Ministry of Higher Education and Scientific Research University of Manouba
Higher Institute of Multimedia Arts
2017-2018

VR JETMAN GO

GET IT ON Google Play

NOURI MOHAMED

Table of contents

Introduction :

1. Project Framework

As gamers we have always enjoyed playing video games since an early age but also as gamers we have always been wondering how all these amazing games were made. I have noticed that game development is one of the few creative careers that rely on science and math so in order to make a game, being scientific and artistic at the same time is the key.

This project is an opportunity to expand my knowledge

2. Project Purpose

The main purpose of our project is to make and conceive a 3D video game with VR Technology for people who want to have fun, optimized in terms of graphic and coding, this game aims to let the player explore a very beautiful world Using VR Technology

3. The Game Target Audience

This game is made for kids of all ages!
The game contients :
Playful music ,colorful environmentn ,simlpe and freindlly ui

4. System,Hardware Requirements

The game Platform:
Android devices: 5.0 or higher
Iphone Users: 5s or higher

vr headset type :
Google Cardboard,Google Daydream,Samsung vr gear

Introduction To Virtual Reality :

1. what is virtual reality

Virtual reality (VR) is a computer technology that uses virtual reality headsets or multi-projected environments, sometimes in combination with physical environments or props, to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual or imaginary environment.

A person using virtual reality equipment is able to "look around" the artificial world, and with high quality VR move around in it and interact with virtual
features or items. The effect is commonly created by VR headsets consisting of a head-mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens.

VR systems that include transmission of vibrations and other sensations to the user through a game controller or other devices are known as haptic systems.
This tactile information is generally known as force feedback in medical, video gaming and military training applications. Virtual reality also refers to remote
communication environments which provide a virtual presence of users with through telepresence and telexistence or the use of a virtual artifact (VA).

The immersive environment can be similar to the real world in order to create a lifelike experience grounded in reality or sci-fi. Augmented reality systems may also be considered a form of VR that layers virtual information over a live camera feed into a headset, or through a smartphone or tablet device.

Figure 2 Researchers with the European Space Agency i

2 Etymology And Terminology :

In 1938, Antonin Artaud described the illusory nature of characters and objects in the theatre as "la réalité virtuelle" in a collection of essays, Le Théâtre et son double. The English translation of this book, published in 1958 as The Theater and its Double,[1] is the earliest published use of the term "virtual reality".
The term "artificial reality", coined by Myron Krueger, has been in use since the 1970s. The term "virtual reality" was used in The Judas Mandala, a 1982 science fiction novel by Damien Broderick.
Virtual Reality Industry research report is a meticulous investigation of current scenario of the market, which covers several market dynamics.
The Global market research report is a resource, which provides current as well as upcoming technical and financial details of the industry. "Virtual" has had the meaning "being something in essence or effect, though not actually or in fact" since the mid-1400s, "...probably via sense of "capable of producing a certain effect" (early 1400s)".[2] The term "virtual" has been used in the computer sense of "not physically existing but made to appear by software" since 1959.[2]

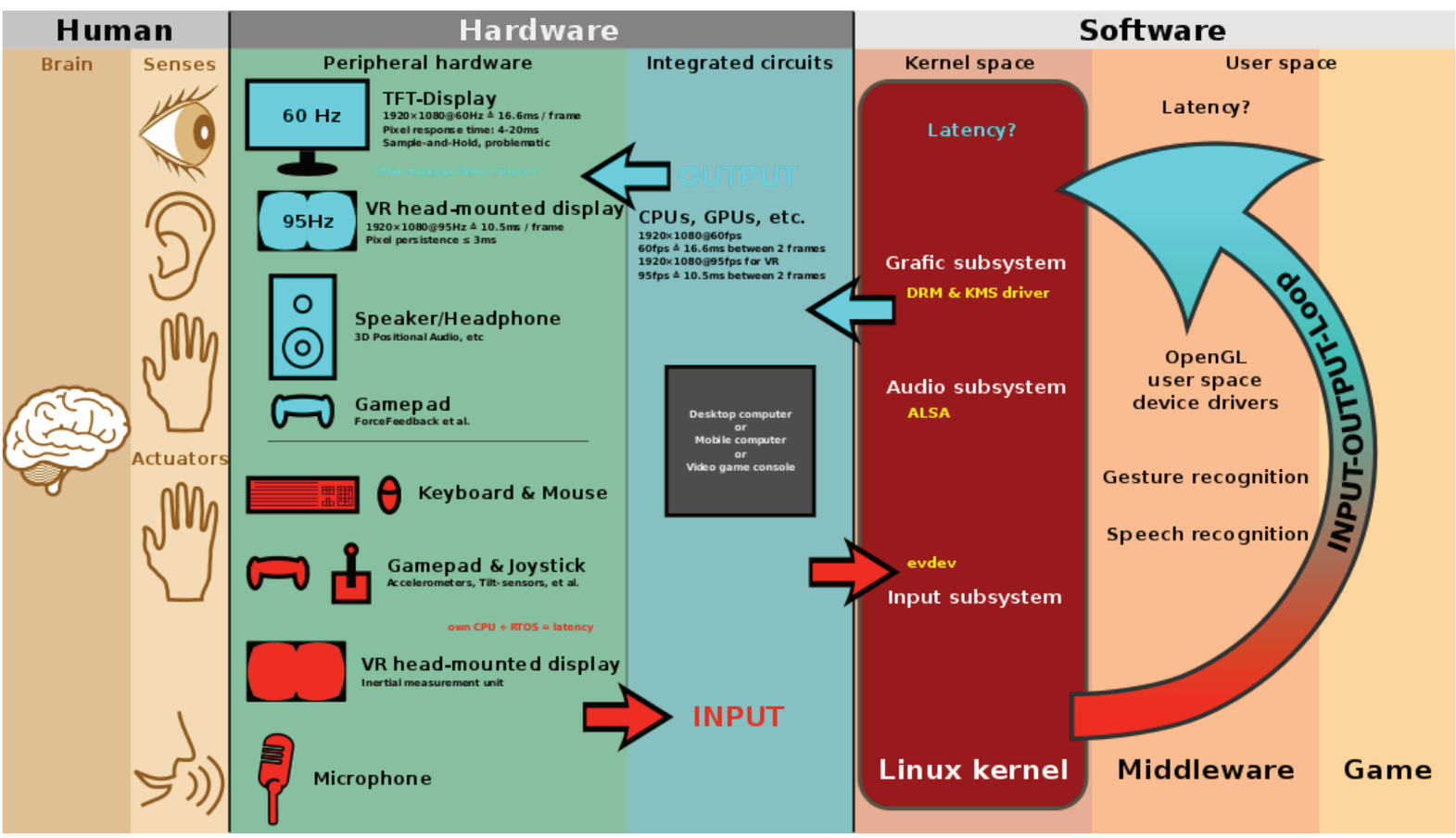


Figure 3: Paramount for the sensation of immersion into virtual reality

3 Technology

The Virtual Reality Modelling Language (VRML), first introduced in 1994, was intended for the development of "virtual worlds" without dependency on headsets.[5] The Web3D consortium was subsequently founded in 1997 for the development of industry standards for web-based 3D graphics. The consortium subsequently developed X3D from the VRML framework as an archival, open-source standard for web-based distribution of VR content.[6]

All modern VR displays are based on technology developed for smartphones including: gyroscopes and motion sensors for tracking head, hand, and body positions; small HD screens for stereoscopic displays; and small, lightweight and fast processors. These components led to relative affordability for independent VR developers, and lead to the 2012 Oculus Rift kickstarter offering the first independently developed VR headset.[7]

4 History

Before the 1950s

The first references to the more modern concept of virtual reality came from science fiction. Stanley G. Weinbaum's 1935 short story "Pygmalion's Spectacles"[13] describes a goggle-based virtual reality system with holographic recording of fictional experiences, including smell and touch.

1950–1970

Morton Heilig wrote in the 1950s of an "Experience Theatre" that could encompass all the senses in an effective manner, thus drawing the viewer into the onscreen activity. He built a prototype of his vision dubbed the Sensorama in 1962, along with five short films to be displayed in it while engaging multiple senses (sight, sound, smell, and touch).
Predating digital computing, the Sensorama was a mechanical device. Heilig also developed what he referred to as the "Telesphere Mask" (patented in 1960).

1970–2000

Also notable among the earlier hypermedia and virtual reality systems was the Aspen Movie Map, which was created at MIT in 1978. The program was a crude virtual simulation of Aspen, Colorado in which users could wander the streets in one of the three modes:
summer, winter, and polygons.
The VR industry mainly provided VR devices for medical, flight simulation, automobile
industry design, and military training purposes from 1970 to 1990

2000-2015

In 2001, SAS3 or SAS Cube became the first PC based cubic room, developed by Z-A Production (Maurice Benayoun, David Nahon), Barco, Clarté, installed in Laval France in April 2001. The SAS library gave birth to Virtools VRPack. By 2007, Google introduced Street View, a service that shows panoramic views of an increasing number of worldwide positions such as roads, indoor buildings and rural areas. It also features a stereoscopic 3D mode.

2017

By 2017 there were at least 240 companies developing VR-related products. Facebook has 400 employees focused on VR development; Google, Apple, Amazon, Microsoft, Sony and Samsung all had dedicated AR and VR groups. Dynamic binaural audio was common to most headsets released that year

Virtual reality job openings are up 800%

5 Uses of Virtual Reality

Virtual reality technology holds enormous potential to change the future for a number of fields, from medicine, business, architecture to manufacturing.

Psychologists and other medical professionals are using VR to heighten traditional therapy methods and find effective solutions for treatments of PTSD, anxiety and social disorders. Doctors are employing VR to train medical students in surgery, treat patients' pains and even help paraplegics regain body functions.

In business, a variety of industries are benefiting from VR. Carmakers are creating safer vehicles, architects are constructing stronger buildings and even travel agencies are using it to simplify vacation planning.

introduction to google Cardboard

1.Introduction

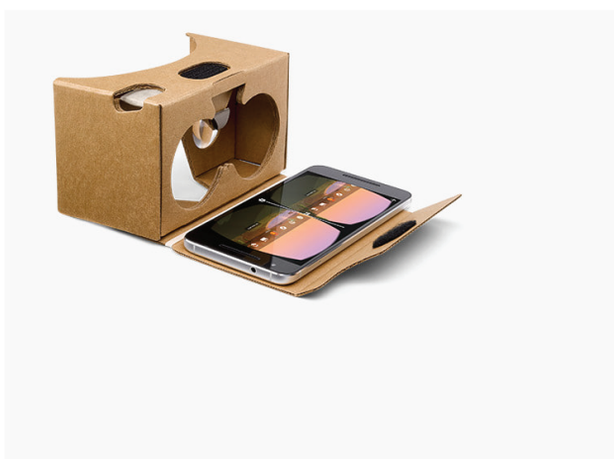
Google Cardboard is a virtual reality (VR) platform developed by Google for use with a head mount for a smartphone. Named for its fold-out cardboard viewer, the platform is intended as a low-cost system to encourage interest and development in VR applications.[1][2] Users can either build their own viewer from simple, low-cost components using specifications published by Google, or purchase a pre-manufactured one. To use the platform, users run Cardboard-compatible applications on their phone, place the phone into the back of the viewer, and view content through the lenses.

The platform was created by David Coz and Damien Henry, Google engineers at the Google Cultural Institute in Paris, in their 20% "Innovation Time Off".[3] It was introduced at the Google I/O 2014 developers conference, where a Cardboard viewer was given away to all attendees. The Cardboard software development kit (SDK) is available for the Android and iOS operating systems; the SDK's VR View allows developers to embed VR content on the web as well as in their mobile apps.[4]

Through March 2017, over 10 million Cardboard viewers had shipped and over 160 million Cardboard app downloads had been made. Following the success of the Cardboard platform, Google announced an enhanced VR platform, Daydream, at Google I/O 2016.

2.Viewer assembly and operation

Google Cardboard headsets are built out of simple, low-cost components. The headset specifications were designed by Google, which made the list of parts, schematics, and assembly instructions freely available on their website, allowing people to assemble Cardboard themselves from readily available parts. Pre-manufactured viewers were only available from third-party vendors until February 2016, when Google began selling their own through the Google Store.[5]



3.Software

Google provides three software development kits for developing Cardboard applications: one for the Android operating system using Java, one for the game engine Unity using C#, and one for the iOS operating system.[8] After initially supporting only Android, Google announced iOS support for the Unity plugin in May 2015 at the Google I/O 2015 conference.[9] Third-party apps with Cardboard support are available on the Google Play store[10] and App Store for iOS. In addition to native Cardboard apps, there are Google Chrome VR Experiments implemented using WebGL; phones, including Apple's, that support WebGL can run Google's web experiments.[11][12] A port of the Google Cardboard demonstration app to iOS was released at Google I/O 2015.[13] In January 2016, Google announced that the software development kits would support spatial audio, a virtual reality effect intended to simulate audio coming from outside of the listener's head located anywhere in 3D space.[14][15]

In March 2016, Google released VR View, an expansion of the Cardboard SDK allowing developers to embed 360-degree VR content on a web page or in a mobile app, across desktop, Android, and iOS.[16] The JavaScript and HTML code for web publishing VR content is open source and available on GitHub, allowing developers to self-host their content.[17]

CHAPTER 2

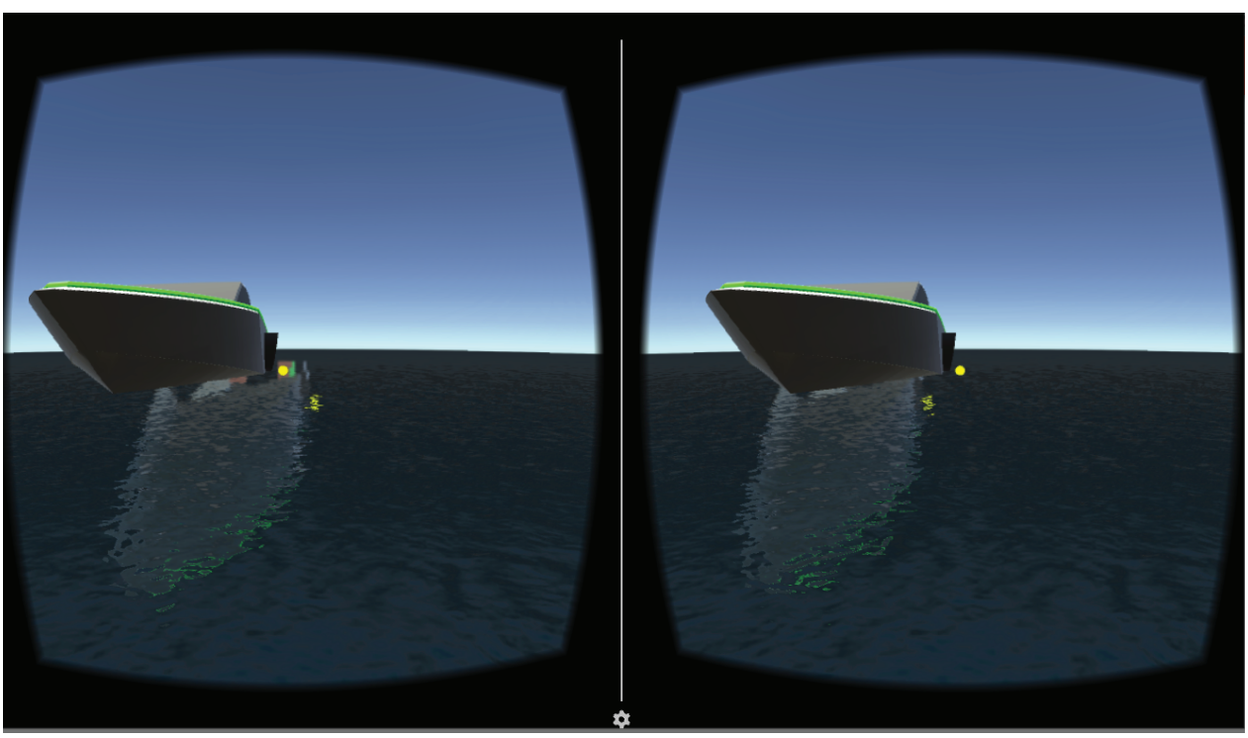
HELLO, VR WORLD

Google VR in Unity on Android :

1.Introduction

Google Cardboard is a technology platform to develop virtual reality (VR) applications for mobile phones. A main component of the platform is a low-cost headset with dual lenses to produce "stereoscopic 3D view" of the scene displayed on the phone screen. A Google Cardboard compatible application generates slightly different versions of a (2D) image on a split screen that, when viewed with the headset, creates the illusion of 3D depth.
Google has published specifications for platform compatible headsets and provided two development kits for Cardboard applications.

-Android SDK: Based on Android Java, applicable to only Android devices.
-Unity SDK: Based on Unity Editor and C#, applicable to both Android and iOS devices.

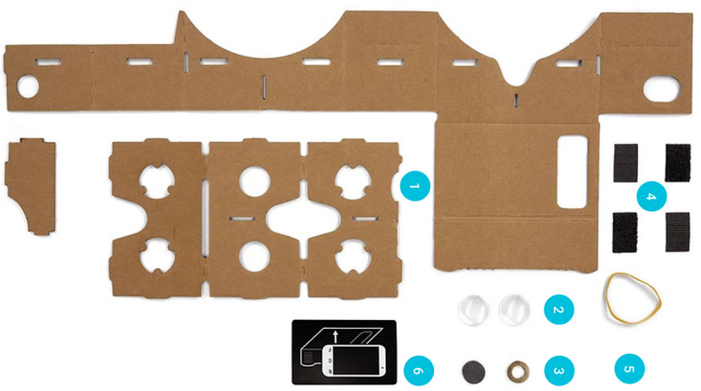


2.Hardware requirements:

Daydream:
Daydream-ready phones are built for VR with high resolution displays, ultra smooth graphics and high-fidelity sensors.

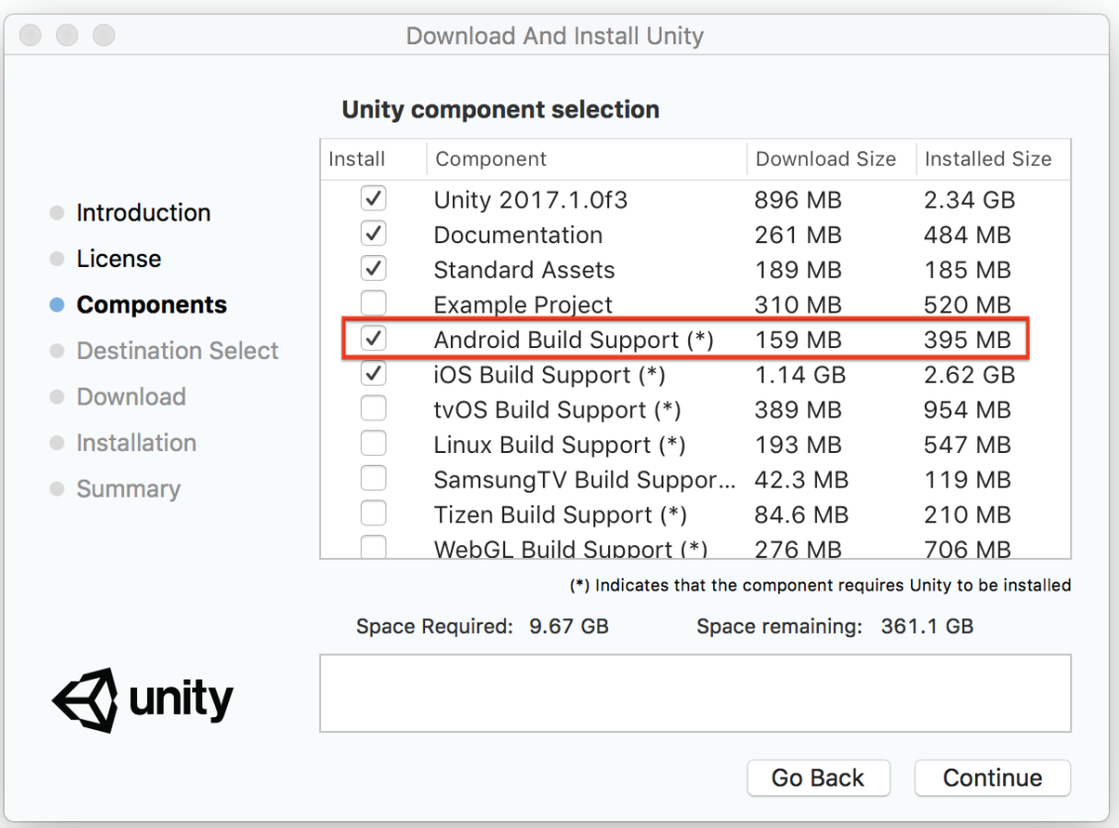
Cardboard:

Google Cardboard brings immersive experiences to everyone in a simple and affordable way. Whether you fold your own or buy a Works with Google Cardboard certified viewe



3.Software requirements:

we need Unity 5.6 or later. to use the sdk.Also Android Build Support Compnents



4.Controller support in Unity

The Google VR SDK for Unity provides support for the Daydream controller, including the following capabilities:

Arm model: Mathematical model that predicts the location of the user's controller, based on the controller orientation and by using virtual shoulder, elbow, wrist, and pointer joints.

Input system: Integration with Unity's standard EventSystem. Integrates with the laser and includes custom graphic and physics raycasters and reticle visualizations to allow users to interact with Canvas UI elements and game objects in your scene.

Reskinnable visualizations: Default visualizations that can be reskinned or used as is:

Controller: 3D model of the Daydream controller model that indicates which button the user is currently pressing and/or where the user is currently touching the controller touchpad.

Laser and reticle: Displays a laser and reticle to help users interact with the VR environment.

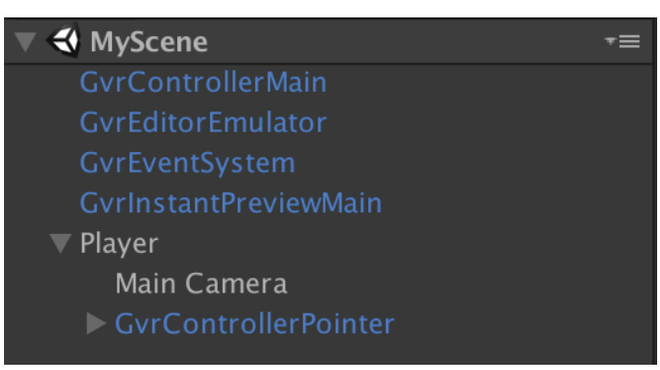
Tooltips: Customizable tooltips indicating how to use the controller buttons and touchpad.

The SDK includes several prefabs to support the Daydream controller and integrate with the Unity's Input Sytem

5.Controller and Input System prefabs

Prefab name	Location in scene	Description
GvrControllerMain	*Anywhere in scene*	*Main controller prefab, responsible for managing controller state. Includes the GvrControllerInput component, which is the main entry point to the*
GvrControllerPointer	*Sibling to the Main Camera*	*Daydream controller prefab. Provides controller, laser, and reticle visualizations and serves as attachment point for tooltips and custom visualiza-*
GvrEventSystem	*Anywhere in scene*	*Lets the Daydream controller use the Unity event system.*
GvrEditorEmulator	*Anywhere in scene*	*Editor play mode camera controller. Lets you simulate user's head movement with your mouse or trackpad*
GvrInstantPreviewMain	*Anywhere in scene*	*Editor play mode Instant Preview controller. Lets you stream a stereo preview to your phone and use a physical Daydream controller in the editor.*

The resulting scene will look something like this:



CHAPTER 3

THE BRAINSTORMING PROCESS

Introduction :

In this chapter i'am going to talk about the game purpose and the brainstorming process that took place before starting the game development. Also the functional, quality requirements and the game conception.

1. About the Game

In this part I will introduce the details about the game in order to give concrete and precise idea about the different aspects of «JETMAN GO».

2.Game Purpose

Adventure game the player should try to collect a bigger score every time

3.Character

The character a man with a jetpack.

4.Enemies

different type of enemies static the player should avoid ,and live enemies they will attack the player.

5.Scenario

The game is about a funny characterwith with jetpack ,he is trying to kill evil robots and get back his home village

6.Software to Use

These are mainly the software we will be using during the game development process.

Unity 3D 2018 as the game engine.
Maya for modelling, texturing and Animation.
Adobe Photoshop and Illustrator CC 2018 for graphic conception and texturing.
MonoDevelop as an open source integrated development environment for game programming(C#).

7.Hardware to use

Lenovo Y510p
RAM 16GO
CPU intel i7
GPU Nividia GEFORCE

FOR TESTING
Main Device
SAMUNG NOTE 5
SAMSUNG S7 EDGE

Use Case Diagram

A use case diagram at its simplest is a representation of a player's interaction with the game that shows the relationship between the player and the different use cases in which the player is involved.
Below is the use case diagram of our game [See Figure 21].

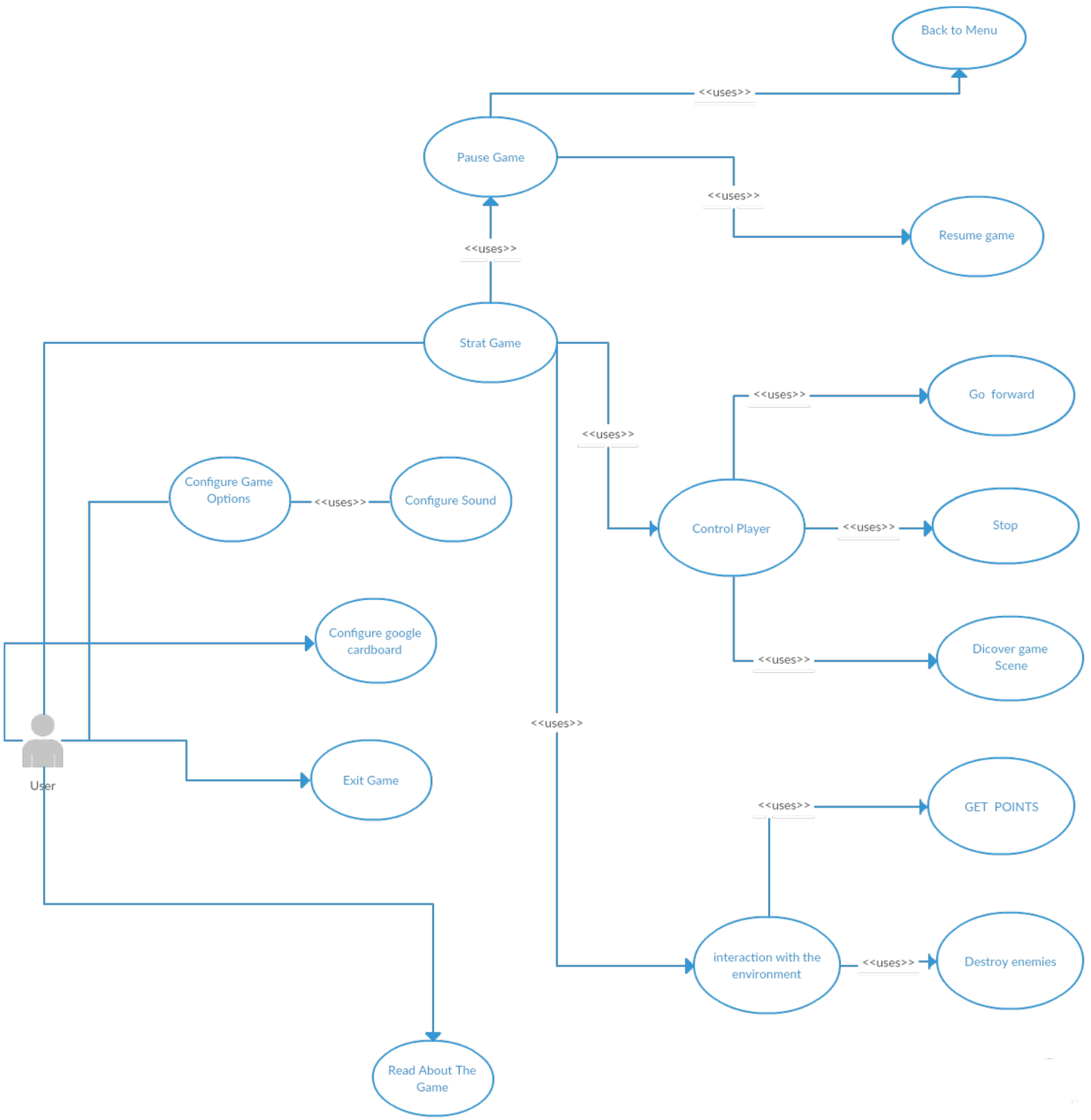


Figure 21 Use case diagram

b. Textual description of the use case diagram

For further enlightenment of the use case diagram, we are going to add a textual description for the main use case. Textual description of the use case “Play first floor”:
Identification:

Use Case Name: Player
Intent: the player must collect and kill Enemies
Summary: the player opens all rooms and collects the masterpieces.
Primary Actor: Player.
Person in Charge:Me
Main Flow of Event:
1) «Player» discovers the Game World
2) «Player» kill Enemies.
5) «Player» collects Points.

Exception n°1: Death of the character
1) One of the enemies kills the character.
2) The character dies but the player can resume where he stopped if the game was saved.

Exception n°2: Player leaves the game before finishing the game

1) The player pauses the game.
2) The system shows the pause menu.
3) The player quits the game.

Exception n°3:Flow n°1: Time is Up
1) The Player must finish the game before the Time gets up

9. sequence diagram

A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart.

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

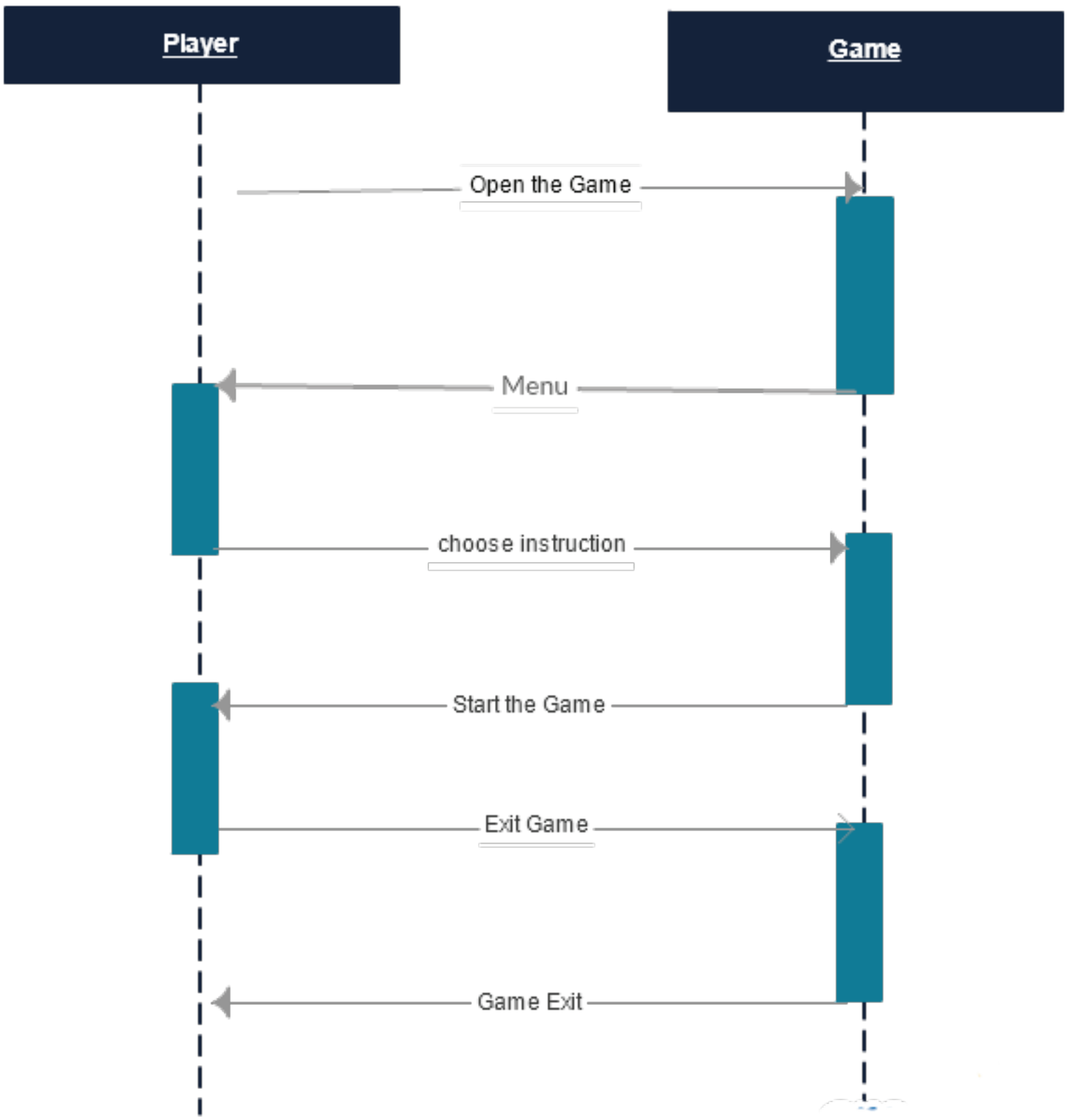


Figure 22:**sequence diagram :Main Game**

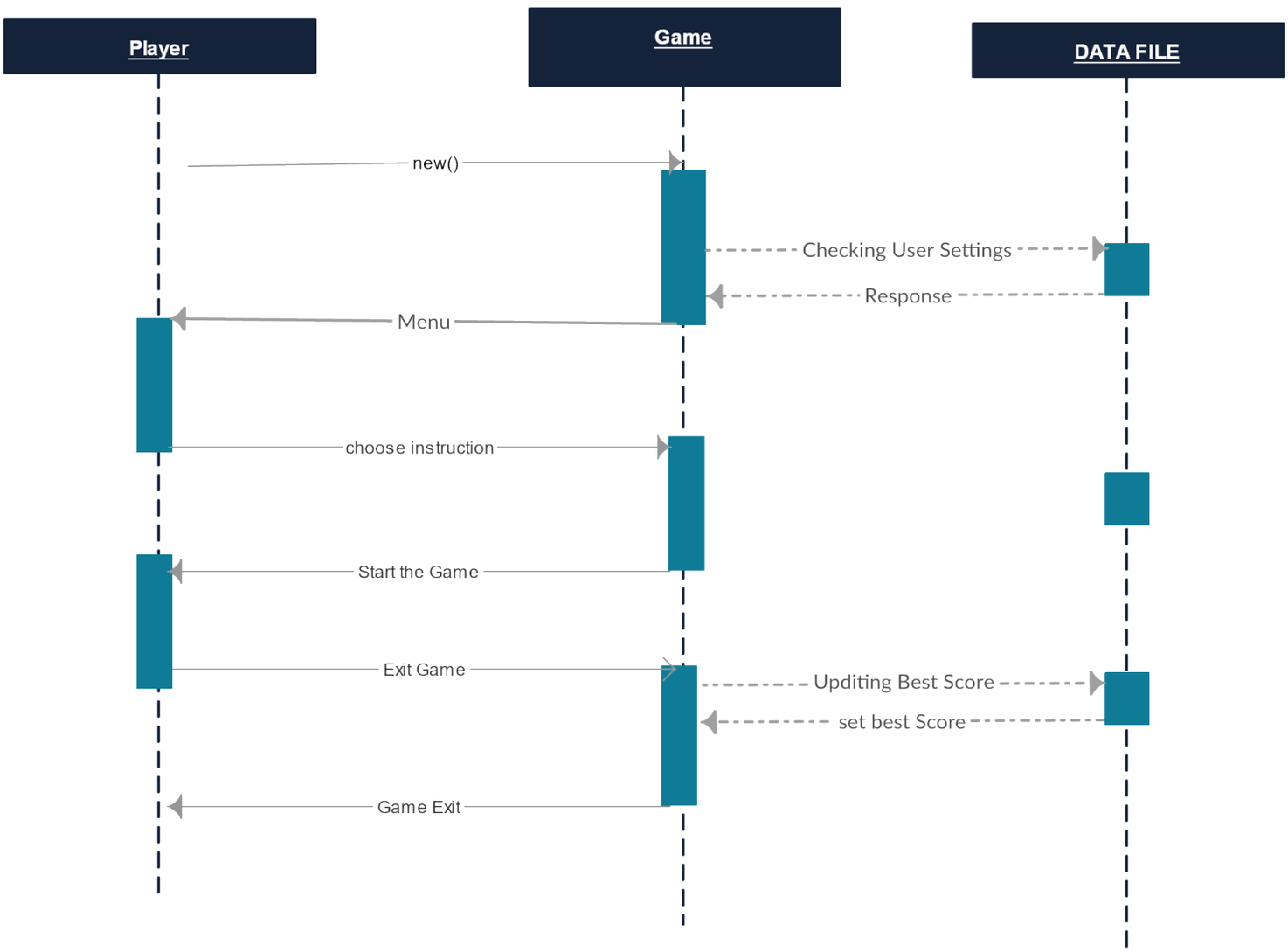


Figure 23:**sequence diagram :Using A Data File To store Simple Information**

10.Product Quality Requirements

a. SQuaRE Standard

The model for External and Internal software Quality (Software product quality model) that we will highlight in the following is the first part of The SQuaRE quality model.

Figure 24 Software product quality model

The model for Quality in Use that we will highlight in the following is the second part of The SQuaRE quality model.

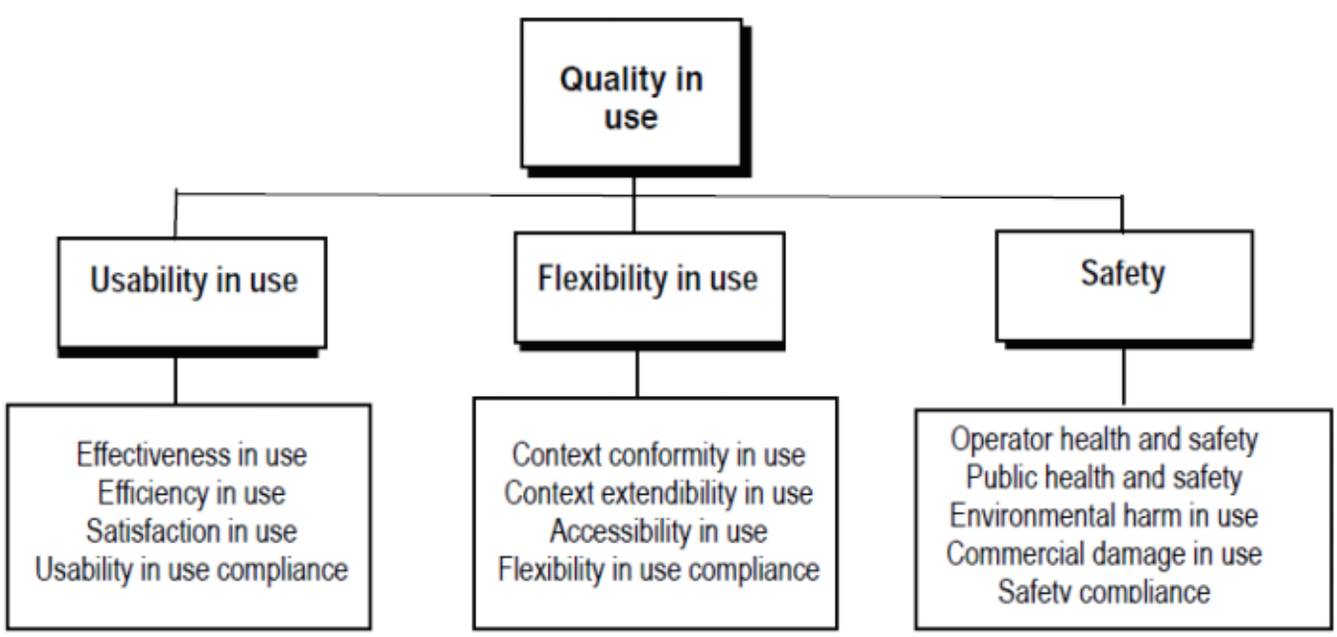
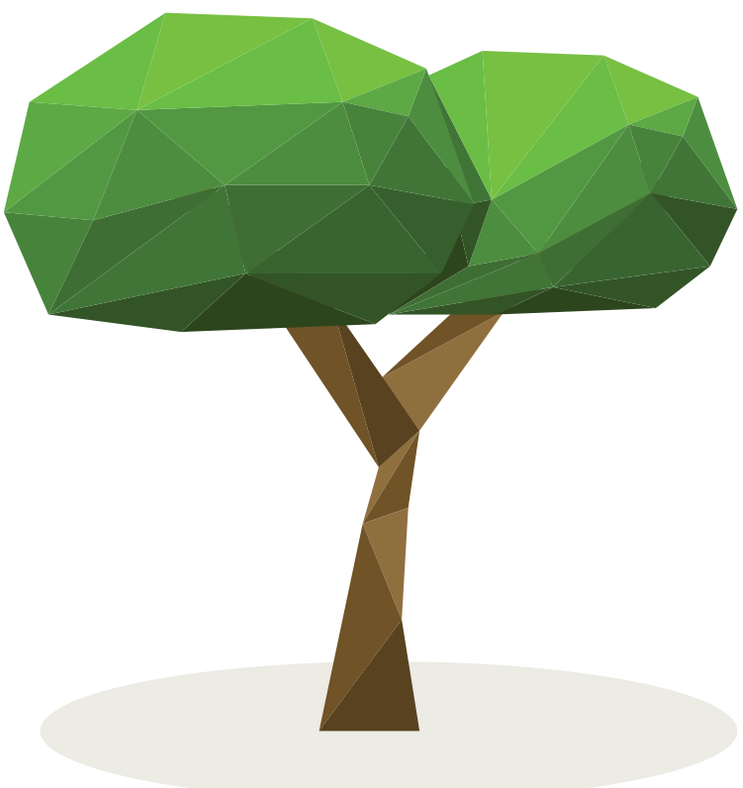


Figure 25 Quality in use model

CHAPTER 4



START WORKING WITH LOW POLY GAME OBJECTS

Introduction to 3D :

1. Modelling

Low poly

Low poly is a polygon mesh in 3D computer graphics that has a relatively small number of polygons. Low poly meshes occur in real-time applications (e.g. games) as contrast with high poly meshes in animated movies and special effects of the same era. The term low poly is used in both a technical and a descriptive sense; the number of polygons in a mesh is an important factor to optimize for performance but can give an undesirable appearance to the resulting graphics.

Low poly as a relative term

There is no defined threshold for a mesh to be low poly; low poly is always a relative term and depends on (amongst other factors):

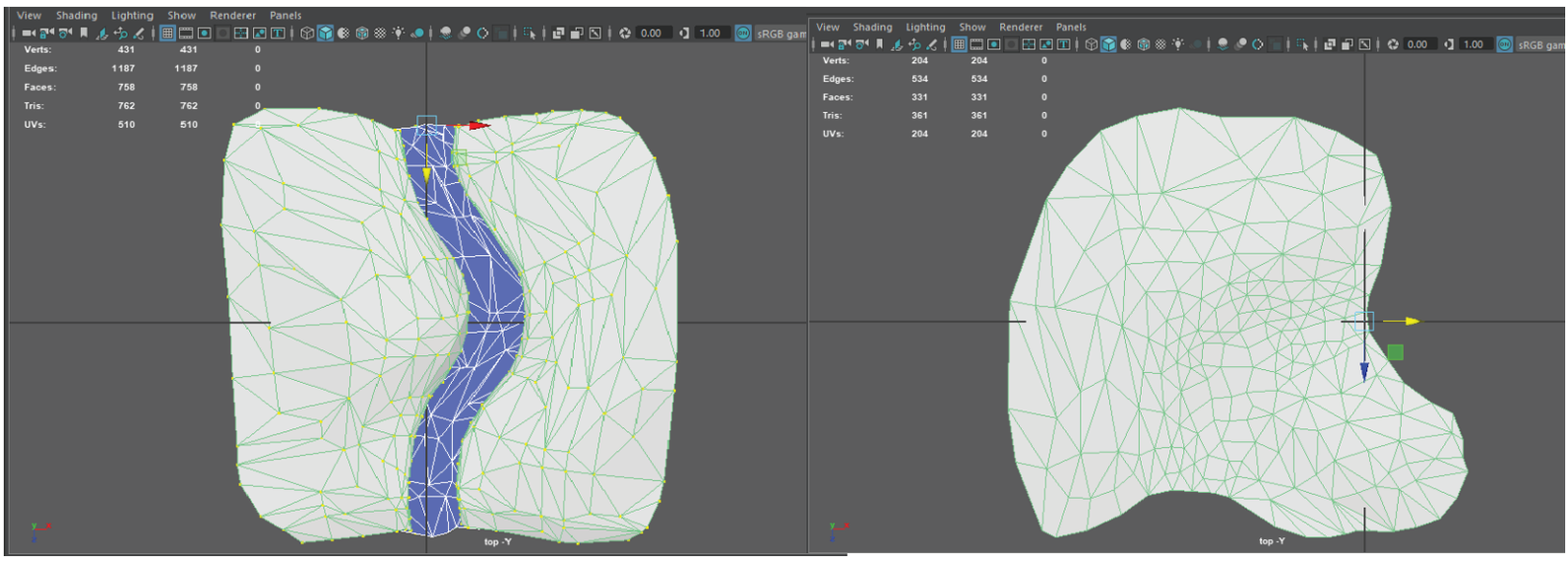
The time the meshes were designed and for what hardware
The detail required in the final mesh
The shape and properties of the object in question
As computing power inevitably increases, the number of polygons that can be used increases too. For example, Super Mario 64 would be considered low poly today, but was considered a stunning achievement when it was released in 1996. Similarly, in 2009, using hundreds of polygons on a leaf in the background of a scene would be considered high poly, but using that many polygons on the main character would be considered low poly.

Low poly meshes in physics engines

Physics engines have presented a new role for low poly meshes. Whilst the display of computer graphics has become very efficient, allowing (as of 2009) the display of tens to hundreds of thousands of polygons at 25 frames per second on a desktop computer, the calculation of physical interactions is still slow. A low poly simplified version of the mesh is often used for simplifying the calculation of collisions with other meshes, in some cases this is as simple as a 6 polygon bounding box.

For modelling i have used MAYA 2017,in order to facilitate the modelling process,i have come with the result shown in the screenshot below

2.SOME TERRAINS EXAMPLES



VERTS 431
EDGES 11787
FACES 578

VERTS 204
EDGES 534
FACES 331

Figure 26 Main TERRAIN in Maya

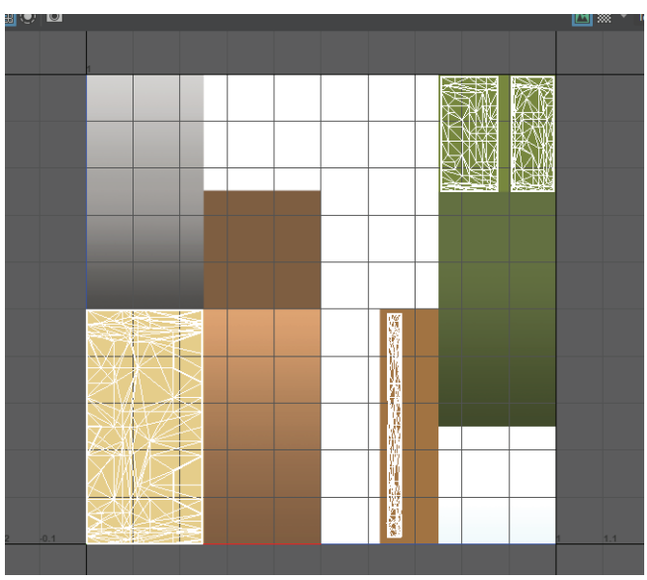


Figure 27 Main TERRAIN UV MAPPING

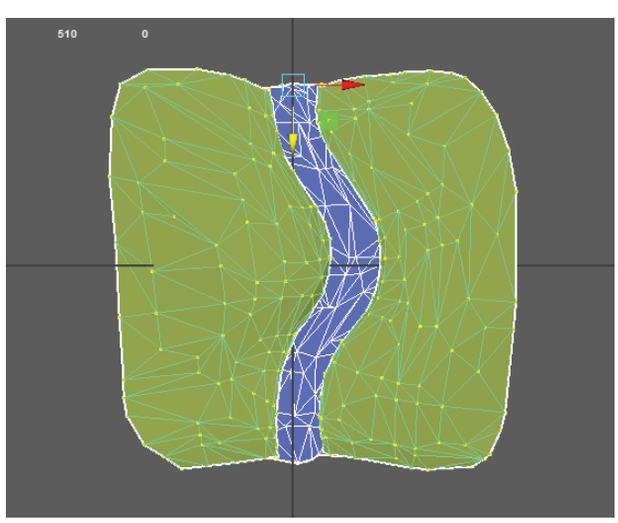


Figure 28 Main TERRAIN RESULT

3 . Some Environment Objects

During this phase i have used Maya to create objects, i tried to make the number of polygons as much low as possible in order to get optimized models that will be eventually used on the unity game engine later, Some Examples :

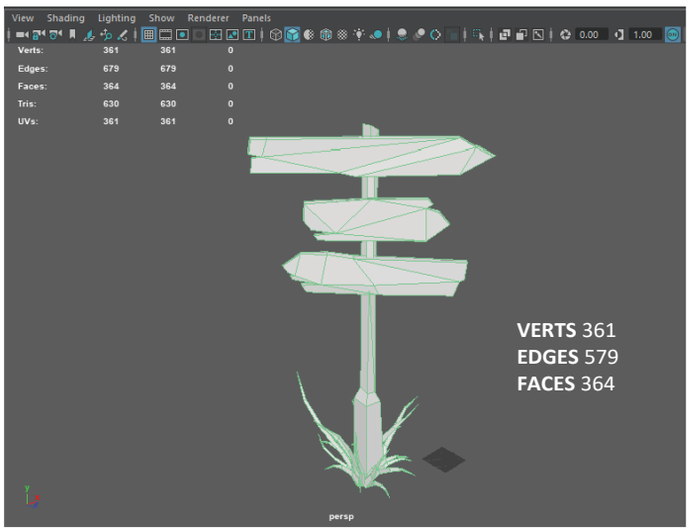


Figure 29: Low Ploy Environment Object

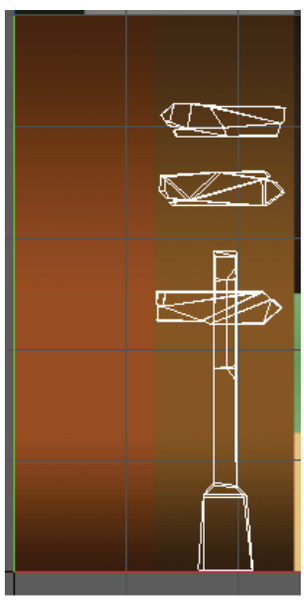


Figure 30: Low Ploy Uv Mapping

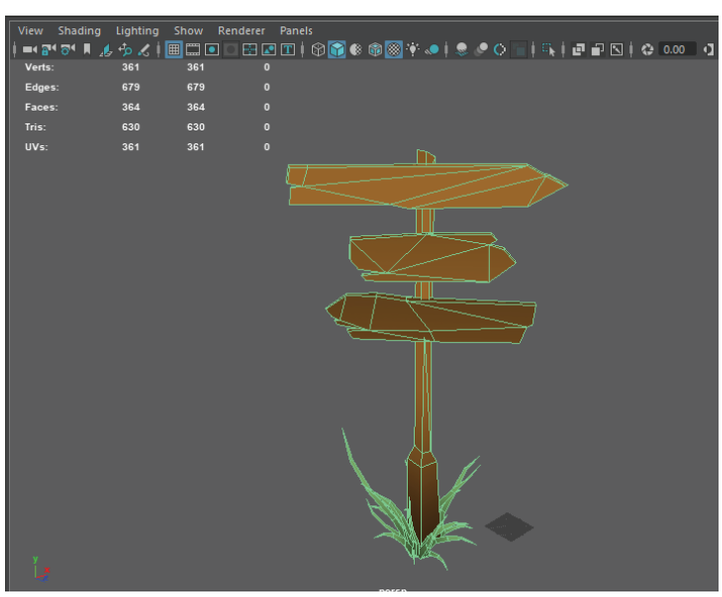


Figure 31: Final Result

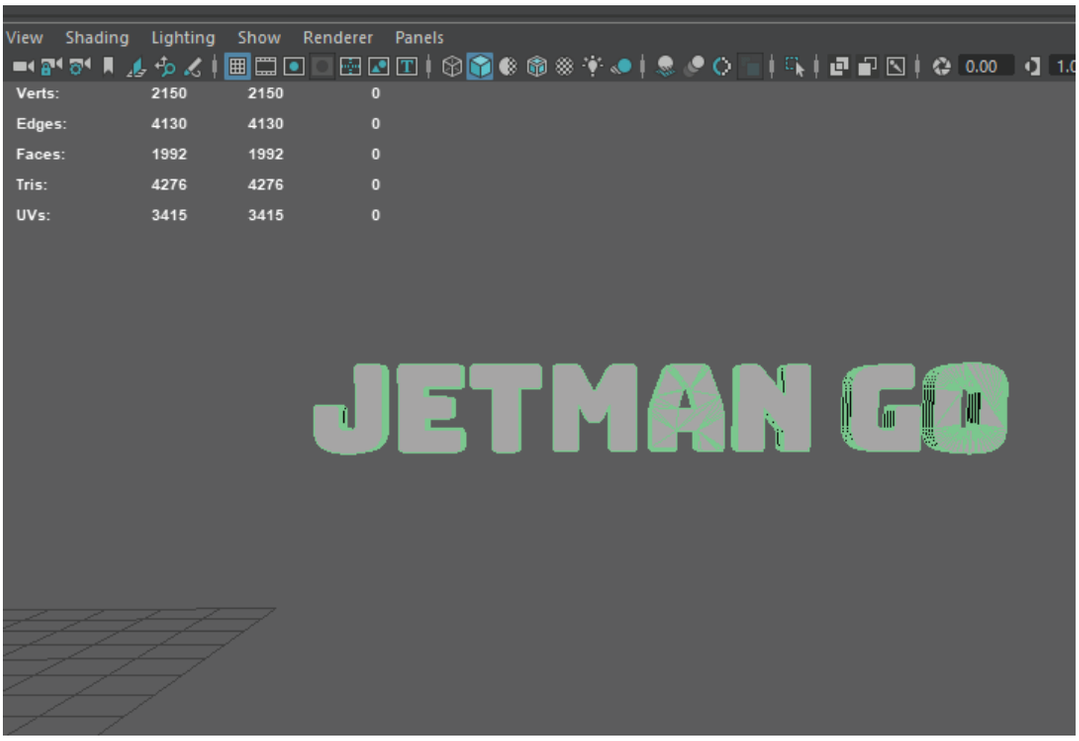


Figure 32: Game Logo

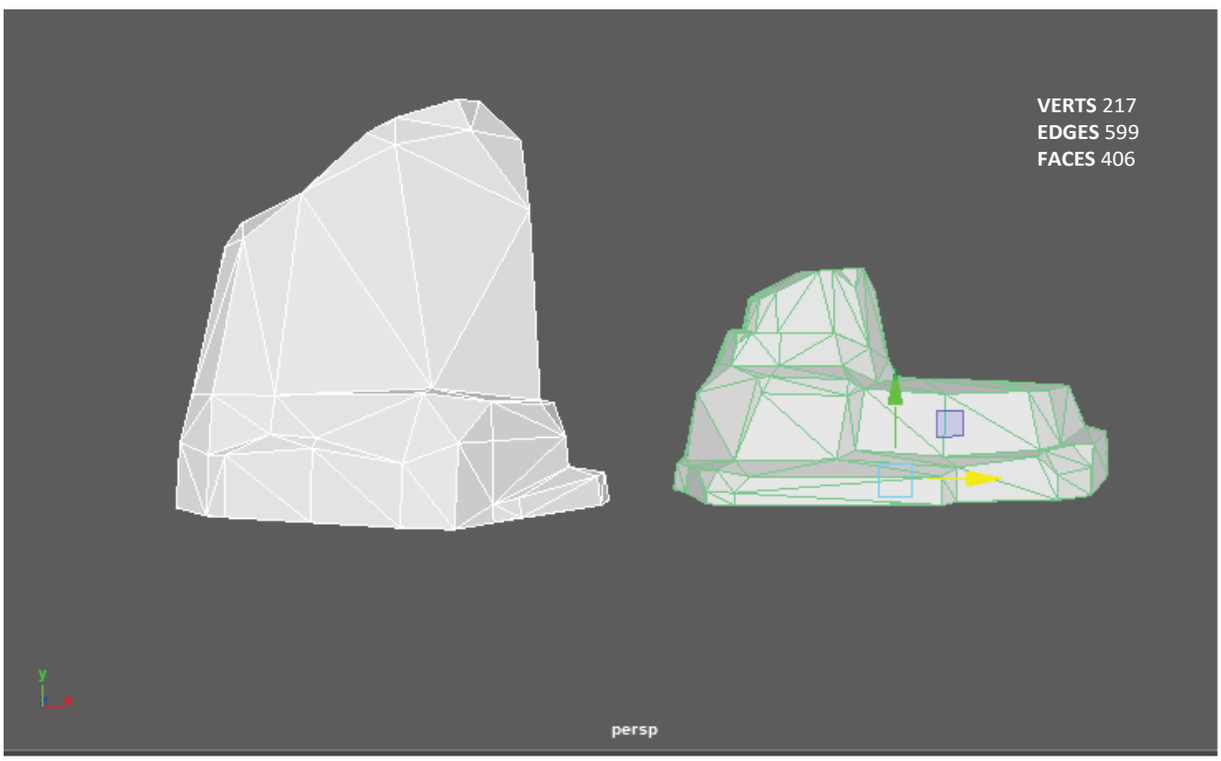


Figure 33: Ploy Environment Object Example

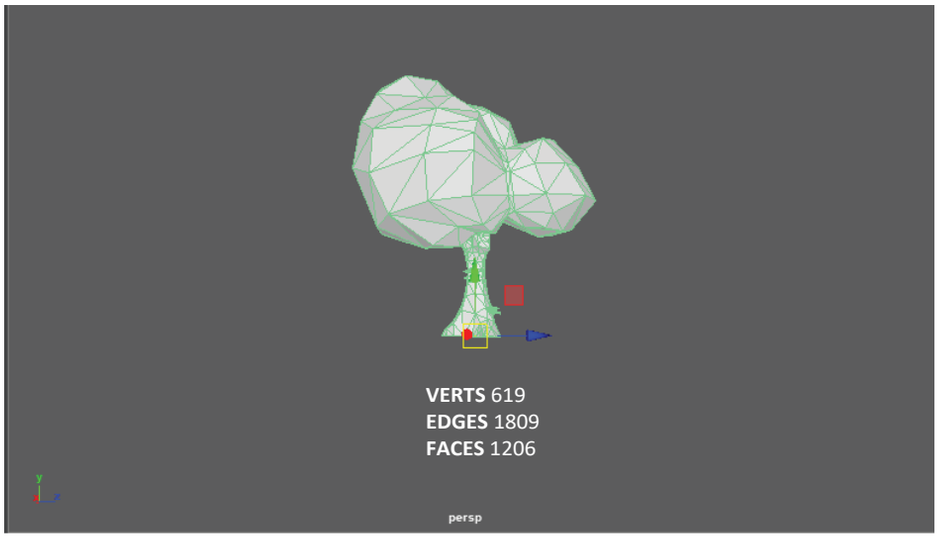


Figure 34: TREE EXAMPLE

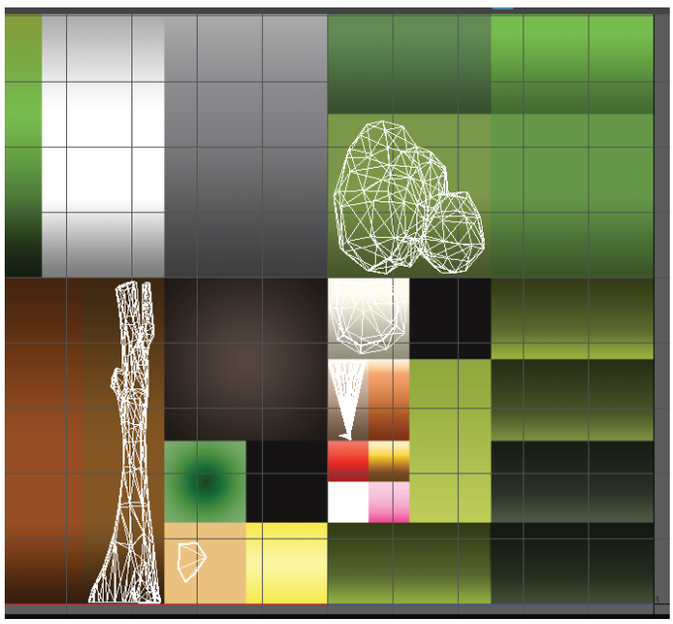


Figure 34: TREE UV

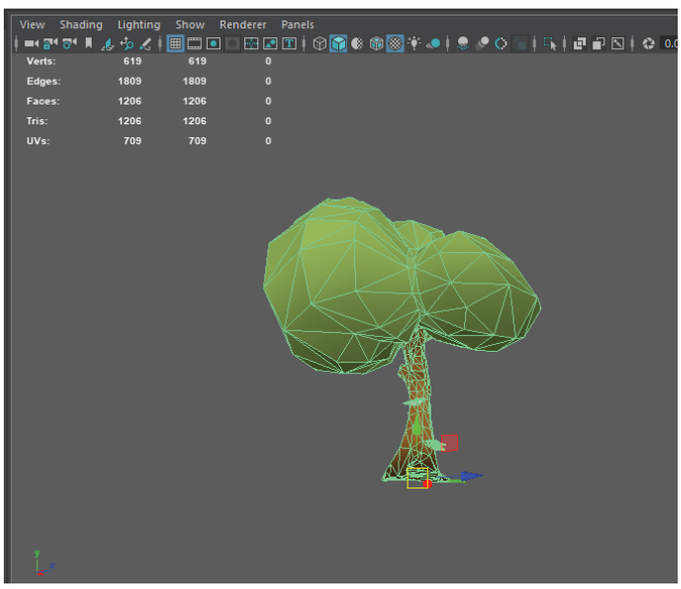


Figure 35: FINAL TREE

4.Bringing assets From MAYA to unity

After modelling, texturing process,
Here is the result
i have also used some packages to make the game looks more attractive

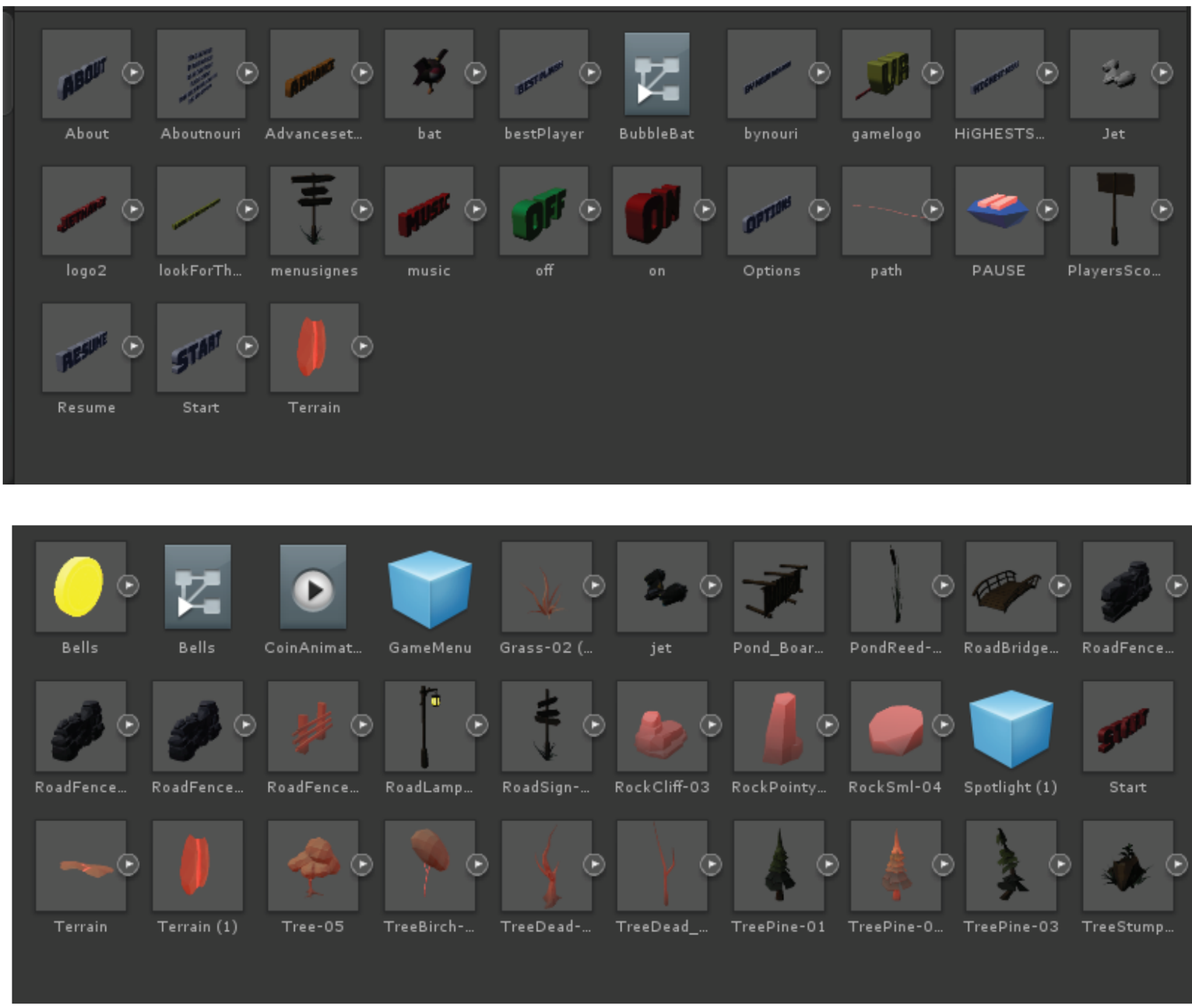


Figure 35: unity Assests

5.Simple Rigging and animation

The Game has animations such as this example

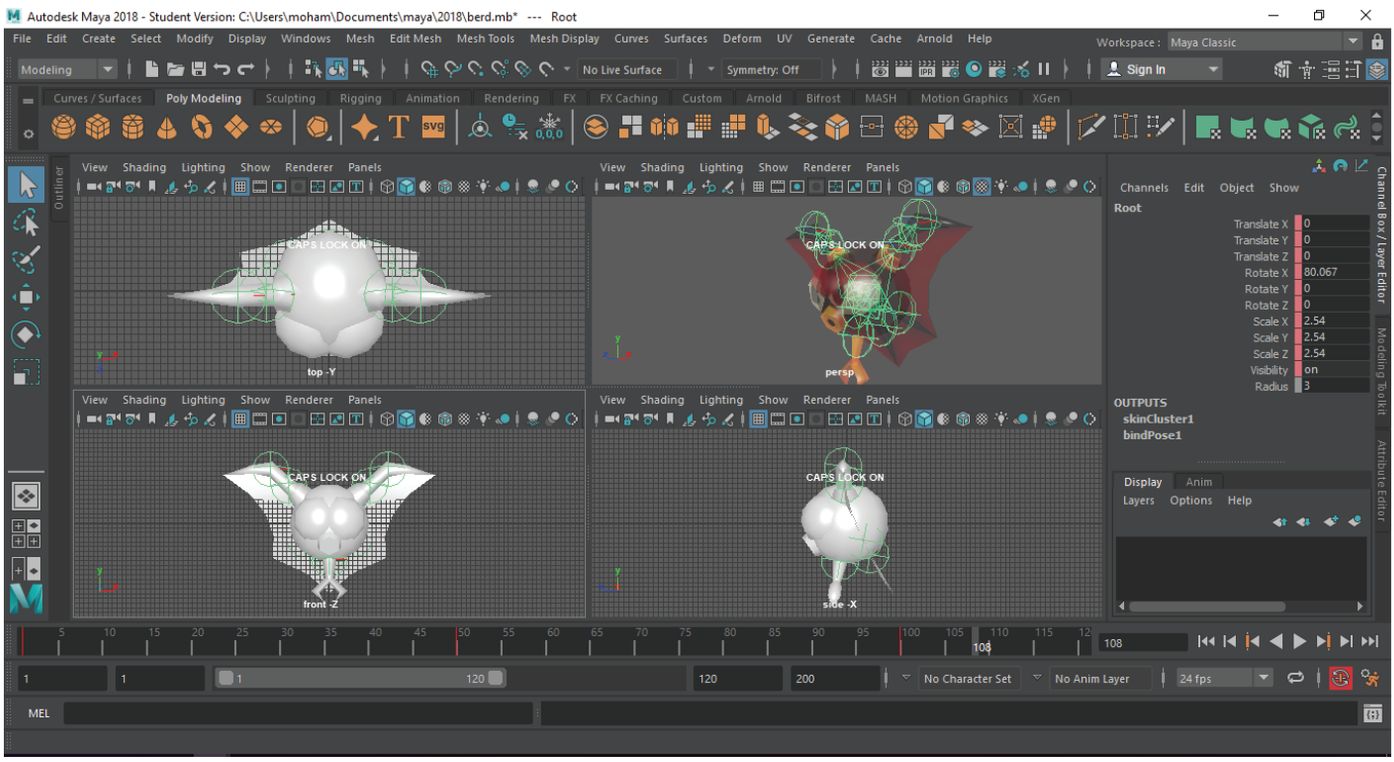


Figure 36: Simple Rigging Maya 2018 (ik)

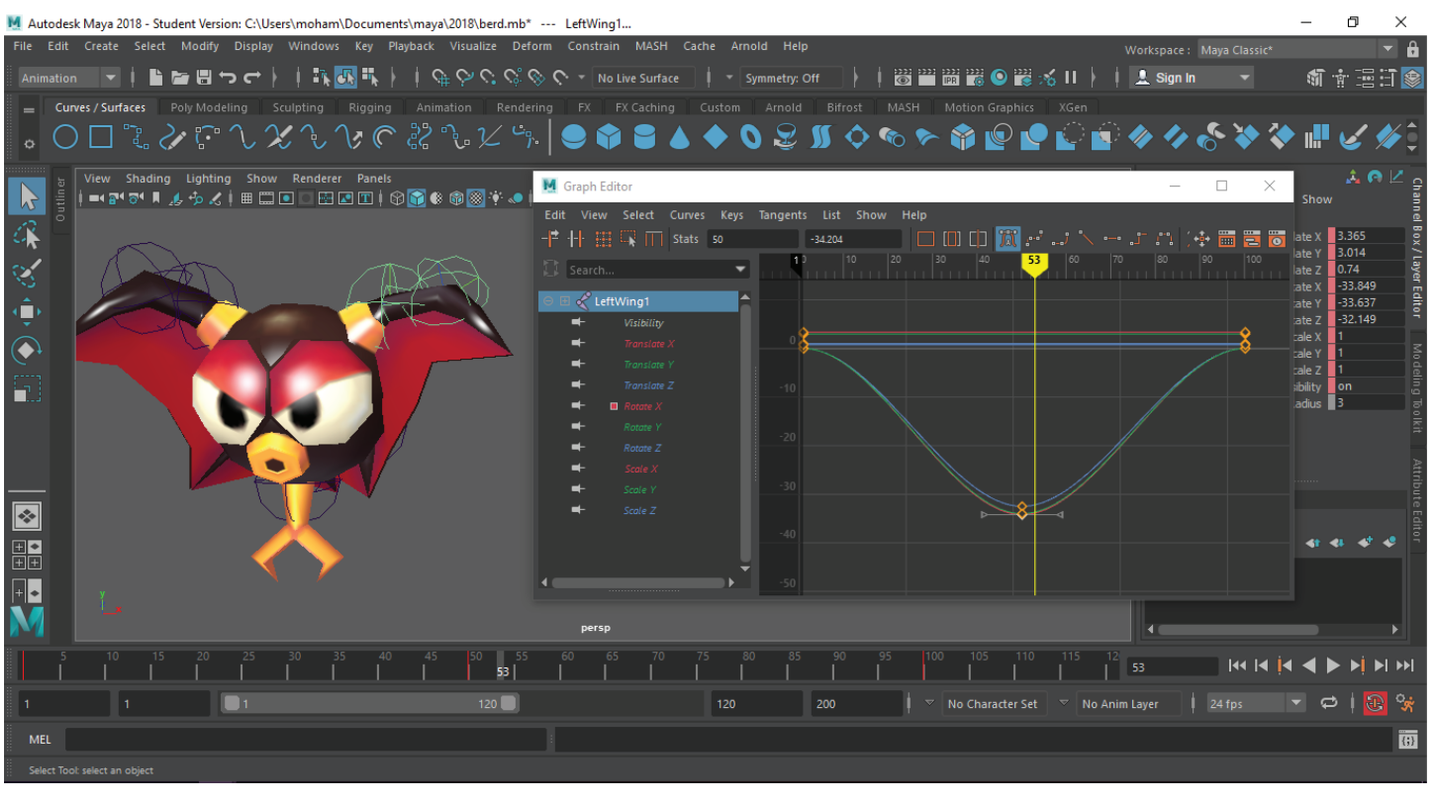


Figure 37: Simple Animation USing Key Frame

6.Bringing Animation From MAYA to unity

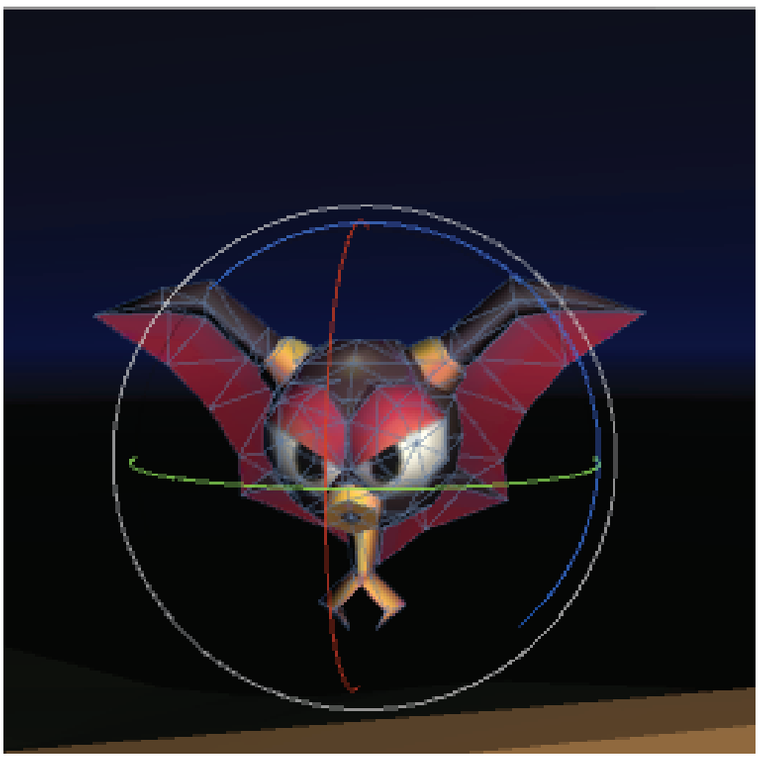


Figure 38: Animation From Maya To Unity Example

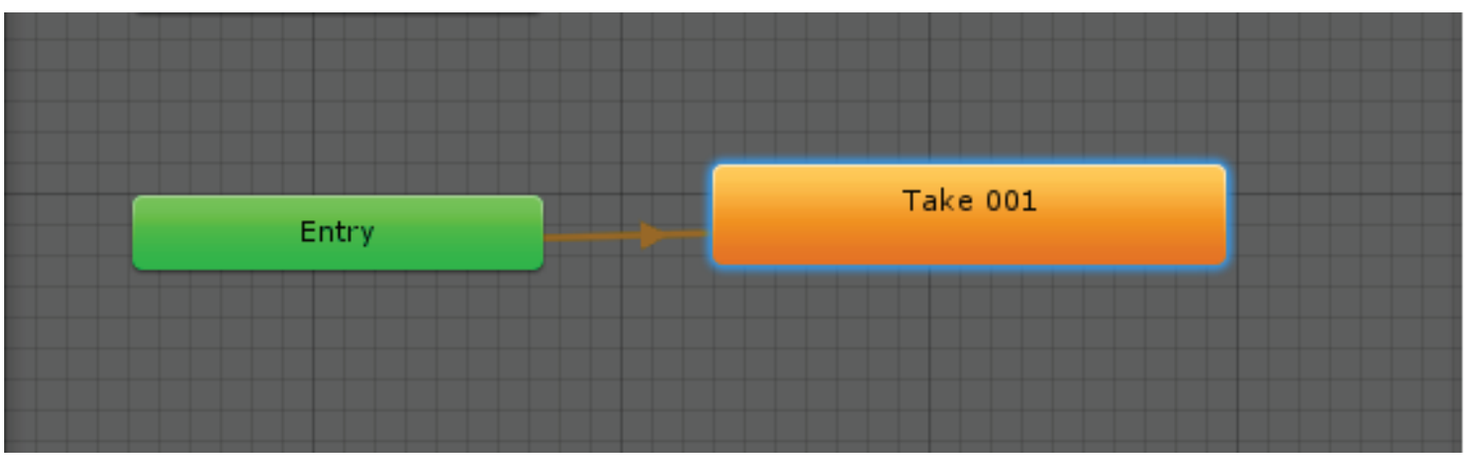


Figure 39: Setting Animator Controller

CHAPTER 5

BUILDING SCENES

The Main Menu

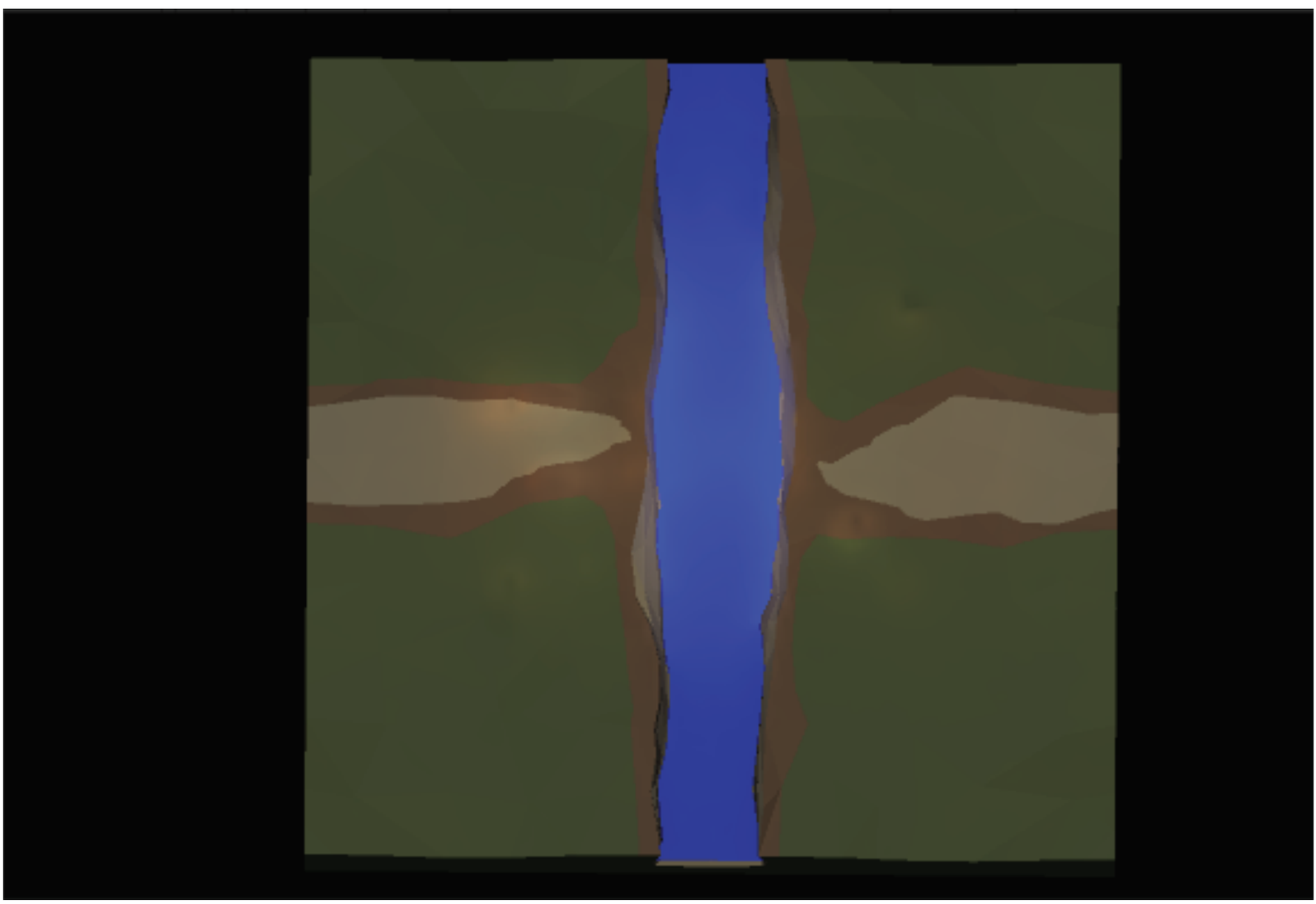


Figure 40: **Starting with An Empty Terrain**

Keeping Gameobject Organised is important

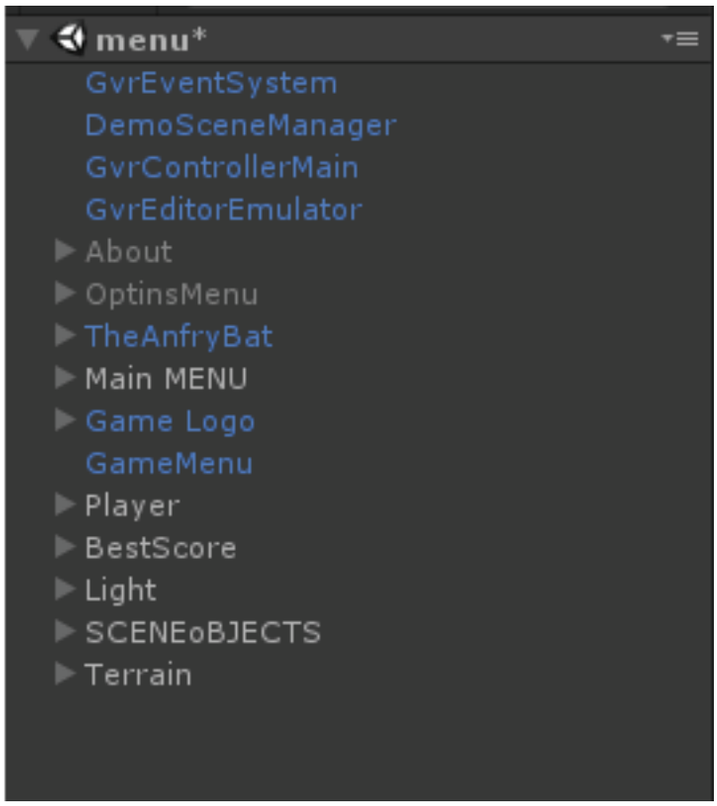


Figure 41: **Menu hierarchy**

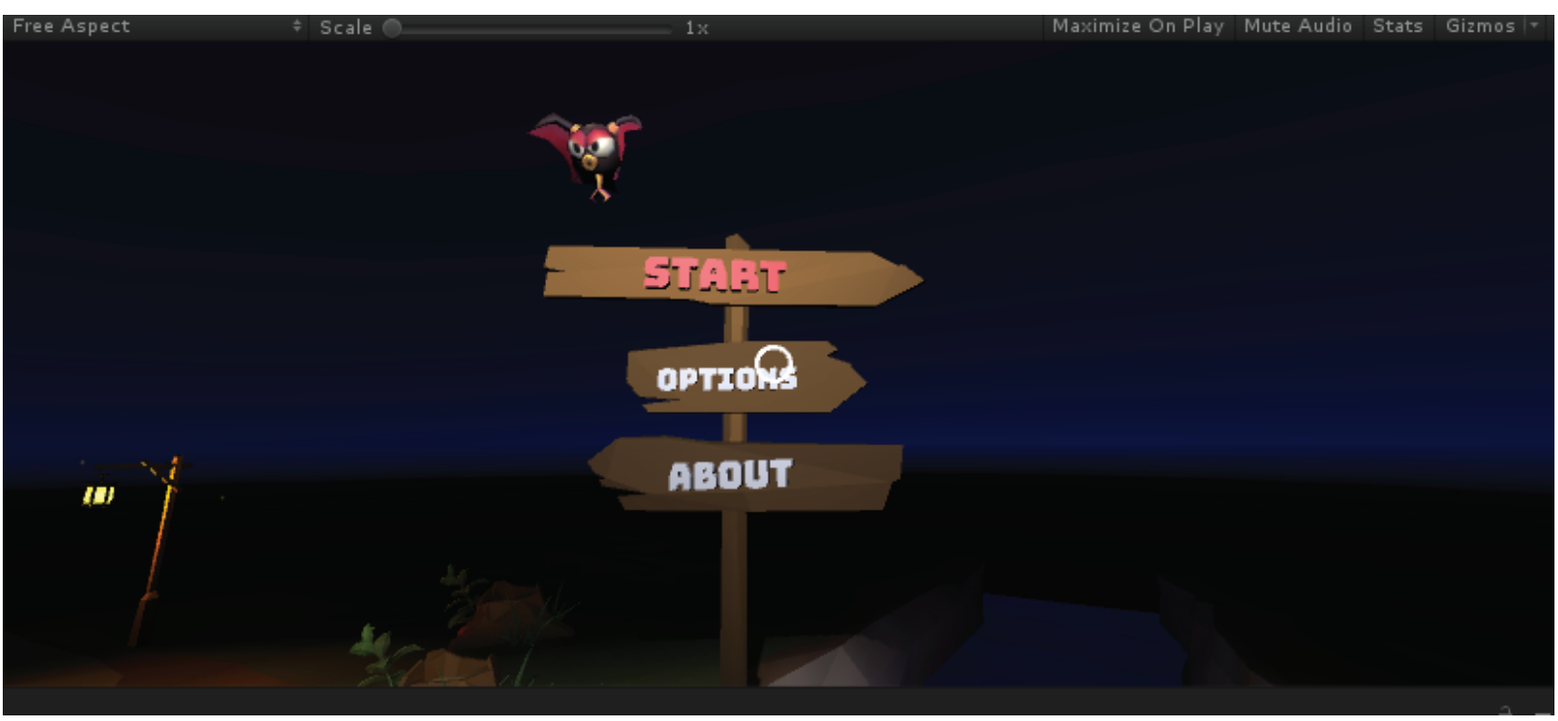


Figure 41: **Menu UI**

Figure 42:**IF THE Player IS ALREADY PLAYED HE CAN RESUME THE GAME**

Figure 43:**The Player can hide/show the options Menu**

HANDLING USER INPUT

GvrControllerPointer prefab

Using GVRControllerPointer Prefab Provides a rendered Daydream controller with built-in laser pointer. Must be a sibling to the main camera, so that the camera and controller transforms share a common parent game object.

Requires an instance of the GvrControllerMain prefab to exist in the scene in order to be use the Google VR input system.

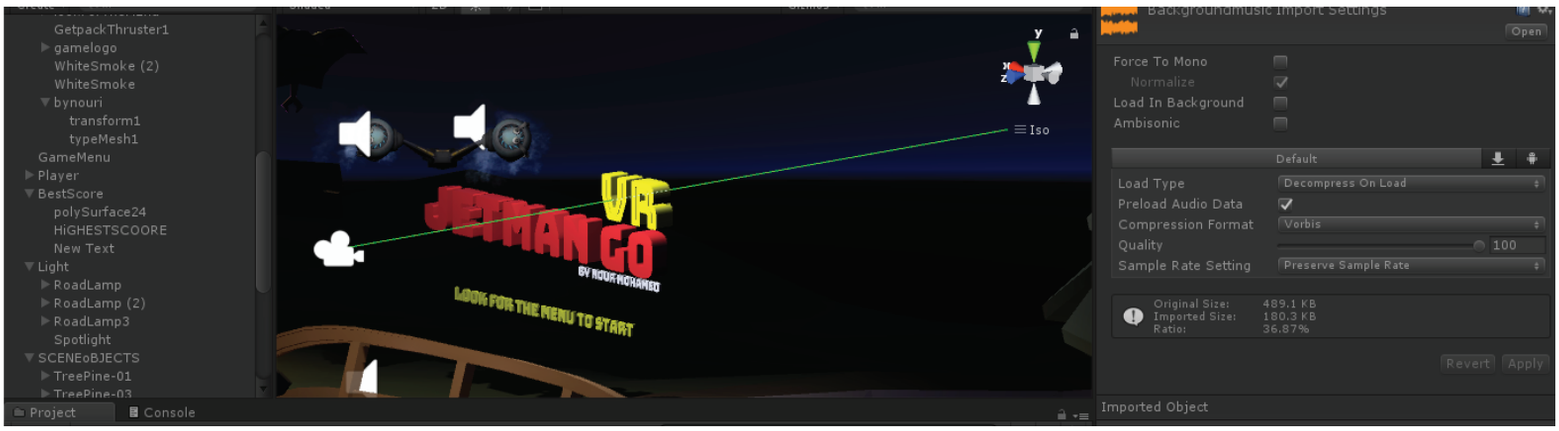


Figure 45:**GVRControllerPointer Example**

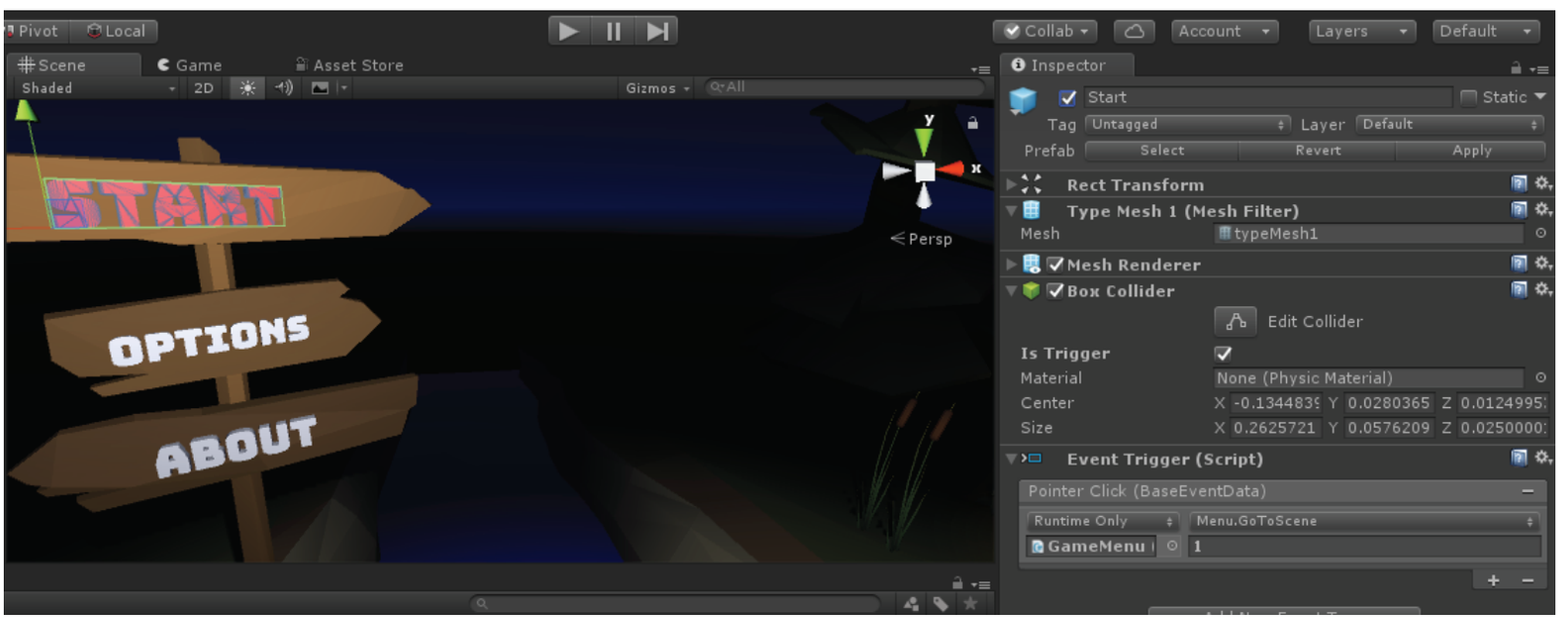


Figure 46:**Making The Menu interactive**

```
120
121
122     IEnumerator LoadNewScene(int scene ) {
123
124         yield return new WaitForSeconds(3);
125         AsyncOperation async = Application.LoadLevelAsync(scene);
126
127         while (!async.isDone) {
128             yield return null;
129         }
130
131     }
132
133
```

Figure 46:**Examlpe Loading A Scene Async thread**

Forcing the player Camera to see the Options menu

```
iTween.RotateTo(camera,iTween.Hash
    ("y",276,"time",3,"delay",0,"onupdate","myUpdateFunction","looptype",
        iTween.LoopType.none));
//camera.transform.Rotate(276f * Time.deltaTime, Space.World);
```

Figure 49: **Also i'm Using itween to force the user to see the Menu**

```
72
73      public void SoundOff(){
74          //Adding clicking Sound
75          AudioSource audio = GetComponent<AudioSource>();
76          audio.Play();
77
78          Debug.Log ("My methode Sound is here ");
79          if (soundon==false) {
80              soundon = true;
81              oFFObject.SetActive (false);
82              oNObject.SetActive (true);
83              audio.Play();
84          }
85          else if(soundon==true){
86              soundon = false;
87              oFFObject.SetActive (true);
88              oNObject.SetActive (false);
89
90          }
91
92      }
93
```

Figure 50: **Script to show/hide On Off button when the user clicks**

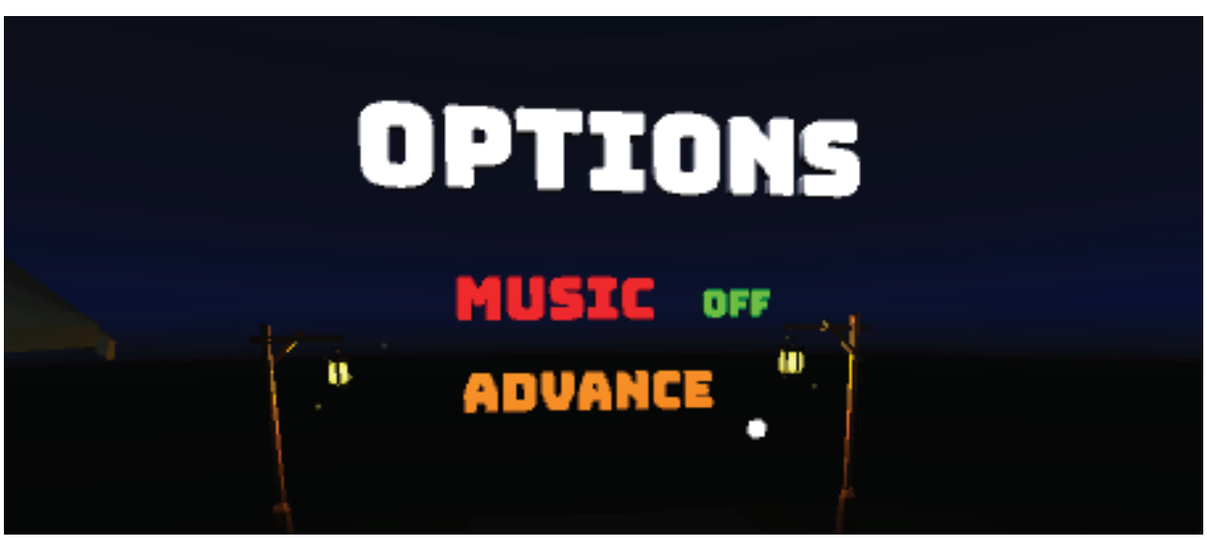


Figure 50: **On Off button**

```
21
22      public void GoToScene(int Number){
23
24
25          StartCoroutine(LoadNewScene(Number));
26
27      }
28
```

Figure 47: **Loading Scene Methode**

Options Menu

Figure 48: **Options Menu**

```
31      public void GoToSceneOptions(){
32          AudioSource audio = GetComponent<AudioSource>();
33          audio.Play();
34          Debug.Log ("Show Options");
35          if (OptionMenu.activeInHierarchy) {
36              OptionMenu.SetActive (false);
37          } else {
38
39              OptionMenu.SetActive (true);
40
41              if (soundon == true) {
42                  oFFObject.SetActive (false);
43                  oNObject.SetActive (true);
44
45              }
46              else if(soundon==false) {
47                  oFFObject.SetActive (true);
48                  oNObject.SetActive (false);
49
50              }
51               iTween.RotateTo(camera,iTween.Hash("y",276,"time",3,"delay",0,"onupdate","myUpdateFunction","looptype",iTween.LoopType.none));
52              //camera.transform.Rotate(276f * Time.deltaTime, Space.World);
53          }
54
55      }
56
```

Figure 49: **Options Menu Methode**

Figure 51: **Highest Score**

```
7  public class Playerscore : MonoBehaviour {
8
9      public float speed;
10     public Text countText;
11     public Text winText;
12     public static int PlayerHealth=5;
13
14     public static int count = 0;
15
16     public float myTime=99;
17     public Text timertext;
18
19
20     void Awake ()
21     {
22         try
23         {
24             TextMesh t = (TextMesh)gameObject.GetComponent(typeof(TextMesh));
25             string Score = PlayerPrefs.GetString ("Score");
26             if (Score != null) {
27                 t.text=Score;
28             } else
29                 t.text= "No Recorder yet";
30
31         }
32
33        catch
34     {
35             // Error: Use No Score Found ERROR
36     }
37
38     }
39
40
```

Figure 52: **Highest Score Script Reading Data From An external File**

Figure 53: **Highest Score Result**

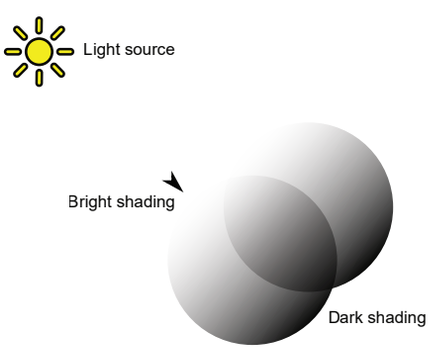
Visual Effects AND Particle Systems

Particle Systems

In a 3D game, most characters, props and scenery elements are represented as meshes, while a 2D game uses sprites for these purposes. Meshes and sprites are the ideal way to depict “solid” objects with a well-defined shape. There are other entities in games, however, that are fluid and intangible in nature and consequently difficult to portray using meshes or sprites. For effects like moving liquids, smoke, clouds, flames and magic spells, a different approach to graphics known as particle systems can be used to capture the inherent fluidity and energy. This section explains Unity’s particle systems and what they can be used for.

Using Point lights

A point light is located at a point in space and sends light out in all directions equally. The direction of light hitting a surface is the line from the point of contact back to the center of the light object. The intensity diminishes with distance from the light, reaching zero at a specified range. Light intensity is inversely proportional to the square of the distance from the source. This is known as ‘inverse square law’ and is similar to how light behaves in the real world.



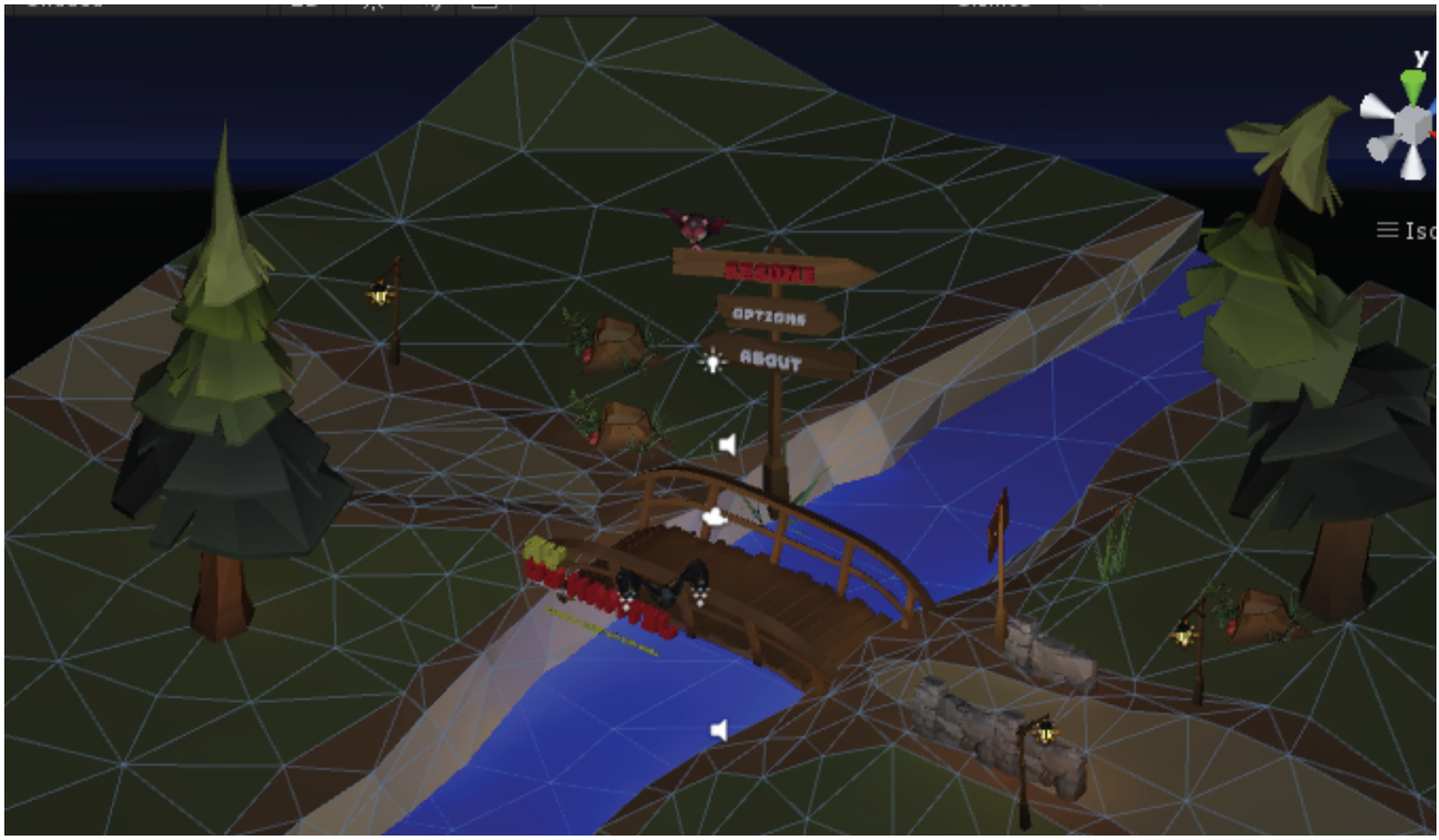


Figure 54: **The Scene With No Lights**

Figure 55: **Adding Particle System To the Lamp**

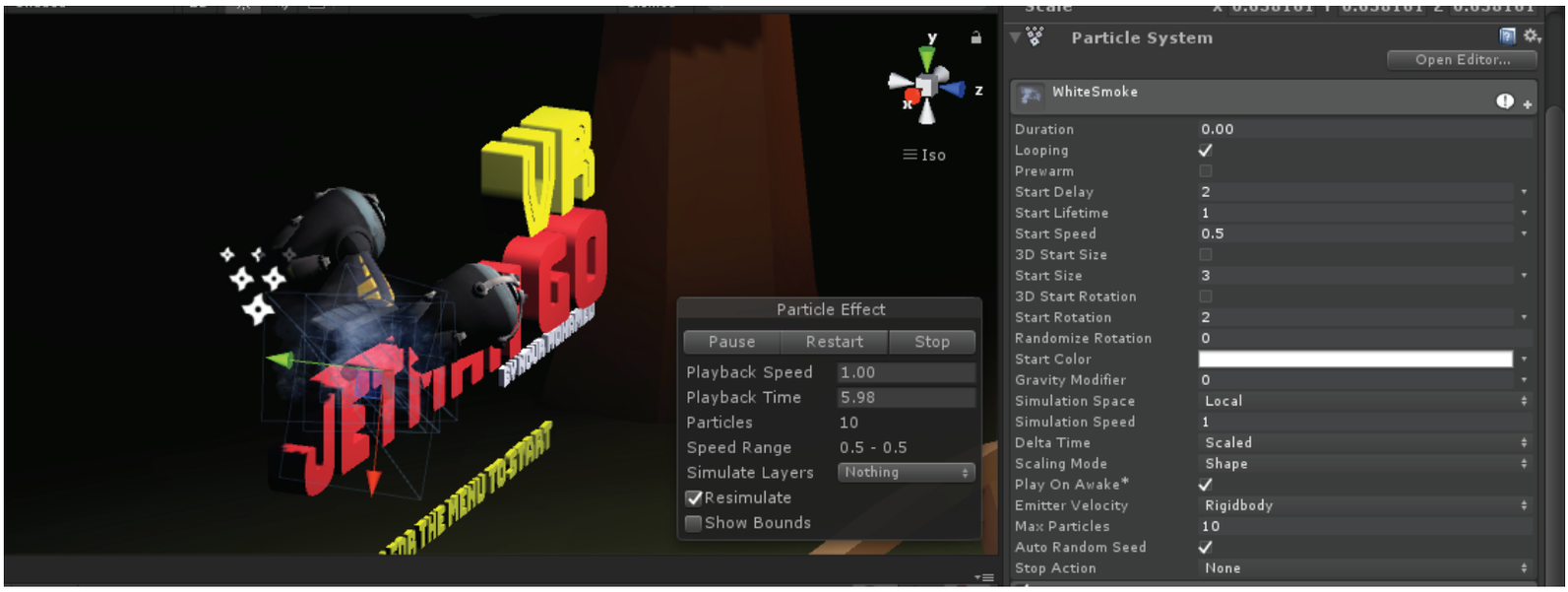


Figure 56: **Adding Smoke To The jet**

Figure 57: **configuring the Lighting**


```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Floating : MonoBehaviour {
6     public float amplitude;
7     public float speed;
8     public float tempVal;
9     public Vector3 tempPos;
10    private AudioSource _audio;
11    void Start ()
12    {
13        tempVal = transform.position.y;
14          _audio = GetComponent<AudioSource>();
15    }
16
17    void Update ()
18    {
19        if (_audio != null) {
20            if (Menu.soundon == false) {
21                _audio.Stop ();
22            }
23
24        }
25        tempPos.y = tempVal + amplitude * Mathf.Sin(speed * Time.time);
26        transform.position = tempPos;
27    }
28 }
29
```

Figure 58:**Floating Animation + checking settings before playing Sound**

The Audio Source

The Audio Source plays back an Audio Clip in the scene. The clip can be played to an audio listener or through an audio mixer.

The audio source can play any type of Audio Clip and can be configured to play these as 2D, 3D, or as a mixture (SpatialBlend).

The audio can be spread out between speakers (stereo to 7.1) (Spread) and morphed between 3D and 2D (SpatialBlend). This can be controlled over distance with falloff curves. Also, if the listener is within one or multiple Reverb Zones, reverberation is applied to the source. Individual filters can be applied to each audio source for an even richer audio experience.

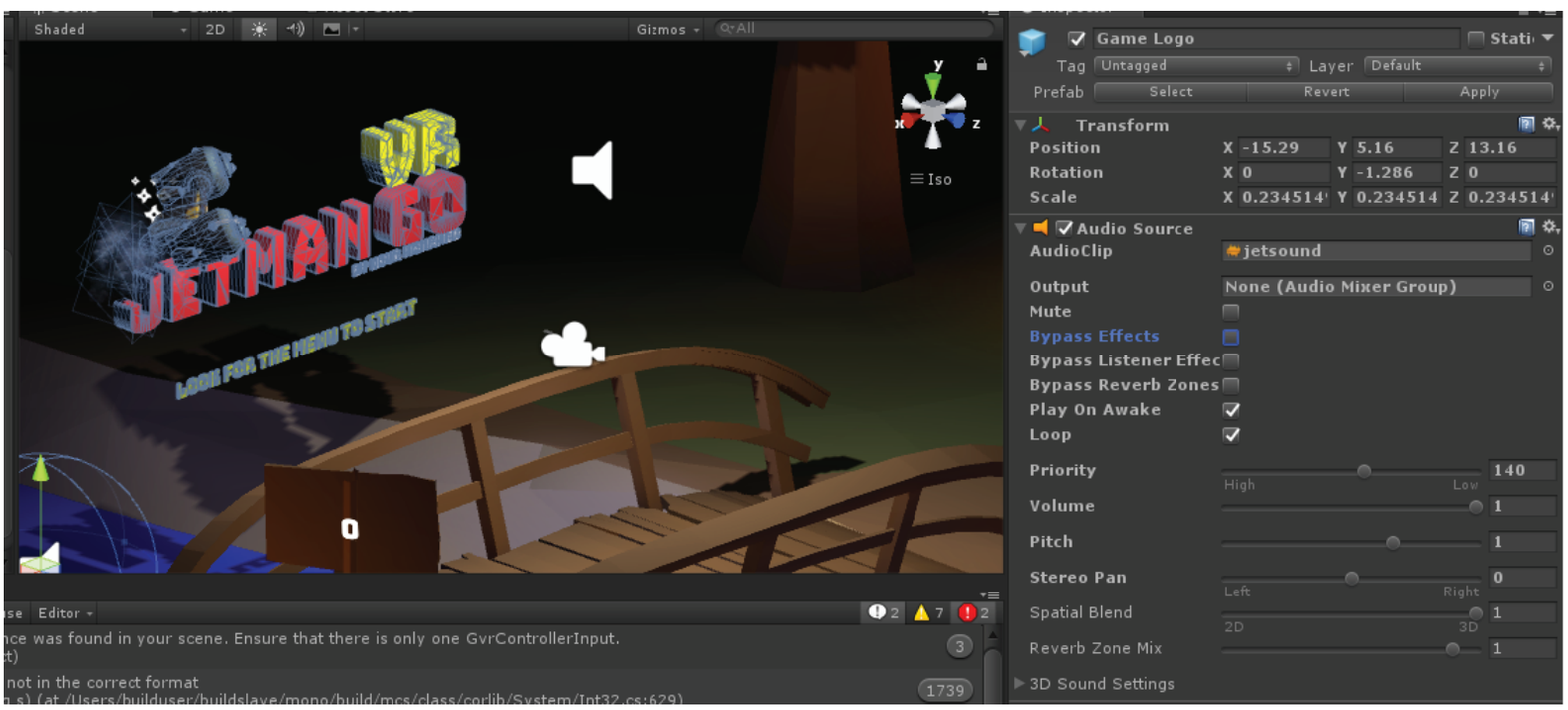


Figure 59: Adding Audio Source

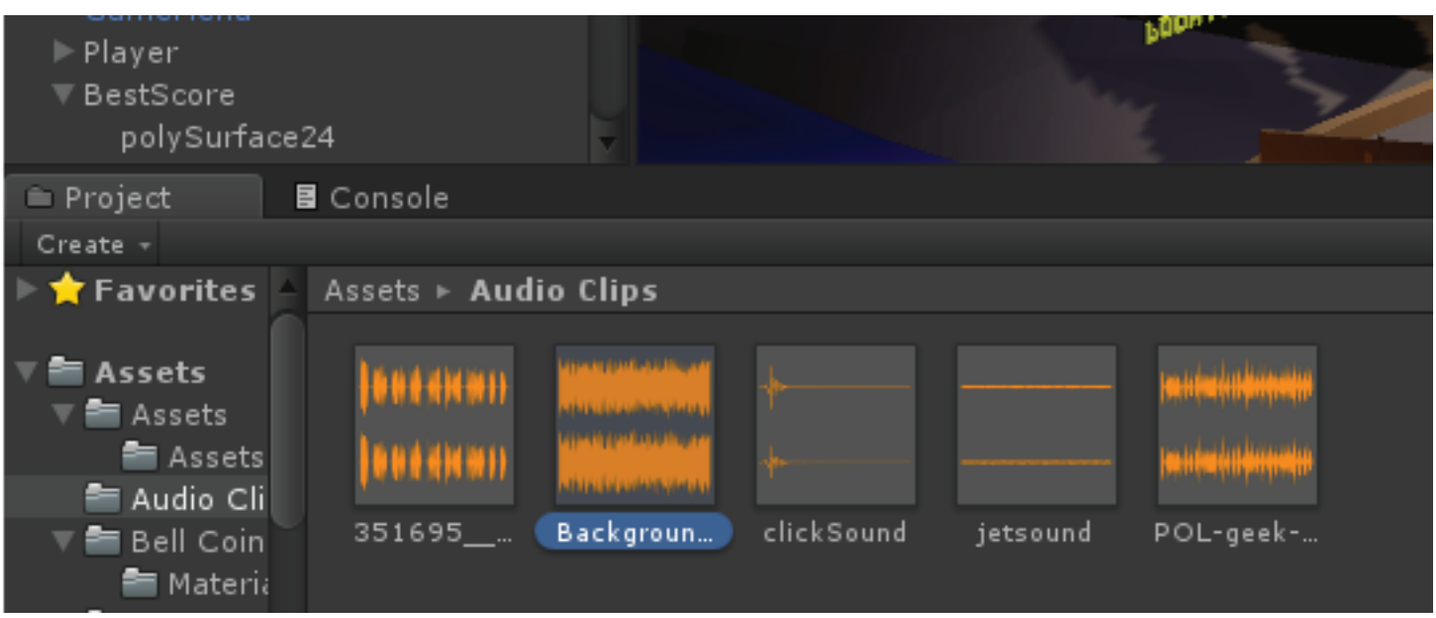
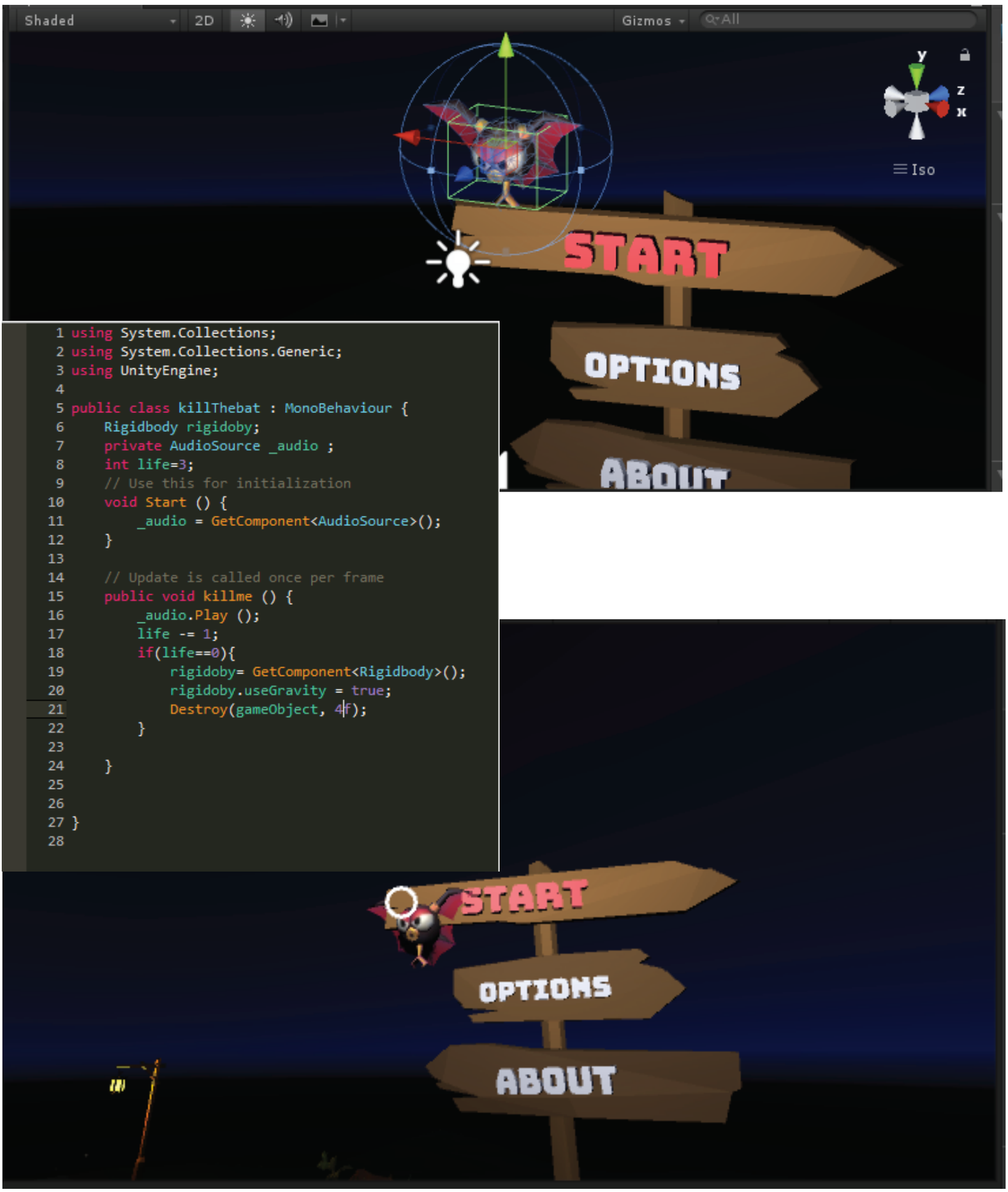


Figure 60: Adding Audio clips to the Audio Clips Folder

Funny interaction

if the player get annoyed by the angry bat he can simply end his life by clicking on him 3 times




```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class killThebat : MonoBehaviour {
    Rigidbody rigidoby;
    private AudioSource _audio ;
    int life=3;
    // Use this for initialization
    void Start () {
        _audio = GetComponent<AudioSource>();
    }

    // Update is called once per frame
    public void killme () {
        _audio.Play ();
        life -= 1;
        if(life==0){
            rigidoby= GetComponent<Rigidbody>();
            rigidoby.useGravity = true;
            Destroy(gameObject, 4f);
        }

    }


}

```

Figure 61: Killing the Angry bat just For Fun

Making sure that the scene is going to Run Smooth on Mobile

The Unity Profiler

The Unity Profiler Window helps you to optimize your game. It reports for you how much time is spent in the various areas of your game. For example, it can report the percentage of time spent rendering, animating or in your game logic.
You can analyze the performance of the GPU, CPU, memory, rendering, and audio.
To see the profiling data, you play your game in the Editor with Profiling on, and it records performance data. The Profiler window then displays the data in a timeline, so you can see the frames or areas that spike (take more time) than others. By clicking anywhere in the timeline, the bottom section of the Profiler window will display detailed information for the selected frame.

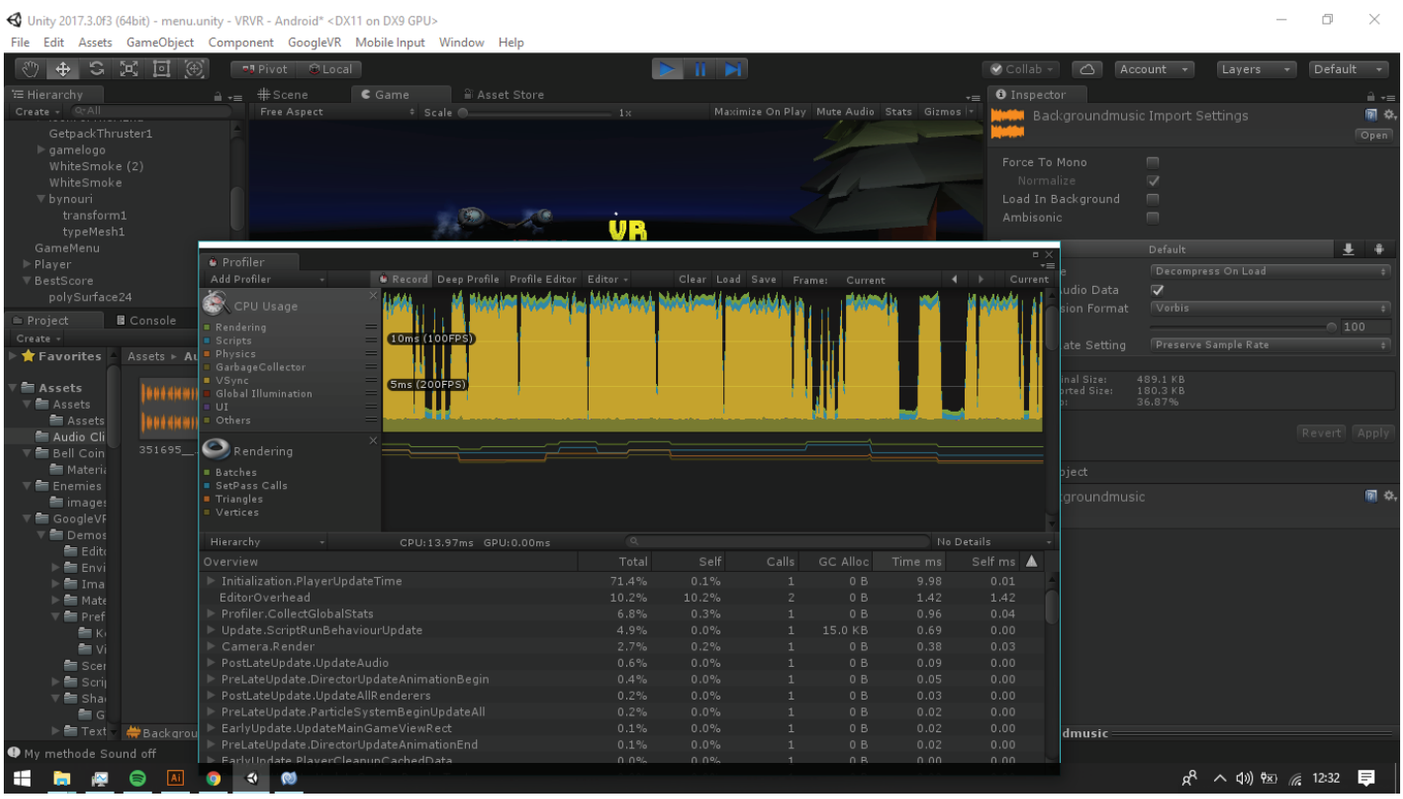


Figure 62: Using The Unity Profiler

TIME TO SEE THE RESULT

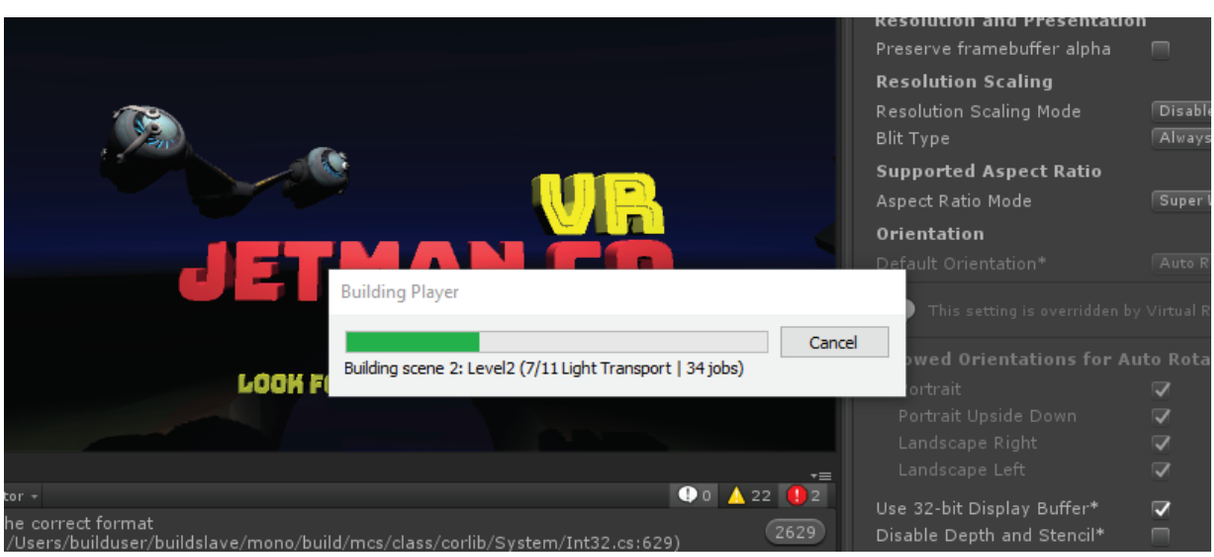


Figure 63: Building For Android For The First Time

FIRST LEVEL

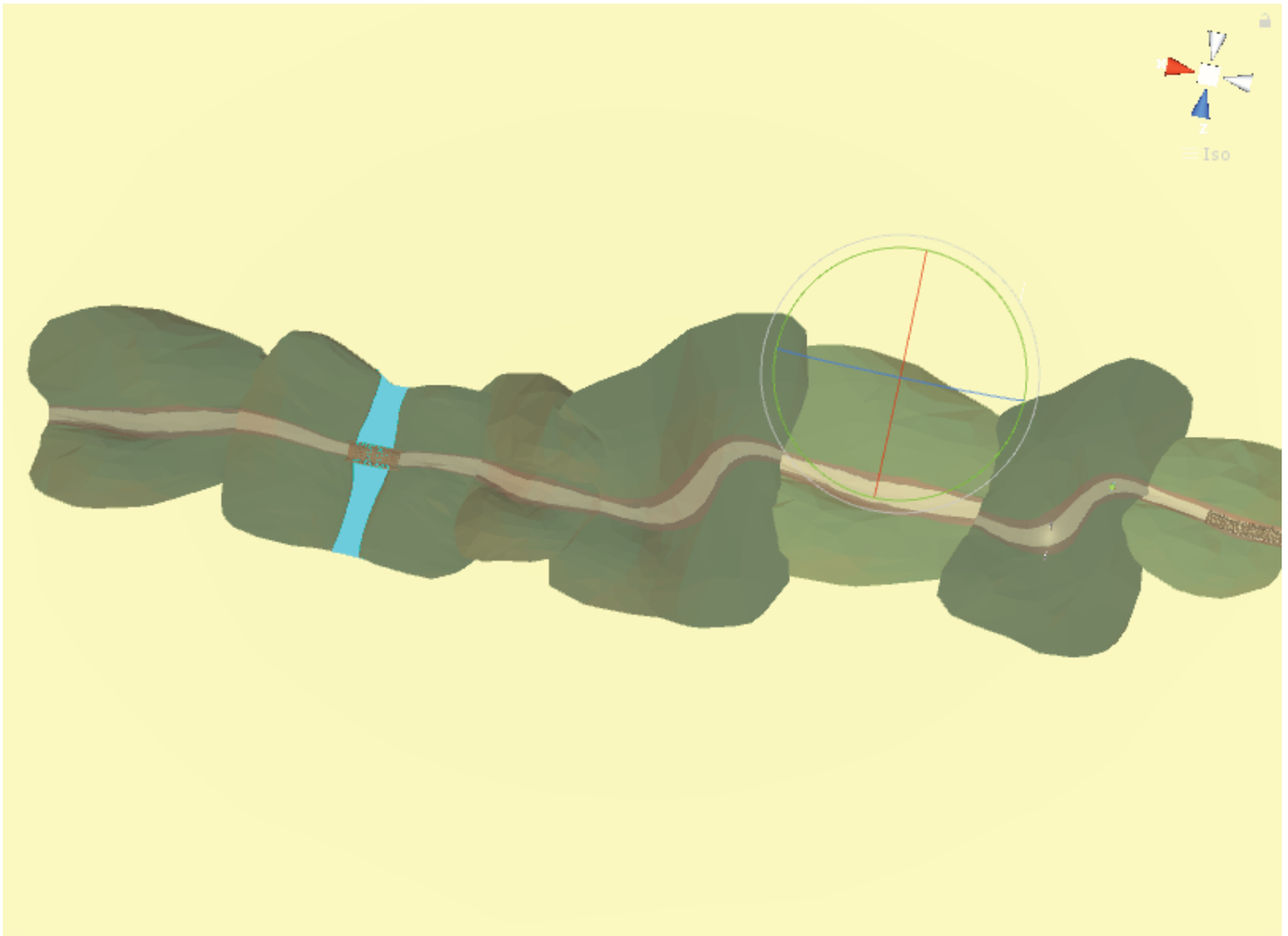


Figure 65: placing Terrains to design the level

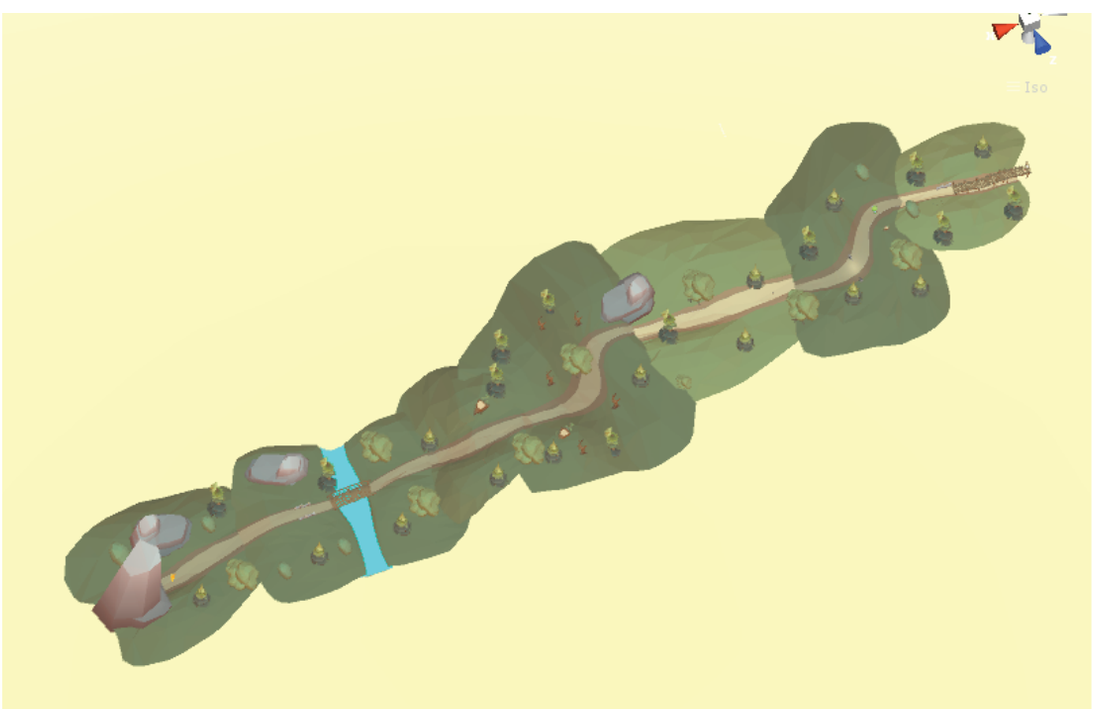


Figure 66: Adding Environment Objects

Keeping Gameobject Organised is important

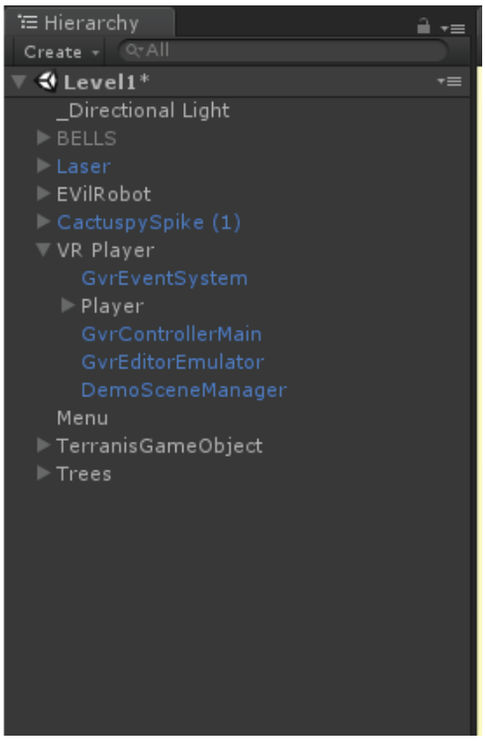


Figure **67 Scene hierarchy**

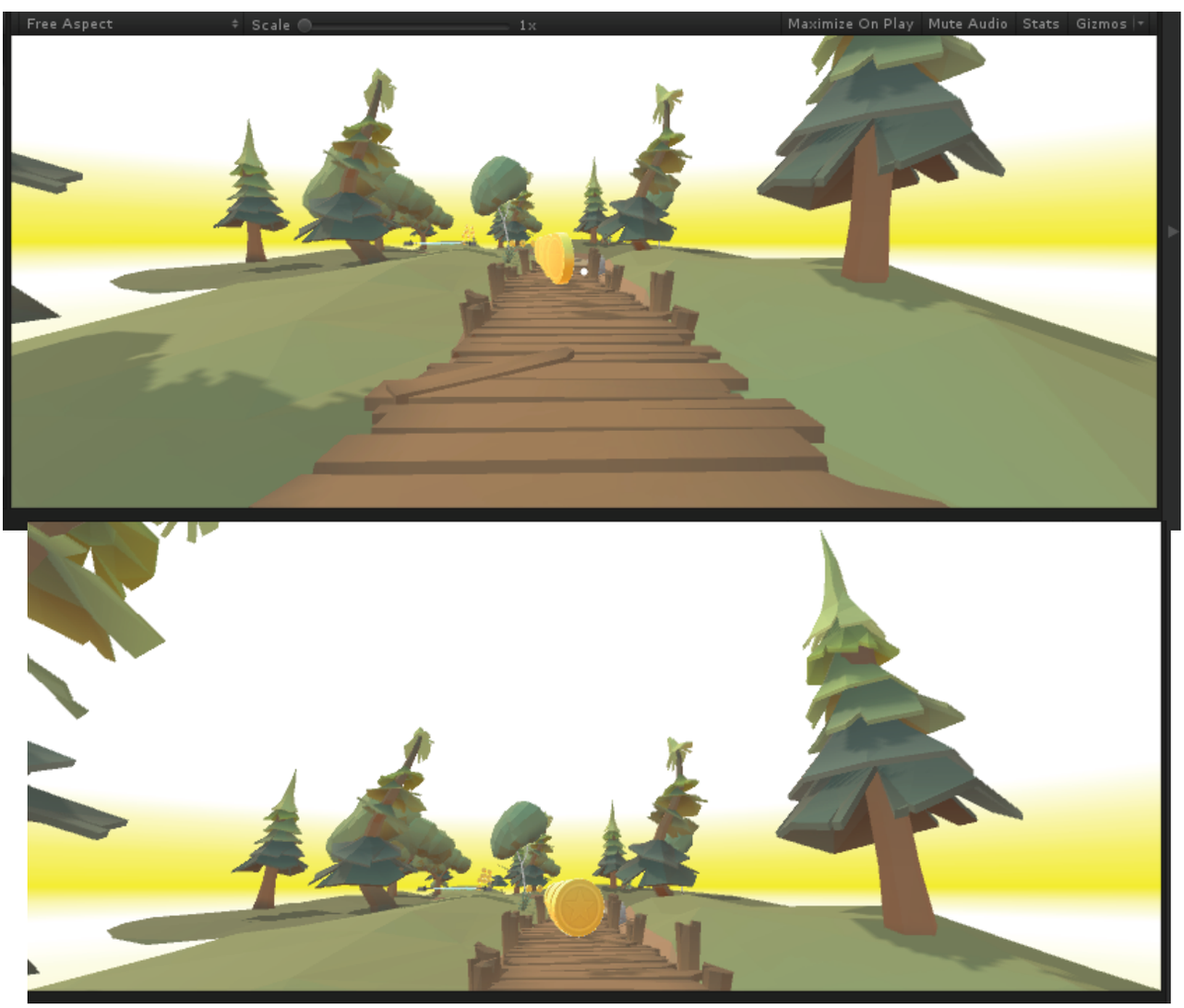


Figure **68 Scene hierarchy**

HOW TO PALY Interface for The first Time

Figure **68 : HOW To PLAY instructions UI**

```
    public float speed;
    public int forceConst = 50;
    private bool canJump;
    public GameObject LearnHwotoPlay;



    void Awake(){
        try
        {

            int firstTime = PlayerPrefs.GetInt("firstTime");
            if (firstTime != 10) {
                // No Recorder yet  See HOW to Paly "
                LearnHwotoPlay.SetActive (true);
                Debug.Log ("First time !");

            }  else{
                // No need For
                GameObject.Destroy (LearnHwotoPlay);
                Debug.Log ("Not the First time Remove for the Game !");
                startlevel();
            }
        }


        catch
        {
            Debug.Log ("ooopss !");
        }


    }
```

Figure **69 : Script "HOW To PLAY instructions "**

```
47
48      public void endFirstTime(){
49          PlayerPrefs.SetInt("firstTime", 10);
50          GameObject.Destroy (LearnHwotoPlay);
51          startlevel();
52      }
53
54
55
```

Figure **70 : Script "Methode dont show instructions UI next Time "**

Pausing The Game Is Super Easy For The Player

Figure **71 :ui "When the Player just looks at the pause button the Game will pause"**

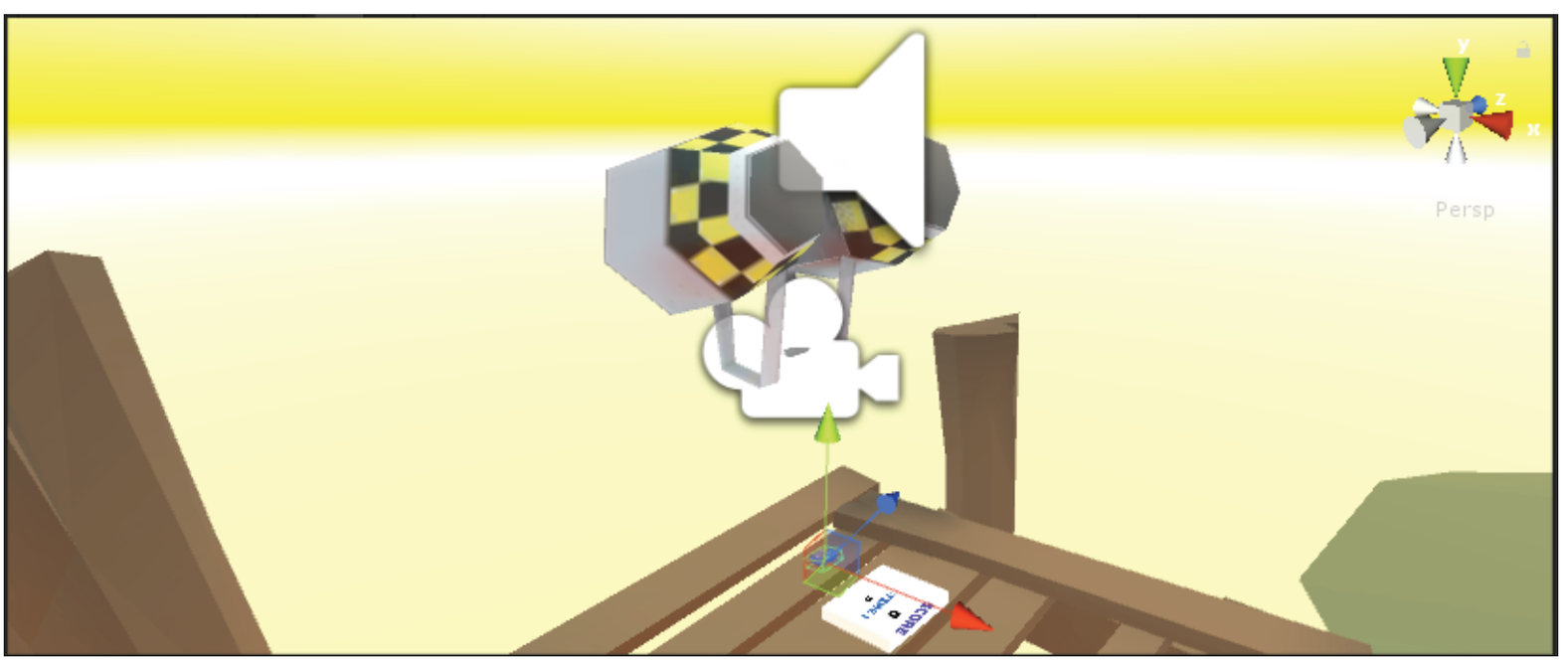


Figure **72 : Adding The Player jetpack**

Figure **73: interactive Pause ui**

```
 97
 98    public void pauseGame () {
 99        if (pauseMenu.activeInHierarchy) {
100            pauseMenu.SetActive (false);
101
102        } else {
103            pauseMenu.SetActive (true);
104
105
106        }
107        iTween.Resume ();
108        Gamepause ();
109    }
110
111
112
113
114    public void resumeGame () {
115        if (pauseMenu.activeInHierarchy) {
116            pauseMenu.SetActive (false);
117
118        } else {
119            pauseMenu.SetActive (true);
120
121            //camera.transform.Rotate(-276f * Time.deltaTime, Space.World);
122        }
123
124        iTween.Resume ();
125        Gamepresume ();
126    }
127
128
129
```

Figure **74: Script for the pause Menu GamePuse/GameResume Method**

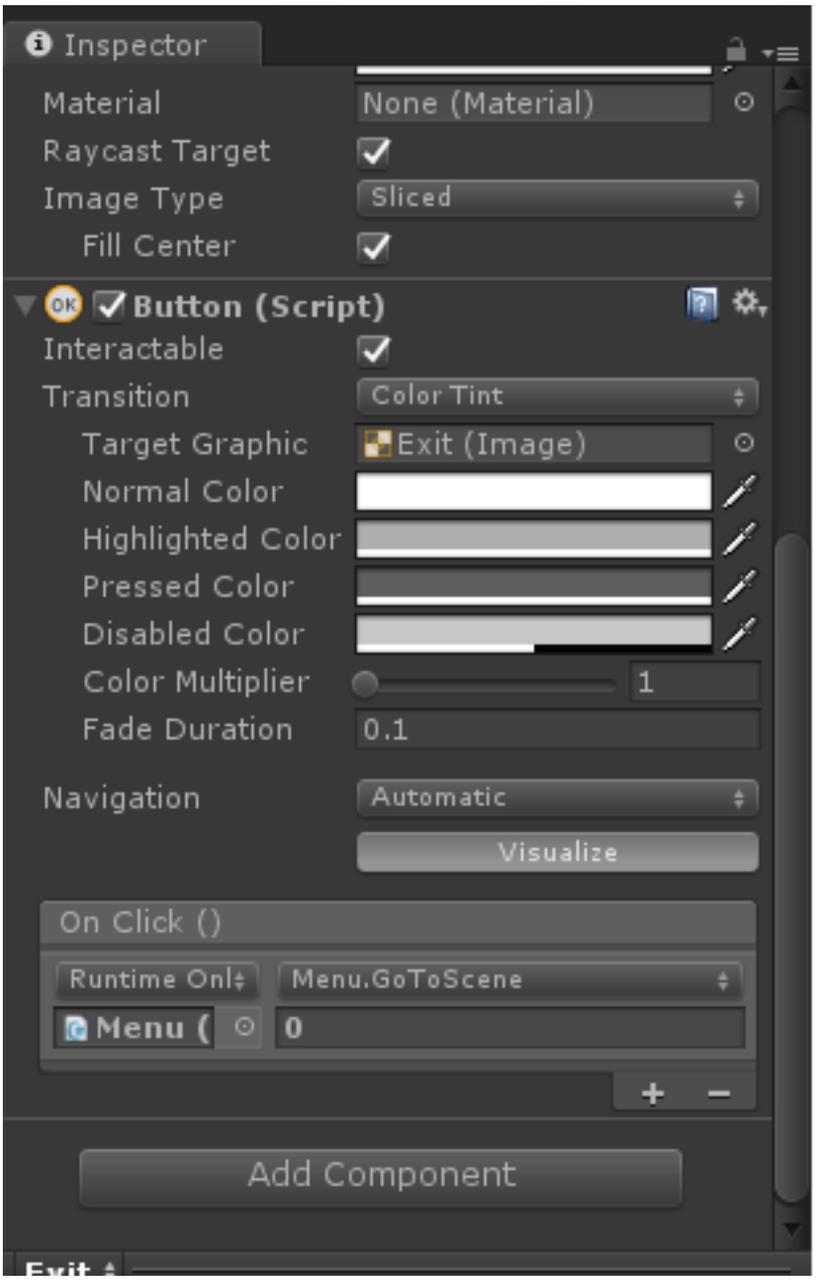


Figure 75: unity interactive Button,
HANDLING USER INPUT

Figure 76: Adding Coins

```
16
17      void Start () {
18          count=0;
19           anim = GetComponent<Animator> ();
20          _audio = GetComponent<AudioSource>();
21      }
22
23
24          void Update () {
25          transform.Rotate (Vector3.up *speed* Time.deltaTime, Space.Self);
26
27          }
28
29
```

Figure 78: Make Coins live by Rotate The object

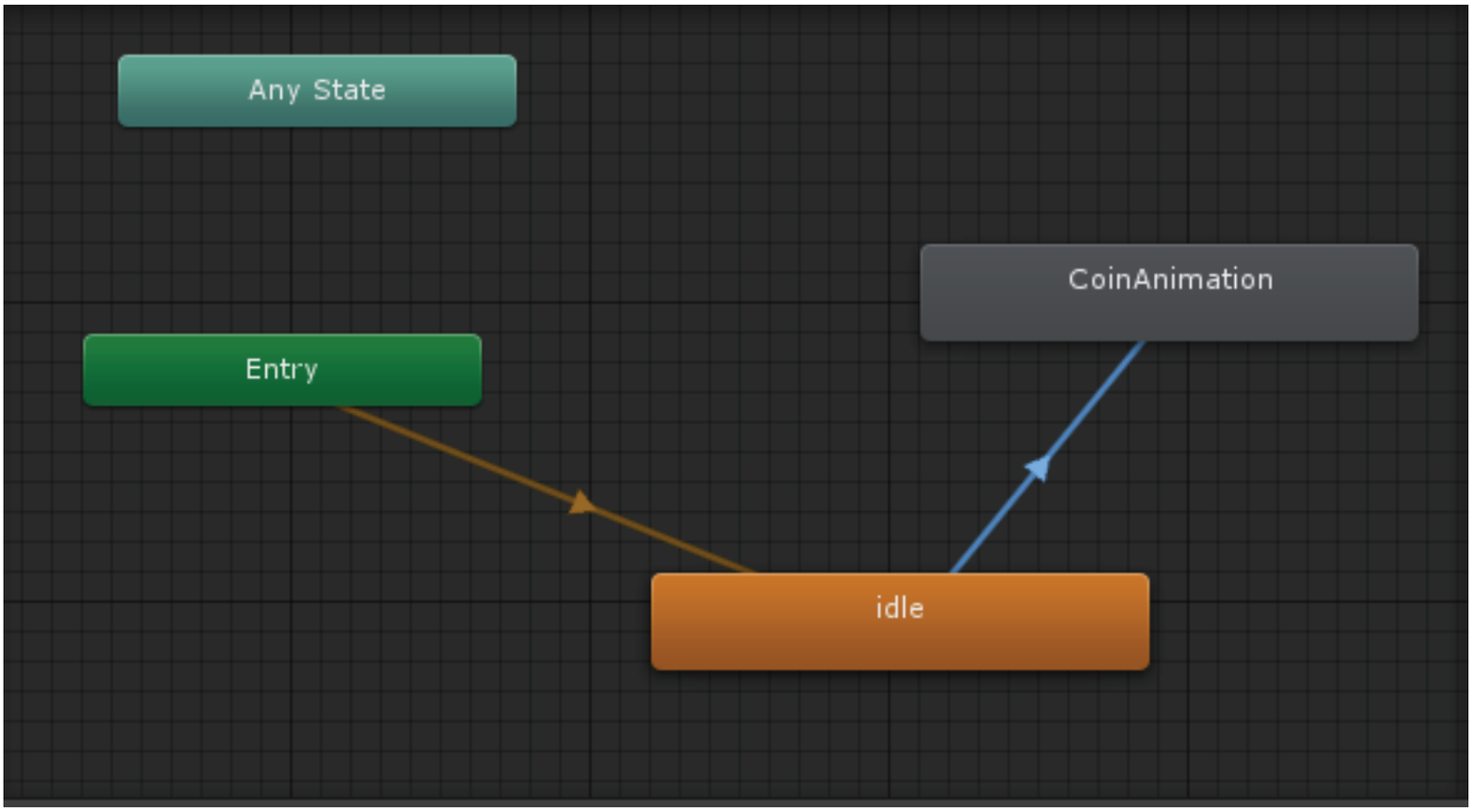


Figure 79: Make Coins live by Rotate The object

```
public void getcoin(){
    anim.SetBool ("coin",false);
    Playerscore.count += 100;
    Debug.Log (Playerscore.count);
     //countText.text =  count.ToString ();
    _audio.Play ();
     Destroy(gameObject, 0.2f);


}


public void getBiggercoin(){
    Playerscore.count += 1000;
    Debug.Log (Playerscore.count);
    //countText.text =  count.ToString ();
    // anim.SetBool ("goteaten",true);
    Destroy(gameObject, 0.5f);


}
```

Figure 80: Script when the player collects coins
the coin should play a simple animation and play a soun

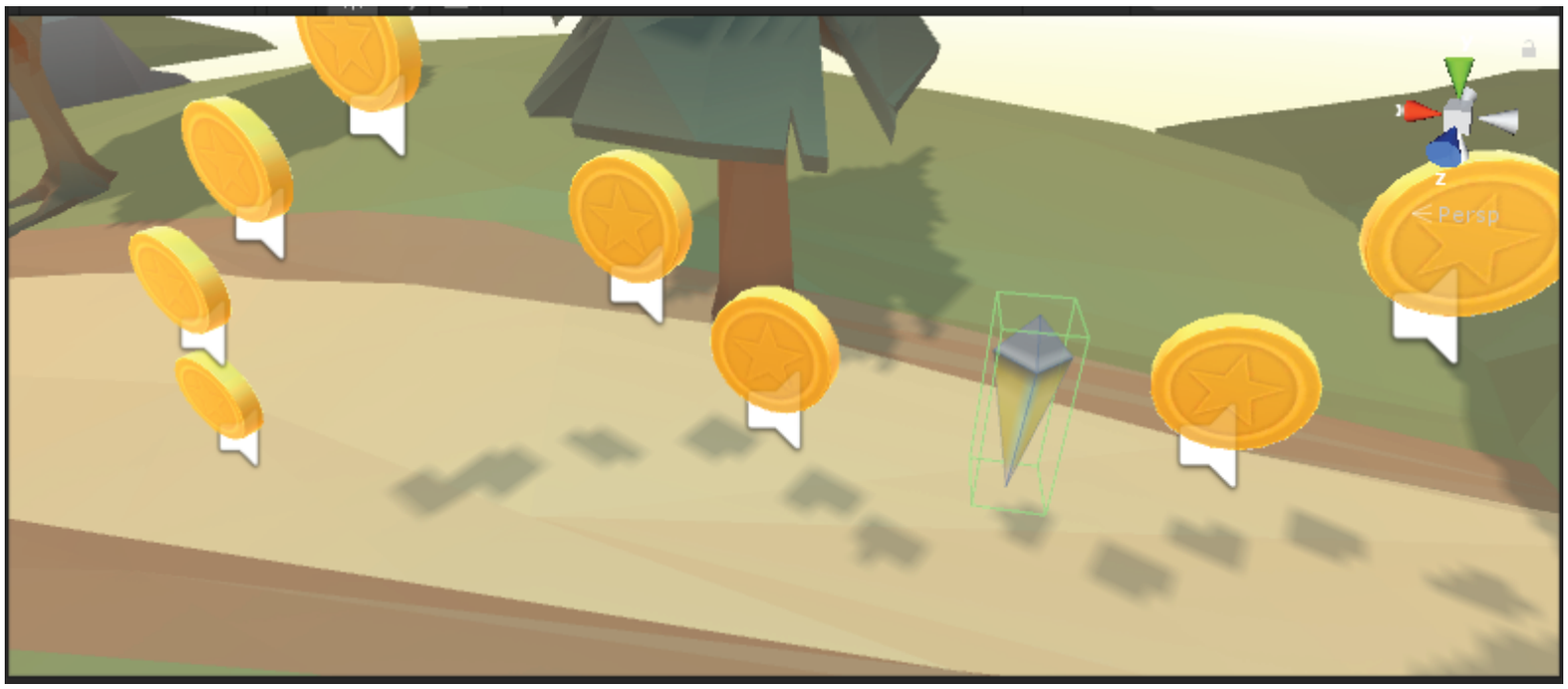


Figure 2end type of coin gives the player 1000 points

Enemies

The Rebot

Figure 81: Adding Enemies to the scene + adding colliders

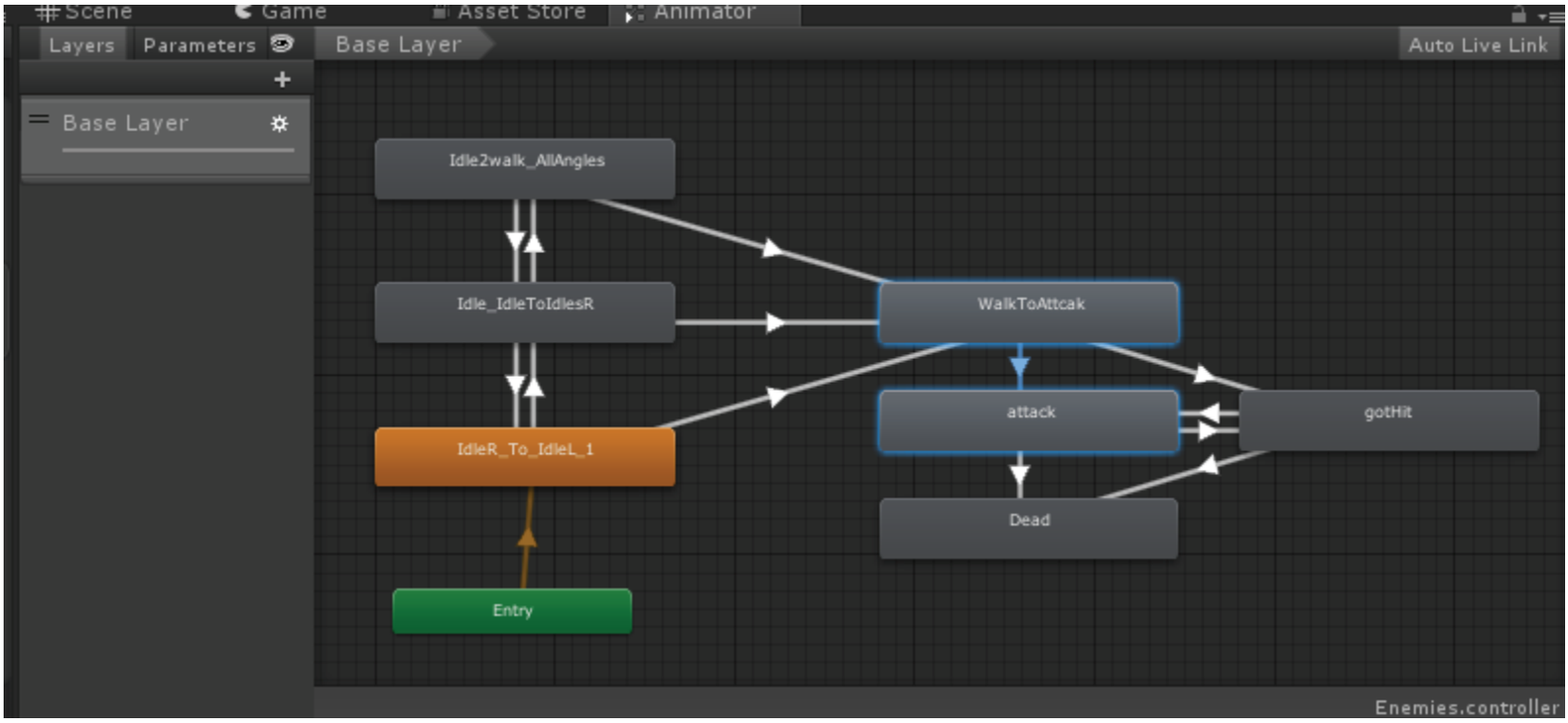


Figure 82: Adding the Robot Animator

```
24
25      void OnTriggerEnter(Collider col){
26
27          if(col.gameObject.tag=="player"){
28          Debug.Log ("Player is here");
29          wayPoint = GameObject.Find("Player");
30          animator.SetBool ("attack",true);
31          wayPointPos = new Vector3(wayPoint.transform.position.x, transform.position.y, wayPoint.transform.position.z);
32          transform.position  = Vector3.MoveTowards(transform.position, wayPointPos, speed * Time.deltaTime);
33
34          Playerscore.PlayerHealth -=1;
35          animator.SetBool ("fight",true);
36          }
37          }
38
39
40
41
42
43          public void attackEneimes(){
44          gun.Play ();
45          life -= 1;
46          animator.SetBool ("gothit",true);
47          if (life == 0) {
48              animator.SetBool ("dead",true);
49              Destroy(gameObject, 0.9f);
50          }
51
```

Figure 82: Script: attack Player with animation, and Flow him

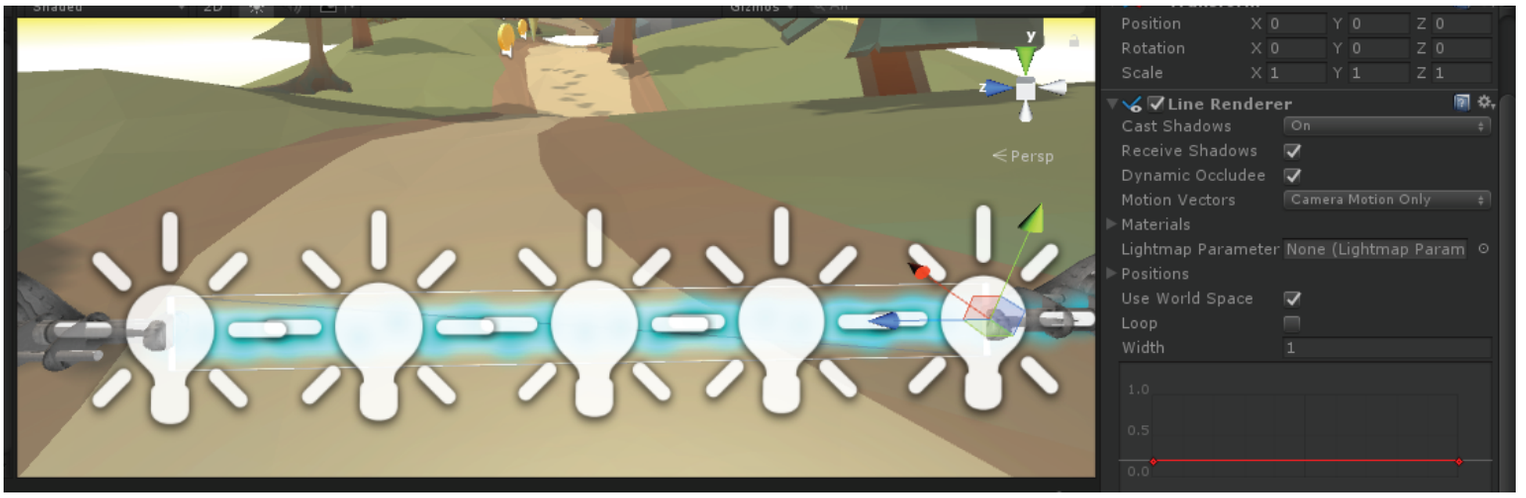
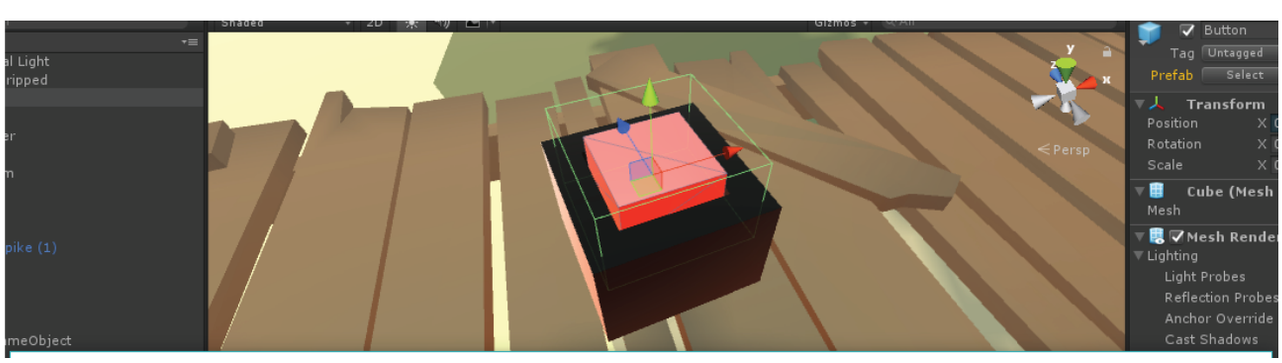


Figure 83: Adding A laser gun with Using Line Renderer

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class laserScript : MonoBehaviour {
5      public Transform startPoint;
6      public Transform endPoint;
7      LineRenderer laserLine;
8      // Use this for initialization
9      void Start () {
10         laserLine = GetComponentInChildren<LineRenderer> ();
11         laserLine.SetWidth (.2f, .2f);
12     }
13
14     // Update is called once per frame
15     void Update () {
16         laserLine.SetPosition (0, startPoint.position);
17         laserLine.SetPosition (1, endPoint.position);
18
19     }
20 }
21
```

Figure 83: Script Adding A laser gun with Using Line Renderer



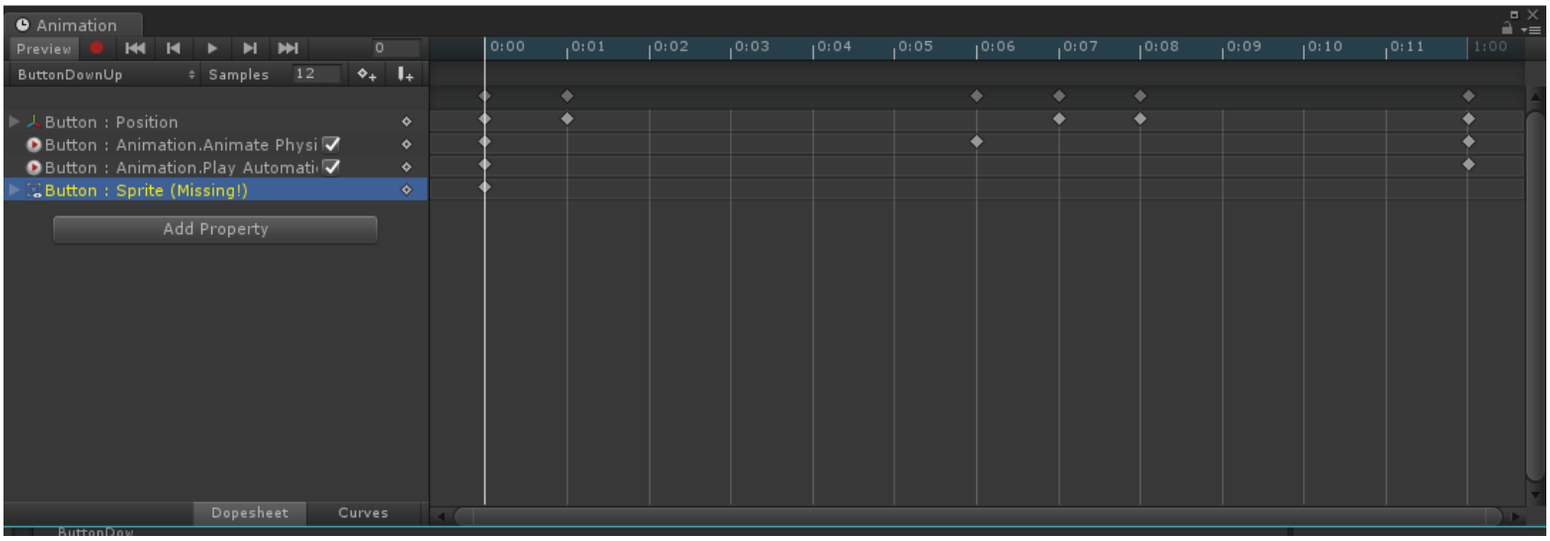


Figure 85:BUTTON "Animation phusics"

```
15      }
16
17      public  void buttonControll(){
18          anim.Play ();
19          closeLaiser ();
20
21      }
22
```

Figure 86:BUTTON Script Clode the Laiser

```
33
34      void OntriggerEnter(Collider col){
35
36          if (col.gameObject.tag == "Palyer") {
37
38              Playerscore.PlayerHealth = 0;
39          }
40
```

Figure 87:player health= 0 if he passed Laser without pressing the button

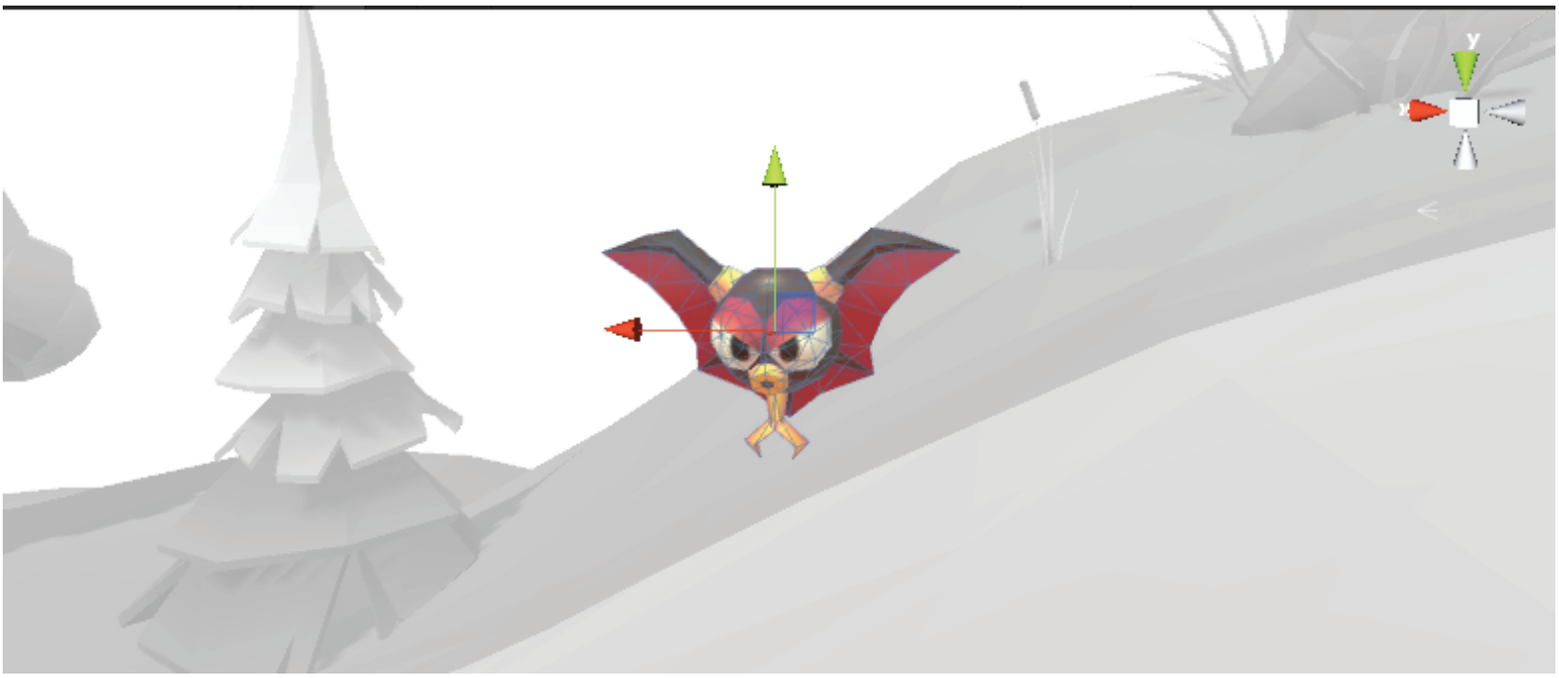


Figure 88:The Angry Bat

```
    // Update is called once per frame
    public void killme () {
        _audio.Play ();
        life -= 1;
        if(life==0){
            rigidoby= GetComponent<Rigidbody>();
            rigidoby.useGravity = true;
            Destroy(gameObject, 4f);
        }

    }

    void OnTriggerEnter(Collider col){

        if(col.gameObject.tag=="player"){
            Debug.Log ("Player is here");
            wayPoint = GameObject.Find("Player");
            animator.SetBool ("attack",true);
            wayPointPos = new Vector3(wayPoint.transform.position.x, transform.position.y, wayP
            transform.position  = Vector3.MoveTowards(transform.position, wayPointPos, speed *
            Playerscore.PlayerHealth -=1;

        }
    }
}
```

Figure 99:Script forThe Angry Bat

THE END POINT

Figure 89: the End Point The Player needs to click to go to the next Level

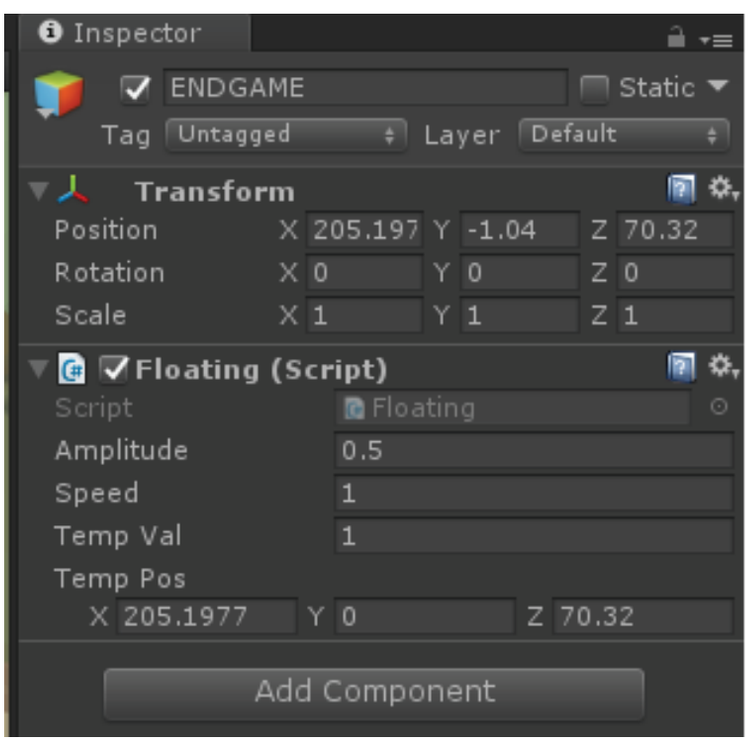


Figure 90: Adding Floating Animation

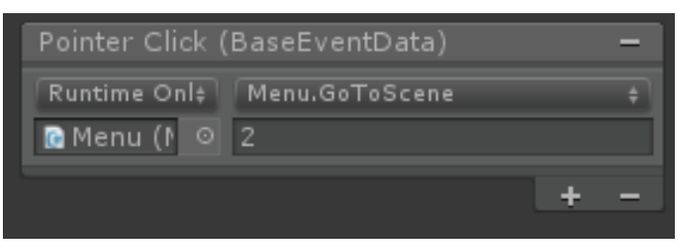


Figure 91 :
Loading Level Number 2

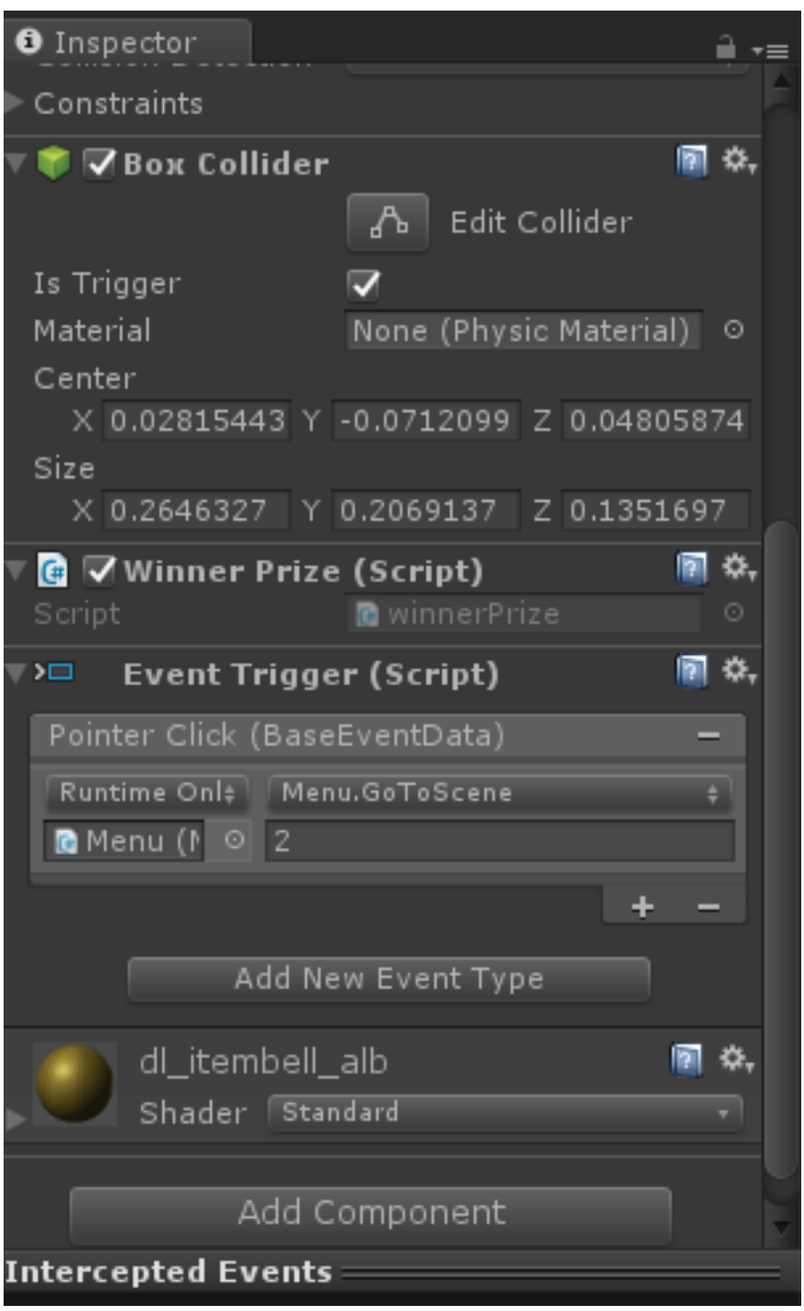


Figure 92: Adding A box Collider

Controlling The Player

Using iTween

iTween can be used with all scripting languages that Unity supports including JavaScript, C# and Boo. iTween offers the ability to create animations with full control over things such as delay, looping, callback functions and much more. To use iTween's more robust features you must provide the parameters you would like in the form of a Hashtable.

Figure 93: Adding VISUAL ITWEEN PATH EDITOR

```
    // Sound Of the Jet
    _audioMoving=GetComponent<AudioSource>();

    iTween.MoveTo(gameObject, iTween.Hash ("path", iTweenPath.GetPath ("playerpath"), "time", speed,"easetype",iTween.EaseType.easeInOutSine));

//iTween.Pause ();
```

Figure 94: itween Script

```
63
64
65      void playermove(){
66
67
68           iTween.Resume ();
69          _audioMoving.Play ();
70
71
72      }
73
74      void playerStop(){
75
76
77          iTween.Pause ();
78          _audioMoving.Stop ();
79
80
81      }
82
83
```

Figure 95 Script player Move/ Player Stop

```
    void SetCountText ()
    {
    //  countText.text = "Count: " + count.ToString ();
        string Score =PlayerPrefs.GetString("Score");
        int BestScore =System.Int32.Parse(Score);
        if (count >=BestScore)
        {

            Debug.Log ("You Cracked the Best Scoor Congrats ");
            PlayerPrefs.SetString ("Score",count.ToString ());
        }
    }


}
```

Figure 96: Updating the Player max Score

```
void Update(){

    if (PlayerHealth==0){
        PlayerPrefs.SetInt("lastScene",0);
        SceneManager.LoadScene( SceneManager.GetActiveScene().name );
    }

    SetCountText ();
}
```

Figure 96: SCRITP IF THE PLAYER HEALTH+0 BACK TO THE FIRST SCENE

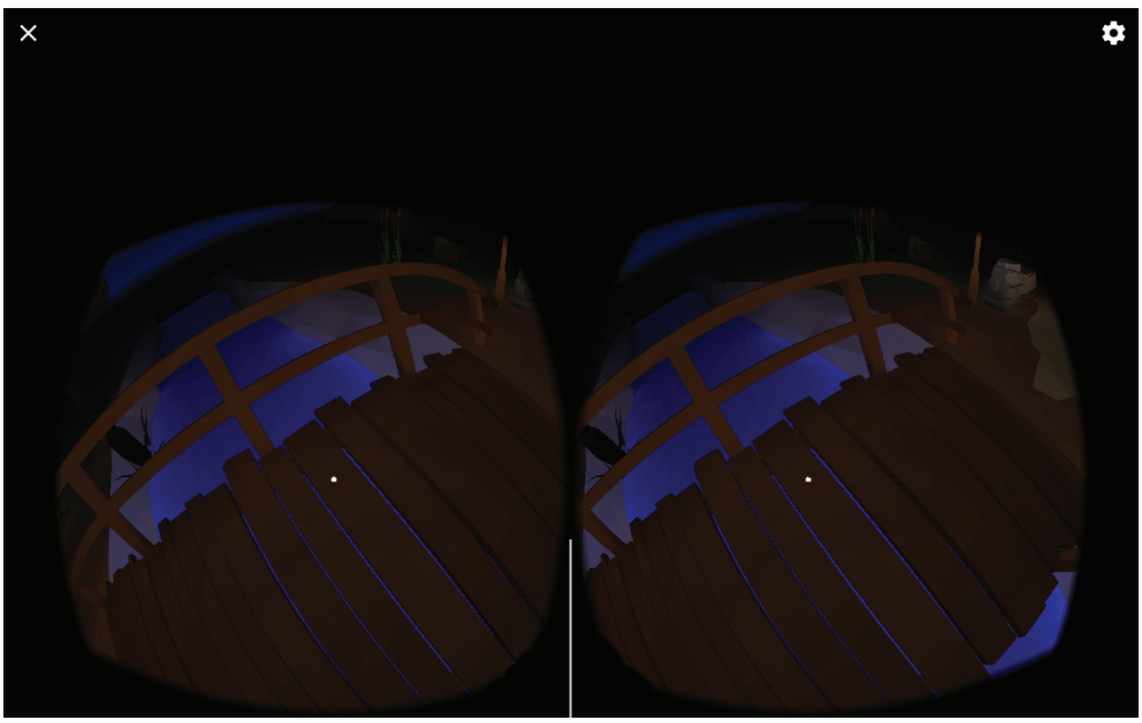
```
void OnTriggerEnter(Collider col){



        Debug.Log ("Your Are A Winner");
        PlayerPrefs.SetInt("lastScene",2);

    }
```

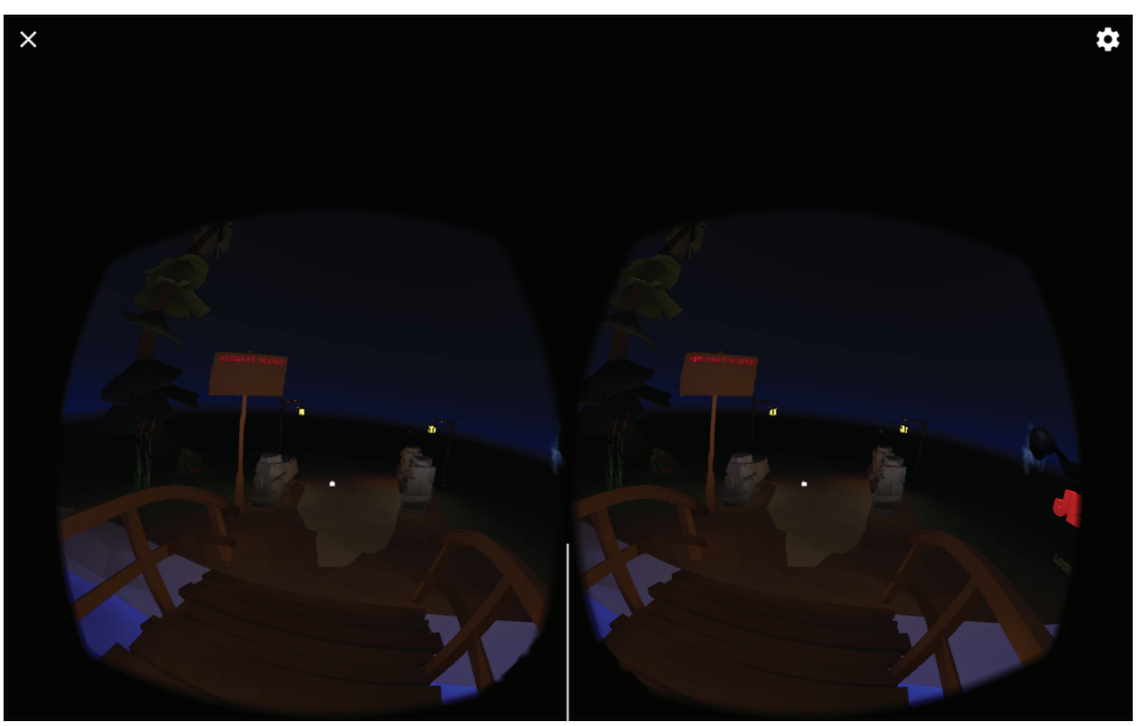
Figure 97 : SCRITP for saving last scene

```
void Awake(){

    try{
        int lastlevel= PlayerPrefs.GetInt("lastScene");
        if(lastlevel==2){
        START.SetActive (false);
        RESUME.SetActive (true);
        }


    }catch{
        START.SetActive (true);
        RESUME.SetActive (false);

    }

}
```

Figure 98: SCRITP TEST IF THE PLAYER SHOULD START FORM LEVEL 1 OR 2

THE END RESULT





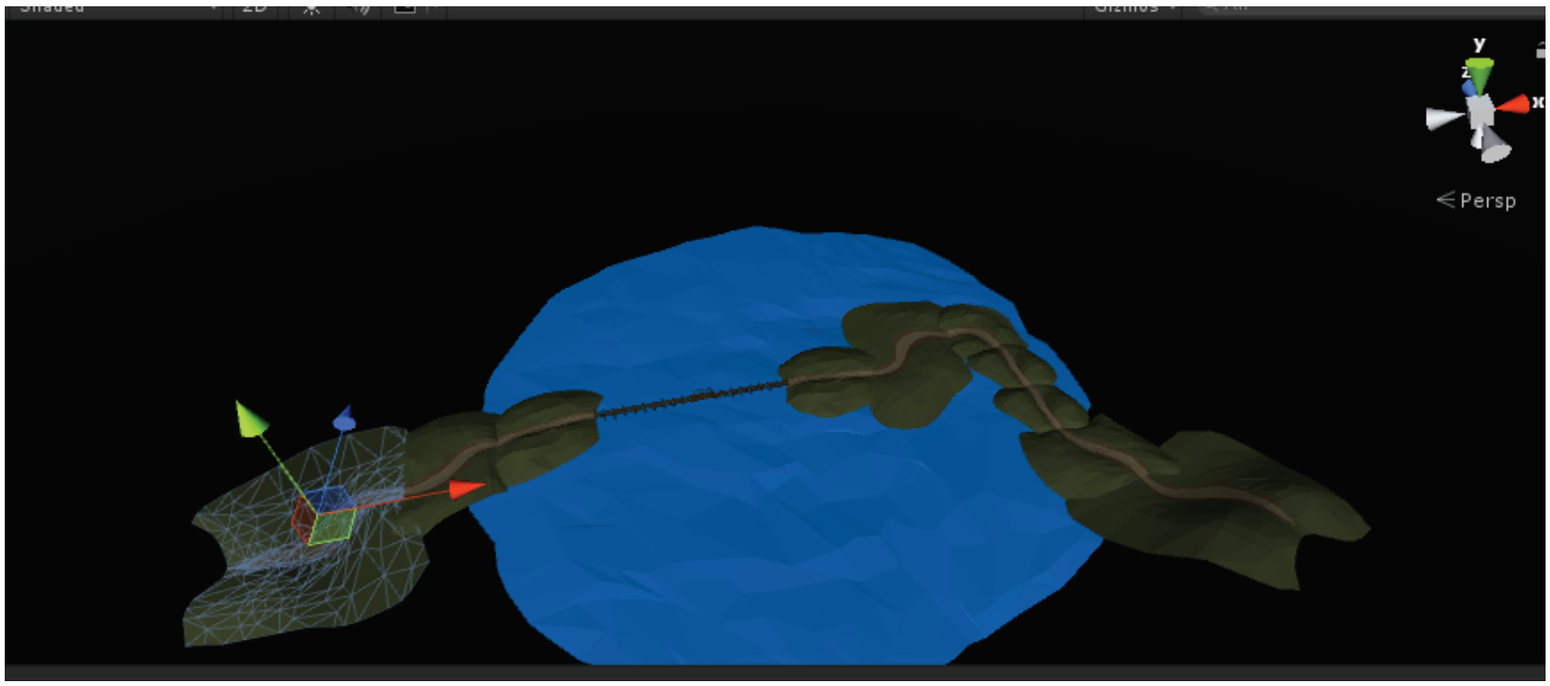


Figure 100:**adding Terrrains**

Figure 101:**adding Game Objects**

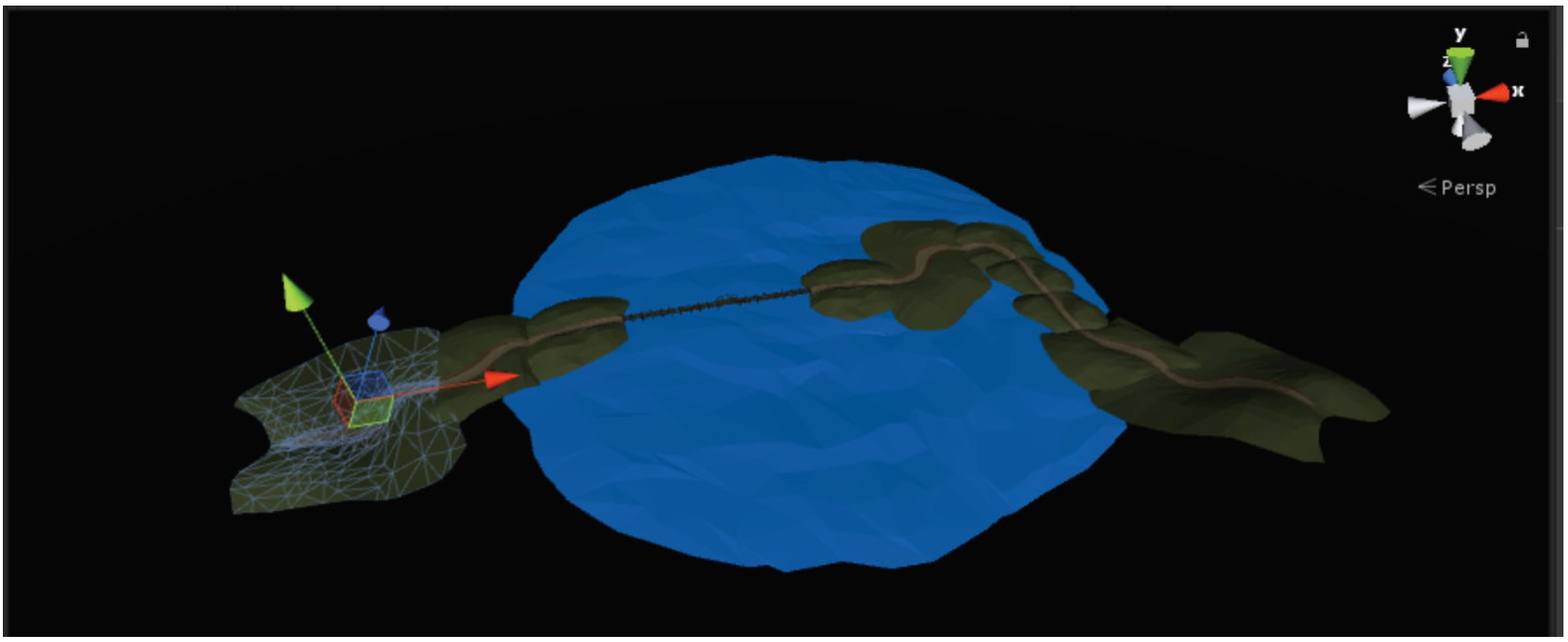


Figure 102:**adding Terrrains**

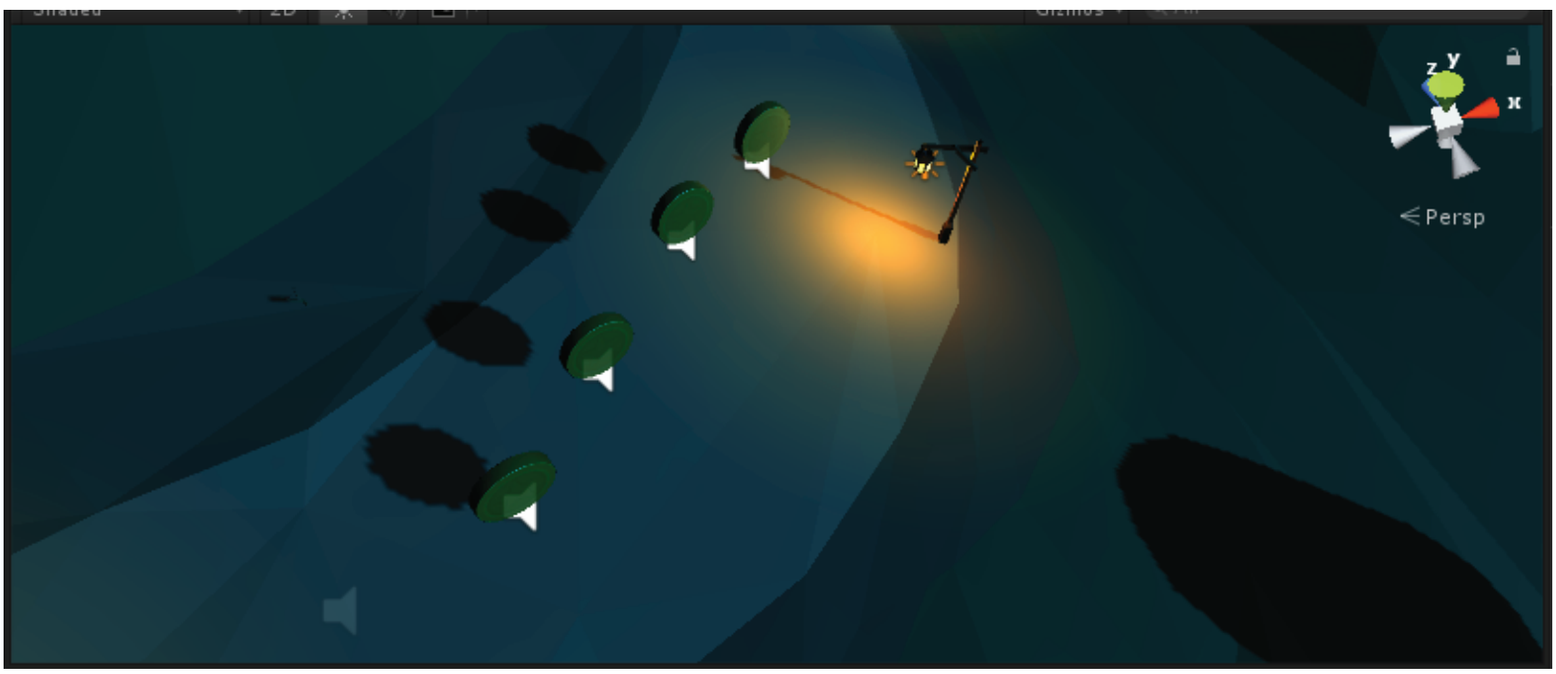


Figure 103:**Adding Coins**

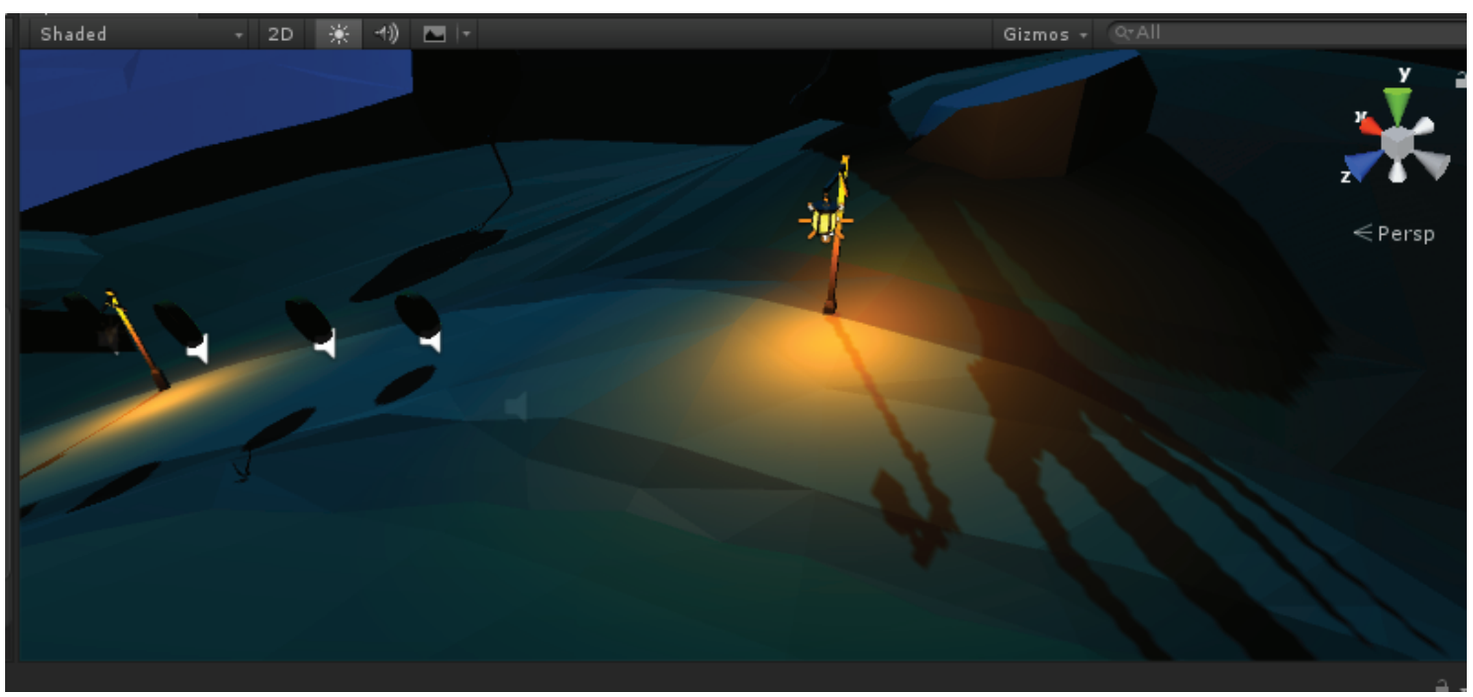


Figure 103:**Adding Lights**

ANDROID LOGO

188*188

80*80


THE END
THANKS