COSC 729 Virtual Reality and its Applications

Project Title: Side Effects

Assignment Due: May 5th, 2021

Faculty: Dr. Sharad Sharma

Team Members:

Syed Ali

Contents

1. Goal and Objectives

1.1 Introduction

My anticipated project is to build a first-person shooter game called Side Effects. Side Effects is a first-person shooter game with a psychological horror-infected theme implemented to it. The main idea behind the game is that several years after receiving the COVID-19 vaccination, patients will either turn into infected like zombies or will have serious side effects which will make them hallucinate. The main character will be a nurse that will have hallucinogenic attacks throughout the game. The goal of the game will be to cure these infected patients using a vaccine that utilizes DNA synthesis technology to cure these patients. One of the objectives of the game is to survive and find more doses of the vaccine in the asylum. Most of this game module is built in Unity and 3ds Max.

1.2 Target Audience

The main target audience are gamers who are craving for a psychological horror game. Currently there are not many videogames in the market which have a psychological horror theme implemented to them. Companies like Xbox game studios, Amazon Lua, Google Stadia (when it existed), and Steam are actively seeking independent developers to make video games like these for their platforms and implementing a VR capability to them increases their market value since there are not many horror games (especially psychological horror games) in the market which have a VR capability associated to them.

1.3 Application Usefulness

Currently this game is in its pre-alpha state, so there are not many puzzle solving features implemented to the game. In the future more puzzle solving features will be added which will improve the players problem solving skills. The players playing this game will also gain knowledge about DNA synthesis and gene therapy which are two new evolving fields in the field of genetics.

2. Modeling

2.1 Planned Geometry

Assets such as battery, ammo and health pickups were modeled in 3ds Max. Other assets such as NPCs, infected zombies, syringe, boxes, and the asylum itself and its assets were imported from websites such as CG trader and Turbosquid, and most of these models were either scaled down or up on 3ds Max. The environment and the terrain were built inside the Unity Engine. Figures below describe the building of these assets.

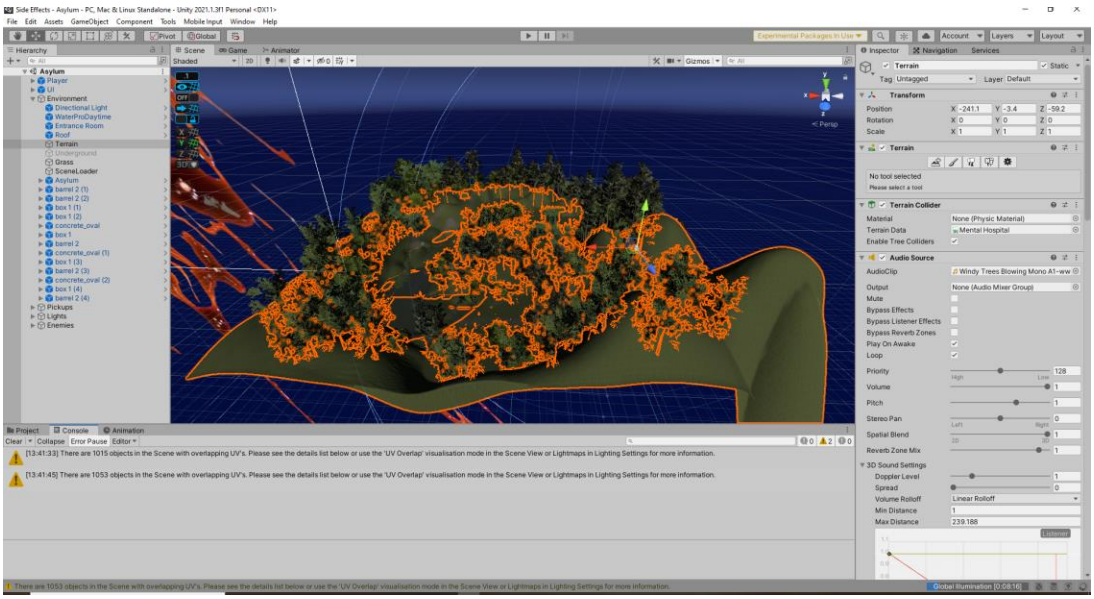


Figure 1: Terrain and Environment building inside Unity

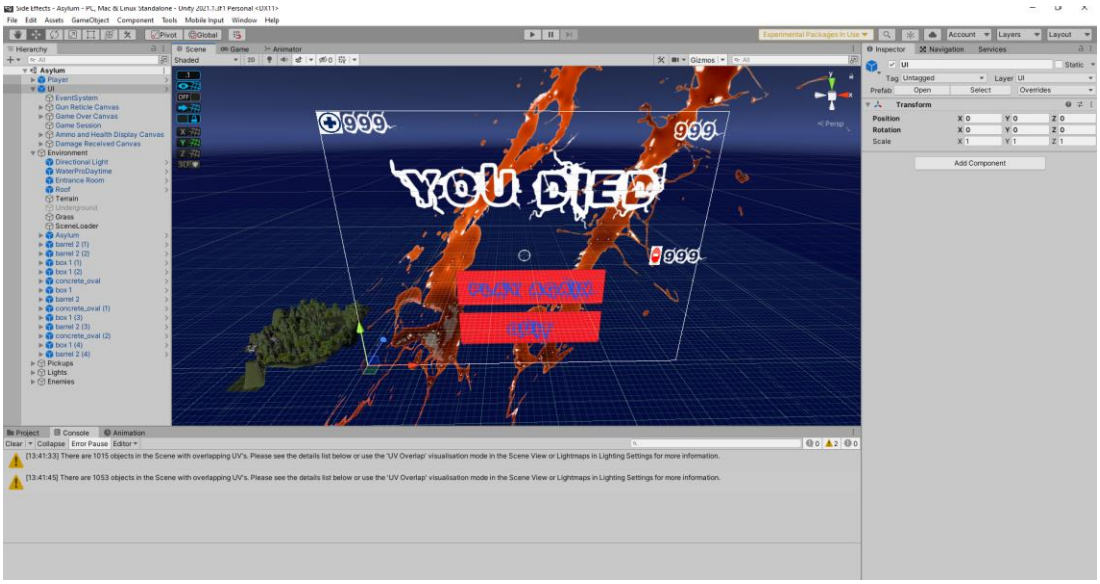


Figure 2: The building of UI canvas in Unity

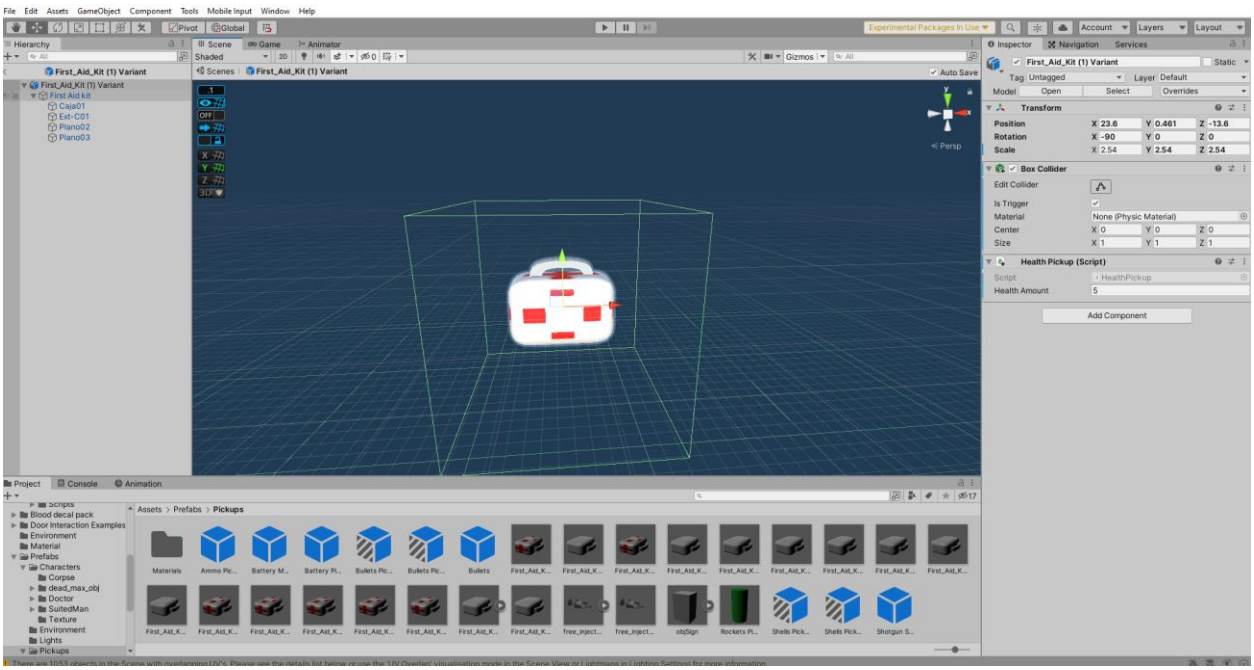


Figure 3: Health pack imported from CG Trader and scaled down in Unity along with applying textures to the model

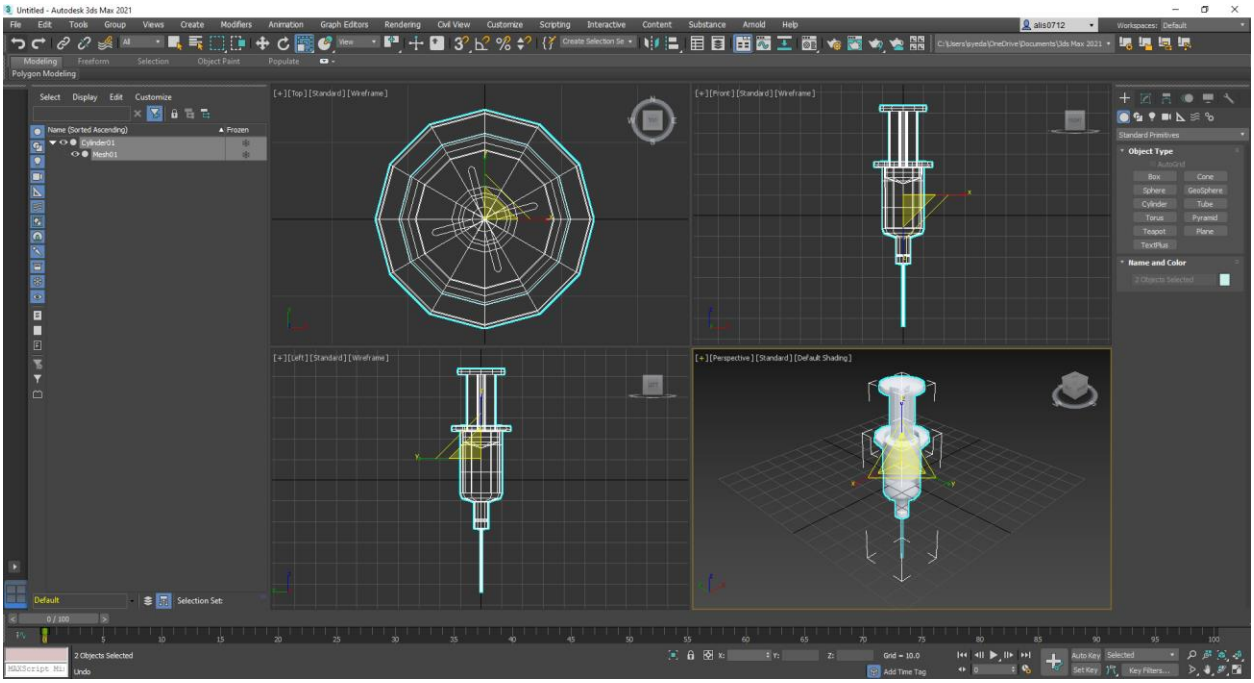


Figure 4: Syringe Model imported from turbosquid and scaled down in 3ds Max

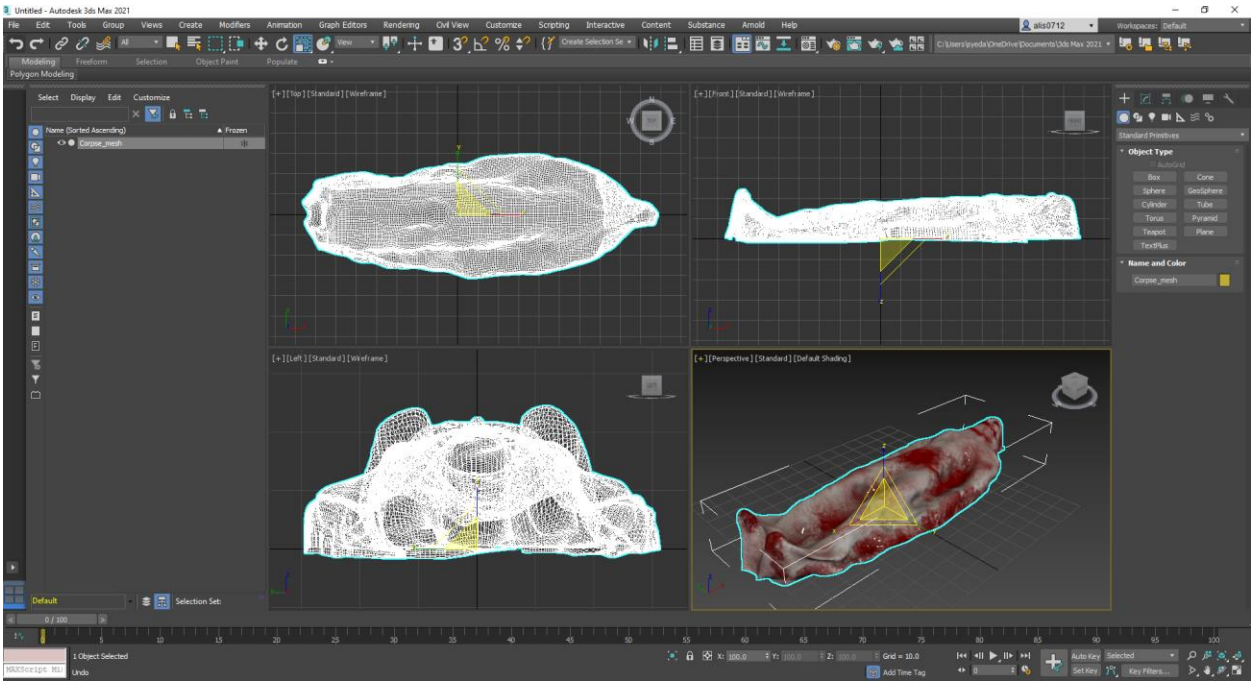


Figure 5: Corpse Model imported from CG trader and scaled down in 3ds Max.

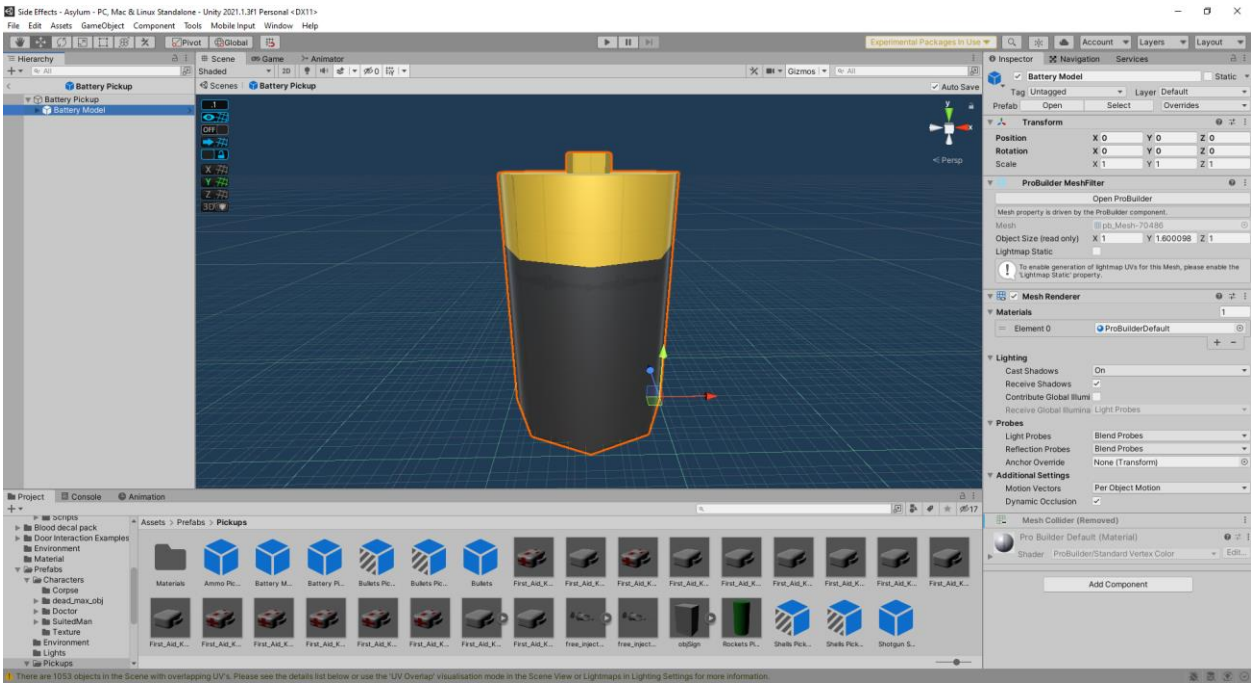


Figure 6: Applying textures to the battery model initially made in 3ds Max.

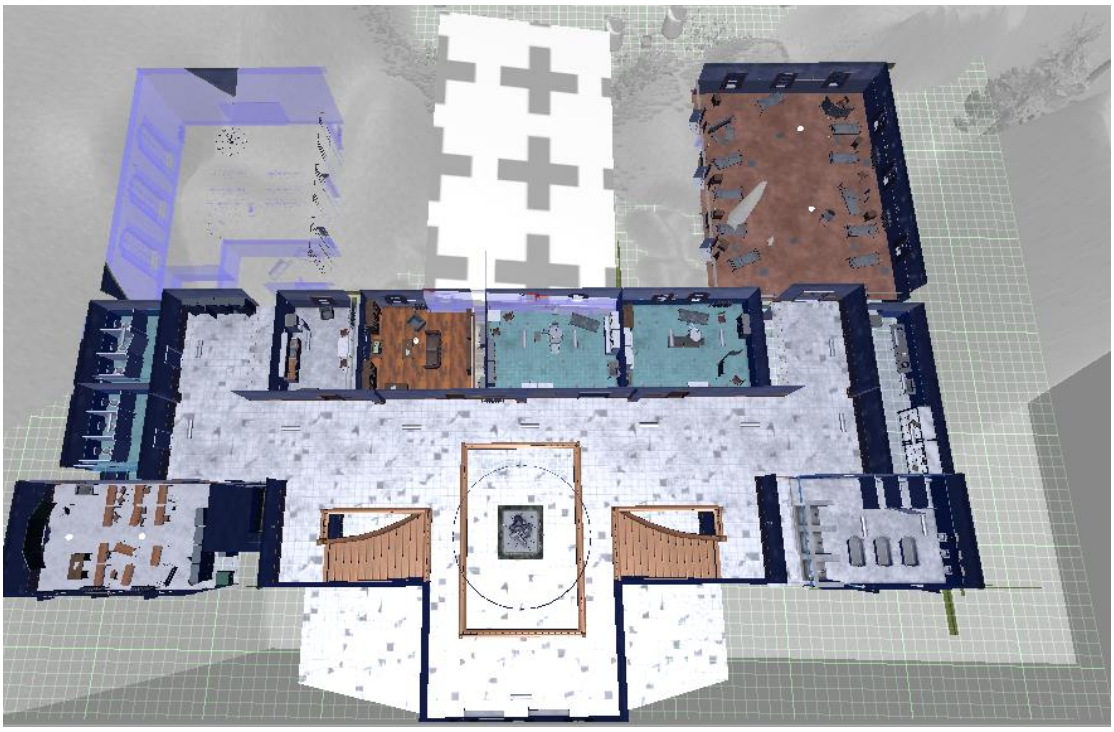


Figure 7: The placement of the Asylum model in the terrain in Unity initially imported from Turbosquid and scaled down in 3ds Max.

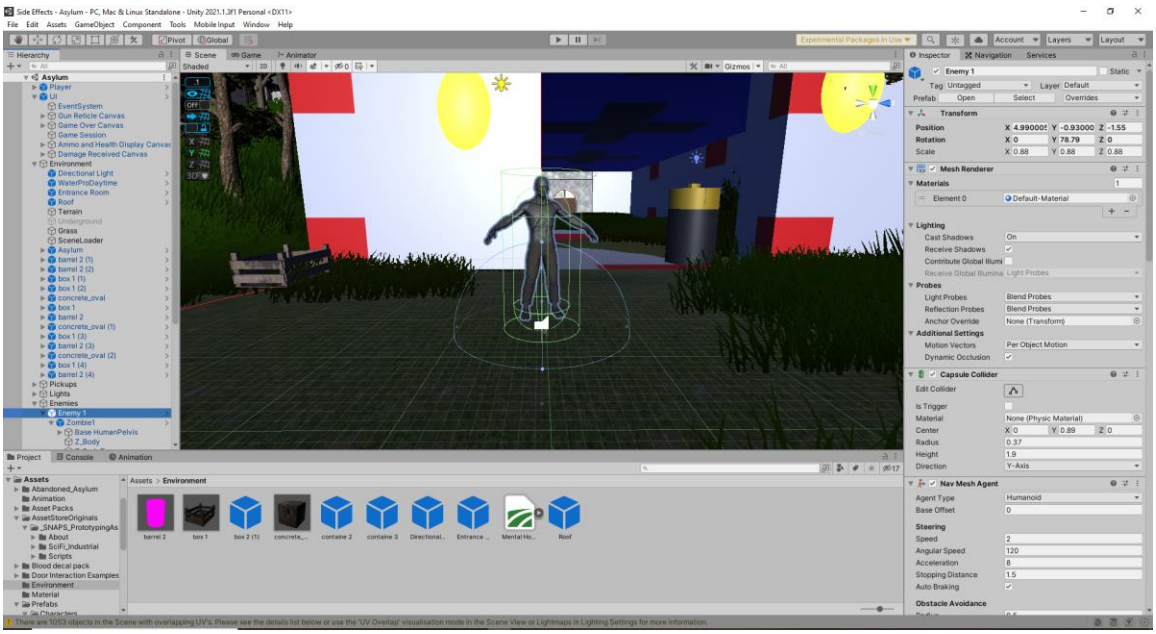


Figure 8: Zombie asset imported from Unity asset store. The figure shows implementation of Nav mesh agent, animator controller, and capsule collider.

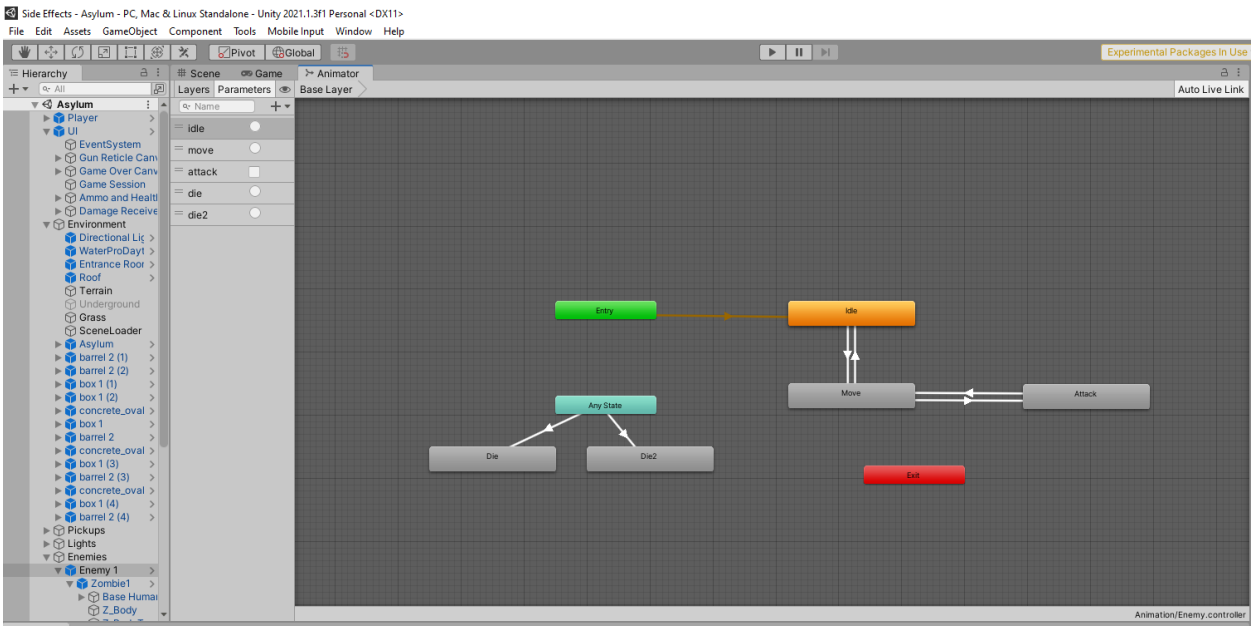


Figure 9: Animator controller for the zombie avatar and the parameters implemented to it.

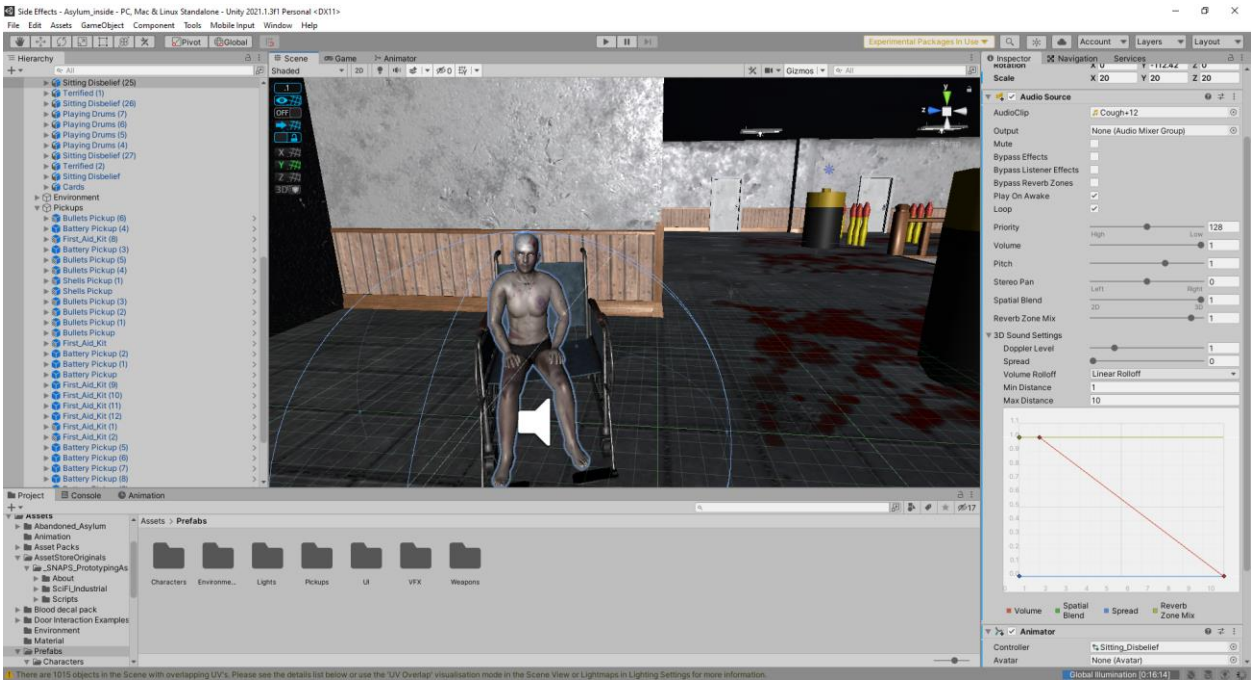


Figure 10: NPC asset imported from Turbosquid. The figure shows implementation of sound and animator controller.

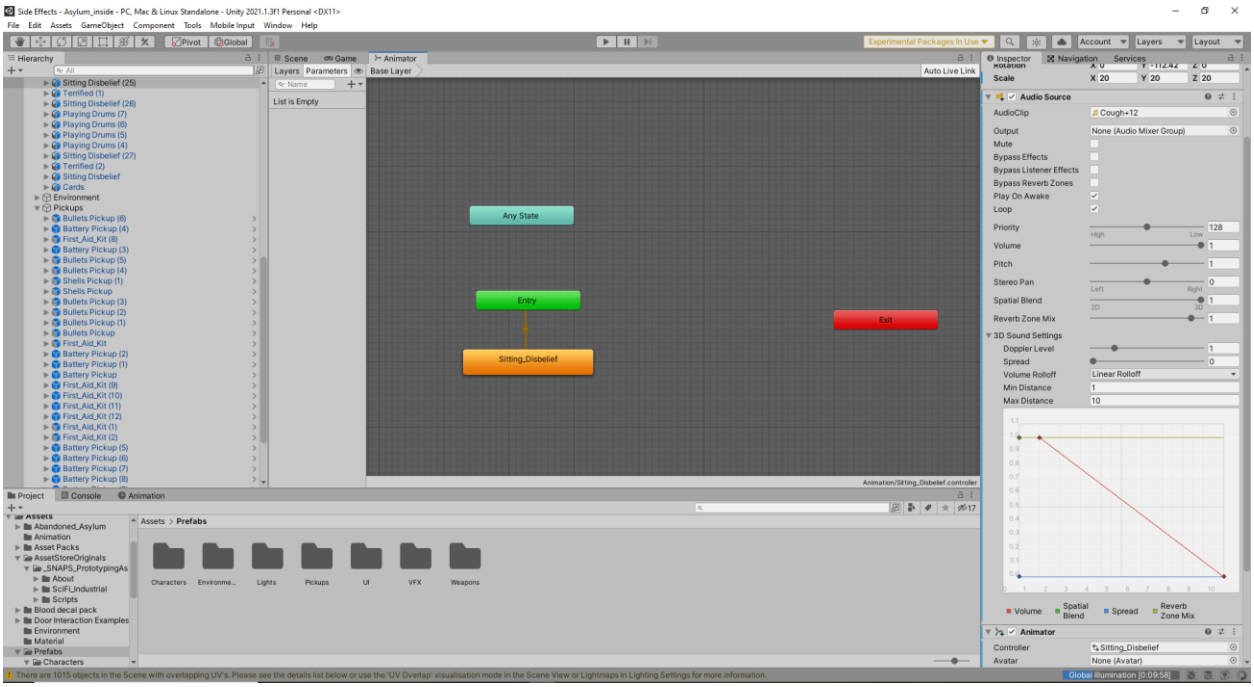


Figure 11: An example of an animator path for the NPC asset in figure 10.

2.2 Application Usage

As described in section 1.2, the main target audience are gamers. As the game begins, the player will have to navigate through the terrain and enter an abandoned asylum. Inside the asylum the goal of the player will be to navigate around the asylum and find more doses of the vaccine that will cure the infected.

The player will be given a limited amount of vaccine initially when starting the game and will have to be careful when using it, inventory management will play a key factor when navigating through the terrain and asylum. There is a score counter implemented in the game which will let the player know how many infected have been treated. The player will also have a choice to shoot the infected, however, if the player decides to do so, the score will not be updated. Figures below describe the game mechanics.

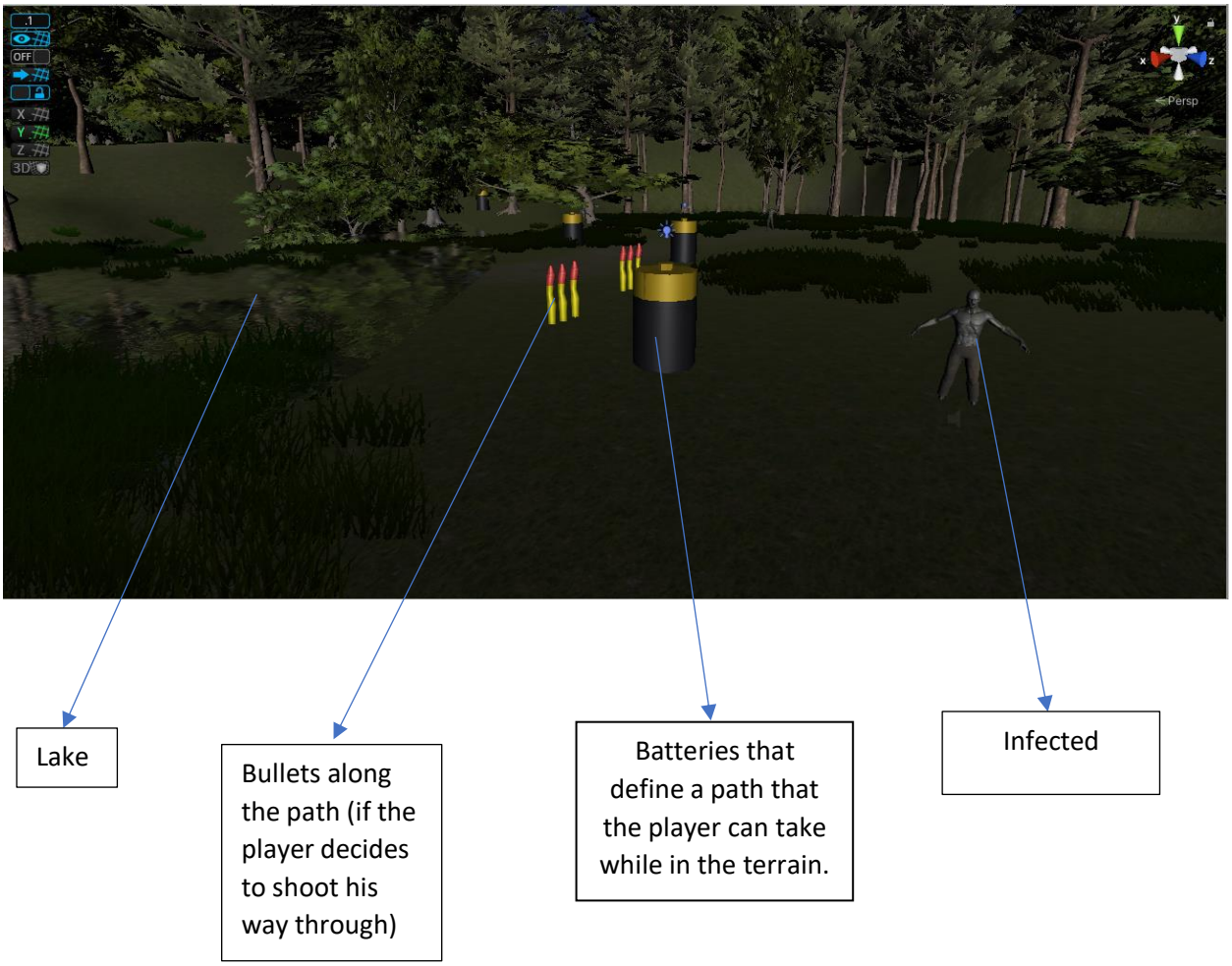


Figure 12: A figure describing the path the player undertakes while in the terrain

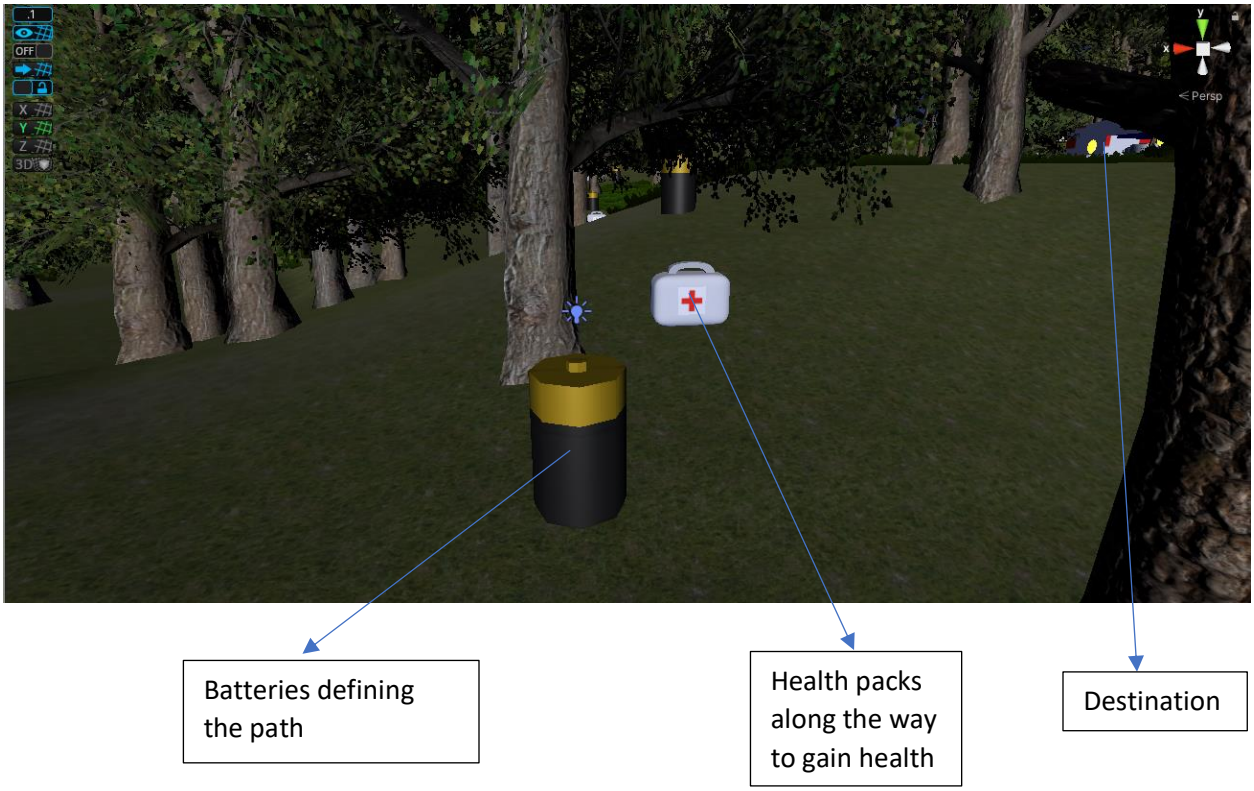


Figure 13: A figure describing the path the player undertakes while in the terrain. Health packs are added along the path to gain health

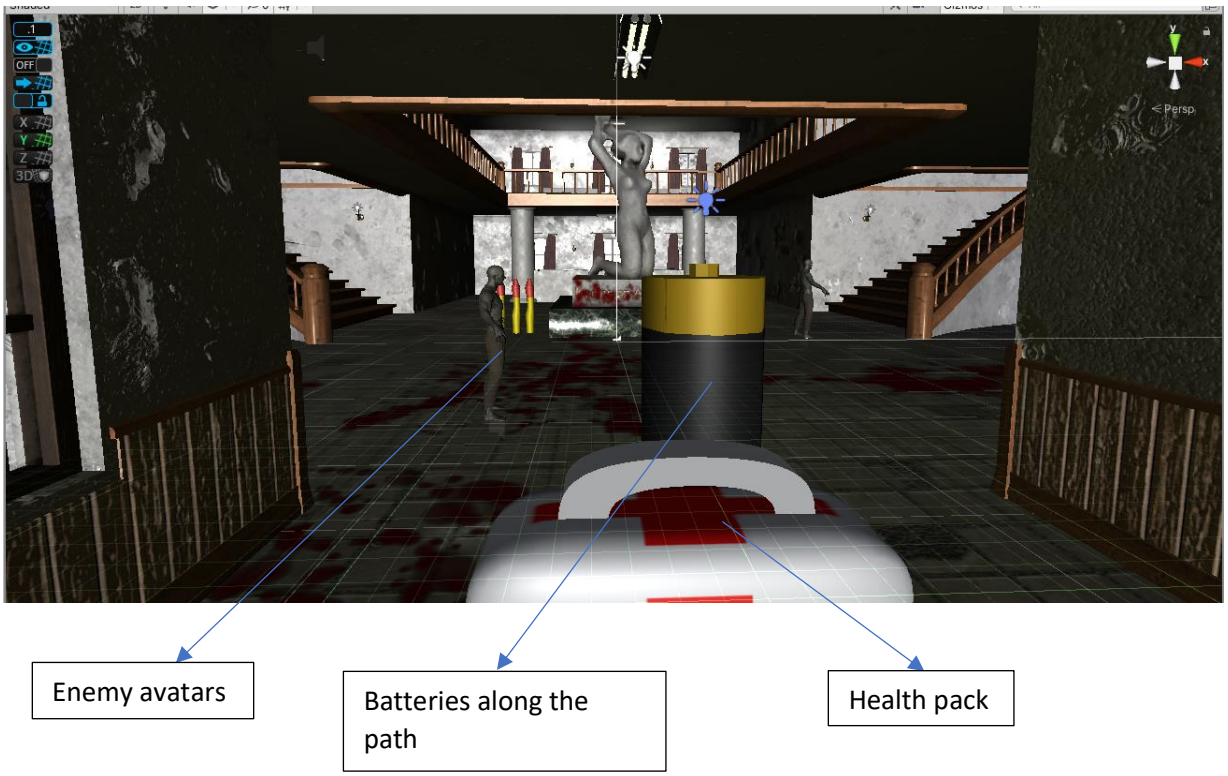


Figure 14: A figure describing the path the player undertakes while in the asylum. Health packs and batteries are added along the path to gain health and luminosity.

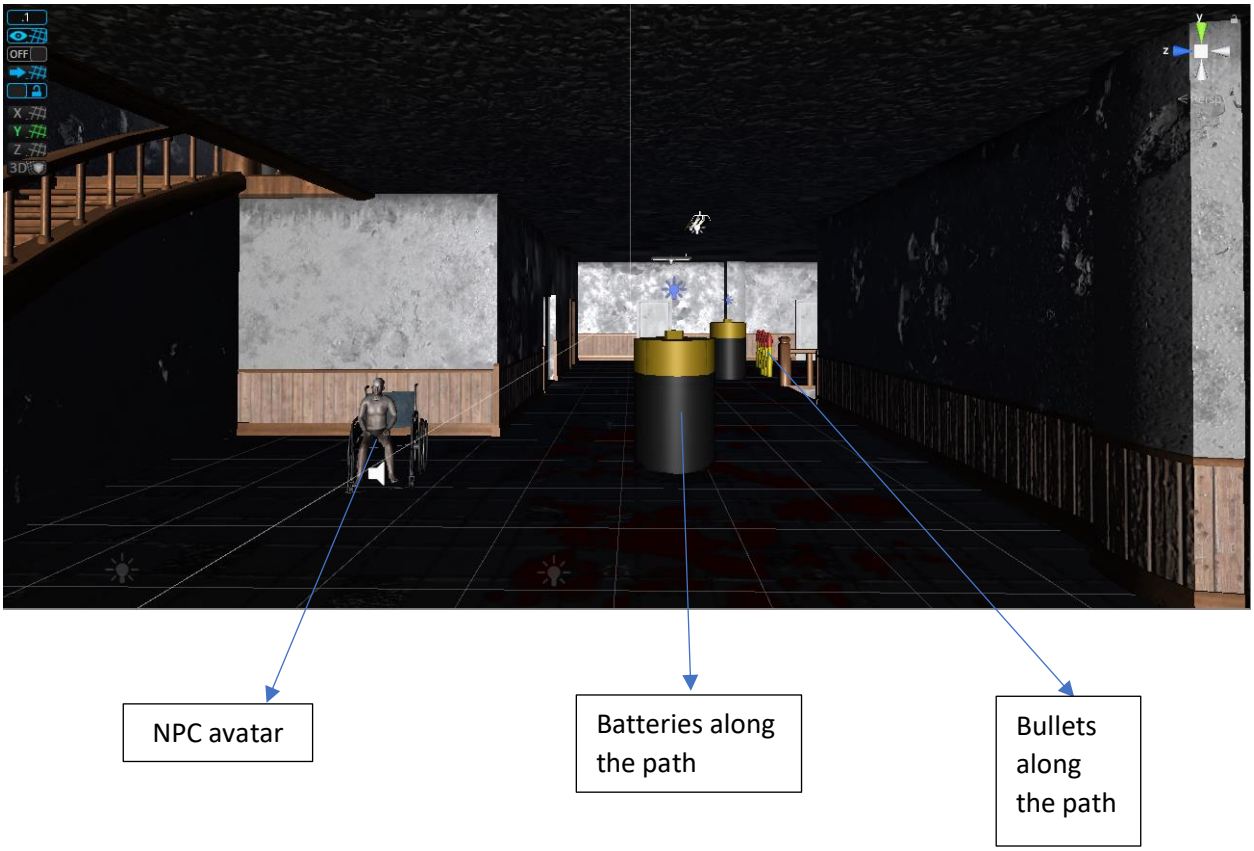


Figure 15: A figure describing the path the player undertakes while in the asylum. Batteries and bullets are added along the path to gain luminosity and bullets.

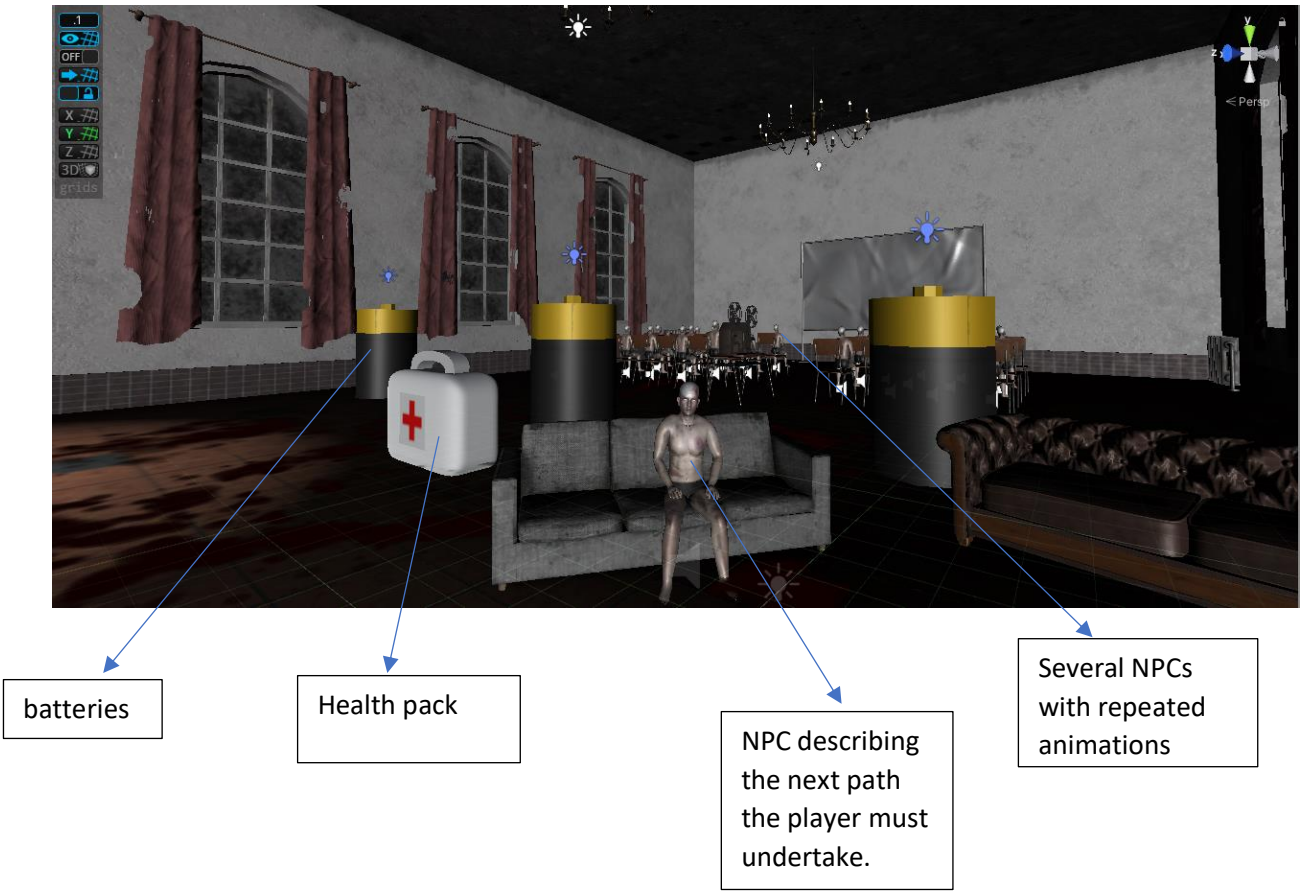


Figure 16: A figure describing the path the player undertakes while in the asylum. The player will meet an NPC which will guide him to meet with Andre.

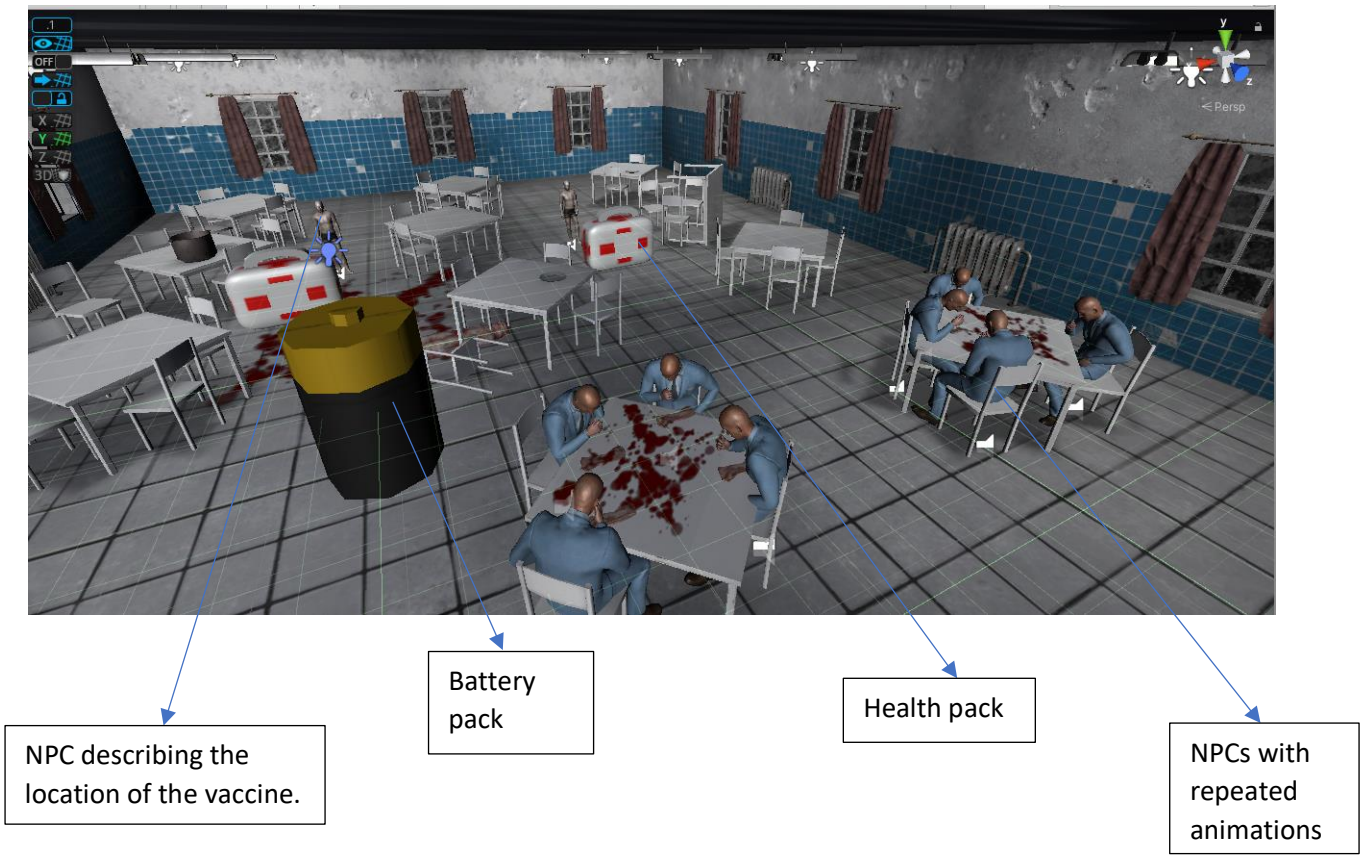


Figure 17: A figure describing the path the player undertakes while in the asylum. The player will meet an NPC which will guide him to the location of the vaccine.

Figure 18: A figure describing the path the player undertakes while in the asylum and the location of the vaccine.

2.3 Programming

Several programs were written in C# which provided specific competences within the game. The list of programs created along with their descriptions are provided below:

Ammo: The ammo program contained several functions that allowed the player to reduce or increase their current ammo. If the player decided to shoot an enemy, a ReduceCurrentAmmo function is called which reduces the amount of ammo. If the player goes near a pick-up the IncreaseCurrentAmmo function is called which increases the ammo amount. A sample snippet for this code is given below:

Figure 19: A figure describing the Ammo C# program.

AmmoPickup: This code allows the player to increase the magazine size of the weapon. A sample code snippet is shown below:

Figure 20: A figure describing the AmmoPickup C# program.

AmmoType: This small program allows the player to cycle through the magazine type. For example, the syringe has a magazine type of a vaccine. A code snippet for this program is given below:

```
public enum AmmoType
{
    Bullets,
    Shells,
    Rockets,
    Vaccine
}
```

Figure 21: A figure describing the AmmoType C# program.

AsylumInside: This program allows the player to change the sound to “sneaker shoes walking on concrete”. There is a trigger component in the asylum hallway that changes to this sound. The sound is based on keyboard inputs. If W is pressed, then the sound will be played. A snippet of this code is given below:

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

Unity Script | 0 references
public class AsylumInside : MonoBehaviour
{
    public AudioSource Asylum;
    private bool AsylumEntered = false;

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        AsylumEntered = true;
    }

    Unity Message | 0 references
    private void Update()
    {

        if (Input.GetKeyDown("w") || Input.GetKeyDown("s") && AsylumEntered == true)
        {
            Asylum.Play();
        }

        if (Input.GetKeyUp("w") || Input.GetKeyUp("s") && AsylumEntered == true)
        {
            Asylum.Stop();
        }
    }
}
```

Figure 22: A figure describing the AsylumInside C# program.

BatteryPickup: This code allows the player to pickup batteries along the given path. It also restores the flashlight's luminosity and angle. A code snippet of this code is given below:

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

Unity Script | 0 references
public class BatteryPickup : MonoBehaviour
{
    [SerializeField] float restoreAngle = 90f;
    [SerializeField] float addIntensity = 1f;

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject.tag == "Player")
        {
            other.GetComponentInChildren<FlashLightSystem>().RestoreLightAngle(restoreAngle);
            other.GetComponentInChildren<FlashLightSystem>().AddLightIntensity(addIntensity);
            Destroy(gameObject);

        }
    }

}
```

Figure 23: A figure describing the BatteryPickup C# program.

DeathHandler: This C# code is triggered when the player's health reaches zero. It is essentially a game over screen letting the player to either quit the game or restart the game from level I. A code snippet is provided below

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

Unity Script | 1 reference
public class DeathHandler : MonoBehaviour
{
    [SerializeField] Canvas gameOverCanvas;

    Unity Message | 0 references
    private void Start()
    {
        gameOverCanvas.enabled = false;
    }

    1 reference
    public void HandleDeath()
    {
        gameOverCanvas.enabled = true;
        Time.timeScale = 0;
        FindObjectOfType<WeaponSwitcher>().enabled = false;
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }
}
```

Figure 24: A figure describing the DeathHandler C# program.

DisplayDamage: This C# code is triggered when the player receives damage from the infected avatar. A code snippet is provided below:

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

Unity Script | 1 reference
public class DisplayDamage : MonoBehaviour
{

    [SerializeField] Canvas impactCanvas;
    [SerializeField] float impactTime = 0.3f;


    Unity Message | 0 references
    void Start()
    {
        impactCanvas.enabled = false;
    }

    1 reference
    public void ShowDamageImpact()
    {
        StartCoroutine(ShowSplatter());
    }

    1 reference
    IEnumerator ShowSplatter()
    {
        impactCanvas.enabled = true;
        yield return new WaitForSeconds(impactTime);
        impactCanvas.enabled = false;
    }
}
```

Figure 25: A figure describing the DisplayDamage C# program.

DoorScript: This simple script triggers the opening of the door when the player reaches close to the door. A code snippet is provided below:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
   Unity Script | 0 references
5  public class DoorScript : MonoBehaviour
6  {
7      private Animator animator;
8
       Unity Message | 0 references
9      void Start()
10     {
11         animator = GetComponent<Animator>();
12     }
13
14
       Unity Message | 0 references
15     void OnTriggerEnter(Collider other)
16     {
17         if (other.gameObject.tag == "Player")
18         {
19             animator.SetBool("open", true);
20         }
21     }
22
23 }
```

Figure 26: A figure describing the DoorScript C# program

EnemyAI: This C# script allows the enemy avatar to follow and attack the player. It is also containing a function that provokes the enemy to confront the player if it is attacked. A code snippet is provided below:

```
public class EnemyHealth : MonoBehaviour
{
    [SerializeField] float hitPoints = 100f;
    public TextMeshProUGUI scoreText;
    bool isDead = false;
    private AudioSource infected;
    private static int currentScore = 0;


    Unity Message | 0 references
    private void Start()
    {
        infected = GetComponent<AudioSource>();
        scoreText.text = currentScore.ToString();



    }


    1 reference
    public bool IsDead()
    {
        return isDead;
    }

    1 reference
    public void TakeDamage(float damage)
    {
        BroadcastMessage("OnDamageTaken");
        hitPoints -= damage;
        if(hitPoints <= 0)
        {
          currentScore += 1;
          scoreText.text = currentScore.ToString();
          Die();
          infected.Stop();
        }
    }
    1 reference
    private void Die()
    {
        if(isDead)
```

Figure 27: A figure describing the EnemyAI C# program

EnemyAttack: This simple C# code reduces the player's health when the player is attacked. It also calls on the DisplayDamage C# code. A code snippet is provided below:

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

Unity Script | 0 references
public class EnemyAttack : MonoBehaviour
{
    PlayerHealth target;
    [SerializeField] float damage = 40f;



    Unity Message | 0 references
    void Start()
    {
        target = FindObjectOfType<PlayerHealth>();
    }

    0 references
    public void AttackHitEvent()
    {
        if (target == null)
        {
            return;
        }
        target.TakeDamage(damage);
        target.GetComponent<DisplayDamage>().ShowDamageImpact();
    }
}
```

Figure 28: A figure describing the EnemyAttack C# program

EnemyHealth: This C# script is for the enemy's health, when enemy's health reaches zero it triggers a death animation and stops playing the enemy's sound. It also has a score counter which updates every time an enemy is vaccinated. A code snippet is provided below:

```
public class EnemyHealth : MonoBehaviour
{
    [SerializeField] float hitPoints = 100f;
    public TextMeshProUGUI scoreText;
    bool isDead = false;
    private AudioSource infected;
    private static int currentScore = 0;

    Unity Message | 0 references
    private void Start()
    {
        infected = GetComponent<AudioSource>();
        scoreText.text = currentScore.ToString();


    }


    1 reference
    public bool IsDead()
    {
        return isDead;
    }

    1 reference
    public void TakeDamage(float damage)
    {
        BroadcastMessage("OnDamageTaken");
        hitPoints -= damage;
        if(hitPoints <= 0)
        {
          currentScore += 1;
          scoreText.text = currentScore.ToString();
          Die();
          infected.Stop();
        }
    }
    1 reference
    private void Die()
    {
        if(isDead)
```

Figure 29: A code snippet showing the enemy's health C# script.

FlashLight: This C# script implements a flashlight system, where the flashlight's luminosity decreases over time. It also calls on several functions used in BatteryPickup C# code. A code snippet for this is provided below:

```
Unity Script | 2 references
public class FlashLightSystem : MonoBehaviour
{
    [SerializeField] float lightDecay = 0.1f;
    [SerializeField] float angleDecay = 1f;
    [SerializeField] float minimumAngle = 40f;

    Light myLight;

    Unity Message | 0 references
    private void Start()
    {
        myLight = GetComponent<Light>();
    }

    Unity Message | 0 references
    private void Update()
    {
        DecreaseLightAngle();
        DecreaseLightIntensity();
    }

    1 reference
    public void RestoreLightAngle(float restoreAngle)
    {
        myLight.spotAngle = restoreAngle;
    }

    1 reference
    public void AddLightIntensity(float intensityAmount)
    {
        myLight.intensity += intensityAmount;
    }
    1 reference
    private void DecreaseLightAngle()
    {
        if (myLight.spotAngle <= minimumAngle)
        {
            return;
        }
        else
        {
            myLight.spotAngle -= angleDecay * Time.deltaTime;
```

Figure 30: A figure describing the FlashLight C# program

HealthPickup: Like other pickups, this script allows the player to gain their health back. A code snippet for this is provided below:

```
Unity Script | 0 references
public class HealthPickup : MonoBehaviour
{
    [SerializeField] int healthAmount = 5;

    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {

        if (other.gameObject.tag == "Player")
        {

            FindObjectOfType<PlayerHealth>().IncreaseCurrentHealth(healthAmount);
            Destroy(gameObject);

        }
    }
}
```

Figure 31: A figure describing the HealthPickup C# program

Move: This C# script allows one of the NPCs in the cinema room to move from point a to b. A code snippet for this is provided below:

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

Unity Script | 0 references
public class Move : MonoBehaviour
{
    public Vector3 pointB;
    public float maxSpeed = 1;
    private CharacterController m_CharacterController;
    private CollisionFlags m_CollisionFlags;
    // this is an object, so that you can move it around in the editor.

    // units per second

    Unity Message | 0 references
    void Update()
    {
        var change = maxSpeed * Time.deltaTime;
        transform.position = Vector3.MoveTowards(transform.position, pointB, change);
    }

    Unity Message | 0 references
    private void OnControllerColliderHit(ControllerColliderHit hit)
    {
        Rigidbody body = hit.collider.attachedRigidbody;
        //dont move the rigidbody if the character is on top of it
        if (m_CollisionFlags == CollisionFlags.Below)
        {
            return;
        }

        if (body == null || body.isKinematic)
        {
            return;
        }
        body.AddForceAtPosition(m_CharacterController.velocity * 0.1f, hit.point, ForceMode.Impulse);
    }
}
```

Figure 32: A figure describing the Move C# program

PlayerHealth: Like Enemy's Health script, this C# script is for the player health which updates when the player is either attacked or is close to health pack. It also contains a function that allows it to be displayed on the UI. A code snippet is provided below:

```
public class PlayerHealth : MonoBehaviour

{
        [SerializeField]  float hitPoints = 100f;
        [SerializeField] TextMeshProUGUI healthText;


    Unity Message | 0 references
    void Update()
    {
        DisplayHealth();
    }

    1 reference
    public void IncreaseCurrentHealth( float healthAmount)
    {
        if (hitPoints < 100)
        {
            hitPoints += healthAmount;
        }
    }

    1 reference
    private void DisplayHealth()
    {
        float currentHealth = hitPoints;
        healthText.text = currentHealth.ToString();
    }
    1 reference
    public void TakeDamage(float damage)
        {
            hitPoints -= damage;
            if (hitPoints <= 0)
            {
            GetComponent<DeathHandler>().HandleDeath();
            }
        }
}
```

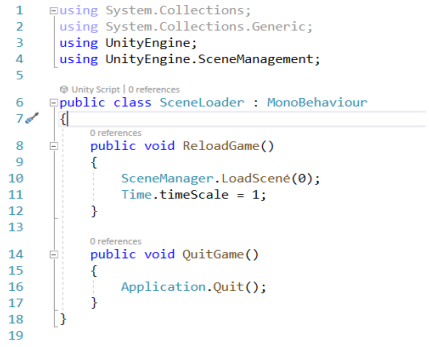
Figure 33: A figure describing the PlayerHealth C# program

SceneChanger: This simple C# script allows the player to change from scene to the next. In the first scene there is a trigger element that allows the player to change the scene. A code snippet is provided below:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
    Unity Script | 0 references
5   public class SceneChanger : MonoBehaviour
6   {
        Unity Message | 0 references
7       public void OnTriggerEnter(Collider other)
8       {
9           SceneManager.LoadScene(1);
10      }
11  }
12
```

Figure 34: A figure describing the SceneChanger C# program

SceneLoader: This simple C# script allows the player to either quit the game or restart the game when the player's health approaches zero. A code snippet is provided below:



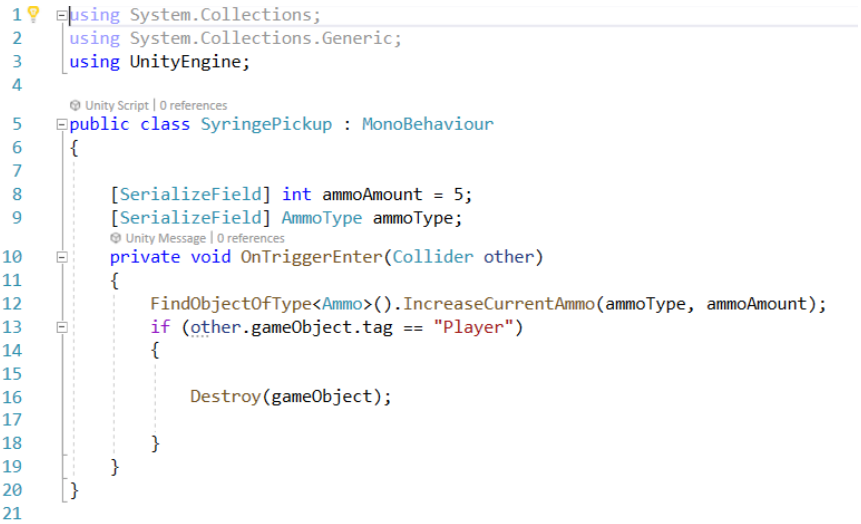
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script | 0 references
public class SceneLoader : MonoBehaviour
{
    0 references
    public void ReloadGame()
    {
        SceneManager.LoadScene(0);
        Time.timeScale = 1;
    }

    0 references
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

Figure 35: A figure describing the SceneLoader C# program

SyringePickup: Like ammoPickup, this C# script allows the player to pick-up vaccine doses scattered around the environment. A code snippet is provided below:



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

Unity Script | 0 references
public class SyringePickup : MonoBehaviour
{

    [SerializeField] int ammoAmount = 5;
    [SerializeField] AmmoType ammoType;
    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        FindObjectOfType<Ammo>().IncreaseCurrentAmmo(ammoType, ammoAmount);
        if (other.gameObject.tag == "Player")
        {

            Destroy(gameObject);

        }
    }
}
```

Figure 36: A figure describing the SyringePickup C# program

TerrainSounds: Like AsylumInside, this program allows the player to change the sound to "sneaker shoes walking on grass". There is a trigger component in the terrain that changes to this sound. The sound is based on keyboard inputs. If W is pressed, then the sound will be played. A snippet of this code is given below:

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

Unity Script | 1 reference
public class TerrainSound : MonoBehaviour
{
    public AudioSource Grass;

    private bool TerrainEntered = false;


    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
            TerrainEntered = true;

    }

    Unity Message | 0 references
    private void Update()
    {

        if (Input.GetKeyDown("w") || Input.GetKeyDown("s") && TerrainEntered == true)
        {
            Grass.Play();

        }

        if (Input.GetKeyUp("w") || Input.GetKeyUp("s") && TerrainEntered == true)
        {
            Grass.Pause();
        }


    }
}
```

Figure 37: A figure describing the TerrainSounds C# program.

Weapon: This C# script allows the player to shoot or vaccinate at enemy avatars. It utilizes raycasting while vaccinating or shooting the enemy. It also creates a hit impact which notifies the player that he has hit the enemy avatar. It also contains a muzzle flash which is again used to notify the player that they are indeed shooting or vaccinating the enemy avatar. It also contains a DisplayAmmo function that allows the doses or the magazine size to be displayed on the UI. A sound function is also implemented into the code that plays a sound when the trigger is released and stops when the cartridge is empty. A code snippet is provided below:

```
7   public class Weapon : MonoBehaviour
8   {
9       [SerializeField] Camera FPCamera;
10      [SerializeField] float range = 100f;
11      [SerializeField] float damage = 30f;
12      [SerializeField] ParticleSystem muzzleFlash;
13      [SerializeField] GameObject hitEffect;
14      [SerializeField] Ammo ammoSlot;
15      [SerializeField] AmmoType ammoType;
16      [SerializeField] float timeBetweenShots = 0.5f;
17      [SerializeField] TextMeshProUGUI ammoText;
18
19      bool canShoot = true;
20      AudioSource shootingSound;
21
            Unity Message | 0 references
22      void Start()
23      {
24          shootingSound = GetComponent<AudioSource>();
25      }
            Unity Message | 0 references
26      private void OnEnable()
27      {
28          canShoot = true;
29      }
            Unity Message | 0 references
30      void Update()
31      {
32          DisplayAmmo();
33          if(Input.GetMouseButtonDown(0) && canShoot == true)
34          {
35              StartCoroutine(Shoot());
36              shootingSound.Play();
37              if (ammoSlot.GetCurrentAmmo(ammoType) == 0)
38              {
39                  shootingSound.Stop();
40              }
41          }
42      }
43
            1 reference
44      private void DisplayAmmo()
45      {
```

Figure 38: A figure describing the Weapon C# program.

WeaponSwitcher: This C# code allows the player to switch between weapons either by scrolling on the mouse or through keyboard inputs. A code snippet is provided below:

```
public class WeaponSwitcher : MonoBehaviour
{
    [SerializeField] int currentWeapon = 0;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        SetWeaponActive();
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        int previousWeapon = currentWeapon;
        ProcessKeyInput();
        ProcessScrollWheel();

        if(previousWeapon != currentWeapon)
        {
            SetWeaponActive();
        }

    }

    1 reference
    private void ProcessScrollWheel()
    {
        if(Input.GetAxis("Mouse ScrollWheel") < 0)
        {
            if (currentWeapon >= transform.childCount - 1)
            {
                currentWeapon = 0;
            }
            else
            {
                currentWeapon++;
            }
        }

        if (Input.GetAxis("Mouse ScrollWheel") > 0)
```

Figure 39: A figure describing the WeaponSwitcher C# program.

3. Functionalities

There were several key functionalities added to the game that allows the player to feel more immersed into the environment and they are provided below:

3.1 Vision

In terms of vision the game has the following features:

- Terrain with grass, trees, and hills
- Abandoned Asylum
- Abandoned health containers
- Ammo placements
- Battery placements
- Syringe placements
- Infected Avatars
- NPC Avatars
- First person character with a syringe/weapon

An example figure below summarizes the vision aspect of the game:

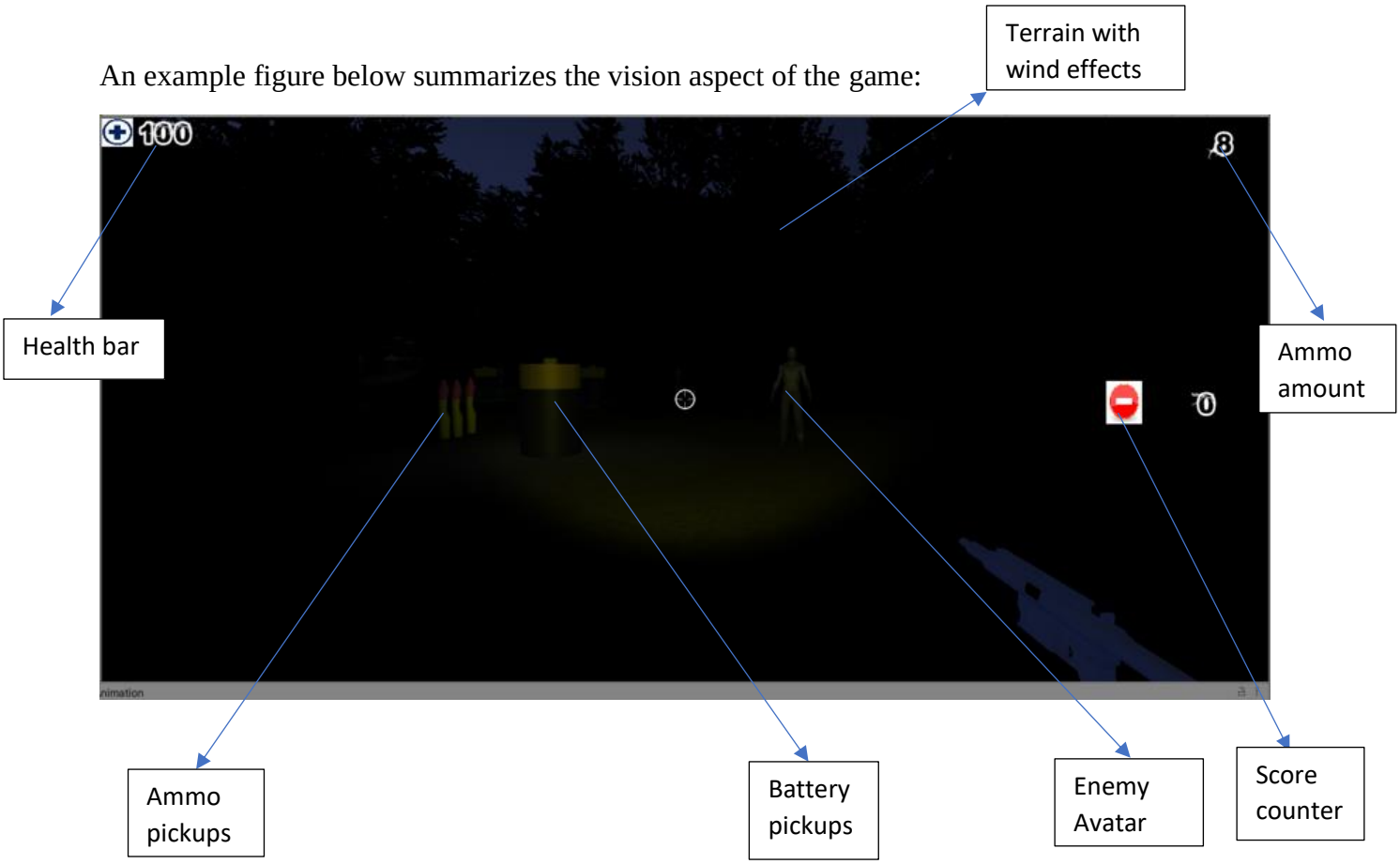


Figure 40: Example screenshot obtained upon play mode.

3.2 Sound

There is an eerie music playing in the background while in the asylum and in the terrain along with sounds of winds, ocean waves, grass, and random hallucinogenic sounds. The weapons or syringe have their own sound effect implemented to them, and they are played upon firing. The infected avatars have their own sound effects such as howling or growling when they find the player. There are footstep sounds in the terrain and the asylum.

3.3 Animation

There are animations associated to the enemy avatar along with several NPCs. The enemy avatars have an idle, walk, and attack animations as provided in the figure below. The NPCs animations range from terrified to eating. There are animations associated to the terrain itself such as moving of the trees, grass, etc. When the player approaches a door there is a door opening animation.

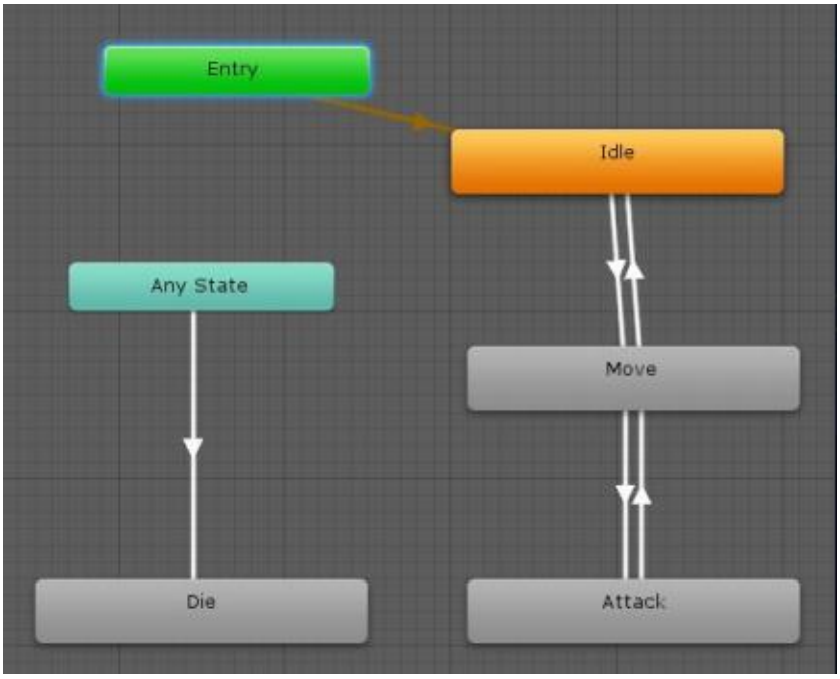


Figure 41: Animation path for zombie avatar.

Figure 42: Attack animation from the zombie avatar along with NPC's terrified animation. The speaker around the avatars verify that a sound effect is implemented to it.

3.4 Interactivity

There are several trigger events in the game. When the player approaches close to the asylum door it transitions the player from the terrain environment to the asylum. Similarly, there are triggers placed in the asylum and the terrain which upon entering produces footstep sounds. Also, when the player attacks the enemy avatar, it will cause the enemy avatar to come towards the player. Similarly, when the player is within the proximity AI of the infected avatar, it triggers the infected to come towards the player. When the player dies, it triggers a game over scene and will prompt the player to restart the game or exit the game. Also, when the player picks up a battery, it triggers the flashlight luminosity to increase in value and angle. Likewise, when the player's approaches close to one of the NPCs in the mess hall or cinema room it triggers a voice that guides the player. Examples of triggers are given below:

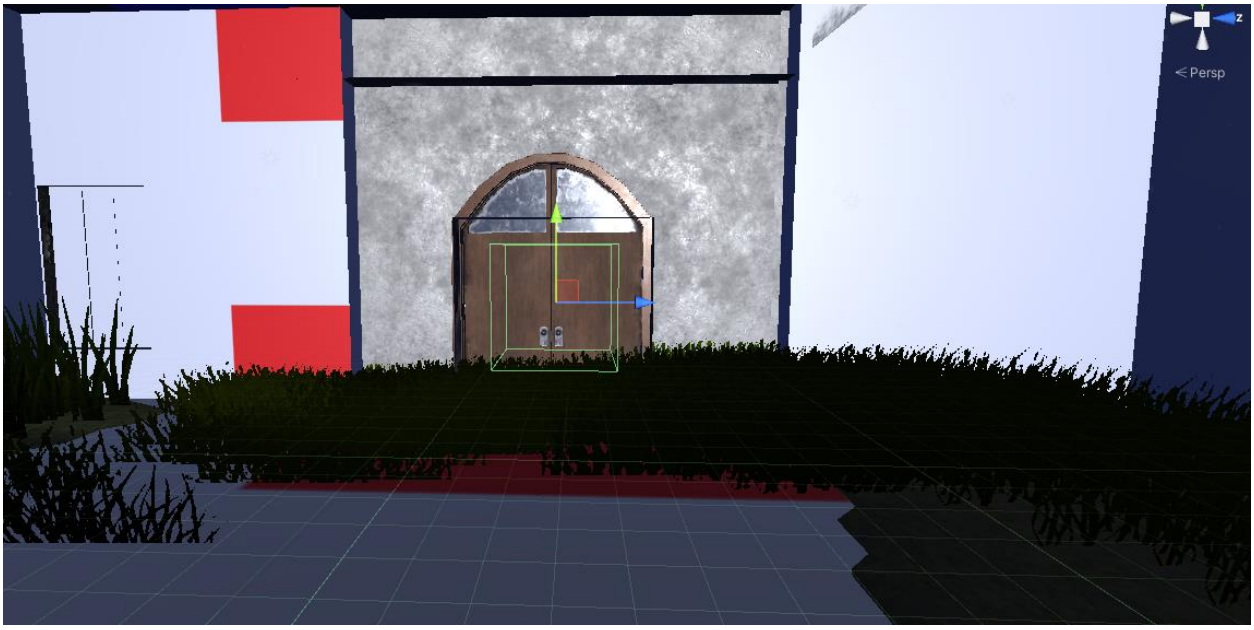


Figure 43: A trigger event that upon entering changes the scene from the terrain to the asylum.

Figure 44: When the player's health approaches zero, it triggers a game over screen.

3.5 Sensor

There are several sensors in the game. The infected avatar has a proximity AI associated to them which are constrained to a radius, and so when the player comes closer to that radius, it triggers the infected to come towards the player. There are proximity sensors around the pickups, so when the player approaches close to the pickups it destroys the gameobject notifying the player that they have indeed picked up something. Also, the flashlight's luminosity reduces overtime which will notify the player to pickup more batteries to maintain the flashlight's radiance.

Figure 45: The player's flashlight decreases overtime.

3.6 Avatars

There are several avatars in the game which include infected avatars along with NPCs with different animations. When the player approaches one of the NPC avatars in the mess hall and cinema hall, and presses the T key on the keyboard, the NPC provides guidance on where the player needs to go next.

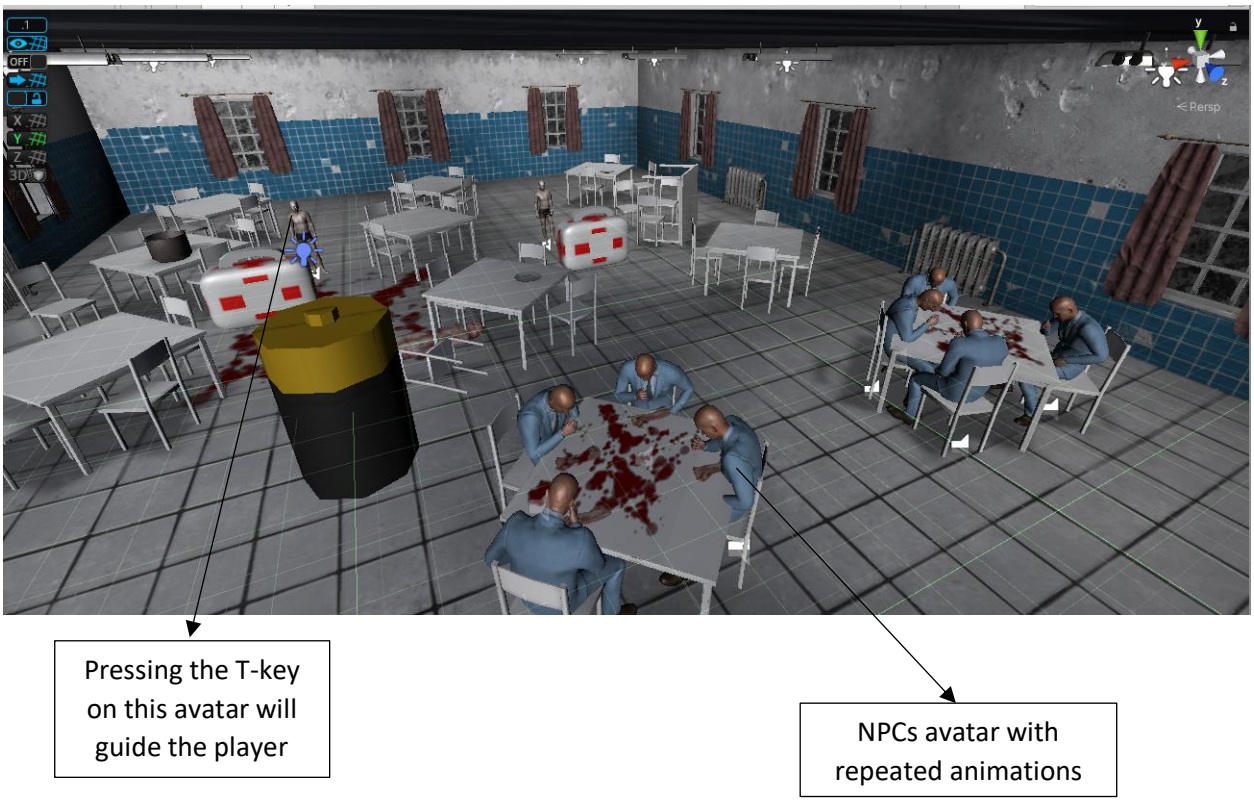


Figure 46: Avatar with animations. Pressing the T-key on one of the avatars guides the player.

4. Unity3D Functionality

- Lights: This game has a darker theme associated to it so there are less lights. The only proper source of light is the player's flashlight as described in figure 45. There are also point lights added on the batteries for player's guidance.
- Timers: There is a health bar associated to the player which will drain based on enemy avatar attacks. There is also an ammo counter, which is based on the amount of doses/weapon fired on the enemy avatar. There is also a score counter which provides a feedback on how many enemy avatars have been vaccinated, as described in figure 40.
- Keyboard functionalities: There are keyboard inputs for each action conducted by the player. For example, the spacebar will be used for jumping, left and right mouse buttons for shooting, w to

walk, s to walk backwards, shift+w to run, etc. Also, while in the asylum, pressing the T-key to one of the avatars in the mess hall and cinema room will guide the player on where to go next.

- Proximity Sensor: Each enemy has a proximity AI associated to them, when the player enters that proximity radius it will trigger the enemy to come towards the player. Similarly, when the player is close to a pickup it destroys the gameobject notifying the player that they have picked up something.
- Avatar and Avatar Animation: There are fifteen to twenty NPC avatars along with enemy avatars, with most of them having repeated animations from running to talking to attacking the player.
- Environment: There is sky, moon, grass, trees, and a lake added to the terrain. In the asylum there are blood spots around the asylum along with dimmed lights.
- Audio: There is an eerie music playing in the background, along with wind sounds, footstep sounds, the infected sounds, and gun fire sounds, along with sounds coming from NPCs. This applies to both the terrain and the asylum.

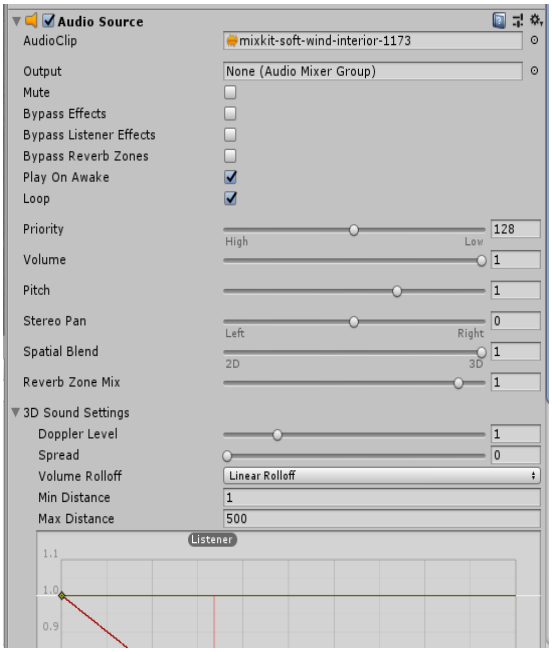


Figure 47: An example of an audio source added to the terrain.

5. Avatar behaviors

The enemy avatar's behavior ranges from idle to attack. When they are either provoked or when the player is within the proximity AI their behavior changes from idle to move to attack if they are close to the player. Likewise, for the NPCs the behaviors range from terrified to eating. They are several NPCs who are terrified and are screaming and shouting. There are several who are in pain or are hallucinating, and lastly there are several NPCs who are viciously eating.

Figure 48: NPCs viciously eating at the mess hall.

6. Conclusion

As mentioned in the goals and introduction section of this report, this game module is currently in its pre-alpha state. There is still a lot more work that needs to be done. I am hoping to implement more sensory features such as flashing of the health bar when the player's health approaches close to zero or the flashing of the ammo bar, when the bullets or the doses reach close to zero.

I am also hoping to improve the overall graphics of the game (if it is possible in Unity) to make it more interesting and real. There will be puzzle elements in the game that will enhance the player's problem-solving capabilities, along with information pertaining to genetics, microbiology, and DNA synthesis. I am also hoping to implement a cutscene at several points in the game to make it more interactive and entertaining.

As discussed in the target audience section of this report, currently there are not many games in the market that have a psychological horror theme implemented to them, and it is a rarity when it comes to implementing VR capability to these genres of videogames, which is one of the reasons why companies like Xbox game studios are seeking independent developers to help make these genres of games for their platforms.

7. References:

Assets:

- https://www.turbosquid.com/3d-models/aid-kit-fbx-free/622126
- https://www.turbosquid.com/3d-models/3d-model-pharmacy-sign-1174966
- https://www.turbosquid.com/3d-models/corpse-creepiness-model-1540195
- https://www.turbosquid.com/3d-models/scanned-man-surgical-doctor-3d-model-1624551
- https://www.turbosquid.com/3d-models/3d-character-medical-worker-uniform-model-1703700
- https://www.turbosquid.com/3d-models/3d-packed-meat-model-1486220
- https://www.turbosquid.com/3d-models/cartoon-injector-3ds-free/411969
- https://www.turbosquid.com/3d-models/syringe-c4d-free/505091
- https://www.turbosquid.com/3d-models/victim-patient-obj-free/834793
- https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232
- https://www.cgtrader.com/free-3d-models/interior/hall/asylum-hallway

Animations:

- https://www.mixamo.com/#/

Code:

- https://answers.unity.com/questions/938221/basic-enemy-ai-in-c.html
- https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html
- https://forum.unity.com/threads/ammo-pick-up.253459/
- https://answers.unity.com/questions/589666/how-to-switch-weaponsc.html
- https://answers.unity.com/questions/977751/how-to-make-a-flashlight-toggleable-in-c.html
- https://learn.unity.com/tutorial/let-s-try-shooting-with-raycasts