Advanced games engine programming

A\* Pathfinding Project

@00744684

[Project Plan & Design 2](#_Toc164357479)

[Introduction 2](#_Toc164357480)

[Overview 2](#_Toc164357481)

[Objective 2](#_Toc164357482)

[Conceptual Model 2](#_Toc164357483)

[Entity Relationship Diagram 2](#_Toc164357484)

[Project Outline 3](#_Toc164357485)

[Game Loop 3](#_Toc164357486)

[Enemy 3](#_Toc164357487)

[Player 4](#_Toc164357488)

[Level Editor Tool 4](#_Toc164357489)

[UML OUTLINE 5](#_Toc164357490)

[Project Plan 5](#_Toc164357491)

[Agent/Enemy 5](#_Toc164357492)

[Player and weapon 9](#_Toc164357493)

[Game Manager 9](#_Toc164357494)

[Level Editor 10](#_Toc164357495)

[Milestones 10](#_Toc164357496)

[Project Implementation 10](#_Toc164357497)

[Milestone One 10](#_Toc164357498)

[Milestone Two 11](#_Toc164357499)

[Milestone Three 12](#_Toc164357500)

[Milestone Four 13](#_Toc164357501)

[Project Conclusion 14](#_Toc164357502)

[Introduction 14](#_Toc164357503)

[What Went Well? 14](#_Toc164357504)

[What could have been better ? 14](#_Toc164357505)

[Learnings from the project 14](#_Toc164357506)

[Conclusion 14](#_Toc164357507)

[Appendix 15](#_Toc164357508)

[Appendix A 15](#_Toc164357509)

[Game Controls 15](#_Toc164357510)

[Appendix B 16](#_Toc164357511)

[Code Changes 16](#_Toc164357512)

[Appendix C 17](#_Toc164357513)

[Third Party Assets 17](#_Toc164357514)

[Reference List 17](#_Toc164357515)

[Bibliography 18](#_Toc164357516)

# Project Plan & Design

## Introduction

### Overview

This project will involve the creation of a survival/hide-and-seek game with agents that you use A\* pathfinding to navigate the map. Additionally, the project should ideally include custom advanced editor features to aid project development.

### Objective

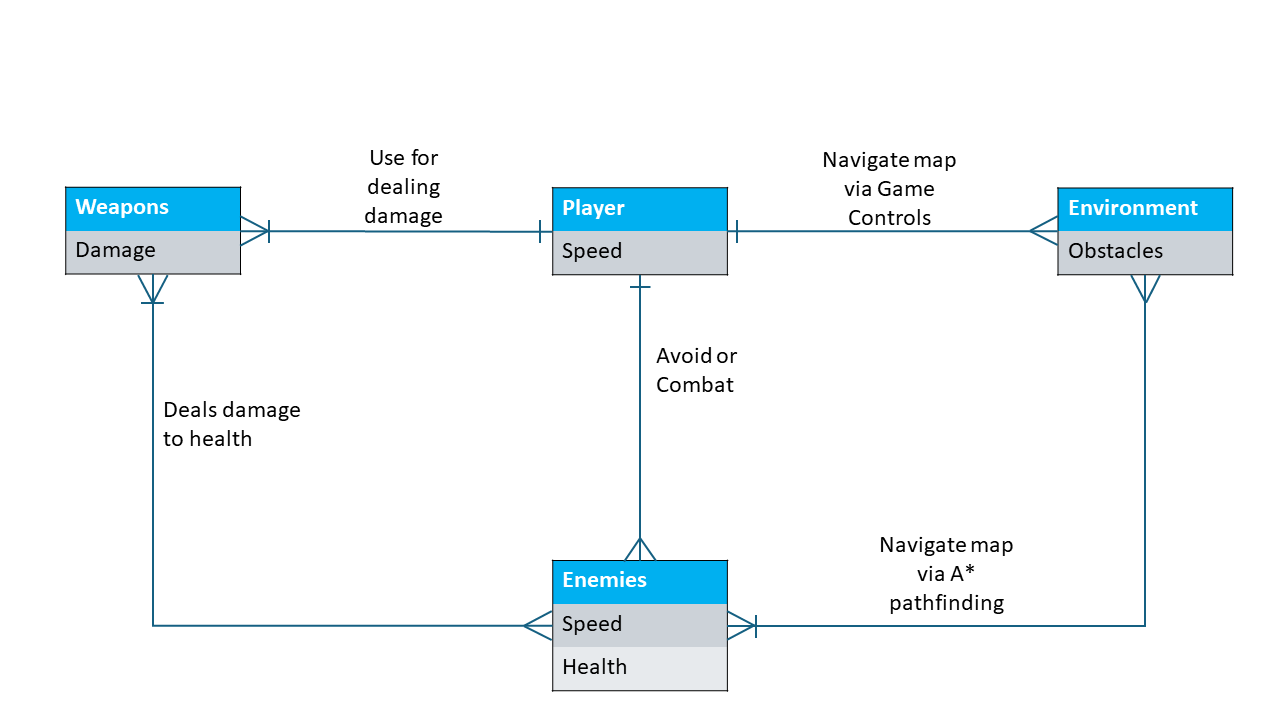
The main goal of this project is to develop a cost-effective A\* pathfinding system as an alternative to the unity navmesh system.

## Conceptual Model

For the game I will develop the following:

* **Components**: Player, Enemies, Weapons, Environment, Game Manager & Level Editor
* **Attributes**: Player ( speed), Enemies (speed, health), Weapons (damage), Environment (obstacles).
* **Relationships**: The player navigates through the Environment, avoiding enemies and combating Enemies. Enemies will end the game when they are in close range of the player, while the weapon will allow the player to eliminate enemies from a safe distance.
* **Tools**: The level editor will allow for creating/placing prefabs, aligning and distancing objects.

### Entity Relationship Diagram



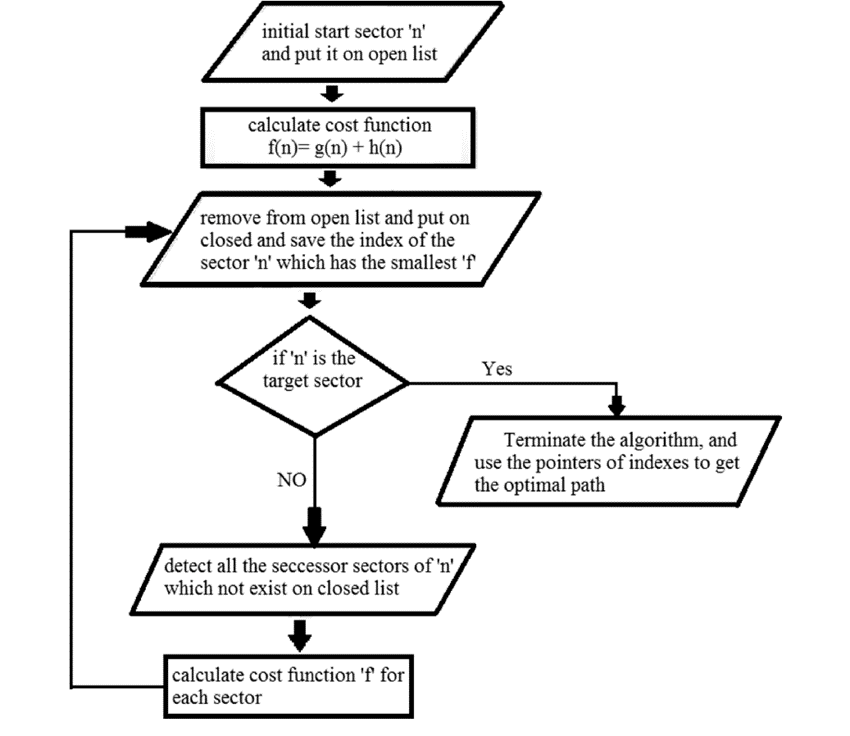
This ERD showcases the interactions between the different entities within the game.

## Project Outline

### Game Loop

The central game loop will focus on the player surviving as many rounds as possible, each round the number of enemies will be doubled from that of the previous until the game ends. The player will survive by moving around the map and killing every enemy still alive. I plan to implement this as a 3d first-person shooter.

### Enemy

The enemy entity will navigate to a random point within the map, if enemy comes within a set range of the player the game ends. As per the main object of the project the enemies will navigate the map using A\* Pathfinding similar to the algorithm shown below: 

(Zidane & Ibrahim, 2018)

As for interaction with the player, enemies can be destroyed by the player’s weapon, a gun as the game end will be triggered by being in proximity and line of sight to the player.

### Player

The player is an important part of this project as it will allow for interactivity within the project. To do this there will be a variety of movement controls including (movement, sprinting, jumping), camera movement and shooting mechanics(aiming and firing) which will be implemented using standard key binds, see appendix A.

### Level Editor Tool

It is my plan to implement a level editor tool to assist with the creation of the map for the project. This tool will include functionality such as:

1. Placing prefabs in the project files
   1. Searching Prefabs, icons for visual feedback
   2. Setting prefab position, scale and rotation
2. Selecting objects withing the hierarchy/scene
   1. Aligning these objects on selected axis
   2. Distributing these objects on selected axis
   3. Snapping selected object to a gird
   4. Creating a prefab from the selected objects
   5. Deleting Selected objects

### UML OUTLINE

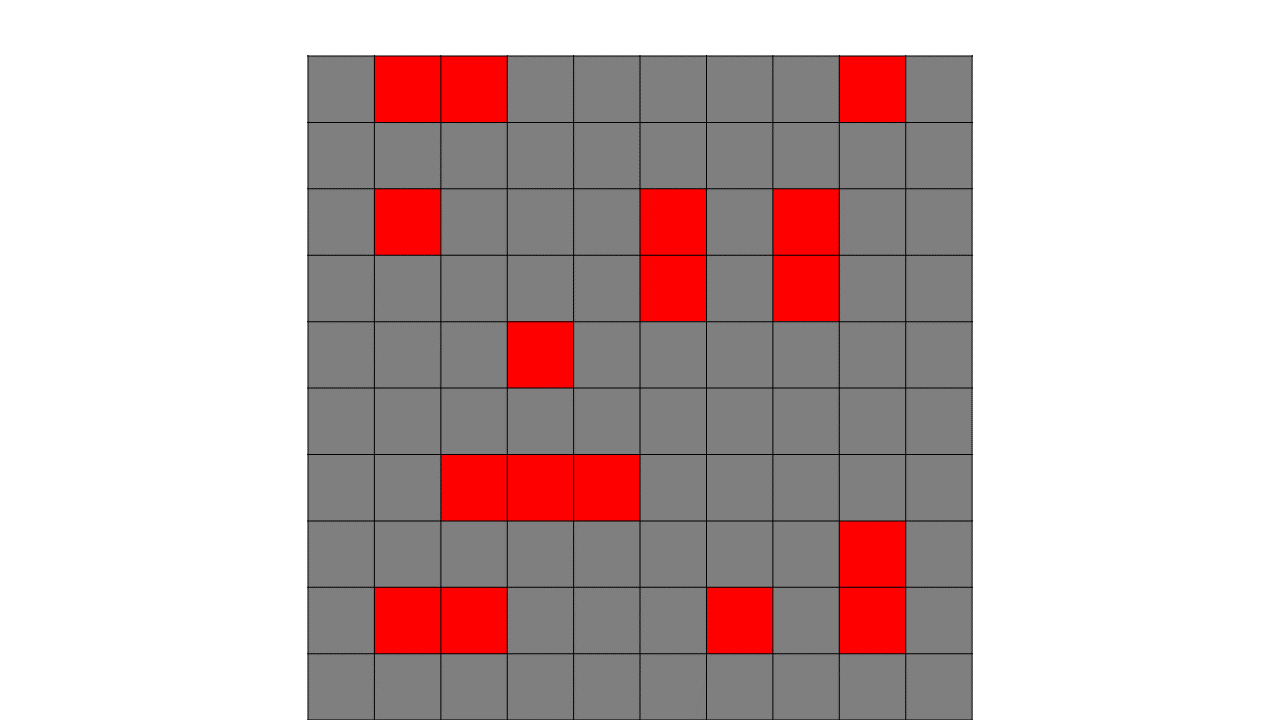
This uml is how I plan to implement my systems/entities. I have added an Idamageable interface and a singleton pattern for this project, although it’s likely that changes may be made during development to keep the project object-oriented. All classes currently are monobehaviour apart from the editor. Purpose of each class is as follow:

* Idamageable – an interface to deal damage to entity
* Singelton – a generic used to ensure only one game manager instance is active
* Weapon- a class that will be attached to a weapon gameobject for the player to use
* Player – a controller to allow the player to move within the environment
* Gridcreator – a class that allows us to create a grid of nodes
* Game manager – responsible for the game loop
* Agent – a class that control how the a\* navigates the environment
* Node- -a class that defines what a node is
* Level editor – an editor window allowing us to edit gameobjects transform and prefabs.

## Project Plan

Agent/Enemy

It is my intention to focus development on that of the agent and the agent’s navigation of the world making it as performant as possible. As mentioned earlier the agent will be navigating via a\* pathfinding.



As a prerequisite the grid creator will have already initialised a 2d array of nodes, Node[,]. Additionally, these nodes will already have been set if they are walkable via a raycast that will detect if there is anything above. The agent will also not be able to walk diagonally so neighbours are only north, east, south and west.

Now the a\* algorithm needs to be understood before it can be implemented so I will lay out how it works with a start node of 1,9 and an end node of 9,3.

//Lists are created and initialised with the addition of the starting node.

openList = Node(1,9)

closedList = Node()

//We loop until we have exhausted all options or found the goal node

WHILE openList.LENGTH > 0

currentNode = node FROM openList with the LOWEST fCost //currentnode is the lowest fcost node

openList.REMOVE(currentNode)

closedList.ADD(currentNode)

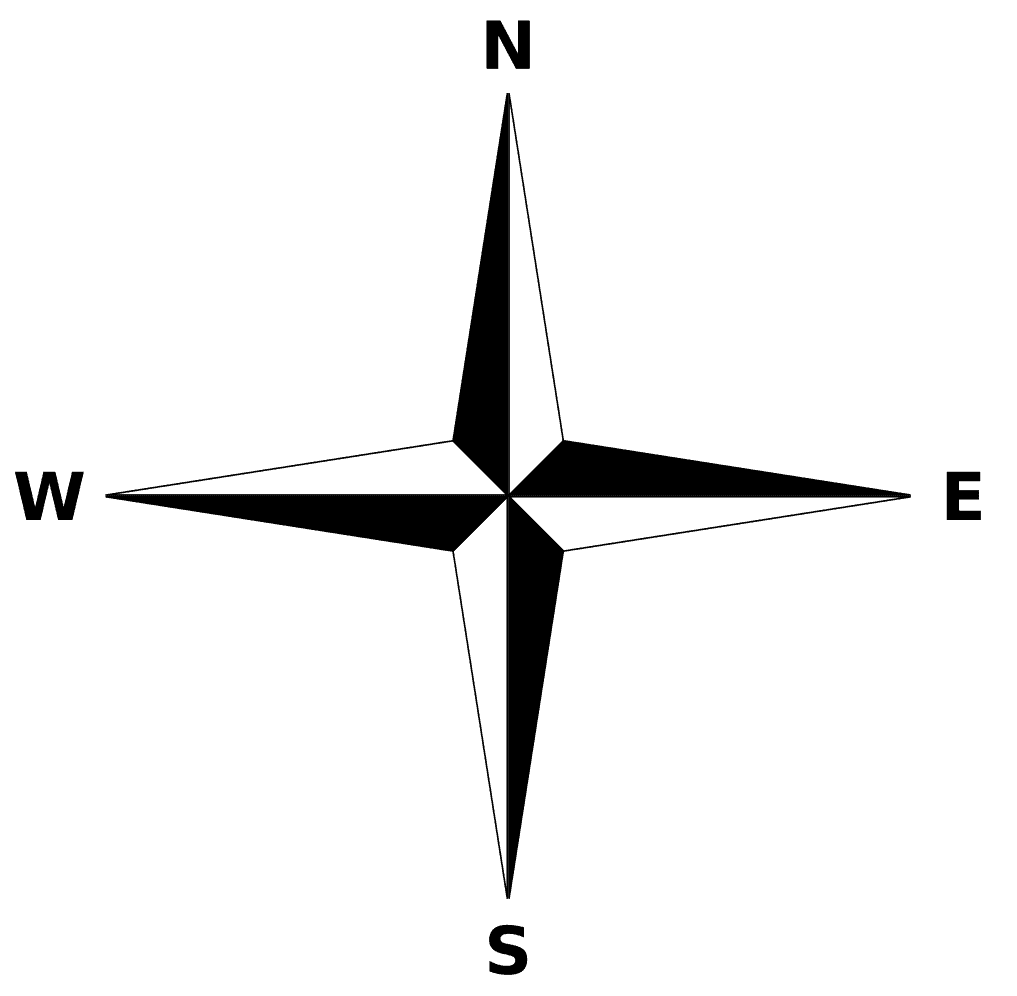
//Check if goal is reached

IF currentNode == goalNode THEN

RETURN Path //return the reversed path of parents of each node

ENDIF

We now use a function to get the neighbour of the current node. In this algorithm we will not be able to travel diagonally so the neighbours would be north, east, south and west.



1,9

1,8

1,10

2,9

Out of Bounds

//Loop through currentnode’s neighbours

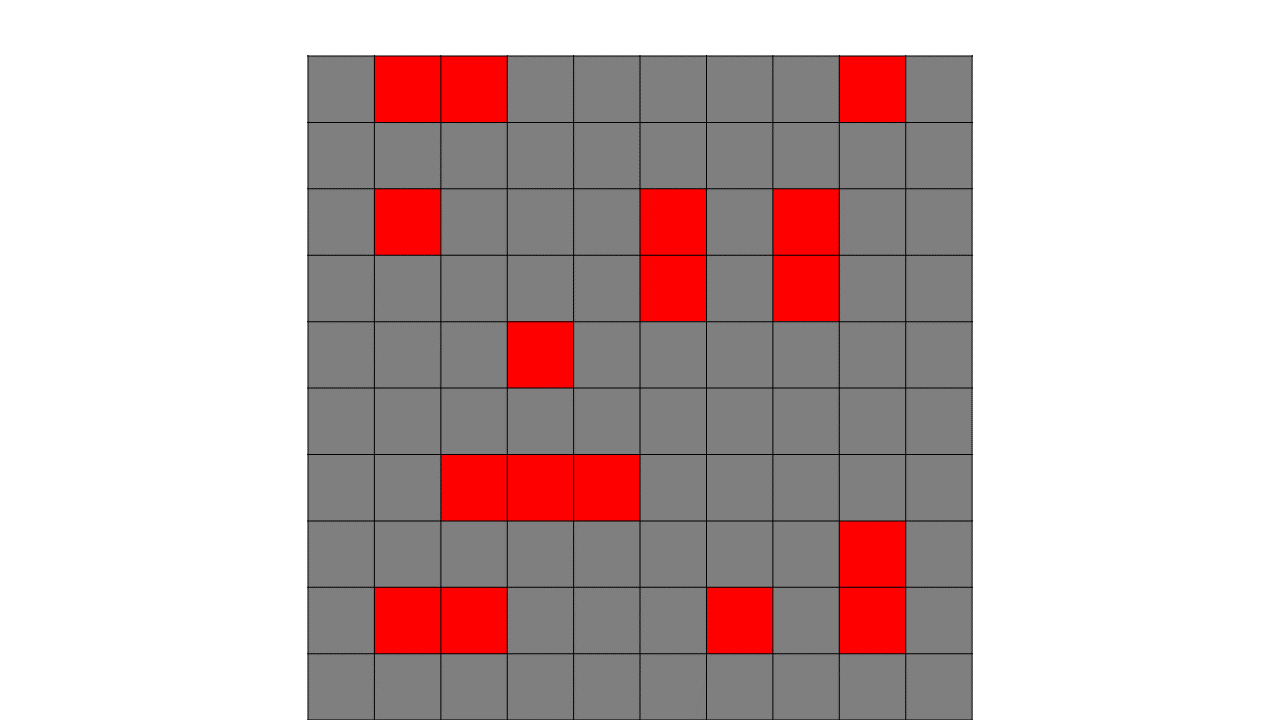
FOREACH neighbour of currentnode

IF neighbour.walkable == FALSE OR neighbour IN closedList THEN

RETURN

ENDIF

Now we update the costs of the nodes, these are then added the open list for evaluation in the next iteration



**G = 1**

**H =13**

**F = 14**

**G = 1**

**H =15**

**F = 16**

//Calculate new costs for the neighbour

newGCost = currentNode.gCost + DISTANCE(currentnode, neighbour)

IF newGCost < neighbor.gCost OR neighbour NOT IN openList THEN

//Update costs

neighbour.gCost = newGCost

neighbour.hCost = DISTANCE(neighbour, goalnode)

neighbour.fCost = neighbour.hCost + neighbour.gCost

IF neighbour NOT IN openlist THEN

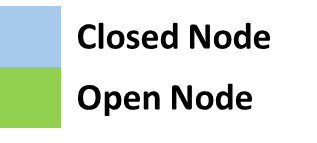
openList.ADD(neighbour)

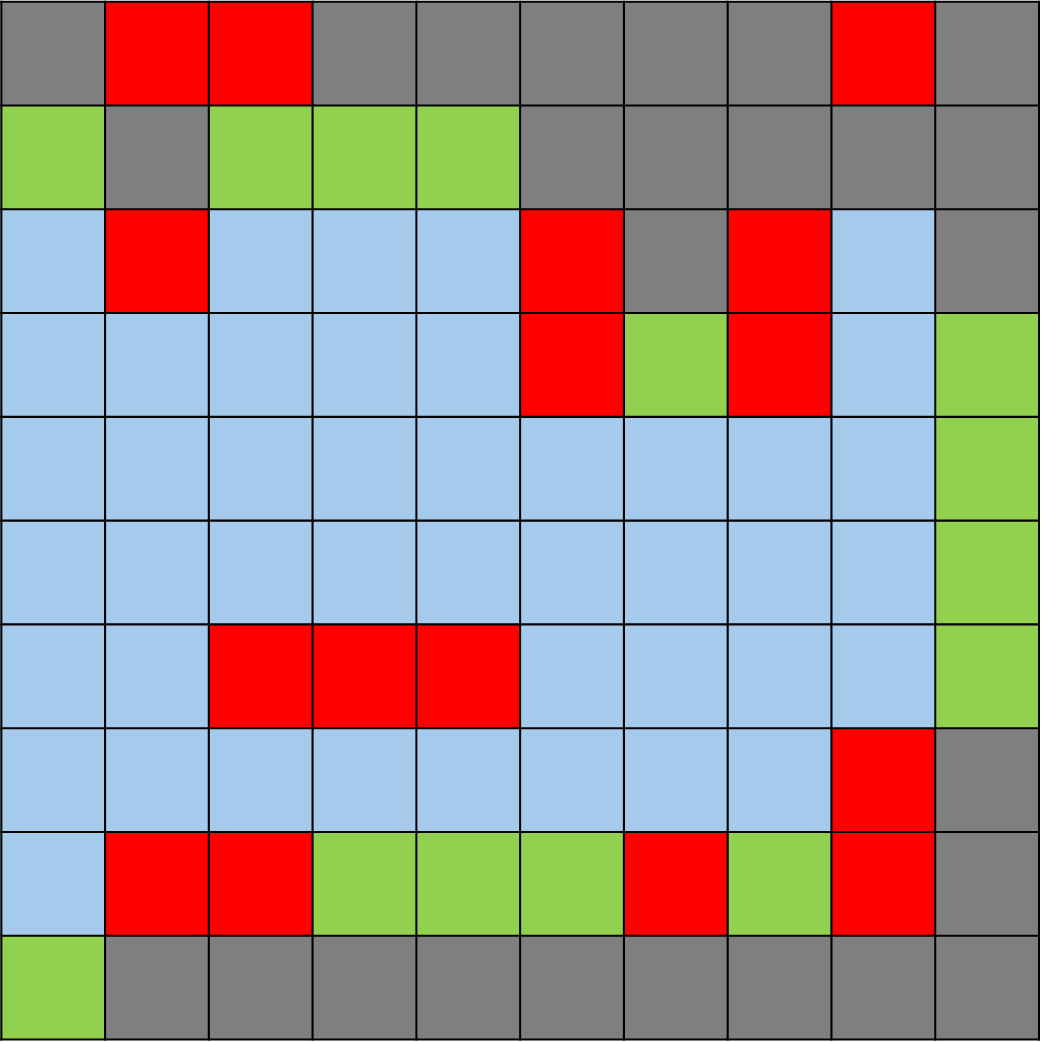
ENDIF

ENDIF

ENDFOREACH

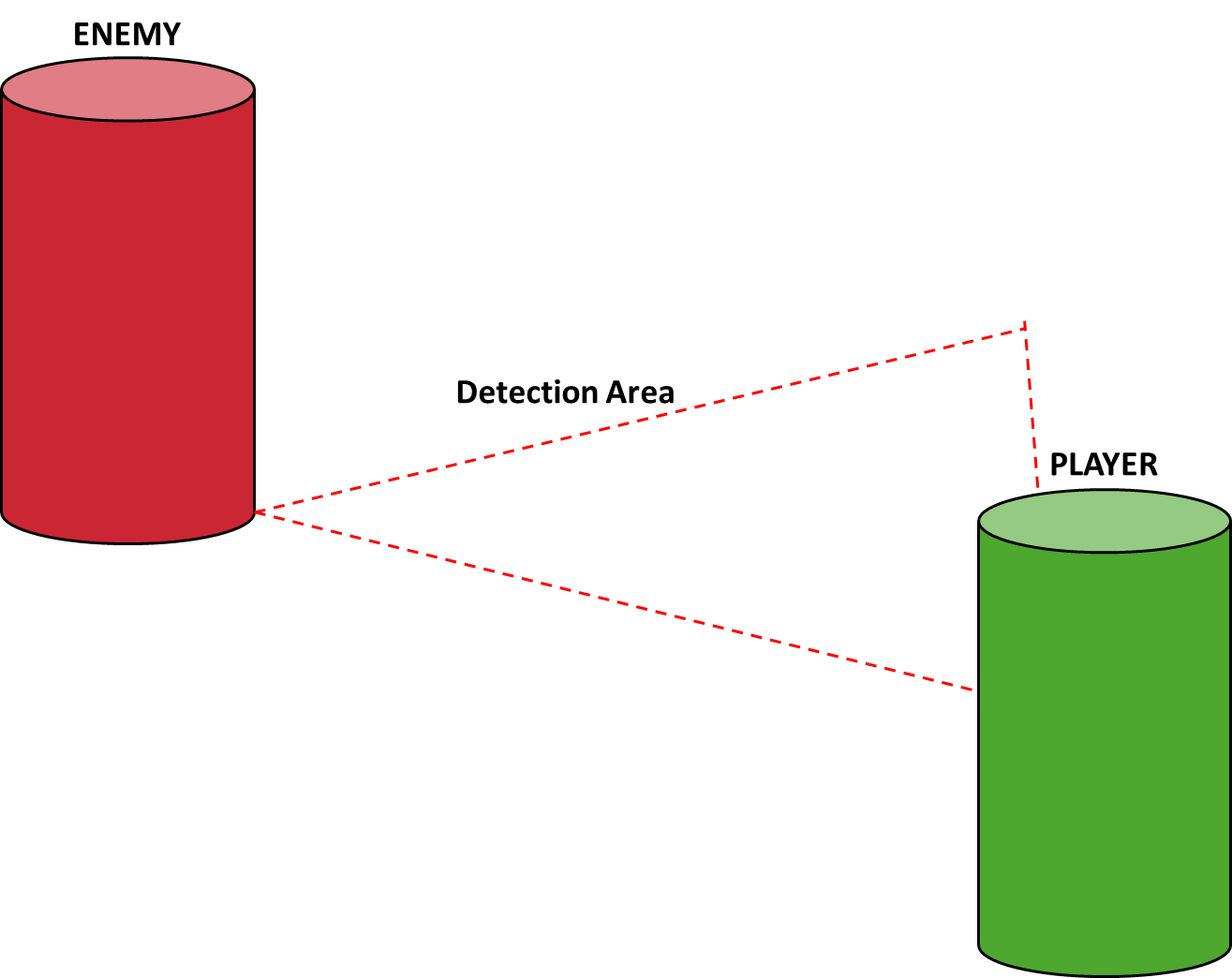
ENDWHILE

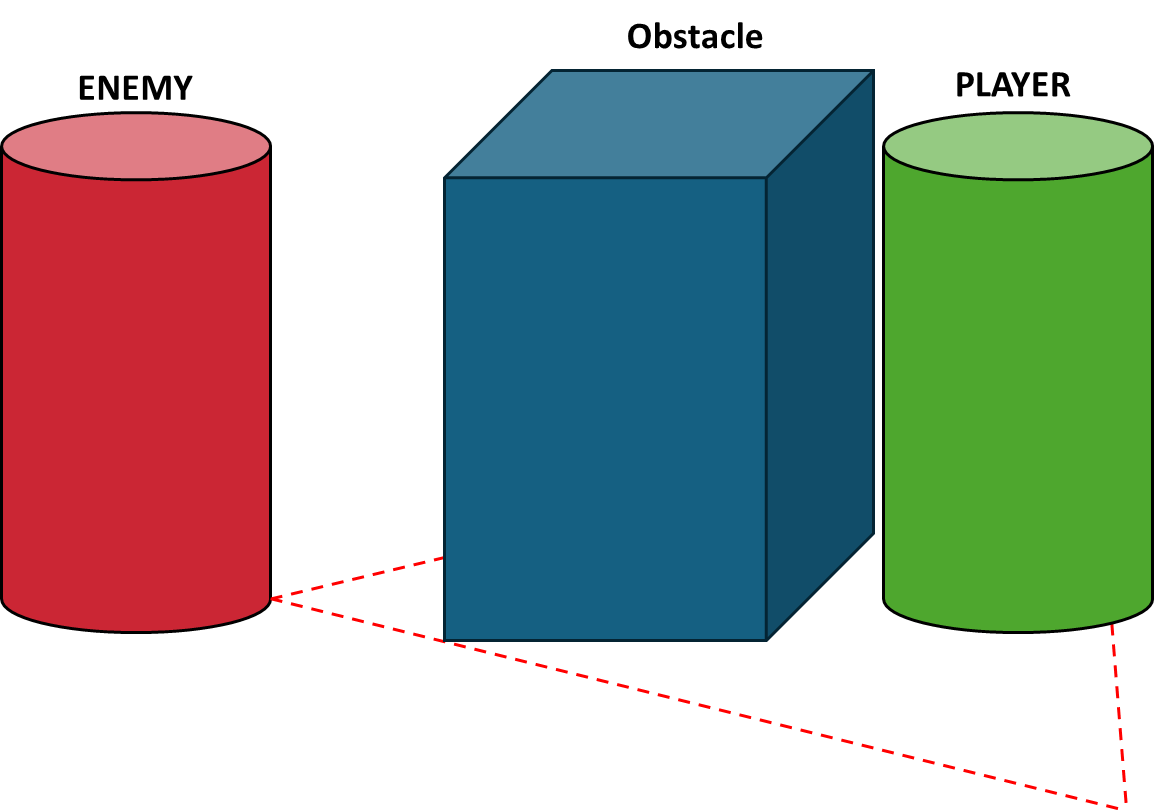
The process is then repeated until the path is found or all nodes have been exhausted.



*How the pathfinding algorithm would have found the path.*

Additionally, as discussed earlier there will be an endgame scenario based on proximity.

The enemy will have a detection area that is triggered when the player enters.

In addition, to being the detection area, the enemy must have line of sight of the player to prevent detection via walls.

### Player and weapon

For the player I will develop movement and combat mechanics.

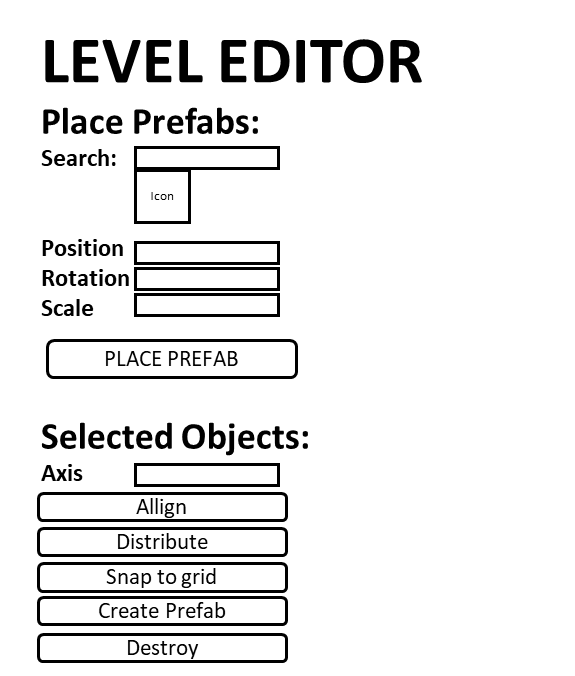
For basis movement I will use input.getaxis and use this input to calculate forces to apply a character controller using controller.move(), I will be using this component as it the most suitable for player controller rather than a rigid body. I will also add a sprint modifier that changes speed and jumping which will apply force to the Y axis.

For shooting, I will be instantiating a bullet with a rigidbody and applying force to “shoot” the bullet. The bullet with a script that allows it to do damage to object with an idamageable interface. The aiming will be lerped between two positions based on if the input is pressed.

### Game Manager

My game manager will be responsible for controlling the game loop; starting rounds and ending the game. I will be implemented it using the singleton pattern as this will ensure only one instance exist exits, this would also mean I could reference the instance without manually connecting scripts/objects together.

### Level Editor

As I am unfamiliar with developing custom tools and editor windows, I will have to consult relevant documentation however I know what I intend for the window to look like.

*Level Editor Concept art*

### Milestones

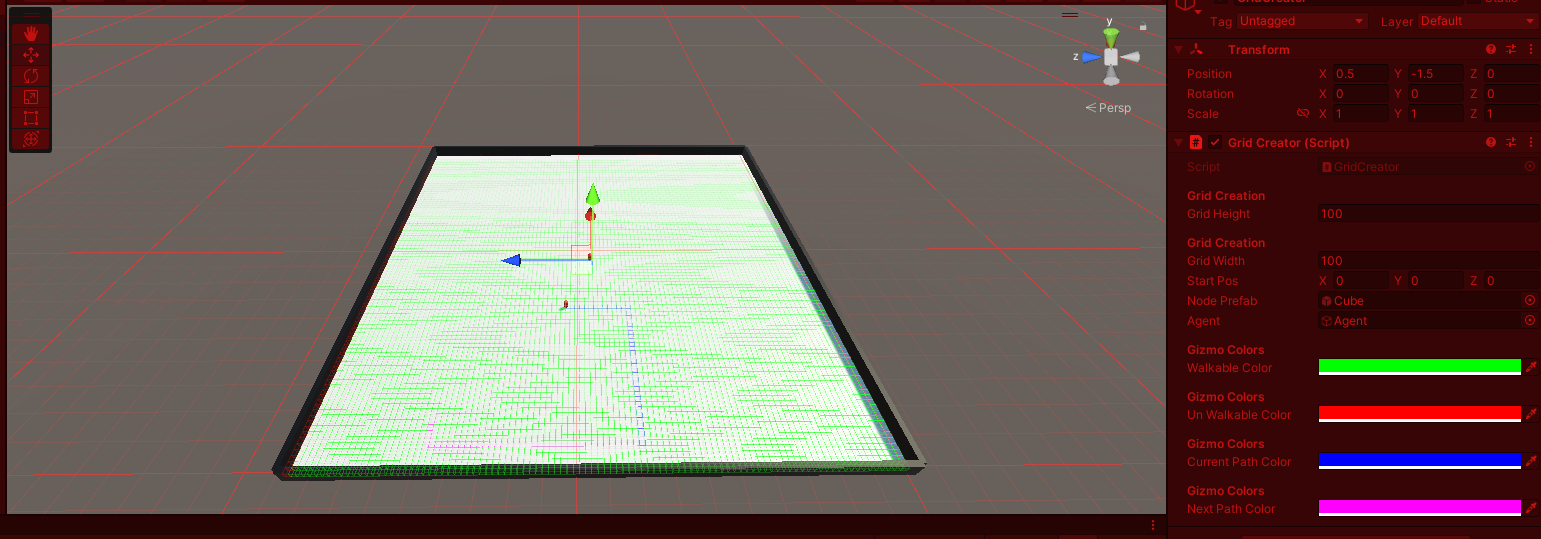
|  |  |
| --- | --- |
| Milestone One | Implement player and agent movement |
| Milestone Two | Create level editor tool and design level |
| Milestone Three | Create game loop and add specified mechanics |
| Milestone Four | Project refinement, optimisation and changes |

# Project Implementation

## Milestone One

Implement player and agent movement went pretty much as planned with some optimisations such as, using a Manhattan distance calculation instead of the Euclidean distance(Appendix B, A1), ensuring grid initialisation before calculating a path and changing the closednode list to a hashset to increase the framerate performance. My biggest challenge was converting the algorithm from pseudocode a unity script.

One issue I overlooked was the computational load of pathfinding for even one agent, I reduced the computational load of calculating a path by performing 100 iterations of pathfinding per frame. This solved an issue of noticeable frame drops, however now there is a brief pause whilst the agent performs its next calculations. I solved this problem by starting the calculation for the nextpath so that there’s always a path to follow.

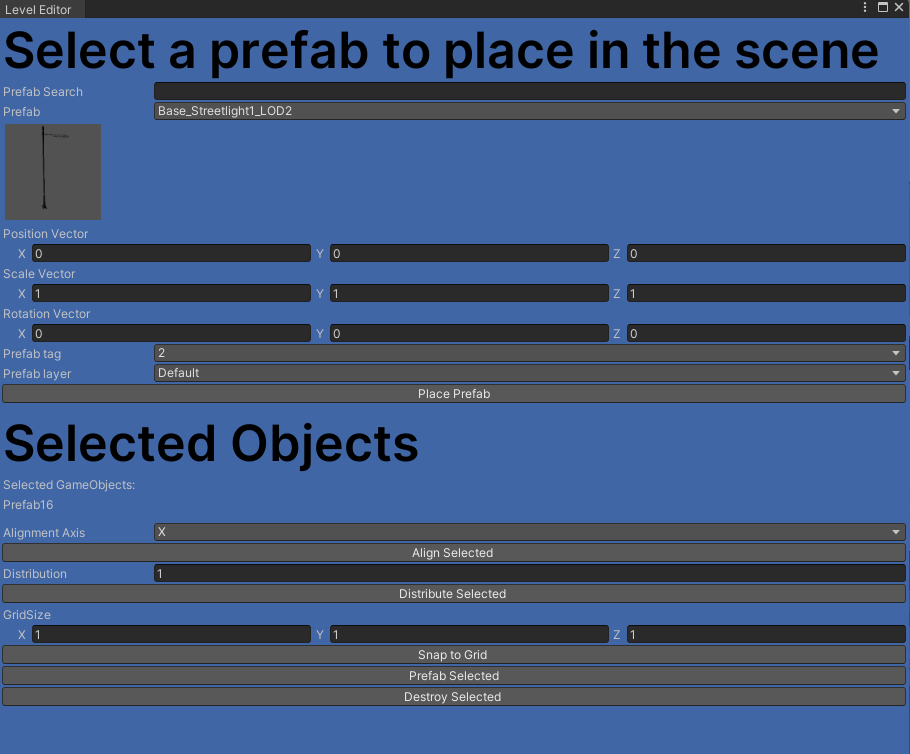
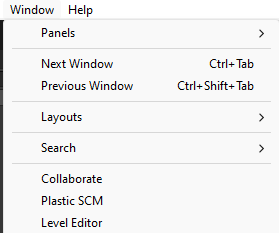
I added gizmos to visualise the grid, walkable/unwalkable are green/red and path/nextpath are blue/magenta. 

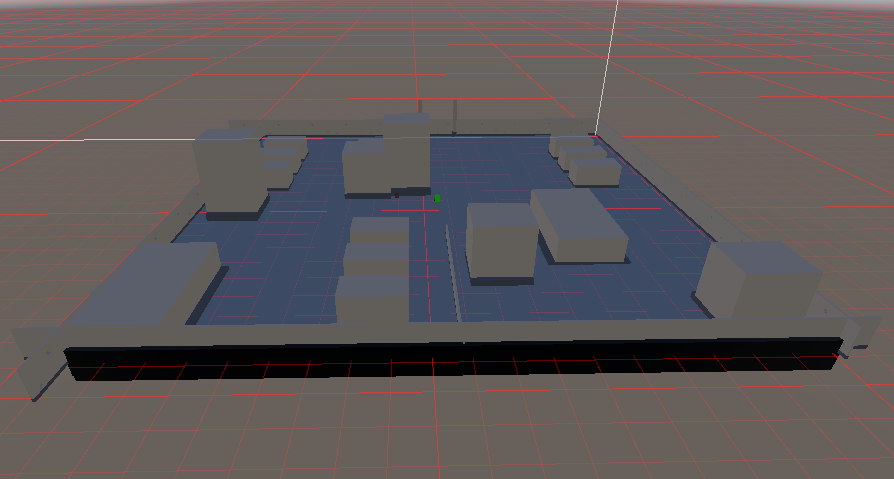
This allowed me to discover that nextpath was constantly being regenerated due to no boolean for detecting if the calculation was happening, also short paths could cause the path to become nulled if a path wasn’t generated in time, so I implemented fixes for both.

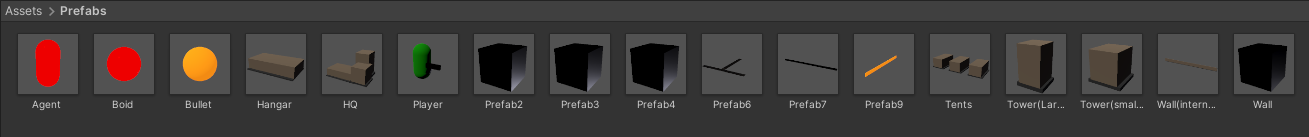
One final thing I did was add the node class see appendix B, A2. Previously, I used gameobjects as this allowed me to visualise the grid but with gizmos this was no longer a requirement and node being gameobject was a waste of resources.

## Milestone Two

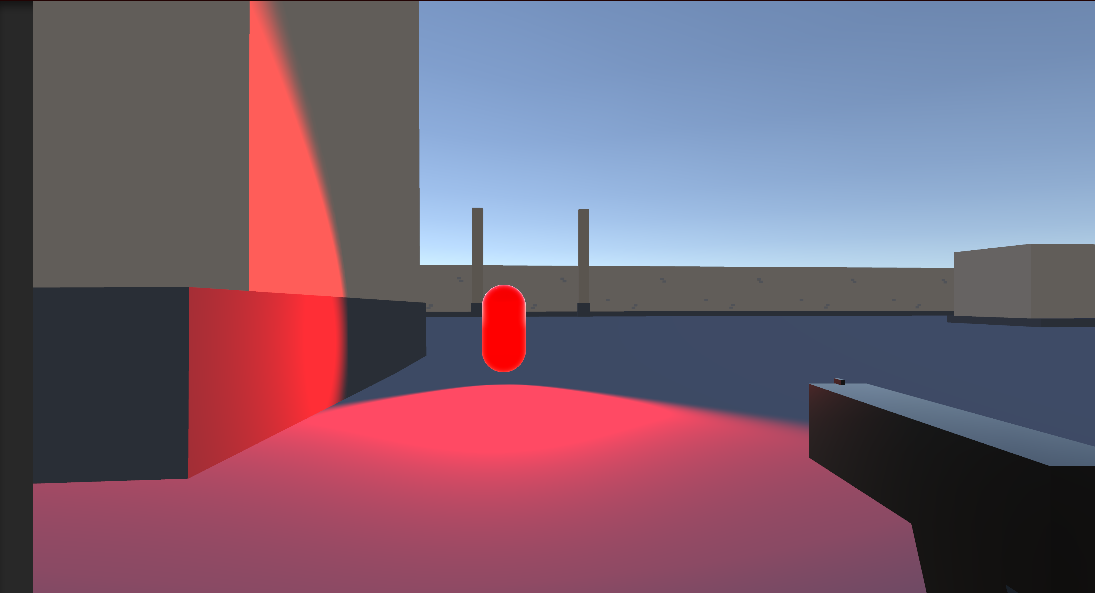
For the level tool I had to consult a lot of unity editor window documentation, especially in dealing with properties such as prefabs. The window ended up similar to my original concept and I used it to create the map/level as well as various prefabs shown below. The biggest challenge was learning how to access a variety of properties in the project via an editor script.







## Milestone Three

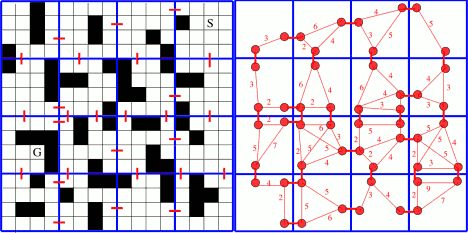
In this stage I implemented the main game loop, mechanics and some UI elements to communicate this to the player. For the implementation of the detection mechanic rather that use a gameobject or define the area in code I used a spotlight for the detection area as this could make the bounds visually clear in addition to being easy to work with. The biggest challenge of this stage was setting up shooting/aiming mechanics as well the game manager. Optimally I placed most references in the game manger so that reference could be accessed via the singleton instance in scripts.

*An agent with spotlight as detection*

## Milestone Four

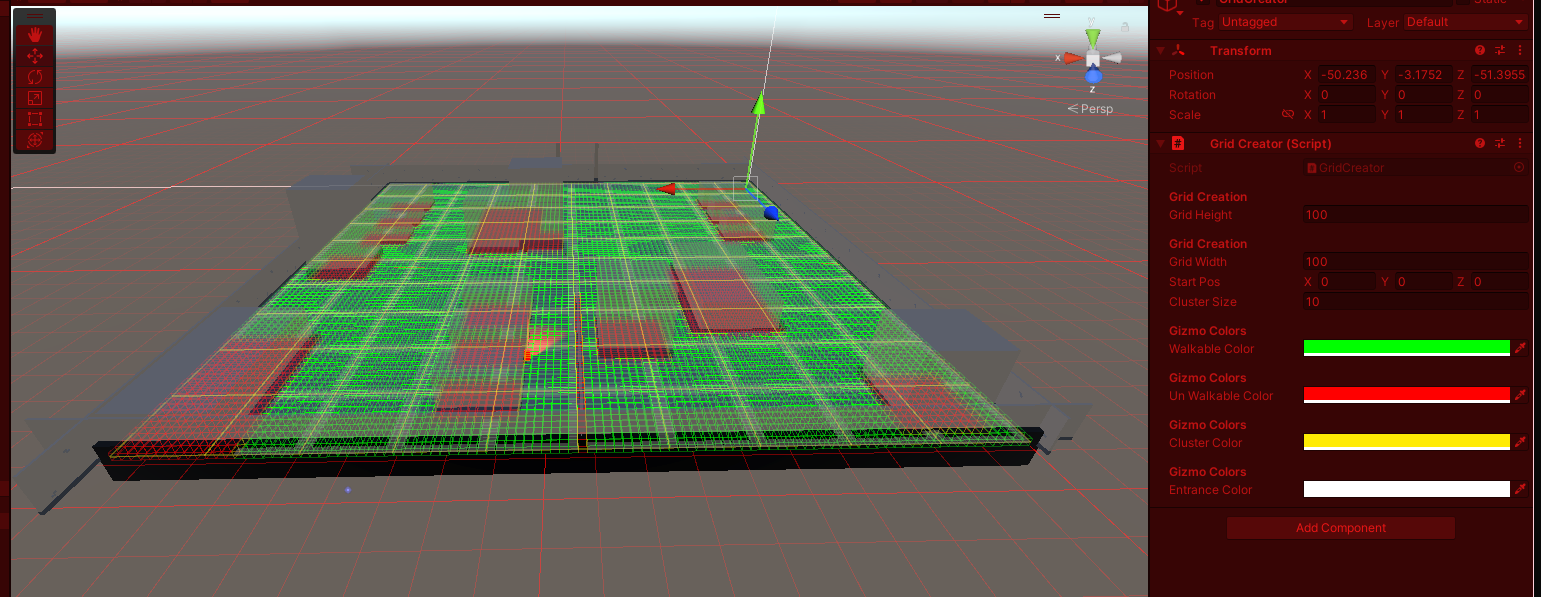
This milestone was improving and adding what I hadn’t initially considered in the original plan.

The first issue I had was upon testing the game I discovered performance issues with the number of agents. I went over many solutions but eventually decided on implement hierarchal pathfinding a\*. This involved separating the gird into cluster, essentially large nodes made up of nodes except these and entrance/exits. A\* pathfinding would take place on the cluster level, require a connection of an entrance/exit to be an acceptable neighbour before then being complement within each cluster.



Chowdhury, X. (2021, May 9)

As an added bonus this new efficacy meant there was no need for a regeneration of path/nextpath and now the game could easily handle 40+ agents.



*Gizmos showcasing cluster and their entrance/exits*

Finally, I added bullet pooling to further improve performance and resource management. I also added boids upon agent death to make kills more interesting.

# Project Conclusion

## Introduction

The main aim was to develop a game that implemented a\* pathfinding and I believe that scope has been met although there is still room for improvement.

## What Went Well?

I was able to follow the created design documentation that I followed closely which helped speed up the development process, I managed to create the game loop that I set out to make that included a seamless implementation of a\* pathfinding to meet the laid out criteria. The project is very modular made use of design patterns, object-oriented programming and relevant coding conventions to allow for readability and reusability.

## What could have been better ?

Whilst I quite satisfied with my implementation there are some improvements that could be made. Firstly, performance could have been further increase by implementing multithreading for the pathfinding tasks. The pathfinding also has several limitations including but not limited to the walkable does not allow for multiple levels, new scenes or regeneration/changes in general, as an advantage of a\* pathfinding is that changes to the walkability of areas are relatively cheap, this is something that might’ve been suitable to add. Furthermore, there were also some development optimisations that could be made, in order to add clusters, I created a new class for it and many functions for this class do the same functions as the node classes but just deal with data for the cluster class. A solution would be to changes these functions to generics which would make the code more efficient and object-oriented.

In addition, the playability of the game loop has some issues, no audio ques, no way to know if an agent is behind or about to spawn on top of you, whilst these are not specified in the brief improvement to this would improve the prototype I developed. The level editor tool I developed alongside this project has some issues, it doesn’t allow for resizing of the text within the window and the background/custom heading is often temperamental not loading at all sometimes.

## Learnings from the project

There were a variety of learnings from this project.

* How a\* pathfinding works and how to implement it in a relevant scenario
* How HPA works and how to implement it in a relevant scenario
* How to make custom editor tools and editor windows including using niche scripting function and learning about the information stored within, for example the layers can only store custom layers and the detail layers have to be added manually.
* The difficulty and customization of implementing a custom pathfinding system in relationship to the simplicity of using unity’s built in navmesh – its more complex but more reliable/customisable

## Conclusion

In conclusion, the project successfully met the laid out criteria in a functional fairly optimised solution implementing a viable alternative to unity navmesh.

Key Achievements:

* Efficient A\* Pathfinding
* Level editor tool development
* Robust Game loop/mechanics

Challenges addressed and resolved:

* Performance
* Complex editor tool development

Areas for improvement:

* Improvement
* Dynamic grids for pathfinding
* Player experience enhancement.

Overall, as the project design documentation were followed closely the project is robust, scalable and modular. There is potential to expand upon and refine this project.

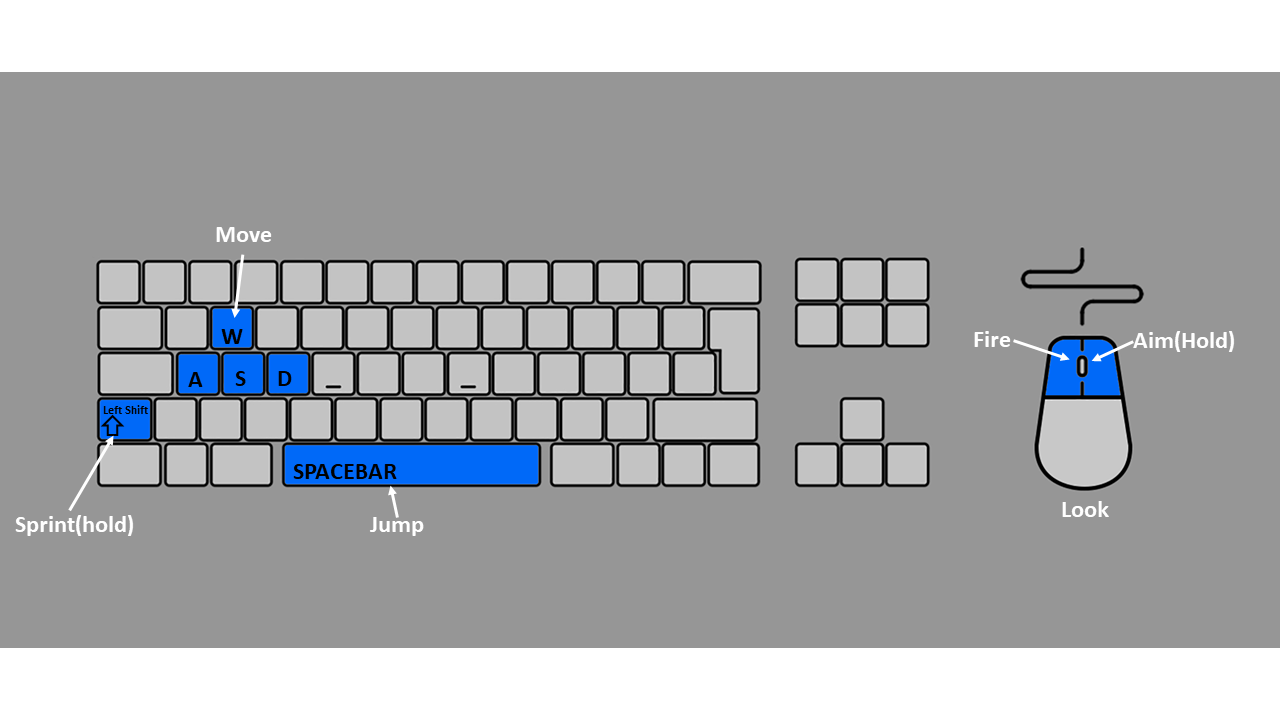
# Appendix

## Appendix A

### Game Controls

For the game controls I chose to use standard first-person controls (Microsoft, 2019; Unity Forum, 2011). These controls are:

* WASD – Move
* Left Shift(hold)- Sprint
* Space – Jump
* Mouse – Look
* Left Mouse Button – Fire
* Right Mouse Button(hold) - Aim



*Game control visualisation*

## Appendix B

### Code Changes

Appendix B showcases changes in code

#### Code A1

Code snippet, change of distance calculation

A black screen with white text

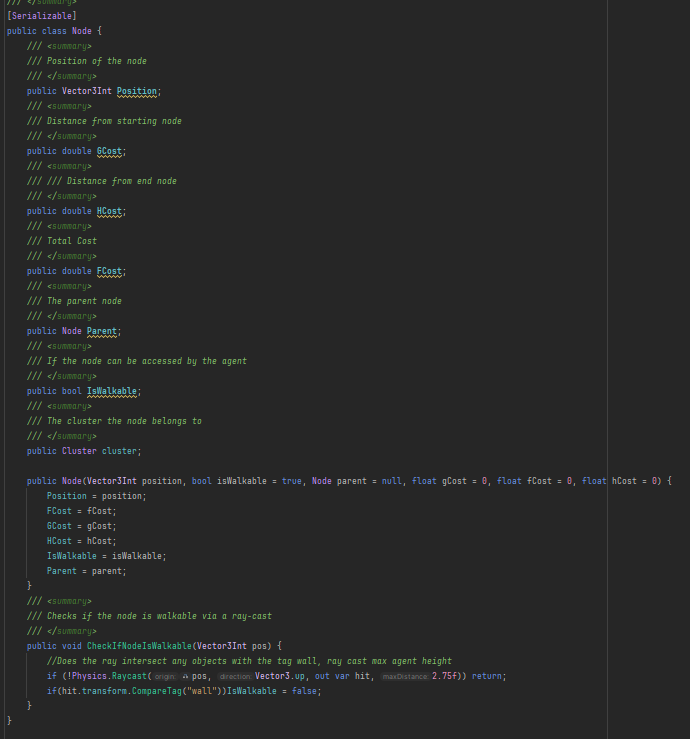
Description automatically generated

A screen shot of a computer

Description automatically generated

#### Code A2

Node Class



## Appendix C

### Third Party Assets

Asset Name: Low Poly FPS Map Lite

Creator: JustCreate

Source: Unity Asset Store

URL: <https://assetstore.unity.com/packages/3d/environments/low-poly-fps-map-lite-258453>

# Reference List

Chowdhury, X. (2021, May 9). How does HPA\* (Hierarchical Pathfinding A\*) really work? Game Development Stack Exchange. <https://gamedev.stackexchange.com/questions/192183/how-does-hpahierarchical-pathfinding-a-really-work>

Microsoft. (2019, April 18). Design for the Senses. Gameuxmasterguide.com. https://gameuxmasterguide.com/2019-04-18-DesignSenses/

Unity Forum. (2011, May 26). Most common / intuitive 1st person control method? Unity Forum. https://forum.unity.com/threads/most-common-intuitive-1st-person-control-method.436560/

Zidane, I., & Ibrahim, K. (2018, September 1). Wavefront and A-Star Algorithms for Mobile Robot Path Planning [Conference paper]. ResearchGate. <https://www.researchgate.net/publication/319404402_Wavefront_and_A-Star_Algorithms_for_Mobile_Robot_Path_Planning>

# Bibliography

Game Dev Guide. (2019 , July 8). Easy Editor Windows in Unity with Serialized Properties[Video]. Youtube. <https://youtu.be/c_3DXBrH-Is?si=cpX6iepns6fMAoj8>

Implementing A-star(A\*) for Pathfinding. (2016, March 1). Insight into Programming Algorithms. <https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/implementing-astar-for-pathfinding/>

‌

Kemble Software. (2020, April 9). Unity Custom Inspectors in 100 seconds[Video]. Youtube. <https://youtu.be/J_Q6Bpk4XDc?si=BHNCoYGeRmylDGbO>

Korsgaard, M. T. (2010, June 17). How to change text color on GUI label? Unity Forum. https://forum.unity.com/threads/how-to-change-text-color-on-gui-label.51791/

Lague, S. (2019, August 26). Coding Adventure: Boids [Video]. Youtube. <https://youtu.be/bqtqltqcQhw?si=VBbQmkLC6lv97NQ7>

Ravikiran, A. S. (2024, February 16). A algorithm in artificial intelligence you must know in 2024. Simplilearn. <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm>

Reynolds, C. (2001, September 6). Boids: Background and Update. Retrieved from <https://www.red3d.com/cwr/boids/>

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. In Computer Graphics (ACM SIGGRAPH '87 Conference Proceedings) (Vol. 21, No. 4, pp. 25-34). Anaheim, California, July 1987. Retrieved from <https://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/>

RSDevelopment. (2022, May 22). Spawner - Unity EditorWindow Tutorial (Random Spawn Position, Avoiding Collisions) [Video]. Youtube. <https://youtu.be/GGwrDQakG00?si=lv6uR8OINAHPNVDL>

Sharpcoderblog. (n.d.). Unity FPS Controller. Sharp Coder Blog. Retrieved February 1, 2024, from <https://www.sharpcoderblog.com/blog/unity-3d-fps-controller>

Stevenson, M. (2012, December 18). Generic Singleton classes for Unity. Use MonoBehaviourSingletonPersistent for any singleton that must survive a level load. Github Gist. https://gist.github.com/mstevenson/4325117

Unity Discussions. (2013, September). Color custom editor window. Unity Discussions. <https://discussions.unity.com/t/color-custom-editor-window/82068/1>

Unity Technologies. (2024, March 29). Asset Database. Unity Manual. <https://docs.unity3d.com/Manual/AssetDatabase.html>

Unity Technologies. (n.d.). Styled Text. Unity Manual. Retrieved February 10, 2024, from https://docs.unity3d.com/Packages/com.unity.ugui@2.0/manual/StyledText.html

Unity Technologies. (2024, March 29). AssetDatabase. Unity Documentation. <https://docs.unity3d.com/ScriptReference/AssetDatabase.html>

Unity Forum. (2010, January 4). Here is an Editor Script to Help Place Objects on Ground. Unity Forum. <https://forum.unity.com/threads/here-is-an-editor-script-to-help-place-objects-on-ground.38186/>