General Video Game Playing with Goal Orientation

Blaine Ross

September 2014

Master Thesis

Advanced Computer Science

University of Strathclyde

Supervisor:

Dr. John Levine (Senior Lecturer, University of Strathclyde)

Abstract

Artificial Intelligence of the past has been primarily designed to deal with specific tasks, but within the field of general video game playing intelligent agents must be created that can cope with multiple different unknown scenarios. The AAAI started a competition in 2005 that allowed board games to be the focus of this research, and a new competition has been started in 2014 to help further this which now focuses on video games, and will be concluded at the IEEE CIG conference in Germany. This competition features a library of Java based games where anyone can try and create general video game playing agents and then submit them for evaluation. To date, focus has been given to tree search based approaches such as Monte-Carlo Tree Search, but this method has its drawbacks which does not make it applicable to all types of gameplay. In this report a possible solution is suggested to cope with one of these limitations by looking at a more goal oriented approach in combination with Monte-Carlo Tree Search. By using this newly designed method a comparison is able to be drawn with a variety of other techniques that do not have this goal based focus. The results are promising and show potential not only in the scores achieved during gameplay, but in the overall behaviour of AI controlled agents when playing video games with greater goal orientation.

Contents

1. Motivation

Artificial intelligence (AI) within games has been worked on for over half a century even since the term was coined by John McCarthy in 1956 [67]. It quickly found prevalent use within board games like chess, checkers and GO where the learning and search based problems faced within these games made them a perfect fit for AI based research [9]. The challenge soon became not to play the game, but to create artificial intelligence strong enough to be able to challenge human opponents. It was not realised what a difficult challenge this would prove to be and the initial goals set out for trying to accomplish this were far beyond what was actually achievable [38].

Claude Shannon's paper[5] in 1954 was one of the first that proposed how a computer program could be designed to compete at chess, and ground breaking research was done soon after by Samuel [39], with the game Checkers, who showed that it was now possible through AI for computer opponents to consistently outperform their human counterparts. This work has continued on board games up to the present day where most notably Deep Blue, a computer specifically designed to play chess, was developed. It went on to have one of the defining crowning moments of game related AI when it defeated chess world champion Garry Kasparov[6]. Deep Blue was able to evaluate 200 million positions a second compared to just the 2 that a human player is capable of [41].

The problem with these game playing systems like Deep Blue is that they don't serve a great purpose apart from being able to play one particular game, albeit incredibly well. Try and make one of these systems play a different game and they would not even be able to function, let alone be able to compete with a human player. It would be much more efficient if there was a system in place that was able to play any game set before it.

This is why human players can claim to be better game players than computer programs because regardless of whether they are playing a slow based, tactical board game, or a dynamic unpredictable video game, humans are able to learn and adapt their style of play dependent on what is set before them. It's in this sense that a human's intelligence can be seen as more *general* [2]. This idea when applied to board games became known as general game playing (GGP), and in more recent times when applied to video games is now referred to as general video game playing (GVGP).

GVGP is a new concept, with the Java framework supporting it being used in this report only being completed in mid-2014 and being the first of its kind. Using this framework it should be possible to analyse how GVGP could successfully be achieved, and also focus on one of the major talking points of any general game playing system, which is how to identify goals [51].

1.1 Objectives

Below are listed the main objectives and aims of this report.

- Discuss the main principles of current GVGP design in relation to video games.
- Study previous work done in the field and discuss the main advantages and disadvantages of the methods being used.
- Explain how a more goal oriented approach could lead to more successful video game playing agents in relation to these methods.
- Design an agent based around a goal based approach to play the games that are available in the GVG-AI framework.
- Make a comparison with how the agents designed improve on the ability of current agents to play video games.
- Discuss the future of GVGP based research and what the next steps are likely to be in its development.

2. Background

In this section of the report a brief background is given behind GGP and how it is being adapted to be used with video games.

2.2 General Game Playing

GGP is the notion of AI being used so that it's able to play any game that it has never seen before without any human intervention [26]. This is possible through the use of an agent that takes the place of what would normally be a human player and then learns how to play the game by itself. The agent will have access to a list of definitions that describe the game, and a list of all the possible actions that are available to be taken. It is then up to the agent to use these game definitions and possible actions to evaluate the best course of action to take, in the same way as a human player would also have to do so.

From continuous playing of a game an agent will be able to learn from its experiences and improve its ability to play a game in the same way that a human player would [32]. With GGP the AI will have no predetermined idea of what problem it is that's trying to be solved, therefore the tasks that need to be accomplished by the agent cannot be solved using predesigned algorithms, but the algorithms must be discovered by the AI [1]. This becomes much more efficient, as the work is being done by the computer as opposed to a human programmer who would normally be having to take into account every eventuality before the agent is able to compete.

Thielscher and Schiffel [51] suggest that there are four functions that a complete general game player must have: (1) they must have adequate interpretation skills as to decipher game rules and know what legal and non-legal moves are. (2) The ability to search a game tree and if this tree is not available then (3) an evaluation process must be established to compensate. (4) There needs to be a way of recognising game-specific knowledge if any sort of strategic play is to be attempted.

The major advantage of this GGP approach is that you don't have to design multiple different agents for multiple different games. Instead you design an agent that is able to adapt and play any finite number of game that is laid before it [69]. This bypasses one of the limitations of traditional AI where large complex algorithms are often needed to be

completely recoded for games, and also systems outside of video games, for the new domains they have to be used in.

This key concept of GGP of allowing AI to adapt to different scenarios is not just of use to games as the idea can be applied to a variety of other fields including robotics where currently research is incredibly restrictive to the one or few tasks that a robot is being designed for [2]. GGP could also find use in wide range of other possible areas including business process management, electronic commerce, and military operations [52] showing the possible potential this area of research has.

2.1 General Game Playing with Video Games

Much like how board games were a precursor for video games when it came to “regular” AI this is also the case with general game playing. It was seen by many coming into the 21st century that the problem of playing many board games with AI had been “solved” [7], including games like Connect 4, Backgammon, Scrabble, and GO and it was at this point that research began to focus more heavily on trying to achieve the same success with video games.

Video games have become an increasingly popular method of developing AI because of the diverse scenarios that they provide. As technology improved a new generation of types of video games were developed with genres like real time strategy and role playing games offering new challenges for AI development compared to the arcade games that had been seen before. The importance that AI was starting to have in games can be seen by how much computational time it was starting to be given. In 1997 only 5% of CPU resources were put towards calculating artificial intelligence, but only 3 years later it was stated that it had gone up to as much as 25%[40] in some cases.

When applying GGP to video games it does not take long to appreciate how challenging of a problem it is. In a fighting game the agent will be given a list of fighting moves and told how to react to its opponent’s moves accordingly, while in a racing game the AI will be given specific instructions on how to “drive”. Striving for the same agent to be able to play two vastly different games like this is something that is yet to be achieved. Research at the moment is primarily based within 2D games like Pac-Man, Space Invaders and Frogger, with the goal being that if this is successful then eventually research will be moved on into 3D games [2].

With GVGP it's not a simple case of the agent performing solely as well as the programmer coded it. A successful GVGP agent must integrate various AI technologies together, including, reasoning, learning, representation and rational decision making [3] and be able to display some kind of autonomy in its actions that will only become apparent when the agent can be seen playing games. Combining all of these elements together successfully is still something that is far from being fully achieved.

GVGP could also help to solve the problem of iteration within video games. Annual releases of video games following the same series is a common occurrence within the video game world and many of these games are of a similar type, yet will require completely different sections of AI code for the game to run. Now instead of redesigning an agent, the only thing that would need to be changed is the definition of the game and the agent would handle the rest.

2.2 General Game Playing Competitions

One of the biggest contributors towards AI based research are the amount of annual competitions that are held to try and push research forward. Some of these competitions include: the Arimma [54] competition to develop an agent to beat human opponents at a board game; The Google AI challenge [55] held from 2009-2011 where game playing agents were created for a specific game each year; and the 2K Australia competition [56] to develop agents for first person shooter games, just to name a few.

One organisation that saw the potential in this research was AAAI [1] who set up an annual competition for general game playing (among other game related AI projects) in 2005 for programmers to submit their own solutions to the problem. The most successful GGP agents submitted for this competition have primarily involved tree based searches [26] such as the CadiaPlayer [53] which has previously won the AAAI GGP competition in 2007 and 2008 and was the first agent to use Monte-Carlo Tree Search approach that has now become the standard in GGP research.

This competition has only focused on board games, however the GVG-AI competition [56] was set up in 2014 which now focuses on creating general game playing agents that are able to play video games. In this competition agents created are run against a number of different video games to see if they exhibit the type of general intelligence that we would normally associate with a human player.

More specifically, competitors are given a training set of 10 games to test their agent on, after which the agents can be submitted to be trialled against a secret validation set of an additional 10 games on a submission server that cannot be seen in advance. At the competitions conclusion between 26th and 29th of August 2014 the agents will be tested against a final set of games, which again, are unseen by the competitors.

2.3 Conclusion

In this section a brief summary was given detailing how GGP started and then how GVGP was able to evolve from it. Following on from this the key parts that make up a GVGP system will be looked at in relation to the GVG-AI framework which is being used throughout this report.

3. GVG-AI Framework

The GVG-AI framework is a Java based library designed for use by competitors in the GVGP competition to be held at CIG 2014. In this section of the report the parts that make up that framework are analysed and shown how they contribute towards GVGP.

3.1 The Game Description Language

For GVGP to be effective it requires a game description language (GDL) so that the rules of the games can be described in a standard format across all the games that the agent will have to play [50]. These languages have been attempted in the past such as the GDL Gala created by Koller and Pfeffer [28] which was used to great effect in games like poker, but none had been made to work with video games.

It was in 2013 when Tom Schaul [4] developed the VGDL (Video Game Description Language), a high level descriptive language making the process of creating arcade style games possible through the use of a small amount of key words and rules. Originally developed in Python, the language was ported over to Java which makes it useable within the GVG-AI framework used in this report.

Schaul highlights that the goal of the definition language is that it should be clear, human-readable and unambiguous. The language should also be easy to extend to take into account the many new types of games that may need to be created, and be easily parsed so that newly created games should run without much effort. This also has the added benefit that users with no programming experience will be able to develop games using the language. A big motivation for the language is that it could also be used as a benchmark [14] within the AI community for continual advancement in the field of GVGP through events similar to the AAAI competitions.

The language has a list of features ranging from teleportation to gravity effects (a lot of which can be seen in the GVG-AI framework of games) which can easily be added to if a greater range of games wanted to be created. The GVG-AI framework has a large amount of ontology classes that are used to hold all these different definitions. For the moment the framework is designed around fast paced arcade games that do not have the complex graphical or gameplay requirements of more modern games.

The language condenses all the code that would normally be needed to create individual games and instead you are left with a single file split into four different sections. An example of this can be seen below for the game Missile Command which is one of the games featured in the GVG-AI framework.

```
SpriteSet
  city  > Immovable color=GREEN img=city
  explosion > Flicker limit=5 img=explosion

  movable >
    avatar  > ShootAvatar stype=explosion
    incoming >
      incoming_slow  > Chaser stype=city color=ORANGE speed=0.1
      incoming_fast  > Chaser stype=city color=YELLOW speed=0.3

LevelMapping
  c > city
  m > incoming_slow
  f > incoming_fast

InteractionSet
  movable wall  > stepBack
  incoming city > killSprite
  city incoming > killSprite scoreChange=-1
  incoming explosion > killSprite scoreChange=2

TerminationSet
  SpriteCounter stype=city   win=False
  SpriteCounter stype=incoming win=True
```

Figure 1 Game description for Missile Command.

For the VGDL the rules within the game environment have been split up into four different sections [4], an example of which can be seen in Figure 1.

Sprite Set - This holds all the classes of objects that will be used in the game. Indentation is used to show inheritance within the objects, so the incoming enemies will also be movable.

Level Mapping - Gives a list of all the different character objects in the game and how they should be translated from the starting game map. For example, the letters m and f are both used to show enemies that move at different speeds.

Interaction Set - The interaction set holds the list of all the possible interactions that can occur between sprites during a course of a game. So here you can see that an enemy sprite can collide with a city that will cause a negative score change in the game.

Termination Set - Tells the game under what conditions the agent will either win or lose.

When a game has been defined levels can be created in a similar simplistic fashion with the layout of a level being a grid of characters referring to different sprite types (W = wall, A = agent/human controlled player). This makes the process of creating new levels based on already existing games very easy to achieve.

```
w    m     f    m   f  w
w                      w
w                      w
w                      w
w                      w
w                      w
w            A         w
w                      w
w                      w
w                      w
w   c              c   w
wwwwwwwwwwwwwwwwwwwwwwwww
```

Figure 2 Description of a single level for Missile Command.

According to Theilscher [29] game description languages require three things: Information about the players starting position, details on the termination and winning criteria of the game, and a list of all the possible actions available to the player. The first two of these points have been handled, in addition to other information like sprite interactions and level mapping which Schaul thought necessary to add for a video game based language. The last of these points mentioned by Theilscher is also achieved, and is talked about in section 3.3 of this report.

The overall aim of a GDL like the one described in the GVG-AI framework is that new games and levels can be created with no additional code being required [4].

3.2 The GVG-AI Games

The GVG-AI framework comes including ten different games which are referred to as a *training set*, each of which are styled around, or play, like a classic arcade. Every game has a playable avatar that is controlled via a keyboard, and every game is displayed from a top down perspective looking down on the avatar. The entirety of any level is always

visible from the start of the game, though dynamic elements will cause their layouts to change.

To examine how each of these games offer a different challenge for the agent a brief description of each game will be given, describing how the stochastic nature of the game affects the gameplay and also the goal and score based events within the games.

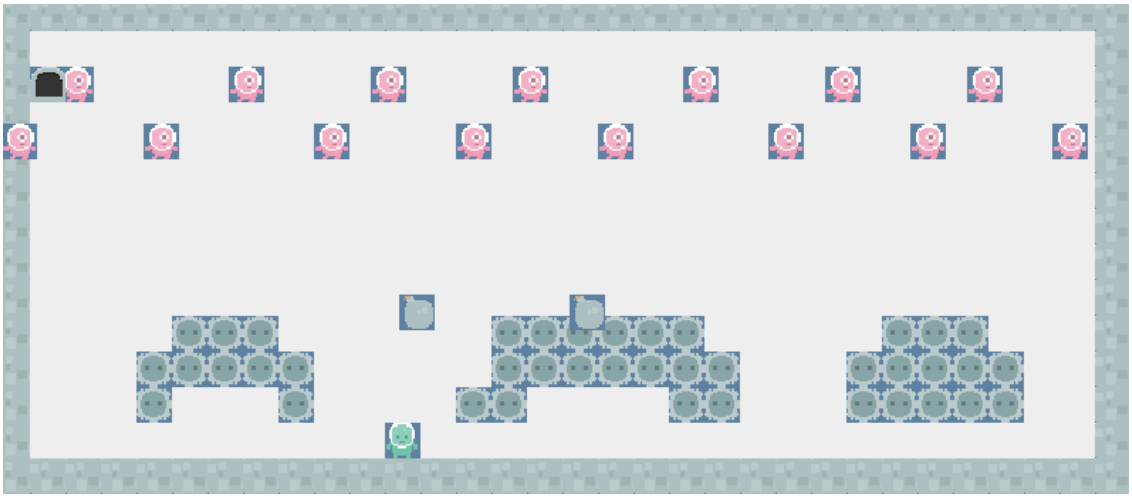

Figure 3 An in game shot of the game Aliens from the GVG-AI framework.

1. *Aliens*

Based on the arcade game space invaders the player must move either left or right while shooting projectiles up the screen. The agent can use blocks for cover and must try and shoot all of the enemy sprites before they reach the bottom of the screen and kill the agent. Points can be earned by shooting blocks in front of the user or killing the enemy sprites.

2. *Boulder Dash*

The agent must collect gems on the map then make their way to an exit portal. This must be done while avoiding enemy sprites and getting crushed by boulders which can fall on top of the agent if they move below them. Points are only scored from collecting gems (not level completion) and a minimum of 10 gems must be collected, each of which are worth 2 points. The number of gems available on each level varies, but some are unattainable.

3. *Butterflies*

The agent must move about the screen and stop enemy sprites from landing on mushroom sprites that must be protected. New enemies will spawn over time, and all the enemies must be defeated before the game time runs out or the game is lost. Points are gained for each butterfly the agent comes in contact with and subsequently removes, but points can also be lost for each mushroom that a butterfly lands on. If all the mushrooms get “eaten” the agent will lose the game.

4. *Chase*

The agent can move around a maze like course where other enemy sprites will try and run away from the agent. The agent must find and track down all of the enemy sprites within the time limit for the game to end. A point is awarded for each enemy sprite the agent comes in contact with, with a maximum of 7 points on offer.

5. *Frogs*

The agent must reach a goal point by traversing across moving platforms and avoiding stepping on damaging terrain. A point is only scored by reaching the end point of the level.

6. *Portals*

The agent must use a range of different teleportation portal squares on the map to travel to the exit point of the level whilst avoiding missiles. Levels range in complexity with a large amount of predictable, but also random missiles being fired across the map that the agent must avoid. The agent can only get a maximum of 1 point in any level for reaching the goal point.

7. *Missile command*

The agent must protect cities at the bottom of the screen by defeating enemies before they move down on them from the top of the screen. The game is lost if all the cities get destroyed and won if all the enemies are defeated. The agent can remove enemy sprites by moving towards them and using a weapon when next to them. Points are gained for each enemy removed, and points are lost for each city which is destroyed.

8. *Sokoban*

The agent is presented with a map with a number of portals and a number of boxes that must be pushed into them. Any amount of boxes can be pushed into any portal and 1 point is gained for each box that is removed this way. The game is only successfully completed when every box has been removed from the screen.

9. *Survive Zombies*

The agent must survive for a certain amount of time on a map where there is an endless wave of zombies that will spawn eventually filling up the entire map. The agent has the option of trying to collect coins worth 1 point each while they are trying to survive. Collecting coins also acts as an incentive by filling up the agents' health meter and allowing them to make brief contact with zombies. Coins will also re-spawn on the level in a random location a short period after they have been collected. No additional points are gained by completing the level.

10. *Zelda*

The agent must collect a key then navigate enemies on the way to the exit door. There are a few enemy sprites on each level that can be defeated through the use of a sword held by the avatar. A minimum of 2 points can be gained when completing a level by (1) collecting the key and then (2) reaching the exit door. Additional points can be gained by killing off the enemy sprites.

3.2.1 Game Goals within the AVG-AI framework

Goals are an element that appear in almost every form of gameplay, give games a sense of purpose, and give an idea to the player of how they're meant to play the game [8]. One board game which proved especially challenging for computers is Go, primarily because solutions can only be found when the game is close to conclusion [7]. This problem is only intensified when applied to video games, because the goals are diverse and will vary greatly from game to game.

What becomes immediately obvious looking at the list of games in the GVG-AI framework is that they offer a large range of gameplay techniques, playing mechanisms, and different scoring and goal possibilities.

The types of goals that can be established from these games are listed below. As more games are added to the framework this list will continue to grow which shows the importance in developing an agent that has a better chance of identifying them.

- Survival based: There is no way to force an end to the game, merely to survive till the game is completed.
- End Point Based: A certain location must be reached at which point the game will automatically end.
- Defeat all enemies/collect all resources: The game will end when all the enemy/resource sprites have been removed from the game.
- Puzzle based: The game will only end when a certain task has been completed by the agent.
- Time Based: The user has a set amount of time to complete a task, or alternatively must try and do something in the quickest amount of time. Every game is limited to a set number of game cycles so this goal applies to any of the games run. Speed is a necessity.

The games also raise another significant difference from board games in that there can be seen to be primary and secondary goals depending on the game being played. Every game listed implements at least one of these goal types, and some games use multiple, like in the case with Zelda where you must collect resources and also navigate to an end point on the map. Here the complexity inherent in video games is again evident, as unlike board games that generally have one overall goal, video games can have a multitude of different types of goals that need to be taken into consideration.

3.3 The Agent

An agent within the GVG-AI framework is always a playable avatar that can be seen in the game domain, never an overlooking player that you would have controlling a game like in chess. The character has a possibility of 6 actions to choose from: *left, right, up, down, use* and *nil.* Use is activated by pressing the spacebar and triggers an event created by the agent, however it is not active in every game. In the case of the games in the framework that use it, this will either mean firing a projectile or using a close combat

weapon. In the case of the weapon, it will take up one square next to the agent and will always be pointed in the direction the agent was last moving. This can sometimes make choosing actions troublesome as you may need to go further away from an enemy briefly before you are able to then move towards them at the right direction needed to hit them. The nil action simply means that there will be no action taken and the agent will remain where it currently is.

3.4 Game Observation

Game observation plays a crucial role in any GVGP scenario as the agent needs some way of being informed about the current state of the game. In GVGP this becomes slightly trickier as you have to be able to return general information that can apply to all games. The GVG-AI framework offers a variety of methods that are able to return useful information about the game state back to the agent [58]. There are two parameters that are sent to the agent, one being an instance of the *state observation* class and another being a *timer*.

In the framework there is a time limit in place for every game cycle. The timer is primarily used to tell the agent how much computational time they have used this game cycle, and how much time they have remaining.

The purpose of the state observation class is to give information about the state of the game to allow the agent to make better decisions. This can return *Avatar* (the agent) information like its position and speed, a list of *events* that have taken place in the game (sprite collisions), and also access to *observations* in the game.

A debatable point to be made from what the agent has access to, is just how much information the agent is given about particular sprite types. The agent can know in advance whether a sprite is a resource or a portal, and therefore importantly establish if a sprite is *not* a resource or a portal. This could prove crucial as an agent could be told to first prefer sprites that are resources and to avoid any other types of sprites in the risk of them being damaging to the overall score before the game has even started. This could be seen as not fitting in with the one of the main themes of general game playing where the agent has no prior information about the game domain [59]. Even a human player will have to interact with a sprite that looks like a resource (for example, a coin or a key) to confirm it is a resource despite how intuitive it may seem.

It may be expected that a more advanced agent would be able to learn whether these sprites should be avoided or not, or even be able to learn their purpose in the game. Given the extensibility of the library it may be tempting to add in additional rules for more sprite types if they were added into games instead of developing a way for the agent to work it out itself, which may set a dangerous precedent. However, it has been stated that “At the minimum, the agent will be given the current state of the world and told what actions are applicable” [2] which suggests there may still be some debate as to the level and indeed volume of information that the agent should have access to. In the implementation done in this report the types of sprites are purposefully ignored until the agent has come in contact with them so no advantage can be seen to be given.

3.5 Observation grid

One of the most important observations available to the agent from the state observation class is the observation grid. This has the ability to return a grid which can be referenced by coordinates allowing instant access to all observations in the game domain for each square in the game grid. In the solution proposed in this report thorough use is made of this feature so that the game state and all its observations can be scanned to help build up an idea of the agents surroundings.

3.6 Forward Model

One of the most vital abilities that the agent has is to be able to advance the game, which is given access to by a *Forward Model* of the game. The advance method is what allows the game to be played out so it’s possible to see what would happen if the agent was to choose certain actions. In essence this is allowing the agent to look in to the future of the game to see what the best possible option is for them to take. This method is used in all the agents that are mentioned in this report and a vital part in almost all GVGP search methods.

An important part to note about the advance method is that stochastic events in the game will not necessarily happen the same way when the game is played out for real [60]. For example, when advancing the game an enemy sprite may move to the left but when running the game for real the enemy may move to the right. This is why it is important to repeatedly advance the game multiple times to get as predictable an outcome as possible.

In a game with absolutely no unpredictability this method could theoretically return the same action, depending on the heuristic method used for evaluation. This does not apply to most games, and none of the games previously listed within the GVG-AI framework as they all have elements of randomness built in to them.

3.7 Conclusion

In this section of the report the GVG-AI framework has been thoroughly examined to show how it can make GVGP a possibility. Following on from this the methods which can then be used to take advantage of a system like this will be discussed.

4. Search Methods for GVGP

In this section of the report an analysis is given to some of the more popular and well known search methods used in designing GGP agents and should therefore also be of use when designing GVGP agents. Focus is especially given to Monte-Carlo Tree Search (MCTS) and genetic algorithms as they are used within the GVG-AI framework.

4.1 MCTS

One method which has become increasingly popular and is now seen as the forerunner in GVGP is using the MCTS algorithm which was spawned from the initial Monte-Carlo method proposed by Metropolis and Ulam in 1949 [10]. The method is described very broadly, and Metropolis and Ulam show how it could feasibly be applied to a wide range of mathematical and science based problems. Even at this early stage the use of the method in gameplay was already evident as one of the examples they used was with a game of solitaire and how to calculate the probability of successful outcomes.

It wasn't until 2006 [9] that the MCTS method was established as a way of successfully playing board game and video games. Its use can be seen in all types of games from arcade to even modern 3D real time strategy games [62] which was demonstrated by Champandard in 2014. Champandard outlines three reasons why the MCTS has become so popular: (1) Its incredibly well suited to cope with the unpredictability of video games, (2) the chance of making bad decisions is significantly lowered from repeatedly attempting different options, and (3) it's very computationally efficient as it can be told to stop and return a result at any point.

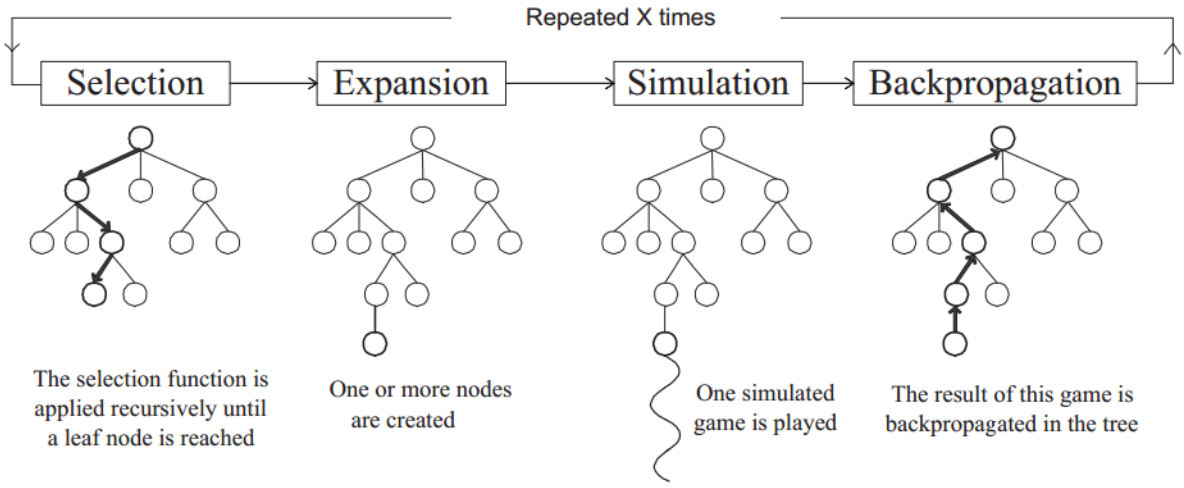

Figure 4 Outline of Monte-Carlo Tree Search process [9].

The highlighted example in Figure 4 shows the same 4-step process that any MCTS algorithm will follow.

Selection: To begin with a lead node will be chosen and from that lead node a child node and then successive child nodes will be chosen until a leaf node is reached. The ways in which these child nodes can be chosen make up the main differences between the variations of MCTS. Many different techniques have been attempted and found to work such as simulated annealing and progressive pruning [44] for this initial selection process.

Expansion: When expanding the tree, unless the leaf node encountered leads to an end state of the game then this node should try and be expanded. The most common method for expansion is that one node will be added per simulated game [11], and although there are other methods this seems efficient for most game based problems. In gameplay this would mean that one node would be added for each available action from that chosen node.

Simulation: One of the leaf nodes is chosen and then will subsequently be played out. This is commonly known as a rollout, and actions from this point will be selected at random until the end of the game [9].

Back propagation: After the simulation run has been completed the nodes on the tree are updated to show the amount of times that they were visited and will also hold some sort

of score based counter to record how successful they have been during the simulations. The node that is chosen to be executed will be the one that was followed the most times, as this will mean it has the highest probability for a good score.

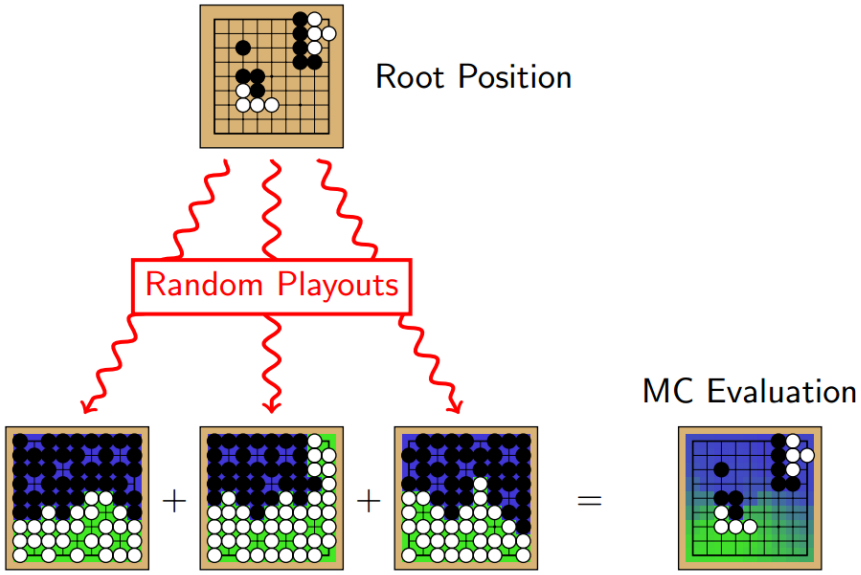

Figure 5 MCTS rollout example using the board game GO [43].

The rollout is one of the most crucial aspects of any MCTS algorithm. Figure 5 shows an example of how MCTS can be used in the board game GO to try and evaluate the next best possible board game state. A game state will be chosen and from that game state a group of random playouts (rollouts) of the game will take place. It's the end result of these rollouts that then mean an evaluation can be made deciding what the next best possible action is.

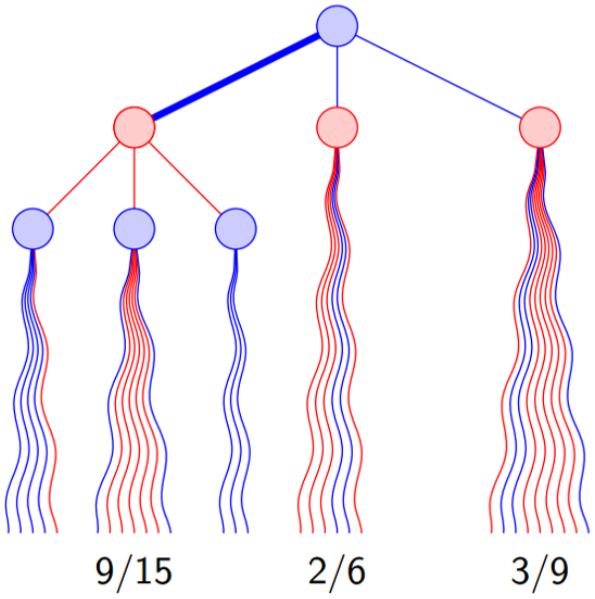

Figure 6 MCTS rollout example showing the branching of different nodes [43]. Blue curved lines indicate rollouts that led to a win and red curved lines indicate a loss.

Of course an important aspect to this rollout method is what part of the tree actually gets expanded. In Figure 6 you can see that the depth of the tree is increased in one section because it is achieving better results. So if an initial move by an agent is proving more beneficial, then that section of the tree will be expanded further. The left node in the example given has been shown to have been providing the best results so it's been chosen to be expanded further and then a random rollout executed from each of its child nodes. The depth of the tree can be expanded more before the rollout for additional accuracy. This does highlight a potential weakness of the approach with possible pathways being disregarded too quickly [9]. Just because a path hasn't managed to reach a solution at the first few levels does not mean that they won't have a better overall solution when going further down the tree. Therefore it is best to continue expanding the tree down as many branches as possible even after a few bad results to try and confirm that they should be disregarded.

4.1.1 Locality in MCTS

Although a MCTS based approach has proven to be very successful it also has some downfalls such as that it can be slow to calculate [44] and this can contribute towards making the depth of the tree a lot less than would be preferred. Shepherd indicates two problems that can be encountered from game domains are imperfect information, and

high branching factors [45]. Both of these issues are evident when trying to apply MCTS to video games, as there is a plethora of unknown informational knowledge that will only become apparent as a game progresses. This is in contrast to a game like chess where all the possible moves and information regarding the game can always be seen. However, it's the branching factors that can prove to be the most problematic when it comes to actual implementation.

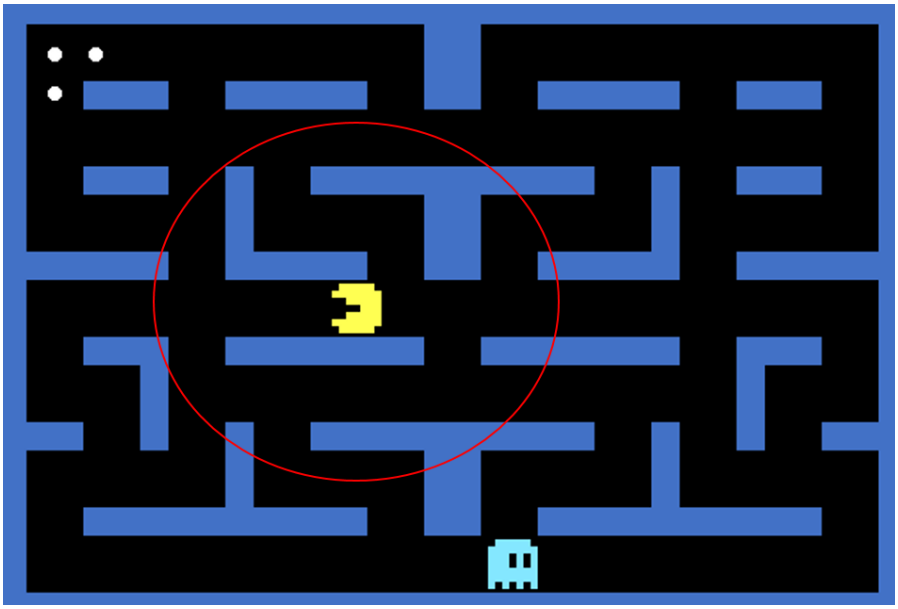

Figure 7 Area of locality in a game of Pac-Man. The red circle signifies everything that the MCTS rollout is able to reach.

To see this problem taking effect in a game based scenario we can look at the game Pac-Man (Namco, 1980) in Figure 7. We can see that the agent has removed all the pills surrounding it and there are no ghosts currently nearby threatening its safety. The obvious next move is to continue to corner of the level and consume the remaining pills but how does the agent know to do this? It doesn't, unless the rollout happens to expand long enough to reach that far across the game domain, but realistically this is never always 100% computationally feasible. The issue is that the agent will only be able to deal with "localised" information [47] which is highlighted by the red circle in Figure 7 surrounding Pac-Man. This is a hypothetical tree depth to which the game could be played out to. Anything outside of that area of locality will have no bearing on how the agent behaves so in reality the agent can't even see the pills at the other side of the map. Now the agent will seemingly get stuck in this section of the map with only random moves being chosen since no one move is seen as better than any other. The best way that the agent could possibly become "activated" again is if the ghost was to come near to the agent into its area of locality causing the agent to make more deliberate movements. This would hopefully end

up pushing the agent into the direction of the pills. A lot of games do not have anything to push them back into action and the game would come to an end at this point. This highlights the need for some form of goal based direction to be used to help stop problems like this occurring.

4.2 Learning

One ability of many GVGP implementations is the agents' ability to learn. This adds onto the concept of trying to recreate a human players more general intelligence, as this is a big part of what allows human players to switch between and play different types of games efficiently. Ideally with a learning approach an agent should become more optimised to play the game the longer that they're faced with it.

4.2.1 Genetic Algorithms

One method that has been used to try and achieve learning is with the use of genetic algorithms, one of which comes included within the GVG-AI framework. A genetic algorithm (GA) is a form of evolutionary algorithm that generates solutions based on natural evolutionary techniques like mutation and crossover [23]. It achieves optimisation by continually taking different sets of data then combining parts of them together to produce a genetically better solution over time. The GA used in the GVG-AI framework will take an initial population of individuals, each of which refers to a sequence of actions that are then heuristically evaluated [58], and the most successful populations will get evolved together. The action played out will be the first action from the most successful individual that was evolved.

GAs have been used in combination with video games before including when trying to play Pac-Man which is a game that has proved to be a focus point of exploring and teaching AI design for over 30 years [46]. An agent is still trying to be developed which is able to achieve the highest possible score possible in a game of Pac-Man, which currently has only been achieved by human players [64]. In an example by Alhejali & Lucas [25] agents would be given a specific game domain to play on and they would have to learn which was the best play style to complete the level. It's pointed out how well the Pac-Man agent is able to generalise his behaviour to different maps, which given the large range of

maps available in GVG-AI framework, it becomes evident why a GA approach may be of use.

Most game AI relies on heavily intensive rule-based systems, and a GA can be used as a way of trying to work around that. Injection into the evolutionary chain can be used as way of helping the AI learn as the game continues [24] without the need for endless conditional statements. This is especially useful compared to more traditional AI where a human player may be able to work out how to trick the AI depending on its predictable decisions that are hard-coded into the game. Using this method the AI would be able to keep learning and keep changing crucial behavioural parameters such as *aggressiveness* and *weapon choice* [63] in a game like Counter Strike (Valve, 1999) to be able to play the game much more efficiently. An issue with both this approach and the Pac-Man method is that they both use specific variables highly confined to their game type. Finding variables that will apply to all games in a GVGP context is more challenging.

4.2.2 Reinforcement Learning

An alternative approach which has been used to try and achieve learning is with the use of Reinforcement Learning (RL) where the aim is to reward the agent for actions that it takes that lead to a positive outcome [31]. In the case of the GVG-AI framework of games this could be done by noting that there has been a score increase in the game state. The RL method will then begin to learn what actions will result in the score being maximised for the game.

However, there are issues with reinforcement learning such as their inability to cope well with incomplete state information, scaling to realistic problems, and non-stationary environments [33]. These issues would obviously be problematic within a GVGP context, but they are also determent on how the RL is implemented.

Sutton and Barto [36] also highlight four key elements that can be applied to all applications of reinforcement learning that could in turn be applied to games within the GVG-AI framework.

- *Policy:* This dictates the way the agent chooses to act at any current time.

- *Reward Function:* This is the goal that the agent is trying to achieve. The reward will be for the agent reaching the goal point that it has been set. Depending on the result of the reward this can alter the policy.
- *Value Function:* Rewards are only interested in the immediate future of the game, taking each goal one step at a time; however the value function is concerned with what the end result of all this will lead to. For example in some games it is beneficial if the agent realises that it should be trying to collect resources so the level can be completed. Achieving this through normal gaming AI would be a straightforward process using a conditional statement but trying to achieve it for general gameplay remains a complex challenge.
- *Model of the Environment:* Models are used for planning in the environment and allow us to play out the action to predict what state the game will be in after. We have a model that can do this that can be accessed through the State Observation class using the advance method as mentioned in section 3.6.

Within branches of neural networks and machine learning one of the most popular methods of learning is supervised learning. Supervised learning requires data sets so that the agent it is able to learn from examples [36] and that can make it impractical for use within many areas of AI. This also applies to GVGP where it is important there does not have to be any past experience of gameplay for the agent to perform successfully. In this sense reinforcement learning applies itself well to GVGP in comparison because it requires no prior information at all to begin playing a game [67].

4.2.3 Exploration v Exploitation

One of the key learning problems that Sutton and Barto [36] mention is the trade-off that has to be made between *exploration* and *exploitation* when it comes any form of learning. Using the game Boulder Dash as an example, the agent would find a gem resource gave a score increase so it could be told to start prioritising this type of sprite over any other. However this means that the agent is also ignoring the possibility that other resources could be worth more points. This game and the other games in the GVG-AI library are simplistic enough that there are generally not more than one collectable type of resource, but it still raises an interesting dilemma for the agent to deal with. There is also the possibility that the agent could learn something which may be deemed as good, but may not actually be completely true. If the agent was to attack and kill an enemy it may gain points, but at the same time it may be more beneficial for the agent to actually ignore

enemies unless necessary because of the inherent risk with going near them. Finding a balance between *exploring* for better possible scoring options or *exploiting* the knowledge that has already been gained is a constant struggle in GVGP design.

4.2.4 Learning Scenario

To establish the level of complexity involved even within simplistic arcade games, the game Boulder Dash included in the GVG-AI framework will be analysed to show the types of decisions and scenarios that can occur.

Boulder Dash has a range of different information that the agent would ideally like to be aware of when making decisions:

- The agent starts underground and is effectively digging through terrain and making any square that they go onto moveable.
- Enemy sprites are trapped in certain sections of the map and should preferably not be released unless necessary. They cannot dig like the agent and can only move on the already moveable squares.
- The agent must collect a minimum amount of resources on the map - only at this point will the exit portal be active.
- Boulder sprites in the game domain will drop if the agent tries to move to the square beneath the boulder. This will either block the path of the agent if they move under a boulder from the side, or it will kill the agent if the agent tries to come towards the boulder from underneath.

What's evident is there is a multitude of different learning factors that need to be taken into consideration when trying to play the game. Unlike some other games where the agent may have to learn to collect certain types of sprites, this game requires more detail as the agent will have to actually learn, not only to head towards certain sprites, but also *how* to go towards them based on the other sprites that are surrounding them.

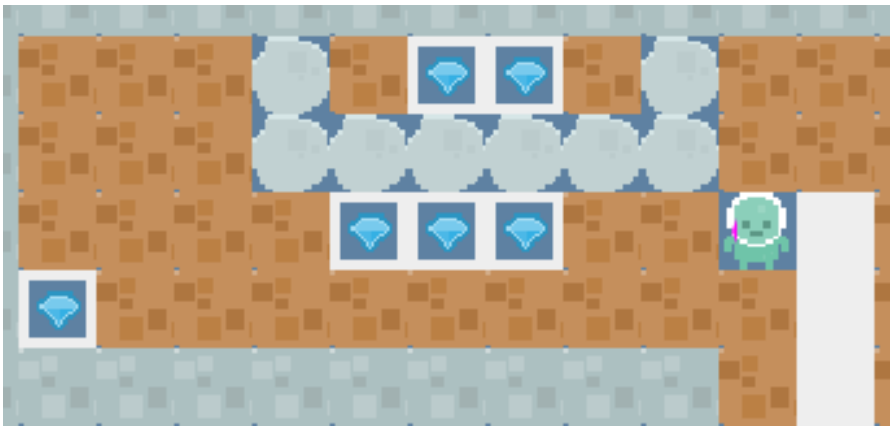

Figure 8 an in game shot of Boulder Dash from the GVG-AI framework

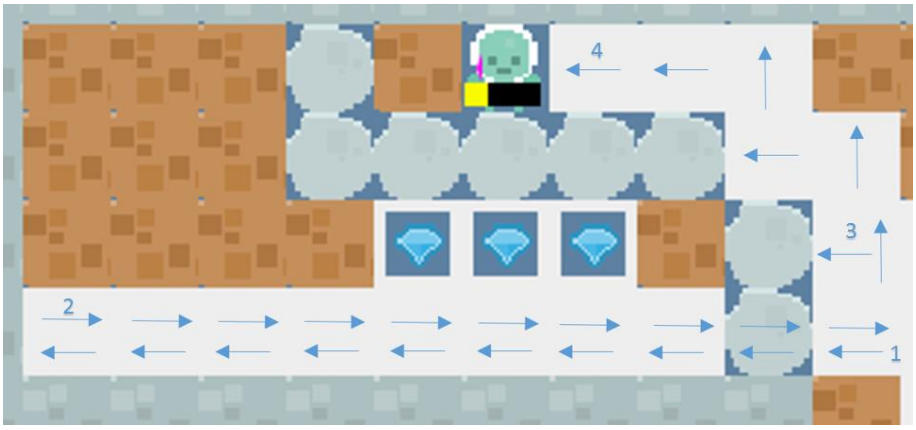

Figure 9 an in game shot of Boulder Dash from the GVG-AI framework. Arrows are used to show the way to solve this part of the level

From Figure 8 we can see the type of dilemmas that are faced by the agent as ideally it would want to collect every gem possible. In most cases it would be assumed the agent is going to head towards the three gems in a row that are closest to it but, as explained above, moving under a boulder cannot be done because it will fall down and block your path. The correct move for this section of the game is actually to ignore all three gems because they are impossible to collect and even for a human player this may require some trial and error. The best possible play is highlighted in the list below.

1. The agent must collect the gem in the bottom left hand corner of the screen at point 2 as seen in Figure 9.
2. The agent must move back to the starting position at point 1, then move up to point 3.

3. The agent must now try and move left to trigger the boulders to fall down while at point 3.
4. The path is now open to go up to point 4 and collect the two gems at the top of the screen.

It's important to note that the agent does not *have* to learn to achieve the steps listed to complete this part of the game. A MCTS approach may find its way around this but it would be a time consuming process and leads to a lot of uncertainties and sporadic movements on behalf of the agent. It would be much more efficient if the agent was able to learn to avoid these types of squares by default so any sort of MCTS evaluation process could just ignore these options to begin with.

4.3 Conclusion

In this section of the report some of the main methods of applying search techniques to GVGP have been discussed. Following on from this the design of a new agent for GVGP will be discussed, with focus being given on how to use the MCTS method discussed in coordination with a goal based approach.

5. Design of an Agent

In this section of the report details are given about how the GVG-AI framework was used to create agents that have a greater emphasis on goal orientation. The primary method being used is to implement a form of path finding for the agent, which was done using A* and also enforced hill climbing. The advantages and disadvantages of both these methods are discussed as well as a description of how the goals within the game levels are able to be identified.

5.1 Step Controllers

Before work began on developing an agent with goal direction, the one-step controller given within the GVG-AI framework was experimented with. The reason for this was to gain an understanding of how the framework is able to advance games, and evaluate different actions.

```
public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer) {

    Types.ACTIONS bestAction = null;
    double maxQ = Double.NEGATIVE_INFINITY;
    SimpleStateHeuristic heuristic =  new SimpleStateHeuristic(stateObs);
    for (Types.ACTIONS action : stateObs.getAvailableActions()) {

        StateObservation stCopy = stateObs.copy();
        stCopy.advance(action);
        double Q = heuristic.evaluateState(stCopy);

        if (Q > maxQ) {
            maxQ = Q;
            bestAction = action;
        }
    }
    return bestAction;
}
```

Figure 10 One-Step agent code taken GVG-AI library sample agent.

The one-step look ahead agent shown in Figure 10 quite simply advances the game one action ahead for each of the available actions. So in essence the agent would have to be one action away from another sprite for there to be any difference from this and a random agent. This process is highlighted below:

1. Each of the available actions for the agent are returned.

2. A copy of the game state is created. The game is advanced for each one of these actions in this copied game state.
3. A heuristic function is used to calculate a score value for the current state of the game.
4. After the state of the game has been evaluated for each action, the initial action which provided the best future state is returned to be played out in the actual game.

By using this agent it was then able to be modified with a simple adaptation to produce a two-step agent, which can be seen in Figure 11.

```
for (Types.ACTIONS action : stateObs.getAvailableActions()) {

    StateObservation stCopy = stateObs.copy();
    stCopy.advance(action);

    for (Types.ACTIONS action2 : stCopy.getAvailableActions()) {

        StateObservation stCopy2 = stCopy.copy();
        stCopy2.advance(action2);
        Q = heuristic.evaluateState(stCopy2);

        if (Q > maxQ) {
            maxQ = Q;
            bestAction = action;
        }
    }
}
```

Figure 11 Part of the code for a Two-Step agent.

The main principle here is that all the available actions are advanced like you would do for the one-step agent, but then each one of these actions are advanced a following time for the second step. This process was also extended using the same method so more controllers with additional steps were made. The results of this can be seen in the next section of this report.

5.2 Sensor Grid

The sensor grid allows us to bypass a lot of the issues with imperfect information that are heavily related to other games like poker [28] where information is hidden from the player. An equivalent for this scenario would be if the games had a fog of war element in place where the agent, or any human player, could only see within a certain radius of their location. Importantly, the agent and player can both see the whole game domain from the

start of any game in the GVG-AI framework so the agent is getting no advantage over a human player with building up this initial sensor grid.

The sensor grid allows us to take the observation data being held for each cell of the game grid and then store these as a list of nodes. Each node holds the following information about its grid square:

Variable	Value
Position	x and y coordinates
ID	Integer
Category	Integer ranging from 0 - 7
Type	Integer
Parent	Node reference (only applicable to A*)
Distance From Goal	Integer

Figure 12 Values held by nodes in the sensor grid

While the sensor grid is being created every single node in the game domain is looked through which allows us to build up a list of possible *explorable sprites*. To be classed as an *explorable sprite* a number of requirements must have been abided by.

Explorable sprite: A sprite in the list of game nodes created through the sensor grid that is (1) not an immovable sprite like a wall, (2) an empty node (i.e. a blank square with no type that can only be walked on), (3) the player, (4) any player created sprite like a missile or sword, or (5) a sprite that has already been previously explored.

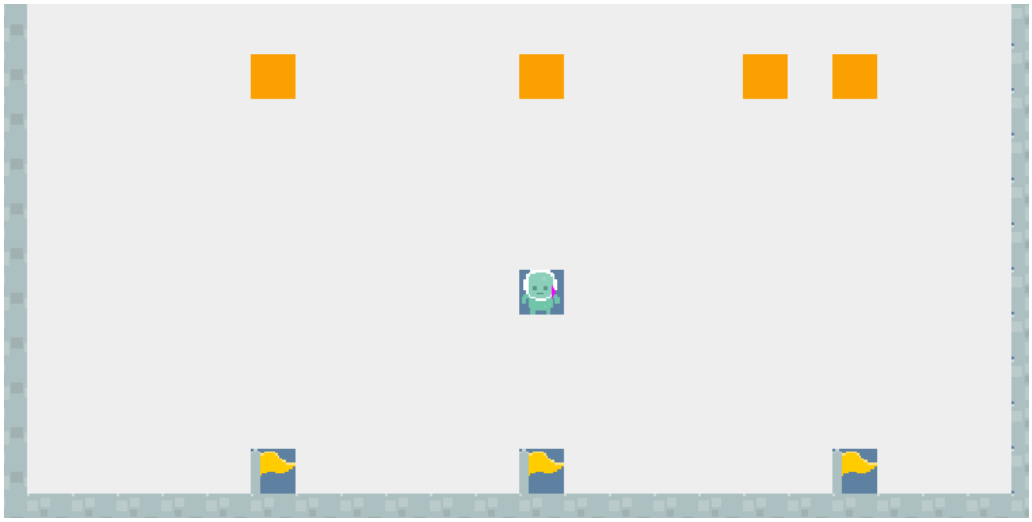

Figure 13 an in game shot of Missile Command from the GVG-AI framework

This means that the list of explorable sprites becomes a list of sprites that the agent is able to have *possible* interactions with, or in other words, a list of *goals* that the agent can head towards. Using the example above in Figure 13 for Missile Command this would mean there would be a list of seven different possible explorable sprites. These comprise of four different “chaser” enemies and three different “city” sprites.

5.2.1 Applying the Sensor Grid to Goals

Theoretically, in a more deterministic game world, one by one each of these explorable sprites would be set as the next goal until they are each successfully explored and the game would then hopefully be concluded. Video games are not normally so straightforward and there are a lot of non-deterministic issues that need to be taken into account.

1. New explorable sprites may spawn in the game domain that were not originally found.
2. Conversely, current sprites could disappear through no interaction with the agent causing the agent to head towards no goal.
3. Some sprites may be able to unpredictably change their type.
4. Sprites may have no interaction with an agent until a later point in the game so they will need to be explored multiple times. One of these trigger events takes place in the Zelda game where the end game portal will not become active until the player has first collected the key to open the door.

5. Given the random nature of some of the sprites in the games it is not guaranteed that they will be explored properly when first checked. By the time the agent gets to the sprite location it may have moved out of range of the agent.
6. If two sprites are close together the agent might go towards a different sprite rather than the one that was initially intended if using MCTS. It could then be registered that the wrong sprite was actually explored in the game.

Point 3 does not actually apply to any of the games in the training set of the GVG-AI framework, since no sprite is able to change their type. For example, in the game Butterflies when the agent collides with one of the butterflies the sprite transforms into a still non-moving sprite of a different type, which can't be interacted with. This however is not registered as the sprite changing type, but a new sprite entirely. This can be seen given by the fact it's given a new ID which would not be the case if it had merely changed type.

Despite not applying to the training set of 10 games it is still important to take this consideration into account in the case of the framework being extended, but also because the unseen games in the validation set of games on the submission server may include some of these features.

5.3 Path Finding with A*

The A* algorithm was first put forward almost 50 years ago[15] where it was described how it could be used for graph traversal, including that of working out a way through a maze which now has an obvious link with how it can be used in video games. However, it wasn't until 1996[20] when the video game community first started to see its potential and began to use it. The A* algorithm allows for the heuristic determination of the minimum cost paths [65] for an agent to follow. It does this by looking at the domain area like a grid and referring to each square on the grid as a node, each of which has their own heuristic value dependent on the goal location. We can see from Figure 12 that each node stores its distance from the current goal which is used to work out this heuristic value.

Prior to A* two of the most used graph traversal techniques were breadth-first search and depth-first search. The difference being that breadth-first search would go down a graph searching every sibling node before moving on to the next level of nodes whereas in

depth-first search a node and then all of its ancestor nodes would be explored before backtracking up the tree for another node to expand [70]. These methods will find a solution but are inefficient because the solution that they find will not necessarily be the best one. A* is known as a Best-First approach, and what makes this approach so useful, especially when applied to video games, is that they take into account domain knowledge to help guide the search[20]. This means it now becomes possible for the agent to find the best possible path to its location.

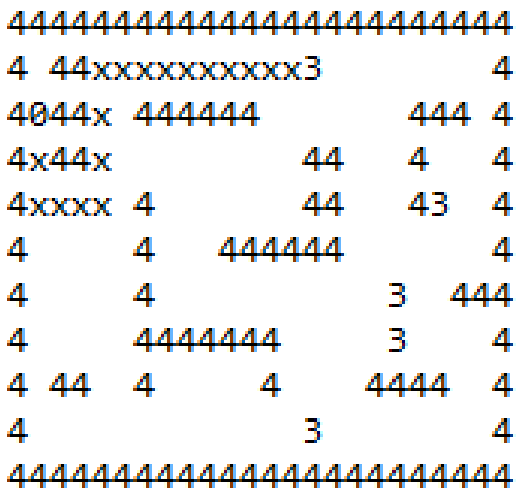

Figure 14 an image produced via the sensor grid showing a path for the agent in the game Chase

Using the sensor grid an A* algorithm is able to reference the grid as a map of nodes and thereby calculate the best possible path to the goal as seen in Figure 14. The game domain has been outputted displaying each nodes category. Category 4 nodes are walls and are therefore ignored, whereas category 3 nodes are the sprites that the agent has to chase down. The player is represented as category 0 across all games. The sensor grid has also been modified to show the result of the outcome of the path found by the A* algorithm, each step of the path being represented by the letter ‘x’.

The A* code used in this implementation is a modified version of source code made freely available online [66]. The main adaptions that had to be made were modifications to what information the nodes should hold (as listed in Figure 13), being told how to avoid sprites of certain type, and general modifications so that it was compatible with the sensor grid being used.

This algorithm has added advantages aside from letting the agent navigate successfully around any domain. (1) The speed of the gameplay should be vastly improved compared to the random sporadic movements of the Monte-Carlo approach, allowing the agent to complete the game in less game cycles, and (2) there may be added score benefits to the agent finding normally hard to reach sprites on the map.

5.3.1 Computational Issues

It would look like A* should be a good fit for the games in the GVG-AI framework, but there is one very limiting factor that makes its use questionable. Although it was previously mentioned that A* should help speed up how quickly the agent plays the game, which may be true in terms of the number of game cycles used, it definitely is not true in terms of computation. A huge amount of effort has gone into making accurate path finding, which is still a big issue in modern AI design [16] with the main issue being that it's incredibly hard to achieve advanced movement without using too much computational power. Although the power that can be used has increased over time with technological development, games themselves use increasing amounts of power on graphical effects and more objects being displayed on the screen. The additional objects makes the path finding job even more computationally intensive as paths have to be reworked for their unpredictable nature of movement. A yearly competition is run to develop AI for Starcraft: Brood War (Blizzard Entertainment, 1998) [17] which is a real time strategy game where the user must control up to two hundred different units. The game involves a huge amount of path finding since the units available will be split up in to separate groups and not only have to navigate around the map but around each other as well so they can fit up narrow slopes and also engage with enemies. To add to that, there can be upwards of 4 players in a game so the amount of possible permutations involved becomes immense. For each path that is created for every unit there will be thousands of calculations performed and this in most cases becomes infeasible. Even 16 years after the games release perfecting the AI in this game is still a challenging problem partly due to the complexity of the path finding issue. The need to keep this process computationally low is also added to by the fact games are generally given a low percentage of time per game cycle to do their calculations [18].

There have been different interpretations of the A* algorithm such as IDA*[19] (Iterative Deepening A*) where the memory usage is kept lower than traditional A* however the

general principles behind the process remain unchanged. Another method created by Botea and Muller called Hierarchical Path-Finding A* (HPA*) [21] also tries to solve this problem by splitting a game domain up into sections and calculating parts of the path separately. They claim that their method can reduce the effort required to calculate a path by up to ten times, which would obviously be very beneficial, but unfortunately problems do still remain, especially with GVGP, such as trying to deal with the randomness of unseen levels which is not something HPA* can easily cope with. The computational effort of both of these methods is also still dependent to some extent on the size of the game domain, which is not ideal.

5.4 Path Finding with Enforced Hill Climbing

Due to this problem with the A* algorithm a less computationally intensive method called enforced hill climbing (EHC) was also designed with a different agent. What makes this method useful in computational terms is that the time it needs to perform calculations will remain constant regardless of the map size. Instead of calculating an entire path for the agent, it now "pushes" the agent one square closer to its destination one action at a time. The process for this method is highlighted below.

1. *The Manhattan distance is calculated between the position of the player and the position of the chosen goal node.*

```
int manhattanDistance = Math.abs(playerLocation.X – goalLocation.X)
+ Math.abs(playerLocation.Y – goalLocation.Y);
```

2. *On each game cycle every square surrounding the agent will be evaluated and the one with the lowest Manhattan distance will be the next chosen square.*
3. *This process continues until the agent reaches the goal square.*

What this method is essentially doing is removing the complexities of having to save details about every node in a path and instead applying a movement bias on the agent towards a goal.

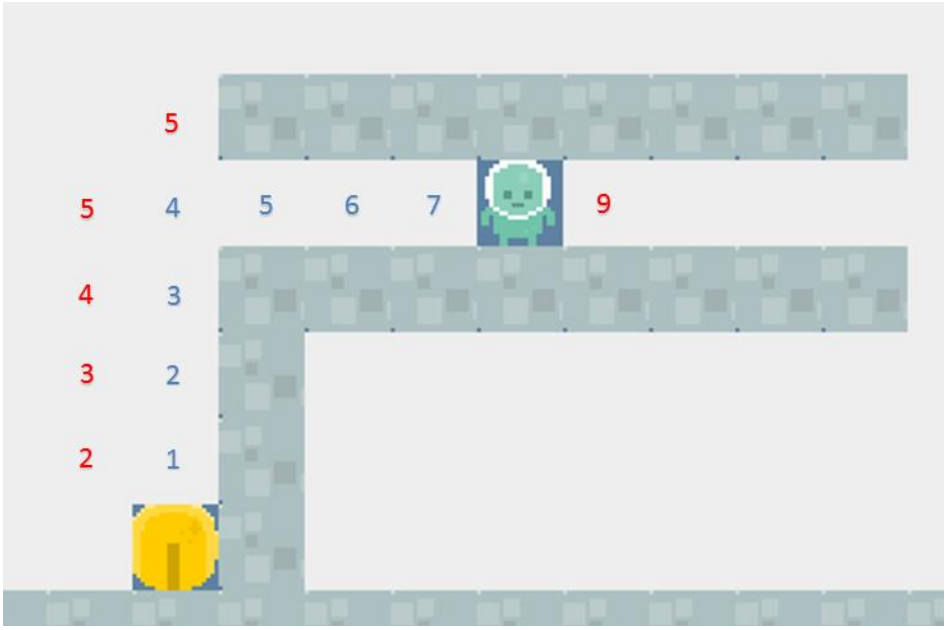

Figure 15 an example of the EHC process of finding a path

The way this process works in a game scenario can be seen in Figure 15. The numbers represent the Manhattan distance of each square that has been evaluated. The blue numbers show the path that will be taken by the agent as they have the lowest Manhattan distance, whereas the red numbers show the squares that will be evaluated but their Manhattan distance will be found to be larger than another possible square.

However there are obvious issues with this approach that will become apparent on certain game levels. The agent is not guaranteed to take the best possible path to the goal. Using the example in Figure 12, going to the left is obviously best path, but this is not always the case. On some occasions going slightly further away from the target is necessary and is part of the best possible path. This issue can be worsened by games with rooms or corners as it's very easy for the agent to get stuck if the goal is on the other side as the agent will not realise it has to go further away from the target before it can reach the goal.

This problem was able to be lessened to an extent in this implementation because if an agent recognises that it is trying to move into a wall it will then make an attempt to stick to the wall and go around it. So if we take the example shown in Figure 15, the agent will check the square below and above it to see if it is moveable, but on recognising that they aren't it will calculate the Manhattan distance for going left or right and choose one of these actions instead. In this case the agent would continue moving to the left towards the coin until it can find a way to move down. This does not fix the problem entirely, given that

the agent can still get trapped in a corner of a room, and can still struggle to get around walls depending on where the goal is, as depicted in Figure 16.

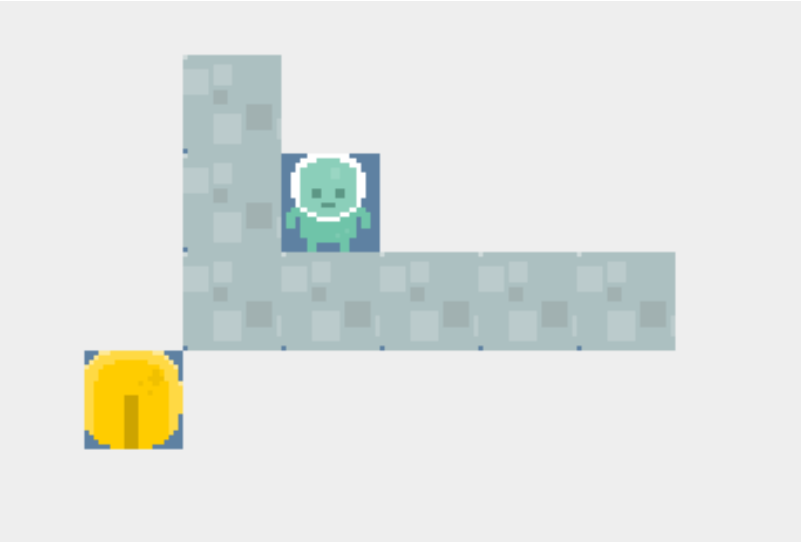

Figure 16 EHC issues with manoeuvring around walls

An advantage of using EHC is that every step of the path is created separately which means it is much easier to adapt and change the path. The agent is told when going towards a goal in EHC mode that it cannot move on any other explorable sprites on its way. This is to stop the agent running in to an enemy or any other damaging sprite by accident. However, in A* mode the path can only be calculated once at the start of the path due to the computational issues so if an enemy walks across your path then the agent will get hit.

5.5 Switching between MCTS and Path Finding Mode

The problem with just using path finding with agents is that there is an assumption that the agent wants to land on the goal. This is not always true, as the goal could be any multitude of things from damaging terrain to enemy sprites. Therefore a method is proposed to use MCTS in coordination by switching between it and path finding at optimum points. This method will be referred to as *switching modes* and is applied to both A* and EHC and should help solve the issue of locality that has been described with previous implementations of MCTS.

The MCTS code that comes with the GVG-AI framework is also the same code being used in both of the goal oriented agents being designed. This will aid in drawing a better comparison with how much the goal based method is able to improve the results.

	A*	EHC
Switch Distance (squares)	3	3
Time allowed in MCTS mode	1500ms	3000ms
Time allowed in path finding mode	NA	2000ms

Figure 17 Key variables in switching agent modes.

The switch distance of an agent is how close an agent will get to its goal before going into MCTS mode. It plays an important role in maintaining the safety of the agent in its environment. The result being that if the target is a collectable resource then the MCTS will guide the agent on to it, otherwise the MCTS will be able to take control and steer clear of it. Importantly, the area of locality the agent has when it switches to MCTS mode will leave it in range of the MCTS area of locality.

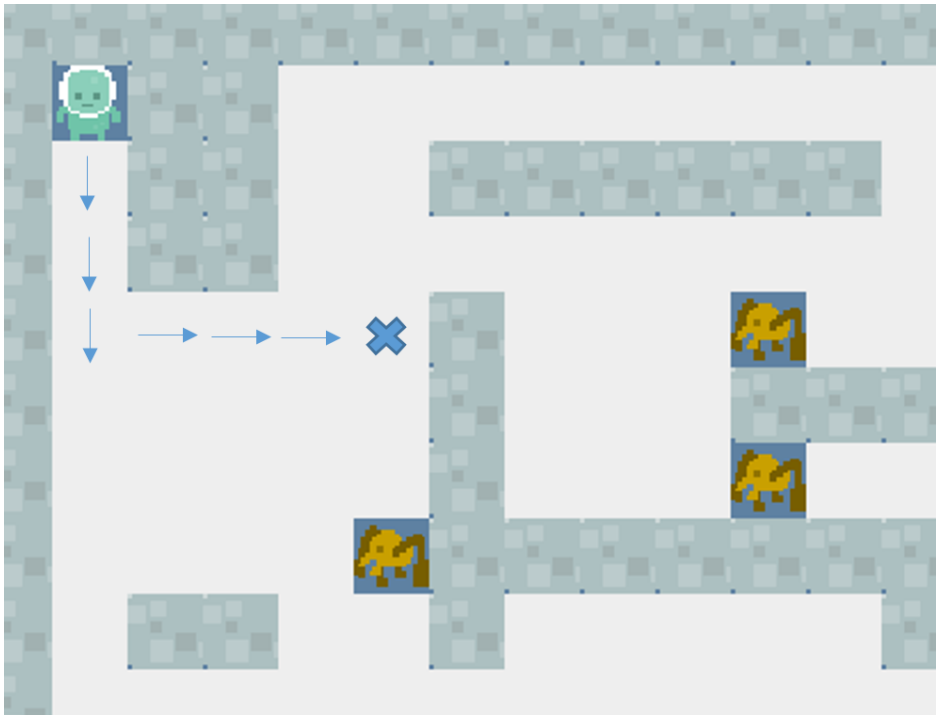

Figure 18 An in game shot of Chase from the GVG-AI framework. The blue cross marks the point at which the agent will switch into MCTS mode.

Experimentation was done with the values that produced the best results possible, although it should be noted that further testing on additional games outside of the training

set available may provide differing results. It was found if the agent was put into MCTS mode with the agent two or less squares distance from a goal and that goal was an enemy sprite then there would often not be enough time for the agent to react properly and the agent would collide with the sprite causing an end to the game. Putting the value to four or more meant that if the agent was near a favourable sprite the agent would have a tendency to take a long time to reach the sprite or sometimes miss it completely and start moving in a different direction. Therefore, a value of three seemed like a happy medium between keeping the agent safe, and allowing the MCTS to guide the agent towards favourable sprites as quickly as possible. The switch distance and the reasoning behind the switch distance remains the same for both EHC and A*.

5.5.1 Timers for Switching Modes

After guiding the agent towards its goal the next challenge comes from controlling how long to stay in MCTS mode, and then moving on to the next goal. As we can see from figure 17 this differs for the respective agents.

The MCTS is very quick to take affect when the agent is within range of a sprite which is why we can limit the time to only 1500ms when using A*, knowing that the goal is *likely* to get explored. However there is a need to stay in MCTS mode slightly longer with EHC mode to try and combat the issue highlighted in Figure 16 with the agent getting stuck in corners of the map. How this effectively works is that if the agent gets stuck in a corner then the additional time given to MCTS mode should hopefully be enough to allow the agent to have moved out of its current location, due to the somewhat sporadic movement when in MCTS mode, however this is not always 100% successful.

The time allowed in EHC mode is quite small as the agent moves incredibly quickly when in this mode due to the lack of computation time. There is rarely ever a situation where the 2000ms would be exceeded trying to reach a goal. The purpose of this path timer is that if the agent surpasses this time we can make an assumption that the agent has got stuck at some point in the map so the MCTS can be activated in an attempt to help. Even if the agent does switch out of path finding mode to early, it will resume the path again after the time spent in MCTS mode, so the effect is mainly negligible. There is no timer in place like this for the A* approach because we know the agent has already found the best

path. If for some reason a section of the path becomes obstructed then a new path will be calculated.

5.6 Choosing Goals

Now that methods to find pathways to goals and also how to explore them have been discussed more detail will be given to how the goals are being chosen. Below is listed the overall process of selecting goals for an agent. This process is repeated until the game is won or lost.

1. *Create a sensor grid with a list of explorable sprites.*
2. *Set the new goal to the closest explorable sprite.*
3. *Create a path to the goal.*
4. *When within switch distance change to MCTS mode.*
5. *When time in MCTS mode is more than the time allowed the goal has been explored, now switch back to path finding mode.*

One issue that was previously highlighted was that goals may need to be explored multiple times by an agent. However some sprites should only be explored once, so how do we try and take this into account? A solution used in the implementation allows us to visit past goals to see if they have now become active. If a new goal is needed sprites that have not yet been explored will always be prioritised. However, if there are no explorable sprites left then we can make an assumption that we need to start looking through past explorable sprites. An example of this process can be seen below.

```
IF a new goal is needed
    SET current goal explored = true
    REMOVE current goal from explorable sprites list
    IF explorable sprites list size = 0
                    FOR EACH sprite where explored = true
                        SET explored = false
    END IF
    GET closet sprite from list of explorable nodes
    Goal = closest sprite
END IF
```

This process essentially refreshes the list of possible explorable sprites so the agent will continually loop round rechecking old sprites if the game is still running. This fixes many of the issues mentioned in section 5.2.1 with the MCTS with regard to it occasionally missing sprites, or, with the example of the Zelda game, certain sprites not being active if you haven't explored a previous node first.

5.7 Prioritisation

For the solution in this report, since goal direction is being focused on, complex learning abilities will be beyond the scope of what our agent will be able to achieve, however the agent will have a form of learning to help to try and identify goals in a better order.

From the state observation class that was mentioned previously in section 3.4 there is the ability to return an event history from the game. What this holds is all of the different interactions that have occurred during the course of the game between different sprites and the agent. This gives us an opportunity for applying a form of reinforcement learning with the agent as we can analyse the state of the game before and after sprite interactions and learn if the agent should prioritise those interactions over another.

When choosing the initial goal for the player to head towards the agent looks at the game domain as a human player would with no prior information on how any of the sprites will influence the game. However, we want to be able to tell the agent to head towards resources that is has managed to establish will improve the game state. This is done using a process where the game score is continually checked to establish if there are certain sprites types that have more value than others. The process for this can be seen below.

```
REPEAT on each game cycle
        IF current game score > previous game score
                IF score increase amount > any saved past score increase
                amount
                        GET last sprite entered in event history list
                        SET preferred category to category of returned sprite
                END IF
        END IF
UNTIL game is ended
```

What this means is that when the next goal is being attained there will first be a check to see if a prioritised sprite category has been set. If so, instead of deciding the goal based on how far away the agent is it is, now a sprite will be picked that is of the same category as what has been prioritised. The closest sprite of that prioritised sprite category will be set as the next goal.

5.8 Conclusion

In this section of the report we have covered how an agent can be designed to take into account goal orientation by using a grid to explore all the possible goal options in the game domain. Following on from this the results of this solution will be looked at to gain a better understanding of how this method is affecting the behaviour of the agent.

6. Results

In this section it will be discussed how the ideas mentioned previously in the report have been implemented within the GVG-AI framework. Of particular interest is how the different types of agents already in place within the framework compared to the more goal oriented agents. It can be difficult to effectively evaluate the successfulness of a GVGP agent given the range of different factors involved [51], such as the type of game being played, the amount of time the game is being played over, and the description of the game. Due to the freshness of the GVGP research there are also no real benchmarks for comparisons to be made. Thankfully the sample agents that come with the GVG-AI framework allow for a good comparison to be made with the agents designed in this report.

For the results shown in this section of the report all of the agents are run once on each level of every game. There are five different levels for each game so each agent will have five attempts at playing any one game, totalling fifty game attempts in any one evaluation cycle. This method is what is used to test the agents on the GVG-AI submission server, so the same method was decided to be used for evaluating all the other results reported for a clearer comparison.

6.1 GVG-AI Framework Agents

The GVG-AI framework comes with four sample agents that can be used to play any of the games available. In particular the MCTS agent will be used for more heavy comparison here since it is the same MCTS algorithm being used in the both of the path finding solutions. The four sample agents are listed below.

- Random
- One step look ahead
- Monte-Carlo Tree Search
- Genetic Algorithm

6.1.1 Results with Sample Agents

The agents are evaluated in Table 1 by looking at the score the agent achieved on each level and also the number of wins. It's important to note that the scores achieved in games are not directly comparable to other games as they have different scoring systems in place. Five points for the game Portal would be a maximum possible score (1 point for completing each level) whereas a game like Survive Zombies does not have an obvious maximum score as the agent can keep collecting coins until the time runs out. The agent is also awarded negative points for dying or for allowing vital resources in the game domain to get destroyed by enemy sprites, which is only possible in some games. This is why the total score across all the games has not been totalled as it cannot be seen as a clear comparison. To compensate for this the total number of wins for each agent is shown to give a clearer estimate of how an agent performed overall. For the sake of brevity the games are also referenced to by number in the same way that can be found in section 3.2.

Game No. :		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
MCTS	Score	36	8.2	27.6	1.4	-0.8	-1.6	0.2	0.2	8.8	-0.6	
	win rate/5	0	0	2	0	2	1	1	0	3	1	**10**
GA	Score	50.2	15.8	24.8	6	-1.2	2	0.4	0.6	11.2	1.4	
	win rate/5	1	1	5	2	0	2	2	0	2	1	**16**
1 Step Look Ahead	Score	39.6	0.8	16	0.4	0	0	0	0	6.2	-0.6	
	win rate/5	2	0	0	0	0	0	0	0	1	0	**3**
Random	Score	22	3	18	1.4	-1.6	1	0	0.2	4.6	-0.4	
	win rate/5	0	0	2	0	0	0	0	0	0	0	**2**

Table 1 Results of sample agents

Unsurprisingly random was the worst performing agent, and as the name would suggest it generates a list of all the possible actions available to the game and will choose one at random. Its main use is as a benchmark to use against other controllers to see how well other agents are playing the game. Any agent should be outperforming random to at least some extent for it to be seen as successful.

What is also noticeable from the results in Table 1 is that despite MCTS and GA being much better solutions on paper their results are not particularly overwhelming in a lot of cases. Results are definitely improved, but it's easy to identify that these methods were not able to solve some of the problems set before them as efficiently as required.

6.1.2 Behavioural Observations

It's important to not just look at the scores the agents achieved but to observe the agents as they are playing the game to get a better understanding of their behaviour, because a score cannot tell the whole story of how well an agent is performing. One agent could be almost completing a game on every attempt whereas another might be failing instantly but they could still be scoring the same amount of points. On closer inspection it becomes evident why the MCTS or GA are not successfully able to complete some of the games:

Chase: The enemy sprites move away from the agent leaving the agent stranded in the middle of the screen unable to identify any nearby targets. The GA performs quite well here as it doesn't suffer from the same sort of issue that the MCTS does where the radius will almost quite literally stop the agent from moving. The GAs movement is a lot more exploratory so it does on a lot of occasions end up winning the game.

Missile Command: The enemy sprites come down towards the ground at a fast rate and most are missed before they collide with the cities below. The agents do have time to react but normally end up diverging off in the wrong direction towards a corner of the map where there are no sprites to interact with at all. This is because it takes a small amount of time before the sprites get within range, so the agent might already have chosen to go a random direction which will lead to the enemy sprites being missed.

Portals: Both agents manage to move around the maps very quickly, and are able to navigate between missiles much faster and more effectively than any human player would manage. It's these missiles that allow the agents to travel so well as there is almost always something nearby that needs to be avoided and this will lead the agent to, by chance, finding their way to the end portal on some occasions. There is no sort of heuristic reward for going through portals so the agents will only find their way to the goal by chance. Portal also has an extremely large amount of spites to avoid, and the agent will quite often end up running into one of the missiles as their movement options deteriorate.

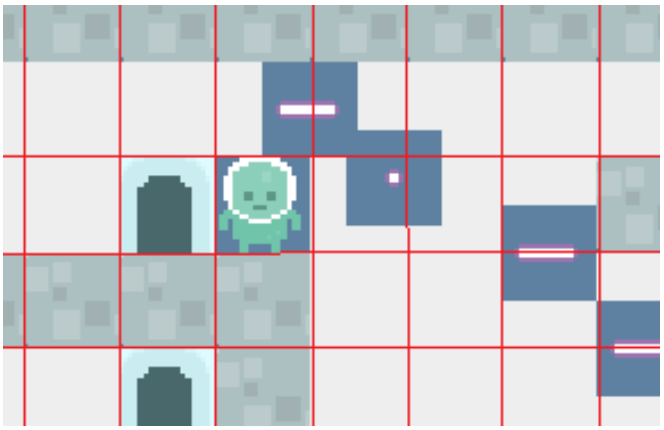

Figure 19 An in game shot of the game Portal showing the movement difficult

Another movement issue is pointed out in Figure 19 where the game grid has been emphasized. Whereas the agent must move between the red bordered squares for each movement action, enemy projectiles do not have to abide by this rule. As you can see a single projectile can occupy four squares at the same time, landing on any of which would trigger the end of the game for the agent.

Sokoban: The controllers quite simply are not able to distinguish that boxes need to be pushed onto certain squares. An improved game state is only given when a box has been pushed into a portal, but this could require upwards of 10+ actions and the controllers are not able to predict this. The MCTS will often get stuck in a section of the map and both the MCTS and GA will end up on a lot of occasions moving boxes into the corners of the levels which will mean that the game cannot be completed. It is often the case that even before the agent would have a chance of working out that the boxes being moved into portals would be beneficial that they have already moved them into unwinnable positions.

Survive Zombies: The agent is not able to identify that it should be trying to find resources, and so will struggle to increase its score unless the coins are nearby. More often than not the end result will be the agent getting trapped in a room and eventually surrounded by zombies. Ironically the way the agent gets trapped in a room can sometimes lead to its success in surviving the game as the zombies are not smart enough to reach certain points on the map.

Zelda: The sporadic enemy movement proves too challenging for the agent to avoid and so most runs result in the agent being killed. There is also the high possibility of the agent getting trapped in a room unable to escape. The odds of the agent being able to successfully find its way to the two separate key points on a level required are slim, although not impossible given that the map sizes are not overly large.

6.2 Comparison of Step Controllers

Since there was a one-step controller in place a two-step controller was developed to see just how much difference one extra step in advance would make to the results.

Game No. :		1	2	3	4	5	6	7	8	9	10	Total
One-Step	Score	39.6	0.8	16	0.4	0	0	0	0	6.2	-0.6	
Look Ahead	Win rate/5	2	0	0	0	0	0	0	0	1	0	**3**
Two-Step	Score	46.4	1	22.8	0.6	0	-0.8	0	0	12.8	-1	
Look Ahead	Win rate/5	4	0	0	0	0	0	0	0	0	0	**4**

Table 2 Results of step controllers

Now the agent is able to see two moves in front of where it would be going, and although this may seem fairly negligible the results do show a marginal difference with the two-step controller outperforming the one-step version on a lot of the games available, when comparing only the scores that are being achieved in the individual games.

It's important to realise that the agent has to be one action away, and not only one square away. If the agent is firing a missile from across the screen and it connects with an enemy this is still just one action ahead. This can be seen when you look at the results of the Aliens game where the one step-ahead agent does comfortably outperform the random agent because it recognises it should, more or less, continue shooting upwards on most occasions. The evidence of this can also be seen when comparing the results of the step controllers against the MCTS and GA where for this reason they perform just as well as those controllers on this game.

With the two-step agent performing marginally better then why can a 100 or 200 step agent simply not be made to keep increasing the success rate? Experimentation was also done up to a five-step agent to try and test this theory. This works as you would expect

with the agent now calculating five steps in advance. Unfortunately the number of calculations that needed to be run were far too high even for just five steps ahead causing a huge amount of delay. Games that would previously last under a minute would now be taking upwards of 30 minutes to complete. This effectively made any results by this solution obsolete since they hold no value towards an efficient or even feasible solution. It does only prove to highlight why the approach of evaluating every possible branch of a tree cannot be achieved and why methods such as MCTS are a necessity.

6.3 Results with Goal Oriented Controllers

In this section of the report the results previously discussed with those of the goal based controllers that have been designed are compared.

6.3.1 Table of Results

In Table 3 we can see the same results for the MCTS seen in Table 1 but compared this time against the results of the EHC and A* agent.

Game No. :		1	2	3	4	5	6	7	8	9	10	Total
MCTS	Score	36	8.2	27.6	1.4	-0.8	-1.6	0.2	0.2	8.8	-0.6	
	win rate / 5	0	0	2	0	2	1	1	0	3	1	**10**
EHC	Score	30.2	5.2	24.8	5.2	-1.6	-1.6	0.2	0	16	0.4	
	Win rate / 5	0	0	4	4	0	0	1	0	3	2	**14**
A*	Score	36	6.2	27.6	7	-0.8	-1.6	0.2	0.2	8.8	-0.6	
	win rate / 5	0	0	5	5	0	1	1	0	1	2	**15**

Table 3 Results of goal oriented controllers compared to MCTS

What we can see is that the results show little difference with both methods excelling on the same games. However, there is a marginal difference with the A* method in that it achieves perfect scores on two of the games whereas the EHC method does not. This is due to the issue that was discussed previously with the agent on occasions getting stuck, making the A* method much more consistent as it should always manage to achieve perfect results on these games. On the other hand the EHC scores will vary to a small degree over successive runs.

6.3.2 Approximate Computation Times

Here a quick rundown of the computation times involved for the agents are given to show why the A* method was found to not be a computationally viable option.

Process	Time
Creating the sensor grid + path movement in EHC	20 milliseconds
Evaluation with MCTS	30 milliseconds
Creating a path with A* (5 squares):	20-30 milliseconds
Creating a path with A* (10 squares+):	200+ milliseconds

Table 4 Computation times

The A* approach improves greatly on what has been achieved with MCTS in relation to score on a selection of the games, but unfortunately these results cannot be submitted to the GVG-AI competition because of the large computation times required, as on many occasions it greatly exceeds the 40ms time limit that is enforced by the competition.

It's evident just how quickly the computation times get unfeasibly hard to deal with as the size of the path progresses. The amount of time to create a path also varies a lot depending on the space that the user has around them. On a game such as Missile Command there are almost no walls or pathways to navigate around which means many more path possibilities have to be calculated.

Although the A* method was not able to run within the competition rules, the EHC method was able to run very smoothly leaving up to 20 milliseconds of time to spare from the 40 millisecond time limit at the end of most cycles when in EHC mode. The time being used would increase slightly when going into MCTS mode, but in neither mode would the 40 millisecond limit be surpassed.

6.3.3 Behavioural Improvements

Table 3 shows how the EHC and A* method were able to improve the most of the scores achieved with the MCTS, but to any great degree. A score, as mentioned, can sometime be an unfair representation of an agent however, so again the behavioural changes of the

agent will be looked at now that a goal based approach has been implemented to see what behavioural improvements have been made.

Aliens: As expected a goal based approach makes little difference in this game because the game is designed in a way that the actions available don't allow you to move towards any other sprites, so you are trapped at the bottom of the screen. As such this type of game was not really the focus of the approach used, as it is the only game where an agent does not have full movement. The goal based approach is actually a slight hindrance since it goes into path finding mode and the agent is unable to shoot until it switches back to MCTS mode.

Boulder Dash: The agent will now move around the game domain more and be able to identify resources very quickly. However, the levels are all extremely hazardous and without there being some adjoining method of allowing the agent to learn to protect itself, a goal oriented approach does not prove overly effective in the actual completion of the levels. On most occasions the agent will end up moving into dangerous situations that will lose the game.

Butterflies: The goal based approach shows a marked improvement on the previous behaviour with the agent immediately identifying and moving towards the target sprites one at a time. The movement is quick enough that on most occasions the agent does not get stuck (although this still can be an issue for EHC) and is able to take out all of the enemy sprites before more appear, and thus winning the game. The prioritisation technique in place stops the agent from getting fixated on the mushroom sprites that have to be protected and so only going after the butterflies on the majority of occasions.

Figure 20 an in game shot of the game Butterflies from the GVG-AI framework

Chase: This game shows the biggest improvement, as the MCTS proved very poor with the enemies immediately moving beyond its viewable range. Now the agent follows them and activates MCTS when it is within a range where it can be effective. The only negative point is that the maze like levels do lead to the agent getting stuck on fairly regular occasions using EHC mode, slowing down the level completion.

Frogs: The goal based approach is not able to achieve any success in this game. There are a multitude of different sprites that need to be avoided which means the goal based approach will almost always lead the agent into a situation that it cannot recover from. The type of slow patient play required in this game is just not suited to the speed that the goal based agent moves around the level.

Missile Command: There is little improvement in this game from a goal based approach despite the fact that it intuitively would seem like it would benefit a lot from this method. The main reason being that the agent will push its way towards the enemy sprite, but the sprite will continue to quickly move down the screen giving the MCTS mode very little time to hit the enemy before it goes out of reach. The chances of the agent following the enemy in MCTS mode are very slim given that this game has wide open spaces that allow for more movement options, unlike Chase where the levels are more maze like, with corridors giving the MCTS very few options. This problem is exacerbated by the fact you do not have to just move over to the enemies to kill them but instead have to use a different action (use a weapon) when next to the enemy. This makes it harder for the MCTS to correctly evaluate the best set of actions, and it's likely if this game was similar to butterflies where

only contact was required with enemies the success rate on this game would have increased massively. The path finding is also a hindrance in this case because it will evaluate the cities at the bottom of the screen as possible explorable sprites, when in actual fact there is absolutely zero benefit to the agent going there. All this accomplishes is allowing the enemy sprites to move closer to the cities before the agent can begin trying to go towards them instead.

Portal: When playing portals a similar issue arises to that of Frogs, with the agent being told to head towards sprites that are missiles and on a lot of occasions causing the agent to get killed. It is the case that if the level was purely a maze with portal entrances and exits the agent would be very efficient, but the agent is unable to identify when missile sprites are active on the screen that portals are the only sprites it should be taking into consideration. Without some heuristic benefit being given to moving through portals this will be very hard to achieve.

Sokoban: There is some small improvement in that the agent itself will explore every single box on the map instead of getting stuck at certain points when no box is in range, which improves the chances of scoring but not greatly, since most boxes will almost always get pushed into immovable locations. This is partly due to the agent not being able to correctly identify the long list of actions required to achieve scoring a point. An example of how this game could be made a lot easier for the agent is if some sort of scoring benefit was given every time a box was moved closer to a portal. This is obviously unrealistic, but it highlights the issue of games where scoring incentives cannot be seen by the agent. Another issue here is that the map sizes have been made purposefully small so that incredibly precise movements are required by the agent. It's likely that with maps that had more margin for error this agent would be able to outperform the MCTS agent.

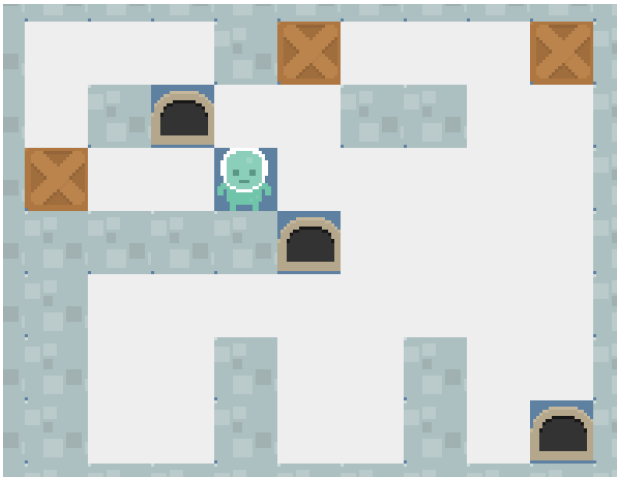

Figure 21 A common issue on Sokoban. The agent now has no way to complete the level

Survive Zombies: There is a large improvement in this game for the EHC in terms of score which has a large part to do with the prioritisation that was put in place. On most maps there are a large amount of coin resources which make it likely that the agent will come across one and identify that it should continue collecting these resources. It then tries to ignore the zombies in favour of all the coins that are scattered throughout the map. However, the game is very quick in spawning new enemies and the agent is not smart enough to avoid locations where it is likely to get trapped, which will lead to the game ending. Noticeably the A* approach fairs far worse on this game than EHC, which is in a large part due to the EHC approach being able to adapt its paths to avoid zombies, which the A* approach cannot do since each path is only calculated once.

Zelda: There has been significant improvement in the agent when trying to complete this game. The MCTS previously had very little chance of completing the game and would only do so on very rare occasions. Now the agent will move towards each of the sprites in turn and on about 50% of occasions successfully get the key and exit the level. The main issue is that the agent will still run into the moving enemies quite often and end the game. The reason for this is because unlike games like Survive Zombies and Portals where the agent rapidly moves around enemies and missiles for long periods the enemies in Zelda actually have some score worthy benefit. Using a sword next to an enemy will result in the enemy dying and the game score being increased, but, like in Missile Command, the movement of the enemies makes this difficult for the MCTS to cope with and will quite often lead to the agent colliding with the enemy sprite. With greater ability to take out the enemy sprites,

or if enemies were told to be ignored from the very start of the game, the path finding approach would almost always be able to complete this game.

6.4 Comparison to Submission Server

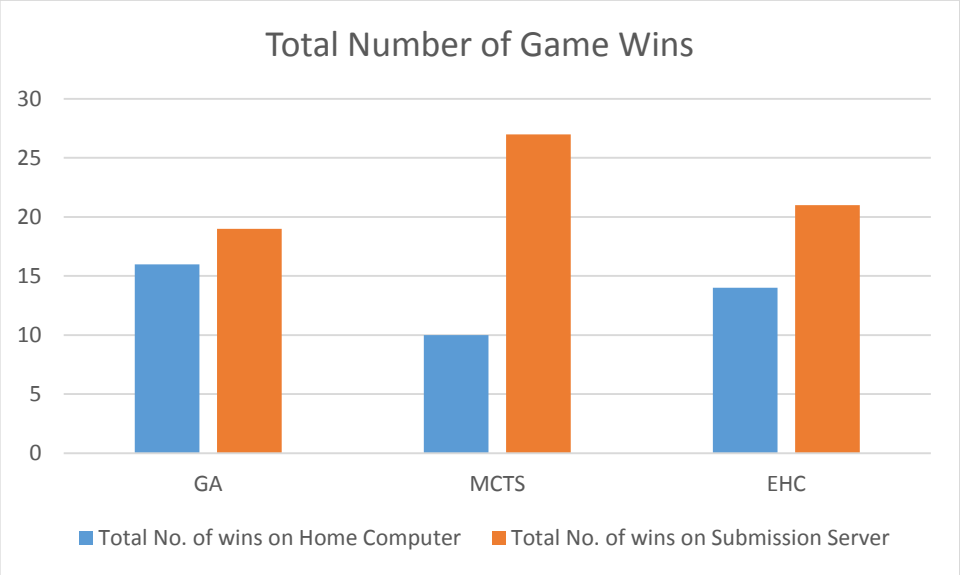

Figure 22 Number of wins achieved by agents using the training set of games on the validation server

In Figure 22 the results showed are compared with the results when submitting the same agents to the GVG-AI submission server on the same training set of games. There is an increase in wins for all the agents when ran on the submission server, which is presumably because the submission server must use a computer to calculate the results which greatly outperforms any standard home computer setup.

The crucial point to note about the results shown is the huge difference in terms of results achieved by the MCTS. On the server the MCTS now performs much better than the GA which differs from the results shown in Table 3 showing that the GA approach is largely not dependent on computational effort. The reason why the MCTS has improved is because the speed of the computations has greatly increased the area of locality of the agent. This means in a lot of cases the agent is able to see the whole game domain in its area of locality, judging by the successful rate of the results, which it was never able to do with the previous experiments. So in games like Chase and Butterflies where the goal based approach did notably better this method is not even necessary because the agent is able to see essentially everything around it. This will also have another effect in

increasing the size of the rollout depth meaning more branches can be explored and an overall increased accuracy across all the games being played. It was discussed how the MCTS had a problem in games like Zelda and Missile Command because the agent had to be positioned in a certain way to correctly attack enemies, but with this improved rollout depth this problem is reduced significantly which is evident in the results.

It's important to note that the reason the MCTS now performs better than EHC on the submission server is because the EHC method is a hindrance in some of the games and requires greater modification to be useful such as in the game frogs, aliens and portal. Despite this, it is still the case that if the level size of the games was increased it is extremely likely the EHC would once again outperform the standard MCTS.

6.4.1 Number of Game Cycles

Looking at the score and win rate are not the only ways of trying to evaluate agents' successfulness. One useful way of analysing an agents' potential is to look at the number of game cycles that are being used. When the score is evaluated for an agent the number of cycles do not influence the score, but they can be a large indicator to how well the agent is playing the game. If two agents are achieving the same score on a game we can assume that the agent achieving that outcome faster is playing the game more efficiently. The total number of game cycles being used on the games can be found on GVG-AI submission server and some of these have been selected to be shown below. These games have been picked because they are the games where the EHC and MCTS approaches were both performing well and achieving similar win rates.

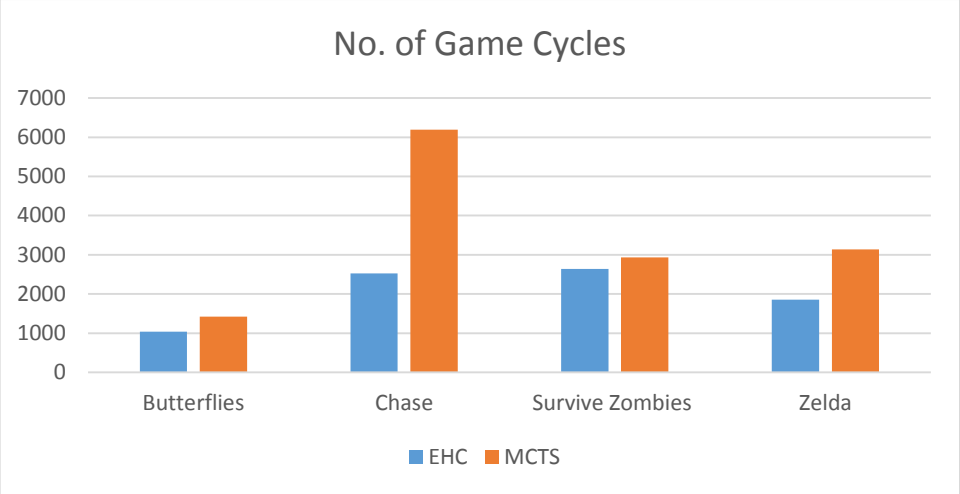

Figure 23 Number of game cycles being used by agents using the training set of games on the validation server

It can be seen from Figure 23 that the EHC method was on average 30% faster completing Zelda, roughly 40% faster in completing butterflies, and roughly 110% faster in completing Chase.

A big reason for wanting an increased speed in the agent is that the quicker an agent is able to move about the game domain the quicker they are able to interact with the game sprites and the quicker they are able to prioritise certain sprites, thereby speeding up the game process even more. Methods like this have already been attempted in GGP in games like GO with Rapid Action Value Estimation (Rave) [32] being used to speed up the learning process. Whereas methods such as RAVE require a different heuristic technique to be used on top of MCTS, the path finding used in this report just makes the heuristic method become activated a lot sooner than it otherwise would have done. It is possible that techniques like these could be combined together.

6.4.2 Results on Validation Set

When an agent is sent to the GVG-AI submission server it will be run on the standard ten games from the training set and evaluated with a score. However, as previously mentioned there is also a validation set of games that are used which cannot be seen in advance. The programmers creating the agents will be equally unaware of what the agents are about to face as the agents are.

The scoring on the submission server works slightly differently, as the agent is given a score based on the ranking of itself compared to other agents. The score for first place in each game is awarded 25 points, and then 18, 15, 12, 10, 8, 6, 4, 2, 1 for the subsequent placing's.

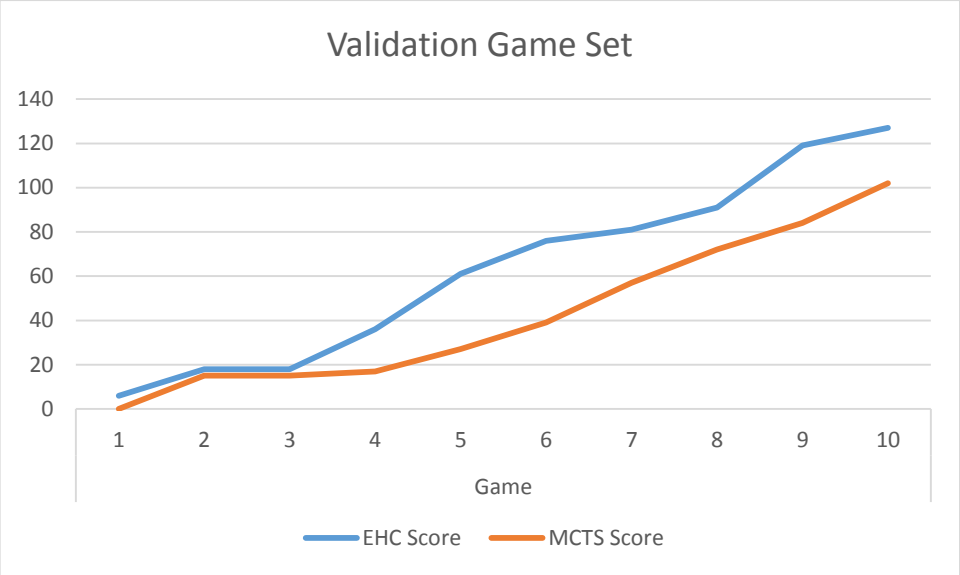

Figure 24 Results on the submission server using the validation set of games.

The results for the EHC method are very promising with the agent outperforming the normal MCTS approach on most games, and achieving a better overall score at the conclusion of game 10 by 25 points. Unfortunately since this is a validation set and there is no access to the games available there can be no comments made about the actual behaviour of the agent in these games. Although an assumption could be made that many of the same behavioural differences observed in the training set of games would also be prevalent here. The score does show that a goal based approach is improving the results of the agent but it's also quite probable the behaviour of the agent is also being significantly improved in ways that we can't tell just from looking at the score. Some probable assumptions that can be made are that in games 1, 4, 5 and 9, where there was considerable improvement with the EHC approach, that these games may have been more resource intensive or search based, as this was where the agent was also most successful in the previous set of games. This follows what would be naturally expected given a more goal based approach should lend itself to these types of games more effectively. Conversely the agent performs worse on games 8 and 10 suggesting these

games either have a large amount of dangerous sprites where the EHC method is leading the agent into inescapable situations, or that these games may be of a type where goals cannot be moved towards by the agent like in Aliens.

An interesting comparison to be made here is with the results on the submission server that were achieved with the training set and displayed in the previous section. Now the EHC is able to outperform the MCTS using the validation set of games where before it wasn't, despite the increase in computation. This suggests that the games being played on the validation set may have a much larger domain (level size) than those in the training set so even increased computation cannot help the MCTS see enough of the level in some cases without goal direction support.

This is just suggestive reasoning behind the scores, and much further analysis would ideally be needed with the ability to see the agent play the games to truly establish how well the agent is performing.

6.5 Conclusion

In this section of the report the results of the goal oriented controller designed in section 5 have been analysed in detail, giving a better understanding of what improvements have been made and also what aspects still require further development. Following on from this a discussion of the possible future work to be done, and brief conclusion will be given.

7. Discussion

In this section of the report some closing statements are given regarding the future work that is still to be accomplished in GVGP, and also a brief analysis of what has been learnt by using a goal based approach during the course of this research.

7.1 Future Work

The learning done by the agent in this implementation was basic enough that it allowed for an easy extension onto the goal orientation that was being aimed for. As such, there is a lot of room for improvement in this regard that could aid the agent further in finding goals. One improvement that could be made is to add a better prioritisation technique when defining new goals. The solution used in this experiment was adequate for some scenarios but it does require customisation to be more efficient. At the moment the agent is only affected by positive reinforcement by additions to the overall game score, but there should also be a way of applying negative reinforcement as well so the agent can be told to avoid sprites. This was a big factor in the results of the games where the path finding approaches were detrimental in some games as the agent was just being directed straight into danger. Additionally, there needs to be a method of identifying sprites that are *always* negative towards the user so that they can always be ignored. In the game Frogs where there is only one valuable sprite (the end portal) and dozens of sprites where their only purpose is to hurt the user, it would help out significantly if the agent could be aware to always avoid these types of sprites. However, by saying that a type of sprite will always be bad you are ruling out the possibility that it will become useful to the agent in the near future, and given the stochastic nature of video games this is not something you can do lightly. It's already been pointed out that none of the ten games in the GVG-AI framework have the ability to change sprite types, but this does not mean that future games won't, or that games in the unseen validation set won't either.

Another point to add on to this idea is telling the agent when to play the game or when to complete the game. If the agent is aware that it can complete the level should it do so or should it first try to maximise its potential score first? These type of questions raise the issues of exploration v exploitation that were touched upon briefly earlier in the report, and they do not seem to have definite answers. Specialised algorithms such as UCT [61] have been designed to try and solve this problem but issues of too little or too much exploration

still remain. As work moves on from GGP to GVGP a prediction is that this in particular will start to receive a large amount of focus in successful game players.

The EHC path finding method although showing some success is a method that can be improved upon for greater results. It was mentioned how the agent was able to stick to walls in an attempt to get around the level, but this method only works to an extent in this solution. Making the agent go the opposite direction to a goal when in EHC mode is not possible here, but would be a great adaption to make that should increase the consistency of results. What could be required is to implement a best-first approach similar to A* but to find ways to greater the computational efficiency of this approach. The most intuitive way may be to use a method to split up the path finding process and not try to calculate a complete path in one game cycle, which is required of A*. It has been briefly touched on how this may be possible but the concept of GVGP always having different unpredictable environments makes this a very difficult challenge to accomplish. If a best first approach was to be attempted then there would need to be another consideration taking into account regarding when to create a path. With the implementation of A* done in this report because of the lengthy computation time a path was only calculated once at the start of the path. Really the path should be recreated, or at least revaluated on every subsequent action that the agent makes, due to random events that may change the game state. Whether it is possible to be able to achieve this efficiently remains to be seen with future attempted agents.

It was also evident that in general the path finding approach used in the games did not noticeably improve results on some of the games being played. This is partly due to the path finding approaches not being as optimised as is possible, but also because using a path finding approach on its own to achieve some goals looks like it will never be truly possible. It seems to work very well on certain types of games, but obviously with GVGP this is not good enough, it needs to ideally work well with ever type of game. There is a necessity to have some other method used in coordination due to the vast number of different goals that are possible in video games. It may be the case that multiple different methods have to be combined to fully achieve this. It could also be argued that the path finding method used in this report is not general enough, and using path finding as a secondary method instead of a primary method of goal identification may help lessen some of the issues that get encountered.

Another aspect with learning within most games and within our approach is that every time the game is run the agent is essentially refreshed as if it is has not seen the game before. Human players have the added benefit that they can draw from past experiences not only in the current game but from past games as well. This would greatly improve the performance of the agents in GVGP if there was the ability to run a game *n* number of times but still have a learning mechanism in place to retain what the agent has previously experienced. The agent would be able to remember what sprites to avoid, what resources to prioritise, how enemies behave, and also the most efficient paths to take based on events that take place in the game. This could also be applied in a much more general sense. If you take the example of a human player who has played a first person shooter (FPS) game such as Call Of Duty (Activision, 2003), and if they were then to play another FPS game such as Battlefield (Electronic Arts, 2002) they would automatically use their past experience of playing a different game of the same type to try and play this new game. Ideally an agent would have a central hub of information that it can store information to relate back to, and then use to help it play any future games.

Another point to be made is that it's likely that results could be improved by modification to the MCTS algorithm itself. As it stands, at the conclusion of the competition, (15/08/2014) the top submission is a modified version of a MCTS algorithm which greatly outperforms the current sample MCTS used in this report by doubling the score it managed to achieve. Although it can't be seen exactly how the agent is managing to improve on the MCTS it would definitely be interesting if this was possible. It may be the case that experimentation could be done to combine the work done in this report with that of other improved MCTS approaches for even greater results.

7.2 Concluding Thoughts

From the research done in this report we can come to the following working conclusions:

1. A path finding method can help improve goal based GVGP, but without major adaptations will not be useful on all games.
2. Computational limitations with the amount of time allowed for calculating AI will continue to be a challenge to future progress being made.
3. Even with a very basic form of path finding the behaviour of the agent still showed an improvement in its ability to play games set before it.
4. The limitation of locality in MCTS can be greatly circumvented with use of a goal oriented approach.
5. Vast improvements can be expected to be made with the ongoing research being done in the field of GGP not just through new techniques but also continuing to improve existing methods that have still not yet been fully explored.

The three key factors towards creating a successful GVGP agent seem to be a *learning mechanism*, a *heuristic evaluation method,* (whether it be MCTS or some other method) and also *goal orientation.* This report has mainly focused on goal orientation and how that can affect the efficiency of an agent, but with combination with alternative heuristic and learning approaches the results could show interesting and beneficial differences in gameplay.

Despite the improvements that are needed, the agent in this report could serve as a useful benchmark to compare future agents against that focus on goal orientation. In future editions of the CIG competition for GVGP it should be expected that agents will greatly improve on these results when different approaches are combined and optimised. At the conclusion of the 2014 competition all the submitted agents (including the EHC agent used in this report) will compete on a final test set of 10 games which will hopefully give even more interesting and beneficial results to aid next year's competition and any other future GVGP research that takes place.

Finally, it can be concluded from the research done in in this report that the success of GVGP can be increased by not just relying on tree based search approaches, but by including a much heavier emphasis on goal orientation.

References

[1] Genesereth, M., & Björnsson, Y. (2013). The international general game playing competition. *AI Magazine*, *34*(2), 107.

[2] Levine, J., Congdon, C. B., Ebner, M., Kendall, G., Lucas, S. M., Miikkulainen, R., & Thompson, T. (2013). General Video Game Playing. *Dagstuhl Follow-Ups*, *6*.

[3] Genesereth, M., Love, N., & Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI magazine*, *26*(2), 62.

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

[5] Shannon, C. E. (1988). *Programming a computer for playing chess* (pp. 2-13). Springer New York.

[6] Deep Blue versus Garry Kasparov Challenge – Wikipedia, the free encyclopedia. Accessed online at: http://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov Last accessed at: 25/07/2014

[7] Van den Herik, H. J., Uiterwijk, J. W., & Van Rijswijck, J. (2002). Games solved: Now and in the future. *Artificial Intelligence*, *134*(1), 277-311.

[8] Krzywinska, Tanya & Atkins, Barry. (2007). Without a Goal: On open and expressive Games. Accessed online at: http://www.jesperjuul.net/text/withoutagoal/ Last Accessed: 30/07/2014

[9] Chaslot, G., Bakkes, S., Szita, I., & Spronck, P. (2008). Monte-Carlo Tree Search: A New Framework for Game AI. In *AIIDE*.

[10] Metropolis, N., & Ulam, S. (1949). The monte carlo method. *Journal of the American statistical association*, *44*(247), 335-341.

[11] Chaslot, G. (2010). *Monte-carlo tree search* (Doctoral dissertation, PhD thesis, Maastricht University).

[12] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., & Colton, S. (2012). A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, *4*(1), 1-43.

[13] Mehat, J., & Cazenave, T. Ary: A Program for General Game Playing. *Univ. Paris*, *8*.

[14] A video game description language (VGDL) - a Description. Available Online at: http://pygame.org/project-A+video+game+description+language+%28VGDL%29-2397-.html Last Accessed at: 03/08/2014

[15] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, *4*(2), 100-107.

[16] Nareyek, A. (2004). AI in computer games. *Queue*, *1*(10), 58.

[17] Champandard, Alex J. (2007). Near-Optimal Hierarchical Pathfinding (HPA*). Available Online at: aigamedev.com/open/review/near-optimal-hierarchical-pathfinding/ Last accessed: 05/08/2014

[18] Yap, P. (2002). Grid-based path-finding. In *Advances in Artificial Intelligence* (pp. 44-55). Springer Berlin Heidelberg.

[19] Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, *27*(1), 97-109.

[20] B. Stout. (1996). Smart moves: Intelligent path-finding.Game Developer Magazine: 28–35

[21] Botea, A., Müller, M., & Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of game development*, *1*(1), 7-28.

[22] Forward Model and State Observation. Available Online at: http://www.gvgai.net/forwardModel.php Last accessed: 06/08/2014

[23] Genetic Algorithm. Available Online at: http://en.wikipedia.org/wiki/Genetic_algorithm Last Accessed: 07/08/2014

[24] Louis, S. J., & Miles, C. (2005). Playing to learn: case-injected genetic algorithms for learning to play computer games. *Evolutionary Computation, IEEE Transactions on*, *9*(6), 669-681.

[25] Alhejali, A. M., & Lucas, S. M. (2010). Evolving diverse Ms. Pac-Man playing agents using genetic programming. In *Computational Intelligence (UKCI), 2010 UK Workshop on* (pp. 1-6). IEEE.

[26] Finnsson, H., & Björnsson, Y. (2008, July). Simulation-Based Approach to General Game Playing. In *AAAI* (Vol. 8, pp. 259-264).

[27] Ebner, M., Levine, J., Lucas, S. M., Schaul, T., Thompson, T., & Togelius, J. (2013). Towards a video game description language. *Dagstuhl Follow-Ups*, *6*.

[28] Koller, D., & Pfeffer, A. (1995). Generating and solving imperfect information games. In *IJCAI* (pp. 1185-1193).

[29] Thielscher, M. (2010). A General Game Description Language for Incomplete Information Games. In *AAAI* (Vol. 10, pp. 994-999).

[30] Thompson, Tommy. (2014). GVG-AI. The Challenge of General Intelligence in Games. Available Online at: http://t2thompson.com/tag/gvg-ai/ Last accessed at: 16/08/2014

[31] Rosca, J. P. (1996). Generality versus size in genetic programming. InProceedings of the First Annual Conference on Genetic Programming (pp. 381-387). MIT Press.

[32] Finnsson, H., & Björnsson, Y. (2010, April). Learning Simulation Control in General Game-Playing Agents. In AAAI (Vol. 10, pp. 954-959).

[33] Grefenstette, J. J., Moriarty, D. E., & Schultz, A. C. (2011). Evolutionary algorithms for reinforcement learning. arXiv preprint arXiv:1106.0221.

[34] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. arXiv preprint cs/9605103.

[35] Erev, I., & Roth, A. E. (1998). Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. American economic review, 848-881.

[36] Barto, A. G. (1998). Reinforcement learning: An introduction. MIT press.

[37] Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. Artificial Intelligence, 136(2), 215-250.

[38] Schaeffer, J., & Van den Herik, H. J. (2002). Games, computers, and artificial intelligence. Artificial Intelligence, 134(1), 1-7.

[39] Samuel, A. L. (2000). Some studies in machine learning using the game of checkers. IBM Journal of research and development, 44(1.2), 206-226.

[40] Middleton, Zak. (2002) Case History: The Evolution of Artificial Intelligence in Computer Games. Available online at: http://web.stanford.edu/group/htgg/cgi-bin/drupal/sites/default/files2/zmiddleton_2002_1.pdf Last accessed: 12/08/2014

[41] Xu, Siyuan. History of AI design in video games and its development in RTS games. Available online at: https://sites.google.com/site/myangelcafe/articles/history_ai
Last accessed: 12/08/2014

[42] Nguyen, K. Q., & Thawonmas, R. (2013). Monte carlo tree search for collaboration control of ghosts in ms. pac-man. Computational Intelligence and AI in Games, IEEE Transactions on, 5(1), 57-68.

[43] Rémi Coulom (2008). "The Monte-Carlo Revolution in Go". Japanese-French Frontiers of Science Symposium. Available Online at:
http://remi.coulom.free.fr/JFFoS/JFFoS.pdf Last Accessed: 16/08/2014

[44] Chaslot, G. M. J. B., Saito, J. T., Bouzy, B., Uiterwijk, J. W. H. M., & Van Den Herik, H. J. (2006). Monte-carlo strategies for computer go. In Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium (pp. 83-91).

[45] Sheppard, B. (2006). Efficient control of selective simulations. In Computers and Games (pp. 1-20). Springer Berlin Heidelberg.

[46] DeNero, J., & Klein, D. (2010). Teaching introductory artificial intelligence with pac-man. In Proceedings of the Symposium on Educational Advances in Artificial Intelligence.

[47] Gallagher, M., & Ryan, A. (2003). Learning to play Pac-Man: An evolutionary, rule-based approach. In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on (Vol. 4, pp. 2462-2469). IEEE.

[48] Ramon, J., & Croonenborghs, T. (2006). Searching for compound goals using relevancy zones in the game of Go. In Computers and Games (pp. 113-128). Springer Berlin Heidelberg.

[49] Robles, D., & Lucas, S. M. (2009). A simple tree search method for playing Ms. Pac-Man. In Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on (pp. 249-255). IEEE.

[50] Genesereth, M., & Love, N. (2006). General game playing: Game description language specification.

[51] Schiffel, S., & Thielscher, M. (2007, July). Fluxplayer: A successful general game player. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 22, No. 2, p. 1191). Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

[52] Chapter 1 - Introduction - Stanford Logic Group. (2010). Available Online at: http://logic.stanford.edu/games/chapter01.html
Last Accessed: 18/08/2014

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

[54] Wikipedia. Arimma – Wikipedia, the free encyclopedia. Available Online at: http://en.wikipedia.org/wiki/Arimaa#Arimaa_Challenge
Last Accessed: 18/08/2014

[55] Wikipedia. 2K Australia – Wikipedia, the free encyclopedia. Available Online at: http://en.wikipedia.org/wiki/2K_Australia Last Accessed: 18/08/2014

[56] Wikipedia. AI Challenge – Wikipedia, the free encyclopedia. Available Online at:http://en.wikipedia.org/wiki/Google_AI_Challenge Last Accessed: 18/08/2014

[57] The General Vide Game AI Competiton – 2014. Available Online at: http://www.gvgai.net/ Last Accessed at: 18/08/2014

[58] The GVG-AI Framework – Code Structure. Available Online at: http://www.gvgai.net/code.php Last Accessed at: 20/8/2014

[59] Clune, J. (2007, July). Heuristic evaluation functions for general game playing. In*AAAI* (Vol. 7, pp. 1134-1139).

[60] The GVG-AI Framework – Forward Model and State Observation. Available Online at: http://www.gvgai.net/forwardModel.php Last Accessed: 20/08/2014

[61] Gelly, S., & Wang, Y. (2006). Exploration exploitation in go: UCT for Monte-Carlo go.

[62] Champandard, Alex J. (2014). Monte-Carlo Tree Search in TOTAL WAR: ROME II's Campaign AI. Available Online at: http://aigamedev.com/open/coverage/mcts-rome-ii/ Last Accessed: 21/08/2014

[63] Cole, N., Louis, S. J., & Miles, C. (2004, June). Using a genetic algorithm to tune first-person shooter bots. In *Evolutionary Computation, 2004. CEC2004. Congress on* (Vol. 1, pp. 139-145). IEEE.

[64] Thompson, Tommy. (2014). GVG-AI. The Challenge of General Intelligence in Games. Available Online at: http://t2thompson.com/2014/02/10/ai-and-pacman/ Last accessed at: 16/08/2014

[65] Ferguson, D., Likhachev, M., & Stentz, A. (2005, June). A guide to heuristic-based path planning. In Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS) (pp. 9-18).

[66] a-star-java. (2010). Available Online at: https://code.google.com/p/a-star-java/source/browse/?r=7#svn/AStar/src/aStar Last Accessed at: 23/08/2014

[67] Levinson, R. (1996). GENERAL GAME-PLAYING AND REINFORCEMENT LEARNING. Computational Intelligence, 12(1), 155-176.

[68] Timeline of Artificial Intelligence. Available Online at: http://en.wikipedia.org/wiki/Timeline_of_artificial_intelligence Last Accessed on: 24/08/2014

[69] Thielscher, M. (2009). Answer set programming for single-player games in general game playing. In Logic Programming (pp. 327-341). Springer Berlin Heidelberg.

[70] Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. Artificial intelligence, 27(1), 97-109.

Appendix 1 – Additional Detailed Results

Table of results for comparison of number of wins achieved on the submission server, displayed in Figure 22.

	Total No. of wins on Home Computer	Total No. of wins on Submission Server
GA	16	19
MCTS	10	27
EHC	16	21

Table of results for the number of game cycles that are being used to complete the games in the training set on the submission server, displayed in figure 23.

Agent		Game 1	Game 2	Game 3	Game 4	Game 5	Game 6	Game 7	Game 8	Game 9	Game 10
EHC	No. of Cycles	4141	909	**1040**	**2523**	8	654	447	6209	**2635**	**1854**
MCTS	No. of Cycles	2636	5566	**1418**	**6196**	4396	478	6657	8007	**2932**	**3135**
GA		3313	10000	**2541**	**8996**	1774	843	1727	8229	**1917**	**3232**

Table of results comparing EHC with MCTS using the validation set of games on the submission server, displayed in Figure 24.

Agent		Game 1	Game 2	Game 3	Game 4	Game 5	Game 6	Game 7	Game 8	Game 9	Game 10	Total
EHC	Score	6	12	0	18	25	15	15	10	18	8	**127**
MCTS	Score	0	15	0	2	10	12	18	15	12	18	**102**