

Rolling Horizon Evolutionary Algorithms in General Video Game Playing

Raluca D. Gaina

Submitted in partial fulfilment of the requirements of the
Degree of Doctor of Philosophy

Game AI Group
School of Electronic Engineering and Computer Science
Queen Mary University of London
United Kingdom

14 April 2021

Statement of Originality

I, Raluca Daniela Gaina, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Signature: Raluca D. Gaina

Date: 14 April 2021

Abstract

General video game playing aims to design an artificial agent capable of rational thought, which would achieve high-level play in any game, thus needing to remove domain knowledge and introduce techniques to gather information and statistics about the previously unknown game. While Monte Carlo Tree Search has dominated the area, Rolling Horizon Evolutionary Algorithms (RHEA) were shown in early work to have the potential of reaching an even better performance. This thesis presents a series of experiments carried out to analyse the performance and behaviour of RHEA, which evolves, online, a sequence of actions to play in a game. We analyse its various properties and parameters, as well as combinations with other algorithms. Results obtained are favourable and outperform previous state-of-the-art in several games. A deeper visual analysis tool, VERTIGØ, was created to enable the capture of statistics live during any of the games within the General Video Game AI framework. The features extracted were also used to predict RHEA's performance, with great results even from the very early stages of a game. The multitude of parameters resulting from the several studies led to work on automatic optimisation, using the N-Tuple Bandit Evolutionary Algorithm and several other simpler methods. The algorithm's parameters were tuned both offline and online with mixed results, but high promise is found in helping the algorithm generalise better across a wider range of games, and even observe first win rates in extremely difficult environments. Applications of the algorithm in different games are also explored: RHEA is very aggressive in Pommerman, competitive in Tribes and a top contender in tabletop and real-life physics-simulating games. The thesis finally discusses new research directions and how RHEA could interact with humans and other artificial systems within the context of a present, continuous, 'always-on', interactive game-playing entity.

Acknowledgements

There are so many people who have contributed in small or larger parts to my reaching this point in my life, that I am sure I have forgotten some, though I've tried to name as many as I could. That in no way reduces their impact in my life and career, it's simply a sign of a tired mind after rushing to finish this thesis in time during a global pandemic. I am very grateful to each and everyone I've met in the past 4 years and I wish to see them all again soon to properly express my gratitude in person.

I would like to thank everyone in the **QMUL Game AI group**, who have built a great community of researchers, supportive and playful, with long nights of games, beers, parties and research discussions, even in pandemic times. In particular, I would like to highlight Diego, who I hope is happy to see his first PhD student graduating soon, his tireless and eager support in everything I've done so far in my career, including searching for and finding my own interests. Special thanks go to Simon Lucas as well, who has given me many opportunities that have helped me meet amazing people, work on great projects and develop into the researcher I am today. I will also give special mentions to Jeremy, Simon Colton and Mike Cook, who I'm grateful to for treating me as a colleague and an equal, showed me nothing but kindness and have been truly great to work with. Lastly, I'll give thanks to Melissa, who has always brightened my days while at the Mile End campus and has been a great help and warrior on the students' side.

Next, I would like to thank those that have been with our QMUL Game AI group for shorter times, yet all have been a joy to have around and work with. Alex, who has happily joined in our various projects and has provided great insight into crossing the PhD finish line. Olve, who combines the best confidence, humility, eagerness to learn and an absolute joy for life, lighting up any space with his energy - Raluca is Thanks! Raul, who has been fun to work with on many different projects during his visit. Mike Preuss, who is a great partner in event organisation and the quickest person to reply to emails I know. Matthew, who is always fun to be around and one person I look forward to seeing again at conferences.

I also have to give thanks to the **IGGI** family, not only for the funding allowing me to do the PhD, but also for the great adventures we've shared as a community. Cris and Rokas, who I've started this journey with in Essex and I'll close it with thanks to both together as well. Ivan, James and Martin, who have been always up for research talks, working on various projects, games and fun. Carlos and Cristina, who are some of the kindest people I know and have been the most amazing support in very difficult times, I am so thankful I've met them both. And last but not least, Jo, who is the absolute warrior behind all IGGI students.

Three special months during my PhD were spent at Microsoft Research Cambridge, and I wish to give huge thanks to everyone who has made it possible, as well as to everyone I've met there. Sam, who is incredibly kind and supportive. Carol, Jarek and Adrien, who I've shared an office with and who have offered great support and friendship throughout my time there. Andre, Dave, Ben, Haiyan and Sean, who have been great colleagues, bosses and/or always happy to have a laugh at lunch time.

I send thanks to the women in science who I look up to and have inspired me along the way. Jialin, first introduced to me as "the postdoc you'll probably work with a lot", a great role model and close friend. Vanessa, who is only *slightly* older than me (!), brilliant, has taught me confidence and strength, has been a great friend and has brought much happiness and laughter in my life. Chiara, who is another amazing young woman and a kind presence. Katja, brilliant strong and joyful, striking the perfect balance between a strict yet understanding boss and a best friend.

And the final people I would like to mention in the professional plan are great leaders in their domains who I've had a chance to meet, chat and laugh with, and have each offered me friendship and opportunities which have helped me grow. Jeff, Mark, Christoph, Julian and the NYU crew, Tommy, Dan Ashlock, Richard. I give you all very special thanks for the role you've had in my life and career!

I've saved two people that deserve the warmest heartfelt thanks for a separate mention. We've shared view and research talks at conferences, laughs over beers, late night pizza in New York (sorry Phil!), late night donuts in Maastricht, castle adventures in Edinburgh and PacMan magnet jokes. Damien and Phil, the closest friends that I see way too rarely, but who are the best people who make every conference we've attended together absolutely wonderful and have always been there for me. I thank you dearly!

Last but not least, I would like to thank my friends and family for all their unconditional support and always being on my side. Kamolwan, who would have still been my closest friend and research partner

had he stayed in the UK, and who I miss dearly, but who I know is building a great life in Thailand. John, who is another of my closest friends, though we don't meet as often as we should, and is always happy to play late night games, or share pictures of his cats. My mum and dad, who are my absolute biggest fans, and have sacrificed so much for me to be where I am today. I have to give them a big shoutout here to let the world know how grateful I am to them for all they've given me, how much I love them, and that I hope I've made them proud. I'll also mention my niece here, who is almost 1 and a half at the time of writing, coincidentally shares a name with a research paper I wrote not long before she was born (Project Thyia) and I hope she'll randomly find this one day and smile when she sees her name - hi Thea!

And of course, I'll thank Mia and Mini the cats, who have purred in support while I finished this thesis.

This work was funded by the EPSRC CDT in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1

Licence

This work is copyright © 2021 Raluca D. Gaina, and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported Licence. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Contents

1	Introduction	21
1.1	Contributions	22
1.2	Definitions	24
1.3	Associated Publications	25
1.4	Thesis Structure	28
2	Background	29
2.1	General Video Game Playing	29
2.1.1	General Video Game AI Framework	30
2.1.2	Game set	30
2.1.3	Sparse reward systems	31
2.2	Monte Carlo Tree Search	32
2.2.1	Implementation Details	33
2.2.2	State evaluation	33
2.3	Rolling Horizon Evolutionary Algorithms	33
2.3.1	RHEA improvements	34
2.3.2	RHEA hybrids	35
2.3.3	Population diversity	35
2.3.4	Macro actions	36
2.4	Visual Game Analysis	37
2.5	Win Prediction	37
2.6	Optimisation	38
3	Rolling Horizon Evolutionary Algorithm	41
3.1	Vanilla RHEA	41
3.2	Modifications and Parameters	43
3.2.1	Genetic Operators	43
3.2.2	Fitness Assignment	45
3.2.3	Initialisation	46
3.2.4	Frame-skip	46
3.2.5	Shift Buffer	47
3.2.6	Dynamic Depth	47
3.2.7	Monte Carlo Rollouts	48
3.2.8	Bandit-Based Mutation	49
3.2.9	Statistical Tree	49
3.2.10	Diversity	50
3.2.11	Other Parameters	50
4	RHEA Benchmarking	51
4.1	Population Size and Individual Length	51
4.1.1	Results and Discussion	54
4.2	Population Initialisation	60
4.2.1	ISLA Seeding (Algorithm <i>RHEA-ISLA</i>)	62
4.2.2	MCTS Seeding (Algorithm <i>RHEA-MCTS</i>)	62
4.2.3	Results and Discussion	62
4.2.4	Overall Seeding Comparison	63
4.3	Hybrids	67
4.3.1	Results and Discussion	70
4.4	Conclusions	73

5	RHEA Analysis	77
5.1	Visual Analysis	77
5.2	Sparse Reward Landscapes	79
5.2.1	Baseline Methods	81
5.2.2	Experiments	82
5.2.3	Results and Discussions	83
5.3	General Win Prediction	85
5.3.1	Classification	86
5.3.2	Data set	86
5.3.3	Predictive models	88
5.3.4	Live play results	90
5.4	Conclusions	94
6	Automatic Parameter Optimisation	97
6.1	Offline	98
6.1.1	N-Tuple Bandit Evolutionary Algorithm	98
6.1.2	Experiments	99
6.2	Online	106
6.2.1	Approach	106
6.2.2	Experimental Setup	110
6.2.3	Results and Discussion	110
6.3	Conclusions	114
7	Applications	117
7.1	RHEA in Pommerman	118
7.1.1	Related work	119
7.1.2	Framework	120
7.1.3	Agents	121
7.1.4	Experimental Setup	122
7.1.5	Results	122
7.2	RHEA in Tribes	126
7.2.1	Strategy Games	127
7.2.2	The Framework	128
7.2.3	Agents	128
7.2.4	Experiments and Analysis	130
7.3	RHEA in Tabletop Games	132
7.3.1	Tabletop Games	133
7.3.2	The Framework	134
7.3.3	Discussion	138
7.3.4	Challenges and Opportunities	140
7.4	Conclusions	141
8	Further Research Pathways	143
8.1	Project Thyia: RHEA as AI Entity	143
8.1.1	AI Entities	144
8.1.2	Continual Learning	145
8.1.3	Learning while Planning	146
8.1.4	Planning to Learn	146
8.1.5	Proposed System: Thyia	147
8.1.6	Ethical Implications	150
8.2	Representation Types	151
8.2.1	rhNEAT	152
8.2.2	Experiments	154
8.3	Physics-Based Games	158
8.3.1	Real-World Physics Games in GVGAI	158
8.3.2	Experiments	159
8.3.3	Results and Discussion	159
8.4	Conclusions	162
9	Thesis Conclusions	165
9.1	Future work	170

A	20 GVGA Games	189
B	Deceptive Games	193
C	Physics-Based Games	195

List of Figures

3.1	Rolling Horizon Evolutionary Algorithm cycle, repeated for several generations.	42
3.2	Evaluation of one individual in the Rolling Horizon Evolutionary Algorithm, given the current game state.	42
3.3	Selection options: roulette, rank and tournament. The first two can be seen as wheels spinning and the slice (or individual) next to the selection point is the one chosen; they are both representations of the same population. In the tournament visualisation, the yellow individuals were randomly chosen for the tournament, and the one highlighted green had the highest fitness amongst them and was selected.	45
3.4	One Step Look Ahead (ISLA).	46
3.5	Monte Carlo Tree Search (MCTS)	47
4.1	Solution space explored by a simulated evolutionary algorithm with varying population sizes . Individual length set to $L = 2$ for visualisation purposes, with values for each gene (0-5) on the X and Y axes. Each colour is a different individual mutating over 20 generations (1-bit mutation), each point plotted with 0.05 opacity (thus more intense colours signify the point was sampled more times).	52
4.2	Level space explored by a simulated evolutionary algorithm with varying individual lengths . Population size set to $P = 1$ for visualisation purposes, axes corresponding to level space X and Y axes, in a grid of size 10. Each point is plotted with 0.05 opacity (thus more intense colours signify the point was sampled more times).	52
4.3	Level space explored by a simulated evolutionary algorithm with varying individual lengths . Population size set to $P = 1$ for visualisation purposes, axes corresponding to level space X and Y axes, in a grid of size 10. Each point is plotted with 0.05 opacity (thus more intense colours signify the point was sampled more times). Featuring both positive (green) and negative (red) rewards, which give the colour to the whole individual once encountered; blue remains neutral.	53
4.4	Change of win rate as population size increases, for different individual lengths, in all deterministic games tested. The standard error is shown by the shaded boundary.	55
4.5	Change of win rate as population size increases, for different individual lengths, in all stochastic games tested. The standard error is shown by the shaded boundary.	56
4.6	Change of win rate as individual length increases, for different population sizes, in all deterministic games tested. The standard error is shown by the shaded boundary.	57
4.7	Change of win rate as individual length increases, for different population sizes, in all stochastic games tested. The standard error is shown by the shaded boundary.	58
4.8	Win rate per game for best win rate per game of RHEA, MCTS and RS from experiments presented in Table 4.4, $L = 20$	59
4.9	Change of win rate as population size and individual length increase, in all deterministic games tested. The standard error is shown by the shaded boundary.	63
4.10	Change of win rate as population size and individual length increase, in all stochastic games tested. The standard error is shown by the shaded boundary.	64
4.11	Win rate per game for <i>RHEA-R</i> , MCTS, <i>RHEA-MCTS</i> and <i>RHEA-ISLA</i> . * symbol shows the best result per game over all configurations tested for that algorithm and therefore does not reflect the performance of a single algorithm.	67
4.12	Win percentage for all configurations. The colour bar denotes in how many unique games row was significantly better than column. Legend: A = Vanilla, B = EA-shift, C = EA-tree, D = EA-tree-shift, E = EA-bandit, F = EA-bandit-shift, G = EA-bandit-tree, H = EA-bandit-tree-shift	70

4.13	Win percentage for configuration 10-14. The colour bar denotes in how many unique games row was significantly better than column. Legend: A = Vanilla, B = EA-roll, C = EA-shift, D = EA-shift-roll, E = EA-tree, F = EA-tree-roll, G = EA-tree-shift, H = EA-tree-shift-roll, I = MCTS	72
5.1	VERTIGØ graphical user interface.	78
5.2	VERTIGØ example plots, including positional heatmap (squares with reduced opacity) and simulations (circles of different sizes and colours).	80
5.3	Win rate in sparse rewards games for RHEA and MCTS with extreme length rollouts. Shaded areas indicate the standard error of the measure.	84
5.4	Variations in dynamic rollout length (blue) and fitness landscape flatness (orange) for RHEA agent “Butterflies”, level 0. Note that the scale for rollout length is on the secondary (right) Y axis.	84
5.5	Feature correlation.	88
5.6	Class predictions by features for the different game phases. Red (left side) signifies the model feature predicts a loss, green (right side) a win. The probability of a class being selected based on individual feature recommendation is plotted on the X axis.	90
5.7	Model F1-scores in the game “Ghost Buster”, trained on variations of RHEA and RS (80 training games) and tested on MCTS.	91
5.8	Live play results (part 1), averaged over up to 1400 runs, 14 agents, 100 runs per game. Game ticks on the X axis, maximum 2000. 3 predictor models trained on early, mid and late game data features, as well as the baseline rule-based predictor. If F1-scores were 0 for all models, accuracy is plotted instead. Win average reported for each game.	91
5.9	Live play results (part 2).	92
5.10	Live play results (part 3).	93
6.1	Parameter search space, size 5.36×10^8 (excluding dark grey boxes). Possible values for a parameter in curly brackets, default value in underline green. Parameters not in the 1 st column are dependent on others (denoted with arrows from parent to dependent; parent value required for dependent to affect phenotype is noted in blue angled brackets). Dark grey parameters are not included in the experiments for this paper (default values used instead). In yellow parameters previously analysed in literature, with citation.	100
6.2	Progression of solution fitness in the internal n -tuple model during NTBEA optimisation process in 2 games, “Missile Command” (left) and “Intersection” (right): plotting the value in the model for the solution evaluated at each iteration (light blue). The darker line indicates the value in the model for the seeded solution at each iteration (after n -tuple updates). Vertical lines indicate when the algorithm changes the best solution recommendation, based on its internal n -tuple model.	101
6.3	1-tuple: rollout length percentage parameter. Colours show average fitness for each data point, with blue being highest (1.0) and white being lowest (0.0). Each data point is highlighted with a black circle; the larger the circle, the more times that parameter value was sampled.	103
6.4	1-tuple: genetic operator parameter. 0 - crossover and mutation. 1 - mutation only. 2 - crossover only.	103
6.5	1-tuple: offspring count parameter.	103
6.6	1-tuple: frame-skip type parameter. 0 - repeat. 1 - null. 2 - random. 3 - sequence.	103
6.7	2-tuple: individual length and population size. Colours show average fitness for each data point, with blue being highest (1.0) and white being lowest (0.0). Each data point is highlighted with a black circle; the larger the circle, the more times that combination of values was sampled.	104
6.8	2-tuple: mutation type and crossover type.	105
6.9	Win rate of all tuners (using RHEA configuration 10-15-1000), compared against RHEA state-of-the-art.	111
6.10	Win rate of RND tuner, with all RHEA configurations, compared against RHEA state-of-the-art.	111
6.11	Normalised count of times considered best, per parameter value, one game per row. Parameter value index corresponds to list from Table 6.2. All use 5-10-1000 RHEA configuration.	112
6.12	Average parameter values in winning game instances. All use 5-10-1000 RHEA configuration.	113
7.1	Original “Pommerman” (left) and Java version (right).	118

7.2	Events recorded during the FFA games played (results for TEAM are very similar). All charts show values for $VR = \{1, 2, 4, \infty\}$. Shaded area shows the standard error of the measure.	125
7.3	Bomb placement by RHEA and MCTS in TEAM mode. From left to right: $VR = \infty$; $VR = 4$; $VR = 2$; $VR = 1$. Agent starts in the top left corner.	125
7.4	“Tribes” (left) and The Battle of Polytopia (right).	126
7.5	Example games in the TAG framework.	132
7.6	GUI for “Love Letter”, red outline shows current player (player 3).	136
7.7	Action space size in “Uno” with all player number versions; 1000 runs per version played by random players.	138
8.1	Thyia system. Composed of 3 core modules: a game player (planning AI agent), a learner and a game set. Additional modules include external communications with the “real world” for game sharing and human interaction; and an optimisation module for tuning the game player and learner’s parameters. Further enhancements include a knowledge base, detailed analytics and forward model learning. We include a possible connection with a game designer, which would be providing games for the system to play.	148
8.2	Two parents lined up for crossover according to their innovation numbers. Dark genes are <i>disjoint</i> , while the dark cell with its innovation number underlined is <i>excess</i>	152
8.3	Summary of an individual evaluation.	153
8.4	Win rates per game of different rhNEAT variants: baseline rhNEAT, with population carrying (+cp), speciation (+sp) and with both (+cp,+sp).	155
8.5	Win rates per game of different rhNEAT variants: calculating the fitness as the value of the last state reached through the rollout (rhNEAT), as the sum of the values of all states visited (-acc) or as the discounted sum of the values of all states visited (-accdisc).	156
8.6	Win rates per game of different rhNEAT variants: individual fitness assigned as the last evaluation only (rhNEAT), averaged for all evaluations (-avg) or adjusted with a learning rate (-lr).	157
8.7	Win rates per game for rhNEAT, MCTS, RHEA and RHEA state-of-the-art (SotA).	157
8.8	Victory rate (with standard error bars) in the game “Bird”, for all algorithms and configurations.	161

List of Tables

2.1	Game set including feature analysis. The last 3 columns show clusters as depicted in previous works; games with the same value are denoted as part of the same cluster. As (1) do not include all games we use in their study, column (2) shows the game indexes between which the missing games are placed by Mark Nelson (lower-higher); (3) shows more recent work clustering all GVGAi games.	31
3.1	Parameter Search Space. Parameters noted with † are dependent on the value of other parameters. Last column shows default values for each parameter; unless otherwise specified, the parameters in all experiments are fixed to this value when not varied for studies.	44
4.1	Average win rate over all 20 games tested, for different values of population size (P) and individual length (L). Standard errors in brackets. Highlighted in bold style is the best result.	54
4.2	Average win rate over the 10 deterministic games tested, for different values of population size (P) and individual length (L). Standard errors in brackets. Highlighted in bold style is the best result.	54
4.3	Average win rate over the 10 stochastic games tested, for different values of population size (P) and individual length (L). Standard errors in brackets. Highlighted in bold style is the best result.	54
4.4	Comparison of win rates achieved by RHEA, RS and MCTS with higher budgets. Win rates for all games (T), deterministic (D) and stochastic (S). All algorithm use individuals (or iterations for MCTS) of length 20, as many as possible within the given budget.	59
4.5	Win rate and average score achieved (plus standard error) in 20 different games with configuration $P = 1$ and $L = 6$. Bold font shows the algorithm that is significantly better than both others in either victories or score.	64
4.6	Pairwise significance comparison between vanilla RHEA and ISLA-seeded RHEA.	65
4.7	Significance comparison between vanilla RHEA and MCTS-seeded RHEA.	65
4.8	Significance comparison between ISLA-seeded RHEA and MCTS-seeded RHEA.	65
4.9	Average of victories in all 20 games. Bold style indicates a significantly better average <i>win rate</i> than both the other two seeding variants. If the * symbol is present additionally, it indicates instead a significantly better average <i>score</i> than both the other two seeding variants. The bottom of the table adds up the number of games in which the algorithm was significantly better than the other two variants in average victories, with counts for games with significantly better average scores in brackets. The non-parametric Wilcoxon signed-rank test ($p\text{-value} < 0.05$) was used to test significance.	66
4.10	Significance comparison of algorithms <i>RHEA-R</i> , <i>RHEA-MCTS</i> and MCTS in all 20 games and all configurations.	67
4.11	Configuration 5-10. Rankings table for part 1 algorithms across all games. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.	71
4.12	The best algorithms (by Formula-1 points and win rate) in all configurations and rollout repetitions (R), as compared against the other variants in the same configuration and the same R value (includes variants without rollouts).	72
4.13	Configuration 10-14, $R = 5$. Best algorithm found (EA-shift-roll) compared with MCTS. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.	73
4.14	Configuration 10-14, $R = 1$. Algorithm most similar to MCTS (EA-tree-roll) compared with MCTS. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.	73
5.1	Deceptive game set including feature analysis.	81

5.2	Extreme length rollout budget allocation. Default configuration in bold.	82
5.3	Average win rate for long rollouts variations. Distinction is made between sparse and dense rewards games, with the final column averaging over all games. Budget for each algorithm is $L \times 60$. RHEA obtains significantly better results than MCTS in all but sparse reward games ($L = 200$).	83
5.4	Win rates for RHEA and MCTS, vanilla and dynamic variants (non-shift RHEA). Distinction is made between sparse and dense reward systems, with the last column averaging win rates over all games.	85
5.5	GVGAI-style Formula-1 point ranking of all methods. Type and configuration (rollout length L if one value, population size P and rollout length L if two values) are reported, followed by the sum of Formula-1 points across 20 games and the average win rate.	87
5.6	Global rule-based classifier report. Global model tested on all game ticks of all instances in the test set.	89
5.7	Global AdaBoost classifier report. Global model tested on all game ticks of all instances in the test set.	89
5.8	Feature importances extracted from global model. ϕ_x represents a feature and its associated importance.	89
5.9	F1-Scores each model per game phase over all games, accuracy in brackets. Each row is a model, each column is a game phase. Highlighted in bold is the best model on each game phase, as well as overall best phase and model.	90
6.1	RHEA best win rate (and standard error) recorded in all games. “opt” rows show NTBEA optimisation results, other rows show previously best recorded (with corresponding citation, highlighted in yellow). Parameters using default values (as per Figure 6.1) highlighted in green. Enhancements include values for dependants in brackets. Win-rates in bold are the higher values observed, if different.	102
6.2	Parameter Search Space, total size 270, or 99 valid combinations if parameter dependency is taken into account.	107
6.3	Results of all tuners for all RHEA configurations tested. Showing average win rate in all 20 games; average difference in win rate to RHEA SotA in games in which the tuned agent is better (Δ_{Better}) and worse (Δ_{Worse}), with number of games in brackets. Highest win rate, highest Δ_{Better} and lowest Δ_{Worse} are highlighted for each tuner.	111
6.4	Tuples considered best most times and Shannon entropy $H(x)$ for all tuners, one row per game, with RHEA configuration 5-10-1000; showing only the genetic operator 1-tuples. Parameter index and value index correspond to order in Table 6.2 and game index corresponds to list in Section 6.2.2. Majority agreement across tuners highlighted for 1-tuples.	112
7.1	Games including a mapping to relevant features in Table 2.1. <i>Opp</i> refers to the opponent. <i>Solo</i> refers to the player being the only one left in the game. Most <i>Play card</i> action spaces are abstracted, and often involve further decisions to be made, or different phases. <i>Counter</i> win conditions involve having the highest amount of a specific game component. <i>Line</i> refers to completing a grid line (row, column or diagonal) with the player’s symbol. Other concepts (e.g. <i>Death</i>) are also abstracted and the reader is recommended to read the full game description. Games with ‘x’ in the stochasticity column have a stochastic setup and randomness affects gameplay after the game has started as well. Games with ‘**’ in the stochasticity column only have a stochastic setup, but are deterministic thereafter.	117
7.2	Experimental Setup. $All \equiv VR \in \{1, 2, 4, \infty\}$. Each set up is repeated 200 times (10×20 fixed levels). There are 32 different configurations, totalling 6400 games played.	122
7.3	FFA win rate (W), ties (T) and losses (L). 1^{st} column indicates vision range $\in \{1, 2, 4, \infty\}$. Names in italics represent results averaged across players of the same type.	123
7.4	TEAM win rate (W), ties (T) and losses (L). 1^{st} column indicates vision range (VR). Results include 2 agents of the same type on a team and average across them. The row agent team plays against the column opponent team.	124
7.5	Win rate for each row player averaged across 500 games against the column player. The values between brackets indicate standard error.	131
7.6	Statistics for all games averaged across 2500 game ends. The values between brackets indicate standard error.	131
7.7	Analysis of games, played 1000 times for each possible number of players on each game, using random agents.	138

7.8	AI player performance, 100 game repetitions. Highest win rate in bold. “Tic-Tac-Toe” played in a round-robin tournament. “Pandemic” uses 4 instances of the same agent. All others played in their 4-player variants, with 1 instance of each agent.	140
8.1	rhNEAT parameter values.	154
8.2	Summary of results showing, for each approach, average win rate (and standard error) in the 20 games, the number of games it achieved the highest positive win rate in the subset (including the absolute highest count, i.e. no ties), and the number of games in which it achieved the highest score.	155
8.3	Physics game set including feature analysis. Symbols as used previously in Section 2.1.2. Actions key: Rot = rotate; Move = move; LR = left and right; UD = up and down; Shoot = create a missile sprite with the same direction as the avatar and a particular speed; AD = accelerate and decelerate; Jump = add force in up direction.	159
8.4	Results of the controllers with their default $L = 10$	160
8.5	Results for all algorithms and configurations. Indicated values are the average of victories across all games, with the standard error between brackets. Results in bold mark the best performances for RHEA and MCTS.	160
8.6	Results per game in the configuration $L \times M = 30$. Averages and standard errors of the measures indicated in bold when better than the other variants (<i>italics</i> where the best result is shared).	161
9.1	20 GVGAI games state-of-the-art win rate. Games are indexed 0-19 in the following order: “Dig Dug”, “Lemmings”, “Roguelike”, “Chopper”, “Crossfire”, “Chase”, “Camel Race”, “Escape”, “Hungry Birds”, “Bait”, “Wait for Breakfast”, “Survive Zombies”, “Modality”, “Missile Command”, “Plaque Attack”, “Sequest”, “Infection”, “Aliens”, “Butterflies”, “Intersection”. Full game descriptions in Appendix A. MCTS column hows highest MCTS win rate, with the implementation described in Section 2.2. RHEA column shows highest RHEA win rate, obtained with the configuration described by the rest of the columns. P.Size = population size; I.Len = individual length; Offspring = offspring count; 1 elite for all games; Init. = initialisation method; Selection = selection type; Crossover = crossover type; Mutation = mutation type; Fit. = fitness assignment type; DD = dynamic depth; SB = shift buffer (discount in brackets); MC = Monte Carlo rollouts (length and number of repetitions in brackets); Skip = frame skip (type in brackets, optionally with number of frames skipped); Fit.Div = diversity in fitness (weight in bracket).	168
9.2	“Pommerman” state-of-the-art win rate (FFA game mode; 200 games per vision range option between MCTS, RHEA, one step look ahead and rule-based players; vision range is 2 for best RHEA result, and 1 for best MCTS result), see Section 7.1 for details.	168
9.3	“Tribes” state-of-the-art win rate (averaged across 2500 two-player games from a round-robin tournament between RHEA, MCTS, Monte Carlo search, rule-based, one step look ahead and random players), see Section 7.2 for details.	169
9.4	5 GVGAI deceptive games state-of-the-art win rate. Games are indexed 0-4 in the following order: “Decepti Coins”, “Flower”, “Invest”, “Sister Saviour”, “Wafer Thin Mints Exit”. Full game descriptions in Appendix B.	169
9.5	10 GVGAI physics-based games state-of-the-art win rate. Games are indexed 0-9 in the following order: “Artillery”, “Asteroids”, “Bird”, “Bubble”, “Candy”, “Lander”, “Mario”, “Pong”, “PTSP”, “Racing”. Full game descriptions in Appendix C. MCTS column includes rollout length \times frames skipped.	169
9.6	8 Tabletop Games state-of-the-art win rate (across 100 repetitions per four-player game, played by RHEA, MCTS, one step look ahead and random players; “Tic-Tac-Toe” shows results for two-player games from a round-robin tournament; “Pandemic” shows results for teams of the same player), see Section 7.3 for details. Games are indexed 0-7 in the following order: “Tic-Tac-Toe”, “Dots & Boxes”, “Love Letter”, “Uno”, “Virus!”, “Exploding Kittens”, “Colt Express”, “Pandemic”. RHEA mutation randomly chooses one gene in the individual and mutates all subsequent genes.	169

Chapter 1

Introduction

Artificial Intelligence is concerned with creating an agent capable of rational thought. When applied to games, the agent must be able to make decisions which would lead to fulfilling its goal (usually winning, possibly against an opponent). Academic interest for Artificial General Intelligence has spread across Game AI research during the last years, and researchers are trying to push the boundaries of AI by bringing forth new methods, as well as new testbeds and complex challenges, to obtain an agent capable of achieving high-level play in any given game. Examples of such work in general domains include the Arcade Learning Environment (ALE), where Deep Reinforcement Learning techniques have been able to reach human level of play (4), or the General Video Game AI (GVGAI¹) Framework and Competition (5; 6). GVGAI proposes a benchmark for planning, learning and procedural content generation that has attracted multiple authors within the last few years.

Although tree-based search methods (and in particular, Monte Carlo Tree Search, or MCTS) have been, in most cases, proclaimed winners of different GVGAI game-playing competition tracks (5) and have been generally the preferred method for search-based planning in game-playing research, approaches based on evolutionary algorithms, and in particular Rolling Horizon Evolutionary Algorithms (RHEA), are an excellent alternative to tree search for real-time control in games. First introduced by Perez et al. (7), RHEA evolves a sequence of actions to play in the game over several generations, using genetic operators to combine, modify and discard solutions during its allocated thinking time, at every game step, in order to obtain the best possible solution. The first action of the best sequence found is played in the game, and planning resumes in the next game step. Its name stems from the fact that it keeps a constant sequence length, but at each game tick, as the game progresses, RHEA can see one step further into the future, thus “rolling” the horizon line forward. Previous to the work presented in this thesis, this algorithm was only explored in limited settings and environments, and in a yet very simple form.

This thesis goes beyond previous research on RHEA to bring together several old and novel modifications, as well as extracting several control parameters, to create a highly-customisable algorithm able to perform well in a large selection of games. We look into different parameter settings and present experiments with variations of the algorithm, discussing the most interesting results obtained. Different experiments look at in-depth analysis of the algorithm’s decision-making process, as well as at automatically finding good parameter settings on a wide range of games. The main domain in which we test the algorithm is General Video Game Playing (GVGP) (8), and, in particular, the General Video Game AI framework (9; 10), working with a subset of games which includes varied features to represent a wide array of challenges. Later automatic optimisation work is mainly motivated by the fact that it is hardly possible that the same parameter setting would work equally well for all of the assorted games of the GVGAI corpus: these games can vary across many dimensions, such as their level of stochasticity, average duration of a game, presence or absence of Non-Player Characters, etc. Therefore, one algorithm configuration cannot be expected to perform highly across all games. Benchmarking and exploring various aspects tried in literature previously in a consistent experimental setup is essential to identify which solution works where and the advantages each method brings, to combine them for efficient evolution and exploration of the solution search space.

Lastly, with the intent of exploring the potential of the method beyond the simple games of GVGAI, RHEA is applied in several external domains: Pommernan (11), Tribes (12) and TAG (13). An ample discussion on other novel work that opens up new research pathways is included at the end of the document, together with a summary of results and a statement of the new state-of-the-art obtained through the completion of this thesis. The impact of the research presented here is already evident within the research community, with more and more works adopting RHEA as the algorithm of choice, and testing its performance in different environments. Duarte et al. summarise stand-out learning and planning methods

¹ www.gvgai.net

for game-playing in a recent survey (14), giving RHEA a well-deserved highlight as a promising new and upcoming contender to MCTS-based algorithms.

1.1 Contributions

The contributions in this thesis can be summarised as follows:

- **Extensive literature** on the Rolling Horizon Evolutionary Algorithm, its various modifications and environments it has been tested in, as well as other related concepts is presented in Chapter 2 and throughout the thesis in key chapters. This serves as an easy point of reference for future research in the area, to quickly find relevant work and ideas already explored, or to be explored further. Additionally, some basic literature on Monte Carlo Tree Search, its theoretical concepts and details on its implementation are described as well, together with background information on the General Video Game Playing domain used as the main context for experiments included in this thesis. Furthermore, literature on other topics touched on in the thesis (such as automatic optimisation or diversity within evolutionary algorithms) is included.
- The **game set** first used in (15), then in all subsequent experiments presented in this thesis (with few exceptions) combines a diverse set of challenges for artificial game-players, including various objects for the players to interact with in different ways, different mechanics and control schemes, different winning and losing conditions and a variety of levels with different properties and gimmicks associated. The difficulty of the challenges and the impact of a game’s stochasticity varies across games as well, as highlighted later in the results. This set of games is further analysed in detail in terms of unique or challenging game features, which encourages more thorough future research in similar environments, going beyond win/loss metrics and looking further into environment characteristics that lead to interesting insights or particular behaviours. All games used in the thesis are described in the appendices as well for easier comprehension of the true variety tested here. The following research question is answered through this contribution:

Research Question 1 *What is a varied set of environments appropriate for testing general video game-playing algorithms in?*

- This thesis formalises the **Rolling Horizon Evolutionary Algorithm** first introduced in (7), discussing the many aspects the algorithm deals with, from the perspectives of evolutionary algorithms, as well as general game playing. Many modifications and parameters are brought together, including those previously explored in literature, as well as new ones. This aims to serve as an easy introduction and starting point for anyone unfamiliar with the algorithm, to understand its basic workings; but also as a comprehensive collection of possible enhancements tested so far, with recommendations for particular applications within the experiments. Researchers needing particular behaviours or performance in their environments, or looking for modifications to apply to different algorithms, could use this resource as a handy development guide. The following research question is answered through this contribution:

Research Question 2 *What parameters can be extracted from a Rolling Horizon Evolutionary Algorithm, and what modifications can be integrated into the algorithm for varied behaviour?*

- Further, the various **parameters and modifications** in the algorithm are studied in a series of **experiments** using the same common setting, in isolation (to perform a fair analysis of particular benefits brought by individual enhancements), and in combination (to assess which parameters work well together, or where there might be issues in synergy; both of these situations were identified in the experiments and highlighted appropriately). Starting from simple tests regarding the benefits of shorter or longer lookaheads (for exploring more of the level space, at the expense of less iterations and therefore less accurate information; or the opposite), more or less sequences of actions evolved at a time (for exploring more of the search space, at the expense of less iterations and less accurate information; or the opposite), to more complex population initialisation and management techniques or combinations with other algorithms. These experiments serve as a point of reference for the benefits and drawbacks of each modification, and for details on the exploration and analysis of RHEA’s parameter space. The following research question is answered through this contribution:

Research Question 3 *What is the effect of adjusting the values of the parameters and combining modifications on the performance of the Rolling Horizon Evolutionary Algorithm?*

- Naturally, a more in-depth **analysis** into the algorithm’s decision-making process follows. The thesis presents first a tool, VERTIGØ, which facilitates the analysis process live, while playing the game, with an easy-to-use and complete interface, allowing the user to select the game and level to play out of the large GVGA collection, change the algorithm’s parameters and visualise the thinking process with heatmaps and a series of customisable plots; all detailed data can also be saved for further post-processing. These types of visualisations can lead to interesting remarks about an algorithm’s behaviour, which is studied further in experiments on dynamically adjusting the individual length, according to the density of rewards observed during the game (to prioritise exploration of the levels versus accurate statistics at the correct moments in time). All of the features extracted and readily available for post-processing are also employed in a second study, which uses these decision-making features to predict an algorithm’s overall expected performance in the game with great success; this suggests changing the behaviour of the agent to promote certain trends in its decision-making process could lead to a boost in performance, making tools such as VERTIGØ key in not only better understanding the inner-workings of algorithms, but also in improving them. The following research question is answered through this contribution:

Research Question 4 *What insights can be gained from deeper analysis into the algorithm’s decision-making process?*

- Given the large set of parameters that resulted after the experiments presented, which added more and more modifications to the algorithm, it becomes infeasible to choose correct configurations manually in general, as well as for specific problems. Thus another topic tackled in this thesis is that of **automatic optimisation**. Experiments are presented exploring these concepts first offline, taking several days to learn good algorithm configurations for each of the 20 GVGA games tested. This work was later extended to work online instead, with the agent adjusting its parameters while also searching for good action sequences during the game. These approaches are not only efficient at finding parameter settings that work well (and often, better) in many games, but they also offer further insights into the algorithm’s parameter space, such as which values for parameters are preferable, which values could be better explored in a better-tailored search space, or even highlight particular synergies between parameters that cannot be easily identified through manual tuning or human intuition. The following research question is answered through this contribution:

Research Question 5 *How can the large parameter space of the Rolling Horizon Evolutionary Algorithm be searched effectively for a good configuration of parameters?*

- Moreover, the thesis includes towards the end some in-depth discussions of novel work which opens several exciting **new research pathways**. This refers to testing new representations within RHEA, new environments for the algorithm altogether which better reflect real-world circumstances, as well as building a whole artificial entity framework around the algorithm to allow it to interact with our world better, learn from human players and even share its experiences with other artificial entities, such as generative systems, for an exciting create-play-feedback-improve creation loop. The following research question is answered through this contribution:

Research Question 6 *What research directions into Rolling Horizon Evolutionary Algorithms are opened up and encouraged by novel work?*

- Last but not least, all experiments presented in the thesis include detailed results through tables, figures and external links to software, data or extended results. The thesis ends with a summary of the current state-of-the-art in Rolling Horizon Evolutionary Algorithms, as another easy point of reference for researchers interested in the domain. All experiments are grounded in past work in the area of general video game playing and include direct **comparisons to Monte Carlo Tree Search**, the previous state-of-the-art in the domain and favourite across many environments. The thesis compares not only performance of these algorithms, but also their behaviour and particular differences in thinking process, with the aim of shading some light into their distinctions, similarities, and best-case applications. Several variations of the Rolling Horizon Evolutionary Algorithm are shown to outperform MCTS, and we hope to see RHEA as the algorithm of choice much more often due to its superior performance especially in complex sparse reward environments, its simplicity and high adaptability. The following research question is answered through this contribution:

Research Question 7 *What is the new state-of-the-art in Rolling Horizon Evolutionary Algorithms, and how does it compare to the previous state-of-the-art approaches based on Monte Carlo Tree Search?*

1.2 Definitions

This section defines terms and concepts used throughout the thesis.

Definition 1.1 A **game** is an environment in which several objects (or sprites, in 2-dimensional games) interact. One or more of these objects are controlled by a **player**. The end point of the interactions are determined through end/**termination** conditions, which also define if the player has **won** or **lost** the game. Players may earn some **points** or **score** in the game, which usually indicate how well the player is doing and what their progress is towards the game’s winning condition. **Stochastic** games contain some elements of chance/randomness (e.g. random behaviour of a Non-Player Character), while **deterministic** games do not (an action applied repeatedly at the same moment in time leads to the exact same outcome). Most games used in this thesis are also grid-based: the level is split into a grid of a particular size, with game objects, including the one controlled by the player, usually can only move one grid cell at a time, to the left, right, up or down. Most games used in this thesis are **real-time** games, meaning the decision-making time is restricted so that the game runs without any apparent pauses for the user; this is set to 40ms per decision or game frame (as set by the General Video Game AI framework in 2012), which is the equivalent of a game running at 25 frames per second. Modern games (2021) are often required to run at least 60 frames per second, which reduce the decision-making time even further, yet we choose the 40ms limit to ensure compatibility with different machines and fair comparison to previous results in the domain. **General** game playing refers to being able to play multiple games, often without any prior knowledge of what the game is, how it works or what good strategies there are.

Definition 1.2 A **level** is a particular configuration of objects possible within the game. All levels for the games used in this thesis include one and only one **avatar**, which the players control in order to interact with the game’s environment.

Definition 1.3 A **game state** describes a moment in time, including e.g. current positions of objects in the level and properties of the avatar.

Definition 1.4 A **terminal game state** is a game state in which the winning or losing termination condition has been triggered, and the player has won or lost the game.

Definition 1.5 A **game tick/step** represents how many repetitions of a game’s loop/cycle have been executed so far, or how many time units have passed in the game world.

Definition 1.6 An **action** is one way through which a player can interact with the game’s environment. Actions in GVGAi almost always affect the player’s avatar only (e.g. move, jump).

Definition 1.7 A **reward** is a numerical indication of progress in a game. In the context of this thesis, the rewards always (unless specified) refer to the in-game score obtained by a player.

Definition 1.8 An **agent/player/controller** is the actor in the environment, or an algorithm which decides, at every game tick, what action should be applied in the game.

Definition 1.9 A **policy** is a probability distribution over the actions possible in the game (summing up to 1.0). Choosing an action according to a policy means choosing the action with the highest probability.

Definition 1.10 A **heuristic/value function** is a function able to assign a numerical value to a game state, attending to its current properties.

Definition 1.11 A **forward model (FM)** is a function which takes a game state and an action as input, and returns the resulting game state after applying the action. Conceptually, this is used to simulate the effect of actions in the game (without actually playing them) and the forward model mimics the game engine (although potentially inaccurate, using a different random seed in stochastic games).

Definition 1.12 A **rollout** is repeating these steps L times: 1) choose an action possible in the current game state, 2) use the forward model to advance the game state with the chosen action. L is referred to as the length of the rollout.

Definition 1.13 **Lookahead** takes a numerical value that represents how far into the future an agent can see (i.e. the maximum rollout length).

Definition 1.14 A **parameter** is a numerical, categorical or toggle variable which controls some part of an algorithm (e.g. rollout length).

Definition 1.15 *Optimisation/tuning* is find the best/optimal parameters, i.e. those that allow the algorithm to perform to the best of its abilities (e.g. highest win rate). Automatic optimisation/tuning means using an algorithm to perform this task, instead of a human hand-picking the parameters. Algorithms that adjust their own parameters are referred to as **adaptive**.

Definition 1.16 An **individual** is a potential solution to a problem.

Definition 1.17 The **phenotype** represents the observable traits of the individual. In this context, this is the behaviour of the player-controlled avatar in the game world.

Definition 1.18 The **genotype/genome/representation** is the encoding of the phenotype into data structures easier to process. In this context, the genotype of an individual is a sequence of actions to execute in the game.

Definition 1.19 A **population** is a collection of individuals.

Definition 1.20 **Fitness** is a value assigned to an individual, indicating how good the individual is (the higher the fitness, the better the individual). This is calculated through a **fitness function**.

Definition 1.21 **Evolution** is the process of iteratively improving upon an initial population over several generations, by combining and changing the individuals via **genetic operators** and discarding the less fit solutions, in order to increase the fitness of the population.

Definition 1.22 An **iteration** is the part of an algorithm that is repeated multiple times. In evolutionary algorithms, this is equivalent to moving to the next generation once.

Definition 1.23 A **decision budget** controls how many iterations an algorithm is able to perform.

1.3 Associated Publications

Most of the work detailed in this thesis has been presented in national and international scholarly publications, some of which already highly cited and discussed within the community, making these the largest contribution of my PhD (journal publications highlighted with a * symbol; books highlighted with a ^ symbol; all others are conference publications). The papers included in the thesis are accompanied by statements of personal contributions to those works in bold.

Core first-author publications

R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, “Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing,” in *Springer Lecture Notes in Computer Science, Applications of Evolutionary Computation, EvoApplications*, no. 10199, 2017, pp. 418–434

[Chapter 4, Section 4.1] Contributions: implementation, experiments, writing most of the paper. Second author plotted results and helped write the experiments section. Third and fourth authors participated in discussions and writing the paper.

R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, “Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing,” in *Proceedings of the Congress on Evolutionary Computation*, June 2017, pp. 1956–1963

[Chapter 4, Section 4.2] Contributions: implementation, experiments, writing the paper. Second and third authors participated in discussions and polishing the paper.

—, “Rolling Horizon Evolution Enhancements in General Video Game Playing,” in *Proceedings of IEEE Conference on Computational Intelligence and Games*, Aug 2017, pp. 88–95

[Chapter 4, Section 4.3] Contributions: implementation, experiments, writing the paper. Second and third authors participated in discussions.

—, “VERTIGO: Visualisation of Rolling Horizon Evolutionary Algorithms in GVGAI,” in *The 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018, pp. 265–267

[Chapter 5, Section 5.1] Contributions: implementation, experiments, writing the paper. Second and third authors participated in discussions.

——, “Tackling Sparse Rewards in Real-Time Games with Statistical Forward Planning Methods,” in *AAAI Conference on Artificial Intelligence (AAAI-19)*, vol. 33, 2019, pp. 1691–1698

[Chapter 5, Section 5.2] Contributions: implementation, experiments, writing the paper. Second and third authors participated in discussions and polishing the paper.

——, “General Win Prediction from Agent Experience,” in *Proc. of the IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2018, pp. 1–8

[Chapter 5, Section 5.3] Contributions: implementation, experiments, writing the paper. Second and third authors participated in discussions and polishing the paper.

* R. D. Gaina, S. Devlin, S. M. Lucas, and D. Perez-Liebana, “Rolling Horizon Evolutionary Algorithms for General Video Game Playing,” *IEEE Transactions on Games*, 2021

[Chapter 6, Section 6.1] Contributions: implementation, experiments, writing the paper. Second and third authors participated in discussions and polishing the paper.

R. D. Gaina, C. F. Sironi, M. H. Winands, D. Perez-Liebana, and S. M. Lucas, “Self-Adaptive Rolling Horizon Evolutionary Algorithms for General Video Game Playing,” in *IEEE Conference on Games (CoG)*, 2020, pp. 367–374

[Chapter 6, Section 6.2] Contributions: RHEA implementation, setting direction, experiments, writing the RHEA, experiments, results and conclusions sections of the paper. Second author integrate RHEA with online tuning and wrote the introduction, background and tuning approaches sections of the paper. The rest of the authors participated in discussions and polishing the paper.

R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, “TAG: a Tabletop Games Framework,” in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2020

R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, “Design and Implementation of TAG: a Tabletop Games Framework,” *arXiv preprint arXiv:2009.12065*, 2020

[Chapter 7, Section 7.3] Contributions: main TAG developer, games analysis, writing most of the paper. Third author ran agent experiments and wrote that results section in the paper. All authors participated in developing the framework and polishing the paper.

R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, “Project Thyia: A Forever Gameplayer,” in *IEEE Conference on Games (COG)*, 2019, pp. 1–8

[Chapter 8, Section 8.1] Contributions: wrote most of the paper. Second author wrote the “Planning to learn” section. The third author participated in discussions, shaping and polishing the paper.

Other non-first-author publications included in this thesis

D. Perez-Liebana, R. D. Gaina, O. Drageset, E. Ilhan, M. Balla, and S. M. Lucas, “Analysis of Statistical Forward Planning Methods in Pommerman,” in *Proceedings of the Artificial intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 15, no. 1, 2019, pp. 66–72

[Chapter 7, Section 7.1] Contributions: implemented large parts of the framework, wrote the results section and participated in discussions, shaping and polishing the paper. First author coordinated the project, ran experiments and wrote most of the paper. The other authors participated in developing the framework, discussions and polishing the paper.

D. Perez-Liebana, Y.-J. Hsu, S. Emmanouilidis, B. Khaleque, and R. D. Gaina, “Tribes: A New Turn-Based Strategy Game for AI,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 252–258

[Chapter 7, Section 7.2] Contributions: developed the GUI for the framework, developed the RHEA agent used in experiments, wrote the literature review section in the paper and participated in the polishing of the paper. The other authors developed the framework and wrote the rest of the paper.

D. Perez-Liebana, M. S. Alam, and R. D. Gaina, “Rolling Horizon NEAT for General Video Game Playing,” in *IEEE Conference on Games (CoG)*, 2020, pp. 375–382

[Chapter 8, Section 8.2] Contributions: participated in discussions. Helped with writing, shaping and polishing of the paper.

D. Perez-Liebana, M. Stephenson, R. D. Gaina, J. Renz, and S. M. Lucas, “Introducing Real World Physics and Macro-Actions to General Video Game AI,” in *Proceedings of IEEE Conference on Computational*

Intelligence and Games, Aug 2017, pp. 248–255

[Chapter 8, Section 8.3] Contributions: participated in discussions during framework and games development, and in shaping and polishing of the paper. First author led the project and ran experiments. Second author developed the framework additions and games. The other authors participated in discussions and polishing of the paper.

Other first-author publications not included in this thesis

R. D. Gaina, D. Perez-Liebana, and S. M. Lucas, “General Video Game for 2 Players: Framework and Competition,” in *Proc. of the IEEE Computer Science and Electronic Engineering Conference (CEEC)*, 2016, pp. 186–191

* R. D. Gaina, A. Couëtoux, D. J. Soemers, M. H. Winands, T. Vodopivec, F. Kirchgessner, J. Liu, S. M. Lucas, and D. Perez-Liebana, “The 2016 Two-Player GVGAI Competition,” *IEEE Transactions on Games*, vol. 10, no. 2, pp. 209–220, June 2018

R. D. Gaina, R. Volkovas, C. G. Díaz, and R. Davidson, “Automatic Game Tuning for Strategic Diversity,” in *2017 9th Computer Science and Electronic Engineering (CEEC)*, Sept 2017, pp. 195–200

R. D. Gaina and M. Stephenson, ““Did You Hear That?” Learning to Play Video Games from Audio Cues,” in *IEEE Conference on Games (COG)*, 2019, pp. 1–4

Other non first-author publications not included in this thesis

K. Kunanusont, R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, “The N-Tuple Bandit Evolutionary Algorithm for Game Improvement,” in *Proc. of the IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 2201–2208

C. F. Sironi, J. Liu, D. Perez-Liebana, R. D. Gaina, I. Bravi, S. M. Lucas, and M. H. M. Winands, “Self-adaptive MCTS for General Video Game Playing,” in *Applications of Evolutionary Computation*, K. Sim and P. Kaufmann, Eds. Cham: Springer International Publishing, 2018, pp. 358–375

D. Perez-Liebana, K. Hofmann, S. P. Mohanty, N. Kuno, A. Kramer, S. Devlin, R. D. Gaina, and D. Ionita, “The Multi-Agent Reinforcement Learning in Malmö Competition,” in *Challenges in Machine Learning (CiML; NIPS Workshop)*, 2018, pp. 1–4

* D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, “General Video Game AI: a Multi-Track Framework for Evaluating Agents Games and Content Generation Algorithms,” *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, Sep 2019

^ D. Perez-Liebana, S. M. Lucas, R. D. Gaina, J. Togelius, A. Khalifa, and J. Liu, *General Video Game Artificial Intelligence*. Morgan and Claypool Publishers, 2019

S. M. Lucas, J. Liu, I. Bravi, R. D. Gaina, J. Woodward, V. Volz, and D. Perez-Liebana, “Efficient Evolutionary Methods for Game Agent Optimisation: Model-Based is Best,” *arXiv preprint arXiv:1901.00723*, 2019

S. M. Lucas, A. Dockhorn, V. Volz, C. Bamford, R. D. Gaina, I. Bravi, D. Perez-Liebana, S. Mostaghim, and R. Kruse, “A Local Approach to Forward Model Learning: Results on the Game of Life Game,” *arXiv preprint arXiv:1903.12508*, 2019

A. Dockhorn, S. M. Lucas, V. Volz, I. Bravi, R. D. Gaina, and D. Perez-Liebana, “Learning Local Forward Models on Unforgiving Games,” in *IEEE Conference on Games (COG)*, 2019, pp. 1–4

O. Drageset, R. D. Gaina, D. Perez-Liebana, and M. H. Winands, “Optimising Level Generators for General Video Game AI,” in *IEEE Conference on Games (COG)*, 2019, pp. 1–8

R. Montoliu, R. D. Gaina, D. Perez-Liebana, D. Delgado, and S. M. Lucas, “Efficient Heuristic Policy Optimisation for a Challenging Strategic Card Game,” in *International Conference on the Applications of Evolutionary Computation (EvoStar)*, vol. 12104. Springer, 2020, pp. 403–418

1.4 Thesis Structure

The rest of this thesis is structured as follows:

- Chapter 2 reviews literature related to the work presented in this thesis, introduces the domain and game set used, and includes background on Monte Carlo Tree Search, as the main point for comparison throughout experiments.
- Chapter 3 gives an overview of Rolling Horizon Evolutionary Algorithms, explaining conceptual and implementation details. It includes detailed descriptions on all modifications tested within this thesis and resultant parameters.
- Chapter 4 presents experimental work studying several basic parameters of the algorithm (population size and individual length), as well as testing several enhancements previously described in literature within a common setting (the same 20 GVGAI game), in isolation and in combination.
- Chapter 5 presents experimental work looking deeper into the inner-workings of the algorithm through visual analysis and the VERTIGØ tool built, resulting insights developed into larger pieces of research as well as the use of features describing the agent's thinking process to predict performance.
- Chapter 6 addresses the problem of the large parameter space built through this work via automatic optimisation, both offline and online.
- Chapter 7 reviews applications of the algorithm outside of the GVGAI domain and the adaptations needed for specific games: Pommerman (partial-observable Bomberman), Tribes (turn-based strategy game) and tabletop games (within the TAG framework).
- Chapter 8 describes works opening up new pathways of research for Rolling Horizon Evolutionary Algorithms.
- Chapter 9 concludes the thesis with a summary of the results obtained and comments on possible future extensions.
- Finally, Appendices A,B,C include full descriptions of the games used in the different experiments in this thesis.

Chapter 2

Background

2.1 General Video Game Playing

It is important to first give an overview of the larger problem the algorithms developed in this thesis are trying to solve, General Video Game Playing (GGP). The popularity of this domain has increased since M. Genesereth et al. (39) organised the first GGP competition allowing participants to submit game agents to play in a diverse collection of board games. Sharma et al. (40) motivates research in this area by bringing to attention how agents trained without prior knowledge of the game are excelling in specific games, such as TD-Gammon in Backgammon (41) and Blondie24 in Checkers (42); but they cannot be successfully applied in other scenarios or environments. Furthermore, they also suggest that there are several techniques that do work in a general context, as long as there is a standardised way of describing the games and the agent's interaction, otherwise requiring minor modifications.

The problem is further expanded to video games in General Video Game Playing (GVGP (8)), which provide the agents with new and possibly more complex challenges due to a higher and continuous, in practice, rate of actions. One of the first frameworks to allow testing of such general agents was the Arcade Learning Environment (ALE) (43), later used as benchmark for applying Deep Q-Learning to achieve human level of play on the Atari 2600 collection (4). The way the world was presented to the agents in this framework was via screen capture; they would return an action to be performed and the next game state would be processed by the system.

Moreover, (44) explores creating agents for ALE, looking at two different methods of reinforcement learning and search-based techniques, both of which return satisfactory results when trained on only four games, but tested on 50 randomly selected new games. Their training and testing approach is similar to that used in this work. They look at two different techniques that could be implemented for ALE, using reinforcement learning and search-based algorithms. They were trained on 4 games and tested on 50 randomly selected new games, showing promising results.

Within the larger picture of GGP lies the General Video Game AI Competition (GVGAI) (5; 6), which offers a large corpus of games described in a plain text language, making it easy to run general AI agents in several different environments and analyse their performance. The competition has already completed three editions of its single-player track (starting in 2014), with two additional tracks running in 2016 for two-player games (27), level generation (45) and rule generation (46). Therefore, it is attracting a large interest on an international scale, with close to a hundred participants every year across its different tracks.

Considering the variety of games the competition offers, studying methods of identifying which type of game is currently being played may be a key to a successful solution to the GVGP problem. There are a few works which attempt to classify or cluster the games. One classification was generated by Mark Nelson (47) in his analysis of the vanilla Monte Carlo Tree Search algorithm in 62 of the games in the framework, sorted using the win rate of MCTS as a simple criterion. A clustering approach of 49 games by Bontrager et al. (1) separated the games into groups based on their similarity in terms of difficulty to a large set of VGGAI entries. These two studies are used in my research to determine subsets of games for testing algorithms. Additionally, they lay the groundwork for more in-depth studies which would lead to hyper-heuristics being explored to intelligently manage algorithm usage.

This competition is becoming a popular way of benchmarking AI algorithms such as enforced hill climbing (48), algorithms employing advanced path-finding or using the knowledge gained during the game in interesting ways (49; 50), or dominant Monte Carlo Tree Search techniques (51). All of the authors appear to agree on the complexity of the problem proposed, as well as its importance, going beyond the realm of video games towards that of AGI.

Monte Carlo Tree Search (MCTS) has proven to be the dominating technique out of the sample ones

provided in the GVGAI competition, with numerous participants using it as a basis for their entries before adding various enhancements on top of its vanilla form. Most winners of the first competition employ MCTS-based methods, including the winner in 2014, Adrien Couëtoux (6), who used an Open Loop approach.

2.1.1 General Video Game AI Framework

The framework used for all experiments discussed in this thesis is the General Video Game AI (GVGAI) framework (10). GVGAI is a framework widely used in research (9) which features a corpus of over 100 single-player games and 60 two-player games. These are fairly small games, each focusing on specific mechanics or skills the players should be able to demonstrate, including clones of classic arcade games such as Space Invaders, puzzle games like Sokoban, adventure games like Zelda or game-theory problems such as Iterative Prisoners Dilemma. All games are real-time and require players to make decisions in only 40ms at every game tick, although not all games explicitly reward or require fast reactions; in fact, some of the best game-playing approaches add up the time in the beginning of the game to run Breadth-First Search in puzzle games in order to find an accurate solution (9). However, given the large variety of games (many of which are stochastic and difficult to predict accurately), scoring systems and termination conditions, all unknown to the players, highly adaptive general methods are needed to tackle the diverse challenges proposed.

GVGAI includes several different tracks which tackle different problems: single-player planning (5), two-player planning (28) and single-player learning tracks focus on finding general game-playing AI agents which would be capable of planning (with internal models of the world) or learning across all the games in the framework. More recently, level generation (45) (creating levels for any game) and rule generation (46) (creating rules for any given level) challenges were introduced as well, to push the limits of general game Artificial Intelligence.

For the purpose of the experiments described in this thesis, we will focus on the single-player planning track, although the work could easily be expanded to include two-player games.

2.1.2 Game set

20 single-player games, with 5 levels each, were selected out of the larger GVGAI corpus using two different classifications present in literature in order to balance the game set and analyse performance on an assorted subset of games. The first classification was that generated by Mark Nelson (47) in his analysis of the vanilla Monte Carlo Tree Search algorithm in 62 of the games in the framework, sorted using the win rate of MCTS as a simple criterion. The second classification considered was the clustering of 49 games by Bontrager et al. (1), which separated the games into groups based on their similarity in terms of game features. Combining these two lists and uniformly sampling from both provided a diverse subset, as described in Table 2.1. The resultant game set is varied in features and difficulty (according to the performance of various GVGAI competition entries), but it is also important to highlight that half of the games are deterministic and half are stochastic, introducing additional noise to the agent decision-making process.

The game table includes additional information about each game. They showcase varying reward structures, such as games with *no* rewards (with the possibility of gaining points on win/lose conditions only), games with *dense* rewards (multiple interactions with the environment result in a score change) or games with *discontinuous* rewards (a longer sequence of actions is required to obtain the reward). Four different types of winning conditions are featured, in which the player has to kill certain game objects (*Kill*), reach an exit point (*Exit*), wait for a timer to run out (*Timeout*) or complete a certain more precise sequence of actions (*Puzzle*, such as move a box onto a specific point). Three types of losing conditions are included, which result in the player losing if they run out of time (*Timeout*; note that all games include a default timeout of 2000 game ticks, at which point the game ends and the player loses if they did not complete any of the winning conditions), die (*Death*) or fail to kill specific game objects (*No-kill*).

We note that most experiments in this thesis analyse the win rate of the AI players, and not the game score obtained. While the score is important in some games (a higher score could be better if losing), the variety of reward systems make it an unreliable metric, especially when this is deceptive (52). As such, in this study of AI game players, we are solely interested in the ability to win the game (or solve the problem), regardless of score obtained.

Additionally, the 5 levels included with each game vary in *size* (Large - *L*, Medium - *M* or Small - *S*) and *density* of interactive tiles (that is, tiles which produce some sort of effect when the player interacts with it, such as blocking the player's path, moving or getting destroyed). Some games include Non-Player Character (NPCs) that might either help the player (*F*), hurt the player (*E*) or have no direct influence on the player's win/lose condition or score (*N*) through their behaviour. The player may need to collect resources or pay particular attention to their avatar's hitpoints (HP). Finally, games vary in the actions available to

Table 2.1: Game set including feature analysis. The last 3 columns show clusters as depicted in previous works; games with the same value are denoted as part of the same cluster. As (1) do not include all games we use in their study, column (2) shows the game indexes between which the missing games are placed by Mark Nelson (lower-higher); (3) shows more recent work clustering all GVGAI games.

Idx	Game	Stoch.	Rewards	Win	Lose	Levels	NPCs	Res.	Actions	(1)	(2)	(3)
0	Dig Dug	x	D	Puzzle/Kill	Timeout	L/Dense	E		Move+Shoot	4		5
1	Lemmings		D	Exit/Puzzle	Death	L/Dense	N		Move+Shoot	4		5
2	Roguelike	x	D	Exit	Death	L/Dense	E	x	Move+Shoot	4		4
3	Chopper	x	D+Disq	Kill	No-kill	L/Dense	E	x	Move+Shoot		g4-g1	2
4	Crossfire	x	N	Exit	Death	M/Dense	E		Move	2		4
5	Chase		D	Kill	Death	M/Sparse	F+E		Move	2		4
6	Camel Race		N	Exit	Timeout	L/Sparse	E		Move	2		3
7	Escape		N	Exit/Puzzle	Death	M/Dense			Move	2		3
8	Hungry Birds		Disq	Exit	Timeout	M/Sparse		HP	Move		g7-g10	3
9	Bait		N	Puzzle/Exit	Timeout	S/Sparse		x	Move	4		4
10	Wait for Breakfast		N	Puzzle	Timeout	M/Dense	N		Move	2		3
11	Survive Zombies	x	D	Timeout	Death	M/Dense	F+E		Move	3		4
12	Modality		N	Puzzle	Timeout	S/Dense			Move	3		4
13	Missile Command		D+Disq	Kill	No-kill	M/Sparse	E		Move+Shoot	3		2
14	Plaque Attack		D	Kill	No-kill	L/Dense	E		Move+Shoot	3		2
15	Sequest	x	D+Disq	Timeout	Death	M/Dense	F+E	x	Move+Shoot	3		2
16	Infection	x	D	Kill	Timeout	M/Dense	F+E		Move+Shoot	1		1
17	Aliens	x	D	Kill	Death	M/Dense	E		LR+Shoot	1		1
18	Butterflies	x	D	Kill	Timeout	M/Dense	F		Move	1		2
19	Intersection	x	D+Disq	Timeout	Death	L/Dense	E	HP	Move		g18-g17	1

the players (*Move* includes movement in all 4 directions, up, down, left and right; *LR* includes only left and right movement; a special *Shoot* action might be available in some games, with different effects). If the action chosen by the player at any game step is illegal or cannot reasonably be played (e.g. walking into a wall), it is automatically treated as “*do nothing*” by the game engine.

Full descriptions of the games are included in Appendix A. When discussing parameter choices, we will refer to games as similar based on the features described in Table 2.1, or the clustering identified from previous works.

2.1.3 Sparse reward systems

A distinct problem is that of the variety of reward landscapes in games and how most current general methods are not equipped to handle this. Anderson et al. (52) highlight deceptive reward systems in games (i.e. by introducing score gains which guide the AI player away from winning the game), using agents from the General Video Game AI Competition (GVGAI) to show that AI game players can be easily tricked into not finding the optimal solution. Companez et al. (53) look at enhancements for Monte Carlo Tree Search in Tic-Tac-Toe variations meant to overcome such deceptive issues, highlighting a particular situation where the agent should be able to self-sacrifice in the short run in order to obtain a larger gain in the long run.

The variety of games that general algorithms are expected to achieve a high performance on is noted by Horn et al. in (54). They look at the 2D grid-physics games in the GVGAI Framework and identify the different strengths and weaknesses of Evolutionary Algorithms as opposed to methods based on Tree Search. The authors propose a game difficulty estimation scheme based on several observable game characteristics, which could be used as a guideline to predict agent performance depending on the game type. Some of the metrics they extracted tie in to the fitness values identified by the algorithms, such as puzzle elements or enemy (possibly random) Non-Player Characters (NPC) which may negatively impact state value estimation. They also observe the lower performance of most algorithms on sparse reward games, but their study is limited in terms of overcoming the issues highlighted.

Different authors use macro actions to explore the space in physics based games, where one single action may not have much effect on the environment (55; 26). In both works, simple macro actions are implemented which repeat single actions a number of times M , further increasing the computation budget as well, due to the action to be played repeatedly being planned for M time ticks later. Simply repeating the same action M times (similar to the concept of frame-skipping in Reinforcement Learning) proved very effective in the Physical Travelling Salesman Problem (55), but it did not work in all physics based games tested in the GVGAI Framework (26) due to the coarseness resultant, indicating that a dynamic approach may be better.

One approach to deal with sparse reward landscapes specifically is presented in (28). Vodopivec describes the use of dynamic rollout increase, proportional to the iteration number, and weighted rollouts in his Monte Carlo Tree Search (MCTS) based entry in the 2016 GVGAI Two-Player track. The purpose of this addition is specified as combining quick reaction to immediate threats with better exploration of areas

farther away, if time budget allows. This is an interesting general approach, but computation time is potentially wasted if there are no close rewards to guide the search before rollouts become long enough to retrieve interesting information.

Another approach (56) is to combine Deep and Reinforcement Learning on Atari games to learn, from the reward landscape, a bonus function that modifies the UCT policy on MCTS. The authors showed that it is possible to learn from raw perception and improve the performance of MCTS agents in some of these games, by using policy-gradient for reward design.

Finally, a complementary set of methods, often referred to as intrinsic motivation, encourage exploration in ways that ignore rewards and focus instead on properties of the state space (or state-action space). The aim is to encourage the agent to explore novel or less visited parts of the state space, or areas that maximise the agent's affordances (57). For non-trivial games, most possible states are never visited due to the vast state space, so statistical feature-based approximations can be used to estimate the novelty of a state (58). The rollout length adaptation method described here may complement intrinsic motivation methods, but this has not been investigated yet.

2.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is one of the best methods in GVGP at present, and the method of choice for many works as summarised in this section. As a result, we use MCTS for validation in the studies presented in this thesis, to compare against and measure how performances appearing good actually fit within the larger problem, grounding results in the wider context.

There is a wide literature on MCTS, as reviewed in the survey by Browne et al. (59). Its application in General Video Game Playing is varied as well. One of the first usages of MCTS in this context was CadiaPlayer developed in 2007 (60). Although this was a rather basic implementation of the algorithm, it made use of a heuristic based on the history of events in the game in order to learn correct actions (61). Its success in the GGP competition (winning both 2007 and 2008 editions) attracted the interest of other researchers as well, enhancements to improve performance starting to be explored. Even though Finnsson et al. (61) did not find a suitable combination of techniques which would result in better results in all games instead of specific ones, the general strength of their algorithm did increase over the following few years during further development.

Another general MCTS player is Ary (62), which employed transposition tables, Upper Confidence Bound applied to Trees (UCT) and nested MCTS techniques. These improvements helped this algorithm defeat CandiaPlayer and win GGP in 2009. While exploring different versions of their proposed algorithm, Mehat and Cazenave (62) come to the same conclusion as Finnsson et al., where there is no clear enhancement that does better than others in all the games.

Other approaches to this problem in GVGP include incorporating Answer Set Programming in MCTS in Centurio by Möller et al. (63), using state and action patterns to generate domain knowledge by Sharma et al. (64) and influence maps to better direct the search in games with a large search space, in which standard MCTS would move almost randomly due to lack of rewards; an average performance increase is reported by (51). In addition, Chu et al. (50) take the idea of knowledge-based fast-evolutionary MCTS (49) and combine it with path-finding algorithms for a slight improvement over alternatives.

There have been several MCTS adaptations for multi-player turn-based games, using the Minimax technique, in which the levels of the tree alternate between the players and the algorithm assumes the opponent tries to minimise the agent's reward, while its goal is to maximise it. Research in the area attempts diverse improvements with mixed results (65) (66).

Although MCTS has shown to be a top contender in this domain, Evolutionary Algorithms (EA) show great promise at obtaining just as good, if not better, performance. Perez et al. (7) compare EA techniques with tree search on the Physical Salesman Travelling Problem, and their results are satisfactory, encouraging research in the area. In their work, the authors employ several techniques to improve the state evaluation function, such as avoiding opposite actions, movement blocks and pheromone exploration.

It is important to notice that this work focuses on improving domain-agnostic algorithms on a variety of different games. Other agents submitted to the GVGAI Competition (i.e. YOLOBOT (67), the winner of several editions) obtain higher performance than the methods explored here, but also count on stronger heuristics (mostly adapted to respond well to GVGAI games) and combine several algorithms (tree search, A*, best first search, etc.). The focus of our work is to explore improvements in simpler, game-agnostic algorithms, taking their vanilla form as baseline to analyse the effects of the proposed modifications.

2.2.1 Implementation Details

At every game tick, MCTS works by iteratively building a game tree, in which nodes are represented by *game states* and/or statistics (Q , the value of the node, and N , the number of times the node was visited), and the nodes are connected by *actions* taken within the game. At every iteration, new nodes are added to the tree and statistics are updated, through a series of 4 steps:

1. **Selection:** Navigate the tree from the root, until a node which is not yet fully expanded is found (new actions that have not been tried yet can still be taken from that game state). During tree navigation, at each node in the tree, choose a child according to the *tree policy*.
2. **Expansion:** Add a new child of the node reached during the selection step.
3. **Simulation:** Execute a Monte Carlo simulation starting from the newly added node (choose an action available from the game state using a *default policy*, apply the action, and repeat the process L times to reach a final game state).
4. **Backpropagation:** The final game state is evaluated with a heuristic h and this value is propagated up the tree, updating Q and N statistics in all of the nodes visited during the iteration; Q will reflect for each node the average of all values observed during iterations that visited the node.

After several iterations, a *recommendation policy* is used to select an action from the root to play in the game, and the process repeats at the next game tick. There are several things to note about the algorithm: 1) it can be stopped after any number of iterations and return an approximation of a good action to play; given infinite compute and several constraints, the algorithm was theoretically proven to converge to optimal play (68); 2) depending on the tree policy used, the tree built most often grows asymmetrically, favouring those branches where higher rewards are observed; 3) statistics become more accurate with more iterations; statistics are also most accurate towards the root of the tree, while nodes further down in the tree see less visits and are therefore less accurate.

The MCTS variant used in the work presented in this thesis uses an open loop modification, an idea suggested by Perez et al. (69). It consists of not storing the game states in the nodes of the tree, but only the corresponding statistics instead, while using a simulation model to recalculate game states at each iteration. This technique has proven superior in stochastic games, where the randomness aspect leads to inaccurate results in the closed loop MCTS.

Additionally, we specify the *tree policy* used as UCB1 (using an exploration constant $C = \sqrt{2}$ (70) and rewards normalised in $[0, 1]$ using dynamic bounds), the *default policy* used as random and the *recommendation policy* used as the most visited child. The length of the MC rollouts varies, specified for each experiment, together with any other modifications added.

2.2.2 State evaluation

The heuristic function h is always kept to a generic form, and the same for all algorithms tested, throughout the experiments; this aims to maximise the game score, while favouring wins and discouraging losses, see Equation 2.1 (s is the game state being evaluated and *score* is the game score normalised in $(0, 1)$).

$$h(s) = \begin{cases} 1 & \text{win} \\ 0 & \text{lose} \\ \text{score} & \text{otherwise} \end{cases} \quad (2.1)$$

2.3 Rolling Horizon Evolutionary Algorithms

Evolutionary Algorithms (EAs) (71) provide a simple, robust and generally applicable approach for searching a wide variety of spaces, and have been the subject of intensive research for more than five decades. In terms of their application to Game AI, much of the effort has been focused on evolving game-playing AI agents, or on evolving game content (such as level design) (72), game rules or game parameters. Recently, it was shown that Evolutionary Algorithms could be applied as any-time and real-time decision-making algorithms for use in Game-Playing AI, adopting a similar simulation-driven approach to Monte Carlo Tree Search, while being simpler to implement and offering competitive performance (7). This was initially done for one-player games, but was extended to 2-player games in the form of Rolling Horizon Co-Evolution (73).

Among the techniques employed over the last years of the GVGAI Competition, one of the most promising is that of Rolling Horizon Evolutionary Algorithms (RHEA). These methods, rather than basing the

search on game tree structures, use influences from biological sciences to evolve a population of individuals until a suitable one, corresponding to a solution to the problem, is obtained. The way they are applied to the domain of GVGP is by encoding sequences of in-game actions as individuals, using heuristics to analyse the value of each sequence (69). The individuals are evaluated by simulating moves ahead using a forward model. From the current state of the games, all actions (genes of the individual) are executed in order, until a terminal state or the length of the individual is reached. The state reached at that point is then evaluated with a heuristic function and the value assigned as the fitness of the individual. The most basic form this algorithm can take is that of a Random Mutation Hill Climber (74), where the population size is only 1, using the mutation operator as the only way to navigate through the search space.

Samothrakis et al. (75) compare two variations of the Rolling Horizon setting of EAs in a number of continuous environments, including a Lunar Lander game. The first algorithm uses a covariance matrix, while the second employs a value optimisation algorithm. The Rolling Horizon refers to evolving plans of actions and, at each game step, executing the first action that appears to be the best at present, while starting fresh and creating a new plan for the next move, sequentially increasing the "horizon". Their research suggests EAs to be viable algorithms in general environments, and that a deeper exploration should be performed with an emphasis on heuristic improvement.

Regarding two-player RHEA implementations for multi-player real-time games, Liu et al. suggest an expansion of the single-player algorithm to two players (and possibly multiple others), by employing a co-evolution model of one population for each different player and using the actions from both to simulate possible future states and improve the action sequences (73). Their algorithm is tested on a two-player Space Battle game and returns favourable results.

The first step in studying RHEA algorithms is, however, understanding the effect of the core parameters, the population size and individual length. Not many studies have focused on the implications of these core parameters, nor justified their choices or tuning algorithms, which, as a consequence, may include bias in their results. N. Justesen et al. (76) used online evolution for action decision in Hero Academy, a game in which each player counts on multiple units to move in a single turn, presenting a branching factor of a million actions. In this study, groups of actions are evolved for a single turn, to be performed by up to 6 different units. With a fixed population of 100 individuals, the authors show that online evolution is able to beat MCTS and other greedy methods.

Other authors do not necessarily highlight the reasoning behind their parameter choices either. Horn et al. (54), for example, who depict interesting hybrids between RHEA and MCTS, appear to change the configurations of their algorithms from one test to the next, which causes possible inconsistencies to appear and the results reported may not be necessarily due to the technique used, but the variation in parameters. This is one aspect my work wishes to improve, by keeping studies comparable and grounded in incrementally complex experiments.

2.3.1 RHEA improvements

When it comes to improving upon the baseline algorithm, a good place to start is looking at the algorithm structure and how evolution could be made more effective in the short computation time allowed by real-time games. This could mean creating hybrids, algorithms that take advantage of the strengths of several approaches and combine them for new interesting behaviour (54).

Several instances have been produced, with evolution being integrated into the simulation phase of Monte Carlo Tree Search (49), as well as using an EA to tune MCTS parameters (77). Recent work attempts to look at the problem the opposite way: incorporating trees into EAs or using a bandit approach, similar to the selection step in MCTS, for guiding evolution (78).

RHEA and MCTS have been combined in different ways in the literature as well, in order to make use of both of their strengths and mitigate their weaknesses at the same time, at the cost of increased computational effort. Vasquez et al. (79) combine them in 2 variants: one which runs the algorithms alternatively at every game tick to decide on the next action; and one which runs both at every game tick and chooses the action leading to the best reported result. Both of these hybrids lead to better performance across several games. Anderson et al. explore a similar concept at a larger scale, by building complex ensemble systems (80): these run several algorithms at every game tick and use the outputs returned by all to come to a consensus and play at a higher level than any of the individual parts. Baier et al. integrate evolutionary concepts into an MCTS tree, which holds sets of actions into its nodes instead of single actions, the algorithm named EMCTS (81; 82). Each action set includes one action per unit the player controls, applicable in multi-agent games (where the player controls multiple units and must decide what each must do in a player's turn). The nodes in the tree are then connected by mutation operators, as the "actions" taken to navigate the tree. This algorithm thus keeps a history of all action sets tried in a game tick, increasing the memory required for the run. However, it is able to then make use of the entire history in selecting the next action set to mutate and obtains great results in multi-agent games by considering entire turns for one player in the nodes of MCTS.

The population initialisation is the one thing that all Evolutionary Algorithms have in common, regardless of any additional features or the actual evolutionary techniques used. There have been several attempts at exploring this particular improvement. Kazimipour et al. (83) review different methods present in literature and categorise them according to various factors: randomness, compositionality and generality. They identified several techniques which would work in a general environment; however, they suggest that these methods are computationally expensive, therefore not translating well to real-time games, for example, which is the domain my work focuses on.

In addition, Kim et al. (84) analyse the effects of initialising an EA population using an optimal solution determined by a Temporal Difference Learning algorithm in the game Othello. This addition appears to lead to a significant improvement in performance and future work in the area is encouraged.

The issue with initialising the population with pseudo-random numbers is raised by Maaranen et al. (85), who instead propose a quasi-random sequence method meant to obtain more evenly distributed points in a multi-individual population, in order to better explore the search space. This technique is applied to a genetic algorithm and it is tested on 52 global optimisation problems. Their results are promising, suggesting a higher level of performance over the traditional initialisation method.

Tobias Benecke goes one step further to analyse in-depth the impact of the initial population throughout the evolutionary process, proposing several metrics that track the progress of the population from one generation to the next, as well as measuring the end value of the initial population (86). This work aims to better characterise the behaviour of the algorithm and inform future work in the area of population initialisation.

2.3.2 RHEA hybrids

Perez et al. (69) look at the recommendation policy part of the algorithm and they keep a statistical tree alongside the evolutionary process, in order to record statistics about the actions while evaluating individuals and select the action with the highest value averaged during the evolution. Thus they make use of intermediate states and not only look at the final population obtained. This method is most effective in noisy environments as an alternative to re-sampling, which would be more expensive. Furthermore, they keep the tree from one game step to the next, by using the child selected at the end of the evolution as the new root of the tree in the following step. Their promising results motivated the use of both of these methods in my studies, by combining the stats tree with a shift buffer for the same effect. However, it is worth noting that the authors add a pheromone-based heuristic to their algorithms, which may impact their findings.

A compelling and novel addition to evolutionary algorithms is that of multi-armed bandits applied as a mutation operator to better balance between exploration and exploitation. There is extensive literature on the multi-armed bandit problem (87) and various solutions to it. One possibility is using an Upper Confidence Bound (UCB) method. Powley et al. (88) look at using UCB in Monte Carlo Tree Search as both the tree policy and the simulation policy. When tested on three different problems, two card games (“Dou Di Zhou” and “Hearts”) and a board game (“Lord of the Rings: The Confrontation”), its performance is shown to consistently be at a high level.

A bandit-based mutation system was described in (78; 89). Liu et al. compare this mutation method with the Random Mutation Hill Climber (RMHC) on two simple problems and their results suggest that bandit-based mutation is especially effective in cases where individual evaluation is expensive, therefore applicable to the problems described in this thesis.

Horn et al. (54) look at two different MCTS-RHEA hybrids. In the first method (*EAroll*), Monte Carlo simulations are used at the end of the evaluation of one RHEA individual with a limited depth, the resulting value being averaged with the genome evaluation to determine its fitness. The second variant (*EAaltActions*) uses both RHEA and MCTS to individually search for distinct solutions, the two final recommendations being evaluated and the best one chosen for execution. They analyse the performance of both algorithms on 20 games of the GVGAI corpus (but a different 20 than in my work) and *EAroll* appears to be significantly better than vanilla RHEA and dominating the games used in their experiments.

2.3.3 Population diversity

Some work in literature looked at addressing the problem of solution space exploration. The aim here is to move away from traditional evolution towards objectives (which would often result in several individuals behaving similarly or get stuck in local optima) and instead attempt to obtain a population of individuals as diverse as possible.

Mahfoud analyses several methods for population diversity, while focusing on the particular subset called niching (90). The study suggests that algorithm convergence to local optima is due to three factors: selection noise (random choices between individuals with the same fitness forces the removal of good

individuals), selection pressure (low fitness individuals disappearing from a finite population during evolution) and operator disruption (crossover and mutation might worsen good solutions). Therefore some works focused on reducing the selection noise (stochastic remainder selection with replacement (91) and stochastic universal selection (92)), but Mahfoud’s analysis shows them to not address the problem well enough. Some direct infusion methods (such as increasing mutation rates) are also shown to be ineffective at obtaining useful diversity. Mauldin’s (93) uniqueness-assurance technique mutates new individuals until they are sufficiently different before adding them to the population; although this results in diverse alleles, it does not lead to a meaningful exploration of the solution space.

More details on many more population diversity methods are presented by Gupta and Ghafir in (94), and Gabor presents two different options for integrating population diversity in algorithms very similar to RHEA in (95).

Novelty search is a different approach to the same issue proposed by Lehman and Stanley (96) and applied to several tasks that can be easily translated to the GVGA environment. They replace the objective function depicting the fitness of an individual with a novelty function. Therefore the fitness represents how unique an individual is, instead of how good the solution is to the problem. Their method attempts to avoid deception and leave evolution open-ended, without a clear definition of what a good solution implies, thus not limiting the search or accidentally directing to dead ends. They obtain good results on biped walking and maze navigation problems, their technique outperforming objective-based algorithms. Additionally, they suggest this method might be more general, allowing for multiple diverse solutions to be obtained, the user able to select which one to use in a flexible manner.

An extension to this is work by Gravina et al. (97) which looks at combining novelty search with a surprise metric, leading to a wider exploration of the solution space and outperforming both of the components on robot navigation tasks. Surprise search was first introduced in (98) and comprises of two processes: identification of the expected behaviour, and calculation of the deviation from the expected. Experiments on navigation tasks show this search method to be faster and more robust than novelty or objective search, with overall performance comparable to novelty search. The speed of the algorithm, in particular, lends it naturally to the real-time environments explored in this thesis.

These same concepts can be extended to Quality Diversity algorithms, such as Map-Elites (99), which aim to explore a wide variety of high-performing solutions to a problem. As such, Gravina et al. (100) use surprise to drive the search and show improved performance on several deceptive maze navigation tasks. MAP-Elites was applied in GVGA specifically in the context of procedural content generated, to create diverse set of levels showcasing a variety of mechanics in isolation and in combination (101). A similar approach can be extended to better observe the parameter space of RHEA and better classify its behaviour on the variety of environments tested.

2.3.4 Macro actions

Another solution proposed for the computation time limit and level exploration issues is using macro actions (high-level actions which contain information on strategies or tactics, rather than the granular input required by games). A simple form of this is frame-skipping, which involves returning actions only every N game ticks and using the time in-between for longer decision-making. A variant of this which repeats the action chosen N times was tried in GVGA physics-based games with mixed results (26).

The idea of using macro actions (from a simple action repetition to the design of more complex variants) has been used multiple times when the size of the state space makes search a very costly task. Pioneered in the early days of Reinforcement Learning (102), macro actions have been used in Real-Time Strategy games like Wargus (103) (applying them to simultaneous moves of variable duration), the artificial game P-Game (104) and the card game Dou Di Zhu (105), where the authors split actions in several consecutive decisions in order to reduce the branching factor at the expense of tree depth.

Last but not least, macro actions have also been used in the PTSP game mentioned above (106; 7), both for tree search and evolutionary techniques, as explored in this thesis. In their work, the authors propose a simple repetition of actions as a way to coarse the search and provide a longer thinking time for the agent in this real-time game. Results showed that there was an optimal amount of times an action should be repeated to maximise performance: shorter macro actions would not allow for an effective exploration of the search space, while longer ones did not provide the agent with enough precision to navigate through the maze efficiently. A similar approach has been followed in this work when applying this concept to the new GVGA games, aiming to investigate if the findings there extrapolate to multiple games at once.

2.4 Visual Game Analysis

There is extensive literature on game data mining and gameplay data analysis and visualisation. Wallner and Kriglstein (107) give a large overview of the state-of-the-art in their survey, identifying various classifications of the works, based on applications, target audience and representation type. The applications of visual data analysis are quite extensive as well, ranging from game design (108) to analysing player behaviour (109) and understanding player movement (110). Additionally, gameplay data analysis can be used to identify cheating in various games; one form of cheating is botting, where players use AI agents to play the game instead. Mitterhofer et al. (111) used logs of character movement to identify botting.

Several tools have been created for visual data analysis as well. One example is Scelight (112) for StarCraft II, which provides various statistical information (e.g. game length, speed and other player specific information) and diagrams showing, for example, which and how many actions are performed every minute. Another tool is “Echo”, designed for DotA 2 (113) and launched at ESL One Hamburg. “Echo” gathers statistics about matches played and overlays visual information on top of the game currently played for a more detailed analysis of gameplay, in order to enhance the viewing experience.

However, there are not many stand-alone visualisation tools or in-depth analysis of inner-workings of AI algorithms playing games. Some projects exist which look directly at tools for visualising Monte Carlo Tree Search (MCTS) within specific games, such as Connect 4. Volz et al. (114) proposed a set of algorithm and game measures and prototyped a visualisation tool for general video game playing. Simon Lucas (115) promotes easily accessible games in a web browser with handy visualisations included. The work presented in this thesis extends this proposal by implementing an extended set of measures and formalising a visualisation tool that is decoupled from GVGA.

2.5 Win Prediction

There is extensive literature on extracting various AI gameplay measures. Traditionally, these methods are predominantly used in the area of Procedural Content Generation in order to assess the quality of a level or game created automatically.

Liapis et al. (116) create models of player types called “procedural personas”, which then they use to automatically generate levels of a roguelike puzzle game. For this purpose, they identify several features that the evolved agents will focus on: the number of monsters they kill, the number of treasures collected or reaching the exit of the level. Using these different personas to automatically play-test levels, the authors are able to generate interesting levels which highlight agent strengths.

Some researchers focus more on the area of human-computer interaction and how measures extracted from gameplay can be used in predicting various aspects characterising automatically generated games (engagement, frustration and challenge in (117); or human enjoyment when playing against different ghost teams in the game Ms Pac-Man (118)). The content and gameplay features highlighted by Shaker et al. (117) in the platformer game Super Mario Bros are directly applicable to AI gameplay as well as humans: number of enemies, number and width of gaps in the level, enemies placement, boxes, power-ups and events triggered during play.

Sombat et al. (118) analyse human enjoyment when playing against different ghost teams in the game Ms Pac-Man. They show that they are able to classify ghost team enjoyment and identify several measures to this extent, regarding the ghost team’s play style: level of challenge (time taken for the ghosts to capture the player), level of behaviour diversity (variations in score obtained by the same player in multiple games) and level of spatial diversity (how much of the level the ghosts explore). The authors use these three measures to quantify the perceived level of entertainment of a Ms Pac-Man game.

Isaksen et al. (119) define several metrics characterising dice games: win bias (the difference between the probabilities of player A winning a dice battle and player A losing the battle), tie percentage (the probability of a tie in a given battle) and closeness (how much the result of the battle centred around a tie). Volz et al. (120) evaluate how close the game ended as well in the card game Top Trumps with the objective of automatically balancing the game.

Several authors look at skill depth as a measure for good games, a technique applied to various problems. Liu et al. evolve a Space Battle game with a Multi-Armed Bandit version of a Random Mutation Hill Climber, using the difference between a Monte Carlo Tree Search (MCTS) player and a random shooting one as their measure of depth (89). Kunanusont et al. apply the concept of skill depth for evolving a similar Space Battle game using the N-Tuple Bandit EA in (31), while employing three different agents for comparison (MCTS, One Step Look Ahead and random). Perez et al. automatically generate maps for the Physical Travelling Salesman Problem in (121), assessing the skill depth of each by again having a novice, medium and advanced player test each generated map and calculating the difference in performance (defined as the time taken to complete the task), aiming to maximise the gaps between players.

Nevertheless, for all of the above mentioned cases, two different games need to be played in order for the measures to be computed, which is not feasible in planning scenarios when an AI agent is seeing a game for the first (and only) time.

However, the features explored by these authors are game-specific and applying these methods to other domains is not straightforward. When designing games, Browne and Maire (122) looked at 57 different criteria in judging an evolved game split into 3 categories, intrinsic, viability and quality. The authors use general game-playing agents to test their games, which are written in the Ludi Game Description Language. Most of the quality features analysed are resultant from AI game-play, such as depth, drama, decisiveness or uncertainty. Some of these metrics, where possible to translate to single-player games, were adapted for our study.

Several works move away from the area of PCG and instead focus on extracting measures of player behaviour to specifically tune game-playing agents or perform a deeper performance analysis than the typical win rate investigation. Khalifa et al. (123) used features from human gameplay data to tune a human-like Monte Carlo Tree Search (MCTS) player. Their features mostly focused on actions, such as action repetition, change frequency or pauses, with an additional map exploration metric. The authors applied the features extracted from human data to tune a Monte Carlo Tree Search agent on 3 different games in the General Video Game AI framework (GVGAI), with mixed results.

More general measures for better analysis are depicted by Volz et al. in (114). Their prototype implementing the measures for live game-playing agent analysis also uses the GVGAI framework, allowing for a general application of the method on several different games. Some of these metrics, such as decisiveness or action entropy, were included in this study, excluding multi-player or comparison metrics.

Some researchers use such metrics for machine learning tasks. For example, Bontrager et al. (1) cluster the games in the GVGAI framework based on the performance of several agents submitted to the corresponding competition. In this case, the performance of an agent is simply characterised by the win rate, which is shown to differ between the players. The authors signify that some agents possess skills useful in certain tasks, while other agents lack or make up for them in different ways.

Mendes et al. (124) used this conclusion to construct a hyper-heuristic agent. The authors extracted several game features (number and type of NPCs, resources available, map dimensions and number and types of other sprites) and used a classification method to determine which AI agents, selected from a subset of GVGAI competition entries, achieve highest win rates when specific game features or combination of features are present in a new game tested. The algorithm then decides which agent to query for a solution depending on the recommendation of the classifier (a Support Vector Machine and a Decision Tree). The agent selected will play the entire game with no changes.

A similar approach was employed by Horn et al. in (54) for AI hybrid evaluation (excluding the hyper-heuristic construction step). They propose a game difficulty estimation scheme based on game features (NPC types, puzzle elements, path-finding requirements or traps). These are arguably more open to human bias, as each metric is evaluated manually. Although the game difficulty features identified do not correspond to agent win rates, the authors carry out an analysis which gives a deeper insight into reasons for agent performance levels.

These works are, however, based on game features as defined by human knowledge on the existing data set. My work in (20) proposes a game win predictor based solely on agent experiences, aiming to remove potential human bias resultant from designing features seen on known games.

2.6 Optimisation

In any domain where control parameters exist to change the behaviour of an algorithm, finding the correct settings for the parameters becomes a very important problem. For games, in general, choosing good parameters could affect the flow and balance of a game and make the difference between a great player experience and an unplayable game. For game-playing algorithms, well chosen parameters could lead to high performance and interesting behaviours, whereas other combinations might not work at all. For instance, increasing the population size of an evolutionary algorithm has been shown to increase robustness to noise (125). Automatic optimisation is one solution to this problem, concerned with automatically testing various configurations and choosing those hyper-parameters that lead to the best result, as defined for the given problem.

In the area of Game AI, evolutionary algorithms are a common choice for automatic optimisation algorithms (126), being employed in several areas, from game tuning to agent or heuristic optimisation. However, the main problem many of these algorithms deal with is that of expensive configuration tests. As an example, for agent parameter optimisation (our focus in some experiments presented in this thesis), a parameter configuration test consists of running the agent with the given parameters in one or several games, once or multiple times for more accurate approximation of the agent's performance in that setting. Although

most games used for AI benchmarking are designed to be fast to run, doing so enough times to explore the parameter space of the agent efficiently can be very expensive still. Even more so, if such optimisation is considered for larger problems and complex domains that are maybe not optimised themselves for speed, could pose even greater problems and lead to very long run times before a *good enough* configuration can be found. Note that we use the term *good enough* when referring to such optimisation problems, as the optimal is rarely an option or even a necessity for most cases.

Taking these aspects into account, model-based algorithms are often a more popular choice for computationally expensive domains (127; 128). These approaches store information and statistics internally to be able to *approximate* the value of some solutions, and choose which ones should be more thoroughly evaluated in the real domain, in order to make efficient use of the computation budget given and shorten run times (129).

A different common problem that optimisation methods need to deal with is the noise in the problem domain: in the case of game-playing agent optimisation, this can arise from both the algorithm playing the game, and the game itself. Re-sampling can be an option (i.e. running the evaluation multiple times and using a combination of all values obtained, instead of a single value), although this increases computation time and the budget can become an issue again; without re-sampling, the values obtained for the solutions evaluated might not be indicative of the actual quality of the solution, and therefore the algorithm would explore sub-optimal parts of the search space and recommend solutions that are not consistently *good enough*. A naive approach is using a simple bandit algorithm, Upper Confidence Bound (UCB) (130), to balance between re-evaluating solutions thought to be good to gain more accurate statistics about their true value, and exploring new or less often visited settings. This approach was shown to obtain good results in some cases (131), but it does not account for potential dependencies between the solution dimensions (in this case, dependencies between algorithm parameters).

Approaches based on Hyper-Heuristics have been used in GVGAI by Mendes et al. (132). These approaches train the agent offline to recognise the best strategy for the game at hand from a portfolio. When the agent has to play a new game, it uses the trained mechanism to select the best strategy depending on some of the game’s features. Experiments have shown that these approaches are promising and are able to outperform agents based on standard algorithms.

The N-Tuple Bandit Evolutionary Algorithm (NTBEA) was first introduced in 2017 with an application for automatic game optimisation (31) and later formalised in (133) with an alternative application for game-playing agent parameter optimisation. NTBEA combines evolutionary algorithms with n-tuple systems for its internal model, and bandit-based sampling, following on from the bandit-based Random Mutation Hill Climber proposed previously (78). NTBEA was designed for sample-efficient hyper-parameter optimisation in noisy environments and shows competitive results compared to other state-of-the-art methods (34).

While such works are promising in optimising RHEA’s parameters offline, attention to online adaptation of game-playing agents has increased recently, also due to the increased interest in general game playing (GGP), where the optimal configuration has to be learned online. One of the first attempts at adapting GGP agents online concerned the adaptation of the playing strategy (134). An online mechanism decides how to allocate available samples to evaluate a portfolio of strategies for the agent, and find the one that is best suited for the current game. Results on abstract games showed that a strategy based on a MAB that tries to allocate the highest number of samples to the best playing strategy while still exploring other strategies performs best.

Similarly, an online mechanism was used to find the best parameter configuration for an MCTS agent depending on the game being played (32; 131). In this approach, the result of each MCTS simulation is used to evaluate the quality of the parameter values that control the simulation. Moreover, statistics collected so far on the performance of parameter values are used to choose which values to evaluate next. This approach has been tested both on classic board games (32) and on arcade-style video games (131). On board games, it had positive results, especially when the number of tuned parameters is small. On video games, online tuning was shown to be harder, nevertheless promising for a few of them. Moreover, in GVGP, randomisation of MCTS parameters online was tested and shown to perform closely to online parameter tuning, provided that parameter values are selected from a reasonable subset of feasible values (135). Work by Stefan Bussemaker takes these ideas forward and combines them with multi-objective and macro actions modifications for superior performance (136).

Chapter 3

Rolling Horizon Evolutionary Algorithm

This chapter will describe the basics of the algorithm at the base of the work presented, as well as all modifications and hyper-parameters studied in this thesis.

3.1 Vanilla RHEA

RHEA utilises Evolutionary Algorithms (EA) to evolve an in-game sequence of actions at every game tick, with restricted computation time per execution. This subsection will describe the baseline algorithm, often referred to as *vanilla*; modifications applied are detailed in Section 3.2.

In this application of EAs for game-playing, the genotype is described as a vector of integers of length L (individual length, or sequence length, or horizon), where each integer a is in the range $[0, N)$, with N being the maximum number of actions in a given game state S . This translates to a phenotype as a sequence of actions played in the game starting from state S_0 , or, in other words, the behaviour of the player. In our implementation, the EA considers all individuals, and genes in the individuals, as legal and feasible (this is a potential limitation, as redundant actions are often considered; future work will look at adapting methods to avoid this, such as (137; 138)). In order to evaluate an individual in this context (see Figure 3.2), RHEA uses the forward model of the game, an internal model of the world, to simulate through the actions, one at a time. The game state reached at the end is then evaluated with a heuristic function h and this value becomes the fitness of the individual: therefore, we are evolving action sequences which lead to the best game outcome, limited to the exploration range L . The function h is always kept to a generic form throughout the experiments; this aims to maximise the game score, while favouring wins and discouraging losses, see Equation 3.1 (s is the game state being evaluated and r is the *reward* obtained from the game (or in-game *score*), normalised in $(0, 1)$).

$$h(s) = \begin{cases} 1 & , \text{win} \\ 0 & , \text{lose} \\ r & , \text{otherwise} \end{cases} \quad (3.1)$$

Alternative heuristics were explored by Guerrero et al. (139), with a view of their application for game testing (140), i.e. agents have different goals to achieve such as exploring more of the level space or gaining knowledge about sprite interactions. Santos and Bernardino also propose the use of advanced heuristics to better guide the agent's search process (141). However, the purpose of the work described in this thesis is to investigate the algorithm itself and keeping a consistent focus on solving the given problems between all studies described helps highlight the benefit of various modifications, parameter choices and enhancements.

Using this method to evaluate individuals, the vanilla algorithm follows a typical EA process (see Figure 3.1). It begins by initialising a population of P individuals of length L at random and evaluates them. At every generation, while budget is still available, it promotes E individuals directly to the next generation through elitism. It then generates P offspring by repeatedly selecting parents through tournament selection, crosses them with uniform crossover to create a child, and mutates the child through uniform mutation before adding it to the pool of offspring. The best $P - E$ individuals from both parents and offspring pools are added to the next generation and the process repeats. See Algorithm 1 for pseudocode of the process repeated at every game step.

Typically, a budget of 40ms per game tick is given to the algorithm for real-time decision-making. As the agents have a limited amount of time to make decisions in real-time games, one of the popular methods

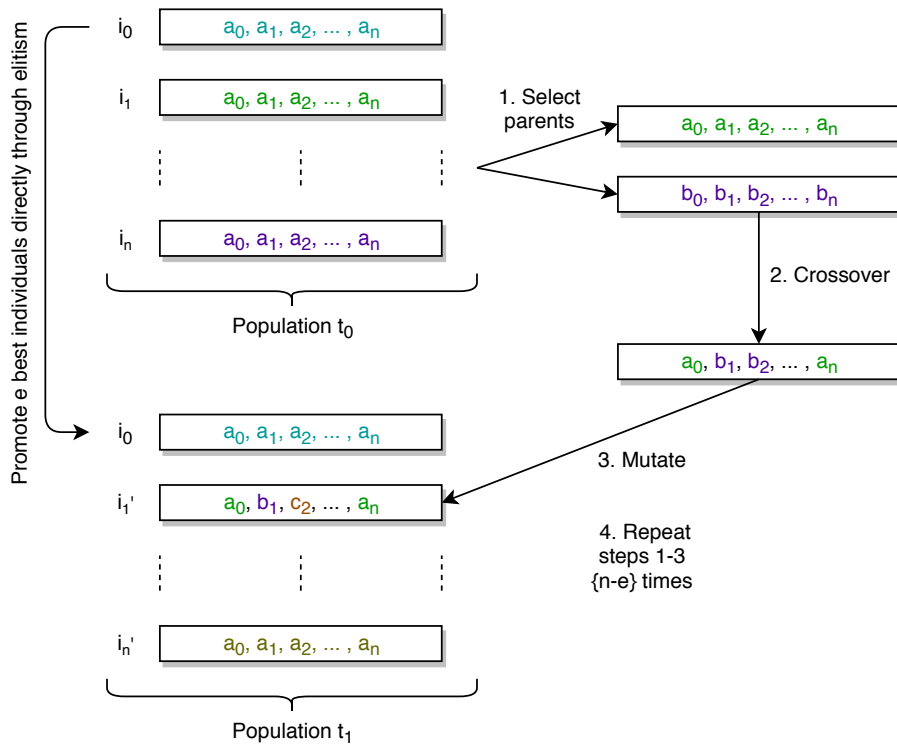


Figure 3.1: Rolling Horizon Evolutionary Algorithm cycle, repeated for several generations.

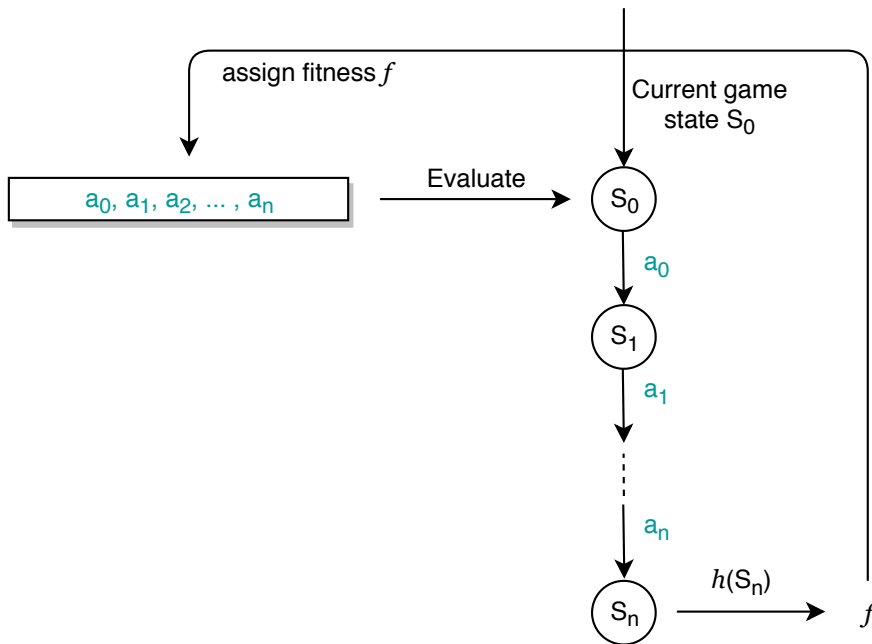


Figure 3.2: Evaluation of one individual in the Rolling Horizon Evolutionary Algorithm, given the current game state.

Algorithm 1 Vanilla Rolling Horizon Evolutionary Algorithm

```
1: procedure MAIN( $s_t, budget$ ) ▷ Game state at time step  $t$  and FM budget available
2:    $k \leftarrow 0$ 
3:    $P_k \leftarrow initializePop(budget)$ 
4:   while  $budget \neq 0$  do
5:      $P_{k+1}.add(\text{first } E \text{ individuals from } P_k)$ 
6:      $O \leftarrow \text{new array}$ 
7:     for  $j = 0 : pop\_size$  do
8:        $p_1, p_2 \leftarrow selectParents(P_k)$  ▷ Tournament selection
9:        $I' \leftarrow cross(p_1, p_2)$  ▷ Uniform crossover
10:       $I' \leftarrow mutate(I')$  ▷ Uniform mutation
11:       $evaluate(I', s_t, budget)$ 
12:       $O[j] \leftarrow I'$ 
13:       $pool \leftarrow P_k + O$ 
14:       $sort(pool)$  ▷ Sort in ascending order based on fitness
15:       $P_{k+1}.add(\text{first } P - E \text{ individuals from } pool)$ 
16:       $k \leftarrow k + 1$ 
17:       $best \leftarrow \text{first individual from } P_k$ 
18:       $a_t \leftarrow best[0]$ 
19:      return  $a_t$  ▷ Action to play at time step  $t$ 
20:
21: procedure INITIALIZEPOP( $s_t, n\_actions, budget$ )
22:    $P \leftarrow \text{new array}$ 
23:   for  $k = 0 : pop\_size$  do
24:      $I \leftarrow \text{new array}$ 
25:     for  $j = 0 : ind\_length$  do
26:        $I[j] \leftarrow random(0, n\_actions)$ 
27:        $evaluate(I, s_t, budget)$ 
28:        $P[k] \leftarrow I$ 
29:   return  $P$  ▷ Initial population
30:
31: procedure EVALUATE( $I, s_t, budget$ )
32:   for  $j = 0 : ind\_length$  do ▷ Iterate through actions in individual
33:      $s_{t+j+1} \leftarrow s_{t+j}.advance(I[j])$  ▷ Use FM to advance game state given action in individual
34:      $budget \leftarrow budget - 1$ 
35:      $f \leftarrow h(s_{t+ind\_length})$  ▷ Use Equation 3.1
36:   return  $f$  ▷ Fitness of individual  $I$ 
```

in the literature consists of generating only one new individual at each generation, therefore making it possible to interrupt the process at any point. The most basic form this algorithm can take is that of a Random Mutation Hill Climber (74), where the population size is only 1, using the mutation operator as the only way to navigate through the search space.

3.2 Modifications and Parameters

This section describes all the hyper-parameters of the algorithm, including hybrids and modifications on/off flags. All algorithm parameters are presented in Table 3.1. Several dependent parameters are highlighted in the table: these are parameters that would not impact the phenotype without specific values taken by other parameters, as detailed below. Default choices for parameters, which would form the vanilla algorithm configurations, are also highlighted in the text: we note that this is the baseline for all experiments in the thesis, kept to its most simple form for consistency and ease of comparison across different experimental setups.

3.2.1 Genetic Operators

There are three main genetic operators used by the evolutionary algorithm in RHEA: crossover, selection and mutation. In our implementation, selection is only used to select parents for offspring, subsequent

Table 3.1: Parameter Search Space. Parameters noted with † are dependent on the value of other parameters. Last column shows default values for each parameter; unless otherwise specified, the parameters in all experiments are fixed to this value when not varied for studies.

Idx	Parameter	Type	Value Range	Default Value
p_0	Population Size	int	$[1 - \infty)$	10
p_1	Individual Length	int	$[1 - \infty)$	15
p_2	Dynamic Depth	bool	{False, True}	False
p_3	Offspring Count	int	$[1 - \infty)$	p_0
p_4	Number Elites	int	$[1 - p_0)$	1
p_5	Initialisation Type	categ.	{Random, 1SLA, MCTS}	Random
p_6	Genetic Operator	categ.	{Crossover Only, Mutation Only, Crossover + Mutation}	Crossover + Mutation
p_7	Selection Type †(p_6)	categ.	{Rank, Tournament, Roulette}	Tournament
p_8	Crossover Type †(p_6)	categ.	{Uniform, 1-point, 2-point}	Uniform
p_9	Mutation Type †(p_6)	categ.	{Uniform, 1-Bit, 2-Bits, Softmax, Diversity(p_{20})}	Uniform
p_{10}	Fitness Assignment	categ.	{Last, Delta, Average, Min, Max, Discount}	Last
p_{11}	Diversity Weight	double	$[0.0 - 1.0]$	0.0
p_{12}	Frame-skip	int	$[0 - game\ length)$	0
p_{13}	Frame-skip Type †(p_{12})	categ.	{Repeat, Null, Random, Sequence}	-
p_{14}	Shift Buffer	bool	{False, True}	False
p_{15}	Shift Discount †(p_{14})	double	$[0.0 - 1.0]$	-
p_{16}	MC Rollouts Length	double	$[0.0 - \infty)$	0.0
p_{17}	MC Rollouts Repeat †(p_{16})	int	$[1 - \infty)$	-
p_{18}	Bandit-based mutation	bool	{False, True}	False
p_{19}	Statistical tree	bool	{False, True}	False
p_{20}	Diversity mutation †(p_9)	bool	{False, True}	False
p_{21}	Diversity fitness	bool	{False, True}	False
p_{22}	Diversity type †(p_{20}, p_{21})	categ.	{Genotypic, Phenotypic}	-

generations being formed directly with the best individuals from the current generation (with no further selection being applied). These three genetic operators each have several implementation options, as discussed below. A hyper-parameter controls which operators should be applied, with options of only using crossover (and selection), only using mutation, or using all three to first obtain an offspring from crossover, and then mutate it as well. It is worth noting that the operator type parameters detailed below are dependent on the choice of genetic operator: changing the mutation type would not have any effect on the phenotype if no mutation is used in the algorithm, and similarly for crossover and selection. The default choice for this parameter is utilising both crossover and mutation.

Selection.

Three types of selection are available in the system: tournament, roulette and rank. All options describe the process for selecting one individual, which is then repeated again for the second parent. See Figure 3.3 for a visual representation of the different options. The default choice for this parameter is tournament selection.

Tournament selection picks a percentage of the population ($t = 50\%$ by default) at random and then chooses the best individuals from these to reproduce.

Roulette selection chooses individuals with probabilities directly proportional to their fitness (therefore, higher fitness individuals have a higher chance of being selected). This can pose a high selection pressure if the fitness of individuals have large differences between them.

Rank selection first assigns inverse-ranks to all individuals in the population according to their fitness (the lowest fitness individual would have rank 1, second lowest rank 2 etc.) and then chooses individuals with probabilities equal to their rank (therefore, higher fitness individuals have a higher chance of being selected, but the selection pressure is reduced by minimising the differences in fitness).

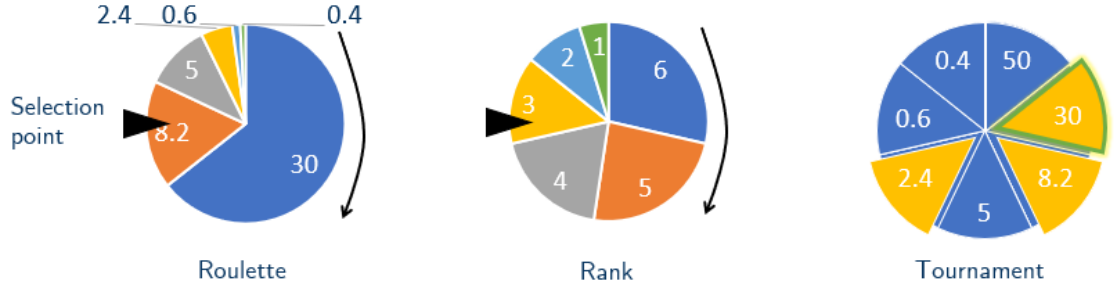


Figure 3.3: Selection options: roulette, rank and tournament. The first two can be seen as wheels spinning and the slice (or individual) next to the selection point is the one chosen; they are both representations of the same population. In the tournament visualisation, the yellow individuals were randomly chosen for the tournament, and the one highlighted green had the highest fitness amongst them and was selected.

Crossover.

Two types of crossover are available in the system: uniform and n -point. Although all options could produce two offspring, depending on which of the two parents is considered ‘first’, only one individual is produced from each pairing in this implementation. The default choice for this parameter is uniform crossover.

Uniform crossover selects genes from either of the parents with equal probability.

n -point crossover randomly selects n points along the individuals which would split all individuals in subsections, the offspring being formed then by alternatively choosing subsections of genes from the parents; we use 1 and 2 as possible values for n , leading to three total values for the crossover parameter.

Mutation.

Three types of mutation are available in the system: uniform, Softmax and n -bits. The default choice for this parameter is uniform mutation.

Uniform mutation assigns each gene an equal probability of mutation ($m = 1/L$, where L is the individual length) and picks a new different value for the genes mutating, the values also being chosen uniformly at random.

Softmax mutation uses the Softmax equation (see Equation 3.2) to bias mutation towards the beginning of the individual, which causes the largest perturbation in the action sequence (changing any gene in the individual, in this context, also changes the meaning of all subsequent genes - therefore changes in the beginning of the genome have the largest impact in the phenotype).

n -bit mutation chooses n genes uniformly at random to mutate to a new and different random value; we use 1 and 2 as possible values for n , leading to a total of four values for the mutation parameter.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (3.2)$$

3.2.2 Fitness Assignment

A key part of the algorithm is deciding how to calculate the fitness resultant from an individual’s phenotype. Individuals are sequences of actions which are executed, in order, during evaluation, resulting in a sequence of game states that the AI player traverses. If all game states traversed are evaluated with a heuristic function h (see Equation 3.1), then we obtain an array of values F .

This array can then be translated to a fitness value for the individual being evaluated in different ways: keeping only the value of the last game state reached, $F[L - 1]$ (prioritises the direct outcome of the action sequence); keeping the difference between the value of the last state and the value of the first state, $\Delta(F[L - 1], F[0])$ (prioritises state improvement); keeping the average of all game state values, \bar{F} (prioritises consistency throughout the action sequence); keeping the minimum value, $\min(F)$ (pessimistic model, could miss very good outcomes but avoid bad ones); keeping the maximum value, $\max(F)$ (optimistic model, could miss very bad outcomes, but catches good ones); or keeping a discounted sum of all values: $\sum_{i=0}^L F[i] \times \gamma^i$, where $\gamma = 0.9$ (prioritises immediate rewards). The default choice for this parameter is assigning the value of the last game state reached, $F[L - 1]$, which is most robust to game-specific reward systems fluctuations.

3.2.3 Initialisation

In the vanilla version, the algorithm is initialised with random individuals (all genes in all individuals are picked uniformly at random from all possible values). We add two different options for obtaining the initial population of individuals.

1SLA.

The One Step Look Ahead (1SLA) algorithm performs an exhaustive search of all possible options for a gene and picks the action which leads to the highest value for the following game state (using the same heuristic function h depicted in Equation 3.1). To form an individual, this process is followed for each gene, an action is chosen, the game state is advanced with the chosen action and the process is repeated again for the next gene until an action sequence of sufficient length is generated. If the end of the game is reached during the creation of an individual, the individual is padded with random actions until it meets the required length.

For initialisation of a RHEA population, the first individual is created with the 1SLA algorithm and the rest are mutations of the first (mutation type used is the same given by the mutation type hyper-parameter). Giving a greedy approach, this reduces the randomness of the initial population and begins search from a local optimum. See Figure 3.4 for a visualisation of this process.

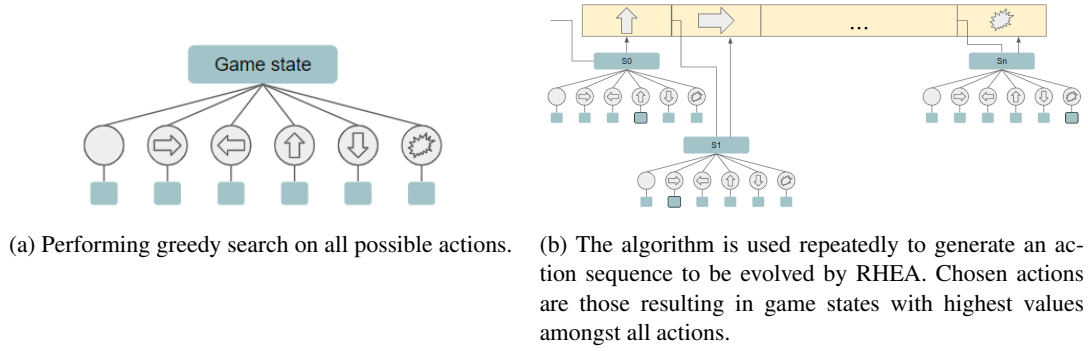


Figure 3.4: One Step Look Ahead (1SLA).

MCTS.

Monte Carlo Tree Search (MCTS, see Figure 3.5) iteratively builds a search tree by following four steps. First, it selects a node in the tree to expand, using the UCB1 formula (see Equation 3.3, where: constant $C = \sqrt{2}$, a is the chosen action from the set of possible actions $A(s)$, s is the current game state, $Q(s, a)$ is the value of choosing action a from state s , $N(s)$ is the number of times state s has been visited and $N(s, a)$ is the number of times state s has been visited and action a was chosen next). It then adds a new child of the selected node to the tree and runs a Monte Carlo simulation from the node (a sequence of random actions up to a maximum tree depth L). Finally, it updates the statistics ($N(s)$, $N(s, a)$ and $Q(s, a)$) of all nodes traversed during the iteration with the value given by the heuristic h for the final game state reached after Monte Carlo simulations. This tree grows asymmetrically as MCTS balances between exploration of uncertain actions and exploitation of seemingly good actions.

It is worth noting that this same MCTS agent is used in several other experiments. As this implementation does not store game states in the nodes of the tree it builds, we consider it an Open Loop approach.

For initialisation of a RHEA population, MCTS 2.2 is run for half the budget and the first individual is selected by greedily traversing the tree created. As the tree would not be fully expanded, the path through the tree is capped when a node with less than 3 visits is reached and actions are added randomly afterwards, up until individual length L . Similarly to the 1SLA initialisation, the rest of the individuals in the population are mutated from the first.

$$a^* = \arg \max_{a \in A(s)} \left\{ Q(s, a) + C \sqrt{\frac{\ln N(s)}{N(s, a)}} \right\} \quad (3.3)$$

3.2.4 Frame-skip

Frame-skipping has become common practice in several Reinforcement Learning works, and key in the success of specific applications (142; 143): grouping N game states when making a decision, to increase

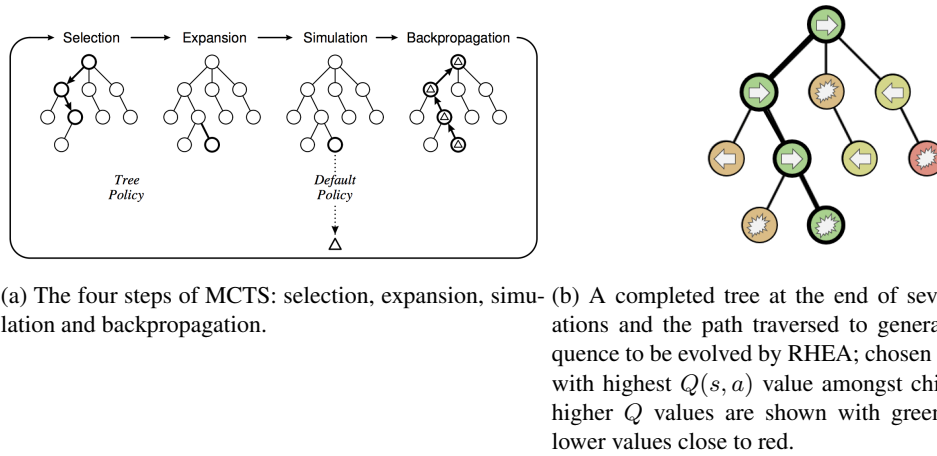


Figure 3.5: Monte Carlo Tree Search (MCTS)

the data available and reduce the frequency of decisions returned to only every N game states. Statistical forward planning approaches, on the other hand, usually make a new decision at every game tick, repeating their search process in the very limited time. With this modification, we test if SFP methods can also benefit from a longer time for making decision by only returning an action every N game ticks, replying according to a specific strategy for the game ticks in-between and using all the time in-between decisions for planning the next move. We test 0 (no frame-skip, decisions at every game tick), 5 and 10 as values for N and four different strategies for actions in-between decisions: *repeat*, *null*, *random* and *sequence*. The default choice for this parameter is 0 (no frame-skip is used in the vanilla algorithm).

The *repeat* strategy simply repeats the previously decided action until a new action is decided. The *null* strategy plays `ACTION_NULL` (does nothing), which more closely mimics human player gameplay with pauses in-between actions. The *random* strategy plays a random action and the *sequence* strategy continues playing the following actions in the best individual returned with the last decision. The frame-skip type parameter is dependant on the frame-skip value: if no frame-skip is used (value 0), then changing the frame-skip type would have no effect on the phenotype.

3.2.5 Shift Buffer

This is a population management technique which avoids repeating the entire search process from scratch at every new game tick, which usually loses information gained in previous iterations of the algorithm; this is meant to make the algorithm more sample-efficient by retaining previous computation information. A shift buffer works by keeping the final population evolved during one game tick to the next. However, as the first action of the best individual has just been played, all first actions from all individuals in the population are removed and a new random action is added at the end. Additionally, there exists the option in our implementation to apply a discount to the values of all individuals in the new population, which can be either 0.9, 0.99 or 1.0 (no discount applied); this would weaken the values of previously obtained sequences in the new context. The shift buffer discount parameter is dependent on the shift buffer toggle: if no shift buffer is used, then changing the shift buffer discount would have no effect on the phenotype. No shift buffer is used in the vanilla algorithm.

3.2.6 Dynamic Depth

When testing Artificial Intelligence agents on multiple distinct games in a general game playing black box setting, the main difficulty the players face is being able to correctly judge and differentiate situations. Most games cannot be fully explored until the end due to their complexity in action space, state space or both.

If the algorithm is targeted at a particular game, human knowledge about the problem can be integrated into the heuristic function in order to effectively guide the search in the right direction, even if no “natural” rewards (from the game) are observed by the agent. However, the lack of domain knowledge in general video game playing poses a significant challenge on how to bias or guide the search effectively in the case of a mostly flat reward landscape (49).

Further, it is often the case that different games benefit from different algorithm parameters and one key aspect is the length of the action plans used in RHEA. We expect games with dense rewards to generally benefit from shorter individuals which would allow for more generations and more statistics gathered to facilitate quick strategic reactions; as opposed to games with sparse or no rewards, where longer individuals

are required in order to be able to find those rewards farther ahead. This difference in reward density can also be observed at a more granular level, during the play-through of only one game: some areas of the game may contain more rewards, whereas others would require more exploration. Therefore, the dynamic depth modification has the option of changing the length of the individuals at every 5 game ticks depending on the flatness of the reward landscape, in order to examine the algorithm’s ability to adapt to the various types of problems proposed, as described below. By default, this modification is not used in the vanilla algorithm.

Algorithm 2 Adjusting rollout length dynamically

Require: t : current game tick
Require: $fitnessLandscape$: the fitness landscape (all fitness values) observed in the previous game tick
Require: f_{Ld} : fitness landscape flatness
Require: L : rollout length
Require: ω : adjustment frequency
Require: SD_- : lower f_{Ld} limit for L increase
Require: SD_+ : upper f_{Ld} limit for L decrease
Require: M_D : rollout length modifier
Require: MIN_D : minimum value for L
Require: MAX_D : maximum value for L

```

1:
2: if  $t \bmod \omega = 0$  then
3:   if  $fitnessLandscape = null$  then
4:      $f_{Ld} \leftarrow SD_-$ 
5:   else
6:      $f_{Ld} \leftarrow \delta(fitnessLandscape)$  ▷ get standard deviation
7:   if  $f_{Ld} < SD_-$  then
8:      $L \leftarrow L + M_D$ 
9:   else if  $f_{Ld} > SD_+$  then
10:     $L \leftarrow L - M_D$ 
11:    $L \leftarrow \text{BOUND}(L, MIN_D, MAX_D)$ 
12:
13: function  $\text{BOUND}(L, MIN_D, MAX_D)$ 
14:   if  $L < MIN_D$  then
15:      $L \leftarrow MIN_D$ 
16:   else if  $L > MAX_D$  then
17:      $L \leftarrow MAX_D$ 
18:   return  $L$ 

```

The pseudocode of the method used to adjust the rollout length is depicted in Algorithm 2. The adjustment is set to occur with a frequency $\omega = 15$ game ticks. The feature used to determine a change in rollout length is the flatness of the fitness landscape observed in the previous game tick (f_{Ld}); this is therefore ignored if the first game tick is currently observed (therefore no fitness landscapes were previously recorded). The fitness landscape is a vector with all fitness values observed in one game tick by any individual in the population (RHEA) or any rollout (MCTS), and its *flatness* is calculated as the standard deviation (δ) of all the elements of this vector (Line 6).

The length L is then increased by the depth modifier $M_D = 5$ if f_{Ld} falls below the lower limit ($SD_- = 0.05$), or is decreased by M_D if f_{Ld} is above the upper limit ($SD_+ = 0.4$) (see Lines 7-10). The length is capped to always stay between a minimum (1) and a maximum (half of the maximum number of FM calls; Line 11). This translates to shorter rollouts when the fitness values observed are highly varied (therefore more sampling and processing of the current situation is needed to determine the right course of action) and longer rollouts when the fitness landscape is flat, to encourage exploration of solutions farther ahead which would give the agent more information to judge which would be the best move. The values for the different variables (ω, SD_-, SD_+) were manually tuned for best performance of both algorithms on a random subset of 5 games.

3.2.7 Monte Carlo Rollouts

We consider the hybridisation of the algorithm and its further combination with MCTS, which has been very successful in many GVGAI games (2). We have previously described MCTS initialisation, but concepts from MCTS can further be borrowed and integrated into RHEA, such as its Monte Carlo (MC) simulation

phase, as initially proposed by (54). Therefore, the evaluation process in RHEA may add MC rollouts of length $\{0.0 \text{ (no rollouts used), } 0.5, 1.0 \text{ or } 2.0\} \times L$ after advancing through the action sequence of length L represented by the individual; these may be repeated 1, 5 or 10 times for more statistics gathered. In order for this to be compatible with the fitness assignment modifications, the values of all game states traversed (or the average value for a particular game tick if there are repetitions performed) R are added at the end of the array of state values F obtained from the individual and all fitness assignment methods are applied to the combined array of values $(F + R)$ instead. The MC rollout repetition parameter is dependent on the rollout length: if the length is set to 0.0, then changing the number of rollout repetitions would have no effect on the phenotype. The motivation behind the use of additional rollouts lies in the fact that the algorithm receives a farther lookahead, without being restricted to only a specific set of actions (as it is the case when the L parameter value is increased directly).

The default choice for this parameter is rollout length of 0.0 (no rollouts used in the vanilla algorithm).

3.2.8 Bandit-Based Mutation

The multi-armed bandit problem (144) is a classic problem, in which a gambler having access to multiple machines needs to make a decision as to which machine's lever they should pull. Each machine produces a random reward from a specific probability distribution. The goal of the gambler is to maximise the sum of rewards obtained through subsequent plays. Therefore they need to balance their exploration and exploitation, in order to learn the different distributions, while getting the maximum benefits from their plays.

One of the solutions to the problem, and the bandit-based mutation included in RHEA (explored previously in (78)), is using the UCB (Upper Confidence Bound) equation (Equation 3.3). The first term ($Q(s, a)$) attempts to maximise the value of the play (exploitation). The second term favours levers which were pulled the least number of times (exploration), $N(s, a)$ indicating the number of times lever a was pulled and $N(s)$ the total number of plays. The constant C ($C = \sqrt{2}$) is that which balances between the two terms and it may be adjusted to fit specific problems. In GVGAI, levers are represented by actions, therefore, from one state, the UCB equation would ensure that good actions are chosen, while exploring those not chosen as often to analyse their effect and build up the knowledge base.

In the RHEA variants with bandit-based mutation, two levels of bandit systems are used.

The first system is at individual level, used to select which gene to mutate. In the exploration term from the UCB equation (Equation 3.3), $N(s, a)$ is the number of times gene a was mutated and $N(s)$ is the total number of mutations. The exploitation term is determined by $\max(\Delta R)$, the maximum difference in rewards observed when mutating gene a . The differences in rewards are updated after each mutation by evaluating the new individual obtained. If the new ΔR is negative (thus there was no improvement in the value of the individual), the mutation is reverted. Therefore, the individuals will never get worse with this mutation operator.

The second system is at gene level (therefore L bandits, one for each gene). The information from all of the individuals in the population is stored in the same set of L bandits as they all aim to find the same optimal action plan. Therefore, a number $P - E$ values are used for updating the bandit information each generation. In this case, the exploration term is made up of the number of times gene X was changed ($N(s)$) and the number of times gene X received value a ($N(s, a)$). The exploration term is the ΔR corresponding to the value a .

When combined with the shift buffer, the gene-level bandits are shifted along with the population in the same manner. Additionally, all ΔR values are discounted by the same discount factor given by the shift discount parameter.

By default, this modification is not used in the vanilla algorithm.

3.2.9 Statistical Tree

This modification keeps a statistical tree alongside the population evolved by RHEA, similar to the work in (69). The tree is initialised at every game tick with the current game state as the root node. Then, each sequence of actions considered by RHEA during its evolutionary process is used to traverse the tree, new nodes being added if new actions are encountered. Each tree node visited in this operation is then updated: the visit count is increased by 1, and the node average value is updated with the fitness of the individual. $(P - E) \times L$ nodes are updated at each generation.

This statistical tree comes into play when choosing which move to make at the end of RHEA's decision-making process. The vanilla version of the algorithm returns the first action of the best individual found. With this enhancement, the action returned is the tree's root node's child with the highest UCB value (Equation 3.3), aiming for a better balance between exploitation and exploration of the search space.

Combining this modification with the shift buffer results in the tree being trimmed and carried forward, the root node's selected child becoming the root of the tree in the next game step, while its siblings are discarded, in a similar manner as in (69). If the tree is kept, the values stored in its nodes are discounted according to the shift discount parameter.

By default, this modification is not used in the vanilla algorithm.

3.2.10 Diversity

Genotypic diversity keeps track of the values of each gene from all individuals explored during evolution (each gene has a fixed list of possible values, and a dynamically-updated list of values visited during evolution). The *diversity score* of an individual is the inverse sum of the number of visits per each of its gene values. The *mutation operator* chooses to mutate the gene that has currently been explored the most, to the value for the gene that has been explored the least.

Phenotypic diversity mutation keeps track of the positions (within the level grid) explored during evolution. Individuals are rolled out to find the positions reached after each action is executed. The *diversity score* of an individual is the inverse sum of the number of visits per each of its gene positions. The *mutation operator* chooses to mutate the gene that leads to the most visited position, to a random different value.

The diversity type operator toggles between these two options. The diversity score is used when assigning the fitness of an individual, as a simple weighted sum with the reward obtained from regular evaluations. Fitness weight w is assigned to the diversity score, and $1 - w$ to the reward obtained.

3.2.11 Other Parameters

Last but not least, we can change several other parameters in the algorithm:

- **Population size:** the number of individuals evaluated at once can vary between 1 and, technically, infinity - the upper limit on the population size is imposed by the budget with respect to action plan lengths and other modifications which might affect how the budget is spent. When the population size is 1, the algorithm becomes a simple Random Mutation Hill Climber: only 1 new individual is created at every generation by mutating the current individual, and the best of the two is carried forward to the next generation to repeat the process. With 2 individuals in the population, crossover is introduced as well, but not selection, as there could be no other choice for parents other than the 2 individuals. All modifications take full effect if the population size is 3 or larger.
- **Offspring count:** this parameter sets the number of individuals to be created at every generation. For each one a process of selection (repeated twice to obtain 2 parents), crossover and mutation is applied. The next generation will be formed of the best individuals from the pool combining parents and offspring (or, alternatively, parents could be ignored and only offspring considered for the next generation). By default, this is set to the size of the population.
- **Number elites:** this parameter sets the number of best individuals (highest fitness) promoted directly to the next generation; these individuals may still spawn offspring, but they are guaranteed to continue existing in the next generation. By default, this is set to 1.
- **Individual length:** this parameter sets the length of action plans evolved, introducing the trade-off between multiple generations being produced (and therefore better chance of obtaining high-fitness individuals) and longer lookahead into possible futures, which could be key in finding rewards in the environment further away from the player.

Chapter 4

RHEA Benchmarking

This chapter presents work done regarding benchmarking of Rolling Horizon Evolutionary Algorithms (RHEA) in the context of the larger domain of General Video Game Playing and specifically on the General Video Game AI (GVGAI) framework. It mostly focuses on improvements or modifications presented in previous literature and integrates them into the same system for fair experiments, in the same environments and under the same constraints. Parameters and modifications are tested in isolation, but some combinations are also explored, as well as their larger effect on the inner-workings of the algorithm.

All of the studies are presented in the context of the General Video Game AI Framework and the games described in Section 2.1.2; it is hardly possible that the same configurations of the algorithm would work well across all games, as they showcase a variety of features and interesting differences requiring different skills from the players, such as navigation, movement or shooting accuracy or execution of carefully constructed plans.

We begin by exploring the two main and most basic parameters of the algorithm, its population size and individual length. Next, we consider the initialisation method for the algorithm and the possibility of starting the search from a better than random point in the space. We further look at combinations of several modifications previously introduced separately in literature, and finish the chapter by exploring a simple case of macro actions in physics-based games. All sections include a comparison to the previous state-of-the-art algorithm, Monte Carlo Tree Search.

We define two concepts for the purpose of analysis and discussions presented in this section:

Definition 4.1 *The **solution space** refers to an individual’s genotype, and is the set SS containing points $p_{SS} \in SS$, with $p_{SS} = (c_0, c_1, \dots, c_L)$, where c_i is one coordinate for the point, corresponding to the value of gene g_i in an individual’s genome. For example, if the individual has length $L = 3$ and all genes can take values in range $[0,3]$, then $(0,1,2)$ and $(2,2,3)$ are valid points in the solution space, but $(1,2,5)$ and $(2,2,0,1)$ are not. If a population contains individuals $\{(0,1,2), (2,2,3)\}$, then this population is said to explore 2 points in the solution space. In other words, the solution space contains all possible combinations of gene values in individuals.*

Definition 4.2 *The **level space** refers to an individual’s phenotype, and is the set SL containing points $p_{SL} \in SL$, with $p_{SL} = (x, y)$, where x is a coordinate on the X axis of the game’s 2D level grid, and y is a coordinate on the Y axis of the level grid. For example, if the player’s avatar is at location $(0,0)$ at game tick t (with origin being in the top-left corner of the level grid) and an individual of length $L = 3$ maps to the sequence of actions (down, down, right), then this individual is said to explore 4 points in the level space, including the starting location: $\{(0,0), (0,1), (0,2), (1,2)\}$. In other words, the level space contains all possible positions the player’s character can take on the level grid.*

4.1 Population Size and Individual Length

The work in this section was published at Evostar 2017:

R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, “Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing,” in *Springer Lecture Notes in Computer Science, Applications of Evolutionary Computation, EvoApplications*, no. 10199, 2017, pp. 418–434.

This section primarily aims at analysing the performance of the vanilla Rolling Horizon Evolutionary Algorithm, as described in Section 3.1, with respect to the different nature of the games used as a testbed and especially their stochasticity. The focus of this section lies in the 2 key parameters of the algorithm: population size and individual length. These control the number of solutions evolved at the same time and

the length of these solutions: higher population sizes mean sampling more solutions and increasing the possibility of finding overall better action plans (see Figure 4.1); higher individual lengths mean simulating the environment further into the future, which can allow the algorithm to find those rewards further away into the future, thus especially useful in sparse reward environments (see Figures 4.2,4.3). However, it is also worth noting that, in stochastic environments, longer rollouts would also lead to more prediction inaccuracies, as the algorithm uses a model of the world for its simulations which would stray further from reality the deeper into the future the algorithm looks.

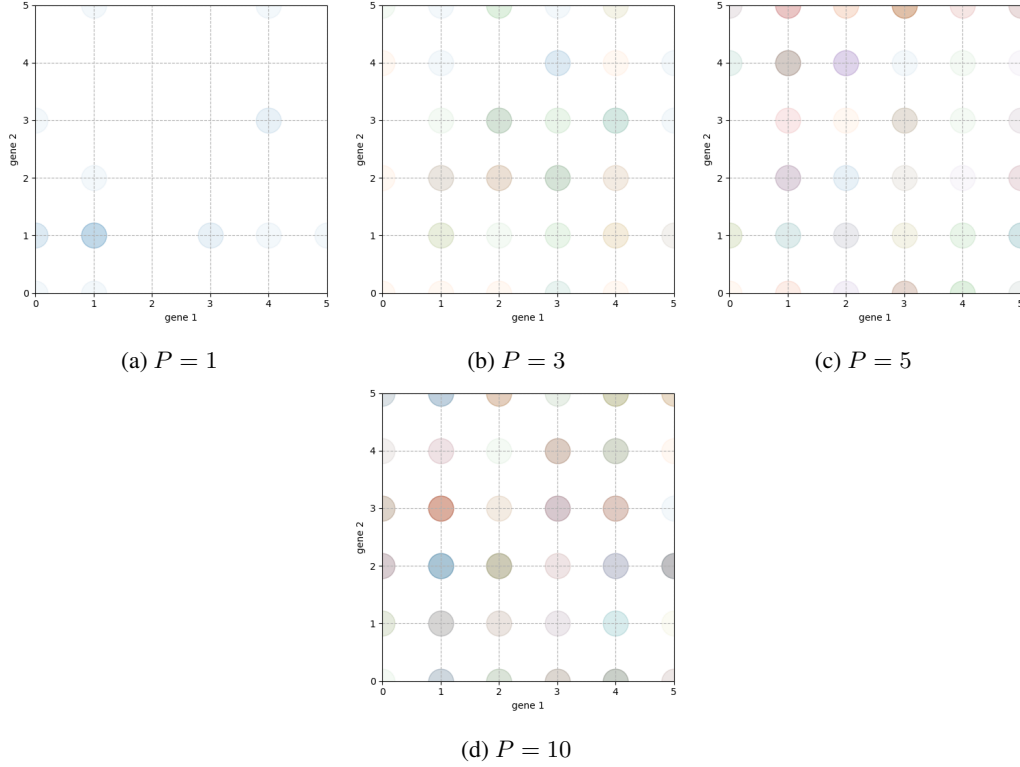


Figure 4.1: **Solution space** explored by a simulated evolutionary algorithm with varying **population sizes**. Individual length set to $L = 2$ for visualisation purposes, with values for each gene (0-5) on the X and Y axes. Each colour is a different individual mutating over 20 generations (1-bit mutation), each point plotted with 0.05 opacity (thus more intense colours signify the point was sampled more times).

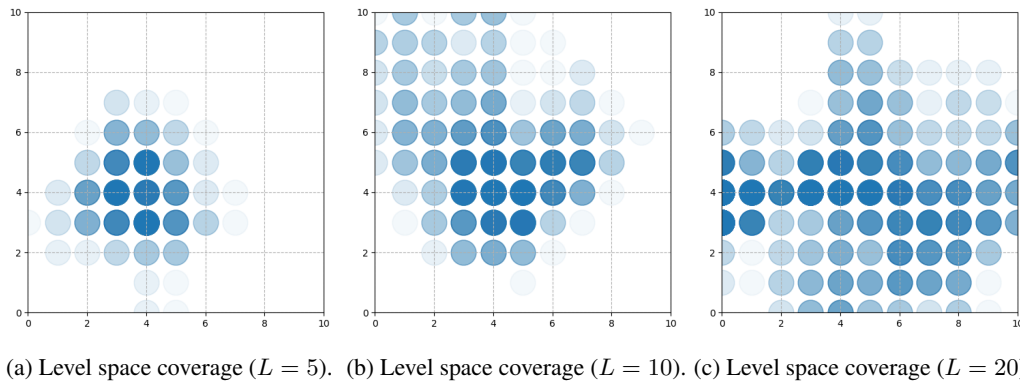


Figure 4.2: **Level space** explored by a simulated evolutionary algorithm with varying **individual lengths**. Population size set to $P = 1$ for visualisation purposes, axes corresponding to level space X and Y axes, in a grid of size 10. Each point is plotted with 0.05 opacity (thus more intense colours signify the point was sampled more times).

And while increasing both of these values might seem like an excellent idea, we are working within real-time constraints and an interesting trade-off arises, where we can consider four separate extreme situations (in these examples, we will denote with S small values for the parameters, and with L large values, using the format $\{\text{population_size}\} - \{\text{individual_length}\}$):

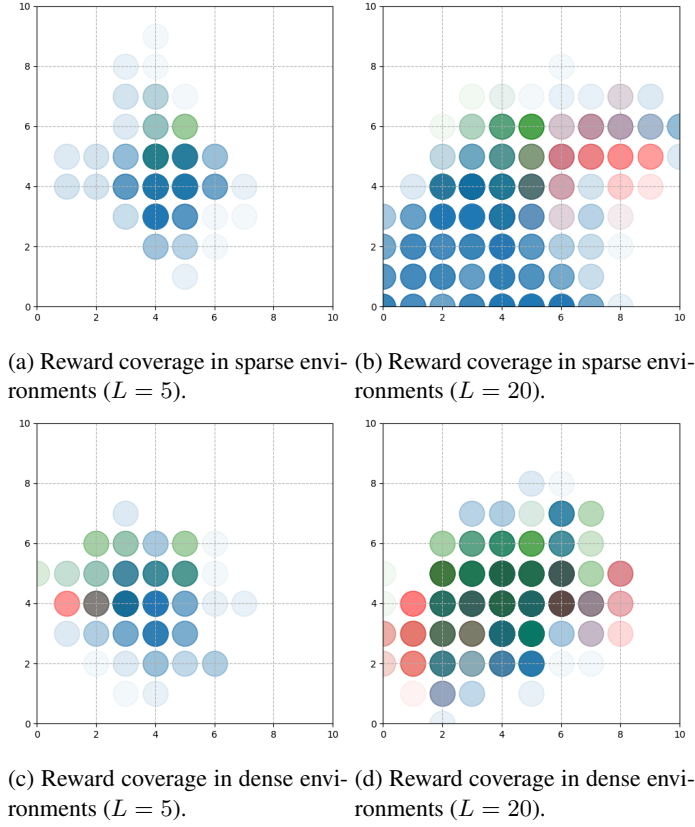


Figure 4.3: **Level space** explored by a simulated evolutionary algorithm with varying **individual lengths**. Population size set to $P = 1$ for visualisation purposes, axes corresponding to level space X and Y axes, in a grid of size 10. Each point is plotted with 0.05 opacity (thus more intense colours signify the point was sampled more times). Featuring both positive (green) and negative (red) rewards, which give the colour to the whole individual once encountered; blue remains neutral.

1. **S-S**: highest number of generations within budget and therefore most accurate statistics obtained from evolution; the algorithm is, however, myopic, and likely to not find rewards further away and therefore unable to navigate a flat reward landscape. It does not explore much of the level space, and it is likely to not explore much of the solution space either (restricted to the neighbourhood of the few initial solutions).
2. **S-L**: medium number of generations within budget, but with longer lookaheads; the algorithm is now more likely to find rewards farther ahead, exploring more of the level space, but it is, similarly to the case of *S-S*, constrained to navigating a small part of the solution space.
3. **L-S**: medium number of generations within budget, this time with more solutions, leading to a wider exploration of the solution space, but the algorithm is again myopic and has a higher chance of getting stuck in a flat reward landscape, not exploring much of the level space.
4. **L-L**: lowest number of generations within budget, resulting in less accurate statistics of which solutions are actually good: however, the algorithm has a very high chance of finding far-away rewards through ample level space exploration, as well as exploring more of the solution space.

In the extreme case, where the budget is only used for initialising and evaluating a single population, and no budget remains for evolution, the algorithm becomes *Random Search* in the space of action plans. In the extreme case of using a single individual in the population, the algorithm becomes a *Random Mutation Hill Climber*, which aims to improve that single solution through repeated mutations.

This section therefore analyses how modifying the population size (P) and individual length (L) configuration of vanilla RHEA impacts performance in a generic setting. Exhaustive experiments were run on all combinations between population sizes $P = \{1, 2, 5, 7, 10, 13, 20\}$ and individual lengths $L = \{6, 8, 10, 12, 14, 16, 20\}$. The budget defined for planning at each game step was set as 480 forward model calls to the *advance* function, the average number of calls MCTS is able to perform in 40ms of thinking time

Table 4.1: Average win rate over **all** 20 games tested, for different values of population size (P) and individual length (L). Standard errors in brackets. Highlighted in bold style is the best result.

P	L=6	L=8	L=10	L=12	L=14	L=16	L=20
1	35.45(2.54)	38.25(2.54)	37.95(2.47)	36.70(2.58)	34.20(2.42)	33.55(2.57)	33.15(2.60)
2	39.95(2.62)	40.95(2.55)	41.05(2.62)	40.25(2.48)	39.50(2.56)	38.75(2.56)	36.80(2.60)
5	42.55(2.57)	43.50(2.39)	44.65(2.40)	44.25(2.38)	43.80(2.34)	44.95(2.53)	46.05(2.54)
7	43.00(2.49)	42.60(2.43)	44.65(2.36)	44.35(2.45)	45.30(2.23)	44.80(2.47)	47.05(2.56)
10	42.25(2.53)	43.60(2.49)	44.05(2.26)	45.80(2.47)	45.05(2.35)	46.60(2.45)	46.80(2.49)
13	42.65(2.43)	45.15(2.48)	45.15(2.47)	45.00(2.42)	46.25(2.41)	47.40(2.30)	47.05(2.42)
20	42.75(2.51)	43.20(2.60)	44.75(2.31)	45.50(2.34)	46.45(2.32)	46.30(2.32)	47.50 (2.33)

Table 4.2: Average win rate over the 10 **deterministic** games tested, for different values of population size (P) and individual length (L). Standard errors in brackets. Highlighted in bold style is the best result.

P	L=6	L=8	L=10	L=12	L=14	L=16	L=20
1	22.30(2.88)	26.80(2.95)	26.90(2.93)	25.30(2.91)	24.20(2.84)	23.00(3.01)	22.50(2.99)
2	26.40(3.13)	26.80(3.08)	27.90(3.05)	27.90(2.92)	27.10(2.91)	26.80(2.93)	24.50(2.99)
5	28.70(3.08)	29.70(3.10)	31.90(3.18)	31.80(2.88)	30.00(2.86)	32.00(3.04)	32.20(3.19)
7	28.80(3.26)	29.00(3.00)	30.80(3.09)	30.40(3.01)	31.70(2.82)	32.00(2.99)	34.30(3.12)
10	27.70(3.18)	31.00(3.27)	29.50(2.90)	33.00(3.03)	32.60(2.94)	32.40(3.11)	33.20(3.05)
13	28.90(3.19)	32.20(3.32)	32.10(3.06)	31.80(3.07)	33.30(3.18)	34.70 (2.88)	34.00(2.97)
20	28.60(3.19)	29.90(3.34)	31.50(2.87)	32.30(3.05)	33.10(3.11)	32.10(2.84)	34.30(3.02)

Table 4.3: Average win rate over the 10 **stochastic** games tested, for different values of population size (P) and individual length (L). Standard errors in brackets. Highlighted in bold style is the best result.

P	L=6	L=8	L=10	L=12	L=14	L=16	L=20
1	48.60(2.20)	49.70(2.13)	49.00(2.01)	48.10(2.25)	44.20(2.00)	44.10(2.12)	43.80(2.22)
2	53.50(2.12)	55.10(2.02)	54.20(2.20)	52.60(2.05)	51.90(2.20)	50.70(2.20)	49.10(2.22)
5	56.40(2.07)	57.30(1.68)	57.40(1.61)	56.70(1.88)	57.60(1.81)	57.90(2.01)	59.90(1.89)
7	57.20(1.72)	56.20(1.85)	58.50(1.64)	58.30(1.90)	58.90(1.63)	57.60(1.95)	59.80(2.00)
10	56.80(1.88)	56.20(1.71)	58.60(1.63)	58.60(1.91)	57.50(1.77)	60.80 (1.79)	60.40(1.93)
13	56.40(1.68)	58.10(1.65)	58.20(1.88)	58.20(1.76)	59.20(1.63)	60.10(1.71)	60.10(1.86)
20	56.90(1.83)	56.50(1.86)	58.00(1.74)	58.70(1.64)	59.80(1.53)	60.50(1.80)	60.70(1.64)

in the games of this framework¹. Larger values for either individual length or population size were not considered due to the limited budget and the complete nature of the experiment (analysis of all combinations); values above 24 would not allow in certain cases for a full evaluation of even one population.

To expand the analysis of the results, a particular configuration was also tested, using $P = 24$ and $L = 20$. Effectively, given the budget of 480 Forward Model calls, this is an equivalent method of *Random Search* (RS). The algorithm only has enough budget to initialise and evaluate the initial population, before applying any genetic operator. In essence, this configuration evaluates 24 random walks and returns the first action of the best sequence of moves found.

In order to validate the results, MCTS was also tested on the same set of 20 games, under the same budget conditions. MCTS has proven to be the dominating technique out of the sample ones provided in the GVGA competition, with numerous participants using it as a basis for their entries before adding various enhancements on top of its vanilla form. The winner of the first edition of the competition in 2014, Adrien Couëtoux (6), employed an Open Loop technique quite similar to this algorithm.

Each algorithm was run 20 times on each of the 5 levels of the 20 GVGA games, therefore 100 runs per game. We consider win rate as the measure for performance and record this for all runs.

4.1.1 Results and Discussion

This section presents and discusses the results obtained from this study. Observations are made attending to the nature of the game and variations of the population size and individual length. Section 4.1.1 compares

¹Using these forward model calls instead of real execution time is more robust to fluctuations on the machine used to run the experiments, making it time independent and results comparable across different architectures.

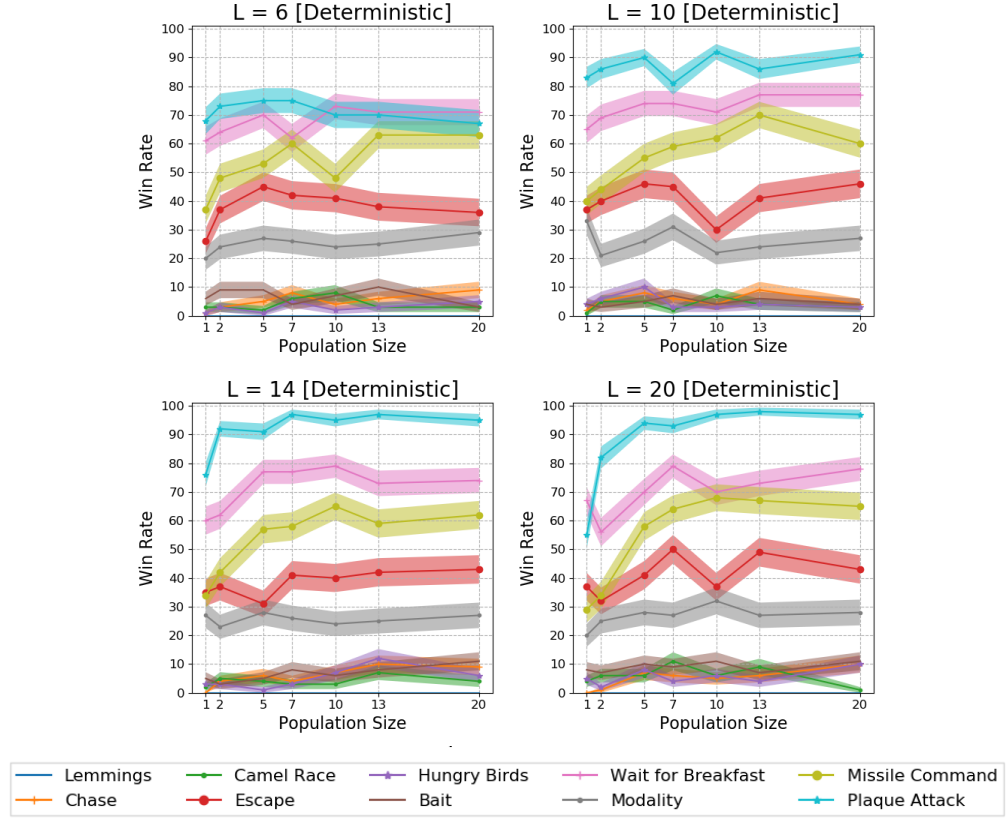


Figure 4.4: Change of win rate as population size increases, for different individual lengths, in all **deterministic** games tested. The standard error is shown by the shaded boundary.

performance regarding the use of smaller or larger population, while Section 4.1.1 discusses the impact of individual length variations. Later, the performance of RHEA is also compared to RS employing different budgets (Section 4.1.1) and to MCTS (Section 4.1.1) in the implementation supplied with the GVGAI framework. As the game set used is divided equally between deterministic and stochastic games, an in-depth analysis is carried out on each game type, although it is not implied the trend would carry through in other games of the same type. Additionally, a Mann-Whitney non-parametric test was used to measure the statistical significance of results for each game (p -value = 0.05). Table 4.1 summarises the win rates of all configurations tested in this study.

Population Variation

Figures 4.4, 4.5 shows the change in win rate as population size increases, for $L = \{6, 10, 14, 20\}$. Each of the 20 games that these algorithm configurations were tested on showed different trends when the parameter values were varied. There is a trend noticed in most of the games, with win rate increasing with the increase in population size, regardless of the game type (see Table 4.1). Exceptions are for games where the win rate starts at 100%, therefore leaving no room for improvement (“Aliens” and “Intersection”) or, on the contrary, when the win rate stays very close to 0% due to outstanding difficulty (“Roguelike”). The winning rate in the game “Crossfire” reaches a peak at 10% for $P = 5$, $L = 5$, compared to 0% for $P = 1$, which suggests that games which a priori seem unsolvable, can be approached by exploring more of the solution space with a larger population - however, there should be a balance between such exploration and the accuracy of statistics through more generations of the evolutionary algorithm, as shown by win rates dropping at higher values for some of the games (e.g. “Sequest”, “Survive Zombies”, “Escape”); these are games with very dense environments full of sprites the player can interact with, which shows a specific need for exploration-accuracy balance when the player has many viable options available.

In general, a conclusion that could be drawn from these experiments is that increasing the population size rarely hinders the performance of the agent. In fact, in some cases it makes the difference between a very poor and a very successful performance (e.g. in the game “Chopper”). Therefore most games benefit from a high exploration of the solution space, although we see some games as much more sensitive than others in choosing the correct balance between the exploration of the two spaces.

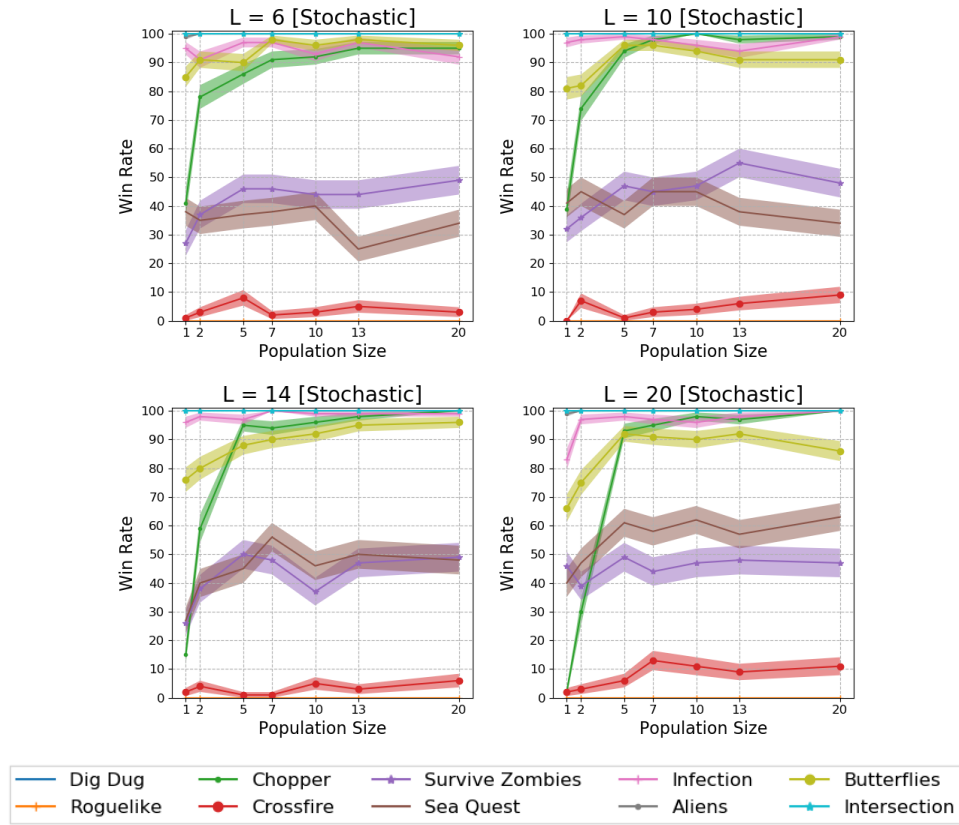


Figure 4.5: Change of win rate as population size increases, for different individual lengths, in all **stochastic** games tested. The standard error is shown by the shaded boundary.

Deterministic games The win rate varies within small bounds around certain values in deterministic games, with a small upwards trend as the population size increases (Figure 4.4). Several games stand out: “Missile Command” and “Wait for Breakfast” show similar curves and the highest increase in performance, although these games do not appear to have much in common (Table 2.1). Additionally, “Plaque Attack” sees much lower performance with $L = 20$, $P = 1$ (55%), which then increases to close to 100% for higher population sizes, showing a clear benefit for exploring both solution space and level space equally during evolution. Lastly, the game “Escape” sees higher variations in performance, with a dip in win rate for population size $P = 10$, especially with individual lengths $L = 10$ and $L = 20$. We hypothesise this is due to this game needing more focus on either level exploration, or solution exploration; both elements are very important here, but especially conflicting in this puzzle game requiring much precision in the execution of the action plans. Dissimilarly, “Escape” sees a curve peaking at $P = 5$, when the individual length is fixed at $L = 6$, and slowly decreasing afterwards, again highlighting the high precision required and the higher sensitivity towards parameter configuration observed in this game.

Stochastic games Regarding stochastic games (Figure 4.5), it is important to separate them based on their probabilistic elements and their impact on the outcome of the game. For example, the game “Survive Zombies”, has numerous random NPCs and probabilistic spawn points for all object types, in contrast with the game “Aliens”, where its stochastic nature comes only from the NPCs dropping bombs in irregular patterns. In the games “Butterflies”, “Chopper” and “Seaquest” (with larger individual lengths), a big improvement in terms of winning rate is observed by increasing the population size from 1 (the case in which there is no tournament) to 5, and this remains stable with larger populations. All of these games are placed in the same cluster by (3) and they are similar in terms of density of environments and winning conditions as well. The other games placed in the same cluster, “Missile Command” and “Plaque Attack” we have similarly observed to show increased performance with larger population sizes. As these are highly dynamic environments, a higher exploration of the solution space is needed to find viable “good” solutions.

When the length of the individual is fixed to a small value, $L = 6$, increasing the population size is not beneficial in all cases, sometimes having the opposite effect and causing a drop in win rate (in “Seaquest”, from 38% for $P = 1$ to 25% at the lowest with $P = 14$). On the contrary, the game “Chopper”, sees a great improvement (from an average of 41% in population size $P = 1$ to 95% in population size $P = 20$). Even

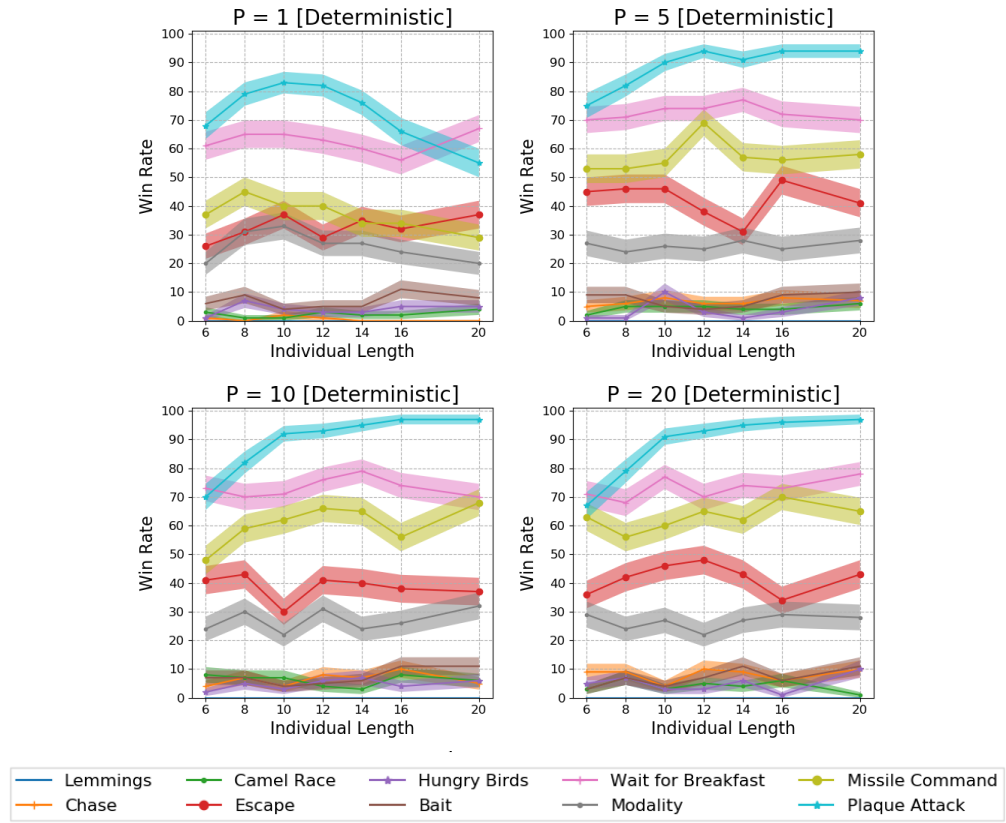


Figure 4.6: Change of win rate as individual length increases, for different population sizes, in all **deterministic** games tested. The standard error is shown by the shaded boundary.

though these games are very similar, the discontinuous rewards in “Seaquest” require a higher exploration of the level space than “Chopper”, thus they observe differences in performance with small individual lengths.

Individual Variation

Figures 4.6, 4.7 illustrates the change in win rate in each of the 20 games as individual length increases, with population sizes fixed to $P = \{1, 5, 10, 20\}$. The trend of increasing win rates along with the parameter value increase observed previously is kept here as well when the population size is larger, but the opposite is true in several games with $P = 1$, more notably for “Chopper”, “Butterflies” and “Plaque Attack”. These are the highly dynamic environments previously discussed which feature fairly dense rewards and many moving sprites - the fact that performance drops along with the increase in individual length suggests that these environments require prioritisation of solution exploration and accuracy of statistics gathered, rather than an increased exploration of the level space.

In general, increasing the length of the individual provides better solutions if the size of the population is high, although the effect of increasing the population size seems to be bigger. This can be clearly observed in the results reported in table 4.1 and suggests level exploration to be a key part in whether the method is successful at a game or not.

Deterministic games When there is only one individual in the population, thus no crossover is involved, the win rate experiences an increase followed by a drop along with the increase of individual length in “Plaque Attack”, “Missile Command” and “Modality”, which are the 3 deterministic games in one of the clusters identified in (1). This is due to the fact that the size of the search space of solutions increases exponentially with the individual length and therefore, with few individuals evaluated, the algorithm struggles to find optimal solutions in these challenging games. This issue can be solved by increasing the population size, and therefore the exploration of the solution space, as shown in Figure 4.6. For instance, the game “Plaque Attack”, sees a variation from 68% to 83% to 55% with population size $P = 1$; while with population size $P = 5$, there is a constant increase from 75% to 94% (and similarly for larger population sizes as well). Small upwards trends or small variations are observed for the rest of the games, more notably “Escape” shows again a preference for extreme individual lengths (either very small or very large).

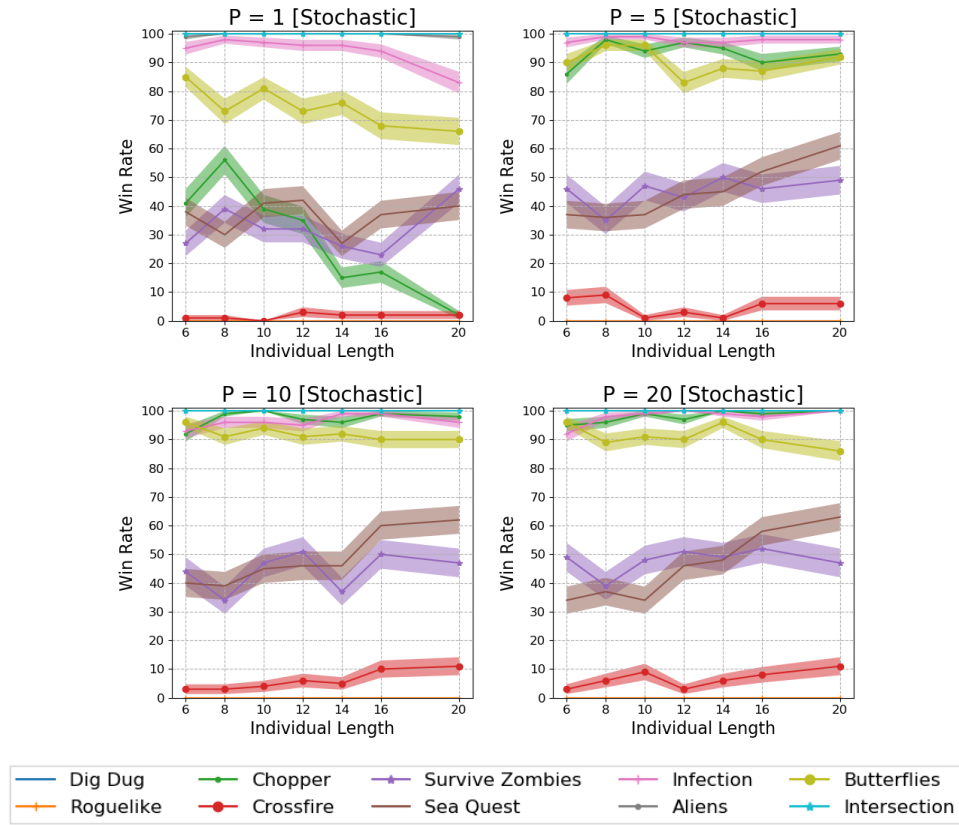


Figure 4.7: Change of win rate as individual length increases, for different population sizes, in all **stochastic** games tested. The standard error is shown by the shaded boundary.

Stochastic games In stochastic games, we see performance dropping for most games when $P = 1$ and increasing for larger population sizes. For instance, in “Butterflies”, win rate drops from 85% ($L = 6$) to 66% ($L = 20$), when evolving a single individual in the population; this game is particularly interesting as curves shown in Figure 4.7 highlight the balance between level and solution explorations by peaking in the middle or for smaller individual lengths. An even bigger difference in win rate can be seen in “Chopper”, which drops from 40% ($L = 6$) to 2% ($L = 20$) with a single individual in the population; however, this game sees close to 100% win rate when the population size is increased and crossover is introduced in the evolution process to introduce more ample exploration of the solution space. “Sequest”, “Survive Zombies” and “Crossfire” show very similar curves in all cases, with performance increasing along with individual lengths. These games have loss conditions in common and medium-sized very dense environments full of threats for the player character: a better exploration of the level space can provide a better mapping of dangers spread out on the map and therefore better inform the agent when making its decisions.

Random Search

The version of RHEA using large values for population size and individual length is reminiscent of the *Random Search* (RS) algorithm, meaning the algorithm only has time to initialise and evaluate P solutions of length L and chooses the best one found, without any evolution occurring. We run RS on the same set of games using $P = 24$ individuals and simulation depth $L = 20$, with the same allocated budget of 480 calls to the forward model. The average win rate over all games is summarised in the last row of Table 4.4.

RS performs no worse than any variant of RHEA studied previously. The vanilla version of RHEA is not able to explore the search space better than (and, in most cases, not even as good as) RS when the budget is very limited. In order to test the limits and potential benefits of evolution with a single configuration, an additional set of experiments was run, with increased FM calls budgets by 2, 3 and 4 times the original 480 call budget (resulting in 960, 1440 and 1920). We run RHEA, MCTS and RS on all games, keeping individual (or rollout, for MCTS) length to 20. RHEA uses a population of size 1: while this tries out as many individuals as RS, the difference is in random sampling of the whole individual as opposed to evolving the initial random solution.

The results, presented in Table 4.4, suggest performance does continue to improve when the algorithm

Table 4.4: Comparison of win rates achieved by RHEA, RS and MCTS with higher budgets. Win rates for all games (T), deterministic (D) and stochastic (S). All algorithm use individuals (or iterations for MCTS) of length 20, as many as possible within the given budget.

Algorithm	Budget	Win Rate (T)	Win Rate (D)	Win Rate (S)
RHEA	1920	49.40(1.58)	35.00(2.13)	63.8(2.15)
RHEA	1440	49.30(1.58)	35.20(2.14)	63.4(2.15)
RHEA	960	51.00(1.58)	38.20(2.17)	63.8(2.15)
RHEA	480	46.70(1.58)	33.60(2.11)	59.8(2.19)
MCTS	1920	44.60(1.57)	27.60(2.00)	61.60(2.18)
MCTS	1440	43.10(1.57)	24.00(1.91)	62.20(2.17)
MCTS	960	45.70(1.58)	27.60(2.00)	63.80(2.15)
MCTS	480	41.45(1.89)	22.20(2.45)	60.70(1.34)
RS	1920	48.60(1.58)	36.20(2.15)	61.00(2.18)
RS	1440	49.70(1.58)	36.80(2.16)	62.60(2.16)
RS	960	49.10(1.58)	37.00(2.16)	61.20(2.18)
RS	480	46.60(2.40)	32.90(3.04)	60.30(1.76)

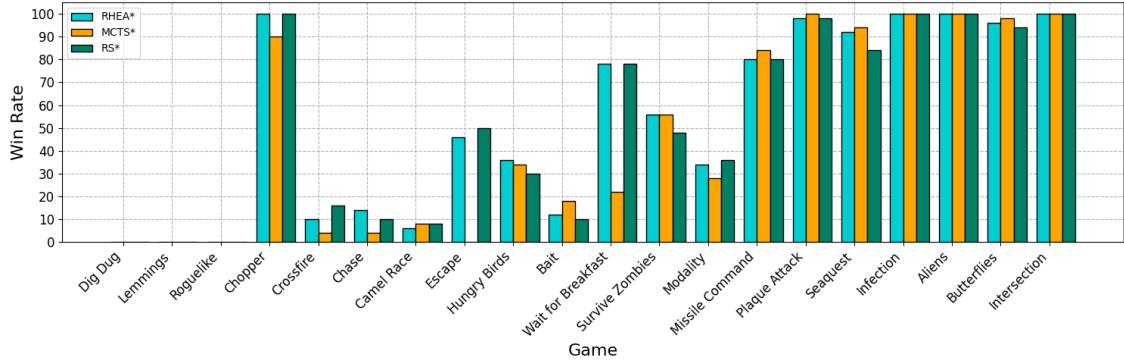


Figure 4.8: Win rate per game for best win rate per game of RHEA, MCTS and RS from experiments presented in Table 4.4, $L = 20$.

receives more thinking time, although it stabilises when reaching the highest budget tested. The difference observed is smaller than that given by the previous experiments presented.

In stochastic games, RHEA obtains similar performance in all higher budgets tested, comparable to MCTS-960 only and outperforming all others variations tested, including Random Search. For deterministic games, we note that the performance of MCTS is much below that of both RHEA and RS. RHEA-960 obtains the highest win rate, although this decays as the budget increases further. RS observes a similar trend and wins more games than RHEA with the highest budgets: this suggests that RHEA can get stuck in local optimum with only 1 individual evolved over so many iterations. Increasing exploration in the level space and accuracy of statistics gathered through more iterations might not be enough, but this is a step forward towards opening the possibility of solving very difficult, or seemingly unsolvable, problems.

Figure 4.8 shows the best win rate of each search method per game - we highlight that these are not the results of a single method, but the best across all variations for RHEA, MCTS and RS. Given this and the overall low performance observed in Table 4.4, we note that MCTS actually obtains the highest win rate in several of the games, such as “Bait”, “Missile Command”, “Plaque Attack” and “Seaquest”. This suggests a lack of consistency for this algorithm, which gives single RHEA instances the advantage when assessing overall performance.

Further, we observe *Random Search* to still be outperforming both RHEA and MCTS in several of the games (“Crossfire”, “Escape”, “Modality”), which, together with the very low win rates in half of the games, speaks to the difficulty of the problems proposed in the GVGAI framework.

RHEA vs MCTS

Table 4.4 also includes the performance of the GVGAI sample MCTS agent. The sample MCTS agent uses a playout depth of 10, hence the comparisons presented here relate to RHEA configurations with individual length $L = 10$. Results show that, although RHEA is significantly worse when its population size is small,

it outperforms MCTS when the number of individuals per population is increased ($P > 5$). It is very interesting to observe that it is possible to create a RHEA agent capable of achieving a higher level of play than MCTS, which is the base of most dominating algorithms in the GVGAI literature.

In addition, MCTS also falls short when comparing it to RS. Although it appears to be quite similar to RS in stochastic games, its performance is much worse than RS in deterministic games, becoming comparable to the worst configuration of RHEA found during these experiments (population size $P = 1$ and individual length $L = 20$). In Figure 4.8 there are several games where the algorithm achieves very low scores compared to other algorithms, deterministic games in particular standing out: “Escape”, “Wait for Breakfast” and the stochastic “Crossfire”. These are games grouped in the same cluster by (1) and feature puzzle elements and navigation skills, indicating RHEA to be better than MCTS in these types of games in particular. However, MCTS does outperform the best RHEA results in “Butterflies” and “Seaquest”, showing MCTS to be better at dealing with highly stochastic and dynamic environments.

4.2 Population Initialisation

The work in this section was published at IEEE CEC 2017:

R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, “Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing,” in *Proceedings of the Congress on Evolutionary Computation*, June 2017, pp. 1956–1963.

While Monte Carlo Tree Search and closely related methods have dominated General Video Game Playing, Section 4.1 has demonstrated the promise of Rolling Horizon Evolutionary Algorithms as an interesting alternative. However, there is little attention paid to population initialisation (or seeding) techniques in the setting of general real-time video games. Therefore, we propose the use of different population seeding to improve the performance of Rolling Horizon Evolution, focused on generating a better than random initial population from which to start the evolutionary process. We present results of using two methods to seed a RHEA population: One Step Look Ahead (ISLA) and Monte Carlo Tree Search (MCTS). We test both options on the 20 games of the General Video Game AI corpus with multiple parameter values for population size and individual length. An in-depth analysis is carried out between the results of the seeding methods and the vanilla RHEA. In addition, we discuss a comparison to Monte Carlo Tree Search.

Algorithm 3 ISLA Seeding

```

1: procedure INITIALIZEPOP( $s_t, n\_actions, budget$ )
2:    $P \leftarrow \text{new array}$ 
3:   for  $k = 0 : pop\_size$  do
4:     if  $k = 0$  then
5:        $I \leftarrow \text{new array}$ 
6:        $s \leftarrow s_t$ 
7:       for  $j = 0 : ind\_length$  do
8:          $Q(s) \leftarrow \text{ISLA}(s, n\_actions, budget)$ 
9:          $I[j] \leftarrow \arg \max_{a \in A(s)} Q(s, a)$ 
10:         $s \leftarrow s.advance(I[j])$   $\triangleright$  use FM to advance game state given best action
11:         $budget \leftarrow budget - 1$ 
12:     else
13:        $I \leftarrow mutate(P[0])$   $\triangleright$  Other individuals are mutations of the first (uniform mutation)
14:        $evaluate(I, s_t, budget)$ 
15:        $P[k] \leftarrow I$ 
16:   return  $P$   $\triangleright$  Initial population
17:
18: procedure ISLA( $s_t, n\_actions, budget$ )
19:    $Q(s_t) \leftarrow \text{new array}$ 
20:   for  $a = 0 : n\_actions$  do
21:      $s'_{t+1} \leftarrow s_t.advance(a)$   $\triangleright$  Use FM to advance game state given action available
22:      $budget \leftarrow budget - 1$ 
23:      $Q(s_t, a) \leftarrow h(s'_{t+1})$   $\triangleright$  Use Equation 2.1
24:   return  $Q(s_t)$   $\triangleright$   $Q$  values for all actions available from state  $s_t$ 

```

Therefore, the aim of this section is to explore whether initialising the population of an Evolutionary Algorithm with individuals better than random produces an improvement in performance, defined as win

Algorithm 4 MCTS Seeding

```
1: input:  $M \leftarrow 3$  ▷ MCTS node visit count limit
2:
3: procedure INITIALIZEPOP( $s_t, n\_actions, budget$ )
4:    $P \leftarrow \text{new array}$ 
5:   for  $k = 0 : pop\_size$  do
6:     if  $k = 0$  then
7:        $I \leftarrow \text{new array}$ 
8:        $A \leftarrow \text{MCTS}(s, n\_actions, budget/2)$ 
9:       for  $j = 0 : \text{length}(A)$  do
10:         $I[j] \leftarrow A[j]$ 
11:       for  $j = \text{length}(A) : \text{ind\_length}$  do
12:         $I[j] \leftarrow \text{random}(0, n\_actions)$ 
13:     else
14:        $I \leftarrow \text{mutate}(P[0])$  ▷ Other individuals are mutations of the first (uniform mutation)
15:        $\text{evaluate}(I, s_t, budget)$ 
16:        $P[k] \leftarrow I$ 
17:   return  $P$  ▷ Initial population
18:
19: procedure MCTS( $s_t, n\_actions, budget$ )
20:    $root \leftarrow s_t$ 
21:   while  $budget \neq 0$  do
22:      $node = \text{MCTS\_select}(root)$  ▷ select node in tree using UCB1 3.3, uses  $budget$ 
23:      $exp = \text{MCTS\_expand}(node)$  ▷ add new child of node to tree, uses 1 FM call
24:      $s' = \text{MCTS\_simulate}(exp)$  ▷ simulation from new node for  $L$  steps, to  $s'$ , uses  $L$  FM calls
25:      $Q \leftarrow h(s')$  ▷ Use Equation 2.1
26:      $\text{MCTS\_backpropagate}(exp, Q)$  ▷ update  $Q, N, N_a$  for all nodes visited during this iteration
27:    $A \leftarrow \text{new array}$ 
28:    $s \leftarrow s_t$ 
29:    $node \leftarrow \arg \max_{a \in A(root)} N(s, a)$  ▷ get child of root recommendation, most visited
30:   while  $N(s, node) > M$  and  $\text{length}(A) < \text{ind\_length}$  do
31:      $A.add(node)$ 
32:      $s \leftarrow s.advance(node)$  ▷ use FM to advance state to chosen node
33:      $budget \leftarrow budget - 1$ 
34:      $node \leftarrow \arg \max_{a \in A(node)} N(s, a)$  ▷ continue to best child recommendation
35:   return  $A$  ▷ Action sequence recommendation
```

rate (or the ability to solve a variety of different problems), when applied to General Video Game Playing. This section experiments with the parameters described in Section 3.2.3.

This hypothesis was tested by using the two initialisation methods to extend vanilla RHEA; we will refer to this algorithm as *RHEA-R* in this section. Algorithm *RHEA-ISLA* is a seeding variant which employs a One Step Look Ahead technique to select a better starting point in the solution space. Algorithm *RHEA-MCTS* uses Monte Carlo Tree Search to seed the RHEA for better analysis of the solution space. A fourth algorithm’s performance was compared against the RHEA variants, an Open Loop Monte Carlo Tree Search (algorithm MCTS), using the implementation described in Section 2.2.

The effect of the initialisation techniques was tested on different configurations of the RHEA algorithm, with population sizes (P) and individual lengths (L) in the following ranges: $P = \{1, 2, 5, 10, 15, 20\}$, $L = \{6, 8, 10, 14, 16, 20\}$, following only the diagonal of the matrix these values would form. In the case of algorithm MCTS, its rollout depth was kept the same as the RHEA individual length in order to make the approaches comparable. The largest value tested was 20 due to the fact that, by allowing half of the budget for MCTS computation in algorithm *RHEA-MCTS*, higher values for P and L would result in the algorithm not being able to evaluate even 1 whole population in the initialisation step.

In order to account for the stochastic aspect of the algorithms used in this study, as well as half of the games included in the set, each algorithm was run 100 times on each game (20 times on each of the 5 levels available). The budget offered for decision-making in each game tick was 900 FM calls, which is the average number of FM calls that *RHEA-R* achieves in 40ms of computational time in the complete 100 games in the GVGAI-1P corpus. Note that this is almost double the budget used in the previous section, which reflected the average number of FM calls used by MCTS in the real-time GVGAI competition limits.

4.2.1 1SLA Seeding (Algorithm *RHEA-1SLA*)

The One Step Look Ahead algorithm is a simple technique which exhaustively searches through the actions available from the current state and associates each a Q value, corresponding to the approximated value of the game state reached after performing each action (the value is defined by the same heuristic employed by RHEA). It then selects for execution the action with the highest Q value.

Algorithm *RHEA-1SLA* uses the same evolutionary process as *RHEA-R* described in Chapter 3, but the first individual in the initial population is the solution recommended by the 1SLA technique. L iterations of the 1SLA algorithm are performed to obtain a first individual, replacing the `initializePop` function in Algorithm 1 as shown in Algorithm 3 (the rest of the algorithm continues as normal). An exhaustive search is carried out through all of the actions available from the current state, the game state is advanced using the FM, through the best action found and the process is repeated until either the end of the individual or the end of the game is reached. In the second case, the rest of the individual is padded with randomly selected actions. If the population size is bigger than 1, the rest of the individuals are obtained by mutating the first individual obtained from the 1SLA algorithm.

This technique uses $L \times N + L \times P$ FM calls from the budget for initialisation and evaluation of initial population (where N is the number of actions available in the game), and is thought to reduce random bias (the vanilla algorithm potentially not being able to find the current best action because of the random seeding) and to provide a better starting point for evolution, guiding the search process towards an initially promising part of the solution space.

4.2.2 MCTS Seeding (Algorithm *RHEA-MCTS*)

Algorithm *RHEA-MCTS* splits the budget received and uses half of it to first run Monte Carlo Tree Search on the current game state, following the steps described in Section 2.2. The rollout depth is set to the same value as the individual length in *RHEA-R* and the UCB1 formula (with constant C taking the value $\sqrt{2}$) is applied as tree policy (see Equation 3.3).

The first individual in the initial RHEA population is then seeded using the solution recommended by MCTS. Only the first relevant nodes are selected, by traversing the tree through the most visited actions (the same method used by algorithm MCTS when selecting its final action to play). A node is relevant if it has been visited at least $M = 3$ times. The rest of the individual (if any genes have not received a value) is padded with randomly chosen legal actions. If the population size is bigger than 1, all other individuals in the population are mutations of the first, similar to the 1SLA procedure. See Algorithm 4 for details of the overridden `initializePop` method from vanilla RHEA shown previously in Algorithm 1 (the rest of the algorithm continues as normal).

This technique uses $\text{budget}/2 + \text{length}(A) + P \times L$ FM calls from the budget for initialisation and evaluation of the initial population (where A is the sequence of relevant nodes recommended by MCTS, with maximum length L), and is thought to also aid in starting evolution from a better than random point in the solution space, but it would further obtain better statistics as to which initial solution is the best, as opposed to the greedy approach taken by 1SLA. This results in a budget and accuracy trade-off, which is highlighted by the decision to choose only nodes visited at least 3 times for the seeding (thus nodes which do have some statistics built up through multiple iterations). However, the individuals resulting from this seeding technique are also more likely to contain more variation from the random actions at the end, depending on how MCTS uses its computation time to expand the tree.

4.2.3 Results and Discussion

The analysis in this section uses a two-tailed Mann-Whitney non-parametric U test to measure the statistical significance of the results for each game (p -value = 0.05), applied to two performance indicators: win rate and game score achieved.

In general, both seeding techniques improve the performance of the vanilla algorithm much more when the population size and individual length are small than when they increase. This is due to less generations being evolved the larger the parameter values, reducing the impact of the seeding that can be observed. We have previously seen that very large population sizes and individual lengths, going to Random Search (RS) within budget limits, emerged as the best options. Random sampling of entire individuals, rather than the focused initial population offered by the seeding methods, are better at exploring more of the solution space and find those solutions that evolution could reach after several iterations, if available. However, the fact that performance is increased when several generations are evolved suggests that this focused start is indeed better for evolving action sequences - but not necessarily for finding the optimum.

Table 4.9 presents an overall win rate comparison between the two seeding variants and vanilla RHEA, across all games and configurations, while Figures 4.9,4.10 show a breakdown of win rate per game for

all methods tested. The bottom of Table 4.9 sums up the number of games in which one algorithm was significantly better than the other two, leading to a total of unique games where a significant improvement was noticed, in all configurations tested. Table 4.5 shows complete results for configuration $P = 1, L = 6$.

4.2.4 Overall Seeding Comparison

The general trend observed in this study is that the MCTS seeding variant performs significantly better than both algorithms *RHEA-R* and *RHEA-ISLA* in 8 unique games for win rate and 13 unique games for scores across all configurations, while being significantly worse than either of the other two in only 4 games for both win rate and score.

It is worth noting that there were a reduced number of games in which *RHEA-R* or *RHEA-ISLA* turned out to be consistently significantly better than *RHEA-MCTS*: “Escape” and “Wait for Breakfast” for both win rate and score and “Intersection” for score only. This is due to the traditional poor performance of MCTS in these games, also observed in Section 4.1, which is improved in the seeded algorithm over MCTS; evolution generally outperforms MCTS in games that require more careful planning of precise sequences of actions.

In addition, the MCTS seeding shows a steady improvement in several games. The win rates in the games “Aliens”, “Butterflies” and “Chopper” see an increase to very close to 100% in all configurations. These are games presenting highly dynamic environments in which accurate statistics of which actions are good are important. The biggest improvement is observed in “Chopper”, where the *RHEA-R* win rate for the smallest configuration ($P = 1, L = 6$) is only 26% to begin with ($p \ll 0.0001$).

This leads to the conclusion that identifying the type of game being played and applying the correct algorithm seeding and parameters through a meta-heuristic would be highly beneficial to a general AI agent. However, there are also games such as “Dig Dug”, “Lemmings” and “Roguelike” in which the win rate for all algorithms remains at 0%, these being particularly difficult games which require greater exploration of both solution and level space that neither technique can provide.

It is interesting to further observe change in win rate along with the increase in population size and individual length for all methods, as showcased in Figures 4.9, 4.10. For most games we see an upwards trend in win rate when no seeding is used (*RHEA-R* using default random initialisation), which is consistent with results presented in Section 4.1. However, the story is different for the seeded methods, which often see the opposite (“Chopper” for ISLA seeding and “Plaque Attack”, for example). This is not a linear drop, however, the curves for the different games (and different methods as well) peaking at different points, although most of these peaks happen when population size and individual length values are small. Some notable exceptions are “Plaque Attack”, “Modality”, “Survive Zombies”, “Seaquest” and “Missile Command” which prefer medium-high values for the parameters instead - all of these games are clustered together by (1) and they are mostly very dense and dynamic environments; we have seen previously that evolution can be tricked to evolve in the wrong direction in these types of games, thus it is not surprising that the algorithm struggles to find better solutions than the initial suggestions offered by the seeding methods.

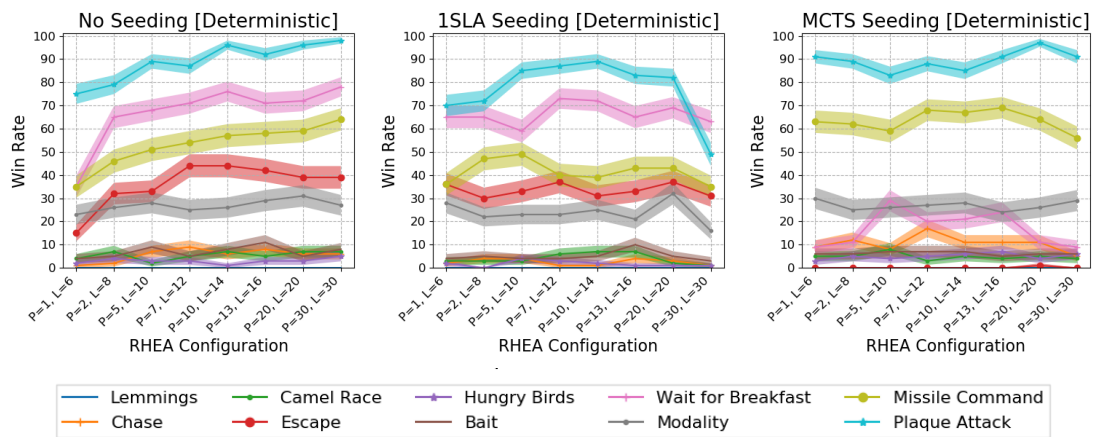


Figure 4.9: Change of win rate as population size and individual length increase, in all **deterministic** games tested. The standard error is shown by the shaded boundary.

Pair-wise Seeding Comparison

Pair-wise significance comparison between algorithms *RHEA-R*, *RHEA-ISLA* and *RHEA-MCTS* on all the configurations tested can be observed in Tables 4.6, 4.7 and 4.8. The values represent the number of games

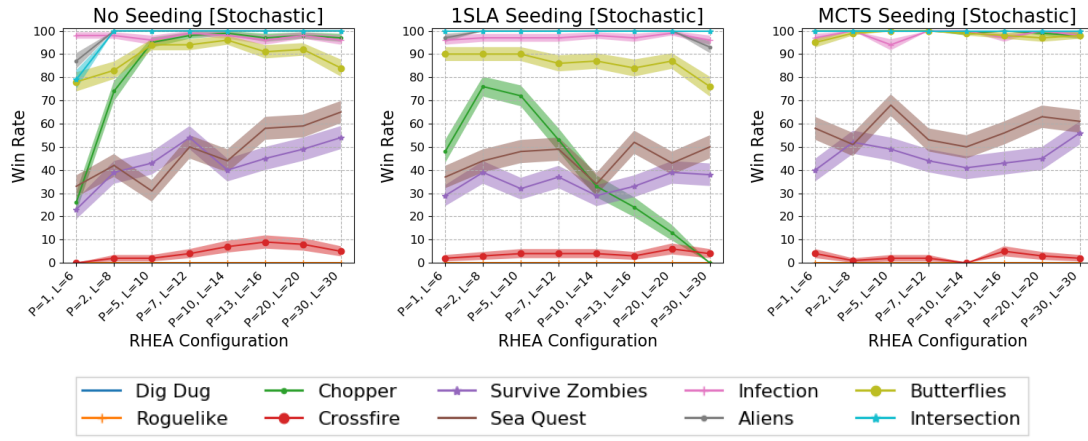


Figure 4.10: Change of win rate as population size and individual length increase, in all **stochastic** games tested. The standard error is shown by the shaded boundary.

Table 4.5: Win rate and average score achieved (plus standard error) in 20 different games with configuration $P = 1$ and $L = 6$. Bold font shows the algorithm that is significantly better than both others in either victories or score.

Initialisation	Game	Win Rate (%)	Score	Game	Win Rate (%)	Score
Random	Aliens	87.00 (3.36)	59.33 (1.62)	Infection	98.00 (1.40)	11.09 (0.61)
1SLA		97.00 (1.71)	61.95 (1.29)		96.00 (1.96)	11.84 (0.71)
MCTS		100.00 (0.00)	68.87 (1.52)		97.00 (1.71)	15.25 (0.86)
Random	Bait	4.00 (1.96)	2.10 (0.29)	Intersection	79.00 (4.07)	-3.03 (1.16)
1SLA		4.00 (1.96)	3.28 (0.51)		100.00 (0.00)	4.25 (0.69)
MCTS		6.00 (2.37)	3.41 (0.41)		100.00 (0.00)	1.00 (0.00)
Random	Butterflies	78.00 (4.14)	33.12 (1.60)	Lemmings	0.00 (0.00)	-9.11 (0.37)
1SLA		90.00 (3.00)	32.48 (1.65)		0.00 (0.00)	-0.11 (0.05)
MCTS		95.00 (2.18)	30.48 (1.46)		0.00 (0.00)	-0.03 (0.02)
Random	Camel Race	4.00 (1.96)	-0.76 (0.05)	Missile Command	35.00 (4.77)	1.47 (0.41)
1SLA		3.00 (1.71)	-0.77 (0.05)		36.00 (4.80)	1.72 (0.43)
MCTS		5.00 (2.18)	-0.75 (0.05)		63.00 (4.83)	4.65 (0.48)
Random	Chase	1.00 (0.99)	2.16 (0.20)	Modality	23.00 (4.21)	0.23 (0.04)
1SLA		2.00 (1.40)	2.14 (0.22)		28.00 (4.49)	0.28 (0.04)
MCTS		9.00 (2.86)	3.20 (0.24)		30.00 (4.58)	0.30 (0.05)
Random	Chopper	26.00 (4.39)	2.39 (0.62)	Plaque Attack	75.00 (4.33)	35.37 (1.60)
1SLA		48.00 (5.00)	4.63 (0.78)		70.00 (4.58)	33.05 (1.75)
MCTS		100.00 (0.00)	16.99 (0.28)		91.00 (2.86)	47.17 (1.87)
Random	Crossfire	0.00 (0.00)	-1.01 (0.01)	Roguelike	0.00 (0.00)	1.60 (0.37)
1SLA		2.00 (1.40)	-0.89 (0.08)		0.00 (0.00)	3.54 (0.52)
MCTS		4.00 (1.96)	0.18 (0.10)		0.00 (0.00)	5.44 (0.62)
Random	Dig Dug	0.00 (0.00)	5.66 (0.76)	Sea Quest	33.00 (4.70)	903.56 (127.82)
1SLA		0.00 (0.00)	9.15 (0.77)		37.00 (4.83)	1130.36 (137.78)
MCTS		0.00 (0.00)	14.93 (1.17)		58.00 (4.94)	1807.79 (177.44)
Random	Escape	15.00 (3.57)	-0.64 (0.07)	Survive Zombies	23.00 (4.21)	0.92 (0.38)
1SLA		36.00 (4.80)	0.34 (0.05)		29.00 (4.54)	0.92 (0.39)
MCTS		0.00 (0.00)	0.00 (0.00)		40.00 (4.90)	2.27 (0.41)
Random	Hungry Birds	2.00 (1.40)	2.00 (1.40)	Wait for Breakfast	35.00 (4.77)	0.35 (0.05)
1SLA		2.00 (1.40)	2.00 (1.40)		65.00 (4.77)	0.65 (0.05)
MCTS		3.00 (1.71)	4.60 (1.85)		9.00 (2.86)	0.09 (0.03)

(out of 20 total) in which one algorithm was significantly better than the other regarding victories, as well as scores, in brackets. The totals sum up the unique games in which one algorithm was significantly better than the other across all configurations (maximum of 20).

RHEA-R vs RHEA-1SLA The One Step Look Ahead seeding appears to produce the best results where the RHEA parameter values are very small (improvements in 6 games for win rate and 7 games for score, see Table 4.6). However, a change is noticed halfway through the table where the seeding variant actually becomes significantly worse than the vanilla version in up to 5 games for win rate and 10 games for score. On average, across all configurations tested, the 1SLA seeding appears to be worse than the baseline algorithm.

Table 4.6: Pairwise significance comparison between vanilla RHEA and 1SLA-seeded RHEA.

Initialisation	$P = 1, L = 6$	$P = 2, L = 8$	$P = 5, L = 10$	$P = 10, L = 14$	$P = 15, L = 16$	$P = 20, L = 20$	Total
Random	1 (1)	0 (0)	0 (1)	3 (5)	5 (8)	5 (10)	8 (11)
1SLA	6 (7)	1 (5)	0 (4)	0 (1)	0 (2)	0 (2)	6 (8)

Table 4.7: Significance comparison between vanilla RHEA and MCTS-seeded RHEA.

Initialisation	$P = 1, L = 6$	$P = 2, L = 8$	$P = 5, L = 10$	$P = 10, L = 14$	$P = 15, L = 16$	$P = 20, L = 20$	Total
Random	2 (1)	2 (3)	2 (3)	4 (3)	2 (4)	2 (4)	4 (5)
MCTS	10 (16)	6 (11)	4 (7)	1 (5)	2 (5)	0 (5)	12 (16)

Table 4.8: Significance comparison between 1SLA-seeded RHEA and MCTS-seeded RHEA.

Initialisation	$P = 1, L = 6$	$P = 2, L = 8$	$P = 5, L = 10$	$P = 10, L = 14$	$P = 15, L = 16$	$P = 20, L = 20$	Total
1SLA	2 (3)	2 (4)	2 (3)	3 (4)	2 (4)	2 (4)	3 (5)
MCTS	6 (11)	8 (11)	4 (9)	6 (12)	6 (11)	6 (11)	10 (13)

A study of the complete matrix of small parameter values ($P = \{1, 2, 5\}, L = \{6, 8, 10\}$), where the difference in performance is most observed, reveals that the variance in individual length and population size have different effects. On the one hand, increasing the size of the population results in a decrease in the number of games *RHEA-1SLA* is significantly better in when compared to *RHEA-R*, which is due to the fact that the seeding variant explores the search space much less, with only one optimal solution mutated for all of its individuals. On the other hand, the performance is proportional to the individual length, suggesting that the directed search provided by 1SLA is more effective in cases with big L values compared to *RHEA-R*'s random sampling.

For configuration $P = 20, L = 20$, the biggest significance is noticed in ‘‘Chopper’’, in which *RHEA-1SLA* reduces the baseline algorithm's performance from 98% to 13% ($p \ll 0.0001$). However, *RHEA-1SLA* succeeds in gaining a higher score in the low-scoring game ‘‘Intersection’’, increasing from 2.24 points to 11.34.

RHEA-R vs RHEA-MCTS With parameter values smaller than $P = 10, L = 14$, the MCTS seeding is significantly better than the vanilla version, the difference being most noticed, again, when the parameter values are smallest. The decrease in performance in larger values is thought to be caused by the rollout depth of MCTS exceeding the optimal range observed in GVGAI games (10 – 12). Across all configurations, MCTS seeding improves the baseline algorithm in 60% of the games for win rate and 80% for score, see Table 4.7.

Comparing the complete matrix of small parameter values shows that the population size has a much greater negative effect on the performance than the individual length. The lack of impact of the individual length can be explained by the proportional increase in the rollout length of MCTS, therefore keeping results comparable. However, the decrease observed with population size increase suggests that the algorithm fails to explore the solution search space as well as RHEA, therefore balancing of other parameters should be considered.

For configuration $P = 5, L = 10$, there are two interesting games to look in-depth at. In the game ‘‘Seaquest’’, *RHEA-MCTS* increases the win rate of the baseline algorithm from 31% to 68% ($p \ll 0.0001$) and the score from 1225.68 average points to 2485.43. Another big effect size is perceived in ‘‘Camel Race’’, in which, although the win rates remain small, there is an increase from 2% to 8% ($p = 0.026$). Both games highly benefit from the balanced exploration and exploitation provided by the MCTS solution which stands at the base of the evolutionary process.

RHEA-1SLA vs RHEA-MCTS When the two seeding techniques are compared directly, *RHEA-MCTS* achieves a consistently better performance in 50% of the games for win rate and 65% for scores, whereas being consistently significantly worse in the games ‘‘Escape’’ and ‘‘Wait for Breakfast’’ for both win rate and scores and game ‘‘Intersection’’ for score only, similar to previous observation (see Table 4.8. In ‘‘Chopper’’, *RHEA-1SLA* with configuration $P = 2, L = 8$ achieves a 76% win rate, while *RHEA-MCTS* increases it to 100% ($p \ll 0.0001$). In addition, significant improvements with large effect sizes can also be noticed in the games ‘‘Missile Command’’ ($p = 0.016$) and ‘‘Survive Zombies’’ ($p = 0.033$). However, in ‘‘Escape’’, in which the MCTS algorithm cannot find a solution, the win rate drops from 30% for *RHEA-1SLA* to 0% for *RHEA-MCTS* ($p \ll 0.0001$).

MCTS Comparison

Although the results indicate algorithm *RHEA-MCTS* to be the best in this setting, an analysis against a pure Monte Carlo Tree Search technique was carried out to validate the findings. This comparison can be seen in Table 4.10, in which the values show the number of games in which one algorithm was significantly

Table 4.9: Average of victories in all 20 games. **Bold style** indicates a significantly better average *win rate* than both the other two seeding variants. If the * **symbol** is present additionally, it indicates instead a significantly better average *score* than both the other two seeding variants. The bottom of the table adds up the number of games in which the algorithm was significantly better than the other two variants in average victories, with counts for games with significantly better average scores in brackets. The non-parametric Wilcoxon signed-rank test (p-value < 0.05) was used to test significance.

Game	Initialisation	$P = 1, L = 6$	$P = 2, L = 8$	$P = 5, L = 10$	$P = 10, L = 14$	$P = 15, L = 16$	$P = 20, L = 20$	Total
0	Random	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	ISLA	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	MCTS	0.00 (0.00)*	0.00 (0.00)*	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
1	Random	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	ISLA	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)*	0.00 (0.00)*	0.00 (0.00)*	0.00 (0.00)
	MCTS	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)*	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
2	Random	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	ISLA	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
	MCTS	0.00 (0.00)	0.00 (0.00)*	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)*	0.00 (0.00)
3	Random	26.00 (4.39)	74.00 (4.39)	95.00 (2.18)	99.00 (0.99)	97.00 (1.71)	98.00 (1.40)	81.5 (11.74)
	ISLA	48.00 (5.00)	76.00 (4.27)	72.00 (4.49)	33.00 (4.70)	24.00 (4.27)	13.00 (3.36)	44.33 (8.43)
	MCTS	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	99.00 (0.99)*	100.00 (0.00)	99.00 (0.99)*	99.67 (0.16)
4	Random	0.00 (0.00)	2.00 (1.40)	2.00 (1.40)	7.00 (2.55)	9.00 (2.86)	8.00 (2.71)	4.67 (1.39)
	ISLA	2.00 (1.40)	3.00 (1.71)	4.00 (1.96)	4.00 (1.96)	3.00 (1.71)	6.00 (2.37)	3.67 (0.33)
	MCTS	4.00 (1.96)*	1.00 (0.99)*	2.00 (1.40)*	0.00 (0.00)*	5.00 (2.18)*	3.00 (1.71)*	2.50 (0.76)
5	Random	1.00 (0.99)	2.00 (1.40)	7.00 (2.55)	6.00 (2.37)	8.00 (2.71)	6.00 (2.37)	5.00 (1.33)
	ISLA	2.00 (1.40)	4.00 (1.96)	4.00 (1.96)	1.00 (0.99)	4.00 (1.96)	3.00 (1.71)	3.00 (0.61)
	MCTS	9.00 (2.86)	12.00 (3.25)	8.00 (2.71)*	11.00 (3.13)	11.00 (3.13)*	11.00 (3.13)	10.33 (1.28)
6	Random	4.00 (1.96)	7.00 (2.55)	2.00 (1.40)	7.00 (2.55)	5.00 (2.18)	7.00 (2.55)	5.33 (0.77)
	ISLA	3.00 (1.71)	3.00 (1.71)	3.00 (1.71)	7.00 (2.55)	7.00 (2.55)	2.00 (1.40)	4.16 (0.83)
	MCTS	5.00 (2.18)	5.00 (2.18)	8.00 (2.71)	5.00 (2.18)	4.00 (1.96)	5.00 (2.18)	5.33 (0.68)
7	Random	15.00 (3.57)	32.00 (4.66)	33.00 (4.70)	44.00 (4.96)	42.00 (4.94)	39.00 (4.88)	34.16 (4.56)
	ISLA	36.00 (4.80)	30.00 (4.58)*	33.00 (4.70)	31.00 (4.62)	33.00 (4.70)	37.00 (4.83)	33.33 (1.11)
	MCTS	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	1.00 (0.99)	0.16 (0.16)
8	Random	2.00 (1.40)	4.00 (1.96)	3.00 (1.71)	1.00 (0.99)	3.00 (1.71)	3.00 (1.71)	2.67 (0.42)
	ISLA	2.00 (1.40)	0.00 (0.00)	4.00 (1.96)	2.00 (1.40)	1.00 (0.99)	1.00 (0.99)	1.67 (0.53)
	MCTS	3.00 (1.71)*	5.00 (2.18)	4.00 (1.96)	6.00 (2.37)	7.00 (2.55)	4.00 (1.96)	4.83 (0.57)
9	Random	4.00 (1.96)	5.00 (2.18)	9.00 (2.86)	8.00 (2.71)	11.00 (3.13)	5.00 (2.18)	7.00 (1.12)
	ISLA	4.00 (1.96)	5.00 (2.18)	4.00 (1.96)	5.00 (2.18)	10.00 (3.00)	5.00 (2.18)	5.5 (1.88)
	MCTS	6.00 (2.37)	6.00 (2.37)	7.00 (2.55)	7.00 (2.55)	5.00 (2.18)	6.00 (2.37)	6.16 (0.33)
10	Random	35.00 (4.77)	65.00 (4.77)	68.00 (4.66)	76.00 (4.27)	71.00 (4.54)	72.00 (4.49)	64.50 (6.05)
	ISLA	65.00 (4.77)	65.00 (4.77)	59.00 (4.92)	72.00 (4.49)	65.00 (4.77)	69.00 (4.62)	65.83 (2.12)
	MCTS	9.00 (2.86)	11.00 (3.13)	29.00 (4.54)	21.00 (4.07)	24.00 (4.27)	11.00 (3.13)	17.50 (3.13)
11	Random	23.00 (4.21)	39.00 (4.88)	43.00 (4.95)	40.00 (4.90)	45.00 (4.97)	49.00 (5.00)	39.83 (4.15)
	ISLA	29.00 (4.54)	39.00 (4.88)	32.00 (4.66)	29.00 (4.54)	33.00 (4.70)	39.00 (4.88)	33.50 (1.68)
	MCTS	40.00 (4.90)*	52.00 (5.00)	49.00 (5.00)	41.00 (4.92)*	43.00 (4.95)	45.00 (4.97)	45.00 (1.92)
12	Random	23.00 (4.21)	26.00 (4.39)	28.00 (4.49)	26.00 (4.39)	29.00 (4.54)	31.00 (4.62)	27.16 (0.87)
	ISLA	28.00 (4.49)	22.00 (4.14)	23.00 (4.21)	25.00 (4.33)	21.00 (4.07)	32.00 (4.66)	25.16 (1.02)
	MCTS	30.00 (4.58)	25.00 (4.33)	26.00 (4.39)	28.00 (4.49)	24.00 (4.27)	26.00 (4.39)	26.50 (0.88)
13	Random	35.00 (4.77)	46.00 (4.98)	51.00 (5.00)	57.00 (4.95)	58.00 (4.94)	59.00 (4.92)	51.16 (3.51)
	ISLA	36.00 (4.80)	47.00 (4.99)	49.00 (5.00)	39.00 (4.88)	43.00 (4.95)	43.00 (4.95)	42.83 (2.02)
	MCTS	63.00 (4.83)	62.00 (4.85)	59.00 (4.92)	67.00 (4.70)	69.00 (4.62)	64.00 (4.80)	64.00 (1.60)
14	Random	75.00 (4.33)	79.00 (4.07)	89.00 (3.13)	96.00 (1.96)	92.00 (2.71)	96.00 (1.96)*	87.83 (3.24)
	ISLA	70.00 (4.58)	72.00 (4.49)	85.00 (3.57)	89.00 (3.13)	83.00 (3.76)	82.00 (3.84)	80.16 (3.27)
	MCTS	91.00 (2.86)	89.00 (3.13)	83.00 (3.76)	85.00 (3.57)	91.00 (2.86)	97.00 (1.71)	89.33 (1.32)
15	Random	33.00 (4.70)	42.00 (4.94)	31.00 (4.62)	44.00 (4.96)	58.00 (4.94)	59.00 (4.92)	44.50 (4.16)
	ISLA	37.00 (4.83)	44.00 (4.96)	48.00 (5.00)	34.00 (4.74)	52.00 (5.00)	43.00 (4.95)	43.00 (2.90)
	MCTS	58.00 (4.94)	51.00 (5.00)	68.00 (4.66)	50.00 (5.00)	56.00 (4.96)	63.00 (4.83)	57.67 (2.69)
16	Random	98.00 (1.40)	98.00 (1.40)	96.00 (1.96)	98.00 (1.40)	96.00 (1.96)	98.00 (1.40)	97.33 (0.57)
	ISLA	96.00 (1.96)	97.00 (1.71)	97.00 (1.71)	98.00 (1.40)	97.00 (1.71)	99.00 (0.99)	97.33 (0.33)
	MCTS	97.00 (1.71)*	100.00 (0.00)*	94.00 (2.37)*	100.00 (0.00)	97.00 (1.71)	100.00 (0.00)	98.00 (1.00)
17	Random	87.00 (3.36)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	97.83 (2.16)
	ISLA	97.00 (1.71)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	99.50 (0.50)
	MCTS	100.00 (0.00)	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)
18	Random	78.00 (4.14)	83.00 (3.76)	94.00 (2.37)	96.00 (1.96)	91.00 (2.86)	92.00 (2.71)	89.00 (2.94)
	ISLA	90.00 (3.00)	90.00 (3.00)	90.00 (3.00)	87.00 (3.36)	84.00 (3.67)	87.00 (3.36)	88.00 (1.04)
	MCTS	95.00 (2.18)	99.00 (0.99)	100.00 (0.00)	99.00 (0.99)	98.00 (1.40)	97.00 (1.71)	98.00 (0.76)
19	Random	79.00 (4.07)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	96.50 (3.50)
	ISLA	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)*	100.00 (0.00)
	MCTS	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)	100.00 (0.00)
Total	Random	0 (0)	0 (0)	0 (0)	2 (0)	0 (0)	0 (1)	2 (1)
	ISLA	2 (3)	0 (2)	0 (1)	0 (2)	0 (2)	0 (2)	2 (4)
	MCTS	6 (11)	6 (10)	3 (7)	0 (5)	2 (4)	0 (4)	8 (13)

Table 4.10: Significance comparison of algorithms *RHEA-R*, *RHEA-MCTS* and MCTS in all 20 games and all configurations.

Algorithm	$P = 1, L = 6$	$P = 2, L = 8$	$P = 5, L = 10$	$P = 10, L = 14$	$P = 15, L = 16$	$P = 20, L = 20$	Total
<i>RHEA-R</i>	2 (1)	2 (3)	2 (3)	2 (3)	2 (4)	2 (4)	3 (4)
<i>RHEA-MCTS</i>	0 (2)	1 (1)	0 (3)	0 (0)	0 (1)	0 (1)	1 (7)
MCTS	2 (3)	0 (0)	0 (1)	1 (3)	0 (3)	0 (3)	3 (7)
Improved Seeding	10 (15)	4 (10)	2 (4)	0 (5)	2 (5)	0 (4)	10 (15)

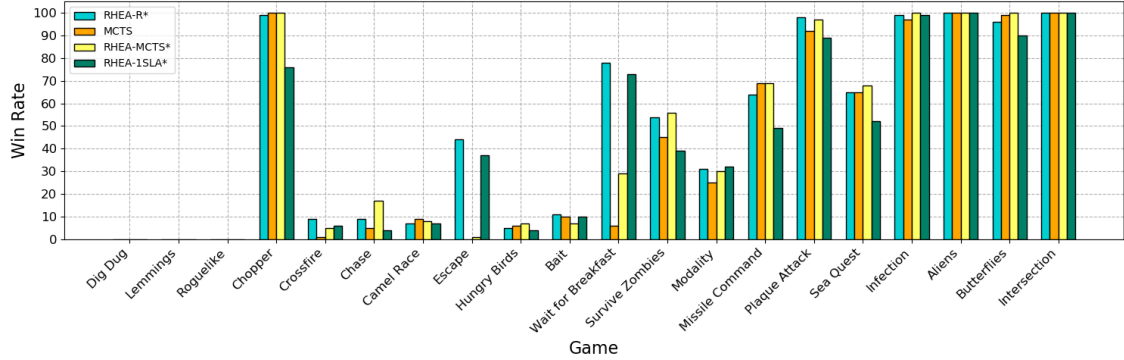


Figure 4.11: Win rate per game for *RHEA-R*, MCTS, *RHEA-MCTS* and *RHEA-ISLA*. * symbol shows the best result per game over all configurations tested for that algorithm and therefore does not reflect the performance of a single algorithm.

better than both the other MCTS variant and vanilla RHEA in win rate (and scores, in brackets), adding up to a total of unique games across all configurations.

The bottom line of Table 4.10 shows the number of games in which, although *RHEA-MCTS* was not the best algorithm, the addition of MCTS seeding to RHEA made it in turn better than the baseline algorithm. This takes into account the cases where *RHEA-MCTS* and MCTS were not significantly better than each other, but they still achieved a higher performance than *RHEA-R*, for a more complete picture.

While *RHEA-R* consistently obtains significantly more victories and higher scores in its best games (“Escape”, “Wait for Breakfast” and “Intersection”), it must be highlighted that the apparent low performance of *RHEA-MCTS* is due to it not always being significantly better than MCTS. For the direct comparison between *RHEA-MCTS* and *RHEA-R*, the reader is referred to Table 4.7. In this case, the MCTS seeding variant shows improvement over a wider range of games, adding up to 50% games in which a larger win rate was observed and 75% games in which the score increased. The conclusion emerging is that MCTS seeding has a highly beneficial effect, especially in low RHEA parameter values, and further exploration of its advantages is encouraged.

Finally, Figure 4.11 shows an overall picture of the algorithms tested in this section, aggregating results over all configurations per algorithm and showcasing the best result per algorithm, for each game. MCTS is rarely better than the vanilla RHEA or the MCTS-seeded RHEA, and vanilla RHEA is further best on most games. However, there are a few interesting cases to note: in the games “Chase”, “Survive Zombies” and “Seaquest”, the MCTS-seeded variant appears to combine the benefits of both RHEA and MCTS the best so that it obtains results better than either of the other methods (and, in fact, significantly improves the quality of the initial solution recommended by MCTS through evolution). These games do not appear to have much in common (see Table 2.1), therefore further investigation is encouraged into the reasons behind this hybrid doing very well in specific cases. This finding accentuates the need for highly adaptive methods that can make use of the strengths of different approaches to succeed in unique circumstances.

4.3 Hybrids

The work in this section was published at IEEE CIG 2017:

—, “Rolling Horizon Evolution Enhancements in General Video Game Playing,” in *Proceedings of IEEE Conference on Computational Intelligence and Games*, Aug 2017, pp. 88–95.

This section aims to explore four enhancements to the vanilla RHEA. Some of the enhancements presented here have been seen in the literature before, either in a General Video Game Playing (GVGP) setting, or in some other domains. However, they have been previously employed under different conditions, heuris-

tics and sets of games, and sometimes combined with other techniques. It is therefore very hard to deduct (if not impossible) which ones of these approaches work well in isolation, which ones do not produce improvements in the vanilla form of the algorithm, if decoupled from the heuristics used, and which ones could work better if put together in combination.

The objective of this section is to formalise and provide a fair analysis on these enhancements. They are all tested in isolation, but also combinations between them are drawn in order to identify potentially good synergies. Furthermore, they are evaluated under the same testing circumstances, provided with a common heuristic to evaluate states visited during search, in the 20 GVGAI games carefully selected to serve as a good representative set of the whole GVGAI corpus.

The baseline algorithm is *Vanilla RHEA*. The population initialisation is kept pseudo-random (each individual receiving random actions for each gene, in the range $0 - (N - 1)$, where N is the number of legal actions in the current game state (therefore each gene corresponds to one in-game action)).

Breeding occurs $P - E$ times in one generation, where E represents elitism (the chosen method for promoting the best individuals, unchanged, to the next generation. $E = 1$ for all cases) and P population size. Each new individual in a subsequent generation is the product of uniform crossover between individuals from the previous generation, selected through tournament (size 2), and mutation (random).

The heuristic used to evaluate game states and determine individual fitness simply returns the game score, dynamically normalised between (0, 1), 1 for winning and 0 for losing. The process of evaluating an individual is as described in Section 3.1.

In the rest of this section, the term “configurations” will refer to different values for the population size (P) and individual length (L) parameters and the term “variants” will refer to RHEA algorithms with enhancements added to the vanilla version. If more than one enhancement is used, the term “hybrid” may be used instead. Several variations of the vanilla RHEA algorithm were analysed on a set of 20 games, playing 20 times on all 5 levels of each game (therefore 100 runs per game per algorithm). Additionally, 4 different core parameter configurations ($P-L = \{1-6, 2-8, 5-10, 10-14\}$) were used for all algorithms, in order to observe the effect of the enhancements across a range of parameter values, comparable with the results presented in previous sections.

The budget given to each algorithm was restricted to 900 FM calls (the average obtained by vanilla RHEA in the current GVGAI corpus), so as to eliminate bias from variations in the machine used to run the experiments. The maximum configuration tested was 10-14 due to the fact that if it were larger, by adding

Algorithm 5 RHEA Hybrids Main

```

1: procedure MAIN( $s_t$ ,  $budget$ )
2:    $k \leftarrow 0$ 
3:   if  $t = 0$  then
4:      $P_k \leftarrow initializePop(budget)$ 
5:      $initialize\ bandits$  ▷ New bandit for population, new bandit for each gene
6:      $initialize\ stat\_tree$  ▷ Empty tree with root from  $s_t$ 
7:   else
8:      $P_k \leftarrow SHIFT(s_t, budget, P_k[t - 1])$  ▷ Shift population from previous tick
9:      $shift\ bandits$  ▷ Bandit for action played removed, new bandit for last action
10:     $shift\ stat\_tree$  ▷ Root becomes previous action recommended
11:  while  $budget \neq 0$  do
12:     $P_{k+1}.add(\text{first } E \text{ individuals from } P_k)$ 
13:     $O \leftarrow \text{new array}$ 
14:    for  $j = 0 : pop\_size$  do
15:       $p_1, p_2 \leftarrow selectParents(P_k)$ 
16:       $I' \leftarrow cross(p_1, p_2)$ 
17:       $I', j', a' \leftarrow BANDITMUTATE(O)$  ▷ Bandit mutation
18:       $f = evaluate(I', s_t, budget)$ 
19:       $update(ind\_bandit, j', f)$  ▷ Update chosen ind\_bandit arm  $Q$ ,  $N$ ,  $N\_a$  values
20:       $update(gene\_bandit[j'], a', f)$  ▷ Update chosen gene\_bandit[j'] arm  $Q$ ,  $N$ ,  $N\_a$  values
21:       $O[j] \leftarrow I'$ 
22:     $pool \leftarrow P_k + O$ 
23:     $sort(pool)$ 
24:     $P_{k+1}.add(\text{first } P - E \text{ individuals from } pool)$ 
25:     $k \leftarrow k + 1$ 
26:   $a_t \leftarrow \arg \max_{a \in stat\_tree.children[root]} N(s, a)$  ▷ Return most visited action from the stats tree
27:  return  $a_t$ 

```

Algorithm 6 RHEA Hybrids Functions

```
1: procedure SHIFT( $s_t, budget, P_{t-1}$ )
2:    $P \leftarrow \text{new array}$ 
3:    $n\_actions \leftarrow s_t.nActionsAvailable()$ 
4:   for  $k = 0 : pop\_size$  do
5:      $I \leftarrow \text{new array}$ 
6:     for  $j = 0 : ind\_length - 1$  do
7:        $I[j] \leftarrow P_{t-1}[k][j + 1]$ 
8:        $I[ind\_length - 1] = \text{random}(0, n\_actions)$ 
9:        $evaluate(I, s_t, budget)$ 
10:     $P[k] \leftarrow I$ 
11:   return  $P$  ▷ Shifted previous population
12:
13: procedure BANDITMUTATE( $I, s$ )
14:    $n\_actions \leftarrow s.nActionsAvailable()$ 
15:   ▷ Choose gene in individual to mutate with highest UCB value, Equation 3.3
16:    $j' \leftarrow \arg \max_{j \in I.genes} ind\_bandit.UCB(j)$ 
17:   ▷ Choose new value for gene with highest UCB value, Equation 3.3
18:    $a' \leftarrow \arg \max_{a \in n\_actions} gene\_bandit.UCB(a)$ 
19:    $I[j'] = a'$ 
20:   return  $I, j', a'$ 
21:
22: procedure EVALUATE( $I, s_t, budget$ )
23:   for  $j = 0 : ind\_length$  do
24:      $s_{t+j+1} \leftarrow s_{t+j}.advance(I[j])$ 
25:      $budget \leftarrow budget - 1$ 
26:      $f \leftarrow \text{MCSIMULATION}(budget, s_{t+ind\_length})$  ▷ Runs MC simulation from last state reached
27:      $stat\_tree.UPDATE(I, f)$  ▷ Adds nodes to tree, or updates  $Q, N, N\_a$  for all actions in individual
28:   return  $f$ 
29:
30: procedure MCSIMULATION( $budget, s$ )
31:    $n\_actions \leftarrow s.nActionsAvailable()$ 
32:    $f \leftarrow \text{new array}$ 
33:   for  $r = 0 : R$  do
34:      $s' \leftarrow s.copy$ 
35:     for  $j = 0 : sim\_length$  do
36:        $s' \leftarrow s'.advance(\text{random}(0, n\_actions))$ 
37:        $budget \leftarrow budget - 1$ 
38:      $f[r] \leftarrow h(s')$  ▷ Use Equation 2.1
39:   return  $\bar{f}$ 
```

rollouts, the limited budget would not allow for even one full population to be evaluated in one game tick.

There are two main parts to the experiments run for this study, the second of which includes a comparison with MCTS. The results presented in Section 4.3.1 correspond to this setup. Algorithm 5 shows the modified main function for RHEA with all modifications introduced, and Algorithm 6 shows the modified function for evaluating individuals, as well as additional functionality introduced by the modifications.

The first part of the experiments explores the first three enhancements described in Section 3.1 (bandit-based mutation, referred to as *EA-bandit*; stats tree, referred to as *EA-tree*; and shift buffer, referred to as *EA-shift*) in isolation, as well as combinations of them, resulting in 8 variants. The best variants in all configurations (4 in total) were kept for the second part.

The second part of experiments looks at the last enhancement (rollouts, referred to as *EA-roll*, see Section 4.3.1), added to the 4 variants promoted previously. Three different values for rollout repetitions were considered: $R = \{1, 5, 10\}$. This resulted in 8 algorithms (with and without rollouts) analysed in this section for each rollout length, therefore 24 per configuration. Finally, the algorithms were compared with MCTS in order to validate their quality on a larger scale in GVGAI.

4.3.1 Results and Discussion

The results presented in this section are based on both rankings following the Formula-1 (F1) point system and a significance comparison in win rate or scores, using a Mann-Whitney non-parametric U test, with $p\text{-value} < 0.05$. The F1 point system is used in GVGAI as a measure of generality: all algorithms are ranked on each of the games played based on win rate (and scores, then game ticks for tie breakers; highest values first), then each rank receives an associated number of points: first receives 25 points, second 18, third 15, then 12, 10, 8, 6, 4, 2 and 0 for all remaining algorithms. The sum of points across all games played decides the overall ranking of algorithms on a given game set and is meant to promote algorithms that perform well in the most games, as opposed to those that are very good in some, but the worst in others. It is important to highlight that this generality measure is highly dependent on the approaches a particular algorithm is compared against and does not paint an absolute picture of an algorithm's overall performance.

Bandit, tree, shift

Overall, in the first part of experiments, the shift buffer appears to offer the biggest improvement in performance, while the bandit-based mutation is in many cases significantly worse than all other algorithms. If all variants across all configurations were to be compared and ranked according to F1 points, *EA-shift* (5-10) would be in first place, with 213 points and 40.05% average win rate, while *EA-bandit* (1-6) would be last, with 0 points and 29.65% win rate. The specific results for configuration 5-10 are depicted in Table 4.11.

The effect of increasing the population size and individual length is noticed in most variants. Although the win rate sees an overall increase proportional to parameter values, the algorithm ranking does not remain consistent.

Figure 4.12 presents the significant wins of all variants in all configurations, counting for each pair in how many games the row algorithm was significantly better than the column one; the darker the colour, the higher the game count. A dark row would therefore signify an algorithm better than the others in most games, while a dark column would mean the algorithm performed worse. It is worth observing how bandit hybrids feature dark columns in most configurations, as well as how *EA-shift* and *EA-tree-shift* rows stand out as the best.

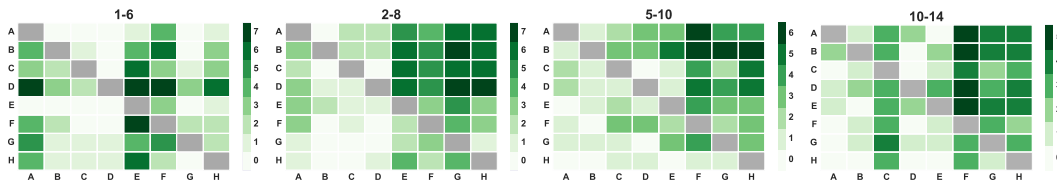


Figure 4.12: Win percentage for all configurations. The colour bar denotes in how many unique games row was significantly better than column. Legend: A = Vanilla, B = EA-shift, C = EA-tree, D = EA-tree-shift, E = EA-bandit, F = EA-bandit-shift, G = EA-bandit-tree, H = EA-bandit-tree-shift

An interesting game to look at in more detail is game 60 (“Missile Command”), where no significance can be observed in low configurations, but in higher ones *EA-shift* is significantly better than vanilla in win rates and all shift hybrids are better than vanilla in scores; *EA-bandit* is significantly worse than both shift and tree hybrids. In the game “Escape”, all variants are significantly better than vanilla in both win rates and scores, except for tree hybrids, in low configurations, while no significance is observed at the opposite end of the spectrum.

In most games, the shift enhancement is significantly better across configurations, *EA-shift* (2-8 and higher) being able to match and surpass the performance of the best *Vanilla RHEA* (10-14). This is a critical finding of this study: the simple shift buffer enhancement, which requires little extra computation time, allows for much better performance without needing to increase core parameter values.

The best 4 algorithms carried forward to the second part of experiments are *EA-shift*, *EA-tree-shift*, *EA-tree* and *Vanilla*.

EA-bandit

The *EA-bandit* algorithm is one of the worst variants tested in this study. In all configurations, it performed worse than *Vanilla* and, in the smallest configuration (1-6), it was outperformed by all of the bandit hybrids as well. However, in higher configurations it increases its average win rate significantly, from 29.65% to 38.50% and even outperforms *EA-tree* in the largest configuration (10-14).

In the game “Plaque Attack”, *EA-bandit* attains a significantly better win rate than most algorithms, increasing from 69% (*Vanilla*, 1-6) to 98% (10-14). However, in the game “Intersection, *EA-bandit* (1-6) is

Table 4.11: Configuration 5-10. Rankings table for part 1 algorithms across all games. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3	G-4	G-5	G-6	G-7	G-8	G-9	G-10	G-11	G-12	G-13	G-14	G-15	G-16	G-17	G-18	G-19
1	EA-shift	373	40.05 (2.50)	25	25	25	25	12	25	15	18	12	15	18	25	6	25	25	8	18	18	18	15
2	EA-tree-shift	293	37.20 (2.48)	10	6	18	8	18	18	18	10	25	8	8	15	12	18	18	25	10	25	15	8
3	Vanilla	290	38.75 (2.53)	18	18	12	18	4	12	8	15	18	10	25	18	10	15	12	18	12	12	10	25
4	EA-tree	243	35.25 (2.36)	15	15	15	15	15	10	12	4	6	12	4	12	18	6	15	15	8	15	25	6
5	EA-bandit-shift	219	32.00 (2.28)	4	8	8	6	8	4	25	25	15	25	10	6	8	4	10	6	25	8	4	10
6	EA-bandit	206	36.35 (2.52)	12	12	10	10	6	15	6	12	10	4	12	10	25	8	8	12	4	10	8	12
7	EA-bandit-tree	174	34.65 (2.44)	8	10	6	12	25	6	4	6	8	18	6	4	4	12	6	10	15	4	6	4
8	EA-bandit-tree-shift	162	33.40 (2.17)	6	4	4	4	10	8	10	8	4	6	15	8	15	10	4	4	6	6	12	18

one of the only algorithms achieving a win rate under 100% (only 89%, the others being *Vanilla*, 91% and *EA-bandit-shift*, 99%).

The worst bandit hybrid is *EA-bandit-shift*, achieving only 32.05% win rate in even the best configuration tested. Bandit mutation is generally most beneficial in larger configurations.

EA-tree

Across all games, *EA-tree* is better than *Vanilla* in the lower half of the configurations tested. In an overall view of all algorithms and configurations, the tree hybrids rank mid-table, outperforming *EA-bandit*, but not *EA-shift*.

There are several games in which *EA-tree* is significantly better than *Vanilla* in either wins or score, although this effect is mostly observed in low configurations, in games such as “Crossfire” and “Butterflies”. However, in “Escape”, *EA-tree* is significantly worse than all other algorithms in win rate, across all configurations.

The worst tree hybrids are *EA-bandit-tree* and *EA-bandit-tree-shift*, both being very close in performance on configuration 10-14, while *EA-tree* ranks second. The statistical tree appears to be most beneficial in low configurations.

EA-shift

The shift buffer is the best enhancement analysed. It outperforms *Vanilla* in all configurations, ranking first in all but 1-6, where tree hybrids are better. Overall, the shift buffer hybrids are good at achieving significantly higher scores than all others. As many games rely on this aspect, the win rates also increase, although not as dramatically.

For example, in “Dig Dug”, *EA-shift* (10-14) is significantly better in score than all other algorithms in all configurations. In “Infection”, even though it is again significantly better than all others in scores (10-14), its win rate is significantly worse than most others. Nevertheless, in the games “Wait for Breakfast” (10-14) and “Aliens” (1-6), *EA-shift* and its hybrids see a significant increase in both win rates and scores (from 86% to 100% win rate in “Aliens”, $p \ll 0.001$).

The worst shift buffer hybrids are *EA-bandit-shift* and *EA-bandit-tree-shift*, which achieve a much lower win rate. This leads to the conclusion that combining bandit mutation with a shift buffer (possibly due to the old information stored by the bandits) is not favourable in this setting.

EA-roll and its hybrids

The overall results of the second part of experiments suggest that the shift buffer enhancement is even better when combined with rollouts, *EA-shift-roll* being the dominating algorithm, while *Vanilla* and *EA-roll* rank the lowest in most settings. It is interesting to observe the gradual increase in performance of *EA-shift* in all R values, being last out of the shift hybrids in 1-6, but moving up in the rankings with the increase in core parameters. Rollouts seem most advantageous in low configurations, as they become too expensive to compute in the limited budget when individual length grows.

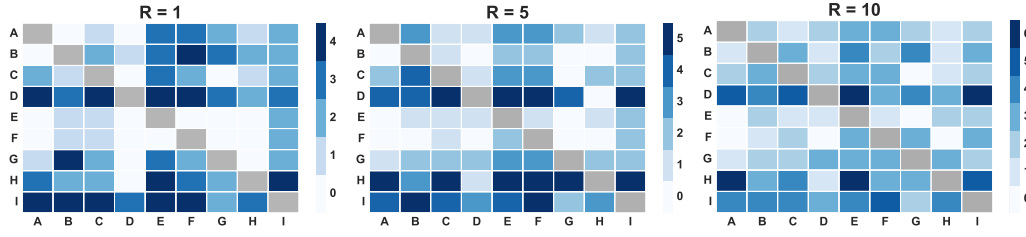


Figure 4.13: Win percentage for configuration 10-14. The colour bar denotes in how many unique games row was significantly better than column. Legend: A = Vanilla, B = EA-roll, C = EA-shift, D = EA-shift-roll, E = EA-tree, F = EA-tree-roll, G = EA-tree-shift, H = EA-tree-shift-roll, I = MCTS

Table 4.12: The best algorithms (by Formula-1 points and win rate) in all configurations and rollout repetitions (R), as compared against the other variants in the same configuration and the same R value (includes variants without rollouts).

Config.	R	Best By F1 Points		Best By Win Rate	
		Algorithm	Avg. Wins	Algorithm	Avg. Wins
1-6	1	EA-shift-roll	38.35 (2.31)	EA-tree-shift-roll	38.60 (2.55)
	5	EA-shift-roll	40.10 (2.51)	EA-shift-roll	40.10 (2.51)
	10	EA-shift-roll	39.35 (2.64)	EA-shift-roll	39.35 (2.64)
2-8	1	EA-shift-roll	40.35 (2.63)	EA-shift-roll	40.35 (2.63)
	5	EA-shift-roll	40.75 (2.46)	EA-shift-roll	40.75 (2.46)
	10	EA-shift-roll	40.20 (2.30)	EA-shift-roll	40.20 (2.30)
5-10	1	EA-shift-roll	43.20 (2.43)	EA-shift-roll	43.20 (2.43)
	5	EA-shift	40.05 (2.50)	EA-shift-roll	41.85 (2.42)
	10	EA-shift	40.05 (2.50)	EA-shift	40.05 (2.50)
10-14	1	EA-shift	39.75 (2.54)	EA-shift-roll	42.80 (2.44)
	5	EA-shift-roll	42.05 (2.48)	EA-tree-shift-roll	42.70 (2.41)
	10	EA-shift-roll	42.35 (2.53)	EA-shift-roll	42.35 (2.53)

EA-tree-roll performs the worst out of the tree hybrids in all configurations and R values, indicating that the deeper look into the future provided by the rollouts does not have a positive impact on the tree statistics. The best tree hybrid is *EA-tree-shift-roll*, surpassing the variant without rollouts.

Figure 4.13 presents the significant wins of all variants in configuration 10-14, with the different repetitions $R = \{1, 5, 10\}$. MCTS is included for comparison as the last row/column. It is interesting to note that *EA-shift-roll* is significantly better than most other algorithms in all R values, matching the performance of *MCTS*, but the most in $R = 5$, then decreasing in $R = 10$. This suggests that the ideal value peaks in the vicinity of 5. *EA-tree* and *EA-tree-roll* also stand out as the worst algorithms in all R variations tested.

The good performance of *EA-shift-roll* is also highlighted in Table 4.12, which summarises the best algorithm in each configuration and R value by both F1 points and win rate. The specific amount of points are not presented due to their high dependence on the other algorithms in the rankings and point distribution, therefore not being comparable independently. *EA-shift-roll* stands out as dominating most settings by both F1 points and win rate, with few exceptions.

One of the interesting games to look at in more detail is “Bait”, which has a generally low win rate. However, both *EA-shift-roll* and *EA-tree-shift-roll* achieve win rates of 19-20% in all R values for configuration 10-14 and 12-14% for 1-6, significantly higher compared to 1-3% of *Vanilla* and *EA-roll*. In “Chopper”, *EA-shift-roll* is significantly better in both win rate and scores than most other algorithms in all configurations and R values, the highest win rate being 54% in 1-6, $R = 5$, compared to 35% maximum for *EA-roll* (10-14, $R = 10$). The high improvement in low configurations is of specific interest, as it allows more thinking time in other parts of the evolutionary process for more complex computations.

Comparison with MCTS

Finally, we carried out a comparison with MCTS, the dominant technique in GVGAI. Overall, only few of the RHEA variants succeed in significantly outperforming MCTS. However, Table 4.13 shows the direct contrast between the best RHEA variant found during these experiments in terms of generality (thus highest F1 points in individual juxtaposition against the other algorithms), *EA-shift-roll* (10-14, $R = 5$) and MCTS

Table 4.13: Configuration 10-14, $R = 5$. Best algorithm found (EA-shift-roll) compared with MCTS. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3	G-4	G-5	G-6	G-7	G-8	G-9	G-10	G-11	G-12	G-13	G-14	G-15	G-16	G-17	G-18	G-19
1	EA-shift-roll	430	42.05 (2.48)	25	18	18	18	18	18	25	25	25	25	25	18	25	25	25	18	18	18	18	25
2	MCTS	430	41.30 (1.76)	18	25	25	25	25	25	18	18	18	18	18	25	18	18	18	25	25	25	25	18

Table 4.14: Configuration 10-14, $R = 1$. Algorithm most similar to MCTS (EA-tree-roll) compared with MCTS. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.

#	Algorithm	Points	Avg. Wins	G-0	G-1	G-2	G-3	G-4	G-5	G-6	G-7	G-8	G-9	G-10	G-11	G-12	G-13	G-14	G-15	G-16	G-17	G-18	G-19
1	MCTS	451	41.30 (1.76)	25	25	25	25	25	25	18	18	25	18	18	25	18	25	18	25	25	25	25	18
2	EA-tree-roll	409	35.90 (2.27)	18	18	18	18	18	18	25	25	18	25	25	18	25	18	25	18	18	18	18	25

(with a comparable rollout length of 14). Highlighted are the games in which one algorithm is better than the other (even if the difference is not significant).

EA-shift-roll matches the generality of MCTS, achieving the same amount of F1 points, but a higher win rate. When looking at individual games, it becomes clear that this RHEA variant is significantly better than MCTS in 5 games for win rate and 6 games for scores, while being significantly worse in 3 and 6 games, respectively. For example, in game 4, MCTS achieves a win rate of 6%, while *EA-shift-roll* obtains 19% ($p = 0.003$).

Table 4.14 shows the comparison between the RHEA variant considered most similar to MCTS (*EA-tree-roll* with $R = 1$) in its best configuration, 10-14, and MCTS (with a rollout length of 14). The fact that the tree is updated passively alongside the RHEA population and it is only used at the end of the evolutionary process to select which action to play leads to a significantly lower performance than MCTS in 5 games for win rate and 11 games for score.

However, there are several games where *EA-tree-roll* is significantly better in terms of win rate: game 36 ($p = 0.012$), a game in which EAs traditionally do better than tree search, and game 91 ($p \ll 0.001$).

4.4 Conclusions

This chapter presented three pieces of work testing the performance of the algorithm in a subset of 20 games of the General Video Game AI corpus, selected based on their difficulty and game features, in order to present a reduced set of challenges as assorted as possible. Various parameters and modifications were tested in these common environments and the same experimental settings, so as to observe baseline performance across several games.

Population size and individual length. First, we detail an analysis of two key parameters for the vanilla version of the Rolling Horizon Evolutionary Algorithm (RHEA): population size and individual length. One of the main findings of this research is the fact no single configuration of RHEA is able to find better solutions than Random Search (RS) in the settings explored, being worse than RS in many cases. However, the different configurations exploring different parts of the level or solution spaces show different strengths in the different games, and the best results over all configurations are better than RS in individual games. These results do suggest that the vanilla version of the algorithm is not able to explore the search space quickly enough given the limited budget, and it may also evolve in the wrong direction given misleading evaluations of individuals, especially in highly stochastic games. Therefore, this finding motivates research in RHEA, in order to find operators and techniques able to evolve sequences of actions in a more efficient and reliable manner. The results presented with higher execution budgets are an indication that this is possible, as well as showcasing the possibility of attacking previously unsolvable problems.

At the same time, this experiment highlighted another interesting conclusion: given the same length for the sequence of actions and the same budget (480 calls to the forward model), RHEA is able to outperform Open Loop Monte Carlo Tree Search (MCTS) when configured with a high population size. In particular, MCTS is outperformed by RHEA in deterministic games, especially those with puzzle elements or requiring precise navigation of complex environments. As most of the entries of the GVGAI competition, including

some of the winners, use MCTS or similar tree search methods as the basis for their entries, RHEA presents itself as a valuable alternative with a potentially promising future.

Finally, this study analyses the performance of the different versions of the algorithm in a game per game basis, and it is clear that in some games the agent performance increases along with the increase in values for the population size or the individual length. For instance, in most games the agent benefits from using larger populations (although exceptions do exist). Similarly, a long sequence of actions typically helps in giving a better indication of rewards available to the agents through more ample exploration of the level space, but some games form the exception and RHEA performs better with shorter individual lengths, especially if more accurate statistics are needed on which moves are best (such as in highly dynamic environments with dense rewards). In general, however, it has been observed that an increase in the population size has a higher impact on the performance than considering a farther lookahead (longer individuals); therefore, exploration of the solution space is more important than exploration of the level space, possibly due to the exponential increase in solution space size when individual lengths become larger.

Therefore, although the general finding is that bigger populations and longer individuals improve the performance of RHEA on average, it should be possible to devise methods that could identify the type of game being played, and employ different (or, maybe, modify dynamically) parameter settings. In a form of a meta-heuristic, an agent could be able to select which configuration better fits the game being played at the moment and increases the average performance in this domain.

The most straightforward line of future work, however, is the improvement of the vanilla RHEA in this general setting. The objectives are twofold: first, seeking bigger improvements in action sequences during the evolution phase, without the need of having too broad an exploration as in the case of RS; and second, being able to better handle long individual lengths in order for them to not hinder the evolutionary process; simulating the effect of actions in the games is the most expensive part of the process, therefore approximating such results could reduce costs and allow computation time to be spent in other parts of the algorithm. Additionally, further analysis could be conducted on stochastic games, considering the effects of more elite members in the population or re-sampling individuals, in order to alleviate the effect of noise in the evaluations or to correct any wrong directions taken by the evolution due to inaccuracies in forward model simulations.

Population seeding. Next, we presented an experiment focused on observing how a better than random population initialisation technique affects the performance of Rolling Horizon Evolutionary Algorithms (RHEA) in General Video Game Playing. Two different seeding techniques were used for testing. First, a One Step Look Ahead method (ISLA), which simply carries out an exhaustive search through all actions available and chooses the one resulting in the best next state, at each game step. Second, a Monte Carlo Tree Search (MCTS), which takes half the budget to process the game from the current state and recommend a solution to serve as a starting point for the evolutionary process. Experiments were carried out in a balanced set of 20 games of the General Video Game AI framework and using various configurations of basic RHEA parameters (population size (P) and individual length (L)).

The results suggest that both seeding variants offer a significant improvement in performance, considering both win rate and in-game score, in particular when the P and L values are small. However, as the parameter values increase, the benefit of seeding decreases, indicating that the unique solution offered by the initialisation methods, which the evolution searches around, loses value compared to the wider search space at the disposal of vanilla RHEA. A conclusion drawn from this is that the seeding-directed evolution should be combined with better exploration of the solution space in order to achieve optimal results. Nevertheless, as the aim of these algorithms is to attain a high level of play on all games, a positive result on a relatively small sample of games negates the null hypothesis and recommends deeper investigation.

An in-depth comparison between vanilla RHEA, the MCTS-seeded RHEA and Open Loop Monte Carlo Tree Search was also performed. The findings of this study pinpoint the fact that, as the evolution parameters increase, so does the performance of RHEA compared to the methods based on tree search, in several games where the search space is too large for MCTS to traverse efficiently enough. Furthermore, the MCTS seeding in RHEA does not produce worse results than simply MCTS. Therefore, this seeding technique is shown to have great promise in this environment. The ideas explored in this study were later taken forward in similar work by Galvan et al. (145).

The next steps could be focused on developing the algorithm's exploration of the game space, through further use of tree structures for hybridisation, additional rollouts and circular buffers. Moreover, a wider range of games will be used to ascertain that the difference in performance would indeed be significant in an even more general setting.

Enhancements. This section studied the effects of four different enhancements applied to the vanilla version of the Rolling Horizon Evolutionary Algorithm (RHEA), aiming to provide a fair comparison between

the methods and identify synergies. They were analysed in four different parameter configurations, with the same general heuristic and in the same set of 20 games of the General Video Game AI (GVGAI) corpus.

The experiments were divided into two parts due to the large scale of the analysis. First, three of the enhancements were tested individually and in all combinations, resulting in 8 algorithms. A bandit system was used to guide mutation (*EA-bandit*); a statistical tree was kept alongside evolution employed in selecting actions at the end of the evolution (*EA-tree*); and a population shifting method was used to carry forward information from one game step to the next (*EA-shift*). Combinations of these methods resulted in interesting hybrids.

The results indicate that the uni-variate bandit system does not work well in this setting where individuals are sequences of actions. This is thought to be due to epistasis: changing one gene in an individual impacts all the subsequent genes as well, therefore the statistics used by the bandits are much less useful. This leads to a line of future work in employing an N-tuple bandit mutation (31) in order to account for the connections between genes. The bandit systems do work better in high configurations, due to fewer evolution iterations, therefore the effect is less pronounced. *EA-shift* and *EA-tree-shift* stood out as the best algorithms in this first part, followed shortly by *EA-tree*. It was observed that the stats tree was more beneficial in small configurations, due to information in small individuals being more accurate than in longer ones. Whereas the shift buffer enhancement led to a significant increase in score gain, as well as raising the win rates in small configurations so as to be similar to those of the vanilla version with large core parameter values. The shift buffer worked so well because reusing information from previous game steps means learning more about the environment in the limited budget available.

The second part of the experiments took the 4 best algorithms found, and added Monte Carlo rollouts at the end of the individual evaluation, repeated $R = \{1, 5, 10\}$ times, to create 4 new variants (named *EA-roll* for the vanilla version). No significant difference was observed across the configurations, except for *EA-shift*, which saw an increase in performance proportional to the individual length, surpassing its rollout counterpart. Therefore, the longer the individual, the less beneficial the rollouts become.

EA-shift and *EA-tree-shift-roll* showed a promising performance, but the best algorithm emerging was *EA-shift-roll* (using a shift buffer and rollouts repeated $R = 5$ times, configuration 10-14). This method was compared to Monte Carlo Tree Search (MCTS) for validation and it outperformed MCTS significantly in several games. The algorithm considered most similar to MCTS (*EA-tree-roll*, employing the stats tree and rollouts repeated $R = 1$ times), in its best configuration (10-14), was not as good as initially estimated, and worse than most other RHEA variants in this second part of experiments.

Another line of future work will be expanding this study to a wider range of games, as 20 remains a relatively small sample and possibly not indicative of the true potential of these methods. Additionally, determining the characteristics of the specific games that lead to changes in the performance of particular methods would be an interesting study in itself, which would open the possibility of dynamically tuning and turning these features on or off in order to gain the maximum benefit from each one, depending on the problem at hand.

Next chapter. In the next chapter, we develop several ideas described here to strengthen and deepen the analysis process applied. This aims to better study the strengths and weaknesses of the algorithm in the variety of environments proposed and give better insights into its inner-workings in order for the algorithm to be more accessible and more understandable to the wider community, and to encourage novel applications of RHEA.

Chapter 5

RHEA Analysis

This chapter presents work carried into analysis of RHEA’s (and other AI players) decision-making process, looking to answer the question of why it does the things it does, and trying to deepen the understanding of the inner-workings of this algorithm. This chapter includes work into the visual analysis of the algorithm (Section 5.1), analysis of behaviour in sparse reward environments (Section 5.2) and analysis of agent features leading to prediction of overall performance (Section 5.3). All works included here use the same set of features extracted from the agent’s decision-making process to show different applications of the analysis: a visual analysis tool (displaying the features extracted directly, in a more user-friendly manner), online adaptation of the agent to improve the decision-making process depending on the state of the features extracted, and predicting the agent’s winning chances depending on the state of the features extracted, respectively.

5.1 Visual Analysis

The work in this section was published at AIIDE 2018:

——, “VERTIGO: Visualisation of Rolling Horizon Evolutionary Algorithms in GVGAI,” in *The 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018, pp. 265–267.

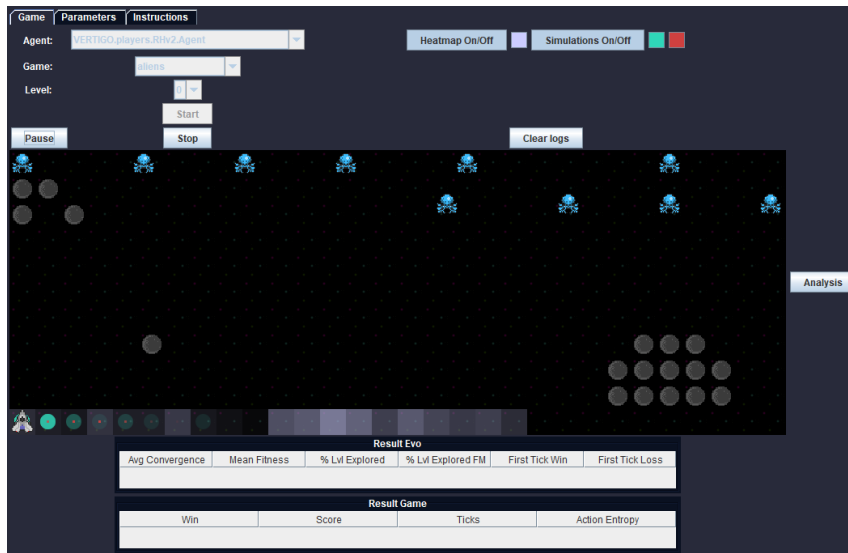
The first type of analysis discussed in this section is visual. Experiment results are often reported in numerical form, allowing for quick comparison between multiple algorithms or running these values through more advanced processing software. Additionally, they may also be accompanied by a visual representation of the same numbers: readers understand information in different ways, and often it is much easier to see bars that are higher or lower than others to understand which algorithm is better, rather than scanning numbers in a multitude of tables to gather the same information.

However, an important part of analysis is looking deeper into an algorithm’s inner-workings, to understand the reasons for its decision-making, identify weak or strong points and adapt it accordingly to obtain the desired behaviour for the given problem. This means storing event logs during the games and condensing this information into a useful and easy to read summary. There have been various algorithms that use this sort of information directly in their decision-making: Perez et al. use domain knowledge gathered during the game to guide their agent (49), or Bravi et al. use statistics on agent agreement (146) and events triggered in the game (147) to gain a clearer image and wider range of agent behaviours. More details on related work can be found in Section 2.4.

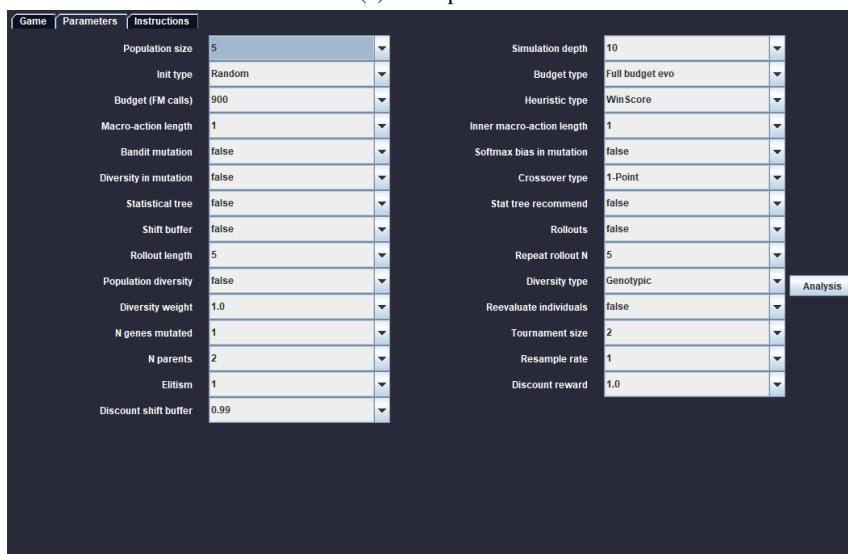
Rolling Horizon Evolutionary Algorithms are newer methods that do not benefit from the ample research behind others such as Monte Carlo Tree Search. In particular, we identify a gap in understanding this algorithm’s thinking process and how its differences lead to large dissimilarities in behaviour to MCTS. New and less understood algorithms are less likely to be rapidly adopted by the wider community, thus shining some light into the unique aspects of this method, as well as similarities and dissimilarities in its thinking process to others, could help it in being more widely adopted. As a result, to address this gap, we present a tool that allows running the algorithm in an easy-to-use application with several features of interest.

VERTIGO is an open-source software publicly available on Github¹ (18). It consists of in-depth analysis and visualisation systems developed independently of the GVGAI framework as stand-alone applications. A Java application allows for integration within GVGAI, as well as easy-to-use interaction with the system while running AI agents on the multitude of games in GVGAI. We highlight several exciting features of the latest version published at the time of writing this thesis (v1.2):

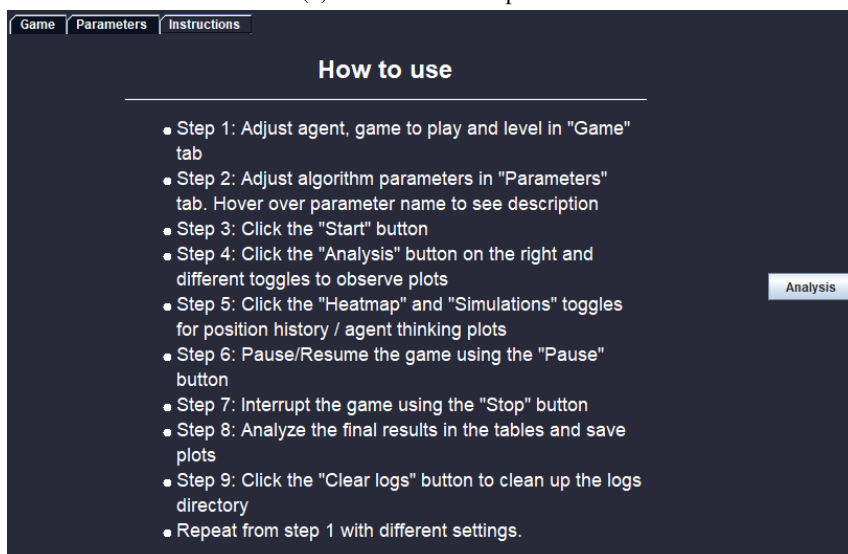
¹<https://github.com/rdgain/VERTIGO>



(a) Main panel.



(b) Parameter choice panel.



(c) Instructions panel.

Figure 5.1: VERTIGØ graphical user interface.

1. All-Java application: watch the agent play any of the GVGAI games, while observing the various plots and analysis available updated live during the game.
2. Pause the games to analyse moments of interest more in-depth.
3. Change game-playing agents and agent parameters in the GUI (*note*: currently only the RHEA agent is properly supported; however, the program was rewritten to easily support plugging in other agents, as long as they implement the required interfaces).
4. Keep track of history of game results.
5. All information and player logs saved into files, so they can be exported and further processed after the games.
6. Several analysis options:
 - **Heatmap** of all positions travelled during the game (with colour adjustment option to fit all games visuals).
 - **Simulation** visualisation, showing positions visited during the agent’s internal simulations for the game tick, with sizes and colours adjusted so as to indicate the value of the game state as well (with larger/greener circles indicating better states, and smaller/redder circles indicating less good states). Colour adjustment options are available to fit all games visuals.
 - **Convergence** line plot: showing at every game tick the iteration number where the agent decided the final action recommended and did not change again for the remainder of its thinking time.
 - **In-game score** progression line plot: showing at every game tick the agent’s in-game score.
 - **Score gain/loss events**: highlighting the game ticks where the agent gained or lost points.
 - **Win/Loss events**: highlighting the game ticks where the agent saw winning or losing game states during its internal simulations.

While the agent plays a game, features describing its experience are recorded in files and then analysed by a custom Java plotting class (making use of the *xchart* Java library². This class presents the information in visual form through a collection of graphs as described above. Figures 5.1 show the basic interface for the application, while Figures 5.2 show example analysis plots for 2 different games: score progression in “Butterflies” (top) and convergence with scoring events in “Chopper” (bottom).

The version of RHEA included in the system is similar to that used in the rest of the experiments in this thesis, combining features previously discussed in various publications (15; 16; 17) and keeping toggles, categorical and numerical hyper-parameters to control the usage of these modifications. These include population size, individual length, budget allocated (in forward model calls), heuristic used etc. There are 29 total parameters, with a search space size of 13.8×10^{12} . All parameters can be found in class `VERTIGO.players.RHv2.utils.RHEAParams.java` and VERTIGO can be run through class `VERTIGO.VERTIGO.java`.

The system was constantly evaluated and tested during development, as well as exposed to human testers at the end of the development process. The feedback received after this trial, largely related to accessibility for users not very familiar with all of the algorithm features, were incorporated in the updated 1.2 version described in this chapter.

5.2 Sparse Reward Landscapes

The work in this section was published at AAI 2019:

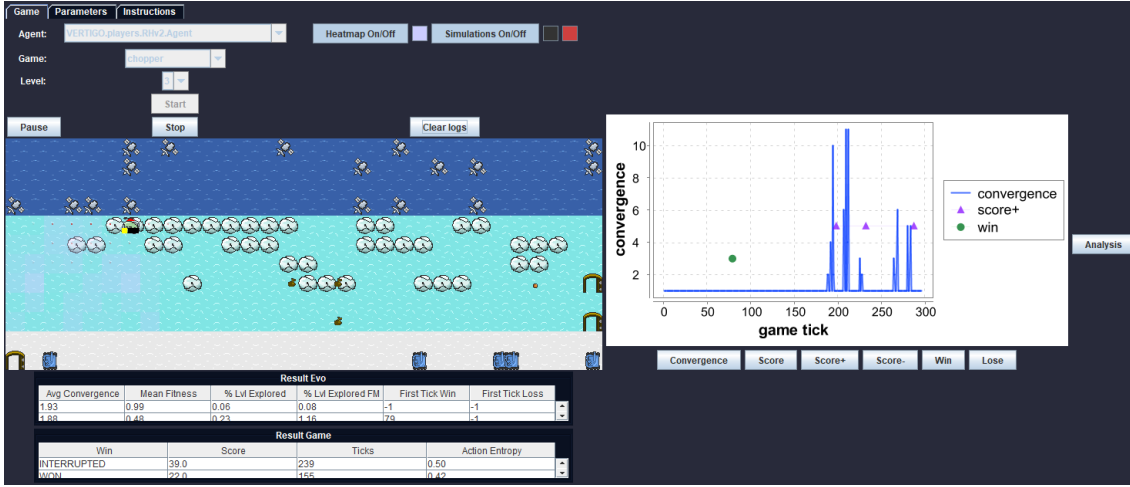
——, “Tackling Sparse Rewards in Real-Time Games with Statistical Forward Planning Methods,” in *AAAI Conference on Artificial Intelligence (AAAI-19)*, vol. 33, 2019, pp. 1691–1698.

Next, we look at observations derived from the visual analysis and resulting follow-up research. In particular, we can often observe on graphs combining convergence and game score events (e.g. Figure 5.2 bottom) that the algorithm converges very quickly in the parts of the environments where the rewards are sparse, due to not being able to find any reward signals and therefore unable to improve on its initially generated solution in the flat reward landscape. However, this is different in the parts where many scoring events occur, where the number of generations before convergence is reached is much higher, as the algorithm

²<https://knowm.org/open-source/xchart/>



(a) Score plot in the game “Butterflies”.



(b) Convergence and score events plot in the game “Chopper”.

Figure 5.2: VERTIGØ example plots, including positional heatmap (squares with reduced opacity) and simulations (circles of different sizes and colours).

requires more iterations and more statistics gathered in order to make the best decisions in environments with dense rewards.

As a result, we propose a method to automatically adjust the rollout length of the algorithm based on the flatness of the reward landscape, so as to encourage longer rollouts in situations with sparse or equal rewards (in order to potentially reach game states further into the future which can be better distinguished), and shorter rollouts otherwise (in order to allow for more generations to be iterated through leading to better statistics gathered and more accurate decisions taken).

The problem of dealing with different reward systems is one of the challenges in the domain of general video game playing, as all games are diverse - while some may present often opportunities for a player to gather points (*dense rewards*, e.g. “Aliens” in GVGAI), others might not offer any rewards at all and only give positive signals on completion of the level (*sparse rewards*, often, puzzle games). Table 2.1 shows the variety of reward systems encountered in the games studied in this thesis and their variety. As such, we expect that an algorithm able to cope with rewards varying not only between games, but between different points in time in the same game as well, would be a step towards more adaptive general AI.

To clarify the specific types of games used in this study, we can differentiate the reward systems used in the 20 games as follows:

1. **Sparse rewards:** Crossfire, Camel Race, Escape, Hungry Birds, Wait for Breakfast, Modality
2. **Dense rewards:** Dig Dug, Lemmings, Roguelike, Chopper, Chase, Bait, Survive Zombies, Missile Command, Plaque Attack, Infection, Aliens, Butterflies, Intersection, Seaquest

For example, the game “Camel Race” shows the agent racing against other characters (camels, in this case), controlled by simple behaviours (e.g. random, consistent movement towards the goal at different

Table 5.1: Deceptive game set including feature analysis.

Idx	Game	Stoch.	Rewards	Win	Lose	Levels	NPCs	Res.	Actions
0	Decepti Coins		D	Exit	Death	M/Dense	E		Move
1	Decepti Zelda	x	Disq	Exit	Death	M/Sparse	E		Move+Shoot
2	Sister Saviour		Disq	Kill	Death	S/Sparse	E		Move+Shoot
3	Invest	x	Disq	Timeout	Score	S/Dense	N		Move
4	Flower		D	Timeout	-	S/Sparse			Move
5	Wafer Thin Mints Exit		D	Timeout/Exit	Score	M/Dense			Move

speeds etc.), and attempting to traverse the level from one end to the other the fastest, while avoiding obstacles on the way. In this game, the agent receives no rewards unless it completes the level successfully, which earns it 1 point.

Differently, the agent plays a fairy trying to collect all the butterflies in the game “Butterflies”, receiving 2 points per butterfly collected; this game is played in an open meadow, with many randomly moving butterflies flying around the agent and possibly spawning even more if they touch the cocoons placed around the level. The cocoon mechanic can further better reward a player that delays their win in order to allow more butterflies (and therefore more sources for points) to spawn.

It is further interesting to note that some games feature non-linear reward systems, such as “Lemmings” or “Plaque Attack” - here, the player can lose points depending on its interactions with the environment, as the games include enemy non-player characters that attempt to hurt the player’s score. Therefore the player could try to prevent these negative interactions, although some may actually be required in order for the player to win.

In addition to the usual set of 20 GVGAI games, we use a further 6 games with reward structures and level layouts designed to specifically deceive regular AI assumptions (e.g. the more points the agent obtains, the better) as well as take advantage of their limitations (e.g. simple generic heuristics, limited thinking time / lookahead lengths) (52). A full description of the games can be seen in (52) and Appendix B, and feature analysis for this game set is presented in Table 5.1. Due to the longer or adaptive rollouts resulting from this work, we hypothesise that our proposed methods will improve performance in these types of games as well.

5.2.1 Baseline Methods

We tested the performance of two different algorithms and two variations of each, adding the dynamic rollout length enhancement on top. We use both Monte Carlo Tree Search 2.2 and Rolling Horizon Evolutionary Algorithms 3.1 to show that this is a flexible enhancement that can work with any algorithm using variable length lookaheads for decision-making purposes. In the case of RHEA, we first chose to use the best variant described in literature up to date, which uses a shift buffer and Monte Carlo rollouts at the end of the individual evaluation (17). The MCTS agent implemented in the GVGAI framework already uses Monte Carlo rollouts; however, a similar method to the shift buffer for keeping the statistical tree between game ticks instead of starting from scratch heavily impacted the base algorithm’s performance, and thus we further test a variant of RHEA without the shift buffer, for fairness of evaluation.

All algorithms tested employ the same parameter configuration: a starting rollout length L of 14 actions (and $L/2 = 7$ actions for Monte Carlo evaluations added in RHEA), a budget of 1000 forward model calls and a population size P of 10 individuals for RHEA variants.

$$f = score + \begin{cases} H^+, & \text{if win} \\ H^-, & \text{if loss} \end{cases} \quad (5.1)$$

Further, all use the same heuristic function to evaluate the final state reached after performing their rollouts, shown in Equation 15.2.1. Here, H^+ represents a very large positive integer, and H^- represents a very large negative integer; these are set to very large values in order to encompass all possible rewards the agent might receive from a game (specifically to account for discontinuous and large increments in rewards observed in the game “Seaquest”). In order to keep the UCB constant $C = \sqrt{2}$ relevant in MCTS and due to the generally unknown minimum and maximum rewards per game, MCTS keeps track of reward boundaries dynamically and normalises all rewards observed according to the previously observed boundaries. This heuristic function aims to encourage winning states and penalise losing states, while using the game score as a guiding feature through the rest of the states. However, it is worth noting that this function often returns values of 0 in games with sparse rewards, which can reduce the agent’s behaviour to random. More complex functions could be used to give more information to the agent and better guidance in its search, but the focus

Table 5.2: Extreme length rollout budget allocation. Default configuration in bold.

Idx	Length	Budget (FM Calls)
1	14	1000
2	50	3000
3	100	6000
4	150	9000
5	200	12000

of this work is to study performance differences in our proposed variants, especially in these very difficult situations.

Lastly, we note an interesting difference in the exploration of the search space by MCTS and RHEA, particularly relevant in this study. MCTS builds its tree incrementally, focusing on nodes closer to the root and slowly expanding outwards in order to build most accurate statistics on which action to take next. In sparse reward games, this algorithm is very likely to evenly expand its tree, not being able to exploit the more promising branches (as they all would look the same in the lack of rewards) and therefore losing its great advantage that sets it apart from other techniques in many other games. However, RHEA spreads its computation efforts evenly across the entire search space, as it samples complete sequences of actions at a time. In the cases where rewards to exist and may be further away on a specific path, or in puzzle games where a precise sequence of action is required to solve the problem, RHEA is much more likely to sample an overall better solution than MCTS; this generally gives RHEA the upper hand in sparse reward games, while MCTS generally excels in environments with dense rewards.

5.2.2 Experiments

We carried out two sets of experiments with these methods. First, we studied the difference in performance when strictly increasing a method’s lookahead, with a directly proportional increase in budget available, so that 40 individuals are evaluated in RHEA and 40 iterations are performed by MCTS, as is the case in the default settings described above. And second, we applied the dynamic rollout length adjustments, constrained within the fixed budget of 1000 FM calls.

Extreme Length Rollouts

For this first experiment, we consider research which uses much longer lookaheads for the algorithms with great success (148). This is often not feasible in real-time with current computation power when running most of the GVGAI games, let alone complex modern video games where simulations of game states would be much slower to run. However, technology advances suggest that it might be possible to run such simulations in the future and it is worth investigating whether simply increasing the rollout length is beneficial for these algorithms, so as not to use up other resources for additional dynamic adjustments calculations.

The longest lookahead previously explored in GVGAI research was 24 in (15). Here, we test up to 4 times this length. See Table 5.2 for details on lengths tested and associated budgets.

We hypothesise that very long rollouts allows the agent to sample action sequences which lead to further away rewards, and also create better plans to reach the rewards, as they do not face the regular trade-off between lookahead length and accuracy of statistics. Therefore performance in sparse reward games should be increased. However, these longer lookaheads might also cause the agents to ignore more immediate threats, as they would tend to focus on longer term goals instead.

Dynamic Length Rollouts

The second set of experiments investigates instead the effects of dynamically modifying the rollout lengths within the fixed 1000 FM calls budgets, which is in line with real-time specifications with regards to GVGAI games. The aim of this enhancement is to promote accurate decision-making in dense reward situations and quick reactions to the unexpected, while a more exploratory approach can be successfully applied so as to identify further away rewards when these are sparse. The details of the inner-workings of the dynamic adjustment of action plans is detailed in Section 3.2.6.

We hypothesise that dynamic rollouts will improve performance in sparse reward games, while not hurting win rates in dense reward environments. Agents should be able to better adapt and cope with a variety of different situations and find successful approaches to solve even more difficult problems.

Table 5.3: Average win rate for long rollouts variations. Distinction is made between sparse and dense rewards games, with the final column averaging over all games. Budget for each algorithm is $L \times 60$. RHEA obtains significantly better results than MCTS in all but sparse reward games ($L = 200$).

Alg	L	Sparse	Dense	Overall
RHEA	50	29.80 (3.30)	61.33 (1.66)	51.80 (2.14)
	100	36.36 (3.59)	61.04 (1.87)	53.55 (2.37)
	150	36.03 (3.59)	62.63 (2.10)	54.60 (2.53)
	200	37.04 (3.85)	60.89 (1.93)	53.70 (2.49)
MCTS	50	14.31 (3.29)	53.54 (2.08)	41.70 (2.42)
	100	22.39 (3.79)	54.18 (1.58)	44.50 (2.23)
	150	26.77 (3.94)	53.75 (1.49)	45.60 (2.21)
	200	30.98 (4.14)	53.61 (1.52)	46.70 (2.29)

5.2.3 Results and Discussions

The results reported in this section mainly focus on the win rate and game scores achieved by the algorithms. One aspect further reported is the algorithm’s sum of Formula-1 points over all games, when compared to all other variants of the same algorithm. The points are awarded per game, based on the algorithm’s rank (by win rate), the first ranked receiving 25 points, then 18, 15, 12, 10, 8, 6, 4, 2 and 0 for the rest, depending on the number of methods included in the tournament. This is the typical ranking system used in the GVGA competition to better approximate generality across a range of games.

Extreme Length Rollouts

Overall, results suggest that the general trend is the longer the rollouts, the better. However, there is a point where the improvement halts in RHEA (see last column in Table 5.3). For MCTS, even though win rate increases with rollout length, the number of F1 points gathered actually decreases (from 379 points to 338 points), suggesting that long rollouts favour some games in the detriment of others. When looking at the different reward systems, the improvement is only noticeable in sparse reward games, whereas the performance in dense reward games remains fairly constant; one exception is “Bait” which increases from 16.5% to 37.4% for RHEA with $L = 150$ (this game is a special incremental scoring system game case featuring puzzle elements; see Section 2.1.2 for details). A similar trend is observed for MCTS: significant improvement in win rate in sparse reward games (from 14.31% to 30.98%), while performance in dense reward games remains constant; thus the performance gain in sparse reward games is not detrimental to the rest of the problems. However, we do notice a striking drop in performance for MCTS in the game “Chopper”, where the algorithm falls from 100% win rate to 4% with $L = 200$; the same is not observed in RHEA, suggesting MCTS to be worse at dealing with immediate threats when considering farther ahead rewards.

Figure 5.3 shows the win rate of both RHEA and MCTS variants with long rollouts in the sparse reward games. It is interesting to observe that in the game “Escape” both methods increase their performance until they peak (at $L=100$ for RHEA and $L=150$ for MCTS), following which the win rate drops again. In most other games we see a steady increase as rollout length goes up. This could suggest that the rollout length should not be pushed to too high values and instead more carefully considered based on the problem at hand.

Dynamic Length Rollouts

The two algorithms tested in this study show very different reactions to dynamic variations of their rollout length. This adjustment halves win rate in RHEA (from 48.60% to 21.05%), but it improves performance in MCTS, from 40.40% to 44.00% overall.

The explanation for the large drop in win rate suffered by RHEA is the use of the shift buffer. In fact, it is reasonable that altering previously evolved sequences of actions by cutting or increasing them (with new random actions added at the end) changes the sequence (and importantly, the phenotype) too much for the algorithm to be able to handle. This theory was tested and it showed that, by removing the shift buffer, dynamically adjusted rollout lengths in RHEA lead to a 39.55 win rate. This is still lower than the baseline method, but it is at the level of the default MCTS method without dynamic rollouts, suggesting that this adjustment does have the potential of improving performance, with possibly tweaked (or dynamically adjusted as well) parameters.

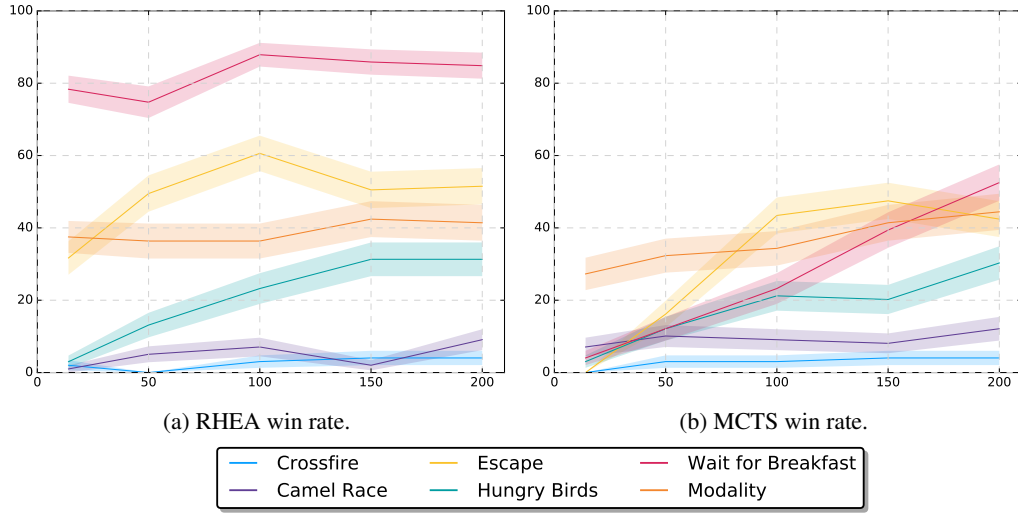


Figure 5.3: Win rate in sparse rewards games for RHEA and MCTS with extreme length rollouts. Shaded areas indicate the standard error of the measure.

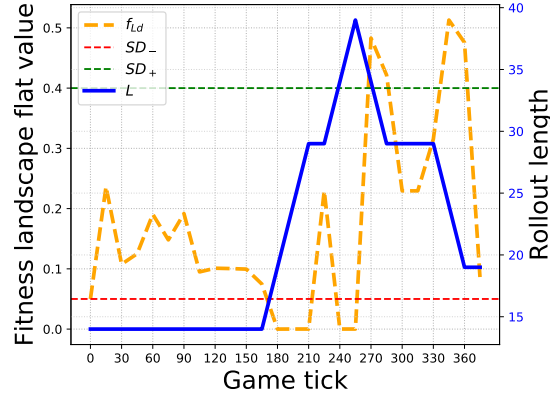


Figure 5.4: Variations in dynamic rollout length (blue) and fitness landscape flatness (orange) for RHEA agent “Butterflies”, level 0. Note that the scale for rollout length is on the secondary (right) Y axis.

Table 5.4 summarises the win rates of the two methods and their variations on the set of 20 games. Looking more in-depth at the two types of reward systems paints an interesting picture. The performance of RHEA remains similar in sparse reward games when dynamic rollouts are employed, whereas the noticeable drop in performance comes from the side of dense reward games, notably “Chopper”, from 100% to 56.57%, and “Intersection”, from 100% to 43.43%. This indicates dynamic rollouts to be harmful for RHEA in environments dependent on quick reactions.

However, MCTS sees a similar story as in the case of extreme length rollouts: the performance in sparse reward games is significantly improved (from 6.90% to 19.70%), with no detriment to the rest in the set. Some notable examples here are “Escape”, which sees an increase in win rate from 0% to 29.29%, and “Wait for Breakfast”, from 4% to 42.42%. This suggests dynamic rollouts to be greatly beneficial to MCTS in sparse rewards landscapes.

Figure 5.4 shows an example of how RHEA varies its rollout length L in the game “Butterflies” and the corresponding fitness landscape flatness f_{Ld} . The upper and lower limits (SD_+ and SD_- , respectively) are the points where the algorithm is expected to adjust its rollout length depending on its assessment of the fitness landscape. It is interesting to note that the rollout length does match the shape of the fitness landscape flatness. The fact that the algorithm reduces its rollout length after a peak at game tick 255 suggests that RHEA is able to successfully use the longer rollouts to adjust its search and find the more interesting parts of the level to win the game.

Deceptive Games

The last experiment was to test these methods on the deceptive games presented by Anderson et al. (52). It is expected that the adjusted variants would perform better than the baseline, as they are less biased and

Table 5.4: Win rates for RHEA and MCTS, vanilla and dynamic variants (non-shift RHEA). Distinction is made between sparse and dense reward systems, with the last column averaging win rates over all games.

Alg	Sparse	Dense	Overall
RHEA	25.59 (2.82)	58.04 (1.73)	48.31 (2.05)
RHEA-dyn	24.41 (3.26)	46.03 (2.84)	39.55 (2.97)
MCTS	6.90 (1.79)	54.76 (1.65)	40.40 (1.69)
MCTS-dyn	19.70 (3.37)	54.47 (1.89)	44.04 (2.33)

better adapt to various situations when making decisions. The most interesting results on the 5 games tested are as follows.

- “Decepti Coins”: RHEA-dynamic performs significantly better than all other RHEA variations, in both win rate and score (55.56% win rate, a significant 40% improvement over baseline). All MCTS variations achieve a 79.8% win rate, although the extreme rollout length variations complete the games the fastest (200 ticks faster than the baseline on average).
- “Flower”: All algorithms achieve 100% win rate, but MCTS with long rollouts is overall significantly better than the baseline in score, with over 200 points improvement for all rollout lengths. MCTS with reward signals gains 50 points less than baseline RHEA, but still 100-300 more points than all other RHEA variations.
- “Invest”: No algorithm manages to solve this game, but all fitness exploratory variations of the algorithms are significantly better than the baseline in score (100-300 point improvement for MCTS, 10-100 points for RHEA).
- “Sister Savior”: The win rate in this game is on average very low ($3.03\% \pm 1.08$), with 4 algorithms unable to solve it: baseline MCTS, MCTS-dynamic, RHEA-150 and RHEA-200. The highest win rate is achieved by MCTS-100 ($10.26\% \pm 4.86$), followed closely by RHEA-50 with $7.69\% \pm 4.27$ win rate.
- “Wafer Thin Mints Exit”: All algorithms achieve 100% win rate. RHEA-dynamic is significantly better in score than the baseline, 2.68 (± 0.36) to 1.16 (± 0.11) points.

RHEA-dynamic performed much better than the baseline method in most of the deceptive games tested. There was not much difference observed in some games in terms of win rate, all variations achieving either 100% or 0% victories, although there were overall improvements in either win rate or game score in all cases over the baseline methods. This indicates our modified methods to be more robust to deceptive reward systems.

5.3 General Win Prediction

The work in this section was published at IEEE CIG 2018:

—, “General Win Prediction from Agent Experience,” in *Proc. of the IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2018, pp. 1–8.

The question of whether the correct algorithm is used for the problem at hand usually comes at the end of execution, when the algorithm’s ability to solve the problem (or not) can be verified and/or its behaviour analysed, as has been done in the works presented up until now in this thesis. But what if this question could be answered in advance, with enough notice to make changes in the approach in order for it to be more successful? This section proposes a general agent performance prediction system, tested in real-time within the context of the General Video Game AI framework. It is solely based on agent features extracted from the analysis of their thinking process, therefore removing potential human bias produced by game-based features observed in known games.

Several previous studies suggest clustering games using game features or performance of agents on them (149; 1; 124; 150). In general, this clustering can be used to select which agent, from a collection of different techniques, should be used to play the game at stake. This is a reasonable approach, but little thought has been put so far into analysing if the algorithm should be changed once the game has already started. The technique used in a particular game may need to be discarded in favour of another one, either because the choice was wrong in the first place, or because the game conditions have changed.

In fact, it is common for a human who is playing a game to have a certain intuition about how well are they doing mid-way through it. A player in Space Invaders can see, before losing the game, that the presence of too many aliens close to the ground is a bad sign. Having most pellets still to be eaten in Pac-Man with no power pills left in the level can also be an indication of the likely (negative) outcome of the game. Our interest is to see if it is possible to give this ability to a general agent, allowing the possibility of changing technique before it is too late in the game.

The use of game features, however, poses an additional problem: including the number of pills or aliens as features is a very specific approach. In fact, even considering GVGA terms (as presence of Non-Player Characters, portals, resources, etc.) is not general enough. This does not only tailor the methods to GVGA (which may be hard to avoid when working with a specific framework), but also to the games the algorithm designer has seen in the past. Other features, however, can be more resilient to this bias, such as agent-based features (114): decisiveness of action selection, speed of convergence to a recommendation or analysis of the fitness landscape.

The work presented in this section explores the idea of designing a game outcome predictor. In particular, we propose building predictors that only focus on agent-based features, in order not to bias the prediction towards already seen games. The question this section tries to answer is if it is possible to train a model solely on agent experiences, so it is able to estimate the probability of victory at the current state for any game within the GVGA framework.

As game situations and agent behaviour can be very different in various parts of the game, we differentiate three different models that can be queried while playing the game to determine whether the agent will win or lose, based on the current game phase: early, mid and late game feature models. The models are trained on 80 games in the framework and tested on 20 new games, for 14 variations of 3 different methods: RHEA, MCTS and Random Search (RS). In this context, RS samples uniformly at random action sequences of length L within the allocated budget and chooses for play the first action in the best solution found. The same RHEA individual evaluation method is applied to evaluate RS sequences of actions, and the algorithm chooses the first action of the best sequence sampled to play in the game. All agents use the same heuristic function (see Equation 5.2.1).

Additionally, we introduce the term *analysis window* W for MCTS, which is represented by the number of iterations included in the analysis - thus grouping the iterations in sets, similar to the concept of population used in RHEA. In RS, this same number is used to determine the number of sequences sampled in a game tick (no evolution occurring afterwards).

5.3.1 Classification

Due to the high variety of the games in the GVGA framework and the low overall performance of the general agents (most games remain too difficult to be solved), as highlighted in the literature review, the $F1$ -Score (see Equation 5.4) will be reported as to the quality of the classifiers employed in this study. It represents the harmonic average between precision and recall, 1 signifying the best value and 0 the worst.

$$precision = \frac{TP}{TP + FP} \quad (5.2)$$

$$recall = \frac{TP}{TP + FN} \quad (5.3)$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (5.4)$$

In Equations 5.2 and 5.3, TP stands for true positives (correctly predicted a win), FP stands for false positives (incorrectly predicted a win) and FN stands for false negatives (incorrectly predicted a loss). This is meant to be a better measure of classifier quality than accuracy when there is an imbalance in data (in this case, a majority of games, approximately 77%, resulting in a loss, see Table 5.5) (151).

5.3.2 Data set

To obtain the set of agents used to generate the data set, 3 rollout values L were tested for RS (10, 30, 90); 2 parameter sets were tested for all 4 RHEA variations ($P=2, L=8$ and $P=10, L=14$); 3 parameter sets were tested for MCTS ($W=2, L=8$; $W=10, L=10$ and $W=10, L=14$). Every experimental setup makes use of the same number of FM calls.

All 14 algorithm variations described previously were run on all 100 games publicly available in the GVGA Framework, 20 times on each of the 5 levels, being given a budget of 900 FM calls. Each run produced 2 log files, recording information about the agent inner processing, as well as its actions played

Table 5.5: GVGAI-style Formula-1 point ranking of all methods. Type and configuration (rollout length L if one value, population size P and rollout length L if two values) are reported, followed by the sum of Formula-1 points across 20 games and the average win rate.

#	Algorithm	Points	Avg. Wins
1	10-14-EA-Shift	1225	26.02 (2.11)
2	10-RS	898	24.33 (2.13)
3	2-8-EA-All	888	23.95 (1.98)
4	30-RS	885	22.49 (2.02)
5	2-8-EA-Shift	866	24.54 (2.00)
6	14-MCTS	780	24.29 (1.74)
7	10-14-EA-All	695	22.66 (2.02)
8	10-14-RHEA	664	23.23 (2.08)
9	10-MCTS	652	24.01 (1.65)
10	2-8-EA-MCTS	621	23.98 (1.73)
11	10-14-EA-MCTS	618	23.99 (1.80)
12	8-MCTS	594	23.42 (1.61)
13	90-RS	457	16.31 (1.67)
14	2-8-RHEA	257	18.33 (1.77)

and game scores, at every game step, in addition to the final game results (win/loss, final score and number of game ticks).

Data set and processing scripts are publicly available³. On each game, Formula-1⁴ points are awarded attending to a ranking determined by win rate. The first 10 ranked entries receive 25 points, second 18, then 15, 12, 10, 8, 6, 4, 2, 1 and 0 for the 11th and below positions. Points across games are summed up for an overall ranking, shown in Table 5.5.

The rest of this section presents the metrics recorded for each agent, as well as the features extracted from the logged data and any pre-processing steps taken.

A list of the 12 features extracted for each (game, player, level, repetition) tuple can be found below. Features $\phi_2, \phi_8, \phi_9, \phi_{10}, \phi_{11}$ and ϕ_{12} compute averages from the beginning of the game up until the current tick t . Features $\phi_5, \phi_6, \phi_{11}$ and ϕ_{12} rely on the FM. Only *agent features* were used in this study, with the exception of the game score:

- ϕ_1 **Current game score**
- ϕ_2 **Convergence**: Iteration number when the algorithm found the final solution recommended during one tick. A low value indicates quick and almost random decisions.
- ϕ_3 **Positive rewards**: Count of positive scoring events.
- ϕ_4 **Negative rewards**: Count of negative scoring events.
- ϕ_5 **Success**: The slope of a line over all the win counts. Win count increases when any solution sees the end of the game with a win, at any point during search. A high value indicates the increase in discovery of winning states.
- ϕ_6 **Danger**: The slope of a line over all the loss counts. Loss count increases when any solution sees the end of the game with a loss, at any point during search. A high value indicates the increase in discovery of losing states.
- ϕ_7 **Improvement**: The slope of a line resultant from all best fitness values plotted over game tick. A high value indicates good fitness improvement.
- ϕ_8 **Decisiveness**: Shannon entropy (SE) (see Equation 5.5) over the percentage of times p_i each of the possible actions i was recommended (it was the first action of a solution in the final population or analysis window). In all cases of distribution-based features, a high value suggests actions of similar value; the opposite shows some to be dominating.

$$H(X) = - \sum_{i=0}^{N-1} p_i \log_2 p_i \quad (5.5)$$

³<https://github.com/rdgain/ExperimentData/tree/GeneralWinPred-CIG-18>

⁴Not to be mistaken with F1 accuracy measure for classifiers.

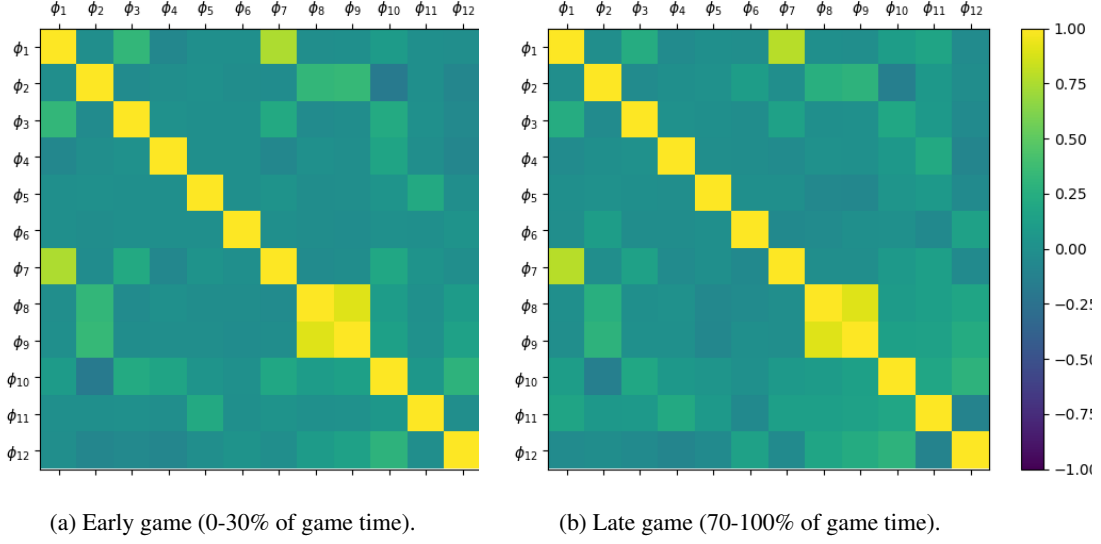


Figure 5.5: Feature correlation.

- ϕ_9 **Options exploration**: SE over the percentage of times p_i each of the possible actions i was explored (it was the first action of a solution at any time during search). A low value shows an imbalance in actions explored.
- ϕ_{10} **Fitness distribution**: SE over fitness per action.
- ϕ_{11} **Success distribution**: SE over win count per action.
- ϕ_{12} **Danger distribution**: SE over loss count per action.

The full feature file (processing all games and algorithms for global classifiers) took approximately 2.5 hours to generate, from 26GB of raw metrics data split over 281.4k files (Dell Windows 10 PC, 3.4 GHz, Intel Core i7, 16GB RAM, 4 cores).

Figure 5.5 shows the pair-wise correlation between the features extracted (using the Pearson correlation coefficient), in a comparison between the early (first 30% of game ticks) and late (last 30% of game ticks) phases of the games. Differences are small, but they do exist. An aspect worth highlighting is the higher correlations in the bottom right corner in the late-game phase versus the early-game phase (i.e. the success distribution appears to increase correlation with all other features).

Another interesting positive correlation that only appears in the late-game phase is that between the sense of danger and the convergence, suggesting agents take longer to settle on their final decision when surrounded by possible losses. A positive correlation that is less strong in the late-game is that between fitness improvement and fitness distribution over the actions, implying that when one action is deemed significantly better than the rest, it is unlikely for the fitness to improve further, possibly due to the other actions not being explored enough. The case of one action appearing to be dominating leads to a persistent negative correlation between convergence and fitness distribution. This suggests that agents are unlikely to change their decision if one action is deemed significantly better than the rest and try a less promising move.

5.3.3 Predictive models

For the purpose of these experiments, we split the games (and all data corresponding to each game) 80/20 for training/test data (thus testing occurs on data points in completely new games). This study aims to build several classifier models from agent features extracted, which would predict a win or a loss during play of a new game. We also show that the system is robust enough to handle new agents with significantly different play styles as well.

It takes approximately 10 seconds to process a full feature file and split the data into train and test, another 10 seconds to train a global model on a full feature file (or 1 minute if cross-validation is used). Predicting the outcome of 28000 instances takes approximately 1 minute, the equivalent of 2.26ms per instance. As the data used in this study is publicly available, adapting the methods to different problems or agents would only involve extracting the relevant features from the newly introduced agents or problems.

Table 5.6: Global rule-based classifier report. Global model tested on all game ticks of all instances in the test set.

	Precision	Recall	F1-Score	Support
Loss	0.83	0.52	0.64	20500
Win	0.35	0.70	0.46	7500
Avg / Total	0.70	0.57	0.59	28000

Table 5.7: Global AdaBoost classifier report. Global model tested on all game ticks of all instances in the test set.

	Precision	Recall	F1-Score	Support
Loss	1.00	0.99	0.99	20500
Win	0.97	0.99	0.98	7500
Avg / Total	0.99	0.99	0.99	28000

Table 5.8: Feature importances extracted from global model. ϕ_x represents a feature and its associated importance.

ϕ_1	0.24	ϕ_2	0.04	ϕ_3	0.08	ϕ_4	0.06
ϕ_5	0.2	ϕ_6	0.1	ϕ_7	0.12	ϕ_8	0
ϕ_9	0.06	ϕ_{10}	0.02	ϕ_{11}	0.02	ϕ_{12}	0.06

Baseline

The baseline model all our classifiers are compared against is a simple rule-based predictor incorporating human knowledge. In classic arcade games and most GVGAI games, gaining score is a good thing and often means the player is on the right path to winning if they increase their score. This idea is implemented as described in Equation 5.6, which compares the count of positive scoring events recorded to the count of negative events. This classifier’s performance on the test set is shown in Table 5.6, where it can be observed that it reaches an F1-Score of only 0.59 despite a high precision (0.70). This model will be referred to as R_g in the rest of this paper.

$$\hat{y} = \begin{cases} win & \text{if } \phi_3 > \phi_4 \\ lose & \text{otherwise} \end{cases} \quad (5.6)$$

Classifier selection - global model

Seven classifiers (with default hyper-parameters if not specified) were trained and tested for proof of concept and classifier analysis. These are K-Nearest Neighbours (5 neighbours), Decision Tree (5 max depth), Random Forest (5 max depth, 10 estimators), Multi-layer perceptron (learning rate 1), AdaBoost-SAMME (152), Naive Bayes and Dummy (simple rule decision-making, very poor general performance to be used as another possible baseline). All classifiers use the implementation in the Scikit-Learn Python 2.7.14 library (153).

Cross-validation with 10 folds was used during training to assess performance (folds created across the entire training dataset, spanning across games where necessary), the classifiers obtaining 0.95, 1.00, 0.98, 0.96, 1.00, 0.95 and 0.66 accuracy during validation, respectively. Both AdaBoost and the Decision Tree classifier achieved high accuracy values during validation and test (0.99, see Table 5.7 for its performance measures) and were deemed equal. Either could be used, but AdaBoost was selected as the main classifier for the rest of the experiments presented.

Feature importances according to AdaBoost can be seen in Table 5.8. It appears that the game score is most important in distinguishing wins and losses, unsurprisingly, but it is followed close behind by the number of wins seen by the agents, the improvement in fitness and the sense of danger. The decisiveness of the agents is considered to have no impact in deciding the outcome of a game.

Model training

All games were split into logical phases for predictions at various points in the games: early-game (0–30%), mid-game (30–70%) and late-game (70–100%). Multiple models were then trained for each of the phases,

Table 5.9: F1-Scores each model per game phase over all games, accuracy in brackets. Each row is a model, each column is a game phase. Highlighted in bold is the best model on each game phase, as well as overall best phase and model.

	Early-P	Mid-P	Late-P	Total-M
E_g	0.22 (0.72)	0.42 (0.74)	0.49 (0.76)	0.38 (0.74)
M_g	0.29 (0.72)	0.57 (0.79)	0.71 (0.83)	0.53 (0.78)
L_g	0.01 (0.73)	0.05 (0.74)	0.22 (0.76)	0.09 (0.74)
R_g	0.42 (0.67)	0.47 (0.61)	0.46 (0.58)	0.45 (0.62)
Total-P	0.24 (0.71)	0.38 (0.72)	0.47 (0.73)	

using agent features based on metrics logged only in the ticks corresponding to each interval. 3 different models resulted, referred to as E_g , M_g and L_g , respectively, in the rest of this paper.

The performance of the models was analysed by testing each on the 20 new games, on their corresponding interval of game ticks. During training with 10-fold cross-validation, they achieve 0.80, 0.82 and 0.99 accuracy, respectively. During test on the new games, they report accuracies of 0.73, 0.80 and 0.99 (0.70, 0.80 and 0.99 F1-Scores), respectively. These results are satisfactory and allow for further exploration.

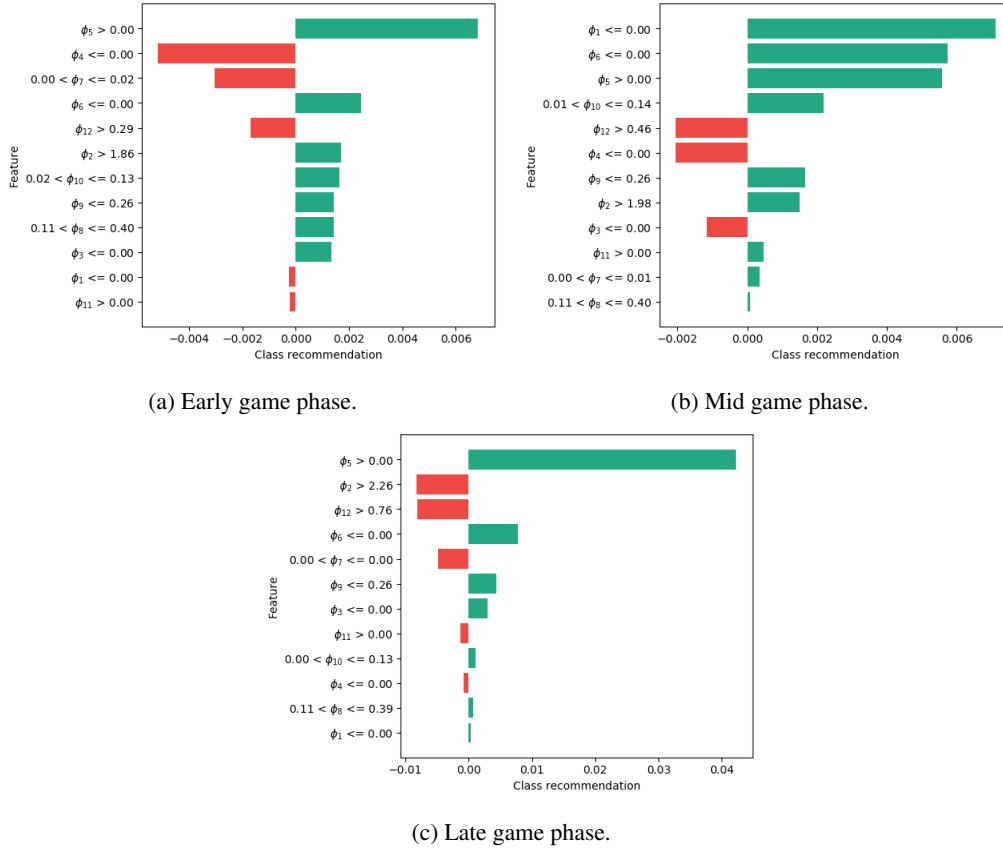


Figure 5.6: Class predictions by features for the different game phases. Red (left side) signifies the model feature predicts a loss, green (right side) a win. The probability of a class being selected based on individual feature recommendation is plotted on the X axis.

5.3.4 Live play results

For the experiments in this paper, we simulated live play by extracting agent features from the log files for a range of ticks ($T = \{100 \cdot a : \forall a \in [1, 20] : a \in \mathbb{N}\}$), all from the beginning of the game until the current tick tested $t \in T$. Gameplay from all 14 algorithms on the 20 test games (20 plays on each of the 5 levels) was used to compute the final results. Each model was tested on each of the feature files, being asked to predict the game outcome every 100 ticks.

Simulated live play results can be observed in Figures 5.8-5.10. The simple rule-based model achieves a high performance in some of the games and it proves better than the trained predictive models (i.e. “Aliens”,

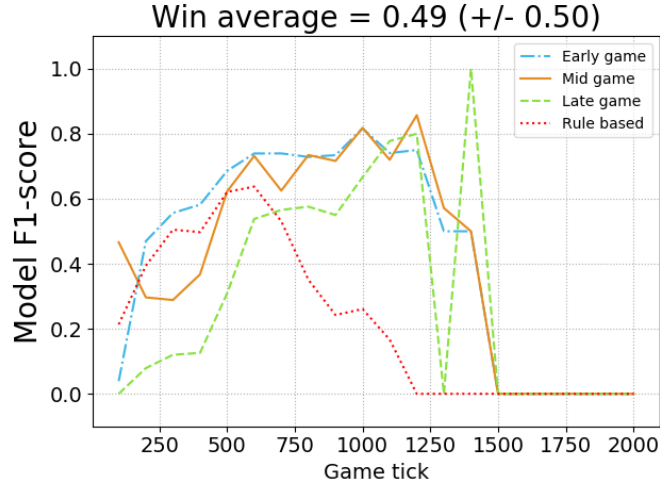


Figure 5.7: Model F1-scores in the game “Ghost Buster”, trained on variations of RHEA and RS (80 training games) and tested on MCTS.

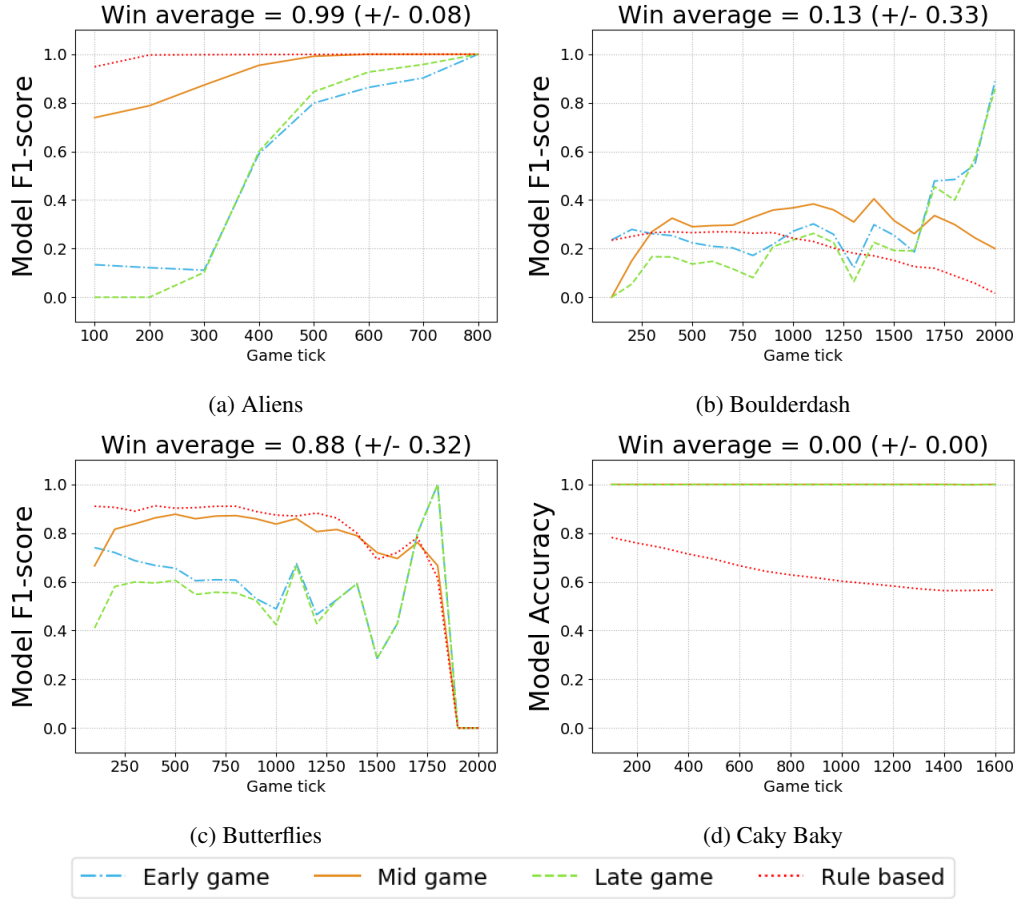


Figure 5.8: Live play results (part 1), averaged over up to 1400 runs, 14 agents, 100 runs per game. Game ticks on the X axis, maximum 2000. 3 predictor models trained on early, mid and late game data features, as well as the baseline rule-based predictor. If F1-scores were 0 for all models, accuracy is plotted instead. Win average reported for each game.

“Defem”, “Chopper”, “Eggomania”). As these are games with plenty of scoring events, it is unsurprising that the simple logic of R_g works in these cases. However, there are games where the trained models achieve much better predictions (“Ghost Buster”, “Colour Escape” or “Frogs”). The reward gain is not linear in these games, meaning the player need not necessarily be phased by a temporary decrease in score, or too optimistic as a result of score gains.

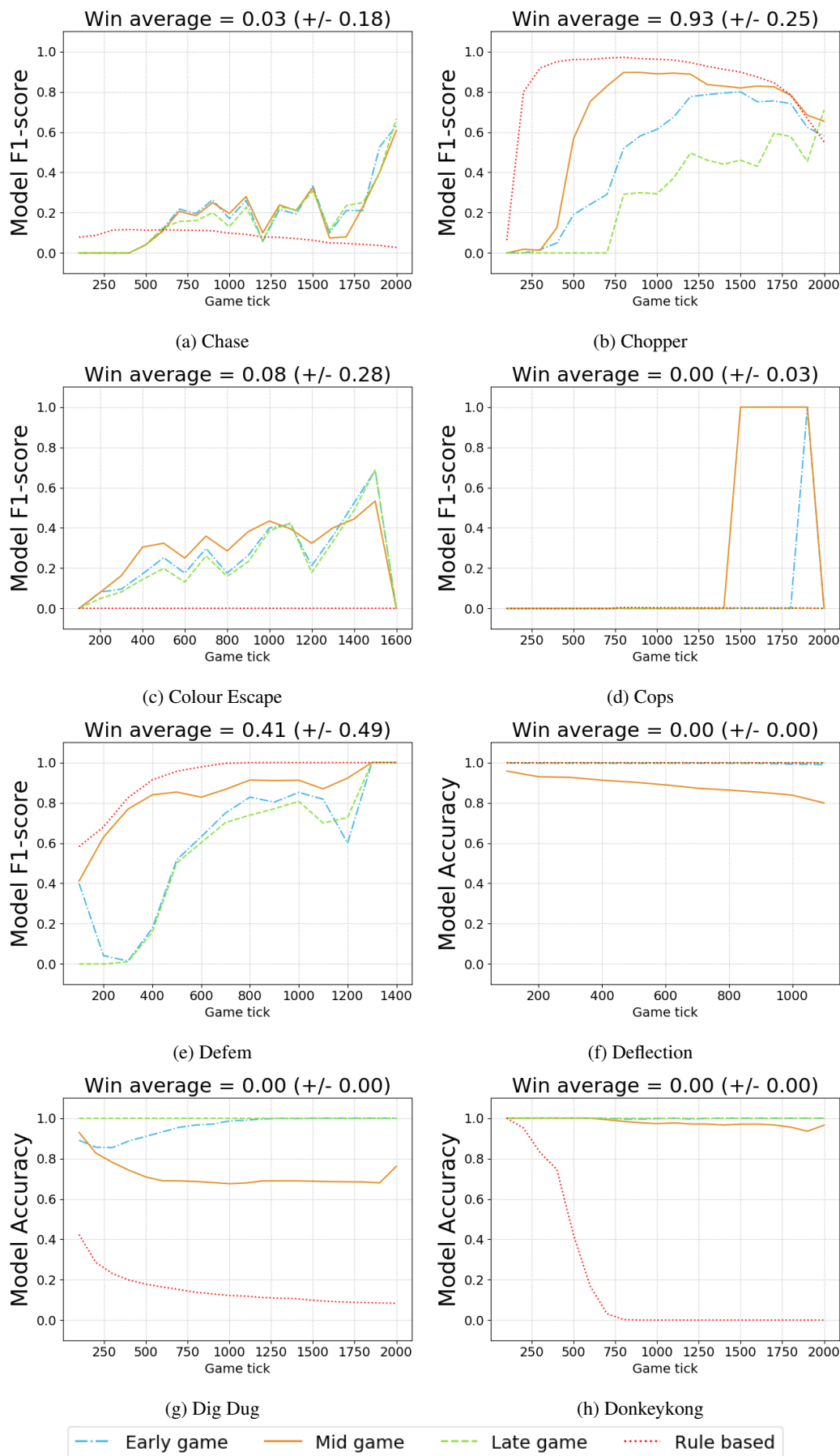


Figure 5.9: Live play results (part 2).

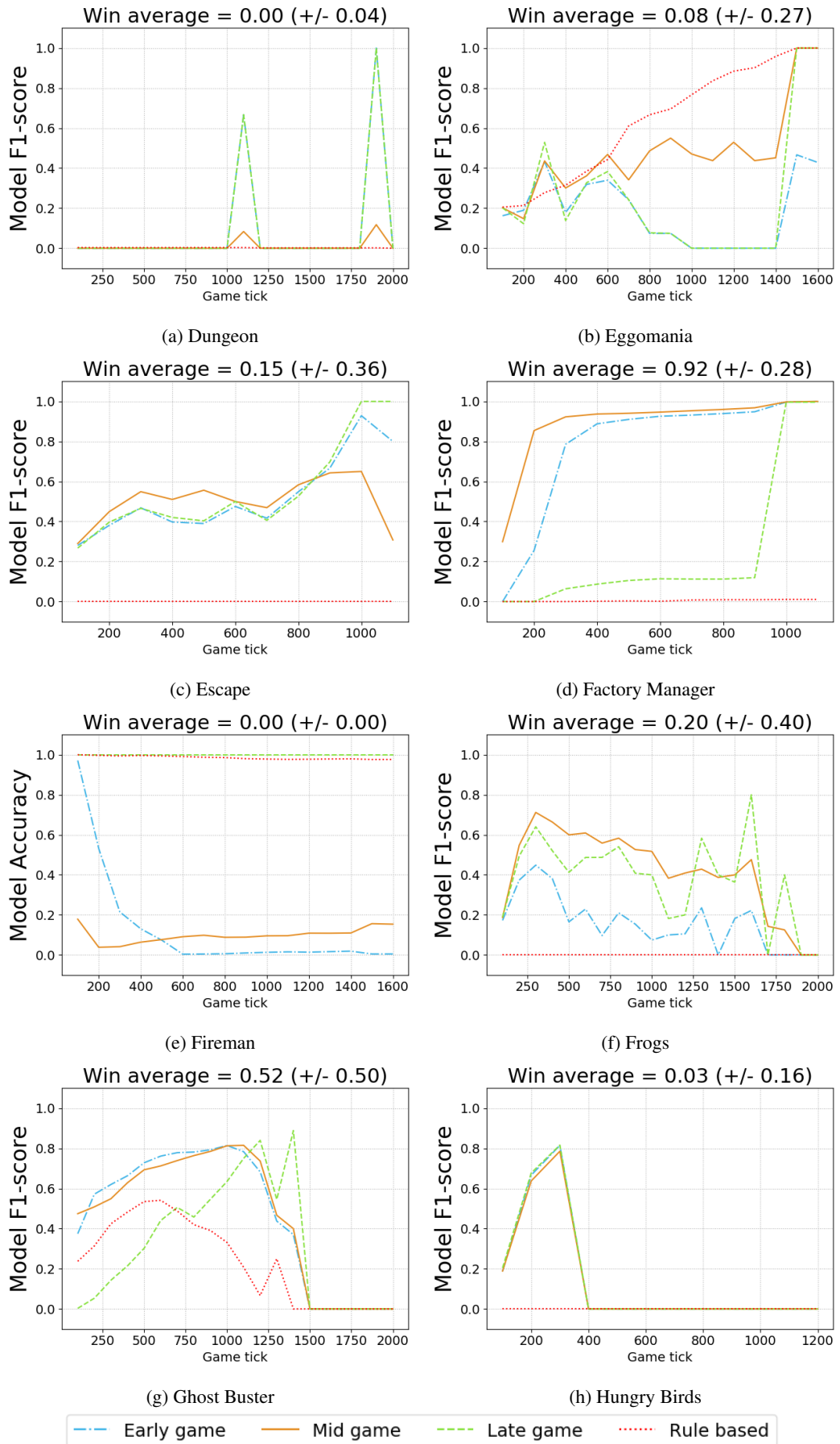


Figure 5.10: Live play results (part 3).

It is interesting to observe that the trained models do not follow the expected curves (E_g being better in the early-game phase and then decreasing, M_g showing a spike in the middle of the game and L_g offering good predictions only towards the end of the game). Instead, the early-game model appears to have a generally low performance compared to the rest, which can be explained by the limited information available for this particular model. The late-game model seems particularly strong in games with very low win rate (“Fireman”, for example, in which both E_g and M_g are predicting wins, yet the overall win rate remains at 0% for this game).

It is most interesting to observe the games with close to 50% win rate, “Defem” and “Ghost Buster”. High F1-Score values here indicate that the predictors are able to correctly judge both wins and losses equally. And indeed, in both games, the trained models achieve F1-Scores of over 0.8 only half way through the game. Model M_g appears to excel in these situations, meaning that it can recommend the game outcome and possibly the better approach to be used.

It is important to highlight at this point the importance of this great result: the predictor is able to foresee with high reliability, after only a fourth of the game has been played, if the agent is going to win or lose the game. In this case, games that are either won or lost with the same probability as a coin flip. And these are truly general models: trained in *different* games, using only *agent experience* features. This shows a great scope for the system’s use within hyper-heuristic methods, as some of the algorithms tested in this study do win at “Defem” and “Ghost Buster”. Devising a procedure that determines which is that better method and switches to it when the prediction is a loss is scope for future work, but having a system that indicates if a change should be made is the first step in that direction.

All predictive models were further analysed as to their average quality considering all games. To this extent, table 5.9 summarises F1-scores for all models on the different game phases identified. The models are the same as discussed in Section 5.3.3, and they are tested in the same previous test setting, with features extracted from the beginning of the game until the current tick which falls at the half point in each game phase (15%, 50% or 85% of the game ticks).

The results indicate the rule-based model to be giving consistent average performance throughout the game phases, being the best in the early phase with an F1-score of 0.42. In the mid and late game phases, model M_g is the best, achieving a 0.57 and 0.71 F1-score, respectively. Overall, the best model is M_g with an F1-score average of 0.53.

It is not surprising that the M_g model is the best in its respective game phase, and it is expected that the prediction quality is generally lower in the early-game phase, when there is less information available and it is harder to judge if the agent’s performance is good enough or not. A significant result extracted from the summarised data is that model M_g achieves high (if not the best) F1-scores across all game phases, indicating that the system can identify with high confidence whether the agent is performing well or not and leaving open the possibility of switching approaches appropriately.

5.4 Conclusions

This chapter offers a deeper look into the types of analysis into the inner-workings of RHEA and other statistical forward planning methods, which give a better understanding of not only the algorithm’s strength, but its overall thinking process as well. We strongly encourage in-depth analysis as more informative for future expansions and adaptations of the algorithms to new modifications or new environments, so as to appropriately exploit the strengths of the algorithm and mitigate its weaknesses.

VERTIGØ. First, we described VERTIGØ, a tool which allows live analysis of an algorithm’s thinking process and its behaviour in a game, while the algorithm is playing, through data overlaid onto the game window, or side-by-side plotting of interesting algorithm features. The data analysis provided by VERTIGØ has already shown an interesting aspect: the lack of game events, otherwise a broad and abstract concept, correlates to quick convergence, an easily measured metric. As algorithms in GVGAI struggle in games with sparse rewards, this finding could be used to identify when exactly the algorithm is not receiving enough information to make intelligent decisions and act in consequence, by actively seeking to explore more of the game space, for example. The follow-up paper on this observation, (19) was later published at AAAI (and detailed in Section 5.2), showing results of using such information in the agent’s decision-making to correct its weakness for better performance.

VERTIGØ could be further expanded in multiple ways. One direction would be to include more game-play data (for example, the long term effect of player actions on the game), and to make more direct comparisons between seemingly correlating aspects.

The games could be analysed in more detail, irrespective of algorithm, to spot when interesting events happen (e.g. a burst of enemies spawning), then comparing this with algorithm data to analyse the algorithm’s reactions to these events and improve heuristic functions. All plots and analysis extracted can be

used directly to not only characterise agent behaviour, but games and their different properties and levels of challenge as well.

Additionally, more algorithms could be added to the system (Monte Carlo Tree Search, for example, by exposing their hyper-parameters and implementing the feature analysis). This would allow users to quickly compare between not only different settings of the same method, but different methods altogether, and answer the question of why one is better than the other in a particular game.

Dynamic length. Next, we looked at analysing various ways to explore the fitness landscapes in 20 games from the General Video Game AI Framework (GVGAI), for two different algorithms, Monte Carlo Tree Search (MCTS) and Rolling Horizon Evolutionary Algorithm (RHEA). Two experiments are carried out to this extent: increasing the rollout length (to 50, 100, 150 and 200 from the baseline 14) and dynamically adjusting the rollout length based on the flatness of the fitness landscapes, in order to allow for quick reactions in busy environments or more exploration in sparse rewards scenarios. All methods were also tested on a set of human-crafted deceptive reward games to analyse whether their fitness exploration variants lead to better results in such games.

Overall, modified methods are shown to perform better than the baseline methods in sparse reward games, without affecting success rates in dense reward games. One exception is RHEA with dynamic rollouts, which halves win rate from 48.60% to 21.05%. Further analysis into this aspect suggested that this was due to the shift buffer enhancement added to the RHEA variant, which is unable to cope with the change in phenotype between game ticks where the sequence length is varied. By removing the shift buffer, the performance of RHEA with dynamic rollouts becomes comparable to baseline MCTS, at 39.55% win rate.

The algorithms reacted well to the increase in rollout length, their performance increasing with the length in sparse reward games, while performance in dense reward games was kept fairly constant; there were two exceptions to this rule in the games “Butterflies” for both methods and “Chopper” for MCTS only, where increased rollout length is actually detrimental. In “Chopper”, this is thought to be due to the immediate rewards needed to be collected in this game which may be ignored when the rollout length becomes too large. When the rollout length was dynamically adjusted, RHEA and MCTS reacted differently, RHEA seeing a general decrease in performance in games based on dense reward systems, whereas MCTS saw an increase in performance in sparse reward games. This shows that RHEA is more sensitive to games requiring quick decision-making, whereas MCTS benefits from the adjustments which aid in its traditionally poor exploration in binary-reward games.

It is worthwhile mentioning that, although these experiments employ the GVGAI framework, the applicability of the findings extend beyond these games. In particular, this work focuses on modifications to overcome the presence of sparse rewards, an issue present not only in other games such as some in the Atari Learning Environment (43) and more complex games, but also in other real life scenarios, such as engineering or robotics.

Regarding future work, although this is a very interesting step towards better understanding of agent behaviour, more analysis can be carried out for the various scenarios proposed in this study, including different metrics (game score, GVGAI generality score) or optimising dynamic rollout adjustment parameters. Additionally, the reactions of the methods to macro actions in this environment and dynamic length macro actions could be studied as well. Last but not least, more interesting problems with various features to their fitness landscapes will be introduced to the methods in order to correctly assess exactly why some algorithms react better to some situations than others.

Win prediction. Lastly, we presented work in extracting agent features from AI gameplay in a generic setting, using the General Video Game AI framework (GVGAI). Game-specific features are specifically excluded in order to avoid potential bias introduced by human knowledge of already known games. 14 total variations of Rolling Horizon Evolutionary Algorithm, Monte Carlo Tree Search and Random Search are used to generate data on 100 games, playing 20 times each of the 5 levels. Three different models corresponding to early, middle (mid) and late game phases are trained on 80 randomly selected games and tested on the remaining 20 through live play simulation and repeated predictions every 100 game ticks.

The results obtained indicate that models are able to correctly predict in most cases the outcome of the game with sufficient time before the end of the game to make appropriate changes in the method employed. Throughout all experiments, it is apparent that some models have better predictions in specific games than others. Additionally, the mid-game phase model proved to have the best overall performance, achieving an F1-Score of 0.53 (0.78 accuracy) across all test games and game phases. It is also the strongest model in the individual mid and late phases, being bested in the early-game phase only by the simple rule predictor implemented (which incorporates the human knowledge that gaining score leads to a win).

Regarding next steps, a hyper-heuristic agent will be built, able to switch between algorithms appropriately while playing the game, based on the predictions given by our system. The task can be split into two: identifying which features need improvement and which method leads to the desired behaviour. A prediction explanatory system could be responsible for the first part of this task and first steps towards this system are presented in Figure 5.6, which uses the LIME system⁵. The example provided is an explanation of the prediction of each model at game tick 300 in “Frogs” level 0, when played by 2-8-RHEA. Although the probabilities given by each of the features are small, which can be explained by the difficulty in predicting game outcome based on a single feature, there is an obvious difference between features and a clear signalling of which features currently indicate a loss. Therefore, a hyper-heuristic method could make use of this analysis to correct the loss indications.

Additionally, new methods could be introduced to the system in order to create stronger models, able to adapt to any style of play. The current system is robust enough to handle testing on new algorithms: Figure 5.7 shows predictions trained with data generated only by RHEA and RS variants, but tested live with an MCTS controller playing the game. If this is compared to Figure 5.10g, it can be seen that all models are able to maintain a similar shape and still accurately predict the outcome half way through the game.

Lastly, more features could be integrated to better describe player experience, such as empowerment (154), spatial entropy or characterisation of agent surroundings (114).

Next chapter. The next chapter brings together the previous manual comparison of different algorithm parameter settings, as well as in-depth analysis as described here, to automatically adjust RHEA for high performance on a variety of environments, both offline to give it more time to find good configurations for a game, and online, to assess its power of adaptability in real-time. Interesting insights into the algorithm’s parameter space are extracted, to inform further developments and future specific applications for RHEA.

⁵<https://github.com/marcotcr/lime>

Chapter 6

Automatic Parameter Optimisation

In this chapter we revisit the application of game-playing Evolutionary Algorithms with a deeper analysis of algorithm modifications, first offline in Section 6.1, then online in Section 6.2. We argue that automatic exploration (or algorithmic optimisation) of algorithm variations is essential for problems with large search spaces, although not exhaustive due to computation speed limitations. This optimisation process can further lead to insights into algorithms, and as such we additionally conduct an in-depth analysis of the parameter space, while highlighting performance gain in various games. There have been several recent advances in game-playing Evolutionary Algorithms (81; 155) and a multitude of modifications proposed to improve performance across a large number of games. The result is that the possibility space for algorithm configurations has grown beyond efficient manual optimisation. Although performing grid-search is sometimes possible for finding good values for some parameters (15), more recent works find the need to reduce more and more the number of parameter combinations chosen for analysis (19). Therefore, the interesting insights into which variation of the algorithm is actually best are limited to human exploration of very small sections of the entirety of the search space.

The specific novel application of Evolutionary Algorithms as game-playing methods (referred to as Rolling Horizon Evolutionary Algorithms, or RHEA) was introduced for the first time in 2013 by Perez et al. (7). The base algorithm has been extended in several works. Gaina et al. (15) performed an in-depth analysis of the algorithm's main parameters (population size and individual length), generally finding that the higher the parameter values (even reaching the extreme of Random Search), the better RHEA performs across several games; this work further highlights an increase in performance with the increase of the available budget and correspondingly higher parameter values. Different population initialisation methods were explored in (16); this work was important in highlighting the benefit of using different options in different game types, as some games saw increased performance with greedy initialisation, while others preferred a statistical approach instead. Furthermore, Gaina et al. tested in (17) various combinations with other techniques, which further pinpointed not only the difference in performance of certain parameter configurations across the different games, but also that the RHEA parameter space was already being expanded beyond the possibility of exhaustively exploring all parameter combinations. Some of these enhancements were further tested by Santos et al. (156) in General Video Game AI (GVGAI) and by Tong et al. (157) in MuJoCo's physical control tasks, both with great success. Finally, a study on dynamically adjusting individual length based on the fitness landscape observed during evolution (19) shows that some parameters might be conflicting with each other and cause poor performance in some games and suggests a need for carefully constructed parameter search spaces.

The work is carried out within the domain of general video game playing, which focuses on finding general-purpose Artificial Intelligence players that are able to play any game, even those unseen previously. The concepts behind this could be further extended to general AI which is able to solve any given task (as opposed to any given game), as methods developed for games have been shown to be applicable to wider domains, such as chemistry (158). Two large categories of players can be differentiated in this domain: planning and learning. The latter requires training for several episodes on a game before it can figure out how to play it, which is often an expensive process leading to narrow results: the agent trained for one game would be unlikely to be able to play another without significant training on the new game. The former category, which RHEA belongs to, refers to methods which work online, during the game, to search for the appropriate solutions. These methods require a forward model to be able to simulate possible futures and effects of their actions. Although planning methods are more generally applicable, they face the drawback of the lack of a FM in some games, as this is not always feasible. The problem of learning any game's model is an active research area (159) which would make our methods even more widely applicable, even in complex commercial games; however, in this work we apply our algorithms to games which do have a model available.

6.1 Offline

The work in this section was submitted to IEEE TOG (July 2020):

R. D. Gaina, S. Devlin, S. M. Lucas, and D. Perez-Liebana, “Rolling Horizon Evolutionary Algorithms for General Video Game Playing,” *IEEE Transactions on Games*, 2021.

Game-playing Evolutionary Algorithms, specifically Rolling Horizon Evolutionary Algorithms, have recently managed to beat the state-of-the-art in win rate across many video games. However, the best results in a game are highly dependent on the specific configuration of modifications introduced over several papers, each adding additional parameters to the core algorithm. Further, the best previously published parameters have been found from only a few human-picked combinations, as the possibility space has grown beyond exhaustive search. In this section, we use a parameter optimiser, the N-Tuple Bandit Evolutionary Algorithm, to find the best combination of parameters in 20 games from the General Video Game AI Framework. Further, we analyse the algorithm’s parameters and some interesting combinations revealed through the optimisation process. Lastly, we find new state of the art solutions on several games by automatically exploring the large parameter space of RHEA.

In this context, Monte Carlo Tree Search (MCTS) had for a long time represented the state-of-the-art in general video game playing. However, RHEA has been shown to outperform MCTS in multiple games in some of its variations (17), while other combinations of modifications led to significantly worse results. As highlighted by Lucas et al. (34), there can be a large difference in performance for the same base algorithm when using different parameters, and optimisation is essential. Ashlock et al. (149) emphasise this in the context of general game playing, where one single method (or single parameter configuration, in our approach) is unlikely to achieve high performance across all possible tasks. Our specific problem is additionally highly noisy: most games are stochastic and the same sequence of actions in a game could lead to different outcomes; furthermore, the algorithm itself is stochastic and may produce different outputs given the same game state.

In this section we use the N-Tuple Bandit Evolutionary Algorithm (NTBEA) (31) for optimising RHEA parameters, an algorithm which has shown robust high performance in noisy optimisation problems, even when compared with alternatives. It features high sample efficiency, fast convergence and good scaling for large search spaces (34). Simulations of AI players on a multitude of games can be very expensive, therefore sample efficiency is key, making NTBEA suitable for optimising RHEA parameters. The algorithm has been previously successfully employed in several noisy optimisation problems, such as tuning game parameters (160) as well as AI game player parameters (133; 34; 161). A highly adaptive system which can optimise its parameters and structure so as to achieve best performance in various games could easily feed into a generic lifelong learning system such as that presented in (24).

A summary of the contributions in this section is as follows:

1. We perform an in-depth analysis and optimisation of the algorithm’s parameters with respect to its performance across the various games tested.
2. We find new configurations which outperform the previous state-of-the-art on a range of GVGAI games.

6.1.1 N-Tuple Bandit Evolutionary Algorithm

NTBEA is a model-based optimiser based on an Evolutionary Algorithm. It begins by randomly initialising a solution o , or with a given solution (referred to as *seed*, and the process as *seeding* the algorithm). It then evaluates one solution at a time, with the fitness function determined by the specific application. Small random noise is added to each solution’s fitness value to distinguish between equally good solutions.

n-tuple Model Update

The internal *n*-tuple model is then updated. In this context, an *n*-tuple is a list of length n , containing non-repeating numbers from 0 to $L - 1$ (where L is the solution length); these numbers map to indexes in the solution evaluated. Statistics on all possible *n*-tuples are registered in the update step, keeping track of the number of times each *n*-tuple was sampled, how many times the indexed genes in the solution evaluated were observed to have those specific values, and the fitness value of the solution evaluated. We use 1, 2 and L values for n .

For example, consider solution (2, 3, 0, 0, 1) with fitness value 5. The 2-tuple (0,3) looks at indexes 0 and 3 in the solution, which map to specific gene values (2, 0). In the model, the number of times the (0,3) tuple was sampled is increased by 1, the number of times it mapped to gene values (2,0) is increased by 1, and value 5 is added to the sum value of the tuple. Similarly, all other combinations of 1, 2 and L tuples are added to the model as well.

Neighbourhood Evaluation

In the next step, several neighbours ($x = 50$) of the solution are generated through uniform random mutation, with probability $1/L$. The fitness of each neighbour is estimated based on the n -tuple model: all n -tuples are extracted from a neighbour o (let m be the total number of n -tuples), and statistics for each n -tuple are looked up in the internal model. For each n -tuple, we can then calculate its UCB value: if T_j is the j^{th} n -tuple, then $Q(T_j)$ is the average value observed for the n -tuple, $N(T_j)$ is the number of times the n -tuple was sampled, and $N(T_j, o)$ is the number of times the n -tuple was sampled and indexed the same gene values as in solution o . The constant $k = 2$ sets the focus of the algorithm, whether more exploitative or more exploratory. To obtain the overall UCB value for solution o , we calculate the average of UCB values for all n -tuples and add small random noise (maximum $\epsilon = 0.5$) to randomly break ties (see Equation 6.1).

$$UCB_o = \frac{1}{m} \sum_{j=1}^m \left(Q(T_j) + k \times \sqrt{\frac{\ln N(T_j)}{N(T_j, o)}} \right) + noise \quad (6.1)$$

Repeat

Finally, we choose the neighbour with the highest UCB value to be the next solution evaluated (o') and the process repeats for a set number of iterations. The final solution recommended is the one with the highest $Q(T)$ value averaged over all n -tuples. We refer the reader to (133) for more details on the NTBEA algorithm.

6.1.2 Experiments

In this study, we consider all RHEA parameters described in Section 3.1. Given the large number of parameter combinations, estimated at 5.36×10^8 , it would take a significant amount of time to test each combination exhaustively in several games and with repetitions for statistical significance. Therefore we choose to analyse the different parameters indirectly through the evolutionary process described by an N-Tuple Bandit Evolutionary Algorithm (NTBEA). We ran NTBEA for 1500 iterations individually on each of the 20 games described in Section 2.1.2 to perform a search through the RHEA parameter space depicted in Figure 6.1, obtaining a (potentially different) parameter set recommendation for each game. Each individual evaluated by NTBEA would therefore be one parameter combination (18 individual length). We seed NTBEA with the previous state-of-the-art (SotA) parameter configuration for each game (see rows with citation in Table 6.1). To evaluate each individual, we run RHEA with the specific parameter configuration on the given game, once in each of the 5 levels of the game and we use the average win rate on the 5 levels as individual fitness. To test the final configuration, we run it 100 times on the given game (20 times per level) and we additionally test the tuned parameter configuration on the entire set of 20 games, similarly with 100 runs per game.

All experiments were run on IBM System X iDataPlex dx360 M3 Server nodes, with one game per node, having one Intel Xeon E5645 processor core allocated to it and a maximum of 3GB of RAM of JVM Heap Memory. The runs took between 43 hours and 6 days to complete, including NTBEA tuning and final configuration testing; one run of a game can take up to 2000 game ticks to complete, with 1000 forward model calls per tick for AI decision-making (plus game engine computations), the fastest game ending after 50 game ticks on average. The budget for all agents was set as 1000 forward model calls instead of time limits (which averages as the equivalent of 40ms in our tests), in order for the experiments to be consistent and replicable across different machines.

In the following sections we aim to analyse not only the performance of the optimised agents on the different games, but also the parameter space explored during the evolution and the parameter choices themselves. We hypothesise that similar games would lead to similar choices in parameters, which would differ across game types.

This section presents and discusses the most interesting aspects observed, but all results, plotting scripts and additional figures are available on Github¹. We refer the reader to (15; 22) for a comparison of this algorithm with MCTS.

Optimisation Effectiveness

We first discuss the effectiveness of the optimisation. We summarise in Table 6.1 the results obtained on all 20 games used for tuning RHEA parameters with NTBEA. For each game, we present the parameter configuration of the previous state-of-the-art (previous highest win rate recorded), its win rate and standard

¹<https://github.com/rdgain/ExperimentData/tree/NTBEA-RHEA-2019>

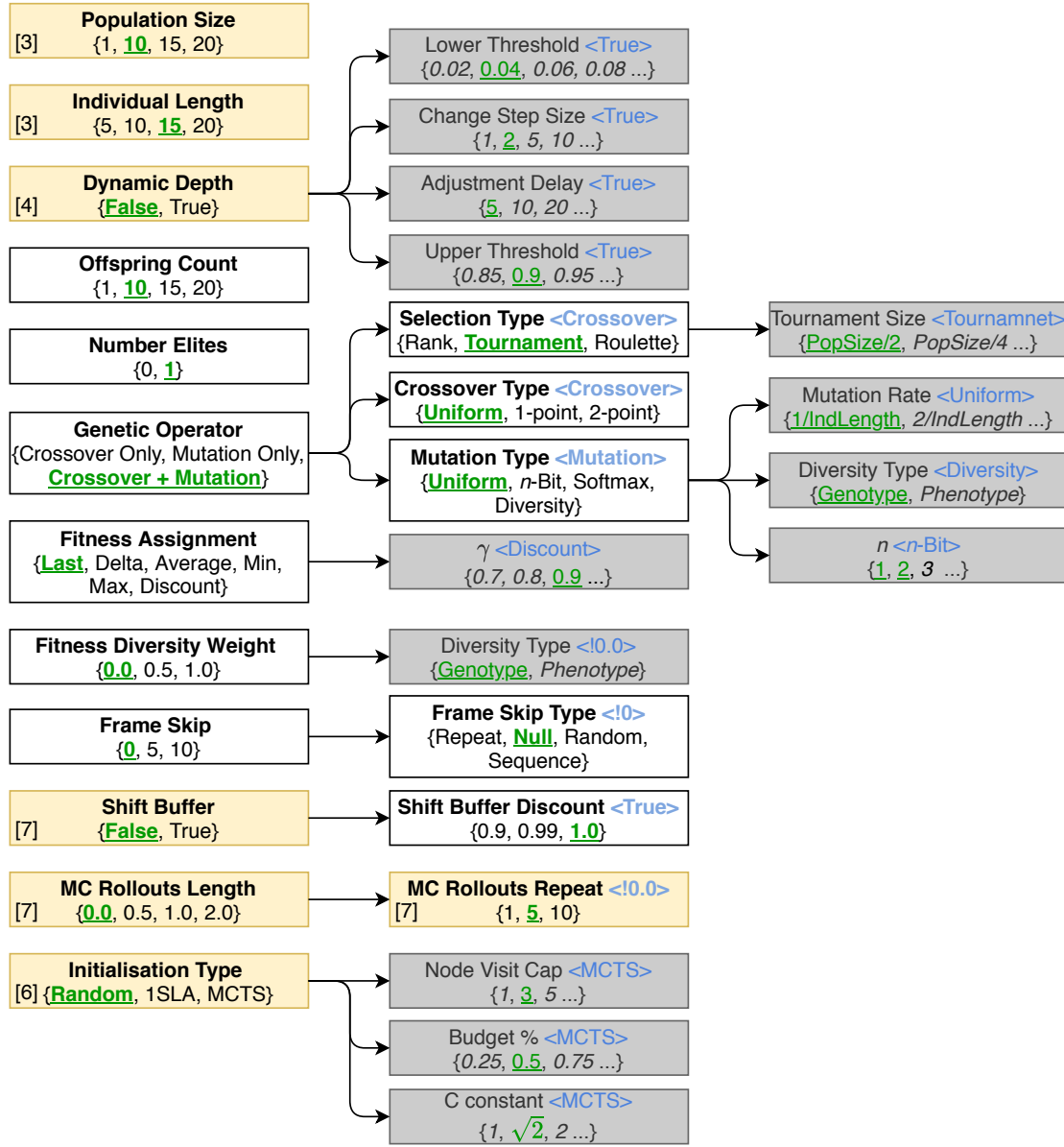


Figure 6.1: Parameter search space, size 5.36×10^8 (excluding dark grey boxes). Possible values for a parameter in curly brackets, default value in underline green. Parameters not in the 1st column are dependent on others (denoted with arrows from parent to dependent; parent value required for dependent to affect phenotype is noted in blue angled brackets). Dark grey parameters are not included in the experiments for this paper (default values used instead). In yellow parameters previously analysed in literature, with citation.

error; similarly, we present the optimised configuration for each game. Out of 20 games, 8 show worse results after optimisation, 6 observe similar results and in 6 we see an increased win rate. There are many games in which the win rate remains at or very close to 0%. This set of games (“Dig Dug”, “Lemmings”, “Roguelike”) remains too difficult for these methods to solve without more game-specific information or better exploration policies.

There are also several games which see win rates at, or very close to, 100% (“Intersection”, “Aliens”, “Infection”, “Chopper” and “Plaque Attack”). We do not see a decrease in performance in these games after optimisation (but a definite increase in “Plaque Attack” to 100% with several modifications in parameter choices, including using dynamic depth, 1SLA initialisation and random frame-skip).

We do see several games improving performance: the win rate in “Seaquest” increases from 65% to 84% by employing longer individual lengths, a larger population size, a shift buffer and MC rollouts. “Missile Command” sees an increase in win rate from 78% to 86% with a shift buffer, MC rollouts and a discounted fitness assignment. And performance in “Camel Race” increases from 11% to 41% by using *repetition* frame-skip, a shift buffer and MC rollouts, among many configuration modifications. These 3 games do not immediately show common features as per Table 2.1, with “Seaquest” standing out due to its stochastic

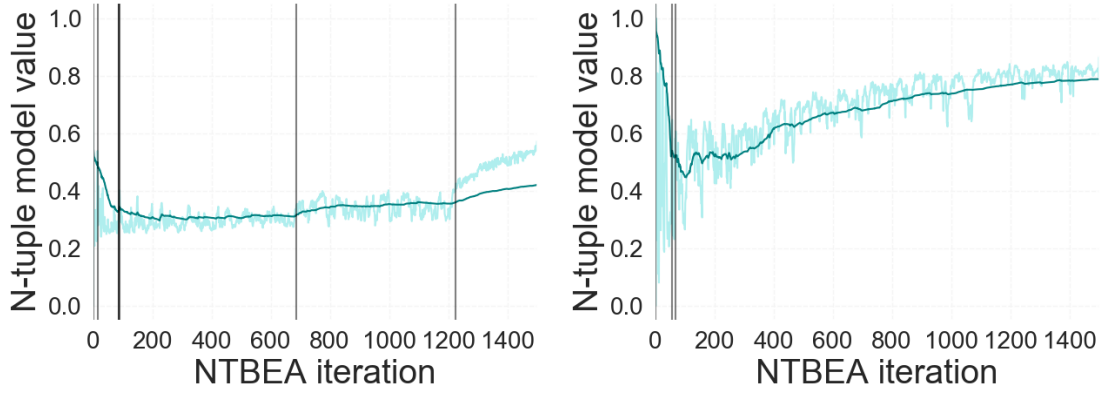


Figure 6.2: Progression of solution fitness in the internal n -tuple model during NTBEA optimisation process in 2 games, “Missile Command” (left) and “Intersection” (right): plotting the value in the model for the solution evaluated at each iteration (light blue). The darker line indicates the value in the model for the seeded solution at each iteration (after n -tuple updates). Vertical lines indicate when the algorithm changes the best solution recommendation, based on its internal n -tuple model.

nature and dense environment, while the other two feature sparser deterministic environments.

We see a considerable decrease in performance in three of the games: “Butterflies” (from 96% to 90%), “Escape” (from 46% to 32%) and “Modality” (from 37.5% to 25%). As NTBEA was seeded with the previously best solution, we believe these are cases in which the noisy fitness evaluation was shown to be most harmful. Solutions are only evaluated once in each level of a game to obtain their true fitness value, while the final solutions are evaluated 20 times per level. Therefore, it is likely that the initial stochastic evaluation of the seed was misleading, leading the algorithm towards other areas of the search space. Additionally, since the value used in solution recommendations by NTBEA is based on its internal n -tuple model, this could also be a problem of credit assignment: all n -tuples are weighed equally, whereas they do not equally impact the phenotype. Future work will consider a better approach for tackling these difficult environments. A similar smaller decrease is also observed in “Lemmings” and “Bait” - all of these, except for “Butterflies”, are games with puzzle elements to them, which appear to be most difficult to optimise and estimate solution quality for, as they require more precise action sequences, with one move possibly making the game unsolvable, and therefore more precise evaluation. For these games, increasing the number of evaluations used in the fitness function would likely reduce the noise, as would changing the balance in the UCB calculations towards exploitation.

Finally, we highlight NTBEA’s optimisation process progression in two games in Figure 6.2, “Missile Command” and “Intersection”. Both of these games see an upwards trend in solution quality, and they represent the games with the slowest and fastest convergence, respectively. We can observe that the algorithm settles on the solution for “Intersection” very quickly, before iteration 100, whereas it uses almost all computation budget for “Missile Command” to find the best option. This could be an indication of not only game difficulty, but also strategic depth: most parameter options work well and obtain very good performance in “Intersection”, while “Missile Command” poses a challenge at which not many options are successful and the finding of those few good configurations is more difficult. We note that the upward trend after settling on the best solution comes from the value of that best solution improving over time as more samples are added into the n -tuple model. Most games converged to a stable solution recommendation before the 1500 budget was exhausted. However, we expect solution quality to improve further with even more resources (e.g. Figure 6.2 shows Missile Command solution quality increasing and finding new bests up to very close to the given budget). Due to stochastic evaluations, different runs of the optimisation process may produce different results.

Overall, the game-specific optimised agents achieve win rates of below 50% when tested on the entire set of games, which is not surprising in the general game playing context; the agents do not use any game-specific information. The best performing tuned agent is that for “Seaquest” (53.4 average win rate on all 20 games), which shares different features with several other games; this appears to make the specific configuration more generally applicable than the others.

1-tuple Analysis

Next, we look into the parameter space explored by NTBEA, starting with 1-tuples, or analysing each parameter and its preferred values in isolation in the different games tested; we use the term *prefer* to mean

Table 6.1: RHEA best win rate (and standard error) recorded in all games. “opt” rows show NTBEA optimisation results, other rows show previously best recorded (with corresponding citation, highlighted in yellow). Parameters using default values (as per Figure 6.1) highlighted in green. Enhancements include values for dependants in brackets. Win-rates in bold are the higher values observed, if different.

Game	Win Rate	Parameters										
		Numerical				Nominal					Enhancements	
		P.Size	ILen	Offspring	Elite	Init.	Selection	Crossover	Mutation	Fit.		
0	0% (0.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	0% (0.00)	1	20	15	1	MCTS	Rank	Uniform	2-bit	Last	SB(0.9); MC(0.5,1); Skip(Rep)	opt
1	4% (1.98)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	DD	(19)
	0% (0.00)	10	10	1	0	RND	Tourn.	1-point	2-bit	Last	SB(0.9); MC(0.5,5); Skip(Rep); DD	opt
2	0% (0.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	0% (0.00)	1	20	15	1	MCTS	Rank	Uniform	2-bit	Last	SB(0.9); MC(0.5,1); Skip(RND)	opt
3	100% (0.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	99% (0.99)	10	15	20	1	ISLA	-	-	Uniform	Disc.	SB(0.9); MC(1.0,5); DD	opt
4	10% (3.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	10% (3.00)	20	10	15	0	ISLA	-	-	2-bit	Disc.	SB(0.9); DD	opt
5	13% (3.39)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	3% (1.70)	10	20	20	1	RND	Tourn.	2-point	-	Average	MC(0.5,1)	opt
6	11% (3.13)	1	10	10	1	RND	Tourn.	Uniform	Uniform	Last	MC(0.5,10)	(17)
	41% (4.92)	15	15	1	1	MCTS	Tourn.	Uniform	Diversity	Disc.	SB(0.99); MC(1.0,5); Skip(Rep); Fit.Div(0.5)	opt
7	46% (4.98)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	MC(0.5,1)	(17)
	32% (4.665)	20	10	10	0	RND	-	-	Diversity	Disc.	SB(0.9); Fit.Div(0.5)	opt
8	12% (3.25)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	SB(0.9); MC(0.5,10)	(17)
	11% (3.13)	10	20	15	0	ISLA	Rank	2-point	2-bit	Max	SB(0.99); MC(2.0,5)	opt
9	20% (4.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	SB(0.9); MC(0.5,10)	(17)
	19% (3.923)	10	20	10	1	RND	Tourn.	1-point	Uniform	Max	SB(0.99); MC(1.0,5)	opt
10	78% (3.76)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	83% (3.76)	10	20	10	1	ISLA	Tourn.	1-point	-	Disc.	SB(0.99); MC(2.0,1); Skip(Null); DD	opt
11	55% (5.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	56% (4.97)	20	15	10	1	RND	Tourn.	Uniform	Uniform	Disc.	SB(0.99); MC(2.0,5); Skip(RND); DD	opt
12	38% (4.42)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	25% (4.33)	10	20	15	0	RND	Tourn.	1-point	Diversity	Max	MC(0.5,10); Skip(Seq); DD	opt
13	78% (4.18)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	86% (3.47)	15	20	15	1	RND	Tourn.	1-point	Softmax	Max	SB(0.9); MC(2.0,1)	opt
14	99% (1.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	100% (0.00)	15	20	1	1	ISLA	Rank	1-point	Diversity	Max	SB(1.0); MC(2.0,1); Skip(RND); DD	opt
15	65% (4.77)	1	5	1	1	RND	Tourn.	Uniform	Uniform	Last	SBer(0.9) MC(0.5,10)	(17)
	84% (3.66)	20	20	1	1	RND	Rank	Uniform	-	Average	SB(0.9); MC(2.0,1)	opt
16	100% (0.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	99% (0.99)	15	20	1	0	RND	-	-	2-bit	Last	SB(0.99); MC(0.5,10); Skip(RND); DD	opt
17	100% (0.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	100% (0.00)	15	20	20	0	RND	-	-	2-bit	Disc.	SB(0.9); DD	opt
18	96% (1.92)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	90% (3.00)	15	5	20	0	RND	Roulette	2-point	-	Disc.	SB(0.99); MC(2.0,5)	opt
19	100% (0.00)	10	15	10	1	RND	Tourn.	Uniform	Uniform	Last	-	(15)
	100% (0.00)	10	5	1	1	ISLA	Roulette	1-point	2-bit	Disc.	MC(2.0,5); DD	opt

the value achieves highest win rate. We group together the solutions in which the parameter had the same value chosen and plot the average fitness of these solutions against parameter values. We note that the parameter may have not had a great influence in the fitness obtained, and we exclude the data points where the parameter had no influence at all, in the case of dependent parameters (see Figure 6.1). The resulting heatmaps show the fitness values observed for each parameter values, as well as how many times each parameter value was explored by NTBEA. The latter is given by the circle size in the figures presented; we cap the maximum number of occurrences of a data point at 100 and normalise all values in $[0, 1]$ for visualisation purposes.

17 games prefer the shift buffer turned on and to keep 1 elite between generations. Additionally, as previously seen in (15), most games prefer long individual length and large population sizes. 17 games further prefer the agent employing Monte Carlo rollouts at the end of its individual evaluation: “Chopper”, “Seaquest” and “Missile Command” in particular prefer very long rollouts ($2.0 \times L$, see Figure 6.3); this could be due to these three games featuring different types of rewards and delays in obtaining rewards. The other similar game in terms of rewards, “Intersection”, does not show a particular preference in this parameter, achieving 1.0 fitness in all values.

In terms of genetic operators, most games prefer the agent to use both mutation and crossover in its evolutionary process. However, there are some exceptions: “Chopper” and “Plaque Attack” prefer to use mutation only, whereas “Missile Command” prefers options that do include crossover and more disturbance in its offspring (see Figure 6.4). Although these games are seen as similar in (3) and obtain high winning rates, the way the agents achieve their good performance does differ in these games, suggesting methods

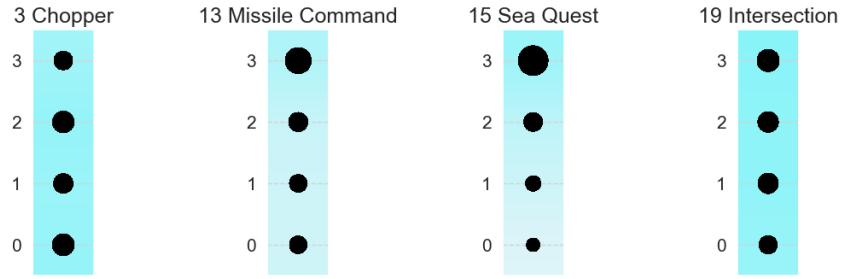


Figure 6.3: 1-tuple: rollout length percentage parameter. Colours show average fitness for each data point, with blue being highest (1.0) and white being lowest (0.0). Each data point is highlighted with a black circle; the larger the circle, the more times that parameter value was sampled.

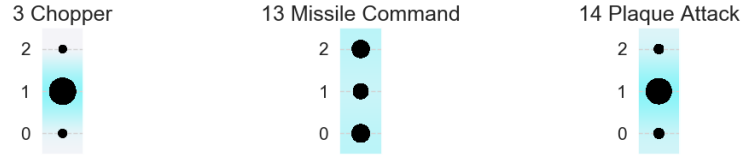


Figure 6.4: 1-tuple: genetic operator parameter. 0 - crossover and mutation. 1 - mutation only. 2 - crossover only.



Figure 6.5: 1-tuple: offspring count parameter.

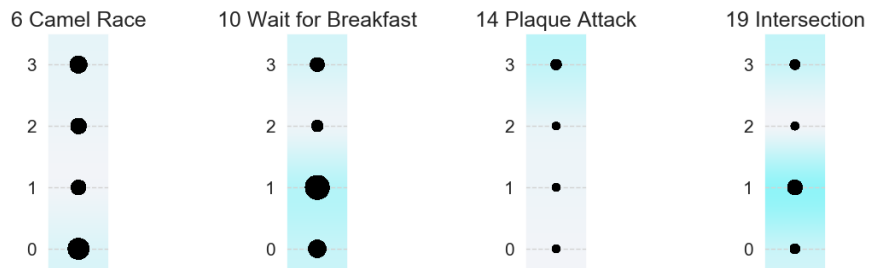


Figure 6.6: 1-tuple: frame-skip type parameter. 0 - repeat. 1 - null. 2 - random. 3 - sequence.

based on win rates could be improved by taking into account agent-based features.

When looking at the number of offspring (see Figure 6.5), “Survive Zombies”, “Missile Command” and “Chopper” prefer more. These games are quite similar in terms of features (win/lose conditions, level sizes, enemy NPCs) and are clustered together in (3). However, in the same cluster, “Butterflies” and “Plaque Attack” do not show strong preference here - as opposed to the others, these two games have a smoother score progression, while “Missile Command” and “Chopper” show more delay in getting rewards, more actions are required from the player to find particular rewarding scenarios. “Sequest” is placed in a similar cluster by (1), but it shows opposite preference, for less offspring instead. In this game we see large

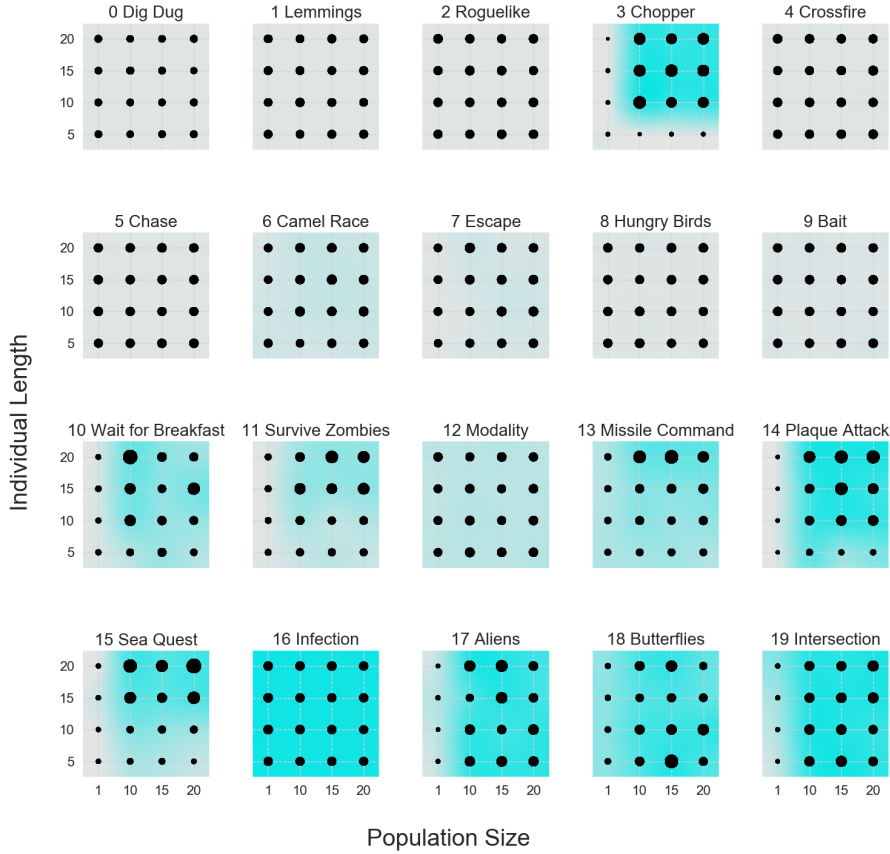


Figure 6.7: 2-tuple: individual length and population size. Colours show average fitness for each data point, with blue being highest (1.0) and white being lowest (0.0). Each data point is highlighted with a black circle; the larger the circle, the more times that combination of values was sampled.

discontinuous rewards as well as many smaller dense rewards - the larger variety in types of rewards could be what leads to favouring less solutions sampled to increase the number of generations in the evolutionary process, and to gain better insight into which reward type is preferable.

Lastly, we highlight that “Intersection” and “Wait for Breakfast” are the only games that benefit from *null* frame-skipping (see Figure 6.6) - in both of these games it is essential to wait for specific events to happen (a way in the road to clear, or the waiter to arrive). “Plaque Attack” prefers *sequence* frame-skipping, as plans evolved are precise enough in line with the constant stream of rewards. And “Camel Race” prefers *repeat* or *sequence*, with more frames skipped being better, which are effective strategies of exploring large sparse environments. Most other games dislike frame-skipping and prefer a more fine-grained search; however, we note that the search space for this parameter is very coarse and it might be that more games could benefit from *some* or dynamic frame-skipping.

2-tuples Analysis

Similarly as with 1-tuples, we can look at how combinations of parameters affect overall solution fitness. In this section we group together solutions which had the same values for each parameter combination, while eliminating the data points where either one or both of the parameter values did not impact solution phenotype, in case of dependent parameters (see Figure 6.1). We plot each parameter against all others in the different games tested, each data point representing the average fitness observed in the respective group of solutions. We further add black circles on each data point to highlight the number of times each combination was explored by NTBEA during the optimisation process.

The first thing that stands out in all resulting figures is that NTBEA explores the best combinations the most, while mostly ignoring less promising options. This can be seen as a direct confirmation of the effectiveness of the bandit-based approach, but also as a potential point of improvement: due to the nature of the very noisy optimisation, it might be beneficial to obtain more accurate estimates of some data points which do not immediately stand out as the best: as discussed in a previous section, it was the case in several games that the optimised solution ended up performing worse than the initial solution given to the

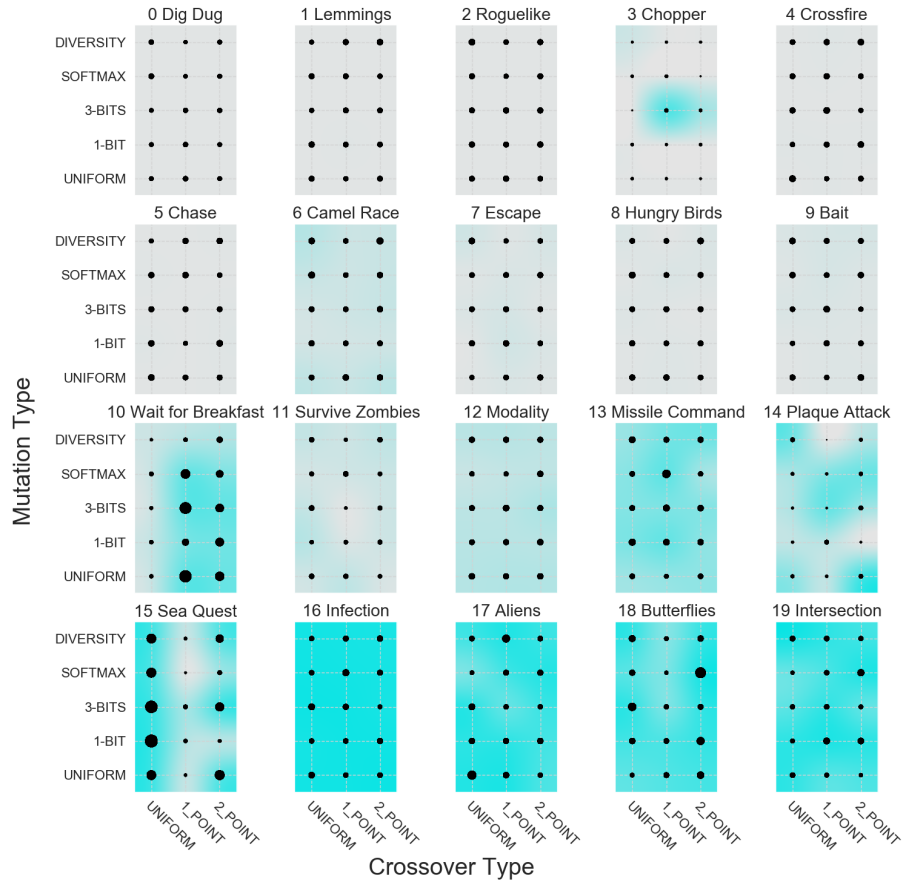


Figure 6.8: 2-tuple: mutation type and crossover type.

algorithm, which could have been avoided had a more accurate evaluation of solution quality been done.

In Figure 6.7 we can observe the combination of individual length L and population size P parameters. We have previously observed that longer individuals lead to higher fitness values, and similar for larger population sizes. It is interesting to see that this holds true also for the combination of L and P , although specific combinations achieve better results in some games (such as $L = 20$ and $P = 15$ in “Chopper”).

Another interesting parameter combination to discuss is that of mutation type and crossover type, shown in Figure 6.8, which largely decides how offspring are created at each generation. Although the overall fitness of solutions differs, games “Hungry Birds” and “Plaque Attack” show a similar distribution of good or bad quality combinations: in particular, 1-point crossover does not agree with diversity mutation, and 2-point crossover does not agree with bit mutation. This could largely be due to the specific modifications n -point crossover wishes to generate, which are modified unexpectedly by bit mutation. However, these two games singled out here do not appear to have much in common according to our feature descriptions and clustering in Table 2.1; it is thus interesting to find game similarities beyond those given by traditionally-employed features.

6.2 Online

The work in this section was published at IEEE CoG 2020:

R. D. Gaina, C. F. Sironi, M. H. Winands, D. Perez-Liebana, and S. M. Lucas, “Self-Adaptive Rolling Horizon Evolutionary Algorithms for General Video Game Playing,” in *IEEE Conference on Games (CoG)*, 2020, pp. 367–374.

As we observed performance improvement and much variety in parameters chosen in the GVGAI games in the previous section, the next step is to try this optimisation process online, while the agent is playing the game, to encourage fast adaptation to different situations. Previous research has looked at such online tuning of MCTS players, exploring some of the parameters controlling the decision-making process of this algorithm during a single play-through of a game (32; 131). This work uses each iteration in the MCTS algorithm as a data point evaluated for a specific configuration of parameters, with new set of parameters chosen for every iteration based on the values returned. Thus, the optimiser and the search algorithm interleave execution to play a game: the search algorithm attempts to find the best solution for the game, while the optimiser attempts to find the best set of parameters for the search algorithm. No such method had been tried in evolutionary agents previous to this study.

This section describes a proposal to adapt the MCTS optimisation method to RHEA in a similar fashion: each RHEA iteration (a full generation of individual) would represent a data point for the optimiser, which will aim to maximise the fitness improvement between generations with the search parameters it selects. Due to the optimisation problem focused on the improvement from one generation to the next, we limit the set of parameters in this study to those which impact the generation of new individuals most directly: genetic operator, crossover type, selection type, mutation type and mutation transducer (see Section 3.2 for details on each and Table 6.2 for a summary).

Several optimisers were tried previously for this purpose with MCTS agents (32; 131) and we adopt all to study not only their performance in combination with RHEA, but also their differences and individual efficacy. The most successful previous approaches were based on evolutionary algorithms: a standard Genetic Algorithm (GA) and a N-Tuple Bandit Evolutionary Algorithm (NTBEA) (133), as seen previously used in offline tuning. We further add to this set of optimisers Naïve Monte Carlo (NMC) (162), which performed best for MCTS out of all non-evolutionary techniques tested. Simple baselines are included as well in the form of a Multi-Armed Bandit (MAB), which does not take into account the relationships between parameters and therefore the combinatorial nature of the problem, and Random (RND), which simply selects random sets of parameters without storing any statistics of their values.

The contributions of this work are twofold. First, we propose an adaptation of online tuning methods for RHEA and test the performance of the tuned RHEA agents on a variety of different problems, highlighting strengths and weaknesses. Second, we perform an in-depth analysis of the algorithm parameters from the perspective of the statistics gathered by the optimisers, with the aim of obtaining more insight into the inner-workings of the algorithm.

6.2.1 Approach

In order to implement a self-adaptive RHEA agent, the parameter space has to be defined and the online parameter tuning method used for MCTS has to be adapted. Moreover, optimisers that decide how to allocate the available samples to evaluate different parameter combinations are necessary.

RHEA Parameter Space

Although several modifications and parameters of RHEA have been previously studied (15; 16; 17; 19), we focus here on those parameters which have most impact in any one iteration of the algorithm - that is, those that impact the generation of offspring, as summarised in Table 6.2.

Genetic operator. This parameter controls which genetic operators are applied: crossover only (offspring are not mutated), mutation only (offspring are obtained by directly mutating each individual in the population), or both (offspring are generated through crossover, and then mutated). The rest of the parameters may only have an effect on the phenotype if this parameter has a particular value.

Selection type. This operator is used to select parents for crossover. Roulette selection chooses individuals based on probabilities directly proportional to their fitness. Rank selection chooses individuals based on probabilities inversely proportional to their rank in the ordered list of individuals (best individuals first). Tournament selection randomly chooses 40% of individuals, then the best out of the random selection.

Crossover type. This operator is used to combine selected individuals and generate offspring. Uniform crossover randomly selects a value for each gene from either of the parents. n -point crossover divides the

Table 6.2: Parameter Search Space, total size 270, or 99 valid combinations if parameter dependency is taken into account.

Idx	Name	Values
0	Genetic Operator	Crossover + Mutation, Mutation Only, Crossover Only
1	Selection Type	Rank, Tournament, Roulette
2	Crossover Type	Uniform, 1-point, 2-point
3	Mutation Type	Uniform, 1-Bit, 3-Bits, Softmax, Diversity
4	Mutation Transducer	False, True

Algorithm 7 RHEA action selection with online parameter tuning.

```

1: function GETACTION( $\mathcal{T}, \lambda, \mu$ )
2:   Input: Tuner  $\mathcal{T}$ , population size  $\lambda$ , elite size  $\mu$ .
3:   Output: First action of best individual in the population
4:    $pop \leftarrow \text{GETCURRENTPOPULATION}()$ 
5:   order individuals in  $pop$  by decreasing fitness
6:   while time not elapsed do
7:      $\vec{p} \leftarrow \mathcal{T}.\text{SELECTPARAMVALUES}()$ 
8:      $\text{SETPARAMVALUES}(\vec{p})$ 
9:      $offspring \leftarrow \text{GENANDEVALOFFSPRING}(pop_{[0, \dots, \mu]})$ 
10:    order individuals in  $offspring$  by decreasing fitness
11:     $r \leftarrow offspring_0.\text{fitness} - pop_0.\text{fitness}$ 
12:     $\mathcal{T}.\text{UPDATEVALUESTATS}(\vec{p}, r)$ 
13:     $pop_{[\mu+1, \dots, \lambda]} \leftarrow offspring_{[0, \dots, \lambda-\mu]}$ 
14:    order individuals in  $pop$  by decreasing fitness
15:  return  $pop_0.\text{GETFIRSTACTION}()$ 

```

individual into $n + 1$ slices and alternatively selects the corresponding slices from each parent; we use $n = \{1, 2\}$.

Mutation type. This operator is used to modify offspring. Uniform mutation changes each gene of the individual with a probability of $1/L$. n -bit mutation changes n random genes; we use $n = \{1, 3\}$. Softmax mutation uses Equation 6.2 to bias mutation towards the beginning of the individual, where changes in genes most affect the phenotype. Diversity mutation logs values explored for all genes and chooses to mutate the gene explored the least, to its least visited value.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (6.2)$$

Mutation transducer. This parameter is used to decide new values for mutating genes (no effect in diversity mutation). If this flag is off or the gene is first in the individual, the gene will take a random new value. Otherwise, it will take the value of the previous gene in the sequence; this aims to decrease the jitteriness of the agent, by encouraging action repetition.

Online Parameter Tuning for RHEA

The parameters we are aiming to tune in RHEA control how the entire population of individuals is evolved. Therefore, we need to use the entire population to evaluate the quality of a parameter combination. Algorithm 7 shows how online parameter tuning is implemented by the RHEA agent when it has to choose an action for a given game state. Note that the procedure assumes that the RHEA population has already been initialised and evaluated once. Until the search budget expires, the procedure repeats the following steps:

1. Select a new combination of parameter values using the tuner \mathcal{T} (line 7).
2. Set the tuned parameters to the selected values (line 8).
3. Using the elite individuals in the current population, generate new offspring, evaluate (line 9) and order them by decreasing fitness (line 10).
4. Compute the payoff r for the selected combination of parameter values as the difference between the fitness of the best individual in the offspring and the fitness of the best individual in the previous generation (line 11).

5. Update the statistics of the selected parameter combination using the computed payoff (line 12).
6. Update the population by replacing the worst $\lambda - \mu$ individuals with the generated offspring and order it by decreasing fitness (lines 13 and 14).

When the budget expires, the first action of the best individual in the population is returned (line 15).

Note that, to compute the payoff of a parameter combination, we only consider the fitness of the individuals in the offspring and not of the individuals in the entire new population. This is because the parameter values set for an iteration of RHEA only influence how such offspring is generated. Moreover, to compute the payoff of a combination of parameter values we consider only the best individual in the offspring and in the population, because we are interested in finding the parameters that can generate the best possible individual, even if the rest of the population has a low fitness. The action that will be played in the real game is taken from such individual, therefore it will be the only individual that will influence the real game.

Multi-Armed Bandits

The MAB problem (130) is characterised by m independent arms, each of which is associated with a reward distribution. When an arm is played, a reward is obtained as a sample of the corresponding distribution. The goal of a sampling strategy for a MAB is to maximise the sum of rewards obtained by successive plays of the arms. Thus, the strategy has to balance exploration of less sampled arms in order to learn their distribution, with exploitation of arms that produced a high reward. A variety of sampling strategies have been proposed. One of the most used is UCB1 (130), which, in each iteration, selects the arm a^* as shown in Equation 6.3.

$$a^* = \arg \max_{a \in A} \left\{ \bar{q}_a + C \times \sqrt{\frac{\ln n}{n_a}} \right\} \quad (6.3)$$

Here, A is the set of available arms, \bar{q}_a is the average payoff over all the plays of arm a , n is the total number of samples from arms, n_a is the number of samples from arm a , and C is the constant that controls the balance between exploitation of good moves and exploration of less visited ones.

Optimisers

The parameters of a game-playing agent can be seen as a vector. Therefore, the problem of tuning these parameters for each new game consists in searching the optimal vector of parameters values in a combinatorial search space. Previous work (131) defined this problem as a Combinatorial Multi-Armed Bandit, characterised by the following three components:

- Vector of d variables, $\vec{P} = \{P_1, \dots, P_d\}$, where each variable P_i can take m_i different values $V_i = \{v_i^1, \dots, v_i^{m_i}\}$.
- Reward distribution $R : V_1 \times \dots \times V_d \leftarrow \mathbb{R}$ that depends on the combination of values assigned to the variables.
- Function $L : V_1 \times \dots \times V_d \rightarrow \{true, false\}$ that determines which combinations of values are legal.

There are various optimisers that decide which combinations of parameter values should be evaluated, how many times and in which order (32; 131). The ones considered in this study are described below.

Multi-Armed Bandit (MAB) A straightforward solution to deal with a combinatorial multi-armed bandit problem is to translate it to a Multi-Armed Bandit (MAB) (130). Each arm of the MAB corresponds to a possible legal combination of values for the parameters. Therefore, selecting the next combination of parameter values to evaluate corresponds to choosing one arm of the MAB. Here, UCB1 is used as sampling strategy for the MAB, with exploration constant $C_{\text{MAB}} = 0.7$ (this value is taken from previous work (135)). Note that, differently from other optimisers used here, the MAB optimiser ignores the information on the combinatorial structure of the parameter values. This strategy does not exploit the fact that often a value that is good (or bad) for a parameter in a certain combination of values, is also good (or bad) in general or in many other combinations.

Naïve Monte Carlo (NMC) First proposed to play real-time strategy games (162), it was later applied as an optimiser to tune MCTS parameters online (32; 131). The NMC optimiser is based on the naïve assumption that the reward associated to a combination of d parameter values, $\vec{p} = \langle p_1, \dots, p_d \rangle$, can be approximated by a linear combination of the rewards associated to single parameter values p_i ($i \in \{1, \dots, d\}$). This means $R(\vec{p}) \approx \sum_{i=1}^d R_i(p_i)$.

When choosing a combination of parameter values, NMC alternates between *exploration* and *exploitation*. In a given iteration, NMC performs exploration with probability ϵ_0 and exploitation with probability $(1 - \epsilon_0)$. During exploration, for each parameter being tuned NMC considers a local multi-armed bandit problem, where each arm corresponds to one of the possible values for the associated parameter. A value for each parameter is selected independently from each local multi-armed bandit using the UCB1 sampling strategy with exploration constant C_L . During exploitation, NMC considers a global multi-armed bandit, where each arm corresponds to a possible combination of parameter values. A combination is selected from the global multi-armed bandit using the UCB1 sampling strategy with exploration constant c_G . At the start of the execution of NMC, the global multi-armed bandit has no arms. A new arm is added every time a new combination of parameter values is generated during exploration with the local multi-armed bandits. After evaluating a selected combination of parameter values, NMC uses the obtained payoff to update both the statistics of the combination in the global bandit and the statistics of each single parameter value in the local bandits. In these experiments, the settings for this strategy are the same used in GVGP to tune MCTS in previous work (32): $\epsilon_0 = 0.75$, $C_L = 0.7$, $c_G = 0.7$.

Genetic Algorithm (GA) Genetic Algorithms (EAs) (163) are optimisation algorithms that search for an optimal solution by evolving a population of candidate solutions using a variety of genetic operators. As shown in (32) and (131), a GA with population size λ_{GA} and elite size μ_{GA} can be used as optimiser for the CMAB that represents the parameter tuning problem. A combination of parameter values can be considered as an individual in the population and each single parameter as gene. The GA optimiser starts with a randomly generated population of parameter combinations and then evolves it multiple times until the computational budget expires.

During each iteration, GA first evaluates each combination in the population as shown in Algorithm 7, i.e. using it to control the generation of the new population in one iteration of the RHEA algorithm. This means that the fitness of a combination of parameters corresponds to the payoff computed at line 11 in Algorithm 7. Subsequently, GA keeps the μ_{GA} parameter combinations with the highest fitness in the current population (the elite) and uses them to generate the remaining $\lambda_{GA} - \mu_{GA}$ new combinations. Each new combination is generated with probability p_{cross} by uniform random crossover between two randomly selected elite individuals, and with probability $(1 - p_{cross})$ by uniformly mutating one bit of a randomly selected elite individual. In these experiments, the settings for this strategy are the same used in GVGP to tune MCTS in previous work (32): $\lambda_{GA} = 50$, $\mu_{GA} = 25$, $p_{cross} = 0.5$.

N-Tuple Bandit Evolutionary Algorithm (NTBEA) First proposed for game parameter tuning (31), NTBEA has also been successfully used for offline optimisation of a game-playing RHEA agent (133) and has been applied to online parameter tuning for MCTS (32; 131).

Like the GA optimiser, NTBEA considers each combination of parameter values as an individual and each single parameter as a gene. In addition, NTBEA uses an *N-Tuple fitness landscape model* to memorise statistics about the parameters. The implementation of the landscape model used in these experiments, similarly to the NMC approach, keeps a local multi-armed bandit for each parameter and a global bandit for the combination of all the parameters. The fitness model can be used to quickly evaluate a parameter combination by computing its average UCB1 value over all the bandits. This quick evaluation is used by the evolutionary algorithm to speed up the evolutionary process, while balancing exploration and exploitation of the various parameter combinations.

More precisely, the NTBEA algorithm starts with a randomly generated combination of parameter values. During each iteration of the algorithm, the current combination of parameter values is used to control an iteration of RHEA. The obtained payoff is used to update the statistics in the fitness model that correspond to the evaluated combination. At this point, x neighbours of the evaluated combination are generated, each by mutating the value of a randomly selected parameter in the combination. The x neighbours are evaluated using the fitness model and the one with the highest average UCB1 value becomes the new considered combination. This process is repeated until the computational budget expires. As for the other optimisers, the settings used in these experiments are the same as in previous work (32): $x = 5$, $C_{NTBEA} = 0.7$ (exploration constant used to compute the UCB1 value with the fitness model).

Random (RND) The random optimiser has already been evaluated for online parameter adaptation both in abstract games (164) and video games (135). Despite its simplicity, parameter randomisation has been

shown to be beneficial in some games, especially when the fixed parameter settings are not optimal, or when time settings are short. The random optimiser selects parameter combinations randomly among all the feasible combinations of parameter values. This means that each iteration of the RHEA algorithm, and thus the generation of each new population, is controlled by a randomly selected combination of parameter values. This also means that no statistics collected about the quality of previously tested parameter values or parameter combination are exploited.

6.2.2 Experimental Setup

We test the performance of each optimiser in the same 20 games from the General Video Game AI framework (GVGAI (9)), as detailed in Section 2.1.2: “Dig Dug”, “Lemmings”, “Roguelike”, “Chopper”, “Crossfire”, “Chase”, “Camel Race”, “Escape”, “Hungry Birds”, “Bait”, “Wait for Breakfast”, “Survive Zombies”, “Modality”, “Missile Command”, “Plaque Attack”, “Seaquest”, “Infection”, “Aliens”, “Butterflies”, “Intersection”. Each of these games has 5 levels, which vary the positions and presence of sprites, as well as the map size.

We use different configurations for RHEA in this setting, varying budget, population size and individual length. Larger values for population size and individual length allow for less iterations during the agent’s thinking time, and therefore less data points for the tuners, but were generally shown to perform better (15). We set the default budget for the agent to 1000 forward model calls, which is the average non-tuned RHEA can perform in 40ms; this allows for robust results across different machines. Further experiments halve the budget, or increase the budget by 5 times to observe tuner performance outside of GVGAI bounds and with varying numbers of iterations. If we format tested algorithm names as “{individual length}—{population size}—{budget}”, we obtain 6 configurations: 5-10-500, 5-10-1000; 5-10-5000; 10-15-500; 10-15-1000 and 10-15-5000. Given that there are a total of 270 possible parameter combinations (see Table 6.2), none of the configurations are able to even sample all points at least once during a game tick, let alone gather accurate statistics on all the points, making sample efficiency key.

Each agent (combining a RHEA configuration and an optimiser) is run 20 times on each of the 5 levels of the 20 games, or 100 times per game. We record the final result of each game (win/loss, score and game tick). The results are compared with current state-of-the-art (SotA) in RHEA, i.e. highest win rates obtained by any previously explored configuration in each game; as a result, different games may have different RHEA configurations as SotA. Given the nature of the experiment, with parameters varied in tuned agents at every iteration, we consider these SotA results a very high bar, but a good comparison. It is worth noting that some of the enhancements that led to SotA results (e.g. Monte Carlo rollouts (17)) are not used in any of the tuned agents presented here, in order to increase the number of iterations available.

Further, we log the number of visits and average score for all combinations of parameters (5-tuples), and for individual parameters (1-tuples), at every game tick, for all tuners (even if they do not use statistics, e.g. RND).

We note that tuner statistics are not reset between game ticks. Initial experiments showed performance to be very similar regardless of discount factor used (0.0 and 0.8 experimented with). We speculate that this is due to the game states not varying widely from one game tick to the next and thus the statistics on parameter choices are more generally applicable. The only exception we noted was in the game “Crossfire”, where a discount of 0.8 showed an increase in performance; this warrants further investigation for highly dynamic games.

6.2.3 Results and Discussion

This section presents and discusses some of the more interesting results obtained. Full results, log summaries and plots are available on Github².

Win rate

We first look at the performance of the tuned agents in the 20 GVGAI games. For the purpose of this analysis, we only consider win rate (i.e. the agent’s ability to solve the problem). This is summarised for all RHEA configurations and tuner combinations in Table 6.3, with a particular RHEA configuration (10-15-1000) visualised in Figure 6.9. No agent is able to beat SotA results on all games, but win rates are largely comparable, and there are several which do perform better in some of the games, with some interesting cases standing out.

In the game “Crossfire”, 9 of the agents with population size 5 and individual length 10 are able to beat SotA by up to 8%, whereas none of the other variants with increased values in RHEA configuration are able

²<https://github.com/rdgain/ExperimentData/tree/RHEA-Online-Tuning-20>

Table 6.3: Results of all tuners for all RHEA configurations tested. Showing average win rate in all 20 games; average difference in win rate to RHEA SotA in games in which the tuned agent is better (Δ_{Better}) and worse (Δ_{Worse}), with number of games in brackets. Highest win rate, highest Δ_{Better} and lowest Δ_{Worse} are highlighted for each tuner.

Tuner	$P = 5, L = 10$				$P = 10, L = 15$			
	FM Calls	Win Rate	Δ_{Better}	Δ_{Worse}	FM Calls	Win Rate	Δ_{Better}	Δ_{Worse}
GA	500	40.80 (± 8.51)	0.00 (0)	12.25 (17)	500	43.75 (± 8.84)	0.00 (0)	8.78 (17)
	1000	43.15 (± 8.76)	0.00 (0)	11.52 (14)	1000	45.90 (± 8.79)	0.00 (0)	7.09 (15)
	5000	44.30 (± 8.83)	3.00 (1)	10.87 (13)	5000	46.05 (± 8.99)	2.00 (1)	7.52 (14)
MAB	500	43.05 (± 8.49)	0.00 (0)	9.60 (17)	500	43.45 (± 8.76)	1.00 (1)	9.77 (16)
	1000	43.20 (± 8.61)	2.00 (1)	10.82 (15)	1000	45.10 (± 8.97)	1.00 (1)	7.70 (16)
	5000	44.80 (± 8.84)	0.00 (0)	33.90 (34)	5000	46.05 (± 9.14)	2.34 (2)	7.71 (14)
NMC	500	42.20 (± 8.57)	0.00 (0)	11.27 (16)	500	43.90 (± 8.74)	0.00 (0)	9.14 (16)
	1000	42.35 (± 8.51)	0.00 (0)	10.43 (17)	1000	45.75 (± 8.99)	2.34 (2)	7.60 (15)
	5000	44.40 (± 8.69)	3.00 (1)	8.71 (16)	5000	45.85 (± 8.80)	1.30 (3)	7.94 (14)
NTBEA	500	43.05 (± 8.53)	2.00 (1)	11.81 (14)	500	42.45 (± 8.72)	0.00 (0)	10.31 (17)
	1000	43.30 (± 8.54)	0.00 (0)	9.89 (16)	1000	44.60 (± 8.79)	0.00 (0)	7.78 (17)
	5000	45.20 (± 8.69)	8.00 (1)	8.55 (15)	5000	46.30 (± 8.86)	2.00 (1)	6.27 (16)
RND	500	43.10 (± 8.48)	1.00 (1)	12.56 (13)	500	43.35 (± 8.81)	0.00 (0)	9.83 (16)
	1000	44.75 (± 8.73)	2.00 (1)	9.38 (14)	1000	45.90 (± 8.91)	1.33 (3)	7.88 (14)
	5000	45.15 (± 8.89)	2.84 (2)	9.77 (13)	5000	45.60 (± 9.02)	2.96 (3)	9.32 (13)

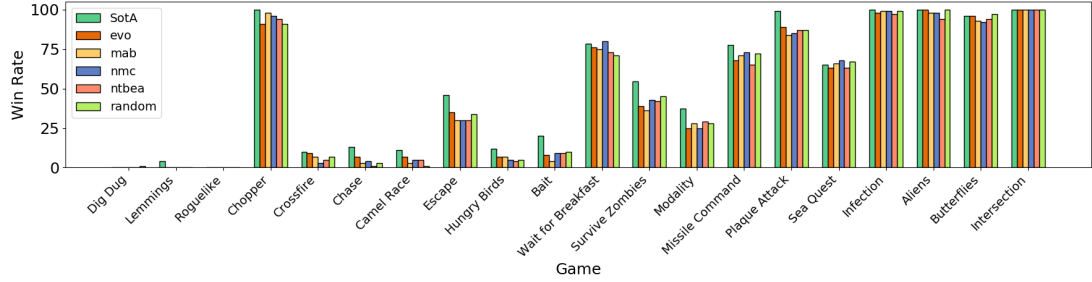


Figure 6.9: Win rate of all tuners (using RHEA configuration 10-15-1000), compared against RHEA state-of-the-art.

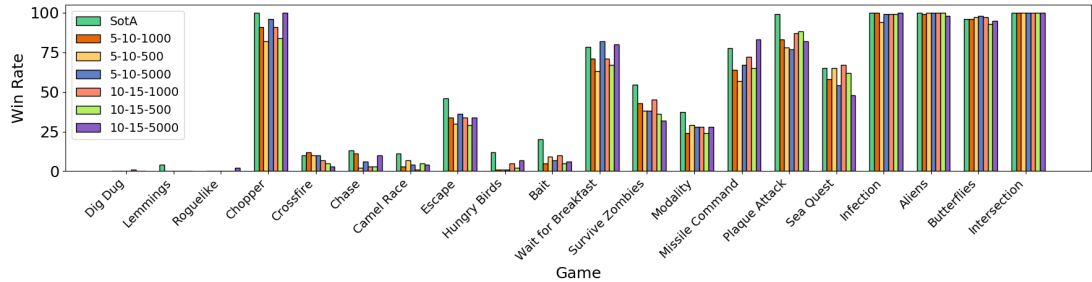


Figure 6.10: Win rate of RND tuner, with all RHEA configurations, compared against RHEA state-of-the-art.

to perform very well. This is thought to be largely due to the nature of the game, where long rollouts are not needed, but more accurate statistics over several generations are beneficial instead; the tuning helps further adapt to the highly dynamic nature of the game. Opposite to this, the larger RHEA configuration (10-15) is able to beat SotA results in “Seaquest” and “Butterflies” in 4 agent variations each. These two games generally benefit from longer rollouts as they feature delayed rewards (“Seaquest”) and an increasingly sparser environment (“Butterflies”). Both of these games are similarly very dynamic and featuring a variety of rewards and changes to the agent’s environment which require the high adaptability offered by online tuning.

We can also observe a difference in performance when the budget is varied. Win rates generally increase with higher budgets, with particular improvement observed in the games “Wait for Breakfast” and “Missile Command” when the budget is set to 5000 FM calls. These games require different skills, but both show

Table 6.4: Tuples considered best most times and Shannon entropy $H(x)$ for all tuners, one row per game, with RHEA configuration 5-10-1000; showing only the genetic operator 1-tuples. Parameter index and value index correspond to order in Table 6.2 and game index corresponds to list in Section 6.2.2. Majority agreement across tuners highlighted for 1-tuples.

G	GA		MAB		NMC		NTBEA		RND	
	5-tuple : $H(x)$	1-tuple	5-tuple : $H(x)$	1-tuple	5-tuple : $H(x)$	1-tuple	5-tuple : $H(x)$	1-tuple	5-tuple : $H(x)$	1-tuple
0	(0, 0, 0, 0, 1) : 5.87	1	(1, 0, 0, 0, 1) : 5.37	1	(0, 2, 1, 3, 1) : 4.41	2	(2, 2, 2, 4, 1) : 4.81	1	(0, 0, 1, 1, 1) : 4.92	2
1	(0, 2, 1, 1, 0) : 6.35	1	(1, 2, 2, 3, 0) : 5.24	0	(0, 1, 0, 3, 1) : 4.44	2	(1, 1, 1, 3, 1) : 5.42	1	(1, 1, 1, 3, 0) : 4.68	1
2	(1, 0, 1, 0, 0) : 5.27	1	(0, 0, 2, 3, 1) : 5.25	1	(2, 0, 0, 4, 0) : 4.25	2	(0, 2, 0, 5, 0) : 5.32	1	(0, 2, 1, 1, 0) : 5.11	1
3	(1, 2, 2, 1, 0) : 5.99	0	(0, 1, 0, 0, 0) : 5.09	0	(2, 2, 1, 5, 1) : 4.21	2	(2, 1, 2, 0, 1) : 5.46	2	(0, 2, 0, 3, 1) : 4.32	0
4	(0, 2, 2, 3, 1) : 6.06	1	(0, 1, 2, 0, 1) : 4.98	1	(1, 1, 1, 3, 1) : 4.28	1	(1, 2, 2, 0, 1) : 5.28	1	(0, 2, 2, 3, 1) : 4.90	1
5	(1, 1, 2, 3, 1) : 6.05	0	(0, 2, 2, 0, 1) : 4.68	1	(1, 1, 1, 4, 0) : 3.73	1	(0, 1, 1, 0, 0) : 5.69	1	(1, 2, 1, 0, 0) : 5.57	2
6	(1, 0, 0, 3, 1) : 5.97	1	(1, 0, 0, 1, 0) : 5.21	1	(2, 0, 0, 1, 1) : 4.61	0	(1, 0, 1, 1, 0) : 5.27	0	(2, 2, 0, 3, 1) : 5.05	1
7	(0, 0, 1, 3, 1) : 6.28	1	(0, 2, 0, 3, 1) : 4.73	1	(1, 0, 0, 3, 1) : 4.18	2	(0, 0, 2, 3, 1) : 5.72	1	(0, 1, 2, 5, 1) : 4.70	0
8	(1, 0, 0, 0, 1) : 6.07	0	(0, 0, 2, 0, 1) : 4.63	1	(1, 0, 2, 3, 1) : 4.32	1	(1, 0, 0, 4, 1) : 5.70	1	(0, 0, 0, 3, 1) : 5.03	2
9	(0, 0, 1, 3, 1) : 6.11	1	(2, 0, 1, 3, 0) : 4.97	2	(0, 0, 2, 3, 1) : 4.93	1	(0, 2, 0, 4, 1) : 5.76	1	(0, 1, 2, 3, 1) : 5.08	2
10	(0, 0, 1, 0, 0) : 5.77	0	(2, 0, 1, 3, 0) : 4.88	2	(0, 1, 2, 4, 1) : 4.20	1	(1, 2, 0, 1, 0) : 5.19	1	(0, 0, 2, 1, 1) : 4.23	2
11	(0, 1, 1, 5, 0) : 6.20	0	(1, 2, 0, 0, 1) : 5.33	1	(2, 2, 0, 3, 0) : 4.68	1	(2, 0, 2, 0, 1) : 5.51	1	(1, 1, 2, 3, 1) : 5.09	0
12	(1, 1, 2, 3, 1) : 6.24	1	(0, 1, 1, 3, 1) : 5.02	1	(0, 1, 1, 0, 1) : 4.51	2	(1, 0, 2, 3, 1) : 5.50	1	(2, 2, 0, 3, 1) : 5.07	1
13	(0, 2, 2, 3, 1) : 5.82	1	(1, 0, 1, 3, 0) : 4.59	1	(1, 0, 2, 0, 0) : 4.11	1	(0, 2, 0, 4, 0) : 5.65	1	(0, 0, 1, 5, 0) : 5.25	2
14	(0, 0, 0, 3, 0) : 6.10	1	(1, 1, 2, 0, 1) : 4.41	0	(1, 0, 0, 5, 1) : 3.81	1	(0, 1, 0, 3, 1) : 5.19	2	(0, 1, 0, 0, 1) : 5.24	2
15	(1, 1, 1, 1, 1) : 5.95	2	(2, 0, 0, 0, 1) : 5.15	0	(1, 1, 1, 4, 0) : 4.46	2	(1, 2, 0, 4, 1) : 5.27	1	(0, 2, 0, 3, 1) : 4.92	2
16	(1, 2, 1, 0, 1) : 5.42	1	(0, 2, 1, 3, 1) : 5.15	1	(2, 2, 0, 0, 0) : 4.42	1	(1, 1, 0, 4, 1) : 5.49	0	(0, 1, 1, 0, 1) : 5.46	1
17	(1, 2, 0, 0, 1) : 5.86	0	(1, 1, 2, 1, 0) : 4.09	2	(1, 1, 0, 1, 0) : 3.83	1	(0, 1, 1, 3, 1) : 5.14	0	(1, 1, 1, 3, 1) : 5.51	2
18	(0, 2, 0, 3, 1) : 5.81	1	(0, 0, 0, 5, 0) : 4.61	2	(1, 1, 1, 5, 0) : 4.21	2	(2, 1, 1, 1, 0) : 5.49	0	(2, 0, 0, 5, 0) : 4.98	1
19	(1, 2, 0, 3, 1) : 6.20	0	(1, 2, 2, 1, 1) : 4.82	1	(2, 2, 2, 3, 0) : 4.09	1	(2, 0, 2, 3, 0) : 5.33	2	(1, 1, 1, 3, 1) : 5.23	1

long-term effects of actions and require precision in decision-making. Thus the increased budget not only allows for RHEA to find better plans, but also allows the tuners to increase their accuracy in recommending good parameters.

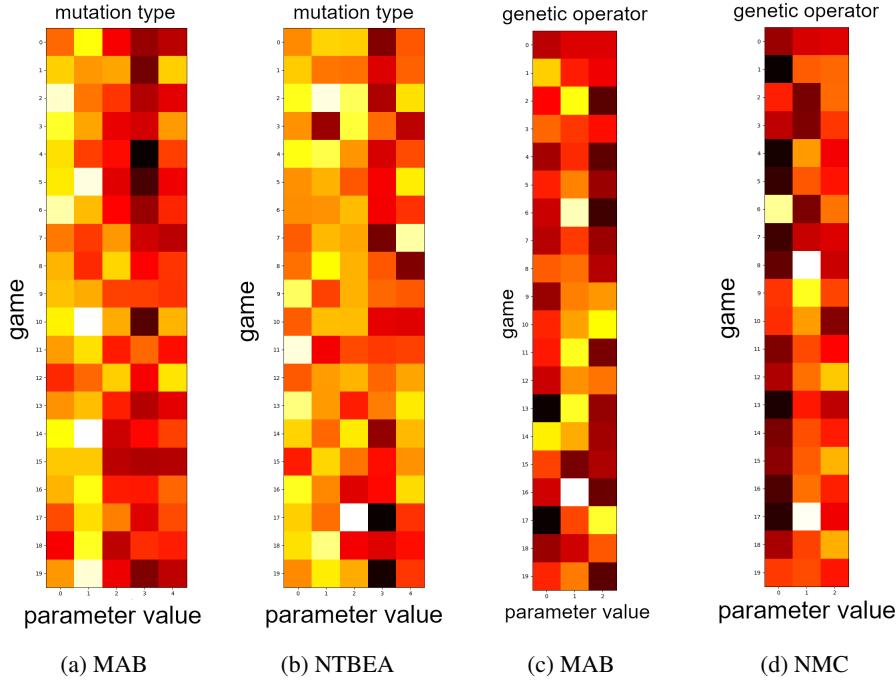


Figure 6.11: Normalised count of times considered best, per parameter value, one game per row. Parameter value index corresponds to list from Table 6.2. All use 5-10-1000 RHEA configuration.

Most interestingly, we remark first win rates from some agents in very difficult games where SotA remains at 0%: RND-10-15-1000 wins one game of “Dig Dug”, and NMC, NTBEA and RND (with RHEA configuration 10-15-5000) win 2 games of “Roguelike” each. More in-depth analysis is needed to find out how to further boost performance in these games, but we consider these clear examples of the benefits of online adaptability for tackling very difficult problems. A similar result was previously seen in (32).

We see no large differences in overall performance for the tuners, although they do appear to have different strengths. The RND tuner achieves the most better-than-SotA results across all RHEA configurations (3 games for both 10-15-1000 and 10-15-5000), and performs worse in least games as well (13 games for both 5-10-5000 and 10-15-5000). However, NTBEA obtains the highest difference to SotA in winning games (8%), as well as the lowest difference in losing games (6.27%).

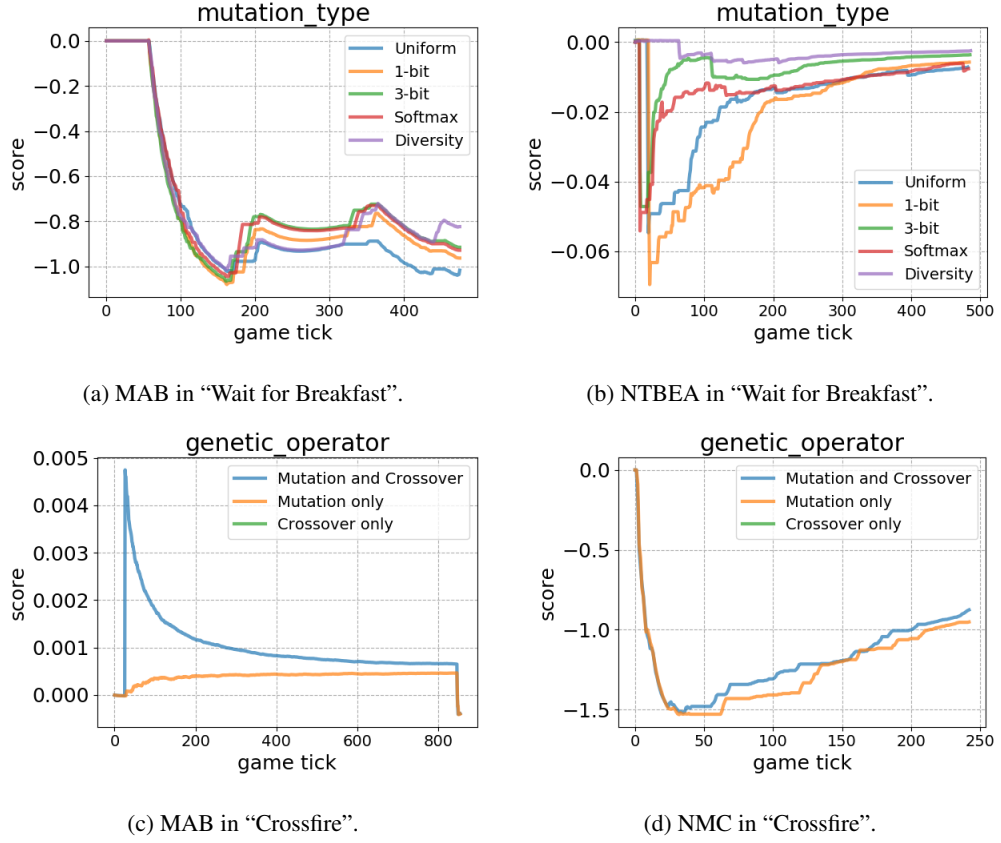


Figure 6.12: Average parameter values in winning game instances. All use 5-10-1000 RHEA configuration.

Parameter Combinations Analysis (5-tuples)

These experiments can also give further insight into what works, and what does not, in the algorithm being tuned. Table 6.4 shows the parameter combination chosen most often as the best option by each of the tuners, in all games, for RHEA configuration 5-10-1000. We further note the Shannon entropy $H(x)$ over all game ticks, which gives an indication of the tuner’s consistency. Using this measure, the NMC tuner seems most consistent in its recommendations, while the EA tuner the least; given their win rates (Table 6.3), this could suggest consistency might not be essential to success (e.g. if consistently recommending bad parameter combinations), although there is no strong evidence in this direction.

No tuner seems to agree with others on the best parameter combination for any game; this is likely due to the low number of data points each tuner is able to sample during a game, which do not allow for significant statistics. However, some partial agreements can be observed, which could indicate good combinations of parameters. In particular, we can observe a preference in several games and tuners for the (3, 1) 2-tuple (mutation type, mutation transducer), corresponding to values (Softmax, true). There is not a similar case for the crossover parameters (selection type, crossover type), where a wide variety of values are chosen in the best 5-tuples.

Individual Parameter Analysis (1-tuples)

Lastly, we can analyse each parameter individually and the values each tuner considers best. In Table 6.4, we also highlight the value chosen as the best option for the genetic operator parameter. We can observe more agreement between the different tuners on individual values of parameters, with 12 games showing 3 or more tuners choosing the same value for the genetic operator (mutation only, with two exceptions). In “Chopper”, they all show a preference for using both crossover and mutation. This game generally benefits from ample exploration of the action space, which is best obtained with both genetic operators enabled. Whereas in “Seaquest”, crossover only is considered best, leading to smaller disturbances, which could be key in a stochastic game with many possible deaths to the agent - a point strengthened by the increase in performance of several tuned agents in this game.

This is similar for most other parameters, with rank selection, 1-bit mutation and mutation transducer enabled being chosen most often; there is no agreement on which crossover type works best in any of the games. All tuners show similar and fairly high levels of consistency (given by Shannon entropy) and

variance in their recommendations.

Figure 6.11 shows that, interestingly, both MAB and NTBEA consider option 3 (Softmax) the worst value for mutation in many games, which we previously saw chosen most often in 5-tuples. Although this could be a question of credit assignment, as parameters receive associated values even if they do not impact the phenotype, we consider this an important highlight of the combinatorial problem, as opposed to choosing the best 1-tuple values. In contrast, MAB and NMC do not feature a similar ranking of options for the genetic operator, although they do both favour option 1 (mutation only) in most games.

In Figure 6.12 we take a look one level deeper into parameter choices per game tick, and how the scores given to each parameter value progresses over the course of a game. The examples included highlight the different approaches of the tuners, with shapes similar across all other parameters for the same tuners as well. MAB seems to always start with over-estimations before settling on lower scores, whereas NTBEA and NMC both show upwards trends and are able to make better use of information from previous game ticks to continue improving their parameter recommendations.

6.3 Conclusions

In this chapter we used several optimisation algorithms to automatically search the large parameter space of RHEA in order to find the most high-performing variant for a variety of environments. We extract information from the data logged by the optimisation algorithms in order to better analyse RHEA's parameters, leading to interesting insights into which parameter values are effective on which types of environments, in order to better inform future specific applications of the algorithm.

Offline. First, the N-Tuple Bandit Evolutionary Algorithm (NTBEA) was used to optimise the performance of Rolling Horizon Evolutionary Algorithm (RHEA) in 20 GVGAI games, by modifying the configuration of RHEA's 18 parameters, offline, ran for longer periods of time. The various values possible for all parameters form a large search space of 5.36×10^8 , which makes manual optimisation or exhaustive search difficult with limited compute, thus we choose to use NTBEA to attempt to improve the win rate of the agent in each of the 20 games.

As a result of the optimisation, the performance increases in several games. However, puzzles appeared to be the games where NTBEA struggled to estimate the quality of different agent configurations and the solution returned was worse than the state-of-the-art, although NTBEA's evolutionary process was run with SotA as the initial solution. The optimisation process differed in the games tested, NTBEA being able to converge in under 100 iterations in some games, while taking most of its 1500 iteration budget to find good solutions in others: this strengthens the idea that one specific method is unlikely to perform well across all games, and that games might require specialised parameter search spaces to ensure fast optimisation or even the possibility of a high-performing solution being found.

We further analysed RHEA's parameters through the evolutionary process, by looking at some 1-tuples and 2-tuples and the values explored for each. Several games with similar features in common were found to prefer similar parameter values, although exceptions do exist of game clusters shown in parameter values, but not in the traditional game features considered. This suggests that game clustering methods can be further enhanced by considering agent-based features.

To further expand on the work carried out here, we propose exploring larger and more complex search spaces, with an enhanced NTBEA which is able to handle tree structures: we have seen several parameters dependent on others and optimisation would be more sample-efficient if this was taken into account during the evolutionary process. NTBEA parameters can be better adjusted as well: decreasing the exploration constant over time could lead to better results. More enhancements can also be added into the system, as well as optimising RHEA on a larger set of games (including multi-player games and games with hidden information), with the possibility of testing approaches at optimising a generally applicable player. Moreover, other optimisers could be tested, such as Bayesian Optimisation, or other parameter selection approaches, such as (165), to observe difference in results and insights gained. Lastly, information gathered during optimisation and in-depth analysis can be used for designing hyper-parameter methods which would be able to identify game features, relate these to previously seen situations and adapt to new unknown environments.

Online. Second, we presented the use of various optimisation methods in choosing parameters for the Rolling Horizon Evolutionary Algorithm (RHEA) online (i.e. during one play-through of a game, in real-time). Five different tuners were used in this context, a standard Genetic Algorithm (GA), a Multi-Armed Bandit (MAB), Naive Monte Carlo (NMC), the N-Tuple Bandit Evolutionary Algorithm (NTBEA) and Random (RND). Several budget options, population sizes and individual length were used for RHEA. The tuned agents were tested in 20 games from the General Video Game AI framework and win rate, as well as parameter combinations and individual parameter choices were analysed in detail.

Victory rates of tuned agents were comparable to the high bar set by the RHEA state-of-the-art, surpassing it in several games. We highlight that this approach is more general and highly adaptive, as opposed to hand-picked SotA results. Generally, longer RHEA rollouts and higher budgets led to better outcomes, although the opposite led to specific improvements in the game “Crossfire”. Tuner performance was very similar, with RND and NTBEA standing out in a few cases; a deeper analysis into the different strengths and weaknesses of the tuners, and better initialisation methods, is one avenue for future work; further, could the tuner itself be tuned, or, the choice of tuner included as a parameter, for a branching hyper-parameter optimisation algorithm?

There did not appear to be a particular combination of parameters, which worked best in all games, or even the same recommendation by all tuners in a game. This emphasises the difficulty of this highly stochastic problem. However, some combinations of parameters did stand out as better than others, such as Softmax mutation, used with a mutation transducer. 1-tuple analysis suggested 1-bit mutation to be the best instead when dependencies are ignored, using mutation only as the genetic operator and keeping the mutation transducer enabled.

We also observed interesting behaviour and novel results in very difficult problems such as “Dig Dug” and “Rogue”; obtaining more data points where agents win these games could give important information on strategies for tackling such problems. Similarly, we saw a difference in highly dynamic games when discounting the tuner statistics between game ticks, which is worth further investigation and could boost performance in this class of problems.

Next chapter. In the next chapter, we take forward the insights gained through in-depth analysis from the previous two chapters and show applications of RHEA, with minimal modifications and adaptations, in 3 different environments. All of the environments are much more complex than the simple GVGAI games tested so far, posing additional challenges such as multi-player scenarios, high strategy requirements or a completely novel type of environments, tabletop games.

Chapter 7

Applications

So far we have looked at using the General Video Game AI framework to develop and study the Rolling Horizon Evolutionary Algorithm, together with several improvements and combinations with other techniques. These studies have shown RHEA to be a new state-of-the-art in general video game-playing methods. However, the games it has been tested on so far, although diverse in their features and skills required from players, remain fairly small arcade-style games, which might cause doubt in the applicability of this method in real-world problems or more complex environments.

This chapter aims to address this concern by discussing several works applying the algorithm in specific domains. All of the chosen domains detailed in this chapter (Pommerman in Section 7.1, Tribes in Section 7.2 and Tabletop Games in Section 7.3) have several features in common, all of which make them much more complex domains than previously explored: they are multi-player partially-observable environments, with large and dynamic action spaces (reaching hundreds of actions possible in a single game state). All domains further include specific other important challenges of interest to the game-playing AI research community.

See Table 7.1 for a summary of the features of the games discussed here mapping to the previously presented Table 2.1.

We use the knowledge gained from previous studies and analysis to choose favourable configurations for each domain, for a “plug in and play” approach. Each domain does add a custom heuristic and/or reward function which all the AI algorithms tested use. This is the only domain-specific adaptation employed in RHEA. We show that this general algorithm is able to handle more complex environments well, showing performance competitive with other players, both artificial and human.

This chapter includes projects I have contributed to, but where I was not the primary author.

Table 7.1: Games including a mapping to relevant features in Table 2.1. *Opp* refers to the opponent. *Solo* refers to the player being the only one left in the game. Most *Play card* action spaces are abstracted, and often involve further decisions to be made, or different phases. *Counter* win conditions involve having the highest amount of a specific game component. *Line* refers to completing a grid line (row, column or diagonal) with the player’s symbol. Other concepts (e.g. *Death*) are also abstracted and the reader is recommended to read the full game description. Games with ‘x’ in the stochasticity column have a stochastic setup and randomness affects gameplay after the game has started as well. Games with ‘*’ in the stochasticity column only have a stochastic setup, but are deterministic thereafter.

Idx	Game	Stoch.	#Players	Win	Lose	Board	Actions
0	Pommerman	*	4	Solo	Death	M/Dense	Move+Bomb
1	Tribes	x	2-4	Kill	Death	L/Dense	Dynamic
2	Tic-Tac-Toe		2	Line	Opp-Line	S/Sparse	Choose grid cell
3	Dots & Boxes		2-5	Counter	Opp-Count	M/Dense	Choose grid edge
4	Love Letter	*	2-4	Counter/Solo	Death/Opp-Counter	-	Play card
5	Uno	*	2-10	Counter	Opp-Counter	-	Play card
6	Virus!	*	2-6	Counter	Opp-Counter	-	Play card
7	Exploding Kittens	x	2-5	Solo	Death	-	Play card
8	Colt Express	x	2-6	Counter	Opp-Counter	S/Dense	Play card
9	Pandemic	x	2-4	Counter	Timeout	L/Sparse	Dynamic

7.1 RHEA in Pommerman

The work in this section was published at **AIIDE 2019**:

D. Perez-Liebana, R. D. Gaina, O. Drageset, E. Ilhan, M. Balla, and S. M. Lucas, “Analysis of Statistical Forward Planning Methods in Pommerman,” in *Proceedings of the Artificial intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 15, no. 1, 2019, pp. 66–72.

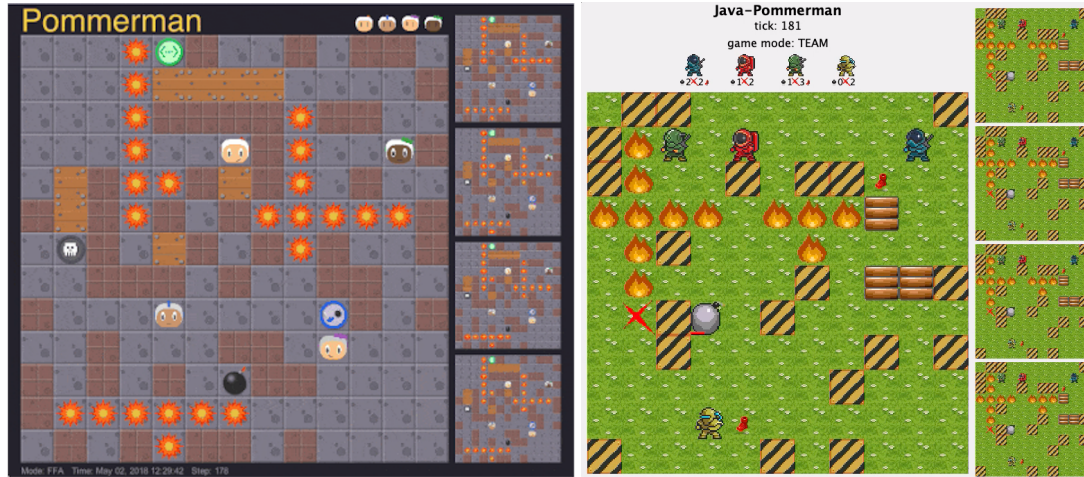


Figure 7.1: Original “Pommerman” (left) and Java version (right).

The first environment tested in this chapter is “Pommerman” (see Figure 7.1 for a visualisation). “Pommerman” is an environment originally developed in Python¹ which implements the mechanics of the game “Bomberman” (Hudson Soft, 1983) in a partially-observable setting. As such, four players each start in a corner of a randomly generated 2D 11×11 grid map containing permanent obstacles (walls), temporary obstacles (wooden boxes) and power-ups. The players can each move orthogonally (up, down, left, right), do nothing, or use a special action to place a bomb that will explode in a cross of a particular range (initially 1 square up, down, left and right) after 10 game ticks. Such bomb explosions can destroy the temporary obstacles, which may reveal one of the available power-ups: increase in bomb explosion range, increase in the number of bombs the player can place (once a bomb explodes, it goes back into the player’s inventory and can be placed again; players can only place 1 bomb at the beginning of the game), or gain the ability to kick bombs. Kicking bombs (by moving against them) causes bombs to move with constant velocity (1 square/tick) in the direction opposite to the kicker, until they encounter an obstacle, the edge of the play area, or another player. Bomb explosions can also destroy these power-ups, and kill players. This last game mechanic leads to the goal of the game: in order to win, a player has to avoid bomb explosions and be the last one alive at the end.

Several different game modes exist which offer some small variety in the games being played. The first variation was introduced as a means to force a winner in games where the players play too safely and simply avoid dying (which means the game ends in a tie). To avoid such scenarios, a possible toggle to the environment is to add shrinking level space: after game tick 800, the outer edge of the remaining playable space turns into permanent obstacles every 10 game ticks, resulting in a slow reduction of the available play area and forcing players into much closer proximity. Any players on the edge when the level shrinks is automatically killed and loses the game. This method forces more aggression between players and rewards those who are still able to avoid flames in a much tighter play area.

Additionally, the original “Pommerman” competition distinguishes two different tracks that can both make use of this environmental toggle: **free for all (FFA)** and **team (TEAM)**. A third track was tried at the 2019 competition (team radio, similar to the TEAM game mode, but allowing messages to be sent between team members) - however, this was not tested in our study and therefore will not be further elaborated on here. The **FFA** game mode has each player competing individually against 3 other players, and the last player alive is declared the winner; alternatively, the players alive at the end tie, and those that died lose. The **TEAM** game mode has players competing in teams of 2, so that both players in a team win at the end of the game if at least one member of their team survives, and both lose otherwise. If at least one member from both teams is alive at the end, all players tie.

Although the overall game rules are fairly simple, this environment has several emergent complexities:

¹<https://www.pommerman.com/>

- **Multi-player:** As opposed to all games tested so far, there are 4 players acting simultaneously in “Pommerman”. It follows that, although the environment itself is deterministic after the initial procedural generation, each play-through (and therefore each simulation performed by our statistical forward planning methods) is very noisy, depending on the behaviour of the other players. Thus opponent modelling and high adaptability to rapidly changing environments are two important challenges to overcome.
- **Partial observability:** The game can be played with full observability (therefore the agents observe the entirety of the game board at all times), or with partial observability (where vision is restricted to a square of a particular range around the player, and all other unknown parts of the board are returned as “hidden” objects in agent observations). This encourages flexible action plans that can be adapted to react to unknown situations, as well as opening a potential research area into memory and belief systems, to recognise e.g. that permanent obstacles are not going to move, thus any internal simulations could fill in this information as previously observed.
- **Delayed actions:** As detailed earlier, bombs take several game ticks to explode. For simulation-based agents, this means that the result of a “place bomb” action will not be observed until several ticks in the future, by which point it may be too late for the agent to actually react appropriately. This can be especially troublesome when trap scenarios emerge, and the agent is stuck within several obstacles with no way to escape an incoming explosion (possibly even caused by their own bomb). Longer planning horizons could aid in this issue, but given the limited thinking time (40ms available per tick) and previous observations in the study of statistical forward planning methods, short horizons allowing for more iterations are needed in highly dynamic environments such as “Pommerman”.
- **Cooperation and competition:** Lastly, the TEAM game mode specifically adds another complexity: the agents must not only learn how to kill other players and avoid dying themselves, but they must also work together with their teammate and help each other as needed in order to maximise their chances of winning. In the lack of communication, action coordination and using actions to communicate intent are the key for success, and another interesting area for further research and analysis.

Taking all of this into account, we consider this environment a very interesting testbed for AI, and very different to GVGAI games explored so far, providing interesting research directions into opponent modelling, communication and game theory. As such, we experiment with RHEA and MCTS against two simpler approaches: greedy action selection, and a rule-based system. The contributions brought by this study are not only about showing first applications of Statistical Forward Planning methods in “Pommerman” and reporting their performance, but also a deeper analysis into the games played and the behaviour of each of the methods included, which can lead to important insights and hints as to how to improve the algorithms further. Overall, MCTS and RHEA are shown to outperform the simpler approaches, although MCTS appears to be the best in several game settings. We further discuss in this section some related work, the experimental setup, results, conclusions and interesting avenues for future work.

7.1.1 Related work

The first “Pommerman” competition was organised in 2018 and focused on the FFA game mode, in which 4 agents play individually against each other. This competition saw a Finite State Machine Tree Search approach come in first, with a rule-based AI in second (166). The second competition organised received more entries, including both planning and learning approaches. Organised as part of NeurIPS 2018, this competition focused on the TEAM mode instead, in which 2 teams of 2 agents play in a partially observable environment (166). In (167), the authors describe their 2 MCTS-based agents which ranked first and third, and compare them against the 2nd place agent (another MCTS implementation with a depth $D = 2$). Osogami fixes random seeds on the first level of the tree and perform deterministic rollouts with $D = 10$.

Zhou et al. (168) compared different search techniques, such as MCTS, BFS and Flat Monte Carlo search in this game. The authors show, in the fully observable mode, that MCTS is able to beat simpler and hand-crafted solutions. The results reported in our study align with these findings, but we expand the work to partially observable settings, add RHEA to the pool of agents tested, and report a more detailed analysis on how the agents played the games.

An important body of literature on “Pommerman” is on the challenge of learning to play offline. Peng et al. (169) used continual learning to train a population of advantage-actor-critic (A2C) agents in “Pommerman”, beating all other learning agents in the 2018 Competition. A Deep Neural Network (DNN) is updated using A2C in a process that allows the agent to progressively learn new skills, such as picking items and hiding from bomb explosions. Another Deep Learning approach is proposed by (170), which uses Relevance Graphs obtained by a self-attention mechanism. This agent, enhanced with a message generation system,

analyses the relevance of other agents and items observed in the environment. The authors show that their Multi-Agent network outperforms all other tested agents. Resnick et al. (171) proposed *Backplay*, which speeds up training by starting on the terminal states of an episode. By backtracking towards the initial state, the agent improves on sample-efficiency and can learn faster using curriculum learning. (172) implemented Skynet, second-best learning agent in the NeurIPS 2018 Team Competition. Each agent of this team is a neural network trained with Proximal Policy Optimisation, reward shaping, curriculum learning and action pruning.

Finally, some relevant work on hybrid methods combines Deep Learning with MCTS. In (173), also later expanded in (174), the authors train a DNN using Asynchronous Advantage Actor-Critic (A3C) enhanced with temporal distance to goal states. They also integrate MCTS as a demonstrator for A3C, which helps reduce agent suicides during training via Imitation Learning. It is interesting to observe here that these findings align with the lower suicide rate shown by MCTS in the experiments performed for this paper (see Section 7.1.5).

7.1.2 Framework

The experiments described use our own Java implementation of the game², as opposed to the original Python Pommerman framework³. See both games depicted in Figure 7.1. Our Java implementation follows closely all of the game rules, observations, game modes and level generation, in order to allow for fair comparison with agents tested in the Python framework previously. However, our implementation is 45 times faster than the original, running at 241.4k ticks a second, compared to the original at 5.3k ticks a second⁴. This aspect is key for SFP methods, which can show better performance if able to run more simulations within the same real-time thinking budget (40ms). The re-implementation of the game further allows us to easily build more variations and logging systems on top of the original, and we use this for a better analysis of the results shown later on.

The game begins by generating a board, using a given random seed (therefore the same board can be used for multiple game runs as long as the seed is known). This process first places the agents in the corners of the board at distance $A = 2$ from the edge of the board, and $E = 2$ empty tiles around them to ensure some degree of freedom of movement in the beginning of the game. Next, obstacles are placed semi-randomly on the map (ensuring at all times that there exists a path of empty tiles connecting all 4 players), symmetrically distributed along the main diagonal, having $X = 20$ temporary obstacles and $Y = 20$ permanent obstacles. The algorithm then checks the number of inaccessible tiles on the map - if this is bigger than $Z = 4$, the whole process repeats. $W = 10$ power-ups are placed under randomly selected temporary obstacles (1 item per obstacle), with types also chosen at random. The board is then complete, with all stochastic selections made uniformly at random. A, E, X, Y, Z and W are parameters in our level generator, but the experiments presented use the default values from the original framework, as depicted here.

With the board game ready, the game environment becomes deterministic for the duration of the agents' play. The game loop iterates until the end condition for the selected game mode is met (1 agent alive for FFA, or 1 team alive for TEAM). In each iteration, all agents are asked for an action to execute, given their observation of the current game board, their avatar's properties (bomb range, number of bombs carried and ability to kick) and the alive status of the other players in the game. This observation may be restricted by the vision range (VR); a VR of n indicates that a square with size of $n * 2 + 1$ board tiles is visible to the agent, centred around the agent position; everything outside of this range is replaced with *Fog* tiles and treated as empty tiles in agent simulations. With this information, agents have 40ms to decide which of the 6 actions available to do: STOP (does nothing), move in one of four directions (UP, DOWN, LEFT, RIGHT) and BOMB (place a bomb, if possible, in the current position). The actions of all agents are then executed, respecting rules of simultaneous decision-making (e.g. no position swapping of avatars).

Besides the game state observation, the agents further receive a forward model. Even though the environment itself is deterministic, there is information hidden from players (e.g. where the power-ups were placed and which they are), which is randomised into a deterministic observation before this is passed on to the players. However, the main challenge comes from deciding what the other players might do in the situation (opponent modelling), which is unknown, and often stochastic (depending on the algorithms controlling the opponents).

²<https://github.com/GAIGResearch/java-pommerman>

³<https://www.pommerman.com/>

⁴Timings recorded on a machine using Windows 10 Intel Core i7 16GB RAM.

7.1.3 Agents

We describe next the 2 simple agents included in the experiments (ISLA and Rule-Based), and the heuristic function used to evaluate states by ISLA, RHEA and MCTS. All agents using the FM for game simulations employ a random opponent model (i.e. they assume the opponents will always return a random move out of the 6 available).

One Step Look Ahead (ISLA): This greedy algorithm exhaustively tries each of the actions available in game simulations, using the FM to advance the game state once with each action. Each of the resulting states are evaluated with the heuristic function, and the action which led to the highest valued state is chosen to be played in the game. Ties are broken uniformly at random.

Rule-Based: This agent is a re-implementation of the Simple Agent in the original “Pommerman” framework⁵. This agent uses Dijkstra’s algorithm (with depth limited to 10 so as to avoid over-timing) to calculate distances to different game objects in the board observation received. It then follows several simple steps to decide on the next best action to execute, in the specified order of priority:

1. **Escape:** If there are upcoming bomb explosions, choose the *move* action which leads the agent away from the explosions.
2. **Attack:** If adjacent to an enemy, place a *bomb*.
3. **Move to attack:** If there is an enemy within 3 steps, choose the *move* action which allows the agent to get closer to the enemy.
4. **Power up:** If there is a power-up within 2 steps, choose the *move* action which allows the agent to get closer to the power-up.
5. **Destroy:** If adjacent to a wooden box, place a *bomb*.
6. **Move to destroy:** If there is a wooden box tile within 2 steps and a bomb could be placed next to it, choose the *move* action which allows the agent to get closer the wooden box.
7. **Explore:** Lastly, choose a random *move* action towards a board position which has not been recently visited.

Custom State Heuristic: ISLA, MCTS and RHEA use the same heuristic to evaluate a state. It is calculated as the difference between the root game state (observation received from the environment at the beginning of its decision-making process) and the evaluated game state (obtained through FM simulations), and is based on a series of features:

1. Δ_t : number of teammates who died during simulations — weight $w_t = -0.1$ (0 for FFA)
2. Δ_e : number of enemies who died during simulations — weight $w_e = 0.13$ (0.17 for FFA)
3. Δ_w : number of wooden blocks that were destroyed during simulations — weight $w_w = 0.1$
4. Δ_b : increase in bomb range during simulations — weight $w_b = 0.15$
5. $k \in \{0, 1\}$: ability to kick bombs (0 = can kick, 1 = can not kick) — weight $w_k = 0.15$

,
The state value is the weighted-sum $\sum_i \Delta_i \times w_i$. The weights were manually chosen by observing agent performance prior to the experiments.

⁵https://github.com/MultiAgentLearning/playground/blob/master/pommerman/agents/simple_agent.py

Table 7.2: Experimental Setup. $All \equiv VR \in \{1, 2, 4, \infty\}$. Each set up is repeated 200 times (10×20 fixed levels). There are 32 different configurations, totalling 6400 games played.

Game mode: FFA	
VR	Agents
∞	RHEA vs 1SLA vs 1SLA vs 1SLA
	RHEA vs Rule-Based vs Rule-Based vs Rule-Based
	MCTS vs 1SLA vs 1SLA vs 1SLA
	MCTS vs Rule-Based vs Rule-Based vs Rule-Based
<i>All</i>	1SLA vs Rule-Based vs RHEA vs MCTS
	RHEA vs MCTS vs RHEA vs MCTS
Game mode: TEAM	
VR	Agents
<i>All</i>	RHEA \times 2 vs 1SLA \times 2
	RHEA \times 2 vs Rule-Based \times 2
	MCTS \times 2 vs 1SLA \times 2
	MCTS \times 2 vs Rule-Based \times 2
	RHEA \times 2 vs MCTS \times 2

7.1.4 Experimental Setup

We define a level as a game with a fixed board. We generated 20 fixed levels with 20 different random seeds, sampled uniformly at random within the range $[0, 100000]$. All experiments described in the rest of this section play each of these 20 levels 10 times, hence 200 plays per configuration. We test 2 different game modes, Free For All (FFA) and TEAM, in 4 different observability settings: vision ranges $VR \in \{1, 2, 4\}$, or fully observable (denoted here as ∞).

1SLA, Rule-Based, RHEA and MCTS were used in the tests. No communication is allowed between agents. For RHEA and MCTS, their rollout depths (L and D) are set to 12 moves. RHEA and MCTS use a budget of 200 iterations per game tick to compute actions, with a uniform random opponent model and the same custom state heuristic (described in the previous section) for evaluating states found at the end of action sequences. MCTS uses $K = \sqrt{2}$ and RHEA evolves a single individual ($N = 1$). New individuals are created every iteration via mutation (rate 0.5), keeping the best individual and a shift buffer is used.

Given that “Pommerman” is a 4-player game, the number of combinations of agents and modes is prohibitively high, thus we made a selection of the most interesting settings for our tests. Table 7.2 shows all configurations tested. For each game mode, first we aim to confirm our initial hypothesis that RHEA and MCTS have a higher performance than the other two simpler methods. Afterwards, we try to determine which of these two algorithms achieves better results in direct confrontation. In order to account for possible biases due to symmetry along the main diagonal, we tested RHEA versus MCTS ($VR = \{1, 2, 4, \infty\}$) with swapped positions. These tests showed that there is no relevant difference on the performance of the teams after the position exchange, thus swapped experiments were excluded from this report. All experiments were run on IBM System X iDataPlex dx360 M3 server nodes, each with an Intel Xeon E5645 processor and a maximum of 2GB of RAM of JVM Heap Memory.

7.1.5 Results

Overall, the results indicate that RHEA and MCTS both outperform the simpler methods tested, 1SLA and Rule-Based. This section presents and discusses results in the two game modes tested, FFA and TEAM⁶.

FFA

In FFA games with full observability (first section of Table 7.3), it is interesting to observe that, while the difference in win rate against the Rule-Based AI is quite small (rows 1 and 3) at 46.5% for MCTS and 33.0% for RHEA, MCTS tends to end more of its non-winning games in ties rather than losses, as opposed to RHEA. This suggests RHEA to adopt more aggressive or risky strategies, whereas MCTS plays safer and often avoids dying until the end of the game. This is corroborated by the average number of bombs these agents lay (RHEA uses about 30 more per game, see Fig. 7.2).

⁶Full data and plots are available here: <https://github.com/GAIGResearch/java-pommerman/tree/master/data/>

Table 7.3: FFA win rate (W), ties (T) and losses (L). 1st column indicates vision range $\in \{1, 2, 4, \infty\}$. Names in italics represent results averaged across players of the same type.

VR	Agents	% Wins	% Ties	% Losses
∞	MCTS	46.50 (4.0)	42.00 (3.0)	11.50 (2.0)
	<i>Rule-Based</i>	3.00 (1.0)	16.50 (3.0)	80.50 (3.0)
∞	MCTS	91.50 (2.0)	5.00 (2.0)	3.50 (1.0)
	<i>1SLA</i>	1.00 (0.3)	2.00 (1.0)	97.00 (1.0)
∞	RHEA	33.00 (3.0)	22.00 (3.0)	45.00 (4.0)
	<i>Rule-Based</i>	12.50 (2.3)	12.67 (2.3)	74.83 (3.0)
∞	RHEA	65.50 (3.0)	1.00 (1.0)	33.50 (3.0)
	<i>1SLA</i>	11.17 (2.3)	0.33 (0.0)	88.50 (2.3)
1	RHEA	20.50 (3.0)	5.00 (2.0)	74.50 (3.0)
	<i>1SLA</i>	2.50 (1.0)	0.50 (0.0)	97.00 (1.0)
	MCTS	67.50 (3.0)	7.50 (2.0)	25.00 (3.0)
	<i>Rule-Based</i>	1.50 (1.0)	3.00 (1.0)	95.50 (1.0)
2	RHEA	21.00 (3.0)	43.00 (4.0)	36.00 (3.0)
	<i>1SLA</i>	0.00 (0.0)	3.50 (1.0)	96.50 (1.0)
	MCTS	18.00 (3.0)	54.00 (4.0)	28.00 (3.0)
	<i>Rule-Based</i>	4.50 (1.0)	23.00 (3.0)	72.50 (3.0)
4	RHEA	19.00 (3.0)	57.00 (4.0)	24.00 (3.0)
	<i>1SLA</i>	0.00 (0.0)	2.00 (1.0)	98.00 (1.0)
	MCTS	16.50 (3.0)	59.00 (3.0)	24.50 (3.0)
	<i>Rule-Based</i>	3.00 (1.0)	17.00 (3.0)	80.00 (3.0)
∞	RHEA	13.00 (2.0)	51.50 (4.0)	35.50 (3.0)
	<i>1SLA</i>	0.00 (0.0)	3.50 (1.0)	96.50 (1.0)
	MCTS	21.00 (3.0)	61.50 (3.0)	17.50 (3.0)
	<i>Rule-Based</i>	1.50 (1.0)	32.00 (3.0)	66.50 (3.0)
1	<i>RHEA</i>	8.50 (2.0)	6.00 (2.0)	85.50 (2.5)
	<i>MCTS</i>	19.50 (3.0)	40.50 (3.0)	40.00 (3.5)
2	<i>RHEA</i>	8.00 (2.0)	38.50 (3.5)	53.50 (3.5)
	<i>MCTS</i>	5.50 (1.5)	56.50 (3.5)	38.00 (3.0)
4	<i>RHEA</i>	1.25 (1.0)	67.00 (3.0)	31.75 (3.0)
	<i>MCTS</i>	2.00 (0.5)	74.25 (3.0)	23.75 (3.0)
∞	<i>RHEA</i>	1.75 (0.5)	73.25 (3.0)	25.00 (3.0)
	<i>MCTS</i>	1.00 (0.5)	83.75 (2.5)	15.25 (2.5)

It is also worth taking into account the fact that the number of deaths by suicide in RHEA is higher than MCTS, as shown in Figure 7.2 (left). We refer to suicides to those cases in which an agent is killed by its own bomb. Note that in some cases this also includes chain reactions from other bombs. Suicides in “Pommerman” are seen as one of its most challenging aspects (174). In general, MCTS achieves the lowest suicide rate in all VR settings and modes, which helps explain the success of this method.

The win rate percentage for both RHEA and MCTS increase when facing 1SLA, with very few ties in both cases. The low number of ties compared to deaths in non-winning games suggests more aggressiveness, although 1SLA is hindered by its short look-ahead in trying to avoid bomb explosions and suicides, as its horizon does not reach beyond the 10 game ticks bombs take to explode. MCTS clearly dominates 1SLA with a 91.5% win rate. When all 4 methods play against each other (second section of Table 7.3), win rate heavily shifts in favour of MCTS with very short VR (1). In all VR options, 1SLA and Rule-Based keep a win rate no higher than 5%. RHEA’s performance is similar for all VR values, 10% and 21% of winning. It is interesting to observe that the number of losses is higher with low visibility (74.50%, $VR = 1$) than in the other cases, where ties happen more often.

Finally, MCTS achieves a higher win rate when playing only RHEA in the shorter VR option (1; see

Table 7.4: TEAM win rate (W), ties (T) and losses (L). 1st column indicates vision range (VR). Results include 2 agents of the same type on a team and average across them. The row agent team plays against the column opponent team.

VR	Agents	% Wins	% Ties	% Losses
Opponent: 1SLA				
1	<i>RHEA</i>	76.50 (3.0)	1.50 (1.0)	22.00 (3.0)
	<i>MCTS</i>	97.00 (1.0)	1.00 (1.0)	2.00 (1.0)
2	<i>RHEA</i>	88.50 (2.0)	3.50 (1.0)	8.00 (2.0)
	<i>MCTS</i>	95.00 (2.0)	2.00 (1.0)	3.00 (1.0)
4	<i>RHEA</i>	90.50 (2.0)	2.50 (1.0)	7.00 (2.0)
	<i>MCTS</i>	97.00 (1.0)	2.50 (1.0)	0.50 (0.0)
∞	<i>RHEA</i>	81.00 (3.0)	0.50 (0.0)	18.50 (3.0)
	<i>MCTS</i>	98.50 (1.0)	1.50 (1.0)	0.00 (0.0)
Opponent: Rule-Based				
1	<i>RHEA</i>	76.50 (3.0)	3.00 (1.0)	20.50 (3.0)
	<i>MCTS</i>	68.50 (3.0)	19.50 (3.0)	12.00 (2.0)
2	<i>RHEA</i>	45.00 (4.0)	27.00 (3.0)	28.00 (3.0)
	<i>MCTS</i>	74.00 (3.0)	22.50 (3.0)	3.50 (1.0)
4	<i>RHEA</i>	55.00 (4.0)	22.50 (3.0)	22.50 (3.0)
	<i>MCTS</i>	70.00 (3.0)	28.00 (3.0)	2.00 (1.0)
∞	<i>RHEA</i>	40.50 (3.0)	23.50 (3.0)	36.00 (3.0)
	<i>MCTS</i>	73.00 (3.0)	23.00 (3.0)	4.00 (1.0)
Opponent: RHEA				
1	<i>MCTS</i>	68.50 (3.0)	14.00 (2.0)	17.50 (3.0)
2	<i>MCTS</i>	22.50 (3.0)	59.00 (3.0)	18.50 (3.0)
4	<i>MCTS</i>	7.50 (2.0)	85.50 (2.0)	7.00 (2.0)
∞	<i>MCTS</i>	9.00 (2.0)	84.00 (3.0)	7.00 (2.0)

third section of Table 7.3), but very similar to RHEA in the other vision ranges. RHEA, with VR= 1, loses 85.5% of games and ties rarely, a trend that changes for the other values of VR where ties become more frequent than losses.

TEAM

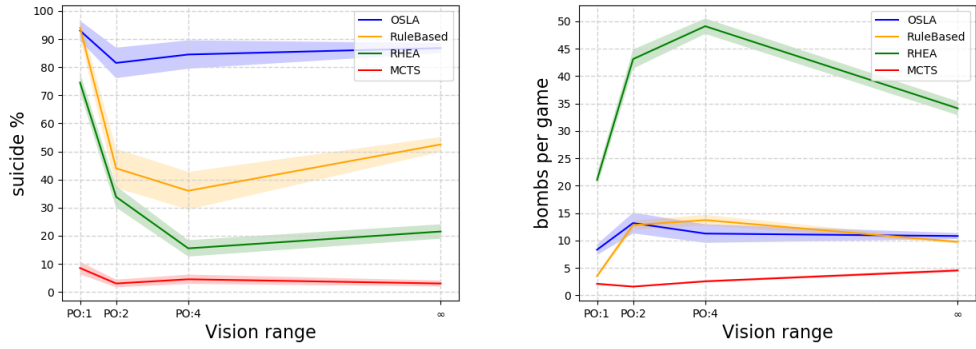
In team games, we can observe a similar performance of the algorithms with interesting differences. When playing against 1SLA (top of Table 7.4), both RHEA and MCTS achieve high victory rates and low ties, as seen in FFA, with MCTS close to 100% win rate in all VR options. RHEA appears to perform better with high VR (and still wins fewer games than MCTS). RHEA appears to perform better with low VR with the exception of VR = 1, highlighting the algorithm’s problems with restricted PO.

The average MCTS win rate when playing against the Rule-Based AI is higher than previously observed in FFA games, around 70% vs 46.5%. This is probably due to having two strong agents on the same team. The performance of MCTS is kept fairly consistent regardless of VR (second section in Table 7.4). When MCTS plays RHEA, the former algorithm dominates when VR= 1, but they achieve a similar performance with higher visibility, increasing the number of matches finished in ties.

This doesn’t mean, however, that playing style does not vary when VR is changed. Figure 7.3b shows heatmaps of bomb locations by MCTS for all VR options. As can be observed, low observability leads to significantly less and very specific bomb placements. When VR = 1 there are less bombs placed than with higher VR values, but they are more localised around the starting position. In higher visibility, the bombs are more spread out around the level. However, making the game fully observable encourages the agent to explore more and scatter bombs across the entire map. It seems clear that the presence of PO hinders the capability of the agents to use many bombs.

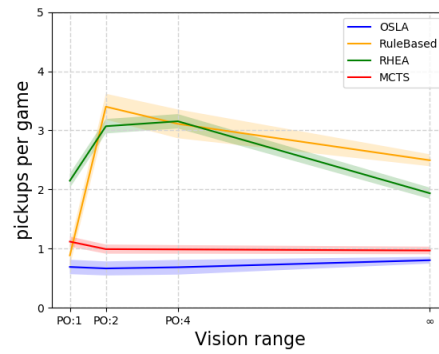
RHEA’s performance is again low with reduced vision range, but similar to MCTS with vision range 4 (the default option in the “Pommernan” competition). In terms of bomb placement, however, one can see a clear difference with MCTS. Figure 7.3a shows a difference between RHEA and MCTS, especially in intermediate VR values, showing a higher number of bombs being dropped by the former (which agrees with the observations seen in Figure 7.2. In the case of RHEA, bombs are more concentrated around the starting position and around the edges of the board than MCTS, where we see a more even spread in the starting corner and towards the centre of the map. This clearly shows that both SFP algorithms behave (i.e.

explore the search space) differently. The large amount of bombs placed by RHEA may also explain why it tends to suicide more often.



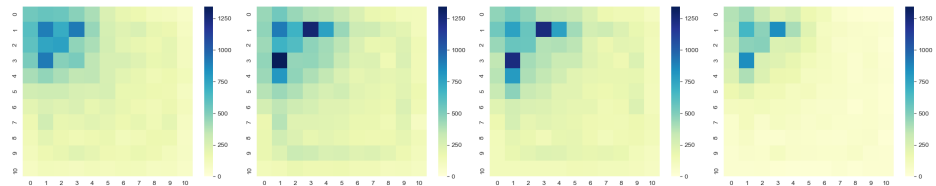
(a) Percentage of deaths caused by the own agent bombs.

(b) Number of bombs placed per game.

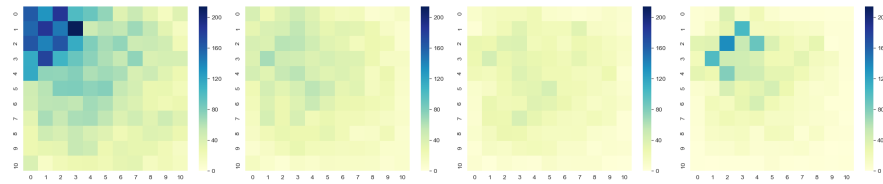


(c) Number of pick-ups collected.

Figure 7.2: Events recorded during the FFA games played (results for TEAM are very similar). All charts show values for $VR = \{1, 2, 4, \infty\}$. Shaded area shows the standard error of the measure.



(a) RHEA.



(b) MCTS.

Figure 7.3: Bomb placement by RHEA and MCTS in TEAM mode. From left to right: $VR = \infty$; $VR = 4$; $VR = 2$; $VR = 1$. Agent starts in the top left corner.

7.2 RHEA in Tribes

The work in this section was published at AIIDE 2020:

D. Perez-Liebana, Y.-J. Hsu, S. Emmanouilidis, B. Khaleque, and R. D. Gaina, “Tribes: A New Turn-Based Strategy Game for AI,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 252–258.



Figure 7.4: “Tribes” (left) and The Battle of Polytopia (right).

The second environment explored in this chapter is “Tribes”, looking at the strategy games category. “Tribes” is a clone of the popular award-winning game *The Battle of Polytopia* (175), which can be seen as a simplified version of Sid Meier’s *Civilization* (176). Both the original game and the “Tribes” framework are depicted in Figure 7.4. “Tribes” is a multi-player, multi-agent, stochastic, partially observable, highly strategic environment. Play happens on a randomly generated level (2D grid of $N \times N$ tiles), initially covered by fog of war except for the spawning locations of each player. Each tile has one terrain type associated (plain, mountain, shallow or deep water) and one type of resource, if any present (forest, cattle, food, crops, ore, ruins, fish and whales). There are two special types of terrains as well: villages (which are neutral and may be conquered by the players and converted into cities), and cities (which are owned by a player, may be conquered by other players, may level up and are able to spawn units). Only one unit may occupy a tile at any time (no overlapping is allowed). The tiles within a city’s border (a square of 5×5 tiles initially, with the city in its centre) are considered to belong to that city (and therefore the tribe owning the city). Tiles outside all city borders are considered neutral.

The **board generator** is a configurable rule-based system, which takes into account the tribes taking place in the game in order to generate their unique biomes around their starting locations, with different terrains or resources to fit their starting technologies. The generation process begins by distributing land and water tiles in the level; capital cities are then placed in land tiles, as far as possible from each other. All other terrain types, neutral villages and resources are added in last. The “Tribes” level generator is a Java port of the generator developed by Pierre Schrodinger⁷.

As the name might suggest, tribes battle during the game for control of the board. Each tribe mainly consists of several controllable units and cities. There are several types of **units** with different properties (attack power, defence, attack range and health points). They can be melee (warrior, rider, defender, swordsman and knight) or ranged (archer and catapult). All can move into a port in the water in order to embark onto a boat (a special type of unit that can move in water tiles; this can be upgraded to a ship, and then a battleship, for increased movement, defence and attack power). All units can become a *veteran* by defeating 3 other units, which increases their health points. All units can move on land tiles (and water tiles if on a boat). All units (except for the mind bender) can also attack enemy units. The mind bender can heal friendly units, or convert enemy units to their tribe. In a single turn, units may execute one or more of their possible actions, depending on the unit type. All units (except for the mind bender) conquer neutral villages or enemy cities by starting their turn in the corresponding tile.

Cities are the source of the game’s currency and produce *stars*, awarded to the player at the beginning of each turn. Stars may be spent on spawning new units in a city (which are not able to execute actions until

⁷<https://github.com/QuasiStellar/Polytopia-Map-Generator>

the player’s next turn), collecting resources within a city’s borders, improving owned tiles with buildings, or researching technologies. Collecting resources and constructing buildings adds population to the city they belong to, and reaching $L + 1$ population (where L is the city’s level, initially 1) results in the city levelling up to produce more stars, and offering a choice between 2 bonuses such as extra star production, resources, city border growth etc. After a city reaches level 5, a *super unit* or *giant* may be spawned, or a park built in the city which awards extra points. Buildings may offer additional bonuses depending on neighbouring tiles. Buildings may also be destroyed to free up a tile and retrieve some of its star cost back.

Each tribe has access to a **technology tree**. Researching a new technology is available if the player has enough stars and if its parent technology has been researched already, and unlocks new abilities for the tribe, such as constructing new buildings, collecting more resources, movement in deep water or mountain tiles, or building roads (which increase unit movement and may add population if cities become connected through roads).

Gameplay: Tribes each start in a location designated as a potential starting place on the given board, with different unit types and different technology already researched. Each player (minimum 2, maximum 4 currently supported) controls one of the tribes (Xin Xi, Imperius, Bardur and Oumaji), starting with 1 capital city and 1 unit (depending on the tribe selected: warrior for Xin Xi, Imperius and Bardur; and rider for Oumaji) placed on the board. Additionally, each tribe also begins with a technology already researched (Climbing, Organisation, Hunting and Riding, respectively). During the game, the players take turns to choose actions. During a player’s turn, any number of actions may be taken; a turn ends when no more actions are available, or the player chooses the `End Turn` action; alternatively, turns may be time-constrained in the “Tribes” framework.

The framework supports two different **game modes**, which enforce different end of game conditions. In the *Capitals* game mode, a player wins once they conquer all capital cities on the board. In the *Score* game mode, the game ends after 30 rounds (where 1 round consists of 1 turn played by each player), and the player with the highest number of points wins. Points can be gained through collecting resources, exploring the map (and revealing information hidden by fog of war) or capturing neutral villages. A player loses in both cases if their capital is captured.

Summary: “Tribes” is overall a complex game, with many aspects to it, including technology research, economy management, build orders and combat. Full details of the game rules and all game parameters (unit properties, technologies etc.) are available in the fandom page⁸. The framework code and documentation is available on GitHub⁹. The rest of this section gives an overview of strategy games, the place “Tribes” has within this domain and the research opportunities it brings, as well as showing the performance of several game-playing AI methods, including RHEA, in this environment.

7.2.1 Strategy Games

Some of the first applications of AI in turn-based strategy games were seen in 2004, by Arnold et al. (177) in the game “Freeciv”¹⁰, an open-source free game inspired by Sid Meier’s “Civilization” series and including most of the complexities of the original game in the interactions between potentially hundreds of players. The game comes with in-built AI, which uses a set of rules and a goal priority system to decide its actions. However, the intricacy of the problem and the many sub-problems involved (e.g. troop movement, battles, resource gathering and management, city development, technology advancements) led researchers to only tackle some of the aspects involved.

(177) tackled the initial placement of settlements for the player’s civilisation by using a Genetic Algorithm (GA) to adjust the parameters of the city placement algorithm provided with the game. The authors found that achieving good performance even in just this part of the problem was very difficult, although seemingly good policies are general enough to be applicable on new maps. (178) discussed instead the problem of city development in “Freeciv”, and proposed using an online GA to evolve a city development strategy; here, they use a 2-layer genome, where the top layer considers the cities to focus on, and the bottom layer considers development factors (e.g. happiness, food supply) for each city. Later, (179) applied Q-Learning to “Civilisation IV” for the city placement problem, to replace the previously explored rule-based system and create a more dynamic and adaptive approach. Their results show their method is able to outperform the rule-based system in short games and on small maps, but begins to struggle when the complexity increases. With “Tribes” we aim to make it more attainable to create an AI player able to fully undertake complex decision-making in large dynamic and multi-faceted environments.

⁸polytopia.fandom.com/wiki/The_Battle_of_Polytopia_Wikia

⁹<https://github.com/GAIGResearch/Tribes>

¹⁰The Freeciv Project, 1996-2020, <http://www.freeciv.org/>

One critical aspect of “Tribes” is the challenge of multiple actions executed per turn, and planning turns accordingly to maximise results. (76) address this problem in “Hero Academy” (180), where the player controls several agents with different actions available, with 5 action points per turn each. The authors introduce Online Evolutionary Planning (OEP), which is able to outperform tree search methods due to managing the large turn planning problem much better (155). Later, (81) propose an alternative which combines evolutionary algorithms and tree search (EvoMCTS) in order to take advantage of the benefits of both methods to outperform OEP. Both EvoMCTS and OEP were further applied to other strategic games with moderate success, although large action spaces prove challenging to both approaches (38).

This problem of very large action spaces in turn-based strategy games was recently highlighted in the Bot Bowl framework (181), which presents an implementation of the board game “Blood Bowl” (182). In this game, players control an entire football team, where each unit can execute several actions, leading to a very large turn-wise branching factor. In the first competition using this framework in 2019, *Grod-Bot*, the rule-based baseline agent, outperformed all other submissions (two Actor Critic methods and an evolutionary-tuned *GrodBot*), showing the difficulty of the problem. “Tribes” has a large but smaller action space, but adding several complexities to the decision-making problem beyond troop actions.

Similar tasks can be found in real-time strategy games. (183) review challenges presented in these games, of which “Starcraft II” has recently seen great success with the development of *AlphaStar* (184), a deep-learning agent able to play the game with high proficiency. The general and practical applicability of such methods remains a question, however, and the microRTS competition attempts to promote research into general methods able to handle different scenarios and maps within a simplified version of a real-time strategy game (185). This is similar to the approach taken in “Tribes”, where the AI is challenged on procedurally generated maps, with more variation in units available, their interactions, a very high variance in action space per step and the addition of technology research and the economic system.

To sum up, there have been previous frameworks focused around strategy games, including access to a forward model to enable the instant powerful play of statistical forward planning methods (Hero-Academy (76), Bot Bowl (181), Santiago Ontañón’s μ RTS (185)). However, to the knowledge of the authors, there was no other framework that captures all of the complexities of real-time or turn-based strategy games, while also providing forward model access.

“Tribes” aims to fill in this gap. The framework facilitates research on procedural content generation for levels and automatic game balancing as well, as it exposes a very large set of parameters that adjust the behaviour of the game. This is an interesting research direction that would open the door to creating more interesting levels and game variants.

7.2.2 The Framework

“Tribes” is implemented in Java and includes possibilities for full customisation of game parameters, as well as additional features to those already mentioned. Games can be run either as fully observable, or partially observable, as in the original game. If partially observable, hidden tiles are indicated in player observations as “fog” terrain types and treated as plain terrain types in game simulations. Fog tiles are revealed when a friendly unit moves within range of it. We define vision range as the maximum Chebyshev distance d from a unit that a tile needs to be within, so as to be observed by that unit. Vision range is set to 1 by default, but is increased to 2 when a unit is on a mountain tile.

The players are repeatedly asked for actions by the engine on their turn. Each time they have to return an action, they receive a game state observation (copied instance of the real game state), with reduced information about enemy cities and units regardless of observability settings: the number of kills and city of origin for units, and the population, star production and units originated there for cities, are hidden. The observation can be queried for available actions. Additionally, players can run simulations of the potential effect of actions using the forward model provided with their observation.

We note that executing actions could make new actions possible in the resulting game state (e.g. spawning new types of units, as a result of researching a technology), or may make some impossible (e.g. spending stars would make higher cost actions unavailable, or moving a unit to a tile would block access to that tile for other units). Thus the action space during a player’s turn is non-linear and may increase or decrease over time, depending on their decisions and resource management.

7.2.3 Agents

The framework code includes 7 AI players, 2 of which are very simple: *DoNothing* returns End Turn actions only, and *Random* returns one action out of all possible, chosen uniformly at random. *MCTS*, *RHEA* (with a shift buffer enabled) and the previously seen *ISLA* player (greedily selecting actions) are also included. The other 2 players are described next:

Rule-Based (RB): This player follows a series of rules to decide what to do next, depending only on the current game state (it does not consider future game states and does not use the forward model for simulations). All available actions are assigned a score between 0 and 5, the one with the highest score being chosen to play (with ties broken uniformly at random). To calculate these scores, the action type is considered first, then the action’s parameters, as follows.

Attack actions are scored based on the attack power and health points of the attacking unit, and the defence power and health points of the defending unit (therefore anticipating the result of this attack). The action is scored higher if the attack is going to succeed, and lower otherwise. This action is valued lowest if attacking strong units that are not defeated by the attack, and the resulting counter-attack would defeat the attacker instead.

Move actions are scored based on the distance to powerful or weak enemy units, villages and cities. The action is scored higher if it reduces the distance to weaker enemy units, villages or enemy cities. The action is scored lower if it brings the unit in attack range of strong enemy units.

Capturing cities and villages, *examining* ruins and *making a unit a veteran* are always given the highest possible score. These are special actions that always provide bonuses for the player, thus they are played as soon as they become available. When *levelling up*, the following bonuses are chosen at each level, in order: extra production, extra resources, city border growth, and spawning a super unit (levels 5 and up).

Upgrading units and *spawning* units actions are scored based on the type of unit, stars available and the presence of enemy units within city borders (the last of which reduces the priority of these actions). Similarly, *building* and *resource gathering* consider stars available and population gain, calculating the price-quality trade-off.

The Mind Bender’s *convert* action is scored higher for stronger enemy units and its *heal* action is relative to the number of units affected. *Researching* a technology is scored based on the tier the target technology is in, giving higher priority to those in lower tiers (therefore this player focuses on breadth of technologies, rather than depth).

Disbanding a unit and *destroying* a building are never executed. These actions are often available, but they are useful only in specific circumstances that require more careful planning than what this player is able to do.

As actions are evaluated independently, this player lacks an overall outlook on the situation and strategic planning. This player also shows a lack of unit coordination and poor resource management. However, it incorporates human knowledge and is therefore able to make good tactical decisions, and serves as a baseline for comparison with all other players.

Monte Carlo (MC): This player implements Monte Carlo search, which repeatedly executes rollouts (sequences of random actions) from the current game state to a predetermined depth. The last state reached by the rollout is evaluated with the state evaluation function detailed below. This value is associated with the first action in the random sequence. MC returns the action from the current game state that achieved the highest average value over all iterations.

State Evaluation Function RHEA, MCTS, 1SLA and MC all use the same heuristic function for evaluating game states. In “Tribes”, this function considers 7 different features (ϕ , as the difference (or change) from a state to another:

- Δ_{ϕ_1} star production increase/decrease
- Δ_{ϕ_2} number of technologies researched
- Δ_{ϕ_3} game score gained/lost
- Δ_{ϕ_4} cities conquered/lost
- Δ_{ϕ_5} units spawned/lost
- Δ_{ϕ_6} enemy units defeated
- Δ_{ϕ_7} city level increase (sum of all owned cities’ levels)

A linear combination of these features is calculated, weighted by the weight vector $W = \{5, 4, 0.1, 4, 2, 3, 2\}$ (manually adjusted to reflect the relative importance of each feature), for each player in the game. This calculation represents the progress of a player from one game state to another. We denote the value for the player evaluating the game state as v_p . The values for all enemy players are averaged as v_e , and the final value of the game state becomes $v = v_p - v_e$. This final value represents the relative progress of the player compared to the others in the game.

In practical terms, the features are calculated as the difference between the current game state and the game state reached at the end of a rollout (for MC and MCTS), at the end of an individual in RHEA, or after one action applied in ISLA.

7.2.4 Experiments and Analysis

The parameters of the AI players were first adjusted after small experiments and observation of behaviour in the game. We report here some considerations and configurations (some unsuccessful and discarded), for completeness.

Root prioritisation: MC and MCTS both select a subset of actions (unit actions, city actions or tribe actions) at random at the beginning of each iteration. The first action taken from the root is selected as normal, but only from this subset of actions. From the second action on, all actions in the game state are available to select from again. This suggested stronger playing performance, allowing the algorithms to better focus their search in circumstances with many actions possible. This can be seen as an extremely simplified version of Progressive Widening (186).

Forcing End Turn actions: The game has a large branching factor, which means the trade-off between rollout length and number of iterations (in the cases of MC, MCTS and RHEA) needs to lean towards a higher number of iterations, in order to be able to explore as many actions as possible and build some reliable statistics as to which is best. This, together with the fact that players may play as many actions as they want / are available on their turn, also means that rollouts would rarely simulate the actions of other players (beyond the player’s own turn), and even more rarely would they see a second turn of their own in their simulations (which would give a better indication as to the effect of their actions in the first turn, and therefore allow for more consistent planning that holds up better in the long term). In order to address this limitation, we considered forcing a play of the `End Turn` action every X steps (therefore imposing a limit on how many actions a player can take in a turn). However, this did not improve results and was discarded in final experiments. This limitation could prove useful in combination with other techniques, such as Progressive Widening, move ordering functions or First-play urgency (187).

MCTS simulation step removal: It has been seen in recent works (188; 81) that skipping the simulation step in MCTS and not performing any rollouts from the node added in the expansion step (evaluating this game state instead, and backing up the value of this state) can boost overall performance. This was observed in small tests, thus the MCTS agent does not use rollouts during its iterations in the final experiments described here.

Algorithm parameters: In MCTS, we set the C constant in our UCB equation to $\sqrt{2}$ (as all rewards observed are already normalised by the heuristic function). The rollout length in MC, MCTS and the individual length in RHEA is set to 20 (values of 5 and 10 were tried and consistently provided worse results). The population size in RHEA is set to 1 (therefore no selection or crossover is used; the 1 individual is simply mutated, and the best sequence is kept for the next generation). Some previous work showed that larger RHEA populations tend to provide better results in General Video Game Playing (15), however, sizes of 5 and 20 did not outperform a population size of 1 in “Tribes”. A possible explanation for this is an observed tendency by RHEA of generating invalid actions, which is likely to be caused by the crossover operator. Future work will look into adding repair operators for RHEA and a closed loop more similar to the MCTS implementation.

Action limits: The actions `Disband` and `Destroy` were removed from the list of possible actions for all players, as no AI player used them for useful strategic decisions.

Experimental setup Each AI player played 500 2-player games against each other player, 20 repetitions in the same 25 procedurally generated levels (using 25 different random seeds), all with a 11×11 board size and a 1 : 1 water/land tile ratio. As levels are not guaranteed to be balanced (starting positions may be better for some tribes than others), the 20 repetitions in a level are split in half so that players play 10 times in a match-up, and 10 times more with positions swapped. The games are played only with the *Xin_Xi* and *Imperius* tribes, and in the *Capitals* game mode, with full observability. All games are terminated after 50 rounds - if no winner is declared by round 50, the player with the most points wins instead. MC, MCTS and RHEA have a set budget of 2000 calls to the `next` function on their observations for every decision.

Results We present the results in Tables 7.5 and 7.6, with rows sorted from strongest to weakest players. A head-to-head analysis of Table 7.5 shows that both RHEA and MCTS are able to reliably win against simple users of the forward model (ISLA and MC; $> 75\%$ victories and $> 60\%$ victories, respectively). It is worth noting, however, that the rule-based agent is surprisingly strong against both of these methods, beating MCTS in 56.20% of the games and managing to rank slightly above it in overall standings as well. This showcases the ample room for improvement available within the current players included in

Table 7.5: Win rate for each row player averaged across 500 games against the column player. The values between brackets indicate standard error.

Player	RHEA	RB	MCTS	MC	ISLA	RND
RHEA	*	58.60% (2.20)	63.00% (2.16)	77.80% (1.86)	74.80% (1.94)	100.00% (0.00)
RB	41.40% (2.20)	*	56.20% (2.22)	62.40% (2.17)	70.20% (2.05)	98.80% (0.49)
MCTS	37.00% (2.16)	43.80% (2.22)	*	62.00% (2.17)	60.80% (2.18)	98.60% (0.53)
MC	22.20% (1.86)	37.60% (2.17)	38.00% (2.17)	*	54.80% (2.23)	99.00% (0.44)
ISLA	25.20% (1.94)	29.80% (2.05)	39.20% (2.18)	45.20% (2.23)	*	99.40% (0.35)
RND	0.00% (0.00)	1.20% (0.49)	1.40% (0.53)	1.00% (0.44)	0.60% (0.35)	*

Table 7.6: Statistics for all games averaged across 2500 game ends. The values between brackets indicate standard error.

Agent	Wins	Rank	Score	Techs	Cities	Production
RHEA	74.84% (1.50)	1.25 (0.03)	11610.56 (134.34)	88.95% (1.78)	2.68 (0.05)	20.68 (0.41)
RB	65.80% (1.32)	1.34 (0.03)	8076.32 (81.58)	71.14% (1.42)	2.98 (0.06)	20.29 (0.41)
MCTS	60.44% (1.21)	1.40 (0.03)	9966.55 (106.19)	85.27% (1.71)	2.23 (0.04)	16.96 (0.34)
MC	50.32% (1.01)	1.50 (0.03)	8065.96 (71.22)	82.35% (1.65)	2.23 (0.04)	14.73 (0.29)
ISLA	47.76% (0.96)	1.52 (0.03)	8927.05 (88.84)	82.31% (1.65)	2.05 (0.04)	15.17 (0.30)
RND	0.84% (0.02)	1.99 (0.04)	3708.76 (12.51)	58.54% (1.17)	0.39 (0.01)	0.01 (0.00)

the framework, as well as the strength of the heuristics employed by the rule-based player, which could be better employed within the SFP methods as well. All achieve close to 100% win rate against random, and RHEA shows as the best out of the current set of players, beating MCTS in 63% of the games.

We can see more detailed statistics of each player in Table 7.6, with data averaged over all games played by the row player, against all opponents. 4 measures are highlighted, which normally correlate with the win rate (and used in the players' heuristics): game score (final score achieved), percentage of technologies researched, the number of cities owned at the end of the game, and the star production at the end of the game. Although most measures show no particular surprises, the rule-based player stands out as an outlier here, with less technologies researched and a lower game score than most other players (excluding random): this player appears to focus most of its efforts in conquering enemy cities and increasing their cities' power, which is a strong strategy in the game mode chosen for the experiments. This indicates that the heuristics used by the more advanced players can be further improved and take into considerations more aspects of the game and more specific contexts as well. However, RHEA does still win the most even with less cities owned (2.68 compared to 2.98 of the rule-based player), although they achieve similar levels of star production, and technologies and game score in favour of RHEA. This suggests that different strategies are also effective, with a potential rock-paper-scissors effect in play, although a deeper analysis is required to analyse this phenomenon.

A last test carried out was pitting the strongest AI player (RHEA) against a human player (self-reported as moderately good at the original *The Battle of Polytopia* game). Playing once in all 25 levels used for the experiments, with tribes assigned randomly, RHEA shows its weakness against more advanced strategic planning, being clearly dominated by the human player (100% win rate for the human player). However, this test does show interesting insights into RHEA's behaviour: although it is able to make strong tactical short-term decisions (e.g. defending their own cities), it largely lacks unit coordination and shows fairly poor resource management. Taking this information forward and integrating it into more advanced heuristics could further improve the strength of this algorithm.

7.3 RHEA in Tabletop Games

The work in this section was published at EXAG 2020 and on arxiv:

R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, “TAG: a Tabletop Games Framework,” in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2020.

R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, “Design and Implementation of TAG: a Tabletop Games Framework,” *arXiv preprint arXiv:2009.12065*, 2020.

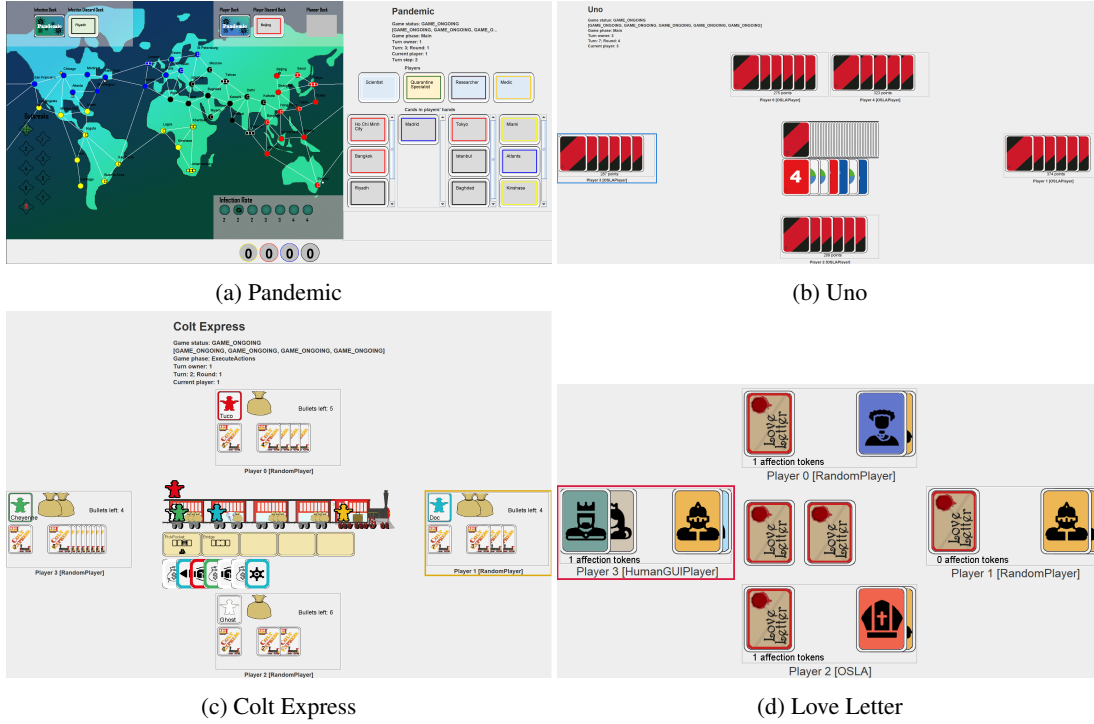


Figure 7.5: Example games in the TAG framework.

The last environment explored in this chapter moves away from video games and into the realm of tabletop games instead. In the last few decades, tabletop games have gone through a ‘Renaissance’, gaining more popularity than ever: thousands of them are published each year and welcomed by an expanding audience of gamers (189). Many of these games are typically designed with richer mechanics and rules, and less focus on chance-based elements. Modern tabletop games are very diverse and complex in general, which can provide various challenges, such as unique game state representations, partial observability on various levels, actions outside of a player’s turn, cooperation, and competition in the same game, etc. Therefore, they have a very different and complex set of mechanisms that would require a lot of effort to develop using previous approaches.

AI Research in board games has mostly been, with some exceptions, focused on traditional board games, either in isolation (*chess*, *Othello*, *Go*, etc.) or as part of general game playing (GGP) frameworks, such as GGP (190), Ludii (191) and OpenSpiel (192). While these frameworks also allow the definition of additional games, they are limited to common mechanics or require extensive development effort.

In this section, we discuss the Tabletop Games (TAG) framework, which is a collection of tabletop games, agents, and development tools meant to provide a common platform for AI research. This work is mainly motivated by three factors: i) the characteristics of modern tabletop games (multi-player, partially observable, large action spaces, competition, and collaboration, etc.) provide an interesting challenge to AI research. These modern games provide many characteristics not implemented in existing frameworks, e.g. changing player roles or varying forms of cooperative and competitive game-play; ii) TAG presents tabletop games from a GGP perspective, by providing a common API for games and playing agents; and iii) we aim to provide a framework that can incorporate different games under a common platform, making it possible for the research community to implement their own games and AI players to expand TAG’s collection. De Araujo et al. (193) highlighted the need for such a framework in a survey that described the different schemes and data structures used in the literature of digital board games.

In order to provide the necessary flexibility to support the great variety of existing tabletop games, TAG requires the user to implement games via a programming language (Java), instead of using a game description language as other general frameworks do. TAG provides many customisable components to

simplify the development of additional games not currently in the framework. It is able to handle partial observability, providing simple means of adding custom observation schemes, game state heuristics, and agent statistics, as well as supporting the development of graphical user interfaces for human play with computer players.

While we are working on extending the framework with additional game mechanics, games and agents, the current version is publicly available¹¹. The contributions of this section are twofold: firstly, we present the framework, its structure, games and AI players implemented (see Section 7.3.2). Secondly, we provide a discussion on a baseline experimentation (Section 7.3.3), aimed at illustrating insights into the games implemented, their features, and performance of vanilla AI players. In Section 7.3.4 we discuss challenges and opportunities for this framework.

7.3.1 Tabletop Games

AI research and board games have been closely related since the beginnings of the field, with game-playing agents for “Tic-Tac-Toe”, Checkers and chess (194), and the more recent breakthroughs in Go (188). One of the most well-known contests, the General Game Playing (GGP) competition (190), featured classical board games written in the Game Description Language and promoted research into generic game players. (191) later introduced the “ludemic” general game system *Ludii*, which builds upon ideas from GGP. *Ludii* defines games as structures of *ludemes*, high-level, easily understandable game concepts, which allows for concise and human-understandable game descriptions, as well as easy implementation of new games. The current main focus of the *Ludii* project is on classical and ancient board games. Kowalski et al. presented in (195) a new GGP language, called Regular Boardgames (RBG), with a similar focus as *Ludii*, but which describes games as regular expressions instead, for increased efficiency.

We consider direct code implementations to be more accessible, faster to execute and easier to work with in many cases. More similarly to our framework in this regard, *OpenSpiel* (192) provides a collection of games and algorithms written in C++ and exposed to Python. Most of their games are still traditional board games, with some exceptions, such as the inclusion of “Hanabi”. Differently, TAG shifts the development effort onto the framework, rather than the games, by making a wide range of components, rules, actions, etc. available to users. Our system allows for fast prototyping of new games and immediate extraction of insights and features of interest in each game through readily-available game and AI-facilitated analysis. We further support many research directions, from simulation-based game-playing to parameter optimisation of games and artificial players.

Kowalski et al. presented in (195) a new GGP language, called Regular Boardgames (RBG), with the objective of joining key properties such as expressiveness, efficiency, and naturalness of GGP languages, compensating certain drawbacks of the existing languages. According to authors, the RBG language allows efficient encoding and playing games with complex rules and large branching factors (such as “Amazons”, large chess variants, Go, paper soccer, etc.).

Here, we focus on more types of games, including board, card, dice and role-playing games, among others, often grouped under the *tabletop games* umbrella term. “Tabletop Simulator” (196) is an example of software facilitating implementation of tabletop games components in a physics-based simulated environment; this allows players to interact with the games as they would in the real world, but the many games implemented lack support for automatic rule execution, nor does the software facilitate AI research as targeted with TAG. However, tabletop games research has been gaining popularity in recent years. Research in game-playing agents for card games is common in competitive (poker (197) and bridge (198)) and cooperative (“Hanabi” (199)) games.

Asymmetric player roles is one feature often encountered in modern tabletop games, and these have been studied in games such as “The Resistance” (200) and “Ultimate Werewolf” (201). This is just another complexity added in modern tabletop games, yet more lead to the need for intricate strategic planning. To this extent, Monte Carlo Tree Search (MCTS) methods have been tried in “Settlers of Catan” (202) and “Risk” (203), and Rolling Horizon Evolutionary Algorithms (RHEA) in “Splendor” (161), all showing a great improvement in performance. Other games have been more recently highlighted as important challenges for AI players due to their strategic complexity (“Pandemic” (204; 205)) or very large action spaces (“Blood bowl” (181)).

Research has not only focused on game-playing AI, however. “Ticket to Ride” (206) was used as an example for employing AI players for play-testing games, characterising their features based on different play-styles and finding possible bugs or gaps in the rule-set. Further, the use of Procedural Content Generation for such games is highlighted by (207); given the rule complexities and the multitude of components in modern tabletop games, AI methods can provide a more efficient way of searching the possibility space for interesting variations.

¹¹<https://github.com/GAIGResearch/TabletopGames>

The Tabletop Games (TAG) framework introduced in this paper brings together all of these different research directions and provides a common ground for the use of AI algorithms in a variety of tabletop games, removing the effort of creating different frameworks for different purposes and simplifying the overall development process. As far as we know, TAG is the first framework that allows the development of multiple games and AI players under a common API for complex modern tabletop games.

7.3.2 The Framework

TAG was designed to capture most of the complexity that modern tabletop games provide, with a few games implemented already and more in progress.

Concepts

Our framework includes handy definitions for various concepts and components common across many tabletop games (189).

We define an **action** as an independent unit of game logic that modifies a given game state towards a specific effect (e.g. player draws a card; player moves their pawn). These actions are executed by the game players and are subject to certain **rules**: units of game logic, part of a hierarchical structure (a game flow graph). Rules dictate how a given game state is modified and control the flow through the game graph (for instance, checking the end of game conditions and the turn order). This **turn order** defines which player is due to play at each time, possibly handling player reactions forced by actions or rules. At a higher level, games can be structured in **phases**, which are time frames where specific rules apply and/or different actions are available for the players.

All tabletop games use **components** (game objects sharing certain properties), whose state is modified by actions and rules during the game. TAG includes several predefined components to ease the development of new games:

- **Token**: A game piece of a particular type, usually with a physical position associated with it.
- **Die**: A die has a number of sides N associated with it and can be rolled to obtain a value between 1 and N (inclusive).
- **Card**: A card usually has text, images or numbers associated with any of its 2 sides, and is the most common type of component used in decks.
- **Counter**: An abstract concept used to keep track of a particular variable numerical value; usually represented on a board with tokens used to mark the current value, but recognised as a separate object in this framework. It has a minimum, maximum and current value associated with it, where the current value can vary between the minimum (inclusive) and maximum (inclusive).
- **Graph board**: A graph representation for a board, as a collection of several board nodes connected between each other.
- **Board node**: A node in a graph board which keeps track of its neighbours (or connections) in the board.
- **Grid board**: A 2D grid representation of a board, with a width and height associated with it. It can hold elements of any type.

Components can also be grouped into collections: an **area** groups components in a map structure in order to provide access to them using their unique IDs, while a **deck** is an ordered collection with specific interactions available (e.g. shuffle, draw, etc.). Both areas and decks are considered components themselves.

Structure

The TAG framework brings together all of the concepts and components described previously and allows quick implementation and prototyping of new games. Its structure consists of several packages:

- `core`: All core framework functionality.
- `evaluation`: Classes for running tournaments and evaluations of games or AI players.
- `games`: Specific implementations of core functionality for each game.
- `gui`: Generic Graphical User Interface (GUI) helper classes. Each game can extend this to implement customised GUIs.

- **players:** All players (human and AI) available.
- **utilities:** Various utility classes and generic functionality shortcuts.

A flexible API is provided for all functionality needed to define a game, with multiple abstract classes that can be extended for specific implementations. The framework provides some generic functionality, from ready-made components, rules, actions, turn orders and game phases, to a fully functional game loop and a prototyping GUI. The GUI allows users to start interacting with the games as soon as they have the two main classes required set up: a *Game State* (GS) class, and a *Forward Model* class.

GS is a container class, including all variables and game components which would allow one to describe one specific moment in time. It defines access methods in the game state to retrieve existing game components, make custom and partially observable copies of the state, and define an evaluation function that can be used by the playing agents. The **FM** encompasses the logic of the game: performs the game setup, defines what actions players can take in a particular game state, applies the effect of player actions and any other game rules applicable, uses a turn order to decide which player is due to play next (or may wait for all players to return an action before processing for simultaneous-actions games), and checks for any end of game conditions. The FM is available to AI players for game simulations.

For each game, users can further implement specific actions, rules, turn orders, game parameters (for easy modification of game mechanics), a GUI and provision of game data. The last is useful when the game requires large amounts of data such as tile patterns, cards and board node connections, and it is provided via JSON files. A full guide on using the framework and implementing new games is available in the wiki provided with the code and in (23).

TAG’s game-loop is presented in Algorithm 8. Given a list of agents and parameters of the game to be played, the framework performs an initial setup of the game state (s_0) and the game’s forward model. While the game state is not terminal, the turn order selects the next player to act. To ensure partial observability, we generate an observation object for the current agent which hides the state of unobserved components. Hence, a list of available actions is generated and the agent is queried to provide the next action to be executed (a_t). Finally, the forward model is used to modify the game state given the action (producing s_{t+1}), and the graphical user interface is updated accordingly.

Algorithm 8 Overview Game Loop

```

1: Input: list of agents, game parameters  $gp$ 
2: Output: win rate statistics

3:  $s_0, FM = \text{SETUPGAME}(gp)$ 
4: while not ISTERMIAL( $s_t$ ) do
5:    $agent \leftarrow \text{GETCURRENTPLAYER}(s_t)$ 
6:    $observation \leftarrow \text{GETOBSERVATION}(s_t, agent)$ 
7:    $actions \leftarrow FM.\text{GETAVAILABLEACTIONS}(s_t, agent)$ 
8:    $a_t \leftarrow agent.\text{GETACTION}(observation, actions)$ 
9:    $s_{t+1} \leftarrow FM.\text{NEXT}(s_t, a_t)$ 
10:   $GUI.\text{UPDATE}()$ 

```

Games

There are currently 8 games implemented in the framework, varying from very simple test games (“Tic-Tac-Toe”) to strategy games (“Pandemic” (208)), as well as diverse challenges for AI players. A few games are currently in active development (“Descent” (209), “Carcassonne” (210) and “Settlers of Catan” (211)), and many more are in the project’s backlog, including games from other frameworks to allow for easy comparison (see Section 7.3.1). Further development plans also include adding easy to use functionality for wrapping external games, so that a direct comparison can be carried out under the same conditions without the need to re-implement games under our framework’s full restrictions.

All games implemented can be found in the `games` package, each registered in the `games.GameType` class; this class allows specifying properties for each game, to allow for automatic listing for experiments (e.g. a list of all games with the “cooperative” tag). We highlight next some particularities of the games currently implemented in the framework.

Tic-Tac-Toe 2 players alternate placing their symbol in a $N \times N$ grid until one player completes a line, column or diagonal and wins the game; if all cells in the grid get filled up without a winner, the game is a draw.

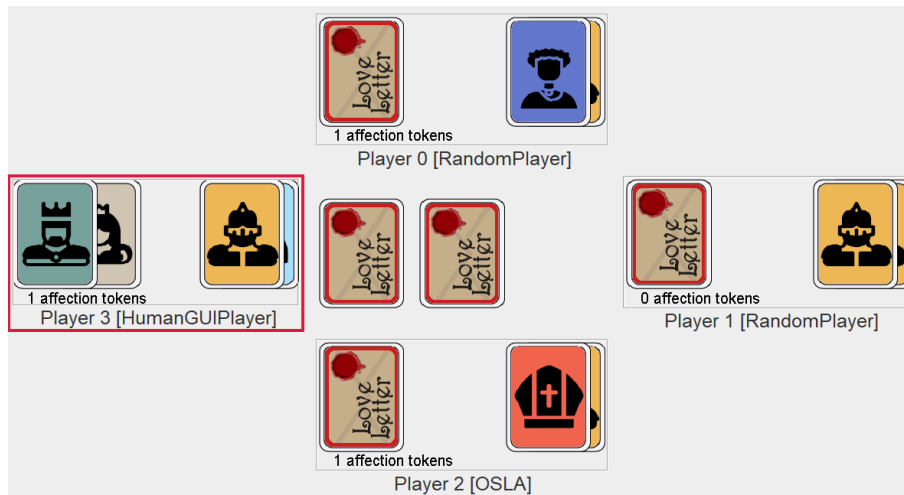


Figure 7.6: GUI for “Love Letter”, red outline shows current player (player 3).

This is the simplest game included in the framework, meant to be used as a quick reference for the minimum requirements to get a game up and running. Its implementation makes use of mostly default concepts and components, but it implements a scoring heuristic and a custom GUI for an easier interaction given the specific game mechanics.

Dots and Boxes The game starts with a 2D grid with dots in the corners of each grid cell. 2 to 5 players alternate placing cell edges (or lines connecting the dots), with the aim of adding all edges to a cell, which earns the player a point and awards them another turn. The player with most points at the end wins.

This is a fairly simple game used for illustration of game implementation in video tutorials¹². Its implementation is very similar to “Tic-Tac-Toe”, but includes further GUI customisation and a more advanced game state representation.

Love Letter (212) 2 to 4 players start the game with one card each, representing a character, a value and a unique effect. A second card is drawn at the start of each turn, one of which must be played afterwards. Card effects can target other players and may exclude them from the remaining game. After the last card of the deck is drawn, the player with the highest valued card wins the current round. A player wins the game after winning 5 rounds. “Love Letter” features partial observability, asymmetric and changing player roles and a point system over several rounds. Figure 7.6 shows an example game state.

Uno (213) The game consists of coloured cards with actions or numbers. Numbered cards can only be played in case either the colour or the number matches the newest card on the discard pile. Action cards let 2 to 10 players draw additional cards, choose the next colour to be played or reverse the turn order. A player wins after gaining a number of points over several rounds (computed as the sum of all other players’ card values). “Uno” features stochasticity, partial observability and a dynamically changing turn order. This game has the potential of being the longest game in the framework, since players need to draw new cards in case they cannot play any.

Virus! (214) 2 to 6 players have a body each that consists of four organs, which can be: infected (by an opponent playing a virus card), vaccinated (by a medicine card), immunised (by 2 medicine cards) or destroyed (by opponents playing 2 consecutive virus cards). The winner is the first player who forms a healthy and complete body. “Virus!” features stochasticity and partial observability, with the draw pile and opponents’ cards being hidden.

Exploding Kittens (215) 2 to 5 players try to avoid drawing an exploding kitten card while collecting other useful cards. Each card gives a player access to unique actions to modify the game state, e.g. selecting the player taking a turn next and shuffling the deck. This game features stochasticity, partial observability and a dynamic turn order with out-of-turn actions: in contrast to previous games, “Exploding Kittens” keeps an action stack so that players have the chance to react to cards played.

¹²<https://youtu.be/-U7SCGN0csg> and <https://youtu.be/m7DAFdViywY>

Colt Express (216) 2 to 6 players control a bandit each, with a unique special ability. Their goal is to collect the most money while traversing the two-level compartments in a train and avoiding the sheriff (a non-playing character moved by players and round card events). The game consists of several rounds, each with a planning (players play action cards) and an execution (cards are executed in the same order) phase. This processing scheme forces players to adapt their strategy according to all the moves already played, in an interesting case of partial observability and non-determinism: the opponents' type of action may be known (sometimes completely hidden in a round), but not how it will be executed. Additionally, the overall strategy should be adapted to a bandit's unique abilities.

Pandemic (208) "Pandemic" is a cooperative board game for 2 to 4 players. The board represents a world map, with major cities connected by a graph. Four diseases break out and the objective of the players is to cure them all. Diseases keep spreading after each player's turn, sometimes leading to outbreaks. Each player is assigned a unique role with special abilities and is given cards that can be used for travelling between cities, building research stations or curing diseases. In each turn, the player can play up to 4 consecutive actions, with a changing action space (e.g. moving to a new city may result in new actions). Additionally, they have access to special event cards, which can be played anytime (also out-of-turn). All players lose if they run out of cards in the draw deck, if too many outbreaks occur or if the diseases spread too much.

"Pandemic" features partial observability with face-down decks of cards and asymmetric player roles. It employs a reaction system to handle event cards and is the only game currently using the graph-based rule system.

AI Players

All implemented players follow a simple interface, only requiring one method to be implemented: `getAction`. This receives a game state object reduced to the specific player's observation of the current state of the game. How this reduced game state is built is game-dependent, usually randomising unknown information. This method expects an action to be returned out of those available and is called whenever it is the player's turn and they have more than 1 action available (i.e. the player actually has a decision to make). If no decision is required, the agent can choose to still receive and process the information on the game state (in the `registerUpdatedObservation` function) but an action is not requested. They may also choose to implement the `initializePlayer` and `finalizePlayer` functions which are called at the beginning and end of the game, respectively. Each player has a player ID assigned by the game engine, and they receive the forward model of the game currently being played. The FM can then be used to advance game states given actions, compute actions available, or reset a game to its initial state. The rest of this section defines the sample players implemented in the framework. These agents use the game's score to evaluate game states (as implemented on the game side), but their heuristic functions may be swapped with a different object implementing the `IScoreHeuristic` interface. Custom heuristics take the current state as input and return a scalar number representing the value of that state without any other restrictions. Agents can be given a heuristic function on initialisation and then instead of using the reward directly from the game they process every state they visit using the provided heuristic.

We include in the framework MCTS and RHEA players, as well as 2 simple AI players and 2 options for playing games as a human.

Human Players Two types of human interaction are available, both of which interrupt the game loop to wait for human actions on their turn. **Console** allows human play using the console. It outputs the game state and available actions in the console and the player inputs the index of the action they choose to play. **GUI** allows human play with a Graphical User Interface, which is game-specific. It uses an `ActionController` object to register player action requests, which are then executed in the game.

Random The simplest automatic player chooses random actions out of those available on its turn.

One Step Look Ahead (ISLA) A greedy exhaustive search algorithm, it evaluates all actions from a given game state and picks that which leads to the highest valued state.

MCTS and RHEA Adjustments The MCTS player runs up to depth 10 and without the simulation step, as we have seen previously that this helps its performance in games with variable and non-linear action spaces, as is the case for several games in the TAG environments. The forward model of the game is only used when expanding a leaf node. The resulting node is immediately evaluated using the heuristic and its

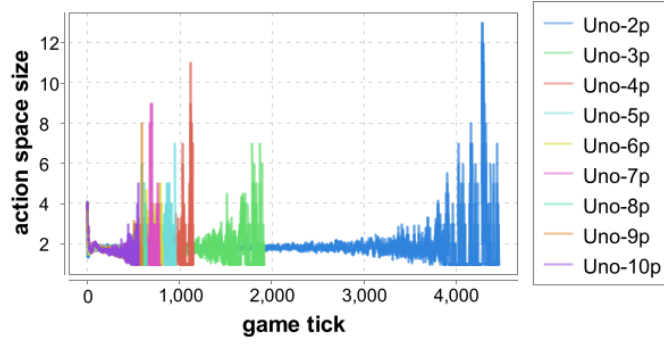


Figure 7.7: Action space size in “Uno” with all player number versions; 1000 runs per version played by random players.

value is backed up through the tree. Further, the version implemented in the framework is closed-loop: it stores game states in each of the nodes.

The RHEA player runs with 1 individual only of length 10, thus using only the mutation operator to evolve its sequence. Given the variable action spaces and that actions available are highly dependent on the current game state, the mutation operator chooses a gene in the individual (i.e. position in the action sequence) and changes all actions from that point until the end of the individual to new random valid actions. The game’s forward model is therefore used in both mutation (to advance the game state given the last action, in order to find the available actions for the given position in the sequence) and evaluation (game states reached through the sequence of actions are evaluated using the game’s heuristic, added up for a discounted total with discount factor $\gamma = 0.9$, and this total becomes the fitness of the individual). It is important to note that RHEA evolves only its own actions and opponents are given a random model (with intermediate states after opponent actions ignored in fitness evaluations).

Given the differences in implementation, we expect an equal budget of forward model calls to result in worse performance for RHEA, as its decision-making process is much more expensive than MCTS and it can perform fewer iterations.

7.3.3 Discussion

This section gives an overview of game analytics which can be extracted from all games, as well as some preliminary results for the sample AI players described in Section 7.3.2.

Game Analysis

Table 7.7: Analysis of games, played 1000 times for each possible number of players on each game, using random agents.

	μ_1	μ_2	μ_3	μ_4	μ_5				μ_6				μ_7
					Setup	Next	Actions	Copy	#decisions	#ticks	#rounds	#APT	
Tic-Tac-Toe	5.69	6.79	1	0%	10^5	10^6	10^5	10^6	7.24	7.61	3.17	1	[0.0, 0.103] sd=0.02
Love Letter	4.74	10.78	24.00	62.96%	10^5	10^6	10^5	10^6	53.22	109.48	6.89	1.96	[0.0, 0.97] sd=0.20
Uno	1.88	4.32	116.00	92.3%	10^4	10^6	10^4	10^6	193.51	540.78	6.07	1	[-0.15, 0.27] sd=0.05
Virus!	9.70	11.03	78.00	89.1%	10^4	10^6	10^5	10^5	317.09	319.56	75.75	1	[0.0, 0.8] sd=0.17
Exploding Kittens	3.17	4.99	60.00	84.5%	10^5	10^6	10^5	10^6	51.86	73.23	8.98	1.07	[-0.5, 0.75] sd=0.11
Colt Express	2.91	5.46	87.60	87.67%	10^4	10^6	10^4	10^6	91.71	176.13	5.00	1.00	[-0.5, 0.5] sd=0.14
Pandemic	11.15	17.97	138.00	64.97%	10^2	10^5	10^4	10^5	108.62	173.94	8.25	5.58	[-1.0, 0.42] sd=0.12

All games in the framework can be analysed to illustrate the challenge they provide for AI players, with several metrics currently readily available. Here, we present measurements currently implemented and applied to the existing games described in Section 7.3.2. We measure the following *averages*, observed in our experiments:

Action space size: the number of actions available for a player on their turn (e.g. Figure 7.7). **Branching factor:** the number of distinct game states reached through player actions from a given state. **State size:** the number of components in a state. **Hidden information:** the percentage of components hidden from players on their turn. **Game speed:** the execution speed of 4 key functions (in number of calls per second): setup, next, available action computation and state copy. **Game length:** measured as the number of decisions taken by AI players, the total number of game loop iterations (or ticks), the number of rounds in the game and the number of actions per turn for a player. **Reward sparsity:** granularity of the heuristic functions provided by the game, measured by min, max and standard deviation of rewards seen by players.

More complete information and graphical visualisations can be obtained by running the `evaluation.GameReport` class included with the framework. Results are presented in Table 7.7 (“Virus!” games were limited to 100 rounds, as random play can lead to infinite games).

When looking at the games currently implemented, the first thing to note is that all games are very fast to execute: most games can execute over 1 million calls per second to the (usually) most expensive functions (`next` and `copy`). The games vary in length, with only 7.61 ticks for the simplest game, “Tic-Tac-Toe”, but 540.78 for “Uno”. We further see variations in the state size, with “Pandemic” showing most complex, while “Uno” includes the most hidden information. “Love Letter” shows its strategic complexity through the higher branching factor (10.78), while “Exploding Kittens” boasts one of the largest spread of rewards.

Many of the metrics reported do not paint a complete picture if only their average is given: action spaces, for example, vary widely during play for most of the games presented. An example which highlights this is “Uno”, where the action space is dependent on the number of cards in a player’s hand, increasing on average as the game goes on (see Figure 7.7). It is further interesting to note here that games get shorter with more players (as players would earn more points per round if they have more opponents, and thus more cards to total the sum of) - an example of insights which can be obtained through analysis of the games themselves, readily available for any newly implemented games in the framework.

Baseline AI Player Performance

We tested the performance of the sample agents on each of the implemented games. For “Tic-Tac-Toe”, we report the win rate per agent when playing 100 times against every possible opponent. Since “Pandemic” is a cooperative game, we report results from games played with a team of 4 instances of the same agent, e.g. 4 MCTS players. For the remaining games, we report the average win rate per agent when playing in a 4-player match against one instance of all other agents. All but the random agent are using state evaluation functions that are provided with each game. For both search-based algorithms, we use a budget of 4000 calls to the `FM.next()` function.

The average win rate per games is shown in Table 7.8. Our results indicate the MCTS agent to be the best, achieving the highest average win rate in 5 out of 7 competitive games, with an overall win rate of 41% in these games - thus clearly dominating the other agents. While RHEA also outperforms random in most of the games (with the exception of “Uno”), it still falls behind the ISLA agent. This could be due to the large uncertainty built up in its rigid sequences of actions (as opposed to the flexible game trees built by MCTS) in these games with partial observability and stochasticity, where a greedy approach appears to be preferable. Further, some games tested are simple enough that a greedy approach works best and are highly advantaged by the heuristics provided by each game (e.g. “Tic-Tac-Toe”). However, we do observe RHEA to clearly dominate all other agents in “Dots & Boxes”. This is an interesting case to be analysed in more details.

Additionally, we note that no agent is able to win in the cooperative game “Pandemic”, as they are unable to perform the multi-faceted long-term planning required to avoid all of the loose conditions and win the game. Further analysis, such as the distance from winning game states for each AI team, could show interesting insights into these agents’ capabilities (205). We note that Sfikas and Liapis (205) obtained good results in the game, but with several simplifications to the environment and abstractions meant to aid the decision-making process: *No Event cards*, which we do include (and often significantly increase the action space for the player). *Limited and fixed-order player roles*, whereas we test all player roles, randomly allocated at the beginning of the game (some of which significantly increase the action space). *RHEA controls all players in the game*, as opposed to our version where 4 instances of RHEA control the different players and create their own plans. Player actions are abstracted into *macro-actions* of different types, which are ranked in terms of importance and used to seed RHEA; we do not use such domain knowledge in our implementation and our RHEA agent only considers single actions. RHEA uses a *determinization* enhancement (the hidden information in the state is randomised for each individual evaluation).

While such an experimental setting could be implemented in TAG, we consider these to be very specific domain adaptations and simplifications which move away from the general game-playing AI concept. However, adopting an approach such as curriculum learning to start from a simplified setup and slowly add features back in until we reach our full game implementation could aid in bringing the great results seen in (205) into our more complex environment and will be explored in future work.

Moreover, we observe “Uno” and “Colt Express” as cases where the performance between all players, including random, is very close (22 – 26% in “Uno” and 19 – 29% in “Colt Express”). This highlights the difficulty of the types of problems proposed, as well as the importance of the heuristic chosen for a game, as some features of a game state may prove deceiving.

However, the statistical forward planning methods described here (MCTS and RHEA) benefit from ample literature with a large parameter space each, which could be tuned for increased performance.

Table 7.8: AI player performance, 100 game repetitions. Highest win rate in bold. “Tic-Tac-Toe” played in a round-robin tournament. “Pandemic” uses 4 instances of the same agent. All others played in their 4-player variants, with 1 instance of each agent.

	Tic-Tac-Toe	Dots & Boxes	Love Letter	Uno	Virus!	Expl. Kittens	Colt Express	Pandemic	Total
Random	0.12	0.00	0.00	0.26	0.01	0.05	0.19	0.00	0.08
ISLA	0.45	0.14	0.24	0.26	0.27	0.37	0.29	0.00	0.25
RHEA	0.44	0.69	0.32	0.22	0.3	0.21	0.28	0.00	0.31
MCTS	0.98	0.17	0.44	0.26	0.42	0.37	0.26	0.00	0.36

7.3.4 Challenges and Opportunities

The presented framework opens up several directions of research and proposes a variety of challenges for AI players, be it search/planning or learning algorithms. Its main focus is to promote research into General Game AI that is able to play many tabletop games at, or surpassing, human level. Related, the agents should be able to handle both **competitive** (most common testbeds in literature), **cooperative** and even **mixed** games. For instance, a future planned development is the inclusion of the game “Betrayal at House on the Hill” (217), in which the players start off playing cooperatively to later split into teams mid-way through the game, from which point on they are competing instead with newly given team win conditions and rules. Most tabletop games include some degree of **hidden information** (e.g. face-down decks of cards) and many more players compared to traditional video-game AI testbeds, introducing higher levels of uncertainty. However, such games often make use of similar mechanics, even if in different forms: thus **knowledge transfer** would be a fruitful area to explore, so that AI players can pick up new game rules more easily based on previous experiences, similar to how humans approach the problem. Some tabletop games further feature **changing rules** (e.g. “Fluxx” (218)) which would require highly adaptive AI players, able to handle changes in the game engine itself, not only the game state. Many others rely on large amounts of content and components, for which the process of creating new content or modifying the current one for balance, improved synergies etc. could be improved with the help of Procedural Content Generation methods (e.g. cards for the game “Magic the Gathering” (219) were previously generated in a mixed-initiative method by (220)).

Specific types of games can also be targeted by research, an option highlighted by TAG’s categorisation and labelling of games and their mechanics. Thus AI players could learn to specialise in games using certain mechanics or in areas not yet explored, such as **Role-Playing** or **Campaign** games (i.e. games played over several linked and progressive sessions). These games often feature **asymmetric player roles**, with a special one highlighted (the dungeon master) whose aim is to control the enemies in the game in order to not necessarily win, but give the rest of the players the best experience possible and the right level of challenge. Strategy AI research could see important applications in this domain, as many tabletop games include elements of strategic planning. Role-playing games focused more on the story created by players (e.g. “Dungeons and Dragons”) rather than combat mechanics (e.g. “Gloomhaven”) would also be a very engaging and difficult to approach topic for AI players, where Natural Language Processing research could take an important role.

The framework enables research into **parameter optimisation**: all parameter classes for games, AI players or heuristics can implement the `ITunableParameters` interface; parameters can then be automatically randomised, or more intelligently tuned by any optimisation algorithm. This allows for quick and easy exploration of various instances of a problem, a potential increase in AI player performance, or adaptation of AI player behaviour to user preference.

We have mentioned previously that the games implemented offer reduced observations of the game state to the AI players, based on what they can currently observe. These hidden information states (usually) do not keep a history of what was previously revealed to a player. Instead, the AI players should learn to memorise relevant information and build **belief systems**, as humans would in a real-world context - a very interesting direction of research encouraged by TAG.

Lastly, the framework includes the possibility for games to define their states in terms of either **vector observations** (`IVectorObservation`), which enables learning algorithms to be easily integrated with the framework; or **feature-based observations** (`IFeatureRepresentation`), which allows for more complex algorithms which can perform a search in the feature space of a game, rather than the usual game state space approached.

7.4 Conclusions

This chapter describes the application of RHEA in several specific environments. All of the environments are much more complex than the GVGAI games used in previous experiments. RHEA is adapted slightly to each scenario, in particular by using custom heuristics to evaluate game states. Its performance is then compared against several algorithms in all cases, and shown to achieve great results and interesting behaviours.

Pommerman. First, we presented a study on the performance of statistical forward planning (MCTS and RHEA) agents on the game “Pommerman”. The first conclusions that can be drawn are that SFP methods are stronger than the Rule-Based and One Step Look Ahead agents they were compared against. Additionally, the configuration tested for MCTS seems to provide a better performance than that of RHEA. The analysis carried out on the different algorithms shows that more offensive strategies (like RHEA dropping more bombs on average) are normally also riskier, due to the known challenge of suicides in this game. In fact, MCTS tends to place less bombs than the other agents, but achieves a higher winning percentage in most modes and visibility settings. Partial observability increases the number of suicides for all agents, and full observability brings the performance of MCTS and RHEA quite close. The PO setting also influences where and how often bombs are placed (as seen in the heatmaps presented), and shows differences of behaviour between the different SFP methods. Adding heuristics and belief systems to remember where and when bombs were placed may lead to a performance boost, as the agents could make better use of information gathered in previous game ticks and use its experiences to the maximum benefit they can provide. Another interesting observation is that many games end up in ties, especially when the visibility range of the agents is greatly limited. One possibility to alleviate this is to adopt the collapsing boards methodology followed in the “Pommerman”, by which winners are enforced.

RHEA is an algorithm with a large parameterisation space. Further tests done in “Pommerman” with other parameters (P , L , mutation rate, etc.) have shown different results for different game settings. A possibility of future work is to automatically tune RHEA parameters to boost the strength of this method, which in many other games has shown competitive performance with MCTS. A similar approach can be taken for MCTS, as in (32), although this algorithm has a smaller parameter space (comparing vanilla versions). Our initial tests on this matter suggest that this is indeed possible.

Other lines of future work, which tackle directly performance in “Pommerman”, are of a wider interest for the Game-Playing AI community. One of them is to learn an effective opponent model of the other agents, which improves upon the random modelling assumed in this paper. The random model was seen to often lead agents to take reckless decisions, as they might observe in their simulations that their opponents suicide, leaving the agent to believe that whatever they do, they can outlast their opponents (even if that means dying themselves). Simple statistical modelling based on the frequency of actions have shown promising results in the past for 2-player GVGAI (221) or other games (222). Other interesting avenues for future work are to tackle partial observability by introducing assumptions of the unknown tiles of the FM (167), or learning value functions that identify trap states or moves (that can cause suicides in the game). Moreover, it would also be interesting to compare these planning SFP approaches to learning agents submitted to the “Pommerman” competition (223; 224), or even investigate more hybrid methods for this game. Lastly, a deeper analysis into the agent behaviour could help paint a clearer picture of their strengths and weaknesses and better inform future developments (225; 226).

Tribes. Next, we discussed the “Tribes” environment, a multi-agent, multi-player, stochastic, partially observable strategy game. It poses many challenges for AI players, including technology and resource management, build orders, unit coordination, opponent modelling and long-term strategic planning, with a large and variable, non-linear action space.

In the context of this thesis, we highlight the high performance of RHEA in this game, when faced with several other players, including MCTS, Monte Carlo, One-Step Look Ahead and random. RHEA is successful in beating all other players and achieves 74.84% overall win rate in all of its games. However, this agent fails a test against human play and shows the need for higher-level planning for better unit coordination and resource management. A surprising finding of the study was the strength of the simple rule-based system, which ranks second out of all players tested, with a different strategy highlighted by end of game measures of several game features. This indicates further analysis is needed to find methods able to surpass the performance of this simple player.

Follow-up work to this study is focused around testing performance in the other game mode available, as well as more of the tribes, more than 2 players, partial observability and other tweaks to the many game parameters available in the “Tribes” framework. Adding other advanced AI game-playing methods would also be straightforward, with those showing high performance in recent multi-agent games research being targeted, such as *Online Evolutionary Planning* (76) and *Evolutionary MCTS* (81). Another line of future

works looks at extending the framework to allow for easy integration of model-free reinforcement learning or deep reinforcement learning techniques. The current visualisation of the game poses an excellent starting point for integrating methods that use screen capture for decision-making. The framework also opens and promotes work into procedural content generation, in particular creating levels with varied biomes that fit the characteristics of a particular tribe and facilitate their growth in the environment - or, on the contrary, pose interesting environmental challenges.

Tabletop Games. Lastly, we discussed the Tabletop Games (TAG) framework, which aims to promote research into general Artificial Intelligence with features for easy implementation and bridge-building between tabletop games and artificial players, with some examples already included. We further analysed the games in the framework, showing a wide variety of action spaces, information available to the AI agents, duration etc., as well as tasks and challenges introduced. The AI player performance analysis shows Monte Carlo Tree Search to dominate all other sample agents in the framework, with simple greedy methods being surprisingly competitive in some of the games. Despite this, the RHEA agent does show high performance in several of the games, suggesting promise for the application of evolutionary algorithms in tabletop games. Overall, however, the problems proposed are far from being solved.

The framework opens up and facilitates many directions of research, and yet many more developments are possible and planned. More measurements for both games and AI players can be added, to paint a more complete picture of the challenges the players would face, as well as the current state of available methods for such games: skill depth, overall state space size, stochasticity, size of search trees, player role asymmetry - all would give a much more in-depth view of the framework as a whole, aiding in future developments of both tabletop games and general AI players.

Further, we aim to facilitate interfacing external games with our framework in order to gain the full benefits of the in-depth analysis and interaction with the implemented players without the need to re-implement everything from scratch: this would open up the framework to many more already existing games, and also increase the number and complexity of environments the AI players can be exposed to, improving their quality as well.

Next chapter. In the next chapter, we take a look at novel research directions that expand upon the work presented so far in this thesis: work which is at the fore-front of innovations in RHEA and which discusses in-depth a variety of topics and potential further advancements, as well as showing preliminary results obtained when applying the concepts in existing or key new environments.

Chapter 8

Further Research Pathways

In the end of this thesis, we look towards the future, and the exciting areas of research that can build on top of the work presented so far. This chapter includes both visionary work describing paths for the future, but also work already published extending from my work in interesting new directions - projects that I have been a part of, but not a main contributor of.

8.1 Project Thyia: RHEA as AI Entity

The work in this section was published at IEEE CoG 2019:

R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Project Thyia: A Forever Gameplayer," in *IEEE Conference on Games (COG)*, 2019, pp. 1–8.

The quest for artificial general intelligence (AGI) has been pursued for many years. Yet "no free lunch" stands true to this day (149) and there exists no one method that is able to solve all problems. Some researchers have been trying to model the human brain in order to give algorithms the power of learning that humans have (227). Generally, humans are fairly good at learning how to perform a variety of tasks ranging in difficulty, from using our body (walking, picking things up, dancing), to maths (counting, fractions, solving equations), to producing creative works (writing, drawing, painting, designing games) and all in-between. Not all of us are the best at all the different tasks, but most of us are fairly good at various tasks within the same domain and at figuring out how new problems work given the knowledge base built over our lifetime.

Although a lot of important advances in AI have been made in games, and games are still actively used as testing environments for AI, algorithms are only able to solve (in this case, win or achieve a high score) a subset of existing games (9). Planning and learning algorithms alike are unable to act in an intelligent manner in all given games, unless they use human-tailored heuristics or features (often game-specific). They do excel in some games, and different methods are better at different types of tasks. Here, we look at it from the perspective of human intelligence: humans do not only learn, or do not only plan, when faced with a new problem. They plan based on existing knowledge, execute the plan, and use the new experiences to update their knowledge. We believe that combining planning and learning methods is key to AGI.

We do notice one drawback in game-playing AI research that is rarely addressed, to our knowledge. The usual steps of running a game-playing AI algorithm are as follows:

1. Write / obtain algorithm.
2. Set up problem domain.
3. Press run.
4. Run ends with some result, maybe statistics.
5. Instance of AI no longer exists.
6. Rerun new instance of AI for new result.

Point 5 here is where our interest lies. Even in the case of learning algorithms, they run for a limited number of steps or episodes, however long the researcher can afford to spend testing the method (in either time or money). The algorithm may converge in the given time, therefore, even if given longer, it would not do any better, but often it does not. If any bugs are found or thought to exist, the knowledge acquired previously is scrapped and it all starts from zero again. The fact that the AI is able to learn to play well

some games when starting from scratch is seen by many as a positive (228). Others give the AI a starting point considered good, whether it is human knowledge or large amounts of existing data (229; 230). But the knowledge stored in a trained model would not be reused in subsequent runs. Most planning methods do not have memory, thus they start from scratch at every run of a game. As most algorithms are stochastic, it is possible to store the random seed of a good run to reuse later on; we consider this to be a small attempt at copying the AI instance, instead of preserving it.

A different scenario is presented by bots hosted on servers and interacting with humans, such as general chat bots, Twitter or Slack bots (231). These bots are given mostly social intelligence, so that they can respond to human requests, or maybe even initiate conversation (more rarely). We take from these the concept of (almost) permanent existence in the cyberspace, as well as their ability to interact with humans. Other examples of popular entities include Alexa, Siri, Cortana and other voice assistants (232), which incorporate human speech understanding to be able to communicate with humans, be it to offer information or a clever joke.

A similar example from the game AI domain is ANGELINA, the game designer (233). Unless Michael Cook decides to take her down for updates or a break, she continuously and autonomously creates games, improves them, names them, or throws them away if she decides they are unworthy. She also sometimes interacts with humans, either through Twitter messages, streaming the design and test process on Twitch or by simply sharing the games it creates¹. In (233), Cook and Colton discuss the benefits of continuous creative systems, highlighting long-term growth and development.

In this section, we present Eileithyia (or Thyia for short), the game player. Inspired not only by previous research and internet trends, but also by Greek mythology. Eileithyia is the Greek Goddess of childbirth, or life, as this would be a more interesting framing in this context. She is also the daughter of *Hera*, which may seem an insignificant detail, but as an anagram of RHEA, we consider it a happy coincidence. Thus Thyia is now Goddess of artificial life in a game-playing context, a system combining learning and planning. Thyia would act in a similar way to ANGELINA: she exists in cyberspace (or, simply put, continuously running on a server) and her purpose in her potentially endless life is to play games and become the best player humans have seen. Thyia uses planning methods to play games, informed by the knowledge she gathers over her lifetime, and learns from her experiences to improve her performance over time.

We highlight that Thyia would be the first step towards combining multiple areas of research and increasing their presence and potential impact in the real world. We envision that game-playing agents would be able to use the knowledge and experience gathered by Thyia to further improve their own performances even in time-limited scenarios. Thyia is meant to showcase the true strength of modern techniques when used together for long-term development.

We can summarise our contributions as follows. We propose a new way of thinking about game-playing Artificial Intelligence as entities that exist in cyberspace. We introduce Project Thyia, which centres around such an AI entity. We combine existing planning and learning methods to allow Thyia to not only plan through games, but also learn from experiences and improve over time, by using knowledge acquired as well as by tuning its large parameter space and structure. Finally, we discuss difficulties imposed by a continuous game player.

We will first review related literature in Sections 8.1.1, 8.1.2, 8.1.3 and 8.1.4. The concepts behind Project Thyia are described in Section 8.1.5 and Section 8.1.6 addresses ethical concerns related to the project.

8.1.1 AI Entities

Creating artificial life that we, as humans, can interact with in meaningful ways is the topic of many books and films, with ongoing research trying to make such concepts into the real world. We will explore some of the advances in this area in this section, with a focus on interactive ‘always-on’ AI.

Perhaps a most commonly accessible form of interactive AI is chatter bots. They are generally AI algorithms running continuously on a server, accepting some form of human input (i.e. text or speech) and returning some output in response to the input received. Conversational bots (234) are largely based on natural language processing and databases of appropriate responses which could be manually designed or automatically extracted (235). An early program with such intent is Weizenbaum’s Eliza (236), a chatterbot built to respond to certain keywords in order to facilitate communication between man and machine. This concept is also used in the development of platform-specific bots, such as Twitter bots, which may post various content on its intended platform with the aim of interacting with other users (231).

Some games adopt this concept and research on social, interactive characters in order to create more immersive experiences for their players. A great example here is Emily Short’s game “Galatea” (237), which focuses on interactive storytelling. Emily Short writes about NPC conversation systems (238), showcasing

¹<https://gamesbyangelina.itch.io>

different functionalities these can take. We are interested in the tutorial system most, although in our case it would be reversed: the humans would be giving the AI hints, and not the other way around, as in Matt Wigdahl’s “Aotearoa” (239) or Santiago Ontañón’s “SHRDLU” (240). These characters, although becoming more and more impressive with the inclusion of memory, personality and adaptation to different players and play styles, they do exist only within the game. Our vision wishes to take this concept further and bring more presence into the real world to such characters.

Darius Kazemi, known as Tiny Subversions², is an internet artist who creates interesting ‘continuous’ bots. One example is his “Random Shopper” project, which consists of an AI entity that buys a new random item from Amazon every month, within a certain budget, and has it shipped to Kazemi’s home for a regular surprise.

Zukowski and Carr recently used Deep Learning to create an AI entity which live-streams music it generates on YouTube (241). The virtual band, Dadabots, creates death metal pieces with the aims of showing that AI is able to capture interesting differences between various music genres.

Perhaps the most notable AI entity with a presence in the real world from the games domain is ANGELINA. Initially created as an automatic system for designing entire games, with a focus of exploring the limits of a software’s creativity and novelty while creating interesting playable experiences (242; 243). Cook and Colton describe in (233) the extension of their vision for autonomous continuous game creation, with a highlight to the opportunities this methodology opens for long-term improvement. We base our concepts largely on their ideas, extending further to a multi-faceted game-playing system.

8.1.2 Continual Learning

An interesting research area which addresses similar problems to our domain is that of continual learning (also known as lifelong learning, or sequential learning), which focuses on long-term learning for continuous development, often on a sequence of different tasks. One definition of this domain is the study of agents capable of interacting with their environment (in our case, a game), with limited computational resources, that is started once and run for a long time (once started, no more changes are allowed) with the aim of continuously improving at fulfilling its goals over a period of time (244). The main differences to our system are the lack of changes allowed once it starts (we wish to allow for updates and changes in the modules part of our system), as well as the lack of interaction with the wider world outside of the agent’s environment (we wish our agent to not only be getting better at the games it plays, but also to have a presence in the real world).

However, a lot of the concepts described in continual learning research can be applied in our case as well. This section will review several recent works with relevant and interesting results and/or takeaways.

Parisi et al. (244) review several works in the area. They note that current approaches are still facing several issues, including flexibility, robustness and scalability. Interestingly, most learning models rely on large amounts of annotated data to function in supervised domains. We wish to emphasise our focus on efficient learning and gathering of data for learning from our planning agent, as well as the modularity of our proposed system which would allow for new games to be added in, which may not respect the same assumptions of our current corpus of games. Thus flexibility and robustness are key aspects we consider, with scalability to more complex game domains an interesting path for future developments.

Lopez et al. (245) address the problem of forgetting as well as knowledge transfer (backwards, to apply new knowledge to previous tasks, and forwards, to use current knowledge to learn new tasks more efficiently). They propose a Gradient Episodic Memory (GEM) model which shows good performance on a range of MNIST (a large database of handwritten digits) and CIFAR-100 tasks (a set of classification tasks). Later, Chaudhry et al. (246) improve upon this algorithm to focus its efforts on the average loss over all previous tasks, resulting in similar performance to GEM, while being much more computationally and memory efficient.

Aljundi et al. (247) introduce the concept of selfless continual learning, which refers to allowing for future tasks to be added to the sequence of tasks being solved. This idea is essential to our system, which we envision to be continuously learning over a continuously expanding set of games.

One example of applying continual learning methods to the games domain is the work by Schwarz et al. (248), who propose an algorithm which compresses its memory after learning each new task so as to preserve key concepts, both old and new. They test their method on the Atari suite and show it to be better than other knowledge preservation methods like Elastic Weight Consolidation (EWC) (249) on several games. These methods, as well as those in (245), (246) and others can be used to enhance our learning component, although in this section we choose to focus on simpler Neural Network approaches.

As highlighted by Diaz et al. (250), most of the focus on continual learning is on memory retention, shaping the knowledge acquired and selectively deciding what, when and how to expand the knowledge, so

²<http://tinysubversions.com>

as to improve performance on new tasks without affecting previously learned ones. Diaz et al. suggest that evaluation of these methods is also very important and propose using several metrics applied at intervals during the learning process: accuracy, knowledge transfer, memory and computational efficiency. They combine these into an overall weighted-sum score based on which they can rank various methods on the iCIFAR-100 dataset. Farquhar et al. (251) approach this same problem of evaluating continual learning methods, further stressing the importance of choosing the right metrics for evaluation. They suggest that current metrics used have a bias towards methods which use previous models as priors for new learning tasks, making these appear superior although a different choice of experimental design would alter rankings. To this extent, the authors propose several principles that should be followed or acknowledged as missing when evaluating continual learning algorithms, such as the usage of several (more than 2) tasks (251). We consider these notes important for future evaluations of our system.

8.1.3 Learning while Planning

There has been increasing interest in the research community regarding the combination of learning and planning methods. Each have their own strengths and weaknesses. Learning methods not only require some form of game state feature extraction in order to be able to play games; but they also need significant resources for training and they lack generalisation across different tasks (252). Planning methods have proven to be very good at a variety of different tasks and they can work online, with no training, and in real-time; but they do require a model of the game in order to simulate possible future scenarios, and they struggle in sparse reward environments (19). One way forward is to draw upon the strengths of both techniques in order to build a fast, effective and general algorithm.

There have been several advances in this direction in board games, where the approach is to use a search algorithm (often Monte Carlo Tree Search) as an expert to generate gameplay data, and a learning algorithm (often a Deep Neural Network) which uses the gameplay data to train and perform better than either algorithm would individually. A prime example is AlphaGo (253), followed by AlphaGoZero (188) and AlphaZero (228), all of which combine Monte Carlo Tree Search (which generates gameplay data) and Neural Networks (which use the gameplay data to train policies and value estimates) to successfully beat the state-of-the-art in the game Go (and Chess and Shogi in the case of the latest AlphaZero program).

Anthony et al. (254) split the task of playing the game of Hex into two areas: planning efficiently and generalising the plans across different boards and opponents. They use tree search to plan, aided by a neural network policy to guide the search, and Deep Learning to further generalise the plans. They pinpoint the benefits of their approach and the great results of combining the two approaches, which mean the agent is capable of winning against previous champions.

These ideas were further developed and applied to video games, which differ mainly through their real-time aspect, as well as increased complexities of dynamic worlds: AI methods only have a limited time to make decisions. In this section, we also focus on video games, although the concepts described could be extended to any games or problems. Jiang et al. (255) apply a combination of Monte Carlo Tree Search (MCTS) and Neural Networks to obtain a competitive “King of Glory” player, where the heuristic used by MCTS to evaluate leaf nodes is improved incrementally based on the results returned.

Lowrey et al. (256) developed a framework based around the idea of combining planning and learning, called POLO, which consists of several continuous control tasks, such as humanoid locomotion or hand manipulation. Most interestingly, they suggest that this combination of methods brings several benefits including reducing the planning horizon, while finding good solutions beyond the local space currently being explored.

A particularly relevant recent work is a combination of Neural Networks (NN) and Rolling Horizon Evolutionary Algorithms (RHEA) applied to a series of MuJoCo control tasks (157). Tong et al. apply the idea used in AlphaGo works by replacing the MCTS with RHEA, and using NNs to generate policies and state value estimations. The policies are used to initialise RHEA, while the state value estimations are used in the fitness evaluations of plans generated by the evolutionary algorithm. The network is then updated after the evolutionary process completes in order to improve value estimations and, implicitly, the policies as well, so that the whole system learns to generate better plans over time. They suggest their method is efficient in learning interesting behaviours. We adopt these ideas into the learning part of our system with several extensions, as detailed in the next section.

8.1.4 Planning to Learn

With a different perspective, we consider active learning as a form of increasing the autonomy of our system. Within pattern classification and language learning, active learning is used to describe cases where the learning agent generates its own patterns to submit to an Oracle which then informs the learner of the

class of pattern (257). Active learning algorithms can use this approach to formulate highly relevant queries that may enable more sample-efficient learning.

Reinforcement learning (RL) agents are already in control of their own destiny within a game, since what they experience depends on the actions they take. In particular, RL algorithms can use concepts such as intrinsic motivation (which may be related to novelty search) to take an active approach to learning in the absence of sufficient external reward signals. This has been used to good effect to boost performance on games with sparse reward landscapes such as Montezuma's revenge (258).

Beyond this, we envisage an even more active type of RL in at least two cases: setting in-game scenarios in order to improve performance on a particular game, and deciding which games to play in order to maximise "personal" development. In the first case, an agent would reason about its current state of ignorance, and set up scenarios to test various hypotheses. Any game that allows user-defined levels would support a form of this, though the ideal would be to have a set of learning games with an active interference API (Application Programming Interface), allowing for alterations of the game state mid-game in order to explore specific consequences. This would enable a much more direct manipulation of the game state than could be achieved by the normal process of taking actions within a game, in turn leading to more sample-efficient learning of highly performing strategies.

In the second case, the agent would analyse its more general short-comings and select games with which to hone its skills. As far as we are aware, neither approach has been tried within the field of AI and Games.

8.1.5 Proposed System: Thyia

Thyia is a large system comprised of several modules. The modularity aims to allow for different parts to be maintained, improved, updated or rebooted independently, in order to avoid potential loss of data in the larger system. See Figure 8.1 for an abstract representation of the system envisioned in this section.

Game Set Module

As we are targeting a planning agent which also learns over multiple runs, an essential part of the system is the set of games. Multiple frameworks for general-purpose game-playing exist. One of the earliest general game frameworks was Metagamer, which used a Game Description Language to define the rules of chess-like games (i.e. Chess, Chinese Chess, Checkers, Draughts and Shogi) and automatically generate variations for game players to attempt to beat (259). A similar project was developed by Jeff Mallett and Mark Lefler, called Zillions of Games³. They expanded the range of games to general board games, using a LISP-like language for game definition. The AI received a lot of information about the game: not only the actions available, but also the board structure and the goals of the game. Humans could use the system to not only create new games, but they could also choose to play their games against an AI using alpha-beta pruning and transposition tables.

This idea evolved further into the General Game Playing competition (GGP) (190), which includes turn-based deterministic board and puzzle games and offers the entire rule set to the agents in order for them to determine their strategy. The Arcade Learning Environment (ALE) (43) improves on this challenge with their focus on real-time deterministic arcade games. ALE in particular has received much attention in the last few years from AI researchers (260; 143; 258).

However, we choose to use the General Video Game AI framework (GVGAI) (261), which comprises of a large (and increasing further on a yearly basis) set of games with different properties of interest, such as partial observability, stochasticity and a variety of game mechanics and score systems (including dense and sparse rewards). The method proposed in this could be used in any of the GVGAI games, as they all use a common interface and no game-specific knowledge is embedded in our system - thus it presents an emergent quality for generalisation across games. Additionally, GVGAI uses the Video Game Description Language (VGDL) to define its games, which allows for easy creation of new games or variations of existing ones, leading to a potentially infinite supply of games varying in features and complexity.

A possible line of work to increase the presence of this system would be an integration with a game designer (e.g. ANGELINA (233)). It follows naturally that ANGELINA and Thyia could make an excellent team, one creating games based on the results and feedback of the other which plays them.

Planning Module

The core part of the system will be the actual game player: the algorithm which is able to play unknown games. We choose to base the game player on a planning method, due to their flexibility, adaptability and lack of training necessary, as well as high performance across multiple games (9). The downsides of these methods are twofold: they do not usually learn between games (i.e. the performance of the method is likely

³<http://www.zillions-of-games.com/ZOG.html>

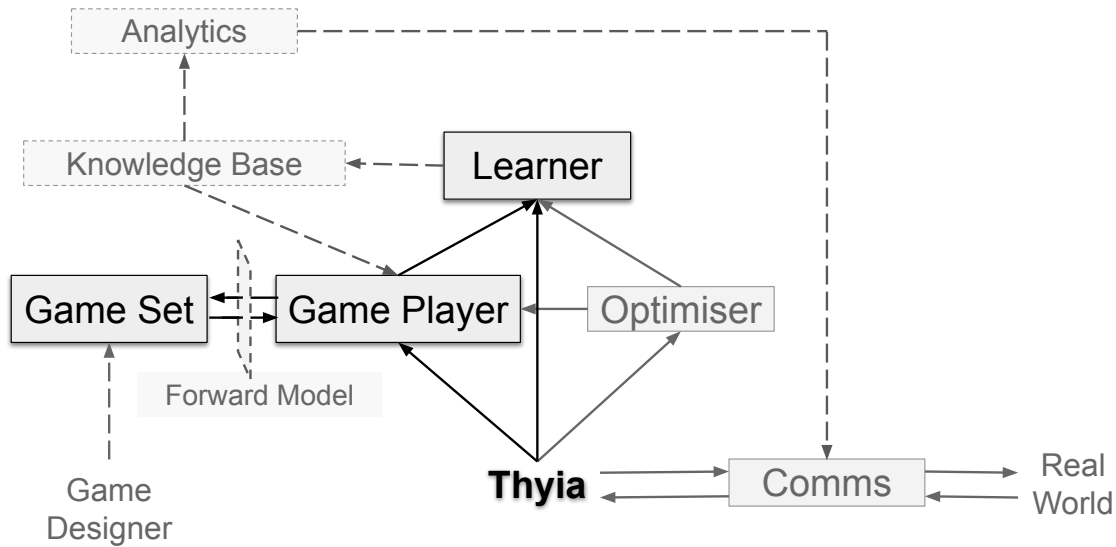


Figure 8.1: Thya system. Composed of 3 core modules: a game player (planning AI agent), a learner and a game set. Additional modules include external communications with the “real world” for game sharing and human interaction; and an optimisation module for tuning the game player and learner’s parameters. Further enhancements include a knowledge base, detailed analytics and forward model learning. We include a possible connection with a game designer, which would be providing games for the system to play.

to be the same the first and the hundredth time it plays a game, save for game or algorithm stochasticity) and they require a game model to be able to simulate possible future states. The first issue is addressed throughout this section; the second will be discussed further in the Chapter conclusions.

Although Monte Carlo Tree Search is a commonly used in game-playing research (59), as well as some commercial games (e.g. Creative Assembly’s “Total War: Rome II” (262)), recent work has shown Rolling Horizon Evolutionary Algorithms (RHEA) to achieve a higher performance in many games (17), as well as offering many customisation opportunities: the underlying Evolutionary Algorithm can range from a 1+IEA (133), compact GA (263), CMA-ES (157) and population-based GA (15). We choose to start with RHEA for the planning module, but other methods such as MCTS could also be easily integrated.

Many RHEA hybrids can be created for interesting and diverse results (16; 17), while the idea behind the algorithm remains the same: evolving sequences of actions at every game tick and evaluating each sequence with the use of a game model, to simulate through the actions and assess the final state reached by following the given actions. The value of this state, given by a heuristic, becomes the fitness of the individual. At the end of the evolutionary process (when some budget has been reached), the first action of the best individual is chosen to be played in the game.

In our implementation, RHEA includes over 30 parameters, allowing for not only operational settings to be modified (i.e. individual length, mutation rate), but also the very structure of the algorithm (keeping the population evolved from one game tick to the next with a shift buffer, including or excluding evolutionary operators, adding Monte Carlo rollouts at the end of the individual when evaluating, etc.). These options are all collected from past literature (15; 16; 17; 19) for a resulting EA with a parameter search space size of $1.741E12$.

The internal state of the planning agent can be fully represented by the random seed and its parameter settings (i.e. given parameters and random seed, the exact same behaviour would be achieved in multiple runs of the algorithm). Therefore, in order to preserve the internal state of the planning module, we need to store its parameter settings and random seed. In the case of updates being required, we would then be able to pause the system, disconnect the planning module, perform any updates and hook it back in with access to its previous parameters and seed. In order to avoid complications with changes in parameter space, the parameter space itself is built separately and modularly, so as new parameters may be added in without impacting any other part of the system.

Learning Module

Given the success of several works of combining planning methods with Neural Networks (NN), the learning part of our system would also take the form of a NN. However, there are two main aspects to consider here: the state representation and the network architecture. In GVGAI, the typical state representations used are: grid observations (NxM matrix with each cell representing one or multiple sprites at that location) (264; 265; 266) or compressed feature vectors extracted automatically from an image representation of the state with an Object Embedding Network (267). Various learning methods are used with different architectures, such as: Value Iteration Networks (266), Q-Learning (267) and Deep Reinforcement Learning (265; 268). We propose using an architecture similar to that of the successful AlphaGo (188), although the increased game complexity should be taken into account.

Due to the aims of general game playing, a limitation to consider is differences in games which make it harder (if not impossible) to apply a model learned in one game to another. In the simplest case, the agent would be told whether it is playing a different game, and learn different models per game. However, in order for the system to be autonomous, not relying on human information, as well as learn efficiently, another key aspect to consider for the learning module is transfer learning or extraction of key concepts which are generally applicable across games (i.e. walking into walls is generally not allowed) similar to the work of Narasimhan et al. in (266), for example.

To describe the internal state of the NN, we would need to save the generated model in order to be able to rerun the exact same instance of the algorithm.

Combining planning and learning. There are various ways in which we can combine the planning and learning approaches. Our proposed method extends from the AlphaGoZero (228) and p-RHEA (157) approaches in literature, as detailed below.

- **Initialisation:** We replace the uniform distribution used in the random population initialisation with external distributions provided by the NN. Starting from the current game state S_t , we query the NN for the action distribution π_t . This policy is followed using the Softmax function and the next state is simulated according to the action a_t selected from π_t , giving us S_{t+1} . The same process is repeated until we generate a full action sequence of the desired length for use within RHEA. We then generate the rest of the individuals in the initial population as mutations of the first.
- **Mutation:** We replace the uniform distribution used in selecting a new value for a gene g being mutated with an external distribution provided by the NN. Given the game state obtained by simulating through the sequence in the individual gene g belongs to, up until (and including) gene g , noted as S_g , the NN return the action distribution π_g . We modify π_g to set the weight of the current value of gene g to 0, and we perform weighted sampling from this distribution to obtain the new value for gene g (guaranteed different from previous).
- **Fitness:** We include in the individual fitness the external state value v_t provided by the NN, weighted by α (see Equation 8.1, where R_{value} is the rollout value obtained by RHEA individual evaluation and N_{value} is the NN value output for the final state reached through the rollout).

$$f = (1 - \alpha) \times R_{value} + \alpha \times N_{value} \quad (8.1)$$

Optimisation Module

A different line of work in improving the performance of game players (as opposed to relying on learning) is the optimisation of their parameters. Given the large parameter search spaces for both the RHEA and NN components of our system, it is highly unlikely that a human user would be able to select the perfect combination of parameters which would result in the highest performance, or most efficient learning. Therefore, we add an optimisation module to Thyia. Although several optimisation methods have been explored in literature and any could be integrated with our system, we choose to focus on the N-Tuple Bandit Evolutionary Algorithm (NTBEA) (31), which has been shown to perform well and robustly on various problems, in tuning game parameters (160) as well as game player parameters (133; 34; 161).

NTBEA is a model-based optimiser based on an Evolutionary Algorithm, which uses bandit-based sampling and detailed statistics on combinations of parameters in order to optimise hyper-parameters. It highlights fast convergence in noisy optimisation problems even with small computational budgets and it scales well for large search spaces (34), although it has yet to be tested on a search space as large as Thyia's.

The addition of this module does raise additional difficulties for the learning system: since the parameters of either the playing or learning algorithm would change, the data received by the learner could vary significantly in terms of the game player's behaviour. Therefore, the learner needs to be general enough to not make any assumptions in the data it receives, in order to be able to cope with high variations.

Human Interaction Module

Another important part of the vision for this system is the ability to interact with the “real world”. Ultimately, the goal of AGI is to bring benefits to humans in a multitude of real-world problems, thus the humans must be brought into the loop. There are several ways in which this could be achieved.

Direct interaction. One of the clearest cases would be a direct communication between the user and the AI, where the AI would use natural language processing to understand humans and reply to them intelligently (or, in the simplest case, building a database of keywords which trigger certain responses from the system). The purposes of this interaction form could vary from asking Thyia to play certain games, or asking for its thoughts on games its played. This leads to further extensions of AI able to form opinions supported by solid arguments or facts. Additionally, given its support for multi-player games, humans could play alongside the AI for another form of direct interaction.

Knowledge and statistics display. Depending on the specific chosen representation for Thyia’s knowledge, it may be hard to understand by humans: it could end up being an endless string of numbers which would mean little to us without the capabilities of fast computation. Therefore, visualising statistics about the games, the game player’s behaviour or the knowledge gathered would be interesting and useful to gain a better understanding about the system’s inner-workings.

Live streaming. A common way for human game players to interact with others is through live-streams (e.g. via popular platforms like Twitch and YouTube) and video sharing. There is a large community which revolves around the concept of sharing gameplay with others that watch and comment on the game being played, suggest strategies for the game or offer helpful information. In a similar way, Thyia could be streaming the games it plays to enter this community of human game players, while opening a direct communication channel through which it could even receive direct feedback for knowledge enhancement. Being part of the human society is a widely studied interesting challenge (269). Human streamers mainly attract audiences through their personality, thus attention should be given to Thyia’s audience interactions. Further studies will look more into how human streamers interact with their audience, to enhance Thyia’s abilities in this area (e.g. what information to present, what conversation it could be involved in etc.).

There is an important factor to take into account: the internet troll phenomenon. On the internet, many humans are generally inclined to give purposefully misleading information. When faced with an AI eager to learn from what the internet has to offer, we speculate these humans to be even more eager to fool our system. Therefore, content filtering, moderation and maintenance are necessary to ensure the system does not fall into traps. These will be further discussed in Section 8.1.6.

8.1.6 Ethical Implications

A large limitation of the project is its possible ethical implications. The fact that Thyia would be open to outside world interaction poses a problem. The ideal scenario is the interaction taking place in a controlled environment where it would be ensured that the tasks the AI is given to solve are not ethically questionable. However, as discussed at the end of Section 8.1.5, the internet is nowhere close to a controlled environment and humans interacting with the system could be supplying various malicious information:

- Suggestions for unethical strategies (e.g. destroying the human race before running to the finish line).
- Unethical game proposals: the games sent to the system to play could contain harmful content, hate speech or unethical themes such as killing a particular race.
- Malicious injections taking advantage of the natural language parser to generate unexpected and harmful behaviour (e.g. teaching the agent to reply in a harmful way to the humans interacting with the system).

There have already been cases of abuse towards interactive AI. A prime example is Tay, Microsoft’s chatter bot which was given open communication via Twitter, with the result being the Twitter community teaching the chatbot to become offensive and racist in only 16 hours (234). The benefit of such incidents is that subsequent attempts at general-public interaction include safety precautions against malicious intent. Moderation and content filtering are, therefore, very important to integrate within our human interaction module. One form of filtering for textual and speech-based content is sentiment analysis (270), which we would use to identify possibly harmful messages received by Thyia before she gets to process them and react accordingly. However, even though research in the area of textual sentiment analysis is plentiful, it is harder to apply the same tools for game content: how could one identify if a given game is unethical? We suggest this as an interesting path for further work.

8.2 Representation Types

The work in this section was published at IEEE CoG 2020:

D. Perez-Liebana, M. S. Alam, and R. D. Gaina, “Rolling Horizon NEAT for General Video Game Playing,” in *IEEE Conference on Games (CoG)*, 2020, pp. 375–382.

The work described in this thesis exclusively uses a single representation type for RHEA: an individual is a sequence of actions to play in the game. This is the classic form first introduced in (7) and the most direct interpretation for a game-playing agent, similar to the process in MCTS. However, this is not the only phenotypical translation possible. Several options that could be considered are as follows:

1. **Actions:** the classic form, where individuals represent sequences of actions. Very granular level of control, yet small mutations lead to large changes in actual agent behaviour in the game, especially when mutations happen towards the beginning of the sequence.
2. **High-level actions:** also referred to as macro actions or macro-goals, the individual could represent higher-level concepts (e.g. “kill enemy X” or “go to door”). This representation requires a low-level control method in charge of achieving the chosen goal, which could be another RHEA using the low-level action representation. This variant is more similar to human decision-making and could be extended to several levels of granularity (e.g. first choose between type of action, then specific object to target, then execute), while allowing for the larger problem to be split into smaller and more easily achievable tasks. This option could show great progress in very complex games, such as almost half of the GVGAI games which still remain at close to 0% win rate.
3. **Paths:** to address the second limitation of the actions representation, individual could represent paths for the agent to take within the level, with mutations adding or removing positions from the planned path (and connecting the remainders appropriately). Further, this concept could be abstracted to consider paths within a game state tree, leading into the area of Genetic Programming and making use of the vast research in that area to apply in this new domain.
4. **Weights for heuristics:** individuals could also encode weights for different functions, such as heuristics, or parameters for a (simple, to avoid computational cost inflation) agent. This approach would likely encounter the problem of working within continuous spaces, yet be more flexible in deciding high-level behaviours that should be executed at a point in time.
5. **Weights for a neural network:** extending from the previous point and taking advantage of recent popularity of neural networks for learning to play games, the weights used by the network could be encoded as individuals in an evolutionary algorithm, using then the network to create action sequences, evaluate these as in the classic RHEA and use the resultant value as the fitness associated to the weights tested. This option would lead to more complex and flexible decision-making, at a similar computation cost to the classic actions representation.
6. **Combination:** lastly, we acknowledge that it could very well be that the ideal and best performing method to be a combination of several representations. For example, half of the individual could represent action sequences, while the other half could be weights for the heuristic function used to evaluate the game state reached through the action sequence. These methods would benefit from high adaptability and better control, with the potential drawback of needing more time to figure out good solutions.

Exploring all of these options in-depth would require a significant amount of work. In the rest of this section, we describe work that explores option 5 by combining RHEA with NeuroEvolution of Augmenting Topologies (rhNEAT) (25): the algorithm now evolves weights for the neural network (NEAT), uses NEAT to generate action sequences and evaluates these by rolling them out with the forward model and assessing the game state reached with a heuristic function h ; the value returned is used as the fitness of the individual (or weight for the neural network) and the best NEAT configuration is chosen at the end of the decision-making process to generate one action to play in the game. Different variations of the algorithm are explored within the context of the same 20 GVGAI games.

NeuroEvolution (NE) is a field concerned with evolving the weights and configurations for a neural network (NN), concepts which have been used extensively in games (271). Early work in the area looked at evolving the topologies of NNs (272). However, Stanley and Miikkulainen proposed evolving both the weights and the topology of NNS through a direct encoding (a binary vector indicating if a connection between 2 neurons exists or not). This has become one of the most popular systems: NeuroEvolution of Augmented Topologies (NEAT) (273). This work was later extended for several domains, such as

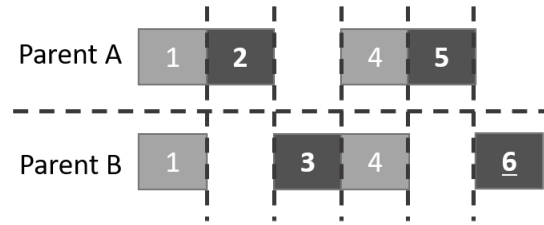


Figure 8.2: Two parents lined up for crossover according to their innovation numbers. Dark genes are *disjoint*, while the dark cell with its innovation number underlined is *excess*.

robotics (274) or checkers (275). Yet more relevant to this study, NEAT has also seen several applications the general (video) game playing context. Reisenger et al. used NEAT to evolve a population of game state heuristic functions in the General Game Playing competition with great success (276), allowing for less human-designed domain knowledge embedded in the system, and more accurate opponent modelling. Hausknecht et al. used NEAT in a similar way, to extract game objects from raw screen input and approximate their values based on the game score achieved (277; 278); using on-screen objects and their associated value as the input to the evolved NN produces best action recommendations, the overall approach achieving high performance on several Atari games. In GVGAI, Samothrakis et al. (279) test several combinations of Separable Natural Evolution Strategies evolving function approximators and neural networks and show their abilities of learning to play several games efficiently. These ideas are taken forward in integrating NEAT within RHEA for online decision-making.

8.2.1 rhNEAT

NEAT starts with the simplest network (all input and output nodes, and an empty list of connections), which becomes incrementally more complex through evolution, in order to find more efficient solutions to the given problem. At each generation, it keeps a population of P individuals, which is sorted based on individual fitness. It then uses tournament selection to choose parents and uses crossover, then mutation, to obtain offspring. Each one is evaluated and a percentage R of the lowest-fitness individuals are discarded for the next generation. The process repeats within the allocated budget and the action returned by the best individual at the end of a game tick is played in the game.

A form of shift buffer is available for this algorithm as well, by keeping the previously evolved population from one game tick to the next; in this section (as no shifting actually occurs and the population is unchanged), we refer to this process as *population carrying*. All parameters of rhNEAT are described in Table 8.1. The rest of this subsection details several key concepts, both for NEAT in general and the specific implementation for playing GVGAI games.

Genome: each genome (individual) is made up of 2 lists: one list of node genes (input, hidden, and output nodes), and one list of connections between the nodes (specifying in-node, out-node, weight, innovation number and a boolean variable indicating if the connection is enabled or not). Crossover specifically looks at the list of connection genes, adjusting the list of node genes as needed according to the result.

Innovation numbers: index of a connection gene, according to the step where it appeared in the evolution process (280).

Crossover: the genes in the chosen parents are lined up according to their innovation numbers; genes *match* if they have the same innovation number; if they do not match, but are within the range of the other parent's innovation numbers, they are labelled as *disjoint*; if they do not match and are outside of the range, they are considered *excess*; see an example in Figure 8.2. Disjoint and excess genes are only taken from the fitter parent (or from both if they are of equal fitness). The matching genes are either chosen uniformly at random from either parent (uniform crossover), or both are kept and their weights are averaged (blended crossover (281)).

Mutation: there are several mutation options possible, each chosen with some probability:

- *Add connection* (μ_t): adds a new connection gene to the list, between two existing nodes.
- *Add node* (μ_n): adds a new node gene C to the node gene list; the connection list is updated so that the newly added node splits an already existing connection between nodes A and B into two, with the new node C in the middle. The original connection ($A \rightarrow B$) becomes disabled. The new connections receive innovation numbers increased by 1 each from the current highest in the algorithm and associated weights ($A \rightarrow C$ gets weight 1.0 and $C \rightarrow B$ gets the weight of the original connection $A \rightarrow B$).

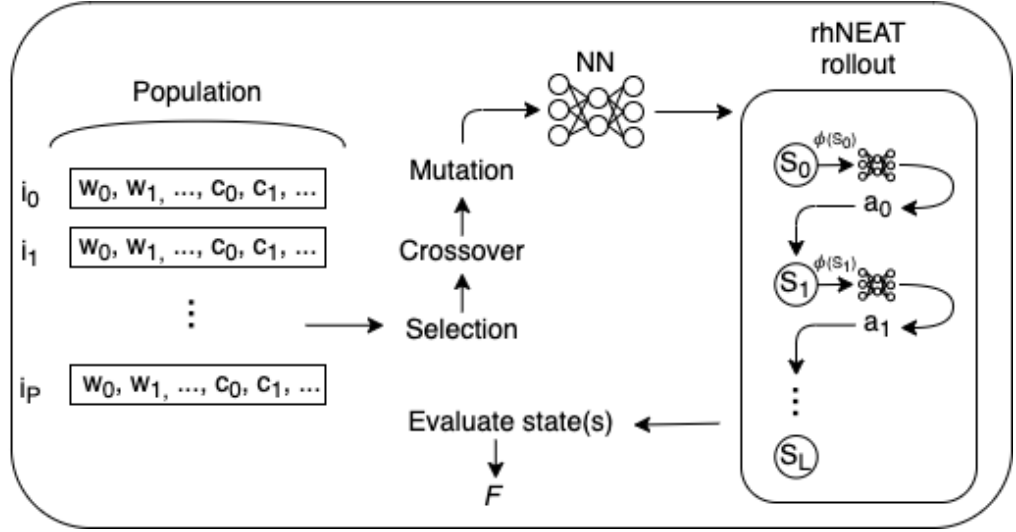


Figure 8.3: Summary of an individual evaluation.

- *Connection weight shift* (μ_{ws}): adjusts the weight of a randomly chosen connection by a value picked randomly in range $[-W_s, W_s]$.
- *Connection weight change* (μ_{wr}): replaces the weight of a randomly chosen connection by a value picked uniformly at random, in range $[-W_r, W_r]$.
- *Connection toggle* (μ_{tl}): toggles the enabled/disabled property of a randomly chosen connection.

Speciation: individuals are grouped into *species* based on their topology complexity and they only compete against other individuals in the same species. To separate individuals from a new generation into species, first a representative from each species in the previous generation is chosen uniformly at random. A distance function is then used to measure the distance of each other individual to each representative, and they are clustered based on the smallest resultant difference. The distance function used is described in Equation 8.2. In this equation, N is the total number of genes in both individuals (normalised to 1 if less than 20), $Excess$ is the number of excess genes, $Disjoint$ is the number of disjoint genes, \bar{W} is the average weight difference between matching genes (273) and c_1, c_2 and c_3 are constants which can be tuned to change the impact of each variable in this linear combination. A maximum distance threshold CP is used, and if an individual is not close enough to any species representative, then it forms a new species.

$$\delta = \frac{c_1 Excess}{N} + \frac{c_2 Disjoint}{N} + c_3 \cdot \bar{W} \quad (8.2)$$

Small networks are generally faster to optimise and obtain good solutions quickly. However, the nature of NEAT results in more and more complex networks. This can lead to newer generations seeing initially lower fitness and a smaller chance of survival throughout generations if compared to the whole population, thus the need for speciation arises. When a species loses all individuals from one generation to the next, that species is removed from the algorithm.

NN Input: the input to each NN encoded in an individual is a series of game state features:

- Avatar x, y position, normalised in $[0, 1]$.
- Avatar x, y orientation.
- Avatar health points, normalised in $[0, 1]$, using 0 as minimum and M_{hp} as the maximum health points achievable in each game.
- Up to three resource types r_1, r_2, r_3 gathered by the avatar, where each r_i is normalised in $[0, 20]$.
- Distance d and orientation o to the closest instance of a sprite in all categories distinguished in GV-GAI: NPCs, immovables, movables, resources, portals, sprites produced by the avatar. Distances are normalised in $[0, 1]$, using 0 as the minimum and M_d as the maximum possible distance in a game level. Orientation is normalised in $[-1, 1]$, where 1 represents that the distance vector to the sprite is aligned with the avatar's orientation and -1 the opposite.

Table 8.1: rhNEAT parameter values.

Parameter	Name	Value
P	Population Size	10
L	Rollout length	15
R	Individuals discarded per generation	20%
CP	Speciation Threshold	4
c_1	Excess coefficient	1.0
c_2	Disjoint coefficient	1.0
c_3	Weight difference coefficient	1.0
μ_l	Mutate Link Probability	0.5
μ_n	Mutate Node Probability	0.3
μ_{ws}	Mutate Weight Shift Probability	0.5
W_s	Weight Shift Strength	0.4
μ_{wr}	Mutate Weight Random Probability	0.6
W_r	Weight Random Strength	1.0
μ_{tl}	Mutate Toggle Link Probability	0.05
FM_b	Forward Model calls budget	1000

We note that due to the different nature of GVGAI games, not all include all of the features described here (e.g. health points, resources or sprites in specific categories are only used in some games). In these cases, those inputs are not included for the network, to reduce its size and encourage faster optimisation. If some of these features appear later on in the game and the population carrying is enabled in the algorithm, then the population is reinitialised for the game tick.

NN Output: distribution over actions available in the game.

Evaluation: to evaluate an individual, the NN it encodes is constructed and used to roll the current game state forward for L steps. At each step, input features are extracted from the game state, fed into the network and the action with the highest probability is extracted from the output. This action is applied in the state using the forward model, obtaining a new game state to repeat the process in (see Figure 8.3). The heuristic function h in Equation 8.3 can be used to evaluate any of the game states traversed during the rollout, and these values combined to obtain the fitness of the individual. Several configurations are tested in the experiments presented here.

$$h(s) = \begin{cases} 10^6 & \text{win} \\ -10^6 & \text{lose} \\ \text{game score} & \text{otherwise} \end{cases} \quad (8.3)$$

8.2.2 Experiments

For the experiments, each algorithm configuration is run 100 times on each of the 20 games, 20 repetitions for each of the 5 levels. The budget given is 1000 FM calls and each configuration uses a population size of 10 individuals and a rollout length of 15. The rest of the parameters are set as in Table 8.1.

The configurations tested include enabling/disabling speciation and population carrying and different options for fitness calculations. Finally, the best overall rhNEAT agent is compared to RHEA and MCTS. All code and results are available on Github⁴.

rhNEAT Additions

The first set of experiments looks at how different enhancements (speciation and population carrying) affects the performance of the algorithm, compared with its vanilla version (referred to as *baseline rhNEAT*). The variant using population carrying is referred to as *rhNEAT(+cp)*, and *rhNEAT(+sp)* is the variant using speciation. *rhNEAT(+sp,+cp)* combines both enhancements. All algorithms in these experiments use only the value of the final game state reached at the end of a rollout as the fitness of individuals (the default in RHEA).

The first row group in Table 8.2 shows an increase in both win rate and score as more enhancements are added, with a higher increase for population carrying compared to the baseline. This suggests that keeping the population evolved between game ticks is very important, to make the most efficient use of the short time

⁴<https://github.com/GAIGResearch/rhNEAT/>

Table 8.2: Summary of results showing, for each approach, average win rate (and standard error) in the 20 games, the number of games it achieved the highest positive win rate in the subset (including the absolute highest count, i.e. no ties), and the number of games in which it achieved the highest score.

	Algorithm	Win Rate (Std Err)	Highest positive win rates (absolute)	Highest score
Additions	baseline rhNEAT	15.54% (6.13)	0 (0)	0
	rhNEAT(+sp)	22.69% (7.01)	2 (1)	2
	rhNEAT(+cp)	30.45% (7.93)	2 (2)	6
	rhNEAT(+sp,+cp)	36.50% (8.52)	13 (12)	12
Fit.C.	rhNEAT	36.50% (8.52)	8 (6)	12
	rhNEAT-acc	35.15% (8.12)	9 (7)	5
	rhNEAT-accdisc	34.20% (8.14)	3 (2)	4
Fit.A.	rhNEAT	36.50% (8.52)	10 (8)	11
	rhNEAT-lr	35.25% (8.72)	5 (4)	3
	rhNEAT-avg	31.55% (8.10)	5 (2)	6
SFP	rhNEAT	36.50% (8.52)	3 (2)	1
	RHEA	44.80% (8.89)	2 (0)	0
	MCTS	42.65% (9.56)	6 (3)	4
	SotA	51.21% (8.72)	13 (10)	16

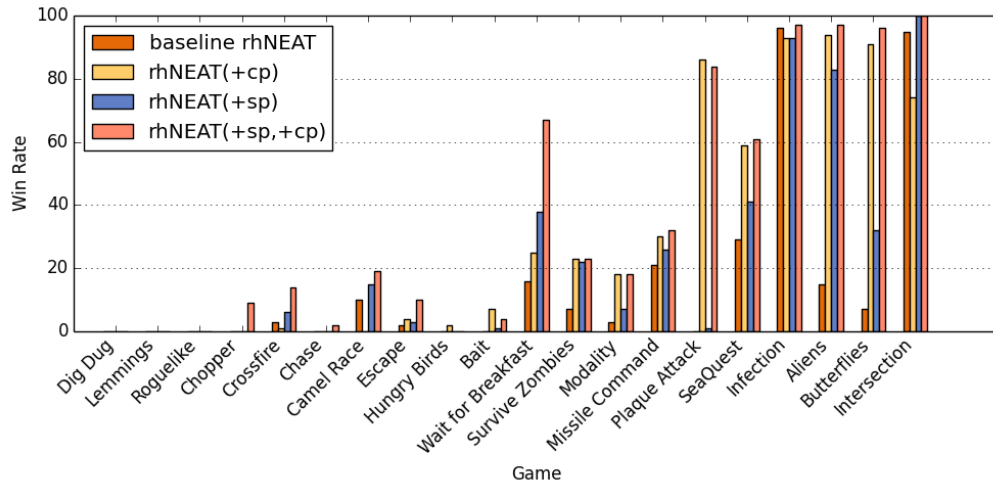


Figure 8.4: Win rates per game of different rhNEAT variants: baseline rhNEAT, with population carrying (+cp), speciation (+sp) and with both (+cp,+sp).

given for decision-making - especially as, with this representation, the population remains highly relevant throughout the game and the networks evolved keep getting better over time. We do note that resetting the population is important if the features present in the game change, or the game state changes drastically (e.g. as in complex games, where multiple levels with different mechanics could be combined into one play session). We see the baseline method performing quite poorly, at 15.54% win rate, but both enhancements added results in a boost in performance to 36.5%, obtaining the highest win rate in 13 out of 20 games and the highest score in 12.

In Figure 8.4 we detail the results per game for all configurations tested here. We can see again that *rhNEAT(+sp,+cp)* achieves the highest win rate in most cases. However, it is worth highlighting that performance varies largely between the different games, with some seeing win rates of or close to 100% (“Infection”, “Aliens”, “Butterflies”, “Intersection”), while others remain at 0% (“Dig Dug”, “Lemmings”, “Roguelike”). Another interesting game to showcase is “Plaque Attack”, in which speciation is actually highly detrimental to the agent’s performance, its win rate (as well as that of the baseline algorithm) being close to 0%; however, adding population carrying raises the win rate to almost 90%. We see this as a clear example of different parameters being beneficial in different games, and optimisation similar to that presented in Chapter 6 could be employed here as well to automatically find good settings for the variety of games.

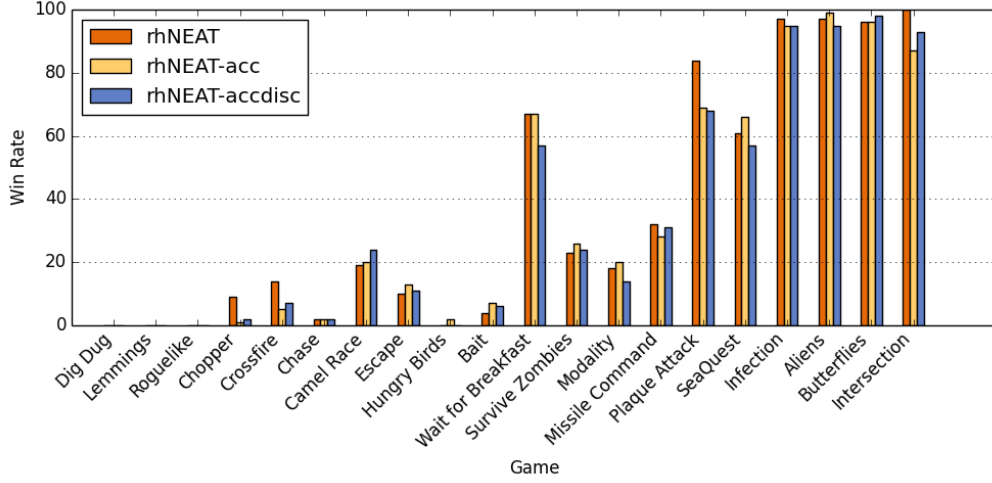


Figure 8.5: Win rates per game of different rhNEAT variants: calculating the fitness as the value of the last state reached through the rollout (rhNEAT), as the sum of the values of all states visited (-acc) or as the discounted sum of the values of all states visited (-accdisc).

rhNEAT Fitness Calculation

Next, we take a look at how the fitness is calculated during the evaluation process of one individual. Here, *rhNeat* uses only the value of the last state reached at the end of the rollout; *rhNEAT-acc* uses a sum of the values of all game states visited during the rollout. *rhNEAT-accdisc* uses a discounted sum of the values of all game states visited during the rollout, with discount factor $\gamma = 0.9$, meant to differentiate between short and long sighted versions of the algorithm. All variants tested here use the best option from the previous set of experiments, rhNEAT(+cp,+sp).

Results in the second row group in Table 8.2 show very similar performance between all variations here, with *rhNEAT* achieving the best win rate and score. However, performance is not affected significantly by whether only the final state is considered, or all in-between. Figure 8.5 presents again a detailed comparison on each of the games. Here we observe *rhNEAT-accdisc* to achieve worse results on most games, except for “Camel Race” and “Butterflies”. These are fairly sparse games with rewards for the agent scattered across larger levels, where a discounted sum of rewards gives better focus to the agent and allows it to explore the space more efficiently.

rhNEAT Fitness Assignment

Lastly, we look at how the fitness calculated (kept as the value of the final state reached from the previous experiment) should be assigned to an individual, based on the assumption that the same individual is going to be evaluated multiple times throughout the course of a game. *rhNEAT* replaces the old fitness value of an individual with a new one. *rhNEAT-avg* assigns the arithmetic average of all rewards seen by the individual, to consider all of its past experiences. *rhNEAT-lr* uses a learning rate ($\alpha = 0.2$) to adjust the individual’s fitness with the new value obtained, as $F = F + \alpha \times (f_i - F)$ (where f_i is the new fitness and F is the old fitness), in order to consider all past experiences, but give more weight to more recent ones.

The third row group in Table 8.2 shows a very similar performance between *rhNEAT* and *rhNEAT-lr*, with *rhNEAT* being slightly better. However, considering all past experiences equally (*rhNEAT-avg*) leads to a drop in performance to 31.55% win rate; this suggests that games do differ in various game ticks, enough that it is often most beneficial to only consider the most recent evaluations when choosing between individuals. Further, these results highlight the baseline algorithm’s ability to adapt well to changing scenarios within a single play session.

In Figure 8.6 we observe the split of win rates per game for this set of experiments. Here it is interesting to note that *rhNEAT-lr* does outperform the baseline in “Survive Zombies” and “Seaquest”, games with very dynamic, stochastic and busy environments, in which using the learning rate to slowly adjust the individual fitness is a more robust approach.

SFP Comparison

Lastly, we discuss how rhNEAT compares in the larger GVGP context. To this extent, we show its performance against RHEA and MCTS; both RHEA and MCTS use the same population size, individual/rollout

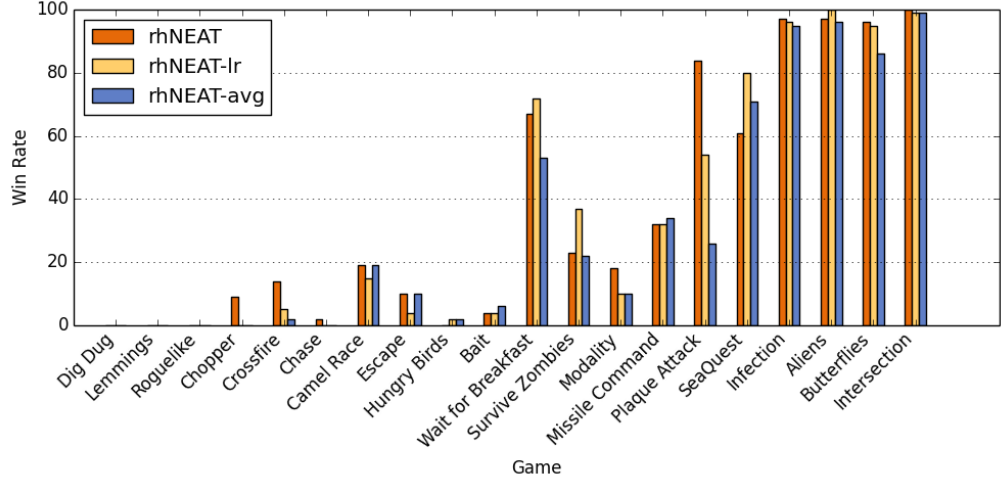


Figure 8.6: Win rates per game of different rhNEAT variants: individual fitness assigned as the last evaluation only (rhNEAT), averaged for all evaluations (-avg) or adjusted with a learning rate (-lr).

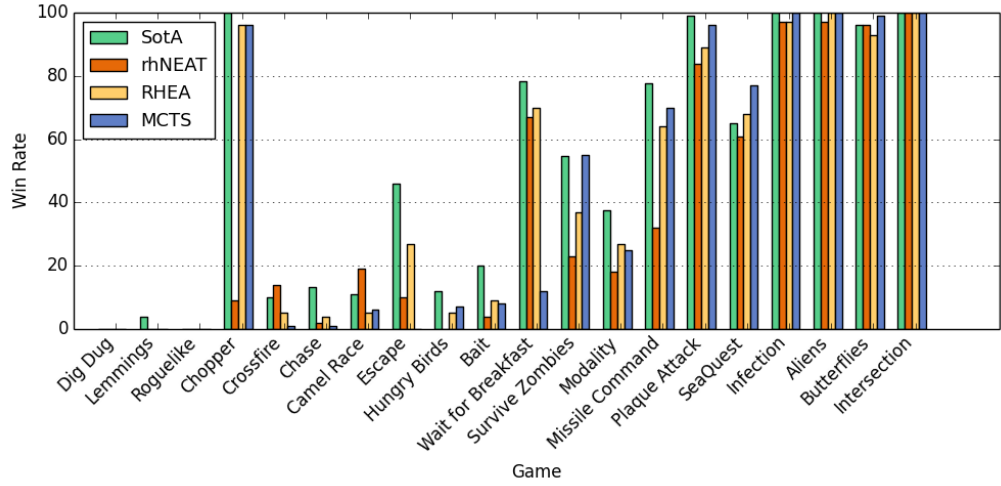


Figure 8.7: Win rates per game for rhNEAT, MCTS, RHEA and RHEA state-of-the-art (SotA).

length, budget and state evaluation as described in this section for rhNEAT for a fair comparison (all other parameters not specified are default as described previously in this thesis in Chapter 3 and Section 2.2, respectively). We further add into the comparison state-of-the-art (*SotA*) results, as described in (21).

The last row group in Table 8.2 show rhNEAT to achieve an overall lower win rate than the other algorithms, although it outperforms them in 2 out of the 20 games. Taking a closer look at the division per game, in Figure 8.7 shows these games to be “Crossfire” and “Camel Race”, in which most algorithms which do not use custom heuristics better informing them of which actions are favourable do not perform very well. Both games also feature sparse rewards and thus posing an extra challenge to general game players (19), making the performance achieved by rhNEAT noteworthy.

8.3 Physics-Based Games

The work in this section was published at IEEE CIG 2017:

D. Perez-Liebana, M. Stephenson, R. D. Gaina, J. Renz, and S. M. Lucas, “Introducing Real World Physics and Macro-Actions to General Video Game AI,” in *Proceedings of IEEE Conference on Computational Intelligence and Games*, Aug 2017, pp. 248–255.

The work presented in this chapter so far focuses on fairly simple, arcade-style 2D grid-based games, as has been the focus of the General Video Game AI framework. Although the games (and the subset used predominantly throughout this thesis) include many different features (different reward systems, win conditions, mechanics etc.) and genres (puzzles, shooters, navigation etc.).

This section expands the domain we are concerned with to games using physics simulations to cover a wider range of mechanics available, as well as increasing the challenge for the AI players: previously, when applying an action, the agents would see (in most cases) an immediate effect, e.g. moving one tile to the left. In games affected by physics, this is much more granular: a movement to the left could mean rotating a sprite only 1 pixel, depending on the force applied and any other forces acting on the object at the same time, for example (or even moving the sprite downwards instead due to gravity!). To put this idea in perspective, the grid-based implementation of “Aliens” sees sprites moving consistently based on their speed (even if this is below 1, so moving less than a tile at a time and giving the impression of more precise movement, this never varies); in a physics-based implementation, the distance moved would depend on the state of the sprite and the environment, i.e. what other forces are acting on the sprite, its current speed, acceleration etc. The level space that the agents have to explore is therefore much larger in these games. However, these games also offer a more realistic simulation of real-life scenarios (282), allowing games research to be more impactful across several other areas, such as robotics and engineering.

The first part of this section will describe additions made to the General Video Game AI framework to support such complex games and the specific game set explored in the experiments, which are the first 10 physics-based games implemented with the new engine. These will be referred to as *Continuous* or *Real-World* physics-based games, in contrast with the games implemented in the 2D grid system.

The second part of this section will detail experiments testing Rolling Horizon Evolutionary Algorithms on the new set of games and comparing its performance to Monte Carlo Tree Search. Both agents receive the same enhancement to be able to handle the larger search space better, a *repeat* frame-skip (where each chosen action is repeated several times). The effect of different lookahead lengths is also tested and analysed in the context of the budget - information gain trade-off.

This type of frame-skip in physics-based games is meant to, in a way, discretise and reduce the search space for the agents, relying on an approximation that an action repeated N times in continuous physics would produce a similar result to the action executed once in grid-based physics. This also increases the lookahead of the agents, as they would now execute N times as many actions in their simulations to account for each one being repeated. Additionally, it also artificially inflates the agent’s thinking time: since an action is repeated N times after it is chosen (from time tick t to $t + N$), the agent will only need to return a new action to play every N game ticks (at game tick $t + N + 1$). This means that the agent has N game ticks to decide the action it is going to take next, and N times as big a budget to do so. This is not a straightforward benefit, as the game state could change unpredictably (in stochastic games) between 2 decision points; yet we explore if the approximations obtained and the longer decision-making time is enough to make up for the potential inaccuracies. This turned out to be the case in previous research, where RHEA saw a large boost in performance in the Physical Travelling Salesman Problem (7).

8.3.1 Real-World Physics Games in GVGAI

The GVGAI framework was enhanced for this work (with modifications publicly available in the open-source engine), allowing for further customisation in the game description files. Sprites can be assigned to use the new continuous physics and specify 3 new properties as a result: *mass*, *friction* and *gravity*. Typically, the *gravity* property will be the same across all sprites in a game (indicating the gravity force affecting them, which one would expect to be the same), but it can differ between sprites for interesting effects and gameplay.

Several forces are therefore defined to act on each sprite at any one time, and their movement is calculated accordingly, instead of being restricted to the previous 4 directions (up, down, left, right). Each sprite has instead a direction unit vector (set in screen pixels) and a speed, making movement and trajectories more flexible and natural. The resultant velocity vector is combined with any other forces acting on the sprite, and then used to update the sprite’s position at every game tick. Inertia naturally arises due to the addition of forces, factored by the object’s mass and friction.

Table 8.3: Physics game set including feature analysis. Symbols as used previously in Section 2.1.2. Actions key: **Rot** = rotate; **Move** = move; **LR** = left and right; **UD** = up and down; **Shoot** = create a missile sprite with the same direction as the avatar and a particular speed; **AD** = accelerate and decelerate; **Jump** = add force in up direction.

Idx	Game	Stoch.	Rewards	Win	Lose	Levels	NPCs	Res.	Actions
0	Artillery		D	Kill	Death	M/Dense	E		Rot:LR+Shoot
1	Asteroids		D	Kill	Death	M/Sparse			Rot:LR+AD+Shoot
2	Bird		D	Exit	Death	L/Sparse			Jump
3	Bubble		D	Kill	Death	M/Sparse	E		Move:LR+Shoot
4	Candy		D	Exit	Death	M/Dense	E		Move:LR+Jump+Shoot
5	Lander		N	Exit	Death	M/Sparse			Rot:LR+AD
6	Mario		D	Exit	Death	L/Dense	E		Move:LR+Jump
7	Pong		N	Kill	Kill	M/Sparse			Move:UD
8	PTSP	x	N	Kill	Death	L/Dense	E		Move:UDLR
9	Racing		D	Kill	Death	L/Sparse			Rot:LR

Several new avatar types and new interactions can be defined as well, setting the possible actions for the player and the resulting forces they would create on the object they control. We refer the reader to (26) for full details on the new framework modifications.

Table 8.3 presents a summary of the features of the physics games tested in the experiments described here. Most of the games are deterministic (except “PTSP”) and include dense rewards, with a variety of actions available to the player and different types of interactions. Full descriptions of the games can be found in Appendix C.

8.3.2 Experiments

An extensive experimental work has been put in place to study the performance of the algorithm in the games detailed in Table 8.3.

These algorithms are MCTS and RHEA, although a deeper study has been performed in the latter case, varying the size of the population for the EA. The population sizes P tried are 1, 5 and 10, and results are reported for these agents as RHEA-1, RHEA-5 and RHEA-10, respectively.

Different lookaheads have been explored for both RHEA and MCTS, aiming to explore how longer rollouts affect the victory rates on these games. The lookahead values employed are 30, 60, 90 and 120 steps from the current game state. These lookahead values are reached by a combination of frame-skip length M and simulation depth L (or individual length for the RHEA agents), with different configurations for both values:

- $L \times M = 10$; $(L, M) = (10, 1)$.
- $L \times M = 30$, with: $(30, 1)$,⁵ $(5, 6)$, $(3, 10)$, $(2, 15)$.
- $L \times M = 60$, with: $(60, 1)$, $(6, 10)$, $(4, 15)$, $(2, 30)$.
- $L \times M = 90$, with: $(90, 1)$, $(9, 10)$, $(6, 15)$, $(3, 30)$.
- $L \times M = 120$, with: $(120, 1)$, $(12, 10)$, $(8, 15)$, $(4, 30)$.

For RHEA, we further test different population sizes P : 1, 5 and 10; results refer to these agent variations as RHEA-1, RHEA-5 and RHEA-10, respectively. Additionally, an extra configuration has been run for all agents, where no frame-skip and a simulation depth of 10 has been used. This is a useful setting for comparisons, as it is the default depth of the controllers as provided in the GVGAI framework.

Each one of these configurations is tested in 100 repetitions of each game: 20 repetitions of the 5 levels in the 10 available games. A budget of 900 forward model calls is assigned for each game tick.

8.3.3 Results and Discussion

Overall Victory Rates

Table 8.4 shows the results with the default depth of 10 single actions. The last row of the table (average victory rate over 100 games played, plus standard error between brackets) highlights that MCTS appears to perform better than RHEA in this setting. However, higher population sizes help RHEA achieve a better

⁵Note that the first configuration per lookahead is equivalent to no frame-skip, but longer individual length / simulation depth.

Table 8.4: Results of the controllers with their default $L = 10$.

Game	RHEA-1	RHEA-5	RHEA-10	MCTS
Artillery	39.00 (4.88)	35.00 (4.77)	32.00 (4.66)	41.00 (4.92)
Asteroids	3.00 (1.71)	10.00 (3.00)	17.00 (3.76)	31.00 (4.62)
Bird	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
Bubble	23.00 (4.21)	53.00 (4.99)	72.00 (4.49)	77.00 (4.21)
Candy	3.00 (1.71)	2.00 (1.40)	2.00 (1.40)	4.00 (1.96)
Lander	0.00 (0.00)	2.00 (1.40)	3.00 (1.71)	8.00 (2.71)
Mario	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
Pong	69.00 (4.62)	68.00 (4.66)	63.00 (4.83)	67.00 (4.70)
PTSP	1.00 (0.99)	4.00 (1.96)	3.00 (1.71)	4.00 (1.96)
Racing	30.00 (4.58)	52.00 (5.00)	55.00 (4.97)	66.00 (4.74)
Total	16.80 (2.27)	22.60 (2.72)	24.70 (2.75)	29.80 (2.98)

Table 8.5: Results for all algorithms and configurations. Indicated values are the average of victories across all games, with the standard error between brackets. Results in bold mark the best performances for RHEA and MCTS.

L × M:	30 × 1	5 × 6	3 × 10	2 × 15
RHEA-1	14.50 (2.10)	21.80 (3.29)	19.60 (3.16)	14.60 (2.46)
RHEA-5	31.60 (3.12)	32.90 (4.18)	22.40 (3.51)	14.70 (2.62)
RHEA-10	36.50 (3.15)	28.80 (4.06)	20.40 (3.43)	14.10 (2.67)
MCTS	46.00 (3.02)	48.40 (4.28)	41.30 (3.72)	20.50 (2.74)
L × M:	60 × 1	6 × 10	4 × 15	2 × 30
RHEA-1	14.10 (1.93)	22.70 (3.30)	19.10 (2.88)	11.80 (2.15)
RHEA-5	33.40 (3.05)	31.50 (4.08)	27.30 (3.74)	12.20 (2.42)
RHEA-10	39.50 (3.10)	24.90 (3.73)	23.00 (3.62)	10.00 (2.23)
MCTS	46.90 (3.01)	46.40 (4.40)	26.70 (2.60)	5.10 (1.45)
L × M:	90 × 1	9 × 10	6 × 15	3 × 30
RHEA-1	12.00 (2.11)	19.40 (3.09)	17.70 (2.88)	14.00 (2.31)
RHEA-5	32.80 (3.06)	30.90 (4.04)	28.70 (3.76)	17.70 (2.74)
RHEA-10	37.30 (3.10)	25.70 (3.89)	24.60 (3.70)	13.70 (2.52)
MCTS	46.90 (3.08)	45.80 (4.56)	25.90 (3.28)	9.00 (1.82)
L × M:	120 × 1	12 × 10	8 × 15	4 × 30
RHEA-1	14.10 (2.13)	20.50 (3.18)	17.10 (2.79)	12.80 (2.14)
RHEA-5	33.40 (3.07)	30.50 (3.92)	27.30 (3.64)	20.00 (2.74)
RHEA-10	36.00 (3.03)	25.10 (3.69)	24.20 (3.53)	15.90 (2.64)
MCTS	44.40 (3.14)	48.40 (4.55)	24.40 (3.15)	11.00 (2.15)

performance, very close to MCTS in $P = 10$. Furthermore, these results offer a glimpse into the wide variety of challenges attacked through the game set presented here. Most of the games are dominated by MCTS, although in some games the $P = 10$ version of RHEA achieves a similar performance. In some games, such as “Bird”, “Lander”, “Mario” and “PTSP”, we observe a very low victory rate and even 0% for all configurations.

Table 8.5 shows the results of all configurations averaged across games, for all the runs. The first conclusion quickly observed is that the results with $M = 1$ are far better than their counterparts. This is true for most cases, and, in fact, there is a regular trend that can be observed where victory rates drop as the frame-skip length increases.

There are other interesting remarks to make about these results, especially compared to those presented in Table 8.4. In general, it can be observed that RHEA usually sees a higher performance when small frame-skip is added (second column of Table 8.5); however, improvement decreases as M gets higher, suggesting this algorithm to need a more granular search in order to maximise its performance. Regarding the change in population size for RHEA, it can be seen that performance typically increases when P is higher (which was also noted in (15)), but only if the frame-skip length is short. Larger populations with more frame-skip reduce the performance of the algorithm to its worst results in this comparison. This strengthens the

Table 8.6: Results per game in the configuration $L \times M = 30$. Averages and standard errors of the measures indicated in bold when better than the other variants (*italics* where the best result is shared).

Algorithm:	RHEA-1				RHEA-5			
L × M:	30 × 1	6 × 5	3 × 10	2 × 15	30 × 1	6 × 5	3 × 10	2 × 15
Artillery	34.00 (4.74)	12.00 (3.25)	3.00 (1.71)	3.00 (1.71)	52.00 (5.00)	11.00 (3.13)	3.00 (1.71)	6.00 (2.37)
Asteroids	0.00 (0.00)	3.00 (1.71)	2.00 (1.40)	0.00 (0.00)	36.00 (4.80)	19.00 (3.92)	0.00 (0.00)	0.00 (0.00)
Bird	0.00 (0.00)	22.00 (4.14)	30.00 (4.58)	15.00 (3.57)	0.00 (0.00)	45.00 (4.97)	22.00 (4.14)	13.00 (3.36)
Bubble	18.00 (3.84)	37.00 (4.83)	27.00 (4.44)	13.00 (3.36)	68.00 (4.66)	45.00 (4.97)	35.00 (4.77)	14.00 (3.47)
Candy	1.00 (0.99)	4.00 (1.96)	0.00 (0.00)	0.00 (0.00)	6.00 (2.37)	22.00 (4.14)	7.00 (2.55)	0.00 (0.00)
Lander	1.00 (0.99)	7.00 (2.55)	7.00 (2.55)	0.00 (0.00)	4.00 (1.96)	29.00 (4.54)	16.00 (3.67)	0.00 (0.00)
Mario	0.00 (0.00)	0.00 (0.00)	5.00 (2.18)	4.00 (1.96)	0.00 (0.00)	2.00 (1.40)	16.00 (3.67)	8.00 (2.71)
Pong	60.00 (4.90)	61.00 (4.88)	42.00 (4.94)	52.00 (5.00)	79.00 (4.07)	65.00 (4.77)	45.00 (4.97)	35.00 (4.77)
PTSP	1.00 (0.99)	36.00 (4.80)	42.00 (4.94)	35.00 (4.77)	13.00 (3.36)	50.00 (5.00)	48.00 (5.00)	42.00 (4.94)
Racing	30.00 (4.58)	36.00 (4.80)	38.00 (4.85)	24.00 (4.27)	58.00 (4.94)	41.00 (4.92)	32.00 (4.66)	29.00 (4.54)
Algorithm:	RHEA-10				MCTS			
L × M:	30 × 1	6 × 5	3 × 10	2 × 15	30 × 1	6 × 5	3 × 10	2 × 15
Artillery	57.00 (4.95)	13.00 (3.36)	9.00 (2.86)	3.00 (1.71)	51.00 (5.00)	33.00 (4.70)	4.00 (1.96)	0.00 (0.00)
Asteroids	48.00 (5.00)	12.00 (3.25)	1.00 (0.99)	1.00 (0.99)	85.00 (3.57)	36.00 (4.80)	24.00 (4.27)	4.00 (1.96)
Bird	0.00 (0.00)	36.00 (4.80)	22.00 (4.14)	8.00 (2.71)	0.00 (0.00)	30.00 (4.58)	27.00 (4.44)	20.00 (4.00)
Bubble	79.00 (4.07)	28.00 (4.49)	17.00 (3.76)	12.00 (3.25)	97.00 (1.71)	50.00 (5.00)	87.00 (3.36)	38.00 (4.85)
Candy	6.00 (2.37)	17.00 (3.76)	5.00 (2.18)	1.00 (0.99)	9.00 (2.86)	56.00 (4.96)	8.00 (2.71)	1.00 (0.99)
Lander	10.00 (3.00)	17.00 (3.76)	9.00 (2.86)	0.00 (0.00)	49.00 (5.00)	58.00 (4.94)	35.00 (4.77)	0.00 (0.00)
Mario	0.00 (0.00)	8.00 (2.71)	9.00 (2.86)	8.00 (2.71)	0.00 (0.00)	18.00 (3.84)	14.00 (3.47)	1.00 (0.99)
Pong	80.00 (4.00)	69.00 (4.62)	52.00 (5.00)	35.00 (4.77)	80.00 (4.00)	100.00 (0.00)	94.00 (2.37)	62.00 (4.85)
PTSP	15.00 (3.57)	48.00 (5.00)	47.00 (4.99)	40.00 (4.90)	16.00 (3.67)	47.00 (4.99)	61.00 (4.88)	36.00 (4.80)
Racing	70.00 (4.58)	40.00 (4.90)	33.00 (4.70)	33.00 (4.70)	73.00 (4.44)	56.00 (4.96)	59.00 (4.92)	43.00 (4.95)

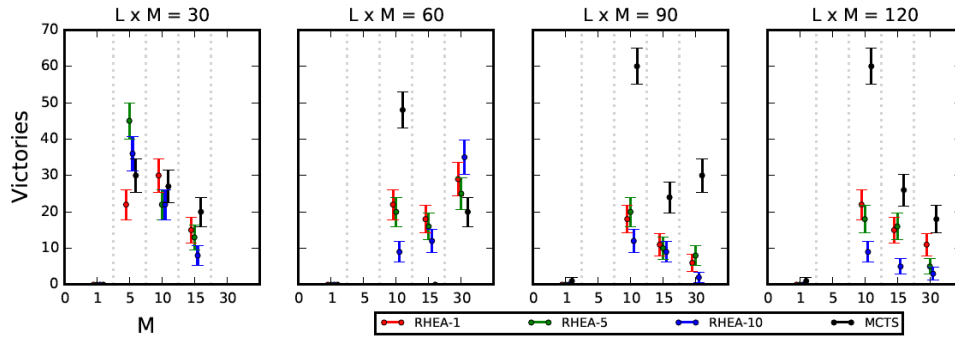


Figure 8.8: Victory rate (with standard error bars) in the game “Bird”, for all algorithms and configurations.

conclusion that finer control benefit this agent’s gameplay.

Similarly, MCTS also obtains the best overall results with small amounts of frame-skip, typically $M = 1$ and $M = 10$, and results get worse when M increases beyond that point. Next, we look deeper into each particular game.

Game Victory Rates

When looking at the results per game from the experimental work described here, it is noticeable that there is a strong dependency on the game in terms of algorithm, lookahead and the use of frame-skip. Table 8.6 shows the results for configuration $L \times M = 30$ in all games separately. Similar trends can be observed in most other games.

In many games, as expected after presenting the overall results, average victory rates are higher with frame-skip $M = 1$ and $M = 10$. However, some games show a different behaviour. The most interesting result is the performance shown in those games with very low win rates without frame-skip: “Bird”, “Lander”, “Mario” and “PTSP”. In all of these games, the use of frame-skip somewhat increases the win rate for some of the agents. In “Bird”, “Lander”, and “PTSP”, the use of frame-skip improves the performance from very close to 0%, to values between 30% and 50% in different settings. These games include scenarios where a longer lookahead may be beneficial, but not without a finer degree of control. A lookahead of 30 (rather than 10, as in the results of Table 8.4) does not guarantee an improvement in performance, but using frame-skip to simultaneously reduce the search space does (e.g. RHEA-5 increasing to 45% of victories). In fact, it is not surprising that “PTSP” is a game where frame-skip works well, it was particularly in that game where the concept used here was originally introduced (283).

There is also a small performance increase observed in “Mario”, from 0% to less than 20%. A possible

explanation why the performance gain is smaller here is that “Mario” requires longer planning in conjunction with fast reaction to avoid enemies and hazards in the levels. A deeper look into platformer games may be required to tackle this type of games in a general context, which is also a new type of games added to the GVGAI framework.

In the game “Bird” (see Figure 8.8), there is a clear difference between using distinct lookaheads. The best performance is achieved when the lookahead is 90 and 120, with 60% victory rate, achieved by MCTS. However, all RHEA approaches are better than MCTS with the shorter lookahead ($L \times M = 30$), and $M = 1$ shows practically no victories for all agents.

8.4 Conclusions

In this chapter we describe several works which open up novel research directions for RHEA, covering several concepts and describing in-depth discussions of the topics as well as initial experimental results as proof of concept.

Project Thyia. First, we presented our vision for Thyia, a large system comprised of several modules with different focus and functionality. The core concept is based on the idea of an Artificial Intelligence entity which continuously plays games, using planning to solve the problems imposed and learning to improve its performance over time. The learning can be seen twofold: in terms of the knowledge of the system and its ability to solve problems of diverse complexities and interact with dynamic environments; but also in terms of adjusting its parameters and structure so as to evolve and adapt to the worlds it encounters. We also see interaction with the real world as an aspect of key importance: an AI entity able to interact with humans in meaningful ways (such as direct communication, knowledge exchange, experience sharing) is much more interesting to study than algorithms which only exist in their constrained environments.

We described several limitations of the system throughout the section and we highlight that combining many different components is bound to raise several issues. Planning methods require models of the game worlds to be able to simulate ahead and Rolling Horizon Evolutionary Algorithms, as well as Neural Networks, offer great flexibility and customisation with the risk of manually tailored parameters may not be the best choice. Hyper-parameter optimisation methods such as the N-Tuple Bandit Evolutionary Algorithm can help find good combinations of parameters, yet the game-playing and learning methods need to be able to cope with a change in parameters across the system and limit their assumptions. Lastly, human interaction brings natural language processing complications and raises some ethical concerns that should be taken into account when building a system such as Thyia.

There are many ways in which the concept of this system could be expanded even further. Knowledge representation is an interesting aspect to consider: we might want the knowledge of our system to be stored in a way such that it is more easily interpreted than a Neural Network. To this extent, Hierarchical Knowledge Bases (284) seem like a natural addition.

Furthermore, in order for the system to be a truly general game player, it should be able to play games even when a game model is not provided. Learning forward models in the general game-playing context is an active area of research, with several impressive advances (285; 286; 287; 35). With the addition of such a module, we speculate the system could even receive games from external sources (thus not adhering to any accidental assumptions included in the building of the system) and learn how to play them.

An important aspect to be considered in future work is the evaluation of the system. Given its complexities and ‘always-on’ characteristics, evaluation would have to be done based on the system’s outputs to user queries to analyse its current knowledge and skills. Given its lack of compatibility with traditional benchmarks and evaluation systems, new benchmarks could be considered for continuous evaluation of complex systems such as Thyia. Additionally, it would be interesting to explore the algorithm’s ability to produce not only ‘intelligent’ game-play, but also meaningful, creative, fun or inspiring experiences for the players it interacts with.

Lastly, we acknowledge analytics as an important possible addition. With AI systems becoming more intelligent, but also large black boxes, it is important to be able to understand their thinking process that leads to certain behaviours. Answering the question of why a decision was made would be arguably more important than making the right decision. There have been several approaches taken to extract features from a planning agent’s own experience while playing a game (20), as well as to visualise these features to give a better insight into the agent’s decisions (18).

Representation types. Next, we discussed the topic of different representations within RHEA, with a detailed study in one particular option: the rhNEAT algorithm combines RHEA and NeuroEvolution of Augmented Topologies (NEAT) so as individuals encode neural networks, which receive game features

as input and output an action to play in the game; the evolutionary process then evolves the weights and topologies of these networks. The evaluation of such an individual keeps the rollout concept within the algorithm, by using the network encoded in an individual to repeatedly generate actions given an input state and advance the state with the game’s forward model, for several steps; the game state reached is then evaluated with a heuristic function, the value becoming the fitness of the individual.

The experiments presented test out several configurations of the algorithm in 20 GVGAI games: population carrying and speciation enhancements are both observed to lead to a large boost in performance (population carrying the most), especially when combined, from 15.54% win rate for the baseline method, to 36.5%. This overall best performance of the algorithm is still lower than previously preferred methods (44.8% for RHEA and 42.65% for MCTS), but the research led into some interesting insights about the inner-workings of this algorithm, which showcases widely different behaviours than the traditional SFP methods. Given that research into this topic is in its early stages, it already shows great promise for carrying out further investigation into different representation options in RHEA. This work opens up research into more indirect representations, such as Compositional Pattern Producing Networks (CCPNs) or HyperNEAT.

Further, we highlight that there is vast literature on NEAT and the many modifications that can be brought to the base algorithm (271); the entire setup presented here could be further improved for better performance, as well as automatically optimised for the wide range of games in GVGAI. Looking at different options for population control (288) or at the use of convolutional layers to extract features from the game screen could lead to breakthroughs as well.

Physics-based games. Lastly, we discussed the new type of physics-based games in the General Video Game AI (GVGAI) Framework, which aim to diversify the challenges presented to the agents and create more realistic simulations of real-world environments and situations. Elements such as inertia, gravity or friction are now part of the framework, as well as a set of new games making use of these properties. We presented experiments with both RHEA and MCTS agents within this context, making use of the *repeat* frame-skip concept to aid in their decision-making in these now much larger level spaces. Different lookaheads and frame-skip values were tested to analyse the benefit of various algorithm configurations.

One main conclusion drawn from the experiments described here is that RHEA and MCTS have different strong and weak points depending on the game, yet none dominates the other, as is typical in many experimental settings in GVGAI. MCTS performs better on average in these games in the vanilla form, yet the (limited) use of frame-skip boosts performance in both RHEA and MCTS. Too much frame-skip, and the planning becomes very inaccurate and difficult to control for both algorithms. In 4 out of 10 games tested, victories can only be achieved through the use of frame-skip, highlighting the benefit of this enhancement in particularly difficult or sparse games, or those games with control systems (player actions / input effects on the in-game object controlled) most different to grid-based games. Choosing the correct parameters for the best performance depends on the game attacked, as finer or coarser control may be preferred in different instances - a potential area to explore more through automatic optimisation.

Future work in this area is twofold: first, the frame-skip enhancement explored here could be better analysed in grid-based games as well. Although generally those are smaller games where repeated actions might not be beneficial, the increased thinking time could help agents make more informed decisions in sparse reward environments in particular. A better exploration of the variety of options for the type of frame-skip (as discussed in Section 3.2.4) could also advantage agents in both grid-based and physics-based games, including combinations of the different types or dynamic adjustments similar to those explored in Section 5.2.

Second, the unique properties of physics-based games could be further analysed and agents developed to take advantage of these and develop their decision-making process accordingly. The games themselves could be further diversified (Table 8.3 shows that only a subset of the different types of features are present in these types of games so far), as well as the physics engine itself improved to handle more complex interactions between sprites accurately and realistically. Looking at the application of these algorithms in completely new types of games, such as (289) could lead to exciting new insights. Continuous action input from agents could also be supported in this type of games, which would add a new layer of challenge and open a whole different area of research which has not yet been approached in GVGAI.

Chapter 9

Thesis Conclusions

This thesis gives an overview of previous literature relevant to Rolling Horizon Evolutionary Algorithms and their application in General Video Game Playing, as game-playing agents in single-player games. Various topics were covered to include work carried out regarding benchmarking in a common setting of previously published approaches, analysis of base parameters (population size and individual length), the effects of population seeding and hybridisation. Additionally, experiments are presented promoting in-depth analysis of the algorithms tested (visual and feature-based, used in performance-enhancing techniques or win prediction) and automatic optimisation. Applications in diverse environments are reviewed, and the end of the document begins a discussion of novel work carried out which opened opportunities for new research directions to further study and apply Rolling Horizon Evolutionary Algorithms.

We make a note of the role randomness plays throughout this thesis. Random initialisation for RHEA is the best choice in most environments tested; randomising some of the parameters online led to better results in many games, even compared to more complex optimisation methods; environments with random elements are often more complicated to tackle; and Random Search was shown to outperform the vanilla versions of RHEA and MCTS in most of the environments. Often, the good performance obtained due to randomness is due to the highly constrained experimental settings: real-time online decision-making does not offer much leeway in employing complex time-intensive mechanisms, raising the need for not only innovation in simple ideas, but also for implementation optimisation (e.g. the shift buffer, which is efficient in preserving information as well as very fast to compute). It may well be that the solution to general video game playing is a combination of offline and online decision-making process, with randomness playing a big part in the success obtained.

The rest of this section summarises conclusions and the main takeaways through answers to the research questions introduced in Chapter 1.

Research Question 1 *What is a varied set of environments appropriate for testing general video game-playing algorithms in?*

Section 2.1.2 describes and analyses a subset of games from the General Video Game AI framework, first introduced in (15). This game set combines a diverse set of challenges for artificial game players, including various objects for the players to interact with in different ways, different mechanics and control schemes, different winning and losing conditions and a variety of levels with different properties and gimmicks associated. The difficulty of the challenges and the impact of a game's stochasticity varies across games as well, as highlighted later in the results. This set of games is further analysed in detail in terms of unique or challenging game features, which encourages more thorough future research in similar environments, going beyond win/loss metrics and looking further into environment characteristics that lead to interesting insights or particular behaviours. All games used in the thesis are described in the appendices as well for easier comprehension of the true variety tested here. Most experiments presented in this thesis (with few exceptions) use this main set of games and show varied performance across the environments, with some games (e.g. "Aliens", "Intersection") often seeing win rates of 100%, while others (e.g. "Dig Dug", "Roguelike") are at the opposite end of the spectrum at 0% wins. Although some methods do see occasional wins on the latter problems mentioned (see Chapter 6), they generally remain too difficult for a general approach to tackle efficiently.

We further test the application of the algorithm in specific environments in Chapter 7. First, we test RHEA in "Pommerman", a complex multi-player partially-observable game, with variations for competitive or a mix of both competitive and cooperative play. We provide insights into the behaviour of the agent in this environment, as well as analysing its performance in terms of win rate. MCTS generally outperforms RHEA in several of the settings tested, through a safer approach than RHEA's aggressive behaviour in this

game. The work shows promise in the direct application of the algorithm in new environments and suggests further optimisation of the algorithm could be greatly beneficial, as well as improving opponent modelling, identifying trap moves and introducing assumptions for partial observability settings.

Next, we experiment with the application of RHEA in “Tribes”, a multi-player, multi-agent, stochastic and partially observable strategy game. This environment is very complex and requires strategic play as well as tactical decisions while managing an army, technology research, civilisation development and overall economy. Results show that RHEA is able to achieve high performance in this game, yet it is still far from human playing strength. Lastly, we look at tabletop games in the context of the Tabletop Games Framework (TAG). Preliminary results show some moderate success for RHEA in competitive games, with MCTS outperforming it in some of the games, both with plenty of room for improvement. None of the agents are able to win in the cooperative game “Pandemic” in the generic form proposed, suggesting special focus should be given to such complex and cooperative environments in future developments.

Research Question 2 *What parameters can be extracted from a Rolling Horizon Evolutionary Algorithm, and what modifications can be integrated into the algorithm for varied behaviour?*

Chapter 3 formalises the Rolling Horizon Evolutionary Algorithm first introduced in (7), discussing the many aspects the algorithm deals with, from the perspectives of evolutionary algorithms, as well as general game playing. Many modifications and parameters are brought together, including those previously explored in literature (e.g. shift buffer, Monte Carlo rollout-based evaluations of individuals), as well as new ones (e.g. dynamic individual length adjustments, statistical trees built during evolution and used in action recommendation). Values explored in follow-up studies are detailed, leading to a search space of 5.36×10^8 total variations of the algorithm.

Research Question 3 *What is the effect of adjusting the values of the parameters and combining modifications on the performance of the Rolling Horizon Evolutionary Algorithm?*

Chapter 4 studies the various parameters and modifications in the algorithm are studied in a series of experiments using the same common setting, in isolation (to perform a fair analysis of particular benefits brought by individual enhancements), and in combination (to assess which parameters work well together, or where there might be issues in synergy; both of these situations were identified in the experiments and highlighted appropriately). We first carried out simple tests regarding the benefits of shorter or longer lookaheads (individual length; for exploring more of the level space, at the expense of less iterations and therefore less accurate information; or the opposite), and more or less sequences of actions evolved at a time (population size; for exploring more of the search space, at the expense of less iterations and less accurate information; or the opposite). We critically analyse different configurations for population size and individual length in the set of 20 GVGAI games. Distinctions are made between deterministic and stochastic games, and the implications of using superior time budgets are studied. Results show that there is scope for the use of these techniques, which in some configurations outperform Monte Carlo Tree Search, and also suggest that further research into these methods could further boost their performance.

Next, we proposed the use of population seeding to improve the performance of Rolling Horizon Evolution and present the results of two methods, One Step Look Ahead and Monte Carlo Tree Search, tested on the 20 GVGAI games, in different population size and individual length configuration. An in-depth analysis is carried out between the results of the seeding methods and the vanilla Rolling Horizon Evolutionary Algorithm. In addition, we present a comparison to Monte Carlo Tree Search, with promising results: seeding is able to boost performance significantly over baseline methods and even match the high level of play obtained by the Monte Carlo Tree Search.

Lastly, we proposed a fair juxtaposition of four enhancements applied to different parts of the evolutionary process: bandit-based mutation, a statistical tree for action selection, a shift buffer for population management and additional Monte Carlo simulations at the end of an individual’s evaluation. These methods are studied individually, as well as their hybrids, on the 20 GVGAI games, and compared to the vanilla version of the Rolling Horizon Evolutionary Algorithm, in addition to the dominating Monte Carlo Tree Search. The results show that some of the enhancements are able to produce impressive results, while others fall short. Interesting hybrids also emerge, encouraging further research into this problem.

Research Question 4 *What insights can be gained from deeper analysis into the algorithm’s decision-making process?*

Chapter 5 presents first VERTIGØ, a tool developed for the analysis and visualisation of Rolling Horizon Evolutionary Algorithms, featuring an easy-to-use and complete graphical user interface, which allows integration within the GVGAI framework. Users are able to select the game and level to run, customise the parameters of the agent between runs and observe an in-depth analysis of its performance through various visual information extracted from gameplay data, live while playing the game. All detailed data can also be saved for further post-processing. This visualisation aims to inform a deeper analysis into algorithm behaviour, in an attempt to justify its decisions improve its performance based on this knowledge.

This analysis is used to inform further studies on dynamically adjusting the individual length, according to the density of rewards observed during the game (to prioritise exploration of the levels versus accurate statistics at the correct moments in time). Modifications were proposed for two algorithms, Monte Carlo Tree Search and Rolling Horizon Evolutionary Algorithms, aiming at improving performance in games with sparse rewards, while maintaining high overall win rates across those games where rewards are plentiful. Results show that longer rollouts and individual lengths, either fixed or responsive to changes in fitness landscape features, lead to a boost of performance in sparse-reward games, without being detrimental to non-sparse-reward scenarios.

All of the features extracted by VERTIGØ are employed in a second study as well. We propose a general agent performance prediction system, tested in real time within the context of the General Video Game AI framework. It is solely based on agent features, therefore removing potential human bias produced by game-based features observed in known games. Three different models can be queried while playing the game to determine whether the agent will win or lose, based on the current game state: early, mid and late game feature models. The models are trained on 80 games in the framework and tested on 20 new games, for 14 variations of 3 different methods. Results are positive, indicating that there is great scope for predicting the outcome of any given game. This further suggests that changing the behaviour of the agent to promote certain trends in its decision-making process could lead to a boost in performance, making tools such as VERTIGØ key in not only better understanding the inner-workings of algorithms, but also in improving them.

Research Question 5 *How can the large parameter space of the Rolling Horizon Evolutionary Algorithm be searched effectively for a good configuration of parameters?*

Chapter 6 tackles the topic of automatic optimisation. Given the large set of parameters that resulted after the experiments presented, which added more and more modifications to the algorithm, it becomes infeasible to choose correct configurations manually in general, as well as for specific problems. Experiments are presented exploring these concepts first offline, taking several days to learn good algorithm configurations for each of the 20 GVGAI games tested. We use a parameter optimiser, the N-Tuple Bandit Evolutionary Algorithm, to find the best combination of parameters in each of the 20 games. Further, we analyse the algorithm's parameters and some interesting combinations revealed through the optimisation process. Lastly, we find new state-of-the-art solutions on several games by automatically exploring the large parameter space of RHEA.

This work was later extended to work online instead, with the agent adjusting its parameters while also searching for good action sequences during the game. We propose adapting online tuning methods for Rolling Horizon Evolutionary Algorithms, and test the effect on the agent's win rate. Online tuned agents are able to achieve results comparable to the state-of-the-art, including first win rates in very difficult problems, while employing a more general and highly adaptive approach. We additionally include further insight into the algorithm itself, given by statistics gathered during the tuning process and highlight key parameter choices.

We conclude that these approaches are not only efficient at finding parameter settings that work well (and often, better) in many games, observing first win rates in extremely difficult problems, but they also offer further insights into the algorithm's parameter space, such as which values for parameters are preferable, which values could be better explored in a better-tailored search space, or even highlight particular synergies between parameters that cannot be easily identified through manual tuning or human intuition.

Research Question 6 *What research directions into Rolling Horizon Evolutionary Algorithms are opened up and encouraged by novel work?*

Chapter 8 includes in-depth discussions of novel work which opens several exciting new research pathways. This refers to testing new representations within RHEA, new environments for the algorithm altogether which better reflect real-world circumstances, as well as building a whole artificial entity framework around the algorithm to allow it to interact with our world better, learn from human players and even

share its experiences with other artificial entities, such as generative systems, for an exciting create-play-feedback-improve creation loop.

We first introduce the vision behind a new project called Thyia, which focuses around creating a present, continuous, ‘always-on’, interactive game-player. Next, we present first experiments combining Rolling Horizon Evolutionary Algorithms with NeuroEvolution of Augmenting Topologies, where we evolve the weights and connections of a neural network in real-time, planning several steps ahead before returning an action to execute in the game. Different versions of the algorithm are explored in the 20 GVGAI games. Although results are overall not better than other statistical forward planning methods, the algorithm better adapts to changing game features and sets new state-of-the-art records in some very difficult problems, as well as opening up research into new representations for RHEA.

Lastly, we discuss an extension of GVGAI supporting real-world physics. We then test a version of RHEA and MCTS which discretise the now larger level space by repeating actions chosen several times (and artificially inflating their thinking time as well), in 10 environments making use of the new physics features. Results show that the simple action repetition enhancement helps the agents in some of the games, although the performance of each variant is highly dependent on the game being played and win rates vary widely.

Research Question 7 *What is the new state-of-the-art in Rolling Horizon Evolutionary Algorithms, and how does it compare to the previous state-of-the-art approaches based on Monte Carlo Tree Search?*

Table 9.1: **20 GVGAI games** state-of-the-art win rate. Games are indexed 0-19 in the following order: “Dig Dug”, “Lemmings”, “Roguelike”, “Chopper”, “Crossfire”, “Chase”, “Camel Race”, “Escape”, “Hungry Birds”, “Bait”, “Wait for Breakfast”, “Survive Zombies”, “Modality”, “Missile Command”, “Plaque Attack”, “Seaquest”, “Infection”, “Aliens”, “Butterflies”, “Intersection”. Full game descriptions in Appendix A. MCTS column shows highest MCTS win rate, with the implementation described in Section 2.2. RHEA column shows highest RHEA win rate, obtained with the configuration described by the rest of the columns. P.Size = population size; I.Len = individual length; Offspring = offspring count; 1 elite for all games; Init. = initialisation method; Selection = selection type; Crossover = crossover type; Mutation = mutation type; Fit. = fitness assignment type; DD = dynamic depth; SB = shift buffer (discount in brackets); MC = Monte Carlo rollouts (length and number of repetitions in brackets); Skip = frame skip (type in brackets, optionally with number of frames skipped); Fit.Div = diversity in fitness (weight in bracket).

Game	MCTS	RHEA	Parameters										Source
			Numerical			Nominal					Enhancements		
			P.Size	I.Len	Offspring	Init.	Selection	Crossover	Mutation	Fit.			
0	0% (0.00)	0% (0.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
1	0% (0.00)	4% (1.98)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	DD	(19)	
2	0% (0.00)	0% (0.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
3	100% (0.00)	100% (0.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
4	1% (0.99)	10% (3.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
5	5% (2.18)	13% (3.39)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
6	9% (2.86)	41% (4.92)	15	15	1	MCTS	Tourn.	Uniform	Diversity	Disc.	SB(0.99); MC(1.0,5); Skip(Rep); Fit.Div(0.5)	(21)	
7	0% (0.00)	46% (4.98)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	MC(0.5,1)	(17)	
8	6% (2.37)	12% (3.25)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	SB(0.9); MC(0.5,10)	(17)	
9	10% (3.00)	20% (4.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	SB(0.9); MC(0.5,10)	(17)	
10	6% (2.37)	83% (3.76)	10	20	10	ISLA	Tourn.	1-point	-	Disc.	SB(0.99); MC(2.0,1); Skip(Null); DD	(21)	
11	45% (4.97)	56% (4.97)	20	15	10	RND	Tourn.	Uniform	Uniform	Disc.	SB(0.99); MC(2.0,5); Skip(RND); DD	(21)	
12	25% (4.33)	38% (4.42)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
13	69% (4.62)	86% (3.47)	15	20	15	RND	Tourn.	1-point	Softmax	Max	SB(0.9); MC(2.0,1)	(21)	
14	92% (2.71)	100% (0.00)	15	20	1	ISLA	Rank	1-point	Diversity	Max	SB(1.0); MC(2.0,1); Skip(RND); DD	(21)	
15	65% (4.77)	84% (3.66)	20	20	1	RND	Rank	Uniform	-	Average	SB(0.9); MC(2.0,1)	(21)	
16	97% (1.71)	100% (0.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
17	100% (0.00)	100% (0.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
18	100% (0.00)	96% (1.92)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	
19	100% (0.00)	100% (0.00)	10	15	10	RND	Tourn.	Uniform	Uniform	Last	-	(15)	

Table 9.2: **“Pommernan”** state-of-the-art win rate (FFA game mode; 200 games per vision range option between MCTS, RHEA, one step look ahead and rule-based players; vision range is 2 for best RHEA result, and 1 for best MCTS result), see Section 7.1 for details.

Game	MCTS	RHEA	Parameters									Source
			Numerical			Nominal					Enhancements	
			P.Size	I.Len	Offspring	Init.	Selection	Crossover	Mutation	Fit.		
0	68% (3.00)	21% (3.00)	1	12	1	RND	-	-	Uniform	Last	SB(0.99)	(11)

Table 9.3: “**Tribes**” state-of-the-art win rate (averaged across 2500 two-player games from a round-robin tournament between RHEA, MCTS, Monte Carlo search, rule-based, one step look ahead and random players), see Section 7.2 for details.

Game	MCTS	RHEA	Parameters								Source	
			Numerical			Nominal				Enhancements		
			P.Size	I.Len	Offspring	Init.	Selection	Crossover	Mutation			Fit.
0	60% (1.40)	75% (1.25)	1	20	1	RND	-	-	Uniform	Last	SB(0.99)	(12)

Table 9.4: **5 GVGAI deceptive games** state-of-the-art win rate. Games are indexed 0-4 in the following order: “Decepti Coins”, “Flower”, “Invest”, “Sister Saviour”, “Wafer Thin Mints Exit”. Full game descriptions in Appendix B.

Game	MCTS	RHEA	Parameters									Source
			Numerical			Nominal					Enhancements	
			P.Size	I.Len	Offspring	Init.	Selection	Crossover	Mutation	Fit.		
0	80%	55.56%	10	14	10	RND	Tourn.	Uniform	Uniform	Last	MC(5;0,5); DD	(19)
1	100%	100%	10	14	10	RND	Tourn.	Uniform	Uniform	Last	MC(5;0,5); DD	(19)
2	0%	0%	10	14	10	RND	Tourn.	Uniform	Uniform	Last	MC(5;0,5); DD	(19)
3	10%	8%	10	50	10	RND	Tourn.	Uniform	Uniform	Last	MC(5;0,5)	(19)
4	100%	100%	10	14	10	RND	Tourn.	Uniform	Uniform	Last	MC(5;0,5); DD	(19)

Table 9.5: **10 GVGAI physics-based games** state-of-the-art win rate. Games are indexed 0-9 in the following order: “Artillery”, “Asteroids”, “Bird”, “Bubble”, “Candy”, “Lander”, “Mario”, “Pong”, “PTSP”, “Racing”. Full game descriptions in Appendix C. MCTS column includes rollout length \times frames skipped.

Game	MCTS	RHEA	Parameters								Source	
			Numerical			Nominal				Enhancements		
			P.Size	I.Len	Offspring	Init.	Selection	Crossover	Mutation			Fit.
0	51% (5.00) - 30 × 1	57% (4.95)	10	30	10	RND	Tourn.	Uniform	Uniform	Last	-	(26)
1	85% (3.57) - 30 × 1	48% (5.00)	10	30	10	RND	Tourn.	Uniform	Uniform	Last	-	(26)
2	30% (4.58) - 6 × 5	45% (4.97)	5	6	5	RND	Tourn.	Uniform	Uniform	Last	Skip(Rep-5)	(26)
3	97% (1.71) - 30 × 1	68% (4.66)	5	30	5	RND	Tourn.	Uniform	Uniform	Last	-	(26)
4	56% (4.96) - 6 × 5	22% (4.14)	5	6	5	RND	Tourn.	Uniform	Uniform	Last	Skip(Rep-5)	(26)
5	58% (4.94) - 6 × 5	29% (4.54)	5	6	5	RND	Tourn.	Uniform	Uniform	Last	Skip(Rep-5)	(26)
6	18% (3.84) - 6 × 5	16% (3.67)	5	3	5	RND	Tourn.	Uniform	Uniform	Last	Skip(Rep-10)	(26)
7	100% (0.00) - 6 × 5	80% (4.00)	10	30	10	RND	Tourn.	Uniform	Uniform	Last	-	(26)
8	61% (4.88) - 3 × 10	50% (5.00)	5	6	5	RND	Tourn.	Uniform	Uniform	Last	Skip(Rep-5)	(26)
9	73% (4.44) - 30 × 1	70% (4.58)	10	30	10	RND	Tourn.	Uniform	Uniform	Last	-	(26)

Table 9.6: **8 Tabletop Games** state-of-the-art win rate (across 100 repetitions per four-player game, played by RHEA, MCTS, one step look ahead and random players; “Tic-Tac-Toe” shows results for two-player games from a round-robin tournament; “Pandemic” shows results for teams of the same player), see Section 7.3 for details. Games are indexed 0-7 in the following order: “Tic-Tac-Toe”, “Dots & Boxes”, “Love Letter”, “Uno”, “Virus!”, “Exploding Kittens”, “Colt Express”, “Pandemic”. RHEA mutation randomly chooses one gene in the individual and mutates all subsequent genes.

Game	MCTS	RHEA	Parameters									Source
			Numerical			Nominal				Enhancements		
			P.Size	I.Len	Offspring	Init.	Selection	Crossover	Mutation		Fit.	
0	98%	44%	1	10	1	RND	-	-	branching	Disc.	-	(13)
1	17%	69%	1	10	1	RND	-	-	branching	Disc.	-	(13)
2	44%	32%	1	10	1	RND	-	-	branching	Disc.	-	(13)
3	26%	22%	1	10	1	RND	-	-	branching	Disc.	-	(13)
4	42%	30%	1	10	1	RND	-	-	branching	Disc.	-	(13)
5	37%	21%	1	10	1	RND	-	-	branching	Disc.	-	(13)
6	26%	28%	1	10	1	RND	-	-	branching	Disc.	-	(13)
7	0%	0%	1	10	1	RND	-	-	branching	Disc.	-	(13)

All experiments presented in the thesis include detailed results through tables, figures and external links to software, data or extended results. The best results obtained on each of the environments tested are summarised in Tables 9.1-9.6. We note that random initialisation is sufficient in most environments,

although MCTS initialisation was suggested by automatic optimisation as the best option in “Camel Race” (a game with large and sparse levels, where MCTS can be used for the initial assessment of the state space and potential dangers around the player, for RHEA to then take over and discover those further away rewards); ISLA initialisation is recommended in 2 other games: “Wait for Breakfast” and “Plaque Attack”, both of which need first a thorough assessment of the immediate next states possible (to avoid sudden losses). In real-time environments with fixed action space sizes, a population size of 10 and individual length of 15 is often preferred (or slight variations up or down, generally the higher values, the better, given a constraint budget), with offspring equal to the population size. In environments with dynamic action spaces, however, we recommend keeping a single individual in the population with length sufficient to catch interesting changes in the game state (e.g. length 12 in “Pommernan” is just about longer than the time a bomb takes to explode). The shift buffer enhancement aids performance in most environments, although it should not generally be used with the dynamic depth enhancement. Generally, RHEA is strong in environments with sparse rewards and/or large dynamic action spaces. We recommend starting from the variant described here for the environment that most closely matches the desired new testbed, to then add or remove enhancements and modify the population size and individual length so as to maximise these within the experimental constraints.

All experiments are grounded in past work in the area of general video game playing and include direct comparisons to Monte Carlo Tree Search, the previous state-of-the-art in the domain and favourite across many environments. The thesis compares not only the performance of these algorithms, but also their behaviour and particular differences in thinking process, with the aim of shading some light into their distinctions, similarities, and best-case applications. Several variations of the Rolling Horizon Evolutionary Algorithm are shown to outperform MCTS, and we hope to see RHEA as the algorithm of choice much more often due to its superior performance especially in complex sparse reward environments, its simplicity and high adaptability.

9.1 Future work

Regarding topics not yet covered of this vast problem, one fairly straightforward line of future work is the improvement of the vanilla Rolling Horizon Evolutionary Algorithm in this general setting. The objectives are twofold: first, seeking bigger improvements of action sequences during the evolution phase, without the need of having too broad an exploration as in the case of Random Search; and second, being able to better handle long individual lengths in order for them to not hinder the evolutionary process. Additionally, further analysis could be conducted on stochastic games, considering the effects of more elite members in the population or re-sampling individuals, in order to alleviate the effect of noise in the evaluations.

Additionally, no work has been done in the use of macro actions in grid-based games in GVGAI, although it seems natural that different macro action lengths will also be needed per game. Ideally, an approach that could bring good performance to the agents would be one where each action is more involved, performing moves such as path-finding to the closest sprite of a given type, or escape from a given location.

An approach that could inspire future work was that by Kelly et al. (290). They looked at a genetic programming algorithm applied to a similar problem, but using the Arcade Learning Environment and screen capture as input instead. Their proposed technique, Tangled Program Graph (which creates and evolves a graph of sub-programs meant to solve different sub-problems) outperforms deep learning, while using significantly lower computation power.

Further, it would be interesting to explore different versions of the algorithm which could improve its search efficiency. For example, Harik et al. (291) attempted to make better use of the evolution budget by using an estimation of a distribution vector instead of a population for binary optimisation problems. Only two individuals would be sampled and evaluated from this vector at each iteration to determine a winner and a loser and adjust the distribution vector to match the winner - thus eliminating evolutionary operators such as mutation or crossover. This method was shown to be effective in noisy environments by Friedrich (292). It is possible to expand this method from binary problems to the multi-action GVGAI games my studies are concerned with, therefore possibly giving the algorithm an advantage in stochastic games. Lucas et al. (263) have also proposed modifications to this algorithm which were suggested to outperform the original.

The rest of this section reviews other interesting works that could be used for future research in the area, in particular to extend the algorithm to multi-objective and multi-player scenarios.

Multi-objective. Most of the games in the GVGAI Framework have multiple (possibly conflicting) objectives, starting from the simplest case in which an agent wants to win, but also achieve the highest score possible. This can be further looked into as an agent wanting to explore as much of the level as possible, eliminating Non-Player Characters, collecting resources etc.

Thus rises the need to balance all the different aspects of one game and direct the search so as to satisfy the multiple objectives. One game competition that addressed this problem specifically was the Multi-Objective Physical Travelling Salesman Problem (MO-PTSP) Competition organised in 2013 (293). In this modification of the Physical Travelling Salesman Problem, the players control a ship that must not only collect all the waypoints in the shortest time possible, but also minimise fuel consumption. The winning entry used a Monte Carlo Tree Search controller, with the addition of several other algorithms for distance mapping and TSP route planning. The parameters of the algorithm were adjusted with a Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (294) in order to balance between the different objectives.

There are several studies in the literature on Multi-Objective Monte Carlo Tree Search (MO-MCTS) methods. Perez et al. introduced one technique in (295; 296) and measured it, in both offline and online scenarios, against a previous MO-MCTS algorithm (297), as well as the state-of-the-art in Multi-Objective Evolutionary Algorithms, Non-dominated Sorting Evolutionary Algorithm 2 (NSGA-II) (298). In their algorithm, Pareto-MO-MCTS, each node in the search tree additionally keeps track of a local pareto front (a set of non-dominated solutions), updated after a new vector of rewards is obtained from Monte Carlo simulations. They suggest that Pareto-MO-MCTS performs similarly to the previous MO-MCTS implementation, but they outperform NSGA-II on both test problems, Deep Sea Treasure (an episodic puzzle game) and Puddle Driver (similar to PTSP). However, it is reported that NSGA-II does not always provide worse solutions and provides a better solution distribution on the pareto front due to its use of population diversity methods.

Later, Perez et al. (299) introduced multi-objective methods to GVGAI for some interesting outcomes. The conflicting objectives they identified were the game score and the level exploration (approached through a pheromone trail heuristic), while weighing in appropriately a win or a loss for the agent. These methods are only applied to Monte Carlo Tree Search algorithms, leaving it an open question whether the same would work in EAs. As such, they tested the MO-MCTS presented in their previous work against the sample controller MCTS (which only takes into account the game score), a weighted-sum MO-MCTS and a mixed strategy MO-MCTS (which randomly chooses a different strategy for each game tick). The results indicate that the first controller achieves much better win rates overall and better average scores in most games than the other three, highlighting that the way the objectives are combined impacts performance.

Khalifa et al. (300) developed a multi-objective benchmark extended upon the existing GVGAI framework. This problem would focus on adjusting the parameters in a complex UCB equation for a simple Monte Carlo Tree Search algorithm, so that it performs well on several problems. The UCB equation used combined several game aspects to guide tree node selection, such as number of or proximity to certain sprites. Therefore, this agent would be a weighted-sum MO-MCTS trained offline (using the SMS-EMOA algorithm (301)). Their results indicate that it is possible to configure this agent so that it performs better across the games, although performances are not reported.

An overview of Multi-Objective Evolutionary Algorithms and the state-of-the-art in 2011 is given in a survey by Zhou et al. (302). MOEAs have been widely used for multi-objective problems, with the inner objectives of converging to the true Pareto front, as well as obtaining a population as diverse as possible. They discuss methods such as decomposition or co-evolution based algorithms, following the idea of divide and conquer, or hybrid algorithms aiming to make the best of different methods. The tuning of the algorithm's control parameters is discussed, the issue of applicability in real world problems being raised, especially domains where specific knowledge may not be available (e.g. the general setting my research is concerned with). To this extent, methods of adjusting mutation or crossover rates depending on population diversity are suggested, or learning and adjusting the lower and upper bounds of parameters to search within a more relevant solution space.

Noisy MO optimisation is covered in one of the sections as well, this being relevant to the GVGAI domain as most games present some aspect of stochasticity. Most commonly used methods to handle noise in such problems are indicated to be re-sampling (a costly solution; alternatives include Syberfeldt et al.'s (303) iterative re-sampling guided by a confidence metric, in turn based on the amount of noise in the neighbouring area in the solution space) or probabilistic ranking process introduced by Hughes in (304), which includes in the evaluation of a solution the standard deviation to minimise the effect of noise.

Multi-player. Multi-player games involve two or more players controlling one game object each. A distinction is made here to multi-agent games (where one or two players control several game objects each).

The idea of co-evolution presented in (305) can be used in multi-player games, where secondary populations of individuals are assigned to the other players. Panait and Luke identify the difference between cooperative and competitive co-evolution, where individuals evolve against each other to either help or be better than others; a typical way of achieving this effect is stated to be splitting the computation into different populations corresponding to sub-parts of the problem and evolving them in parallel, while allowing

them to interact in ways appropriate for the problem analysed.

Liu et al. (73) suggest an expansion of the single-player RHEA algorithm to two players (and possibly multiple others) in real-time games, by employing a co-evolution model of one population for each different player and using the actions from both to simulate possible future states and improve the action sequences. Their algorithm is tested on a two-player Space Battle game and returns favourable results.

Although Rolling Horizon Evolutionary Algorithms have not been traditionally applied to adversarial games due to their nature of not considering the presence of an opponent (co-evolutionary methods do so, but not the vanilla algorithm), they have seen limited success in multi-agent games. These are multi-player games in which the player does not only give one avatar to control (as is the case in the GVGAI Framework used in my studies), but several units that one or multiple actions may be given to. This leads to a very large branching factor, a difficult challenge for an AI agent to handle.

For example, Wang et al. (306) employed a modified version of online evolution using a portfolio of scripts to play Starcraft micro. In this work, rather than evolving groups or sequences of actions, the algorithm evolved plans to determine which script (among a set of available ones) each unit should use at each time step. Each gene in the individual represents a script that will be executed by a given unit in the next turn. Their results show evolution to be better than the other methods tested (Portfolio Greedy Search and Script and Cluster based UCT, the last of which attempts to cluster several units that are close together so they are assigned the same script) in moderate and large battles.

Additionally, Justesen et al. (76) used online evolution in Hero Academy, a turn-based game in which each of the 2 players controls multiple units and has 5 action points to distribute between them each turn. Their EA (Online Evolutionary Planning, or OEP) is compared against simpler methods (GreedyAction and GreedyTurn which consider the next best possible legal action or turn, respectively), as well as Monte Carlo Tree Search. They show in their work that Evolution wins over 80% of the games against the best of the other methods. However, they do consider the fact that MCTS was not built for this type of scenario, its low performance being due to it not being able to explore enough states and expand its tree efficiently enough (it is rarely able to even consider the opponent's subsequent move). Their results are promising, but more research in the area of multi-agent games and games with high branching factors is strongly encouraged.

This work was later expanded (307) to compare OEP with variations of MCTS, including a Non-Exploring MCTS (NE-MCTS, all children nodes are visited at least once before expanding, but the exploration term in the UCB equation is nullified; deterministic rollouts are used as well, controlled by a greedy policy) and a Bridge-Burning MCTS (BB-MCTS, an aggressive pruning strategy that cuts unpromising parts of the search tree out at set intervals and ignores them for the rest of the search). The NE-MCTS algorithm turns out to be the best performing out of all the methods analysed, but with no overall significance over OEP or BB-MCTS.

A second experiment was performed where the game complexity was increased (by increasing the number of action points available per turn). These results showed the Online Evolution method to have the best scalability, ending up with an over 55% win rate when playing against all MCTS variants in 100 games. When tested against human players, OEP won 25% of the games that were completed by the human testers, but 67% of total games, including those where players abandoned the game before finishing (indicating that they left because they were behind and predicted a loss). This shows the algorithm may be competitive against human players, although experiments in a more rigorous scenario are needed to clarify this aspect.

Co-evolution could be considered a natural way of dealing with a multi-agent problem, as suggested by Panait and Luke (305) in their survey on cooperative multi-agent methods.

Bibliography

- [1] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, “Matching Games and Algorithms for General Video Game Playing,” in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016, pp. 122–128.
- [2] M. J. Nelson, “Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–7.
- [3] M. Stephenson, D. Anderson, A. Khalifa, J. Levine, J. Renz, J. Togelius, and C. Salge, “A continuous information gain measure to find the most discriminatory problems for ai benchmarking,” in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.
- [4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level Control Through Deep Reinforcement Learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [5] D. Pérez Liébana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, “General Video Game AI: Competition, Challenges and Opportunities,” in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [6] D. P. Liébana, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, “The 2014 General Video Game Playing Competition,” in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, 2015, p. 1.
- [7] D. P. Liébana, S. Samothrakis, S. M. Lucas, and P. Rolfshagen, “Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2013, pp. 351–358.
- [8] J. Levine, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, “General Video Game Playing,” in *Artificial and Computational Intelligence in Games, Dagstuhl Follow-Ups*, vol. 6, 2013, pp. 1–7.
- [9] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, “General Video Game AI: a Multi-Track Framework for Evaluating Agents Games and Content Generation Algorithms,” *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, Sep 2019.
- [10] D. Perez-Liebana, S. M. Lucas, R. D. Gaina, J. Togelius, A. Khalifa, and J. Liu, *General Video Game Artificial Intelligence*. Morgan and Claypool Publishers, 2019.
- [11] D. Perez-Liebana, R. D. Gaina, O. Drageset, E. Ilhan, M. Balla, and S. M. Lucas, “Analysis of Statistical Forward Planning Methods in Pommern,” in *Proceedings of the Artificial intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 15, no. 1, 2019, pp. 66–72.
- [12] D. Perez-Liebana, Y.-J. Hsu, S. Emmanouilidis, B. Khaleque, and R. D. Gaina, “Tribes: A New Turn-Based Strategy Game for AI,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 252–258.
- [13] R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, “TAG: a Tabletop Games Framework,” in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2020.
- [14] F. F. Duarte, N. Lau, A. Pereira, and L. P. Reis, “A Survey of Planning and Learning in Games,” *Applied Sciences*, vol. 10, no. 13, p. 4529, 2020.

- [15] R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *Springer Lecture Notes in Computer Science, Applications of Evolutionary Computation, EvoApplications*, no. 10199, 2017, pp. 418–434.
- [16] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," in *Proceedings of the Congress on Evolutionary Computation*, June 2017, pp. 1956–1963.
- [17] —, "Rolling Horizon Evolution Enhancements in General Video Game Playing," in *Proceedings of IEEE Conference on Computational Intelligence and Games*, Aug 2017, pp. 88–95.
- [18] —, "VERTIGO: Visualisation of Rolling Horizon Evolutionary Algorithms in GVGAI," in *The 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2018, pp. 265–267.
- [19] —, "Tackling Sparse Rewards in Real-Time Games with Statistical Forward Planning Methods," in *AAAI Conference on Artificial Intelligence (AAAI-19)*, vol. 33, 2019, pp. 1691–1698.
- [20] —, "General Win Prediction from Agent Experience," in *Proc. of the IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2018, pp. 1–8.
- [21] R. D. Gaina, S. Devlin, S. M. Lucas, and D. Perez-Liebana, "Rolling Horizon Evolutionary Algorithms for General Video Game Playing," *IEEE Transactions on Games*, 2021.
- [22] R. D. Gaina, C. F. Sironi, M. H. Winands, D. Perez-Liebana, and S. M. Lucas, "Self-Adaptive Rolling Horizon Evolutionary Algorithms for General Video Game Playing," in *IEEE Conference on Games (CoG)*, 2020, pp. 367–374.
- [23] R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, "Design and Implementation of TAG: a Tabletop Games Framework," *arXiv preprint arXiv:2009.12065*, 2020.
- [24] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Project Thyia: A Forever Gameplayer," in *IEEE Conference on Games (COG)*, 2019, pp. 1–8.
- [25] D. Perez-Liebana, M. S. Alam, and R. D. Gaina, "Rolling Horizon NEAT for General Video Game Playing," in *IEEE Conference on Games (CoG)*, 2020, pp. 375–382.
- [26] D. Perez-Liebana, M. Stephenson, R. D. Gaina, J. Renz, and S. M. Lucas, "Introducing Real World Physics and Macro-Actions to General Video Game AI," in *Proceedings of IEEE Conference on Computational Intelligence and Games*, Aug 2017, pp. 248–255.
- [27] R. D. Gaina, D. Perez-Liebana, and S. M. Lucas, "General Video Game for 2 Players: Framework and Competition," in *Proc. of the IEEE Computer Science and Electronic Engineering Conference (CEEC)*, 2016, pp. 186–191.
- [28] R. D. Gaina, A. Couëtoux, D. J. Soemers, M. H. Winands, T. Vodopivec, F. Kirchgessner, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 Two-Player GVGAI Competition," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 209–220, June 2018.
- [29] R. D. Gaina, R. Volkovas, C. G. Díaz, and R. Davidson, "Automatic Game Tuning for Strategic Diversity," in *2017 9th Computer Science and Electronic Engineering (CEEC)*, Sept 2017, pp. 195–200.
- [30] R. D. Gaina and M. Stephenson, "'Did You Hear That?' Learning to Play Video Games from Audio Cues," in *IEEE Conference on Games (COG)*, 2019, pp. 1–4.
- [31] K. Kunanusont, R. D. Gaina, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The N-Tuple Bandit Evolutionary Algorithm for Game Improvement," in *Proc. of the IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 2201–2208.
- [32] C. F. Sironi, J. Liu, D. Perez-Liebana, R. D. Gaina, I. Bravi, S. M. Lucas, and M. H. M. Winands, "Self-adaptive MCTS for General Video Game Playing," in *Applications of Evolutionary Computation*, K. Sim and P. Kaufmann, Eds. Cham: Springer International Publishing, 2018, pp. 358–375.
- [33] D. Perez-Liebana, K. Hofmann, S. P. Mohanty, N. Kuno, A. Kramer, S. Devlin, R. D. Gaina, and D. Ionita, "The Multi-Agent Reinforcement Learning in Malmö Competition," in *Challenges in Machine Learning (CiML; NIPS Workshop)*, 2018, pp. 1–4.

- [34] S. M. Lucas, J. Liu, I. Bravi, R. D. Gaina, J. Woodward, V. Volz, and D. Perez-Liebana, “Efficient Evolutionary Methods for Game Agent Optimisation: Model-Based is Best,” *arXiv preprint arXiv:1901.00723*, 2019.
- [35] S. M. Lucas, A. Dockhorn, V. Volz, C. Bamford, R. D. Gaina, I. Bravi, D. Perez-Liebana, S. Mostaghim, and R. Kruse, “A Local Approach to Forward Model Learning: Results on the Game of Life Game,” *arXiv preprint arXiv:1903.12508*, 2019.
- [36] A. Dockhorn, S. M. Lucas, V. Volz, I. Bravi, R. D. Gaina, and D. Perez-Liebana, “Learning Local Forward Models on Unforgiving Games,” in *IEEE Conference on Games (COG)*, 2019, pp. 1–4.
- [37] O. Drageset, R. D. Gaina, D. Perez-Liebana, and M. H. Winands, “Optimising Level Generators for General Video Game AI,” in *IEEE Conference on Games (COG)*, 2019, pp. 1–8.
- [38] R. Montoliu, R. D. Gaina, D. Perez-Liebana, D. Delgado, and S. M. Lucas, “Efficient Heuristic Policy Optimisation for a Challenging Strategic Card Game,” in *International Conference on the Applications of Evolutionary Computation (EvoStar)*, vol. 12104. Springer, 2020, pp. 403–418.
- [39] M. Genesereth, N. Love, and B. Pell, “General Game Playing: Overview of the AAAI Competition,” in *AI Magazine*, vol. 26, no. 2, 2005, p. 62.
- [40] S. Sharma, Z. Kobti, and S. D. Goodwin, “General Game Playing: An Overview and Open Problems,” in *IEEE International Conference on Computing, Engineering and Information*, 2009, pp. 257–260.
- [41] G. J. Tesauro, “Temporal Difference Learning and TD-Gammon,” in *IEEE Conference on Computational Intelligence and Games*, 1995, pp. 58–68.
- [42] B. Al-Khateeb and G. Kendall, “The Importance of a Piece Difference Feature to Blondie 24,” in *UK Workshop on Computational Intelligence (UKCI)*, 2010, pp. 1–6.
- [43] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The Arcade Learning Environment: An Evaluation Platform for General Agents,” *J. Artif. Intell. Res.(JAIR)*, vol. 47, pp. 253–279, 2013.
- [44] Y. Naddaf, “Game-Independent AI Agents for Playing Atari 2600 Console Games,” MSc Thesis, University of Alberta, 2010.
- [45] A. Khalifa, D. Pérez Liébana, S. M. Lucas, and J. Togelius, “General Video Game Level Generation,” in *Proc. of the Genetic and Evolutionary Computation Conference 2016*. ACM, 2016, pp. 253–259.
- [46] A. Khalifa, M. C. Green, D. Pérez Liébana, and J. Togelius, “General Video Game Rule Generation,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 170–177.
- [47] M. J. Nelson, “Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus,” in *Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games*, 2016.
- [48] A. Babadi, B. Omoomi, and G. Kendall, “EnHiC: An Enforced Hill Climbing Based System for General Game Playing,” in *IEEE Conference on Computational Intelligence and Games*, vol. 1, 2015, pp. 193–199.
- [49] D. P. Liébana, S. Samothrakis, and S. M. Lucas, “Knowledge-based Fast Evolutionary MCTS for General Video Game Playing,” in *IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–8.
- [50] C. Chu, H. Hashizume, Z. Guo, T. Harada, and R. Thawonmas, “Combining Pathfinding Algorithm with Knowledge-based Monte-Carlo Tree Search in General Video Game Playing,” in *IEEE Conference on Computational Intelligence and Games*, vol. 1, 2015, pp. 523–529.
- [51] H. Park and K. J. Kim, “MCTS with Influence Map for General Video Game Playing,” in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2015, pp. 534–535.
- [52] D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, “Deceptive Games,” in *EvoStar*, 2018.
- [53] N. Companez and A. Aleti, “Can Monte-Carlo Tree Search Learn to Sacrifice?” *Journal of Heuristics*, vol. 22, no. 6, pp. 783–813, Dec 2016.

- [54] H. Horn, V. Volz, D. P. Liébana, and M. Preuss, “MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–8.
- [55] D. Perez-Liebana, P. Rohlfshagen, and S. Lucas, “The Physical Travelling Salesman Problem: WCCI 2012 Competition,” in *IEEE Congress on Evolutionary Computation (CEC)*, 2012, pp. 1–8.
- [56] X. Guo, S. Singh, R. Lewis, and H. Lee, “Deep Learning for Reward Design to Improve Monte Carlo Tree Search in Atari Games,” *arXiv preprint arXiv:1604.07095*, 2016.
- [57] C. Guckelsberger, C. Salge, and S. Colton, “Intrinsically Motivated General Companion NPCs via Coupled Empowerment Maximisation,” in *Proc. IEEE Conf. Computational Intelligence in Games*. IEEE, 2016.
- [58] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, “Unifying Count-based Exploration and Intrinsic Motivation,” in *Advances in Neural Information Processing Systems*, 2016, pp. 1471–1479.
- [59] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. P. Liébana, S. Samothrakis, and S. Colton, “A Survey of Monte Carlo Tree Search Methods,” in *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4, no. 1, 2014, pp. 1–43.
- [60] Y. Björnsson and H. Finnsson, “CadiaPlayer: A Simulation-Based General Game Player,” in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, 2009, pp. 4–15.
- [61] H. Finnsson, Y. Björnsson, D. Fox, and C. P. Gomes, “Simulation-Based Approach to General Game Playing,” in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, vol. 1, no. 1, 2008, pp. 259–264.
- [62] J. Mehat and T. Cazenave, “Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing,” in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, 2010, pp. 271–277.
- [63] M. Möller, M. Schneider, M. Wegner, and T. Schaub, “Centurio, a General Game Player: Parallel, Java- and ASP-based,” in *Künstliche Intelligenz*, vol. 25, no. 1, 2010, pp. 17–24.
- [64] S. Sharma, Z. Kobti, and S. Goodwin, “Knowledge Generation for Improving Simulations in UCT for General Game Playing,” in *Proc. Adv. Artif. Intell., Auckland, New Zealand*, vol. 1, 2008, pp. 49–55.
- [65] J. P. A. M. Nijssen and M. H. M. Winands, “Enhancements for Multi-Player Monte-Carlo Tree Search,” in *Proceedings on Computers and Games, LNCS 6515, Kanazawa, Japan*, 2010, pp. 238–249.
- [66] T. Cazenave, “Multi-player Go,” in *Proceedings on Computers and Games, LNCS 5131, Beijing, China*, 2008, pp. 50–59.
- [67] T. Joppen, M. Moneke, N. Schroder, C. Wirth, and J. Furnkranz, “Informed Hybrid Game Tree Search for General Video Game Playing,” *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.
- [68] L. Kocsis and C. Szepesvári, “Bandit Based Monte-Carlo Planning,” in *European conference on machine learning*. Springer, 2006, pp. 282–293.
- [69] D. P. Liébana, J. Dieskau, M. Hünermund, S. Mostaghim, and S. M. Lucas, “Open Loop Search for General Video Game Playing,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2015, pp. 337–344.
- [70] W. Hoeffding, “Probability Inequalities for Sums of Bounded Random Variables,” *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.
- [71] T. Back, U. Hammel, and H. P. Schwefel, “Evolutionary Computation: Comments on the History and Current State,” in *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997, pp. 3–17.
- [72] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-Based Procedural Content Generation: A Taxonomy and Survey,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.

- [73] J. Liu, D. P. Liébana, and S. M. Lucas, “Rolling Horizon Coevolutionary Planning for Two-Player Video Games,” in *Proceedings of the IEEE Conference on Computational intelligence and Games (CIG)*, 2016.
- [74] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
- [75] S. Samothrakis, S. A. Roberts, D. Pérez Liébana, and S. M. Lucas, “Rolling Horizon Methods for Games with Continuous States and Actions,” in *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 2014, pp. 1–8.
- [76] N. Justesen, T. Mahlmann, and J. Togelius, “Online Evolution for Multi-action Adversarial Games,” in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 590–603.
- [77] S. M. Lucas, S. Samothrakis, and D. P. Liébana, “Fast Evolutionary Adaptation for Monte Carlo Tree Search,” in *EvoGames*, 2014.
- [78] J. Liu, D. P. Liébana, and S. M. Lucas, “Bandit-Based Random Mutation Hill-Climbing,” in *Proceedings of the Congress on Evolutionary Computation (CEC)*, 2017.
- [79] A. E. Vázquez-Núñez, A. Fernández-Leiva, P. García-Sánchez, and A. M. Mora, “Testing Hybrid Computational Intelligence Algorithms for General Game Playing,” in *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 2020, pp. 446–460.
- [80] D. Anderson, P. Rodgers, J. Levine, C. Guerrero-Romero, and D. Pérez Liébana, “Ensemble Decision Systems for General Video Game Playing,” in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.
- [81] H. Baier and P. I. Cowling, “Evolutionary MCTS for Multi-Action Adversarial Games,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.
- [82] —, “Evolutionary MCTS with Flexible Search Horizon,” in *AIIDE*, 2018, pp. 2–8.
- [83] B. Kazimipour, X. Li, and A. K. Qin, “A Review of Population Initialization Techniques for Evolutionary Algorithms,” in *IEEE Congress on Evolutionary Computation (CEC)*, 2014.
- [84] K. J. Kim, H. Choi, and S. B. Cho, “Hybrid of Evolution and Reinforcement Learning for Othello Players,” in *2007 IEEE Symposium on Computational Intelligence and Games*, April 2007, pp. 203–209.
- [85] H. Maaranen, K. Miettinen, and M. Mäkelä, “Quasi-Random Initial Population for Genetic Algorithms,” *Computers and Mathematics with Applications*, vol. 47, no. 12, pp. 1885–1895, 2004.
- [86] T. Benecke, “Tracing the Impact of the Initial Population in Evolutionary Algorithms,” *Otto von Guericke University*, 2020.
- [87] A. J. C. Gittins and J. C. Gittins, “Bandit Processes and Dynamic Allocation Indices,” *Journal of the Royal Statistical Society, Series B*, pp. 148–177, 1979.
- [88] E. J. Powley, D. Whitehouse, and P. I. Cowling, “Bandits All the Way Down: UCB1 as a Simulation Policy in Monte Carlo Tree Search,” in *IEEE Conference on Computational Intelligence and Games*, 2013, pp. 1–8.
- [89] J. Liu, J. Togelius, D. Pérez Liébana, and S. M. Lucas, “Evolving Game Skill-Depth Using General Video Game AI Agents,” in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 2299–2307.
- [90] S. W. Mahfoud, “Niching Methods for Genetic Algorithms,” Ph.D. dissertation, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1995, uMI Order No. GAX95-43663.
- [91] A. Brindle and U. of Alberta. Department of Computing Science, *Genetic Algorithms for Function Optimization*, ser. University of Alberta Department of Computing Science Technical Report Tr. Thesis (Ph.D.)—University of Alberta, 1981.
- [92] J. E. Baker, “Reducing Bias and Inefficiency in the Selection Algorithm,” in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987, pp. 14–21.

- [93] M. L. Mauldin, "Maintaining Diversity in Genetic Search," in *Proceedings of the National Conference on Artificial Intelligence*, 1984, pp. 247–250.
- [94] D. Gupta and S. Ghafir, "An Overview of Methods Maintaining Diversity in Genetic Algorithms," in *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 5, 2012, pp. 56–60.
- [95] T. Gabor, L. Belzner, T. Phan, and K. Schmid, "Preparing for the Unexpected: Diversity Improves Planning Resilience in Evolutionary Algorithms," in *2018 IEEE International Conference on Automatic Computing (ICAC)*. IEEE, 2018, pp. 131–140.
- [96] J. Lehman and K. O. Stanley, "Abandoning Objectives: Evolution Through the Search for Novelty Alone," *Evolutionary Computation*, vol. 19, no. 2, pp. 189–223, June 2011.
- [97] D. Gravina, A. Liapis, and G. N. Yannakakis, "Coupling novelty and surprise for evolutionary divergence," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 107–114.
- [98] D. Gravina, A. Liapis, and G. Yannakakis, "Surprise search: Beyond objectives and novelty," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 677–684.
- [99] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.
- [100] D. Gravina, A. Liapis, and G. N. Yannakakis, "Quality diversity through surprise," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 4, pp. 603–616, 2018.
- [101] M. Charity, M. C. Green, A. Khalifa, and J. Togelius, "Mech-elites: Illuminating the mechanic space of gvg-ai," in *International Conference on the Foundations of Digital Games*, 2020, pp. 1–10.
- [102] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.
- [103] R.-K. Balla and A. Fern, "UCT for Tactical Assault Planning in Real-Time Strategy Games," in *IJCAI*, vol. 40, 2009, p. 45.
- [104] G. Eyck and M. Müller, "Revisiting Move Groups in Monte-Carlo Tree Search," in *Springer Advances in Computer Games*, 2011, pp. 13–23.
- [105] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Determinization in Monte-Carlo Tree Search for the Card Game Dou Di Zhu," *Proc. Artif. Intell. Simul. Behav*, pp. 17–24, 2011.
- [106] D. Perez-Liebana, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the Physical Traveling Salesman Problem: Tree Search and Macro Actions," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, 2013, pp. 31–45.
- [107] G. Wallner and S. Kriglstein, "Visualization-Based Analysis of Gameplay Data — A Review of Literature," *Entertainment Computing*, vol. 4, no. 3, pp. 143 – 155, 2013.
- [108] A. Drachen and A. Canossa, "Towards Gameplay Analysis via Gameplay Metrics," in *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, ser. MindTrek '09. New York, NY, USA: ACM, 2009, pp. 202–209.
- [109] B. G. Weber, M. Mateas, and A. Jhala, "Using Data Mining to Model Player Experience," in *FDG Workshop on Evaluating Player Experience in Games*, ACM. Bordeaux, France: ACM, 06/2011 2011.
- [110] N. Hoobler, G. Humphreys, and M. Agrawala, "Visualizing Competitive Behaviors in Multi-User Virtual Environments," in *Proceedings of the Conference on Visualization '04*, ser. VIS '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 163–170.
- [111] S. Mitterhofer, C. Kruegel, E. Kirda, and C. Platzer, "Server-Side Bot Detection in Massively Multi-player Online Games," *IEEE Security Privacy*, vol. 7, no. 3, pp. 29–36, May 2009.
- [112] A. Belicza, "Scelight," in *Available online: <https://sites.google.com/site/scelight>*, 2013.
- [113] U. of York DC Labs and ESL, "Capturing the Extraordinary: Echo Brings Esports Statistics to Life," in *available online: <https://tinyurl.com/ybo7xbe4>*, October 2017.

- [114] V. Volz, D. Ashlock, S. Colton, S. Dahlskog, J. Liu, S. M. Lucas, D. P. Liébana, and T. Thompson, “Gameplay Evaluation Measures,” in *Artificial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471)*, E. André, M. Cook, M. Preuß, and P. Spronck, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 36–39.
- [115] S. M. Lucas, “Cross-Platform Games in Kotlin,” in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 774–775.
- [116] A. Liapis, C. Holmgård, G. N. Yannakakis, and J. Togelius, “Procedural Personas as Critics for Dungeon Generation,” in *European Conference on the Applications of Evolutionary Computation*. Springer, 2015, pp. 331–343.
- [117] N. Shaker, S. Asteriadis, G. N. Yannakakis, and K. Karpouzis, “Fusing Visual and Behavioral Cues for Modeling User Experience in Games,” *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1519–1531, Dec 2013.
- [118] W. Sombat, P. Rohlfshagen, and S. M. Lucas, “Evaluating the Enjoyability of the Ghosts in Ms Pac-Man,” in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 379–387.
- [119] A. Isaksen, C. Holmgård, J. Togelius, and A. Nealen, “Characterising Score Distributions in Dice Games,” *Game and Puzzle Design*, vol. 2, no. 1, 2016.
- [120] V. Volz, G. Rudolph, and B. Naujoks, “Demonstrating the feasibility of automatic game balancing,” in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 269–276.
- [121] D. P. Liébana, J. Togelius, S. Samothrakis, P. Rohlfshagen, and S. M. Lucas, “Automated Map Generation for the Physical Traveling Salesman Problem,” *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 5, pp. 708–720, Oct 2014.
- [122] C. Browne and F. Maire, “Evolutionary game design,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.
- [123] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, “Modifying MCTS for Human-Like General Video Game Playing,” in *IJCAI*, 2016, pp. 2514–2520.
- [124] A. Mendes, J. Togelius, and A. Nealen, “Hyper-Heuristic General Video Game Playing,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–8.
- [125] A. Auger and N. Hansen, “A Restart CMA Evolution Strategy with Increasing Population Size,” in *IEEE Congress on Evolutionary Computation*, 2005, pp. 1769–1776.
- [126] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, 2018, <http://gameaibook.org>.
- [127] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential Model-Based Optimization for General Algorithm Configuration,” in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523.
- [128] T. Bartz-Beielstein, *Experimental Research in Evolutionary Computation – The New Experimentalism*. Springer, 2006.
- [129] Y. Jin, “Surrogate-Assisted Evolutionary Computation: Recent Advances and Future Challenges,” *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.
- [130] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-Time Analysis of the Multiarmed Bandit Problem,” *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.
- [131] C. F. Sironi, J. Liu, and M. H. M. Winands, “Self-adaptive Monte-Carlo Tree Search in General Game Playing,” *IEEE Transactions on Games*, 2020, in press.
- [132] A. Mendes, J. Togelius, and A. Nealen, “Hyper-Heuristic General Video Game Playing,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 94–101.
- [133] S. M. Lucas, J. Liu, and D. Pérez Liébana, “The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation,” in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–9.

- [134] M. Świechowski and J. Mańdziuk, “Self-Adaptation of Playing Strategies in General Game Playing,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 367–381, Dec. 2014.
- [135] C. F. Sironi and M. H. Winands, “Analysis of Self-Adaptive Monte Carlo Tree Search in General Video Game Playing,” in *Conf. on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 397–400.
- [136] S. Bussemaker, “Self-Adaptive Multi-Objective Option Monte-Carlo Tree Search for General Video Game Play,” Ph.D. dissertation, Otto von Guericke University, 2019.
- [137] E. H. dos Santos and H. S. Bernardino, “Redundant Action Avoidance and Non-Defeat Policy in the Monte Carlo Tree Search Algorithm for General Video Game Playing,” *Proceedings do XVI Simpósio Brasileiro de Jogos e Entretenimento Digital*, 2017.
- [138] S. O. Kimbrough, M. Lu, and D. H. Wood, “Exploring the evolutionary details of a feasible-infeasible two-population ga,” in *International Conference on Parallel Problem Solving from Nature*. Springer, 2004, pp. 292–301.
- [139] C. Guerrero-Romero, A. P. Louis, and D. P. Liébana, “Beyond Playing to Win: Diversifying Heuristics for GVGAI,” in *Proc. of the Conference on Computational Intelligence and Games*, 2017.
- [140] C. Guerrero-Romero, S. M. Lucas, and D. Pérez Liébana, “Using a Team of General AI Algorithms to Assist Game Design and Testing,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.
- [141] B. S. Santos and H. S. Bernardino, “Game State Evaluation Heuristics in General Video Game Playing,” in *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2018, pp. 147–14709.
- [142] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen, “Frame Skip is a Powerful Parameter for Learning to Play Atari,” in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [143] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, “Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents,” *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.
- [144] D. A. Berry and B. Fristedt, *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability)*, *Population and Community Biology*, 1st ed. Springer, Oct. 1985.
- [145] E. Galván, O. Gorshkova, P. Mooney, F. V. Amenyro, and E. Cuevas, “Statistical Tree-based Population Seeding for Rolling Horizon EAs in General Video Game Playing,” *arXiv preprint arXiv:2008.13253*, 2020.
- [146] I. Bravi, D. Pérez Liébana, S. M. Lucas, and J. Liu, “Shallow Decision-Making Analysis in General Video Game Playing,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.
- [147] I. Bravi and S. M. Lucas, “Rinascimento: using event-value functions for playing splendor,” in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 283–290.
- [148] S. M. Lucas, “Game AI Research with Fast Planet Wars Variants,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–4.
- [149] D. Ashlock, D. P. Liébana, and A. Saunders, “General Video Game Playing Escapes the No Free Lunch Theorem,” in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2017, pp. 17–24.
- [150] A. Gustafsson, “Winner Prediction of Blood Bowl 2 Matches with Binary Classification,” *Malmö universitet/Teknik och samhälle*, 2019.
- [151] H. He and E. A. Garcia, “Learning from Imbalanced Data,” *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

- [152] J. Zhu, S. Rosset, H. Zou, and T. Hastie, “Multi-class AdaBoost,” *Ann Arbor*, vol. 1001, no. 48109, p. 1612, 2006.
- [153] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API Design for Machine Learning Software: Experiences from the Scikit-learn Project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.
- [154] C. Guckelsberger, C. Salge, J. Gow, and P. Cairns, “Predicting Player Experience Without the Player.: An Exploratory Study,” in *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY ’17. New York, NY, USA: ACM, 2017, pp. 305–315.
- [155] N. Justesen, T. Mahlmann, S. Risi, and J. Togelius, “Playing Multiaction Adversarial Games: Online Evolutionary Planning Versus Tree Search,” *IEEE Transactions on Games*, vol. 10, no. 3, pp. 281–291, 2017.
- [156] B. Santos, H. Bernardino, and E. Hauck, “An Improved Rolling Horizon Evolution Algorithm with Shift Buffer for General Game Playing,” in *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2018, pp. 31–316.
- [157] X. Tong, W. Liu, and B. Li, “Enhancing Rolling Horizon Evolution with Policy and Value Networks,” in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.
- [158] M. H. Segler, M. Preuss, and M. P. Waller, “Planning Chemical Syntheses with Deep Neural Networks and Symbolic AI,” *Nature*, vol. 555, no. 7698, p. 604, 2018.
- [159] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, “Mastering atari, go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.
- [160] K. Kunanusont, S. M. Lucas, and D. Pérez Liébana, “Modelling Player Experience with the N-Tuple Bandit Evolutionary Algorithm,” in *Artificial intelligence and Interactive Digital Entertainment (AI-IDE)*, 2018.
- [161] I. Bravi, D. Perez-Liebana, S. M. Lucas, and J. Liu, “Rinascimento: Optimising statistical forward planning agents for playing splendor,” in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.
- [162] S. Ontañón, “Combinatorial Multi-armed Bandits for Real-Time Strategy Games,” *Journal of AI Research*, vol. 58, pp. 665–702, March 2017.
- [163] D. Ashlock, *Evolutionary Computation for Modeling and Optimization*. Springer Science & Business Media, 2006.
- [164] C. F. Sironi and M. H. Winands, “Comparing Randomization Strategies for Search-Control Parameters in MCTS,” in *Conference on Games (COG)*. IEEE, 2019.
- [165] M. Dube, S. Houghten, and D. Ashlock, “Parameter Selection for Modeling of Epidemic Networks,” in *2018 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE, 2018, pp. 1–8.
- [166] C. Resnick, W. Eldridge, D. Ha, D. Britz, J. Foerster, J. Togelius, K. Cho, and J. Bruna, “Pommerman: A Multi-agent Playground,” in *MARLO Workshop, AIIIDE-WS Proceedings*, 2018, pp. 1–6.
- [167] T. Osogami and T. Takahashi, “Real-time tree search with pessimistic scenarios: Winning the neurips 2018 pommerman competition,” in *Asian Conference on Machine Learning*. PMLR, 2019, pp. 583–598.
- [168] H. Zhou, Y. Gong, L. Mugrai, A. Khalifa, A. Nealen, and J. Togelius, “A Hybrid Search Agent in Pommerman,” in *Proceedings of the 13th International Conference on the Foundations of Digital Games*. ACM, 2018, p. 46.
- [169] P. Peng, L. Pang, Y. Yuan, and C. Gao, “Continual Match Based Training in Pommerman: Technical Report,” *arXiv preprint arXiv:1812.07297*, 2018.
- [170] A. Malysheva, T. T. Sung, C.-B. Sohn, D. Kudenko, and A. Shpilman, “Deep Multi-Agent Reinforcement Learning with Relevance Graphs,” *arXiv preprint arXiv:1811.12557*, 2018.

- [171] C. Resnick, R. Raileanu, S. Kapoor, A. Peysakhovich, K. Cho, and J. Bruna, “Backplay: “Man muss immer umkehren”,” *arXiv preprint arXiv:1807.06919*, 2018.
- [172] C. Gao, P. Hernandez-Leal, B. Kartal, and M. E. Taylor, “Skynet: A Top Deep RL Agent in the Inaugural Pommerman Team Competition,” *arXiv preprint arXiv:1905.01360*, 2019.
- [173] B. Kartal, P. Hernandez-Leal, and M. E. Taylor, “Using Monte Carlo Tree Search as a Demonstrator within Asynchronous Deep RL,” *arXiv preprint arXiv:1812.00045*, 2018.
- [174] B. Kartal, P. Hernandez-Leal, C. Gao, and M. E. Taylor, “Safer Deep RL with Shallow MCTS: A Case Study in Pommerman,” *arXiv preprint arXiv:1904.05759*, 2019.
- [175] Midjiwan AB, “The Battle of Polytopia,” 2016.
- [176] Firaxis, “Civilization,” 1995 – 2020.
- [177] F. Arnold, B. Horvat, and A. Sacks, “Freeciv Learner: a Machine Learning Project Utilizing Genetic Algorithms,” *Georgia Institute of Technology, Atlanta*, 2004.
- [178] I. Watson, D. Azhar, Y. Chuyang, W. Pan, and G. Chen, “Optimization in Strategy Games: Using Genetic Algorithms to Optimize City Development in FreeCiv,” *University of Auckland*, 2008.
- [179] S. Wender and I. Watson, “Using Reinforcement Learning for City Site Selection in the Turn-Based Strategy Game Civilization IV,” in *2008 IEEE Symposium On Computational Intelligence and Games*. IEEE, 2008, pp. 372–377.
- [180] Robot Entertainment, “Hero Academy,” 2012.
- [181] N. Justesen, L. M. Uth, C. Jakobsen, P. D. Moore, J. Togelius, and S. Risi, “Blood Bowl: A New Board Game Challenge and Competition for AI,” in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.
- [182] Jervis Johnson, “Blood Bowl,” 1986.
- [183] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, “A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft,” *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.
- [184] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell *et al.*, “Alphastar: Mastering the Real-Time Strategy Game Starcraft II,” *DeepMind blog*, p. 2, 2019.
- [185] S. Ontañón, N. A. Barriga, C. R. Silva, R. O. Moraes, and L. H. Lelis, “The First Microrts Artificial Intelligence Competition,” *AI Magazine*, vol. 39, no. 1, pp. 75–83, 2018.
- [186] G. M. J. Chaslot, M. H. Winands, H. J. V. D. HERIK, J. W. Uiterwijk, and B. Bouzy, “Progressive Strategies for Monte-Carlo Tree Search,” *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.
- [187] S. Gelly and Y. Wang, “Exploration Exploitation in Go: UCT for Monte-Carlo Go,” in *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*. hal-00115330, 2006.
- [188] D. Silver *et al.*, “Mastering The Game of Go Without Human Knowledge,” *Nature*, vol. 550, no. 7676, p. 354, 2017.
- [189] G. Engelstein and I. Shalev, *Building Blocks of Tabletop Game Design: An Encyclopedia of Mechanisms*. CRC Press LLC, 2019.
- [190] M. Genesereth, N. Love, and B. Pell, “General game playing: Overview of the AAAI competition,” *AI Magazine*, vol. 26, no. 2, p. 62, 2005.
- [191] E. Piette, D. J. Soemers, M. Stephenson, C. F. Sironi, M. H. Winands, and C. Browne, “Ludii–The Ludemic General Game System,” *arXiv preprint arXiv:1905.05013*, 2019.
- [192] M. Lanctot *et al.*, “OpenSpiel: A Framework for Reinforcement Learning in Games,” *CoRR*, vol. abs/1908.09453, 2019.

- [193] L. J. P. de Araujo, M. Charikova, J. E. Sales, V. Smirnov, and A. Thapaliya, “Towards a Game-Independent Model and Data-Structures in Digital Board Games: an Overview of the State-of-the-Art,” in *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 2019, pp. 1–8.
- [194] M. Campbell *et al.*, “Deep Blue,” *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.
- [195] J. Kowalski, M. Mika, J. Sutowicz, and M. Szykuła, “Regular Boardgames,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1699–1706.
- [196] J. Henry, *Tabletop Simulator*. Berserk Games, 2015.
- [197] M. Moravčík *et al.*, “Deepstack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker,” *Science*, vol. 356, no. 6337, pp. 508–513, 2017.
- [198] T. Cazenave and V. Ventos, “The alpha-mu Search Algorithm for the Game of Bridge,” *arXiv preprint arXiv:1911.07960*, 2019.
- [199] N. Bard *et al.*, “The Hanabi Challenge: A New Frontier for AI Research,” *Artificial Intelligence*, vol. 280, p. 103216, 2020.
- [200] J. Serrino, M. Kleiman-Weiner, D. C. Parkes, and J. Tenenbaum, “Finding Friend and Foe in Multi-Agent Games,” in *Advances in Neural Information Processing Systems*, 2019, pp. 1251–1261.
- [201] M. Eger and C. Martens, “A Study of AI Agent Commitment in One Night Ultimate Werewolf with Human Players,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, 2019, pp. 139–145.
- [202] I. Szita, G. Chaslot, and P. Spronck, “Monte-Carlo Tree Search in Settlers of Catan,” in *Advances in Computer Games*. Springer, 2009, pp. 21–32.
- [203] R. Gibson, N. Desai, and R. Zhao, “An Automated Technique for Drafting Territories in the Board Game Risk,” in *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2010.
- [204] P. S. Chacón and M. Eger, “Pandemic as a Challenge for Human-AI Cooperation,” in *Proceedings of the AIIDE workshop on Experimental AI in Games*, 2019.
- [205] K. Sfikas and A. Liapis, “Collaborative Agent Gameplay in the Pandemic Board Game,” in *International Conference on the Foundations of Digital Games*, 2020, pp. 1–11.
- [206] F. de Mesentier Silva, S. Lee, J. Togelius, and A. Nealen, “AI-Based Playtesting of Contemporary Board Games,” in *Proceedings of the International Conference on the Foundations of Digital Games - FDG’17*. ACM Press, 2017.
- [207] M. Guzdial, D. Acharya, M. Kreminski, M. Cook, M. Eladhari, A. Liapis, and A. Sullivan, “Tabletop roleplaying games as procedural content generators,” in *International Conference on the Foundations of Digital Games*, 2020, pp. 1–9.
- [208] M. Leacock, *Pandemic*. Z-Man Games, Inc., 2008.
- [209] Fantasy Flight Publishing, Inc., *Descent: Journeys in the Dark 2nd Edition*. Diamond Comic Distributors, 2012.
- [210] K.-J. Wrede, *Carcassonne*. Hans im Glück, 2000.
- [211] K. Teuber, *The Settlers of Catan*. Mayfair Games, 1995.
- [212] S. Kanai, *Love Letter*. Alderac Entertainment Group, 2012.
- [213] M. Robbins, *Uno*. AMIGO, 1971.
- [214] D. Cabrero *et al.*, *Virus!* El Dragón Azul, 2015.
- [215] M. Inman *et al.*, *Exploding Kittens*. Ad Magic, Inc., 2015.
- [216] C. Raimbault, *Colt Express*. Ludonaute, 2014.
- [217] B. Glassco *et al.*, *Betrayal at House on the Hill*. Avalon Hill Games, Inc., 2004.
- [218] A. Looney and K. Looney, *Fluxx*. Looney Labs, 1997.

- [219] R. Garfield, *Magic: The Gathering*. Wizards of the Coast, 1993.
- [220] A. J. Summerville and M. Mateas, “Mystical Tutor: A Magic: The Gathering Design Assistant Via Denoising Sequence-To-Sequence Learning,” in *Twelfth artificial intelligence and interactive digital entertainment conference*, 2016.
- [221] J. M. Gonzalez-Castro and D. Pérez Liébana, “Opponent Models Comparison for 2 Players in GV-GAI Competitions,” in *2017 9th Computer Science and Electronic Engineering (CEECE)*. IEEE, 2017, pp. 151–156.
- [222] Z. Tang, Y. Zhu, D. Zhao, and S. M. Lucas, “Enhanced rolling horizon evolution algorithm with opponent model learning,” *IEEE Transactions on Games*, 2020.
- [223] R. Sun, “Environmental curriculum learning for efficiently achieving superhuman play in games,” Ph.D. dissertation, University of Illinois, 2020.
- [224] H. Meisheri, O. Shelke, R. Verma, and H. Khadilkar, “Accelerating Training in Pommerman with Imitation and Reinforcement Learning,” *arXiv preprint arXiv:1911.04947*, 2019.
- [225] S. Agarwal, G. Wallner, and F. Beck, “Bombalytics: Visualization of competition and collaboration strategies of players in a bomb laying game,” in *Computer Graphics Forum*, vol. 39, no. 3. Wiley Online Library, 2020, pp. 89–100.
- [226] H. Meisheri and H. Khadilkar, “Sample Efficient Training in Multi-Agent Adversarial Games with Limited Teammate Communication,” *arXiv preprint arXiv:2011.00424*, 2020.
- [227] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil *et al.*, “Vector-Based Navigation Using Grid-Like Representations in Artificial Agents,” *Nature*, vol. 557, no. 7705, p. 429, 2018.
- [228] D. Silver *et al.*, “A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go Through Self-Play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [229] S. Yeo, S. Oh, and M. Lee, “Accelerating Deep Reinforcement Learning Using Human Demonstration Data Based on Dual Replay Buffer Management and Online Frame Skipping,” in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2019, pp. 1–8.
- [230] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, “Imitation Learning: A Survey of Learning Methods,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.
- [231] T. Veale and M. Cook, *Twitterbots: Making Machines that Make Meaning*. MIT Press, 2018.
- [232] M. B. Hoy, “Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants,” *Medical Reference Services Quarterly*, vol. 37, no. 1, pp. 81–88, 2018.
- [233] M. Cook and S. Colton, “Redesigning Computationally Creative Systems for Continuous Creation,” in *International Conference on Computational Creativity*, 2018.
- [234] A. Schlesinger, K. P. O’Hara, and A. S. Taylor, “Let’s Talk About Race: Identity, Chatbots, and AI,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 315.
- [235] J. Huang, M. Zhou, and D. Yang, “Extracting Chatbot Knowledge from Online Discussion Forums,” in *IJCAI*, vol. 7, 2007, pp. 423–428.
- [236] J. Weizenbaum, “ELIZA – A Computer Program For The Study Of Natural Language Communication Between Man and Machine,” *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.
- [237] E. Short, “Galatea,” *Electronic Literature Collection: Volume One*, 2000.
- [238] —, “NPC Conversation Systems,” *IF Theory Reader, Transcript On Press*, pp. 331–358, 2011.
- [239] M. Wigdahl, “Aotearoa,” 2010.
- [240] S. Ontanon, “SHRDLU: A Game Prototype Inspired by Winograd’s Natural Language Understanding Work,” in *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.

- [241] Z. Zukowski and C. Carr, “Generating Black Metal and Math Rock: Beyond Bach, Beethoven, and Beatles,” *arXiv preprint arXiv:1811.06639*, 2018.
- [242] M. Cook, S. Colton, and J. Gow, “The ANGELINA Videogame Design System—Part I,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 2, pp. 192–203, 2017.
- [243] —, “The ANGELINA Videogame Design System—Part II,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 3, pp. 254–266, 2017.
- [244] G. I. Parisi *et al.*, “Continual Lifelong Learning with Neural Networks: A Review,” *Neural Networks*, 2019.
- [245] D. Lopez-Paz *et al.*, “Gradient Episodic Memory for Continual Learning,” in *Adv. in Neural Inf. Processing Systems*, 2017, pp. 6467–6476.
- [246] A. Chaudhry *et al.*, “Efficient Lifelong Learning with A-Gem,” *arXiv preprint arXiv:1812.00420*, 2018.
- [247] R. Aljundi, M. Rohrbach, and T. Tuytelaars, “Selfless Sequential Learning,” *arXiv preprint arXiv:1806.05421*, 2018.
- [248] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, “Progress & compress: A scalable framework for continual learning,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 4528–4537.
- [249] J. Kirkpatrick *et al.*, “Overcoming Catastrophic Forgetting in Neural Networks,” *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.
- [250] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni, “Don’t Forget, There Is More Than Forgetting: New Metrics For Continual Learning,” *arXiv preprint arXiv:1810.13166*, 2018.
- [251] S. Farquhar and Y. Gal, “Towards Robust Evaluations of Continual Learning,” *arXiv preprint arXiv:1805.09733*, 2018.
- [252] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [253] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the Game of Go with Deep Neural Networks and Tree Search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [254] T. Anthony, Z. Tian, and D. Barber, “Thinking Fast And Slow With Deep Learning And Tree Search,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5360–5370.
- [255] D. Jiang, E. Ekwedike, and H. Liu, “Feedback-based tree search for reinforcement learning,” in *International conference on machine learning*. PMLR, 2018, pp. 2284–2293.
- [256] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch, “Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control,” *arXiv preprint arXiv:1811.01848*, 2018.
- [257] B. Settles, “Active Learning Literature Survey,” University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.
- [258] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, “Go-Explore: a New Approach for Hard-Exploration Problems,” *arXiv preprint arXiv:1901.10995*, 2019.
- [259] B. Pell, “A Strategic Metagame Player for General Chess-Like Games,” in *Computational Intelligence*, vol. 12, no. 1, 1996, pp. 177–198.
- [260] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, “Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning,” in *Advances in Neural Information Processing Systems*, 2014, pp. 3338–3346.






- [261] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, “The 2014 General Video Game Playing Competition,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, pp. 229–243, 2015.
- [262] A. J. Champandard, “Monte-Carlo Tree Search in TOTAL WAR: ROME II’s Campaign AI,” *AIGameDev.com*: <http://aigamedev.com/open/coverage/mcts-rome-ii>, 2014.
- [263] S. M. Lucas, J. Liu, and D. Pérez Liébana, “Efficient Noisy Optimisation with the Multi-Sample and Sliding Window Compact Genetic Algorithms,” in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–8.
- [264] A. Braylan and R. Miikkulainen, “Object-Model Transfer in the General Video Game Domain,” in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016, pp. 136–142.
- [265] K. Kunanusont, S. M. Lucas, and D. Pérez Liébana, “General Video Game AI: Learning from Screen Capture,” in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 2078–2085.
- [266] K. Narasimhan, R. Barzilay, and T. S. Jaakkola, “Deep Transfer in Reinforcement Learning by Language Grounding,” *CoRR*, vol. abs/1708.00133, pp. 849–874, 2017.
- [267] W. Woof and K. Chen, “Learning to Play General Video-Games via an Object Embedding Network,” in *Computational Intelligence and Games (CIG), 2018 IEEE Conference on*, 2018, p. to appear.
- [268] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Pérez Liébana, “Deep Reinforcement Learning for General Video Game AI,” *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug 2018.
- [269] I. Leite, C. Martinho, and A. Paiva, “Social Robots for Long-Term Interaction: A Survey,” *International Journal of Social Robotics*, vol. 5, no. 2, pp. 291–308, 2013.
- [270] S. K. Jain and P. Singh, “Systematic Survey on Sentiment Analysis,” in *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. IEEE, 2019, pp. 561–565.
- [271] A. Baldominos, Y. Saez, and P. Isasi, “On the Automated, Evolutionary Design of Neural Networks: Past, Present, and Future,” *Neural Computing and Applications*, pp. 1–27, 2019.
- [272] G. F. Miller, P. M. Todd, and S. U. Hegde, “Designing Neural Networks using Genetic Algorithms,” in *ICGA*, vol. 89, 1989, pp. 379–384.
- [273] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks Through Augmenting Topologies,” *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.
- [274] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, “Real-Time Neuroevolution in the NERO Video Game,” *IEEE transactions on evolutionary computation*, vol. 9, no. 6, pp. 653–668, 2005.
- [275] J. Gauci and K. O. Stanley, “Autonomous Evolution of Topographic Regularities in Artificial Neural Networks,” *Neural computation*, vol. 22, no. 7, pp. 1860–1898, 2010.
- [276] J. Reisinger, E. Bahceci, I. Karpov, and R. Miikkulainen, “Coevolving Strategies for General Game Playing,” in *2007 IEEE Symposium on Computational Intelligence and Games*. IEEE, 2007, pp. 320–327.
- [277] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone, “HyperNEAT-GGP: A HyperNEAT-Based Atari General Game Player,” in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 2012, pp. 217–224.
- [278] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A Neuroevolution Approach to General Atari Game Playing,” *IEEE Trans. on CI and AI in Games*, vol. 6:4, pp. 355–366, 2014.
- [279] S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, “Neuroevolution for General Video Game Playing,” in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015, pp. 200–207.
- [280] N. J. Radcliffe, “Genetic Set Recombination and Its Application to Neural Network Topology Optimisation,” *Neural Computing & Applications*, vol. 1, no. 1, pp. 67–90, 1993.

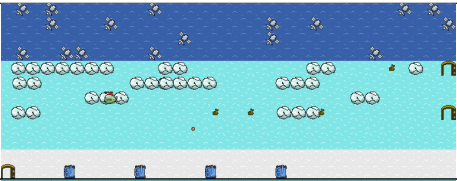




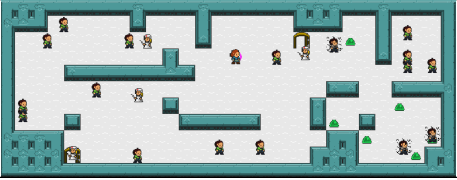

- [281] A. H. Wright, “Genetic Algorithms for Real Parameter Optimization,” in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 205–218.
- [282] J. Renz and X. Ge, “Physics Simulation Games,” *Handbook of Digital Games and Entertainment Technologies*, pp. 77–95, 2017.
- [283] D. Pérez Liébana, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, “Solving the Physical Traveling Salesman Problem: Tree Search and Macro Actions,” *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 6:1, pp. 31–45, 2014.
- [284] D. Apeldoorn and G. Kern-Isberner, “Towards an Understanding of What is Learned: Extracting Multi-Abstraction-Level Knowledge from Learning Agents,” in *The Thirtieth International Flairs Conference*, 2017.
- [285] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, “Learning Latent Dynamics for Planning from Pixels,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 2555–2565.
- [286] L. Buesing, T. Weber, S. Racaniere, S. M. A. Eslami, D. Rezende, D. P. Reichert, F. Viola, F. Besse, K. Gregor, D. Hassabis, and D. Wierstra, “Learning and Querying Fast Generative Models for Reinforcement Learning,” *arXiv*, feb 2018.
- [287] A. Dockhorn and D. Apeldoorn, “Forward Model Approximation for General Video Game Learning,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.
- [288] A. Baldominos, Y. Saez, and P. Isasi, “Evolutionary Convolutional Neural Networks: An Application to Handwriting Recognition,” *Neurocomputing*, vol. 283, pp. 38–52, 2018.
- [289] D. Ashlock, E.-Y. Kim, and D. Pérez Liébana, “Toward General Mathematical Game Playing Agents,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–7.
- [290] S. Kelly and M. I. Heywood, “Emergent Tangled Graph Representations for Atari Game Playing Agents,” in *Genetic Programming - 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings*, 2017, pp. 64–79.
- [291] G. Harik, F. G. Lobo, and D. E. Goldberg, “The Compact Genetic Algorithm,” in *IEEE Transactions on Evolutionary Computation*, 1998, pp. 523–528.
- [292] T. Friedrich, T. Kötzing, M. S. Krejca, and A. M. Sutton, “The Compact Genetic Algorithm is Efficient Under Extreme Gaussian Noise,” *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 3, pp. 477–490, June 2017.
- [293] D. Perez-Liebana, E. Powley, D. Whitehouse, S. Samothrakis, S. Lucas, and P. I. Cowling, “The 2013 Multi-objective Physical Travelling Salesman Problem Competition,” in *IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 2314–2321.
- [294] N. Hansen and A. Ostermeier, “Completely Derandomized Self-Adaptation in Evolution Strategies,” *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, Jun. 2001.
- [295] D. P. Liébana, S. Samothrakis, and S. M. Lucas, “Online and Offline Learning in Multi-Objective Monte Carlo Tree Search,” in *2013 IEEE Conference on Computational Intelligence in Games (CIG), Niagara Falls, ON, Canada, August 11-13, 2013*, 2013, pp. 1–8.
- [296] D. P. Liébana, S. Mostaghim, S. Samothrakis, and S. M. Lucas, “Multiobjective Monte Carlo Tree Search for Real-Time Games,” *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 7, no. 4, pp. 347–360, 2015.
- [297] W. Wang and M. Sebag, “Multi-Objective Monte-Carlo Tree Search,” in *ACML*, ser. JMLR Proceedings, S. C. H. Hoi and W. L. Buntine, Eds., vol. 25, 2012, pp. 507–522.
- [298] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, *A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 849–858.




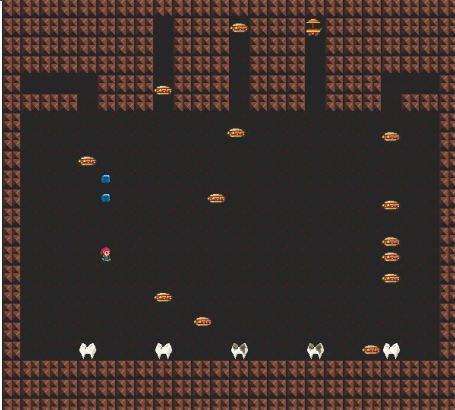
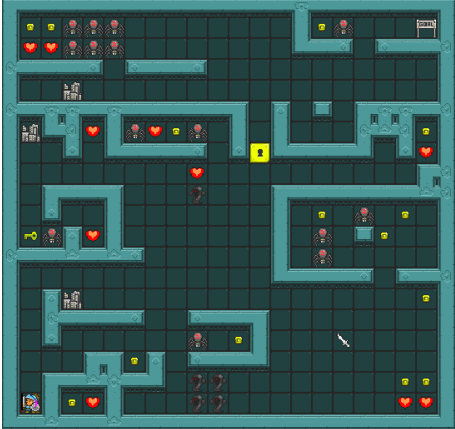
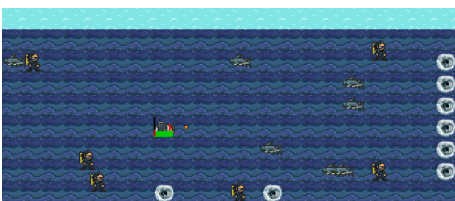
- [299] D. P. Liébana, S. Mostaghim, and S. M. Lucas, “Multi-Objective Tree Search Approaches for General Video Game Playing,” in *IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016*, 2016, pp. 624–631.
- [300] A. Khalifa, M. Preuss, and J. Togelius, “Multi-objective Adaptation of a Parameterized GVGA Agent Towards Several Games,” in *9th International Conference on Evolutionary Multi-Criterion Optimization – Volume 10173*, ser. EMO 2017, New York, NY, USA, 2017, pp. 359–374.
- [301] N. Beume, B. Naujoks, and M. Emmerich, “SMS-EMOA: Multiobjective Selection Based on Dominated Hypervolume,” *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653 – 1669, 2007.
- [302] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, “Multiobjective Evolutionary Algorithms: A Survey of the State of the Art,” *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011.
- [303] A. Syberfeldt, A. Ng, R. I. John, and P. Moore, “Evolutionary Optimisation of Noisy Multi-Objective Problems Using Confidence-Based Dynamic Resampling,” *European Journal of Operational Research*, vol. 204, no. 3, pp. 533 – 544, 2010.
- [304] E. Hughes, “Evolutionary multi-objective ranking with uncertainty and noise,” in *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, ser. EMO '01. London, UK, UK: Springer-Verlag, 2001, pp. 329–343.
- [305] L. Panait and S. Luke, “Cooperative Multi-Agent Learning: The State of the Art,” *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, Nov. 2005.
- [306] C. Wang, P. Chen, Y. Li, C. Holmgård, and J. Togelius, “Portfolio Online Evolution in StarCraft,” in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
- [307] N. Justesen, T. Mahlmann, S. Risi, and J. Togelius, “Playing Multi-Action Adversarial Games: On-line Evolutionary Planning versus Tree Search,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, pp. 1–1, 2017.

Appendix A

20 GVGA! Games

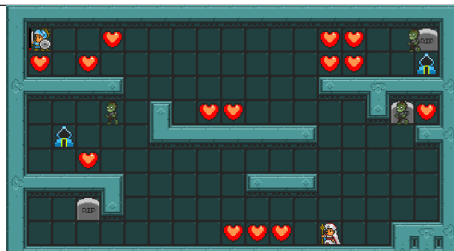
Game	Description	Image
Aliens	The player controls a spaceship at the bottom of the screen, and must shoot all aliens which spawn at the top and slowly descend towards the player, while avoiding the bombs the aliens drop. They score points for killing aliens and for destroying protective bases.	
Bait	The player navigates a small sokoban-style maze to find a key and exit through the door.	
Butterflies	The player must collect all butterflies flying randomly around a meadow, gaining points for each they collect. If butterflies collide with cocoons spread out through the level, more butterflies spawn - if all cocoons are open, however, the player loses.	
Camel Race	The player must reach the exit on the opposite side of the level first, before all of the other NPCs.	
Chase	The player must catch NPC sprites which run away from them, and win when they catch them all. However, the NPCs leave carcasses behind when caught, and if another NPC collides with a carcass, they become angry and hunt the player, killing them if caught.	

Chopper	The player must protect satellites at the top of the screen from tanks spawning and shooting from the bottom. They have limited ammo and must collect more to be able to destroy all tanks and win the game.	
Crossfire	The player must navigate a dangerous maze to the exit, avoiding the cannons which shoot each in a chosen direction.	
Dig Dug	A complex game where the player can destroy obstacles to make new paths through the level. In doing so, they should collect all loot and kill all enemies in order to win.	
Escape	The player must navigate a sokoban-style maze to reach the exit, while avoiding the holes.	
Hungry Birds	The player must navigate the maze to reach the exit. They have a limited time to do so, dictated by a hunger bar. Hunger can be replenished with resources around the level, to help the player make it to the end.	
Infection	The player must help control a disease rapidly spreading through citizens.	
Intersection	The player finds themselves in the corner of a busy intersection, which they must traverse (avoiding the incoming cars) to make it to the exit. If the player gets hit by a car or arrives at the exit, they respawn at the beginning. Each trip to the exit earns them points, and the goal is to obtain as many points as possible in the time limit.	

Lemmings	Lemmings spawn in one corner of the level, and the player must carve a safe path for them to the exit.	
Missile Command	Cities at the bottom are threatened by falling missiles. The player must destroy the missiles before the missiles destroy all of the cities, earning points in doing so.	
Modality	The player can only move onto the colour they are currently on, or change colour through the transition point (pictured in the middle). Their goal is to move the tree into the hole.	
Plaque Attack	Similar to Missile Command, teeth at the bottom are attacked by fast food spawning from the top. The player can destroy the food approaching, or heal the broken teeth (teeth break if they come into contact with the food). The player wins if they survive, and lose if all teeth break.	
Roguelike	The player must navigate a complex dungeon to find keys which unlock doors to the exit, while fighting enemies, collecting coins and health points and avoiding a multitude of traps.	
Seaquest	The player navigates a submarine, with the mission of picking up divers which spawn at the bottom (4 at a time maximum) and bring them to the top for a large reward. They must avoid the sharks and other dangers in the waters, while managing their oxygen bar (replenished at the surface).	

**Survive
Zombies**

The player must survive until the time runs out by avoiding the zombies spawning at different points in the level, collecting hearts to increase their health and using the priests to defend against the zombies.



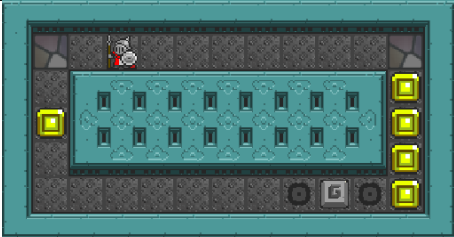

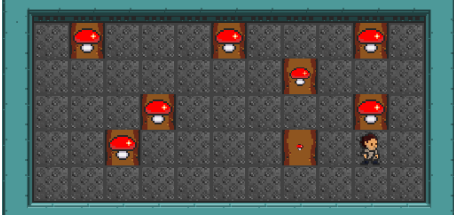

**Wait for
Breakfast**

At some point during the game, a waiter comes from the kitchen and delivers food at a table. The player must sit at that location (and only there) and wait at that location until the end of the game to win.



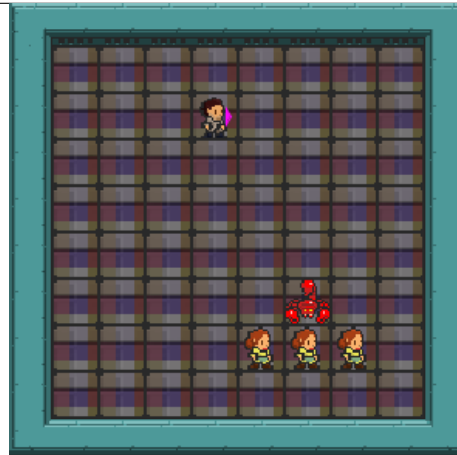
Appendix B

Deceptive Games

Game	Description	Image
Decepti Coins	The player must reach the exit. There are 2 paths to do so, one with few immediate rewards, one longer with more rewards. Once a path is chosen, the access to the other is blocked.	
Decepti Zelda	The player must navigate a Zelda-like dungeon, find a key and exit through the door, while fighting enemies on the way. There are 2 doors to choose from, one on a safe path that gives few points, and another behind several enemies and challenges which gives more points.	
Flower	Several flowers in the level start from the stage of seed, growing through several stages until a maximum. Once collected, the flower resets to the seed stage. They give more points the bigger they are. The goal of the player is to collect as many points as possible in the time limit.	
Invest	The player can collect coins in the level and give them to one of the 3 NPCs, which trigger a random event at some point in the future rewarding the player with more coins than they initially lost. The goal of the player is to obtain as many points as possible.	

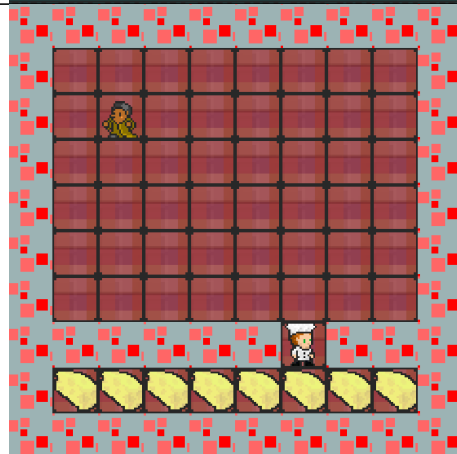
**Sister
Savior**

Several hostages are guarded by enemies. The player can save the hostages for a small reward, or kill them for a larger reward. If all hostages are saved, the player can kill the enemies and win the game.




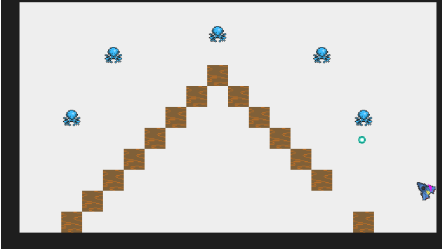
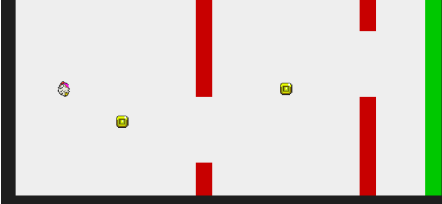
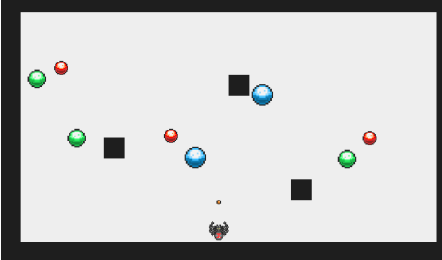
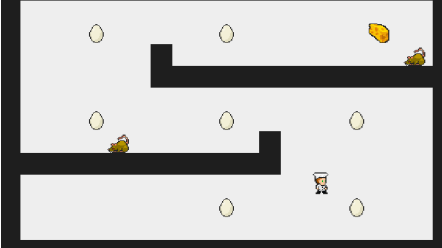
**Wafer
Thin
Mints**

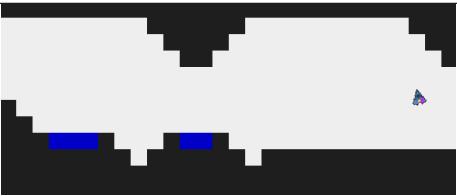



The player can collect rewards through the level, but if they collect too many they die and lose the game.



Appendix C

Physics-Based Games

Game	Description	Image
Artillery	The player controls a cannon with a fixed position that can rotate left or right, and shoots balls (affected by gravity) which destroy some parts of the level (and earns 1 point), as well as targets. The player wins when all targets have been destroyed.	
Asteroids	The player controls a spaceship that can rotate, accelerate and decelerate, flying in space with the mission of destroying all targets, and any other breakable objects for 1 point each. The player wins when all targets have been destroyed. Clone of classic "Asteroids".	
Bird	The player controls a bird affected by gravity, with only 1 action for control (jump). They must navigate a series of pipes forming narrow corridors and collect coins for points. The player wins when they reach the exit. Clone of "Flappy Bird".	
Bubble	The player controls a shooter at the bottom of the screen which can move left and right and shoot straight up. There are balls of 3 sizes bouncing around the level. Hitting bigger balls splits them into 2 of the next smaller size. Hitting the smallest balls destroys them. Each hit earns the player points and they win when all balls are destroyed. Clone of "Bubble Trouble".	
Candy	The player controls a jumping avatar which can also move left and right. They navigate several platforms, collecting points and avoiding enemies, to reach the exit.	

Lander	The player controls a spaceship which falls with a particular speed, and they can rotate to redirect it and adjust the speed. The player wins when the spaceships lands on particular platforms, avoiding the obstacles in the level. Clone of “Lunar Lander”.	
Mario	The player controls a jumping avatar that can also move left and right. They must navigate several platforms in complex levels, collecting coins and avoiding enemies to reach the exit. Clone of “Super Mario”.	
Pong	The player controls a paddle on the left of the screen and must catch a ball bouncing around the level, so as it hits the opponent wall (behind their paddle), and not the player’s wall. Clone of “Pong”.	
PTSP	The player controls a spaceship that can rotate, accelerate and decelerate. They must collect all targets in the level as fast as possible, while avoiding aliens. Clone of “Physical Travelling Salesman Problem”.	
Racing	The player controls a car moving at constant speed (they can only rotate) and must navigate a maze, collecting points along the way, until they reach the exit.	