How to survive in a barren wasteland in a supernatural setting: the game

How to survive in a barren wasteland in a supernatural setting: the game

---

COSC 729

Group Members: Andrew Benya and Aishwarya Reehl

Faculty: Dr. Sharad Sharma

Bowie State University

How to survive in a barren wasteland in a supernatural setting: the game

Overview

The goal of our project is to create a zombie game where the player has to escape the zombie attack and try to reach the helicopter to win.

Health bar is used to track if the player has been attacked by the zombie, if so then the health bar reduces.

In case the player gets stuck he can press T to get evacuated.

This paper contains the details about how the environment is defined and the technologies used in creating this game. As the game designed involves a lot of hypothetical situations, virtual reality is the best option to design it. This game is extremely fun to play and is definitely a stress buster. The purpose of this game is to enjoy the game and have fun.

Environment

The environment resembles a haunted place. The environment is mostly wilderness and it includes mountains, trees, sea and dust storms to make it look more like a haunted place. The abandoned houses have been added to create a feel in the game. Terrain was “painted” onto the plane with a preview package. This package allows us to use painting tools to elevate and depress the ground, apply textures with discretion, and paint any prefab tree available in the scene.

Vision

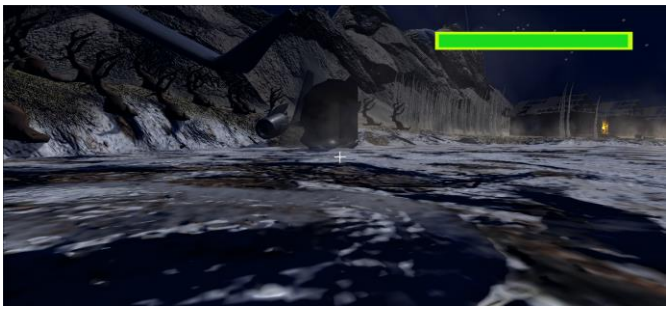


Figure 1 Environment showing the dry lake

Figure 2 Terrain with water, haunted house and zombies

How to survive in a barren wasteland in a supernatural setting: the game

Figure 3 Terrain with billboard, house and zombies

Figure 4 Scary trees in the terrain

For sky we added the night sky with stars to create the perfect environment. A scary music is added throughout the game and helicopter hovering sounds for evacuation.

Inputs

Player can use keyboard to control the navigation. The player can use A to turn left and D to turn right or the Left Arrow and Right Arrow keys. The player can use W to go up and S to go down or the Up Arrow and Down Arrow keys.

Sensors

Several proximity sensors have been used to detect if the player is going near the zombie. There is time sensor for evacuation and touch sensor for radio.

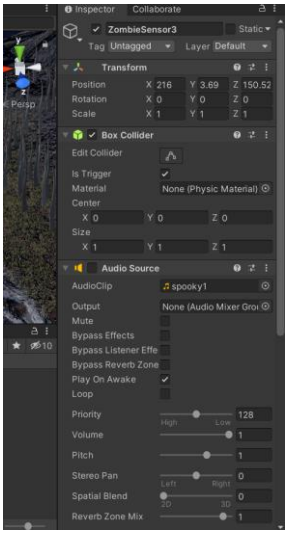


Figure 5 Zombie proximity sensor

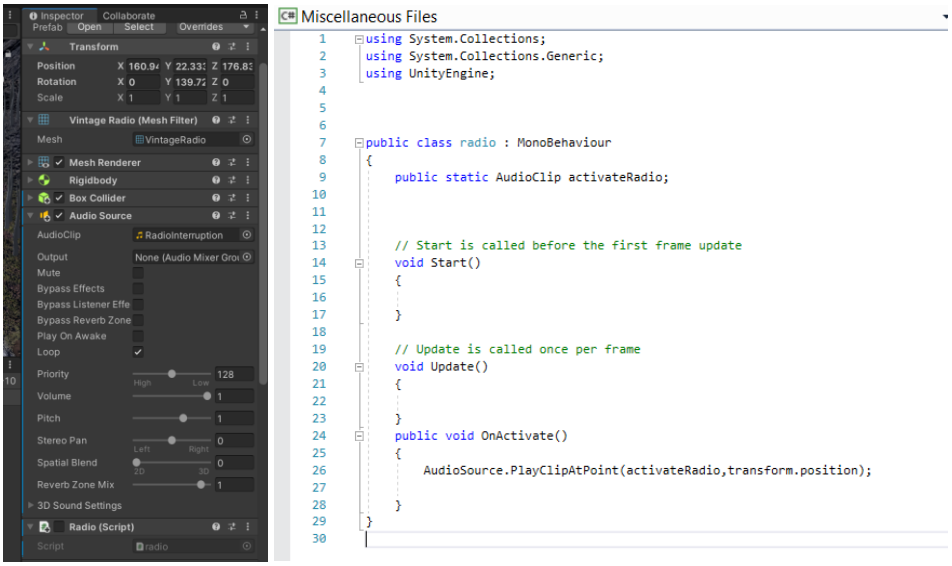


Figure 6 Radio sensor and script

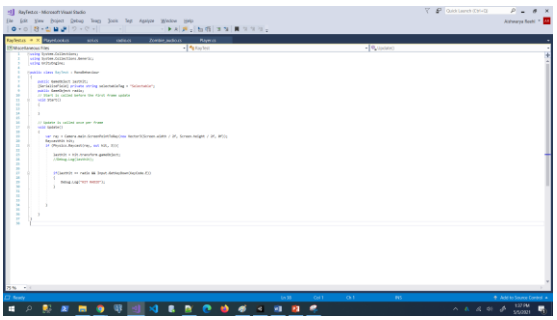


Figure 7 Hit radio script

How to survive in a barren wasteland in a supernatural setting: the game

Animation and scripting

The zombies are animated using custom and predefined scripts. Script to make the zombie look at the player as shown in Figure 8.

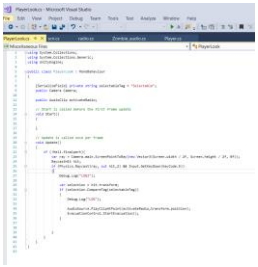


Figure 8 Script to make zombie look at the player

Zombie fall animation at the start of the game:

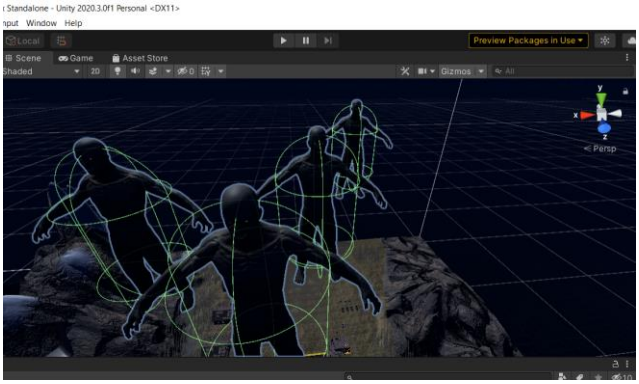


Figure 9 Zombies falling at the start of the game

Helicopter navigation using animation:

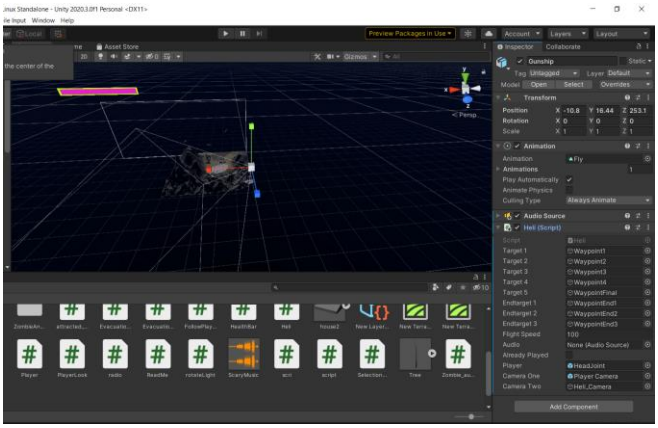


Figure 10 Helicopter navigation animation

Zombie walking animation is as shown in Figure 11:

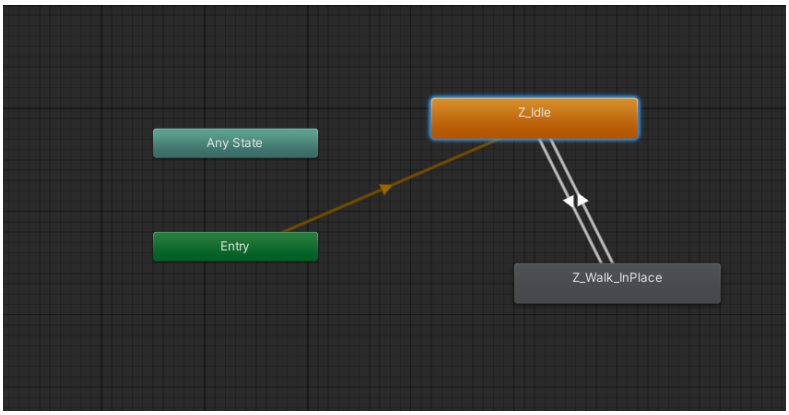


Figure 11 Zombie animation to walk

Zombie various other animations are as shown in Figure 12:

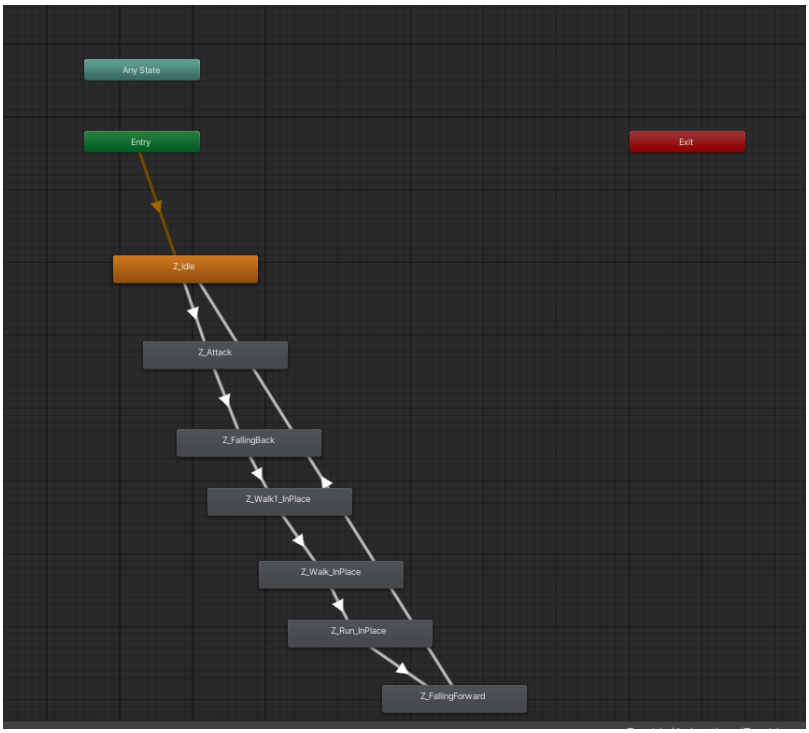


Figure 12 Zombie various animation

Health bar script:

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

public class Player : MonoBehaviour
{
    public int maxHealth = 1000;
    public int currentHealth;

    public HealthBar healthBar;
    // Start is called before the first frame update
    void Start()
    {
        currentHealth = maxHealth;
        healthBar.SetMaxHealth(maxHealth);
    }

    // Update is called once per frame
    public void Update()
    {
        //TakeDamage(20);
        if(currentHealth <= 0)
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        }
    }

    public void TakeDamage(int damage)
    {
        currentHealth -= damage;

        healthBar.SetHealth(currentHealth);
    }
}
```

Figure 13 Health bar manipulation script

Rotate light script to follow the player:

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

public class rotateLight : MonoBehaviour
{
    public float rotateSpeed = 2f;
    Transform transform;
    // Start is called before the first frame update
    void Start()
    {
        transform = GetComponent<Transform>();
    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(Vector3.down * Time.deltaTime * 100);
    }
}
```

Figure 14 Rotate light script

Zombie audio script that is triggered with proximity:

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

public class Zombie_audio : MonoBehaviour
{
    Transform playerTransform;
    AudioSource audioSource;
    public GameObject player;
    //bool sound_play;
    //bool toggle;
    // Start is called before the first frame update
    void Start()
    {
        audioSource = GetComponent<AudioSource>();
        //sound_play = false;
    }

    // Update is called once per frame
    void Update()
    {
        playerTransform = player.transform;
        var distance = Vector3.Distance(playerTransform.position, transform.position);
        if(distance <= 10.0f)
            audioSource.Play();
        else
        {
            audioSource.Stop();
        }
    }
}
```

Figure 15 Zombie audio script

Script to control the zombie's movement thc6:

```
using UnityEngine;
using System.Collections;

public class THC6_ctrl : MonoBehaviour {


    private Animator anim;
    private CharacterController controller;
    private int battle_state = 0;
    public float speed = 6.0f;
    public float runSpeed = 3.0f;
    public float turnSpeed = 60.0f;
    public float gravity = 20.0f;
    private Vector3 moveDirection = Vector3.zero;
    private float w_sp = 0.0f;
    private float r_sp = 0.0f;


    // Use this for initialization
    void Start ()
    {
        anim = GetComponent<Animator>();
        controller = GetComponent<CharacterController> ();
        w_sp = speed; //read walk speed
        r_sp = runSpeed; //read run speed
        battle_state = 0;
        runSpeed = 1;

    }

    // Update is called once per frame
    void Update ()
    {
        if (Input.GetKey ("1"))  // turn to still state
        {
            anim.SetInteger ("battle", 0);
            battle_state = 0;
            runSpeed = 1;
        }
        if (Input.GetKey ("2")) // turn to battle state
        {
            anim.SetInteger ("battle", 1);
            battle_state = 1;
            runSpeed = r_sp;
        }

        if (Input.GetKey ("up"))
        {
            anim.SetInteger ("moving", 1);//walk/run/moving
        }
        else
            {
                anim.SetInteger ("moving", 0);
            }


        if (Input.GetKey ("down")) //walkback
        {
            anim.SetInteger ("moving", 12);
            runSpeed = 1;
        }
```

```
        if (Input.GetKeyUp ("down"))
        {
            if (battle_state == 0) runSpeed = 1;
            else if (battle_state >0) runSpeed = r_sp;
        }

        if (Input.GetMouseButtonDown (0)) { // attack1
            anim.SetInteger ("moving", 2);
        }
        if (Input.GetMouseButtonDown (1)) { // attack2
            anim.SetInteger ("moving", 3);
        }
        if (Input.GetMouseButtonDown (2)) { // attack3
            anim.SetInteger ("moving", 4);
        }
        if (Input.GetKeyDown ("space")) { //jump
            anim.SetInteger ("moving", 6);
        }
        if (Input.GetKeyDown ("c")) { //roar/howl
            anim.SetInteger ("moving", 7);
        }
        if (Input.GetKeyDown ("y")) { //powerhit
            anim.SetInteger ("battle", 1);
            battle_state = 1;
            runSpeed = r_sp;
            anim.SetInteger ("moving", 5);
        }


        if (Input.GetKeyDown ("u")) //hit
        {
            battle_state = 1;
            runSpeed = r_sp;
            anim.SetInteger ("battle", 1);

            int n = Random.Range (0, 2);
            if (n == 1)
                {
                    anim.SetInteger ("moving", 8);
                }
            else
                {
                    anim.SetInteger ("moving", 9);
                }
        }

        if (Input.GetKeyDown ("k")) { //rising
            anim.SetInteger ("battle", 1);
            battle_state = 1;
            runSpeed = r_sp;
            anim.SetInteger ("moving", 15);
        }


        if (Input.GetKeyDown ("i")) anim.SetInteger ("moving", 13); //die/fall
        if (Input.GetKeyDown ("o")) anim.SetInteger ("moving", 14); //die2


        if (controller.isGrounded)
        {
            moveDirection=transform.forward * Input.GetAxis ("Vertical") * speed * runSpeed;
            float turn = Input.GetAxis("Horizontal");
            transform.Rotate(0, turn * turnSpeed * Time.deltaTime, 0);
        }
        moveDirection.y -= gravity * Time.deltaTime;
        controller.Move (moveDirection * Time.deltaTime);
        }
```

Figure 16 Script to animate and control the zombie

Script to make zombies look and follow the player as shown in Figure 17:

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

public class ZombieFake2 : MonoBehaviour
{

    public GameObject player;
    private CharacterController controller;
    public Vector3 characterVelocity;
    private bool groundedCharacter;
    private float characterSpeed = 2.0f;
    private float gravityValue = -9.81f;

    public float attackDistance=10f;
    public float bufferDistance=1f;

    // Start is called before the first frame update
    void Start()
    {
        controller = gameObject.GetComponent<CharacterController>();

    }

    // Update is called once per frame
    void Update()
    {
        groundedCharacter = controller.isGrounded;
        if (groundedCharacter && characterVelocity.y < 0)
        {
            characterVelocity.y = 0f;
        }

        var distance = Vector3.Distance(player.transform.position, transform.position);
        if (distance <= attackDistance)
            {
                if (distance >= bufferDistance)
                {

                    transform.LookAt(new Vector3(player.transform.position.x,player.transform.position.y-1.6f,player.transform.position.z));
                    transform.position = Vector3.MoveTowards(transform.position,player.transform.position,characterSpeed * Time.deltaTime);
                    //transform.position += transform.forward  * characterSpeed * Time.deltaTime;

                }
            }


    }
}
```

```
    // Update is called once per frame
    void Update()
    {
        if(EvacuationControl.EvacuationStatus == false){
                attackSpeed = 1.5f;
                followDistance = 10.0f;
                bufferDistance = 3.0f;
        }
        else{
                attackSpeed = 3.0f;
                followDistance = 5000.0f;
                bufferDistance = 3.0f;
        }

            var distance = Vector3.Distance(playerTransform.position, transform.position);
            // Debug.Log("Distance to Player" + distance);
            if (distance <= followDistance)
            {
                if (distance >= bufferDistance)
                {
                    anim.SetBool("isWalking", true);
                    Vector3 towardPlayer = new Vector3(player.transform.position.x,0f,player.transform.position.z);
                    transform.LookAt(player.transform.position);

                    transform.position = Vector3.MoveTowards(transform.position, new Vector3(playerTransform.position.x,playerTransform.position.y-.5f,playerTransform.position.z), attackSpeed * Time.deltaTime);
                    if(distance <= bufferDistance)
                        playerBar.TakeDamage(1);
                }
            }
            else
            {
                anim.SetBool("isWalking", false);
            }
    }
}
```

Figure 17 Script to make zombies look and follow

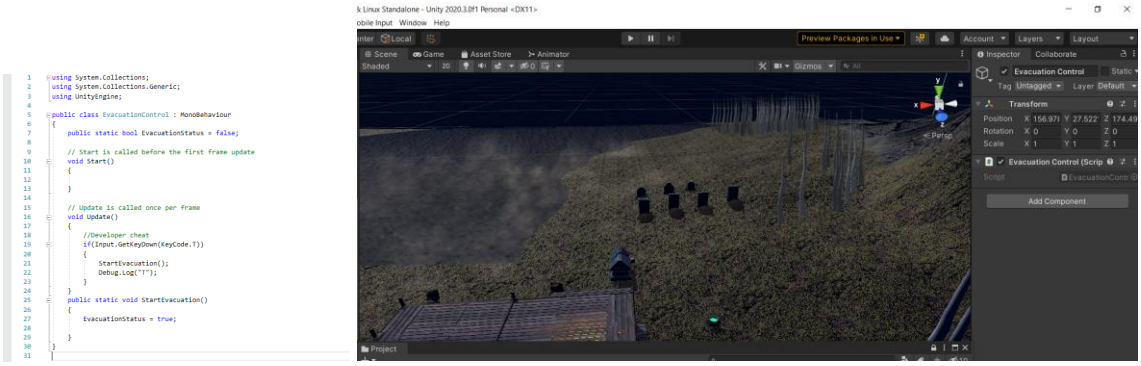
Interactivity

Evacuation can be triggered in two different ways. When the player interacts with the radio in the scene, a helicopter will come to rescue the player. The radio interactivity is caused by detecting 3 things: Object is 2 meters away from the center of the player's camera, the player has pressed E, & the object in question has the label "SelectableTag". Having all three of these conditions satisfied initiates evacuation.

This is the major focus of the game. Escape the area by summoning the helicopter with the radio.

As a developer tool, we have left in pressing T to start evacuation.

Developer implementation shown below. We have implemented this as shown in Figure 17, 18 and 19



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

public class EvacuationControl : MonoBehaviour
{
    public static bool EvacuationStatus = false;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        //Developer cheat
        if(Input.GetKeyDown(KeyCode.T))
        {
            StartEvacuation();
            Debug.Log("T");
        }
    }
    public static void StartEvacuation()
    {
        EvacuationStatus = true;

    }
}
```

Figure 18 Evacuation script

Helicopter evacuation:

How to survive in a barren wasteland in a supernatural setting: the game

The helicopter speed is a major point of difficulty in this game. Setting the helicopter speed slower dramatically increases the difficulty of the game, since the zombies are all alerted and are all persuing the player. Upon reaching the helicopter, the ending "cut scene" will begin where the camera is fixed to the helicopter and the player can watch from the helicopter as they escape.

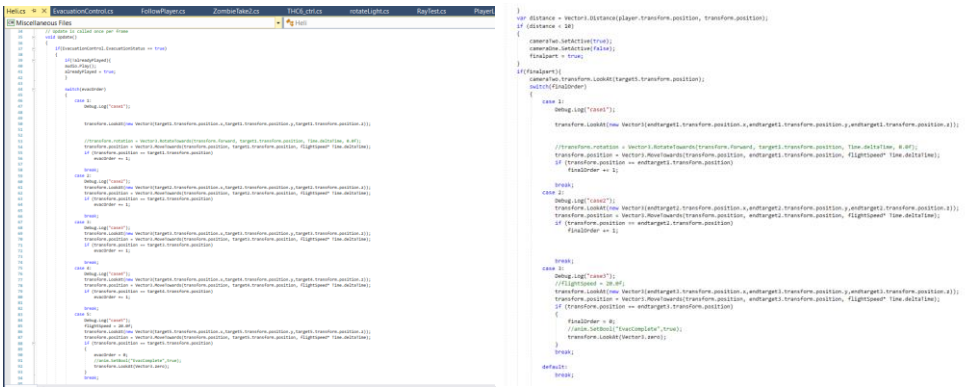


Figure 19 Helicopter animation/navigation

Avatars

Various zombie avatars are used to create this game where zombies would follow the player to create a scary effect like shown in Figure 20

Figure 20 Zombie Avatars

Details of the Virtual Environment

To create the perfect zombie land we added various UI elements. All UI elements used are shown below:

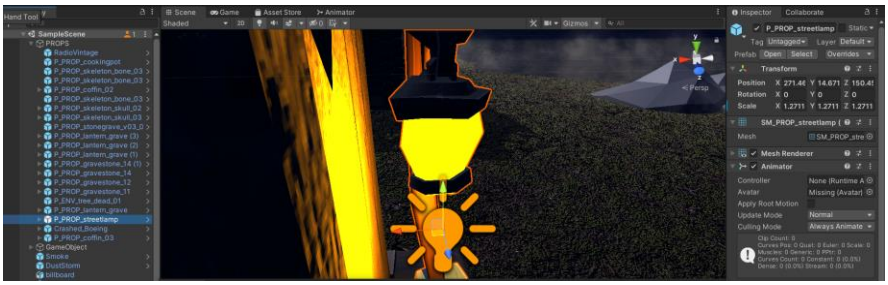


Figure 21 Street lights

Figure 22 Haunted house

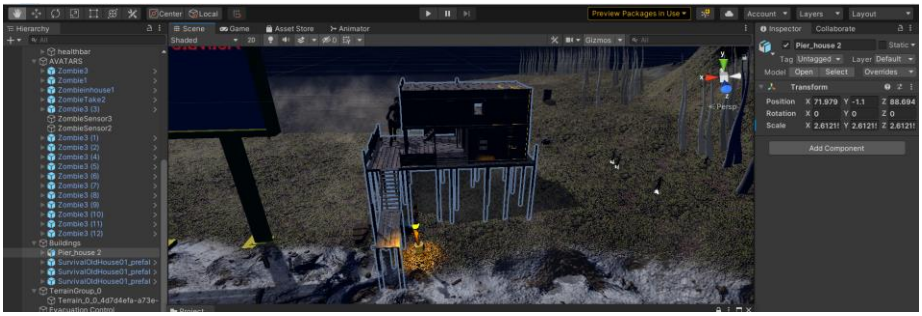


Figure 23 Pier House

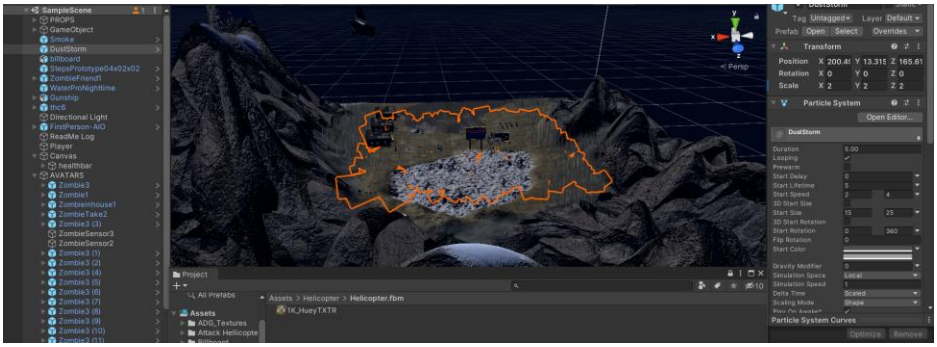


Figure 24 Dust storm to create the fog effect

Figure 25 Billboard with evacuation details

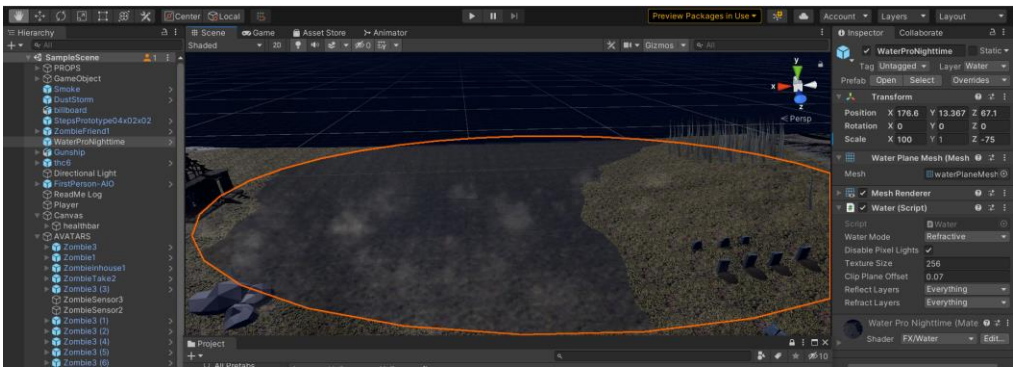


Figure 26 Water effect

Figure 27 Gravestones and crashed Boeing

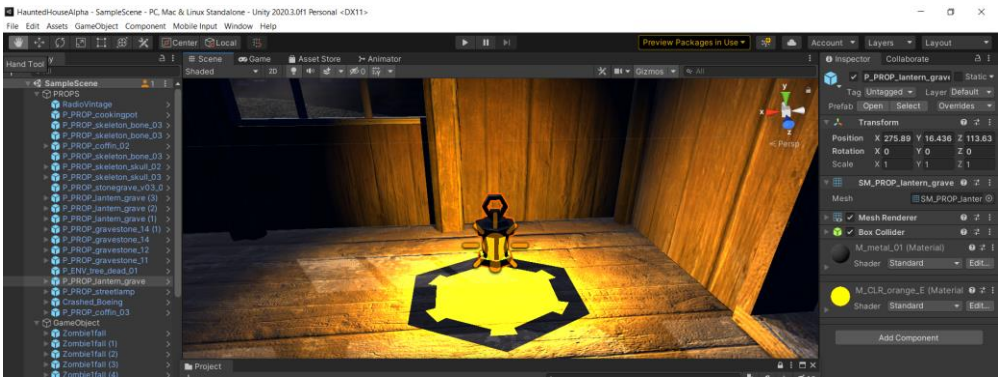


Figure 28 Lantern

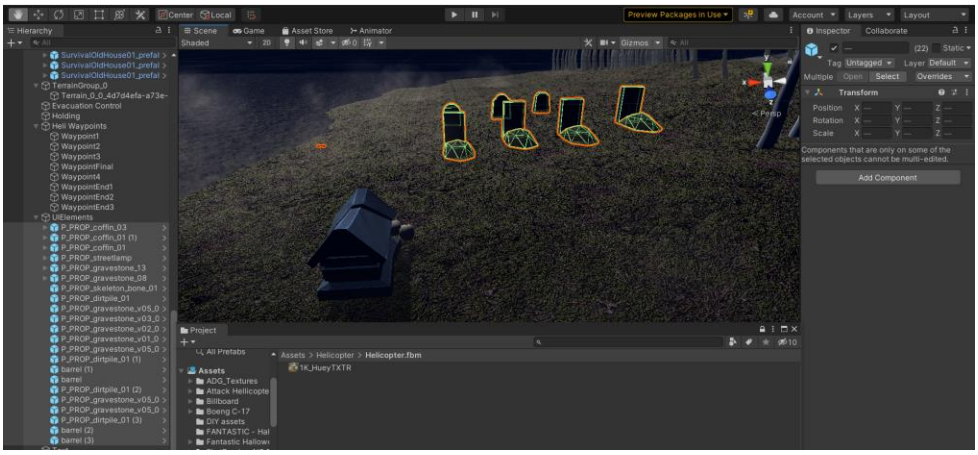


Figure 29 Dirt pile, grave stones and coffins

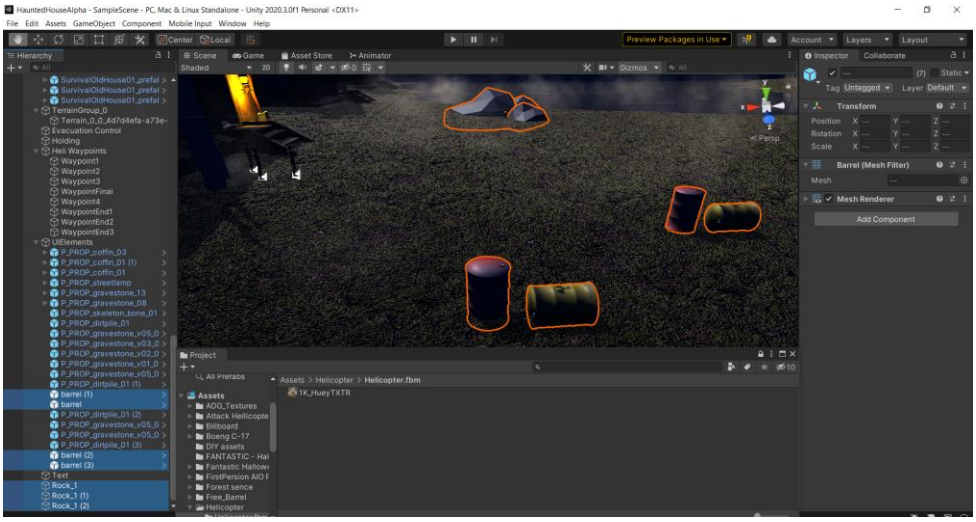


Figure 30 Barrels and rocks

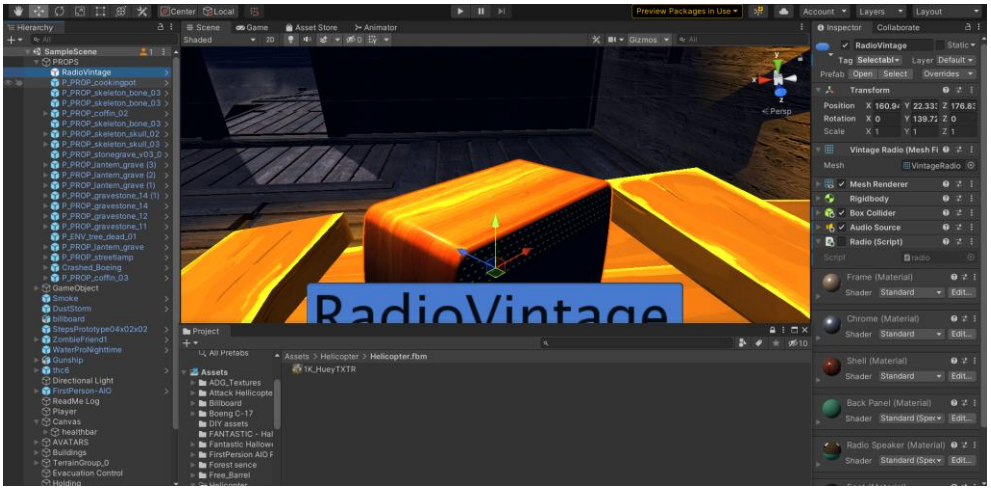


Figure 31 Vintage Radio

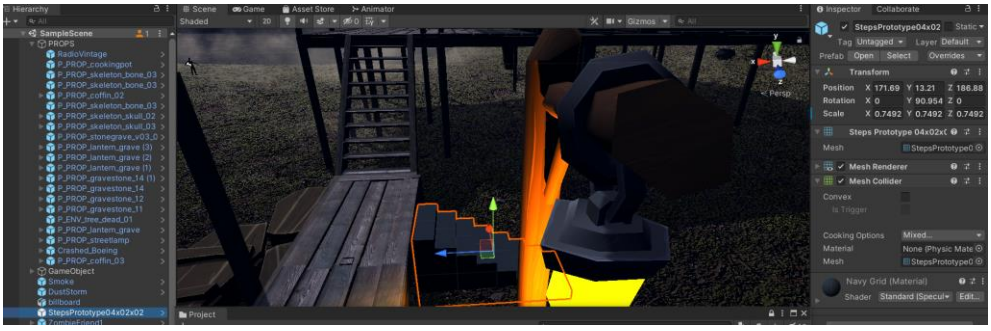


Figure 32 Steps for the pier house

How to survive in a barren wasteland in a supernatural setting: the game

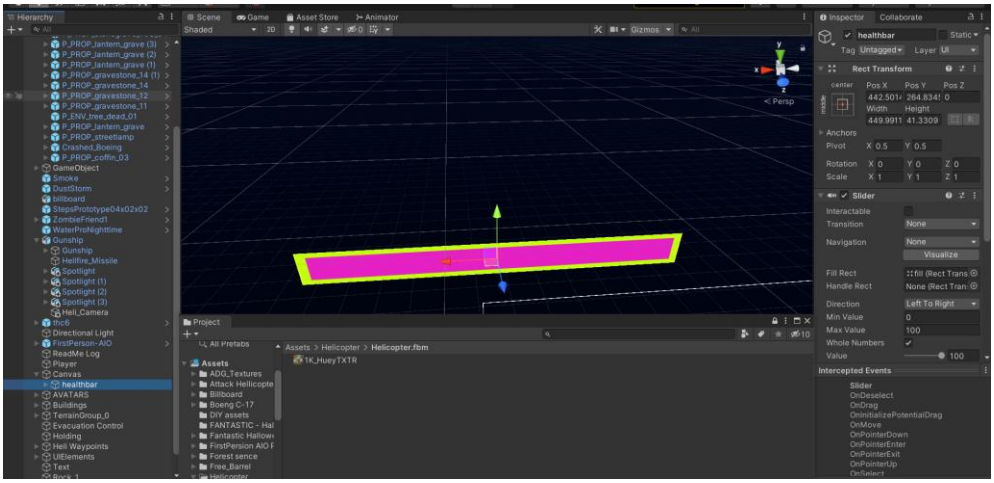


Figure 33 Health Bar

Audio file used:

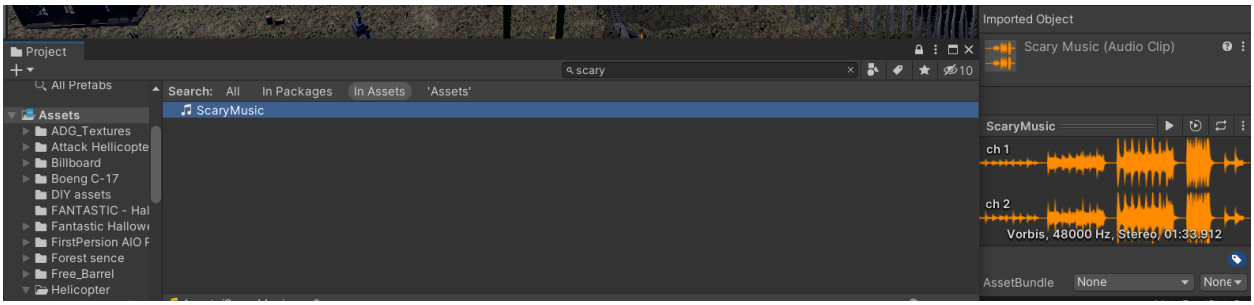


Figure 34 Scary music

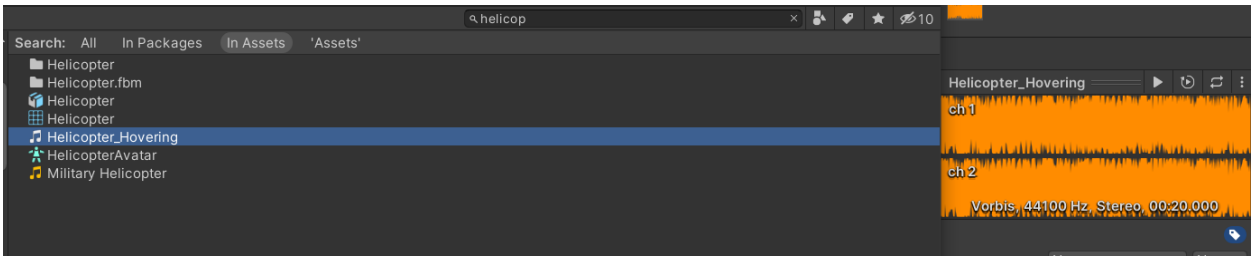


Figure 35 Helicopter music

Further Improvements

We can add how many zombies are tracking the player and also a remaining and elapsed time to create the sense of urgency

Software and Hardware Used

- Windows 10
- 16 bit RAM, 512 GB RAM
- Unity 2020.3.0f1

How to survive in a barren wasteland in a supernatural setting: the game

- C#
- VS 2019

Demo

https://www.youtube.com/watch?v=-SkJBfldQM4