Foreword

I would like to thank LiquidX for the time and consideration of my application by moving it forward with this assessment test.

I have read the Document carefully and have developed this small prototype based on the guidelines given.

It must be noted that although the project is done in Unity, the principles that are implemented on this project are engine agnostic, and can be implemented in other game engines, such as Unreal engine.

This document will mainly be separated into 2 parts.
-**Project Report**: a report as to how I develop the prototype
-**Set Up Guide**: a Users guide to help set up the project

Thank you for your time and looking forward to your feedback and further discussions.

PROJECT REPORT

Thought Process (General Approach):

My goal is to create a simple yet modular AI and gameplay system that can are easy to read and implement while still up to the standards required.

This project is using the simple Third Person project provided by unity as a baseline.

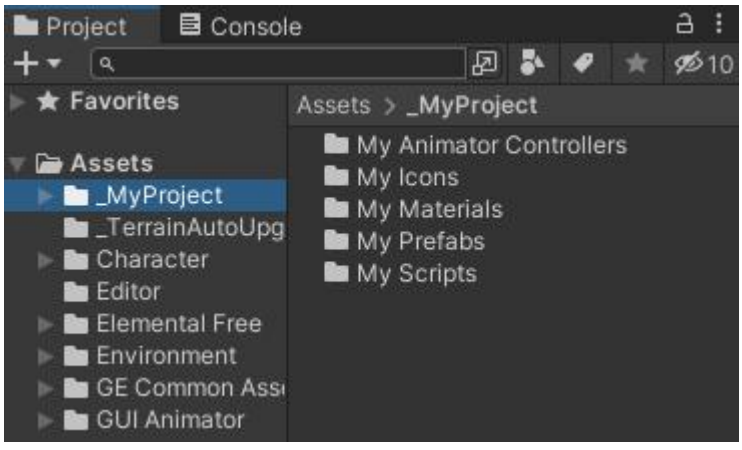
My approach to designing the system is by looking at ways to use as minimal line or code as possible while also using event based systems to mitigate the use of continuous functions such as Update function, on trigger stay, etc, to make the game lighter.

This is why I decided to use the FSM (Finite state machine) approach for the AI system as I believe it is sufficient for a simple system such as this helped in bundling functions together and made it easier track which functions are running

Having FSM also reduces errors from overlapping functions as the nature of FSM only lets 1 state to run at a time.

Aside from the AI system, I have also Included several helper scripts to help manage game states such as game manager script and visualizer scripts to help team members visualize variables (sight distance, view angle, sound travel distance, etc)

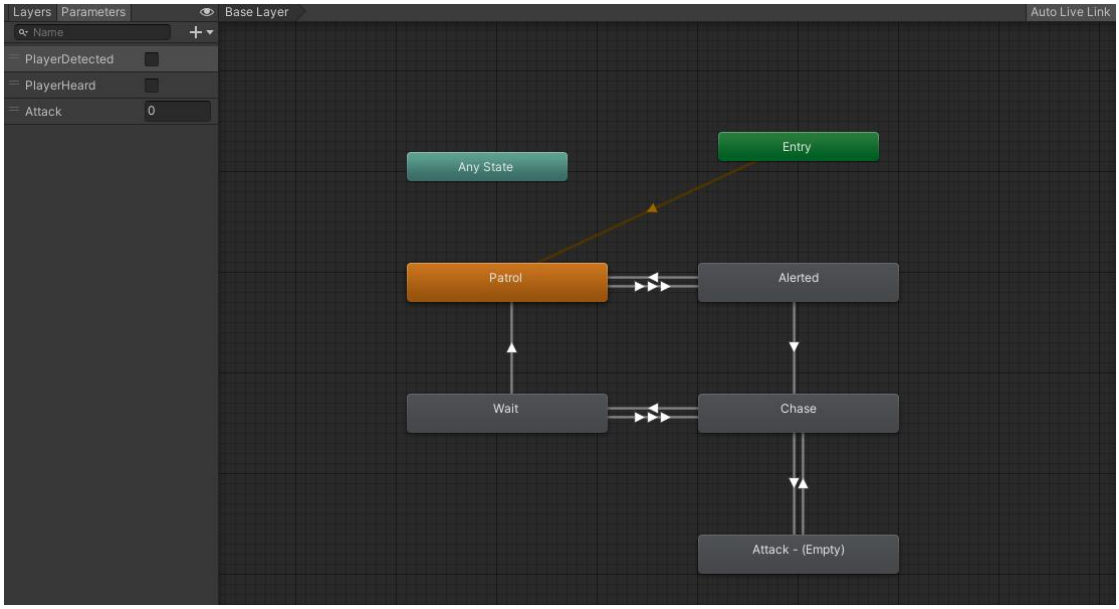
For better organization, I have put all elements that I created (aside from the scenes) for this prototype under the **_MyProject** folder.



As for the scenes created, I used the default directory which is under **Assets**, **Scenes** folder.

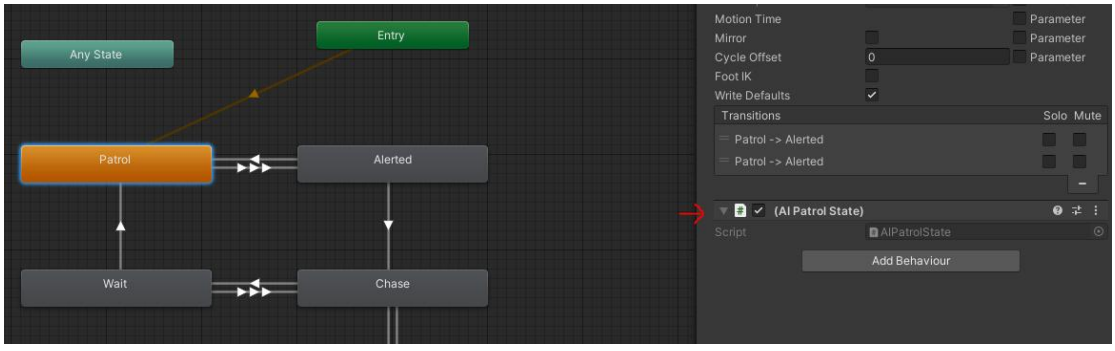
AI system

In order to visualize the FSM I used unity's mechanim, which allows for the visualization of active state and their transition.



In addition to managing the AIs animation, I used mecanim to implement and visualize AI states and state transition that will govern how and when the AIs functions are called

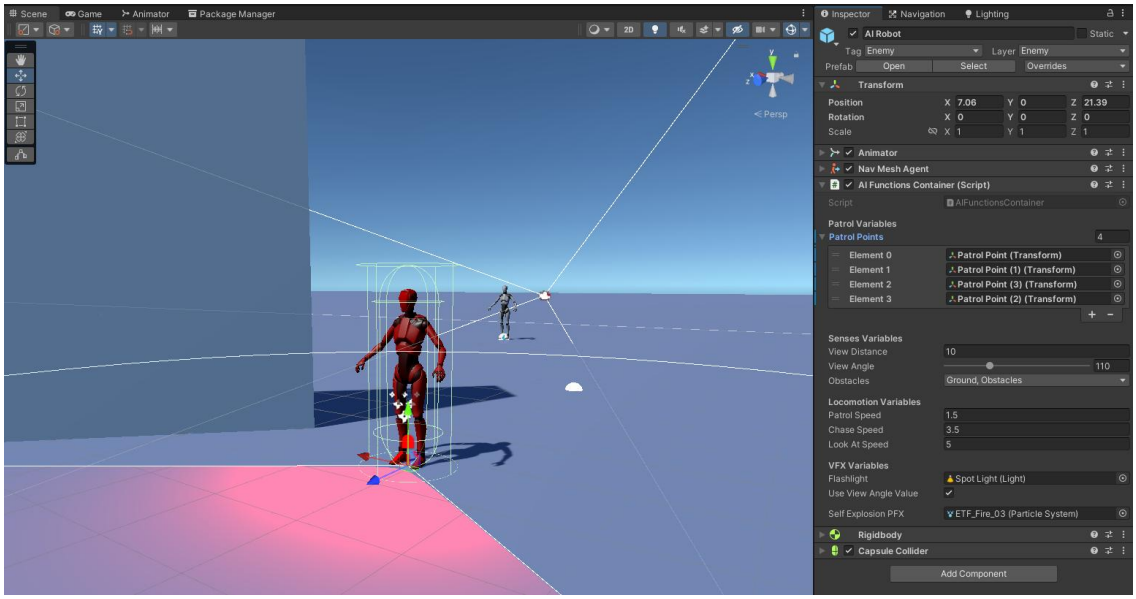
In the picture we can see that there are 5 states: patrol, alerted, chase, attack, and wait. A script is attached to each of these states that will ultimately control which functions are called along with their transition.

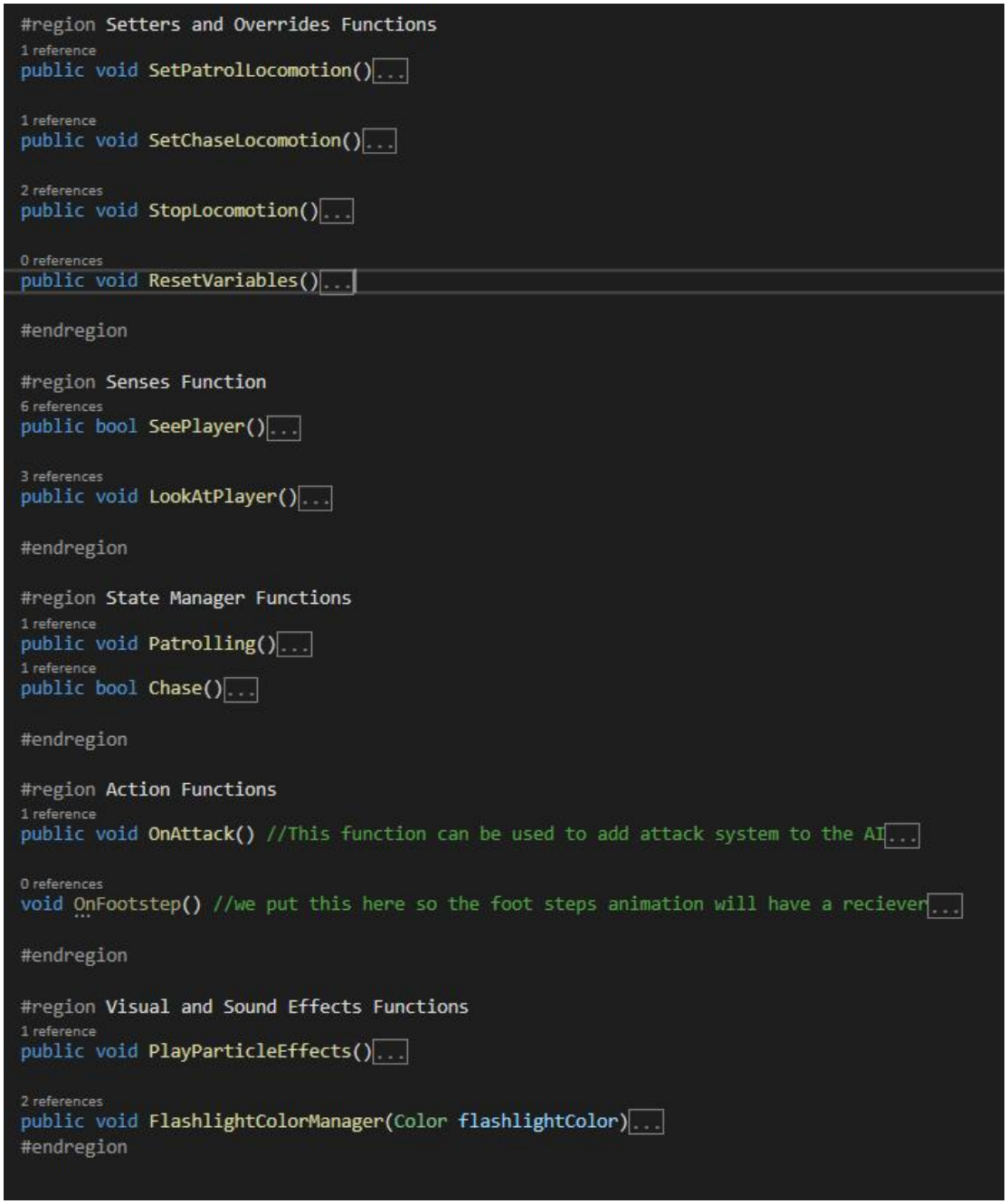


In general the system works like this:

- The AI patrols a fixed point by default
- If the player is detected (heard or seen) the AI will switch from patrol to Alert, which will rotate the AI to face the player. Note: if the player hides behind a wall (obstacle) they will not be seen or heared
- If the player is still within the AIs field of View the AI will chase the player
- If the player is within attacking range of the AI, the AI will switch to the attack state. However, if the player is out of the AIs field of view or hides behind an obstacle, the AI will go to the player's last see position and wait for a few seconds before returning back to its patrol stage

While mecanim handles the function management the actual functions itself is stored in the AIFunctionsContainerScript. This script is attached to the AI game object where, as its name implies, it stores all necessary functions and variables for the AI to use.





This script is ultimately the building block for the AIs functionality and is where variables are adjusted to have.. effects within the game.

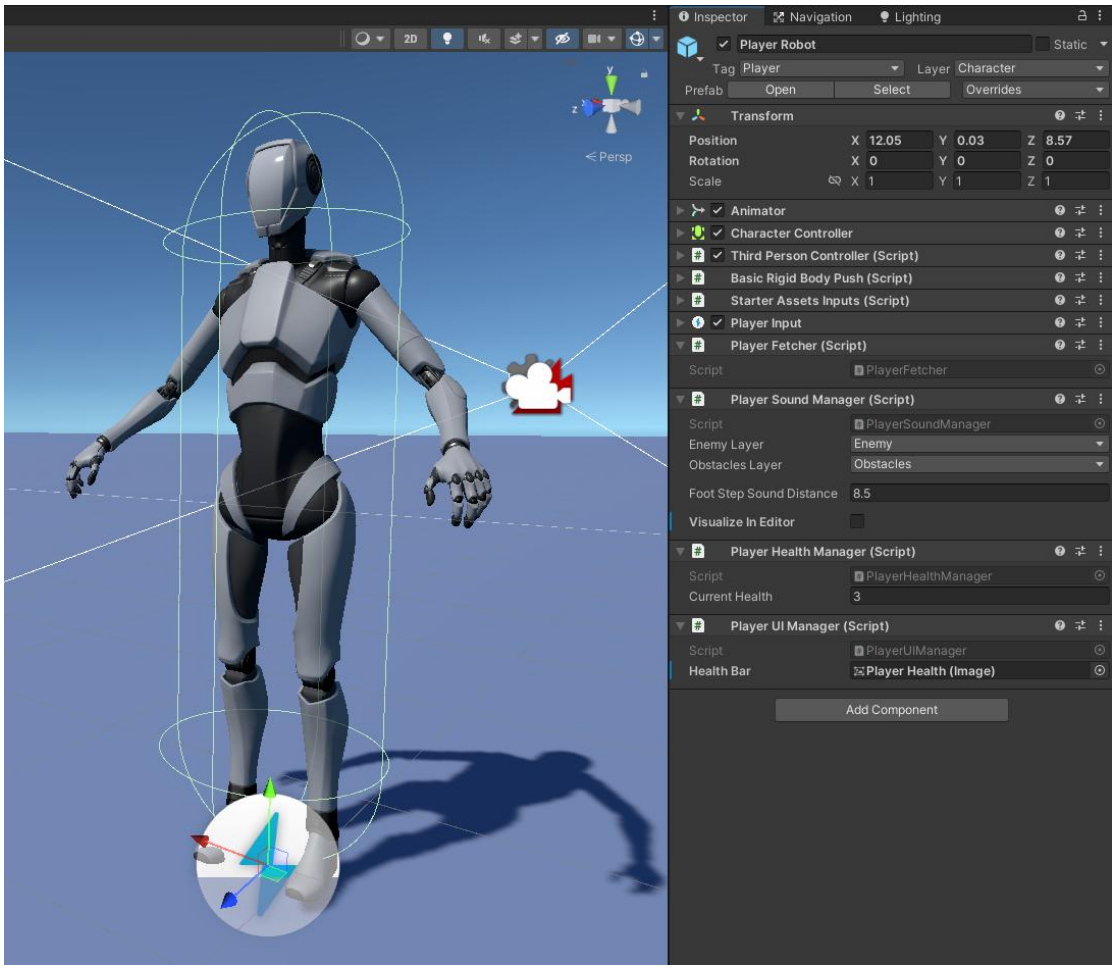
Some notable variables are the
- View Distance: which handles the AIs sight and how far away they can detect the player
- View Angle: this variable handles the AIs field of view and determines how wide or narrow the cone of vision should be
- Obstacles: this determines which layer an object should be in order for them to block the line of vision of this AI

The View Distance and View Angle variables are visualized using an editor script

Player System

As mentioned above, this project uses the simple third person project by unity as a baseline. As a result much of the players component and system are already built.

However, I have extended some functionalities in order to adhered to the assignment documents guidelines.



The first is the inclusion of the PlayerFetcher script. This is a static script that assigns the player (player gameobject) on awake. As a static script it can directly be referenced by other scripts hence boosting the performance of the game.

This script is used by the AI to get the player without using the FindGameObjectWithTag or FindObjectOfType system which may hinder performance as more AIs are added into the scene.

The second is the addition of a PlayerSoundManagerScript. This script governs the distance to which the players foot step sound is heard by the AI while the player is running. This script can also be visualized in the editor.

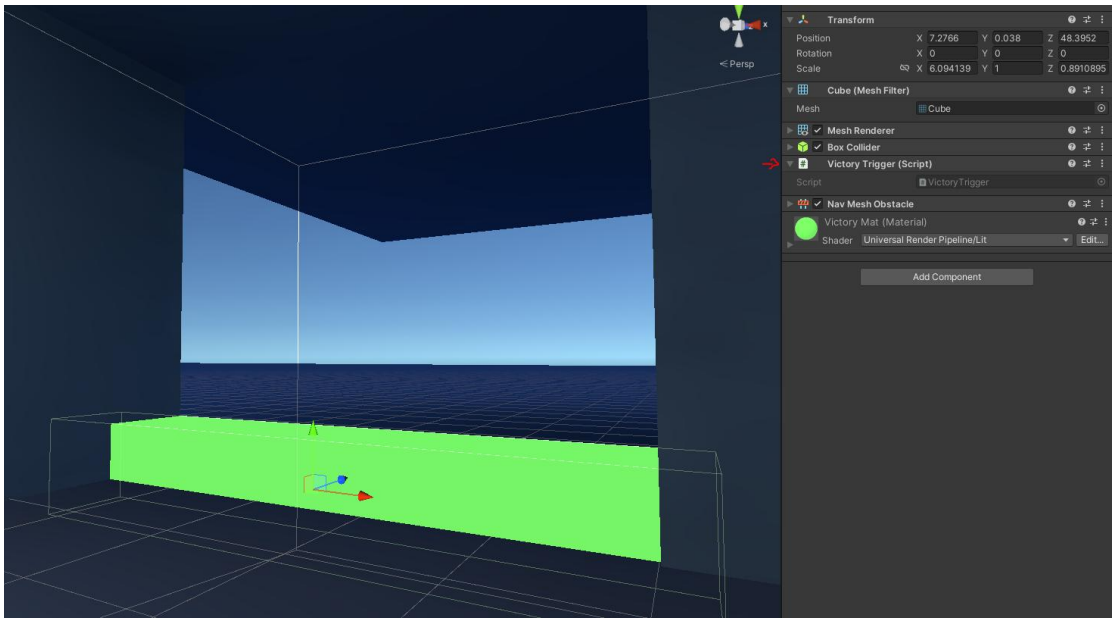
The third and fourth are the PlayerHealthScript, which will be damaged by the AI when the AI is attacking the player, and the PlayerUIManager which displays the player's Health Bar. In general, the health script manages the death state when its triggered (disable controls) for the character, however for the purposes of this showcase, the health script only triggers the game over state and visualizes the health bar UI.

Additional Systems

Aside from the AI and player system, i have also added a game manager script to control the victory and game over state.

This script is a static script (singleton) and therefore can be accesses by any script within the scene.

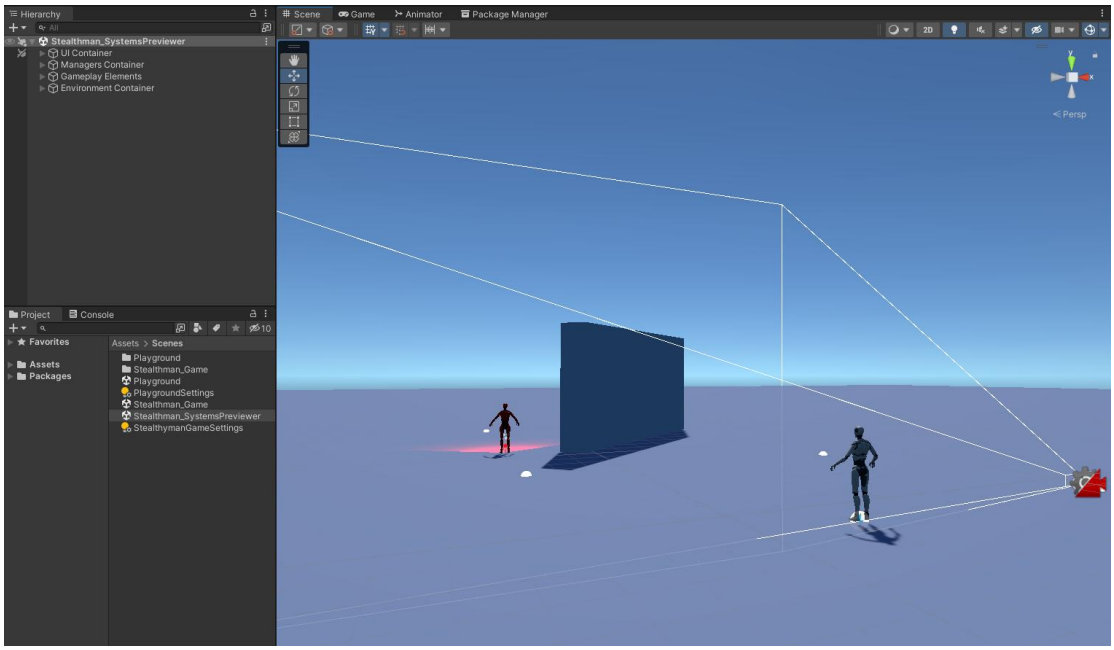
The game over state will be called when the player's health reaches 0 while the victory state will be called when the player reaches the victory line (represented as a green line in the scene) via Victory Trigger Script attached to it.



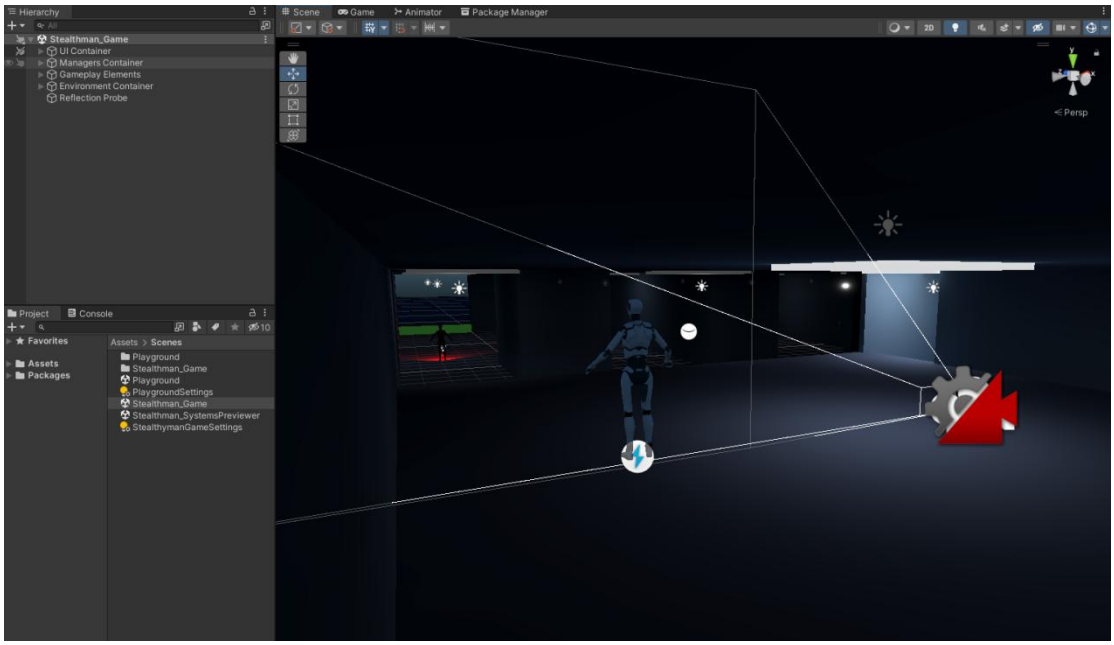
Demo Scenes

For this prototype, I have included 2 demo scenes. Both of these scenes are located under the **Assets**, **Scenes** Folder:

The **Stealthman_SystemsPreviewer** scene focuses on highlighting how the functionality works and is mainly used for initial testing.



The **Stealthman_Game** scene showcases how the system is integrated into a very simple level.



Project Extensibility

As mentioned previously, the function structure of this project are designed in a way where they are similar to blocks of code. This makes it easier for users to extend the function as they only need to adjust the function (block) they needed.

In addition, I have also integrated several functions (empty functions) in place with the purposes of future extension. In order to extend the functionality, a user can simply create a custom system within these functions. Here's a few examples:

```
//Note:
//The functionality of this function is currently only made for this Showcase
//Hoverer, it can be extended by adding additional functionalities as development progresses
1 reference
public void GameOverState()
{
    gameOverScreen.SetActive(true);

    //Do Game Over functions here
}

1 reference
public void VictoryState()
{
    victoryScreen.SetActive(true);

    //Do Victory functions here
}
```

```
1 reference
public void Dead()
{
    if (GameplayManager.gameplayManager != null)
    {
        GameplayManager.gameplayManager.GameOverState();

    }

    //Do Death Function Here

}
```

```
1 reference
public void OnAttack() //This function can be used to add attack system to the AI
{
    //Simple Damage example:
    player.GetComponent<PlayerHealthManager>().Damage(1); //Damage value is currently hardcoded for showcase only
                                                          //Can be adjusted with custom variables later


    //Do Attack / Damage Functions

}
```

Having functions such as these (blocks) makes the system more modular as it can be extended in a non destructive manner.

Polishing

The things i would like to polish further would probably be more towards the animation such as attack animations, alerted animations and also extending the attack and Death functionality for both AI and player.

In addition I would also like to integrate further game design elements such power ups or special abilities (reward elements), spawner system to add additional layer of the optimization into the game, and other system elements such as pause system, settings, scene management, etc, to further enhance player experience when the project is built.

Issues and Difficulties

Currently the biggest challenge I had was concepting the system to make it as simple yet as modular as possible.

This is ultimately the reason why I chose basic (Minimal) direct function to function call rather than use system such as interfaces or events as I believe that this system is more friendly to read while still modular enough that it allows flexibility for extension

Timeline

In general these are the main steps I used and the duration for each of them to build and document this prototype:

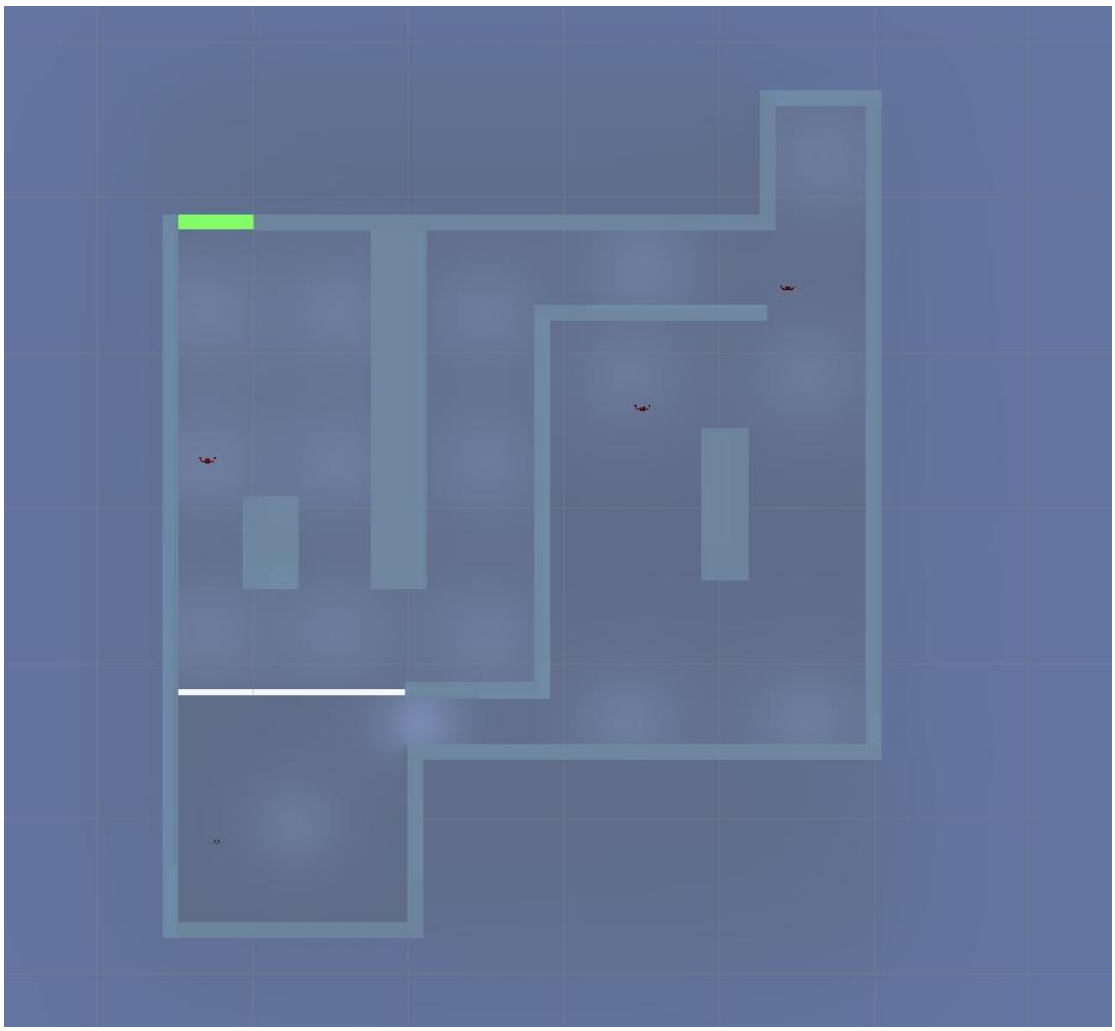
- system concept / flowchating (+3 Hours)
- base system implementation and testing (+6 Hours)
- additional elements integration (30 Minutes)
- demo scene construction + Light Baking (1 Hour)
- documentation (+5 Hours)
- video documentation and Editing (+1.5 Hours)

SET UP GUIDE

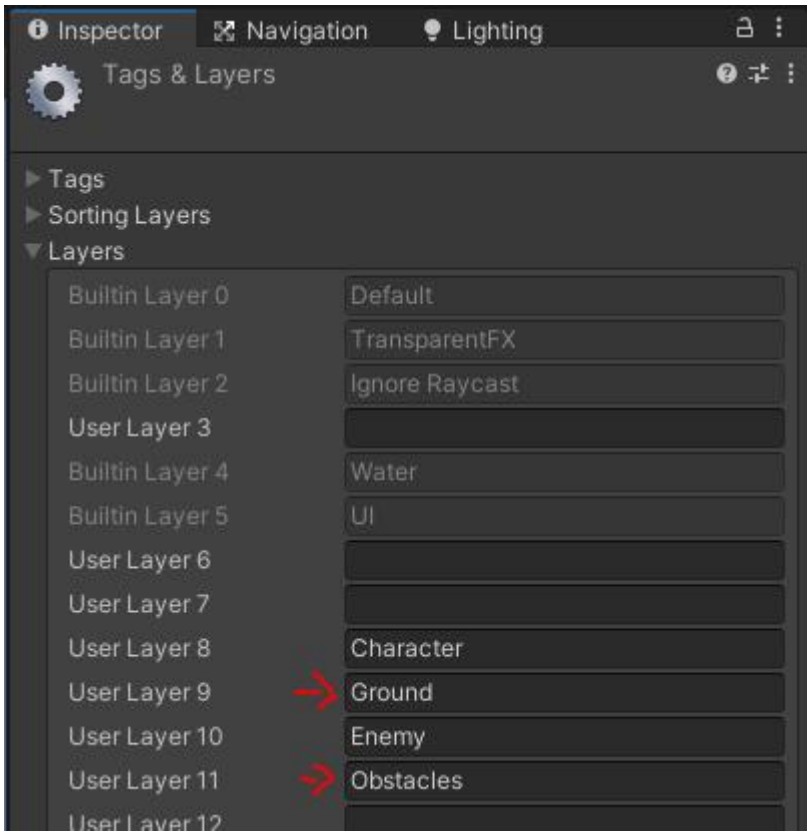
Welcome to the Set Up Guide section of this Document. This section showcases how to set up this asset and its integration, so lets get started.

Scene Set Up

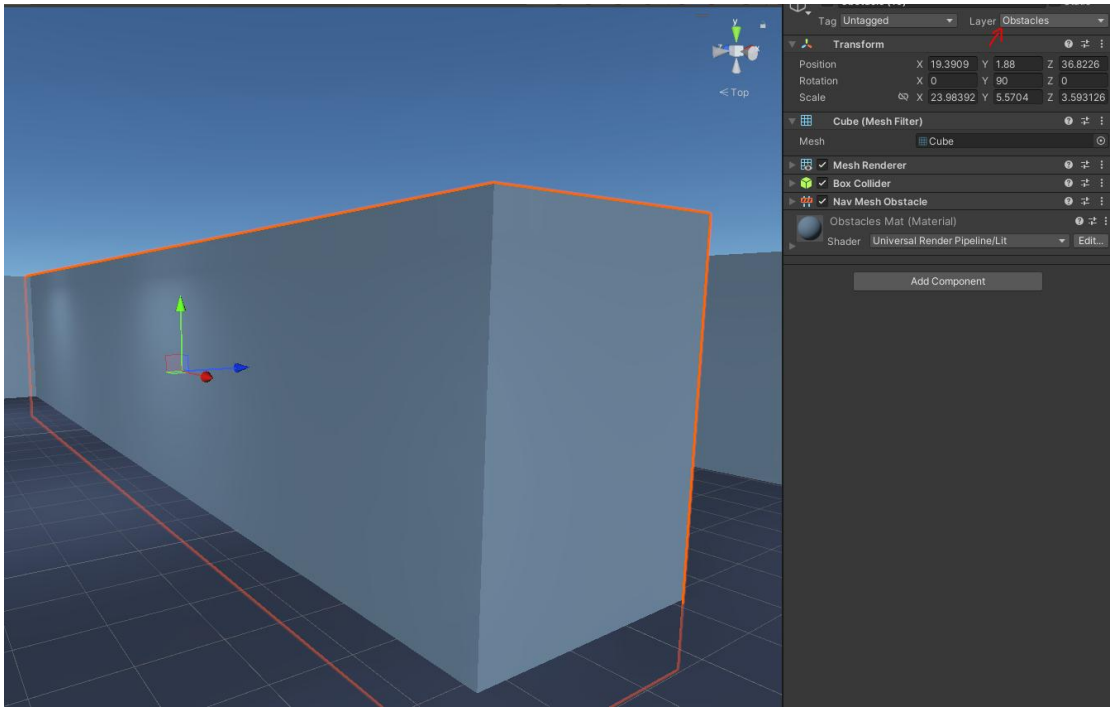
- Design your level to however you like

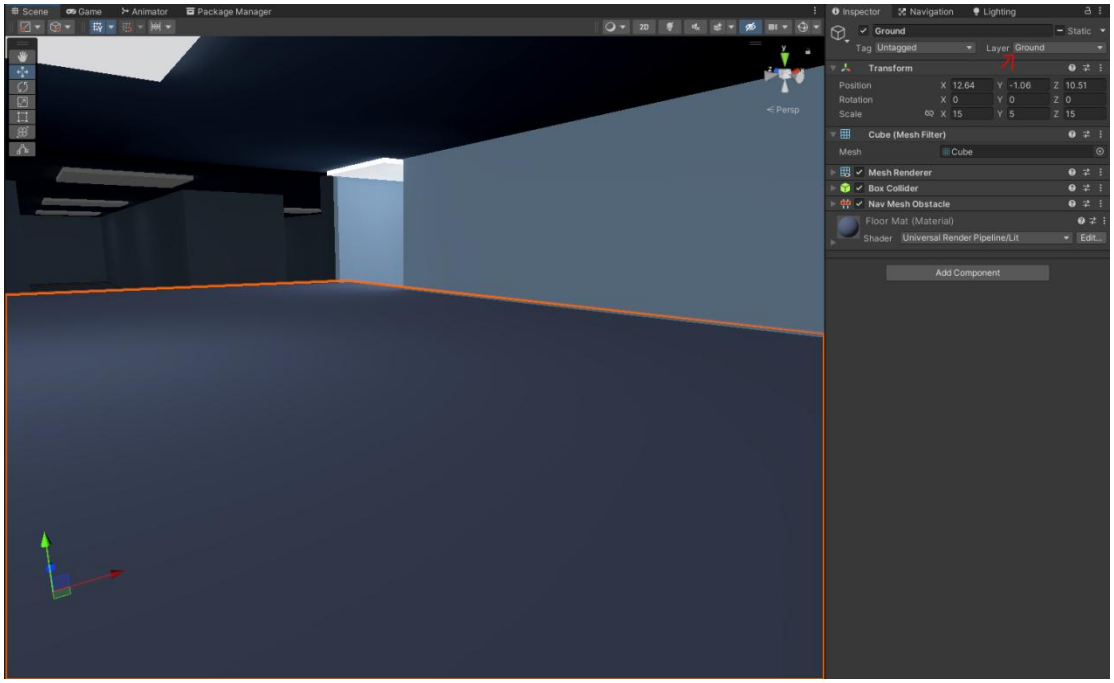


- Create a new layer called ground and obstacles

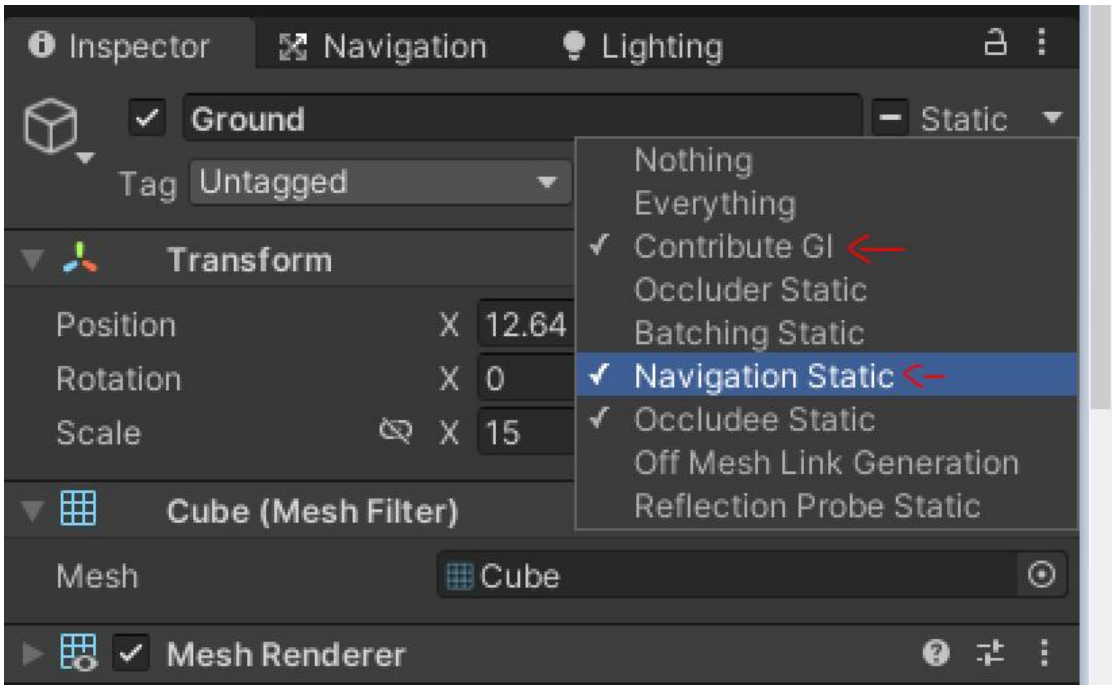


- Set all Wall objects layer to obstacles while Floor object layer to ground

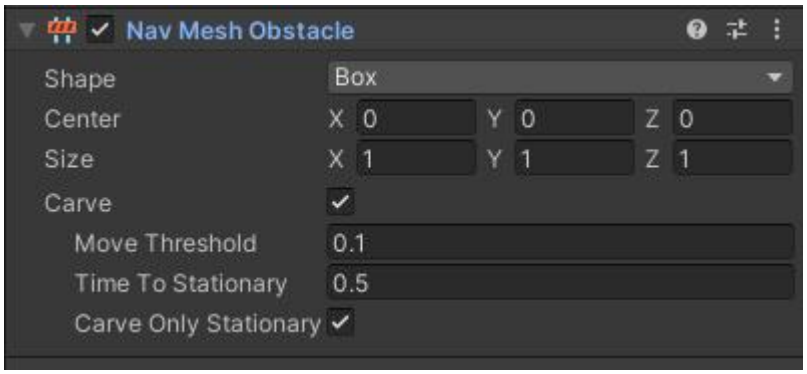




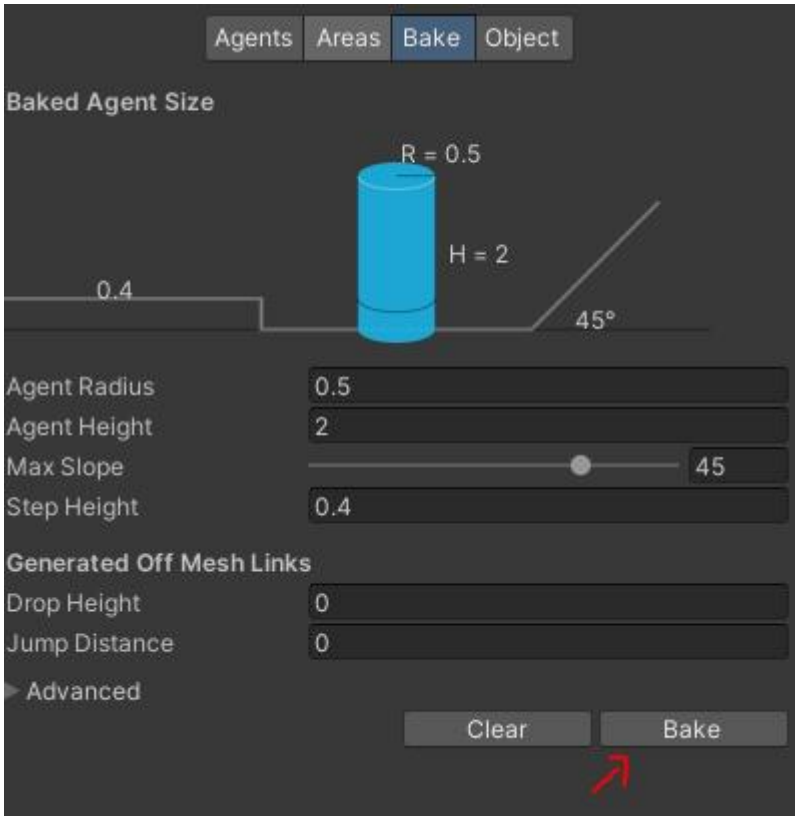
- Set Wall and Floor objects to Contribute GI and Navigation static



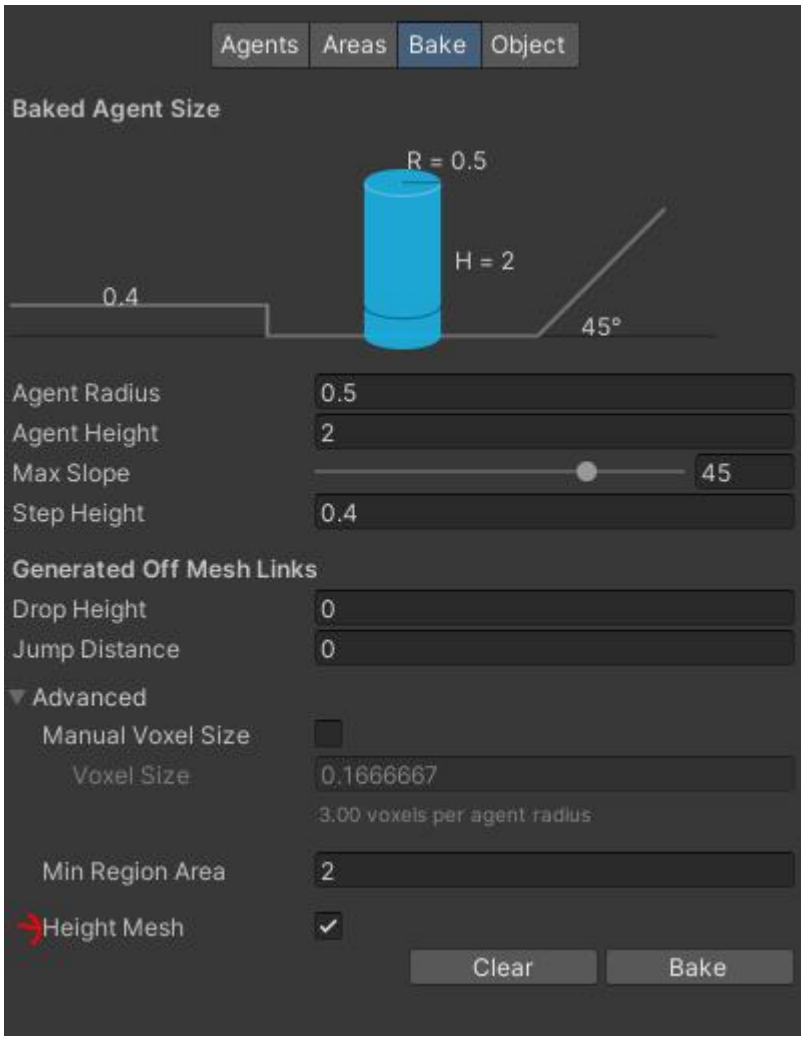
- Alternatively if the Wall objects are simple in shape you can add a Nav Mesh Obstacle component to it, set the shape that best represent the object (Box or Capsule) and set the Carve variable to True



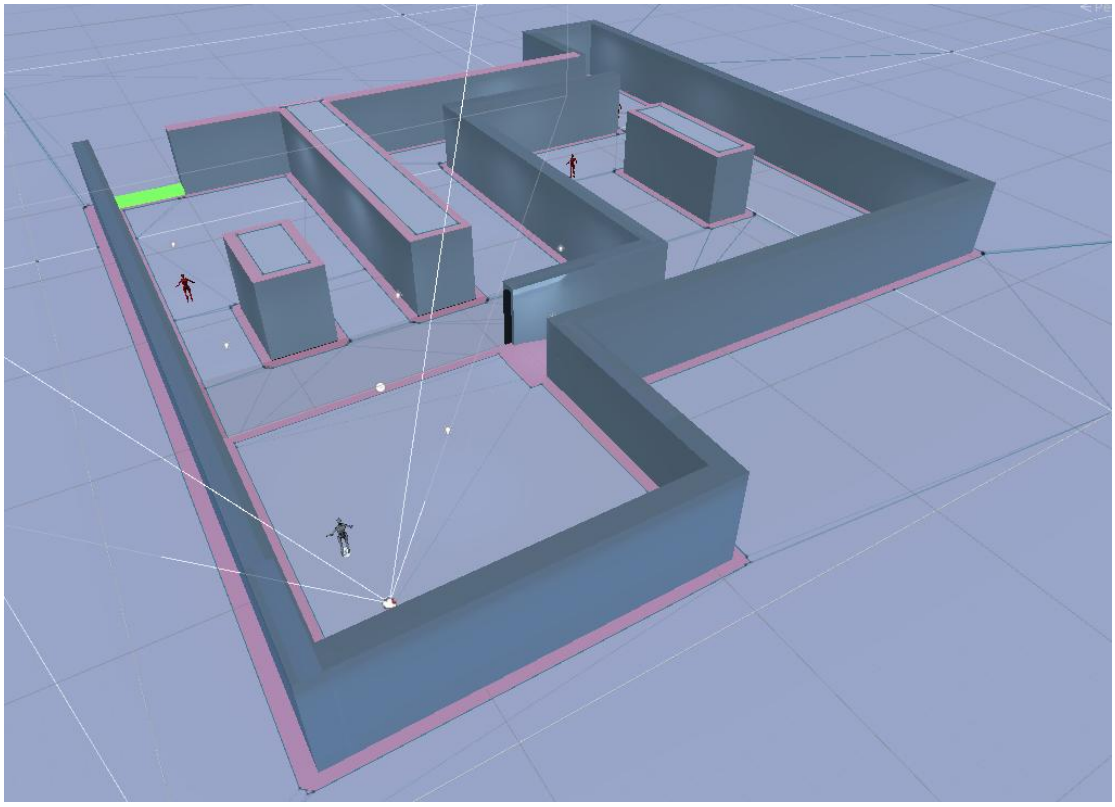
- Under the Navigation tab, click on Bake and Bake navmesh.



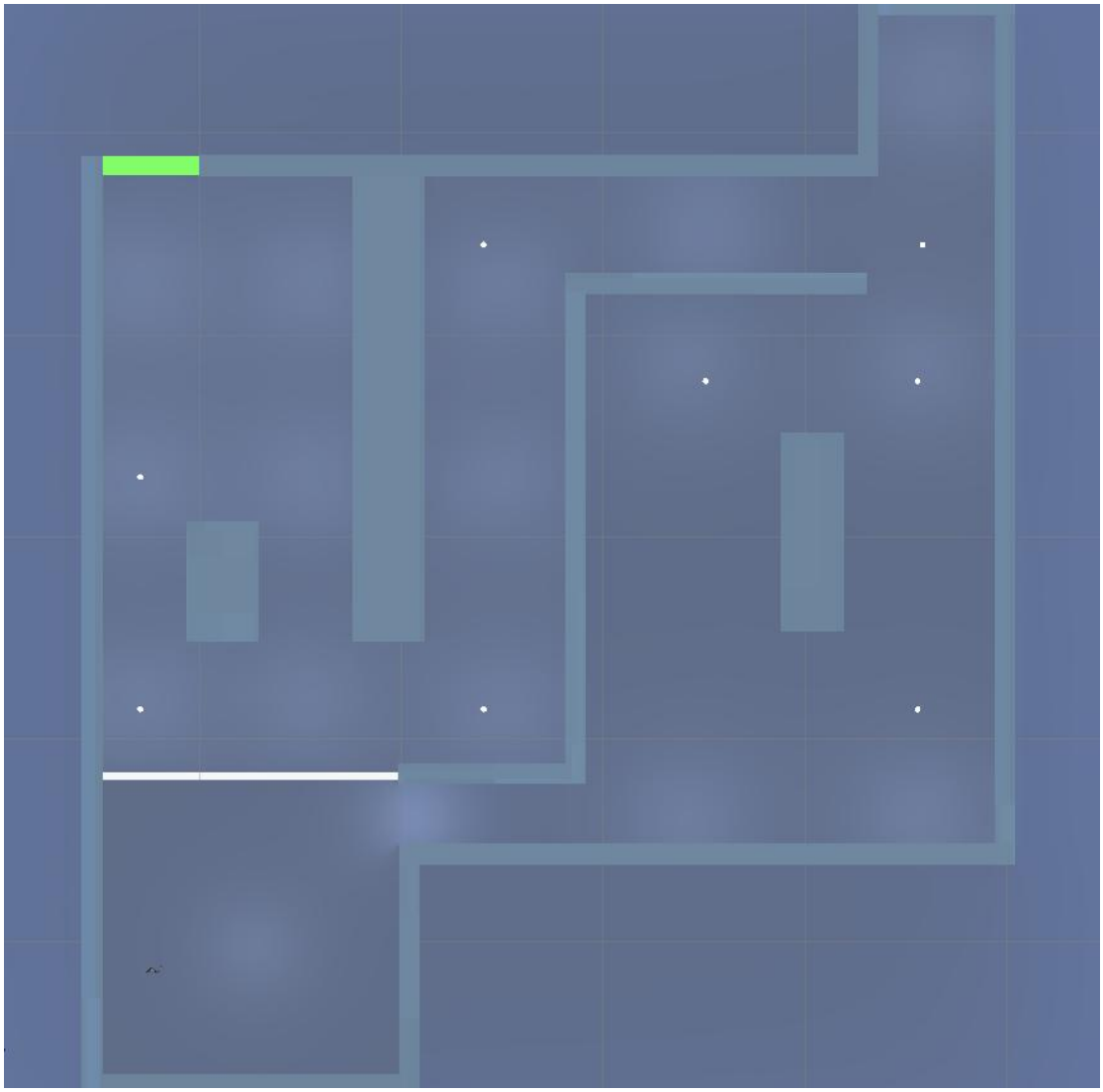
- Alternatively you can set Height Mesh to true to stop agents (AI) from floating above the ground. Note, changing this value requires you to rebake the NavMesh



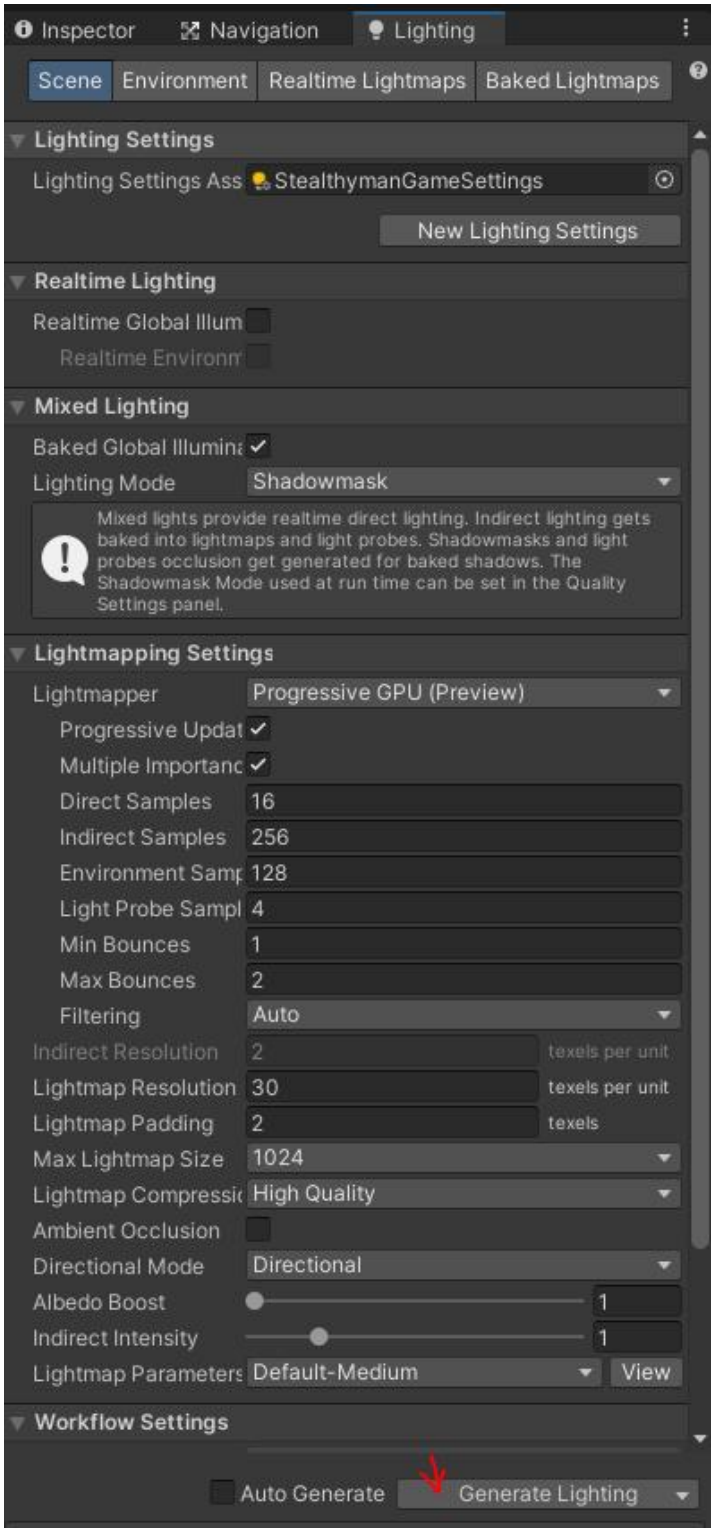
- If everything is done correctly The navmesh of your scene should look something like this



- Set patrol points for the AI to follow. Note: Do Not put patrol points above the ground or on non walkable path.

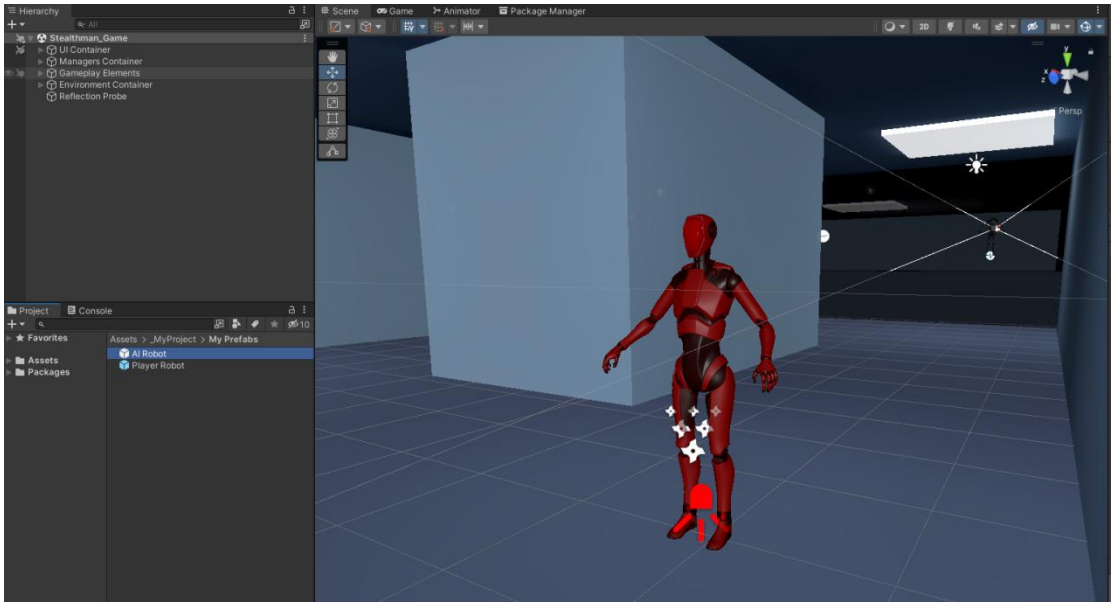


- (Optional) You can bake the lighting of your scene by going to the Lightings tab, create New Lighting Settings, set Bake Global Illumination to true, and click the Generate Lighting button.Note: make sure that you set the light (Light Mode) you want to include in the baking process to **Baked**

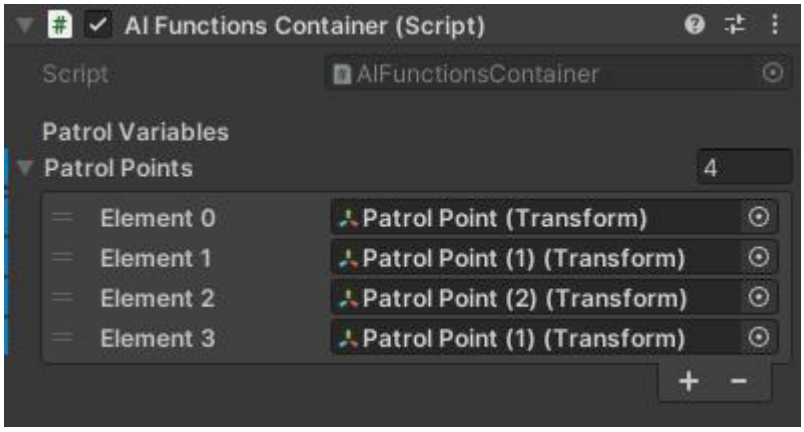


AI Set Up

- Drag AI prefab into the scene



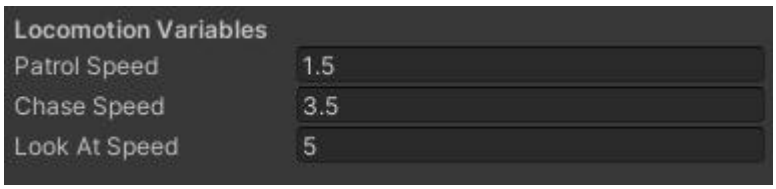
- Set AI patrol points



- Set Obstacles variable layer to **Obstacles**

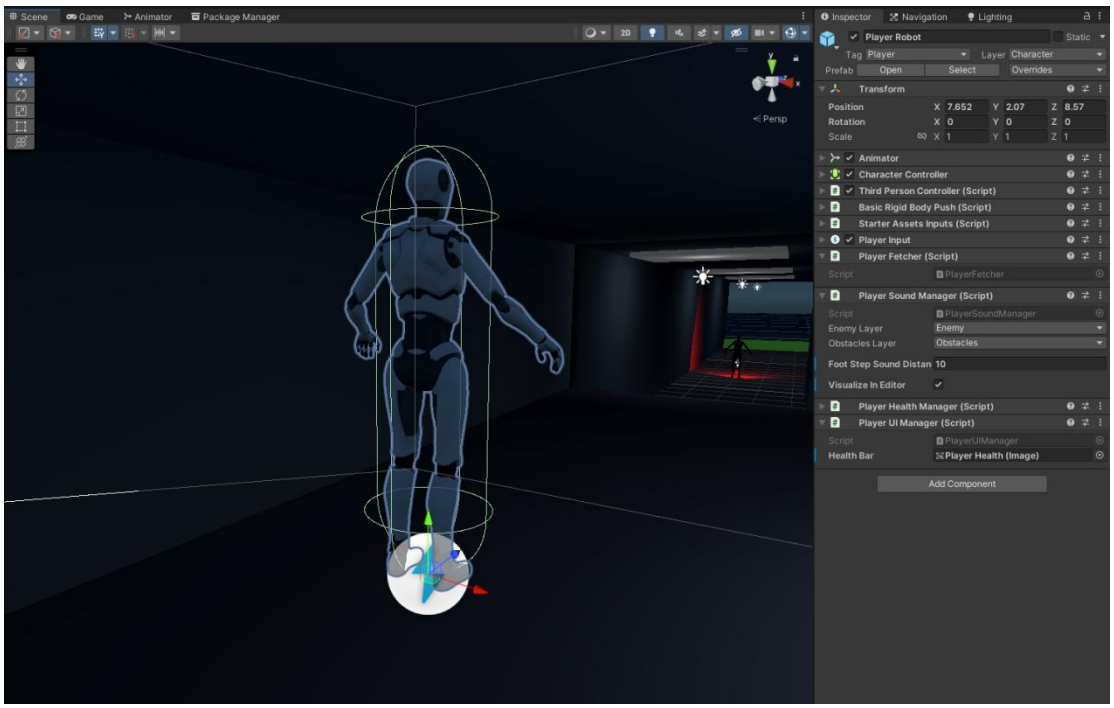
- Adjust AI Sight Distance and View Angle (optional)

- Adjust AI Patrol or Chase Speed (optional)



Player Set Up

- Delete the Main Camera Component of the scene and drag and drop the MainCamera prefab, the Player Robot, and Player Follow Camera the My Prefabs folder into the scene



- Set the Enemy Layer to Enemy and Obstacles Layer to Obstacles

- Adjust player Foot Step Sound Distance variable (optional)

Additional Elements (Optional)

- Create an empty game object and add GameplayManager script

- Create Victory and Game Over Screen

- Create Health Bar

- Assign Victory and Game Over Screen to Gameplay Manager

- Add a Cube, in its Box Collider component set Is Trigger to true, and attach the Victory Trigger Script