ENGR-UH 1000

Computer Programming for Engineers

Term Project

Virtual Reality Fire Drill Simulator

Software Report

Emmanuel Fashae (eif213)

Kenechukwu Ezeifemeelu (kee301)

Kojo Vandyck (kev276)

Fall 2019

INTRODUCTION

This project is a fire drill simulation with the purpose of informing the users how to act in the scenario of a fire drill. According to the Center for Fire Statistics, fire outbreaks cause approximately 20.7 million deaths annually and are cause many untimely deaths and injuries. Whether in the form of a major gas explosion on an oil rig or a small kitchen outbreak, fire incidents occur everywhere. However, when it comes to reacting to a fire outbreak, large groups are usually slow to respond or ignorant on how to respond.

Hence, the need for fire drills in large or medium corporations became necessary in order to save lives in case of an emergency, but there still are some inefficiencies in the traditional fire drills. Therefore, the use of a fire drill simulator can reduce these inefficiencies to some extent.

Why VR Fire Drills Over Traditional Fire Drills

1. **VR drills are more engaging** - Firstly, the monotony of the normal fire drills creates a situation that renders people reluctant to comply to the drill. These people believe that whenever the fire alarm rings, it is just a fire drill and not a real fire. Thus, in the case of a real fire, these individuals will not respond accordingly, endangering lives. However, by practicing in VR, the individual can easily separate reality from practice.
2. **Enough Practicality (More Engaging)** - Secondly, these conventional fire drills are not realistic enough to give the partakers the required sense of emergency required in a fire outbreak. Most of the time, these people just follow the instructions in order to complete the task and not because they want to learn about how to react in case of a fire. However, with our interactive fire drill simulator, the individual will be more involved and thus, learn more.
3. **VR drills are less time consuming** - Individuals can schedule appointments to practice fire drills on VR at conventional times. The fulfillment of this fire drill can be made a requirement in the organization. This solves the problem of having random fire drills at inconvenient hours.
4. **Multiple scenes can be generated** - The VR fire drills can be used to practice fire drills in different locations, opposed to the conventional fire drill that is usually run repeatedly in the same location.

Risks of Virtual Reality Fire Drills

It is important to know that VR is associated with some health risks and should not be used by people suffering from specific medical conditions such as photosensitive epilepsy. Some of these health risks are anxiety, dizziness, nausea, eye strain, and radiation exposure.

However, in all, for short periods of time, Virtual Reality is relatively safe, and its advantages outweigh its disadvantages.

Project Objectives

1. To teach the user how to react and respond in the case of a fire extinguisher
2. To teach the user how to put out fires.
3. To teach the user how to calmly exit the scene of the accident
4. To instill a sense of urgency in a fire emergency through Virtual Reality
5. To create a more engaging way to learn how to act in case of fire
6. Learning how to use Unity Software, the Oculus Rift, C# language and generally Virtual Reality in Specific Situations in Life

Project Development

The equipment used for this project include the Oculus Rift headset, the Oculus touch left and right-hand controller, two Oculus sensors and an Alienware laptop.

Use of Each Device

1. Left and Right Hand Touch Controllers: This device enabled user interaction with game objects.
2. Oculus Sensors: These enabled the Unity Software to track user movements.
3. Oculus Rift Headset: This device enabled visualization of game environment.
4. Alienware laptop: This device's QUAD HD 17-inch display, 4GB GDDR5 standard memory, 8^{th} Gen Intel Core processor and optimum gaming graphics made this the ideal laptop.

Oculus Rift Setup

1. Download and install Unity Software
2. Plug-in and configure headset
3. Set-up sensor

- Create play area
- Configure standing height
- Placing sensors: Make sure play area is at least 3 by 3 feet. Make sure nothing is blocking line of sight between headset and sensors. Sensors are about 6 feet apart from each other.

4. Set up touch controllers

   To achieve optimum PC-gaming graphics, the left and right hand controllers had to be calibrated with the sensors to obtain a workable range or the user and the oculus signal to operate.

5. Set up guardian system

The Guardian system maps out where you will be using the Rift and warns the user when user is near boundaries of that area.

Assets Used

Some of the assets were gotten from the Oculus Integration package and VRTK plugin.

1. 3D Models
2. Environment
3. Fire Extinguisher Model
4. Start Message
5. Second Message
6. Fire
   - Trigger Cubes
7. Extinguisher Fumes
   -Projectile
8. Number Pad
9. Custom Hands
10. Images
11. Congratulatory Message
12. Images: -The congratulatory message was sampled from Pinterest.

Audio

1. Fire Extinguisher Spraying Sound
2. Fire Alarm Sound

   To produce the fire alarm sound, this fire alarm sound (https://www.youtube.com/watch?v=QYBVHJm6tS4&t=10s) was juiced from YouTube using mp3juices.cc
3. Crackling Fire Sound

The fire crackling sound was also juiced from YouTube using mp3juices.cc

Code Breakdown

1. **Menu Script**

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

public class Menu : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
       Destroy(gameObject, 10);

    }

    // Update is called once per frame
    void Update()
    {

    }

}
```

Function: This script was to destroy a UI panel that had been activated at the start of the game.

Explanation: The game object here is the UI panel and it was meant to be destroyed after 15 seconds. The 15 seconds by our calculations is enough time for the user to read the game directives on the panel. This function is only executed at the start of the game.

2. **Particle Launcher Script**

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

public class ParticleLauncher : MonoBehaviour
{
    // Start is called before the first frame update
    public ParticleSystem particleLauncher;
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetButton("Fire1"))
        {
            particleLauncher.Emit(1);
        }
    }
}
```

Function: This script was to perform the “extinguishing” function of a fire extinguisher.

Explanation: Upon hitting the ‘A’ button, tiny box colliders (mesh renderer removed) are emitted from the fire extinguisher which destroy the fire upon collision. This function is updated once per frame which gives it the fire extinguishing effect of multiple particles being emitted.

3. **Projectile Launcher Script**

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

public class Launcher : MonoBehaviour
{
    // Start is called before the first frame update
    [SerializeField]
    private Transform firePoint;
    [SerializeField]
    private Rigidbody projectilePrefab;
    [SerializeField]
    private float launchForce = 700f;
    void Start()
    {

    }

    // Update is called once per frame
```

```
    void Update()
    {
        if (Input.GetButtonDown("Fire1"))
        {
            LaunchProjectile();
        }
    }
    private void LaunchProjectile()
    {

            var projectileInstance = Instantiate(
                projectilePrefab,
                firePoint.position,
                firePoint.rotation);

            projectileInstance.AddForce(firePoint.forward * launchForce);

    }
}
```

Function: This script was to make the fire extinguishing more realistic by deactivating the fire crackling sound upon fire quenching.

Explanation: This script uses a setActive function which deactivates sound upon fire quenching. If fire is active, setActive activates sound.

4. Collision Script

```
0 references
void Start()
{//Disable Instructions After Putting Off Fires
    window.SetActive(false);
}

// Update is called once per frame
0 references
void Update()
{   //Enable Instructions After Putting Off Fires
    if (Fire == null && Fire2 == null && Fire3 == null && Fire4 == null && Fire5 == null && Fire6 == null && Fire7 == null && Fire8 == null)
    {
        window.SetActive(true);
        Destroy(window, 10);
    }
}

0 references
private void OnCollisionEnter(Collision col)
{   //Destroying Fires on Contact With Extinguisher Fumes
    if (col.gameObject.name == "Enemy")
    {
        Debug.Log("Collision Detected");
            Destroy(Fire);
            Destroy(Cube);
     }
```

Function: This script is in two parts. It destroys fires upon collision and prints a timed message for the user for directives to exit the building.

Explanation:

a. A box collider was placed at the base of the fire, upon collision with the box colliders from the fire extinguisher, the fire is destroyed.

b. To do this, a ‘OnCollisionEnter’ function to detect this collision. A message printed on the Debug.Log was used to verify the function.

c. Upon quenching the fire, a UI panel is displayed for 10 seconds for directives to exit the building. The UI panel is only activated upon quenching all three fires.

5. Numpad Logic Script

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

public class NumpadLogic : MonoBehaviour
{
    public string winCode = "4236";
    protected string currentCode = "0000";
    public UnityEvent onWin;
    public TMPro.TextMeshProUGUI display;


    void Awake() {
        currentCode = display.text;
    }

    public void ButtonPressed(int val) {
        currentCode = currentCode.Substring(1) + val.ToString();
        display.text = currentCode;

        if (currentCode == winCode) {
            Debug.Log("WINNER!");
            if (onWin != null) {
                onWin.Invoke();
            }
        }
    }
}
```

Function: This script is in two parts. It obtains user input to exit the building and prints a congratulatory message upon completion of the drill.

Explanation: A fixed code to exit is saved as ‘WINCODE’ with the code being ‘1111’. If the user enters the right code, the door opens. Simultaneously, a UI panel displaying a congratulatory message is activated.

Results and Evaluation

Throughout the development process, we tested the software progressively by evaluating each aspect of the project in the Virtual Reality as it was being developed.

To evaluate the software's locomotion and ergonomics every member of the group took turns immersing themselves in the VR environment and moved around in it using the traditional mode of locomotion .i.e. walking around in the environment. However, while testing this functionality we discovered that due to high tendency of a variation between the size of our virtual environment and the amount of space available to the user it would be impossible to interact with certain parts of the environment. Hence, we implemented a teleportation functionality in our environment to eliminate this problem. The teleportation functionality allows the user to move to parts of the virtual environment that they would not be able to access if motion was restricted to real-world movement. It is activated by through the left or right thumbstick on the touch controllers and allows the user to select what point in the environment and what direction they wish to face in the virtual environment.

Also, while testing our project we discovered that designing the program with only locomotion as the only form of user interaction was unrealistic. Hence, we implemented a grabbing functionality which allows the user to grab, hold, throw and poke certain objects in the virtual environment by pushing the right touch button on the right touch controller. Whilst developing this functionality and testing it, we had difficulties picking up the objects. However, after implementing a collider and the interactable prefab to the objects, the grabbing functionality worked as expected.

Throughout the development of the project, implementing the fire-smoke extinguishing functionality proved the most challenging due to failed attempts at putting out the fire object (fire not extinguishing and extinguishing wrongly) after collision with the extinguisher smoke object. We solved this problem by implementing a collision system between a hidden box collider object at the bottom of the fire and projectile objects embedded in the fire extinguisher smoke. The fire extinguisher smoke embedded within projectiles is instantiated by the user on button press and is aimed at the base of the fire where there is a hidden box collider. As the extinguisher particles collide with the fire (box collider), the fire is gradually reduced until it is destroyed. Testing the fire extinguishing was done using the touch controllers to activate the extinguisher fumes.

After making some of the game objects interactable and adding a collision functionality we discovered that the virtual environment wasn't immersive enough due to the lack of sound. In a real-world situation during a fire the fire-alarm goes off, the fire makes crackling sounds and the fire extinguisher makes a sound when it is being used. This prompted us to add fire alarm, fire crackling and fire extinguisher sounds to the virtual environment, fire and fire extinguisher game objects, respectively. After adding the sound objects, we observed that the fire extinguisher sound and fire sounds were still active even after their respective objects were deactivated. Hence, we made changes to the scripts that activated the sound and made it such that the fire state and sound was set to true on game start but after putting out the fire, the fire state and sound was set to false (turn off the fire sound). The fire extinguisher fumes had a similar problem. By modifying the script such that on game start the fire extinguisher fumes and sound are set to false and are only set to true (active) on button hold, we were able to solve the problem.

Finally, while testing with the Oculus Rift headset and the touch controllers, we discovered that the hand presence objects were not accurate enough and did not simulate the motion of the user's hands in the real-world well enough. Hence we tested the disparity between the VR hands and the user's hands by using an index finger testing technique (touching the two index fingers together in the real-world while holding the touch controllers and wearing the VR headset to check the disparity between the real-world action and the virtual environment action). We then adjusted the hand placements such that the virtual hand movement mimicked the real-world action as realistically as possible.

Conclusion and Future Work

Conclusion

After the production of the project, our group was able to achieve the following skills:

1. A good understanding of the Unity software.
2. A grasp on how the Oculus Rift, sensors and touch controllers work.
3. Extending our programming skills and knowledge from C++ and MATLAB to C#.
4. A good understanding of public, private and even to an extent protected class objects.
5. A firm understanding of Unity-Associated Functions such as OnEnterCollision, Start, Update, Awake, etc. and functions in general added by ourselves during the project.

Future Work

Though we learned a lot from this project, much can still be done to make it more efficient in different fields. In some areas, the complexities of a real-life fire were not captured. Hence, we propose the following improvements

1. Incorporating different types of fire extinguishers to be used in the simulation. Hence, the user will learn about the different types of fires (e.g. chemical fires, electrical fires) and the proper way to put them out.
2. Widening the environment of the project and adding new environments.
3. Introduction of a timed setting that forces the user to complete the course in a specified amount of time, to enforce the emergency of the situation.

Reflection on Learning

Throughout the course of this project we learnt some valuable lessons that will be useful while undertaking future projects. The list below captures the major lessons learned.

1. This project cemented our understanding of object-oriented programming. It helped us understand better the concept of private and public class objects and accessor functions.
2. It also instilled in us best-practice programming techniques such as giving our variables, objects and scripts easily understandable names for ease of reference in the future.
3. If helps the technique of debugging our code in steps and printing short debug messages in the console using the Debug.Log function to determine if functions work as expected.
4. Understanding the physics behind rigid bodies and devising creative techniques to ensure realistic simulation in virtual reality.

5. Understanding the logic of object-oriented code as opposed to normal code that runs sequentially.