ASSIGNMENT

of

Augmented Reality and Virtual Reality

CS 437

Bachelor of Technology (CSE)

By

Thelotham Ravi (21124050)

Last Year, Semester 7

Course In-charge: Prof. Darshan Parmar

Department of Computer Science and Engineering

School Engineering and Technology Navrachana University, Vadodara

Spring – 2024.

GitHub link - https://github.com/ThilsaRavi9/AxeMaster-VR

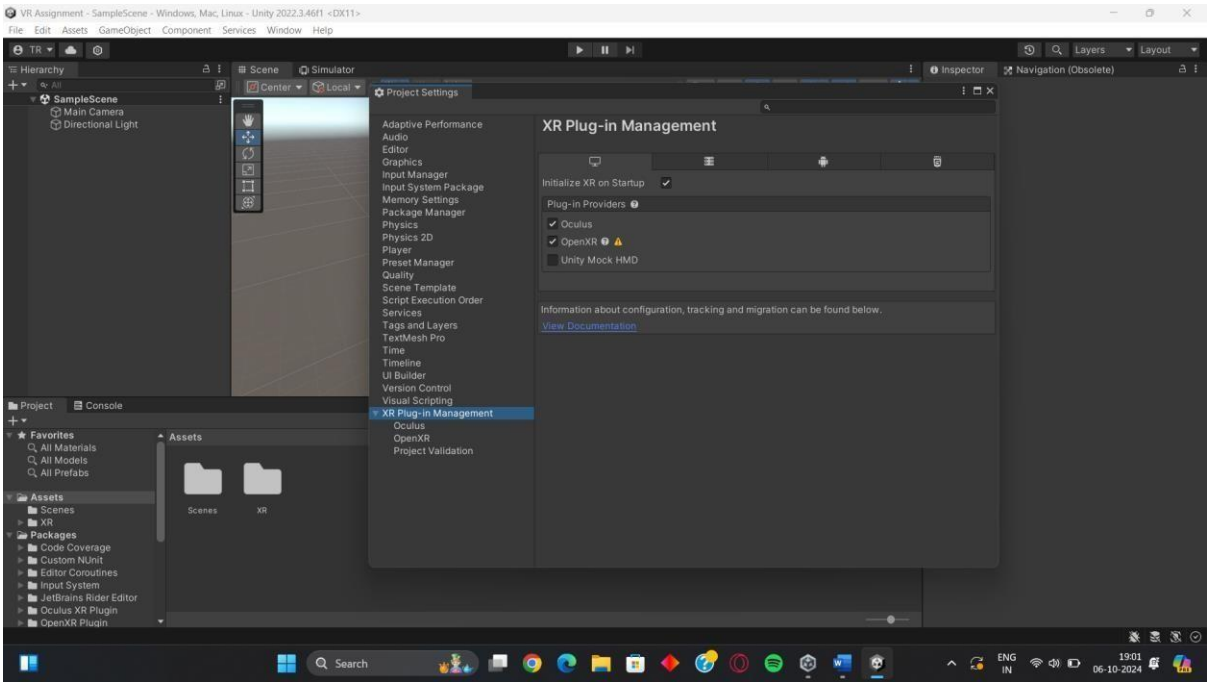
Objective : To create a basic VR game with a virtual environment in Unity that includes a ground plane, a skybox, environmental objects, lighting, and simple VR interaction. The player should be able to grab and move the grabale objects in the environment to score points.

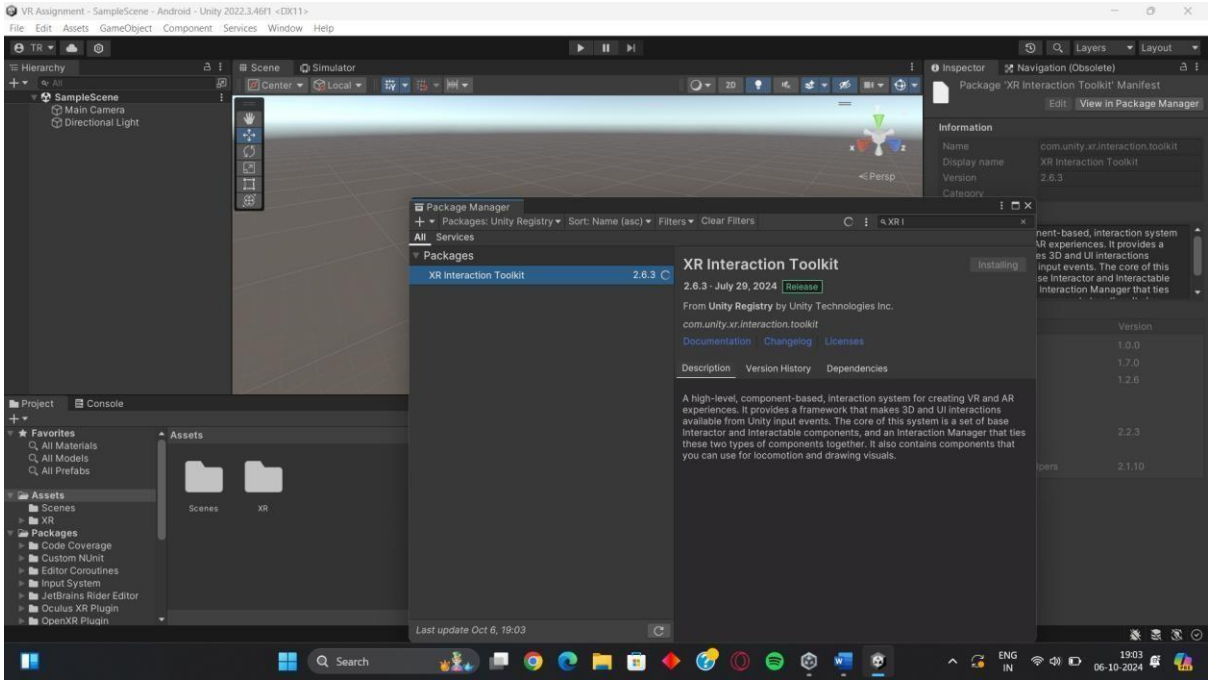
Task 1: Setup your unity project and configure the VR Environment.

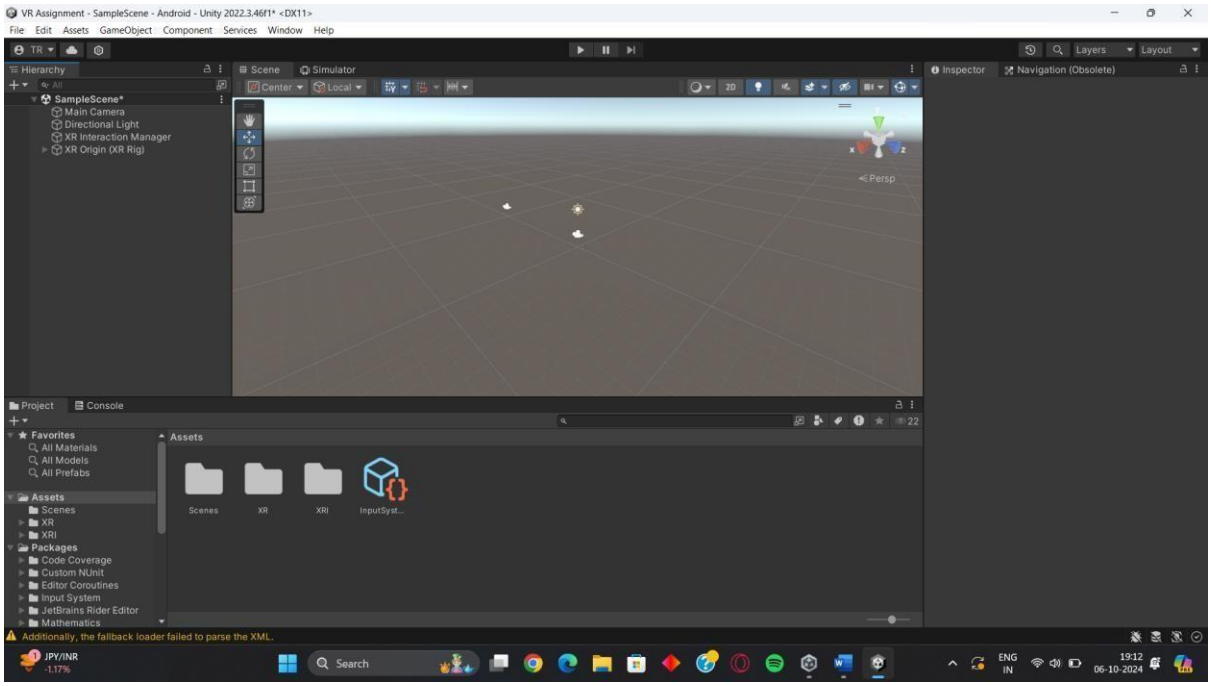
- **Goal**: Initialize a Unity project for VR.
- **Steps**:

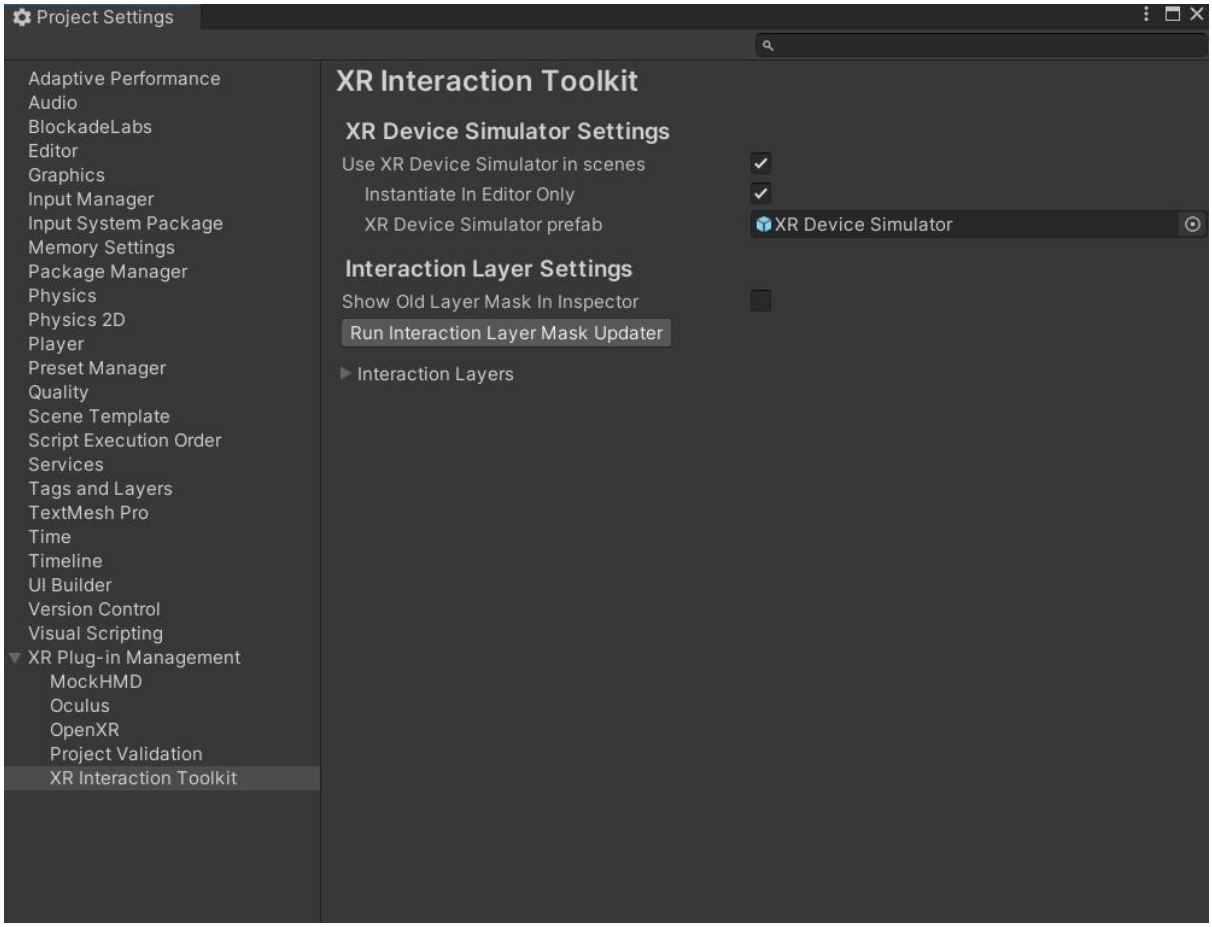
1. Open Unity and create a new 3D project.
2. Set up your VR environment using XR plugins (for example, Oculus or OpenXR).
3. Import the XR Interaction Toolkit from Unity's Package Manager.
4. Configure the necessary settings to ensure VR functionality (like enabling VR support in Player Settings).

#Screenshot :




VR Assignment - SampleScene - Android - Unity 2022.3.46f1 <DX11>
File Edit Assets GameObject Component Services Window Help
Hierarchy
SampleScene
Main Camera
Directional Light
Scene
Simulator
Inspector
Navigation (Obsolete)
Package 'XR Interaction Toolkit' Manifest
Edit
View in Package Manager
Information
Name
com.unity.xr.interaction.toolkit
Display name
XR Interaction Toolkit
Version
2.6.3
Package Manager
Packages: Unity Registry
Sort: Name (asc)
Filters
Clear Filters
All
Services
Packages
XR Interaction Toolkit
2.6.3
XR Interaction Toolkit
Installing
2.6.3 - July 29, 2024
Release
From Unity Registry by Unity Technologies Inc.
com.unity.xr.interaction.toolkit
Documentation
Changelog
Licenses
Description
Version History
Dependencies
A high-level, component-based, interaction system for creating VR and AR experiences. It provides a framework that makes 3D and UI interactions available from Unity input events. The core of this system is a set of base Interactor and Interactable components, and an Interaction Manager that ties these two types of components together. It also contains components that you can use for locomotion and drawing visuals.
Last update Oct 6, 19:03
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Scenes
XR
Packages
Code Coverage
Custom NUnit
Editor Coroutines
Input System
JetBrains Rider Editor
Oculus XR Plugin
OpenXR Plugin
Persp
Search
ENG
IN
19:03
06-10-2024


VR Assignment - SampleScene - Android - Unity 2022.3.46f1* <DX11>
File Edit Assets GameObject Component Services Window Help
Hierarchy
SampleScene*
Main Camera
Directional Light
XR Interaction Manager
XR Origin (XR Rig)
Scene
Simulator
Inspector
Navigation (Obsolete)
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Scenes
XR
XRI
InputSyst...
Packages
Code Coverage
Custom NUnit
Editor Coroutines
Input System
JetBrains Rider Editor
Mathematics
Additionally, the fallback loader failed to parse the XML.
Persp
JPY/INR
-1.17%
Search
ENG
IN
19:12
06-10-2024



Task 2: Create the Ground Plane and add a Skybox

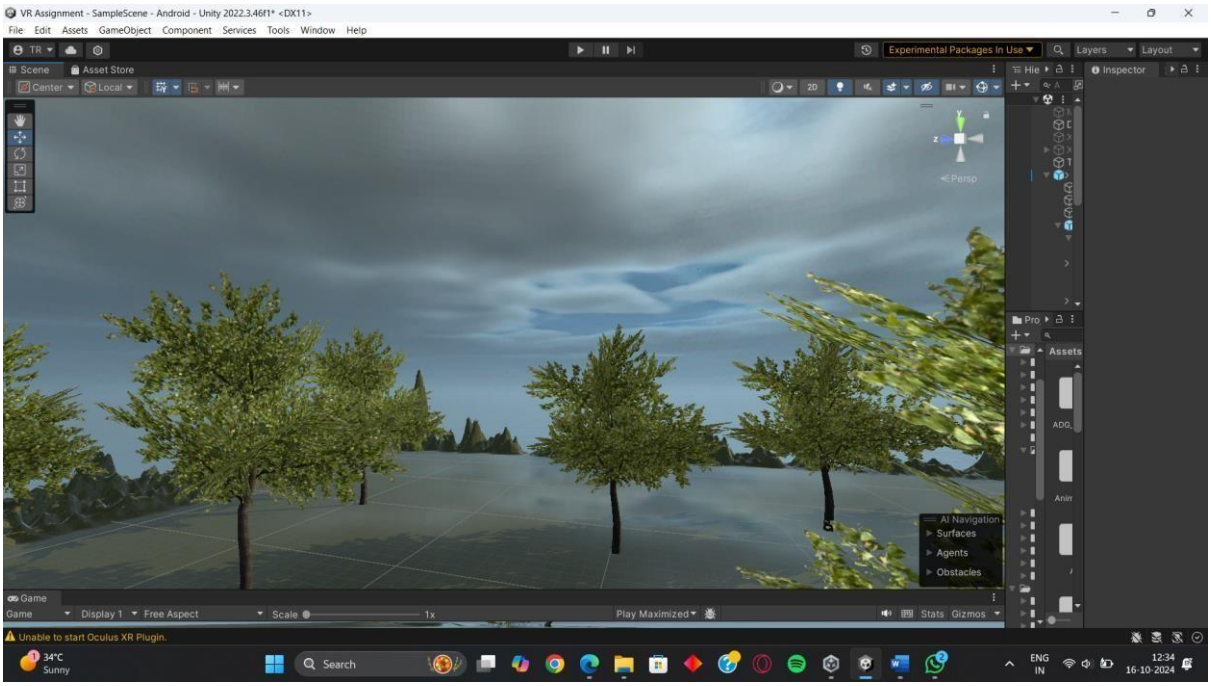
- **Goal**: Establish a base environment.
- **Steps**:

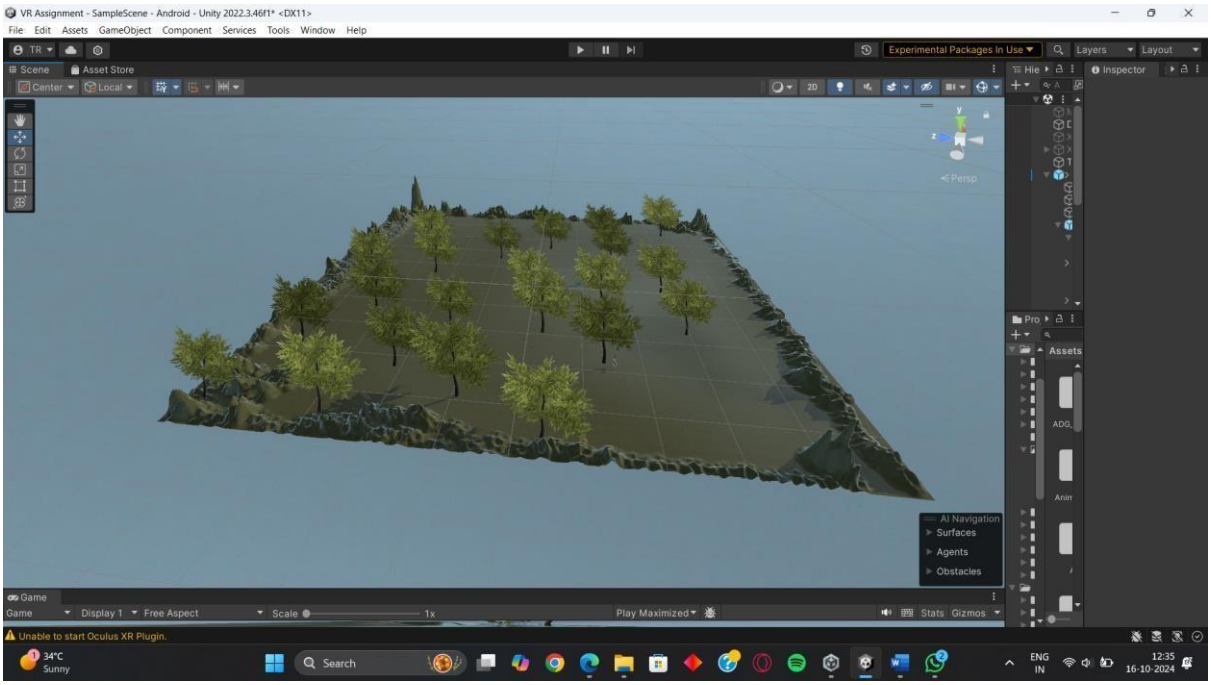
    1. Scale a 3D plane to represent the ground.
    2. Download assets from the Unity Asset Store for terrain textures and skyboxes.
    3. Add your terrain texture to the plane and set up a skybox using the asset's material.
    4. Verify the visual rendering by adjusting the scale and positioning.

Skybox link : Skybox Terrain

texture : Terrain link

#Screenshot :





Task 3: Add Environment Objects

- **Goal**: Populate the scene with interactive objects.
- **Steps**:

1. Import environment objects (like trees, animals) from the asset store.
2. Drag and drop these objects onto the scene, adjusting their positions.

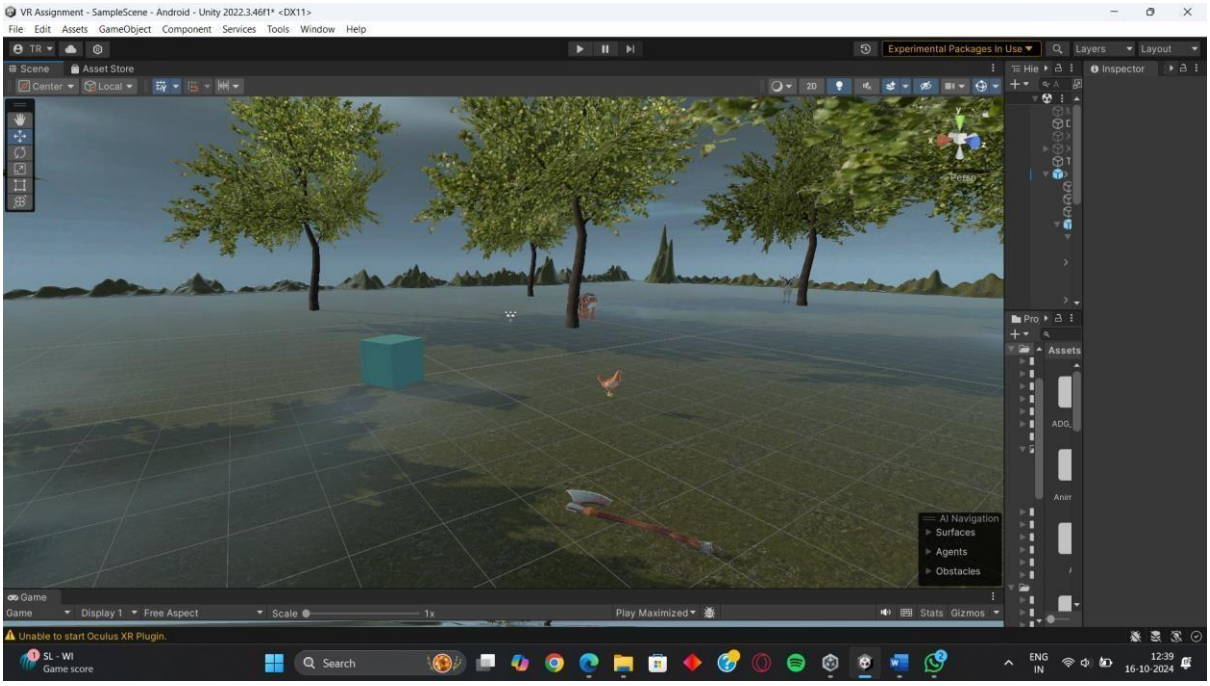
3. Add interactable objects like the axe for the player to grab.
4. Ensure each object has the appropriate collider for interaction.

Trees link : [Tree asset link]

Animal asset link : [Animal asset link]

Axe asset link : [Axe link]

#Screenshot :



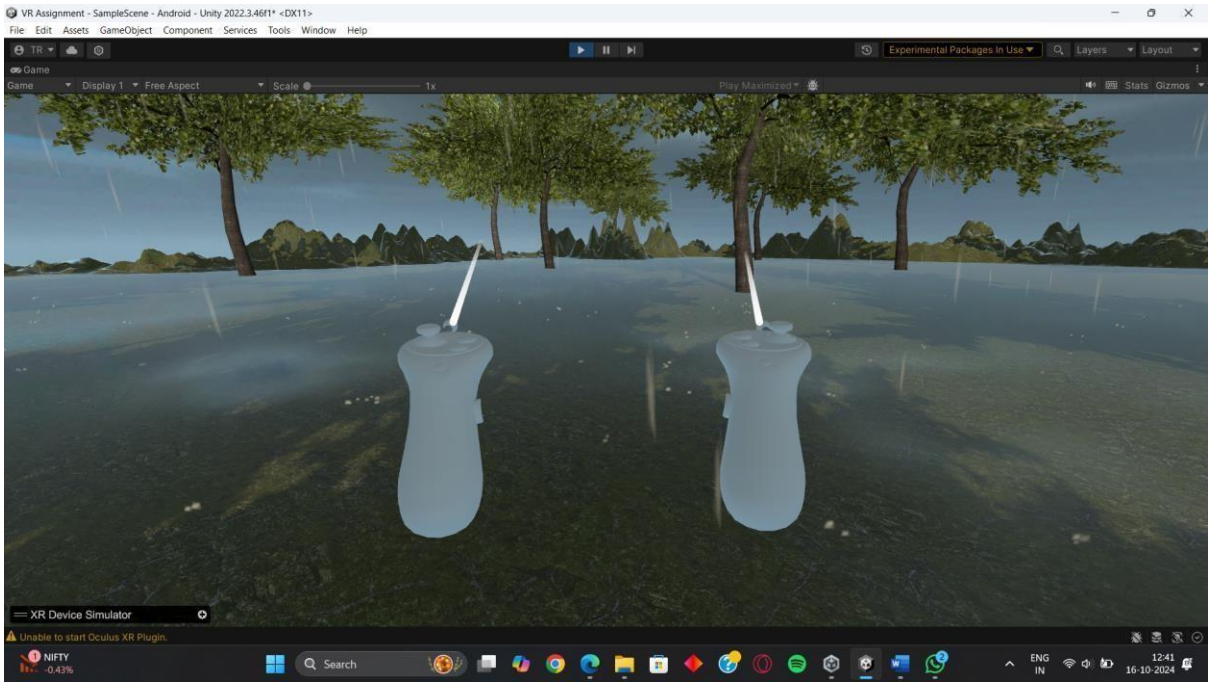
Task 4: Configure Lighting and Shadows

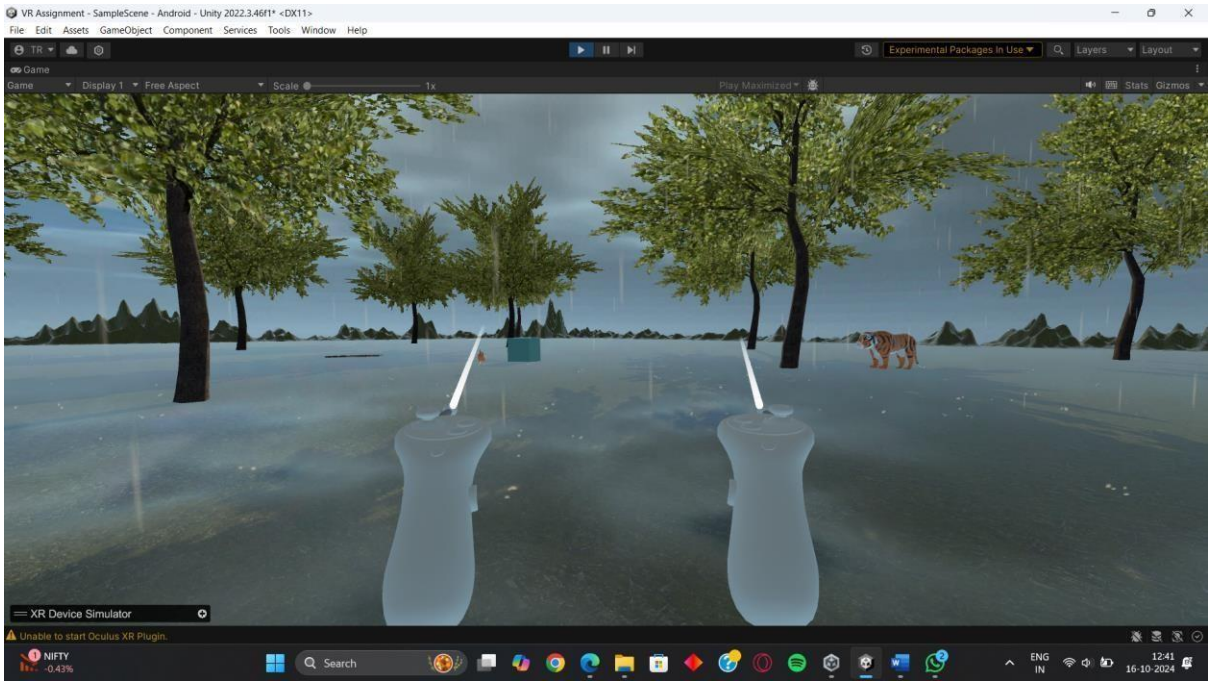
- **Goal**: Improve the scene's realism.
- **Steps**:

1. Use Unity's lighting system to add directional lights.
2. Adjust shadows and reflections to fit your environmental scene.
3. Import and apply the "Rain Maker" particle system to add rain effects.

Rain maker asset link : [Rain maker]

#Screenshot :



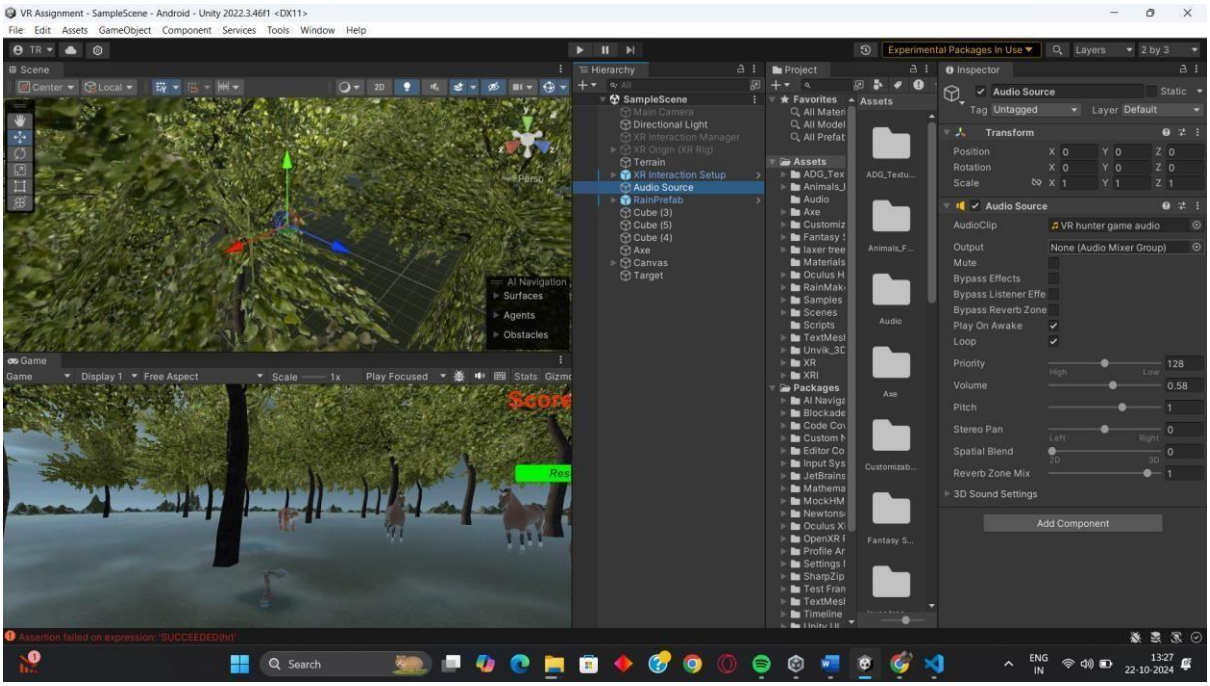


Task 5: Add Audio

- **Goal**: Introduce immersive sound to your VR experience.
- **Steps**:

1. Trim your desired audio clip using an audio editing tool.
2. Import the audio clip into Unity.
3. Attach the audio clip to an empty GameObject or specific object (like background music or sound effects for interactions).
4. Ensure audio sources are set to play in 3D space if spatial audio is required.

Here’s the audio link : [Hunter audio.mp3]



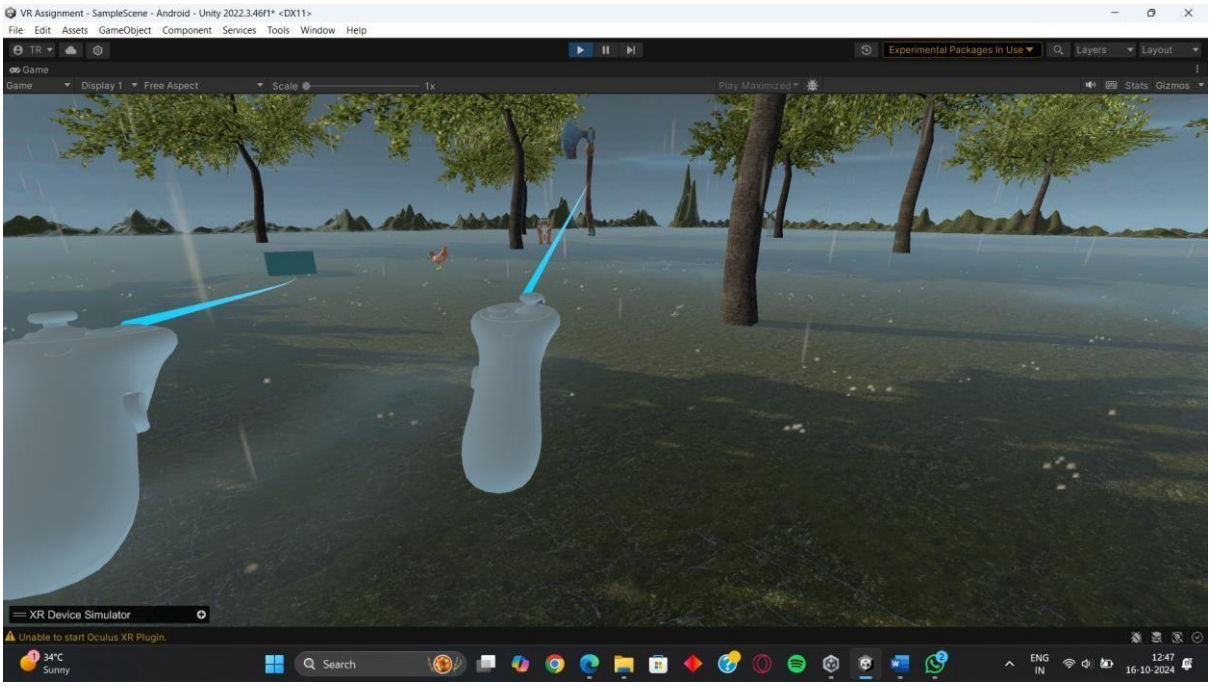
Task 6: Implement Basic VR Interaction

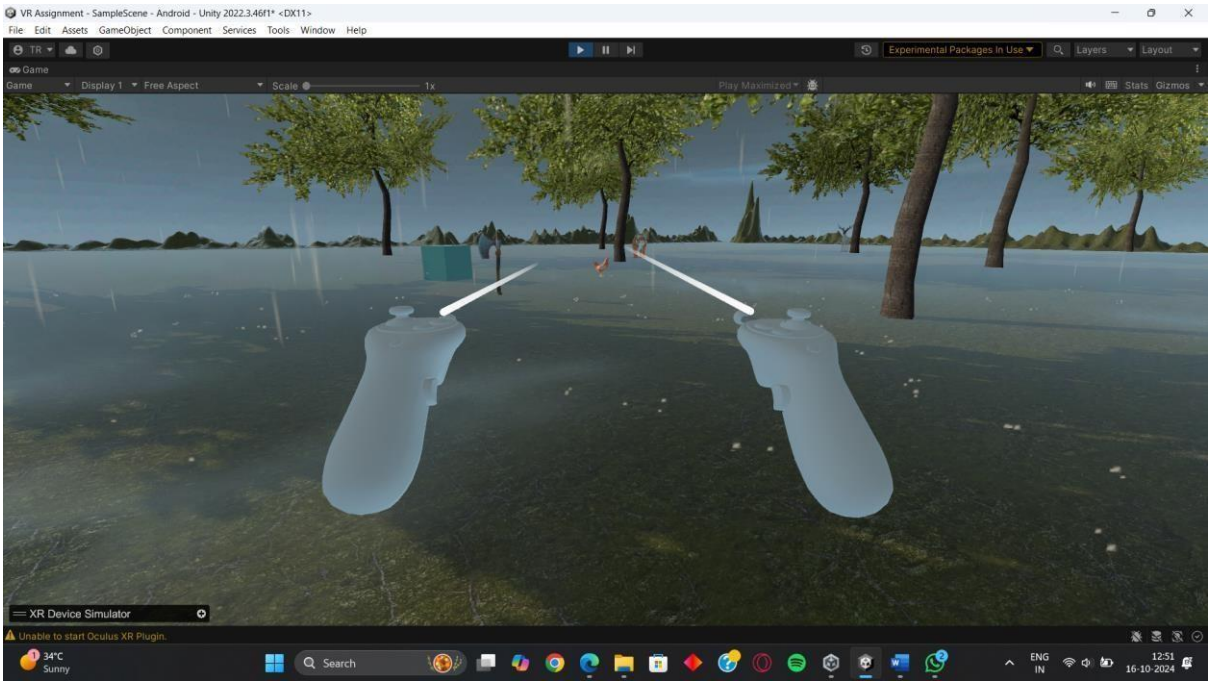
- **Goal**: Make objects interactable in the VR environment.
- **Steps**:

1. Add XR Grab Interactable components to objects like the axe.
2. Set up input actions (e.g., joystick or hand tracking) to allow players to grab and interact with objects.
3. Test the interactions to ensure smooth grabbing and releasing of objects.

Here you can see that with the VR joystick we can grab an axe and hit it at the animals.

#Screenshot :



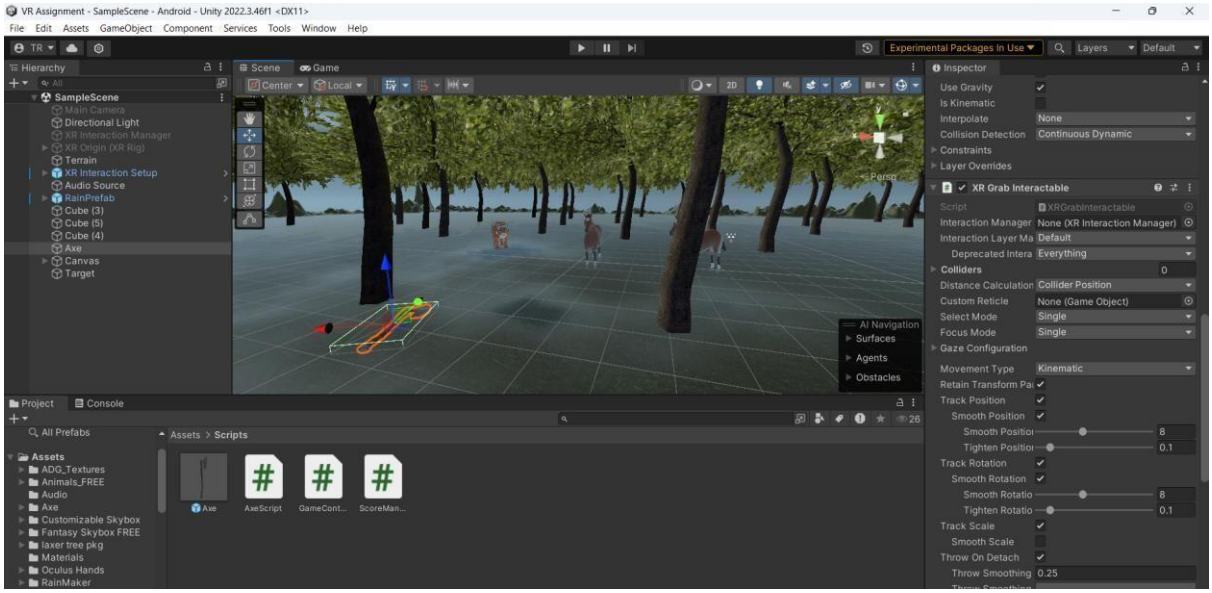


Task 7: Write the VR Interaction Script

- **Goal**: Script interactive behavior for objects.
- **Steps**: 1. Write scripts like `AxeScript.cs` to define what happens when the player interacts with objects (e.g., hitting an animal with an axe). `GameController.cs` script manages the game's overall state. It displays the "You Won!" message when the player reaches the target score and pauses the game.

2. Ensure your objects have appropriate tags (e.g., “Tiger” or “Horse”) to trigger specific behaviors upon collision.
3. Use debugging logs to track collision events and resolve issues.
4. Make use of `XR Grab Interactable` and `Rigidbody` components for physicsbased interactions.

#Screenshot :



AxeScript.cs Script:

```
using UnityEngine;

public class AxeScript : MonoBehaviour
{
   private ScoreManager scoreManager;
public GameController gameController;
   public int winScore = 40;

   void Start()
   {
      // Find the ScoreManager and GameController
scoreManager = FindObjectOfType<ScoreManager>();
gameController = FindObjectOfType<GameController>();
   }

   void OnCollisionEnter(Collision collision)
   {
      // Debug to see what the axe is hitting
      Debug.Log("Axe collided with: " + collision.gameObject.name + " (Tag: " + collision.gameObject.tag +
")");
```

```
        bool hitTarget = false;  // Flag to check if we hit a target

        // Check if the object has the "Tiger" tag
        if (collision.gameObject.CompareTag("Tiger"))
        {
            // Debug to confirm that the tag is correct
            Debug.Log("Hit the target!");

            // Destroy the target object
            Destroy(collision.gameObject);

            // Add score for hitting the target
scoreManager.AddScore(10);
            hitTarget = true;  // Set the hit flag
        }
        else if (collision.gameObject.CompareTag("Horse"))
        {
            // Debug to confirm that the tag is correct
            Debug.Log("Hit the target!");

            // Destroy the target object
            Destroy(collision.gameObject);

            // Add score for hitting the target
scoreManager.AddScore(15);
            hitTarget = true;  // Set the hit flag
        }
        else if (collision.gameObject.CompareTag("15")) // Assuming this is another target type
        {
            // Debug to confirm that the tag is correct
            Debug.Log("Hit the target!");

            // Destroy the target object
            Destroy(collision.gameObject);

            // Add score for hitting the target
scoreManager.AddScore(10);
            hitTarget = true;  // Set the hit flag
        }

        if (hitTarget)  // Check if a target was hit
        {
            // Check for winning condition
            if (scoreManager.GetScore() >= winScore)
            {
                gameController.ShowWinMessage();  // Show the "You Won!" message
            }
}
else
```

```
        {
            // If it's not a target, debug what was hit
            Debug.Log("Collided with non-target object: " + collision.gameObject.tag);
        }
    }
}
```

GameController.cs Script(Displaying the message):

```
using UnityEngine;
using UnityEngine.SceneManagement;  // For reloading scenes
using TMPro;  // For TextMesh Pro (replace with UI Text if using default UI Text)

public class GameController : MonoBehaviour
{
    public TMP_Text winText;  // Reference to the TextMeshPro text (use UI Text if needed)

    void Start()
    {
        // Ensure the win message is hidden at the start
if (winText != null)
        {
            winText.gameObject.SetActive(false);
        }
    }

    // Function to display the win message
    public void ShowWinMessage()
    {
        if (winText != null)
        {
            winText.gameObject.SetActive(true);  // Display the "You Won" text
winText.text = "You Won!";
            Time.timeScale = 0f;  // Freeze the game
        }
    }

    // Function to reset the game
    public void ResetGame()
    {
        Time.timeScale = 1f;  // Unfreeze the game
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);  // Reload the current scene
    }
}
```

ScoreManager.cs Script:

```
using UnityEngine;  using TMPro;  // For TextMesh Pro (replace
with UI Text if needed)  public class ScoreManager :
MonoBehaviour
{
```

```
    public TMP_Text scoreText;  // Reference to the UI text for the score
private int score = 0;  // Start the score at 0

    void Start()
    {
        UpdateScore();  // Update the score display when the game starts
    }

    // Function to increase the score
    public void AddScore(int points)
    {
        score += points;
        UpdateScore();  // Update the score display after adding points
    }

    // Update the score UI
    private void UpdateScore()
    {
        if (scoreText != null)
        {
            scoreText.text = "Score: " + score;
        }
    }
    // Function to check if the player has won (if you have a specific score to win)
public bool HasWon(int winScore)
    {
        return score >= winScore;
    }

    public int GetScore()
    {
        return score;
    }
}
```

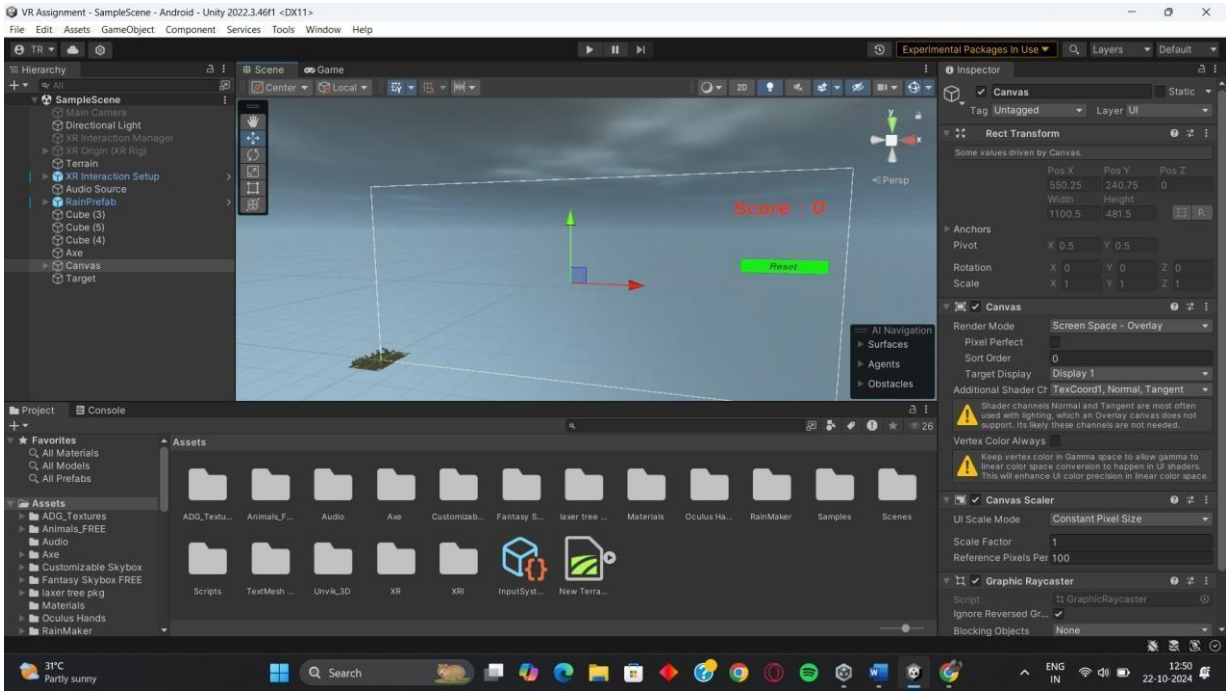
Task 8: Create a scoring mechanism

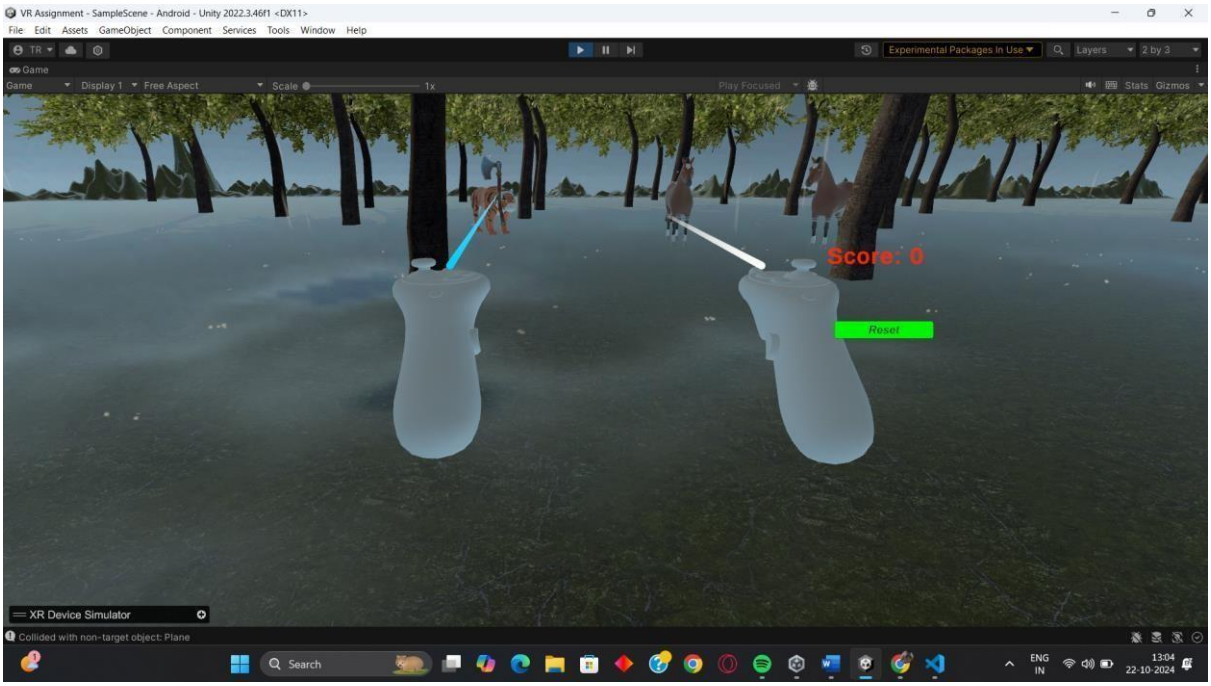
- **Goal**: Track player progress with a score system.
- **Steps**:

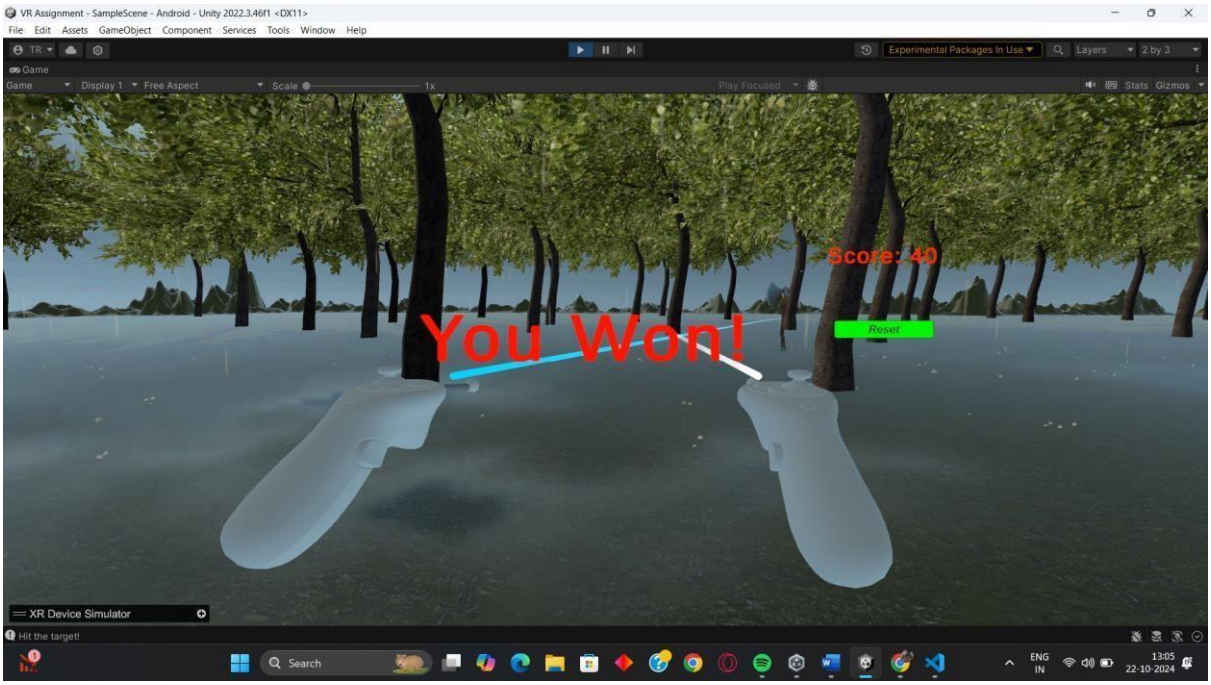
    1. Create a `ScoreManager` script to handle score updates when objects are hit.
    2. Use UI Canvas to display the score on the screen, updating it dynamically as the game progresses.

3. Add a "Reset" button to restart the game and a "You Won" message that displays when the player reaches a target score.

#Screenshot :







Here’s a link for the final demo video : [Demo Video]

Feedback(Difficulties faced):

I encountered challenges with the collision detection between the axe and various objects in the game. Initially, the objects would not respond to the axe, and even when they did, all

objects were destroyed without any interaction. After troubleshooting, I resolved the issue by adjusting the collision logic and ensuring the objects had the correct colliders and tags.