MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR

(Deemed University)

(Declared under Distinct Category by Ministry of Education, Government of India)

NAAC Accredited With A++ Grade

A

Project Report

On

AR Flashcards Recognition and VR Shooting Game

Augmented Reality - Virtual Reality Lab

(2280721)

Branch

Artificial Intelligence and Machine Learning

Submitted By:

Anushka Gupta	**0901AM221077**
Yuvraj Dawande	**0901AM221076**
Shivam Singh kushwah	**0901AM233D05**

Faculty Mentor:

Dr. Vibha Tiwari

Assistant Professor

CENTRE FOR ARTIFICIAL INTELLIGENCE

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE

GWALIOR - 474005 (MP) est. 1957

JUL-DEC 2025

CERTIFICATE

This is certified **Anushka Gupta** (0901AM221077), **Yuvraj Dawande** (0901AM221076) and **Shivam Singh Kushwah**(0901AM233D05) has submitted the project report titled **AR Flashcards recognition and VR Shooting Game** under the mentorship of **Dr. Vibha Tiwari**, in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in **Artificial Intelligence and Machine Learning** from Madhav Institute of Technology and Science, Gwalior.

Dr.Vibha Tiwari
Faculty Mentor
Assistant Professor
Centre for Artificial Intelligence

DECLARATION

We hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of Bachelor of Technology in **Artificial Intelligence and Machine Learning** at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of our work under the mentorship of **Dr.Vibha Tiwari**, **Assistant Professor**, Centre for Artificial Intelligence.

We declare that we have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Anushka Gupta (0901AM221077)
Yuvraj Dawande (0901AM221076)
Shivam Singh Kushwah (0901AM233D05)

PLAGIARISM CHECK CERTIFICATE

This is to certify that I/we, a student of B.Tech. in Centre for AI have checked my complete report entitled "**AR AND VR PROJECT**" for similarity/plagiarism using the "Turnitin" software available in the institute. This is to certify that the similarity in my report is found to be …… which is within the specified limit (30%).

The full plagiarism report along with the summary is enclosed.

Anushka Gupta (0901AM221077)
Yuvraj Dawande (0901AM221076)
Shivam Singh Kushwah (0901AM233D05)

Checked & Approved By:

Dr. Vibha Tiwari
Assistant Professor
Center for AI

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. We are thankful to our institute, **Madhav Institute of Technology and Science** to allow us to continue our disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute.

We are sincerely thankful to our faculty mentor. We are grateful to the guidance of **Dr.Vibha Tiwari**, Assistant Professor, Centre for Artificial Intelligence, for her continued support and guidance throughout the project. We are also very thankful to the faculty and staff of the department.

Anushka Gupta 0901AM221077
Yuvraj Dawande 0901AM221076
Shivam Singh Kushwah 0901AM233D05

Table of Contents

AR Flashcards for Language Learning 2

Chapter 1: Introduction 3

Chapter 2: Tools & Assets 4

Chapter 3: Instructions 6

Chapter 4: Output 7

Chapter 5: Code 9

Chapter 6: Conclusion and Future Scope 16

References 17

VR SHOOTING GAME 18

Chapter 1: Introduction 19

Chapter 2: Tools and Assets 20

Chapter 3: Instructions to Play 23

Chapter 4: Output 24

Chapter 5: Code Explanation 26

Chapter 6: Conclusion and Future Scope 30

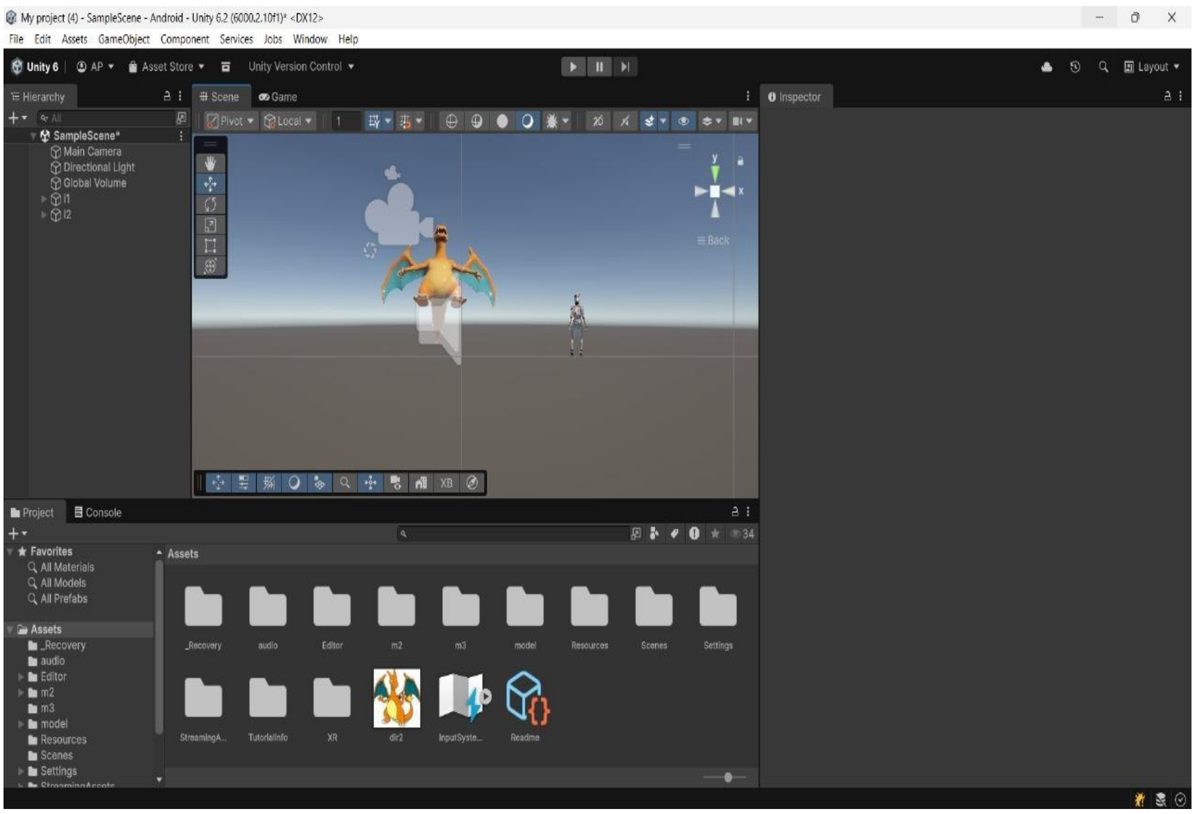
References 31

A

Project report

on

AR Flashcards Recognition



Chapter 1: Introduction

Learning a new language, especially for children or absolute beginners, often feels like a steep climb. The traditional tools we rely on, like paper flashcards, are fundamentally static and can become monotonous. They offer a one-dimensional experience, making it easy for a learner's interest to fade. Our project is an attempt to provide a modern solution to this very problem. We've set out to build an Augmented Reality (AR) based learning system with one primary goal: to transform language learning from a chore into an engaging, interactive experience that feels more like playing a game.

The core concept of this project is to take traditional, lifeless flashcards and breathe life into them, turning them into a rich, multi-sensory tool. We wanted the learner to experience the word, not just memorize its text.

How it works is quite straightforward. When a user, perhaps a child, scans one of our custom-printed flashcards (an image of an apple, for example) using their mobile phone's camera, our system instantly recognizes that image. The moment it does, the application overlays a 3D animated model of that object right onto the live camera feed, making it appear as if the object is sitting in the real world.

But this experience isn't just visual. As that 3D model appears, the application simultaneously plays a clear audio pronunciation of the word (e.g., "Apple"). To further help the user, a text translation also appears on the screen (e.g., "Apple - सेब"). In that single moment, the learner is **seeing** the object, **hearing** its name, and **reading** the word.

This combined approach makes language far more visual and significantly more engaging than traditional methods. When visual (the 3D model), auditory (the pronunciation), and text (the translation) cues all work together, it dramatically improves memory retention for young users. It actively helps refine their pronunciation skills and, most importantly, keeps them genuinely interested in the learning process.

To build this application, we used the **Unity 3D** engine as our foundation. The crucial image recognition and AR magic are handled by the **Vuforia AR SDK**. The final application has been built as a standalone APK file, designed to run easily on any standard Android smartphone.

Chapter 2: Tools & Assets

2.1 Tools and Software

2.1.1 Unity 3D:

- **Description:** Unity 3D is the primary real-time development platform used for designing the 3D scene, scripting interactions, and building the final Android application.
- **Key Features:** Unity provides a comprehensive set of tools for creating 3D applications, powerful scripting with C#, and seamless integration with AR SDKs.
- **Use in Project:** Used for environment setup, asset integration, C# scripting for interactivity, and building the final .apk file.

2.1.2 Vuforia Engine:

- **Description:** Vuforia is the Augmented Reality Software Development Kit (SDK) used to enable image recognition.
- **Key Features:** Provides robust **Image Target** tracking, allowing the app to recognize 2D images (the flashcards) and anchor 3D content to them in real-time.
- **Use in Project:** Used to create the **Image Target Database** from our flashcard images. The AR Camera and Image Target prefabs from Vuforia are the core of the AR functionality.

2.1.3 Android SDK/NDK/JDK:

- **Description:** These are the build tools required by Unity to compile and build the project into a functional Android application (.apk).
- **Key Features:** Provides the essential libraries (SDK, JDK) and native development tools (NDK) that Unity needs to translate C# scripts into code that can run on Android devices. It handles the compiling, packaging of all assets, and signing of the final .apk file.
- **Use in Project:** Integrated into Unity Hub's Android Build Support module to handle the build process, allowing the app to be deployed and tested on physical Android smartphones.

2.2 Asset Packs and 3D Models

2.2.1 3D Models (FBX/GLB):

- **Description:** A collection of low-polygon 3D models corresponding to each flashcard (e.g., Apple, Ball, Cat, Dog).
- **Source:** Models were sourced from online asset libraries such as **Sketchfab** and **Free3D**.
- **Use in Project:** These models are placed as children of their respective Image Targets in Unity, so they appear when the flashcard is scanned.

2.2.2 Image Targets (Flashcards):

- **Description:** A set of high-contrast, feature-rich 2D images used as the physical flashcards.
- **Source:** Images were designed or sourced from asset libraries like **Freepik** and **Pixabay**.
- **Use in Project:** These images were uploaded to the Vuforia Target Manager to create the database. The app actively searches for these images via the phone's camera.

2.2.3 Audio Files (MP3):

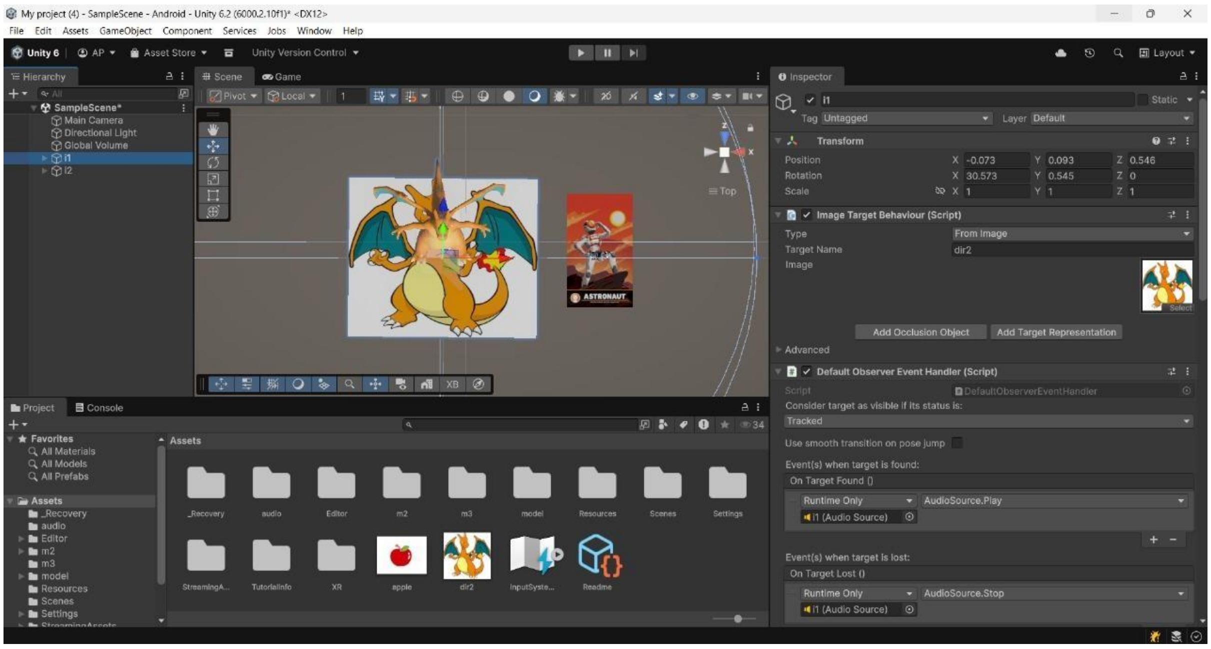
- **Description:** A collection of .mp3 audio clips containing the clear pronunciation of each word in the target language.
- **Key Features:** These are lightweight, high-compression audio files that load quickly and take up minimal space in the final .apk. They are universally compatible with Android devices.
- **Use in Project:** Each audio clip is attached as an Audio Source component to its corresponding 3D model, ready to be played by our C# script.

Chapter 3: Instructions

To play the game:

1. **Starting the Game**
    - Install the AR_Flashcards.apk file on a compatible Android smartphone.
    - Launch the application. You may need to grant **camera permissions** for the app to function.
    - A main menu screen will appear with a "Start AR" button. Press it to begin.
2. **Using the AR Scanner**
    - The app will open your phone's camera.
    - Take one of the printed physical flashcards (e.g., the 'Apple' card).
    - Point your phone's camera directly at the image on the flashcard.
3. **Viewing the AR Content**
    - When the app successfully recognizes the flashcard, a 3D model of the object (the apple) will appear on top of the card on your screen.
    - Simultaneously, the audio pronunciation ("Apple") will play from your phone's speaker.
    - A text box will also appear, showing the word in both English and its translation (e.g., "Apple - सेब").
4. **Learning**
    - You can move your phone around the card to see the 3D model from different angles.
    - Pull the camera away from the card to make the model disappear.
    - Point the camera at a different flashcard (e.g., 'Ball') to see the next 3D model and hear its pronunciation.
5. **Exiting**
    - Press the "Back" button on your phone or an on-screen "Quit" button to exit the application.

Chapter 4: Output



Inspector

i1 Static

Tag Untagged Layer Default

Transform

Position X -0.073 Y 0.093 Z 0.546

Rotation X 30.573 Y 0.545 Z 0

Scale X 1 Y 1 Z 1

Image Target Behaviour (Script)

Type From Image

Target Name dir2

Image

Add Occlusion Object | Add Target Representation

Advanced

Default Observer Event Handler (Script)

Script DefaultObserverEventHandler

Consider target as visible if its status is:

Tracked

Use smooth transition on pose jump

Event(s) when target is found:

On Target Found ()

Runtime Only | AudioSource.Play

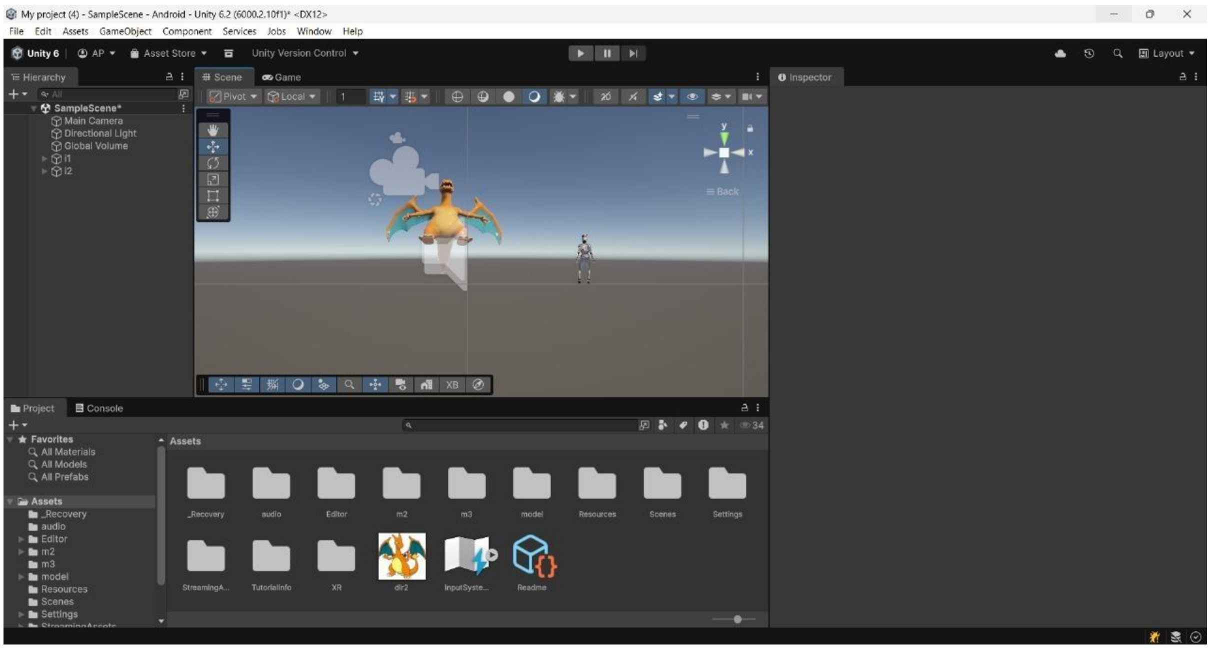
i1 (Audio Source)

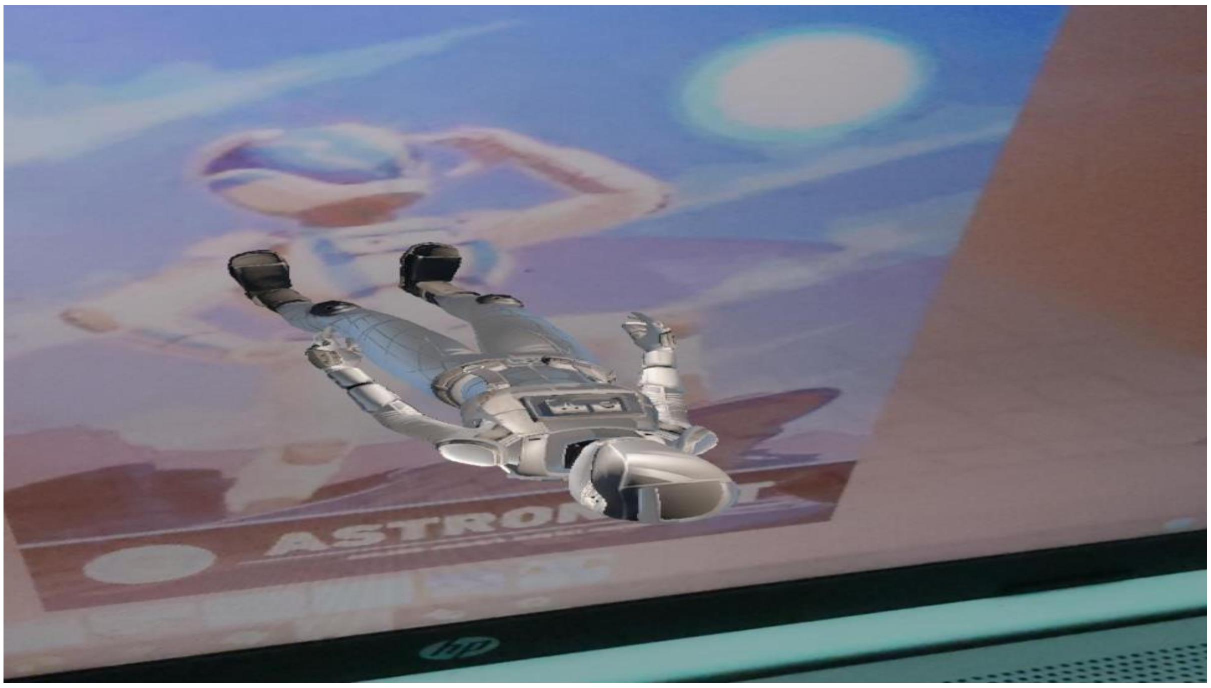
Event(s) when target is lost:

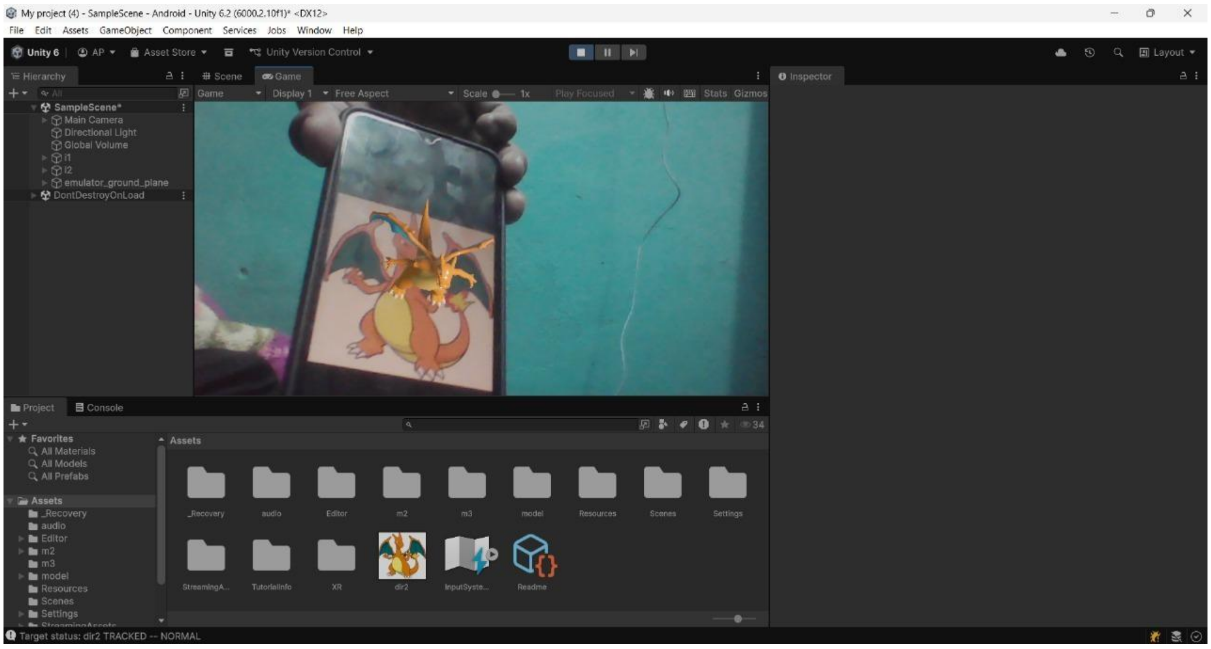
On Target Lost ()

Runtime Only | AudioSource.Stop

i1 (Audio Source)







Target status: dir2 TRACKED -- NORMAL

Chapter 5: Code

5.1 PlaySoundOnDetect.cs:

This script is attached to the **Image Target** object. It uses the ObserverBehaviour from Vuforia to check the target's status. When the target's status changes to TRACKED (meaning the flashcard is visible), it finds the AudioSource component (attached to the 3D model) and plays the audio clip. When the target is lost, it stops the audio.

CODE:

C#

```
using UnityEngine;
using Vuforia;


public class PlaySoundOnDetect : MonoBehaviour
{
    private AudioSource audioSource;
    private ObserverBehaviour observerBehaviour;


    void Start()
    {
        // Find the AudioSource, assuming it's on a child 3D model
        audioSource = GetComponentInChildren<AudioSource>();
        observerBehaviour = GetComponent<ObserverBehaviour>();


        if (observerBehaviour)
        {
            observerBehaviour.OnTargetStatusChanged += OnTargetStatusChanged;
        }
    }


    private void OnTargetStatusChanged(ObserverBehaviour behaviour, TargetStatus status)
    {
        if (status.Status == Status.TRACKED)
        {
            // Target is found
            if (audioSource != null && !audioSource.isPlaying)
            {
                audioSource.Play();
            }
        }
        else
        {
            // Target is lost
            if (audioSource != null)
            {
                audioSource.Stop();
            }
        }
    }
}
```

5.2 AnimateOnDetect.cs:

This script is also attached to the **Image Target**. Its purpose is to trigger an animation on the 3D model when the card is detected. This is useful for making models spin, bounce, or play a specific animation (like a cat purring). It works similarly to the sound script by checking the ObserverBehaviour status.

CODE:

C#

```
using UnityEngine;
using Vuforia;

public class AnimateOnDetect : MonoBehaviour
{
   private Animator modelAnimator;
   private ObserverBehaviour observerBehaviour;

   void Start()
   {
      // Find the Animator on the child 3D model
      modelAnimator = GetComponentInChildren<Animator>();
      observerBehaviour = GetComponent<ObserverBehaviour>();

      if (observerBehaviour)
      {
         observerBehaviour.OnTargetStatusChanged += OnTargetStatusChanged;
      }
   }

   private void OnTargetStatusChanged(ObserverBehaviour behaviour, TargetStatus status)
   {
      if (modelAnimator == null) return;
```

```
        if (status.Status == Status.TRACKED)
        {
            // Target is found, play the animation (e.g., "Spin")
            modelAnimator.SetBool("IsVisible", true);
            // Or use modelAnimator.Play("SpinAnimation");
        }
        else
        {
            // Target is lost, stop the animation
            modelAnimator.SetBool("IsVisible", false);
        }
    }
}
```

5.3 ShowHideUI.cs:

This script controls the visibility of the UI Panel (which contains the text translation). The UI Canvas object is made a child of the **Image Target**. This script finds the Canvas and simply activates or deactivates it based on whether the target is being tracked.

CODE:

C#

```
using UnityEngine;

using Vuforia;


public class ShowHideUI : MonoBehaviour

{

   private Canvas uiCanvas;

   private ObserverBehaviour observerBehaviour;


   void Start()

   {

      // Find the Canvas component on a child object

      uiCanvas = GetComponentInChildren<Canvas>();

      observerBehaviour = GetComponent<ObserverBehaviour>();


      if (uiCanvas != null)

      {

         // Start with the UI hidden

         uiCanvas.gameObject.SetActive(false);

      }


      if (observerBehaviour)

      {

         observerBehaviour.OnTargetStatusChanged += OnTargetStatusChanged;

      }

   }
```

```
    private void OnTargetStatusChanged(ObserverBehaviour behaviour, TargetStatus status)
    g
        if (uiCanvas == null) return;

        if (status.Status == Status.TRACKED)
        {
            // Target is found, show the UI
            uiCanvas.gameObject.SetActive(true);
        }
        else
        {
            // Target is lost, hide the UI
            uiCanvas.gameObject.SetActive(false);
        }
    }
}
```

5.4 SceneLoader.cs:

This is a simple helper script placed on a GameManager object in the **Main Menu** scene. It provides public functions that can be linked to UI Buttons (like "Start" or "Quit") to load different scenes or exit the application.

CODE:

C#

```
using UnityEngine;
using UnityEngine.SceneManagement;


public class SceneLoader : MonoBehaviour
{
    // This function is public so it can be linked to a button's OnClick() event
    public void LoadARScene()
    {
        // Loads the scene named "AR_Scene". Ensure this name matches your scene file.
        SceneManager.LoadScene("AR_Scene");
    }

    // This function is public so it can be linked to a button's OnClick() event
    public void QuitApplication()
    {
        // This will close the application
        Application.Quit();
    }
}
```

Chapter 6: Conclusion and Future Scope

6.1 Conclusion:

In conclusion, this project successfully demonstrates the development of an "AR Flashcards" application for language learning. By leveraging Unity 3D and the Vuforia Engine, we created an interactive and immersive tool that bridges the gap between physical learning materials and digital content.

The application successfully meets its core objectives: it accurately recognizes 2D Image Targets (flashcards) and overlays them with corresponding 3D models, audio pronunciations, and text translations. This multi-sensory approach provides an engaging and effective alternative to traditional memorization, especially for young learners. The project serves as a strong proof-of-concept for the power of Augmented Reality in education.

6.2 Future Scope:

The "AR Flashcards" platform has significant potential for expansion. Future development could focus on the following areas:

- **Expanded Content:**
    - Adding multiple language packs (e.g., French, Spanish, German).
    - Expanding the dictionary to include more complex topics like animals, household items, and verbs.
- **Advanced Mechanics and Modes:**
    - Implementing a **"Quiz Mode"** where the app speaks a word, and the user must find and scan the correct flashcard.
    - Adding a **"Pronunciation"** feature where the user can record their own voice and compare it to the correct pronunciation.
- **Gamification:**
    - Adding a points and rewards system for correctly identifying cards.
    - Creating a "Collection" or "Sticker Book" where users can view all the 3D models they have discovered.
- **Technical Enhancements:**
    - Integrating a cloud-based database to add new flashcards without needing to update the app.
    - Exploring **Object Recognition** as an alternative to Image Targets, allowing users to scan real-world objects.
    -

References

[1] Unity Technologies. (2024). *Unity Real-Time Development Platform*. Retrieved from https://unity.com/

[2] PTC Inc. (2024). *Vuforia Engine AR SDK*. Retrieved from https://developer.vuforia.com/

[3] Sketchfab. (2024). *3D Model Asset Library*. Retrieved from https://sketchfab.com/

[4] Freepik. (2024). *Graphic Resources for Everyone*. Retrieved from https://www.freepik.com/

[5] G. Kipper and J. Rampolla, "Augmented Reality: An Emerging Technologies Guide to AR" (2012).

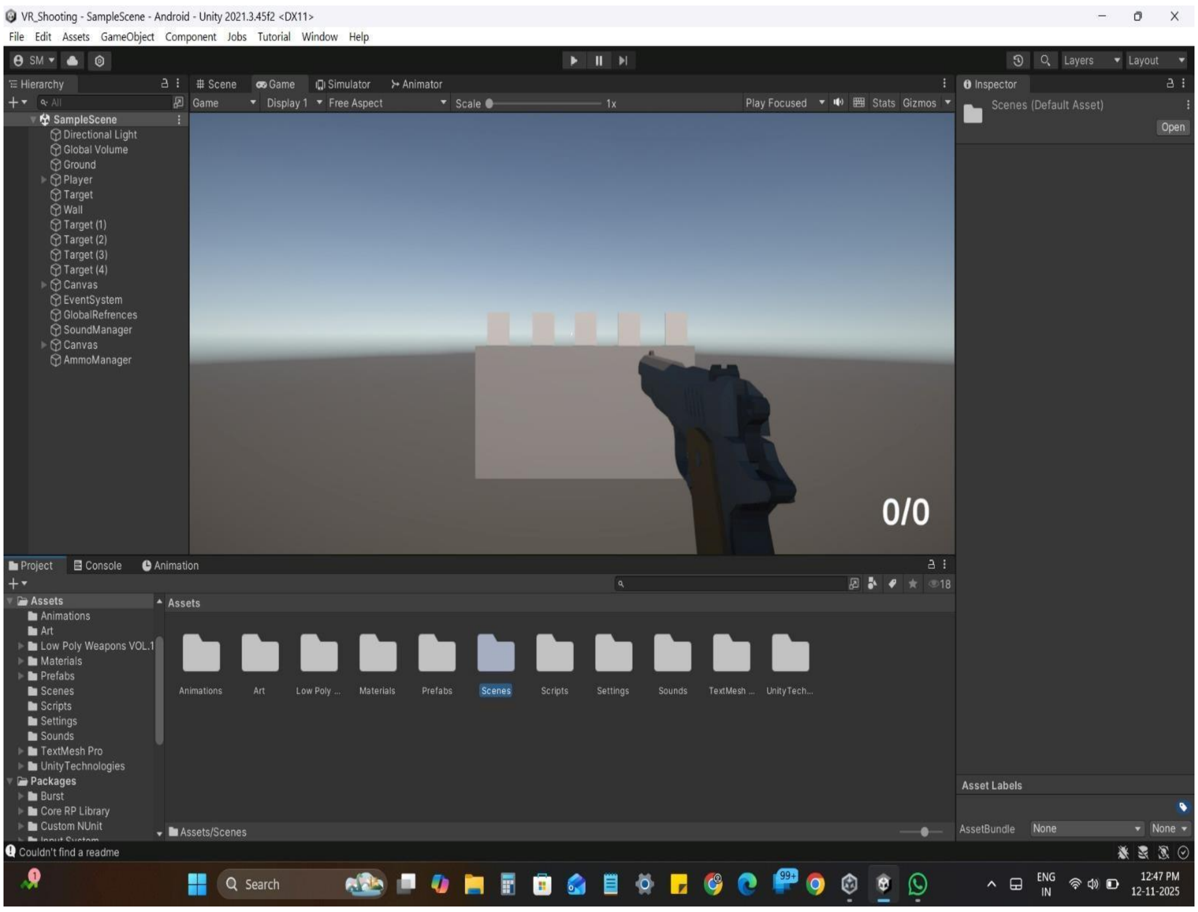
(End of Report Text)

A

Project report

on

<u>VR SHOOTING GAME</u>



Chapter 1: Introduction

For our project, we wanted to dive into Virtual Reality, and we figured the best way to learn was to build something interactive from scratch. We decided to create a classic **VR Shooting Game**. The main goal wasn't to make a highly polished or complex game, but to get a solid grasp of the fundamentals of VR interaction using the modern Unity tools.

The core concept is simple: the player is placed in a basic 3D environment while wearing a VR headset. They hold a virtual gun, which is tracked to their right-hand controller. When they pull the controller's trigger, the gun fires a physics-based bullet. The goal is to shoot and destroy the target cubes that are scattered around the scene. Every time a cube is hit, it disappears, and the player's score—which is displayed on a floating UI screen—goes up by one.

This simple game loop was the perfect way to learn the **Unity XR Interaction Toolkit**. We had to figure out several key concepts:

- Setting up a proper "XR Origin" so the player's real-world movement is tracked.
- Connecting controller inputs (like the trigger pull) to actions in the game.
- Making a physical object (the gun) follow the controller's position and rotation.
- Using Unity's physics system to make a bullet fire forward.
- Handling collisions between the bullet and the targets.
- Displaying a UI in "World Space" so it floats in the VR environment.

This project highlights how VR makes even a simple task like shooting a cube feel much more immersive and engaging than playing on a flat screen.

Chapter 2: Tools and Assets

2.1 Tools and Software Used

- **Unity 3D (Game Engine)**
  - **Description:** Unity is a real-time 3D development platform, or "game engine." It's the main software we used to build the entire project. It gives you a visual editor to build your 3D world and a code editor to script the game's logic.
  - **Why Used:** It's the industry standard for VR development and has the XR Interaction Toolkit, which was perfect for our project. It let us build the 3D world, write C# scripts, and deploy to the headset all from one place.
  - **Key Features Used in This Project:**
    - Physics and Rigidbody system (for the bullets).
    - Prefab system (for making the bullets and target cubes reusable).
    - UI Canvas system (set to World Space for the HUD).
    - Android Build Support (for creating the final APK for the headset).
    - Integration with the XR Interaction Toolkit.
- **XR Interaction Toolkit (Unity Package)**
  - **Description:** This is an official *package* from Unity. It's not the game engine itself, but a high-level framework that sits on top. It's designed to make building VR and AR interactions much, much easier.
  - **Why Used:** This was the most important package for our project. It saved us a *ton* of time by handling all the complicated VR-specific tasks, like tracking the headset/controllers and figuring out what a "trigger pull" or "grip" means.
  - **Key Features Used in This Project:**
    - **XR Origin (Rig):** This prefab is the "player." It contains the camera for the headset and the objects for the left/right hand controllers.
    - **Action-based Input:** We used this new system to map controller buttons to our C# scripts.
    - **XR Grab Interactable:** We used this component to make the gun "grabbable" by the player's hand.
- **OpenXR (Unity Plugin)**
  - **Description:** OpenXR is an open, royalty-free *standard* (or "bridge"). It's not a single tool, but rather a universal language that lets different VR/AR hardware (like Meta, HTC, Pico) talk to different game engines (like Unity) in one common way.

    - **Why Used:** By using this, we made sure our project is cross-platform. We didn't have to write code *just* for the Oculus headset. Any headset that supports the OpenXR standard can run our game.
    - **Key Features Used in This Project:**
        - Cross-platform VR compatibility.
        - Handles the low-level communication between Unity and the headset.
- **Visual Studio Code**
    - **Description:** This is a lightweight but very powerful source code editor from Microsoft. It's where we actually wrote and edited our C# scripts.
    - **Why Used:** It connects perfectly with Unity. It gives us features like syntax highlighting and error checking (IntelliSense) that make coding way faster and less error-prone than using a basic text editor.
    - **Key Features Used in This Project:**
        - C# syntax highlighting and auto-completion.
        - Integrated terminal for any command-line needs.
        - Unity debugging tools.
- **VR Headset (Meta Quest 2)**
    - **Description:** This is the physical, all-in-one (standalone) VR hardware we used. It has the headset, two motion controllers, and doesn't need to be connected to a PC to run.
    - **Why Used:** This was our target platform and our main testing device. We built the .apk file from Unity and loaded it directly onto the Quest 2 to test our game in a real, untethered VR environment.
    - **Key Features Used in This Project:**
        - 6-DoF (Six Degrees of Freedom) tracking for head and hands.
        - Two motion controllers with triggers, grips, and joysticks.
        - Standalone (Android-based) operation.

2.2 Assets Used

- **Gun Model (3D Model)**
    - A free, low-polygon pistol model we downloaded from Sketchfab.
    - **Key Role:** Acts as the player's virtual tool. We attached our GunShoot.cs script to this.
    - **Why Used:** Having a visual gun model that tracks to your hand is much more immersive than just shooting from an invisible point.

- **Bullet Prefab (Unity Primitive)**
  - A simple Unity sphere, scaled down, with a Rigidbody and our Bullet.cs script.
  - **Key Role:** This is the projectile. It gets "instantiated" (created) at the gun's barrel and fires forward.
  - **Why Used:** We made it a prefab so we could spawn unlimited copies of it. Using a Rigidbody makes its movement realistic.
- **Target Cubes (Unity Primitive)**
  - Basic Unity cubes with a bright red material.
  - **Key Role:** These are the targets. The Bullet.cs script looks for objects with the "Target" tag.
  - **Why Used:** They are simple, clear, and easy for the physics system to detect collisions with.
- **Plane (Environment Base)**
  - A large, flat Unity plane that acts as the ground.
  - **Key Role:** Gives the player a sense of space and a floor for the targets to sit on.
  - **Why Used:** The quickest way to create a "room" or "world" for the player to stand in.
- **UI Canvas (Score Display)**
  - A standard Unity Canvas set to "World Space" mode, with a TextMeshPro text element.
  - **Key Role:** Displays the player's score.
  - **Why Used:** In VR, UI can't be stuck to the "screen" like in a 2D game. It has to exist in the world. This floating HUD is easy for the player to see.

2.3 Key Features of the Game

- Real-time VR head and controller tracking.
- Physics-based "projectile" shooting (not just a raycast).
- Interactive targets that can be destroyed.
- A simple game loop: shoot, score, repeat.
- A functional "World Space" UI that displays the score.
- Built as a standalone .apk that runs on the Quest 2.

Chapter 3: Instructions to Play

Starting the Application:

1. The project is built as an .apk file. This file must be sideloaded onto the Meta Quest 2 headset.
2. Once installed, find the game in your "Unknown Sources" app library on the headset.
3. Launch the game. You will be placed directly into the 3D environment.

Gameplay / How to Play:

1. You will see a gun floating in front of you. Reach out with your virtual right hand (by physically reaching) and press the **Grip Button** on the side of the controller to pick up the gun. It will snap to your hand.
2. Look around the environment. You will see several red cubes placed on the ground plane.
3. Aim the gun at a cube by pointing your controller.
4. Pull the **Trigger Button (Index Finger)** on your controller to fire a bullet.
5. When your bullet hits a red cube, the cube will disappear.

Interacting and Scoring:

- Look for the floating scoreboard in your view.
- Every time you successfully destroy a cube, the score will increase by **1 point**.
- You have unlimited bullets. The goal is to destroy all the cubes.

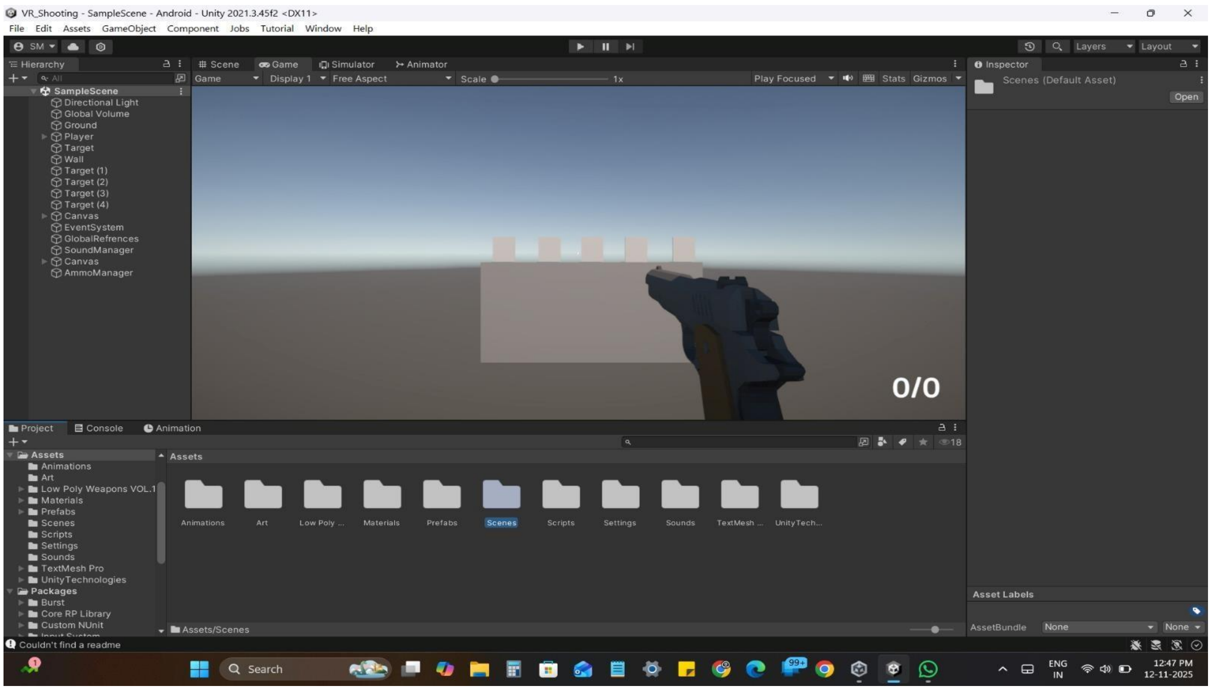
Exiting:

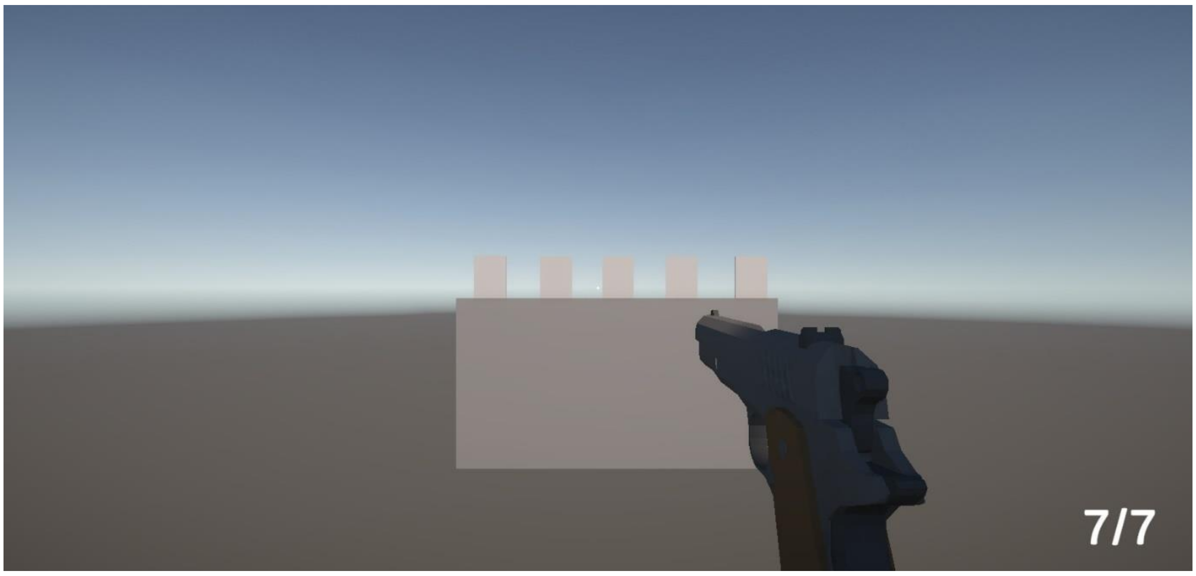
- To exit the game, press the **Oculus/Menu Button** on your left controller to bring up the universal Quest menu and quit the application from there.

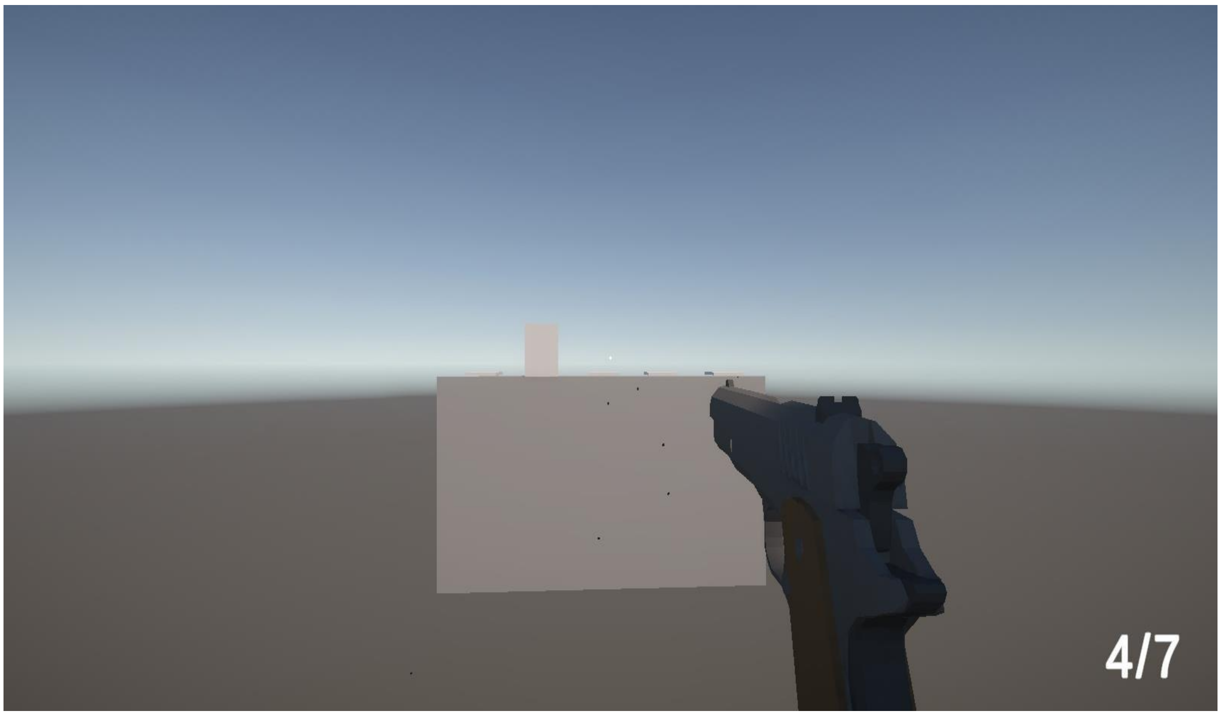
Chapter 4: Output

The final output of this project is a complete, working VR game loop. The player can successfully load into the environment, see their controllers tracked in real-time, grab the virtual gun, and use it to fire projectiles.

The shooting mechanic works as expected: bullets are instantiated at the barrel of the gun and travel forward using Unity's physics. When a bullet collides with an object tagged as "Target" (our red cubes), the collision is correctly detected, the cube is destroyed, and the ScoreManager is notified. The World Space UI correctly updates the score in real-time, providing instant feedback to the player.







Expected Outcomes (and final result):

- The player's head and hand movements are tracked 1-to-1 in VR.
- The gun can be picked up with the grip button and attaches correctly to the hand.
- The trigger button successfully fires a bullet prefab.
- Bullets travel forward and collide with the target cubes.
- Cubes are destroyed on impact.
- The score on the floating HUD increments by 1 for each cube destroyed.
- The game runs smoothly on the Meta Quest 2 without dropping frames.

Chapter 5: Code Explanation

(Here are the main scripts we wrote. You can add 1–2 lines of explanation before each code block in your Word report.)

This was the main script we attached to the gun model. It needs a reference to the bulletPrefab and the spawnPoint (an empty object we placed at the gun's barrel). We linked our "Shoot" input action to the Fire() function.

5.1 GunShoot.cs

C#

```
using UnityEngine;
using UnityEngine.InputSystem;

public class GunShoot : MonoBehaviour
{
    // Assign the bullet prefab in the Inspector
    public GameObject bulletPrefab;

    // Assign an empty 'Transform' object at the gun's barrel
    public Transform spawnPoint;

    // Set the speed of the bullet
    public float bulletSpeed = 20f;

    // This function is called by the 'Input Action' (when trigger is pressed)
    public void Fire(InputAction.CallbackContext context)
    {
        // Check if the button was just pressed down
        if (context.performed)
        {
            // Create a new bullet at the spawn point's position and rotation
            GameObject bullet = Instantiate(bulletPrefab, spawnPoint.position, spawnPoint.rotation);
            // Get the Rigidbody from the bullet we just made
            Rigidbody rb = bullet.GetComponent<Rigidbody>();

            // Add force to the bullet to make it move forward
            rb.AddForce(spawnPoint.forward * bulletSpeed, ForceMode.Impulse);

            // Destroy the bullet after 3 seconds so they don't fill the scene
            Destroy(bullet, 3.0f);
        }
    }
}
```

This script is super simple and is attached to the bulletPrefab. Its only job is to detect when it hits something.

5.2 Bullet.cs

C#

```
using UnityEngine;

public class Bullet : MonoBehaviour
{
    // This function runs automatically when the bullet's collider hits another collider
    private void OnCollisionEnter(Collision collision)
    {
        // Check if the object we hit has the "Target" tag (our red cubes)
        if (collision.gameObject.CompareTag("Target"))
        {
            // Destroy the target object
            Destroy(collision.gameObject);

            // Find the ScoreManager in the scene and tell it to add a point
            // This is a simple way to find it, but for a bigger game, we'd use a singleton
            FindObjectOfType<ScoreManager>().AddPoint();
        }

        // Destroy the bullet itself no matter what it hits
        Destroy(gameObject);
    }
}
```

This script manages the score and updates the UI. We attached it to an empty "GameManager" object in the scene.

5.3 ScoreManager.cs

C#

```
using UnityEngine;
using TMPro; // Import TextMeshPro library

public class ScoreManager : MonoBehaviour
{
    // Assign the TextMeshPro UI element in the Inspector
    public TextMeshProUGUI scoreText;

    // Keep track of the current score
    private int currentScore = 0;

    void Start()
    {
        // Set the initial score text
        UpdateScoreText();
    }

    // This is a public function that other scripts (like Bullet.cs) can call
    public void AddPoint()
    {
        currentScore++;
        UpdateScoreText();
    }

    // A private function to update the UI
    private void UpdateScoreText()
    {
        scoreText.text = "Score: " + currentScore.ToString();
    }
}
```

Chapter 6: Conclusion and Future Scope

6.1 Conclusion

This project was a complete success. We fully achieved our goal of learning the fundamentals of VR development with Unity. We successfully built a stable VR Shooting Game from scratch, implementing core mechanics like player tracking, input handling, physics-based shooting, and a world-space UI.

We learned how the XR Interaction Toolkit works and how to connect all the different pieces (the player rig, the interactable objects, and the input actions) to make a cohesive experience. This project highlights how even simple physics and interactions can feel incredibly immersive in VR, and it gives us a strong foundation for building more complex projects in the future.

6.2 Future Scope

Our game is a simple loop, but it could be expanded in many fun ways. If we were to continue this project, our next steps would be:

- **Add Moving Targets:** Instead of static cubes, we'd create targets that move back and forth to make aiming more challenging.
- **Create a "Wave" System:** Make targets spawn in waves, with each wave getting progressively harder.
- **Add Sound and Effects:** Add a "bang" sound for the gun and a "pop" or explosion particle effect when a cube is destroyed. This would make it feel much more satisfying.
- **Implement a Timer/Challenge Mode:** Add a 60-second timer and see how high a score the player can get.
- **Different Weapons:** Add other guns, like a shotgun or a rifle, that the player could swap between.
- **Real Environment:** Replace the simple plane with a 3D "shooting gallery" environment to improve immersion.

References

[1] Unity Technologies, “Unity User Manual 2022.3 (LTS) - XR Interaction Toolkit,” *Unity Documentation*, 2024. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.5/manual/index.html

[2] Khronos Group, “OpenXR Specification,” *Khronos Group*, 2024. [Online]. Available: https://www.khronos.org/openxr/

[3] Meta, “Introduction to VR Development in Unity,” *Oculus Developer Center*, 2024. [Online]. Available: https://developer.oculus.com/documentation/unity/

[4] J. Gregory, *Game Engine Architecture, Third Edition*. A K Peters/CRC Press, 2018.

[5] C. Anthes, R. J. García-Hernández, M. Wiedemann, and D. Kranzlmüller, "State of the art of virtual reality technology," in *2016 IEEE Aerospace Conference*, 2016.