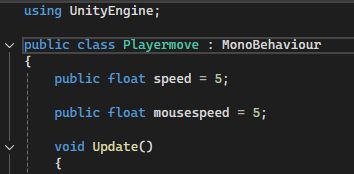
**Game Development Unity Tutorials/Learning Journal – Louie Barnard**

**Open this file using Word for correct formatting!**

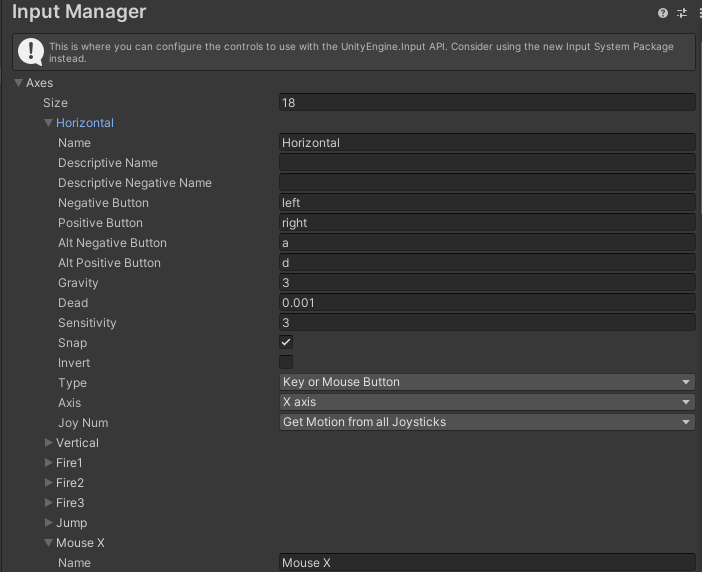
**Unity 20.3.46**

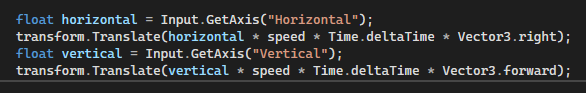
**Tutorial 1 - 3D Player Movement**

To give my player a simple 2 axis movement, I create a new script and name it “Playermove” and then drag the script onto my placeholder character (a cube). To allow for the script to perform multiple tasks at once, I begin by writing “public class Playermove : Monobehaviour.” Before I start scripting any movement, I need to designate the speed at which my character will move, so my next line of code is “public float speed = 5;”. The "public” in this line of code allows for the subject (the speed) to be accessed by all parts of the program (unity). After this step I then write “void Update()”, this step is crucial as it allows the code to run during every updated frame of the game, creating a seamless movement.



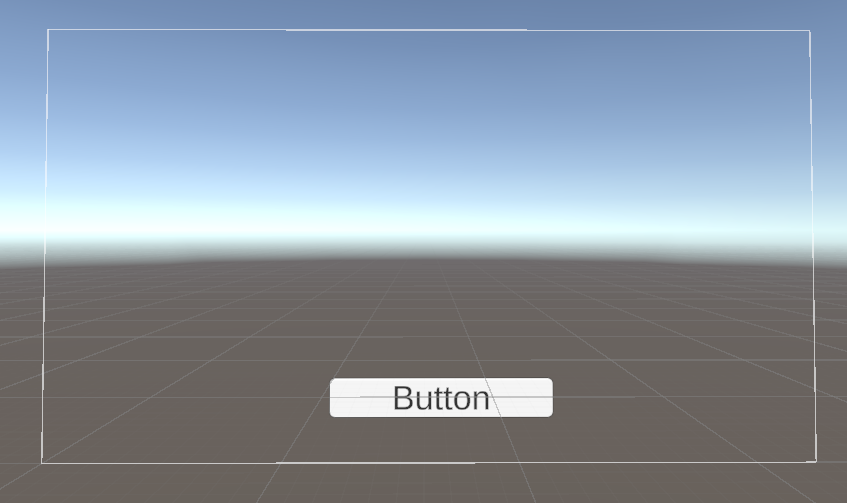
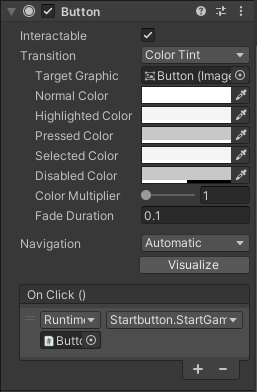
Now for the most important stage, scripting the movement. This is done in two steps as movement forward, backwards, left and right requires two axes to perform. To find what inputs that unity uses to control the axis, I first need to go to edit, project settings and then the input manager.



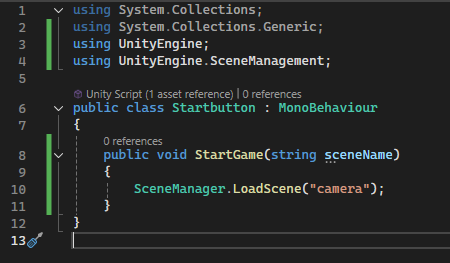
The next line of code begins with “float horizontal = Input.GetAxis(“Horizontal”);”, followed by “transform.Translate(horizontal \* speed \* Time.deltaTime \* Vector3.right”. This simple string of code allows for unity to understand which axis I'm using and the input I want to control it with (right arrow key). The reason I am using “Time.deltaTime” here is to allow for the speed to update independently from the frame rate of the game, the reason for this is obvious as all user machines that will be running my game will vary in processing power, so using this allows for a consistent experience across all computers. Now that we have movement for the horizontal axis, I simply copy and paste the code again, but I replace the axis and input to vertical and forward. 

**Tutorial 2 - Main Menu Start Button**

To start off with my menu, I create a new scene and set my viewport to 2D, I then add a Button into it through the Game object UI menu. To give the button a function I first need to give it an on click property which will allow the button to use a script once it is clicked, I then give it the “Startbutton.startgame” function and create a new script called “Startbutton.”



The next step in the process is writing a script that allows the button to perform a scene change from the main menu to the game.

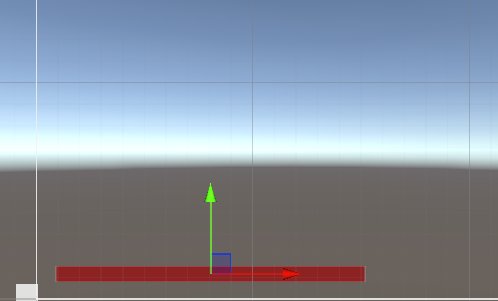


Starting with the script, I add “using UnityEngine.SceneManagement;” to allow for the function to recognize the scenes in my project. My next line of code is public void Startgame(sceneName). This code allows me to add the scene manager function which in turn allows the script to load or start new scenes, I then add my game scene called “camera.”

Reference used - <https://blog.insane.engineer/post/unity_button_load_scene/>

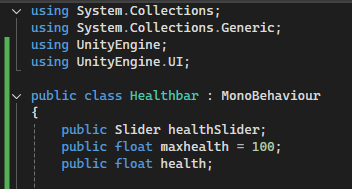
**Tutorial 3 - Interactive Health bar**

For the next tutorial, what I want to achieve is a traditional health bar that will decrease when the player takes damage, to start I add a UI Canvas to the workspace and give it an empty Game Object which I will rename “HealthBar”, I then resize the health bar into a rectangular shape. to create the background of the display, I add a new UI image to the Health bar and change its anchor settings using ALT+Left Click and to stretch it across the health bar, the next step is to change it to a deep red which is the colour I have chosen to represent low health. Now to create the actual health bar I repeat the same steps as the last, add a UI Image but this time will make the colour a more vibrant red. This is what I have so far:



To give the bar the appearance of declining, I must give the health bar a Slider component and assign the fill image to it. Once I do this step, I can adjust the value of the slider in its properties and can see it rising and declining in the viewport. After finalizing the aesthetic of the health bar, I can proceed with giving it functionality. Next i give the health bar a new script and name it “Healthbar”

The first thing I do in the new script is write “using UnityEngine.UI;”, the purpose of this line is to allow the engine to understand functions related to UI. Next, I make the Slider component public by writing “public Slider healthSlider;” this allows the code to be accessed by all classes. I also need to designate the max value of health points my character has, so my next line will be “public float maxHealth = 100;” and to create the health variable I finish with “public float health;”



When someone starts playing my game, I want their health to start at the maximum number of points, so within the “void start” section of the script I write “health maxHealth;”, anything that I write within this section will come into effect within the first frame of starting the game.

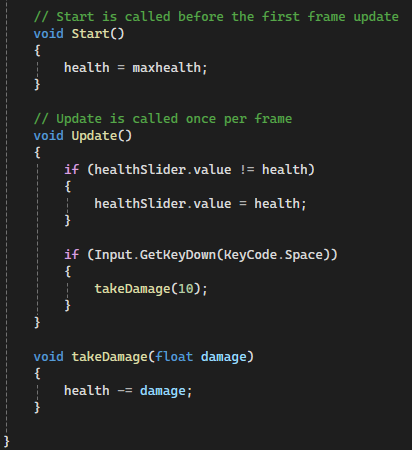
In order to synchronize the slider and health values of the healthbar, I need to create a condition within the code that checks if the slider value is equal to the health and will actively update it to be the same, so this leads my next line to start with “if (healthSlider.value != health)” followed by “healthSlider.value = health;

The script should now be functional however I have no way to test it yet. The way I am going to test the damage function is by binding my spacebar to a function that will remove 20 HP from it in increments of 10, this is easy and only requires a few lines of code.

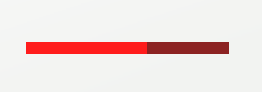
“if (Input.GetKeyDown(KeyCode.Space))

{

takeDamage(10);”.

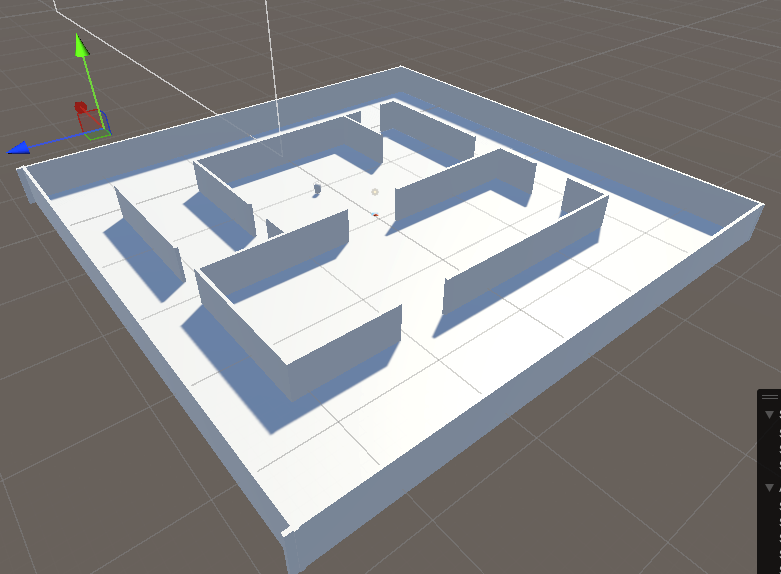


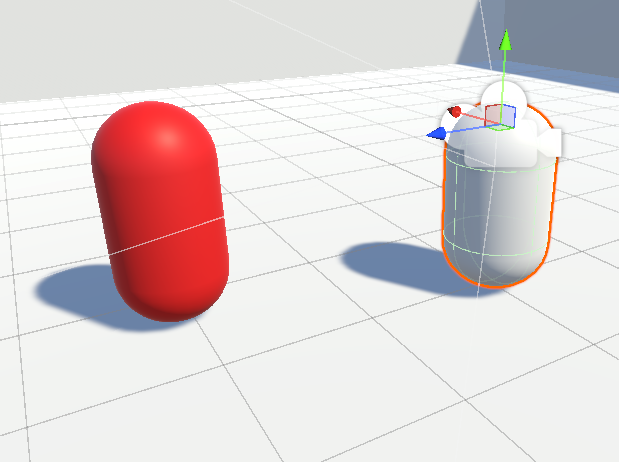




**Tutorial 4 – Enemy AI, Enemy Interactivity and Spawner**

To start off, I begin by creating a simple environment out of cubes that I will be able to test my game and the enemy AI within.

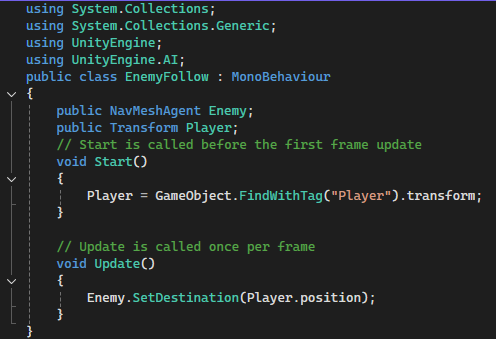




Next, I create a simple capsule and name it “Enemy”, I also give it a red shader to fit the part. For the ai to be able to traverse the surface, I select the floor plane and add a “Nav mesh surface” Component, following on I bake it which will give the surface the correct data in so that the ai will not be able to walk through walls etc. I also give the Enemy a “Nav mesh Agent” component so that it will be recognized as the AI Agent. For the next part I will create a new C# script and name it “EnemyFollow”. The first thing I do with the script is write “using UnityEngine.AI;” at the top so the unity engine can recognize the AI components which I will be using in the script.

Following on I need to create the Enemy variable so that my script will recognize the Enemy as the subject, I do this by writing “public NavMeshAgent Enemy;” I also need to do this with my player so following on the next line will be “public Transform Player;”.

The next part involves the actual behavior of my enemy which will be a simple chase, I did not want to incorporate any sort of roaming ai as of right now as I want my game to be a zombie like shooter or just an aim trainer. As I want the behavior to be a constant function, I must write it in the “void Update” area, so that it is called every frame. This line of code is “enemy.SetDestination(Player.position);. Since my Player and Enemy names are set correctly in the hierarchy, the script should recognize the names and shall function correctly. One last thing I do is give both the player and enemy their respective tags in the inspector, the tags being “Player” and “Enemy.”



When I tried to run the game, my script was not working as intended and the Enemy was not chasing my player. This had confused me. However, I asked my lecturer for help, and he stated that the enemy was not able to find my player and that I would need to add an extra line of code for it to be able to work and thus directed me to the unity manual. I needed to use the “FindwithTag” function so that my players GameObject location could be found. So, following on I added this extra line below

“Player = GameObject.FindWithTag("Player").transform;”

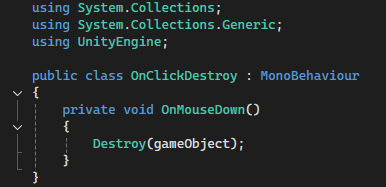
Reference used for code - [How to make an enemy follow player - Unity NavMesh](https://www.youtube.com/watch?v=UvDqnbjEEak)

To give my game some playability, I am going to create a script that destroys the enemy when the player clicks on top of them, i will not include any gun models right now as this is only a test however this is to simulate shooting of some sort. But before i do that, i will need to add a cross hair. I am not going to make a complex crosshair, only a small circle. This is easy as all i need to do is go to the UI menu and add a new image and change it to the "knob” option and then resize it accordingly. As my game is in the style of an FPS, i will need to lock the cursor to the middle of the screen to align it with my cross hair so that if i have any interactivity with an object it will be in the correct position.

As this is only a small line of code, I am going to add it inside the camera movement script so as not to overcomplicate my workspace, the line of code I add looks like this, “Cursor.lockState = CursorLockMode.Locked;”.

Now to create the click to destroy script, I make a new C# script and rename it to “OnClickDestroy”. my first step is to remove the start frame and update frame functions as i will not need this, below the public class i write “private void OnMouseDown()” which is the input I will be using to destroy my GameObject, and following that I write “Destroy(GameObject);”

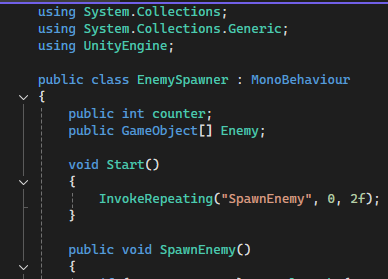
After this i drag the script on top of the Enemy in the hierarchy and then inside of the Enemy’s inspector i give it a 3D Capsule Collider so that the click will be recognized.



Reference used for code - <https://www.youtube.com/watch?v=ql6gnNcTgfM&ab_channel=MegaTechnoYT>

Now for the last part of the fourth tutorial, the enemy spawner. In the future I would like to expand upon this and make it a round-based shooter but for now I only want a spawner that will create a constant flow of enemies. The first step for this is creating an empty game object and I rename it to “EnemySpawner”, this will act as the holder of the script. Next, I create a new C# script which I then also rename “EnemySpawner”.

When i start editing the script the first thing that I do is delete the “start before the first frame” prompt as I will not be needing it for this. My initial line of code begins with “public int counter;”, int stands for integer (meaning whole numbers) this allows for the for counter to count in whole numbers to align with my number of enemies. Next i create the variable for my Enemy, which looks like this - “public GameObject [] Enemy;”. these square brackets “[]” here represent an array, meaning that I can store multiple values inside of a single variable, I need this because I will need multiple enemies to spawn. Next, I delete the default “Update is called once per frame” prompt and in its place, I write “public void spawnEnemy()”, this new variable is necessary so that my next line of code can recognize the spawn feature. Next i need to create a repeating function that will spawn my enemies at a certain rate. This is done with this line “InvokeRepeating("SpawnEnemy", 0, 2f);” as you can see the code will recognise the spawn enemy variable that I created before and repeat it at a designated rate, this being represented by the “2f” (2 seconds). As i want my game to be a test of reaction time and aim skills, I tested it until i found a desired spawn rate and 2 seconds felt perfect.

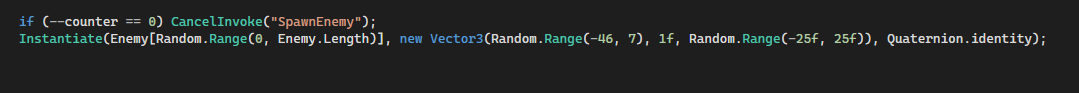


One thing i needed to implement into the script is a system that will stop the spawning of enemies if the player successfully destroys them all fast enough, this will be the way to “win” the game. the way i create this system is by writing “if (--counter == 0) CancelInvoke(“spawnEnemy”);”.

Following on, the next step in the process is writing a line of code that will allow for my enemies to spawn within a certain area, this area being the floor of my environment. To get the values of the area, I place down a cube and move them to the top left corner and write down the value of the x and z value and then I repeat the same step for the bottom right corner. Now that i have the location information I can start writing. The next line looks like this -

”Instantiate(Enemy[Random.Range(0, Enemy.Length)], new Vector3(Random.Range(-46, 7), 1f, Random.Range(-25f, 25f)), Quaternion.identity);”.

The function “Instantiate” simply allows for the script to create a GameObject from a blueprint, the blueprint being my enemy. in order for the script to be able to create clones of the same enemy, I drag the enemy GameObject from the hierarchy into my asset box to convert it into a prefab. Following that i write “Random.Range” so that the script will select a number at random within a given range, these ranges will be determined by the “new Vector3” prompt, which will allow the script to use 3 separate values to determine a certain outcome (hence the 3). Here I can use the location values that I had written down initially, this being “–46,7” and “-25f,25f”. And lastly the “Quaternion.identity” function assures that the enemy will preserve its rotation data when cloned.



To Finalize, I drag the EnemySpawner script on top of its respective Empty GameObject in the hierarchy and the last step is to assign my prefabbed enemy inside of the component tab.



Reference used -

<https://www.youtube.com/watch?v=NZ867Ep3KZs&ab_channel=Pablo>