ZOMBIE OUTBREAK IN DISCOVERY PARK BUILDING

Group Members - Bala Chikkala

Naga Suchandra Tirumalasetti

Srikar Duriseti

Course Title- Special Problems

Course Number- INFO 5900

Faculty- Dr. Sharad Sharma

ABSTRACT

This project is about developing a virtual reality game in which the player must fight off a zombie outbreak in the UNT Discovery Park Building. The player must move around the building and get rid of all the zombies using the Unity game engine, which creates a realistic and immersive environment. Fans of first-person shooter games who enjoy the horror and action genres make up the target market for this game.

The goal of this project is to demonstrate the capabilities of virtual reality technology while also offering a fun and exciting gaming experience. In a secure and controlled setting, the game enables the player to feel the thrill and danger of a zombie outbreak. In addition to being entertaining, the game shows the potential of virtual reality technology in several different fields. The game can be modified for training and simulation purposes in disciplines like disaster preparedness and emergency response.

Overall, this project shows how virtual reality technology could produce captivating and immersive gaming experiences while also having real-world applications in other fields.

INTRODUCTION

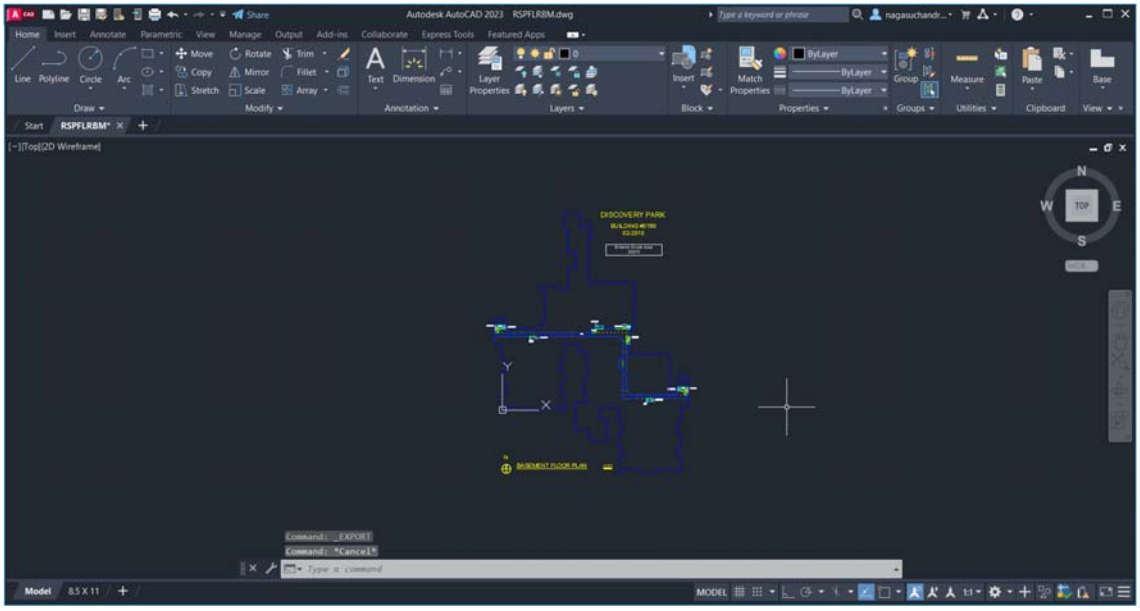
In this game, gamer will find himself surrounded with zombies in UNT Discovery Park building. Your main objective is to stay alive and fight zombies in level 1. But beware, the alive are not the only threat in the building. As you progress, you may uncover clues and information that reveal the cause of the outbreak and difficulty increases with better visual experience. However, the ultimate goal is always to survive and accomplish the tasks.

IMPLEMENTATION

Converting a 2D plan of Discovery Park into a 3D model using AutoCAD

- We are importing the 2D plan of Discovery Park in AutoCAD.
- Creating a 3D workspace: This workspace has different commands and tools for creating 3D models. You can access this workspace by clicking on the "3D Modeling" workspace icon on the status bar.
- Extrude the plan: The next step is to extrude the 2D plan to create a 3D model. These commands allows to convert 2D shapes into 3D objects by extruding them along a specified path and length . You can select the plan and use these commands to create a 3D model.
- Adding Materials: Once you have created a basic 3D model, you can add details such as textures, materials, and lighting to make it more realistic. AutoCAD provides different tools for adding these details.
- Check for accuracy: It's important to check your 3D model for accuracy. You can do this by comparing it to the original 2D plan and making sure that all the dimensions and details are correct.
- Save the 3D model: Once you are satisfied with your 3D model, you can save it as a separate file. AutoCAD supports different file formats for 3D models, such as DWG and STL.

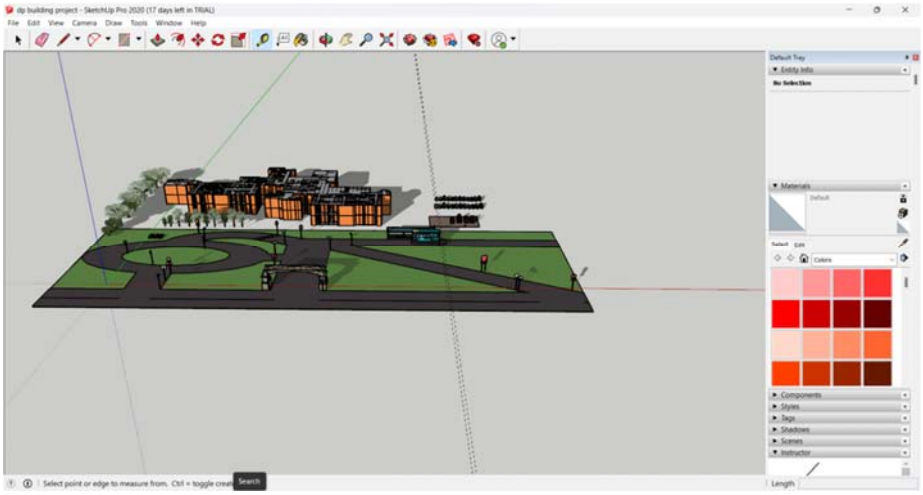
In conclusion, the process of converting a 2D plan of Discovery Park into a 3D model using AutoCAD involves extruding the plan, adding details, checking for accuracy, and saving the model in a compatible file format.



Exporting the 3D model from AutoCAD to Sketch Up:

- The first step is to export the 3D model from AutoCAD in a compatible file format that can be imported into Sketch Up. One such format is the Collada (.dae) format. To export the 3D model, select it in AutoCAD and use the "EXPORT" command to save it as a .dae file.
- Import the 3D model into Sketch Up: Once you have exported the 3D model from AutoCAD, you can import it into Sketch Up. To do this, open Sketch Up and go to "File" > "Import". Select the .dae file that you exported from AutoCAD and click "Import".
- Position the 3D model in SketchUp: When you import the 3D model into SketchUp, it may not be positioned correctly. You may need to use the "MOVE" and "ROTATE" tools to adjust its position and orientation.
- Add other 3D models: Once you have imported the 3D model of Discovery Park into SketchUp, you can add other 3D models to it. To do this, you can either create your own 3D models in SketchUp or import them from 3D warehouse. SketchUp supports different file formats, such as .skp, .obj, and .3ds.
- Arrange the 3D models: Once you have added all the 3D models that you want, you can arrange them in the scene. You can use the "MOVE", "ROTATE", and "SCALE" tools to adjust the position, orientation, and size of the 3D models.
- Apply materials and textures: To make the 3D models look more realistic, you can apply materials and textures to them. SketchUp provides a wide range of materials and textures that you can apply to your 3D models.
- Save the scene: Once you are satisfied with your scene, you can save it as a SketchUp file (.skp) or export it in other formats, such as .dae or .obj.

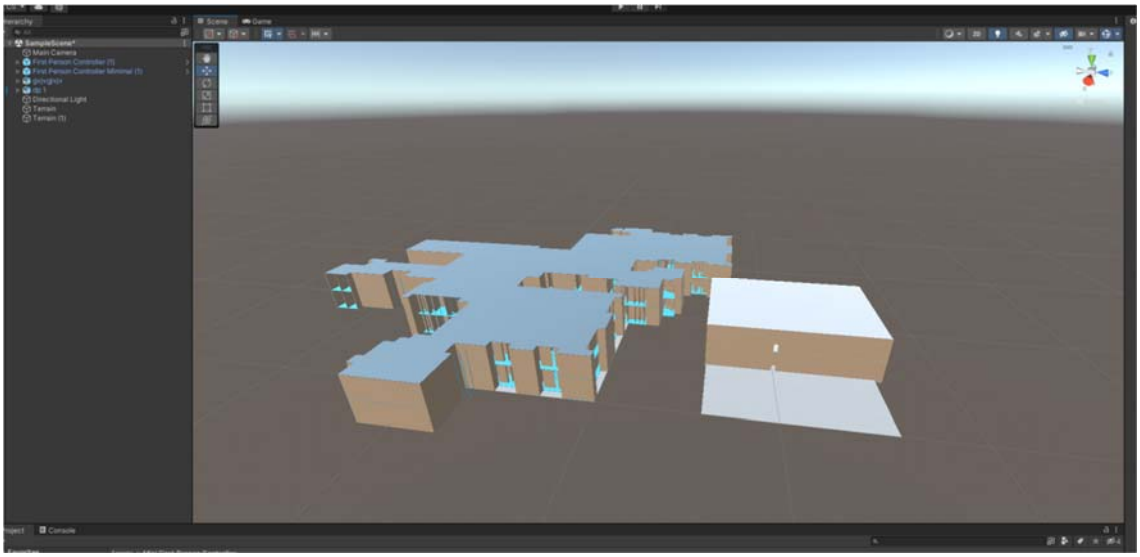
In conclusion, importing the 3D model of Discovery Park from AutoCAD into SketchUp 2020 and adding other 3D models involves exporting the 3D model from AutoCAD, importing it into SketchUp, positioning it, adding other 3D models, arranging them, applying materials and textures, and saving the scene.

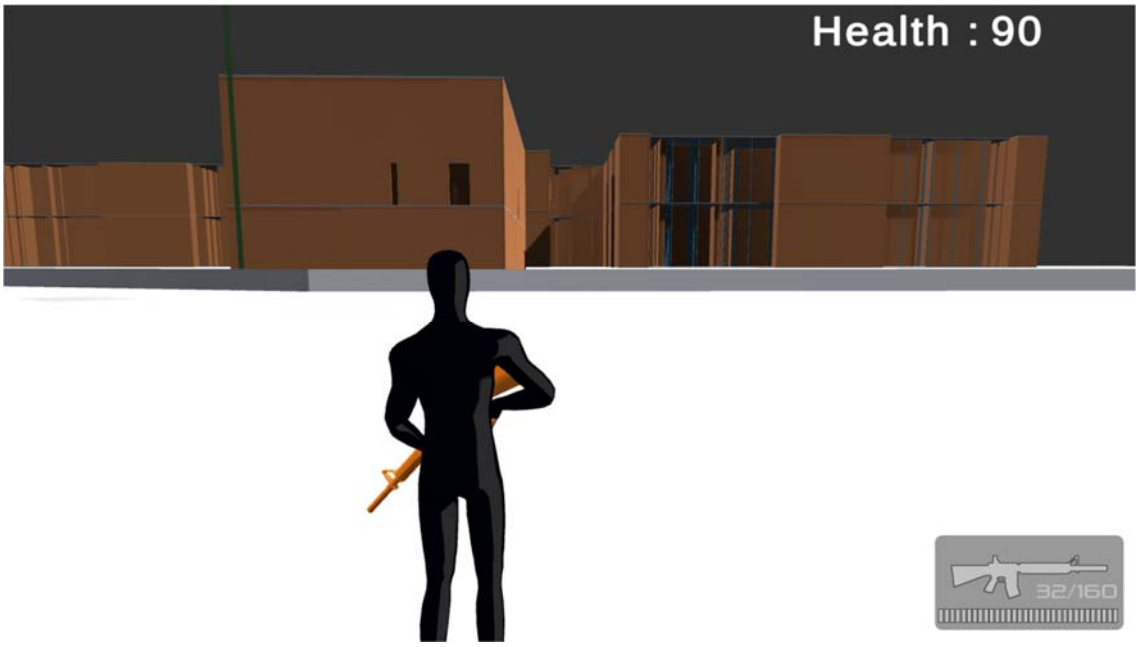


Importing a Sketch Up file into Unity:

- Save the SketchUp file: The first step is to save the SketchUp file in a compatible format that can be imported into Unity. One such format is the FBX(.fbx) format. To save the SketchUp file in the FBX format, go to "File" > "Export" > "3D Model". Select the "FBX File (*.fbx)" option and click "Export".
- Create a new Unity project: Open Unity and create a new project. Give it a name and select a location to save it.
- Import the SketchUp file into Unity: Once you have saved the SketchUp file in the FBX format you can import it into Unity. To do this, go to "Assets" > "Import New Asset". Select the .FBX file that you exported from SketchUp and click "Import".
- Adjust the scale and position of the model: When you import the SketchUp model into Unity, it may not be scaled or positioned correctly. You may need to adjust the scale and position of the model using the "Transform" component in the Inspector window.
- Add materials and textures: To make the SketchUp model look more realistic, you can add materials and textures to it in Unity. You can do this by selecting the model in the Scene window and opening the Materials tab in the Inspector window.
- Save the scene: Once you are satisfied with your scene, you can save it by going to "File" > "Save Scene". Give it a name and select a location to save it.

- Build and run the project: Once you have saved the scene, you can build and run the project to see the SketchUp model in action. To do this, go to "File" > "Build Settings" and select the platform that you want to build for. Click "Build" to create the executable file.





ACKNOWLEDGEMENT

Creating a game requires a collaborative effort, and we would like to express our gratitude to the following individuals and organizations for their contributions to this project:

- Our development team for their tireless efforts in designing, coding, and testing the game.
- The University of North Texas for providing us with access to the UNT Discovery Park building, which served as the inspiration for the game's setting.
- OpenAI for providing us with the GPT-3.5 architecture, which powers the language capabilities of ChatGPT, the AI assistant who helped us create this game introduction.
- Our playtesters, who provided valuable feedback and helped us fine-tune the game mechanics.
- Finally, we would like to thank the gaming community for their support and enthusiasm, which has inspired us to create this immersive and exciting game.

Thank you all for your contributions to this project. We couldn't have done it without you.

REFERENCES

• GitHub - joeaoregan/LIT-Yr4-DigitalGameEngines: Zombie Apocalypse: 3D Unity game based on the theme Shoot 'em up. With VR support for Level 3

- Zombie Apocalypse Trailer - YouTube
- https://github.com/topics/zombie

•https://www.researchgate.net/publication/341287405_Design_and_Implement_an_Artificial_Intelligence_based_Zombie's_Application_using_Unity3D

GOAL & OBJECTIVES

MODELLING

All the floors, rooms, and halls of the UNT Discovery Park Building should be represented in the virtual environment. Also designed in SketchUp and Unity. To evoke a sense of peril and anxiety, the setting should have a dark, ominous mood. Doors that can be opened and shut, windows that can be shattered, and objects that can be moved or used as weapons should all be included in an interactive setting. Players should be able to explore and navigate the terrain via a variety of paths and routes, opening options for both covert and aggressive gaming. The environment should include interactive elements such as computers, files, and clues that can be

discovered and used to uncover the cause of the zombie outbreak and progress through the game.

PROGRAMMING

The application was programmed in Unity 3D and c#.

PlayerCasting2.cs

C: > Users > bnc0168 > Documents > LIT-Yr4-DigitalGameEngines-master > Assets > Scripts > PlayerCasting2.cs

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

public class PlayerCasting2 : MonoBehaviour {

    public float DistanceFromTarget;
    float ToTarget;

    // Update is called once per frame
    void Update () {
        RaycastHit hit;

        if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit)) {
            ToTarget = hit.distance;
            DistanceFromTarget = ToTarget;
        }
    }

    public float getDistanceFromTarget() { return DistanceFromTarget; }                    // Access the distance to a target in OpenDoubleDoors script
}
```

Fig- **Player Casting**

C: > Users > bnc0168 > Documents > LIT-Yr4-DigitalGameEngines-master > Assets > Scripts > LevelComplete.cs

```
// Joe O'Regan
// Levels 1 and 2
// Handle Progressing Between Levels When Level Complete

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

public class LevelComplete : MonoBehaviour {

    public Animator levelCompleteAnimation;
    public Canvas playerHUD;

    GameObject player;
    GameObject gc;

    void Start(){
        player = GameObject.FindGameObjectWithTag ("Player");          // Locate the Player
        gc = GameObject.FindGameObjectWithTag ("GameController");      // Locate the Game Controller
    }

    // When the player enters the trigger point, load level 3
    void OnTriggerEnter (Collider other)                               // Called whenever anything goes into a trigger
    {
        if(other.gameObject == player)                                 // Make sure it is the player we are attacking
        {
            if (SceneManager.GetActiveScene().buildIndex == 3)
                Debug.Log ("Level 1 Complete");
            else if (SceneManager.GetActiveScene().buildIndex == 4)
                Debug.Log ("Level 2 Complete");

            gc.GetComponent<ExitLevel> ().SaveState ();                // Save state between levels

            //SceneManager.LoadScene(5);                               // Load level 3

            playerHUD.enabled = false;
            levelCompleteAnimation.SetTrigger("Fade");
        }
    }
}
```

Fig- **Level Complete**

C: > Users > bnc0168 > Documents > LIT-Yr4-DigitalGameEngines-master > Assets > Scripts > FlameAnimations.cs

```
/*
    Creates random light flicker effect on the torch object
*/

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

public class FlameAnimations : MonoBehaviour {

    public int LightMode;
    public GameObject FlameLight;


    void Update () {
        if (LightMode == 0)
            StartCoroutine (AnimateLight());
    }

    IEnumerator AnimateLight(){
        LightMode = Random.Range (1, 4);     // Never generates the maximum number
        if (LightMode == 1) {
            FlameLight.GetComponent<Animation> ().Play ("TorchAnim");
        }
        if (LightMode == 2) {
            FlameLight.GetComponent<Animation> ().Play ("TorchAnim2");
        }
        if (LightMode == 3) {
            FlameLight.GetComponent<Animation> ().Play ("TorchAnim3");
        }
        yield return new WaitForSeconds (0.99f);
        LightMode = 0;  // So Coroutine can be called again from update()
    }
}
```

Fig- **Flame Animation**

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class ZombieManager : MonoBehaviour
{
    public GameObject Target_;
    public float health;
    void Update()
    {
        float distance = Vector3.Distance(gameObject.transform.position, Target_.transform.position);
        if(distance <= 2)
        {
            gameObject.GetComponent<NavMeshAgent>().enabled = false;
            GetComponent<Animator>().Play("attack");
        }
        else
        {

            gameObject.GetComponent<NavMeshAgent>().enabled = true;
            gameObject.GetComponent<NavMeshAgent>().SetDestination(Target_.transform.position);
            GetComponent<Animator>().Play("walk");
        }
        if(health <= 0)
        {
            GetComponent<Animator>().Play("die");
            gameObject.GetComponent<NavMeshAgent>().enabled = false;
            Invoke("Off", 2f);
        }
    }
    void Off()
    {
        gameObject.SetActive(false);
    }
}
```

Fig- **Zombie Manager**

FUNCTIONALITY

- **VISION**

  A first-person perspective can help the game immerse the user in the setting. For assistance in navigating the pitch-black building, the player might carry a flashlight or night vision equipment. The player may need to use their vision to spot potential hazards because zombies can be hiding in corners or be visible in the distance.

- **SOUND**

  Sound could be utilized to build suspense and provide the player with audible cues. The background music might be ominous and menacing while zombies could make groans and snarls. When the player is in danger, their breathing might quicken, they might become more terrified, and the sound of their own footsteps might help them hear zombies.

- **ANIMATION**

  The game might be given life and made more immersive by using animations. For instance, when a zombie is shot, the player may witness blood splatter and the creature's body falling to the ground. To feel more immersed in the game world, the player's own movements could be animated.

- **INTERACTIVITY**

  A variety of interactive features could be included in the game to keep the user interested. To move through the building, for instance, the player could need to look for guns and ammo, scavenge for food and supplies, and solve riddles. Moreover, the player may be able to barricade windows and doors to keep zombies out or utilize other objects as weapons.

- **CHARACTERS/AVATARS**

  Players can design their own characters by customizing the avatars in the game. To offer players varied gameplay experiences, these avatars might also have various traits like speed, strength, and endurance.

- **SENSORS**

  The usage of sensors could result in a more lifelike and immersive experience. Haptic feedback, for instance, may be used in the game to replicate gun recoil or the force of a zombie onslaught. As the player navigates the hazardous environment, the game may also employ biometric sensors to monitor the

player's respiration and heart rate. It may also use motion sensors to detect the player's motions and modify the game accordingly.

- **PLAYER**

  The application uses a player controller, such as a First-Person Controller to provide a realistic simulation of the situation.

- **AI IMPLEMENTATION**

  By utilizing AI capabilities such as navigation, behaviours, and the quickest path, the software simulates human behaviour in an emergency circumstance.

- **INTERFACE ELEMENTS**

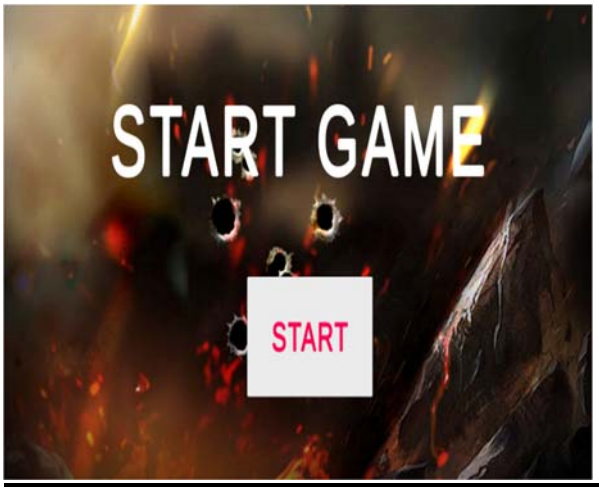
  The surroundings should have interactive features like computers, files, and hints that may be uncovered and used to advance through the game and figure out what caused the zombie outbreak.

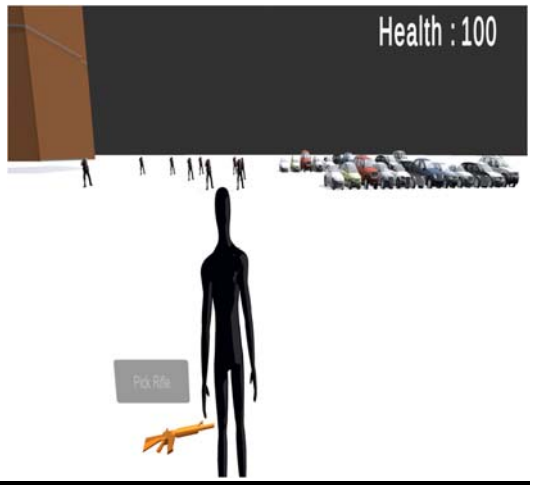
BEHAVIOR IMPLEMENTED FOR DIFFERENT AGENTS/AVATARS

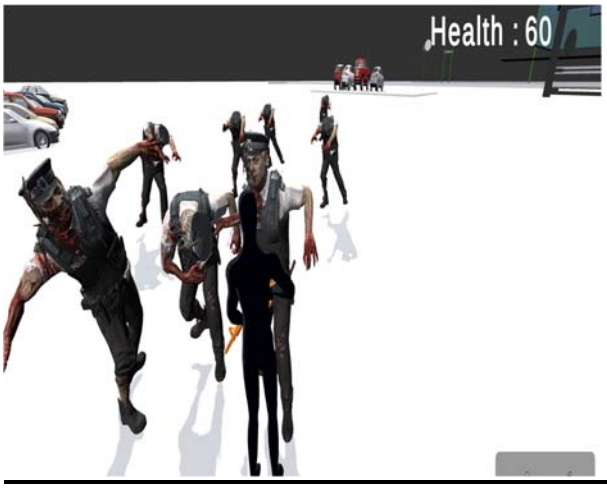
Player: To Pick the rifle click the 'E' key on the keyboard and shoots the zombies by aiming using the mouse.

To reload the rifle press the 'R' Key.

Zombies: When the zombies touches the player the player health gets reduced by 10 points.







WHY THIS APPLICATION IS USEFUL & WHY VR is APPROPRIATE TECHNOLOGY FOR THIS PROJECT.

Unity was used to create the Zombie Outbreak survival application, which has a number of practical uses. For starters, it can be used as a training tool for anyone who wish to learn how to react in an emergency or crisis situation. Second, it can be used as a planning and practice tool for evacuation procedures by emergency personnel. It can also be used as a teaching tool to spread knowledge about emergency planning and response.

This project would benefit from using virtual reality (VR) because it provides a high level of immersion and interactivity. Users can interact with a computer-generated environment that closely simulates a real-world crisis situation, which can improve their engagement and knowledge retention. Users of the Zombie breakout Evacuation program can practice handling a zombie breakout scenario in a secure setting.

Realistic and complex environments can be made using Virtual VR technology. The Zombie Outbreak Evacuation app's ability to faithfully simulate actual locales can help with training and simulation. Additionally, VR technology enables the blending of various interactive components like audio, video, and haptic feedback, which can enhance the simulation's realism and level of immersion.

Explain problems encountered and remaining shortcomings (future work)

1. We must conduct research and compile information in order to construct a convincing and engaging virtual reality experience. This entails looking for examples of genuine zombie behaviour, learning about various weapons and survival techniques, and investigating the surroundings in which the outbreak occurs.
2. Create a plot that moves the project along. To keep the user interested, this may include backstory, character development, and plot twists. Geometrics, textures and functionality can be decided based on the plot of the game.
3. Create and implement user interfaces that let players access and navigate the game.
4. We can choose the gameplay elements to use in the game. This can include things like managing your inventory, using combat techniques, and solving puzzles.