MCTS pruning in Turn-Based Strategy Games

Yu-Jhen Hsu
190279963
Dr. Diego Perez Liebana
MSc Artificial Intelligence

***Abstract*—In a turn-based strategy game, one of the most problematic issues is a significant action space, which damages the performance of AI agents such as Monte Carlo Tree Search (MCTS). Although a number of advanced algorithms, such as Online Evolutionary Planning (OEP) and Evolutionary MCTS (EvoMCTS), have shown progress in dealing this issues, the usage of these algorithms is limited as the representation for the nodes/genomes results in a fixed number of actions for a turn, which is not feasible for most of the strategy games. The action space for a turn might vary during the game and executing the actions might influence the size of the action space in general strategy games. This paper uses a single action instead of a whole turn as the representation for MCTS. In order to manage the high branching factors in a new turn-based strategy game, Tribe, pruning techniques with domain knowledge are used. With the help of these pruning techniques, MCTS is enhanced and outperformed a baseline approached (rule-based agent and Rolling Horizon Evolutionary Algorithms) in the experiment. Moreover, this paper also provides the tree shape and behaviour of MCTS with/without pruning techniques.**

***Index Terms*—Monte Carlo Tree Search, Pruning techniques, Strategy games**

I. Introduction

Monte Carlo Tree search (MCTS) is an advanced search algorithm that demonstrates the profound success in the field of classical board games as well as in general game playing, planning and optimization tasks. These cases rely on evaluation functions which calculate the given state and guide the search. Alpha Go (Silver et al. 2016), for example, achieves elite human-level performance and defeated the world champion by evaluating game states based on the enormous amounts data from expert and self-game playing. The branching factor of classical board games, such as Go and Chess, is considered as moderate (coming up to a few hundred of them) (Baier & Cowling 2018). The increase of branching factors generally adversely affects the performance of MCTS (Justesen et al. 2017, Baier & Cowling 2018). A huge branching factor further increases the difficulty to develop the evaluation function, complicating this process even more (Justesen et al. 2017, Browne et al. 2012). Although MCTS can run large branching factors, it fails when these reach sizes numbering into the billions (Baier & Cowling 2018, Chaslot et al. 2008). One of the reasons is that the tree method requires the tree to visit all of the nodes before exploitation. This requirement makes the rarely-visited and non-visited nodes redundant when an optimal action in game play is obtained (Gelly & Wang 2006).

However, most turn-based multi-action strategy games have huge branching factors. This kind of game requires the player to execute multiple actions within turn rather than just one, significantly increasing the branching factor up to billions of node generations. The method employed to determine the sequence of actions per turn is also important as previous actions can influence later ones. Compared with Go and Chess, branching factors are much higher for turn-based strategy games (Perez et al. 2020); there are numerous games that fall within this range, including XCOM, Civilization and Battle of Polytopia. In this context, there are a few AI frameworks dedicated to such games, including Bot Bowl, HeroAIcamdy and Tribes. Bot Bowl (Justesen et al. 2019) for example, in which the goal is to manage a football team where each unit can execute several actions, has a branching factor that can be as high as 10^{51} in a given turn. The branching factors for Tribes (Perez et al. 2020) and HeroAIcamdy (Justesen et al. 2016) are 10^{31} and 10^{8}, lower than Bot Bowl but still much higher than Go and Chess. Tribes is different from Bot Bowl and HeroAIcamdy as it introduces extra management complexities. Instead of managing only a unit, the player needs to formulate balanced decision in units, cities, tech trees and resources.

There are few search algorithms developed to solve substantial branching factors in strategy games and to obtain excellent performance. Examples are Online Evolutionary Planning (OEP) (Justesen et al. 2017) and Evolutionary MCTS (EvoMCTS) (Baier & Cowling 2018). These two algorithms were developed within the game HeroAIcamdy, in which the player is required to select actions for five units per turn. These two algorithms use a sequence of actions in a turn as the representation rather than representing genomes or nodes in a single action. The method used to obtain an action turn over the search represent the main difference between these two algorithms. OEP uses an evolutionary algorithm and selects the recommended action set by crossover and mutation operators. Since these operators might result in invalid actions sets, the algorithm will check for feasibility during the porcess to ensure their validity. EvoMCTS, on the other hand, combines MCTS and an evolutionary algorithm. It guides the tree by modifying one of the actions in the action set (representation). These two algorithms select a fixed number of actions per turn, and each action is assigned to a specific unit. This approach complicates adaptation for a game like Tribes (or most strategy games) for three reasons. First, the number of actions is changeable within a turn. This is due to the fact that some actions might be invalid or valid after executing the action. Secondly, the size of the action set generally increases

during the game. Lastly, instead of managing units, the agents are required to manage not only unit actions but also cities, researches and tribe actions in the game Tribes. This suggests that it is impossible to select a fixed action size based on the unit size. These factors increase the difficulty to estimate the possible action size required to create fixed size representation for neither OEP nor EvoMCTS.

One possible solution is to use a single action rather than a complete turn as a representation for MCTS. It not only decreases the branching factor but also makes the tree deeper (Baier & Cowling 2018). However, is has been shown that vanilla MCTS without improvement fails and the tree is still shallow (Justesen et al. 2017, Baier & Cowling 2018). Numerous enhancements for MCTS have been proposed to deal with this issue. One possible solution focuses on pruning, that is, reducing the branching factors by removing some weak actions based on evaluation functions or domain knowledge that increases the converge speed to find good actions. Although it is shown that pruning techniques are helpful, research on tree behaviour applying pruning techniques is limited.

This paper aims to investigate pruning techniques and to analyse the tree structure and behaviour of MCTS for Tribes. Specifically, the performance of MCTS is compared by applying pruning techniques with the original MCTS agent, Rolling Horizon Evolutionary Algorithms (RHEA) agent and rule-based agent, which have outperformed MCTS in a previous Tribes investigation (Perez et al. 2020). Moreover, the different MCTS pruning techniques are analysed by their tree shape in terms of the influence of turn and action size.

II. Background and related work

This section reviews MCTS associated pruning techniques for handling huge branching factors and Rolling Horizon Evolutionary Algorithms (RHEA), being the latter one of the baseline approaches.

A. Monte Carlo Tree Search (MCTS)

MCTS (Browne et al. 2012) is a search algorithm that has been applied in multiple games and shows a good performance. It searches the most promising region, resulting in an asymmetric tree, by balancing exploitation and exploration. Four steps are included in an iteration for default MCTS: selection, expansion, simulation and back-propagation. The algorithm first uses Upper Confidence Bounds (UCB1) to guide the search from root until it reaches the node with non-visited nodes in the selection step. Next step, one of non-visited nodes is picked at random and followed by simulation step, which performs a series of rollouts (Monte Carlo simulations) until reaching a terminal state. While this is one of the main steps, some recent studies (Perez et al. 2020, Baier & Cowling 2018) have shown that skipping the rollout can provide better results. The finial step, back-propagation, the terminal state is evaluated and the updates the value of the nodes that are visited in this iteration. The most visited child of the root is returned when the search process runs out of budget.

Numerous of enhancements are used to improve the MCTS performance under large branching factors. These methods either reduce the branching factor or handle the requirement of visiting all nodes before exploitation. First-Play urgency (Gelly & Wang 2006) encourages early exploitation by assigning the initial value to non-visited nodes. It reduces the need for visiting every node before exploitation under UCT and allows the tree to go deeper. Move groups (Childs et al. 2008) is another technique for reducing the branching factor and increasing the performance of MCTS in Go. Instead of choosing a single action, a move group, containing similar moves with the highly-correlated expected value, is picked and evaluated. Rapid Action Value Estimation (Gelly & Silver 2007) updates the value for all of the nodes with the same action and state regardless of its position in the tree. Script approaches, such as Hierarchical Portfolio Search (Churchill & Buro 2015) and Portfolio Greedy Search (Churchill & Buro 2013), specialise in dealing with huge branching factors in real-time strategy games, by searching over a small amount of hand-coded scripts rather than the whole action set.

B. Pruning techniques for MCTS

Pruning is another technique used to manage huge branching factors. There are two main benefits for using pruning. The first one is that poor actions can be ignored so attention is focus on more promising actions; this results in a much deeper tree and, potentially, a quicker convergence. Another benefit is that it can be used without heuristic functions and can be applied in any domain (Browne et al. 2012). The trap states, being states that have a high reward but lead to a loss in the game play, can also be removed by pruning with domain knowledge (Ramanujan et al. 2010).

Pruning techniques can be divided into two categories: hard and soft pruning. The hard version prunes the tree by eliminating actions with the lowest evaluated score. The latter prunes actions after certain iterations, but the eliminated actions will be chosen the later iterations. The benefit of soft pruning consists in alleviating the risk of removing the good actions (Browne et al. 2012); hard pruning, one the other hand, allows the tree to go deeper (Justesen et al. 2017). Progressive widening (PW) (Coulom 2007) or Progressive Unpruning (Chaslot et al. 2008) is an example of soft pruning that achieves the most success in the game Go. N nodes are pruned based on evaluation functions once the iteration number exceeds a threshold. After several simulations, nodes are unpruned and can be chosen in the search; in this way, all actions will be visited given adequate budget. This technique provides a similar effect to First-Play urgency by encouraging early exploitation and allowing the tree to go deeper. While pruning techniques can be used without heuristics, the performance of progressive widening depends on the heuristic function. Previous research (Teytaud & Teytaud 2009) shows that progressive widening without the help of heuristics has a limited impact on improving the strength of the agents in the game Havannah.

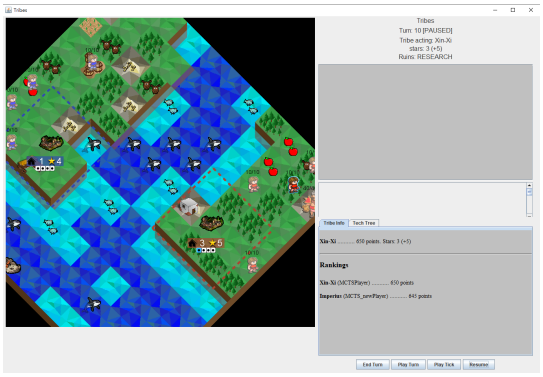

Fig. 1. The user interference of Tribes.

C. Rolling Horizon Evolutionary Algorithms (RHEA)

RHEA is an evolutionary algorithm that achieves good results in several real-time games (Perez et al. 2013, Gaina et al. 2017). Unlike other evolutionary algorithms that tend to learn offline, RHEA learns online during the game. It first generates the individuals (a sequence of actions). Then, it evaluates the final state resulting from executing the actions in the individual from the current state until the end of the sequence (or the game, whatever occurs first). Evolutionary operators such as selection, crossover and mutation are used to generate new individuals based on old ones. The first action in the individual with the highest value is returned once the budget is exhausted.

III. Methods

This section provides an introduction to the game Tribes and the agents used in the experimental work, including two MCTS variants with different pruning approaches.

A. Tribes

Tribes (Perez et al. 2020) is an implementation of the game The Battle of Polytopia (Midjiwan AB 2016), a popular award-winning turn-based strategy game, which can be viewed as a simplified version of Sid Meier's Civilization. Fig. 1 shows the interface of Tribes.

The game takes place in the form of a $N \times N$ tiles setup. Each tile has a terrain type and might also contain a resource, building and a unit at the same time. There are four types of terrain (plain, mountain, shallow water, deep water), determining which type of resource and building can be existed or constructed. At the beginning of the game, the resources, terrain, villages and ruins will be randomly generated in tiles. The ruins are randomised bonuses (special units, technology, resources, etc.) which can be collected by units. One of four available Tribes (Xin Xi, Imperius, Bardur and Oumaji) is assigned to the player with a starting unit, a researched technique, that differs between tribes, and five stars (resource currency). Moreover, none of the players will control the same tribes in the game. There are two game modes: Capital and Score. Taking all the enemies' capitals can result in victory in both of modes, while the player with the highest score in turn 30 will also win the game in Score mode. The score is accrued from a variety of actions such as exploring, resource gathering, capturing villages and cites, and constructing determined buildings. It is allowed to execute as many actions as possible within a turn until they decide to end it, or no more actions are available.

Economy management, technology and combat are three crucial aspects of the game that determine the playing strategy. The strategy for consuming the stars in building, gathering resources, spawning units and researching technologies is the key element in economy management. Stars are produced by cities on each turn and consuming the stars on building and resource can provide population to the city. The population then can ungraded the cities, which unlocks a bonus (extra production, units, city border growth, etc.) and increases the stars it produced, once it exceeds the level of city. When the city level is over 5, the player can only select from a bonus between a strong unit (super unit or Giant) or a park (extra score). Buildings can only be built in their player's own territory, which is formed by the 3×3 tiles around the city. The types of units and buildings that can be spawned and constructed are limited at the beginning of the game, but more types can be unlocked by researching within the different 24 technologies. There are two types of units can be spawned, namely melee (Warrior, Rider, Defender, Swordsman, Knight) and ranged (Archer, Catapult). Each unit has different attack powers, range and health points (HP). They can also embark in port buildings and become a boat unit which has three levels (boat, ship and battleship). Upgrading the boat unit can increase the attack power, movement range and HP. Moreover, different type of units can perform a distinct combination of actions per turn. Most of them can attack and/or move in a turn, but others can do multiple consecutive actions. Once a unit defeats three other units, the former is upgraded to a veteran, having much higher HP. Units can capture the enemy's city or neutral villages after staying in that city/village for more than a turn. Mind Bender is a special unit which can heal friendly units and covert enemies to their faction. More details about the game can be found in the original paper (Perez et al. 2020).

There are three kinds of actions in Tribes: *unit* (the only set without requiring stars to perform an action), *city* and *tribe*. The *unit* actions include attack, move, capture, convert, disband, examine, upgrade and to become a veteran. Some of these actions are specialised for the particular unit type or can only be performed when its corresponding technique is researched. The *city* actions include constructing and destroying buildings, burning, growing and clearing forests, resource gathering, spawning units and levelling up. The *tribes* action includes research, building a road on non-enemy tiles and ending the player's turn. Roads speed up the units' movement and can be used by any player. Most of the actions will influence later actions. Some actions decrease the action space for a turn. For example, building consumes the star and results in a lack of stars for executing subsequent actions. Other actions, such as researching which unlocks features, increase

the action space for a turn. This makes Tribes a complex environment for decision making, with huge branching factors when a player possesses many units and cities. A good performance requires a balance of the actions related to technology, economy management and combat, as well as a good strategy for in terms of the number and sequence of actions per turn.

B. Baseline Approaches

There are three algorithms from the literature (Perez et al. 2020) used as the baseline approaches, being the original MCTS, RHEA and rule-based actions. The reason for comparing RHEA and rule-based agents is that they outperform the MCTS in the original paper.

Rule Based (RB): The simple rule-based agent is the only agent that does not rely on the forward model in this paper. The forward model allows the agent to obtain the next state based on a particular action. It has the second-highest winning rate and has a 56.2% winning rate when competing against the original MCTS (Perez et al. 2020). The agent only uses the information of the current state, evaluates each action separately and returns the one with the highest value. If there is more than one action with the highest value, it picks an action randomly to break the tie. The evaluation is hand-crafted and relies on domain knowledge. The main weakness of this agent is that it considers every action independently and does not consider the action's effect in the game state. The strategies of this agent are that it disables the destroy and disband action and focuses on actions related to capturing, examining and to become veteran. More details of rule-based agents can be found in the original paper (Perez et al. 2020).

Rolling Horizon Evolutionary Algorithms (RHEA): RHEA, described in Section II-C, achieves the best performance with a 63% winning rate against the original MCTS(Perez et al. 2020). In the experiment, the RHEA agent uses the same parameters: population size of 1 and individual length of 20. Similar to the RB agent, the disbanding action and destroying action is removed and never be considered.

Monte Carlo Tree Search (MCTS): MCTS, described in Section II-A, only loses when competing with the RHEA and RB agent(Perez et al. 2020). This MCTS variant prioritises the root action by picking up different subsets of actions from city, unit and tribe actions. Only roots are applied for prioritization, and the children can execute all actions. The maximum depth of the tree is 20 and the UCB1 (shown in figure 1) is chosen as the tree policy. Instead of evaluating the state being rolled out based on default policy until the tree reaches a certain depth or terminal state, the state of the current node is evaluated, and the value is passed to back-propagation.

$$\text{UCB1} = \bar{X}_i + C\sqrt{\frac{\ln{(n+1)}}{n_i}} \tag{1}$$

where $\bar{X}_i$ is the normalised value for children i. C is the parameter deciding the importance of exploration, and it is set to $\sqrt{2}$. n is the visited times of the current node and n_i is the visited time of children i. In order to break the tie, a small amount epsilon $(1e-6)$ is added to n_i.

Heuristic: The heuristic is used to evaluate a game state. It compares the difference between two states and performs an evaluation depends on seven features (Perez et al. 2020). These include the differences between the production, number of technologies, score, number of cities owned, number of units, number of kills and the total level of cities, which have associated weights of 5, 4, 0.1, 4, 2, 3 and 2 respectively. Two features added in this paper stress the importance of building and occupying the enemy's cities. This first new feature add 5 points for a new building and another one gives 10 points for each unit in the enemy's cities in the evaluation state.

C. MCTS with hard pruning

MCTS with hard pruning modifies the tree policy of baseline MCTS to reduce the branching factor. The tree policy not only guides the search, it also prunes the node's children if they are visited from the root more than the budget (20). The number of remaining nodes, being the number of children retained after pruning, is based on the action size and the limit for minimum nodes as shown in equation 2. It will prune nodes' children until the number of children matches the number of the remaining nodes and, calculate its state based on the heuristic described above. The children with the lowest value are pruned and never visited in the later iteration.

$$remainingNodes = \max{(\alpha \log n, T)} \tag{2}$$

where $\alpha = 5$, controls the speed of pruning when the action size grows. This parameter plays a vital role in deciding the number of the remaining nodes when the action size becomes significant: the higher it is, the more children are retained. T, which has a value of 5, is the minimum number of actions that need to be retained after pruning. If there is any action set smaller than T, pruning is not applied. The reason for using logarithm to prune the tree is to reduce the number of actions when this set becomes extremely large. The parameters are determined by running the experiment under different settings and the one with the highest winning rate is used.

Besides pruning by the action size, the destroy action is excluded as it does not provide any rewards and results in a reduced score. Furthermore, MCTS with hard pruning possesses another parameter to decide whether to prune the move action or not. Moreover, it offers two versions of MCTS with hard pruning: with and without move pruning. The reason for pruning the move is to increase the action space significantly. Every unit has multiple move actions based on its move distance and on the road. When it comes to the later stages of the game, a player tends to have many units, increasing the action space significantly. The move action is pruned by the equation 2. It firstly extracts the move action from all action sets; then the move action whose destination does not result in a ruin or the city is extracted and picked by the equation 2, where $\alpha = 1$, and $T = 5$. The pruned move action will then be removed and never be considered in the

TABLE I
AVERAGED WINING RATE FOR EACH VERSION OF MCTS VS 3 BASELINE AGENTS OVER 5000 GAMES

	Hard Pruning			Progressive Widening		
	RB	**MCTS**	**RHEA**	**RB**	**MCTS**	**RHEA**
Move Pruning	56.9%	63.3%	57.8%	57.8%	62%	57.7%
No Move Pruning	54.7%	61.2%	54.5%	55.8%	60.1%	57.7%

action set of the current node. Although the pruned actions are removed in the current node, they still might appear in the following nodes, meaning that they will be considered in the later nodes of tree.

D. MCTS with progressive widening

This variant is a modification of MCTS with hard pruning. Instead of permanently removing the pruned nodes, the algorithm unprunes the nodes in the last search. This process allows a search for every node and alleviates the risk of pruning the best action. It is done by increasing the number of remaining nodes if the iteration number is over the value given by Equation 3 (Chaslot et al. 2008). If the iteration number from the equation exceeds the number of nodes visited from the root, the algorithm will pick up the action at random in the invalid action sets and move it back to the explored or unexplored action set.

$$unpruneIteration = \alpha\beta^{n-n\_init} \tag{3}$$

Here, α is the iteration of pruning, set to 20, β controls the unpruning ratio and has a value of 1.7, n is the size valid action set of current nodes plus one and n_init is the initial number of remaining nodes, being 5. The higher the value of β is, the longer the iteration required for unpruning is and the closer to the behaviour of hard-pruning will be.

IV. EXPERIMENTS AND RESULTS

This section provides an outline of the experimental setup used to test MCTS with pruning, followed by the results of our tests. Finally, an analysis of the MCTS tree is provided.

A. Experimental Setup

Both versions (hard pruning and progresses widening) of MCTS with domain knowledge pruning (i.e., with and without move pruning) were pitched against the rule-based method, the original MCTS and RHEA agents, as included in the framework. For all search agents, the stopping condition is set to 2000 usages of the forward model's *next* method, which is the same as in the original paper (Perez et al. 2020).

Each pairing agent plays 5000 games (25 seeds with 40 repetitions five times). The seeds are the same as in the original paper (Perez et al. 2020) and generate the different levels of 11×11 tiles. Because the level might not be balanced, players swap starting positions within 40 repetitions of every seed. Each game is independent, meaning that no information is carried from one game to the next. The agents only make decisions based on the information from the current game. The game mode is Capital with full observability, meaning that the winning condition is achieved by either taking over the enemy's capital city or reaching the highest score within 50 turns.

B. Game Results

Although both RB and RHEA agents outperform the original MCTS with a 56.2% and 63% winning rate, respectively, the MCTS with pruning techniques is able to defeat both agents with a winning rate surpassing 50%. There are four statistics recorded for the agents in the original paper: average final score, researched technologies percentage, number of cities owned and final production. These statistics do not change significantly after applying pruning on MCTS. The average score of the original MCTS is 9966.55, which lie between the ones obtained by the RHEA (11610.56) and RB (8076.32) agents. In terms of researched technologies, both the original MCTS (85.27%) and RHEA (88.95%) agents attain a much higher percentage than the RB one (71.14%). Regarding the number of cities, the original MCTS owned 2.23, a figure smaller than 2.68 corresponding to RHEA. Similarly, the former obtained a final production of 16.96, while the latter agent outperformed it with a score of 20.68. Overall, The RB agent achieved the highest number of cities owned (2.98) but the lowest final production (20.29). This suggests that the RB agent focuses more on occupying cities while the RHEA agents are more interested in achieving a higher score.

The results of both versions of the MCTS competing with three baseline models is shown in Table I, where the first row indicates the version of MCTS (hard pruning and progressive widening), and the first column indicates whether the move pruned is applied or not. The MCTS with hard pruning achieves 54.7% and 54.5% winning rates compared with the RB and RHEA agents, respectively; this represents a 10% higher winning rate for the original MCTS compared with both agents. The performance of MCTS with hard pruning is further increased by both move pruning (domain knowledge pruning) and progressive widening techniques. However, applying move pruning on MCTS with progressive widening only increases its performance by around 2% when competing with RB and the original MCTS agents; similarly, the winning rate when competing against the RHEA agent remains the same. The hard pruning MCTS with move pruning shows the highest winning rate when competing with the original MCTS (63.3%) and RHEA (57.8%) agents. In parallel, the highest wining rate (57.8%) compared with the RB agent is the progressive widening MCTS with move pruning. This result might be due to the progressive widening allowing the pruned nodes to be visited

TABLE II
THE AGGREGATED STATISTICS FOR EACH VERSION OF MCTS OVER 15000 GAMES

	Hard Pruning					Progressive Widening				
	Win Rate	**Score**	**Techs**	**Cities**	**Prod.**	**Win Rate**	**Score**	**Techs**	**Cities**	**Prod.**
Move Pruning	59.3%	8500.06	77.88%	2.26	17.38	59.1%	8564.36	79.08%	2.25	17.34
No Move Pruning	56.8%	8458.52	77.43%	2.24	17.16	57.8%	8446.04	78.17%	2.24	17.07

after unpruning. This, in consequence, significantly increases the branching factors and results in an inability to obtain a better estimation value of the nodes from the root when the action space becomes huge. Overall, hard pruning, progressive widening and domain knowledge pruning can improve the performance since using domain knowledge has more notable effects (around 2%) in improving the performance compared with progressive widening (around 1%). Despite significantly improving the performance, the usage of this kind of pruning is limited as the knowledge is designed for a particular game. This means that the creation of a different pruning domain knowledge is needed for different games.

Table II shows the statistics of two versions of MCTS over 15000 games: average winning rates over three baselines approaches, final score, researched technologies percentage, number of cities owned and final production. The winning rate is the average rate based on three different baseline approaches. The hard pruning MCTS with move pruning has the highest winning rate (59.3%), followed by the progressive widening MCTS with move pruning (59.1%). This indicates that the move pruning has a more significant effect in improving the performance as the winning rate increased by around 2.5% compared with the MCTS without move pruning. On the other hand, the wining rate of the progressive widening MCTS is only higher than the obtained by its hard pruning counterpart when the move pruning is not applied. After applying the pruning, the final score and the researched techniques percentage of the MCTS are much closer to the ones obtained by the RB agents, while the number of cities owned, and final production is only slightly increased compared with the original MCTS. The MCTS with moving pruning has better statistics compared with those without moving pruning, while the progressive widening MCTS only increases the researched techniques percentage compared with hard pruning MCTS with/without moving pruning. This statistic suggests that the modified MCTS might be more focused on occupying cities rather than trying to maximise the final score as it displays a similar statistic as the RB agent, the latter being mainly focused on taking over units, ruins and cities. This outcome may be a consequence of the new features in the heuristic functions giving extra value for units in the enemy's cities, increasing, in this way, the possibility of choosing invading actions rather than accumulating the scores.

C. Tree analysis

This section analyses which action group and actions are executed the most, the tree depth, the visited time of a chosen action, and the fully expanded rate of the recommend action nodes in terms of different action spaces, being measured without pruned move actions and destroy actions, and turn in the game. The latter is measured between 0 and 1, where a value of 1 means that the recommend action node has visited all of its possible actions, excluding the pruned nodes, from its current state and vice versa.

During the game, the *unit* action group is the highest frequency chosen, 68% of the time. The *city* and *tribe* action groups are only chosen to be 20.5% and 11.4% of the time respectively. This difference might be because unit actions possess a large proportion in the action space. The most frequently executed action is *move*, with a percentage of 58.9% among all of the actions; this explains why the *unit* action group has the largest proportion. Although the move action does not provide a significant reward, it can gain bigger remuneration when the unit moves to ruins and to the enemy's city. Applying move pruning does not change the proportion between the action groups. This might be due to the fact that move pruning has a minimum move action number, implying that the remaining move actions still can be selected before ending the turn.

Figures 2 to 4 display the influence of the action size in the depth of MCTS, and the visited time and fully expanded rate of the recommended action node. The original MCTS was unable to go deeper than 2 and visited all of the possible actions starting from the recommended one when the action size is over 50. Moreover, the visited time of recommend action is closer to 0 when the action size is over 50; this means that the visited node statistic might not be accurate. With the aid of pruning techniques, this problem appears when the size of the actions is over 180, more than three times larger than original MCTS. The depth of the tree sharply decreases when the action size is small; moreover, progressive widening is more affected by this problem than hard pruning when the action size is between 100 and 150. The reason behind this might be because progressive widening unprunes the tree, resulting in more time used to visit unseen nodes. Comparing the visited time of a chosen action, the hard pruning has a higher visited time prior to an action size of 180. This might be a consequence of the fact that progressive widening unprunes the tree, resulting in an increased focus on exploration compared with the MCTS with hard pruning when the action size increases. Compared to hard pruning, progressive widening is unable to fully explore the chosen action when the action size is over 100. Although hard pruning also faces this issue, it occurs much later. These three figures show that progressive widening performs worse than hard pruning when the action size becomes substantial.

TABLE III
THE STATISTICS OF ACTION SPACE FOR EACH VERSION OF MCTS

	Hard Pruning				Progressive Widening			
	Mean	**Std.**	**Median**	**Max**	**Mean**	**Std.**	**Median**	**Max**
Move Pruning	28.69	34.65	16	566	28.2	34.38	16	474
No Move Pruning	33.12	41.96	18	623	32.02	40.67	17	608

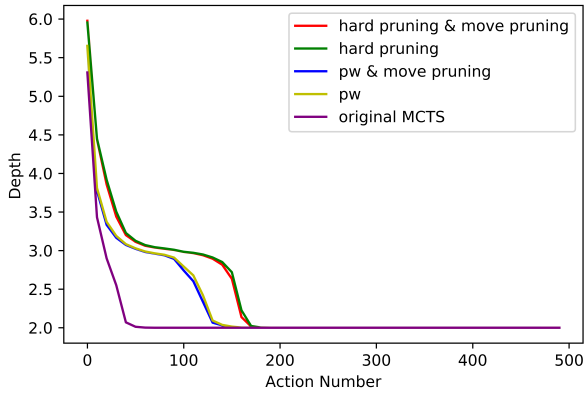

Fig. 2. Depth of MCTS under different action size

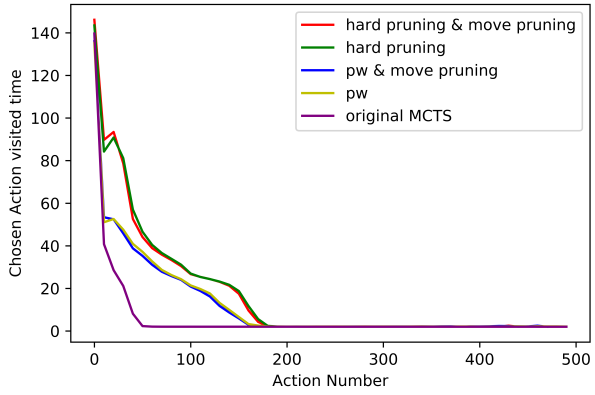

Fig. 3. Visited times of a chosen action in MCTS under different action size

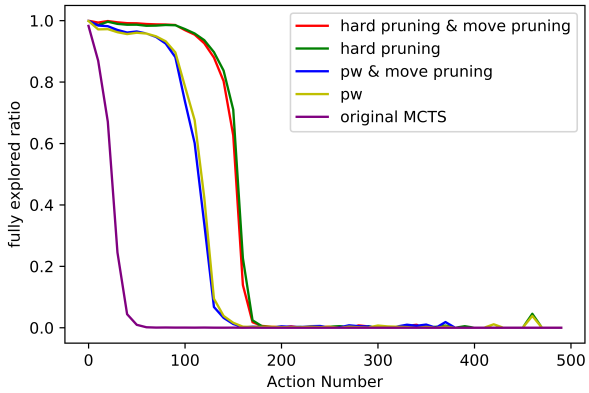

Fig. 4. Fully explored rates for the recommended action in MCTS under different action sizes

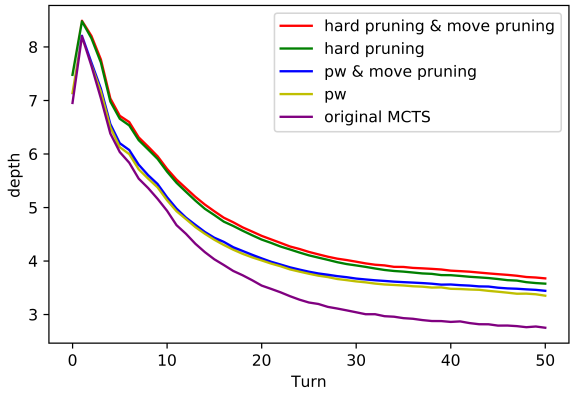

Fig. 5. Depth of MCTS under different turn

Although the moving pruning does not change the trend of neither MCTS with hard pruning nor progressive widening, the size of the action space decreases significantly. Table III shows the statistics (mean, standard derivation, median and maximum) of the size of the action space for each version of this agent. The MCTS with hard pruning and progressive widening has similar statistics, with the ones for former being slightly higher. On the other hand, the move pruning shows a the significantly effect in decreasing the the action space. The average action space decreases to around 4 and the maximum size of the action space drops to approximately 80 and 130 for hard pruning and progressive widening, respectively. This improved performance can be explained by observing that the MCTS prunes a considerable number of move actions, which has a significant effect in decreasing the action space.

Figures 5 to 7 show the depth of MCTS, visited times and fully expanded rate of the recommend action node, respectively. The depth of MCTS drops to around 3 when the turn in the game is 30. The visited time of a chosen action starts at 160 and drops to around 40 after 50 turns, a number 4 times smaller than the initial visiting time. The fully explored rate of a chosen action starts to drop after 10 turns and finishes with 40% in turn 50. The modified versions of MCTS also show a similar trend but drop at a much slower rate than the original MCTS. The depth of the modified MCTS is around 4, and the fully explored rate of the chosen action is at least two times larger compared with the original MCTS in turn 50. The visited time of a chosen action shows the different results in turn 50, the hard pruning MCTS with move pruning has the highest visited time and progressive widening has the smallest visited time among two versions of MCTS. The depth and the explored rate of a chosen action decreases when the turn increases, while the depth drops much earlier than the fully

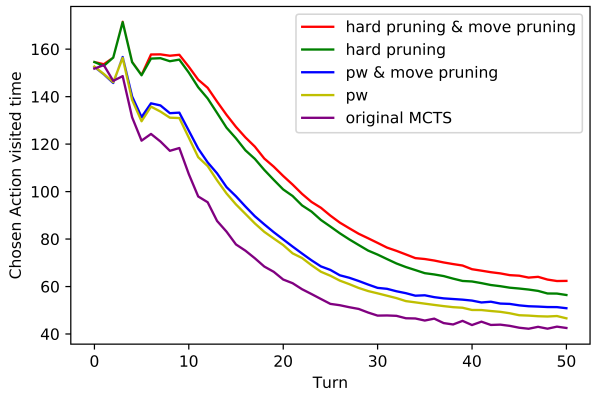

Fig. 6. Visited time of a chosen action in MCTS under different turn

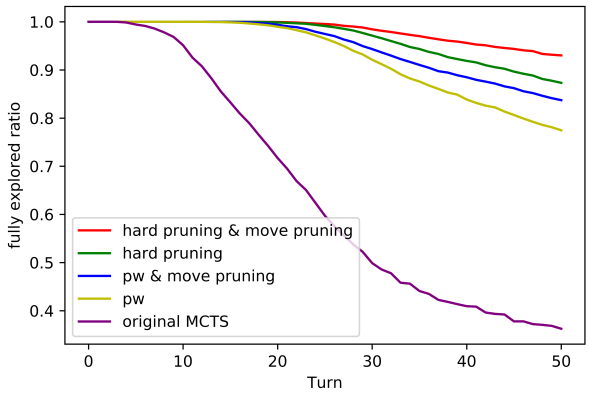

Fig. 7. Fully explored rate for the recommended action in MCTS under different turn

explored rate of the chosen action. This result might be due to action sizes generally increasing with the number of turns. Hard pruning decreases slowly than progressive widening, and move pruning further slows down the decrease. It is shown that move pruning has significant effects in increasing the explored ratio of the chosen action in later turns.

V. Conclusion and Future Work

The main focus of this paper was to investigate the effect of pruning techniques for MCTS in a multi-action strategy game. Compared with the classical board games, such as GO, Chess, this kind of game has a higher branching factor, which causes the drop of agent performance, and allows executing multiple actions in turn. Both OEP and EvoMCTS are good candidates to solve this issue for strategy games, and shows decent performance in the game HeroAIcamdey. However, it is limited to those games that can create a fixed action size representation, which is not the case for Tribes and most of the strategy games. The complexity of Tribes is higher than HeroAIcamdey as the agent requires not only to manage units but also the economy, technology and resources. Moreover, the size of the action space is not fixed and generally increases during each turn.

Because it is unable to create a fixed action size representation for neither OEP nor EvoMCTS, the MCTS using a single action as the representation with pruning techniques is used. Plain MCTS can handle large branching factors up to the order of hundreds (Baier & Cowling 2018, Chaslot et al. 2008). This limitation makes the plain MCTS not a good fit for Tribes. The pruning techniques can ease the huge branching factor issue for plain MCTS. This paper showed that both progressive widening and move pruning (domain knowledge pruning) can increase the performance of MCTS beyond the advantage gained from hard pruning. However, move pruning has a more significant effect in improving the performance; nonetheless, this pruning method is only limited to Tribes. It also analysed the tree behaviour under pruning techniques, indicating that pruning techniques are able to deal with the issue while being limited to an action size up to around 200, a much larger quantity than the original MCTS.

There are two directions for future work. The first one is to foucs on domain knowledge pruning for Tribes. These pruning methods can either be hand-crafted or created by either machine learning or reinforcement learning techniques. The hand-crafted domain knowledge pruning can be done by analysing the actions. In this paper, it is shown that decreasing the proportion of *move* actions increases the performance of the agent. Pruning the action such as Level up and Build action might also be beneficial for MCTS. Pruning based on machine learning or reinforcement learning techniques can learn from the data and environment to prune the actions that tend to give low rewards. This method is favoured as it can be applied to a general game rather than a specific game playing. Another possible direction is to modifying the OEP and EvoMCTS to allow the usage in flexible action size representation. This might be achieved by selecting the the representation for OEP or EvoMCTS based on the action space of the current state and the number of turn in the game. However, the impact of an action should be considered as it will influence later ones, resulting in increasing or decreasing the action size.

References

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

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012), 'A survey of monte carlo tree search methods', *IEEE Transactions on Computational Intelligence and AI in games* **4**(1), 1–43.

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

Childs, B. E., Brodeur, J. H. & Kocsis, L. (2008), Transpositions and move groups in monte carlo tree search, *in* '2008 IEEE Symposium On Computational Intelligence and Games', IEEE, pp. 389–395.

Churchill, D. & Buro, M. (2013), Portfolio greedy search and simulation for large-scale combat in starcraft, *in* '2013 IEEE Conference on Computational Inteligence in Games (CIG)', IEEE, pp. 1–8.

Churchill, D. & Buro, M. (2015), Hierarchical portfolio search: Prismata's robust ai architecture for games with large search spaces, *in* 'Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference'.

Coulom, R. (2007), 'Computing "elo ratings" of move patterns in the game of go', *ICGA journal* **30**(4), 198–208.

Gaina, R. D., Lucas, S. M. & Liebana, D. P. (2017), Rolling Horizon Evolution Enhancements in General Video Game Playing, *in* 'Proceedings of IEEE Conference on Computational Intelligence and Games', pp. 88–95.
URL: *http://ieeexplore.ieee.org/document/8080420/*

Gelly, S. & Silver, D. (2007), Combining online and offline knowledge in uct, *in* 'Proceedings of the 24th international conference on Machine learning', pp. 273–280.

Gelly, S. & Wang, Y. (2006), 'Exploration exploitation in go: Uct for monte-carlo go'.

Justesen, N., Mahlmann, T., Risi, S. & Togelius, J. (2017), 'Playing multiaction adversarial games: Online evolutionary planning versus tree search', *IEEE Transactions on Games* **10**(3), 281–291.

Justesen, N., Mahlmann, T. & Togelius, J. (2016), Online evolution for multi-action adversarial games, *in* 'European Conference on the Applications of Evolutionary Computation', Springer, pp. 590–603.

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

Midjiwan AB (2016), *The Battle of Polytopia*.

Perez, D., Hsu, Y.-J., Emmanouilidis, S., Khaleque, B. D. A. & Gaina, R. D. (2020), Tribes: A new turn-based strategy game for ai research, *in* 'Proceedings AAAI-20'.

Perez, D., Samothrakis, S., Lucas, S. & 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', p. 351–358.

Ramanujan, R., Sabharwal, A. & Selman, B. (2010), On adversarial search spaces and sampling-based planning, *in* 'Twentieth International Conference on Automated Planning and Scheduling'.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016), 'Mastering the game of go with deep neural networks and tree search', *nature* **529**(7587), 484–489.

Teytaud, F. & Teytaud, O. (2009), Creating an upper-confidence-tree program for havannah, *in* 'Advances in Computer Games', Springer, pp. 65–74.

MSc Project - Reflective Essay

Project Title:	MCTS pruning in Turn-Based Strategy Games
Student Name:	**Yu-Jhen Hsu**
Student Number:	**190279963**
Supervisor Name:	Dr. Diego Perez Liebana
Programme of Study:	MSc Artificial Intelligence

My research topic is applying an MCTS agent into a multi-action strategy game: Tribes. Monte Carlo Tree Search (MCTS) (Browne et al. 2012) is one of the state-of-the-art search algorithms in gameplay. Besides applying MCTS in games, it is capable of doing numerical algorithms, planning and optimization. It works by searching the possible move from the root and returning the most promising action. The well-known artificial intelligence (AI) agents using MCTS are AlphaZero (Silver et al. 2017) which combines deep reinforcement learning and MCTS to achieve the human-level agent in playing GO. Besides the AlphaZero, plain MCTS with a well-defined heuristic function, being a method to evaluate the state, also obtains a decent performance in the game GO. The possibility of using these algorithms to achieve human-level performance and the ability to using in general game playing in one of the main goals in-game AI domains.

1. The difference between the theory and practical work

Although MCTS shows the decent performance in classic board games such as Go, MCTS still struggles in some of the games which contain the huge branching factor. The huge branching factor results in the search being unable to go deeper enough in the tree and the rarely-visited nodes are not useful. This issue is highlighted in many papers (Justesen et al. 2017, Baier & Cowling 2018). It is also shown in the game Tribes, in which the agent performance drop when the action space increases. The EvoMCTS (Justesen et al. 2017) and OEP (Baier & Cowling 2018) are two algorithms that deal with this issue in the strategy game and obtain good performance in the game HeroAIcadmy. These two algorithms are limited to some strategy games as they require the fixed action size for the representation. Because the HeroAIcamdy only requires the agent to decide an action for every unit in the game per turn, it allows the algorithm to set up the fixed-size representation based on the unit size, being 5.

However, taking the unit size as the representation for a turn is not feasible for Tribes and most of the strategy games. It is because the unit is not the only factor that influences the action size. In the game Tribe, three factors that might influence the action space. Firstly, the game elements, such as the star, researched technologies, resources and building, affect the action space. These elements affect the action space as it allows the agent to execute some actions on these elements or unlocking the features. The second factor is the sequence of executed actions per turn. It is because the previous actions will influence the later one and result in the change of action space within a turn. The action space might increase or decrease after executing some actions. The research action is one example that might increase the action space by unlocking some features (new unit, building types, or actions). Another example of previous actions that might decrease the action is spawning. In the game Tribe, it is only allowed to spawn a unit in every owned city per turn. However, the action space might contain the multiple spawn action for a city, in which each action is corresponding to the different type of unit. Once the spawn action for a city is executed, the other spawn unit action will disappear in the action space for that turn. The last factor that will influence the action space is turn. The action size will generally increase during the game process. These three factors make it hard to find the fixed action size as the representation for neither EvoMCTS nor OEP.

It shows the gap between theory and practical work. Although these two algorithms show the impressive performance in theory, it is limited to implement these two algorithms in Tribes and most of the strategy games. Although there is an OEP algorithm

implemented in the Tribes, it conquers the problem when using it to play the game. The OEP uses the flexible action size in the Tribes, which do the crossover and mutation operator based on two representations. The one reason OEP has the issue in the game might be because an invalid representation is created during the crossover and mutation. The reason for an invalid representation might be because some actions in the representation are invalid after executing the previous actions. Besides the limitation in creating a valid representation, the new action, which is created after executing some actions, might be ignored. It is because the actions that can be used inside the representation are based on the beginning state of the current turn. This method might result in some actions that appear after executing some actions are ignored. One possibility of solving this issue under OEP might be checking the feasibility of representation after operators. More research on how to adapt the new actions in crossover and mutation operators that might appear within a turn is needed.

Instead of using the turn as representation, using an action as the representation could be a solution. In this way, we can use the forward model, being the model that is rolling over the state based on the action, to obtain the available action sets from the state. However, plain MCTS falls into a huge branching factor. The one possible way to solve this issue is using the pruning techniques to remove some nodes giving the worst reward. There are three pruning techniques used in this paper, being hard pruning (Browne et al. 2012), progressive widening (Coulom 2007, Chaslot et al. 2008) and pruning based on domain knowledge (Ramanujan et al. 2010). The hard pruning, MCTS prunes the tree when the visited time of nodes is over budget. The most of children of nodes that are over budget can then be pruned, leaving the few most promising nodes. The heuristic function, which evaluates the state of the node, is used to decide which children should be pruned. However, the main problem of hard pruning is that we might delete the children that provide a good reward in the long term as we only evaluate the state of children. The progressive widening is used to deal with this problem which unpruned the removed children when the iteration increases. This method allows the agent the alleviate the risk of removing the children with a good reward. Although progressive widening shows better performance in the GO (Coulom 2007, Chaslot et al. 2008), it does not show a significant effect in improving the performance of MCTS for Tribes. This result might be because the pruning progressive conquers the huge branching factor problem in the Tribe.

The pruning with domain knowledge is used on top of either hard pruning or progressive widening. This method is added after the first experiment. The inspiration for using move pruning is by analysing the chosen action under the tree. It shows that the largest proportion of chosen action for MCTS is the move action, which contains more than 50% of the chosen action but without the significant reward. The reward of the move action is in the last five of the unit action group. The reason that move action is chosen the most might be because the action space for a turn contains a significant number of the move action. The move action is decided by the unit number, its moving ability and the road near the unit. In the later of games, the tribe tends to contain a significant number of units. This result allows the possibility the decrease the action space by pruning some of the unhelpful move actions for the nodes. The move pruning is decided by evaluating the state of the move action, which does not move the unit to the enemy's city or ruin. Most of the move actions are pruned while those pruned move actions might be considered in the node's children. It is because the MCTS represents the node by a single action and obtain the action space by the state of the node, which allows the pruned move action might appear in the action space in the later nodes of the tree. The result of cooperating move action is promising, which increases around 3% of winning rate of MCTS using hard pruning without move pruning when competing against the baseline models. Using move pruning to enhance the MCTS with progressive widening is less promising, which only increases by approximately 2% compared with the winning rate of progressive widening MCTS without move pruning. However, the moving pruning might be improved farther by analysing the relationship between the number of units and action space and using the statistics to decide the need of move pruning under different

unit size.

The result of the experiment does not show much difference compared with the previous work. The pruning techniques show the significant effect in dealing with the branching factors. The statistic of the tree shows the original MCTS suffers from the large action size and result in a shallow tree. Moreover, when the action space becomes over a hundred, the chosen action is only visited a few times. With the help of pruning techniques, although the tree is still shallow when the action size becomes huge, it happens much later than the original MCTS. The progressive widening can alleviate the risk of missing any good move in theory which is also true in the experiment. This game results show that MCTS with progressive widening has a better performance compared with hard pruning. However, the MCTS with progressive widening drops the performance much quicker than the MCTS with hard pruning under large action space.

2. The strength and weakness of the agent

The main advantage of the agent in this paper is that it provides the statistics of MCTS during the experiment, which is rarely seen in the previous work. Although the statistic of MCTS does not affect improving the MCTS during play, it provides the possibilities to understand the decision that MCTS agent makes and the tree structure. Recording the statistics of MCTS might slow down the gameplay as the application requires to extract and record the statistics of MCTS. However, these statistics are helpful and can be used to design a more robust MCTS agent. These statistics provide a way to have a good understanding of MCTS in two directions. The first one is the tree structure which is allowed to investigate the depth of a tree. It shows that the MCTS with pruning can go much deeper comparing with original MCTS when the action space over hundreds in the experiment. Another direction is the chosen actions, which records the information of chosen action, such as the turn number in the game, the reward and the visited time. With the aid of the statistic of chosen action, it is much easier to create the pruning method based on domain knowledge.

The limitation of domain knowledge pruning is the main drawback of the agent. Although it shows the significant effect in improving the agent performance, the usage of move pruning is limited to Tribes. Although many strategy games contain the element of moving the unit, the criterion of a good move is different between games. The judge of good move in this paper is the move action can bring a unit to either ruins or enemy's city or obtaining a good reward, while this judgement might not be suitable for the other game. One possible solution for dealing with this issue is allowing the agent to learn which kind of actions should be pruned or kept. Nowadays, it can be done by reinforcement learning, which allows the agent to learn from doing or machine learning, which gains the insights from data. However, applying reinforcement learning might not be an easy job for Tribes, it is because the state of Tribes is significant, which contains a considerable number of features, such as resources, building and unit.

3. Future work

There are two directions I would like to take if I had more time for developing the project. The first direction is related to the domain knowledge pruning, which can ease the problem of enormous action space for Tribes. It is shown that the action space of Tribes can grow to around 500 during the gameplay. This issue drops the performance of agent significantly when the size of action space over 200 as the tree is shallow, and the chosen action is rarely-visited. With the aid of domain knowledge pruning, in which the destroy and move actions are pruned, the performance of MCTS significantly increases compared with original MCTS overturn. I want to try both hand-crafted more domain knowledge pruning and developing the general domain knowledge pruning. The former one is based on the understanding of the statistics of MCTS in the chosen actions. The general domain knowledge pruning relies on building the model, such as a machine learning model and reinforcement learning model, to pick up the actions that should be pruned by itself. This approach is favoured as it is possible for using domain knowledge

pruning in general game playing rather than a particular game.

Another direction is implementing the state-of-the-art algorithms, OEP and EvoMCTS, for Tribes. These two algorithms allow evaluating based on a whole turn rather than a single action and obtain a decent performance in the game HeroAIcamdy. The main problem for implementing these two approaches is representation. The method of creating a valid representation under crossover and mutation operator is needed for OEP. For both EvoMCTS and OEP, the method of deciding the actions that can be added to representation is also important. It is because the action space is changeable after executing some actions.

4. Ethical Issues of game AI

When developing AI applications, the ethical problems are always a critical concern. Although the game AI agent only exists in the game, it might influence human behaviours. The game AI might be able to change the human's behaviour in both a positive and negative way. Some game AI interacts with the users and be able to change their behaviours. The other game AI might be able to ease the negative feeling of the user, which is beneficial for society. Although these agents are designed to improve the personal life, the misusage of game AI agent might lead to the negative result. For example, the personal context of a game which is created by game AI increases the game entertaining but might result in the game addiction. The game AI, which is trained by the biased data, might result in some improper actions and influence the human behaviours. Although we can cooperate AI techniques into personal life and increase the game playability, the evaluation of how these techniques influence human behaviours should be researched first.

References

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

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. (2012), 'A survey of monte carlo tree search methods', IEEE Transactions on Compu- tational Intelligence and AI in games 4(1), 1–43.

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

Coulom, R. (2007), 'Computing "elo ratings" of move patterns in the game of go', *ICGA journal* **30**(4), 198–208.

Justesen, N., Mahlmann, T., Risi, S. & Togelius, J. (2017), 'Playing multiaction adversarial games: Online evolutionary planning versus tree search', IEEE Transactions on Games 10(3), 281–291.

Ramanujan, R., Sabharwal, A. & Selman, B. (2010), On adversarial search spaces and sampling-based planning, in 'Twentieth International Conference on Automated Planning and Scheduling'.

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