|  |  |
| --- | --- |
| Title of Project | Creating a Pathfinding Algorithm that Uses A\* Search and Behaviour Trees to Navigate a Platforming Environment |
|  |  |
| Name of Student | Sean O’Sullivan |
| Student Number | K00180620 |
| Date | 26/03/2020 |
| Word Count | 7,451 |

I declare that no element of this assignment has been plagiarised:

|  |  |
| --- | --- |
| Student Signature |  |



Table of Contents

[1 Introduction 3](#_Toc36117719)

[2 Problem Definition and Background 5](#_Toc36117720)

[3 Literature Review and Research 8](#_Toc36117721)

[4 System Design and Configuration 12](#_Toc36117722)

[5 Example of use 15](#_Toc36117723)

[6 Implementation and Testing 16](#_Toc36117724)

[7 Critical analysis and Conclusions 21](#_Toc36117725)

[8 Appendix 24](#_Toc36117726)

[8.1 User Manual 24](#_Toc36117727)

[9 References 25](#_Toc36117728)

# Introduction

The aim of the project was designing and developing a tool in Unity that would allow developers of platforming games to deploy an AI agent that would move through and complete a level that they have created. The tool used a Pathfinding algorithm that utilized A\* Search and Behaviour Trees to find the optimal path through a level. The project researched and experimented with several different algorithms as the tool was designed to be as universal as possible and work with a wide range of environments within the platforming genre. The tool was chosen to focus on platforming games specifically, not because they are an iconic and popular genre, but because they have dynamically changing environments and moving enemies that the player encounters and must overcome. These factors mean that finding the path to take may be accomplished with an A\* Search but the actual navigation of the levels will contain more complex algorithms.

The tool was originally designed to appeal to players as well. In the first iterations of the tool, there was a plan to incorporate a Replay System that records the players progress throughout a level and compares it to the AI agents’ path. After completing a level, the player could then look back at a recorded replay and see their progress and compete with the AI agent. The Replay System would have featured its own set of controls, players could pause, rewind and fast forward the replay they are viewing so they would be able to focus on whatever part of the level they choose. Developing the Replay System came with its own set of challenges, systems can either be Deterministic or Saved State. A Deterministic system can record player inputs and determine where the objects will be moving. In a Saved State system, everything is recorded, and the system builds the scene from scratch each time. The Replay System was ultimately not used as it was not doing anything new and it detracted from the main function of the tool which was navigating a platforming environment.

A small platforming scene in Unity has been developed for the purposes of showcasing the tool. The scene features a player character that can move, jump and shoot, an enemy character that moves towards the player and can take damage and be destroyed, and a floating coin that is at the end of the scene and is the target of the player character’s pathfinding. The scene can also be customised and changed with the Tile Palette tool in Unity. The Tile Palette can be used to remove blocks from the scene and place more blocks so the scene can be updated to demonstrate how the agent can handle different environments. The project sought to develop this tool as it will be beneficial for developers to see what path players are taking throughout their levels. This data would be invaluable in Level Design and developers can use the tool to design and develop their levels based on player interaction with their prototypes.

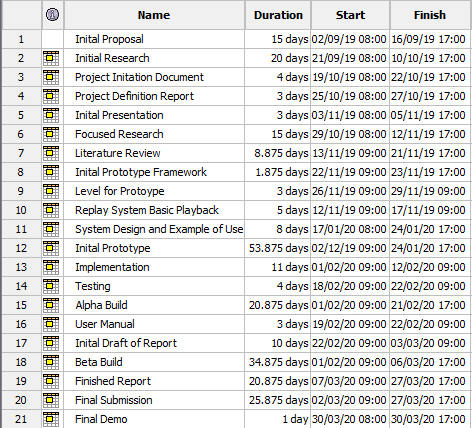
From conducting initial research for the project, there has been some research in this field using the game *Super Mario Brothers,* specifically the *Mario AI* benchmark. Using this as a reference, the tool was designed using similar rules to the iconic franchise, as most platforming games follow these fundamental gameplay rules: The player can move left to right, perform a jump, can duck to avoid projectiles and can jump on enemies to destroy them. Using this initial set of rules, the pathfinding algorithms were built around this to determine the best route a player can take throughout a level. In the final design, the tool had implementations for jumping and shooting mechanics as these were deemed to be more universal in modern platformers.

Unity was chosen specifically because of its powerful built in Physics as well as its popularity and multiplatform support. Unity Developers can design levels using Unity’s integrated level editor and then use the tool to rapidly see if their levels can be completed or if they need to go back and change factors such as: size of platforms or density of enemies that are roaming the level. I also have a personal preference for Unity and have a passion to learn as much about the engine as I can, to build up a portfolio of games that I can use to demonstrate my skills to potential employers.

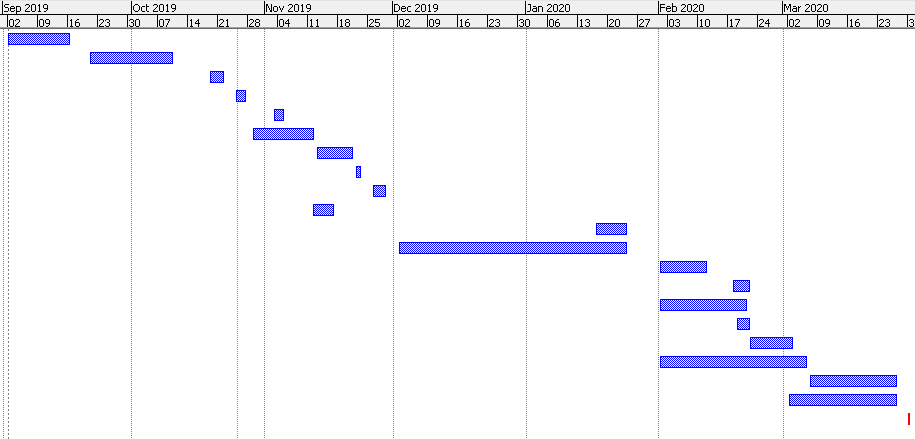
# Problem Definition and Background

The objective of the Final Year Project is to design and develop a polished product that showcases the skills that I have learned from my time in college. By undertaking the project, I have demonstrated that I can work independently and can utilize the skills I have gained by undertaking this course, as well as showing I can use my resources to effectively improve my knowledge to complete any task that I encounter. To complete the project, I have conducted my own research from a large pool of sources that I determined to be credible, and correctly applied and referenced these sources to support my own findings and evaluations. I self-managed and defined a strict schedule that I attempted to adhere to while also managing other assignments and responsibilities. In this report I will demonstrate and explain what I have accomplished as well as what I have learned from the entire process. I acknowledged my own shortcomings and how I attempted to overcome them through rigorous routine, and I have examined and discussed any failures that happened. Failures are an important part of the project as I have learned from them just as much as the rest of project’s deliverables. I have worked with my project supervisor to clearly layout my weekly objectives and have taken guidance as well as criticism to complete the best version of the project that I can accomplish. Having completed the project, its undertaking and the way I have completed it, is something that I can showcase in a portfolio to distinguish me for employment in a competitive job market.

For my project plan, I have used the Agile Development Approach. Software Development projects are complex in nature and require several interdependent parts that are constantly changing and updating. Agile Development allowed me to manage and change my project plan as new techniques and challenges presented themselves. I have laid out my initial schedule in *Figure 1*. The schedule mainly focused on how I would be completing the various deliverables required for the project. As the project progressed, I have edited the file to break down the required programming tasks when they came up. This plan was used as an estimate more than anything and it was, regrettably, not formally adhered to. I have used Project Libre to make the initial file but moved on to use Excel due to its simplicity. Finally, I ended up using the Tasks app on my iPhone as this was an easier user experience as well as a motivational tool to see what tasks needed to be accomplished daily. I have broken down the various tasks in my initial plan in *Figure 1*, as well as included a Gantt chart in *Figure 2*.



*Figure 1: Initial Schedule laying out important delivery dates*

**

*Figure 2: Gantt chart*

The research for the project started by focusing on algorithms that the tool would potentially use. I then provided a Literature Review to further breakdown these areas and glean all the information relevant to my project that I could. Before even attempting any coding, I used pen and paper to hash out a design, wrote a feature list and some pseudocode as well as drew up various Use Case and Class diagrams. Once I had a solid foundation, I then went about designing and creating a simple platforming level in Unity that I used to test the use of the pathfinding algorithms. This level is a rough prototype that I used as a sandbox to develop the tool. It featured basic player inputs and one or two enemy objects that the player can destroy or avoid as well as some basic platforms and obstacles. I then went about writing the scripts that used the pathfinding algorithms to navigate through the level. Once the initial level could be completed by the AI agent, I gradually began making the level more complex and tested the agent against it.

I regularly checked and updated my schedule as well as kept weekly logs to help with the various parts of the final report that I have produced. While keeping logs is a monotonous task, it was invaluable information that I used as a reference in this report. I also used GitHub and used detailed commit messages which also helped keep track of the work that was completed at each stage. I regularly edited and added to my final report draft to provide a clear and engaging piece of content that could accurately convey the work that was put into the project. I planned for all demonstrations and presentations that were completed and I improved my public speaking skills through short speaking exercises which helped me sell myself as well as my project as being something that would be of value to anyone looking to develop platforming games or looking to explore player interaction.

# Literature Review and Research

For the research for my final year project I have explored the various pathfinding algorithms being researched in the field today. I focused on practical examples, particularly pathfinding algorithms being used in Video Games. Video Games are a great environment to create and test pathfinding algorithms as they provide immediate visual feedback as to how the algorithm operates. Pathfinding algorithms are used frequently in Video Games, as the enemies that players encounter use them to navigate the world. Any exploration in this field will be constructive to the medium because making characters in a Video Game appear more intelligent, helps to ground the experience for the player, and provide a satisfying and realistic environment. As stated, the more practical examples and how they are accomplished are what I focused my research on. This was because the more prepared I was to face these challenges, the easier I found it to overcome them. I also explored various testing that is done in Video Games and some tools that developers are using themselves.

This topic was worth researching, as using the tool would save costs in the development of a platforming game. If a developer creates a level, they can use this tool to get instant feedback on whether the level is completable and how long it took to navigate. as well as balancing with new enemies. This data can be used to provide post launch support and in balancing game difficulty. The less time developers are spending testing the more time they can be developing. Testers themselves could use the tool to traverse an environment while they focus on more complex testing.

A lot of the research in this area is based around the *Mario AI* competition. (Shinohara, et al., 2012) The competition, which ran from 2009 to 2012, pits AI bots against each other to complete a series of procedural generated levels. The competition uses *Infinite Mario Bros*, which is an open source clone of Nintendo’s *Super Mario Bros*. As the code is freely available and in public domain, it is used as a benchmark for AI pathfinding in Video Games. Super Mario is such a staple in Video Games, it makes sense to build these systems around this game, as almost all modern platforming games are heavily inspired by the iconic franchise. As such, most games also follow the same gameplay rules and these gameplay rules can help inspire the rules for the pathfinding algorithm to follow. Mario moves left to right across 2D levels, avoiding enemies by jumping over them or on top of them to destroy them. Mario also can pick up power ups to change his state, meaning he can take extra hits before dying while another power up causes him to be able to shoot projectiles. When creating an AI system, all these gameplay elements were imperative for the AI to use, to navigate and overcome any obstacles it encountered.

Many algorithms have been used in the *Mario AI* competition. The most frequently being A\* which also has the best results. (Ortega, et al., 2013) Some more recent research used Grammar Based Genetic Programming. (Freitas, et al., 2018) Genetic Programming is a programming concept inspired by the natural world. Darwin created Natural Selection, a theory that only the strongest of the species survive to produce offspring and Genetic Programming is based on this construct. Given a series of results, based on a tree structure, only the best results are carried over to be tested again. The Grammar Based Genetic Programming ensures that a common language is used across the models to minimize any loss of data, so only paths that can be made would be tested against each other to provide the best results.

*Monte Mario* was another research in which the researchers used Monte Carlo Tree Search, a pathfinding algorithm that uses a tree search to perform random actions until a favourable path can be determined. (Ortega, et al., 2013) The researchers found that utilizing MCTS with Mario performed poorly. However, they adapted the algorithm to use a variety of different techniques to improve the performance and found that incorporating all these techniques together produced the best results.

To move away from Mario, *SpleunkBots API* was a research that sought to create their own benchmark that used the game *Spelunky* to test its AI agent. (Scales & Thompson, 2014) *Spelunky* is a game where the player navigates a series of tunnels collecting loot to get a high score before finding an exit. The game has its own set of unique mechanics, the most prevalent being the ability to use bombs to completely alter the terrain to create new paths entirely. These dynamically changing environments bring up new challenges to the pathfinding, as new routes must be calculated. The research mainly focused on building environments that the bots could navigate and found that MCTS were not effective in this scenario, due to the dynamic nature of the environments. They recommend that custom modules must be built on MCTS like *Monte Mario* accomplished.

A different approach was used in *Monster Carlo*. (Naveed, et al., 2012) This research used MCTS to build a pathfinding algorithm that can be used to make decisions in a game, in order to maximize the high score that can be achieved. They built their own system and tested it with a game they developed themselves, as well as a game they downloaded from the Asset Store. The game they developed themselves was a Tetris clone where the player must assemble monsters into shapes. The AI had a series of decisions to make like where to place the next block or if they needed to rotate the block to get a higher score. The game from the Unity Store was a 2D Roguelike in which the player navigated a 2D space to attack and defeat enemies. *Monster Carlo* was a universal system that worked with Unity, so I utilized this research when I created my own system.

Although I have stated the benefits of building from Mario, there is also a detriment as it is all focused in one area. What if there are games that have their own unique mechanics? A lot of modern platforming games must introduce their own unique mechanics to stand out from the crowd. In *Ori and the Blind Forest*, you have numerous abilities that allow you to dash and glide across levels, *Braid* allows you to control the flow of time to overcome obstacles and the hugely successful *Hollow Knight* has its own unique combat system. Having an AI that can be used as universally as possible must have a way to customize the base rules to allow these mechanics to be built around different environments. The *Mario AI* competition was also 10 years ago, and although there is still recent research ongoing (Freitas, et al., 2018), it is time to move towards a universal AI controller that can handle more difficult and complex tasks.

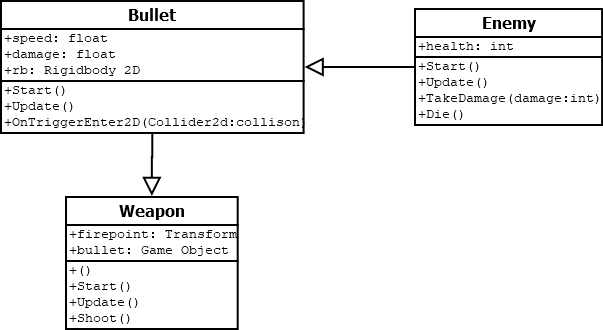
From undertaking this research, I came across *Super Mario* in most papers. This is expected as most platforming games are heavily inspired by the game, as it defines the genre. As such, these systems were built using Mario’s game rules as a framework. From all the different methods, A\* seemed to prevail over most. As such all the algorithms used are some evolution of A\*, for example Moving D\* Lite is a dynamic algorithm that can handle dynamically changing levels.

The results from the research that used MCTS found that this approach did not perform well on its own. However, when using it as a framework and building on it with other algorithms, these systems did produce favourable results. As I planned to build a universal system, similar to what *Monster Carlo* did, I initially hoped to use MCTS as a base and incorporate an evolution of A\*, like Moving D\* Lite, in order to create an algorithm that can be adapted to any environment. In the final project, I used Aron Granburg’s *A\* Pathfinding Project* as it was able to handle any dynamically changing environment which was the biggest concern for using A\*.

To summarize, there has been a lot of research in pathfinding AI built around *Super Mario*. Some more recent research has been moving away from this benchmark and has been creating their own systems. As I wanted to create a universal tool that can be utilized in Unity to support any game, I incorporated techniques used by more recent research like *SpelunkBots API* and *Monster Carlo*. I was also looking for more practical utilizations, such as what goes into developing tools for Unity, but unfortunately was not able to upload anything to the Unity Asset Store.

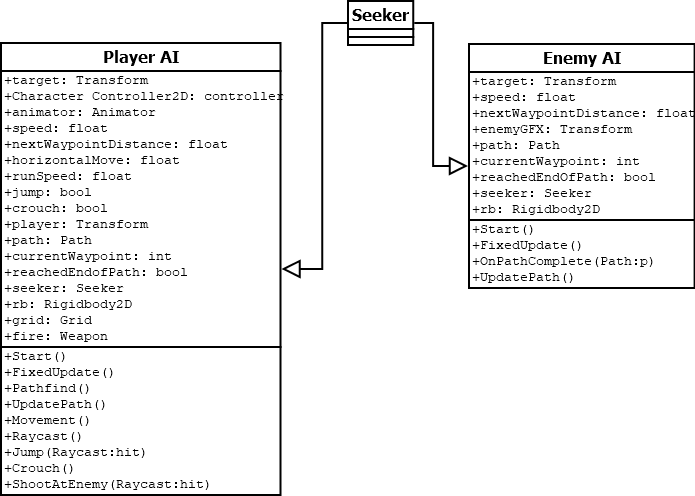
# System Design and Configuration

The following class diagrams represent the overall system of the project. The details of classes and libraries I have used from other sources have not been included here. The first diagram, labelled *Figure 3*, is the basic classes that handle the shooting of the weapon from the AI agent. These classes allow the agent to instantiate a Bullet from the players position, the bullet moves at a desired velocity, and, when it strikes the Enemy, takes damage from the enemy until it is destroyed.



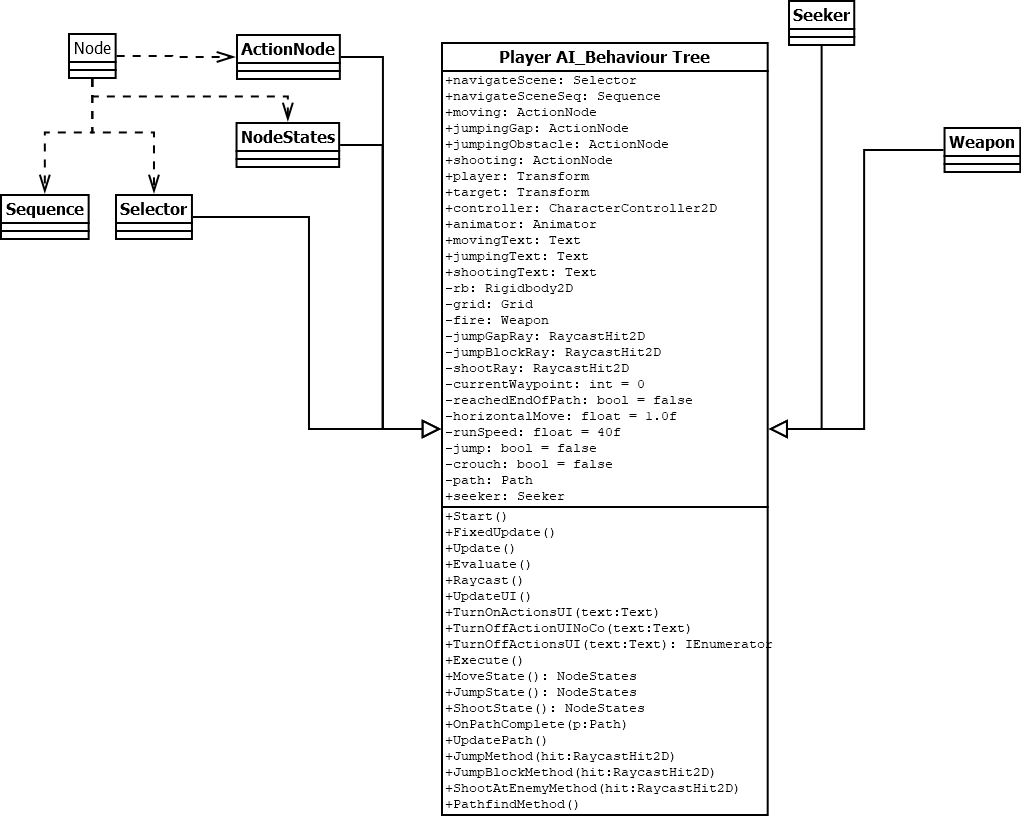
*Figure 3*

The next set of classes, *Figure 4*, were the initial set of classes used to handle the AI agent. The Seeker class handles all the Pathfinding elements, it uses an A\* Search to find the shortest path and the corresponding Player or Enemy class handles all the navigation along the path. The Seeker class here has been imported from Aron Granburg’s *A\* Pathfinding Project* and it is a library I have used for the tool to handle the A\* searching. The PlayerAI class, and the class that replaced it, both use the Seeker class in the Pathfind method to move the agent across the scene. They also both use the Weapon class to handle the shooting. The Raycast method handles the creation of the raycast used for navigation and the Jump and ShootAtEnemy methods use these rays to jump over obstacles and to shoot at enemies.



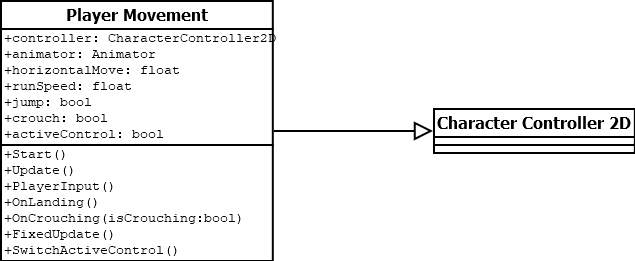
*Figure 4*

The next set of classes, *Figure 5*, are the main classes of the project. The PlayerAI\_BehaviourTree class was an extension and then a replacement of the previous PlayerAI class shown in *Figure 4.* The class has been extended to incorporate a Behaviour Tree to determine the state the AI is currently in and calls the correct method depending on what it encounters in the scene. The blueprint for the tree, like the Node, Action Node and Sequence class, were taken from *Unity 2017 Game AI Programming Third Edition*. These classes helped to set up the tree and I used my own functions to define the actions that the tree takes. In this class I also added some UI elements that showed what action the tree was currently executing. The MoveState, JumpState and ShootState methods would return what branch of the tree had successfully executed when the tree was evaluated in the Evaluate method. The decisions on why I chose to use a Behaviour Tree, as well as a more in-depth explanation on how it works is discussed in the Implementation chapter of this report.



*Figure 5*

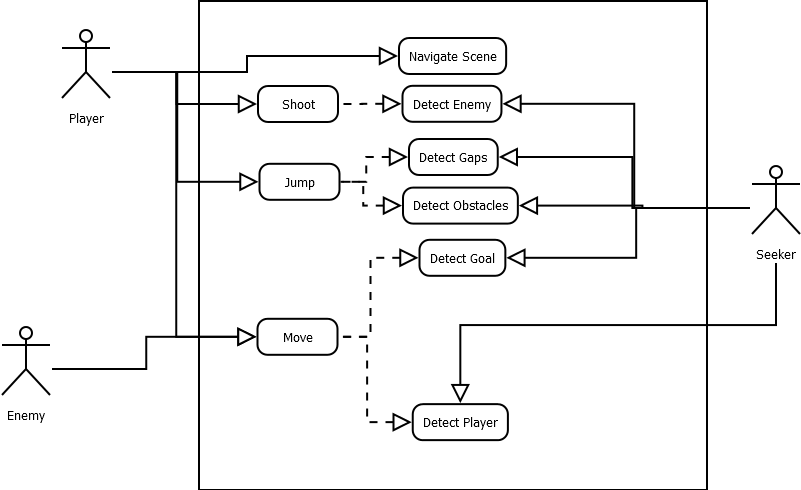
The last class diagram, depicted in Figure 6, handles all the Player Input, this is primarily used in testing and was not used in the final prototype.



*Figure 6*

# Example of use

The Use Case diagram, depicted in *Figure 7*, shows how the various parts of the system interact. The Player object navigates the scene to get to the goal at the end of the Scene. The Seeker handles all the Pathfinding, and detects enemies, gaps and obstacles. The Player then shoots these enemies and jumps over these obstacles. The Enemy also implements the Seeker, to find the Player, and it then moves towards it.



*Figure 7*

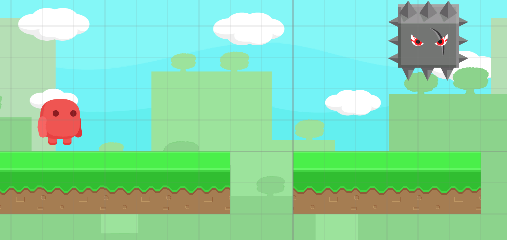
# Implementation and Testing

When beginning the implementation of the algorithm, I had to take a more realistic look at what exactly I wanted the tool to do, as well as what exactly I could accomplish given the time constraints. After some rapid prototyping it became apparent that the idea of a Replay System had to be immediately scrapped. The complexities of designing and developing a deterministic system did not seem worth it for what the output would have been. A Replay System is something that might have only been needed in a small percentage of games as opposed to the main point of the project which could be used in all platforming games. A Replay System itself has already been done in multiple games and my proposed implementation would not have done anything new. The idea was influenced by the game *Super Meat Boy,* which is fun for short and fast levels, but ultimately would not appeal to players in most games. I chose to primarily focus on developers and testers being the prime audience of the tool. Testing can take up a large percentage of development time and streamlining this process can cut costs dramatically. The tool could be a benefit to testers as they could focus on more complex tasks and let the tool handle the mundane task of navigating environments.

As stated in my research, the optimal implementation was a MCTS used in combination with A\* search. I opted to use Aron Granberg’s *A\* Pathfinding Project* as it was a framework that allowed a highly customizable agent that could handle all the pathfinding. This implementation allowed the pathfinding to be updated on the fly and it was a workaround to A\*’s biggest weakness, which was handling dynamically changing environments. In the end, I did not use MCTS but instead opted to create a Behaviour Tree to handle the decision making of the agent. I will now discuss my step by step implementations and the various problems that I encountered.

Before creating the algorithm, I first built a sandbox in which I could test the algorithm. I imported an Asset pack of a Platforming game from the Unity Asset Store. The pack contained sprites for characters, animations and a tile map that I could use to layout the demo level. I also added some basic animations that showed when the Player moved and jumped which helped for demos.

Once I had a test environment set up, depicted in *Figure 8*, I began adding a character that I could control with input from the keyboard as well as create an enemy that I could shoot at. I added an enemy that could be destroyed and begun using A\*. The enemy was the first implementation of pathfinding as it moved towards the player. I used Granberg’s *A\* Pathfinding Project* because I didn’t want to use Unity’s navmesh as I wanted to do some configuration. This library was also open source and contains a vast array of tools that I could use.



*Figure 8. Test environment showing Player, Enemy and Tiles.*

I added functionality to move, jump and shoot, these are standards in any platformer game. Therefore, these were the actions that I wanted the agent to perform. In order to create an agent that could do these tasks autonomously, I had to first visualize what these tasks were and what was required to do these tasks. Why was I jumping when I approached a gap? Why was I shooting at an enemy? It was because I could see them. This made it clear that I had to give the AI sight, and the obvious answer was Raycasts. A raycast is a function that casts a ray from the point of origin that can be directed and its length set. This ray tests what colliders it encounters, and this was exactly what I needed. I would have one ray that shoots ahead of the player and tests for enemies, if it sees an enemy it would then shoot. I would also have another ray that would shoot down, and if it did not detect the ground, it would know that there is a gap and it would then jump. While testing I also discovered that I needed a third ray that would also shoot forward but at a shorter distance. This ray would test if there was an obstacle in front of the player and would jump over it if there was.

I then began creating the first player AI class. This class used Raycasts to detect enemies and gaps. I also added the A\* implementation to the Player and it moved towards the Goal which I represented in the demo with a floating coin. Raycasts were an implementation I wanted to use as it gave the agent sight, it could make decisions based on what it could see, and this was an idea that made the inner workings of the agent more tangible.

The next few weeks were polishing up this class for the first batch of demos. As discussed in my research chapter, a lot of my research pointed towards using A\* with a combination of MCTS. This combination would yield the best results in the works analysed so I went about trying to implement MCTS. I thought that my research was sufficient in this area but unfortunately, I was not able to get a working implementation. I then took a step back and decided on implementing a Behaviour Tree first and going from there. I found a blueprint of a tree that I could use to set up my nodes and began building a tree that would suit my purposes.

A Behaviour Tree is a conceptual model that allows you to execute a set of functions in a hierarchical manner. The root of the tree defines a base action and the various leaves of the tree can be set to complete specific tasks. The leaf nodes I had set in my tree were the actions that the algorithm executes: moving, jumping and shooting. These Action Nodes returned a success or failure to the parent node depending on if they have completed their required task. The parent node can either be a Selector or a Sequence. A Selector node executes each node until one succeeds while the Sequence node returns each node till one returns a failure. So, in a Sequence node each node must succeed to return a successful execution. I experimented with both but opted for a Selector node as the decisions could be executed at any time. (Nicolau, et al., 2017)

The PlayerAI\_BehaviourTree class built off the PlayerAI class and replaced it altogether. In practice, it did the same things as the original class, but it had a more complex behaviour that was more customizable. I would have liked to have spent more time fleshing out the class but could not. I also added some UI elements that would display what the AI is doing at certain points. I could not get a functionality working with movement that I liked, as the movement script would run in a step by step fashion rather than smoothly, as it does in FixedUpdate method. I therefore decided to just have the Tree decide when to shoot and went to jump. I had some difficulties with the tree, when first building the tree I had a misconception with how the tree operated, I was using the tree changing states to run my methods, but this was incorrect. The states should do exactly as they are named and return if the state of the tree has succeeded or failed.

Testing was primarily done with Log messages. Log messages are a way to print code out within Unity. I would step by step through lines of code to see which data was being sent where. I used Log messages to debug issues with Raycasts, mainly to see which entities where being detected by the rays per frame. I would also display what state the Behaviour Tree was currently in and displaying if it has currently succeeded or failed at certain points.

When testing the navigation, it brought to attention a few bugs which needed to be addressed. The agent would become stuck if it ran straight into an obstacle. To fix this, I had to add an additional leaf to the tree and the additional Raycast which was discussed earlier. This ray detected obstacles straight ahead of the agent. After testing this it became apparent that the agent could not handle large gaps as previously thought and it would not jump early enough to overcome them. In the end it led to a scramble like behaviour as the agent would land on edges and appear to climb up them, which was not something I intended and actually looked pretty realistic, however, the agent would fail to make the initial blind faith jump over larger gaps.

The algorithm that I developed was able to complete most levels albeit in a linear fashion. The A\* pathfinding was able to map any walkable terrain and gave a direct route to the Goal but could not factor in any backtracking. So, if the agent needed to go back to jump over an obstacle it would stay going in the direction of the Goal. If you picked up the Goal and moved it, the agent could still follow the Goal, so it could handle either direction. It would just walk towards the Goal in a linear fashion. As stated, there was other issues where the agent would not jump in time to handle larger gaps. The shooting worked fine, as it was instantiating a bullet when it was needed, rather than setting a Boolean value like it was doing in the Jumping Action Node. If I had handled my time more effectively, I would completely rewrite the character controller class and create my own method that simply applied a force to make the character jump.

All things considered I am happy with the outcome. The agent was able to navigate most scenes and the added UI element gave an immediate visual feedback of what action the agent was currently taking. The research pointed to a combination of MCTS and A\* being the best method and I would have liked to have got that working. A highly complex platforming game would have benefited from a MCTS but for a simple run and gun, the Behaviour Tree does more than enough to handle the decisions, buts still I felt that I did not use the Behaviour Tree to its full potential. I felt if I had started working on a solid Behaviour Tree first, I would have produced a more optimal implementation.

I also did not explore any of the options of creating an Asset for the Asset Store. The tool as it stands is a prototype and not a finished product that I can showcase. I will take some time after my studies to work on the tool. The groundwork has definitely been laid to produce a valuable product.

# Critical analysis and Conclusions

The project I have completed is designing and developing a pathfinding algorithm that has utilized A\* search as well as a Behaviour Tree to navigate a platforming level. This report has been an outline of the methods I have used to produce this tool as well as a documentation of my own findings and what I have achieved. I will now close out the report with a summary, as well as a discussion on what I have achieved by undertaking this project.

In my Introduction chapter of this report I had stated that I planned to research and experiment with a few different algorithms. I looked at various implementations of D\* but ultimately chose to stick with A\* as it accomplished exactly what I needed it to do for my agent to find out where it should be going in the scene.

In the Research chapter I discussed the various pathfinding algorithms used with the *Mario AI* competition, and examples of some researchers who moved away from this benchmark and created their own systems. Now that I have completed the project, I feel I should have spent some time with the *Mario AI* benchmark. It would have been a worthy exercise as I would have seen the various components of how an AI agent works. I would have seen the methods that handle all the decision making needed to be processed for an agent to efficiently handle moving through a platforming level. I also researched other projects like *SpelunkBots API* and *Monster Carlo*. These other projects could also have been analysed further and I may have been able to produce something more tangible if I had looked at these examples from a practical standpoint, instead of an analytical one. I focused too much on the results and the theory instead of the code and the practical development, which would have been of a greater benefit to me.

In the System Design chapter, I laid out the design for the system. This was mainly done as a way for me to visualize how the various components interacted with each other. It was also a good way for me to visualize how the Behaviour Tree operated, which was a fundamental step that needed to happen before I could begin trying to create my own. My Implementation chapter contains a step by step account of my development process, and the various changes that my project had to go through, to get to the final result. I discuss the technical problems that I had encountered as well as my solutions to overcome these problems. In this final chapter I will analyse my original goals and what I have achieved by undertaking this project.

The goal of the project was to develop a tool that could be used to rapidly playtest platforming levels and I have achieved a solid prototype that has the potential to become a tool used by developers. Regrettably, I was not able to take this further and produce a marketable project, that I could showcase on my portfolio and release on the Unity Asset Store. This would, however, have been a bonus and not the core objective of the project. In my Introduction chapter, I defined my main objectives to be improving my skills and to examine and overcome of my own shortcomings. I have improved my researching skills by learning how to take a critical evaluation of a study and learning the value of a credible source. I have managed to acknowledge my own work ethic and have altered my routine to reflect this. I have also kept myself healthy, both physically and mentally to maximise my productivity and produce the best possible project that I was capable of. While I had not met all my goals, and made some mistakes along the way, I am proud of what I have produced.

By undertaking the project, I had analysed similar areas and drawn conclusions that affected my developing direction. The research that I had undertaken showed that MCTS and A\* was an effective way to navigate the level but this was only the case in the *Mario AI* benchmark. This narrowed my view and I did not explore more options until it was too late. Rather than creating a universal tool, I should have first created a specific tool and then adapted it to be universal. If I had to do this project again, I would have spent some time trying to create something in the *Mario AI* framework first before attempting to undertake my own algorithm.

I have achieved my main goals of improving my skills through the process of undertaking this project. I now have a lengthy resource that I can utilise in job interviews as an example of how I undertake a software project, as well as highlighting any process that I need to improve on for any future projects that I undertake.

I have outlined the various technical difficulties I encountered in the Implementation chapter and I will now discuss some personal difficulties like motivation and planning. The project was completed in a haphazard manner and was not the final product that I had envisioned. If I had managed my time more effectively and had set out a proper routine, that I would have strictly followed, I might have been able to produce an improved version and something closer to what I had initially hoped. The project was challenging due to having to juggle my other modules and assignments. I had to miss some classes to catch up on deliverables and felt this all could have been avoided if I had set out a proper plan in the beginning. By undertaking this project, I have shown that through perseverance and hard work, I can adapt to any situation and this is an invaluable experience I hope to bring to any workplace in the future. I have shown that my time here in college and the skills I have learned along the way have all amounted to improving myself and have prepared me for a career in the Software Development industry.

In the planning for the project, I had an Agile Development approach. I chose this method as I was treading new ground, in terms of my own abilities, and opted for a plan that could be flexible and change when it was needed. My own understanding of Agile development was flawed, and I did not stick to the schedule at all except for the documentation side which I have kept on schedule with, which greatly helped reduce the workload. If I was to undertake the project again, I would spend more time on the initial tasks as it would give a better direction of where to go and would not lead to the initial directionless approach to the development. I should have stuck to these goals, even in a weekly fashion and it would have helped create a more solid project.

In a final word I would classify the project as a success. I gained valuable experience working independently and I have challenged myself to the best of my abilities. I have learned how to research other works and learned to cut through the data available, to focus on what was relevant to my needs. The tool itself is not a finished product but still is an autonomous AI agent that can navigate a platforming level. In the coming months I hope to further develop this tool, as it will be a worthy addition to my portfolio, as well as something that could be used by Game Developers to help cut costs on testing. The most important lesson that I have learned from this project was that I have the ability to complete any task that is set before me, as long as I have set out a clear plan and have managed my time effectively, to learn and research the task in depth.

# Appendix

## User Manual

The idea was the tool could be as universal as possible. A developer could add the tool and customise behaviours based on their game. Example behaviours include shooting, crouching, using Power Ups, double jumping or gliding. As it stands the tool can only be configured to set a Goal to move to. So, the user would set up their environment and use the tool and set their desired goal, the agent would then move to the goal, jumping over obstacles and shooting enemies that our in the way. The desired implementation was the tool could be customised directly in the inspector, but this is not the case. For the purposes of the demonstration of the agent, the goal has been set to a floating coin which the agent navigates to, shooting any enemies in its way. The scene that has been set up can be customised with the Tile palette, blocks and be added and removed and the goal and be moved around to show how the agent handles different environments. More enemies can also be added by adding from the Enemy prefab or by duplicating any in the scene already.

# References

Algfoor, Z. A., Sunar, M. S. & Abdullah, A., 2017. A new weighted pathfinding algorithms to reduce the search time on grid maps.. *Expert Systems with Applications,* Volume 71, pp. 319-331.

Freitas, J. M., Souza, F. R. & Bernardino, H. S., 2018. Evolving Controllers for Mario AI Using Grammar-based Genetic Programming.. *2018 IEEE Congress on Evolutionary Computation (CEC), Evolutionary Computation (CEC), 2018 IEEE Congress on,* p. 1.

Hu, J., Wan, W. & Yu, X., 2012. A pathfinding algorithm in real-time strategy game based on Unity3D.. *2012 International Conference on Audio, Language & Image Processing,* p. 1159.

Keehl, O. & Smith, A. M., 2018. Monster Carlo: An MCTS-based Framework for Machine Playtesting Unity Games.. *2018 IEEE Conference on Computational Intelligence and Games (CIG), Computational Intelligence and Games (CIG), 2018 IEEE Conference on,* p. 1.

Keehl, O. & Smith, A. M., 2019. Monster Carlo 2: Integrating Learning and Tree Search for Machine Playtesting.. *2019 IEEE Conference on Games (CoG), Games (CoG), 2019 IEEE Conference on,* p. 1.

Kim, S. M. et al., 2017. A review of parameters and heuristics for guiding metabolic pathfinding.. *Journal Of Cheminformatics,* Volume 9, p. 51.

Lim, K. L. et al., 2015. Uninformed pathfinding: A new approach.. *Expert Systems with Applications,* Volume 42, pp. 2722-2730.

Mathew, G. E., 2015. Direction Based Heuristic for Pathfinding in Video Games.. *Procedia Computer Science,* Volume 47, pp. 262-271.

Naveed, M. K., D. Crampton, A. C. & L. Gregory, P., 2012. A Monte-Carlo path planner for dynamic and partially observable environments.. *2012 IEEE Conference on Computational Intelligence and Games (CIG), Computational Intelligence and Games (CIG), 2012 IEEE Conference on,* p. 211.

Nicolau, M., Perez-Liebana, D., ONeill, M. & Brabazon, A., 2017. Evolutionary Behavior Tree Approaches for Navigating Platform Games.. *IEEE Transactions on Computational Intelligence & AI in Games,* Volume 9, p. 227.

Ortega, J., Shaker, N., Togelius, J. & Yannakakis, G. N., 2013. Imitating human playing styles in Super Mario Bros.. *Entertainment Computing,* Volume 4, pp. 93-104.

Perez-Liebana, D. et al., 2019. General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms.. *IEEE Transactions on Games, Games, IEEE Transactions on, IEEE Trans. Games,* p. 195.

Scales, D. & Thompson, T., 2014. SpelunkBots API - An AI toolset for Spelunky.. *2014 IEEE Conference on Computational Intelligence & Games,* p. 1.

Shinohara, S. et al., 2012. *Search Algorithm with Learning Ability for Mario AI -- Combination A∗ Algorithm and Q-Learning.* s.l., IEEE.

Zeng, J. et al., 2019. Navigation in Unknown Dynamic Environments Based on Deep Reinforcement Learning.. *Sensors (14248220),* Volume 19, p. 3837.

Zoubi, O. & Al Awad, M., 2018. Dynamic Area Search with Shared Memory: A Meta-Framework to Improve Pathfinding Algorithms.. *2018 International Conference on Innovations in Information Technology (IIT), Innovations in Information Technology (IIT), 2018 International Conference on,* p. 81.