Investigating MCTS Modifications in General Video Game Playing

Frederik Frydenberg[1], **Kasper R. Andersen**[1], **Sebastian Risi**[1], **Julian Togelius**[2]
[1]IT University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, USA
chbf@itu.dk, kasr@itu.dk, sebr@itu.dk, julian@togelius.com

Abstract

While Monte Carlo tree search (MCTS) methods have shown promise in a variety of different board games, more complex video games still present significant challenges. Recently, several modifications to the core MCTS algorithm have been proposed with the hope to increase its effectiveness on arcade-style video games. This paper investigates of how well these modifications perform in general video game playing using the *general video game AI* (GVG-AI) framework and introduces a new MCTS modification called *UCT reverse penalty* that penalizes the MCTS controller for exploring recently visited children. The results of our experiments show that a combination of two MCTS modifications can improve the performance of the vanilla MCTS controller, but the effectiveness of the modifications highly depends on the particular game being played.

1 Introduction

Game-based AI competitions have become popular in benchmarking AI algorithms [22]. However, typical AI competitions focus only on one type of game and not on the ability to play a variety of different games well (i.e. the controllers only work on one particular game / game type / problem). In this context, an important question is if it is possible to create controllers that can play a variety of different types of games with little or no retraining for each game.

The *general video game AI* (GVG-AI) competition explores this challenge. To enter the competition, a controller has to be implemented in the GVG-AI framework (available at: http://www.gvgai.net/). The framework contains two sets with ten different games in each set. The games are replicas of popular arcade games that have different winning conditions, scoring mechanisms, sprites and player actions. While playing a game the framework gives the controller a time limit of 40 milliseconds to return the next action. If this limit is exceeded, the controller will be disqualified. The competition and framework is designed by a group of researchers at University of Essex, New York University and Google DeepMind [18; 17].

The Monte Carlo tree search (MCTS) algorithm, performs well in many types of games [4; 9; 10; 8]. MCTS was first applied successfully to the Asian board game *Go*, for which it rapidly redefined the state of the art for the game. Whereas previous controllers had been comparable to human beginners, MCTS-based controllers were soon comparable to intermediate human players [10]. MCTS is particularly strong in games with relatively high branching factors and games in which it is hard to develop a reliable state value estimation function. Therefore, MCTS-based agents are generally the winners in the annual *general game playing competition*, which is focused on board games and similar discrete, turn-based perfect-information games [7; 1].

Beyond board games, MCTS has also been applied to arcade games and similar video games. In particular, the algorithm has performed relatively well in Ms. Pac-Man [13] and Super Mario Bros [8], though not better than the state of the art. In the general video game playing competition, the best agents are generally based on MCTS or some variation thereof [16; 17]. However, this is not to say that these agents perform very well – in fact, they perform poorly on most games. One could note that arcade-like video games present a rather different set of challenges to most board games, one of the key differences being that random play often does not lead to any termination condition.

The canonical form of MCTS was invented in 2006, and since then many modifications have been devised that perform more or less well on certain types of problems. Certain modifications, such as *rapid action value estimation* (RAVE) perform very well on games such as Go [6] but show limited generalization to other game types. The work on Super Mario Bros mentioned above introduced several modifications to the MCTS algorithm that markedly improved performance on that particular game [8]. However, an important open question is if those modifications would help in other arcade-like video games as well?

The goal of this paper is to investigate how well certain previously proposed modifications to the MCTS algorithm perform in general video game playing. The "vanilla MCTS" of the GVG-AI competition framework is our basis to test different modifications to the algorithm. In addition to comparing existing MCTS modifications, this paper presents a new modification called *reversal penalty*, which penalizes the MCTS controller for exploring recently visited positions. Given that the games provided with the GVG-AI framework differ along a number of design dimensions, we expect this evaluation to give a better picture of the capabilities of our new MCTS variants than any one game could do.

The paper is structured as follows: Section 2 describes related work in GVG, MCTS and use of MCTS. Section 3 describes the GVG-AI framework and competition and how we used it. Section 4 explains the MCTS algorithm, followed by the tested MCTS modifications in Section 5. Section 6 details the experimental work and finally Section 7 discuss the results and describes future work.

2 Related work

2.1 General Video Game Playing

The AAAI general game playing competition by Stanford Logic Group of Stanford University [7] is one of the oldest and most well-known general game playing frameworks. The controllers submitted for this competition receive descriptions of games at runtime, and use the information to play these games effectively. The controllers do not know the type or rules of the game beforehand. In all recent iterations of the competition, different variants of the MCTS algorithm can be found among the winners of the competition.

The general video game playing competition is a recent addition to the set of game competitions [18; 17]. Like the Stanford GGP competition, submitted controllers are scored on multiple unseen games. However, unlike the Stanford GGP competition the games are arcade games inspired by 1980's video games, and the controllers are not given descriptions of the games. They are however given forward models of the games. The competition was first run in 2014, and the sample MCTS algorithm reached third place. In first and second place were MCTS-like controllers, i.e. controllers based on the general idea of stochastic tree search but implemented differently. The sample MCTS algorithm is a vanilla implementation and is described in Browne et al. [2]. The iterations of the algorithm rarely reach a terminal state due to the time constraints in the framework. The algorithm evaluates the states by giving a high reward for a won game and a negative reward for a lost game. If the game was neither won or lost, the reward is the game's score. The play-out depth of the algorithm is ten moves.

2.2 MCTS Improvements

A number of methods for improving the performance of MCTS on particular games have been suggested since the invention of the algorithm [19; 2]. A survey of key MCTS modifications can be found in Browne et al. [2]. Since the MCTS algorithm has been used in a wide collection of games, this paper investigates how the different MCTS modifications perform in general video game playing.

Some of these strategies to improve the performance of MCTS were deployed by Jacobsen et al. [8] to play Super Mario Bros. The authors created a vanilla MCTS controller for a Mario AI competition, which they augmented with additional features. To reduce cowardliness of the controller they increased the weight for the largest reward. Additionally, macro actions [15; 8] were employed to make the search go further without increasing the number of iterations. Partial expansion is another technique to achieve a similar effect as macro actions. These modifications resulted in a very good performing controller for the Mario game. It performed better than Robin Baumgarten's A* version in noisy situations and performed almost as well in normal playthroughs.

Pepels et al. [13] implemented five different strategies to improve existing MCTS controllers: A variable depth tree, playout strategies for the ghost-team and Pac-Man, long-term goals in scoring, endgame tactics and a last-good-reply policy for memorizing rewarding moves. The authors achieved an average performance gain of 40962 points, compared to the CIG'11 Pac-Man controller.

Chaslot et al. [3] proposed two strategies to enhance MCTS: Progressive bias to direct the search according to possibly time-expensive heuristic knowledge and progressive unpruning, which reduces the branching factor by removing children nodes with low heuristic value. By implementing these techniques in their Go program, it performed significantly better.

An interesting and well-performing submission to the general game playing competition is *Ary*, developed by Méhat et al. [11]. This controller implements parallelization of MCTS, in particular a "root parallel" algorithm. The idea is to perform individual Monte Carlo tree searches in parallel on different CPUs. When the framework asks for a move, a master component chooses the best action among the best actions suggested by the different trees.

Perez et al. [16] used the GVG-AI framework and proposed augmentations to deal with some of the shortcomings of the sample MCTS controller. MCTS was provided with a knowledge base to bias the simulations to maximize knowledge gain. The authors use fast evolutionary MCTS, in which every roll-out evaluates a single individual of the evolutionary algorithm and provides the reward calculated at the end of the roll-out as a fitness value. They also defined a score function that uses a concept knowledge base with two factors: curiosity and experience. The new controller was better in almost every game compared to the sample MCTS controller. However, the algorithm still struggled in some cases, for example in games in which the direction of a collision matters.

This paper uses the GVG-AI framework and builds on the sample MCTS controller. The next section will describe the GVG-AI framework in more detail.

3 GVG-AI Competition & Framework

The GVG-AI competition tries to encourage the creation of AI for general video game playing. The controller submitted to the competition webpage is tested in a series of unknown games, thereby limiting the possibility of applying any specific domain knowledge. The competition is held as part of several international conferences since 2014. Part of the GVG-AI competition is the *video game description language* (VGDL) [5; 21] that describes games in a very concise manner; all of the games used in the competition are encoded in this language. Examples are available from the GVG-AI website.

Users participate in the competition by submitting Java code defining an agent. At each discrete step of the game simulation, the controller is supposed to provide an action for the avatar. The controller has a limited time of 40ms to respond with an action. In order for the controller to simulate possible moves, the framework provides a forward model of the game. The controller can use this to simulate the game for as many ticks as the time limit allows.

For a more detailed explanation of the framework, see the GVG-AI website or the competition report [17].

4 Monte Carlo Tree Search

Monte Carlo tree search (MCTS) is a statistical tree search algorithm that often provides very good results in time restricted

The CIG 2014 Training Game Set	
Aliens(G_1)	In this game you control a ship at the bottom of the screen shooting aliens that come from space. You better kill them all before they reach you! Based on *Space Invaders*.
Boulderdash(G_2)	Your objective here is to move your player through a cave, collecting diamonds, before finding the exit. Beware of enemies that hide underground!
Butterflies(G_3)	You are a happy butterfly hunter. This is how you live your life, and you like it. So be careful, you don't want them to become extinct!
Chase(G_4)	You like to chase goats. And kill them. However, they usually don't like you to do it, so try not to get caught doing that!
Frogs(G_5)	Why did the frog cross the road? Because there is a river at the other side. What would you cross the river as well? Because your home is there, and it's cosy.
Missile Command(G_6)	Some missiles are being shot to cities in your country, you better destroy them before they reach them!
Portals(G_7)	You control an avatar that needs to find the exit of a maze, but moving around is not so simple. Find the correct doors that take you to the exit!
Sokoban(G_8)	In this puzzle you must push the boxes in the maze to make them fall through some holes. Be sure you push them properly!
Survive Zombies(G_9)	How long can you survive before you become their main course for dinner? Hint: zombies don't like honey (didn't you know that?).
Zelda(G_10)	Get your way out of the dungeon infested with enemies. Remember to find the key that opens the door that leads you to freedom!

Table 1: The game descriptions of the training set from the official competition site.

The CIG 2014 Evaluation Game Set	
Camel Race(G_1)	The avatar must get to the finish line before any other camel does.
Digdug(G_2)	The avatar must collect all gems and gold coins in the cave, digging its way through it. There are also enemies in the level that kill the player on collision with him. Also, the player can shoot boulders by pressing USE two consecutive time steps, which kill enemies.
Firestorms(G_3)	The avatar must find its way to the exit while avoiding the flames in the level, spawned by some portals from hell. The avatar can collect water in its way. One unit of water saves the avatar from one hit of a flame, but the game will be lost if flames touch the avatar and he has no water.
Infection(G_4)	The avatar can get infected by colliding with some bugs scattered around the level, or other animals that are infected (orange). The goal is to infect all healthy animals (green). Blue sprites are medics that cure infected animals and the avatar, but don't worry, they can be killed with your mighty sword.
Firecaster(G_5)	The avatar must find its way to the exit by burning wooden boxes down. In order to be able to shoot, the avatar needs to collect ammunition (mana) scattered around the level. Flames spread, being able to destroy more than one box, but they can also hit the avatar. The avatar has health, that decreases when a flame touches him. If health goes down to 0, the player loses.
Overload(G_6)	The avatar must reach the exit with a determined number of coins, but if the amount of collected coins is higher than a (different) number, the avatar is trapped when traversing marsh and the game finishes. In that case, the avatar may kill marsh sprites with the sword, if he collects it first.
Pacman(G_7)	The avatar must clear the maze by eating all pellets and power pills. There are ghosts that kill the player if he hasn't eaten a power pill when colliding (otherwise, the avatar kills the ghost). There are also fruit pieces that must be collected.
Seaquest(G_8)	The player controls a submarine that must avoid being killed by animals and rescue divers taking them to the surface. Also, the submarine must return to the surface regularly to collect more oxygen, or the avatar would lose. Submarine capacity is for 4 divers, and it can shoot torpedoes to the animals.
Whackamole(G_9)	The avatar must collect moles that pop out of holes. There is also a cat in the level doing the same. If the cat collides with the player, this one loses the game.
Eggomania(G_10)	There is a chicken at the top of the level throwing eggs down. The avatar must move from left to right to avoid eggs breaking on the floor. Only when the avatar has collected enough eggs, he can shoot at the chicken to win the game. If a single egg is broken, the player loses the game.

Table 2: The game descriptions of the evaluation set from the official competition site.

situations. It constructs a search tree by doing random playouts, using a forward model, and propagates the results back up the tree. Each iteration of the algorithm adds another node to the tree and can be divided into four distinct parts. Figure 1 depicts these four steps. The first step is the **selection** step, which selects the best leaf candidate for further expansion of the tree. Starting from the root, the tree is traversed downwards, until a leaf is reached. At each level of the tree the best child node is chosen, based on the *upper confidence bound* (UCB) formula (described below). When a leaf is reached, and this leaf is not a terminal state of the game, the tree is **expanded** with a single child node from the action space of the game.

From the point of the newly expanded node, the game is **simulated** using the forward model. The simulation consist of doing random moves starting from this game state, until a terminal state is reached. For complex or, as in our case, time critical games, simulation until a terminal state is often unfeasible. Instead the simulation can be limited to only forward the game a certain amount of steps. After the simulation is finished, the final game state reached is evaluated and assigned a score. The score of the simulation is **backpropagated** up through the parents of the tree, until the root node is reached. Each node holds a total score, which is the sum of all backpropagated scores, and a counter that keeps track of the number of times the score was updated; this counter is equal to the number of times the node was visited.

4.1 Upper Confidence Bound - UCB

The UCB formula selects the best child node at each level when traversing the tree. It is based on the bandit problem, in which it selects the optimal arm to pull, in order to maximize rewards. When used together with MCTS it is often referred to as upper confidence bounds applied to trees, or UCT [2]:

$$\text{UCT} = \overline{X_j} + 2C_p\sqrt{\frac{2\ln n}{n_j}},$$

where $\overline{X}_J$ is the average score of child j. n is the number of times the parent node was visited and n_j is the number of times this particular child was visited. C_P is a constant adjusting the value of the second term. At each level of the the selection step, the child with the highest UCT value is chosen.

4.2 Exploration vs. Exploitation

The two terms of the UCT formula can be described as the balance between exploiting nodes with previously good scores, and exploring nodes that rarely have been visited [2].

The first term of the equation, $\overline{X}_J$, represents the exploitation part. It increases as the backpropagated scores from its child nodes increases.

The second term, $\sqrt{\frac{2ln(n)}{n_j}}$, increases each time the parent node has been visited, but a different child was chosen. The constant C_p simply adjust the contribution of the second term.

4.3 Return Value

When the search is halted, the best child node of the root is returned as a move to the game. The best child can either be the node most visited, or the one with the highest average value. This will often, but not always, be the same node [2].

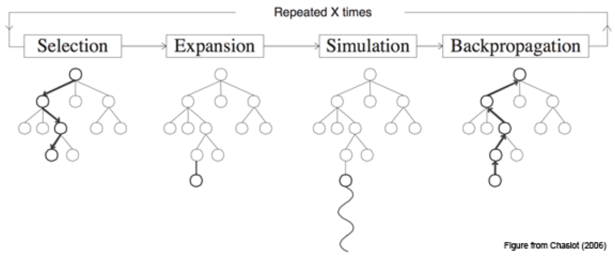

Figure 1: The main steps in the Monte-Carlo Tree Search.

4.4 Algorithm Characteristics

Anytime

One of the strengths of MCTS in a game with very limited time per turn, is that the search can be halted at anytime, and the currently best move can be returned.

Non-heuristic

MCTS only needs a set of legal moves and terminal conditions to work. This trait is very important in the GVG-AI setting, where the games are unknown to the controller. If a playout from any given state has a high probability of reaching a terminal state (win or lose), no state evaluation function needs to be used; as this is not the case in general for games in the GVG-AI set, a state evaluator is necessary.

Asymmetric

Compared to algorithms like minimax, MCTS builds an asymmetric search tree. Instead of mapping out the entire search space, it focuses efforts on previously promising areas. This is crucial in time critical application.

5 Modifications to MCTS

Here we list the particular modifications to the basic MCTS algorithm that we have investigated in this paper.

5.1 MixMax backups

Mixmax increases the risk-seeking behavior of the algorithm. It modifies the exploitation part of UCT, by interpolating between the average score and the maximum score:

$$Q \cdot \text{maxScore} + (1 - Q) \cdot \overline{X_j},$$

where Q is a value in the range $[0, 1]$. A good path of actions will not greatly affect the average score of a node, if all other children lead to bad scores. By using mixmax the good path contributes more to the average than the bad ones, thereby reducing the defensiveness of the algorithm. This modification was proposed in response to a problem observed when applying MCTS to Super Mario Bros, where Mario would act cowardly and e.g. never initiate a jump over a gap as most possible paths would involve falling into the gap. Mixmax backups made Mario considerably bolder in the previously cited work [8].

For the experiments with mixmax in this paper, the Q value was set to 0.1. The Q value was determined through prior experimentation.

5.2 Macro Actions

As stated previously, each action has to be decided upon in a very short time-span. This requirement can often lead to search tree with a limited depth, and as such only the nearest states are taken into consideration when deciding on the chosen action. Macro actions enable a deeper search, at the cost of precision. Powley et al. have previously shown that this is an acceptable tradeoff in some continuous domains [20].

Macro actions consists of modifying the expansion process, such that each action is repeated a fixed number of times before a child node is created. That is, each branch corresponds to a series of identical actions. This process builds a tree that reaches further into the search space, but using coarser paths.

5.3 Partial Expansion

A high branching factor relative to the time limit of the controller results in very limited depth and visits to previously promising paths. Even though a child might have resulted in a high reward, it will not be considered again, before all other children have been expanded at least once. The *partial expansion* modification allows the algorithm to consider "grandchildren" (and any further descendants) of a node before all children of that node have been explored. This allows for a deeper search at the cost of exploration. In the Mario study, Partial Expansion was useful combined with other modifications [8].

5.4 Reversal Penalty

A new MCTS modification introduced in this paper is *UCT reverse penalty*. A problem with the standard MCTS controller is that it would often just go back and forth between a few adjacent tiles. This oscillation is most likely due to the fact that only a small amount of playouts are performed each time MCTS is run, and therefore a single, or a few high scoring random playouts completely dominate the outcome. Additionally, when there are no nearby actions that result in a score increase, an action is chosen at random, which often also leads to behaviors that move back and forth. Instead, the goal of UCT reverse penalty is to create a controller that explores more of the given map, without increasing the search depth. To achieve this the algorithm adds a slight penalty to the UCT value of children that lead to a recently visited level tile (i.e. "physical position" in the 2D game world). However, the penalty has to be very small so it does not interfere with the normal decisions of the algorithm, but only affect situations in which the controller is going back and forth between fields. This modification is similar but not identical to exploration-promoting MCTS modifications proposed in some recent work [12; 14].

In the current experiments, a list of the five most recently visited positions is kept for every node, and the penalty is 0.05; when a node represents a state where the avatar position is one of the five most recent, its UCT value is multiplied by 0.95.

6 Experiments

The experiments in this paper are performed on the twenty games presented in Table 3 and Table 4. In order to make our results comparable with the controllers submitted to the general video game playing competition, we use the same software and scoring method. Each game is played five times for each combination of MCTS modifications, one playthrough per game level. This configuration follows the standard setup for judging competition entries. Each level has variations on the locations of sprites and in some games variations on non-player character (NPC) behavior. There are nine different combinations plus the vanilla MCTS controller, which gives 900 games played in total. The experiments were performed on the official competition site, by submitting a controller following the competition guidelines. All combinations derive from the four modifications explained in the previous section: mixmax scores, macro actions, partial expansion and UCT reverse penalty.

Two measures were applied when analyzing the experiments: the number of victories and the score. The GVG-AI competition weighs the number of victories higher than the achieved score when ranking a controller; it is more important to win the game rather than losing the game with a high score. In both Table 3 and 4 the scores are normalized to values between zero and one.

7 Results

The top three modifications are UCT reverse penalty, mixmax and partial expansion. According to the total amount of wins, the MCTS controller with UCT reverse penalty is the best performing controller. Thirty wins in the training set and seventeen wins in the validation set. The number of wins is slightly better than the number of wins of the vanilla MCTS controller (27). However, the vanilla controller receives a higher number of points (756), compared to the number of points of the UCT reverse penalty modification (601). Videos of the UCT reverse penalty controller can be found here: `http://bit.ly/1INlpF9`.

Compared to the vanilla MCTS controller, the mixmax modification alone does not increase the total amount of wins or points. It does however improve the performance in some games. In Missile Command the vanilla controller scored higher than mixmax, but they have an equal chance of winning. In the game Boulderdash, mixmax wins more often but scores less points. By applying mixmax, the controller wins games faster; points are scored whenever a diamond spawns (time based) and when the controller collects diamonds. Therefore the faster the win, the less points.

Combining UCT reverse penalty and mixmax shows promising results (Table 4). This controller was the highest scoring and most winning controller looking at the total values. It was the only controller winning any playthroughs in the game Eggomania. The gameplay is characterized by a controller that moves from side to side, whereas the other controllers only move as far as they can "see".

Interestingly, whenever a combination of modifications contains macro actions, the controller performs badly, both in terms of total score and total wins. As stated previously macro actions enables a deeper search, at the cost of precision. This ability enables macro controllers to succeed in games like Camelrace; as soon as it finds the goal the agent will move to it. The other MCTS controllers fail in Camelrace because they do not search the tree far enough. However, the MCTS modifications with macro action lose almost every other game type due to lack of precision. For example, in Pac-Man-like games, it searches too deep, likely moving past the maze junctions and never succeeding in moving around properly. The macro action experiments were done with a repeat value of three (i.e. using the same action three times in a row). The repeat value has a profound effect on how the controller performs, and is very domain specific. The repeat value for Camelrace should be very high, but in Pac-Man it should be very low for it to not miss any junctions.

The game Camelrace is uncovering one major problem of the MCTS controllers; the play-out depth is very limited, which is a problem in all games with a bigger search space. If the controller in Pac-man clears one of the corners of the maze, it can get stuck in that corner, therefore never reaching nodes that give points. The only controller that wins any games in Pac-Man is UCT reverse penalty and its combination with mixmax. UCT reverse penalty without mixmax scores most points, but with mixmax it wins all playthroughs and is ranked second in achieved score. The depth cannot be increased due to the time-limitations in the competition framework.

In the game Frogs, the avatar shows problematic behavior. The avatar has to cross the road quickly without getting hit by the trucks in the lanes. Most roll-outs are unable to achieve this. The most common behavior observed is a controller that moves in parallel to the lanes and is never crossing it. No controllers are able to win all the playthroughs, but controllers using mixmax scores or UCT reverse penalty sometimes win.

When comparing our results with the ranking on the official competition site, our controller is performing better on the validation set. The sampleMCTS scores 37 points and wins 16 of 50, where our controllers scores 75 points and wins 20 of 50. This places the controller on seventh place, four places higher than sampleMCTS. In the training set our controller scores less points, but wins three games more than the sampleMCTS. This places our controller on tenth place, three places lower than sampleMCTS.

8 Discussion and Future Work

According to our experiments the [UCT reverse penalty, mixmax] combination was the one that performed best overall, and the only one that convincingly beat Vanilla MCTS on the validation set. It should be noted that while we used the official scoring mechanism of the competition, higher number of playthroughs might have been preferable given the variability between games. Several games contains NPCs, and those have very different behaviors. Additionally, their behaviors are not only different per game, but also per playthrough. In games like Pac-Man (G_7 in the validation set), the enemy ghosts behave very stochastically. Because of this stochastic behavior, the results of five playthroughs will vary even using the same controller.

The presented results show that each MCTS modification only affects subsets of games, and often different subsets. One could argue that the sampleMCTS controller in the framework is rather well-balanced.

One could also argue that the fact that no single MCTS modification provides an advantage in all games shows that the set of benchmark games in GVG-AI provides a rich set of complementary challenges, and thus actually is a test of “general intelligence” to a greater degree than existing video game-based AI competitions. It remains to be seen whether any modification to MCTS would allow it to perform better across these games; if it does, it would be a genuine improvement across a rather large set of problems.

9 Conclusion

This paper investigated the performance of several MCTS modifications on the games used in the General Video Game Playing Competition. One of these modifications is reported for the first time in this paper: UCT reverse penalty, which penalizes the MCTS controller for exploring recently visited children. While some modifications increased performance on some subset of games, it seems that no one MCTS variation performs best in all games; every game has particular features that are best dealt with by different MCTS variations. This confirms the generality of AI challenge offered by the GVG-AI framework.

References

[1] Yngvi Bjornsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(1):4–15, 2009.

[2] Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[3] G.M.J.B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, and B. Bouzy. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[4] David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in starcraft. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[5] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. 2013.

[6] Sylvain Gelly and David Silver. Monte-carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[7] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI magazine*, 26(2):62, 2005.

[8] Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius. Monte Mario: Platforming with MCTS. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO ’14, pages 293–300, New York, NY, USA, 2014. ACM.

[9] Niels Justesen, Tillman Balint, Julian Togelius, and Sebastian Risi. Script-and cluster-based UCT for StarCraft. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–8. IEEE, 2014.

[10] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, J-B Hoock, Arpad Rimmel, F Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The computational intelligence of MoGo revealed in Taiwan’s computer Go tournaments. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(1):73–89, 2009.

[11] Jean Méhat and Tristan Cazenave. A parallel general game player. *KI*, 25(1):43–47, 2011.

Training set (Normalized scores)																						
Controller	**G_1**		**G_2**		**G_3**		**G_4**		**G_5**		**G_6**		**G_7**		**G_8**		**G_9**		**G_10**		**Total points**	**Total wins**
	p	v	p	v	p	v	p	v	p	v	p	v	p	v	p	v	p	v	p	v		
sampleMCTS	**1.00**	5	**1.00**	1	0.89	5	0.88	4	0.83	1	0.69	4	0.67	2	0.88	1	0.44	2	0.24	2	**756**	27
[mixmax]	0.90	4	0.53	3	0.38	5	0.92	4	0.33	0	**1.00**	4	0.33	1	**1.00**	1	0.81	3	0.41	1	727	26
[macroActions]	**0.11**	0	0.26	1	**0.08**	4	**0.00**	1	**0.00**	0	**0.03**	2	**0.00**	0	0.38	0	**0.17**	1	**0.19**	1	448	10
[partialExp]	0.91	5	0.47	1	**1.00**	5	0.62	3	**0.00**	0	0.80	5	0.33	1	**1.00**	2	0.29	2	0.54	2	707	26
[mixmax,macroActions]	**0.00**	4	**0.15**	0	0.54	4	**0.08**	1	**0.00**	0	0.29	2	**0.00**	0	0.50	0	**0.00**	0	**0.11**	0	453	11
[mixmax,partialExp]	0.47	2	0.47	0	0.41	4	0.23	2	0.83	1	0.69	4	0.33	1	0.50	0	0.51	3	0.57	3	615	20
[macroActions,partialExp]	0.84	5	**0.15**	1	0.27	5	0.62	3	**0.00**	0	0.43	2	**1.00**	3	0.25	0	0.23	1	0.68	3	607	23
[macroActions,mixMax,partialExp]	0.27	3	**0.00**	0	0.43	5	0.23	1	**0.00**	0	**0.00**	1	**0.00**	0	0.38	0	0.21	1	**0.00**	0	481	11
[UCT reverse penalty]	**0.15**	1	0.69	3	**0.14**	5	**1.00**	5	**1.00**	2	0.69	5	0.67	2	**1.00**	1	0.56	2	0.62	4	601	**30**
[UCT reverse penalty, macroactions]	0.65	3	**0.13**	0	0.35	5	0.50	4	**0.00**	0	0.31	3	0.33	1	**0.00**	0	**0.09**	1	0.41	1	552	18
[UCT reverse penalty, mixMax]	0.82	5	0.76	3	**0.00**	5	**1.00**	5	**0.00**	0	0.69	3	**0.00**	0	0.75	1	**1.00**	2	**1.00**	5	730	29

Table 3: Results on the CIG2014 training set. Scores are normalized between 0 and 1 - G_1: Aliens , G_2: Boulderdash , G_3: Butterflies , G_4: Chase , G_5: Frogs , G_6: Missile command , G_7: Portals , G_8: Sokoban , G_9: Survive zombies , G_10: Zelda

Validation set (Normalized scores)																						
Controller	**G_1**		**G_2**		**G_3**		**G_4**		**G_5**		**G_6**		**G_7**		**G_8**		**G_9**		**G_10**		**Total points**	**Total wins**
	p	v	p	v	p	v	p	v	p	v	p	v	p	v	p	v	p	v	p	v		
sampleMCTS	**0.00**	0	0.24	0	0.63	2	0.48	2	**1.00**	1	0.62	3	0.60	0	0.54	5	0.82	3	**0.09**	0	7545	16
[mixmax]	**1.00**	1	0.75	0	0.44	0	0.85	4	0.46	0	0.82	2	0.68	0	0.46	5	0.92	4	**0.04**	0	6849	16
[macroActions]	**1.00**	1	**0.03**	0	0.81	0	**0.00**	2	**0.06**	0	0.57	1	**0.08**	0	**0.18**	0	0.30	1	**0.00**	0	2656	5
[partialExp]	**0.00**	0	**0.00**	0	**0.13**	1	**1.00**	3	0.80	0	0.68	3	0.70	0	0.73	5	0.72	3	**0.15**	0	9849	15
[mixmax,macroActions]	**1.00**	1	0.07	0	0.81	0	**0.02**	1	0.29	0	**0.00**	2	**0.00**	0	**0.00**	0	**0.00**	1	**0.02**	0	515	5
[mixmax,partialExp]	**0.00**	0	0.72	0	0.31	0	0.63	3	0.60	0	**0.13**	3	0.20	0	**0.19**	3	0.70	4	**0.04**	0	3172	13
[macroActions,partialExp]	**1.00**	1	0.27	0	0.75	0	0.66	4	**0.00**	0	0.57	2	0.51	0	**0.09**	1	0.56	2	**0.00**	0	2364	10
[macroActions,mixMax,partialExp]	**1.00**	1	0.52	0	0.75	0	0.53	3	0.23	0	0.54	2	**0.15**	0	**0.09**	1	0.32	1	**0.02**	0	1968	8
[UCT reverse penalty]	**1.00**	1	0.62	0	**0.00**	0	0.46	5	0.26	1	0.69	3	**1.00**	1	0.45	3	**1.00**	3	**0.05**	0	6873	17
[UCT reverse penalty, macroactions]	**1.00**	1	0.31	0	**1.00**	1	0.26	4	**0.11**	0	0.68	0	0.73	0	**0.00**	0	0.75	2	**0.01**	0	1460	8
[UCT reverse penalty, mixMax]	**1.00**	1	**1.00**	0	0.69	2	0.52	4	0.23	0	**1.00**	1	0.98	5	**1.00**	4	0.96	1	**1.00**	2	**13358**	**20**

Table 4: Results on the CIG2014 validation set. Scores are normalized between 0 and 1 - G_1: Camel race, G_2: Digdug, G_3: Firestorms, G_4: Infection, G_5: Firecaster, G_6: Overload, G_7: Pacman, G_8: Seaquest, G_9: Whackamole, G_10: Eggomania

[12] Thorbjørn S Nielsen, Gabriella AB Barros, Julian Togelius, and Mark J Nelson. General video game evaluation using relative algorithm performance profiles. In *Applications of Evolutionary Computation*, pages 369–380. Springer, 2015.

[13] Tom Pepels and Mark H. M. Winands. Enhancements for Monte Carlo tree search in Ms Pac-Man. In *CIG*, pages 265–272. IEEE, 2012.

[14] Diego Perez, Jens Dieskau, Martin Hünermund, Sanaz Mostaghim, and Simon M Lucas. Open loop search for general video game playing. 2015.

[15] Diego Perez, Edward J Powley, Daniel Whitehouse, Philipp Rohlfshagen, Spyridon Samothrakis, Peter Cowling, Simon M Lucas, et al. Solving the physical traveling salesman problem: Tree search and macro actions. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(1):31–45, 2014.

[16] Diego Perez, Spyridon. Samothrakis, and Simon Lucas. Knowledge-based fast evolutionary MCTS for general video game playing. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–8, Aug 2014.

[17] Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon Lucas, Adrien Couëtoux, Jeyull Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.

[18] Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul, and Lucas Simon. The general video game AI competition - 2014. `http://www.gvgai.net/`, 2014. [Online; accessed 25-December-2014].

[19] Edward J Powley, Peter I Cowling, and Daniel Whitehouse. Information capture and reuse strategies in Monte Carlo tree search, with applications to games of hidden information. *Artificial Intelligence*, 217:92–116, 2014.

[20] Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling. Monte Carlo tree search with macro-actions and heuristic route planning for the multiobjective physical travelling salesman problem. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[21] Tom Schaul. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[22] Julian Togelius. How to run a successful game-based AI competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 2014.