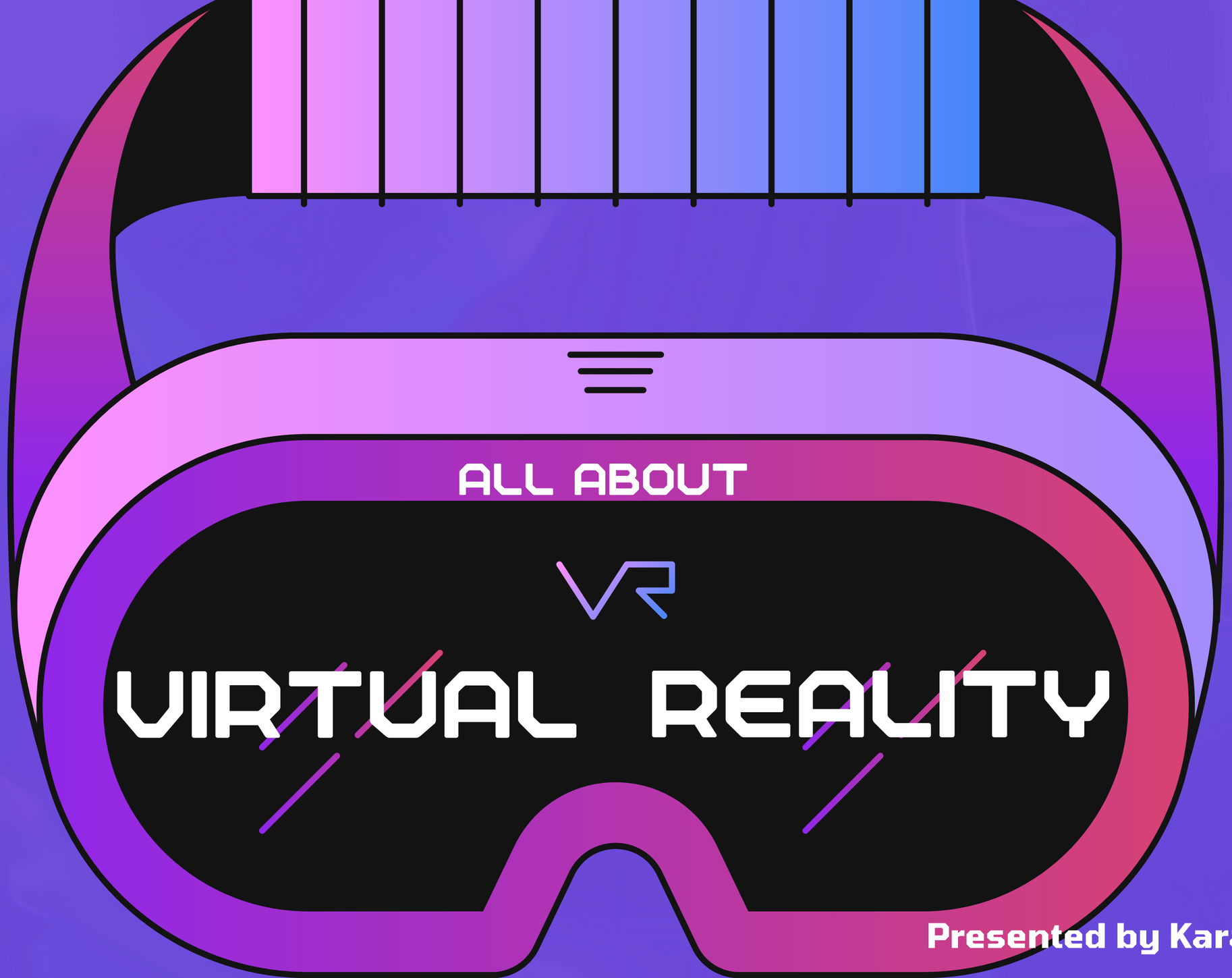


Presented by Karan Grover

MY VR DEVELOPMENT JOURNEY

Welcome to my Virtual Reality Development Journey. This presentation will walk you through four innovative VR projects that demonstrate my skills in Unity development, immersive interaction design, and creative challenges.

Each project showcases a unique aspect of VR, from space exploration to survival horror, and movement mechanics to mixed reality gaming. Let's dive in and explore these exciting experiences together!

LET'S GET STARTED

AGENDA

Project 1: Space Scraper

Project 2: Zombie Survival

Project 3: VR Practice Valley

Project 4: Mixed Reality Archery

TOOLS & TECHNOLOGIES

Unity with XR Interaction Toolkit

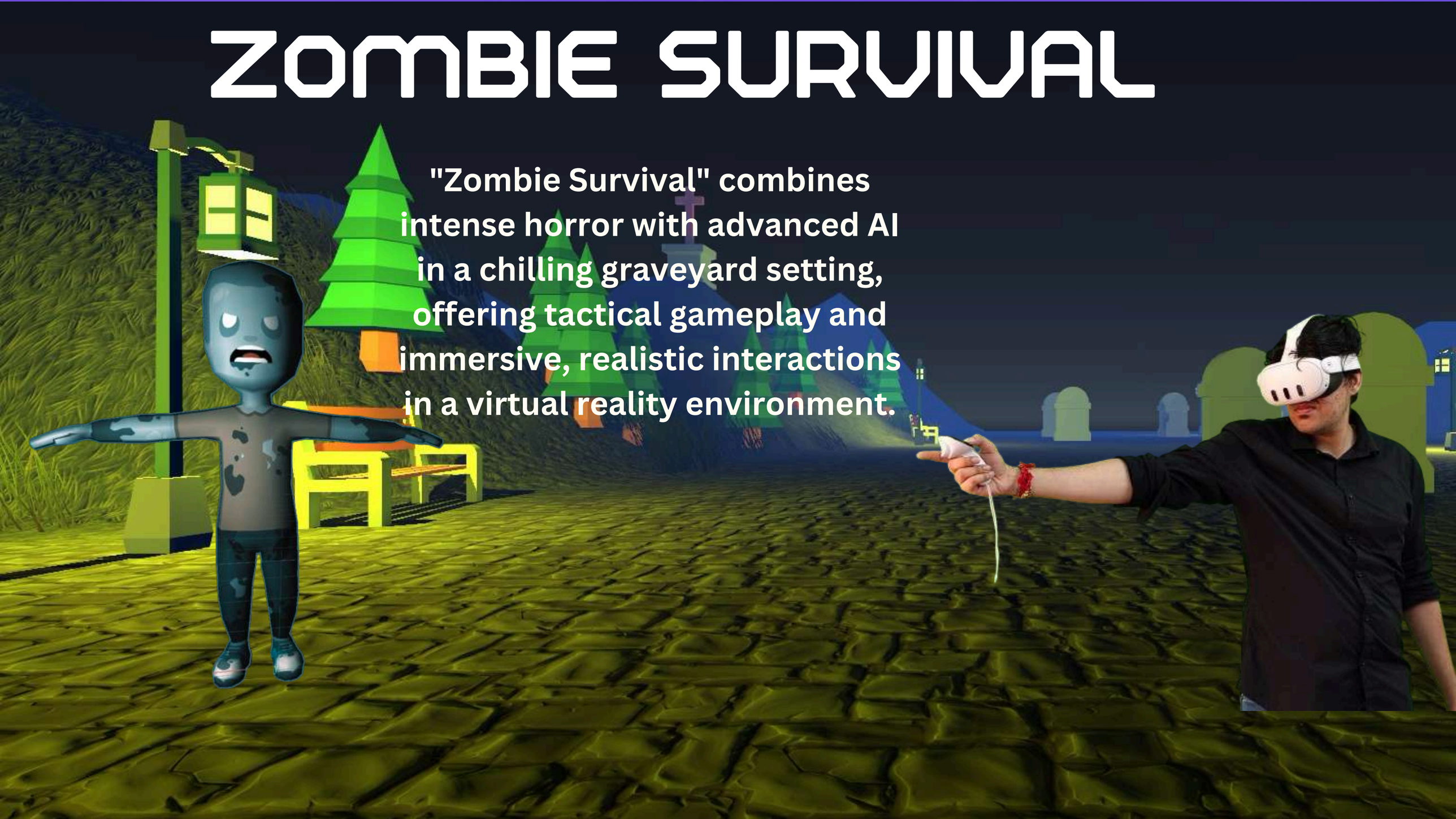
Meta Quest 3

C# Scripting


ZOMBIE
Survival

ZOMBIE SURVIVAL

"Zombie Survival" combines intense horror with advanced AI in a chilling graveyard setting, offering tactical gameplay and immersive, realistic interactions in a virtual reality environment.



ZOMBIE BEHAVIOR SCRIPT

Script Purpose:
Manages zombie behavior in "Zombie Survival VR", ensuring realistic interactions and challenges for players.

Initialization (Start Method):

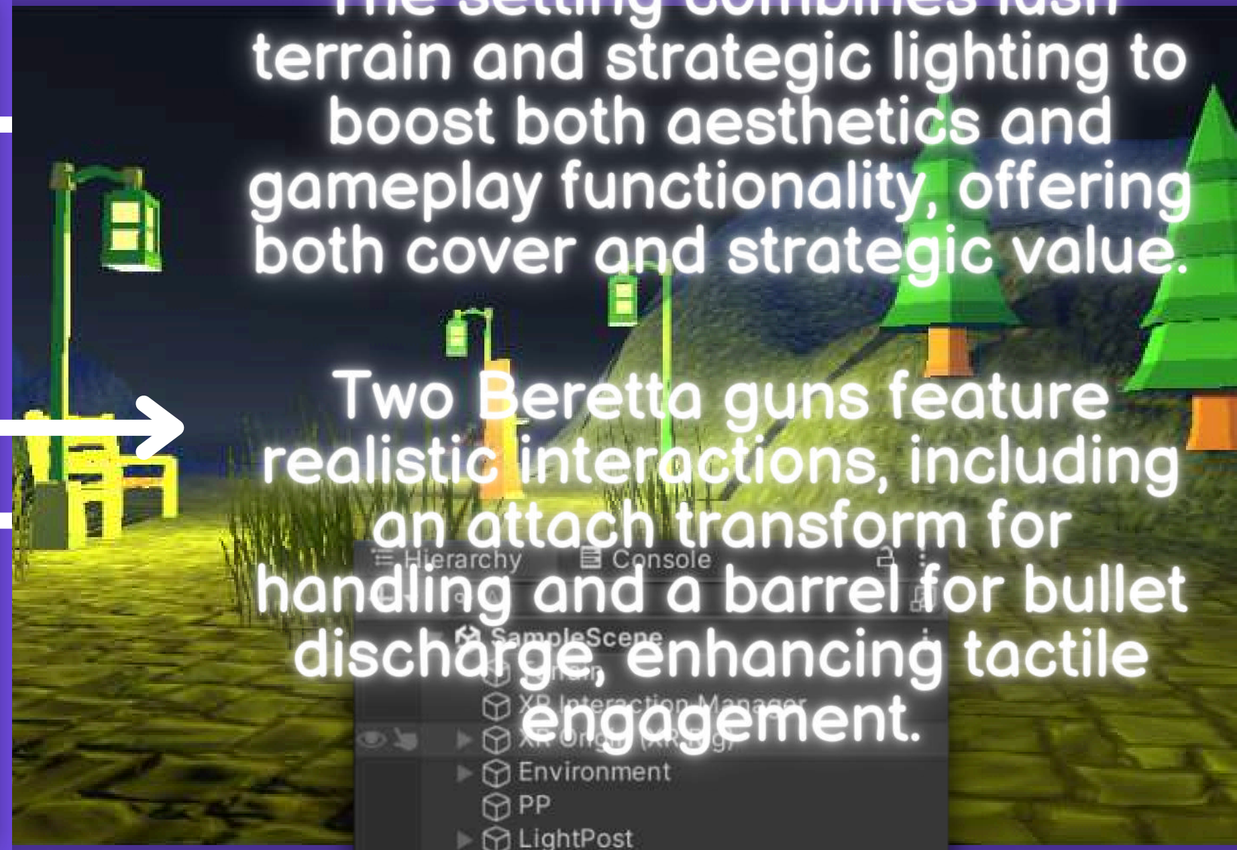
- Retrieves child Rigidbody components for ragdoll physics.
- Sets up NavMeshAgent for AI pathfinding.
- Auto-detects and assigns the player's camera (VR rig) as the target if not manually assigned, ensuring zombies always have a current objective.

Continuous Tracking and Death Mechanics

Continuous Tracking (Update Method):
Continuously updates target position to the player's location, enabling persistent pursuit.
Resets the game scene if a zombie reaches within 1.5 meters of the player, increasing game stakes.

Death Handling (Death Method):
Activates ragdoll effects for realistic death animations.
Disables movement and animation, stops further navigation.
Plays an audio clip for immersive death sounds.
Schedules destruction of the zombie object to optimize resource usage.

The setting combines lush terrain and strategic lighting to boost both aesthetics and gameplay functionality, offering both cover and strategic value.

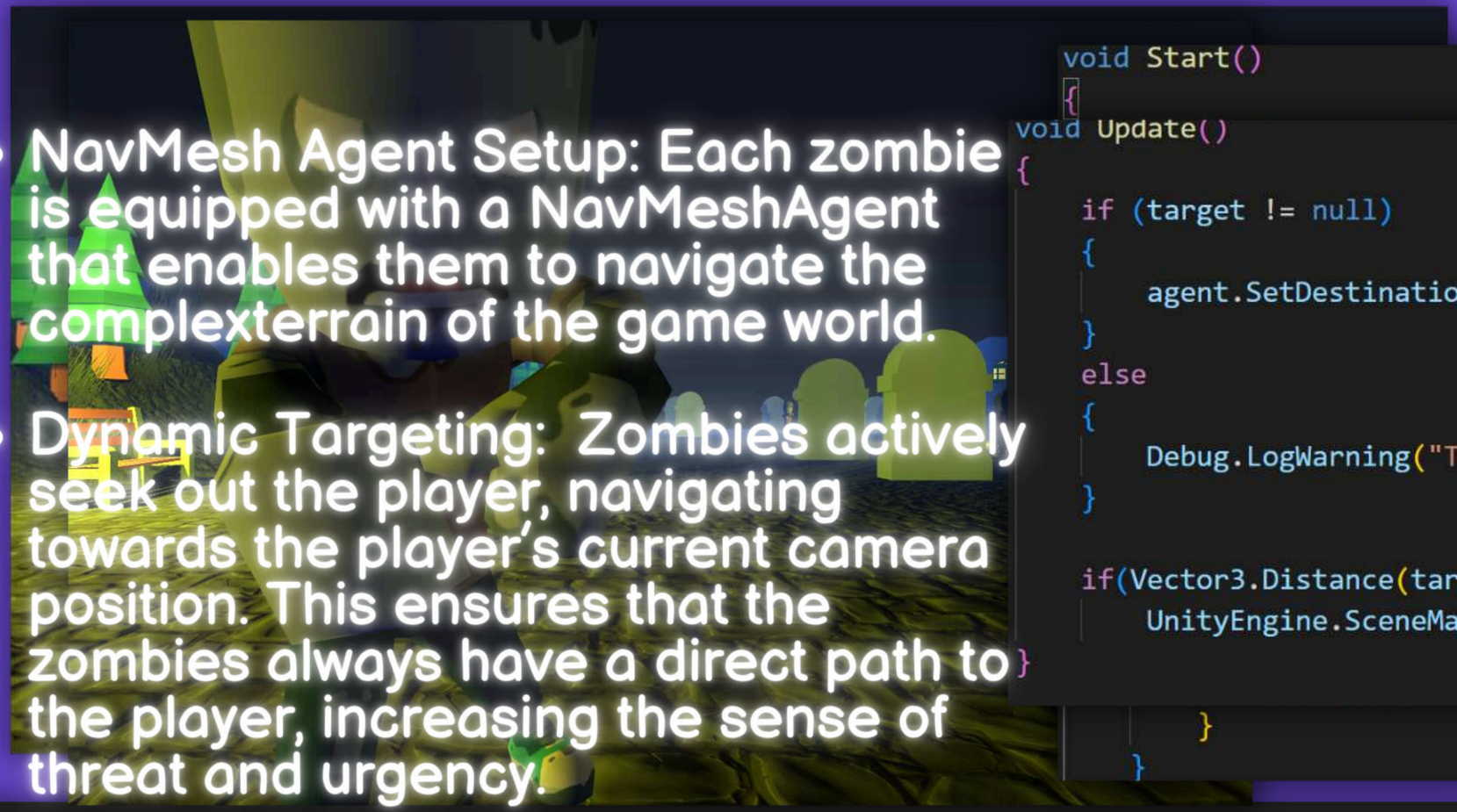


Two Beretta guns feature realistic interactions, including an attach transform for handling and a barrel for bullet discharge, enhancing tactile engagement.

The blue area depicts the NavMesh for zombie AI pathfinding, defining movement capabilities across the terrain.

```
{
    GameObject spawnedBullet = Instantiate(bullet, barrel.posit
    spawnedBullet.GetComponent<Rigidbody>().velocity = speed *
    audioSource.PlayOneShot(audioClip);
    Destroy(spawnedBullet, 2);
}
```

Hierarchy
Console
SampleScene
XR Interaction Manager
Environment
PP
LightPost
Axe
trunkLong
Beretta
Beretta (2)
Beretta (1)
Zombie Limbs
Zombie Spawner
scary-ambience
audio_thrill
rocksTall



- NavMesh Agent Setup: Each zombie is equipped with a NavMeshAgent that enables them to navigate the complexterrain of the game world.
- Dynamic Targeting: Zombies actively seek out the player, navigating towards the player's current camera position. This ensures that the zombies always have a direct path to the player, increasing the sense of threat and urgency.

```
void Start()
{
void Update()
{
    if (target != null)
    {
        agent.SetDestination(target.position);
    }
    else
    {
        Debug.LogWarning("Target is not assigned.");
    }

    if(Vector3.Distance(target.position, transform.position) < 1.5f)
        UnityEngine.SceneManagement.SceneManager.LoadScene(0);
}
        }
    }
```

As zombies draw closer, the threat intensifies. If a zombie reaches within 1.5 meters of the player, it signifies an attack, leading to the game restarting.

```
Transform randomPoint = spawnPoints[Random.Range(0, spawnPoints.Length)];
Instantiate(spawnGameObject, randomPoint.position, randomPoint.rotation );
timer = 0;
```

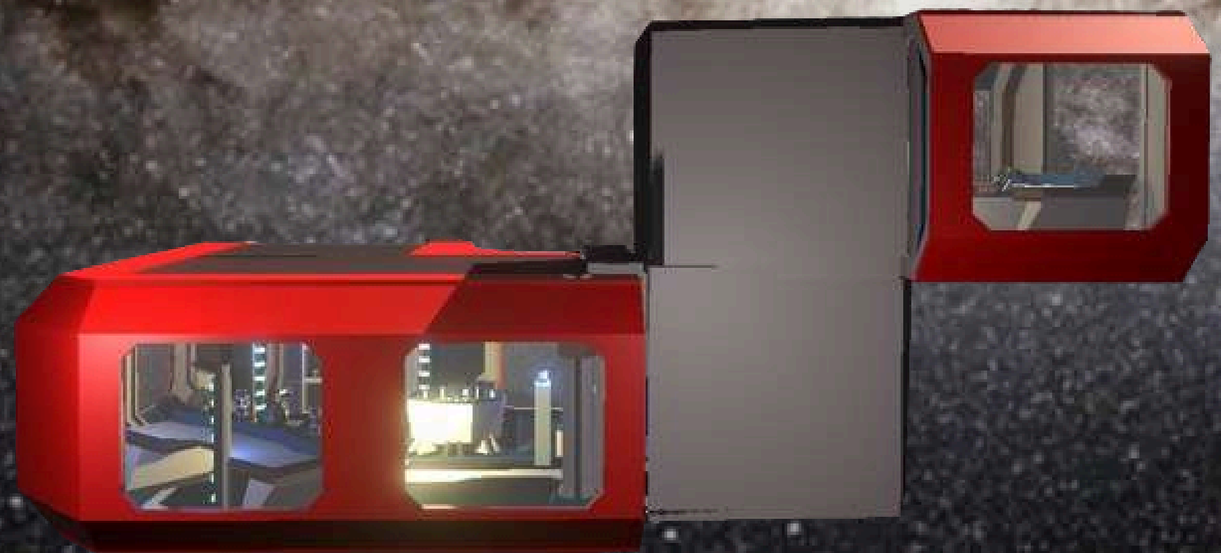
This close-up shot highlights the urgency and immediate danger posed by the approaching zombies.

Zombies are spawned at random positions around the player, utilizing a script that selects a random spawn point from a predefined list of locations.

-

STARSHIP

A SPACE ADVENTURE




DESTINATION PORTAL
CUPBOARD CONTAINING SPACE TRASH
WHEEL FOR DIRECTION
GRABABLE OBJECTS
THE CLIMBING LADDER
OPEN THE DOOR
VORTEX TRASH
GEAR -> FORWARD
THE CONTROL CENTRE

```
void Update()
{
    if (!surface.navMeshData)
        return;

    if(timer > spawnTime)
    {
        Vector3 randomCircle = Random.onUnitSphere;
        randomCircle.y = 0;

        Vector3 center = Camera.main.transform.position;
        center.y = 0;

        Vector3 randomPosition = center + randomCircle * Random.Range(minRange, maxRange);

        bool isOnNavMesh = NavMesh.SamplePosition(randomPosition, out NavMeshHit hit, 0.5f, NavMesh.AllAreas);

        if(isOnNavMesh)
        {
            GameObject spawned = Instantiate(holePrefab);
            spawned.transform.position = hit.position;

            timer = 0;
            Destroy(spawned, 6);
        }
    }
```

- NavMesh Building:
- Event Hook: The NavMesh is built when the scene model is loaded successfully.
- Update Loop: On each frame, if the timer exceeds the spawn time, a random position within the specified range is chosen. The script checks if this position is on the NavMesh.
- Spawning Logic:
- Random Positioning: Objects are instantiated at random valid positions within the NavMesh area.
- Object Lifetime: Spawned objects are automatically destroyed after 6 seconds to keep the scene clean and dynamic.

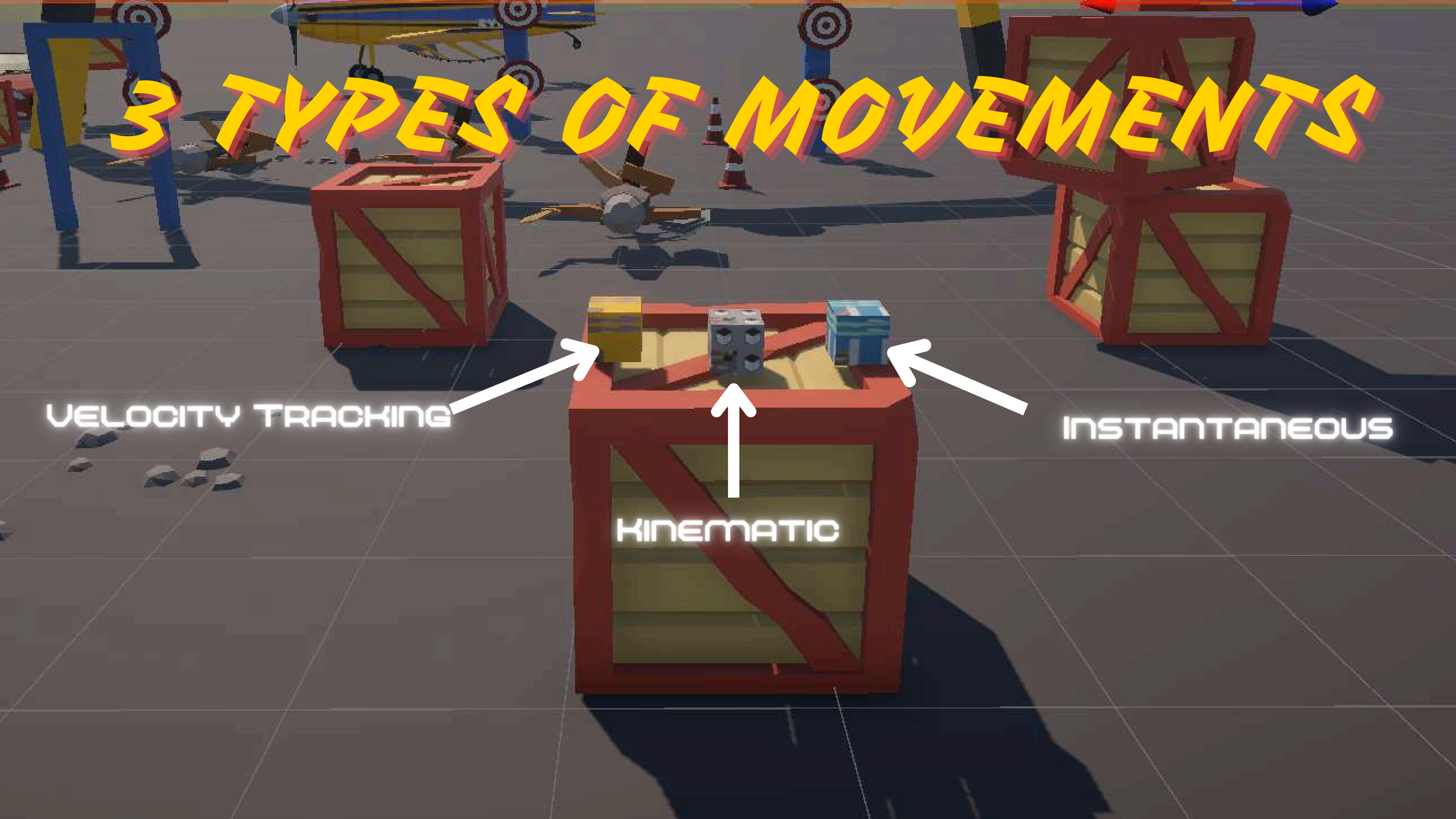
```
void Start()
{
    XRGrabInteractable grabInteractable = GetComponent<XRGrabInteractable>();
    grabInteractable.activated.AddListener(x => StartShoot());
    grabInteractable.deactivated.AddListener(x => StopShoot());
}
1 reference
public void StartShoot()
{
    particles.Play();
    rayActivate = true;
}

1 reference
public void StopShoot()
{
    particles.Stop(true, ParticleSystemStopBehavior.StopEmittingAndClear);
    rayActivate = false;
}
```

- XR Grab Interactable: The script begins by obtaining a reference to the XRGrabInteractable component, which is responsible for detecting when the laser gun is grabbed and released.
- Shooting Activation:
  - When the gun is activated (grabbed), the StartShoot() method is called.
  - This method triggers the particle system to simulate the laser beam and sets rayActivate to true, enabling the laser to interact with objects.
- Shooting Deactivation:
  - When the gun is deactivated (released), the StopShoot() method is called.
  - This method stops the particle system and sets rayActivate to false, turning off the laser.

```
public class ButtonPushOpenDoor : MonoBehaviour
{
    2 references
    public Animator animator;
    2 references
    public string boolName = "Open";
    // Start is called before the first frame update
    0 references
    void Start()
    {
        GetComponent<XRSimpleInteractable>().selectEntered.AddListener(x => ToggleDoorOpen());
    }
    1 reference
    public void ToggleDoorOpen()
    {
        bool isOpen = animator.GetBool(boolName);
        animator.SetBool(boolName, !isOpen);
    }
```

- Animator Reference:
- The script starts by defining a reference to the Animator component (public Animator animator;) that controls the door's animation, as well as a string variable boolName that represents the parameter controlling whether the door is open or closed.
- Event Listener Setup:
- In the Start() method, the script attaches a listener to the selectEntered event of the XRSimpleInteractable component. This event is triggered when the player interacts with the button.
- The AddListener method is used to run the ToggleDoorOpen() function whenever the button is pressed.
- ToggleDoorOpen Function:
  - When the button is pressed the ToggleDoorOpen() method is called.
  - This function checks the current state of the door (open or closed) using animator.GetBool(boolName).
  - It then toggles this state by setting the opposite value (!isOpen) with animator.SetBool(boolName, !isOpen), causing the door to either open or close depending on its previous state.


VR
PRACTICE VALLEY


3 TYPES OF MOVEMENTS
VELOCITY TRACKING
KINEMATIC
INSTANTANEOUS


3 TYPES OF WEAPONS
SHIELD
SWORD
GUN WITH BULLETS

Using Canvas, the dropdown offers "Snap Turn" for discrete, angle-based view changes, and "Continuous Turn" for smooth, ongoing rotation in VR.

These three are the 3 target dummy bots.

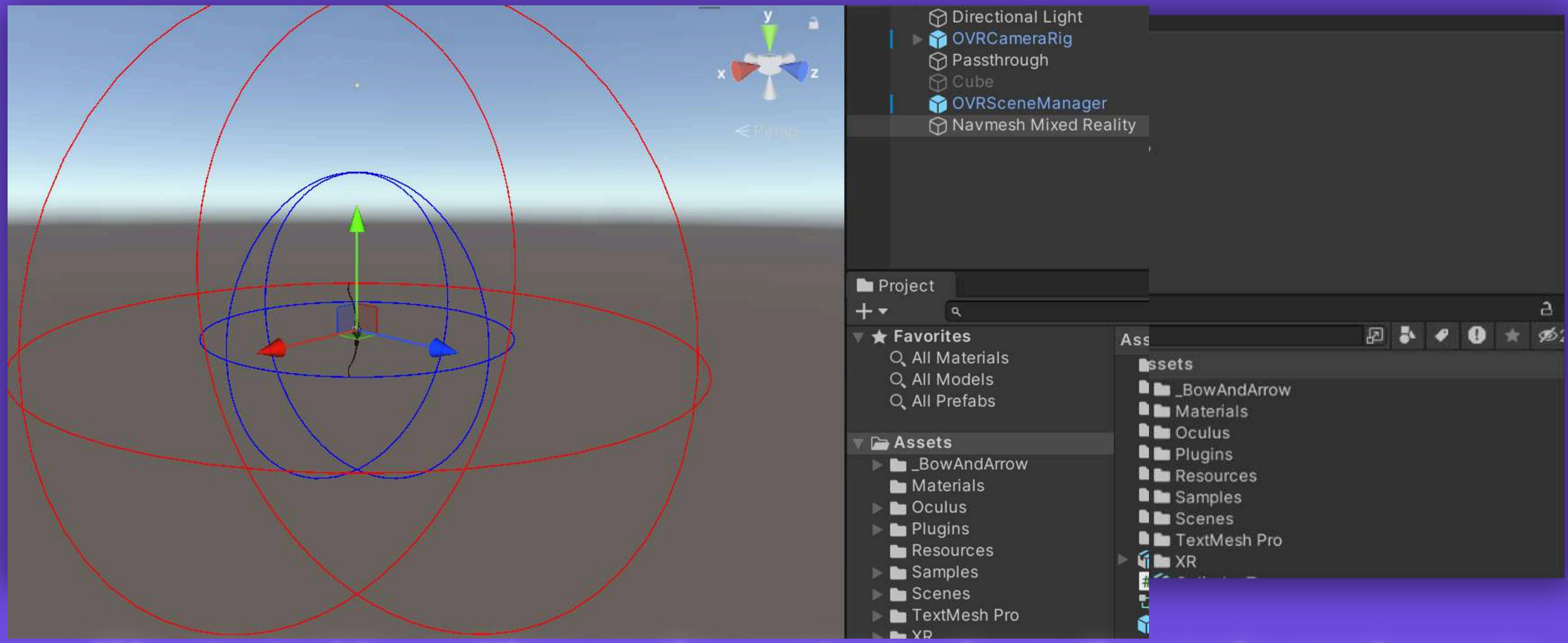
```
public class DummyTrigger : MonoBehaviour
{
    1 reference
    [SerializeField] private GameObject[] targetDummies;

    0 references
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Player"))
        {
            foreach(GameObject dummies in targetDummies)
            {
                dummies.GetComponent<TargetDummy>().ActivateDummy();
            }
        }
    }
}
```

[illegible] a trigger
[illegible] square
[illegible] bots, enabling
dynamic target practice for user
engagement and skill enhancement.

Mixed

reality

ARCHER



We started by setting up the project, where the main goal was to create a mixed reality environment using the Unity XR Interaction Toolkit and OpenXR. This setup allowed us to avoid dependency on Meta's SDK, making our application cross-platform. We configured the project for passthrough, enabling users to see their physical surroundings while interacting with virtual objects. We also explored how to classify different types of planes (e.g., floor, wall, table) to enhance interaction and ensure accurate behavior of virtual objects.

In the next phase, we integrated plane detection, which is crucial for mixed reality applications. We customized the appearance of detected planes, such as floors and walls, and used them for collision detection. This functionality allowed us to interact seamlessly with both the virtual and physical environments. We also highlighted the importance of using Unity's XR Interaction Toolkit for its flexibility and ease of porting to other devices.

```
public void Spawn()
{
    spawned = Instantiate(targetPrefab, spawnPoint.position, Quaternion.identity);

    Vector3 targetCamera = Camera.main.transform.position - spawned.transform.position;
    Vector3 lookCamera = Vector3.ProjectOnPlane(targetCamera, Vector3.up);
    spawned.transform.forward = lookCamera.normalized;

    Rigidbody rb = spawned.GetComponent<Rigidbody>();
    rb.velocity = Vector3.up * Mathf.Sqrt(2*height* - Physics.gravity.y);
}
```

Fire Target Spawning Script:

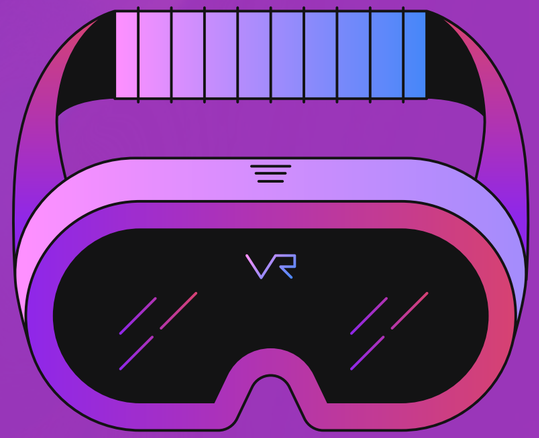
- Instantiation: The script starts by creating (instantiating) a target at a specific spawn point.
- Orientation: It then calculates the vector between the target and the camera, ensuring the target faces the player. This orientation is normalized and set to the target's forward direction.
- Launch Velocity: Finally, the target's Rigidbody component is accessed, and a vertical velocity is applied using a physics formula to achieve the desired height.

```
public void Damage(Vector3 hitPoint)
{
    TMPro.TextMeshPro spawnedText = Instantiate(scorePrefab, transform.position, transform.rotation)
    spawnedText.transform.Rotate(0, 180, 0);

    float hitDistance = Vector3.Distance(hitPoint, center.position);
    float hitPercent = (1 - (hitDistance / radius)) *100;
    int score = (int)hitPercent;
    spawnedText.text = score.ToString();

    Destroy(spawnedText.gameObject, 2);
}
```

- Score Text Display: When a target is hit, the script creates a score text object at the hit location, rotating it to face the player.
- Hit Calculation: The distance between the hit point and the target's center is calculated. The closer the hit is to the center, the higher the score. This score is then converted to a percentage.
- Score Update: The calculated score is displayed on the spawned text, which shows the score visually to the player.
- Cleanup: The score text object is automatically destroyed after 2 seconds to keep the scene clean.

```
void Update()
{
    if (!surface.navMeshData)
        return;

    if(timer > spawnTime)
    {
        Vector3 randomCircle = Random.onUnitSphere;
        randomCircle.y = 0;

        Vector3 center = Camera.main.transform.position;
        center.y = 0;

        Vector3 randomPosition = center + randomCircle * Random.Range(minRange, maxRange);

        bool isOnNavMesh = NavMesh.SamplePosition(randomPosition, out NavMeshHit hit, 0.5f, NavMesh.AllAreas);

        if(isOnNavMesh)
        {
            GameObject spawned = Instantiate(holePrefab);
            spawned.transform.position = hit.position;

            timer = 0;
            Destroy(spawned, 6);
        }
    }
```

- Purpose: This script is designed to spawn objects (like holes) randomly on a NavMesh surface, which is generated dynamically after a scene model is successfully loaded.
- Key Components:
- Scene Manager & NavMesh Surface: The script connects to the OVR Scene Manager and NavMesh Surface to manage when and where objects can spawn.
- Gizmos for Visual Range: Blue and red wire spheres are drawn in the Unity editor to visualize the minimum and maximum spawning ranges relative to the player's position.



THANK YOU

Presented by Karan Grover