Tribes: A New Turn-Based Strategy Game for AI Research

Diego Perez-Liebana, Yu-Jhen Hsu, Stavros Emmanouilidis,
Bobby Dewan Akram Khaleque, Raluca D. Gaina
School of Electronic Engineering and Computer Science
Queen Mary University of London, UK

Abstract

This paper introduces Tribes, a new turn-based strategy game framework. Tribes is a multi-player, multi-agent, stochastic and partially observable game that involves strategic and tactical combat decisions. A good playing strategy requires the management of a technology tree, build orders and economy. The framework provides a Forward Model, which can be used by Statistical Forward Planning methods. This paper describes the framework and the opportunities for Game AI research it brings. We further provide an analysis on the action space of this game, as well as benchmarking a series of agents (rule based, one step look-ahead, Monte Carlo, Monte Carlo Tree Search, and Rolling Horizon Evolution) to study their relative playing strength. Results show that although some of these agents can play at a decent level, they are still far from human playing strength.

1 Introduction

While work in the Game AI field started with domains comprising of simple state representation and action sets (such as Chess, Checkers or Go), nowadays researchers can find a great variety of games with partial observability, multiple players or huge branching factors. Strategy games form a sub-genre that incorporates several of these features at once and are therefore of interest to Game AI research. However, the added complexity in these environments results in difficulties in obtaining the engine functionality necessary to run all classes of AI methods, and, of particular interest, Statistical Forward Planning (SFP) algorithms.

The application of SFP methods to games has been a prolific topic for AI research. During many years, tree search methods like Monte Carlo Tree Search (MCTS) and (more recently) Rolling Horizon Evolutionary Algorithms (RHEA) have been investigated for decision making in board, real-time and turn-based games. One particular characteristic of SFP methods is that they require, in order to allow planning and tree search, a Forward Model (FM) of the game that permits simulating future states, providing a previous state and an action to execute. Having a FM that allows this is not straightforward. Apart from design considerations, the two main methods that provide the functionality required by SFP (`next` - to advance the game state, and `copy` - to clone it) can be computationally expensive in execution time and memory. That is one of the reasons why the more complex environments (e.g Starcraft II (Blizzard Entertainment 2010)) do not provide a FM API for AI agents.

There has been some previous work on frameworks that incorporate FM access for SFP in strategy games. Some of these benchmarks are concerned with the management of multiple units, such as HeroAIcademy (Justesen, Mahlmann, and Togelius 2016) and Bot Bowl (Justesen et al. 2019). Santiago Ontañón's μRTS (Ontañón et al. 2018) also incorporates resource management, partial observability and simple terrain analysis. To the knowledge of the authors, however, there is no other framework that, providing FM access, captures all of the complexities of real-time or turn-based strategy games.

The aim of the present paper is to introduce *Tribes*, a new turn-based strategy game framework that tries to close this gap. This game is an open source re-implementation of the popular award-wining game *The Battle of Polytopia* (Midjiwan AB 2016), which can be seen as a simplified version of Sid Meier's *Civilization* (Firaxis 1995 2020). *Tribes* includes most components that make research in strategy games interesting, such as partial observability, stochasticity, multi-player, multi-unit management and a vast and variable action space. The different factions in the game must also manage resource gathering, economy, terrain analysis, technology trees and build orders. The balance of resource-cquisition, expansion, combat and exploration makes this benchmark interesting for AI research. Although *Tribes* is not limited to SFP methods, this paper explores some initial results with simple versions of Monte Carlo, MCTS and RHEA agents, which can serve as baseline for future research with this game. We also analyze the complexity of *Tribes* by empirically observing the action branching factor and the tree complexity. 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 intersting levels and game variants.

2 Background

Some of the first applications of AI in turn-based strategy games were seen in 2004, by Arnold et al. (Arnold, Horvat, and Sacks 2004) in the game *Freeciv*[1], 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.

(Arnold, Horvat, and Sacks 2004) 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. (Watson et al. 2008) 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, (Wender and Watson 2008) 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.

One critical aspect of *Tribes* is the challenge of multiple actions executed per turn, and planning turns accordingly to maximise results. (Justesen, Mahlmann, and Togelius 2016) address this problem in *Hero Academy* (Robot Entertainment 2012), 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 (Justesen et al. 2017). Later, (Baier and Cowling 2018) 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 (Montoliu et al. 2020).

This problem of very large action spaces in turn-based strategy games was recently highlighted in the Bot Bowl framework (Justesen et al. 2019), which presents an implementation of the board game *Blood Bowl* (Jervis Johnson 1986). 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, GrodBot, 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. (Ontanón et al. 2013) review challenges presented in these games, of which *Starcraft II* has recently seen great success with the development of AlphaStar (Vinyals et al. 2019), 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 (Ontañón et al. 2018). 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.

3 Tribes

Tribes is an open-source re-implementation of the game *The Battle of Polytopia*, a turn-based award-winning strategy game with more than 11 million downloads worldwide. This section summarizes the rules of the game, the implementation of the framework and the AI agents included.

3.1 Rules of the Game

Tribes is a game played for T *game turns*. In each of these, each tribe plays one *turn* t. A tribe can play as many actions as desired on its turn, until it decides to end it. Ending a turn is the only action available if all units have finished their move and the tribe has no more resources available to spend.

The game takes place in a 2D grid of $N \times N$ tiles. Each tile has a terrain type (plain, mountain, shallow or deep water), can hold up to one unit at a time and can have different types of resources (forest, cattle, food, crops, ore, ruins, fish and whales). Tiles may be neutral or owned by a tribe if within one of the faction cities' borders. More cities can be added to the tribe by capturing neutral villages or enemy cities.

Each player controls one of the available tribes (or factions). *Tribes* includes 4 of the original's game tribes: *Xin_Xi*, *Imperius*, *Bardur* and *Oumaji*. Each player starts with one city (capital), a unit (*Warrior* for all but Oumaji, which starts with a *Rider*), a technology already researched (*Climbing*, *Organization*, *Hunting* and *Riding*, respectively).

A game can be played in one of two different modes, *Capitals* and *Score*. In *Capitals*, a tribe wins the game when it captures all of the enemy capitals. In the *Score* mode, the game ends after 30 game turns and the tribe with the highest score wins. Score points can be obtained by various in-game activities such as harvesting resources, exploring the map or capturing villages. In both game modes, a player is defeated if its capital is captured.

Tribes's gameplay can be split into three main components: technology research, economy management, and

[1]The Freeciv Project, 1996-2020, http://www.freeciv.org/

Figure 1: *Tribes* (left) and The Battle of Polytopia (right).

combat. Combining all of them requires effective decision making and enables different playing strategies. Managing the economy revolves around the accumulation of game currency (*stars*). These are produced by all the cities owned by the tribe and are awarded at the beginning of each turn. The collection of resources and construction of buildings adds population to cities, which then level up every time the population reaches a target (city level+1). Leveling up augments the star production of the city and provides bonuses to choose from (extra production, resources, city border growth, etc). If a city reaches level 5 or higher, a specially strong unit (a *superunit*, or *Giant*) can be spawned. Each tribe unlocks new features by spending stars to research up to 24 technologies in the tech tree, including the construction of new buildings, ability to gather new resources within the city borders or traverse water and mountain tiles.

Stars can also be used to spawn units in cities, each with different attack power, defence, attack range and health points (HP). Units can be melee (*Warrior*, *Rider*, *Defender*, *Swordsman*, *Knight*) or ranged (*Archer*, *Catapult*). All can embark onto a *Boat* (which can be upgraded to a *Ship*, then a *Battleship*) and become *veteran* (higher HP) by defeating other units. Some units can move and attack (in any order), some can move or attack multiple times, and others can either move, or attack, in a turn. Finally, the *Mind Bender* can heal friendly units and convert enemy units to their faction. All units, except the *Mind Bender*, can capture enemy cities or neutral villages if they start the turn in the respective tile.

As can be seen, *Tribes* is a fairly complex game that can hardly be fully explained here. For a more detailed description of the game, the reader referred to the fandom page[2]. Figure 1 shows screenshots of *Tribes* and the original game[3].

3.2 Implementation

Tribes is implemented in Java and provides an API for AI agents. The framework can be run either with partial (PO) or with full observability. For the PO mode, fog of war is implemented: tiles are not visible until a friendly unit reveals it. Vision range is set to 1 tile in plains, and 2 in mountains. Tiles not visible to an agent are indicated by a *Fog* terrain type, and they are treated as plain empty tiles by the Forward Model (FM) if revealed in agent simulations. Visible enemy cities and units always hide information in both observability modes: number of kills and original city (for units), and population, production and units created (for cities).

In order to take actions in the game, each agent implements the `act` method, which receives an observation of the current state (copy of the real game state). This copy is also equipped with a FM, allowing the agent to simulate possible future states by providing an action (as if executed by any player). Agents receive one call to the `act` method for each action, until they end their turn returning an *EndTurn* action. Actions can be categorized in three types:

- **Unit actions** (executed by units): Attack unit, move, capture a village/city, convert units to their tribe, disband, examine ruins (provides boosts at random), become veteran, and upgrade (if naval unit). Distinct units can apply one or more of these actions in different order.
- **City actions** (executed within a city's borders): construct building, destroy building, burn forest (to create *crops*), clear forest (to gain stars), grow forest (allows building *lumber huts*), gather resource, spawn unit, and level up.
- **Tribe actions** (global faction actions): research technology, build road (to boost movement) in a non-enemy tile, and end turn.

Actions are returned one by one to the game as `Action` objects, which may contain several parameters (e.g. destination for *Move*, building type and location for *Build*). The game state can be queried for the list of all available actions, which is computed after each action execution in the game or in the FM. Note that executing actions can make some other actions possible (e.g. researching a technology that allows spawning new types of unit) or impossible (e.g. occupying a tile that blocks other units' movement).

Tribes also includes a procedural generator for levels. This generator is a rule-based system that can be configured by defining some initialisation parameters (size of the level, ratio between land and water tiles, which tribes will play the game, etc.). Each tribe starts in its own unique biome, which is more likely to have specific types of terrain or resources, fitting their starting technology. The generator first distributes lands and water tiles around the level, to then place the starting capitals in land tiles located as far as possible from each other. Finally, the level is populated with forest, mountain, village, resource and ruins tiles[4].

3.3 Agents

The framework includes 7 agents. Two very simple ones for testing purposes: ***DoNothing*** (ends its turn at every *act* call), and ***Random*** (returns an available action uniformly at random). The other 5 agents are described next:

***Rule Based (RB)*:** The first agent is a simple rule-based AI that chooses actions based on the current state of the game, without using the FM to reach future game states. It is a

[2]polytopia.fandom.com/wiki/The_Battle_of_Polytopia_Wikia

[3]*Tribes* code and documentation can be found at https://github.com/GAIGResearch/Tribes

[4]This level generator is a Java port of an independent one: https://github.com/QuasiStellar/Polytopia-Map-Generator

hand-crafted heuristic agent modeled with domain knowledge of the game. On each action selection, RB scores each one of the available actions between 0 and 5, to then execute the action with the highest value. In case of a tie, one of the highest-valued actions is selected uniformly at random. Actions are further evaluated by type, prioritizing the most valuable ones to be executed first.

Attack actions are evaluated based on the attacker's attack power, the defender's defence and the health of both units. The action is given maximum priority if the target is weaker in defence and HP than the attacker. Attacks receive a retaliation counter-attack if the defender unit is not defeated, hence lower priority is given to attack actions targeting stronger units with high HP. *Move* actions are penalised if they move the unit in range of stronger enemy units. Higher scores are given to actions that bring units closer to weaker enemy units, villages and enemy cities.

Capturing cities and villages, *examining* ruins and *making a unit a veteran* are always given the highest possible score. For *leveling up*, the following bonuses are preferred at each level: extra production; extra resources; city border growth; and spawning a superunit (levels 5 and up). *Upgrading* and *spawning* a unit are scored based on the type of unit, resources available and the presence of enemy units within city borders. Similarly, *building* and *resource gathering* consider resource and population gain. The Mind Bender's *convert* action is scored higher for stronger enemy units and its *heal* action values 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. *Disbanding* a unit and *destroying* a building are never executed[5].

While this agent incorporates some useful domain knowledge, its main weakness comes from the fact that actions are evaluated independently. Thus, the agent lacks overall planning, unit coordination and an efficient resource management strategy. However, it provides a baseline for comparison with other agents in the game.

***One-Step Look Ahead (OSLA)*:** One-Step Look Ahead is a simple method that uses the FM to advance the current state of the game once for each action available in a given turn. Each action is then given a score based on the evaluation of the state it leads to. This state is scored using the state evaluation function described at the end of this section, breaking ties uniformly at random. The action with the highest value is then executed in the game.

***Monte Carlo (MC)*:** Monte Carlo search repeatedly executes *rollouts* (sequences of random actions) from the current game state (S_0) to a predetermined depth while within budget. The last state reached by the rollout is evaluated with the same state evaluation function. MC returns the action that, starting from S_0, achieved the highest average value over all iterations.

***Monte Carlo Tree Search (MCTS)*:** MCTS is a well known tree search method that grows an asymmetric tree by balancing exploration of new actions and exploitation of the most promising moves (Browne et al. 2014). In each iteration, four main steps are executed in the default MCTS algorithm: *Selection*, *Expansion*, *Simulation*, and *Back-propagation*. The selection step uses a tree policy (such as Upper Confidence Bound for trees; UCB (Kocsis and Szepesvári 2006)) to navigate the tree from the root until reaching a non-fully expanded node. At this point, the expansion step adds a new node at random and starts the simulation step, performing a rollout until reaching the terminal state, which is evaluated with the same state evaluation function described below. The back-propagation step updates all nodes traversed in this iteration with the score of the evaluated state. Once a given budget is expired, MCTS returns the action corresponding to the most visited child of the root.

***Rolling Horizon Evolutionary Algorithms (RHEA)*:** RHEA (Perez et al. 2013)) evolves sequences of actions (individuals) that are evaluated by executing them from the current state until the end of the individual. The state found at the end of this sequence is evaluated with the same state evaluation function and constitutes the fitness of the individual. Once the given budget is exhausted, the sequence with the highest fitness is selected and its first action returned to the game. Evolutionary operators are applied for selection of individuals, crossover, mutation and elitism. It is common practice in RHEA to use a shift buffer, which moves all the genes one position towards the start of the genome after each action selection, adding a random action at the end to keep the pre-determined length.

State Evaluation Function The value of a state V_S is determined by a heuristic function that analyzes the state of the different tribes. This function computes 7 features (Φ), evaluating changes from a state (S_0) to another (S_L). These features are the differences in production (ϕ_1), number of technologies researched (ϕ_2), game score (ϕ_3), cities owned (ϕ_4), number of units (ϕ_5), number of enemy units defeated (ϕ_6) and the sum of the levels of all owned cities (ϕ_7). A "difference" value $v^i_{0\to L}$ is computed, for each player i in the game, as a linear combination between Φ and W, where W is a weight vector $W = \{5, 4, 0.1, 4, 2, 3, 2\}$. These values have been adjusted to represent the magnitudes and relative importance of each ϕ_i. Finally, the $v^i_{0\to L}$ values for all tribes i different to the current player are averaged and subtracted from $v^j_{0\to L}$ where j is the root player, resulting in the final score V_S for state S.

Intuitively, each $v^i_{0\to L}$ represents the *progress* of tribe i from state S_0 to state S_L and V_S indicates the relative progress between the own tribe and the others in the game. This function is used by all agents that use the FM (OSLA, MC, MCTS and RHEA). For them, S_0 is always the current state of the game and S_L is the state reached after each iteration (S_1 for OSLA, a rollout for MC, maximum depth in MCTS and the end of the individual in RHEA).

4 Experiments and Analysis

First, we study the relative performance of the implemented agents (Section 4.1); then we analyze the complexity of the game for AI decision making (Section 4.2).

[5]These actions are often available, but they are useful only in specific circumstances that require more careful planning.

4.1 Agent Performance

An initial piece of work consisted of adjusting the parameters of the different AI agents. For completeness, we report here some considerations and settings that were attempted (and, some, discarded) before the final experimentation.

Root Prioritization: On each action request per turn, MC and MCTS fix a different subset of actions (Unit, City or Tribe) at random to draw from at the start of each iteration (i.e. first action from S_0) during that action selection call. Once S_1 has been reached, all actions become available again for subsequent steps. This provided a stronger playing performance than allowing any available action at the root. This can be seen as an extremely simplified version of Progressive Widening (Chaslot et al. 2008).

Forcing End Turn actions: given the game's large branching factor, MC, MCTS and RHEA rarely have the chance to execute actions as other players in their simulations. It is even less likely that they get to play as themselves again in the same iteration, which may be detrimental to action planning. Forcing the use of an End Turn action every X steps, however, did not provide better results overall and was discarded as an option in the final experiments. Investigating this further is a matter for future research, but it is possible that this approach would work better using Progressive Widening techniques with a move ordering function, or enhancements like First-play urgency (Gelly and Wang 2006).

MCTS Simulation step: Some recent work with MCTS, such as (Silver et al. 2017) in AlphaZero or (Baier and Cowling 2018) in Multi-action MCTS, has shown that skipping the simulation step in MCTS and evaluating the state reached in the expansion phase can provide better results than using rollouts. This was also true for *Tribes*, hence the MCTS agent does not perform rollouts during its iterations.

Algorithm parameters: The exploration constant for the tree policy in MCTS is set to $\sqrt{2}$. The iteration length in MC, MCTS and RHEA is set to 20 (values of 5 and 10 were tried and consistently provided inferior results). The population size for RHEA is set to 1 (hence, RHEA behaves as a Random Mutation Hill Climber). Some previous work showed that larger RHEA populations tend to provide better results in General Video Game Playing (Gaina et al. 2017), 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. Additionally, the actions Disband and Destroy are removed from the selection of possible actions in the SFP methods (as in RB).

Experimental setup Each agent played against each other agent 500 games: 20 repetitions in the same 25 procedurally generated levels, all with a 11×11 size and a $1:1$ water/land tile ratio. As levels are not guaranteed to be balanced, the starting position of one of the tribes can be better than the other in certain levels. To avoid this impacting the results, agents play 10 repetitions with each tribe. All runs are independent, without carrying any information from one game to the next. The tribes used for these games are *Xin_Xi* and *Imperius* and the game mode is *Capitals*[6] played with full observability. OSLA, MC, MCTS and RHEA use the same budget of 2000 usages of the FM `next` method per decision. Once this limit is reached, the algorithms stop and return a desired action. Having this limit based on function calls instead of execution time permits comparing these results in different hardware architectures.

Results Table 1 summarizes the results of these experiments, with rows sorted from stronger to weaker agent. The left part of the table shows the win rate of the row agent versus the column. It can be observed that RHEA is able to win more games against all other agents. It is specially proficient against the simpler agents (MC and OSLA; $> 75\%$ victories), beating MCTS and Rule Based 63% and 58.6% of the time, respectively. It is remarkable that the Rule Based agent ranks second, despite being a very simple heuristic agent. This is useful to put in perspective the strength of the current algorithms included in the framework and the existent room for improvement. They can play at some degree of skill (they all achieve practically 100% versus the random agent), but are still far from optimal play.

Table 1 (right) also shows aggregated statistics for all games played. The last 4 columns indicate end game measures that normally correlate with victory (and as such rewarded by the heuristics employed): final score, percentage of technologies researched, number of cities owned and final production. This also reveals an interesting aspect: RB is strategically an outlier, achieving less score and technologies researched than some agents lower in the ranking.

To verify the strength of the AI players, we paired our top agent (RHEA) against a moderately good human *Tribes* player. A game was played for each of the same 25 levels (same 2 tribes assigned randomly). The results show a clear dominance on the human side (100% wins). Some interesting observations can be made about the play-style of RHEA: it makes reasonable short-term decisions (e.g. attacking enemy units that try to capture their cities), but lacks tactical and strategic depth required to coordinate multiple units in a turn or spend resources efficiently in various situations.

4.2 Game Complexity

One of the main motivations to introduce *Tribes* is its complexity. Apart from the difficulty for AI agents to create long-term plans in a multi-agent, multi-player and partially observable environment, the game provides very large and dynamic action space. As the game progresses, tribes capture more cities, increase their production and research more technologies. All these events allow the tribes to spawn more units and construct new buildings. Therefore, the action space for the players is not only more varied as the game progresses (more *types* of actions) but its size also grows exponentially. Actions are dependant on researched technologies, terrain types, stars and the interaction between the different friendly and enemy units, which makes the analytical computation of the action space not trivial. This section provides an *empirical* analysis of *Tribe*'s action space.

[6]For completion purposes, Capitals games end after 50 *game turns*, declaring the faction with most points the winner.

Agent wining rate (row player vs column player)							**Aggregated results at the end game**					
Agent	**RHEA**	**RB**	**MCTS**	**MC**	**OSLA**	**RND**	**Wins**	**Rank**	**Score**	**Techs**	**Cities**	**Prod.**
RHEA	*	58.60% (2.20)	63.00% (2.16)	77.80% (1.86)	74.80% (1.94)	100.00% (0.00)	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	41.40% (2.20)	*	56.20% (2.22)	62.40% (2.17)	70.20% (2.05)	98.80% (0.49)	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	37.00% (2.16)	43.80% (2.22)	*	62.00% (2.17)	60.80% (2.18)	98.60% (0.53)	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	22.20% (1.86)	37.60% (2.17)	38.00% (2.17)	*	54.80% (2.23)	99.00% (0.44)	50.32% (1.01)	1.50 (0.03)	8065.96 (71.22)	82.35% (1.65)	2.23 (0.04)	14.73 (0.29)
OSLA	25.20% (1.94)	29.80% (2.05)	39.20% (2.18)	45.20% (2.23)	*	99.40% (0.35)	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	0.00% (0.00)	1.20% (0.49)	1.40% (0.53)	1.00% (0.44)	0.60% (0.35)	*	0.84% (0.02)	1.99 (0.04)	3708.76 (12.51)	58.54% (1.17)	0.39 (0.01)	0.01 (0.00)

Table 1: On the left, win rate for each row agent averaged across 500 games. On the right, statistics for all games averaged across 2000 game ends. Values between brackets indicate standard error of the measure.

We ran 400 games for 50 *game turns* between RHEA and RB in the previously set 25 levels. The following was recorded: average number of moves played per turn, average number of available actions per move and branching factor per tribe's turn. In order to appreciate the difference between the start and the end of the game, values are also analyzed during the turn blocks $h_0 = [1, 25]$ and $h_F = [26, 50]$. The number of moves per turn grows linearly, with an average of 9.86 moves for the whole game. This average is 6.47 during h_0 and doubles to 13.11 for h_F.

Figure 2 shows the progression of the action branching factor per move (left) and per turn (right). Every time an agent makes a move, there is an average of 54.47 possible actions to choose from (21.86 for h_0 and 85.83 in h_F). The turn branching factor can be calculated as the product of these values per step, averaging at 10^{15} for the whole game (10^7 for h_0 and 10^{23} for h_F). It is useful to put this in perspective comparing with the branching factor of other games: Chess (30), Go (300), Hero AIcademy (10^8) and Bot Bowl (10^{51}). It is worth noticing, however, the impact that strategy has in these numbers. Figure 2 averages these measures by which bot won each game. It is clear that there is a difference between winning and losing players. Both branching factor and number of actions available are higher in winning players, correlating victory and action space. Winning bots reach a branching factor of 10^{32} in the last 5 turns of the game. Thus one cannot dismiss the idea that more skilled players could experience even larger action spaces, and longer games with more players and larger board sizes are likely to increase this size as well.

The size of the game tree, in a two-player game of 50 turns with an average branching factor would be $((10^{15})^2)^{50} = 10^{1500}$, compared to Chess ($10^{123}$), Go ($10^{360}$), Hero AIcademy ($10^{711}$) and Bot Bowl ($10^{3264}$).

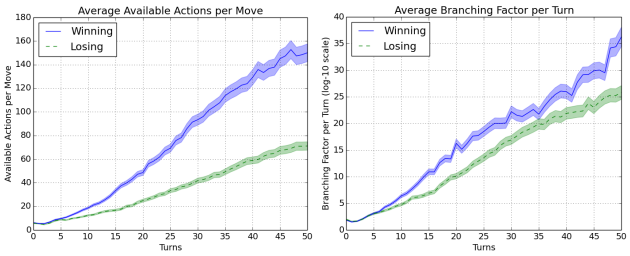

Figure 2: Actions per move and branching factor (log scale).

5 Conclusions and Future Work

This paper introduces a new framework for strategy games, *Tribes*, based on the popular award winning game *The Battle of Polytopia*. This is a multi-agent, multi-player, stochastic and partially observable game that offers several interesting challenges for AI, requiring unit coordination, long-term planning, opponent modeling and build ordering in a vast and variable action space. The game includes a Forward Model to facilitate the creation of SFP methods (like MC, MCTS and RHEA). This paper describes these baseline agents and their relative performance. Although some of these agents are able to play at a decent level (matching a simple heuristic implemented for a rule-based agent), their play strength is clearly inferior to human players.

Already ongoing work is expanding this study to analyze results in the *Score* game mode, games with more than two players and partial observability, as well as implementing other agents that have shown good performance in multi-agent games in recent research, such as Online Evolutionary Planning (Justesen, Mahlmann, and Togelius 2016) and Evolutionary MCTS (Baier and Cowling 2018).

Additionally, other lines of possible research for this work include the implementation of model-free reinforcement learning (RL) or deep RL agents for the game. In particular, the GUI of the current framework provides an ideal input for methods that use screen capture for decision making. The game is also highly parameterizable (about 150 values determine features of units, buildings and economy), facilitating research in automatic game design. Last but not least, this framework also provides the chance of investigating new procedural generations that can create interesting and/or challenging levels for the game.

Acknowledgements

This work is supported by UK EPSRC research grants EP/T008962/1, IGGI CDT EP/L015846/1 and Queen Mary's Apocrita HPC facility, sustained by QMUL Research-IT.

References

Arnold, F.; Horvat, B.; and Sacks, A. 2004. Freeciv learner: a machine learning project utilizing genetic algorithms. *Georgia Institute of Technology, Atlanta*.

Baier, H., and Cowling, P. I. 2018. Evolutionary mcts for multi-action adversarial games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8.

Blizzard Entertainment. 2010. *Starcraft II*.

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2014. A Survey of Monte Carlo Tree Search Methods. In *IEEE Trans. on Computational Intelligence and AI in Games*, volume 4, 1–43.

Chaslot, G. M. J.; Winands, M. H.; HERIK, H. J. V. D.; Uiterwijk, J. W.; and Bouzy, B. 2008. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation* 4(03):343–357.

Firaxis. 1995 – 2020. *Civilization*.

Gaina, R. D.; Liu, J.; Lucas, S. M.; and Pérez-Liébana, D. 2017. Analysis of vanilla rolling horizon evolution parameters in general video game playing. In *European Conference on the Applications of Evolutionary Computation*, 418–434. Springer.

Gelly, S., and Wang, Y. 2006. 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.

Jervis Johnson. 1986. *Blood Bowl*.

Justesen, N.; Mahlmann, T.; Risi, S.; and Togelius, J. 2017. Playing multiaction adversarial games: Online evolutionary planning versus tree search. *IEEE Transactions on Games* 10(3):281–291.

Justesen, N.; Uth, L. M.; Jakobsen, C.; Moore, P. D.; Togelius, J.; and Risi, S. 2019. Blood bowl: A new board game challenge and competition for ai. In *2019 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Justesen, N.; Mahlmann, T.; and Togelius, J. 2016. Online evolution for multi-action adversarial games. In *European Conference on the Applications of Evolutionary Computation*, 590–603. Springer.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Midjiwan AB. 2016. *The Battle of Polytopia*.

Montoliu, R.; Gaina, R. D.; Pérez-Liebana, D.; Delgado, D.; and Lucas, S. 2020. Efficient heuristic policy optimisation for a challenging strategic card game. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, 403–418. Springer.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games* 5(4):293–311.

Ontañón, S.; Barriga, N. A.; Silva, C. R.; Moraes, R. O.; and Lelis, L. H. 2018. The first microrts artificial intelligence competition. *AI Magazine* 39(1):75–83.

Perez, D.; Samothrakis, S.; Lucas, S.; and Rohlfshagen, P. 2013. Rolling horizon evolution versus tree search for navigation in single-player real-time games. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 351–358.

Robot Entertainment. 2012. *Hero Academy*.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Vinyals, O.; Babuschkin, I.; Chung, J.; Mathieu, M.; Jaderberg, M.; Czarnecki, W. M.; Dudzik, A.; Huang, A.; Georgiev, P.; Powell, R.; et al. 2019. Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog* 2.

Watson, I.; Azhar, D.; Chuyang, Y.; Pan, W.; and Chen, G. 2008. Optimization in strategy games: Using genetic algorithms to optimize city development in freeciv. *University of Auckland*.

Wender, S., and Watson, I. 2008. Using reinforcement learning for city site selection in the turn-based strategy game civilization iv. In *2008 IEEE Symposium On Computational Intelligence and Games*, 372–377. IEEE.