

CS7GV5 Real-time Animation

Interactive Game with Animated Characters

Guided by: Dr. Rachel Mc Donnell

Completed by: Yuzhou Shao (Student number 19322035 ARVR)

Introduction

I have used a higher-level Graphics engine which is Unity and referenced its official tutorials and other related academic materials to develop this project. Some of the Codes are programmed through Visual Studio C++.

Youtube Game Demo

The Youtube Scenario demonstration of the project available as below link.

<https://youtu.be/wDm5igGL7eI>

The demo is approximately 8 minutes, and the game can be run longer than 1 minute (5-12 minutes).

Which parts were coded by yourself and which out-of-the-box features were which parts were coded by yourself:

which parts were coded by yourself and which **out-of-the-box features were used**. It should also include a link to an **online repository** containing your code.

Imported Assets

The assets of the entire project are imported from here.

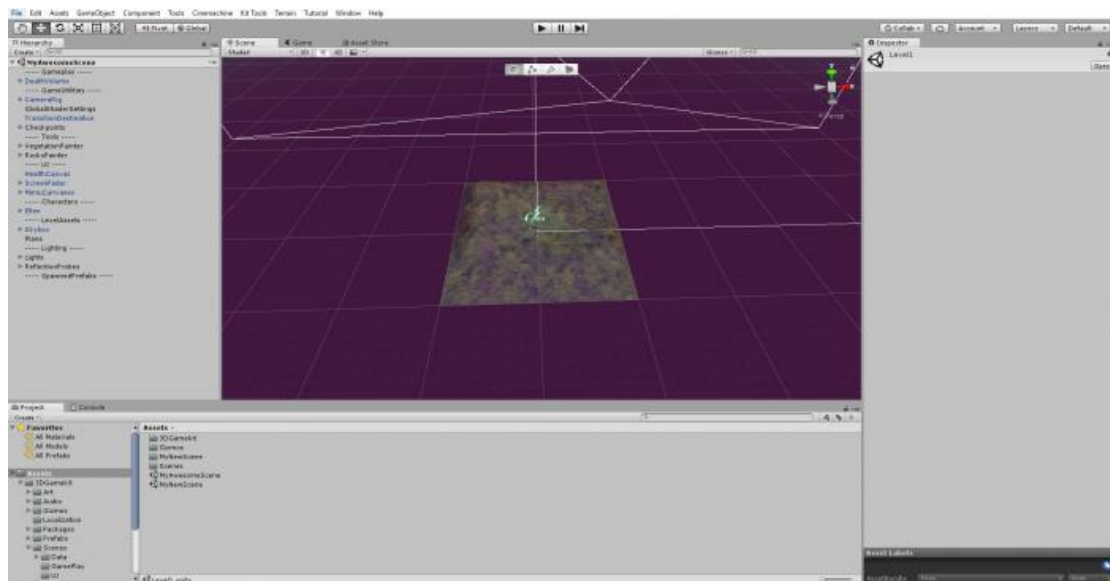
<https://assetstore.unity.com/?q=3d%20game%20kit&orderBy=1>

(Notice: The assets such as the protagonist, enemies, rock, structures, vegetation interactive mechanism ,and so on are all imported from the above link, but the game was entirely constructed and logic was implemented by myself.)

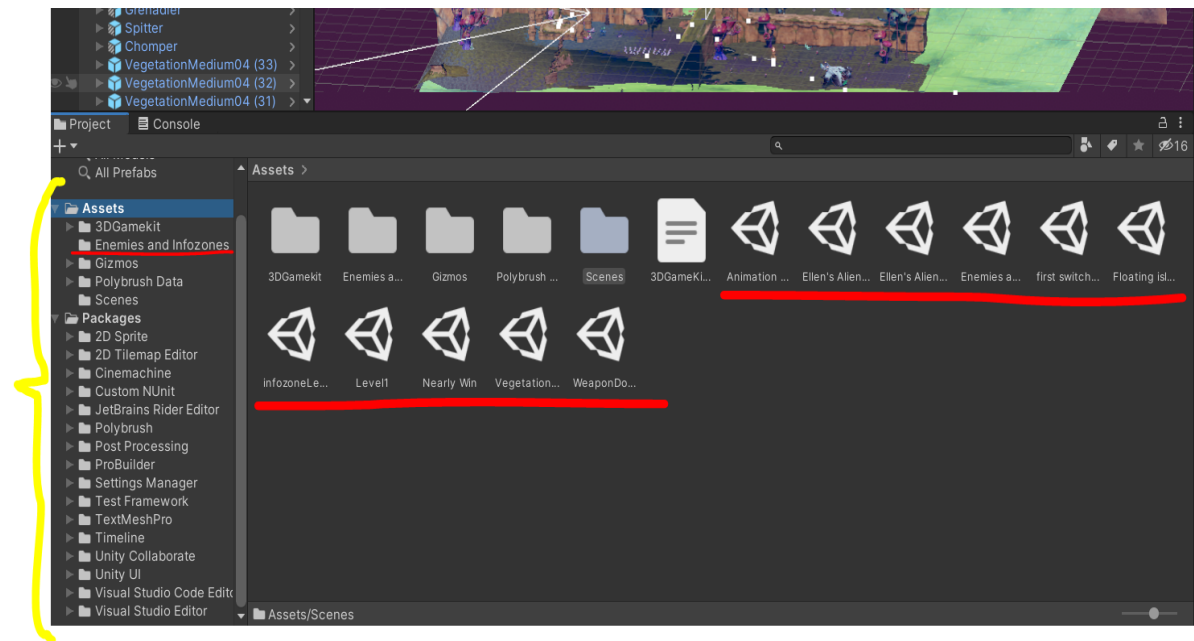
Out-of-Box features and Self-Coded parts:

Seeing the image below, in the project section from the screenshot, the yellow curly braces indicated files, containing Assets Store (containing 3DGamekit, Gizmos Polybrush Data, and Scenes but excluding Enemies and infozones) and files inside Packages are imported from the Unity Asset Store (Out-of-Box features).

As below, this is the original scene just after imported from the asset, it is almost blank.



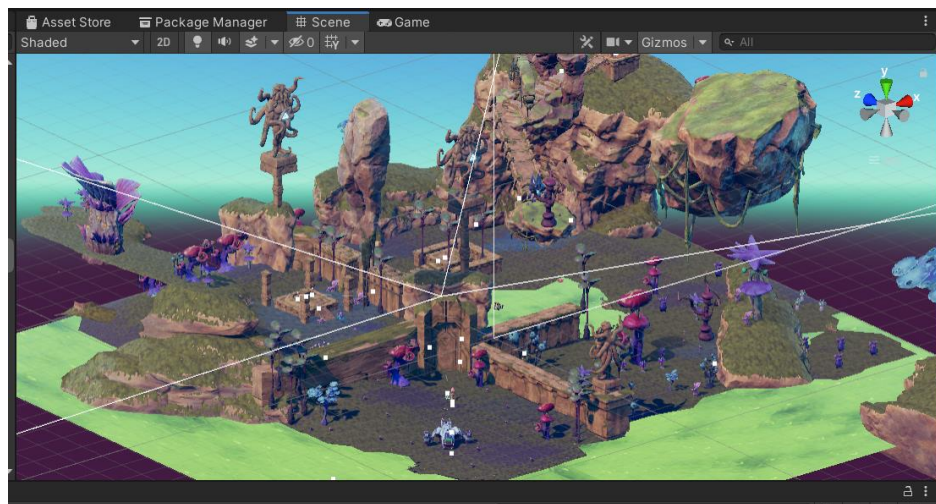
Therefore, the files underlined by the red lines, including Enemies and infozones on the left and those 11 scenes on the right, are developed by myself.



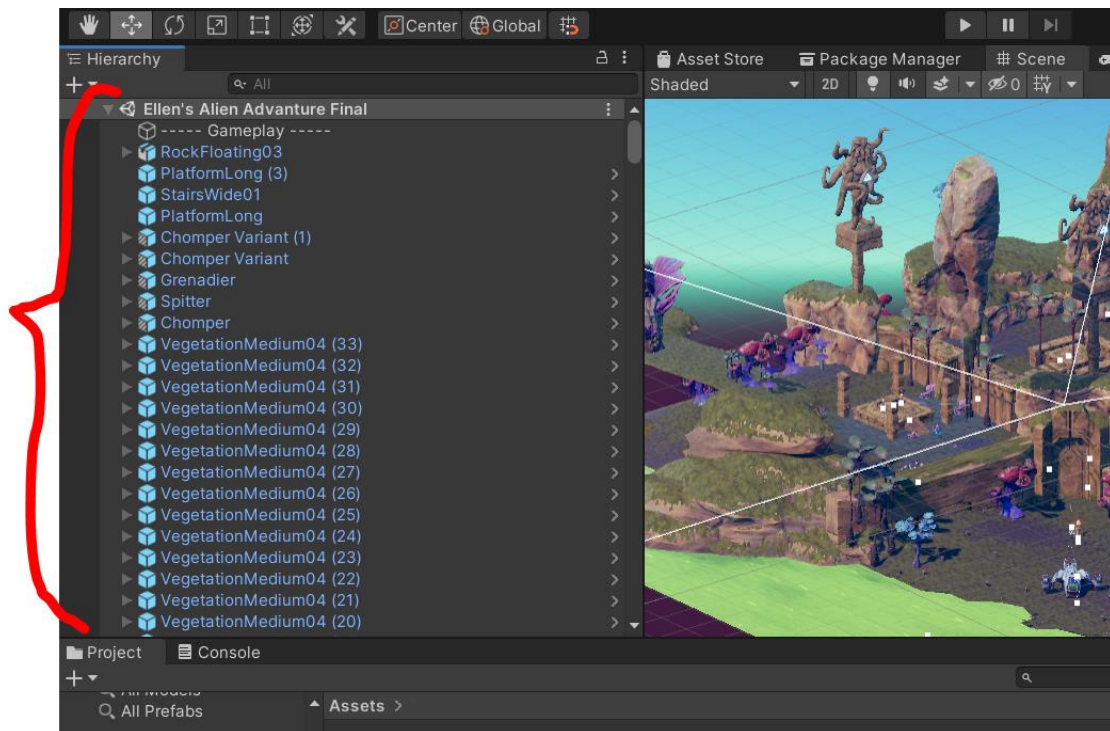
While among those 11 scenes all made by myself, I only select the scene named Ellen's Alien Adventure Final to demonstrate here.

3DGameKit_Third-PartyNotice.txt.me...	2020/7/21 12:54	META 文件	1 KB
Animation Game Project 11 - SceneC...	2020/9/13 16:50	Unity scene file	1,234 KB
Animation Game Project 11 - SceneC...	2020/9/13 16:50	META 文件	1 KB
Ellen's Alien Advanture Final.unity	2020/9/15 18:45	Unity scene file	2,754 KB
Ellen's Alien Advanture Final.unity.me...	2020/9/15 15:47	META 文件	1 KB
Ellen's Alien Advanture.unity	2020/9/15 14:57	Unity scene file	2,754 KB
Ellen's Alien Advanture.unity.meta	2020/9/15 14:57	META 文件	1 KB
Enemies and Infozones.meta	2020/9/14 1:29	META 文件	1 KB
Enemies and Infozones.unity	2020/9/14 2:38	Unity scene file	2,668 KB
Enemies and Infozones.unity.meta	2020/9/14 1:21	META 文件	1 KB
first switch after weapon.unity	2020/9/13 21:26	Unity scene file	1,323 KB
first switch after weapon.unity.meta	2020/9/13 19:13	META 文件	1 KB
Floating island and Faded.unity	2020/9/14 4:19	Unity scene file	2,764 KB
Floating island and Faded.unity.meta	2020/9/14 4:19	META 文件	1 KB
Gizmos.meta	2020/7/21 13:05	META 文件	1 KB
infozoneLeft.unity	2020/9/14 3:22	Unity scene file	2,706 KB
infozoneLeft.unity.meta	2020/9/14 2:38	META 文件	1 KB
Level1.unity	2020/9/13 16:49	Unity scene file	1,234 KB
Level1.unity.meta	2020/9/12 17:27	META 文件	1 KB
Nearly Win.unity	2020/9/15 1:06	Unity scene file	2,769 KB
Nearly Win.unity.meta	2020/9/14 4:28	META 文件	1 KB
Polybrush Data.meta	2020/7/21 13:04	META 文件	1 KB
Scenes.meta	2020/9/12 15:29	META 文件	1 KB
Vegetations and Enemies.unity	2020/9/14 1:20	Unity scene file	2,428 KB
Vegetations and Enemies.unity.meta	2020/9/13 21:27	META 文件	1 KB
WeaponDone.unity	2020/9/13 19:11	Unity scene file	1,242 KB
WeaponDone.unity.meta	2020/9/13 16:58	META 文件	1 KB

Here is the overall scene of Ellen's Alien Adventure:



My work is to develop all the characters, environment, game scenarios (win and lose), and interactive mechanisms (such as the triggerable door, triggerable moving platform, pickable weapons etc.) are developed by myself. Seeing the files under the hierarchy below:



Moreover, as I mentioned in the beginning, the Youtube video link of my Game demonstration is as below:

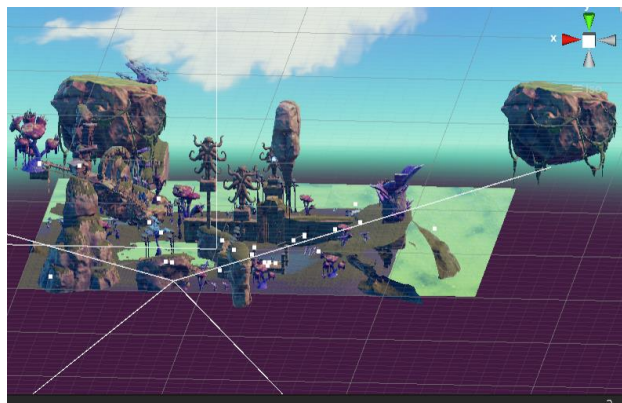
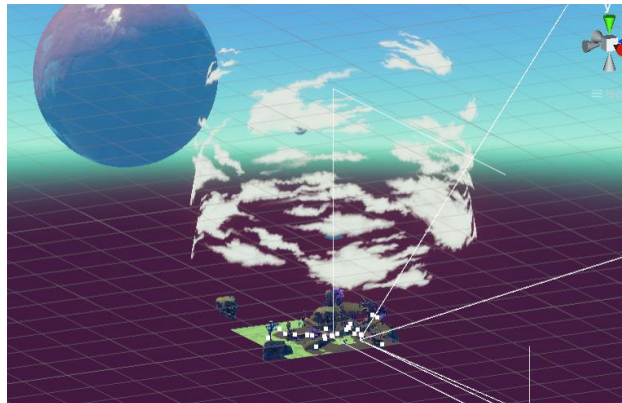
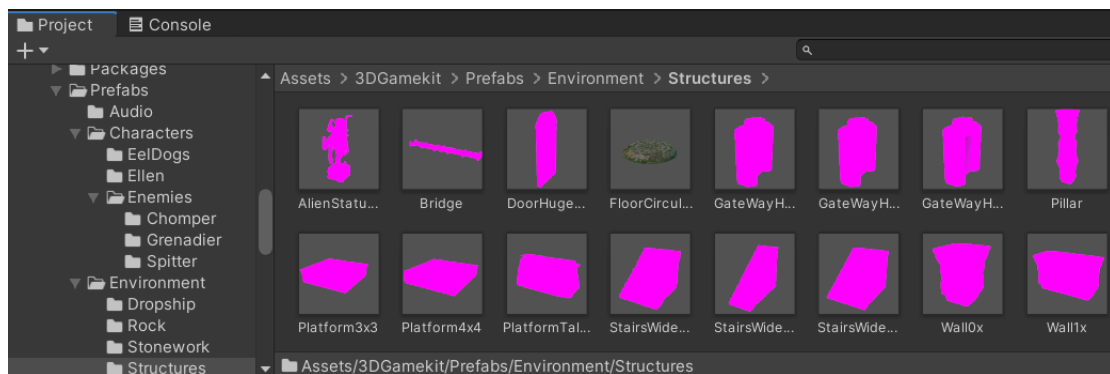
<https://youtu.be/wDm5igGL7eI>

The link to an online repository containing my code is available in the attached file.

Using Environment Prefabs

From Prefabs>Environments, I got resources of multiple level Geometry Objects, such as rocks, walls, cliffs, floating islands, vegetations, acid ocean and lake, and Structures like platform, gates , and alien statues. I placed them and constructed the entire scene.

We have built Level Geometry for you to be able to place where you wish. These can be found in **Prefabs > Environment**.



Game Boundaries

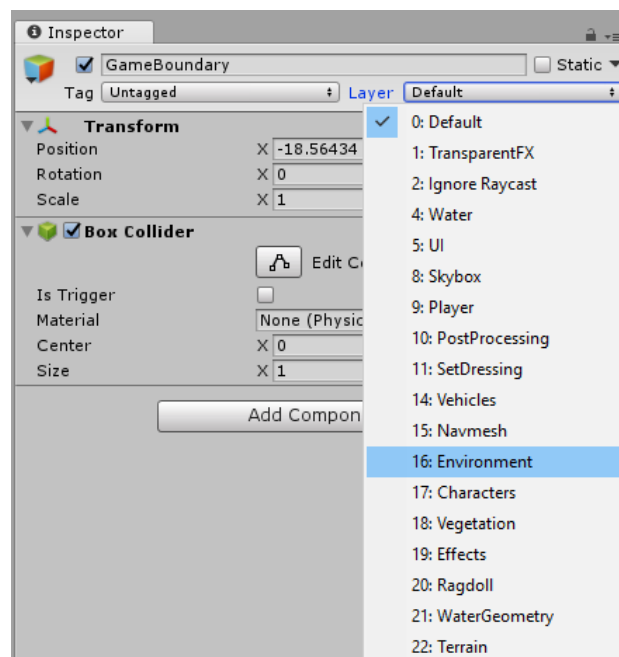
While building a level, I found that it is necessary to make boundaries in avoid that characters escape and get outside, therefore, I added invisible walls here to prevent players and enemy characters from being able to escape.

To implement it:

- At the top of the **Hierarchy** go to **Create > Create Empty**
- In the **Inspector** Rename this to **GameBoundary**
- In the **Inspector** click **Add Component**
- In the search box type in **Box Collider**
- Click on **Box Collider** to add it to the **GameBoundary** GameObject you just created

Then, locate the object in a suitable position of the scene

- Select **GameBoundary** in the Hierarchy
- Hover the mouse over the **Scene View**
- Press **F** on the Keyboard to **Frame Select** the Object.
- In the **Inspector** click on the **Layer** drop down menu
- Left Click on **Environment** to set it to that **Layer**.



Achievements based on the Requirements and Examination

Must have 3D objects and views:

The entire game is developed using Unity 3D, seen as the link to its Youtube Demo above, it can be proved.

Must run for 1 minute or longer:

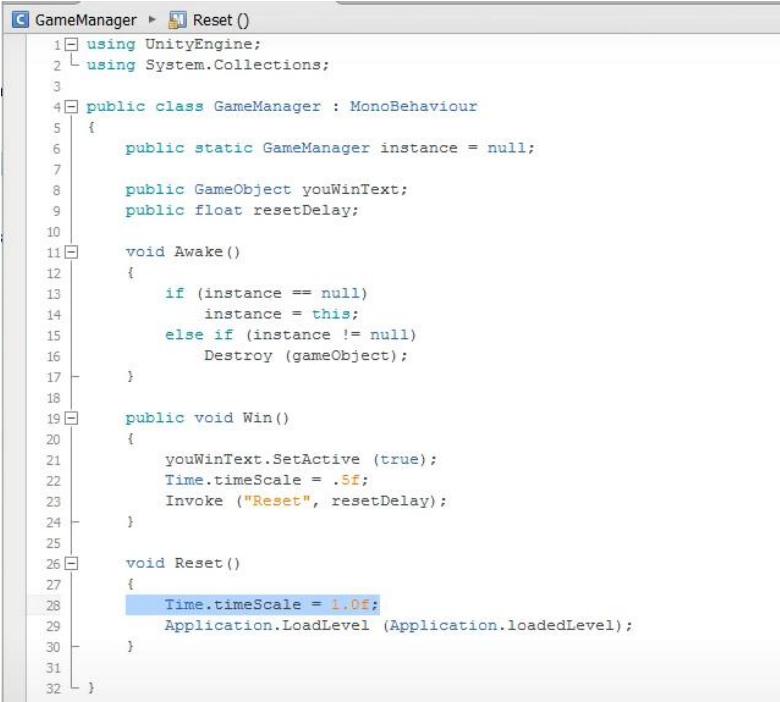
The demo is approximately 8 minutes, and the game can be run longer than 1 minute (5-12 minutes).

Must have scoring and winning/losing:

If the protagonist wins, the time speed reduces to half once it triggers the scenario and it shows "I win!!!!!!" and Game Over.

If the life hit value is zero due to killed by enemies or else if the protagonist walks into the acid ocean or lake, it triggers lose.

To implement **Win**, I created an empty object named GameManager and coded as below, to trigger I Win!!!!!! showing on the game, slow down the time, and reset the game. Then I had linked GameManager to the target 3D object on the game which triggers a win scenario.



```
1 using UnityEngine;
2 using System.Collections;
3
4 public class GameManager : MonoBehaviour
5 {
6     public static GameManager instance = null;
7
8     public GameObject youWinText;
9     public float resetDelay;
10
11     void Awake()
12     {
13         if (instance == null)
14             instance = this;
15         else if (instance != null)
16             Destroy (gameObject);
17     }
18
19     public void Win()
20     {
21         youWinText.SetActive (true);
22         Time.timeScale = .5f;
23         Invoke ("Reset", resetDelay);
24     }
25
26     void Reset()
27     {
28         Time.timeScale = 1.0f;
29         Application.LoadLevel (Application.loadedLevel);
30     }
31 }
32 }
```


Lose can be implemented by this similar concept. While the protagonist has zero life hit point value or walks inside the acid ocean or lake, the program runs similarly as above.

Scoring:

To show the health life point value, here is the code inside the script of the HealthUI which works as the interface showing in the HealthCanvas implementing the function.

```
IEnumerator Start()
{
    if (representedDamageable == null)
        yield break;

    yield return null;

    m_HealthIconAnimators = new Animator[representedDamageable.maxHitPoints];

    for (int i = 0; i < representedDamageable.maxHitPoints; i++)
    {
        GameObject healthIcon = Instantiate(healthIconPrefab);
        healthIcon.transform.SetParent(transform);
        RectTransform healthIconRect = healthIcon.transform as RectTransform;
        healthIconRect.anchoredPosition = Vector2.zero;
        healthIconRect.sizeDelta = Vector2.zero;
        healthIconRect.anchorMin += new Vector2(k_HeartIconAnchorWidth, 0f) * i;
        healthIconRect.anchorMax += new Vector2(k_HeartIconAnchorWidth, 0f) * i;
        m_HealthIconAnimators[i] = healthIcon.GetComponent<Animator>();

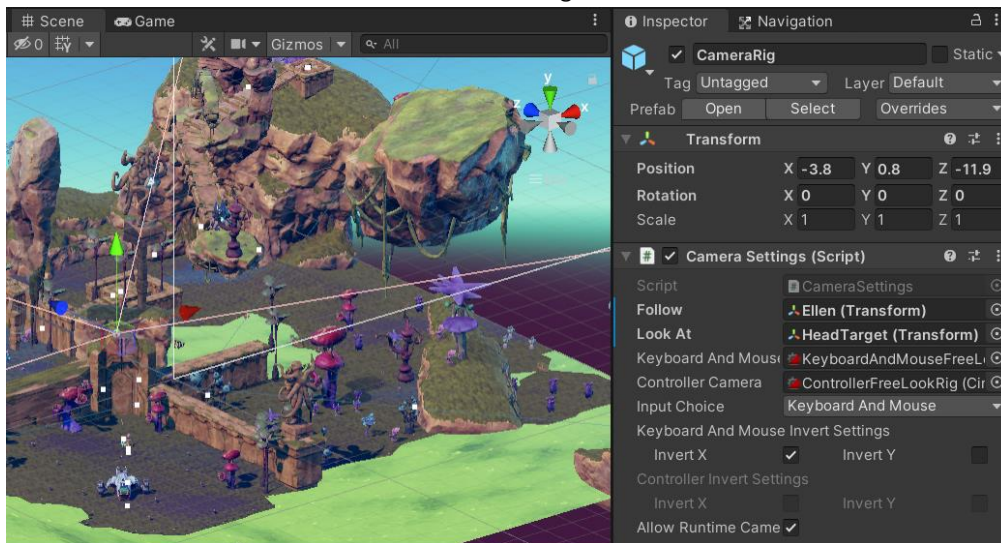
        if (representedDamageable.currentHitPoints < i + 1)
        {
            m_HealthIconAnimators[i].Play(m_HashInactiveState);
            m_HealthIconAnimators[i].SetBool(m_HashActivePara, false);
        }
    }
}
```

Must have user interaction and camera-control

In the game, the camera control is implemented, once the user moves the mouse, the camera perspective moves respectively.

In my hierarchy, objects Camera Rig and Camera Brain implementing the functions.

CameraRig



Under the C# script of Camera Settings inside the CameraRig object, there are 2 functions whose code is very important.

Functions to implement keyboard camera control

```
void Reset()
{
    Transform keyboardAndMouseCameraTransform = transform.Find("KeyboardAndMouseFreeLookRig");
    if (keyboardAndMouseCameraTransform != null)
        keyboardAndMouseCamera = keyboardAndMouseCameraTransform.GetComponent<CinemachineFreeLook>();

    Transform controllerCameraTransform = transform.Find("ControllerFreeLookRig");
    if (controllerCameraTransform != null)
        controllerCamera = controllerCameraTransform.GetComponent<CinemachineFreeLook>();

    PlayerController playerController = FindObjectOfType<PlayerController>();
    if (playerController != null && playerController.name == "Ellen")
    {
        follow = playerController.transform;

        lookAt = follow.Find("HeadTarget");

        if (playerController.cameraSettings == null)
            playerController.cameraSettings = this;
    }
}
```

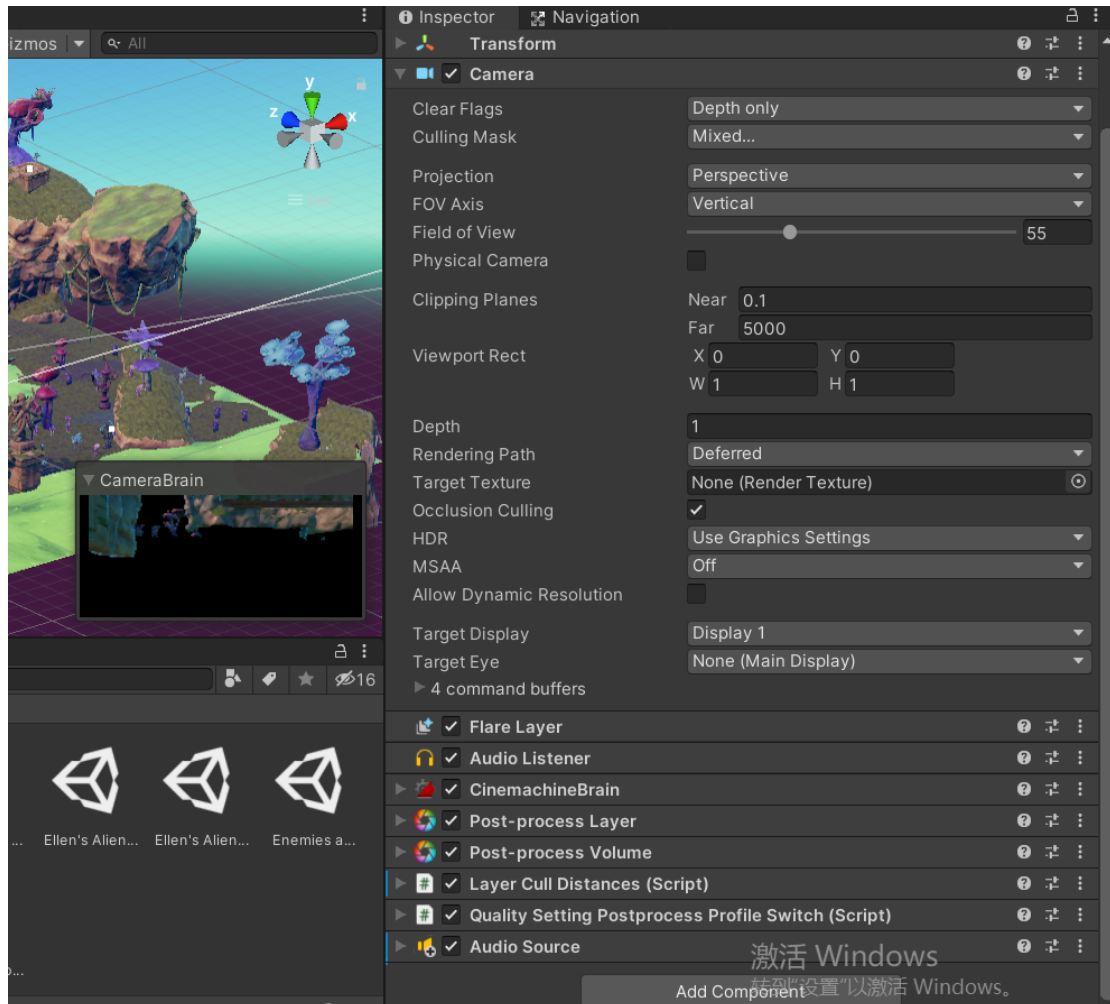
Functions to update camera settings

```
void UpdateCameraSettings()
{
    keyboardAndMouseCamera.Follow = follow;
    keyboardAndMouseCamera.LookAt = lookAt;
    keyboardAndMouseCamera.m_XAxis.m_InvertInput = keyboardAndMouseInvertSettings.invertX;
    keyboardAndMouseCamera.m_YAxis.m_InvertInput = keyboardAndMouseInvertSettings.invertY;

    controllerCamera.m_XAxis.m_InvertInput = controllerInvertSettings.invertX;
    controllerCamera.m_YAxis.m_InvertInput = controllerInvertSettings.invertY;
    controllerCamera.Follow = follow;
    controllerCamera.LookAt = lookAt;

    keyboardAndMouseCamera.Priority = inputChoice == InputChoice.KeyboardAndMouse ? 1 : 0;
    controllerCamera.Priority = inputChoice == InputChoice.Controller ? 1 : 0;
}
```

CameraBrain



Must have at least 5 interactive elements

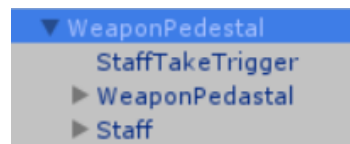
Weapon Pick up

Firstly, because the protagonist can attack with her weapon by default, this setting should be deactivated.

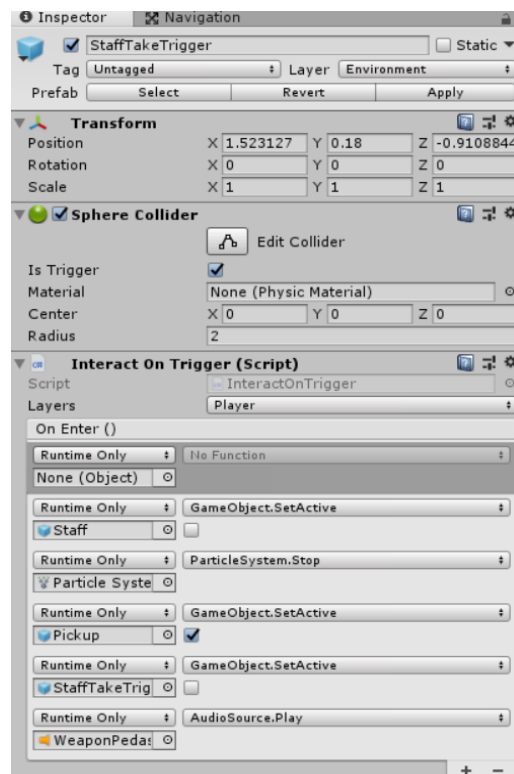


Secondly, drag and place the WeaponPedestal from the asset to a suitable position in the scene.

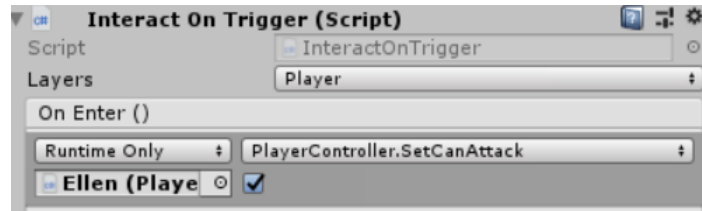
From its hierarchy view, select StaffTakeTrigger



In the **Inspector** there is a **Sphere Collider** component and an **Interact on Trigger** script component with a few predefined values on the properties of the script. This is to save setup time on the pedestal and allow the developer to have ready-made logic.



Drag **Ellen** from the **Hierarchy** view into the top property field. When it accepts the **GameObject**, select a function from the drop-down titled **PlayerController.SetCanAttack** this will default to True (enabled)



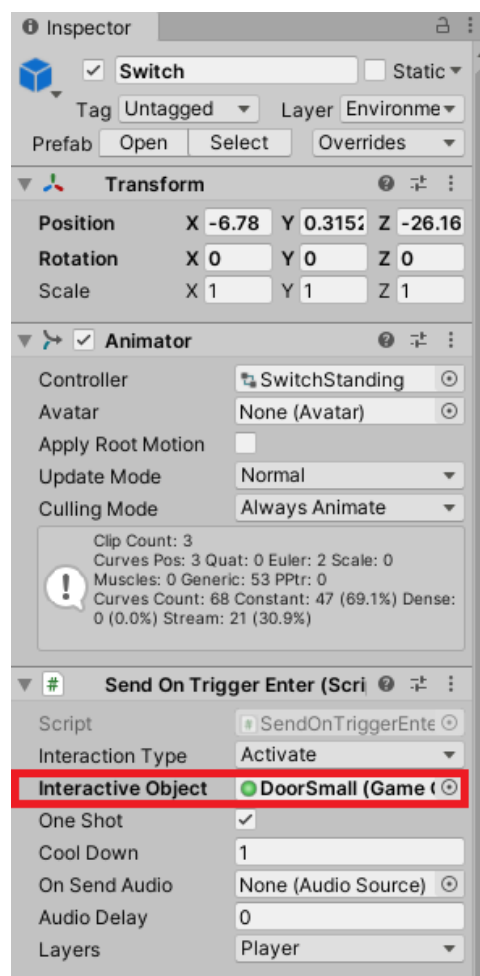
Now, the protagonist Ellen is not able to attack until she picks up the weapon.



Switch triggerable Gate



I have linked the switch to the Gate, now once the protagonist walks close to the switch, the linked gate will be open automatically.



Using Counter and Switches

Here, I want to open the door when all 3 switches are activated.

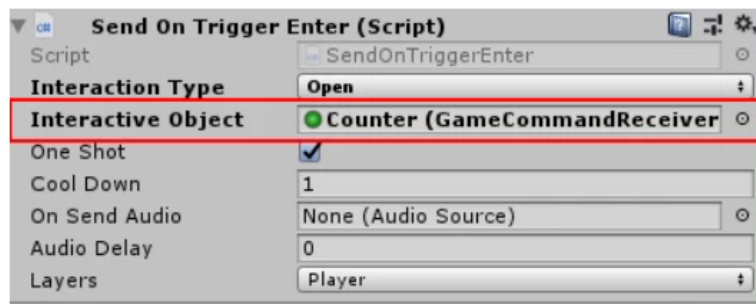


To implement this, I completed the following steps:

Drag the counter prefab and place it in a suitable position of the scene.

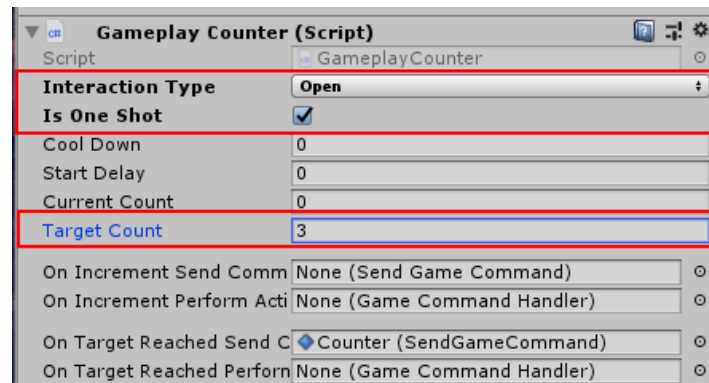
Select switch from the hierarchy

Click and drag the Counter into the Interactive Object slot in the Send on Trigger Enter script.



Repeat the same processes to another two switches.

- **After that,** Select the **Counter** in the **Hierarchy**
- In the **Inspector**, change the following settings on the **Gameplay Counter** script
- **Interaction** Type: Open
- **Is One Shot:** True (tick the box)
- **Target Count:** 3



This will receive a command of type Open, only count once, and set the target amount to fulfill the counter at 3.

Furthermore

- With the **Counter** selected, input the following settings on the **Send Game Command** script
- **Interaction Type**: Open
- Drag **DoorSmall** from the **Hierarchy** to the **Interactive Object** slot
- **One Shot**: Enabled (tick the box)



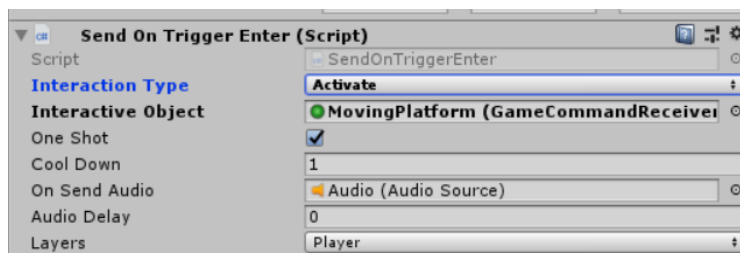
Now, the door will be open as soon as all 3 switches are turned on.

Pressure Pad



It will activate the **MovingPlatform** (This will be explained later) Prefab using Game Commands.

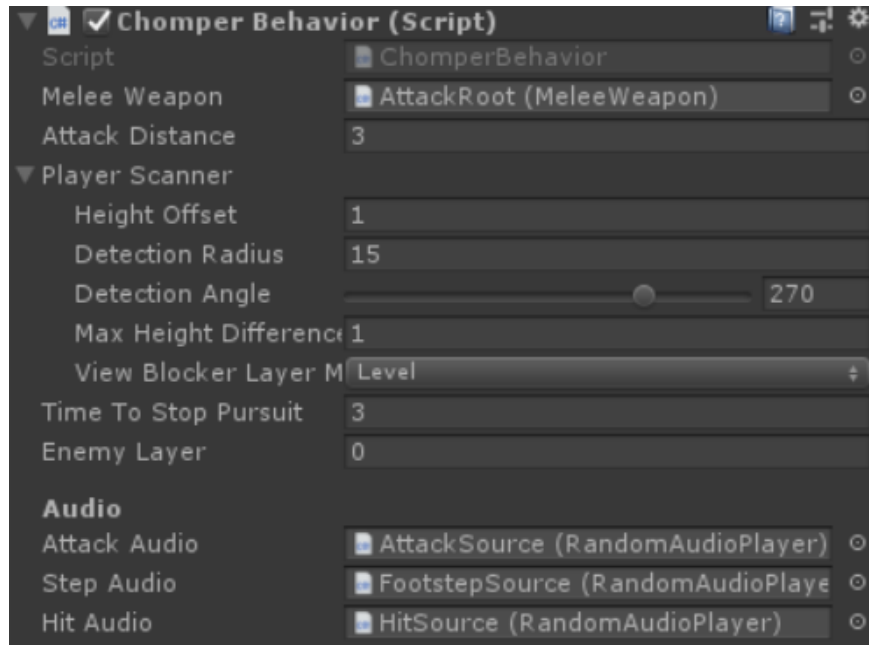
The below image briefly shows how the Pressure Pat is linked to the moving platform.



Adding Enemies

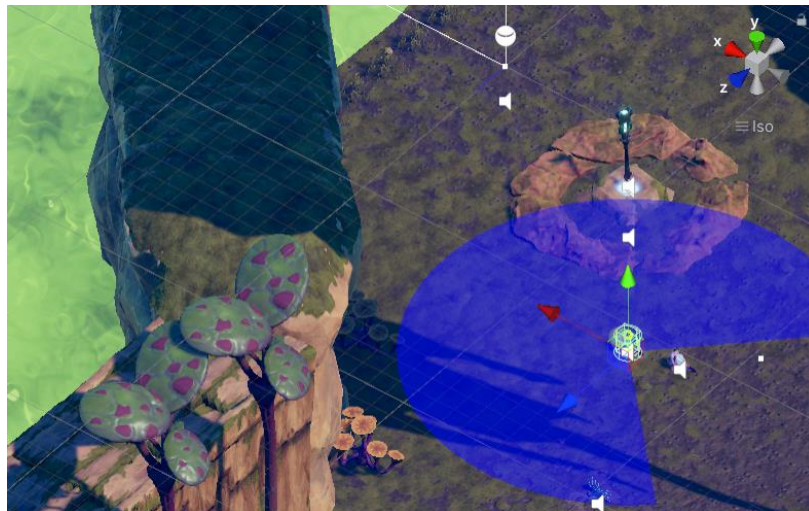
Player Scanner

All enemies have a **Player Scanner** property on their respective behaviour that is used to set-up how they can spot the Player.



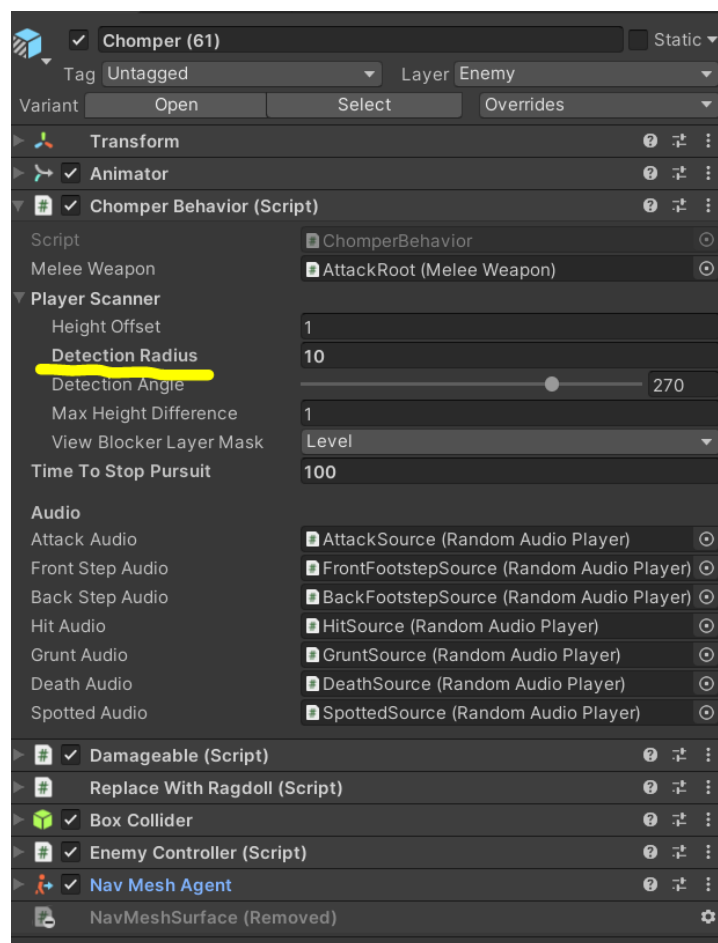
Detecting Radius

Some of those settings can be visualized on the scene view when an Enemy is selected.



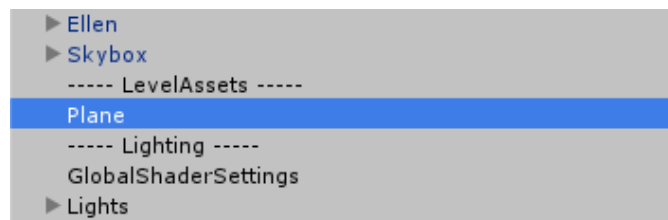
The blue circle represents the Player Scanner detection radius and angle. The player will only be spotted if they walk within that circle.

In the **Inspector**, locate the **Chomper Behaviour** script.



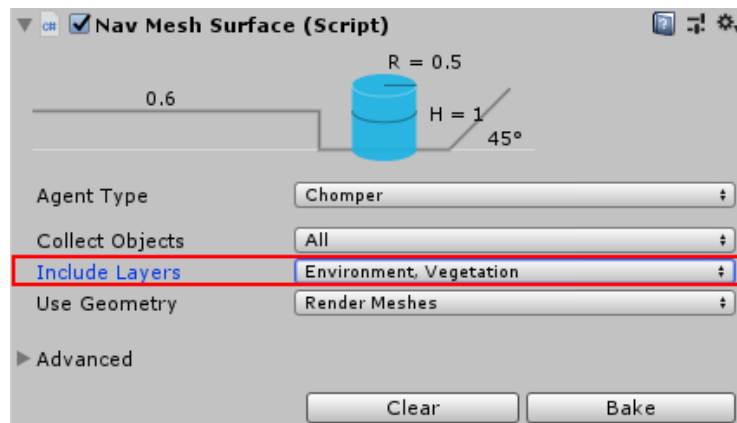
In **Detection Radius**, the value is set to 4. This can also be achieved by quickly scrolling through values, hovering over the word **Detection Radius**.

After this, I need to tell Chomper's AI what surface is walkable. I have used Unity's NavMesh system to do this. Chomper's AI is handled by the NavMesh Agent Component. This gives Chomper the ability to walk on surfaces that we set, avoid obstacles and other Chompers, Spitters and Grenadiers while navigating the world.

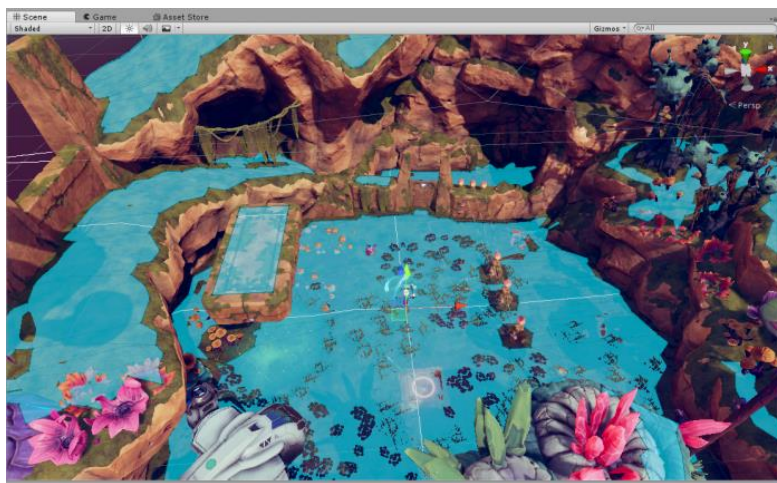


To implement this, we need to go to Plane from the hierarchy, shown as above, then add a new component named **NavMeshSurface**

Then, we need to set the Agent Type to Chomper, and set the include layers as Environment, Vegetation. And Click Bake Button.



This will Bake a NavMesh in the scene and make most surfaces walkable for the enemies. As we can see below, our enemies can walk in the Scene View as it's shown by the light blue highlight.



Similar concepts can be applied for enemy Chomper, Spitter, and Grenadier, showing below from left to right respectively.

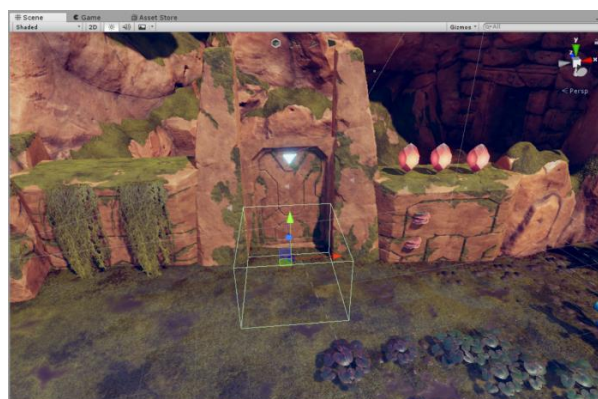


As you can see below, enemies are chasing and trying to attack the protagonist since she went inside the detection range of the enemy.

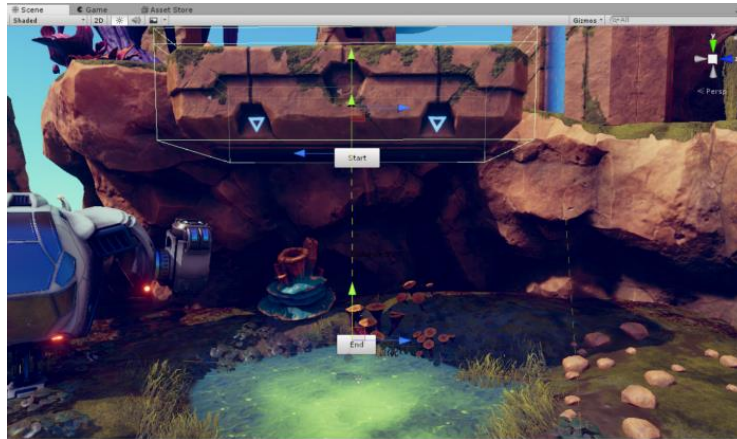


Adding Checkpoints

Checkpoints ensure that when the protagonist dies, she will be respawned at the last checkpoint she touched.



Moving Platforms



The underlying concept's details of it will be explained in the below section.

Motion Editing-blending, transplanting, etc.

There are several objects on the scene achieves this concept. Here, I take the **Moving Platform** as an example.

To find it from the asset, we need to follow the steps below:

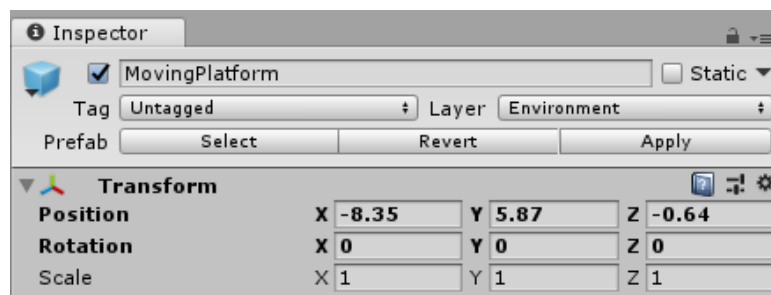
Navigate to the **Project Window**

Go to **Assets > 3D GameKit > Prefabs > Interactables**

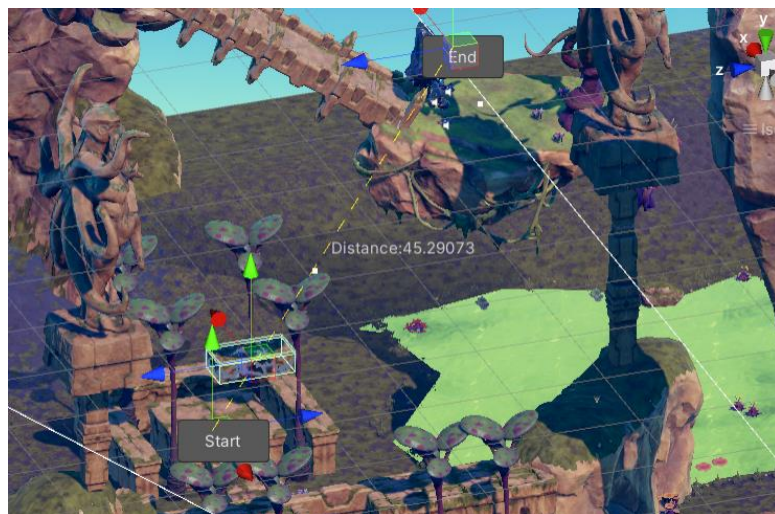
Left Click and drag the **MovingPlatform** Prefab into the **Scene View** OR the **Hierarchy**

Note: if you move this into the **Hierarchy** don't forget to use **Frame Select** to find it (Keyboard Shortcut - F)

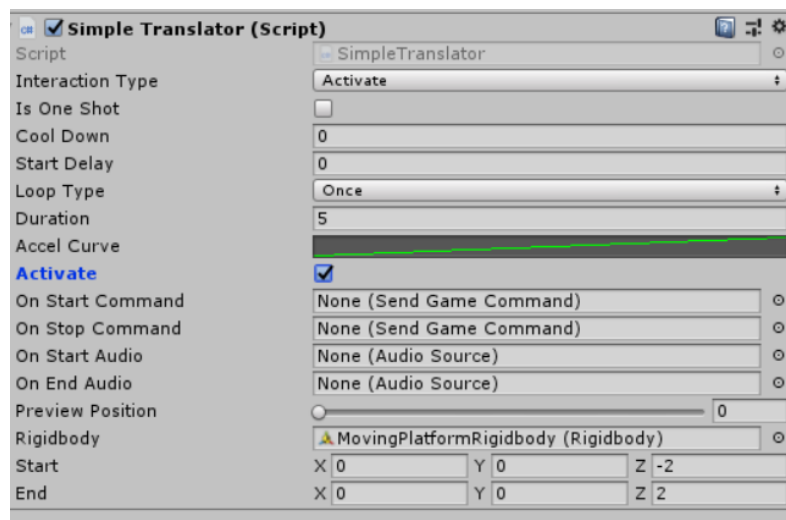
You can also position, rotate and scale the object through the **Inspector** in the Transform Component



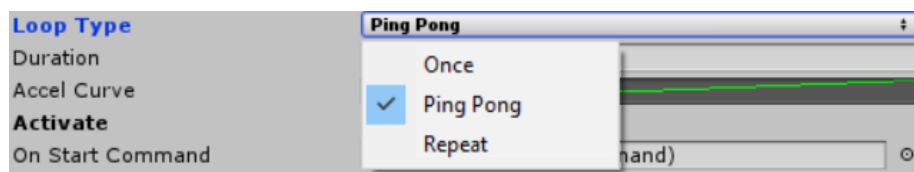
Position the whole object in a suitable position to start from



Click Activate



Select Ping Pong from the loop type. Changing this will repeat the sequence from Start to End and back again.

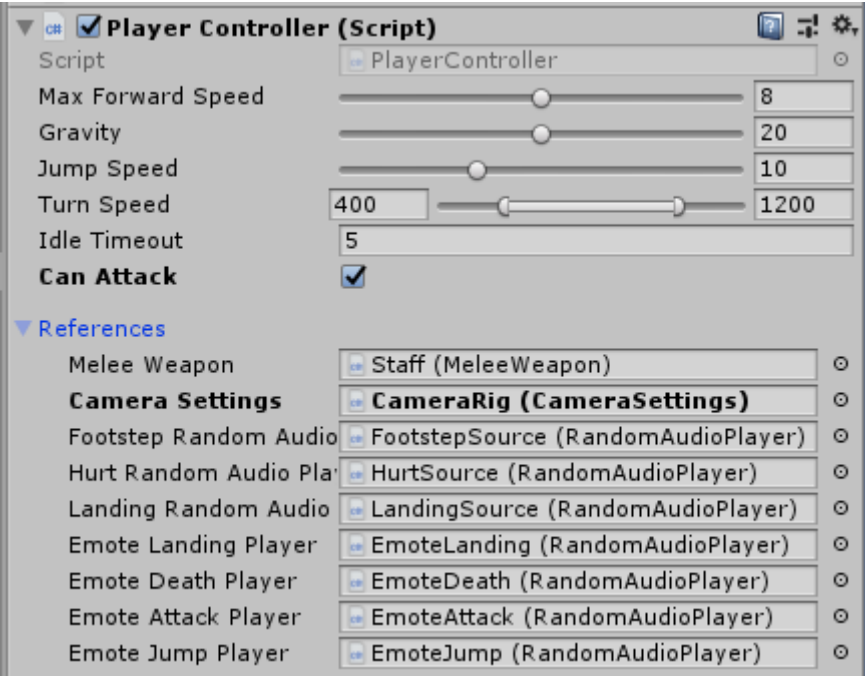


Now, the moving platform is set successfully.



Must have one reasonably realistically moving articulated animated character, as part of the game and visible during gameplay:

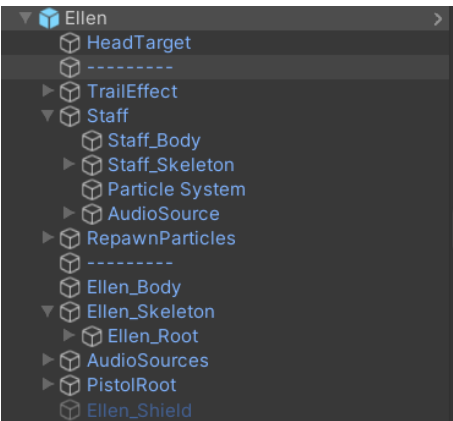
Player Controller:



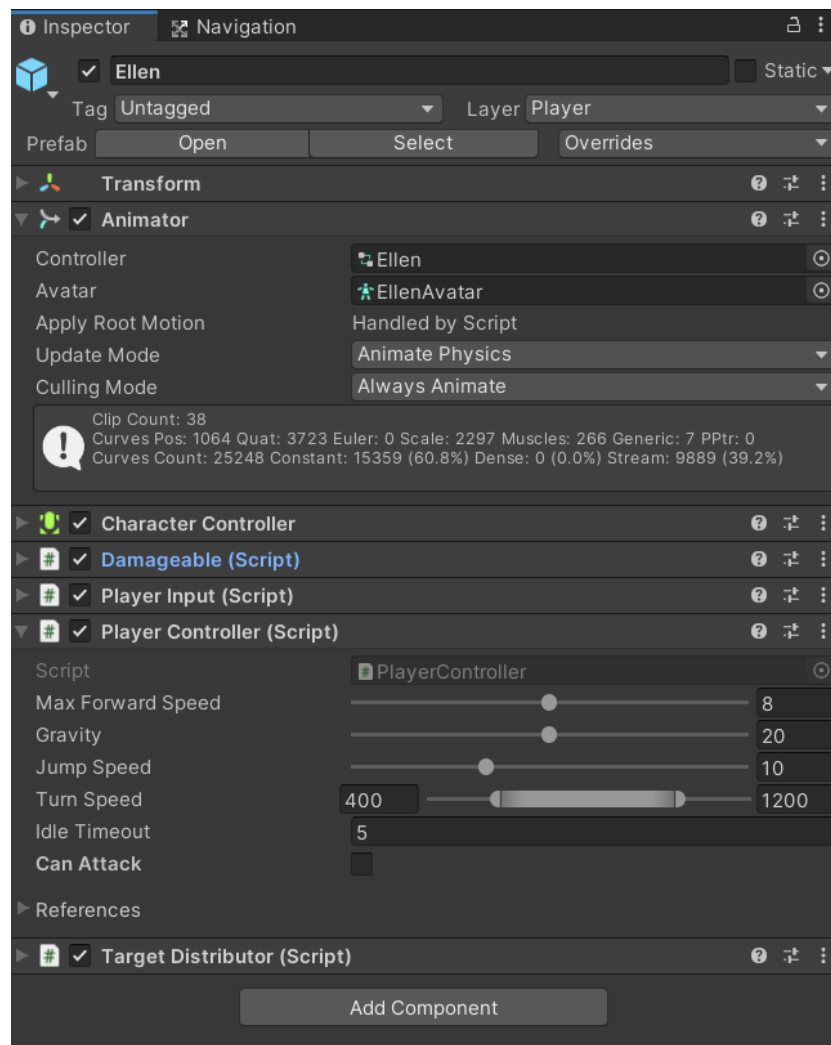
Standard Movement Controls:

Motion	PC	Controller
Move	W, A, S, D	Left stick
Jump	Space	A
Melee	Left Click on Mouse	X
Camera	Mouse	Right stick

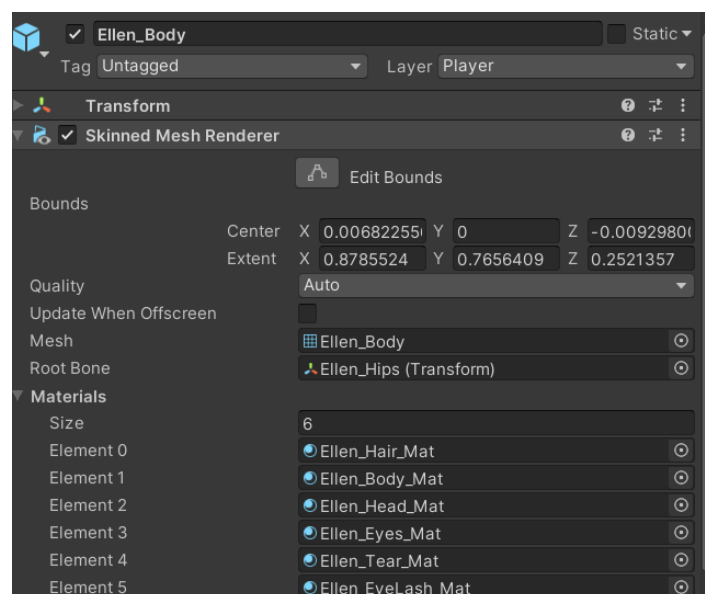
Objects of Ellen from Hierarchy



Inspector of Ellen



Components of different parts of the body of Ellen_Body



The **Player Controller** script holds all the information for how Ellen behaves in the game. The exposed settings here mostly control movement but you can tweak any of the variables within the script itself by opening it.

If you change any settings, the changes are only applied to the Prefab instance in that Scene. If you would like the changes to happen across all levels in your game, click **Apply** at the top of the instance for the change to be applied to the Ellen Prefab.

- **Max Forward Speed:** *Default Setting: 8* How fast Ellen can run.
- **Gravity:** *Default Setting: 20* How fast Ellen falls when in the air.
- **Jump Speed:** *Default Setting: 10* How fast Ellen takes off when jumping.
- **Turn Speed:** *Default Setting: Between 400 and 1200* How fast Ellen turns. This varies depending on how fast she is moving. When stationary, the maximum is used because you want Ellen to turn faster. When she's not running at **Max Forward Speed**, the minimum is used because it's harder to make sharp turns at speed.
- **Can Attack:** *Default Setting: false* Whether or not Ellen can attack with her staff. This can be set externally.
- **References:** These are all the references the script requires to function. They are entirely found on the Ellen prefab and by default you should not need to adjust them.
- **Melee Weapon:** Damages enemies when Ellen swings her staff.
- **Camera Settings:** Gets the rotation of the current camera so that Ellen faces the correct direction. Note: This is the only reference that is not part of the Ellen prefab. It should automatically be set to the Camera Settings script of the CameraRig GameObject when the Prefab is instantiated.

The following features show the personality of the character from her special gesture motion or emotion, which implements the requirement of **Stylized motion, Gesture or personality modelling for the character, Interesting character behaviors/AI and Particularly imaginative gameplay.**

- **Footstep Random Audio Player:** Plays a random sound when Ellen takes a step.
- **Hurt Random Audio Player:** Plays a random sound when Ellen gets hurt.
- **Landing Random Audio Player:** Plays a random sound when Ellen lands.
- **Emote Landing Player:** Plays a random vocal sound when Ellen lands.
- **Emote Death Player:** Plays a random vocal sound when Ellen dies.
- **Emote Attack Player:** Plays a random vocal sound when Ellen attacks.
- **Emote Jump Player:** Plays a random vocal sound when Ellen jumps.

- **Idle Timeout:** *Default Setting: 5* How many seconds before Ellen starts considering random Idle poses.

(Notice: The above Part Research of Player Control is particularly referenced from 3D Game Kit Unity Tutorial whose link can be found in the reference part at the end of this report.)

Applying these underlying mechanism and principles above, the following features have been implemented successfully according to the exam requirement:

Multiple joints and children

It has a correct hierarchy and can move correctly according to its structure

Human being Character

Keyframed facial animation

Kinematics on the arms or feet for foot

Besides, the additional features requirements below are also implemented successfully:

Motion State Machines

Motion Editing-blending, transplanting, etc.

Stylized motion (Also mentioned before)

Gesture or personality modelling for the character (Also mentioned before)

Interesting character behaviors/AI (Also mentioned before)

Particularly imaginative gameplay (Also mentioned before)

Cloth Simulation

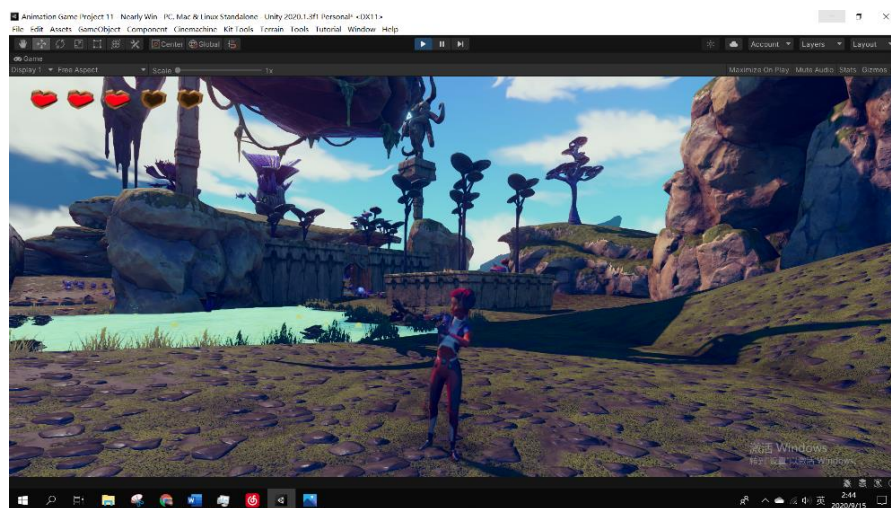
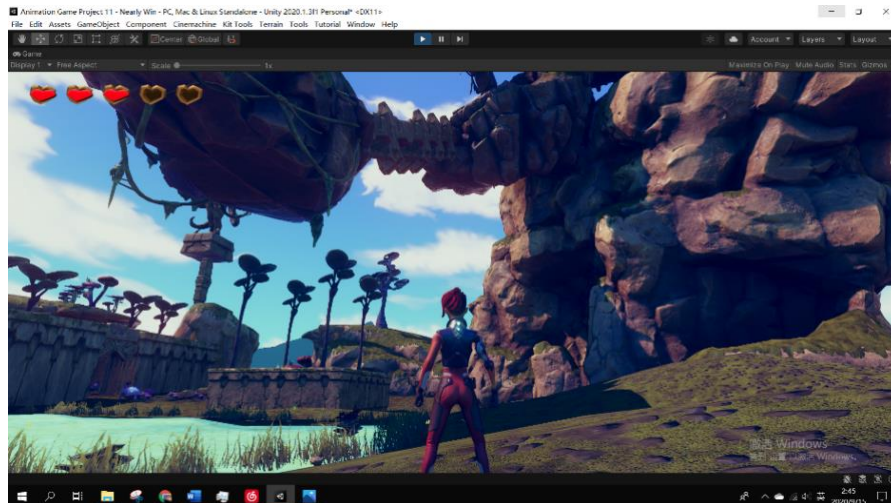
Physically-based animation

In the next pages, the screenshots of the particular behaviors of the protagonist are displayed. It tells how the character features based on the requirements above are implemented.



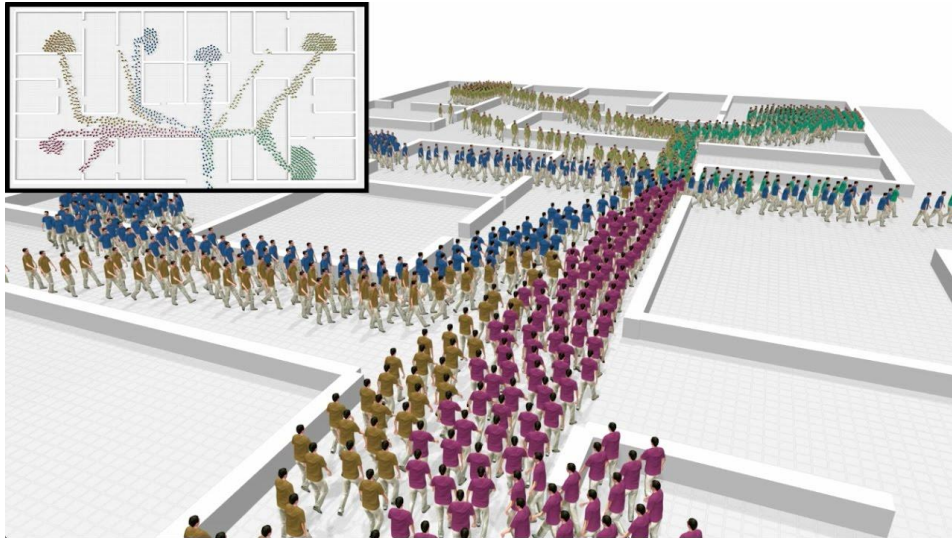






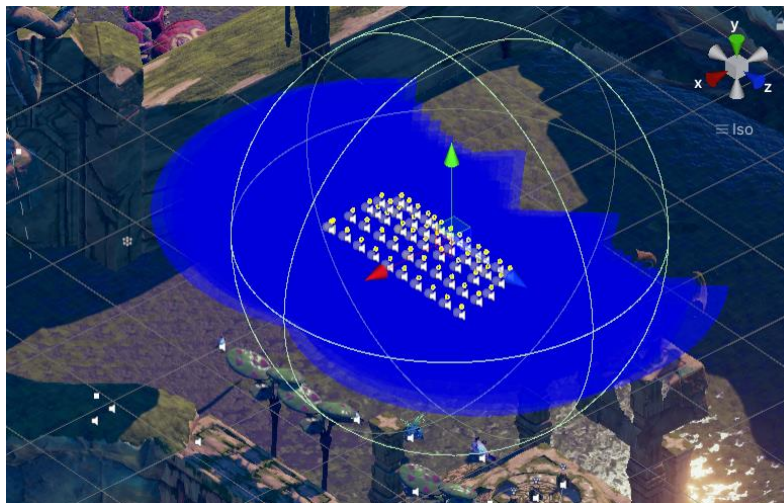
Crowd Simulation

Crowd simulation is the process of simulating the movement of a large number of entities or characters. It is commonly used to create virtual scenes for visual media like films and video games, and is also used in crisis training, architecture and urban planning, and evacuation simulation.



(Referenced from Thalmann, Daniel (2016). "Crowd Simulation". *Encyclopedia of Computer Graphics and Games*. pp. 1–8. doi:10.1007/978-3-319-08234-9_69-1. ISBN 978-3-319-08234-9. Also mentioned in the referenced section)

In our game project, I have added 60 enemies, adjusted their detecting radius (each row or column of enemies are different) and configured the position.



Therefore, once the protagonist reaches within the range, all the massive of the enemies from the crowd will start to chase and attempt to attack the protagonist. In this situation, those enemies simulate a crowd. If we control the moving direction of the protagonist, it affects the moving condition of the enemy crowd as well. As you seen below screenshots.



Referenced resources:

Math for 3D Graphics:
Points and Vectors
Geometric Transformations
(from TCD blackboard)

MotionCapture_Lecture2-Miss Dr. Racheal

<https://tcd.cloud.panopto.eu/Panopto/Pages/Sessions/List.aspx#folderID=%226b4eb516-5364-4735-b880-ab88008cbf49%22>

The Explorer: 3D Game Kit

<https://learn.unity.com/project/3d-game-kit?uv=2018.1>

<https://learn.unity.com/tutorial/quick-start?uv=2018.1&projectId=5c514897edbc2a001fd5bdd0>

<https://learn.unity.com/tutorial/3d-game-kit-walkthrough?uv=2018.1&projectId=5c514897edbc2a001fd5bdd0>

<https://learn.unity.com/tutorial/3d-game-kit-reference-guide?uv=2018.1&projectId=5c514897edbc2a001fd5bdd0>

Unity C# Tutorials

<https://www.youtube.com/watch?v=1yVxdEZvQWo&t=4s>

FPS: Microgame

<https://learn.unity.com/project/fps-template>

Creating Basic Editor Tools

<https://learn.unity.com/tutorial/creating-basic-editor-tools#5cf6c8f2edbc2a160a8a0951>

Researching Human Locomotion

<https://www.youtube.com/watch?v=3K0gmSgOJiw#t=98>

Thalmann, Daniel (2016). "Crowd Simulation". *Encyclopedia of Computer Graphics and Games*. pp. 1–8. doi:[10.1007/978-3-319-08234-9_69-1](https://doi.org/10.1007/978-3-319-08234-9_69-1). ISBN [978-3-319-08234-9](https://www.isbn-international.org/product/978-3-319-08234-9).