Virtual Science Fair - Project Documentation

Extended Reality Mini Project

Team Members

Mahsa NIAZI

Debanjana HALDAR

Project Overview

This Virtual Reality application simulates an interactive science fair environment designed for K-12 education. The project features multiple rooms with different scientific exhibits, focusing on physics and chemistry demonstrations using the PICO VR headset.

Implementation Details

Environment Structure

- Main entrance with an interactive door
- Two separate exhibition rooms
  - Room 1: Physics Exhibit (Solar System and Gravity Demonstration)
  - Room 2: Chemistry Exhibit (Volcanic Reaction Demonstration)

Room 1: Physics Exhibit

Features

- Interactive Solar System display
- Gravity comparison demonstration table
- Components:
  - Main display table
  - Secondary table with reset button
  - Two interactive balls for gravity testing
  - Designated testing zones for different gravitational fields

Interactions

- Ray-casting selection for objects
- Object manipulation through ray-casting
- Comparative gravity zones
- Physics-based interactions for gravity demonstration
- Reset functionality (applied but non-functional)

Room 2: Chemistry Exhibit

Features

- Interactive volcanic reaction demonstration
- Components:
  - Miniature volcano model
  - Two interactive bottles:
    - Soda bottle
    - Vinegar bottle
  - Chemical reaction visualization

Interactions

- Implementation of touchpad-based gliding/flying movement
- Door mechanism at entrance
- Grabbable bottles
- Ray-casting for distant object interaction
- Interactive pouring mechanism
- Chemical reaction trigger system (applied but non-functional)
- Reset functionality (applied but non-functional)

Technical Implementation

Working Features

- Ray-casting interaction system
- Basic physics implementation
- Object grabbing mechanics
- Door interaction mechanism

- Basic scene navigation

Known Issues

1. Reset Functionality:
    - Reset buttons in both rooms are non-functional
    - Scene state persistence issues
    - Canvas and TextMeshPro components not rendering properly in VR space
2. Timer System:
    - Implementation attempted but non-functional
    - Integration issues with main scene
    - Canvas and TextMeshPro components not rendering properly in VR space
3. Volcano reaction System
    - Implementation attempted but non-functional

Technical Challenges

1. Hardware Setup:
    - Difficulties with PICO headset configuration
    - Limited development time due to hardware setup issues
    - Computer compatibility challenges
2. UI Implementation:
    - Canvas and TextMeshPro integration issues in VR space
    - Problems with text rendering and scaling in 3D environment
    - Difficulties in maintaining consistent UI visibility across different viewing angles
3. Development Constraints:
    - Reduced tutorial and development time due to hardware setup issues and limited experience
    - Integration challenges with VR input systems

Future Improvements

1. Bug Fixes:
    - Implement working reset functionality
    - Fix timer system implementation

- Optimize performance for smoother VR experience

2. Feature Enhancements:
    - Add additional interactive elements
    - Improve visual feedback for interactions
    - Enhance physics simulation accuracy

Project Requirements Checklist

✓ Virtual navigation via touch pad

✓ Ray casting Manipulation

✓ Object activation/deactivation

✓ Physics implementation

✓ Lighting controlled during the interaction

✓ 3D audio integrated in the scene

× Canvas + 3D TextMeshPro

Timer functionality (applied but non-functional)

Reset functionality (applied but non-functional)

✓ Interactive object behaviors

Timer Implementation

1. UI Canvas Configuration:
    - Right-click in Hierarchy > UI > Canvas
    - This will create a Canvas with an Event System
    - Select the Canvas in the Hierarchy
    - In the Inspector, find the Canvas component
    - Set Render Mode to "Screen Space - Overlay"
    - Set UI Scale Mode (in Canvas Scaler component) to "Scale With Screen Size"
    - Set Reference Resolution to something like 1920 × 1080
2. Canvas settings for VR:
    - Select the Canvas and add a "Canvas Group" component
    - Settings:
        - Alpha: 1

    - Interactable: false
    - Blocks Raycasts: false
    - Ignore Parent Groups: false
3. Adding the Timer Text:
    - Right-click on the TimerPanel > UI > Text - TextMeshPro
    - Rename to "TimerText"
    - Set font size, style, color, alignment and margins for Timer Text
4. Adding the TimerController script:
    - Create an empty GameObject in the Hierarchy named "TimerManager"
    - Create and add the TimerController.cs script to it
    - Drag the TimerText object onto the "Timer Text" field in the TimerController component

```
using UnityEngine;
using TMPro;
using UnityEngine.UI;

public class TimerController : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI timerText;
    private float currentTime = 0f;
    private bool isRunning = true;

    private void Start()
    {
        // Ensure the timer text component is assigned
        if (timerText == null)
        {
            Debug.LogError("Timer Text component is not assigne
d!");
        }
        UpdateTimerDisplay();
    }

    private void Update()
    {
        if (isRunning)
        {
            currentTime += Time.deltaTime;
```

```
            UpdateTimerDisplay();
        }
    }

    private void UpdateTimerDisplay()
    {
        // Calculate minutes, seconds, and milliseconds
        int minutes = Mathf.FloorToInt(currentTime / 60f);
        int seconds = Mathf.FloorToInt(currentTime % 60f);
        int milliseconds = Mathf.FloorToInt((currentTime * 100f) %
100f);

        // Update the text with formatted time
        timerText.text = string.Format("{0:00}:{1:00}.{2:00}", min
utes, seconds, milliseconds);
    }

    public void StartTimer()
    {
        isRunning = true;
    }

    public void StopTimer()
    {
        isRunning = false;
    }

    public void ResetTimer()
    {
        currentTime = 0f;
        UpdateTimerDisplay();
    }
}
```

Reset Button Implementation

1. Creating UI Button in Canvas
    - Right-click in Hierarchy > UI > Canvas > Button
    - Customize the Button: Change Text, position, style.
2. Adding the script

- Create new C# script called SceneResetter:

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneResetter : MonoBehaviour
{
    // Function to reset the scene
    public void ResetScene()
    {
        // Reloads the currently active scene
        Scene currentScene = SceneManager.GetActiveScene();
        SceneManager.LoadScene(currentScene.name);
    }
}
```

3. Attaching the Script and Connecting the Button
    - Hierarchy > right-click and create an Empty GameObject named "SceneManager"
    - Drag and drop the SceneResetter script onto the GameObject.
    - Select the Button in the Canvas. In the Button's Inspector, add an event to On Click () section.
    - Drag "SceneManager" with the SceneResetter script into the empty field.
    - Choose SceneResetter > ResetScene in the dropdown.

Volcano Reaction Implementation

1. Setting up volcano trigger area:
    - Create an empty GameObject as a child of your Volcano object called "VolcanoTriggerArea"
    - Add a Box Collider component with "Is Trigger" checked
    - Scale and position it to represent the area where bottles can be placed
2. Setting up the bottles:
    - Select Baking soda bottle GameObject and add XR Grab Interactable to it:
        - Set Movement Type to Velocity Tracking
        - Check "Throw on Detach"
    - Select Vinegar bottle GameObject and add XR Grab Interactable to it:

    - Set Movement Type to Velocity Tracking
    - Check "Throw on Detach"

3. Adding VolcanoInteraction script:
    - Attach the VolcanoInteraction script to Volcano GameObject
    - In the VolcanoInteraction component in the Inspector:
        - Drag the VolcanoTriggerArea to the "Volcano Trigger Area" field
        - Drag the baking soda bottle to the "Baking Soda Bottle" field
        - Drag the vinegar bottle to the "Vinegar Bottle" field

```
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;

public class VolcanoInteraction : MonoBehaviour
{
    public Transform volcanoTriggerArea;
    public XRGrabInteractable bakingSodaBottle;
    public XRGrabInteractable vinegarBottle;

    private bool bakingSodaAdded = false;
    private bool vinegarAdded = false;

    private void Start()
    {
        // Ensure bottles have XRGrabInteractable component
        if (bakingSodaBottle == null)
        {
            Debug.LogError("Baking Soda Bottle not assigned!");
        }
        if (vinegarBottle == null)
        {
            Debug.LogError("Vinegar Bottle not assigned!");
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        XRGrabInteractable interactable = other.GetComponent<XRGra
bInteractable>();

        if (interactable == bakingSodaBottle && !bakingSodaAdded)
```

```
        {
            bakingSodaAdded = true;
            interactable.gameObject.SetActive(false);
            CheckForEruption();
        }
        else if (interactable == vinegarBottle && !vinegarAdded)
        {
            vinegarAdded = true;
            interactable.gameObject.SetActive(false);
            CheckForEruption();
        }
    }

    private void CheckForEruption()
    {
        if (bakingSodaAdded && vinegarAdded)
        {
            // Trigger eruption after a delay
            Invoke("TriggerEruption", 5f);
        }
    }

    private void TriggerEruption()
    {
        // Call the eruption effect method
        GetComponent<VolcanoEruption>().StartEruption();
    }
}
```

4. Adding VolcanoEruption script:
    - Attach the VolcanoEruption script to Volcano GameObject
    - In the VolcanoEruption component in the Inspector:
        - Drag EruptionParticles GameObject to “Eruption Particles” field.
        - Drag EruptionAudio GameObject to “Eruption Sound” field.
        - Set Eruption Duration (15 seconds)

```
using UnityEngine;

public class VolcanoEruption : MonoBehaviour
```

```
{
    [SerializeField] private ParticleSystem eruptionParticles;
    [SerializeField] private AudioSource eruptionSound;
    public float eruptionDuration = 15f;

    public void StartEruption()
    {
        if (eruptionParticles != null)
        {
            eruptionParticles.Play();
        }

        if (eruptionSound != null)
        {
            eruptionSound.Play();
        }

        Invoke("StopEruption", eruptionDuration);
    }

    private void StopEruption()
    {
        if (eruptionParticles != null)
        {
            eruptionParticles.Stop();
        }

        if (eruptionSound != null && eruptionSound.isPlaying)
        {
            eruptionSound.Stop();
        }
    }
}
```

Conclusion

While the project successfully implements core VR interactions and educational demonstrations, technical challenges and time constraints impacted the full implementation of all planned features. The foundation for an engaging educational VR experience has been established, with clear paths for future improvements and optimizations.