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). This paper analyses 3 popular tree searching algorithms and how they scale with changing level sizes within the GVG-AI competition. The algorithms are; Breadth First Search (BrFS), Best First Search (BeFS) and Monte Carlo Tree Search (MCTS) and finds that BrFS and BeFS have a higher win percentage on deterministic games, whereas MCTS outshines both BrFS and BeFS on stochastic games.**

Fig. 1. Example of games in GVG-AI Framework: angelsdemons (Top Left), boulderdash (Top Right), frogs (Bottom Right), Zelda (Bottom Left).

I. Introduction

THIS paper compares tree search techniques used within 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 paper will cover the existing work around the GVG-AI competition and compare tree search algorithms that are being used in the competition. 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 1. 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). However there is little research comparing how well basic tree searching techniques compare, this project compares how well basic tree searching algorithms such as BrFS and BeFS perform in comparison to MCTS and it's variants, as mentioned in section III-B3.

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 *How does MCTS compare to more straightforward algorithms?* as well as *How does changing level size effect the performance of each of these algorithms?*

This paper outlines the challenges faced by different tree search algorithms, it also shows that MCTS performs better in stochastic games, compared to BrFS and BeFS. Whereas BeFS and BrFS perform better in deterministic games.

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 [1], [2], [3].

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 (these terms are described in section III-B1). 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].

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 [6], [7]. This lead to the need for general AI for games, hence GVG-AI and Arcade Learning Environment which is described in section III-D.

A long standing goal of AI is to develop algorithms capable of completing various tasks without any need to create domain-specific tailoring.

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. This is a large factor as to what algorithms can be used to play certain types of games, for example; the GVG-AI competition has a maximum branching factor of 7, where as the game Go has a maximum branching factor of 250 [4], [8].

Horizon Effect: This 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 its search "*horizon*" is limited [9].

Evaluation Function: This is used to estimate the value of a position or the current game state, i.e. is the agent winning or loosing the game.

Open Loop & Closed Loop: In the closed loop MCTS the algorithm assumes it is stable to store game sates on the nodes of the tree when expansion is performed, which means that selection can navigate the tree without having to calculate new states. In Open Loop MCTS (OLMCTS) the algorithm only stores the statistics on the tree nodes and then generates the next states using the forward model [10], [11].

In the competition all the games are either deterministic or stochastic, **Stochastic** In stochastic games there is some inherent randomness, meaning any values found by a tree search using the forward model (as explained in section III-D3, may change in following ticks of the game. So an agent that executes exactly the same actions each game will lead to different outcomes.

Deterministic In deterministic games each action is predictable, and any object found using the forward model will not change in forthcoming game ticks.

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

MCTS is a class of decision tree search algorithms discovered independently by several authors [12], [13], [14].

MCTS has been demonstrated to work effectively with classic board games, modern board-games and video games [15], [16].

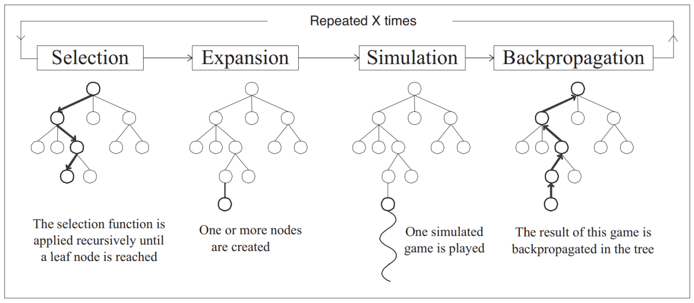

Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Fig. 2. Overview of Monte Carlo Tree Search. Image sourced from [15].

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 with the highest UCT value) 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 [15].

Upper Confidence Bound (UCB) is often a common enhancement for MCTS and is often referred to as Upper Confidence Bounds for Trees (UCT) [18], which is used by the sample MCTS controller, as described in section III-D3. UCT uses UCB1 to build a potentially asymmetric tree that

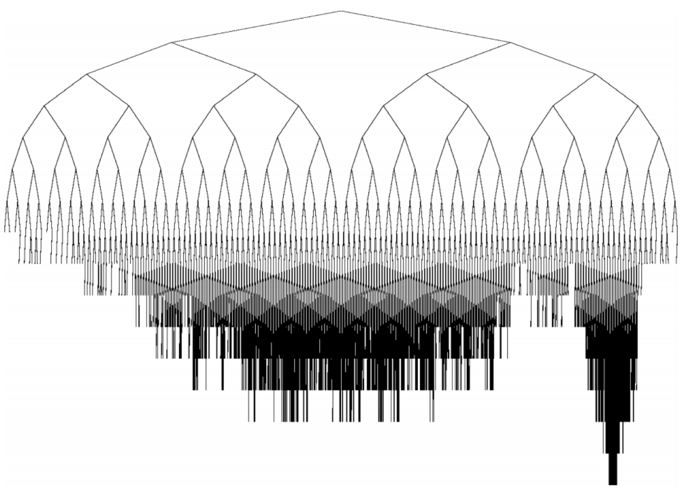


Fig. 3. An Example of an Asymmetric Tree Created by MCTS. Image sourced from [17].

grows towards the more promising parts of the search space as shown in figure 3.

UCB is based on the multi-armed bandit problem, in which it selects the optimal arm to pull in order to maximize rewards [13], [5], [19]. 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 [20]. Typically only one child is added to the tree by many MCTS based agents [15], [20].

Simulation

A Monte Carlo simulation is carried out until a pre-determined depth or the end of a game. 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 [15]. To look for more promising plays, MCTS can use heuristic knowledge to give weights to actions that look more promising [21]. Silver et al. [22] describes the technique of simulation balancing using a gradient decent to bias the policy during the simulation, this aims to provide a more accurate spread of simulation outcomes.

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* [23]. 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.

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 [5], [24], [25], [26], [27], [28], [29].

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 [5], [27], [28].

4) Evolutionary Algorithms: Evolutionary Algorithms (EA) 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 4 shows the stages of how Evolutionary Algorithms work [30]. Rolling Horizon Evolutionary Algorithms are one of the options available to evolve sequences of actions for planning in GVGP [31]. Furthermore within the GVG-AI competition the sampleGA controller, as described in section III-D3 uses a rolling horizon open loop algorithm [8].

Perez et al. in [31] 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 [21], [32] 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 [31], 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 [31]. 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) [33], [30].

A Typical genetic algorithm is composed of several stages; First the algorithm creates a population of random individuals, then it will iteratively go though the sample and calculate the fitness (i.e. the evaluation function as described in III-B1) of every individual in the population. The most fit individuals are then used to form a new generation, and each individual

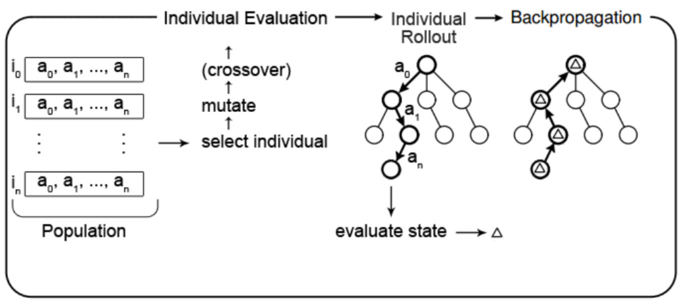


Fig. 4. RHEA statistical tree steps. Image sourced from [30].

is modified with a combination of crossover and mutation. Crossover is representative of biological reproduction where a child is produced based of the parents properties. Mutation is where a random value is changed to explore the search space and is analogous to biological mutation [34], [35], [31], [30], [33]. This process can be shown in figure 4.

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 Lucas et al. [36] 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 attempts to minimize the opponent's maximum reward at each state. The tree search is often stopped prematurely and a value function is used to estimate the outcome of the game, then the Alpha-Beta heuristic is used to prune the search tree [5], [37].

6) Depth First Search: Depth First Search (DFS) is a tree search algorithm that iteratively expands each unvisited child node of the tree until a non-expandable node is reached. The search then navigates back up the tree until it finds the next unvisited node to expand and continues the search until all the nodes have been visited [21]. Depth first search with alpha-beta pruning [37] has achieved superhuman performance in chess, checkers and orthello, however it has not been effective in the game Go [4].

7) Breadth First Search: Breadth First Search (BrFS) searches the tree one level at a time, it starts at the root node and searches neighbouring nodes first, before searching their child nodes. Within the gvg-ai competition, there have been several variations of this, one of them which uses a hash function to improve the performance of the algorithm [38], which is similar to the BrFS Extended controller used in this paper.

8) Best First Search: Best First Search (BeFS) explores a tree by selecting the most promising node and expanding it until a specified limit is reached, or a solution is found. Breadth First Search is used by YOLOBOT, which was developed by T. Joppen et al. [9] and arguably the most successful controller in the competition as of writing this paper, their approach uses Best First Search for deterministic games and MCTS for stochastic games [39].

C. Hyper-Heuristics

A hyper-heuristic approach typically combines several AI methods and automate them to be able to choose the best solution for that situation, a term used to describe them is '*heuristics to choose heuristics*' [40], [41], [42]. During the 2015 competition, the three winning controllers all were a type of hyper-heuristic [43].

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 [7], Super Mario Bros [44], Starcraft [6]. 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 [8].

Another competition that was similar to GVG-AI was the General Game Playing (GGP) competition [45], [46]. 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. [47], [8]

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 [47], [48].

Games in the GVG-AI competition are written using the Video Game Description Language [49], 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 [50]. 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 [49], [46].

The Arcade Learning Environment [51], [52] 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 test bed for benchmarking AI techniques within video games and secondly it is focused around creating agents within video games, as opposed to the GGP competition [45]. 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* [51]. 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 specifications 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 [51]. ALE has been used deep reinforcement learning techniques, that have been able to reach human level of play [53], [30].

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 GVG-AI: The goal of GVG-AI 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 domain-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 [20]. Furthermore, this also is what defines the difference between GVG-AI and GGP, because having such a short amount of time to compute an action is what makes the competition such a challenge, otherwise the agent will be able to brute force the search space to find the best possible solution, which may take minutes or hours to compute each game tick.

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 [8]. 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 [20]. 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 [43], [8].

3) The GVG-AI 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 [54]. The controllers are allowed upto 40ms to compute the agents action(s) [55], [47].

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 [20]. 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 game uses the forward model, which allows the game to copy the current state of the game and advance it by executing a specified action.

A lot of competition winning controllers use a variation of tree search algorithms, i.e. Maast and number27 [20] both use breadth first search for deterministic games, this is due to the game states not changing during the game so extended tree searches work well [39].

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

The GVG-AI 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. [5]. 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 [8]. 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 [50].

sample OLMCTS The Open Loop Monte Carlo Tree Search uses the same techniques as the sampleMCTS, however it does not store the game state within the nodes of the tree search. This is because with non-deterministic games, that contain random elements (i.e. random enemies), the states in the tree will change every frame, so the outcome of the random action will be different on every search.

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 [56], [8]. 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 [8]. 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 [8].

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

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 using the forward model. This controller came in 16th place [8].

IV. Methodology

The simulations were ran on the 2016 variation of the competition framework, which can be downloaded from the official GitHub repository [1].

The framework was modified to be able to collect the data from the different tree search controllers.

A. Hypothesis

Hypothesis One: MCTS has a higher win rate compared to BrFS and BeFS as map size increases in deterministic games

Hypothesis Two: MCTS has a higher win rate compared to BrFS and BeFS as map size increases in stochastic games

Hypothesis Three: BrFS has a higher win rate compared to MCTS and BeFS as map size increases in deterministic games

Hypothesis Four: BrFS has a higher win rate compared to MCTS and BeFS as map size increases in stochastic games

Hypothesis Five: BeFS has a higher win rate compared to MCTS and BrFS as map size increases in deterministic games

Hypothesis Six: BeFS has a higher win rate compared to MCTS and BrFS as map size increases in stochastic games

B. Tools Used

For carrying out the research on the gvg-AI competition, IntelliJ IDEA 2017.3 was used, RStudio Version 1.1.4 was used for compiling the graphs, and the simulations were compiled and executed with java jdk 10, on ubuntu 16.04 LTS. Pycharm 2018.1.2 with python 3.6 was used to create the simulation launcher and json to csv converter.

The computer used to carry out the simulations had Intel(R) Xeon(R) CPU E5-2650 24core @ 2.20GHz - 128GB RAM, 1.1TB SSD.

C. Choosing the Games

There are a lot of games within the GVG-AI framework, and most have multiple levels, so testing with all the different possible configurations is prohibitively expensive, thus a sub-set of games within the framework will be selected in a way that best represent the framework as a whole.

There are papers by previous authors that looked into finding a good selection of games that can be used for benchmarking the framework. [57]

In Nelsons paper [50], he presented a large scale analysis of MCTS with 62 games in the framework, which were sorted based on performance. Bontrager et al. [58] used clustering techniques on 49 games in the goal to obtain rough groups of similar games. The final 20 games that are used in this paper are based of the paper by Gaina et al. [57] where they uniformly sampled from both Nelson and Bontrager et al. papers and balanced a set of 10 stochastic and 10 deterministic games, which can be shown in table I.

Deterministic Games	Stochastic Games
Bait	Aliens
Chase	Chopper
Hungry Birds	Digdug
Missile Command	Intersection
Plaque Attack	Seaquest
Camel Race	Butterflies
Escape	Crossfire
Lemmings	Infection
Modality	Roguelike
Wait For Breakfast	Survive Zombies

TABLE I
TABLE OF GAMES SELECTED BASED OF [59], [57].

D. Choosing the Tree Search Algorithms

In this paper, a total of 6 tree search algorithms were chosen to compare;

Best First Search

This algorithm chosen because it is used in one of the best controllers submitted to this competition, as mentioned in section III-B8.

Breadth First Search

This tree search method was used because it is one of the most basic tree search algorithms, as it essentially brute-force searches the tree, and can be used as a benchmark for the other tree searching algorithms. In this paper, breadth first search starts a new tree search at each game tick.

Breadth First Search Extended

This is the same as the other breadth first search, with the only difference is that it continues the search throughout the game, so the agent will sit idle until the tree search finds a solution.

MCTS

The sample MCTS controller is the vanilla MCTS algorithm as mentioned in section III-B2, this was chosen because it has been one of the most successful algorithms in the competition, and has been used widely within the competition [8], [39]

Flat MCTS

Sample flat MCTS is a monte carlo tree search simulation where actions at the current state are uniformly sampled and no tree is built [5], this was chosen as it is interesting to compare how the different sample MCTS controllers perform.

Open Loop MCTS

The sample open loop variation of MCTS (OLMCTS) as mentioned in section III-D3 and III-B. This was also chosen to compare against the other sample MCTS variations.

E. Design of Data Collection

When running the simulation of all the games in the framework, there were a few issues that arose, the main issue being that when running roughly a total of 600,000 games,

[1] https://github.com/EssexUniversityMCTS/gvgai

$(1000(simulations)\cdot 5(levels)\cdot 20(games)\cdot 6(controllers) = 600,000)$ the java executable would run out of memory. To solve this a simulation launcher was developed where it would run a set amount of games concurrently, until they finished executing, then another one from the list of games would be added to the queue. This means that each java executable ran only 5000 games, compared to 1.5-2M. Moreover, the python launcher program is a lot more scalable in comparison. Typically the total games to concurrently simulate is set to the core count of the processor.

Because this solution is a lot more salable, the results were gathered using a cloud computing solution which allowed many simulations to run in parallel, for long periods without any interference.

As the data collection class outputs a json file, which was intended to be used for nested data structures, however only ended up only outputting flat json, thus a simple json to csv converter was developed in python to be used for importing large datasets into R. This specific converter can be located in the Github directory for this project [2].

To gather the data about the different search trees, I implemented a class that would visualise the time an agent spent in each cell. This gathered insights into the movement patterns within the game, to help analyze if / where the agents got stuck in a level. An example of this can be shown in figure 5

F. Implementation of tree search algorithms

There are two implementations of the breadth first search algorithm in this paper, one which was modified from the *number27* controller, and one I wrote. The one that was modified from number27's controller is one that expands the game tree search over several game ticks and is named Breadth First Search Extended (BrFSE) in this paper, where as the one I wrote creates a new tree search every tick of the game and is named Breadth First Search (BrFS).

The MCTS controller implementations are provided by the competition, as described in section III-D3. The best first search algorithm was extracted and modified from the 2016 ICELab controller.

These algorithms will record various stats about the game they will play, such as; *score, level size, percentage of level explored, total cells visited* and *end game tick.*

G. Game Visualisations

1) Visualising time spent in areas: At the start of this project, to learn how to start gathering data about the different tree search algorithms, I started by creating some visualisation tools to help understand how the sample controllers worked, one which visualized the area in which the agent searched, and one to visualize how much the agent explored in a level. This worked by calculating how long the agent spends in each cell of the game, as shown in figure 6. From this you can see that only best first search and breadth first extended were able to win the game, this is due to them both being able to expand their tree searches over several game ticks, where as all the other algorithms suffer more from the locality limitation, as they start a new tree search every 40ms.

[2]https://github.com/Alli1223/comp320-comp360-dissertation

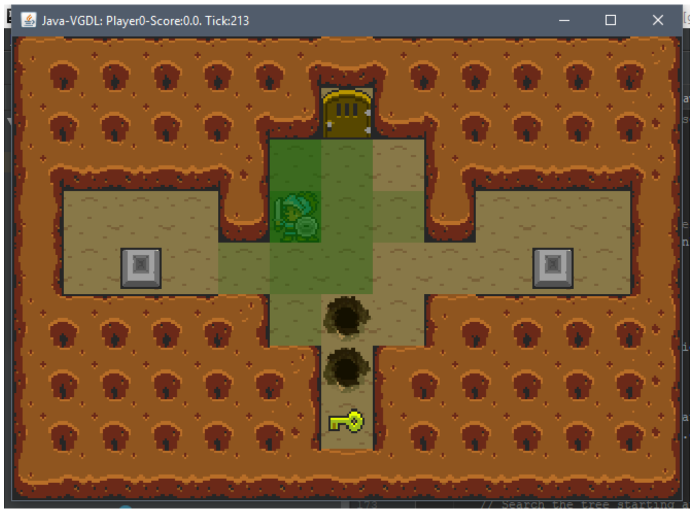


Fig. 5. Tree search locality represented in by green cells

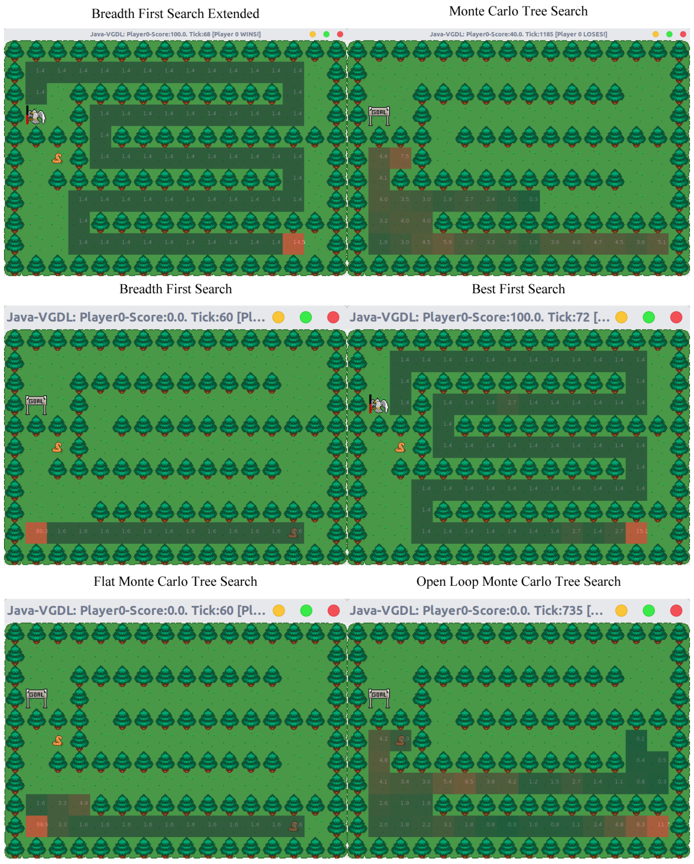

Breadth First Search Extended
Monte Carlo Tree Search
Breadth First Search
Best First Search
Java-VGDL: Player0-Score:0.0. Tick:60 [Pl...
Java-VGDL: Player0-Score:100.0. Tick:72 [...
Flat Monte Carlo Tree Search
Open Loop Monte Carlo Tree Search
Java-VGDL: Player0-Score:0.0. Tick:60 [Pl...
Java-VGDL: Player0-Score:0.0. Tick:735 [...

Fig. 6. Time agent spend in each cell of the game

H. Chosen Development Life Cycle

The software artifact for this project was developed using the agile approach to software development [60], [61], [62], this is a method of software development where the software is developed and tested incrementally, in small iterations.

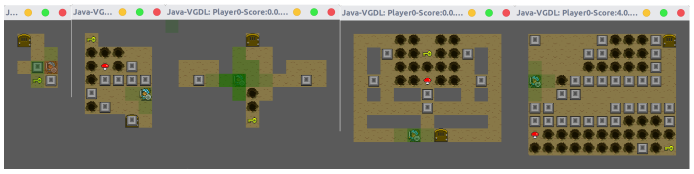


Fig. 7. Tree area search comparison for first five levels on the game Bait

The reason why this development methodology was chosen is because at the start of the project I wasn't sure what direction to take with the project, and agile gives freedom to quickly adapt to any changes [61], [63]. Compared to other development methodologies, such as the waterfall method [64], where it is a lot less flexible to changing requirements.

V. Results

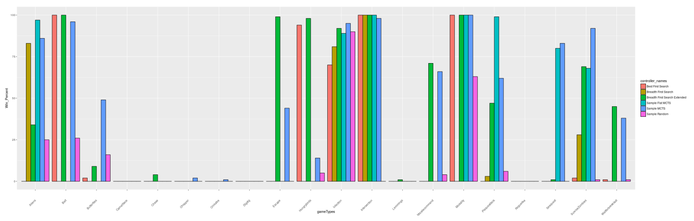


Fig. 8. The win rate of all tree search controllers on the chosen 20 games of the competition

Figure 8 shows the win percentage of the first level of each of the 20 games being tested, this shows that all the different tree search techniques generally struggle with similar games, of which are both deterministic and stochastic games.

One notable finding is how well the random controller did in both infection and modality games, with a $> 50\%$ winrate on those games. This may suggest that those games may be not too challenging compared to the other games selected.

The within the chosen 20 games, four of them have varing level sizes, 3 of those games are deterministic, and one is stochastic.

A. *Anaysis of Deterministic Games*

1) Bait: Bait is a game where there are holes blocking the player from getting a key to unlock the door, and the agent has to move rocks into the holes to get the key to win, figure 7 is what the second level of this game looks like. Figure 9 shows that level size does effect the win percent of the controllers, and it seems to effect all the controllers almost equally. The last level of bait

2) HungryBirds: Hungry birds is a game where the agents have to find a way to the goal, while their health is slowly decreasing, and there are bugs that the player can find and eat to recover health and get a higher score. In figure 11 the extended breadth first search, maintained an average win percentage of 82%, and did not seem to be effected by the level size, this is likely due to the game space being easy to explore over several game ticks. Best first searchs score actually increased very slightly with the level size, however

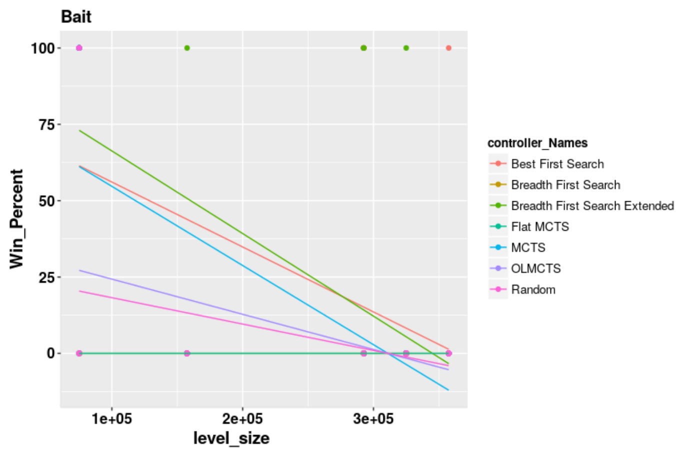


Fig. 9. Scatterplot with regression line of win rate for search algorithms in the game bait

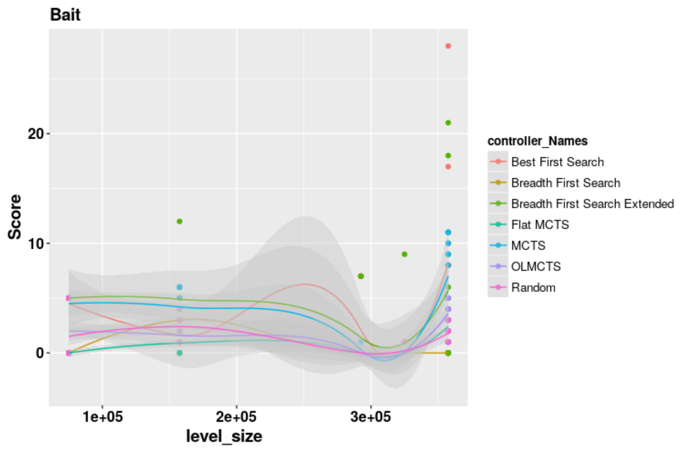


Fig. 10. Comparison of score for search algorithms in the game bait with regression line smoothing and confidence bounds

MCTS and OLMCTS win percentage dropped significantly. Flat MCTS and breadth first search both had an almost 0 % win rate in this game, simply because they couldn't search the game space deep enough.

3) Modality: The game Modality is similar to the game bait, however the agent only has to move one object into a hole by walking into it, figure **??** shows an image of the game. Figure 13 shows a significant decline in all tree search performance as the map sizes increase. The only controllers that were able to even complete the largest level were best first and breadth first extended.

B. *Anaysis of Stochastic Games*

1) Butterflies: In the game butterflies, the agent has to catch all the butterflies before they eat all their food. However the butterflies move around randomly.

Figure 15 shows that MCTS and OLMCTSs performance actually increased as the level size increased. One reason for why this might be is because the level size did not actually increase by much, only by 6%, as discussed in section VI.

However this graph suggests that hypothesis two is true, since both breadth first search had a total win percentage of

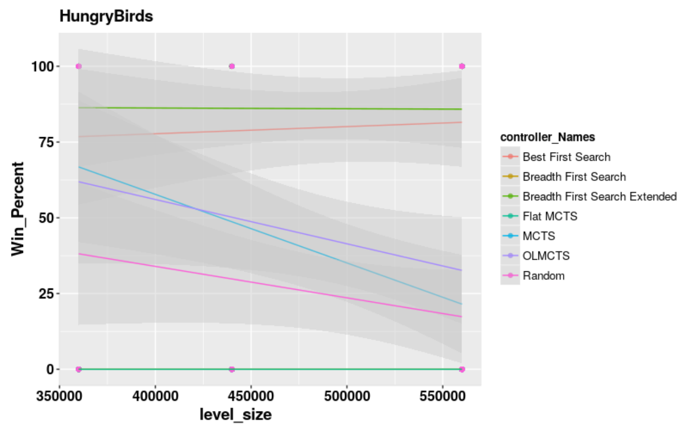


Fig. 11. Comparison of win rate for search algorithms in the game HungryBirds

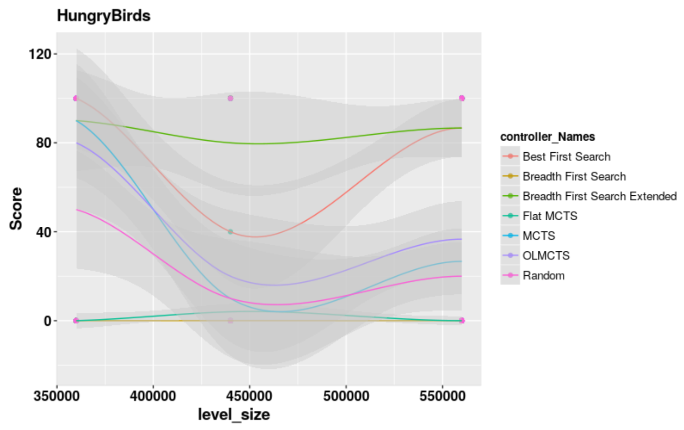


Fig. 12. Comparison of win rate for search algorithms in the game hungrybirds with regression line smoothing and confidence bounds

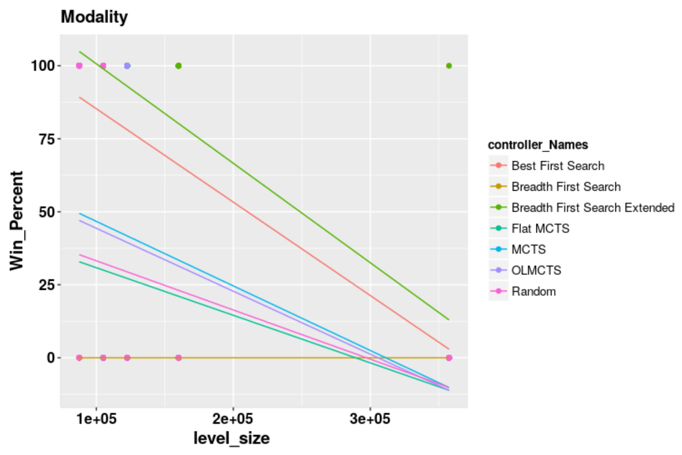


Fig. 13. Comparison of win rate for search algorithms in the game modality with regression line smoothing and confidence bounds

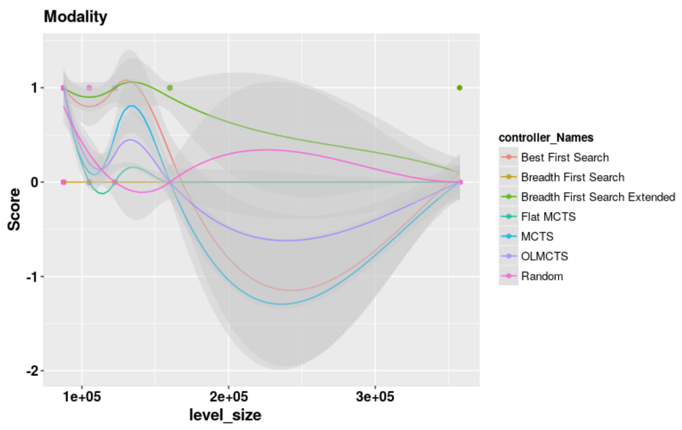


Fig. 14. Comparison of win rate for search algorithms in the game modality with regression line smoothing and confidence bounds

0% in all the games, and best first search had $< 5\%$ over all the levels. Moreover this also proves the null hypothesis to be true for hypotheses four and six.

In figure 16 all the controllers scores decreased almost uniformly, however MCTS and OLMCTS got the highest scores throughout the games.

VI. Possible Issues

A. Game Level Comparisons

When comparing the 4 different games with varying map sizes, each game has different scaling of map sizes, and as mentioned in V-B Butterflies only has a 6% total map size increase, where as HungryBirds has a 36.3% total increase in level size.

B. Not Enough Results

In these results each level was tested only 10 times, so a total of 50 games per game was used, this was chosen as it is the same amount as the official gvg-ai testing servers use. However there were plans to do 5000 games per level, but unfortunately those results turned out to be incomplete, and insufficient time was left to be able to re-run the same amount. Another issue was that only four games that data had been collected from had varying map sizes, in hindsight the creation of a standardised map size variation for each of the 20 games should have been created to give more reliable conclusions. This also lead to an odd number of stochastic and deterministic games being measured, which is not ideal.

VII. Future Work

The experiments in this paper only used 20 of the games in the competition, when there are currently 82 games in the competition, so comparing all the games in the competition may lead to more interesting results. There are also quite a few other tree searching algorithms that are not evaluated in this paper, as this paper only analysed 4-5 when there are a lot of other algorithms that have not been looked at due to the project scope. Moreover, the best first search implementation in this paper searches the game over several game ticks, like the BrFSE controller, however there isn't an BeFS controller that starts a new search every game tick, like BeFS. This gives an unfair comparison for BeFS as there really should be two different versions, one that searches over several game ticks, and one that starts a new search every game tick.

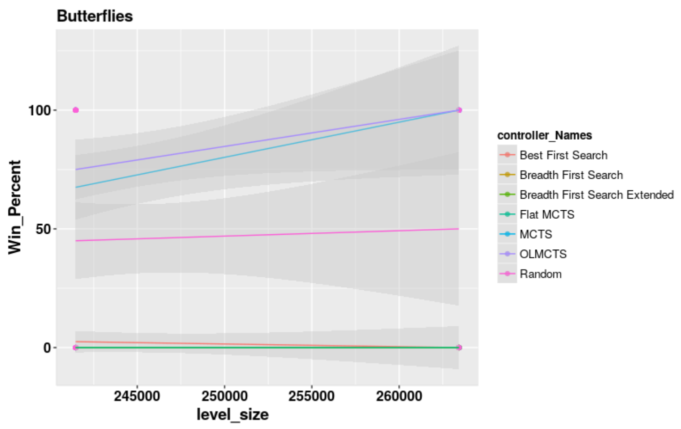


Fig. 15. Comparison of win rate for search algorithms in the game Butterflies with confidence levels

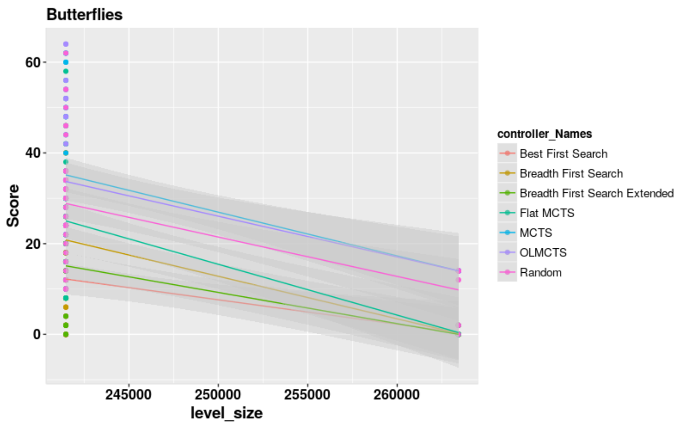


Fig. 16. Comparison of win rate for search algorithms in the game HungryBirds

Another item is measuring the tree search vs level size performance on a wide verity of games, instead of just four, as mentioned in VI.

VIII. Conclusion

While this paper focused on a very small subset of games and tree searching algorithms, these findings can help to lead the development of better AI, not only for games, but also a wide verity of applications outside of games research as well. There has been a lot of attention has been drawn to more complex tree searching algorithms such as MCTS, but there has been very little for basic tree search algorithms, such as depth and breadth searches. This paper found that breadth first search and best first search and both perform better than the sample MCTS for deterministic games as the level size increases. Furthermore, this paper also found that MCTS performs better on stochastic games compared to breadth first search and best first search.

Appendix A
Reflective report

A. *Issues with collecting the data*

One of the big issues I faced when I initially started collecting the data was being able to run many games at one time, and as each game takes on average about 1-2minutes to complete, when there are roughly 600,000 games to simulate, if I was to run one game at a time, it would have taken about 1.14 years (416 days) to gather all my data. This is why the creation of the simulation launcher help so much, the only issue was that it was only created 2 weeks before the deadline, so I ended up having to spin up 4 large servers, that had a total of 56 processing cores and 224GB of RAM, which was able to parallelise the data gathered. This means that the data only took about a week to gather. However when I went to collect the results, I found that a few of the games had errors, which meant I had to throw away all that collected data. I was planning on combining two of my large data sets (one of 100 and one of 1000 simulations) however when loading the data into R, it doesn't like having columns of different lengths. So i had to end up running a quick run through of the games 10 times, just so that I would have results to analyse. This was the biggest problem because I focused too much on gathering a large data set for statistical accuracy, when this wasn't necessary, only a small amount of games actually needed to be simulated to gather an insight to what the controllers stats were. This would have also meant I would have a lot more time for the write up and analysis of the data, because I ended up only having a day to do the write up.

B. *Not having a defined hypothesis before collecting the data, to base my research around*

When I started collecting the data about the different tree searching algorithms I did not have a defined objective that I wanted to prove, this meant that at the start of the development of the artifact I spent a while working on some visualisation stuff that were not relevant to proving my hypotheses.

C. *Outputting json data*

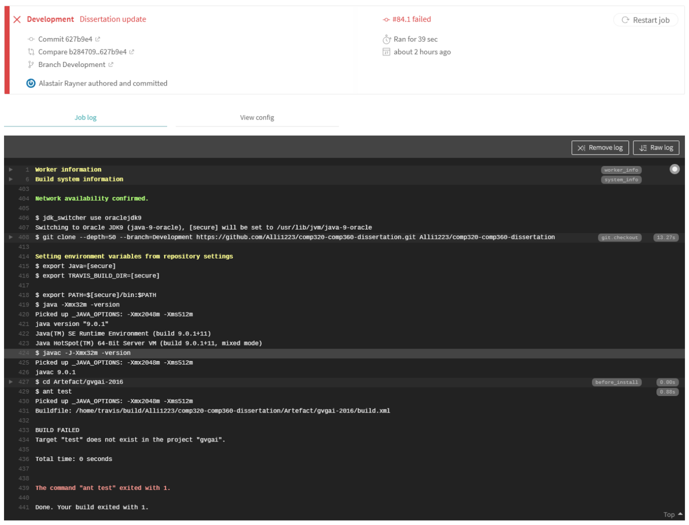
At the start of the project, i planned on outputting nested json data, which would contain the values of where the agents went, which would have been used for a heatmap. However to load data into R I started using an online json to csv converter, but this took a long time, as there were about 120 total conversions I would have had to do by hand. I did this once, but then when I had to do it a second time, I created a simple python script that converted all the json to a csv format.

D. *Unit Testing*

While developing the software, I attempted to set up junit 5 with travisCI early on in the software development lifecycle, however I was not sure how to set up junit to build my project correctly. Moreover, I was unsure what to actually test with unit testing, figure 17 shows the build errors from the TravisCI website [65].

E. *Prioritisation of tasks*

Throughout a large part of this project I ended up working of items that in the end had no relevance to my research question or hypothesis, such as the visualizations class in the project, which i spent about 1-2 months working on.



```
Development  Dissertation update                #84.1 failed                Restart job
  Commit 627b9e4                                Ran for 39 sec
  Compare b284709..627b9e4                      about 2 hours ago
  Branch Development

  Alastair Rayner authored and committed
```

```
Worker information                                                              worker_info
Build system information                                                        system_info

Network availability confirmed.

$ jdk_switcher use oraclejdk9
Switching to Oracle JDK9 (java-9-oracle), [secure] will be set to /usr/lib/jvm/java-9-oracle
$ git clone --depth=50 --branch=Development https://github.com/Alli1223/comp320-comp360-dissertation.git Alli1223/comp320-comp360-dissertation    git.checkout  13.27s

Setting environment variables from repository settings
$ export Java=[secure]
$ export TRAVIS_BUILD_DIR=[secure]

$ export PATH=$[secure]/bin:$PATH
$ java -Xmx32m -version
Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
java version "9.0.1"
Java(TM) SE Runtime Environment (build 9.0.1+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.1+11, mixed mode)
$ javac -J-Xmx32m -version
Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
javac 9.0.1
$ cd Artefact/gvgai-2016                                                        before_install  0.00s
$ ant test                                                                                      0.88s
Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
Buildfile: /home/travis/build/Alli1223/comp320-comp360-dissertation/Artefact/gvgai-2016/build.xml

BUILD FAILED
Target "test" does not exist in the project "gvgai".

Total time: 0 seconds


The command "ant test" exited with 1.

Done. Your build exited with 1.
```

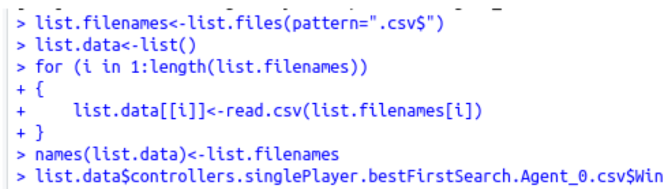
Fig. 17. TravisCI builds failing because project hasn't yet been set up, due to being unsure about what to unit test.

F. conclusion

Although I am happy with the software side to this project, I feel that this project will have very little impact towards the greater scientific community.

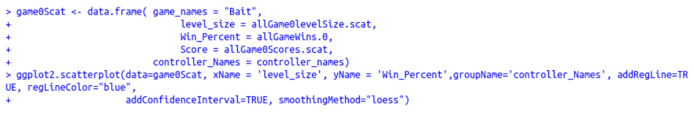
Appendix B
R code snippets

All the graphs were compiled using R, figure 18 shows how the large data sets were loaded into R. As this is a large image, the full size image can be viewed in the git repository for this project [3].



```
> list.filenames<-list.files(pattern=".csv$")
> list.data<-list()
> for (i in 1:length(list.filenames))
+ {
+     list.data[[i]]<-read.csv(list.filenames[i])
+ }
> names(list.data)<-list.filenames
> list.data$controllers.singlePlayer.bestFirstSearch.Agent_0.csv$Win
```

Fig. 18. R code snippet used for loading large game data sets



```
> game0Scat <- data.frame( game_names = "Bait",
+                               level_size = allGame0levelSize.scat,
+                               Win_Percent = allGameWins.0,
+                               Score = allGame0Scores.scat,
+                         controller_Names = controller_names)
> ggplot2.scatterplot(data=game0Scat, xName = 'level_size', yName = 'Win_Percent',groupName='controller_Names', addRegLine=TR
UE, regLineColor="blue",
+                    addConfidenceInterval=TRUE, smoothingMethod="loess")
```

Fig. 19. R code snippet used for creating scatter plot with linear regression line and confidence bounds with smoothing

Appendix C
Acknowledgements

I would like to thank my project supervisor, Dr. Edward Powley, for his guidance and knowledge throughout this project. I would also like to thank all the final year BSc students who helped give peer feedback and support throughout the project. Finally I would like to thank Dr. Michael Scott and all the other BSc Teaching staff at Falmouth University for their excellent teaching throughout the last three years.

[3]https://github.com/Alli1223/comp320-comp360-dissertation/blob/Development/Dissertation/images/allGameBarPlot.png

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.

[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.

[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] 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.

[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] D. Pérez-Liébana, 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.

[9] T. Joppen, M. Moneke, N. Schroder, C. Wirth, and J. Furnkranz, "Informed hybrid game tree search for general video game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, 2017.

[10] D. Pérez-Liébana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the robustness of general video game playing agents," in *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 2016, pp. 1–8.

[11] D. Pérez-Liébana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open loop search for general video game playing," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 337–344.

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

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

[14] 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.

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

[16] 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.

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

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

[19] S. Gelly, Y. Wang, O. Teytaud, M. U. Patterns, and P. Tao, "Modification of uct with patterns in monte carlo go," 2006.

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

[21] D. Pérez-Liébana, 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.

[22] D. Silver and G. Tesauro, "Monte carlo simulation balancing," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 945–952.

[23] 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.

[24] 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.

[25] D. Pérez-Liébana, 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.

[26] 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.

[27] 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.

[28] 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.

[29] K. Subramanian, J. Scholz, C. L. Isbell, and A. L. Thomaz, "Efficient exploration in monte carlo tree search using human action abstractions."

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

[31] D. Pérez-Liébana, 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.

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

[33] 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.

[34] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[35] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: a survey," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, p. 35, 2013.

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

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

[38] 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.

[39] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms," *arXiv preprint arXiv:1802.10363*, 2018.

[40] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[41] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[42] 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.

[43] 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.

[44] 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.

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

[46] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," 2008.

[47] D. Pérez-Liébana, "The general video game ai competition," http://http://www.gvgai.net/, 2017.

[48] J. M. Gonzalez-castro and D. Pérez-Liébana, "Opponent models comparison for 2 players in gvgai competitions."

[49] 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.

[50] 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.

[51] 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.

[52] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari," 2018. [Online]. Available: http://arxiv.org/abs/1802.08842

[53] 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.

[54] 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.

[55] D. Pérez-Liébana, 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.

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

[57] R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana, "Population seeding techniques for rolling horizon evolution in general video game playing," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 1956–1963.

[58] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016, pp. 122–128.

[59] C. Guerrero-Romero, A. Louis, and D. Perez-Liebana, "Beyond playing to win: Diversifying heuristics for gvgai," in *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*. IEEE, 2017, pp. 118–125.

[60] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," 2001.

[61] A. Cockburn and J. Highsmith, "Agile software development, the people factor," *Computer*, vol. 34, no. 11, pp. 131–133, 2001.

[62] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, vol. 50, no. 9-10, pp. 833–859, 2008.

[63] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[64] S. Balaji and M. S. Murugaiyan, "Waterfall vs. v-model vs. agile: A comparative study on sdlc," *International Journal of Information Technology and Business Management*, vol. 2, no. 1, pp. 26–30, 2012.

[65] TravisCI, "Travis ci," https://travis-ci.org/Alli1223/comp320-comp360-dissertation, 2018.