Advanced Interaction Techniques and Technologies - Summative Assessment Report

Cosmic Crisis

Background

‘Cosmic Crisis’ is a virtual reality (VR) escape room developed using the Unity game engine, where the player is placed on a spaceship which is experiencing a number of emergencies. The player has to extinguish fires, repair oxygen leaks, and replace power cores within a five-minute time limit. The purpose of the prototype is to provide an engaging and exciting VR experience that challenges players to act and think quickly in order to complete the game. It immerses the player into a sci-fi environment, and pressures players to use VR-based interactions while in a time-critical scenario. The prototype utilises locomotion techniques like teleportation in order for the player to move around, spatial audio to allow the player to identify points of emergencies and proxy objects combined with haptics to allow the player to interact and manipulate the experience’s environment. The intended audience for the prototype is teens or adult players that are seeking a short, high-intensity VR experience that situates itself in an immersive sci-fi environment that uses an extent of VR interactions to facilitate the game’s progression. A small sci-fi environment in a spaceship was chosen in order to create an imaginative but immersive scenario that suspends disbelief and facilitates unearthly interactions and challenges that are unique to this scenario, such as induced zero-gravity.

Description and Justification

Typical User Interaction

The sci-fi environment is a small spaceship in which the player can teleport to move around. In this environment, there are three main points of emergencies that the player must interact with, including a series of fires usually situated on the spaceship’s floor, oxygen leaks escaping from pipes, and faulty power cores in a small control room that controls the spaceship’s gravity system. There are six fires that need to be extinguished, seven leaks that need to be repaired, and three power cores that must be replaced. To deal with these emergencies, the player must interact with a number of items that can found across the environment. To pick up items, the player must press the back trigger. To drop items, the grip buttons on the side of the controller should be pressed. To extinguish the fires, there are three fire extinguishers placed near the door frames of the main room. These fire extinguishers can be picked up and used by holding the back trigger. The extinguishant will then be fired out the extinguisher, and when aimed at a fire for five seconds, the fire will be extinguished. To deal with oxygen leaks, a welding tool can be used. The welding tool has a similar interaction to the fire extinguisher and can be activated by pressing the back trigger, in which a small flame is fired. When the welding torch is aimed at a leak for five seconds, the leak will be repaired. Twenty seconds after the experience begins, the spaceship’s gravity system will fail. Objects and items placed across the environment will start floating, making the experience slightly more challenging with the player having to take care about where they drop their items. To repair this system, the player must interact with a number of power cores. Situated in a small room next to the large control room, the player can find

three faulty power cores, indicated by the sparks emitting from them. These faulty power cores must be replaced by the coloured power cores placed across the environment. One power core is placed in the main room, while the other two power cores are placed at the end of each corridor that connects to the main room. Each power core's colour matches the colour of the power core slot. The player must first remove the faulty power core from the slots, and then correctly place the power core into the slot. Once all three power cores are placed, the gravity system is turned back online. Once all the fires are extinguished, the leaks are repaired and the power cores are replaced, then the experience will be completed with the player direction to the game completed scene. However, the game consists of a five-minute time limit in which the player must complete all the tasks within this time. This is to provide the experience with an additional challenge of completing the game's main tasks under time pressure. If the player does not deal with all of the emergencies within the time frame, the player will be directed to the game over scene. Restarting the game is encouraged as to allow the player to learn all of the interactions to begin with, allowing them to complete these interactions quicker during a second playthrough and complete the challenge.

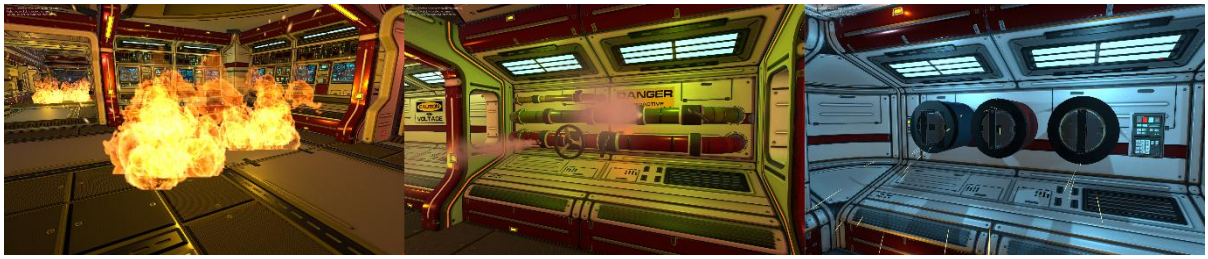

Figure 1. View of all three points of emergencies.

Choice of Interaction Technique

VR was chosen as it provided a number of interesting functionalities that could be used in order to deliver an immersive and engaging experience. Firstly, the 3-dimensional VR display provides the player with the foundation of the prototype's experience. The player must frequently look around and identify emergencies on the ship and the placement of items, especially when they are floating in zero-gravity. The immersive visuals add to the prototype as it fully embeds the player within the hypothetical scenario of dealing with an emergency on a spaceship. Traditional sedentary inputs would not be able to deliver as much of an immersive experience. The head-mounted display also provides the user with realistic spatial audio that contributes to the prototype. Fires and leaks all have their own audio sources which play when they are active. This pairs with VR's spatial audio to allow the player to precisely find where the points of emergencies are present just from the audio alone. In addition to this, the gravity system produces a sound effect when it is enabled or disabled to provide auditory feedback for the objects beginning to float. The items also utilise spatial audio to indicate their status. The fire extinguisher and welding tools produce sounds when they are being used by the player, and the power cores have a sound effect for when they are inserted into the associated slot. The spaceship also emits general sound, such as from the terminals or the ventilation systems to provide additional atmosphere that harmonises well with the spatial audio

and produces the immersive feeling of being present on a spaceship. Additionally, locomotion techniques were utilised for the prototype. The spaceship environment is far too big for the player to simply walk around. Therefore, it is required that the player has some form of traversal within the experience. Because of this, the player has the ability to teleport around the spaceship. This form of locomotion is a conventional technique that can be seen in a lot of VR games and allows the player to traverse with decreased motion sickness. The player can teleport using the controller's touchpad and by aiming at the teleportation areas on the floor. Lastly and most importantly, a number of interaction techniques using the VR controllers are utilised in order to manipulate the prototype's environment and provide feedback through haptics. The player's hand is represented with a proxy hand model. To interact with the various items, the player must physically move their hand in line with the item and grab it by pressing the trigger. This physical interaction amplifies the amount of player agency over the game's environment, creating a great sense of immersion within the experience. In addition to this, the use of haptics strengthens these interactions. When the player picks up or drops an item with a specific hand, a small vibration can be felt to indicate this action. Additionally, when the player is using an item like the fire extinguisher or the welding torch, a large vibration can be felt to provide tangible feedback of an item's state. Haptics are one of the most important features that add to the experience's immersion, especially when providing vibrational feedback when interacting with an item, making them feel tangible and realistic.

Technical Implementation

There are a number of technical implementations required for the prototype to function. The main interaction within the game is picking up, using and dropping items, like the fire extinguisher, welding tool and power cores. SteamVR's predefined scripts were explored initially, but they would not have produced the effect desired for the experience. Items needed to be interacted with, not just simply picked up and dropped. Because of this, 'PickUp.cs' details the implementation of this mechanic. Each hand game object has a trigger collider, and each item is inside of the 'PickUp' layer. When the hand's trigger collides with a game object in the 'PickUp' layer, a variable called 'pickupableObject' gets updated. If this variable is not empty and the user presses the controller's trigger, the 'PickUpItem' function is called. This function sets the items physics accordingly, and the hand's transform becomes the item's parent within the hierarchy. A 'Grip Point' transform is located on the items that can be held, which allows the item to be orientated correctly when it is picked up. When one of the controller's side buttons are pressed while the player is holding an item, the item is dropped. This resets the physics of the item and gets the velocity of the item in order to allow it to be thrown. Portions of this script was generated with generative AI. Given some of the intricacies of SteamVR's implementation, generative AI was a useful tool in understand the complexities of SteamVR and to produce code that pairs well with this predefined functionality, such as with disabling the hand mesh when an item is picked up. Generative AI was also used in order to solve some issues regarding position offsets and the throwing of items, as to ensure behaviour functioned as expected.

```
void PickUpItem()
{
    Rigidbody rb = pickupableObject.GetComponent<Rigidbody>();
    Collider collider = pickupableObject.GetComponent<Collider>();
    rb.isKinematic = true;
    collider.isTrigger = true;

    handRenderers.Clear();

    // The follow foreach loop was generated by Google Gemini in order to disable the hand mesh when an item is picked up
    // Author: Google Gemini
    // Location: gemini.google.com
    // Accessed: 17/05/26
    foreach (Renderer rend in GetComponentsInChildren<Renderer>())
    {
        // If the renderer is enabled, it's the hand. If it's disabled, it's the hidden controller.
        if (rend.enabled)
        {
            handRenderers.Add(rend);
        }
    }

    pickupableObject.transform.parent = transform;

    // The following if statement was generated by Google Gemini for calculating positon offsets
    // Author: Google Gemini
    // Location: gemini.google.com
    // Accessed: 14/05/26
    Transform grip = pickupableObject.transform.Find("Grip Point");

    if (grip != null)
    {
        pickupableObject.transform.rotation = transform.rotation * Quaternion.Inverse(grip.localRotation);
        Vector3 positionOffset = transform.position - grip.position;
        pickupableObject.transform.position += positionOffset;
    }
    else
    {
        pickupableObject.transform.localPosition = Vector3.zero;
        pickupableObject.transform.localRotation = Quaternion.identity;
    }

    isPickedUp = true;

    SetHandVisibility(false);
}
```

Figure 2. 'PickUpItem' function in 'PickUp.cs'

Each functional item has an exclusive script that determines its functionality. The fire extinguisher script becomes active when it is parented to either of the hands. When the back trigger is pressed, the extinguisher becomes active. When it is active, the visual effects of the extinguishant and haptics is produced. Additionally, a raycast is fired in the direction the extinguisher is pointing. If this raycast is fired at an object in the ‘Fire’ layer, the fire begins to be extinguished by reducing the ‘extinguishTime’ in the ‘Fire.cs’ script. This script stops the fire visuals and destroys the game object when a fire is extinguished. The welding torch and leaks have a similar functionality to the fire extinguisher. When a welding torch is held and the trigger is held, the visual effects are shown and a raycast is fired, checking for objects within the ‘O2Leak’ layer. This then reduces the ‘weldTime’ variable in ‘O2Leak.cs,’ which again will disable the leak visuals and destroy the game object.

```
void ExtinguisherFunction()
{
    RaycastHit hit;

    if (Physics.Raycast(firePoint.position, firePoint.forward, out hit, 7.5f, LayerMask.GetMask("Fire")))
    {
        Fire fire = hit.transform.GetComponentInParent<Fire>();

        Debug.Log(fire);

        if (fire != null)
        {
            fire.extinguishTime -= 1f * Time.deltaTime;
        }
    }
}
```

Figure 3. 'ExtinguisherFunction' function in 'FireExtinguisher.cs'

The power cores and power core slots each have their own scripts. ‘PowerCore.cs’ provides functionality for when a power core is placed into the slot. Each power core slot has a trigger collider which checks whether the colliding object is a power core. If so, it will call the ‘PlaceCore’ function. This then sets the core’s position inside of the corresponding slot and updates variables inside of the game manager.

```
public void PlaceCore()
{
    PickUp pickUp = GetComponentInParent<PickUp>();

    transform.position = coreSlot.position;
    transform.rotation = coreSlot.rotation;
    rb.isKinematic = true;
    pickUp.isPickedUp = false;
    transform.parent = null;

    pickUp.SetHandVisibility(true);

    audioSource.Play();

    if (gameObject.name == "Red Power Core")
    {
        gameManager.redCoreIsPlaced = true;
    }
    else if (gameObject.name == "Blue Power Core")
    {
        gameManager.blueCoreIsPlaced = true;
    }
    else if (gameObject.name == "Yellow Power Core")
    {
        gameManager.yellowCoreIsPlaced = true;
    }
}
```

Figure 4. 'PlaceCore' function in 'PowerCore.cs'

The game manager handles the state of the game. It contains a number of variables relating to the condition of the various emergencies in the game, along with the countdown. It also handles the switching gravity system and game completion. The script contains an array called ‘propsRb’, which records an array of rigidbodies in the scene which may be affected by the disabling of gravity. Inside of the ‘DisableGravity’ function, a foreach loop is used to loop through all the rigidbodies within the array. The ‘useGravity’ Boolean is disabled, along with force and torque being added to the items randomly, while also pushing items upwards. This force is added to ensure the objects move once the gravity is disabled. The visuals of the sparks is then added to the faulty power cores, and a sound is played.

```
public void DisableGravity()
{
    foreach (var rb in propsRb)
    {
        rb.useGravity = false;
        rb.AddForce(Random.Range(0f, 1f), 2f, Random.Range(0f, 1f));
        rb.AddTorque(Random.Range(-1f, 1f), Random.Range(-1f, 1f), Random.Range(-1f, 1f));
    }

    faultyCoreVfx1.Play();
    faultyCoreVfx2.Play();
    faultyCoreVfx3.Play();

    audioSource.clip = gravityOfflineClip;
    audioSource.Play();
}
```

Figure 5. 'DisableGravity' function in 'GameManager.cs'

When all the power cores are placed, the function ‘EnableGravity’ is called, again looping through all of the rigidbodies but this time enabling the ‘useGravity’ Boolean, along with another sound effect being played. As well as this, the Boolean ‘hasGravitySwitched’ is updated. This Boolean is checked along with two transform variables relating to the fires and oxygen leaks. Each of the fires and leaks are contained within a parent object. Once both of the parent objects contain no children and the ‘hasGravitySwitched’ Boolean is true, then the game completion scene is loaded. There is another variable called ‘secondsRemaining’ relating to the amount of time before the game is over. Once this variable reaches zero, the game over scene is loaded.

```
void Update()
{
    secondsRemaining -= 1 * Time.deltaTime;

    if (secondsRemaining <= 0 && !isGameComplete)
    {
        GameOver();
    }

    if (redCoreIsPlaced && blueCoreIsPlaced && yellowCoreIsPlaced && !hasGravitySwitched)
    {
        EnableGravity();
        hasGravitySwitched = true;
    }

    if (hasGravitySwitched && fires.childCount == 0 && O2Leaks.childCount == 0)
    {
        isGameComplete = true;
        StartCoroutine(CompleteGameAfterDelay());
    }
}
```

Figure 6. 'Update' function in 'GameManager.cs'

Critical Reflection

The prototype successfully delivers an immersive experience which utilises VR’s capabilities through the use of effective and consistent visuals that pair well with the 3D display, immersive spatial audio that contributes to the overall fidelity of the prototype, the use of traditional locomotion techniques in order to allow the player to traverse around the game’s scene, and precise and predictable interactions that let the user manipulate the environment. The technical implementation is reliable and there are no major bugs that compromise the experience. Item functionality behaves correctly, especially with the fire extinguishers and welding torches that provide tangible feedback when they are used. With additional refinement, the power cores could be updated as to create a more interactive experience for when they are placed into their respective slots, such as pulling and twisting the handle to improve the interaction further. The game’s state could be better conveyed to the player to view how much time is remaining before the game is over. The use of the canvas to convey this information was explored, but this is not permitted within VR experiences, and therefore other methods would have to be implemented. Generative AI was partially used in the project, specifically in the ‘PickUp.cs’ script. The use of it allowed for the pickup interaction to behave fluidly and as intended, whereas without it the interaction may have been less consistent and immersion-breaking for the player.