# Comparing Game Tree Search Techniques for General Video Game Artificial Intelligence

Alastair Rayner - 1507516

***Abstract*—The abstract goes here.**

## I. Introduction

THIS Literature review will cover what questions I will be asking for my dissertation topic as well all the literature I have found that is related to my research questions.

## II. Research Questions

- How does game tree search techniques compare for GVGAI?
- Where does each tree search technique do well in each game?
- What are the strengths and weaknesses of different search techniques and how can they be improved?

## III. Hypothesis

In progress (will deffo change)

Does tree search algorithms perform better than Evolutionary Algorithms? Where does tree search algorithms out perform Evolutionary Algorithms?

## IV. Literature Review

### A. General Artificial Intelligence in games

Games play an important role in the development and bench-marking of Artificial Intelligence (AI). This literature review will cover the existing work around the general video game artificial intelligence competition and the different solutions that are available. There have been a lot of popular games that have been used to benchmark AI, for example games such as chess, or Go. Some of these AI programs have been improved upon until they can defeat the world champion players, such as Deep Blue which defeated Gary Kasparov in 1997 [1], [2], [3].

A long standing goal of AI is to develop algorithms capable of completing various tasks without any need to create domain specific tailoring. Deep blue was developed at IBM during the mid-1990s. It was able to beat the world chess champion by having a massively parallel system that was able to search a very large search space concurrently [1].

The challenge of creating an AI for the game Go is that it has a huge branching factor and it lacks a good evaluation function. There have been more recent breakthroughs within AI, such as AlphaGo [4] which was developed by google deepmind. It was able to beat the a professional Go player in 2015 and in 2017 was able to beat Ke Jie, the world champion player at the time[4]. AlphaGo combined neural networks with MCTS to achieve a 99.8% win rate against other Go programs. The program used a supervised learning policy network that was trained directly from expert human players, then a reinforcement learning policy network was used to improve the supervised learning policy by optimising the final outcome of self played games. Further information on how this works is described in [4]. Monte Carlo Tree Search (MCTS) has had spectacular success in the game Go [5], and is implemented in the top rated go programs.

During the match against Fan Hui, AlphaGo evaluated thousands of times fewer positions than Deep blue did against Kasparov. It did this by selecting those positions more intelligently using the policy network, then evaluating them more precisely using the value network, where as Deep blue used a more brute force approach [4], [1].

Other notable game AI's are IBM's Watson, which won against a human player in the game *Jeopardy!*. This used natural language processing and information retrieval combined with machine-learning. These systems analyse an input question and generate and evaluate candidate answers using a variety of techniques [6].

The reason for these AI's success are often as a result of very domain specific knowledge about the game it is playing and it means they become highly specialized and cannot be ported to other games.

### B. The General Video Game AI Competition

In most modern video games the AI is tailored specifically for that game and can't easily be modified for use in a different game type. However this is what GVG-AI aims to solve, by creating an AI that can play any game.

There have been quite a few AI competitions before in video games, such as Unreal Tournament [7], Super Mario Bros [**?**], Starcraft [8]. However most of the winning AI strategies used in those games are very domain specific and it is often more about knowing the game than developing good general AI [9].

Another competition that was similar to GVG-AI was the General Game Playing (GGP) competition [10]. However almost all of the games in the GGP are board games, and the Game Description Language (GDL) used is not designed for video games.

The GVG-AI Competition is a competition framework that proposes the challenge of creating controllers for general video game playing. The controllers must be able to play a wide variety of video games, many of them will be completetly unknown to the controller. This means the controller must have some general AI to discover the mechanics and goal of the game, so it can increase it's score and win the game. [11], [9]

The framework contains a library of 2D Java based video games some of which are based of classic arcade games, there are currently as of writing this, 62 games that AI controllers can be tested on.

The Arcade Learning Environment [12] provides an interface to hundreds of Atari 2600 game environments. ALE is similar to GVG-AI in a couple of ways; firstly it provides a testbed for benchmarking AI techniques within video games and secondly it is focused around creating agents within video games, as opposed to the GGP competition [10]. The Atari 2600 is a video game console developed in 1977 , it has had over 500 original games released for the console, and nearly all popular arcade games at the time were pored to the console such as; *PAC-MAN* and *SPACE INVADERS* [12]. This provides a large test-bed for AI agents. The hardware for the Atari 2600 is very limited compared to today's standards, it had a 1.19Mhz CPU and 128 Bytes of RAM. These hardware specs limit the complexity of the games that can be played on it, which strikes a balance of challenging but allowing search algorithms to have a small enough search space as to not have a large horizon effect [12].

The growing interest in competitions such as the ones mentioned above clearly reflects a desire for general competency for AI within games.

*1) Challenges and Goals of GVGAI:* The goal of GVGAI is to create a generally intelligent agent that is able to win any game it is placed in, when it doesn't know the game. During the tournament a completely new set of games are used, to avoid the agents becoming too deomain specific. Another challenge is the time limit that an agent can choose an action, this avoids the agent spending too long deciding a task and not making an action. [13]

*2) Competition & Rules:* The winning conditions are decided by three factors:

- Number of games finished with a victory
- Total sum of points
- Total time spent

The fist objective to be considered is the number of victories, however in case of a tie, the next objective is the number of points. Then if those two are a tie then the final decider is the total time spent before the win [9].

In the competition the agent will play 10 unknown games and 5 levels per game. Furthermore each level is played ten times, so each agent plays roughly 500 total games in the tournament [13].

*3) The GVGAI Framework:* The Framework is developed in the Java Environment The controllers are allowed upto 40ms to compute the agents action(s) [14], [11].

The GVGAI framework comes with quite a few sample agents;

These sample agents provide useful insights into how new agents can be created for the competition by applying common AI techniques. The *HUMAN* player and the *REPLAYER* can be used for debugging the game and to help the programmer get a better understanding of how the game can be played.

The framework uses a Video Game Description Language (VGDL) to describe a wide variety of video games. The VGDL is based on a python version developed by Schaul (2014) called PyVGDL [13]. Furthermore in the GVG-AI Competition the AI agent does not have access to the whole games description, where as in GGP the agent was able to see the whole game description. This means that the agent has to analyze and simulate the game in order to figure out the rules and goal of the game.

The framework has an StateObservation object that has an interaction set that consists of *up, down, left, right* and *use*.

**Sample MCTS** The GVG-AI framework proves a sample MCTS controller, this controller has received considerable interest due to it's success in the competition. The sample MCTS controller is an implementation of the vanilla MCTS algorithm, this is described in section IV-C1. The full description of MCTS algorithm is described by Browne et al. [5]. The sample MCTS controller uses a payout depth of 10 moves and an exploration-exploitation constant value of $\sqrt{2}$ and selects the most visited action from the root to pick a move to return for each game cycle [9].

**sample GA** The sample Genetic Algorithm(GA) controller is a rolling horizon open loop implementation for a minimalistic steady state genetic algorithm, known as microbial GA [15]. A tournament takes place between two players and the loser of the tournament is mutated randomly, with the probability of 1/7. Then certain parts of it's genome are recombined with parts from the winners genome, with the probability of 0.1 [9]. This repeats until the time budget has been used. The evaluation function is the same as the sampleMCTS controller. The sample GA controller came 12th in the competition [9].

**Random** This is a very simple controller that is provided with the GVG-AI framework, it simply returns a random action at each game cycle. The random controller came 12th in the competition, which is quite surprising as it managed to beat quite a few of the other, rather complicated controllers.

**OSLA**

### C. Game Search Techniques

*1) Monte-Carlo Tree Search (MCTS) :* Monte Carlo Tree Search is a method for finding the optimal decisions in a specified domain by taking random samples in the search space and building a search tree according the the results [5].

MCTS has been demonstrated to work effectively with classic board games, modern board-games and video games [16].

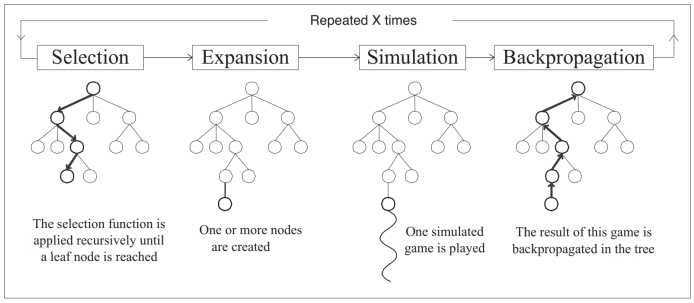


Fig. 1. Overview of Monte-Carlo Tree Search. Image sourced from [16].

The basis of MCTS is the simulation of games where both the player controller and the opponent play pseudo-random moves. From playing a single game consisting of random moves, very little can be learnt about the game. However when simulating a multitude of random games, a good strategy can be inferred. This is what MCTS does, it builds a tree of possible future game states. This can described in four stages, as shown in figure 2.

**Selection** Starting at the root node, a selection policy is recursively applied to descend the tree until the most urgent node is reached. While the state is found in the tree, the next action is chosen in a way that balances between exploitation and exploration. For most of the tasks, the option is to choose exploitation, which leads to the best results so far. However there may still be actions that have not been explored that could lead to good results, thus exploration is used to try and find any promising actions [16].

A common enhancement for selection is Upper Confidence Bounds for Trees (UCT), which is used by the sample MCTS controller, as described in section IV-B3. The main idea of UCT is to use information gathered during previous iterations of MCTS to decide whether to explore a new child or exploit a promising child. Then the child with the highest UCT value is selected.

**Expansion** When the selection strategy returns a node, there are a few different ways to expand the node. This step is similar to the selection step, however this step has no information available from previous iterations of MCTS for a specific child [13]. One of the major tasks for an expansion strategy is to choose the children that should be added to the search tree, whether that child is chosen by an expansion policy or an exploitation policy. Typically only one child is added to the tree by many MCTS based agents after selection and before the simulation [16], [13]. However in the context of the GVG-AI competition where the number of possible actions is limited to a maximum of seven (i.e. right, use etc.) using the all children expansion strategy, where all actions provided by the forward model are used to expand the search tree. Thus, the time necessary for expansion is less than in other domains, where the number of actions is much higher(i.e. chess).

**Simulation** For the rest of the game, actions are selected at random until the end of the game. This means that the weighting of action selection probabilities has a significant effect on the level of play. For example if the game played with equal probability between exploration and exploitation, this will often lead to sub-optimal play [16]. To look for more promising plays, MCTS can use heuristic knowledge to give weights to actions that look more promising.

**Backpropagation** When the simulated game has played out, the tree is updated and each node in the tree that was visited is modified with the win loss ratio of that game. This informs future tree policy decisions.

These steps are then repeated until some predefined computational budget is reached, which most commonly are; time, memory or iteration constraint [5]. At which point the process is halted and the best performing root action is returned. This is the action that the agent will take in the game.

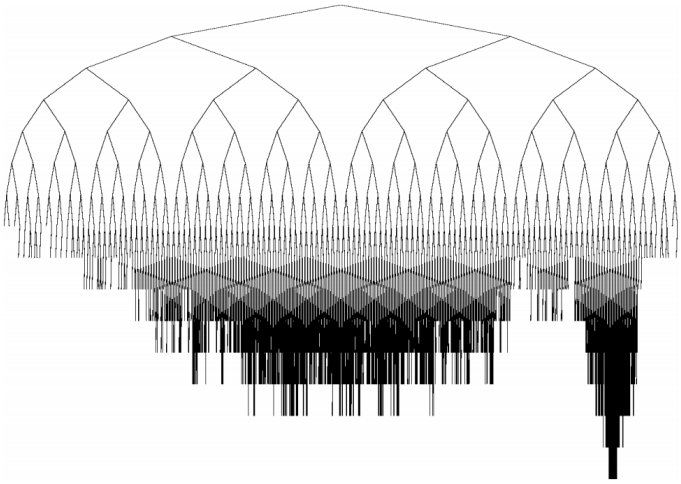

Fig. 2. Overview of Monte-Carlo Tree Search Asymmetry. Image sourced from [17].

*2) MCTS variations:* MCTS is one of the most promising baseline approaches in the literature. There has been a good deal of research on MCTS variants, each providing better results according to different domains [5], [18], [19], [20], [21], [22].

While MCTS has been extensively applied to zero-sum games, with two players that alternate terns in discrete action spaces, it has also been aplied to orther domain types, such as single and multiplayer games, RTS and games with lots of uncertainty [5], [21], [22].

**Bandit Based Methods** Bandit based problems are popular class of sequential decision problems, in which one needs to choose among a set of actions (e.g. the arms of a multiarmed bandit slot machine) in order to maximize the cumulative reward by consistently taking the optimal action [23].

The difficulty arises because the underlying reward distributions are unknown, and potential rewards must be estimated based on past observations.

**Flat UCB** Coquelin and Munos propose a technique called Flat UCB, which treats the leaves of the search tree as a single multiarmed bandit problem, as described below **??** .

*3) Evolutionary Algorithms:* Alpha beta pruning

minimax

*4) Breadth First Search (BFS) :* This paper proposes an efficient implementation of BFS for GVGP [24] uses a hash function.

*5) Depth First Search:* Depth First search with alpha-beta pruning [25] has achived superhuman performace in chess, checkers and orthello. However it has not been effective in Go [4].

OLETS

Evolutionary Algorithms (RHEA)

## V. Methodologies

## VI. Conclusion

The conclusion goes here.

## REFERENCES


[1] M. Campbell, A. J. Hoane, Jr., and F.-h. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, no. 1-2, pp. 57–83, Jan. 2002. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(01)00129-1

[2] C. E. Shannon, "Programming a computer for playing chess," in *Computer chess compendium*. Springer, 1988, pp. 2–13.

[3] F.-h. Hsu, M. S. Campbell, and A. J. Hoane, Jr., "Deep blue system overview," in *Proceedings of the 9th International Conference on Supercomputing*, ser. ICS '95. New York, NY, USA: ACM, 1995, pp. 240–244. [Online]. Available: http://doi.acm.org.ezproxy.falmouth.ac.uk/10.1145/224538.224567

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[6] D. Ferrucci, A. Levas, S. Bagchi, D. Gondek, and E. T. Mueller, "Watson: beyond jeopardy!" *Artificial Intelligence*, vol. 199, pp. 93–105, 2013.

[7] P. Hingston, "A new design for a turing test for bots," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 345–350.

[8] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 5, no. 4, pp. 293–311, 2013.

[9] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[10] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the aaai competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005.

[11] D. Perez, "The general video game ai competition," http://http://www.gvgai.net/, 2017.

[12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents." *J. Artif. Intell. Res.(JAIR)*, vol. 47, pp. 253–279, 2013.

[13] T. Schuster, "Mcts based agent for general video games," Ph.D. dissertation, Masters thesis, Department of Knowledge Engineering, Maastricht University, Maastricht, the Netherlands, 2015.

[14] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General video game ai: Competition, challenges and opportunities," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[15] I. Harvey, "The microbial genetic algorithm," in *European Conference on Artificial Life*. Springer, 2009, pp. 126–133.

[16] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.

[17] P.-A. Coquelin and R. Munos, "Bandit algorithms for tree search," *arXiv preprint cs/0703062*, 2007.

[18] H. Park and K.-J. Kim, "Mcts with influence map for general video game playing," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 534–535.

[19] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary mcts for general video game playing," in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE, 2014, pp. 1–8.

[20] E. İlhan and A. Ş. Etaner-Uyar, "Monte carlo tree search with temporal-difference learning for general video game playing," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 317–324.

[21] M. de Waard, D. M. Roijers, and S. C. Bakkes, "Monte carlo tree search with options for general video game playing," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[22] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, "Investigating mcts modifications in general video game playing," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 107–113.

[23] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*, vol. 6. Springer, 2006, pp. 282–293.

[24] S. Ito, Z. Guo, C. Y. Chu, T. Harada, and R. Thawonmas, "Efficient implementation of breadth first search for general video game playing," in *Consumer Electronics, 2016 IEEE 5th Global Conference on*. IEEE, 2016, pp. 1–2.

[25] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial intelligence*, vol. 6, no. 4, pp. 293–326, 1975.


## APPENDIX A
## FIRST APPENDIX

Appendices are optional. Delete or comment out this part if you do not need them.