2021

AI Being Added to a First-Person Game

COmputer Science for Games

Jake Pascoe – 17828121

SuperVisor - Andrew Blake

Second Reader - James Burton

# Abstract

This project is an in depth look into coding pathfinding and how this can be implemented into a first-person shooting game. This project will look into how pathfinding works and the research behind how the algorithm could be improved to run in real time and handle multiple agents.

1. Contents

[1 Abstract 1](#_Toc71268690)

[3 f 4](#_Toc71268691)

[4 Introduction 4](#_Toc71268692)

[5 Planning 4](#_Toc71268693)

[5.1 Planning Research 4](#_Toc71268694)

[5.1.1 Waterfall Management 4](#_Toc71268695)

[5.1.2 Kanban Project Management 5](#_Toc71268696)

[5.1.3 Work Breakdown Structure 5](#_Toc71268697)

[5.1.4 Chosen Management Style 5](#_Toc71268698)

[5.2 Gantt Chart 6](#_Toc71268699)

[5.3 Planning Discussion 7](#_Toc71268700)

[5.4 Risk Assessment 8](#_Toc71268701)

[6 Literature review of Pathfinding Algorithms 9](#_Toc71268702)

[6.1 A\* Algorithm 9](#_Toc71268703)

[6.1.1 Heuristic Value 10](#_Toc71268704)

[6.1.2 Weighting 11](#_Toc71268705)

[6.1.3 Box Blur Algorithm 11](#_Toc71268706)

[6.2 Dijkstra’s Algorithm 12](#_Toc71268707)

[6.2.1 Heap Optimisation 13](#_Toc71268708)

[7 Methodology 13](#_Toc71268709)

[7.1 Chosen Pathfinding 13](#_Toc71268710)

[7.1.1 Algorithm Speed 13](#_Toc71268711)

[7.1.2 Algorithm Flexibly 14](#_Toc71268712)

[7.1.3 Performance 14](#_Toc71268713)

[7.1.4 Selected Pathfinding 15](#_Toc71268714)

[7.1.5 Nav Mesh VS Pathfinding 15](#_Toc71268715)

[7.2 Game Engine 15](#_Toc71268716)

[7.2.1 Unity 15](#_Toc71268717)

[7.2.2 Unreal 16](#_Toc71268718)

[7.2.3 Selected Development Tool 16](#_Toc71268719)

[8 Implementation 17](#_Toc71268720)

[8.1.1 Grid 17](#_Toc71268721)

[8.1.2 A\* Pathfinding 17](#_Toc71268722)

[8.1.3 Units 18](#_Toc71268723)

[8.1.4 Binary Heap 19](#_Toc71268724)

[8.1.5 Nodes 19](#_Toc71268725)

[8.1.6 Game Mechanics 20](#_Toc71268726)

[9 Testing 20](#_Toc71268727)

[9.1 Pathfinding Testing 20](#_Toc71268728)

[9.2 Amount of Agents 21](#_Toc71268729)

[9.3 Optimisation Testing 21](#_Toc71268730)

[9.3.1 Binary Heap 21](#_Toc71268731)

[9.3.2 Path Request Manager 22](#_Toc71268732)

[9.4 Path Smoothing Testing 22](#_Toc71268733)

[10 Further Work 23](#_Toc71268734)

[10.1.1 Local Avoidance 23](#_Toc71268735)

[10.1.2 Path Smoothing 24](#_Toc71268736)

[10.1.3 Decision making 25](#_Toc71268737)

[10.1.4 3D Grid 26](#_Toc71268738)

[10.1.5 Final Pathfinding Review 26](#_Toc71268739)

[10.2 Game Review 27](#_Toc71268740)

[11 Conclusion 27](#_Toc71268741)

[12 References 28](#_Toc71268742)

[13 Bibliography‌ 28](#_Toc71268743)

# Introduction

This project is an overview of how A\* pathfinding algorithm can be implemented in a 3D space inside of a first-person game. The objective for this project was to add A\* and build a game around it. This report follows the journey of how this was implemented to unity and how the A\* was changed to fit the need of this game. The main objectives for this game were to make a system that would control an agent around the map either following the player or finding other items around the 3D space. This was split into different scopes for the A\* these are as follows: optimising the A\* to run a real time, having the agents follow this path through the 3d space, having the agents decide on what path to take, having multiple agents running optimally at once, and having the ability for users to interact with the game. By optimising the A\* will be ensuring the A\* runs as fast as possible with as little memory usage, this ties in to having multiple agents running at the same time as the A\* need to be heavily optimised for this to become reality. Having the agents follow a path around the 3D space will mean having two script running side by side so each individual agent having its own path. The user interaction side of this project is the game itself this objective was added so the agents are not just running at start time but in fact are running in runtime, the sub objectives in this section is to add a player movement system, shooting system, a health system for the user, and a main menu for this game.

This decision for making this project is because I have tried to add automation to games from the first year of university but did not have the skill to be able to finish it. This project was chosen as this would channel what I have learnt with unity and game design throughout my years of university but as make it so I could have automation inside my game. As unity was the first game engine, I used I feel that this project will improve my knowledge on unity and C# coding in genal.

As this project continue it tested my skills of coding and my use of unity. At the end of this project all of the objective that had been set out where achieved as this game could run multiple agents at a time, this could have been improved as seen in this report. The A\* could run at runtime and the agent could chose a different path towards a weapon rather than the player. The player could move around the map and this mean the any user could interact with my project.

# Planning

## Planning Research

### Waterfall Management

Waterfall management is a way of planning project in a sequential fashion this means that each phase of this project must be complete before the next phase is started on. Waterfall has been around for a long time with root in the manufacturing and construction industries. This is because most part of that have to be finished before the next part is started. This is used in software development and a lot of the parts of software development need prior work done before the user can more on to the next task. The main step for the waterfall methodology is as follows: requirements, design, development, testing, and delivery. With this system the waterfall methodology can be applied to software development. The main advantage of waterfall is it is a well-defined methodology meaning it is tried and tested, the system shows the users each step of the process in very detail level from the start of the project, the project deliverables are more predictable. The disadvantage for this system is waterfall can take up a lot of time designing the planning, the requirements have less room for creativity as the user must stick to the waterfall system.

### Kanban Project Management

This method focuses on visualising all your tasks for the project and setting Work in Progress Limits. This method is useful, as it assists with the workload and limits the tasks, so the user is not working on multiples task and assists with the efficiency of completing the workload. This means you will not move a card over if there are too many things in the tasks currently being completed pile. The importance of this system is managing the flow of cards moving from one section to the next. With the Kanban method, it is very easy to spot potential disruptions in the project as that will be in the doing column for the longest. With this you can also create a backlog column so that if you are stuck on a task for a longer period, you can add it to the backlog and come back to it later in the project when there is more time to complete it.

### Work Breakdown Structure

This method takes a project and breaks it down into smaller parts and then labels them in number order, depending on how many sections there are. This system then breaks them down again in to smaller tasks that can be handle in a few days. These will be labelled 1.1 through to how many sections there are to be completed. This will be shown in a tree-like structure, so the top will be the project as a whole and each layer down will be a smaller part of what is required to complete each individual task. As WBS must be done in a hierarchical fashion, you can get stuck on part of the development and then be delayed and push the outcome to be delayed. This way of planning does make it so you can see each step of the assessment, which still needs completing. This also make creates smaller sections to manage, so it makes it more controllable, as each part is a small chunk of the overall task.

### Chosen Management Style

Out of the three planning methods above the waterfall method has been chosen for this project. all three of these methods are valid for this project but the waterfall method is the style that I have used before and make a visual system that makes sense to read as the users and is simpler to talk about in the planning section of this report. as the waterfall system can be turned in to a Gantt chart with easy this is the clear choice as a Gantt chart is great for showing how the project has been laid out and is easy to refer back to.

## Gantt Chart

Figure 1:Gantt Chart

## Planning Discussion

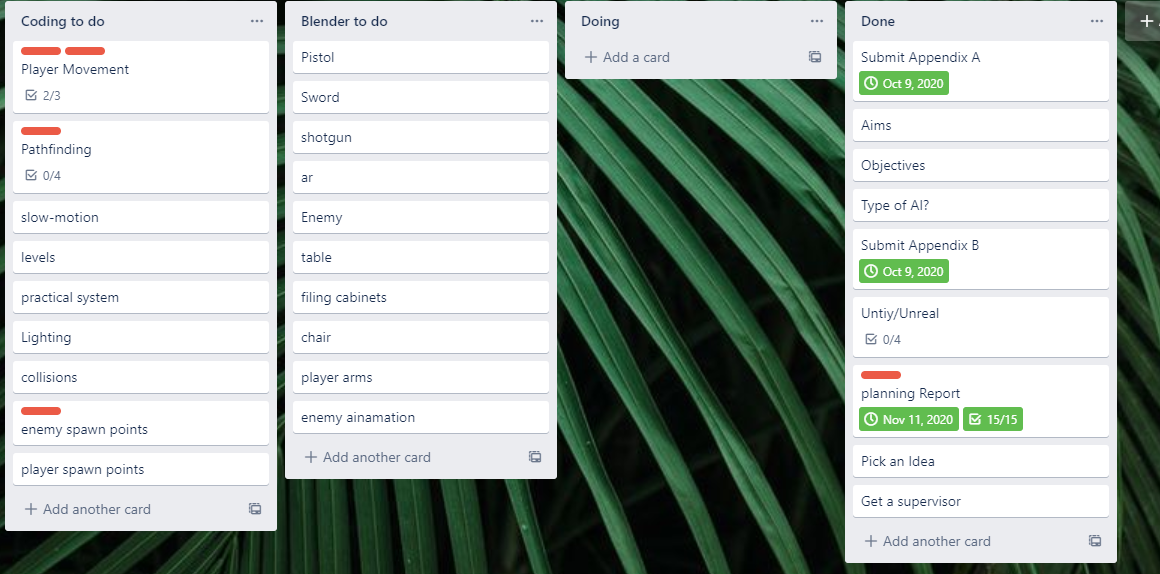
The first section of this project that needs to be completed is the planning, so each part of the project has an end date. This is so that each section could be achieved to a high standard without overlapping with the next task. This project was spilt into three main scopes, consisting of the planning report, coding, and the final write up. All parts of the Gantt chart above were organised in Trello as part of the planning report, where each task was arranged depending on when it needed to be completed. With the website, you can add cards that can be moved around depending which part the users are working on. As seen in Figure 2, the tasks for 3D modelling and coding have been arranged for completion. After this had been made, I could make a Gantt chart based off of all of the tasks in this project. 

Figure 2: Trello Page

The planning report includes finding a supervisor, summiting an ethics check, basic research, and planning draft. This all needed to be done by the 13th of November, and so each part of this report is spilt up into weekly tasks. This is so it would be finished on time and it is not left to the final few days. With this in mind, the first task was choosing a topic to base the project on, as otherwise the following sections would not exist to milestone.

The next part of this plan is setting out what part of the coding need to be done. This was selected by choosing which part of the coding would be fastest and easier to complete first. As the player movement and map design has been worked on in previous projects, this would be straightforward to add to the game as I have had practiced this coding before. The next scope of the project was to get the main deliverables for this project completed as I knew this was going to take the longest to complete, I set my self the more time for this as this part that needed to be completed for this project to be a success. This section was split with five main parts, including being adding the grid, adding A\*, agents following the A\*, and optimising the A\*. The order of these were chosen as each part follows on from the last and therefore improves. I also left time at the end of this planning for bug fixing and improvements in the game. This decision was made as even if the plan were followed perfectly, there would still be issues with the game at the end. With this in mind, I wanted to make sure there was a bit of extra time at the end of the project to fix any major bug in this game.

The final scope of the planning is the is writing the report. This is parallel with the coding as both need to be done for this project to be consider a success. Due to the nature of the project, the write up can only commence once the A\* portion of the coding had been completed, reason for this is because a lot of the write up consist of explain why the coding has been done and what has been done in this project, so as long as the A\* part of the coding was done the report could start. The report planning split this into five-part planning, literature review, methodology, implementation, testing, and further work. With this project, the literature review came first as some of this has been completed in the earlier section of the project, the main part of this was to improve on what had been done in the previous part of this project. After this, the methodology could be started as this uses the information that has been found in the research section. Implementation and testing are done at the same time this is because both of these sections require no research as this explains what has been done in the project and how this works, this makes them one of the easiest parts of the project. Finally, was the further work, as this part of the project improves on the implementation and the research as this requires both of these sections to be completed for this to be done. As this section talks about what could be added to the project as time goes on.

## Risk Assessment

Main Risks with working on a computer for long periods of time are covered by the health and safety (Display Screen Equipment) regulation act that covers any risk for a user who uses any DSE for an hour or more.

|  |  |
| --- | --- |
| Risks | How to avoid Risk |
| Eye Strain for look at a screen for too long of a time period | Make sure the user takes regular break in this case a break every one hour, adjust screen brightness so the glare does not injury eyes. |
| Back strain for not have the correct posture | Make sure the user is following the DSE recommended sitting posture. This can also be fixed by making sure the computer monitor is at the correct height |
| Repetitive strain injury | Maintain good posture, take regular breaks |
| Stress | Make sure a little work is done every day, so the workload does not get too big. Also take regular break of something the user enjoys taking their mind off of the university work. Also make sure the user is having enough sleep, so they are not tried. |
| Mind fatigue | Take breaks while work and also plan rest day where the users do not have to think about the university work. |

All of these risks are accounted for over the run time of this project and have been in force so the user does not encounter any of the risks in this list.

# Literature review of Pathfinding Algorithms

## A\* Algorithm

The A\* algorithm was proposed by Hart, Nilsson, and Raphael in the year 1968 (Hart et al., 1968) in this paper the writers suggest a new method of working out the shortest part to any node on a graph. This new method is the heuristic value of each node; this is an improvement on Dijkstra’s algorithm. The lower the heuristic value is the close A\* will follow Dijkstra’s algorithm but as the value increases get close to the true distance of the path A\* will become more optimal and processed to find the shorter path (Millington and Funge, 2009). The heuristic value h(n) is the true value of an optimal path from the target node(*t*) to the current node(*n*) (Hart, 1968 #34). As this is only part of the equation to work out the shortest path from *n* to *t* the other factor in this equation is the actual cost from start node(*s*) to current node (*n*) this is defined as g(n). Each node is set a movement value as the pathfinding moves through the node this will increase g(n).

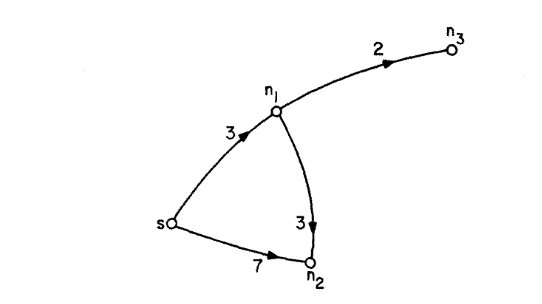


Figure 3:Subgraph For A\*

The graph in Figure 2 show four nodes as the pathfinding will start at *s* this means g(s) = 0 but as A\* moves through the graph, it will increase. With this g(*n*3) = g(*s*) + g(*n*1) + g(*n*3) = 0 + 3 + 2 = 5. The final part of this equation is the evaluation function f(n) this is the two values before added together.

This equation proves that as the heuristic value lower this algorithm acts more like Dijkstra as this only uses g(n) so the closer the heuristic is to zero the more the algorithm is acting on g(n). With this the algorithm changes and chooses which path takes it closer to the target goal.

This paper (Hart et al., 1968) set out how the algorithm can search:

1. Mark as “open” and calculate f(s).
2. Select the next open node n whose f value is the smallest resolve ties arbitrarily, but in favour of the best node.
3. If not, mark n “closed” and terminate the algorithm.
4. Otherwise, mark *n* “closed” and apply the successor operator τ to *n*. Calculate f for each successor node of *n* and mark as “open” each successor not already marked “closed”. Let f(*n*) be the actual cost of an optimal path which got through n, from s to a preferred goal node n. determination of f(*n*) is the primary interest. Remark as “open” any “closed” node n which is a successor of n and for which f(*n*) is smaller now than it was when n was marked as closed.

with this A\* can search for the shortest path from the start node and the target node and always find the quickest path.

### Heuristic Value

A\* behaviour will change depending on the heuristic value and how the user decides to calculate it, it the terms of games this will be the difference between speed and accuracy. With most games the user wants the algorithm to run faster than it is accurate as the user wants a distance that is close to the correct distance but not exact for this trade it will make the algorithm run faster. There are a few main ways of calculating this on a grid-based map; these are The Manhattan distance and Euclidean Distance (Stanford.edu, 2021). All of these have uses on different types of grids.

#### Manhattan Distance

The Manhattan distance was designed for working out the distance between to point on a graph but was used in the city of Manhattan as well (Han et al., 2012). This distance is defined as

This only works if the two points are connected by a right-angle triangle otherwise this equation cannot be used. The reason for this is because it will follow the opposite and adjacent rather than the hypotenuse (RedBlodGames, 2010). This is best for a grid that only allows movement on a square grid in 4 directions this is because of what is stated above about the right-angle triangles (Szabo, 2015). Below is an example of the code that is used to implement this.



#### Euclidean Distance

The other way of calculating the distance between two points is the Euclidean Distance. This is based on Pythagoras theorem and will calculate the distance of the hypotenuse of the triangle made from the two points on the grid (Han et al., 2012). This is defined as:

This will only work on a 2D grid as it needs a right-angle triangle from the equations to work. This is best on a pathfinding system that allows pathfinding to move in any direction (Stanford.edu, 2021). Below is an example of the code used to implement this.



### Weighting

Weighting in A\* Pathfinding is a way to change how the pathfinding makes decisions while running. This can make the pathfinding avoid areas the user does not want the pathfinding to go such as grass and water area on a map. This can be done by changing the equation that is used to pick the next node (Algfoor et al., 2017).

With this equation above the pathfinding will get the heuristic but then multiply that number by the weighting system that has been put in. Below are images from the testing of this system. on the left the pathfinding system without the weighting this shows the pathfinding ignores any path and road that has been added to the map, the right shows the pathfinding with the weighting system added to the game. With this system the users can make pathfinding think the shortest path is along the roads and paths not across the grass.



Figure 4a, b: Example of path weighting

### Box Blur Algorithm

The box blur algorithm is normally used for photos and making the pixels smoother when the picture is being processed but as this works with pixels the method can be changed to work with the nodes on the grid in a game. This system works by taking 9 nodes and getting the average of all of these nodes (Elye, 2020) (Ngo, 2020). Below is the basic matrix that this system uses.

This works with the grid when the weighting needs to be smoother, so the agent walks along the middle of the path. In this it will take all neighbouring nodes/pixels and get the mean of those pixels. This makes it so when there are 2 pixels/nodes that have very different values it can make it more gradual as the values change (Elye, 2020). This system can be used on any size of matrix depending on what the user needs. As the figure 4a below shows that when the box blur is not active the path and the grass are just black and white, figure 4b shows when it is active the paths blended towards the centre of the path.

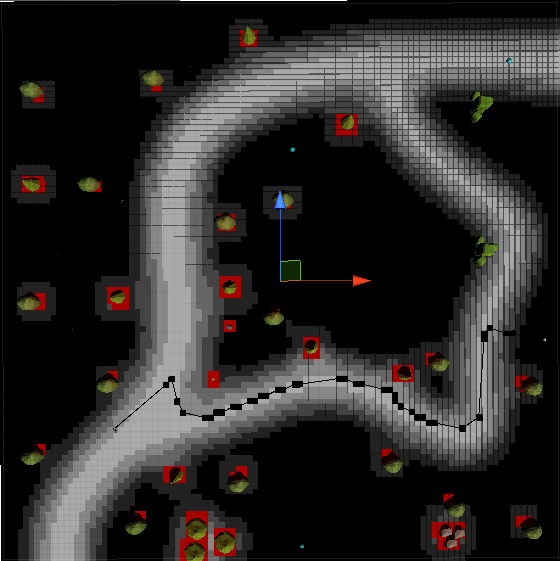
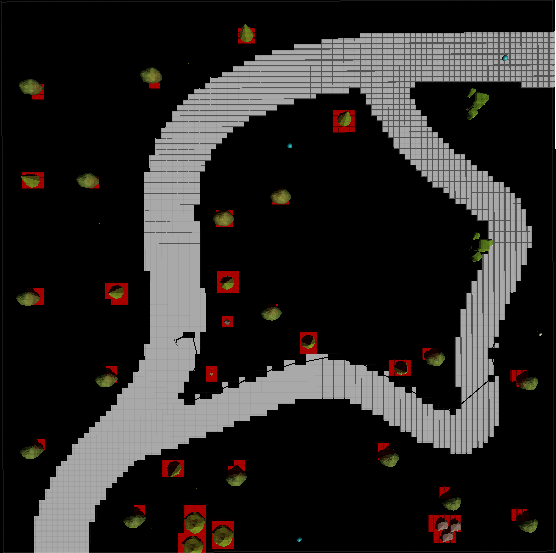


Figure 5a, b: Example of Box Blur Algorithm

## Dijkstra’s Algorithm

Dijkstra’s algorithm proposed in 1959 by E.W Dijkstra and is a way to find the shortest path on any graph using weighting systems (Dijkstra, 1959). The weighting system works by setting a value for all edges on a graph; these edges go from on vertices or nodes. This pathfinding system works by checking most nodes in a graph until it finds the target node and retraces its steps back to the starting node.

Diagram

Description automatically generated

Figure 6: Dijkstra Subgraph

Above in figure 5 is a simple view of a graph that Dijkstra could work through in this example the algorithm would want to find a path from node A and node G. for this algorithm to work all nodes and edges are added are add to the unexplored list, once the algorithm starts node A or starting node is added to checking list. Then these two steps are followed until the shortest path is found (Dijkstra, 1959).

1. Consider all Edges that are connected to the node that have just been added to the checking list, then check if there is a shorter path to any node connected to these edges that are in the explored list and save that path if the path is not shorter reject that path. If the node is in the unexplored list add that node to the checking list.
2. Move to the next node with the lowest path cost from the starting node. If the current node equals the target node the algorithm has found the shortest path if not go back to step one and repeat this until the target node is found.

With this system the algorithm would have found the path A, B, D, G with a cost of 5. this is the system that A\* builds of (See 4.1) as A\* uses the heuristic value (See 4.1.1). this system uses just the g cost in A\* so it will take longer to find the path as it is not moving towards the target node, it just follows the lowest cost so far.

### Heap Optimisation

Heap Optimisation is a way of speeding up Dijkstra’s algorithm that uses a system to change where the values are in the checking list, so the lowest value is always checked. Next there are a few ways of doing this. The simplest way of doing this is with an array that does not sort any of the data that is put into it as it is last in first out, this method can work for smaller graphs but as more nodes are made in the graph this method becomes slower. Below is the equation for this where O is the time complexity of the algorithm and V is the number of nodes. (Dasgupta et al., 2006)

The other method is the binary heap that changes the order in the array, so the lowest value node is always first to leave the array. The time complexity of this method is as follows. This V and O are the same, E is the number of edges in a graph. (Dasgupta et al., 2006)

These two methods are uses depending on the graph if the graph has less edges than node2 then the array method works faster but if the edges are more than that binary heap should be used. (Dasgupta et al., 2006)

# Methodology

## Chosen Pathfinding

This section will summarise the literature review and by the end will explain why the chosen algorithm has been selected. This will explain the speeds of the algorithm, flexible of the algorithms, and performance of both.

### Algorithm Speed

Both algorithms are good and can be used in their own way, but the Dijkstra algorithm falls short when the game map becomes too large as there are too many nodes and edges. In the chosen game there are 1002 nodes on the map with each node having four edges. A\* time complexity is where b is the branching factor and d is the death of the target node in this tree. This is affected by the heuristic value as the closer this is to the true value the smaller the branching factor will be as the user wants the branching factor to be as low as possible as d increases it takes roughly 10 times longer for it to search. Whereas Dijkstra does not have this if the user is using the binary heap the time complexity will be as this algorithm searches more node and edges than A\* it will take longer. This is also a factor when the graph is getting bigger and increasing the node and edges. With this in mind A\* will be a better pathfinding with time taken to find the shortest path. Another reason why A\* is faster than Dijkstra algorithm is because A\* searches in the direction of the target node seen is section 4.1.1 with this the algorithm does not need to search as many nodes making it faster to find the target.

### Algorithm Flexibly

The A\* algorithms can be manipulated as the user needs different maps and different sizes. This is shown by the heuristic and weighting of this algorithm. This makes the algorithm take different paths that are not the shortest part but can be when paths and water are added to the map. This means A\* can be made to make paths that are more realistic and truer to AI. Where the Dijkstra algorithm does not have these features and is harder to find these paths that are needed for games that are not mazes. As this system only uses one cost that is so far this makes it hard for the user to have the pathfinding go on any other than the shortest path. This would be good if the map that was made was a maze but as there are different routes to the player this pathfinding does not fit with this category. With this section A\* performs better flexibly.

### Performance

On this section I found a website ( [https://qiao.github.io/PathFinding.js/visual/](https://qiao.github.io/PathFinding.js/visual/%20) ) that has pathfinding algorithms added on to it so the user can test the speed, accuracy, and number of operations. On this website I tested A\* with Manhattan and Euclidean distance and Dijkstra’s and recorded to results in the table below.

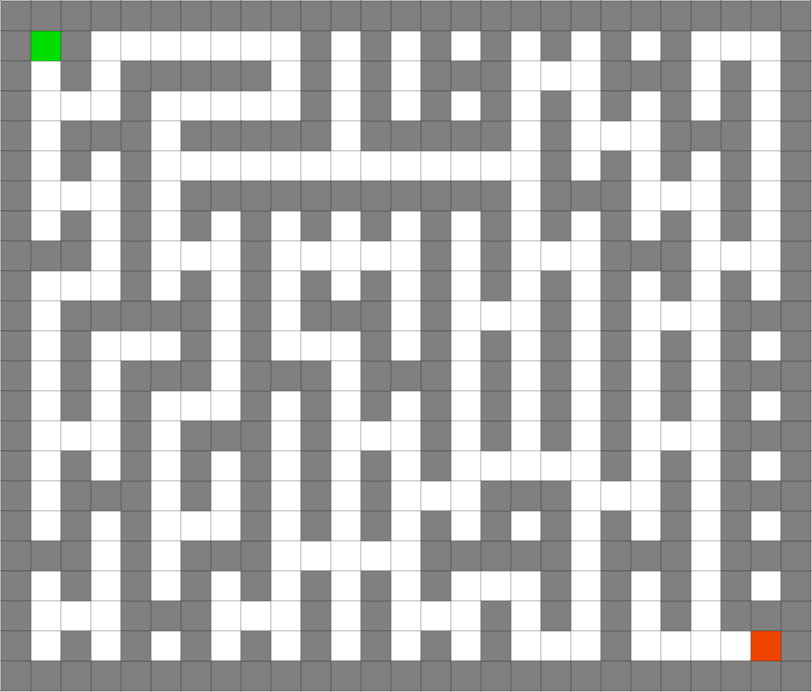


Figure 7: Test Maze

Above shows the results from the test of the website. This shows that the Manhattan distance works the best with this maze that has been made. As this pathfinding has the shortest time and lowest nodes check this makes it a good choice for a pathfinding that has to be run multiple times a second during the run time of the game. Dijkstra still finds the short path but takes longer than the Manhattan distances and checks more nodes. The reason why many nodes are checked is because all of this takes memory, and this will affect how the game runs and how quick the AI can change directions.

### Selected Pathfinding

The Pathfinding picked for this is A\* as this works a lot better than Dijkstra is all categories this mainly comes down to the fact A\* is an improvement on the Dijkstra algorithm and has added features which makes the algorithm perform better. As this pathfinding works faster, works better on bigger maps it checks less nodes and moves towards the target node. With these test it makes A\* they obviously chose out of the two pathfinding algorithms.

### Nav Mesh VS Pathfinding

These two systems work similarly as they do the same pathfinding. The main difference is that the nav mesh is built into unity and can auto generate a grid that avoids any obstacles. This is in place as a way to have agents in the game without having to add a pathfinding algorithm. This makes making a game easier to do. This does have a weighting and cost system where the agent can move along paths. My goal going into this project was to make an agent that uses a pathfinding algorithm, and this was looked into but does not fit the requirements for a final year project as unity hands all of the systems. This also cannot be optimised to use as little memory as possible. Both are viable options when making agents in a game but as I am doing a coding course, I want to make code and add a system to find the shortest path without a system in unity.

## Game Engine

### Unity

Unity is one of the most popular game development tools for indie game developers. The reason for this is because it is easy to start learning and to use as the interface is simple and easy to get around. The main part of unity is with its 3D physics engine, lighting system, and the prefab items. The 3D physic engine called Nvidia PhysX is an engine that can run off of the user’s graphics card and make the game less stressful for the CPU. This system can handle any simulation the users can think of as long as its optimism. The lighting system which can run at real time, has the lighting baked for non-moving objects and this can also be done at run time. With a good reflection and shader system to make the game look a lot better and can also make the game run smoother. The prefab system is the best part of the system as this lets the users save all the values and scripts on a game object. This object can be used multiple times and will all have the same properties each time. This is good for code reusability as the user does not need to rewrite code for each item and make classes easier for the user.

With unity having such a large community this means that there is a system that has been made to improve unity. This is with the asset store or with plugins that the community has to improve or cut down development time. With the asset store the user can get anything from scripts to 3d models this makes the development process easier as anyone can get into it. The plugins make the light and build maps much easier as the users can use this to make whatever is needed. Unity works very well with GitHub and can be pushed to the users’ repositories because of this system the user should not lose the project and will always have a backup to go to in case anything goes wrong with the project. this system can be used with a plugin in unity or with just the desktop app on windows.

Unity has a very good animation where in the application the user can make all animations for their game this does need a model that has been made in another modelling software and is rigged up with a joints and bones once this is set up the user can make all assets in this software. This also includes a visual coding like system for animation switch such as running to walking. This makes it easy for all user to make smooth animation for all characters and models in game.

### Unreal

Unreal is another popular game development platform which offers more features than unity as this has modelling software where the user can create the models inside of unreal and has a very good rendering system where all materials can be made from unreal. This means that this system can be used for everything for making a game. This does make using this system very hard to understand when the user first opens and uses an unreal engine. This engine is built for multi users’ games and is designed for pipeline building. This means that each part of the game is made by someone else. And move one to the next person. With the modelling tool the users can also make all the animations for any model that is built unreal. With key framing and also with a blueprint animation system. This cuts out the use of other modelling systems and exports these assets to unreal. With this modelling system the unreal engine has hair and clothing tools so the user can easily make all parts of the game and any character in this system. This is so the users do not have any export issues and will always have working models.

One of the best reasons a lot of people use unreal is for the blueprint system this means that anyone can make a game without having to write any code. This is a visual way of coding. This means the users can add code blocks to do functions. With this system it makes this engine more inviting for new users with this said most users will not use all of the advanced systems in this engine as there are too many to use all of them. Blueprint is a good system for users when they first start off and there are plenty of tutorials for this system with a smaller community than unity but still big enough for a relatively new user to unreal to make a game with this system.

Another main system to unreal is the built-in multiplayer framework which makes creating a multiplayer game very easy as the system comes with a template for multiplayer games. This can be with client-side hosting and with server-side hosting. With this system in place on unreal it makes multiplayer games simple and easy to set up.

### Selected Development Tool

Both of these engines are valid choices for this game. The main difference is that unreal is built for bigger groups and for bigger games whereas unity is made for smaller teams like indie games. One of the reasons I have picked unity over unreal is because of how much I know each one. In this case I have made multiple game in unity and have only made one game in unreal this means I know unity’s systems a lot better and can make a lot of the game without research whereas in unreal I would have to do a lot of research to make the basic parts of this game as I do not know enough about unreal and C++ to feel confident using this engine and language. Another reason is because unreal has too many features that are not needed in the project where unity has a smaller number of features that will improve on my project. unreal also has a very long loading time this causes me to waste time with waiting for the system to boot whereas unity does not have this issue and can load it up much quicker. This also comes down to blueprint vs code as I wanted to code this game, I would not want to use blueprints as how they are set out confuses me and cannot see what the code is doing this is why I have opted to do all of my project in code as I find it easier to see what is happening and why it is happening. The final reason is to do with the community around unity and unreal, booths have communities, but unity has a much larger one and that means it is much easier to find and ask for solutions when I am stuck in this project. There is also a much larger asset store for unity this means as I am not confident at 3D modelling, I can find free assets on this store to add to my game where in unreal it is not as easy to find answers on your problems and to find free models for projects.

# Implementation

This will explain how the code was done and how this was added to the game. The part this will go over will be the grid creation, pathfinding, moving the agents, and binary heap optimisation.

### Grid

The grid script is designed to map out where the nodes are placed on the map and to set the values of the node on the map. The first part of the script is creating the grid on the map that is set to make the grid to the size of the map and can be changed depending on how the developer needs it. This part also sets the size of each node; this is done by setting the value of a node radius in the unity inspector window. Along with this in the awake function the terrain type is set this is part of the weighting system, this uses a ray cast to check each node and see what layer mask it is overlapping with. In the inspector the developer can change what layer is set and what the value is for this layer mask. This data is stored in a dictionary. The main reason for the dictionary is because 2 bit of data can be added and only one needs to be called to get both bits of data. When the ray cast checks each node in the grid it will try to get the value from the dictionary and output the data of the movement penalty that is set in the inspector; this value is then set to that node.

When the create grid, function is called the function will find the bottom left corner of the map check if the point of the grid is not a wall by checking around the node to see if the unwalkable mask is in the radius then if that node is walkable, it will check what the movement penalty is using the method above. Then this set the node using the node class with the data of; if it's walkable, the X and Y positions and the movement penalty of this node. This runs until all points of the grid have been done. This then gets passed to the box blur algorithm (See 4.1.3).

The Box blur algorithm is a way to get the average value of 9 bit of data in a matrix. The reason for this in this game is because of how the argents move before this point. The agents would just move along the side of the path as this was the shortest path with the lowest weight value. This algorithm combines all values from 9 nodes adjacent to this node because of this the middle of the path becomes the lowest weighting on the path. With this the agents move along the middle of the path and the path becomes smoother.

### A\* Pathfinding

This script handles the A\* and where most of the calculations for the pathfinding is done. The main part of the script is the coroutine that uses the algorithm to find the path for the agents in the game. These take two vector3 positions on the map; these are the agent's position and the player’s position. When this is called these positions are converted from vector3 to nodes on the grid. With this data the algorithm checks if both nodes are walkable. The reason for this is so the algorithm does not run either the target or start cannot be reached as it would waste memory and would not need to run. This then opens a heap list[[1]](#footnote-1) for sorting the nodes and for any open node and opens a hash set for any nodes that have been closed and are not needed anymore. This then adds the start node to the open heap. This then only runs the algorithm while the open heap has more than zero items in it as if there is nothing in the open list there will be nothing for the algorithm to search. This will also check if the target and current node are in the same place, so the algorithm does not run when it is not needed to save on memory. This then checks all the neighbours of the current node while also checking if any of the neighbour’s nodes are in the closed list and walkable as there is no need to check nodes that have been checked and are not walkable. This then gets the G cost of that neighbour node and the H cost multiplied by the movement penalty of that node and add the two values together to get the F cost of that node. It will then check if the neighbours G cost is lower than the current F cost if this is true then the pathfinding will move to that node and run this again. This also sets the neighbour node G cost to the current F cost and sets the H cost again; it will also set the neighbour node to a parent of the current node. This is so the pathfinding can retrace the path and find the shortest path.

This script also handles the simplify path, this was added for testing so when developing I could see if the pathfinding were working correctly. This just makes it so the on-screen gizmos can be seen. This checks if the path has changed direction and will draw a new line to the next node. This makes it easier to see the path. This converts the last node and the new node and checks the direction it is going in if it changes and adds this to the final path array for the retrace path.

The retrace path function just calls the simplify functions and then reverses this path that has been given back from the simplify path function. This is so the unit script can know what path to take the agents on.

### Units

The unit class handles all agents and requests the path for it to follow from the pathfinding script. This script first tried to request the path in the update function. This would request a path sixty times a second. I found this caused the game to lag and it would not be able to handle more than two agents with a push. This was changed to use the invoke repeat function works like the update class but the developer can change how often the function is called this was tested with a few values the best was found to be 0.5 seconds this made it so the agents could get close to the end of the path before a new path was requested. This then checks for visible weapons and will compare the distance of each weapon in the agent’s field of view and find the closest one to the agent. This then calls the function choose path that takes the distance of the player and weapons in the range of the agent and works out which path to go on. If the weapon is closer it will go towards the weapon. If there are now weapons it will always go to the player. Once a path has been found it will follow that path by moving the transform of the agent towards the next node until it has reached the target node then the pathfinding will be terminated until a new target node is made.

### Binary Heap

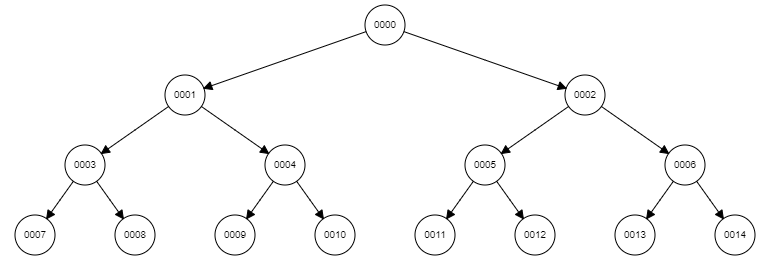
The binary heap is one of the most important bit of cade. As this sort where were each of the node are stored in the list as this make the node with the lowest F cost be at the top of the list. The reason this is important is because when the pathfinding is searching for a path it will always want to check the node with the lowest F cost first as this will bring it closer to the target node. This is done by having each node have two child at a point. The rule of this is that each child has to be less than the parent node. The reason this is importance is so the node with the lowest value is at the top. This is done by adding a new node to the list if the node has a value bigger than the parent nothing happens. Id the parent has a higher value than the child there are swapped until the node has found a point where it is less than its than is child node and more than its parent node. If there is no node that is less than this node then it must be the lowest node in the list. Because of the way this is done less node have to be checked to find the lowest node in the binary heap. If this were not in place and an unsorted array was used the code would have to complete every node. this is fine when there are only 5 nodes in the array as it will only have to check 4 different nodes but when the amount in the array increases the time will increase. 

Figure 8:Binary Heap Tree



Figure 9: Binary Heap in Array

Once the first node in the heap has been removed the code will take the last value in the binary tree this will be the bottom right-hand item and move it to the top of the tree and check each of these node children and swap it with its lowest valued child. Below is how a binary tree is visualized but it is really stored as an array. To find each point in the tree the program only needs to find the children of itself, for the left child its and for the right its this will always find where in the array the child is. The way to find the parent is . This words because when the program divides an integer and gets .5 it will round down to the closest whole value.

### Nodes

The node class handles all the node data. This is what is used in all of the code explained above, this code set the value of each node. This is mainly done in the grid script, but the pathfinding and units scripts use this data. This is where all the data for is the node walkable, movement penalty, position, and gird position. This also holds the data for the G, H and F cost. It is also used to work out the F cost by getting the G cost and H cost and adding them together. This script also has a function to compare nodes together to see which node is more promising to go to next; the pathfinding script uses this function. The main use of the script is to be a constructor class for the grid. This also always allows the node to know which node is its parent so the path can be retracted, and the shortest path can be found.

### Game Mechanics

#### Player Movement

The simplest part of the coding in this game is the player movement. This system takes and inputs from the keyboard, this is handled in the input system inside of unity. With this system, the user can set what each button can do and the name for that command. In this system the player script has two floats these get the horizontal and vertical values of the player and these are increase by the speed value when the movement buttons are pressed. The jump command checks to see if the player is on the ground if this is true then the player can press space to add a force in the upwards direction. The running working but changing the value of the speed to a higher value white the left shift is being held down.

#### Shooting Mechanics

The shooting mechanics uses the unity ray cast system, this just draws a line between the start point and the range that has been set this will then return the value of what is has hit. This is used to check if it has hit the agents in the game if this is true it causes the agents to take damage until the health is at zero where they will flop to the ground and be removed from the map.

#### Spawning Mechanics

This part of the game will spawn a wave of agents around the map at random locations. This works by setting the enemy count and have a while loop running while the enemy count is lower than a set value. This will then find X amount of random X and Y positions on the map and instantiate the agents around the map and increase the enemy count by one.

# Testing

## Pathfinding Testing

Testing the pathfinding was to see if the A\* algorithm could find a path from point A to point B with no obstacles on the map this was just to see if the system can find a path from one point to another. This was done by making the nodes in the grid visible and once the path was found these nodes would turn red showing the route of the path from A to B this would prove that the system can find a path.

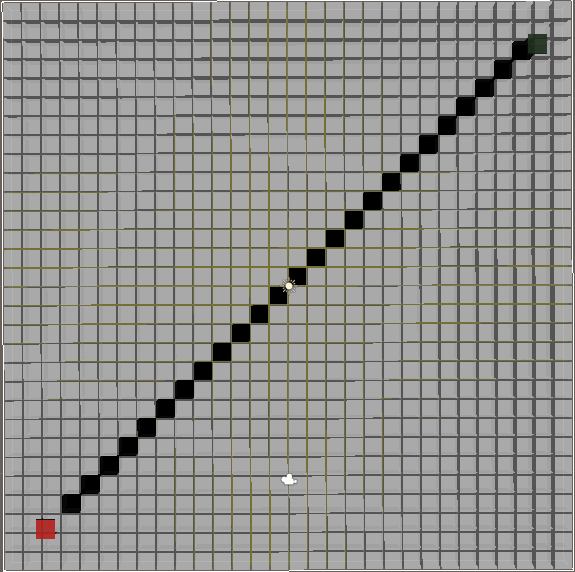


Figure 10: Pathfinding Testing

This then moved to adding an object to the map to see if the pathfinding can find a path with an object around it. this was still done by showing the grid and checking if the node will check the layer mash of the object as walls this was done by making the grid show any nodes will a wall in it as blue this would just check if the node would detect the will in the area of the node. This was then tested with the pathfinding with the same method as above showing the path as red. This was to make sure the pathfinding could find a path around the object in the map.

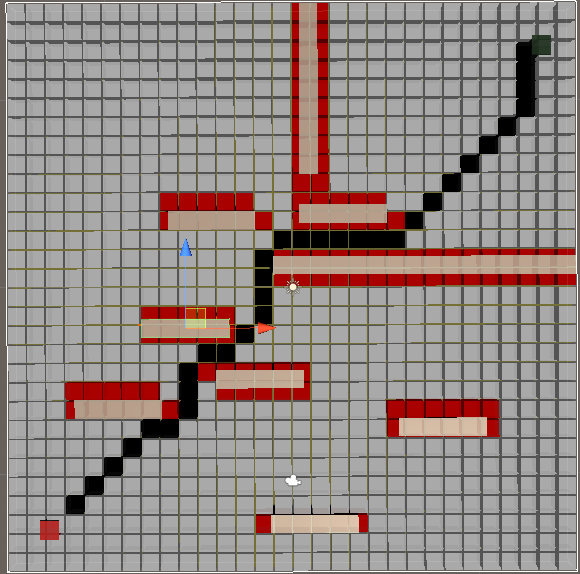
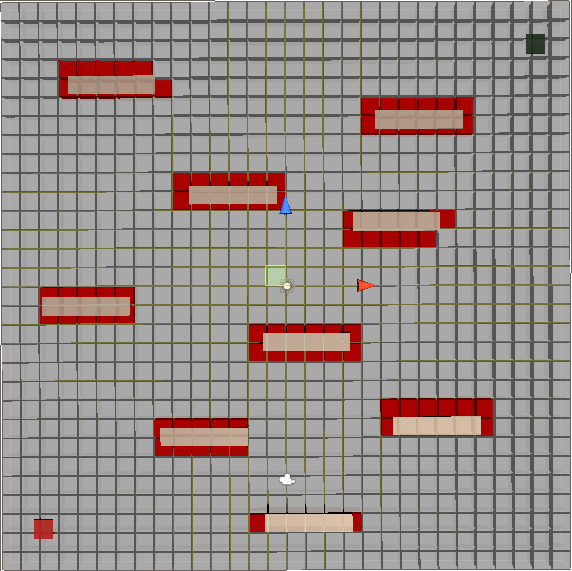


Figure 11a, b: Pathfinding Wall Detection Test

Finally, the pathfinding was tested to see if the pathfinding could find a path of an object that is moving. This was done by changing the pathfinding code to update while the game was running and move the target and seeing if the path changed as the target moved. This work finds with just one pathfinding running.

## Amount of Agents

This was a stress test for my game and what has been made. For this test I add a new agent every time to see how many agents the game could run at one time until the game started lagging, or the pathfinding stopped working. During this test I found that the max number of agents in this game was 15 anything more that this would start to lower the frame rate and lower the performance of the pathfinding as the engine could not handle the amount of requests for a path that were being sent.

With the original code I found as soon as 3 agents were added the pathfinding would not run as well this is where I found that requesting a path sixty times a second was too much and had to lower the amount of time the path was requested per second.

## Optimisation Testing

### Binary Heap

This was this to test if adding binary heap will improve the speed that the pathfinding will find the shortest path this was tested by checking how long the pathfinding took to find the path without binary heap and test how long the pathfinding took with binary heap. With this test the path took 25ms without a binary heap and took 5ms to find the path with a binary heap, so this was a success and was added to the code.

### Path Request Manager

This system was added so more agents could be running at once. This was tested by seeing how many agents can be added before a path request was added which was 5 agents before lagging and once this was added the game could handle 15 agents this system proved to be a success and was kept in the game as more agents could be handled at runtime.

## Path Smoothing Testing

This is a system to make sure the pathfinding goes along the paths I have created in the game and not along the grass. When this was first added the test was to find the best values for the grass, path, road, and water. This was done by setting the road and path as zero as this is where I want the algorithm to follow this route and only leave it when it has no other choice. The values I had to figure out is how to keep the algorithm off of the grass and the water. This is because in the map there is a point in the map where the path can be cut across below you can see this part inside of the red circle.

|  |  |
| --- | --- |
| Weight of Grass | Does the pathfinding cut across |
| 0 | Yes |
| 1 | Yes |
| 2 | Yes |
| 3 | Yes |
| 4 | Yes |
| 5 | No |
| 6 | No |

As this table above shows that when the weighting value for the grass is above 5 the pathfinding no longer moves across the grass as this is what I want the this is the weighting the grass will stay at as it will no longer cut across the grass.

Another test that had to be done while designing this code is the box blur algorithm this makes the agent walk along the middle of the path this had to compare against the pathfinding when this was not in place if the agent did not walk along the middle of the path, then the code had to be changed until the agent would walk along the middle of the path.

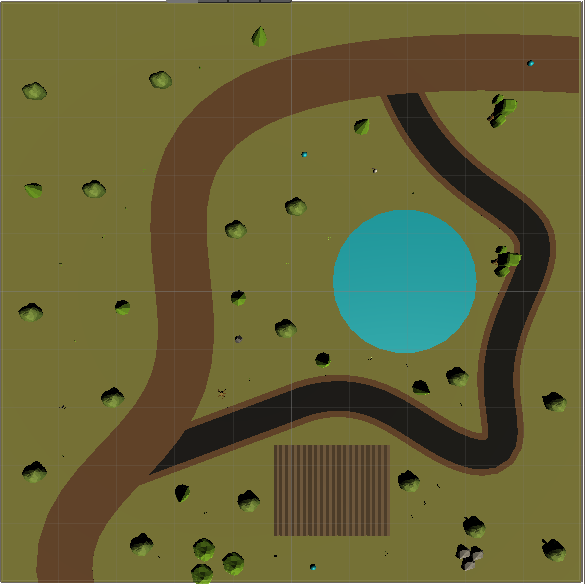


Figure 12: Map Design and Weighting Test

# Further Work

### Local Avoidance

This is an improvement that would have made the pathfinding a lot better than it is as this would make it so the agents in the game would not enter to same node as another agent this would make it so when the game is being played if the agents are not remove of the map, they end up bunching up in to one section and all follow the same path. After doing some research on this I found a paper that goes over what I would have needed if I needed to know about this problem soon in the project. This paper explains how collisions can be avoided by using the velocity of each agent.

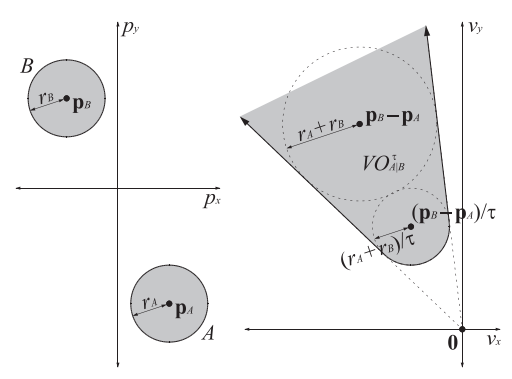


Figure 13: Velocity Avoidance

This figure shows how agent A can avoid agent B. This is done by making a truncated cone from point 0 also where agent A is. This cone is made by having the legs tangent to the disc of the radius of rA + rB at a point if pb – pa and the disc of the radius (ra+rb)/t this means as the time constant increase agent A will not hit agent B (van den Berg et al., 2011). This means that if the agent is traveling in a direction that enters this cone the two agents will collider with each other. This algorithm used the velocity of each agent to work out what direction it can go to avoid a collision with other agents. With this system the user can make a system where all agents can get from point A to point B without hitting another agent. When this is done for agent A and presumes agent B is not moving and will check this angle again at the next time constant. Below is the graph for a moving target where agent B has selected its velocity this time the algorithm has to check each point of the area agent B could end up in (Dark Grey) and perfume the same calculation as before in Figure 12. This will equal an area (Light Grey) where agent A cannot move in (van den Berg et al., 2011).

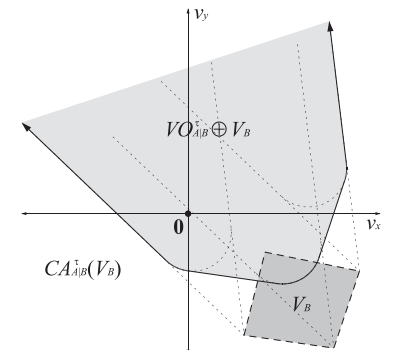


Figure 14:Velocity Avoidance on Moving Object

With this system in place it would make my game agent move in more realistic movement and not go on the same path as each other. This system was found later into the project and seems very advanced to add into this project this is why it was not added to the project. Unfortunately, this was not added as it would have improved my agent and made the game a lot better. While researching I did find a library that could have added this to make a game, but I wanted everything about the agents to be code that was created by myself.

### Path Smoothing

The path soothing in this game works quite well, but this can be improved on as this method just has the agent moving from one point to another in a straight line. With this system the AI can look quite unnatural as it is following each waypoint and does not do any gradual turn. Figure 14 shows how the agents in the game move as they just move in one direction for this to be improved. The agents would have to start turning before they hit the node.

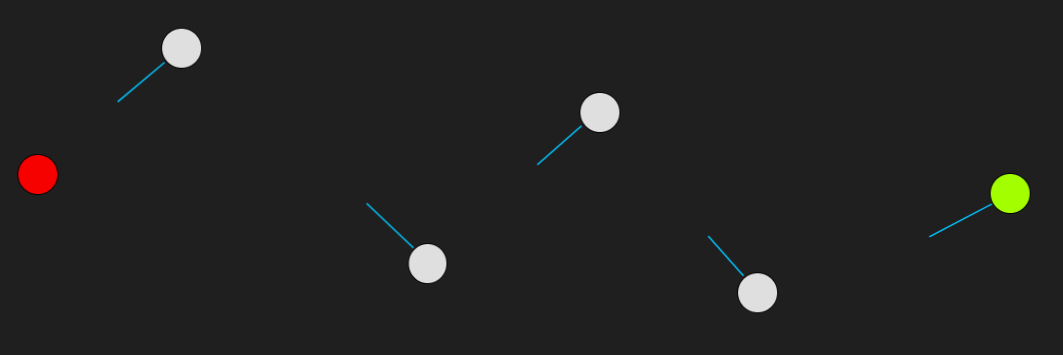


Figure 15:Showing How Pathfinding Move in Game

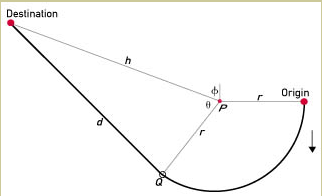
This system works by calculating a turning radius for the agent in the game. This works by finding the centre of the turning radius which would be the radius of the turning circle. This can be done by setting a distance from the node when the agent will start its turn. Once this is done the agent being turned with the turn radius that has been set by the user all it needs to do now it works out how long to keep turning for. They way this is done is by having the code work out the distance from the centre point of the turn radius and the next location this will be defined as h this will make a right-angle triangle with the radius of the circle and to work out the length of the straight line, we use Pythagoras theorem to work that out. The only part left is to work out the angle from the point the circle is made to when the agent needs to start on the straight line. 

Figure 16: Turning Line

This feature could not be added to the game as I do not know enough about circle theory to have this in my game as the website this information explained it a little, but the user needs good knowledge of how circle theory worked to get this in the game. this could have been done if I had found this reason sooner and gave myself more time to do the research on this so I could fully understand how this system works and included it into my game.

### Decision making

The agents in this game have good decision making where they will choose a path depending on what is in their line of sight and what is closer. This could have been improved by having a weighting system on the guns so the agents could line up better guns and decide which gun is the best. This could be done by having a weighting system on each gun and making the agent look for the gun with the highest weighting system. With this the agents chose to change what path they are going on not only on what is closer but what is considered the best gun on the map. This would make the agents react more to the world around them. This could not be added to the game as the game had been finished and the report had to be finished and the coding was done for this project.

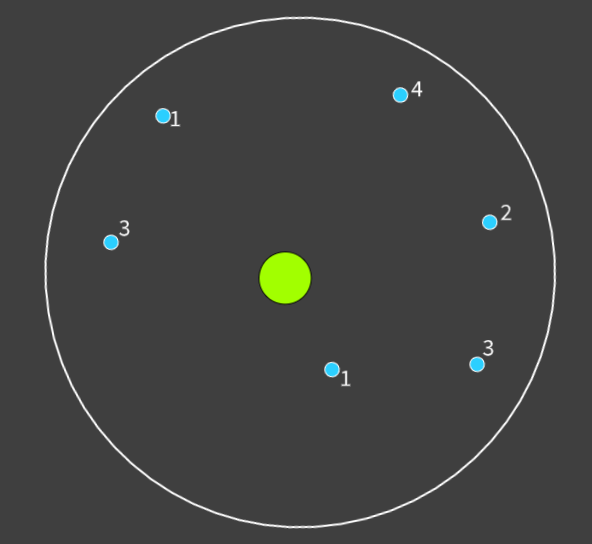


Figure 17: Decision Making Design

In figure 15 it shows the design of this system where all the weapons have a value, but the agent no longer looks for the closest weapon it will look for the weapon that is in its field of view but has the highest value. While this is happening, it will also look for the gun with the highest value that is closest to it. In the example the closest weapon has a value of one, but the agent will get the weapon with the value of four, so it has the best weapon around it. this system would stay in place so if the agent found a weapon with a higher value on the floor it would swap that weapon out for a better one.

### 3D Grid

3D grid is another part of the pathfinding that could be improved on in this system. this would be where the grid goes up in the air as well as just follow the ground. This would allow the agents in this game to use rams and have a map that is not just flat. This would entail changing the grid script in this game to make it, so it has a Z axis as well as the other two. With this the pathfinding calculations would have to be changed to make this work and would slow the process of finding the path as well this is because it would have to check the cube amount of a side path. This system was not added as there was not another time to add this at the end of the project because as long as added this a lot of optimisation would have to go into it to make this system run faster enough to handle multiple agents and the run time of unity this is why this was not worked on as in the time frame was not realistic for this idea and for this project.

### Final Pathfinding Review

Overall I found that the pathfinding in this game was a success as the agents do what I set out for them to do each point above was a way to improve this system further, I found that the A\* Pathfinding once in the game worked well and did all tasks in a quick manner. This meant that the system could be used at run time, this system could also make decisions on what path to take and what target to follow. I found that the path smoothing works very well even though it can be improved on as shown in 8.1.3 but for the game these agents did not need this as they do follow the path and road in the game in the middle of the path which was a goal of the project. Each agent will take different paths and will not follow the same path when they are further away for the player.

## Game Review

Overall, the game was a success as all of the pathfinding worked and the core mechanics works as planned this includes the spawning of agents to follow the player or grab weapons, the player movement and shooting, and map design. The agents have been explained early in this report. The player movement works well as the player can move around the map free and feel responsive while doing so this is in line with the shooting as this works with the agent very well as it damages the agents and stops the then from pathfinding anymore. The agent has a ragdoll system for when they die in the game this will make the agents flop to the ground with the use of physics.

One part of this game I wish I improved on was more weapons and a system for the players to pick up different weapons that do different things in the game. Unfortunately, this could not have been added as this was not planned for in the final plan for this project and therefore did not have enough time to add this to the game. improving on this there would also be power ups for the player so more could be done as the player.

The map design was good but there was more to be desired in this part of the game, as explained in 9.1.4. this would have made the game not feel as flat and would add more challenge to the game and made this a better experience for the user in this game.

# Conclusion

In summary over the course of this project I have worked on implementing A\* pathfinding into a first-person game. This was set out with the goals in the introduction and stated in the planning report earlier in the year. Over this project the agents could find the shortest path at run time using the A\* algorithm, the only issue with this was that they could not avoid each other and so this could have been improved by working on the local avoidance in the game. While the agents could change their path as the game was running and make the decision of going towards a weapon depending on if it were in its field of view or not this could be improved by changing the class-based system and weapons having a higher value, as referred to in Section 9. Overall, I see this project as a success as I did what I set out to do at the beginning in the year even if it did not go fully as planned.

This project has increased my knowledge on how to work with unity and the core mechanic of this system, with this improved knowledge I can improve on what has already been worked on in this project to produce an impressive game for my portfolio in the future. Another skill this project has given is the ability to do correct research on subjects I need to know more on and produce a literature review, as this will help on my other projects this year and to do research in the future to improve my skills.

(Bourg and Seemann, 2004, Rabin, 2008, Dasgupta et al., 2006, Sedgewick, 1983, Millington and Funge, 2009, Iglesias, Mouse, Han et al., 2012, Hart et al., 1968, Ngo, 2020, Dechter and Pearl, 1985, RedBlodGames, 2010, Tree, 2020, Elye, 2020, Cormen, 2009, JustCreate, Mercuzot, PolyMount, Szabo, 2015, ScienceDirect, Algfoor et al., 2017, Dijkstra, 1959, van den Berg et al., 2011)

# 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,* 71**,** 319-331.

BOURG, D. M. & SEEMANN, G. 2004. *AI for game developers,* Sebastopol, Calif, O'Reilly.

CORMEN, T. H. 2009. *Introduction to algorithms,* London;Cambridge, Mass;, MIT Press.

DASGUPTA, S., PAPADIMITRIOU, C. H. & VAZIRANI, U. 2006. *Algorithms*, McGraw-Hill, Inc.

DECHTER, R. & PEARL, J. 1985. Generalized best-first search strategies and the optimality of A\*. *J. ACM,* 32**,** 505–536.

DIJKSTRA, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik,* 1**,** 269-271.

ELYE. 2020. *How to Blur an Image on Android* [Online]. Available: <https://medium.com/mobile-app-development-publication/blurring-image-algorithm-example-in-android-cec81911cd5e> [Accessed 02/03/2021].

HAN, J., KAMBER, M. & PEI, J. 2012. *Data mining: concepts and techniques,* London;Amsterdam;, Morgan Kaufmann.

HART, P. E., NILSSON, N. J. & RAPHAEL, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics,* 4**,** 100-107.

IGLESIAS, K. Basic Motions FREE | 3D Animations | Unity Asset Store. *Basic Motions FREE*.

JUSTCREATE Low Poly Farm Pack Lite | 3D Industrial | Unity Asset Store. *Low Poly Farm Pack Lite*.

MERCUZOT, S. Low Poly Weapons VOL.1 | 3D Guns | Unity Asset Store. *Low Poly Weapons VOL.1*.

MILLINGTON, I. & FUNGE, J. D. 2009. *Artificial intelligence for games,* Burlington, MA, Elsevier Morgan Kaufmann.

MOUSE, K. Customizable skybox | 2D Sky | Unity Asset Store. *Customizable skybox*.

NGO, P. 2020. Game Application Using A\* Pathfinding Algorithm to Help Improving Dementia.

POLYMOUNT LowPoly Vegetation Season Pack Lite | 3D Vegetation | Unity Asset Store. *LowPoly Vegetation Season Pack Lite*.

RABIN, S. 2008. *AI game programming wisdom 4,* Boston, Mass, Charles River Media.

REDBLODGAMES. 2010. *Heuristics* [Online]. Available: [http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#:~:text=A\*'s%20Use%20of%20the%20Heuristic%23&text=If%20h(n)%20is%20always,\*%20expands%2C%20making%20it%20slower](http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#:~:text=A*'s%20Use%20of%20the%20Heuristic%23&text=If%20h(n)%20is%20always,*%20expands%2C%20making%20it%20slower). [Accessed].

SCIENCEDIRECT Manhattan Distance - an overview | ScienceDirect Topics.

SEDGEWICK, R. 1983. *Algorithms,* London;Reading, Mass;, Addison-Wesley.

SZABO, F. E. 2015. M. *In:* SZABO, F. E. (ed.) *The Linear Algebra Survival Guide.* Boston: Academic Press.

TREE, S. 2020. *How Dijkstra's Algorithm Works* [Online]. Available: <https://www.youtube.com/watch?v=EFg3u_E6eHU> [Accessed].

VAN DEN BERG, J., GUY, S. J., LIN, M. & MANOCHA, D. Reciprocal n-Body Collision Avoidance. *In:* PRADALIER, C., SIEGWART, R. & HIRZINGER, G., eds. Robotics Research, 2011// 2011 Berlin, Heidelberg. Springer Berlin Heidelberg, 3-19.

# Appendix

[https://github.com/Pascoe007/ScuffedHot](%20https://github.com/Pascoe007/ScuffedHot)

# Bibliography‌

Amritamaz.net. (2013). *amrita mazumdar | Understanding Box Blur*. [online] Available at: http://amritamaz.net/blog/understanding-box-blur [Accessed 22 Apr. 2021].

Armina Mkhitaryan (2017). *Why Is C# Among The Most Popular Programming Languages in The World?* [online] Medium. Available at: https://medium.com/sololearn/why-is-c-among-the-most-popular-programming-languages-in-the-world-ccf26824ffcb#:~:text=C%23%20has%20many%20features%20that,have%20to%20worry%20about%20them.&text=C%23%20is%20a%20complex%20language,simpler%20languages%20such%20as%20Python. [Accessed 22 Apr. 2021].

Brilliant.org. (2016). *Dijkstra’s Shortest Path Algorithm | Brilliant Math & Science Wiki*. [online] Available at: https://brilliant.org/wiki/dijkstras-short-path-finder/ [Accessed 22 Apr. 2021].

C-sharpcorner.com. (2021). *What Is C#*. [online] Available at: https://www.c-sharpcorner.com/article/what-is-c-sharp/ [Accessed 22 Apr. 2021].

Edpresso Team (2019). *What is the A\* algorithm?* [online] Educative: Interactive Courses for Software Developers. Available at: https://www.educative.io/edpresso/what-is-the-a-star-algorithm [Accessed 22 Apr. 2021].

Friedman, J. (2018). *Understanding the Differences Between C#, C++, and C - C# Station*. [online] C# Station. Available at: https://csharp-station.com/understanding-the-differences-between-c-c-and-c/ [Accessed 22 Apr. 2021].

Gamasutra.com. (2021). *Gamasutra - Toward More Realistic Pathfinding*. [online] Available at: https://www.gamasutra.com/view/feature/131505/toward\_more\_realistic\_pathfinding.php?print=1 [Accessed 22 Apr. 2021].

GeeksforGeeks. (2012). *Dijsktra’s algorithm*. [online] Available at: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/ [Accessed 22 Apr. 2021].

GeeksforGeeks. (2016). *A\* Search Algorithm - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/a-search-algorithm/ [Accessed 22 Apr. 2021].

GeeksforGeeks. (2020). *Box Blur Algorithm - With Python implementation - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/box-blur-algorithm-with-python-implementation/ [Accessed 22 Apr. 2021].

Github.io. (2021). *PathFinding.js*. [online] Available at: https://qiao.github.io/PathFinding.js/visual/ [Accessed 22 Apr. 2021].

GorkaGames (2019). *Nav Mesh vs A\* Pathfinding*. [online] Opsive. Available at: https://www.opsive.com/forum/index.php?threads/nav-mesh-vs-a-pathfinding.678/ [Accessed 22 Apr. 2021].

How do I calculate Euclidean and Manhattan distance by hand (2012). *How do I calculate Euclidean and Manhattan distance by hand?* [online] Mathematics Stack Exchange. Available at: https://math.stackexchange.com/questions/139600/how-do-i-calculate-euclidean-and-manhattan-distance-by-hand [Accessed 22 Apr. 2021].

Isaac Computer Science. (2021). *Isaac Computer Science*. [online] Available at: https://isaaccomputerscience.org/concepts/dsa\_search\_a\_star [Accessed 22 Apr. 2021].

Krish Naik (2019). *Euclidean Distance and Manhattan Distance*. *YouTube*. Available at: https://www.youtube.com/watch?v=p3HbBlcXDTE&ab\_channel=KrishNaik [Accessed 22 Apr. 2021].

Nist.gov. (2019). *Manhattan distance*. [online] Available at: https://xlinux.nist.gov/dads/HTML/manhattanDistance.html#:~:text=Definition%3A%20The%20distance%20between%20two,y1%20%2D%20y2%7C. [Accessed 22 Apr. 2021].

Pluralsight.com. (2019). *Everything you need to know about C# | Pluralsight*. [online] Available at: https://www.pluralsight.com/blog/software-development/everything-you-need-to-know-about-c- [Accessed 22 Apr. 2021].

red (2017). *How to calculate heuristic value in a\* algorithm?* [online] Stack Overflow. Available at: https://stackoverflow.com/questions/47485510/how-to-calculate-heuristic-value-in-a-algorithm [Accessed 22 Apr. 2021].

Redblobgames.com. (2014). *Red Blob Games: Introduction to A\**. [online] Available at: https://www.redblobgames.com/pathfinding/a-star/introduction.html [Accessed 22 Apr. 2021].

Sciencedirect.com. (2011). *Heuristic Value - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/computer-science/heuristic-value [Accessed 22 Apr. 2021].

Sciencedirect.com. (2015). *Manhattan Distance - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/mathematics/manhattan-distance [Accessed 22 Apr. 2021].

Sciencedirect.com. (2018). *Euclidean Distance - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/mathematics/euclidean-distance [Accessed 22 Apr. 2021].

Stanford.edu. (2021a). *Heuristics*. [online] Available at: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html [Accessed 22 Apr. 2021].

Stanford.edu. (2021b). *Introduction to A\**. [online] Available at: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html [Accessed 22 Apr. 2021].

star (2014). *A star pathfinding and discrete/smooth positions*. [online] Game Development Stack Exchange. Available at: https://gamedev.stackexchange.com/questions/81593/a-star-pathfinding-and-discrete-smooth-positions [Accessed 22 Apr. 2021].

Support Forum (2018). *Navmesh vs A\* Pathfinding*. [online] Support Forum. Available at: https://forum.arongranberg.com/t/navmesh-vs-a-pathfinding/5529 [Accessed 22 Apr. 2021].

Unity Forum. (2020). *Unity Navmesh vs A\* Pathfinding project*. [online] Available at: https://forum.unity.com/threads/unity-navmesh-vs-a-pathfinding-project.537619/ [Accessed 22 Apr. 2021].

Unity Technologies (2020). *Unity - Manual: Unity User Manual 2020.3 (LTS)*. [online] Unity3d.com. Available at: https://docs.unity3d.com/Manual/index.html [Accessed 22 Apr. 2021].

Unreal Engine. (2019). *Unreal Engine | Features*. [online] Available at: https://www.unrealengine.com/en-US/features [Accessed 22 Apr. 2021].

Wikipedia Contributors (2021a). *Box blur*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Box\_blur [Accessed 22 Apr. 2021].

Wikipedia Contributors (2021b). *Dijkstra’s algorithm*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm [Accessed 22 Apr. 2021].

Wikipedia Contributors (2021c). *Euclidean distance*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Euclidean\_distance#:~:text=In%20mathematics%2C%20the%20Euclidean%20distance,being%20called%20the%20Pythagorean%20distance. [Accessed 22 Apr. 2021].

Wikipedia Contributors (2021d). *Heuristic (computer science)*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Heuristic\_(computer\_science) [Accessed 22 Apr. 2021].

Wikipedia Contributors (2021e). *Taxicab geometry*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Taxicab\_geometry [Accessed 22 Apr. 2021].

1. A heap list is a way to store data where each item has two children and a parent, this will be sorted as an array with the order of the binary tree set out left to right. [↑](#footnote-ref-1)