# Lab Exercise: Building a simple interactive experience using Blender or Unity.

## Option 01

**Create a Simple Interactive Virtual Environment in Blender**

**Objective:**  
Students will learn to use Blender to create a simple virtual environment that incorporates interactive elements suitable for AI testing or immersive gaming. The exercise will focus on designing a basic virtual room and integrating simple AI-driven interactions, all while considering limited computational resources.

**Part 1: Setting Up the Virtual Environment (45 minutes)**

1. **Create a Room:**
   * Open Blender and create a simple enclosed room. The room should include walls, a floor, and a ceiling.
   * Use basic shapes like cubes and planes to create the room structure.
2. **Add Objects:**
   * Add a few simple objects to the room (e.g., a table, a chair, a lamp). Use primitive shapes to keep computational demands low.
   * Use Blender's Material Editor to apply basic colors to each object. Avoid high-resolution textures to save processing power.

**Part 2: Adding Simple Interactions (45 minutes)**

1. **Create Interactive Elements:**
   * Add an object that can be interacted with, such as a "magic cube" that changes color when clicked.
   * Use Blender’s built-in scripting (use simple Python scripts) to make the cube change color when it is selected. You can use Blender’s "Logic Editor" to set up these interactions.
2. **Animate an Object:**
   * Create a simple animation, like a door opening or the "magic cube" floating. Use Blender’s timeline to create keyframes for movement.
   * Keep the animation short and simple to ensure it runs smoothly on limited hardware.

**Part 3: Integrating AI Concepts (30 minutes)**

1. **Virtual NPC Interaction:**
   * Add a simple NPC (non-player character) in the form of a 3D figure (use low-poly models).
   * Use Blender's basic animation tools to make the NPC wave or move towards the user’s perspective when a specific object is interacted with.
2. **Simulate Simple AI Logic:**
   * The NPC should move when the "magic cube" changes color. Write a basic Python script in Blender that triggers the NPC's movement when this event occurs.
   * Explain that this simple interaction mimics how AI reacts to certain inputs in immersive environments.

**Part 4: Exporting and Testing (30 minutes)**

1. **Test the Environment:**
   * Have students test their virtual environments by interacting with the elements they created.
   * Observe the NPC movement and the color-changing cube to verify that everything functions as intended.
2. **Export for Simple VR Exploration:**
   * If time permits, guide students on how to export their scene for basic VR exploration using tools like a smartphone-based VR viewer or desktop VR emulators, ensuring that computational requirements remain minimal.

**Reflection (15 minutes)**

* Students will write a short reflective journal entry on their experience creating the virtual environment, the challenges they faced, and how these skills could be used in developing AI-driven VR or AR applications.

**Deliverables:**

* A Blender project file (.blend) of the interactive room.
* A reflective journal entry discussing the exercise and its implications for AI in virtual environments.

Option 02

**Lab Exercise: Create a Simple Game in Blender**

**Objective:**  
Students will learn how to create a basic game using Blender, focusing on game mechanics like object interaction and simple AI-driven movement. The exercise will teach them how to build, script, and test a simple, playable game environment with limited hardware requirements.

**Part 1: Setting Up the Game Environment (45 minutes)**

1. **Create the Game Space:**
   * Open Blender and create a simple game space (e.g., a maze or an open area with several obstacles).
   * Use basic 3D shapes such as cubes, spheres, and planes to create walls, pathways, and obstacles to keep the model lightweight.
2. **Add Game Elements:**
   * Place several collectible items throughout the game environment. Use simple models like coins or colored spheres.
   * Add a start and end point, representing the beginning and goal of the game (e.g., a glowing platform for the goal).

**Part 2: Adding Game Mechanics (45 minutes)**

1. **Player Character Setup:**
   * Add a simple 3D model to serve as the player character (e.g., a small cube or sphere). Keep it low-poly to minimize computation.
   * Allow students to add basic materials to the player model to distinguish it in the game.
2. **Character Movement:**
   * Use Blender’s built-in logic bricks or simple Python scripting to allow player control. The character should be able to move forward, backward, left, and right using the keyboard arrows or WASD keys.
   * Explain the concept of input handling and how basic movement controls are fundamental in game development.

**Part 3: Creating Game Objectives (45 minutes)**

1. **Collectibles:**
   * Program the collectible items so that when the player character collides with them, they disappear, and a score is updated.
   * Use Blender’s game engine logic bricks to manage these interactions, which helps students understand game states and simple event-driven programming.
2. **Goal and End Condition:**
   * Add a win condition to the game by scripting the player character’s interaction with the goal platform.
   * Once the player reaches the goal after collecting all items, trigger a message like "You Win!" to appear on the screen.

**Part 4: Adding AI Elements (30 minutes)**

1. **Enemy or Moving Obstacle:**
   * Add a basic enemy character or moving obstacle that follows a simple path or patrols the environment.
   * Use Blender’s animation system or simple Python logic to create a looping movement pattern. Explain that the enemy should "guard" an area, introducing the concept of AI in game design.
2. **Game Over Mechanic:**
   * Script a "Game Over" scenario where if the player character collides with the enemy or moving obstacle, the game ends.
   * Use Blender’s collision detection logic to manage this condition.

**Part 5: Testing and Enhancing the Game (30 minutes)**

1. **Testing Gameplay:**
   * Students will test their games to make sure all mechanics work as intended. They should verify player movement, collectibles, the enemy, and win/lose conditions.
2. **Game Enhancement:**
   * Allow students time to add simple enhancements like changing the colors of obstacles, adjusting enemy speed, or adding simple background music to their game.

**Reflection (15 minutes)**

* Students will write a brief reflection about their game development experience, the challenges they encountered, and how they approached designing game elements such as collectibles and enemies.

**Deliverables:**

* A Blender project file (.blend) containing the complete game.
* A short reflection journal discussing the game creation process and the learned principles of game mechanics.

**Alternative Lab Plan** If using Blender versions 2.8 and above:

* **Create a Virtual Environment with Interactive Elements**: Use Blender to create the models, characters, and environment for the game. Students can script basic animations and interactions to create an interactive scene.
* **Exporting to a Game Engine**: After creating the assets, students can export them to Unity or Godot, where they can use the game engine's tools to implement character control, collisions, and game logic.

## Option 03

**Lab Exercise: Create a Simple 3D Adventure Game in Unity**

**Objective:**  
Students will use Unity's free version to create a simple adventure game. This exercise aims to teach basic Unity skills, scripting interactions, and introducing AI behaviors. By the end, students will have built a playable game environment that demonstrates character movement, interaction with objects, and basic AI logic.

**Part 1: Setting Up the Game Environment in Unity (45 minutes)**

1. **Create a Simple Game Scene:**
   * Open Unity and create a new 3D project.
   * Design a small game environment (e.g., an outdoor scene or maze) using Unity’s built-in terrain tools and GameObjects like cubes, spheres, and planes.
   * Add visual variety by applying different textures to the environment. Keep it simple to ensure it runs smoothly on limited hardware.
2. **Add Player Character:**
   * Import or create a simple 3D model to represent the player character (e.g., a low-poly figure).
   * Use Unity's Character Controller component to give the player character movement functionality.
   * Implement basic player movement using arrow keys or WASD keys, allowing students to get familiar with Unity's physics and input systems.

**Part 2: Adding Game Mechanics (45 minutes)**

1. **Create Collectibles:**
   * Add several collectible objects (e.g., coins or crystals) throughout the scene.
   * Write a C# script to allow the player to "collect" these items on collision. When the player character touches a collectible, it should disappear, and a score should increase.
2. **Display the Player’s Score:**
   * Use Unity's UI tools to create a simple score counter.
   * Update the score in real-time as the player collects items. This teaches students how to work with Unity's user interface components and link game events to UI updates.

**Part 3: Adding Basic AI Behavior (45 minutes)**

1. **Enemy Patrol:**
   * Create a simple enemy character (e.g., a cube or capsule) that patrols back and forth within the game area.
   * Use Unity's NavMesh system to set up a navigation area, and write a script to make the enemy patrol between waypoints.
   * Introduce the concept of AI navigation by showing how the enemy can follow a predefined path, giving students a basic understanding of AI pathfinding.
2. **Enemy Detection and Player Interaction:**
   * Script the enemy so that when it detects the player (using a simple distance-based detection), it starts chasing them.
   * If the enemy catches the player, display a "Game Over" message. This will teach students about collision detection and condition-based game logic.

**Part 4: Enhancing the Adventure Game (30 minutes)**

1. **Adding Win Condition:**
   * Add a "goal" to the scene, such as a treasure chest or a glowing portal that the player must reach after collecting all items.
   * Write a script that checks if all collectibles have been gathered, and when the player reaches the goal, display a "You Win!" message. This reinforces concepts of conditional checks and end-game logic.
2. **Optional Enhancements:**
   * Allow students some creative freedom by adding a personal touch to their game. They could add sound effects when collecting items, enemy growls, or even background music.
   * Students can also adjust enemy difficulty by tweaking the patrol speed or detection range, providing a simple introduction to playtesting and balancing.

**Part 5: Testing and Reflection (15 minutes)**

1. **Testing the Game:**
   * Students should playtest their game to verify that all components work as intended. They should test player movement, item collection, enemy detection, and the win/lose conditions.
2. **Reflection:**
   * Write a reflective journal entry about the game development process. Include what worked well, what challenges were faced, and how the game could be improved or expanded in the future.

**Deliverables:**

* A Unity project folder containing the complete game.
* A reflective journal entry describing the development process and lessons learned.

**Learning Outcomes:**

* **Familiarity with Unity Editor**: Students will learn to navigate Unity’s interface, create GameObjects, and use basic tools like Terrain and Lighting.
* **Understanding Game Logic**: Students will understand how to implement basic game mechanics such as player movement, item collection, and scoring.
* **AI Pathfinding**: Introduction to AI behavior using Unity's NavMesh and scripting simple enemy movement.
* **Game Design and Iteration**: Students will explore game design by adding their creative elements, testing, and iterating on the gameplay to improve it.