**Programming Tutorials Anish Pun Year 2**

For this module, I will be working on a 3D Platform/Action game and writing down tutorials based on what knowledge I have gained. As I am learning Unity, even simpler mechanics will be unfamiliar to me; the tutorials will be great for personal use.

**Health System and HP Bar:**

The first tutorial will tackle the health system within the game.

**Setting the scene:**

In your scene, create a capsule as a representation of the Player. On that capsule, attach a new C# script and name that “HP”. Within the script, write these lines at the top.

A black background with white text

Description automatically generated

We will be using those libraries to transition between scenes and to get access to the UI classes. This will correlate the HP value to the HP Bar UI.

We need to set two important variables that will determine the player’s max and current health, we will use integers for this system:

A screen shot of a computer program

Description automatically generated

In the void start area, write the line above. We are setting the player’s initial health to be the maximum amount.

\*I have used the integer value as I want the player to be able to survive about 3 hits. This is more reliable for small values than a float value. I believe a lower hit point value is appropriate for my game to increase the stakes; each hit is potentially a deadly blow. You could use a float value if you wish to tackle higher, specific numbers for the game you are making. You could also use higher integer values if you are dealing with an RPG-style health value. The feel of the game will change depending on how vulnerable you make your player.

**The HP Bar:**

With just these two variables, we can add a Health Bar to the UI. Right-click on the hierarchy and add a canvas. Right-click>UI>Canvas. Add an empty child object to the canvas and rename this to “Health Bar”. Readjust this bar to where you would like the health to be on the screen. Right-click the Health Bar object: UI>Image. Set it as a dark red colour and adjust the anchor to stretch on both axes. Name this “Background” and repeat this step with “Background” as the parent. The child object should be set to a bright red colour and renamed “Health”. It should look like this so far:

A screen shot of a computer

Description automatically generated

Health UI

Add a slider component to the Health Bar game object and assign the Rect Transform component of the “Health” gameobject to the “Fill Rect” field. The direction should be set to left to right. This is because my health is on the bottom left section. If placing it on the right side of the screen, filling direction from right to left is ideal in that scenario.

A screenshot of a computer

Description automatically generated

Going back to the script, write these lines:

A screen shot of a computer program

Description automatically generated

This makes it so the HP bar moves in correlation to HP changes. It states that when the slider’s value does not match the HP value, the slider value will be set to the HP value.

**Damage and Healing Mechanics:**

With the HP integers and Bar UI in place, we should implement mechanics that damage and heal the player. I am going to create general damage and healing scripts that can be used by different objects in the game.

For healing:

A computer screen with white text

Description automatically generated

The if statement makes it so HP cannot be over the max value. On the other hand, I can call on the “heal amount” integer whenever creating a healing source in the game.

\*I have added a heal amount check if heal amount < 0 and changed the if statement to currentHp + healAmount > maxHp. This will prevent any issues with negative healing numbers and allows for change of maxHp as using 3 in the statement locks

For damage:

A computer screen with white text

Description automatically generated

Likewise, I can call the “damage amount” on any damage sources to the player. The if statement will change the scene to the game over screen when HP is equal to or below zero. As for how to set up menu screens and the buttons needed to navigate them, look at the tutorial below.

\*Another method for showing a game over screen is to hide and unhide its canvas element. This allows the canvas and any scripts associated with it to interact with the scene. For my prototype, I had to move the canvas from the game over scene to the game scene. I did this as it was more efficient, button interactions lead to the game scene so the game over canvas being in the main scene is ideal.

**Menus and UI Buttons:**

For the start and game over screens, we will create different scenes for Unity to transition between. So, create new scenes for each.

**Setting the scene:**

Working in 2D view is ideal when dealing with UI, click the button on the top-right section on the scene view: “2D”. Right-click the hierarchy and create a canvas (UI>Canvas). The canvas is where we will set the UI elements such as text, panels, and buttons.

For the start screen, create a panel as a child object of the canvas (UI>Panel). As child objects of the panel, create a text element and 2 buttons (UI>Legacy>Text/Button). I will only implement two buttons in this example, the “Start” and “Quit” buttons. Manipulate the text element to the desired title of the game and implement two more text elements on the buttons, one each and label them as “Start” and “Quit. This is what the menu could potentially look like.

A screenshot of a video game

Description automatically generated

With the hierarchy set up like this:

A screenshot of a computer

Description automatically generated

I have also created a Game Over screen linked to the HP system in the tutorial above. It was created using the same steps as the Start Screen.

A screenshot of a video game

Description automatically generated

\*You may have noticed that I have used the Legacy UI features such as the buttons and texts. This is simply because the font I am using only functions with the legacy UI features. I recommend using the new TextMeshPro options as they have higher resolutions and more features.

**Buttons:**

There are now buttons on the menus, but they do not do anything yet. Create a new C# script named “Buttons”. At the top, write down “UnityEngine.SceneManagement”. This will allow us to switch between the different scenes we have set up by using SceneManager.LoadScene(“NameofScene”). Make a public void for each button and use this line for each respective scene.

“Application.Quit()” will close the Unity program. It should look something like this, repurposing the Restart button as the start button for the start screen.A screen shot of a computer program

Description automatically generated

Attach this script to the Panel, a new feature should be added to each button:

A grey screen with white text

Description automatically generated A screenshot of a computer

Description automatically generated

Drag the Panel to the bottom section and select the associated statement to the button. When a button is clicked, the statement associated with it will be called. By repeating this, you can add as many menus and buttons as you want to the game.

**Pausing:**

Pausing and unpausing work slightly differently from the other menu systems.

To start off, create a panel in the hierarchy. Following similar steps to the start/game over menus, create a pause menu UI and lower the alpha of the panel. This will make the menu transparent and allow us to view objects behind the camera. It should look something like this:

A screenshot of a video game

Description automatically generated

Notice the transparency of the panel and the repeated use of the main menu and quit buttons. You can easily change the alpha value if you do not want the player to see the scene. The same code above can be placed in for those buttons.

The only new button “Continue” and this is linked to the pausing feature. Create a new C# script and name it “Pausing”.

A black screen with white text

Description automatically generated

These lines must be written at the top to create a paused state and the button used to pause the game. The pause panel canvas also allows for easy hiding/unhiding of UI.

A screen shot of a computer program

Description automatically generated

Both statements are simply a reflection of one another. By understanding how to pause, you can unpause by reversing the values. PausePanel.enabled = true brings the menu up when paused. Time.timeScale = 0 freezes all actions within the game. The cursor lock and hide lines also reveal/hide the cursor when pausing so you can interact with the buttons. The paused bool is true when paused, this allows us to efficiently pause/un-pause with the same button using this code.

A screen shot of a computer program

Description automatically generated

This makes it so pushing the P key will both pause and unpause the game.

Include the subject Unpause() on start as this will hide the menu and unpause right as you press play.

A black screen with white text

Description automatically generated

With this, you have a system that lets you pause, un-pause, return to the menu, and exit the game. You can easily add different buttons and menus for things such as options by following this tutorial.

**Turrets, Damage, and Health Pickups:**

**Turrets:**

In this tutorial, we will call the damage and healing statements from the HP class, while implementing an immobile turret enemy. In a new C# script named “TurretAI”, write the following lines.

A screen shot of a computer program

Description automatically generated

\*In the current version of the script, I have used void Reset instead of void Awake, the player object is located even before the enemy is loaded into the scene. From my understanding Awake is called when the enemy is loaded into the game but Reset is called when the entire scene is loaded in.

Transform player will be used to track the player; the player object should be dragged onto this field in Unity. We will also add a projectile prefab (sphere with script) into the Game object projectile. On void awake, the line above finds the player whenever the object is loaded in.

A screen shot of a computer

Description automatically generated

The attack and sight range are defined in Unity as they are public, and the code above will be used to set the attack range. If the attack range is within the sight range, the turret will attack the player.

A computer screen shot of a code

Description automatically generated

This is the code called when the player is both in attack and sight range. The turret will first face the player and then instantiate a rigid-body projectile with large forward force and slight upward force. The attack rest bool is then set to true. The line below is invoked at a timer we can choose in unity, this sets the attack rest bool back to false and the turret can attack again. You could use values like this. Simply attach this file to the enemy model/primitive shape.

A screenshot of a computer program

Description automatically generated

**Projectile:**

As you can see above, I have placed a prefab on the projectile section of the script. This prefab is made using a simple yellow sphere that has a script attached to it.

A screen shot of a computer code

Description automatically generated

This script states that in a collision between the player object and this sphere, the script will call the HP class and obtain information on the player’s current health. If the health is not null, the projectile will deal 1 damage. After this fact, the object is destroyed. I have also set another destroy game object line with a timer in case the projectile misses. You could also set the project to destroy on any collision.

**Healing:**

Now we have a working health bar and can also lose that health. To regain that health, players can use health pickups. This is yet another simple sphere object with a script attached.

A screen shot of a computer program

Description automatically generatedA green ball in the air

Description automatically generated

This script simply states that on collision, if the object has the player tag and their hp is not null, the object will destroy and heal by heal amount (2). We are calling on the HP class we wrote above and setting a value for the heal amount.

By using this formula, you can also create more pickups such as power ups.

**Item Pickups and Throwing Items:**

With the health system in place and enemies attacking us, it is time to fight back.

**Enemy Attributes:**

Currently, the turret and any other enemy we create do not have health values. Create a new C# script and add it to the enemy object. Write the following code:

A computer screen shot of a program

Description automatically generated

This follows the exact same principle as the player health code, the difference being that when an enemy reaches 0 health, they are destroyed. This can also be changed to add death animations to the enemy and destroy the object after a set time. With the enemy's health set, we need a way to lower their health.

**Player Projectile:**

For my game prototype, I will create a throwing knife as the main method of attacking the enemy. I have created a simple knife prefab in Unity and given it the tag “Item”. I will create a pickup method that will destroy any object with the “Item” tag. This can be used to pick up any item added to the game. Write the following variables in a new script. Attach this script to the player object.

**A computer screen shot of text

Description automatically generated A sword on a stand

Description automatically generated**

The bool will determine if we are able to throw the knife. Pick-up distance will determine just that. Finally, knife origin is where we will instantiate the knife from, this is an empty game object placed near the player’s head. Following is the code for the input and the instantiation of the projectile.

A computer screen with white text

Description automatically generated

A computer screen shot of a program code

Description automatically generated

Upon clicking the left mouse, the knife prefab will be instantiated from the knife origin and rotation. The direction is determined by where the player is facing. Force is then added to the rigid-body component of the knife prefab, multiplied by the throw force we determined. The bool will be false once the knife is thrown.

**Knife Prefab:** A screen shot of a computer code

Description automatically generated

Above is the code for the throwable knife, attach this file to the prefab.

The setup is like the projectile enemies are firing, the projectile will check for the tags “Enemy” and “Level”. On both occasions, velocity will be zero on collision, but enemies will receive a damage value of 2. After this collision, the knife will remain on the ground.

**Pickup system:**

I want the player to walk over to the knife, press a button, and pick it up.

A screen shot of a computer program

Description automatically generated

In this instance, once the player presses E, a sphere around the player at the radius we set (pick-up distance) will check for colliders. Any colliders the sphere encounters will check for the tag “Item”. This means we can use this system for any number of items and interactions if the object is tagged “Item”. In this instance, I have tagged the knife as an item, but it can be tagged “Knife to differentiate between other items. The knife object is destroyed, and the bool is set to true.

If you look at Pick-up and Throw Knife subjects, each contains a knife UI statement.

**Knife UI Toggling:**

I have added a small UI indicator to know when the player has a knife. I created a canvas and placed an image to show this. The canvas is enabled and disabled when the player throws the knife and when they pick it back up.

A screenshot of a video game

Description automatically generated

I have also added a small crosshair to show where the knife is being thrown.

**Conclusion of Tutorials:**

With these tutorials, I have created simple components that will allow me to expand the game while providing some core mechanics. Items, Picking Up Items, Enemies, Projectile Instantiation, UI and Menus, Prefabs, Health Systems, and more can be implemented. For the prototype, I have also included a simple movement system (walking, sprinting, and jumping), first-person camera movement, and moving enemies. However, I have not included them in this tutorial due to the usage of some tutorials.