AI For Games Final Project: “Shadow Protocol”

Description: Unity developed game using AI techniques

Student ID: 23375868

Name: Cian Gleeson

Introduction

Problem:

Create a game with some sort of intelligent objects within a real-time strategy game environment.

Idea:

Create a top-down Hitman like game.

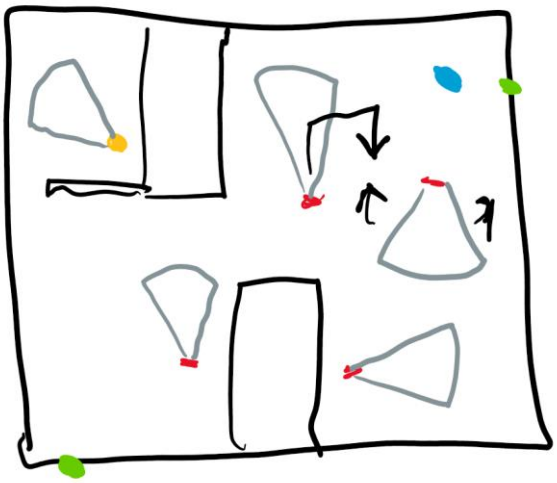
Objective would be to enter a fancy skyscraper with a target on each floor and you, the hitman have eliminate a target and escape, you could go in guns blazing or go in sneakily and quietly, then move on to the next floor of the skyscraper and the enemies would become stronger, faster and “smarter” as you go up the floors of the skyscraper.

Games that Inspired me are Hitman and Enter The Gungeon.

Aspects of AI:

Procedural map generation, pathfinding, player detection, enemy state machine, progressive enemy strength, speed and smarts

Initial Sketch of potential game:



AI Techniques: Procedural Map Generation

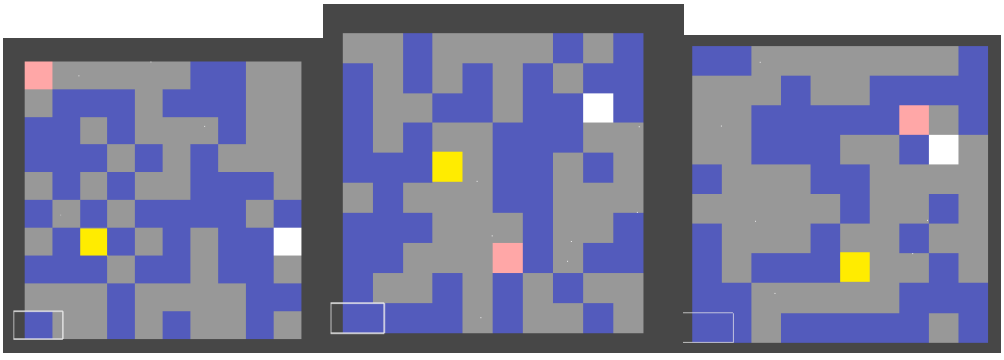
I was tasked with implementing the map generation for the game. To start, I created five prefabs: Entrance, Open Room, Wall, VIP Room, and Exit.

Initial Attempt

My first attempt at random map generation involved creating a matrix where each element was randomly assigned a number. These numbers corresponded to different prefabs. The prefabs were instantiated in a grid using the matrix positions (i, j) as coordinates, with the value at (i, j) determining the prefab type.

The main issue with this approach was that it didn't guarantee a walkable path between the Entrance and Exit prefabs.

Here are some examples of the output:



Revised Approach

To resolve this, I changed the map generation process.

Path Generation

First, I generated a path using a List<String>. This list was populated with strings representing four possible directions: "u" (up), "d" (down), "l" (left), and "r" (right). This was done in a loop using an integer variable which determined the number of directions in the list.

I added if statements to ensure the path didn't immediately reverse on itself (e.g., if the last direction was left, the next direction couldn't be right).

Populating the Matrix

Next, I used the path to populate a matrix. The matrix size was determined by a variable.

The start position was set as the center of the matrix. Using the path's directions, a loop would iterate through the positions in the matrix and giving these positions an int representing the open room prefab.

If issues arose during generation (e.g., the path went out of bounds), the system regenerated the path.

To prevent infinite loops, the path length was reduced until a valid path fit within the matrix dimensions.

At three-quarters of the path's length, the position is saved for the VIP Room prefab. The final position of the path is then saved for the Exit prefab.

While creating the path the program checked the eight surrounding positions for each point. If no rooms were adjacent, those positions would be populated with an int representing the wall prefab.

The system also ensured the Entrance, Exit, and VIP Room didn't overlap.

Once validated, the matrix was used to instantiate the prefabs.
A loop iterated through the matrix, and for each non-zero value, the corresponding prefab was instantiated

Other Details

For the finishing touches I added several utility functions to ensure everything worked smoothly and the maps were playable as well as ensuring every variable corresponded to the game manager which I could then increase when the next level function was called to simulate the map generation AI becoming 'harder', I did cap these variables eventually so after maybe 50 levels you would reach the max difficulty.

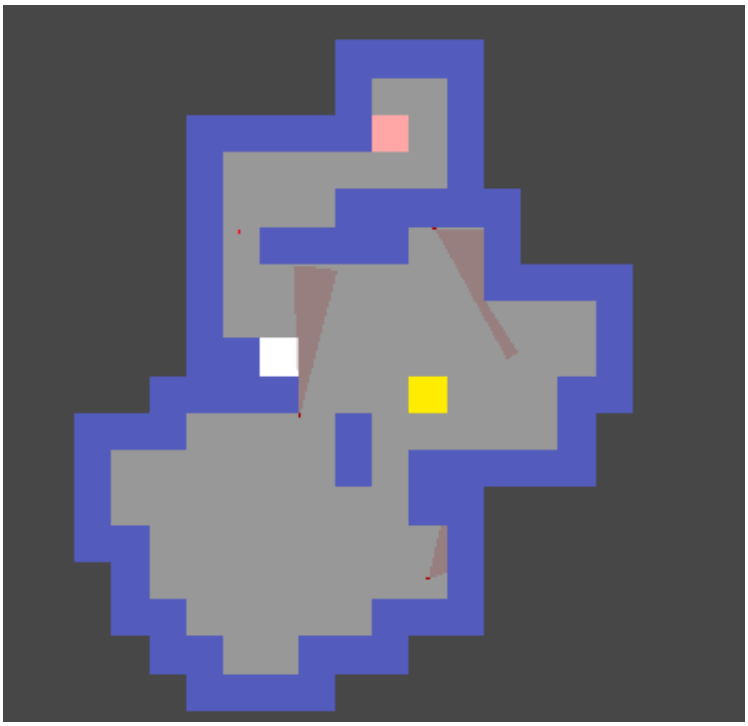
The prefab for open room includes an enemy spawner object that uses game manager variables to change the probability of an enemy spawning.

I did a lot of tweaking with numbers to find what would make for the best maps, for example a small linear one maybe or a large straight line, after some messing around I found that if I made the matrix size small and the path length big, the path would make more open natural feeling areas than a straight linear design, I decided to keep the variables like this for the final result.

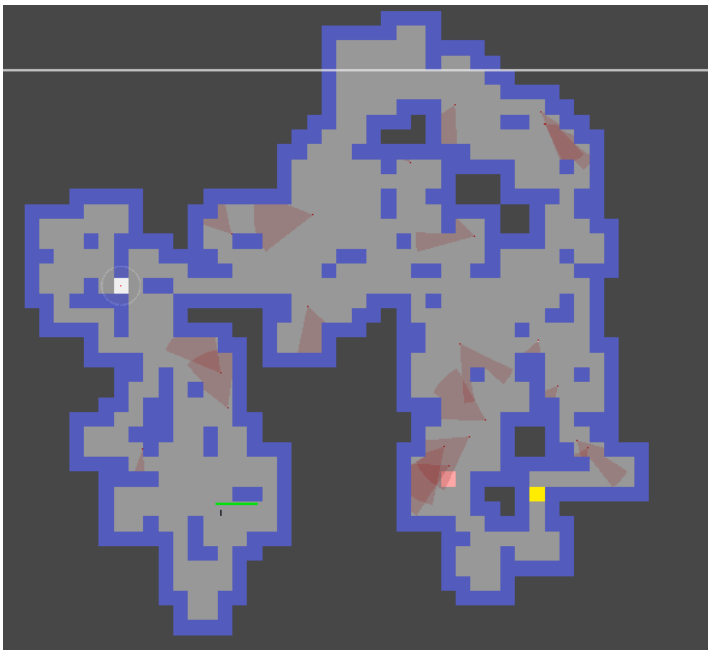
Finalized Map Generator

Here are some examples of the finished map generator.

What the first level would look like when generated:



What the largest possible map would look like:



Improvements

Some improvements and ideas I had for the map generation that I never got to implement were creating multiple open room prefabs that could make the building feel more alive, with offices and big conference rooms, little hallways and a storage room the player could hide in, tables and file cabinets you can hide behind as well as things like reward chests or drawers with secret weapons or perks that you could then keep as you progressed up the building with the items becoming better or stronger as you went higher up to combat the enemy strength and speed increasing.

For graphics I was thinking I could make the walls brick and where there are no walls or floor, we would add the busy Roads beneath the skyscraper, and the higher the level the further away the roads beneath would appear.

AI Techniques: Pathfinding

I was always planning on using Dijkstra's algorithm for pathfinding as at was a part of the applied maths syllabus when I was in 6th year, so I am educated on it,

To do this I would need nodes that the player would use to travel from point to point.

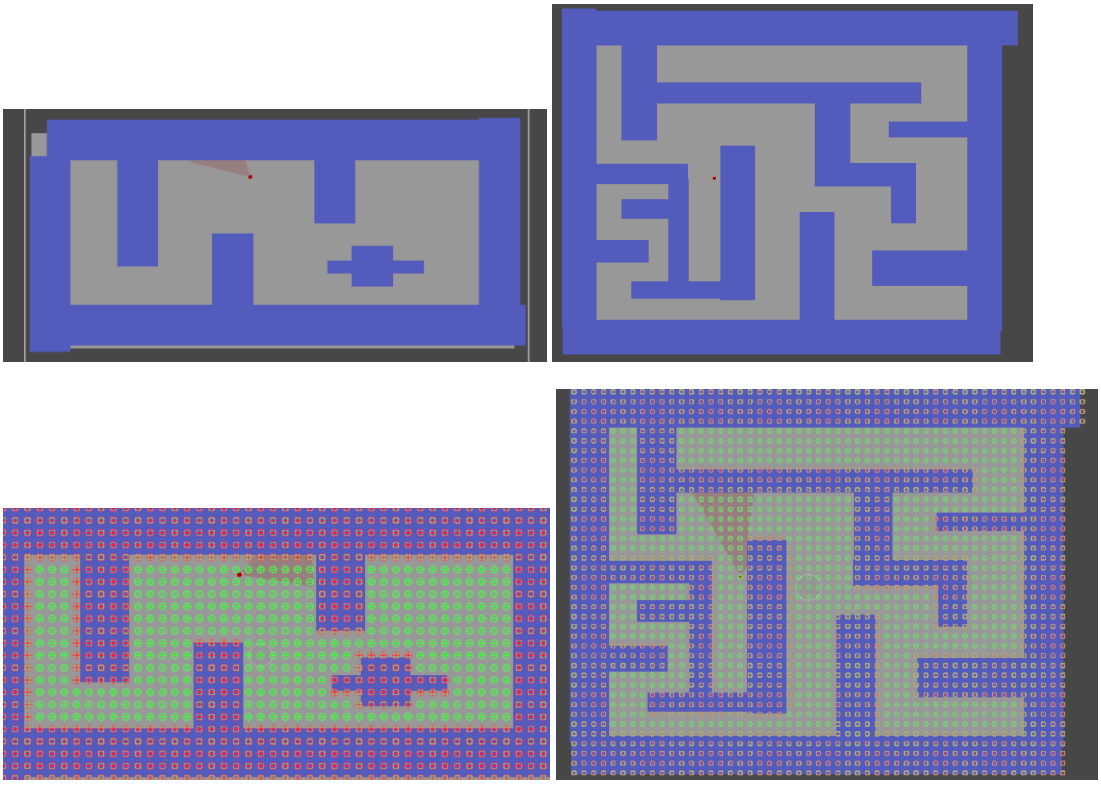
I made a node prefab which draws a circle around its center and if there are any objects within that circle the nodes status is set to not active, if the node doesn't touch anything, it deletes, and if the node touches something that isn't an object it becomes an active node.

To do this I ensured all games objects have box colliders, and I also made corresponding layers and tags for all the games prefabs

I then made a Node Spawner object this object spawns a large grid of nodes and as they spawn, they become active, inactive or delete themselves, I added some visualization code to the NodeSpawner so that I could see what was happening in the scene

I then made two scenes and added some walls, floor and an enemy for testing.

The results of the Node Spawner are shown here:



When the node spawner initializes it makes a list of all active nodes and then checks what nodes are connected to each active node.

For 2 nodes to be connected they must satisfy two conditions:

1. They must be within a set distance of one another.
2. An invisible line drawn between the two nodes doesn't touch any objects.

A dictionary is then initialized with ‘Dictionary<Node, List<Node>>’

This allows us to store every active node as well as the nodes they are connected to.

(Originally, I kept this dictionary in the scene, but I later added it to the game manager and added a couple functions that reset it when a new level is started.)

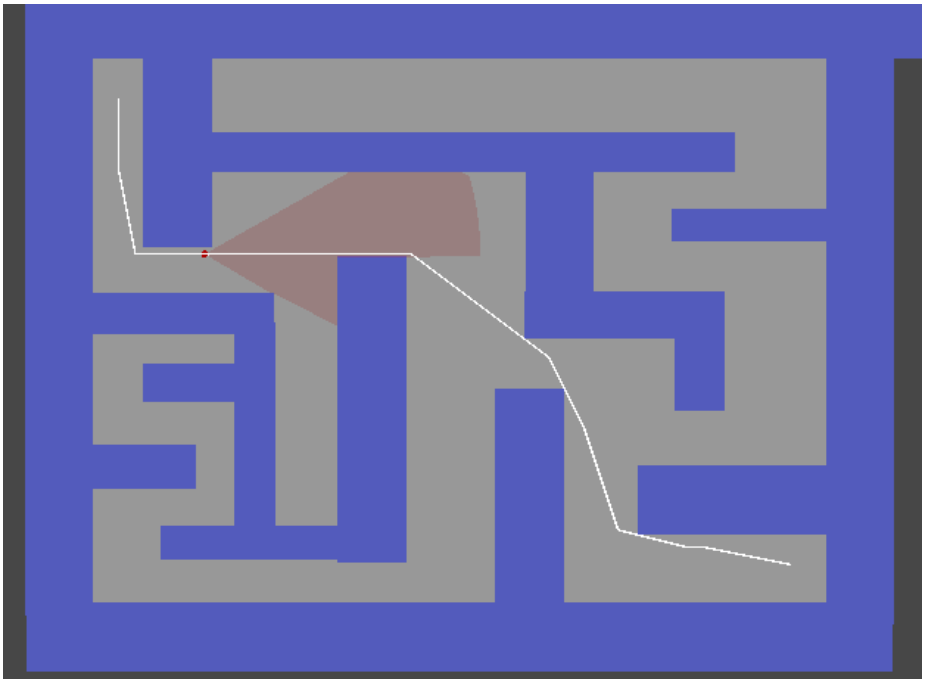
I then made the Dijkstra's algorithm script. The main function in this script takes two nodes and finds the shortest path between them. I also added some functions to find the closest node to any position so that I can get the enemy to the start position of the path.

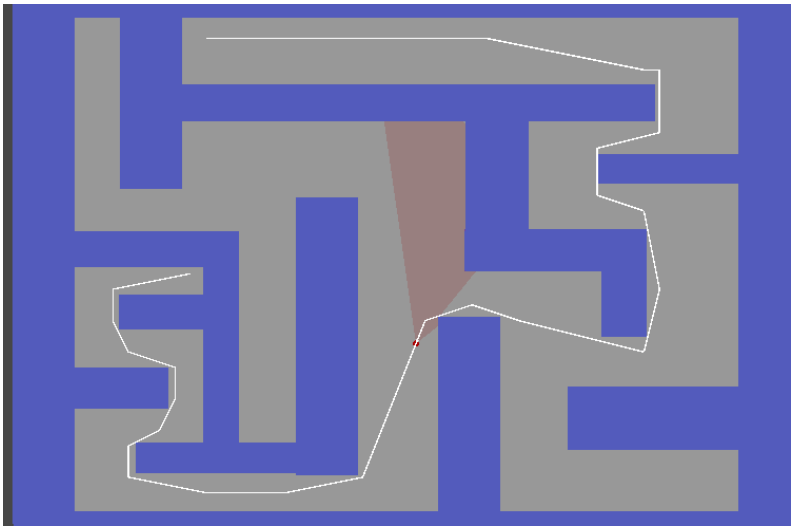
Next, In the enemy controller script, I made a function that when given a position will move the enemy using a path generated by Dijkstra's algorithm, it figures out where the nodes are by calling the dictionary from the game manager.

The function finds two nodes, the closest to the enemy and the closest to the target position, it then uses those and the Dijkstra function to find the shortest path between them, the enemy then goes from node to node along the path.

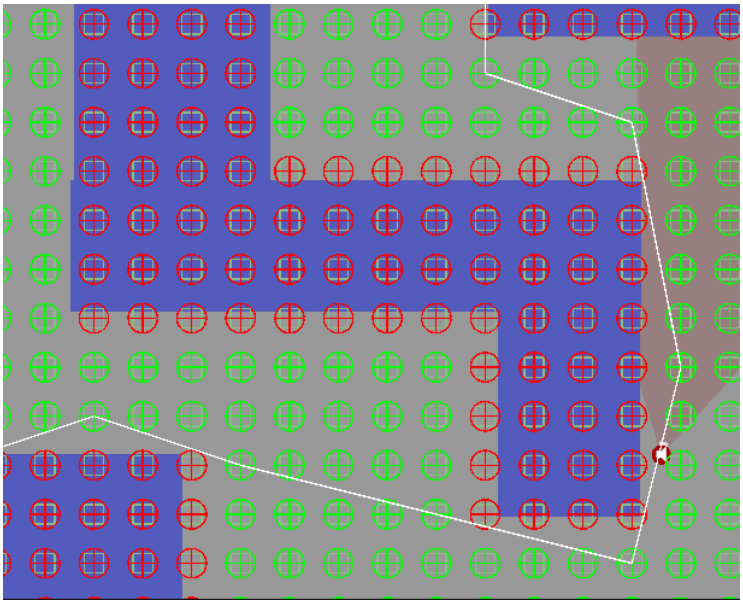
This function also accounts for the direction the enemy is facing, making it face directly towards the next node (Unless as shown later the player is in view).

Example visualizations of the enemy following the path:





Here’s one with the nodes visible



Improvements

Some improvements and ideas I have now after implementing the pathfinding would be to make it smoother and improving where the enemy goes, for example if the enemy knows where the player is and the enemy has a projectile attack, maybe it would only move near the player hiding behind an object and shooting from cover.

AI Techniques: player detection

While Sean Conneally did most of the player detection structure, I added some of the features of it.

When an enemy spawns, a second FOV object spawns as well on the position of the enemy. This object has a mesh renderer, mesh filter and it checks the direction the enemy is facing.

Using a variable for the distance and field of view of the enemy, a mesh is built. Using a ray cast multiple small triangles are made and used to populate the mesh, if any of these ray casts hit an object the vertices of the triangles are then set to the point of contact. The mesh is also set to move with the enemy as they move and look around. The mesh renderer is then used to visualize this area, making a cone that represents the area that the enemy can see.

Then within the enemy controller a ray cast checks if the player is within the FOV of the enemy, it does this by making another ray cast that follows the rules of the FOV object and if this ray cast is colliding with an object on the player layer, then the player is in sight.

AI Techniques: Finite State Machine

While I didn't develop the state machine used in our project, I still had to work with the lads to connect it to my features.

It works by...

Conclusion

This project displays various AI techniques within a real-time strategy environment to create the structure for a top-down stealth-action game. By using procedural map generation, pathfinding, dynamic player detection and finite state machines, we created an

interactive experience where gameplay difficulty scales with the player's progress. Collaboration with my teammates allowed us to come up with implementing all these features within one final model. While there is room for further improvements, such as adding varied room and floor types, more interactive objects, and advanced enemy tactics, the final product successfully showcases AI's potential to make games more immersive and responsive. This project offered a valuable opportunity to apply AI concepts in a practical setting while enabling me to learn new skills and expand my expertise.