Comparing Game Tree Search Techniques for General Video Game Artificial Intelligence

Alastair Rayner
Falmouth Games Academy
Falmouth, UK
Student ID: 1507516
Email: AR185160@falmouth.ac.uk

***Abstract*—Video games have been used for benchmarking Artificial intelligence techniques, however in many cases the AI use very domain specific knowledge to improve their results. The General Video Game AI (GVG-AI) competition aims to address the issue of creating an Artificial General Intelligence (AGI) which in this context means to create an AI that is able to complete video games albeit not up to a human skill level. The games within the GVG-AI competition are all arcade style game, such as *Pac-Man* and *Zelda*. This paper describes the goals of GVG-AI as well as its rules and challenges, it also covers the different types of tree search techniques used in General Game Playing (GGP) as well as other game search techniques for AGI.**

Finish Abstract

I. Introduction

THIS paper will compare the tree search techniques used in the General Video Game Artificial Intelligence (GVG-AI) competition. 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 GVG-AI competition and the different solutions that are available. The GVG-AI competition is a recurring video game playing competition designed to simulate and benchmark general artificial intelligence. For an AI agent to be tested, they are submitted to the competition site, then they are tested against previously unseen arcade-style video games like the ones shown in figure 4. The games that the agents are tested against change every year to stop them from becoming too domain specific.

There is a large amount of tree search techniques used within the GVG-AI competition, many of them are based around Monte Carlo Tree Search (MCTS).This project will compare the success rates of each of the different methods, as well as an in depth comparison as to where each tree search technique succeeds in a set of existing games.

II. Research Questions

There has been considerable literature on GVG-AI and individual AI techniques within the competition, and very little covering an in depth look into the comparison of the different search algorithms.

Furthermore within the GVG-AI competition there are a few different tree search techniques used, a few of which are mentioned in section III-B. These techniques perform quite differently within the competition, this paper will aim to answer *What are the strengths and weaknesses of different search techniques* as well as *What type of game does each search technique succeed at?*

The research gathered from this could lead to a Hyper-Heuristic [1] that can be implemented into a controller for the competition. Furthermore this paper will outline the challenges of using the different search techniques, such as issues of games that have a large search space, or multiple objectives.

III. Literature Review

A. Artificial General Intelligence in games

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 Garry Kasparov in 1997 [3], [4], [5].

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 [3].

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 (these terms are described in section III-B1). There have been more recent breakthroughs within AI, such as AlphaGo [6] 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[6]. 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 [6]. Monte Carlo Tree Search (MCTS) has had spectacular success in the game Go [7], 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

[1] A Hyper-Heuristic contains a portfolio of algorithms and is able to select the most apropriate algorithm to use depending on the game state [1], [2].

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 [6], [3].

The reason for these AI's success are often as a result of very domain specific knowledge about the game it is playing and means they become highly specialized and cannot be easily ported to other games. This is what started the General Game Playing (GGP) competition (described in section III-D) which was to create general AI for board games, this is similar to what happened with GVG-AI where there was lots of specialized AI for games, which were all very domain specficic, such as Starcraft AI [8], [9]. This lead to the need for general AI for games, hence GVG-AI and Arcade Learning Environment which is described in section III-D.

B. Game Search Techniques

1) Common Terms Used in Tree Search Algorithms:
Branching Factor Branching factor is the number of children at each node of a tree.

Horizon Effect The horizon effect is the problem where only a small portion of the search tree can be searched, i.e. the AI can search 4 moves ahead, but the 5th move could be a detrimental move, but the AI doesn't know that because of its "*horizon*".

Evaluation Function Evaluation function is used to estimate the value of a position.

2) 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 [7].

MCTS is a class of decision tree search algorithms discovered independently by several authors [10], [11], [12].

MCTS has been demonstrated to work effectively with classic board games, modern board-games and video games [13], [14].

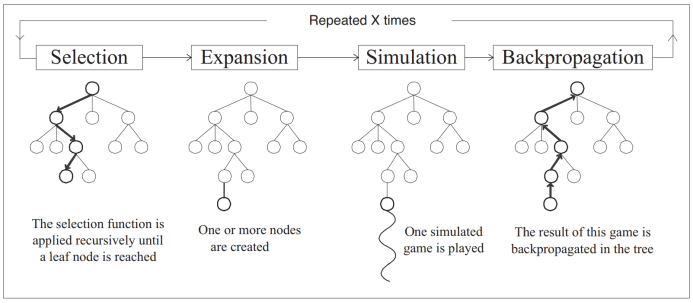

Fig. 1. Overview of Monte Carlo Tree Search. Image sourced from [13].

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 1.

Selection Starting at the root node, a selection policy is recursively applied to descend the tree until the most urgent [2] node is reached. 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 [13].

Upper Confidence Bound (UCB) is often a common enhancement for MCTS and is often referred to as Upper Confidence Bounds for Trees (UCT) [15], which is used by the sample MCTS controller, as described in section III-D3.

UCB is based on the multi-armed bandit problem, in which it selects the optimal arm to pull in order to maximize rewards [11], [7]. The main idea of UCT is to use information gathered during previous iterations of MCTS to decide what the best child node is at each level when traversing the tree. Then the child with the highest UCT value is selected.

$$UCT = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{1}$$

The UCT value is calculated by $\overline{X}_j$ being the average score of child j, and 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 the constant value which adjusts the contribution of the second term. The second term $\sqrt{\frac{2\ln n}{n_j}}$ increases each time the parent node has been visited, but a different child was chosen.

Expansion

The major task for an expansion strategy is to choose which children should be added to the search tree.

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 [16]. Typically only one child is added to the tree by many MCTS based agents after selection and before the simulation [13], [16].

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 [13]. 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.

The expansion and simulation stages are commonly collectively referred to as the *playout* [17]. These steps are then repeated until some predefined computational budget is

[2]Most urgent most commonly means the node with the highest UCT value.

reached, which most commonly are; time, memory or iteration constraint [7]. 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.

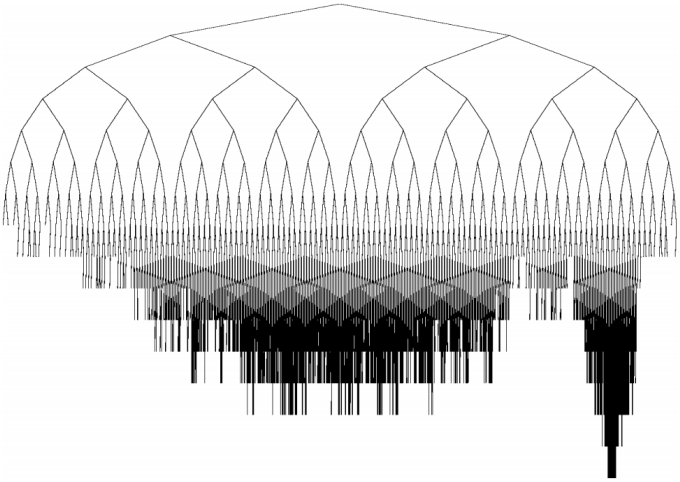

Fig. 2. An Exmple of an Asymmetric Tree Created by MCTS. Image sourced from [18].

3) 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 [7], [19], [20], [21], [22], [23].

While MCTS has been extensively applied to zero-sum games, with two players that alternate terns in discrete action spaces, it has also been applied to other domain types, such as single and multiplayer games, RTS and games with lots of uncertainty [7], [22], [23].

4) Evolutionary Algorithms: Evolutionary Algorithms are largely inspired from the biological sciences. They encode solutions to problems as individuals, part of a population which evolves over generations, until a solution is found or execution limit is reached.Figure 3 shows the stages of how Evolutionary Algorithms work. Rolling Horizon Evolutionary Algorithms are one of the options available to evolve sequences of actions for planning in GVGP.

Perez et al. in [24] compared MCTS with RHEA on the game Physical Traveling Salesman Problem (PTSP). The game is a modification of the popular optimization problem, the Traveling Salesman Problem [25], [26] where the player must visit a series of way points in a 2 dimensional level and the agent has up to 40ms to execute an action [24], which is similar to the GVG-AI competition. The results from that paper show that RHEA is a promising competitor to MCTS. Their approach is used in the same manner as MCTS uses roll-outs and the generative model (i.e. a simulator). Thus an agent will evolve a plan in an imaginary model for some milliseconds, then evolves a new plan repeatedly in a simulation manner, until the game is over. The term Rolling Horizon comes from the fact that the planning has a certain depth that it can search within a game space (i.e. the horizon) [27], [28].

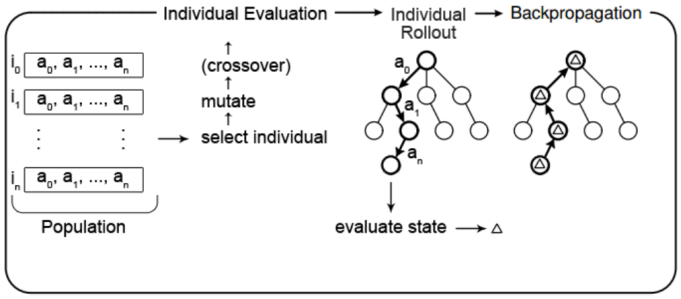

Fig. 3. RHEA statistical tree steps. Image sourced from [28].

Recent literature on AGI have been combining evolution and tree search in interesting ways in order to combine the benefits of both methods, such as Licas et al. [29] applied an EA process to guide the simulation step of MCTS and improve the random default policy. Their results show a significant increase in performance in the game *Space Invaders* and the Mountain Car Problem.

5) Minimax: **Minimax**

Alpha beta pruning Alpha Beta pruning tries to minimax the search space [30] **Depth First Search** Depth First search with alpha-beta pruning [30] has achieved superhuman performance in chess, checkers and orthello. However it has not been effective in the game Go [6].

6) Breadth First Search (BFS) : This paper proposes an efficient implementation of BFS for GVGP [31] uses a hash function.

C. Hyper-Heuristics

During the 2015 competition, the three winning controllers had one thing in common, they were all a combination of techniques that were selected depending on certain characteristics observed in the game state [2].

D. 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 [9], Super Mario Bros [32], 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 [33].

Another competition that was similar to GVG-AI was the General Game Playing (GGP) competition [34], [35]. 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 completely 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 its score and win the game. [36], [33]

The framework contains a library of 2D video games some of which are based of classic arcade games, there are currently as of writing this, 82 different singleplayer games and 20 multiplayer games that AI controllers can be tested on.

The Arcade Learning Environment [37] 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 [34]. 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* [37]. 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 [37]. ALE has been used deep reinforcement learning techniques, that have been able to reach human level of play [38], [28].

Fig. 4. Example of games in GVG-AI Framework: angelsdemons (Top Left), boulderdash (Top Right), frogs (Bottom Right), Zelda (Bottom Left).

Games in the GVG-AI competition are written using the Video Game Description Language [39], which is a high level description language designed to be able to create a wide variety of arcade style games, where the rules take the form of sprite movement and interaction on a 2D grid [40]. VGDL in GVG-AI is a JAVA port from pyVGDL which was programmed in python. The VGDL is a powerful tool for conducting research on computational intelligence and games [39], [35].

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. [16].

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 [33]. 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 [16]. The competition does now allow multithreading that is used by some MCTS enhancements, some MCTS enhancements that are used are discussed in section III-B3. The controllers can also use up to 1 second of CPU time for initialization and 40ms to compute an action each game tick. However if the controller takes between 40ms and 50ms, the action return will be *NIL*(Where no movement will be applied), anymore than 50ms will result in the controller automatically loosing [2], [33].

3) The GVGAI Framework & Sample Controllers: The Framework is developed in the Java Environment and has a few differnet tracks that you can submit AI for, these are; Single Player Planning Track, 2-Player Planning Track and Level Generation Track [41]. The controllers are allowed upto 40ms to compute the agents action(s) [42], [36].

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 [16]. 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, nil, escape* and *use*.

The GVGAI framework comes with quite a few sample agents;

Sample MCTS The GVG-AI framework proves a sample MCTS controller, this controller has received considerable interest due to its success in the competition. The sample MCTS controller is an implementation of the vanilla MCTS algorithm, this is described in section III-B2, and the full description of MCTS algorithm is described by Browne et al. [7]. The sample MCTS controller uses a payout depth of 10 moves and an exploration-exploitation constant value of $\sqrt{2}$ (from equation 1) and selects the most visited action

from the root to pick a move to return for each game cycle [33]. However MCTS isn't the sole answer to the GVG-AI competition, as it has been shown that even with a 30x computational budget, it fails to master games. However it does manage to avoid to explicitly loosing games, but does not win a lot of them either. This shows that for the AI it finds not losing is significantly easier than winning [40].

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 [43]. 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 its genome are recombined with parts from the winners genome, with the probability of 0.1 [33]. 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 [33].

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 14th in the competition, which is quite surprising as it managed to beat quite a few of the other, more complicated controllers. [3]

OSLA One Step Look Ahead is another rather simple controller, it evaluates the position using the same heuristic as Sample MCTS and selects the action with the highest evaluation score, then moves the model one step ahead. This controller came in 16th place.

IV. Research Methodologies

Research Methodology go here. **Hypothesis** In progress..

V. Preliminary results

Preliminary results go here.

VI. Conclusion

The conclusion goes here.

References

[1] A. Mendes, J. Togelius, and A. Nealen, "Hyper-heuristic general video game playing," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[2] H. Horn, V. Volz, D. Pérez-Liébana, and M. Preuss, "Mcts/ea hybrid gvgai players and game difficulty estimation," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[3] M. Campbell, A. J. Hoane, Jr., and F.-h. Hsu, "Deep blue," *Artif. Intell.*, vol. 134, no. 1-2, pp. 57–83, Jan. 2002.

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

[5] 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.

[6] 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.

[7] 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.

[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] 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.

[10] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

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

[12] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. Van Den Herik, "Monte-carlo strategies for computer go," in *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, 2006, pp. 83–91.

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

[14] T. Pepels, M. H. Winands, and M. Lanctot, "Real-time monte carlo tree search in ms pac-man," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 6, no. 3, pp. 245–257, 2014.

[15] I. Bravi, "Evolving uct alternatives for general video game playing," 2017.

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

[17] E. J. Powley, P. I. Cowling, and D. Whitehouse, "Information capture and reuse strategies in monte carlo tree search, with applications to games of hidden information," *Artificial Intelligence*, vol. 217, pp. 92–116, 2014.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "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*. ACM, 2013, pp. 351–358.

[25] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the physical traveling salesman problem: Tree search and macro actions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 31–45, 2014.

[26] M. M. Flood, "The traveling-salesman problem," *Operations Research*, vol. 4, no. 1, pp. 61–75, 1956.

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

[28] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Rolling horizon evolution enhancements in general video game playing," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 88–95.

[29] S. M. Lucas, S. Samothrakis, and D. Perez, "Fast evolutionary adaptation for monte carlo tree search," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2014, pp. 349–360.

[3] The reason for this may be due to more complicated controllers not choosing an action because it isn't able to search the game space, so making a random action is better than no action.

[30] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial intelligence*, vol. 6, no. 4, pp. 293–326, 1975.
[31] 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.
[32] N. Shaker, J. Togelius, G. N. Yannakakis, L. Poovanna, V. S. Ethiraj, S. J. Johansson, R. G. Reynolds, L. K. Heether, T. Schumann, and M. Gallagher, "The turing test track of the 2012 mario ai championship: entries and evaluation," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.
[33] 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.
[34] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the aaai competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005.
[35] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," 2008.
[36] D. Perez, "The general video game ai competition," http://http://www.gvgai.net/, 2017.
[37] 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.
[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[39] T. Schaul, "An extensible description language for video games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 325–331, 2014.
[40] M. J. Nelson, "Investigating vanilla mcts scaling on the gvg-ai game corpus," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–7.
[41] R. D. Gaina, D. Pérez-Liébana, and S. M. Lucas, "General video game for 2 players: framework and competition," in *Computer Science and Electronic Engineering (CEEC), 2016 8th*. IEEE, 2016, pp. 186–191.
[42] 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.
[43] I. Harvey, "The microbial genetic algorithm," in *European Conference on Artificial Life*. Springer, 2009, pp. 126–133.

APPENDIX A
FIRST APPENDIX

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