# Virtual Reality and its Applications

# Mini Project

**Name**: S Madhumedha

**SRN**: PES1PG24CS036

---

**Narnia's Escape: A Unity Puzzle Adventure**

This project is a first-person puzzle game created in Unity. Inspired by fantasy stories, the player must navigate from a mystical forest into a primitive escape room and solve a series of challenges to find their way home.

---

**How to Run**

- **Scene Names**: The '***`SampleScene`***' is the Forest Scene. The '***`RoomScene`***' is the Escape Room Scene.
- **File Structure**: The scenes can be found in the following directory: **`VR_MiniProject > VR_Project > Assets > Scenes.`**

---

**Gameplay Flow**

The player's journey is a linear progression through two distinct scenes with two core puzzles.

1. **The Mystical Forest**: The player spawns in an atmospheric forest and must explore to find a large, custom-built tree with a hollowed-out tunnel.
2. **The Teleportation**: Upon entering the tree tunnel, the player is instantly "teleported" to the second scene.
3. **The Escape Room**: The player arrives in a primitive room containing a locked door and a series of buttons.
4. **Puzzle 1: The Button Sequence**: The player must press the buttons in the correct sequence. Successful completion reveals a glowing crystal ball.
5. **Puzzle 2: The Crystal Throw**: The player must pick up and throw the crystal ball at the locked door.
6. **Escape**: When the crystal hits the door with sufficient force, the door animates open, and the game is won.

---

**Features**

- **First-Person Controller**: Full player movement with mouse-look camera controls. The controller uses raycasting to keep the player grounded on uneven terrain.
- **Scene Management**: A seamless transition system teleports the player between the Forest and Room scenes.
- **Interactive Puzzles**: The game features a logic-based button sequence puzzle and a physics-based throwing puzzle.

- **Physics-Based Interaction**: The throwable crystal ball uses Unity's physics engine, with its Rigidbody.isKinematic state toggled for grabbing and throwing.
- **Dynamic Animations & Effects**: The final door smoothly animates open using Coroutines, and key objects like the crystal have custom lighting effects.
- **Custom Level Design**: The scenes are built with a mix of assets from the Unity Asset Store and custom geometry created with ProBuilder.

---

**Technical Deep Dive**

This section details the core scripts and logic that power the game.

**Player Spawning and Movement**

- **PlayerSpawnManager.cs**: This script handles placing the player correctly in both scenes. On scene load, it casts a **Raycast** downwards from a predefined spawnPosition to find the ground. It then places the player at the ray's hit point plus a playerHeight offset to ensure they start perfectly on the ground.
- **SimpleMovement.cs**: Manages all player movement and camera look.
    - **Movement**: Translates keyboard input (Input.GetAxis) into a movement vector based on the camera's orientation.
    - **Grounding**: To prevent falling through the floor, a continuous **Raycast** is cast downwards to keep the player snapped to the ground's Y-position.
    - **Camera**: Handles mouse look, clamping the vertical (pitch) rotation between -90 and 90 degrees to prevent flipping.

**Puzzles and Interaction**

- **Puzzle 1: Button Sequence (EscapeRoomManager.cs)**
    - Each button has a SimpleInteraction.cs script that detects clicks via OnMouseDown().
    - The click calls a method in the central EscapeRoomManager, which checks if the pressed button matches the sequence in a correctOrder array.
    - If the sequence is completed, the CompletePuzzle1() method is called, which activates the crystal ball GameObject and makes it glow.
- **Puzzle 2: Crystal Ball (ThrowableBall.cs)**
    - **Grabbing**: A raycast from the camera detects a click on the ball, calling GrabBall(). The ball's Rigidbody.isKinematic is set to true, making it immune to physics so it can be controlled by the script for a "hover" effect. A LineRenderer is enabled to provide a visual aiming guide to the target door.
    - **Throwing**: When the mouse is released, ThrowBall() is called. isKinematic is set to false, re-enabling physics. A direction vector is calculated to the center of the target door, a slight upward arc is added, and this force is applied to the Rigidbody's velocity.

**Scene and Object Mechanics**

- **Scene Transition (SceneTransition.cs)**: A SphereCollider on an object in the forest acts as a trigger. When the player enters it, OnTriggerEnter calls SceneManager.LoadScene to load the escape room.

- **Door Animation (DoorController.cs)**:
    - When the OnCollisionEnter method in ThrowableBall.cs detects a hit on the door, it calls OnBallHit() in this script.
    - If the impact force is sufficient, it starts an OpenDoor() **Coroutine**.
    - The coroutine uses Vector3.Lerp or Quaternion.Lerp to smoothly animate the door's position/rotation over time, rather than having it snap open instantly.

**Development & Debugging Practices**

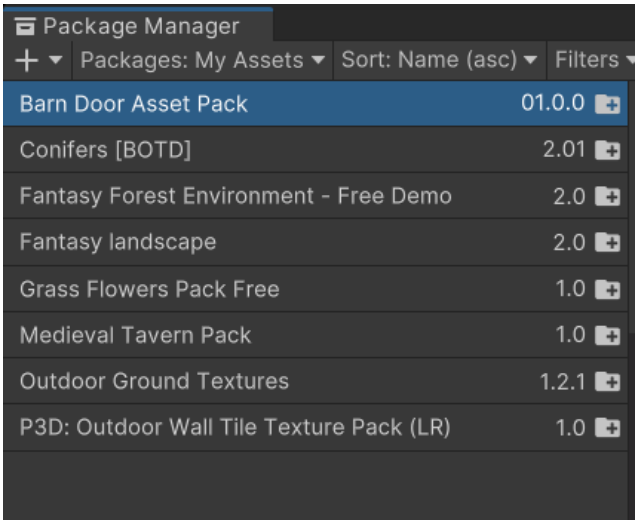
- **OnDrawGizmosSelected()**: This method is used across several scripts (ThrowableBall, DoorController, SceneTransition) to draw visual aids in the Unity editor. It draws wireframes for trigger radiuses and trajectory lines, which helps with debugging and setup without impacting runtime performance.

---

**Asset Credits**

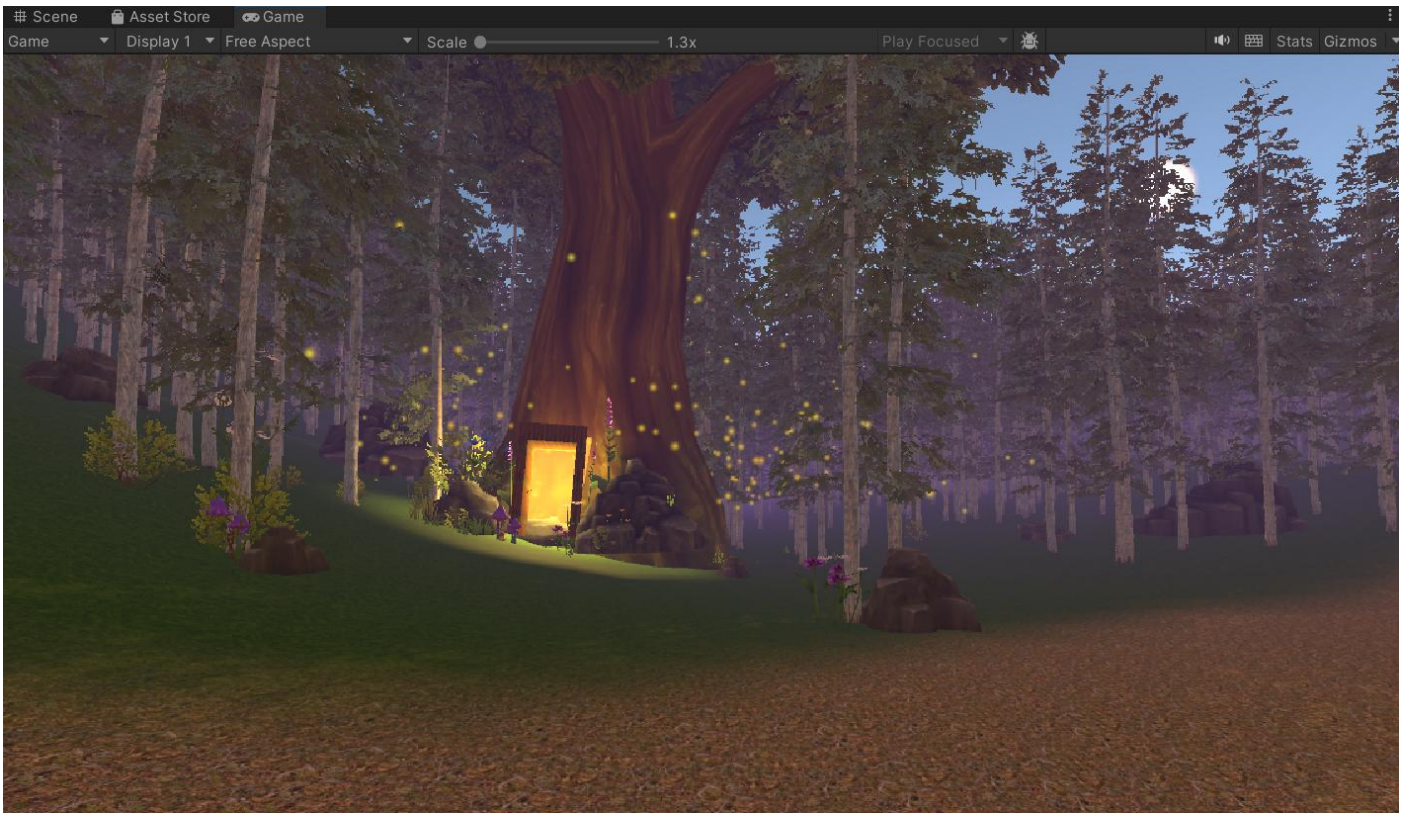
- **3D Models & Textures**: The majority of the environmental assets (trees, rocks, textures) were sourced from the Unity Asset Store.

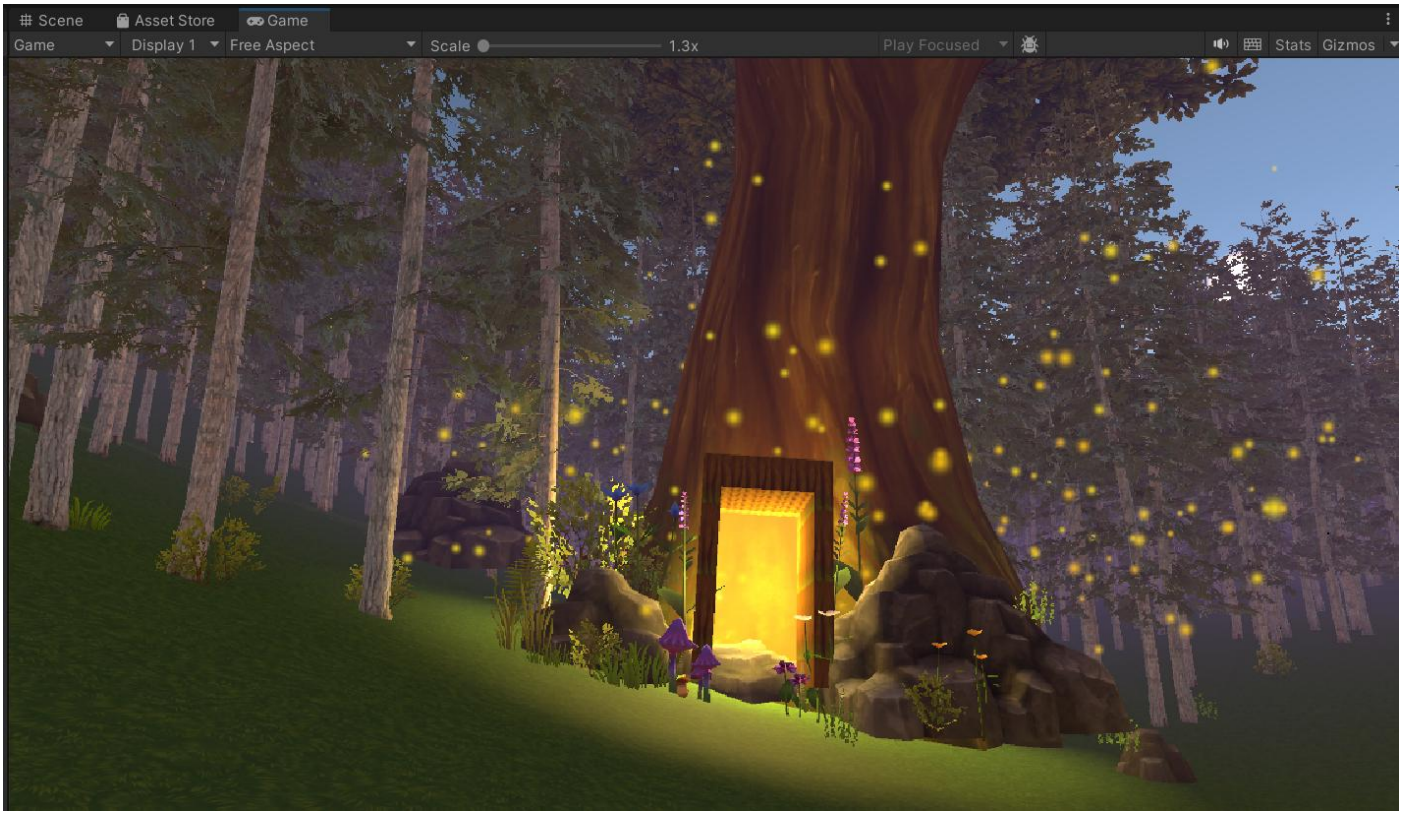
  **Asset Packages Used**:




- **Custom Geometry**: The tunnel in the main tree was created using Unity's ProBuilder package.

# Snapshots of the Forest Scene



# Snapshots of the Room Scene