PROJECT – 2

MR TARGET SHOOTER

(Optimized)

A Mixed Reality Game for Meta Quest 3

by **Heera Menon**

MR Target Shooter – A Mixed Reality Game for Meta Quest 3

1. Introduction

1.1 Motivation

This project explores how Mixed Reality (MR) can transform traditional gameplay by blending digital elements into the player's real environment. Using Meta Quest 3's advanced passthrough, we designed a shooter game that starts instantly, adapts to physical space, and delivers intuitive interaction. Unlike VR games that isolate users, MR enables natural movement, spatial awareness, and immersive feedback through visuals and haptics. The goal was to create a fast, responsive experience with minimal setup—just pull the trigger and play. Inspired by arcade shooters and spatial computing, this game reimagines classic mechanics for the MR era.

1.2 What I Built

I built a Mixed Reality shooter game for Meta Quest 3 using Unity, XR Interaction Toolkit and OVR plugins. The game starts with a trigger pull and spawns virtual targets in passthrough view.

Players aim using a laser pointer from the controller, and raycasts detect hits. Each hit triggers visual and haptic feedback—flashing lasers, hit markers, and vibrations. Targets animate on spawn, react with effects when hit, and optionally float for added challenge. UI panels display score, timer, and game status using TextMeshPro. The codebase is modular, with dedicated scripts for game flow, shooting, target behavior, and UI interaction

VIDEO DEMO: https://drive.google.com/file/d/1Utx3Z-OQBEZzygRk3gbMnfMrHNbDY2mN/view?usp=sharing

1.3 The Game Rules

Start the Game

- Pull the trigger on your right controller to begin.

Choose Mode

- Point to click mode – EASY or HARD
- EASY mode is hitting 25 targets in 15 seconds
- HARD mode is hitting 20 targets in 10 seconds

Time Limit

- You have a fixed number of seconds to complete the challenge.

Scoring

- Each target hit gives you 10 points.
- Your final score is based on total hits and accuracy.

Accuracy Bonus

Accuracy is calculated as ***Targets Hit ÷ Total Targets × 100%***

Game Over

- Game ends when : All targets are hit OR time runs out
- Final stats are displayed: Score, Hits, Accuracy

Restart

- Pull the trigger again or press the Play Again button to restart.

2 Related Work

2.1 Inspiration and Ideation

MR Target Shooter

- Inspired by MR to blend physical and digital world compared to traditional games that isolate players
- Meta Quest 3's advanced passthrough feature holds new design possibilities
- Influenced by arcade shooters but reimagined for spatial computing

GOAL

Build a MR shooter game that starts instantly, feels intuitive and adapts to your space

VIDEO DEMO: https://drive.google.com/file/d/1Utx3Z-OQBEZzygRk3gbMnfMrHNbDY2mN/view?usp=sharing

IDEATION

- Trigger-based start
- Dynamic target spawning based on player position
- Simple UI with score, timer and game over status
- Iterated on player interaction, target behavior and UI polish

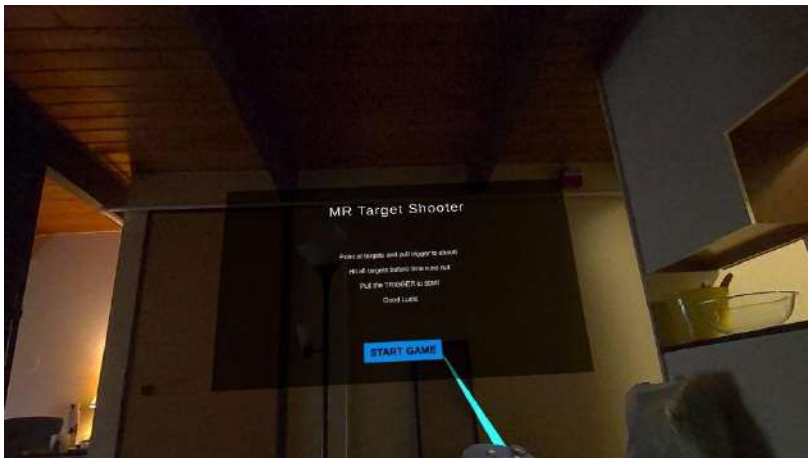
GOALS FOCUSED

- Using passthrough as the default environment
- Trigger-based input for control
- Responsive UI and haptics for immersive feedback
- Modular, scalable architecture open for future expansion

3 Implementation

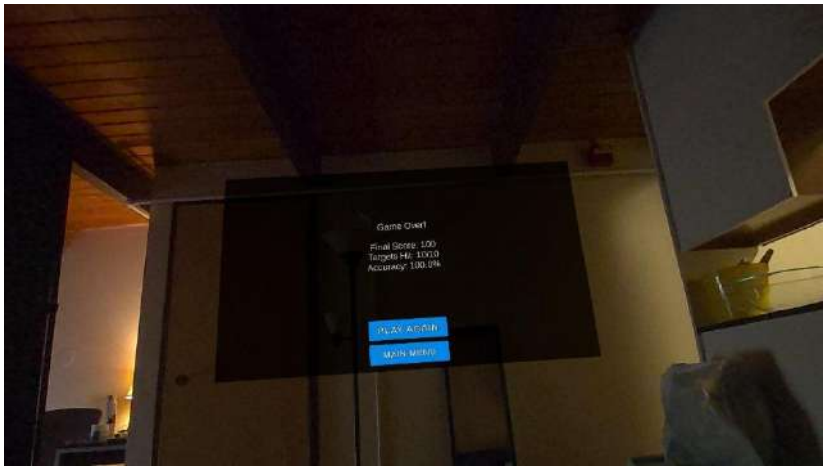
3.1 Design

Game Start Menu



Gaming Mode

Game Restart/ Game Over Menu



3.2 System Overview

```
Assets/
├── Scenes/
│   └── MRTargetGame.unity
├── Scripts/
│   ├── GameManager.cs
│   ├── TargetManager.cs
│   ├── Target.cs
│   ├── OVRShooter.cs
│   └── UIButtonHandler.cs
├── Prefabs/
│   ├── Target.prefab
│   ├── HitEffect.prefab
│   └── HitMarker.prefab
├── Materials/
│   ├── TargetMaterial.mat
│   └── HitMarkerMaterial.mat
├── Plugins/
│   └── Android/
│       └── AndroidManifest.xml
└── [Other Unity/OVR folders]

Builds/
└── MRTargetGame.apk
```

Added: *GreenFireworks* prefab, *TargetRingMaterial* material, *Difficulty* panel – EasyButton, *Difficulty* panel – HardButton

3.3 Updates from First Prototype

The current model **is** the same MR shooter game that starts instantly, feels intuitive and adapts to your space but now comes with:

1. There are 2 levels in menu to choose – EASY and HARD
   - EASY mode is hitting 25 targets in 15 seconds
   - HARD mode is hitting 20 targets in 10 seconds
2. To increase difficulty & have a fun interaction HARD mode feature enables targets to:
   - spawn faster
   - slides across the horizontal plane
3. Targets on being hit will now pulse and bursts into sparks
4. Targets can now spawn ANYWHERE even behind you!
5. Targets can now spawn one behind the other as well
6. Some random targets can hide behind real objects in MR vision
7. Fixed MENU and START functionalities on the screen

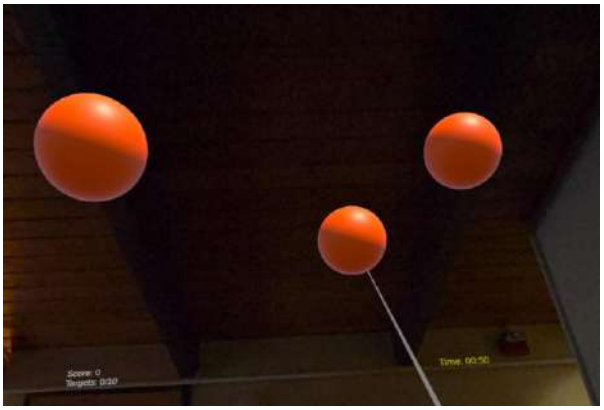
3.4 Scenes and Objects

How Targets Are Designed to React in the Updated Model

- Each target is a small sphere that appears in front of the player
- Targets are placed on a layer called "Target" so the shooter knows what to hit.
- They use a red-orange material to stand out visually
- When spawned, they scale up smoothly to grab attention
- If movement is enabled, they float gently side to side for added challenge
- In hard mode, targets slide sideways on the horizontal plane.

When hit:

- They change color, pulse in size & fade out
- A particle effect plays to show impact
- The target disappears and score increases



How Shooting Feels Responsive

- A laser line comes from the controller, showing where you're aiming

When you pull the trigger:
- A raycast checks if you're pointing at a target

If you hit:
- A red hit marker appears briefly at the impact point
- The controller vibrates for tactile feedback
- The laser flashes green, pulses and bursts into sparks

How the Game Communicates with You

- The game uses TextMeshPro UI panels floating in space:
- Start screen with instructions and a bold title
- Game screen showing score, targets hit and timer
- Game over screen with final stats and restart options

Buttons respond with:
- Color changes and scale-up effects when hovered
- Click vibrations for confirmation
- Particle effects and hit markers add visual clarity to each interaction

3.5 Core Modules

01

GameManager.cs

[central game controller]

Purpose

Controls the entire game lifecycle: starting, scoring, timing, ending and UI transitions

Tasks

- Manages game state (gameActive, targetsHit, currentScore)
- Handles trigger input to start/restart the game
- Updates UI elements (score, timer, game over status)

Main Methods

- **StartGame():** Initializes game variables and activates gameplay UI
- **AddScore(int points):** Increments score and checks win condition
- **UpdateTimerUI():** Formats and displays countdown timer
- **EndGame():** Stops target spawning and shows final stats
- **RestartGame():** Reinitializes the game
- **ShowStartScreen():** Displays the start menu

02
TargetManager.cs
[dynamic target spawner]

Purpose

Spawns and manages targets in 3D space with difficulty scaling

Tasks

- Spawns targets within a defined volume around the player
- Ensures targets are spaced from the player (min/max distance)
- Controls spawn rate and difficulty progression
- Destroys targets after a lifetime or when hit

Main Methods

- **StartSpawning():** Begins coroutine to spawn targets
- **SpawnTarget():** Instantiates a target prefab at a random valid position
- **GetRandomSpawnPosition():** Calculates spawn location with distance checks
- **DestroyTargetAfterTime():** Removes target if not hit
- **ClearTargets():** Cleans up all active targets

03
Target.cs
[target behaviour & effects]

Purpose

Defines how each target behaves, reacts to hits, and animates

Tasks

- Handles hit detection and scoring
- Plays visual effects and animations on hit
- Optional movement logic for dynamic targets

Main Methods

- **OnTargetHit():** Triggers scoring and visual feedback
- **HitEffect():** Plays color fade, scale pulse, and particle effect
- **SpawnAnimation():** Smooth scale-up when spawned
- **Update():** Moves target if movement is enabled

04
OVRShooter.cs
[shooting & interaction]

Purpose

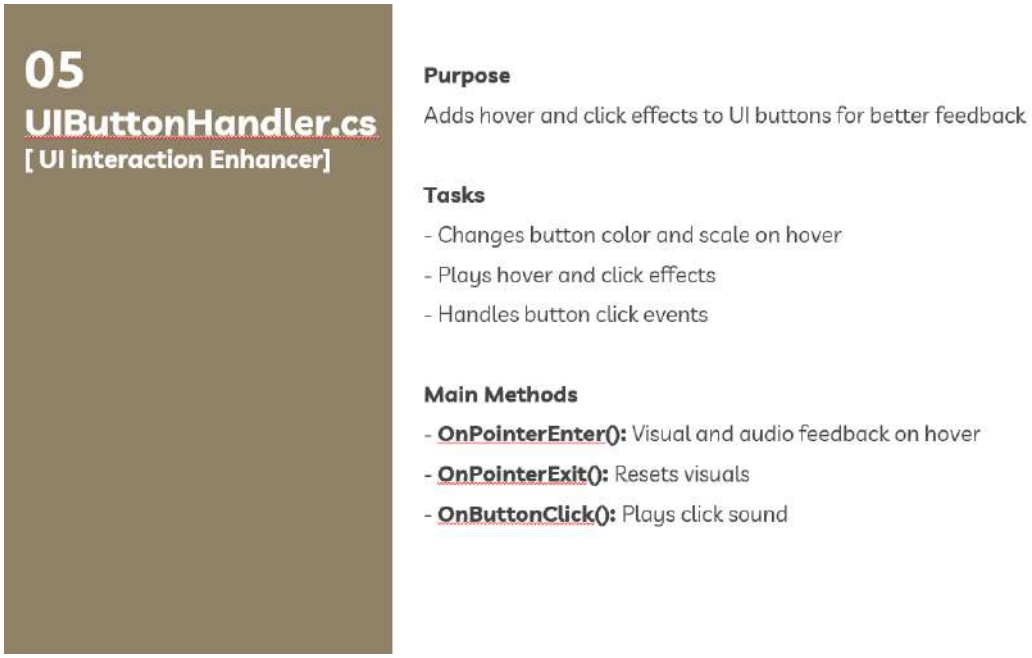
Handles raycast shooting, laser visuals, haptics and hit detection

Tasks

- Shoots ray from controller to detect targets
- Updates laser pointer visuals
- Plays haptic feedback and hit marker
- Detects trigger press to fire

Main Methods

- **UpdateLaser():** Draws laser line and changes color on hit
- **Shoot():** Performs raycast and triggers target hit logic
- **FlashLaser():** Brief color flash for visual feedback
- **StopHaptics():** Ends vibration after short delay



3.6 Unity Setup and Codes

PART 1: UNITY PROJECT SETUP

Step 1: Create New Unity Project

1. Open Unity Hub
2. Click New Project
3. Select 3D (URP) template
4. Name: MRTargetGame
5. Click Create Project

Step 2: Install Required Packages

1. Window → Package Manager
2. Click + button → Add package by name
3. Add these packages:
    - com.unity.xr.interaction.toolkit
    - com.unity.textmeshpro
    - com.unity.inputsystem
4. When TextMeshPro prompts, click Import TMP Essentials

Step 3: Install Meta XR SDK (If Not Already Present)

1. In Package Manager, click + → Add package by name
2. Enter: com.meta.xr.sdk.core
3. Click Add

Note: If your project already has OVR Building Blocks, skip this step.

PART 2: PROJECT SETTINGS CONFIGURATION

Step 1: Configure Build Settings

1. File → Build Settings
2. Select Android
3. Click Switch Platform (if not already on Android)
4. Click Player Settings

Step 2: Player Settings - Android

Edit → Project Settings → Player

Select Android tab (Android robot icon):

Other Settings:

- Color Space: Linear
- Auto Graphics API: UNCHECKED ❌
- Graphics APIs: OpenGLES3 only (remove Vulkan)
    - Click on Vulkan → Click "-" button to remove
- Multithreaded Rendering: Checked ✅
- Package Name: com.yourstudio.mrtargetgame
- Minimum API Level: Android 10.0 (API Level 29)
- Target API Level: Android 13 (API Level 33) or Automatic
- Scripting Backend: IL2CPP
- Target Architectures: ARM64 only ✅ (uncheck ARMv7)
- Install Location: Automatic

Identification:

- Version: 0.1
- Bundle Version Code: 1

Other Settings → Configuration:

- Scripting Define Symbols: (leave as is)
- Debug Symbols: Full

Step 3: XR Plug-in Management

Edit → Project Settings → XR Plug-in Management

Android tab:

- OpenXR: UNCHECKED ❌ (we're using OVR system)

Note: If you keep OpenXR enabled, ensure:

- Graphics API includes Vulkan
- Fix all validation issues

Step 4: Disable Analytics

Edit → Preferences → Analytics

- Enable Editor Analytics: UNCHECKED ❌
- Send Usage Statistics: UNCHECKED ❌

Close all settings windows

PART 3: CREATE FOLDER STRUCTURE

In Project window, create these folders:

1. Right-click Assets → Create → Folder
2. Create folders:
    - Scripts
    - Prefabs
    - Materials
    - Scenes

PART 4: CREATE ALL SCRIPTS

Script 1: GameManager.cs

1. Right-click Scripts folder → Create → C# Script
2. Name: GameManager
3. Open script and replace all content with:

```
using UnityEngine;
using TMPro;

public class GameManager : MonoBehaviour
{
    public static GameManager Instance;

    [Header("Game Settings")]
    public int totalTargets = 10;
    public float gameDuration = 60f;

    [Header("Difficulty Settings")]
    public int easyTargets = 15;
    public float easyTime = 25f;
    public int hardTargets = 35;
    public float hardTime = 15f;

    [Header("UI References")]
    public TextMeshProUGUI scoreText;
```

```
    public TextMeshProUGUI timerText;
    public TextMeshProUGUI gameOverText;
    public GameObject startPanel;
    public GameObject difficultyPanel;
    public GameObject gamePanel;
    public GameObject gameOverPanel;

    [Header("Audio")]
    public AudioSource audioSource;
    public AudioClip hitSound;
    public AudioClip missSound;
    public AudioClip gameOverSound;

    private int currentScore = 0;
    private float timeRemaining;
    private bool gameActive = false;
    private int targetsHit = 0;
    private string currentDifficulty = "EASY";

    void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }

    void Start()
    {
        ShowStartScreen();
    }

    void Update()
    {
        if (gameActive)
        {
            timeRemaining -= Time.deltaTime;
            UpdateTimerUI();

            if (timeRemaining <= 0)
            {
                EndGame();
            }
        }
        else
        {
            if (startPanel.activeSelf)
            {
```

```
            if (OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger) ||
                OVRInput.GetDown(OVRInput.Button.One))
            {
                ShowDifficultyMenu();
            }
        }
        else if (gameOverPanel.activeSelf)
        {
            if (OVRInput.GetDown(OVRInput.Button.PrimaryIndexTrigger) ||
                OVRInput.GetDown(OVRInput.Button.One))
            {
                ShowDifficultyMenu();
            }
        }
    }
}

public void ShowDifficultyMenu()
{
    startPanel.SetActive(false);
    difficultyPanel.SetActive(true);
    gamePanel.SetActive(false);
    gameOverPanel.SetActive(false);
}

public void StartEasyGame()
{
    Debug.Log("Starting Easy Game!");
    currentDifficulty = "EASY";
    totalTargets = easyTargets;
    gameDuration = easyTime;
    StartGame();
}

public void StartHardGame()
{
    Debug.Log("Starting Hard Game!");
    currentDifficulty = "HARD";
    totalTargets = hardTargets;
    gameDuration = hardTime;
    StartGame();
}

public void StartGame()
{
    currentScore = 0;
    targetsHit = 0;
    timeRemaining = gameDuration;
```

```
        gameActive = true;

        startPanel.SetActive(false);
        difficultyPanel.SetActive(false);
        gamePanel.SetActive(true);
        gameOverPanel.SetActive(false);

        UpdateScoreUI();
        TargetManager.Instance.StartSpawning();
    }

    public void AddScore(int points)
    {
        currentScore += points;
        targetsHit++;
        UpdateScoreUI();

        if (audioSource && hitSound)
            audioSource.PlayOneShot(hitSound);

        if (targetsHit >= totalTargets)
        {
            EndGame();
        }
    }

    public void PlayMissSound()
    {
        if (audioSource && missSound)
            audioSource.PlayOneShot(missSound);
    }

    void UpdateScoreUI()
    {
        scoreText.text = $"Score: {currentScore}\nTargets:
{targetsHit}/{totalTargets}";
    }

    void UpdateTimerUI()
    {
        int minutes = Mathf.FloorToInt(timeRemaining / 60);
        int seconds = Mathf.FloorToInt(timeRemaining % 60);
        timerText.text = $"Time: {minutes:00}:{seconds:00}";

        if (timeRemaining < 5f)
            timerText.color = Color.red;
        else
            timerText.color = Color.yellow;
```

```
    }

    void EndGame()
    {
        gameActive = false;
        TargetManager.Instance.StopSpawning();

        gamePanel.SetActive(false);
        gameOverPanel.SetActive(true);

        float accuracy = totalTargets > 0 ? (targetsHit / (float)totalTargets)
* 100 : 0;
        string result = (targetsHit >= totalTargets) ? "VICTORY!" : "GAME
OVER!";
        gameOverText.text = $"{result}\n\nDifficulty:
{currentDifficulty}\nFinal Score: {currentScore}\nTargets Hit:
{targetsHit}/{totalTargets}\nAccuracy: {accuracy:F1}%";

        if (audioSource && gameOverSound)
            audioSource.PlayOneShot(gameOverSound);
    }

    public void RestartGame()
    {
        ShowDifficultyMenu();
    }

    public void ShowStartScreen()
    {
        startPanel.SetActive(true);
        difficultyPanel.SetActive(false);
        gamePanel.SetActive(false);
        gameOverPanel.SetActive(false);
        gameActive = false;
    }
}
```

Script 2: TargetManager.cs

1. Right-click Scripts folder → Create → C# Script
2. Name: TargetManager
3. Replace content with:

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

```
public class TargetManager : MonoBehaviour
{
    public static TargetManager Instance;

    [Header("Target Settings")]
    public GameObject targetPrefab;
    public int maxActiveTargets = 5;
    public float spawnInterval = 1.5f;
    public float targetLifetime = 3.5f;

    [Header("Spawn Area")]
    public Vector3 spawnAreaCenter = new Vector3(0, 1.5f, 2f);
    public Vector3 spawnAreaSize = new Vector3(5f, 2f, 3f);
    public float minDistanceFromPlayer = 1.5f;
    public float maxDistanceFromPlayer = 4f;

    [Header("Difficulty")]
    public bool increaseDifficulty = true;
    public float difficultyIncreaseRate = 0.2f;

    [Header("Difficulty Modifiers")]
    public float easySpawnMultiplier = 1.0f;
    public float hardSpawnMultiplier = 0.3f;

    private List<GameObject> activeTargets = new List<GameObject>();
    private bool isSpawning = false;
    private float currentSpawnInterval;
    private Transform cameraTransform;

    void Awake()
    {
        if (Instance == null)
            Instance = this;
        else
            Destroy(gameObject);
    }

    void Start()
    {
        cameraTransform = Camera.main.transform;
        currentSpawnInterval = spawnInterval;
    }

    public void StartSpawning()
    {
        isSpawning = true;
```

```
        if (GameManager.Instance != null)
        {
            if (GameManager.Instance.totalTargets >= 30)
            {
                currentSpawnInterval = spawnInterval * hardSpawnMultiplier;
                maxActiveTargets = 8;
                StartCoroutine(InitialBurst(5));
            }
            else
            {
                currentSpawnInterval = spawnInterval * easySpawnMultiplier;
                maxActiveTargets = 4;
            }
        }
        else
        {
            currentSpawnInterval = spawnInterval;
        }

        ClearTargets();
        StartCoroutine(SpawnRoutine());
    }

    IEnumerator InitialBurst(int count)
    {
        for (int i = 0; i < count; i++)
        {
            SpawnTarget();
            yield return new WaitForSeconds(0.2f);
        }
    }

    public void StopSpawning()
    {
        isSpawning = false;
        StopAllCoroutines();
        ClearTargets();
    }

    IEnumerator SpawnRoutine()
    {
        while (isSpawning)
        {
            if (activeTargets.Count < maxActiveTargets)
            {
                SpawnTarget();
            }
```

```
            yield return new WaitForSeconds(currentSpawnInterval);

            if (increaseDifficulty)
            {
                currentSpawnInterval = Mathf.Max(0.3f, currentSpawnInterval -
difficultyIncreaseRate);
            }
        }
    }

    void SpawnTarget()
    {
        Vector3 spawnPosition = GetRandomSpawnPosition();
        GameObject target = Instantiate(targetPrefab, spawnPosition,
Quaternion.identity);

        target.transform.LookAt(cameraTransform);
        target.transform.Rotate(0, 180, 0);

        activeTargets.Add(target);

        StartCoroutine(DestroyTargetAfterTime(target, targetLifetime));
    }

    Vector3 GetRandomSpawnPosition()
    {
        Vector3 randomPos;
        int attempts = 0;

        do
        {
            float x = Random.Range(-spawnAreaSize.x / 2, spawnAreaSize.x / 2);
            float y = Random.Range(-spawnAreaSize.y / 2, spawnAreaSize.y / 2);
            float z = Random.Range(-spawnAreaSize.z / 2, spawnAreaSize.z / 2);

            randomPos = spawnAreaCenter + new Vector3(x, y, z);
            randomPos = cameraTransform.position +
cameraTransform.TransformDirection(randomPos - cameraTransform.position);

            attempts++;
        }
        while (Vector3.Distance(randomPos, cameraTransform.position) <
minDistanceFromPlayer && attempts < 10);

        return randomPos;
    }

    IEnumerator DestroyTargetAfterTime(GameObject target, float time)
```

```
    {
        yield return new WaitForSeconds(time);

        if (target != null)
        {
            RemoveTarget(target);
            GameManager.Instance.PlayMissSound();
        }
    }

    public void RemoveTarget(GameObject target)
    {
        if (activeTargets.Contains(target))
        {
            activeTargets.Remove(target);
            Destroy(target);
        }
    }

    void ClearTargets()
    {
        foreach (GameObject target in activeTargets)
        {
            if (target != null)
                Destroy(target);
        }
        activeTargets.Clear();
    }

    void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireCube(spawnAreaCenter, spawnAreaSize);
    }
}
```

Script 3: Target.cs

1. Right-click Scripts folder → Create → C# Script
2. Name: Target
3. Replace content with:

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

```
public class Target : MonoBehaviour
{
    [Header("Target Properties")]
    public int points = 10;

    [Header("Visual Effects")]
    public GameObject hitEffectPrefab;

    [Header("Audio")]
    public AudioClip burstSound;

    [Header("Movement (Optional)")]
    public bool enableMovement = false;
    public float moveSpeed = 1f;
    public float moveRange = 0.5f;

    private Renderer targetRenderer;
    private Color originalColor;
    private Vector3 originalScale;
    private Vector3 startPosition;
    private bool isHit = false;
    private float moveTime = 0f;
    private AudioSource audioSource;

    void Start()
    {
        targetRenderer = GetComponent<Renderer>();
        if (targetRenderer != null)
            originalColor = targetRenderer.material.color;

        originalScale = transform.localScale;
        startPosition = transform.position;

        audioSource = gameObject.AddComponent<AudioSource>();
        audioSource.playOnAwake = false;
        audioSource.spatialBlend = 1f;
        audioSource.maxDistance = 10f;

        if (GameManager.Instance != null && GameManager.Instance.totalTargets >=
30)
        {
            enableMovement = true;
            moveSpeed = Random.Range(0.8f, 1.5f);
        }

        StartCoroutine(SpawnAnimation());
    }
```

```
    void Update()
    {
        if (enableMovement && !isHit)
        {
            moveTime += Time.deltaTime * moveSpeed;
            float offsetX = Mathf.Sin(moveTime) * moveRange;
            float offsetY = Mathf.Cos(moveTime * 0.7f) * moveRange * 0.5f;
            transform.position = startPosition + new Vector3(offsetX, offsetY, 0);
        }
    }

    IEnumerator SpawnAnimation()
    {
        transform.localScale = Vector3.zero;
        float elapsed = 0f;
        float duration = 0.3f;

        while (elapsed < duration)
        {
            elapsed += Time.deltaTime;
            float progress = elapsed / duration;
            transform.localScale = Vector3.Lerp(Vector3.zero, originalScale,
progress);
            yield return null;
        }

        transform.localScale = originalScale;
    }

    public void OnTargetHit()
    {
        if (isHit) return;

        isHit = true;
        GameManager.Instance.AddScore(points);
        StartCoroutine(FireworksBurstEffect());
    }

    IEnumerator FireworksBurstEffect()
    {
        if (burstSound != null && audioSource != null)
        {
            audioSource.PlayOneShot(burstSound);
        }

        if (hitEffectPrefab != null)
        {
```

```
            GameObject effect = Instantiate(hitEffectPrefab, transform.position,
Quaternion.identity);
            Destroy(effect, 3f);
        }

        if (targetRenderer != null)
        {
            targetRenderer.material.color = Color.green;
            targetRenderer.material.EnableKeyword("_EMISSION");
            targetRenderer.material.SetColor("_EmissionColor", Color.green * 2f);
        }

        float burstDuration = 0.15f;
        float elapsed = 0f;

        while (elapsed < burstDuration)
        {
            elapsed += Time.deltaTime;
            float progress = elapsed / burstDuration;
            float scale = Mathf.Lerp(1f, 1.5f, progress);
            transform.localScale = originalScale * scale;
            yield return null;
        }

        yield return new WaitForSeconds(0.1f);

        elapsed = 0f;
        float fadeDuration = 0.2f;

        while (elapsed < fadeDuration)
        {
            elapsed += Time.deltaTime;
            float progress = elapsed / fadeDuration;
            float scale = Mathf.Lerp(1.5f, 0f, progress);
            transform.localScale = originalScale * scale;

            if (targetRenderer != null)
            {
                Color color = targetRenderer.material.color;
                color.a = 1f - progress;
                targetRenderer.material.color = color;
            }

            yield return null;
        }

        TargetManager.Instance.RemoveTarget(gameObject);
    }
```

```
}
```

Script 5: UIButtonHandler.cs

1. Right-click Scripts folder → Create → C# Script
2. Name: UIButtonHandler
3. Replace content with:

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class UIButtonHandler : MonoBehaviour, IPointerEnterHandler,
IPointerExitHandler
{
    [Header("Visual Feedback")]
    public Color normalColor = new Color(0.2f, 0.6f, 1f, 0.8f);
    public Color hoverColor = new Color(0.3f, 0.8f, 1f, 1f);
    public float scaleMultiplier = 1.1f;

    [Header("Audio")]
    public AudioClip hoverSound;
    public AudioClip clickSound;

    private Image buttonImage;
    private Vector3 originalScale;
    private AudioSource audioSource;

    void Start()
    {
        buttonImage = GetComponent<Image>();
        originalScale = transform.localScale;

        if (buttonImage != null)
            buttonImage.color = normalColor;

        audioSource = GetComponent<AudioSource>();
        if (audioSource == null && (hoverSound != null || clickSound != null))
        {
            audioSource = gameObject.AddComponent<AudioSource>();
            audioSource.playOnAwake = false;
            audioSource.spatialBlend = 0f;
        }

        Button button = GetComponent<Button>();
        if (button != null)
        {
            button.onClick.AddListener(OnButtonClick);
```

```
        }
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        if (buttonImage != null)
            buttonImage.color = hoverColor;

        transform.localScale = originalScale * scaleMultiplier;

        if (audioSource != null && hoverSound != null)
            audioSource.PlayOneShot(hoverSound);
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        if (buttonImage != null)
            buttonImage.color = normalColor;

        transform.localScale = originalScale;
    }

    void OnButtonClick()
    {
        if (audioSource != null && clickSound != null)
            audioSource.PlayOneShot(clickSound);
    }
}
```

PART 5: CREATE LAYERS

1. Edit → Project Settings → Tags and Layers
2. Click Layers dropdown
3. Find empty layer (User Layer 6)
4. Click and type: Target
5. Press Enter
6. Close Project Settings

PART 6: CREATE MATERIALS

Material 1: TargetMaterial

1. Right-click Materials folder → Create → Material
2. Name: TargetMaterial
3. Select it, in Inspector:
    - Surface Type: Opaque

- Base Map Color: Orange/Red (R: 255, G: 100, B: 0, A: 255)
- Metallic: 0
- Smoothness: 0.5
- Emission: Check to enable
- Emission Color: Black (will change when hit)

Material 2: GunMaterial

1. Right-click Materials folder → Create → Material
2. Name: GunMaterial
3. In Inspector:
    - Base Map Color: Dark Gray (R: 40, G: 40, B: 40, A: 255)
    - Metallic: 0.8
    - Smoothness: 0.6

Material 3: HitMarkerMaterial

1. Right-click Materials folder → Create → Material
2. Name: HitMarkerMaterial
3. In Inspector:
    - Surface Type: Transparent
    - Blending Mode: Alpha
    - Base Map Color: Red (R: 255, G: 0, B: 0, A: 150)

PART 7: CREATE TARGET PREFAB

Step 1: Create Target GameObject

1. Right-click Hierarchy → 3D Object → Sphere
2. Name: Target
3. In Inspector, set Transform:
    - Position: (0, 0, 0)
    - Rotation: (0, 0, 0)
    - Scale: (0.3, 0.3, 0.3)

Step 2: Apply Material

1. Select Target
2. In Project window, find TargetMaterial
3. Drag TargetMaterial onto Target in Scene view

Step 3: Configure Target

1. Select Target
2. At top of Inspector, set Layer: Target
3. Verify Sphere Collider component exists (should be automatic)
4. Click Add Component
5. Type: Target (script)

6. Select it

Step 4: Create Prefab

1. Drag Target from Hierarchy into Prefabs folder
2. Target in Hierarchy should turn blue
3. Right-click Target in Hierarchy → Delete

PART 8: CREATE GREEN FIREWORKS PREFAB

Step 1: Create Particle System

1. Right-click Hierarchy → Effects → Particle System
2. Name: GreenFireworks
3. Position at (0, 0, 0)

Step 2: Configure Main Module

Select GreenFireworks, in Inspector:

Main Module:

- Duration: 1.0
- Looping: OFF (unchecked)
- Start Lifetime: Random Between Two Constants (0.5, 1.0)
- Start Speed: Random Between Two Constants (3, 8)
- Start Size: Random Between Two Constants (0.05, 0.15)
- Start Color: Green (R: 0, G: 255, B: 0, A: 255)
- Gravity Modifier: -0.5
- Max Particles: 100
- Play On Awake: OFF

Step 3: Configure Emission Module

- Rate over Time: 0
- Bursts: Click + button twice
    - Burst 1: Time 0, Count 50
    - Burst 2: Time 0.1, Count 30

Step 4: Configure Shape Module

- Shape: Sphere
- Radius: 0.2
- Emit from: Volume

Step 5: Configure Color over Lifetime

1. Check box to enable Color over Lifetime

2. Click on Color gradient bar
3. In Gradient Editor:
    - Left key (0%): Green (R: 0, G: 255, B: 0, A: 255)
    - Middle key (50%): Yellow-Green (R: 150, G: 255, B: 0, A: 255)
    - Right key (100%): White (R: 255, G: 255, B: 255, A: 0)
4. Close gradient editor

Step 6: Configure Size over Lifetime

1. Check box to enable Size over Lifetime
2. Click Size dropdown
3. Select curve or "Random Between Two Constants"
4. Set to go from 1 to 0

Step 7: Configure Renderer

Renderer Module:

- Render Mode: Billboard
- Material: Default-Particle

Step 8: Create Prefab

1. Drag GreenFireworks from Hierarchy to Prefabs folder
2. Right-click GreenFireworks in Hierarchy → Delete

PART 9: CREATE HIT MARKER PREFAB

Step 1: Create Quad

1. Right-click Hierarchy → 3D Object → Quad
2. Name: HitMarker
3. Set Transform:
    - Position: (0, 0, 0)
    - Rotation: (0, 0, 0)
    - Scale: (0.1, 0.1, 0.1)

Step 2: Apply Material

1. Select HitMarker
2. In Inspector, find Mesh Renderer → Materials
3. Drag HitMarkerMaterial into Element 0 slot

Step 3: Create Prefab

1. Drag HitMarker from Hierarchy to Prefabs folder
2. Right-click HitMarker in Hierarchy → Delete

PART 10: ASSIGN EFFECTS TO TARGET PREFAB

This is critical - don't skip!

1. In Prefabs folder, double-click Target prefab
2. This opens prefab editing mode
3. Select Target (root object)
4. In Inspector, find Target (Script) component
5. Hit Effect Prefab: Drag GreenFireworks prefab from Prefabs folder here
6. Burst Sound: (Optional - add audio clip if you have one)
7. File → Save (Ctrl+S)
8. Click < (back arrow) in Hierarchy breadcrumb to exit prefab mode

4 More Advancements for Future Work

- Defined structure of a new laser. One idea is a popping laser that reaches out, hits target and come back.
- Work on target and laser skins
- Audio responses to hits and misses
- Multiplayer Mode settings to be created
- Track player status over time which are saved (accuracy, reaction speed)