Mini Project 2

Fall 2025

CSE-411 Intro to Game Development

Submitted by: **Noor-ul-Huda**

Registration No.: **22PWCSE2117**

Class Section: **B**

Student Signature: Noor

Submitted to:

Engr. Abdullah Hamid

January 17, 2025

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Mini Project 2

Project Title: Operation Extraction

Objective of the Project

The objective of this mini project is to design and implement a **third-person stealth game** using **Unity**, where the player acts as a **secret agent** tasked with infiltrating a secure area, collecting a **Data Briefcase**, and safely reaching an **Extraction Helipad** while avoiding enemy guards.

The project focuses on:

- Character animation control without blend trees
- NavMesh-based enemy AI
- Scene management and UI systems
- Game logic for win and loss conditions

Game Overview

In **Operation Extraction**, the player navigates through a maze-like environment representing a secure facility. The player must avoid detection by guards who patrol predefined paths. If the player is detected, guards will chase them. The mission is completed only if the player successfully retrieves the briefcase and reaches the helipad without being caught.

Tools and Technologies Used

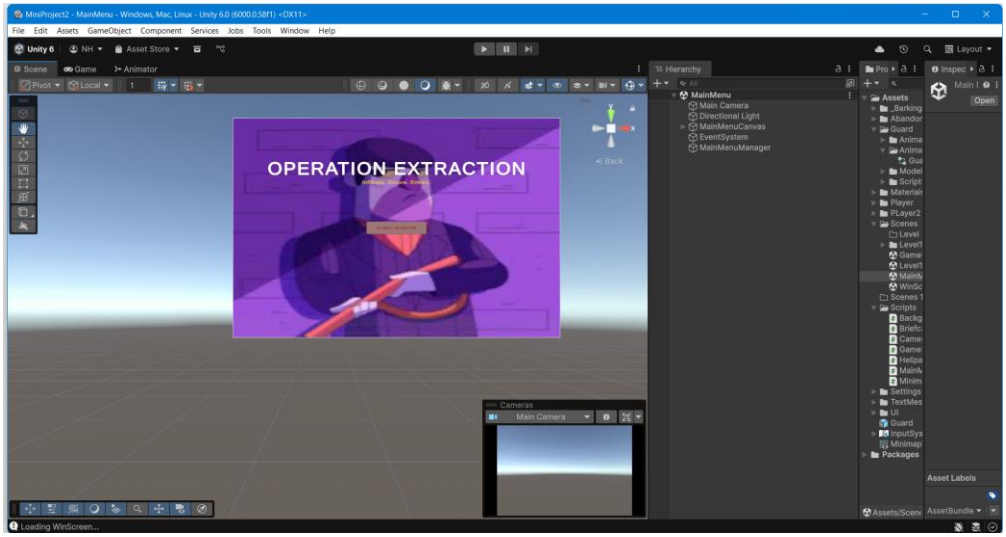
Tool	**Purpose**
Unity (Unity 6.0)	Game Engine
Mixamo	Character models and animations
ProBuilder	Environment and maze design
NavMesh	AI navigation
C#	Game scripting
Unity UI System	Menus, text, and HUD
Animator Controller	Character animations
NavMeshAgent	Enemy movement

Scene Structure

The project consists of the following scenes:

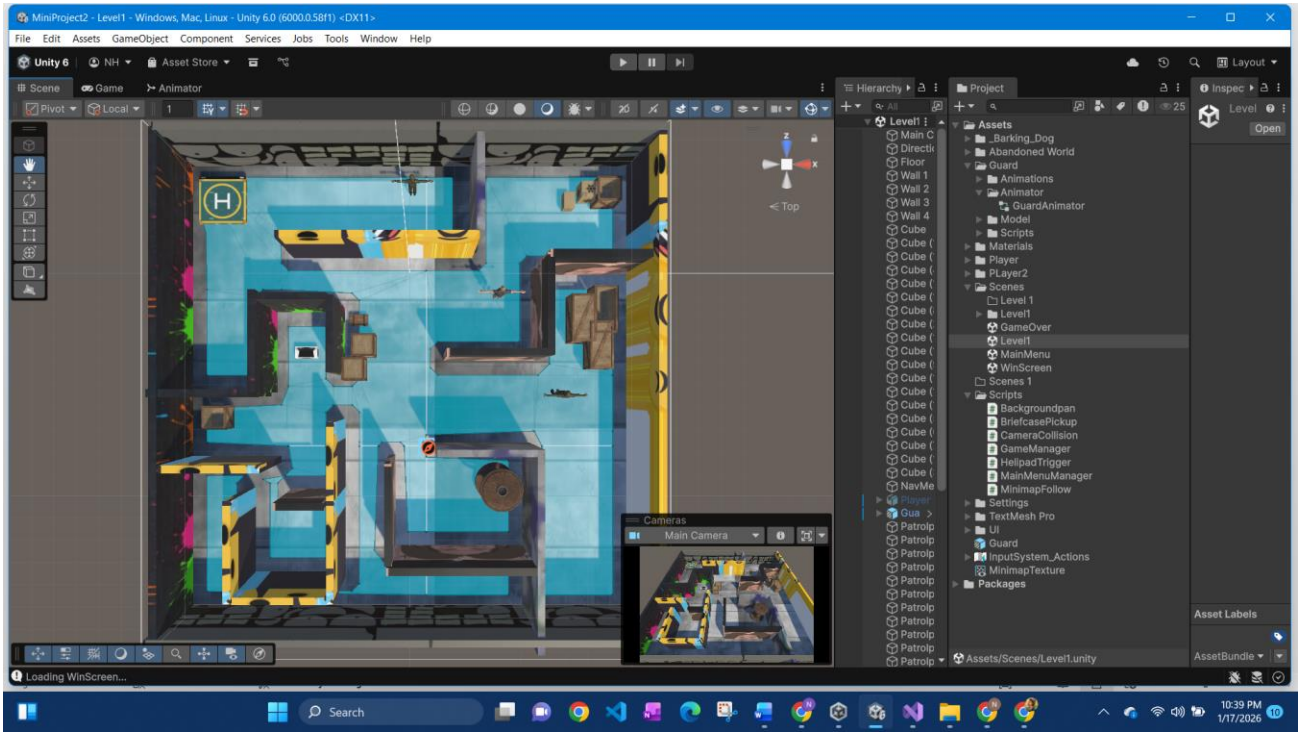
MainMenu

- Start Game



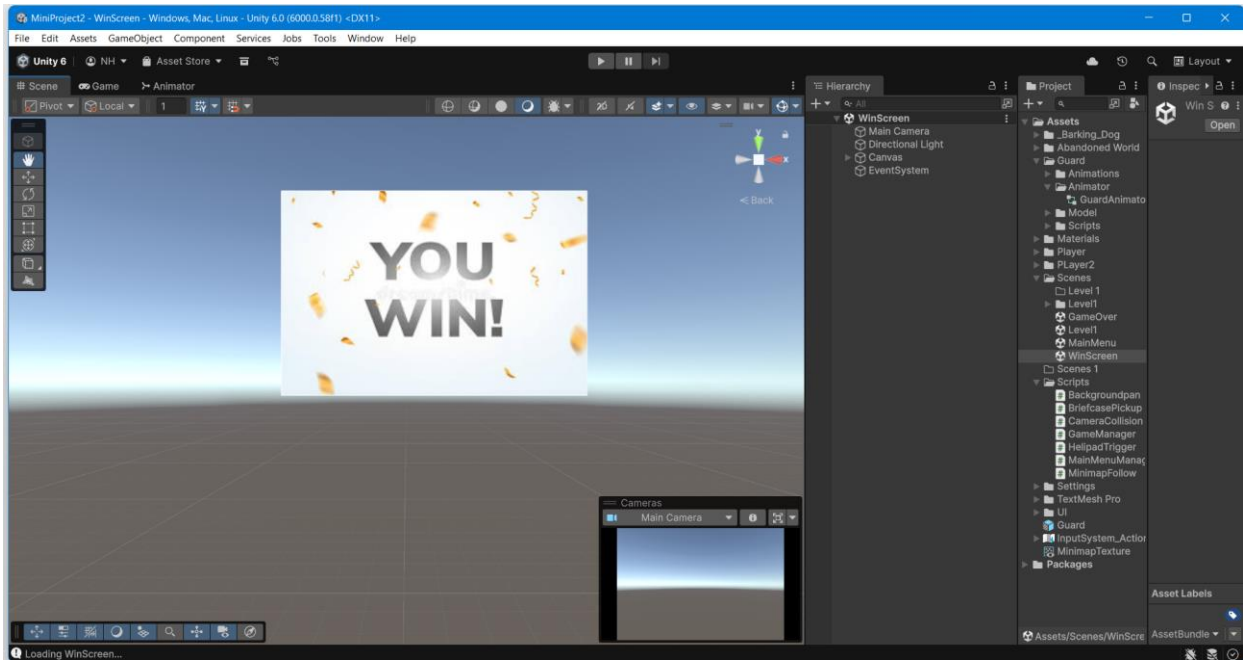
Level1

- Main gameplay scene
- Player, Guards, Maze, Briefcase, Helipad



WinScreen

- Displays “MISSION COMPLETE”



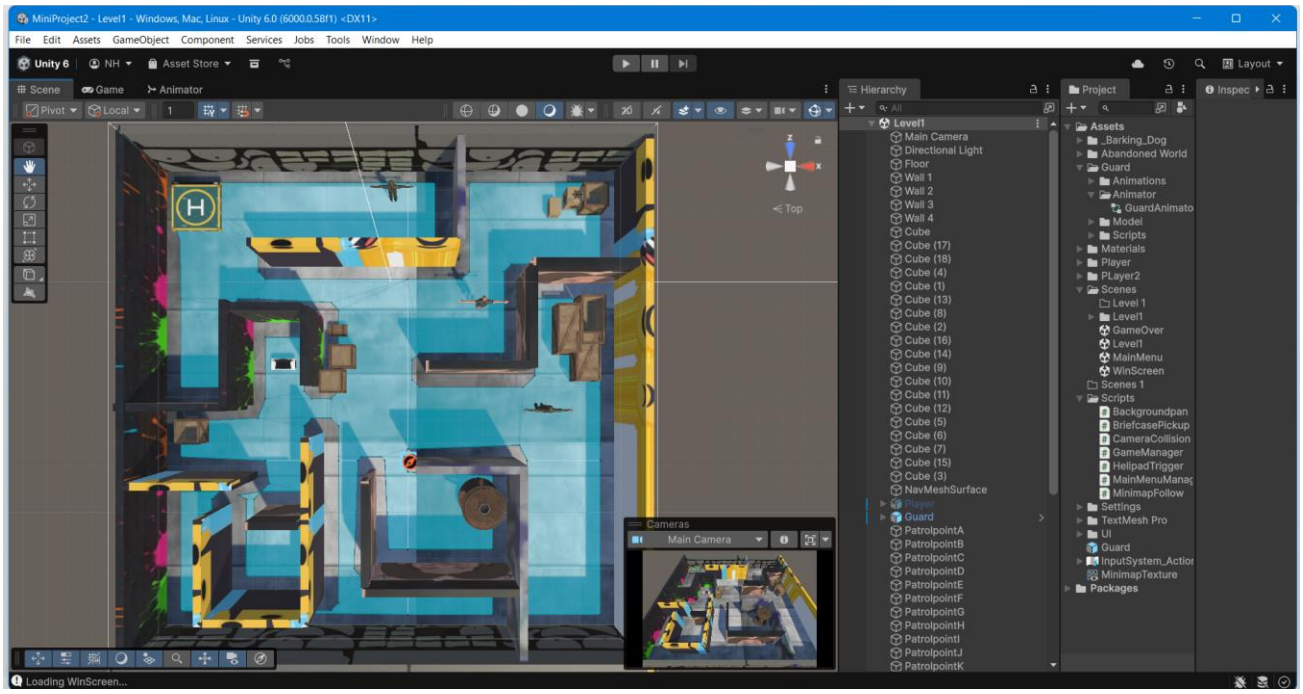
Environment Design

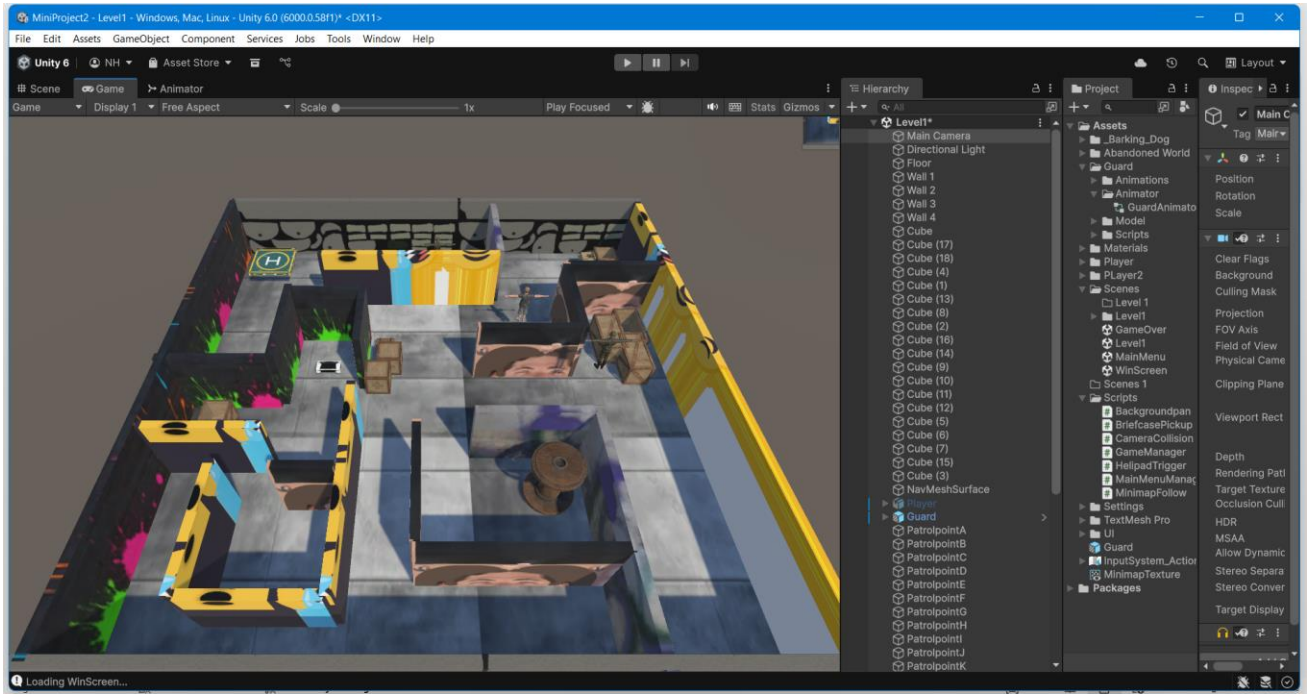
The game environment was created using **ProBuilder**.

Key Elements:

- Floor and walls built using cube meshes
- Maze layout with corridors, turns, and rooms
- Separate area for the briefcase
- Clear path leading to the helipad

All environment objects were marked as **Static** to ensure correct NavMesh baking.





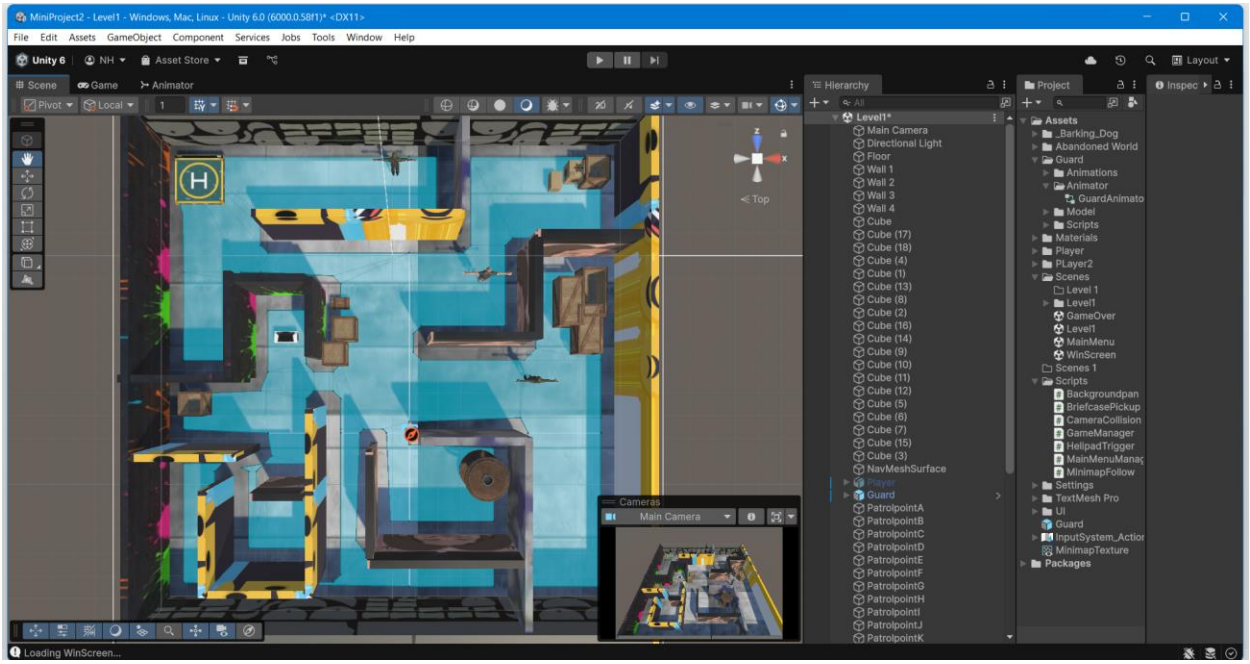
Navigation System (NavMesh)

A **NavMesh Surface** component was added to the scene and baked to define walkable areas for AI characters.

Purpose:

- Enables guards to navigate around walls

- Prevents AI from walking through obstacles
- Ensures realistic movement



Player Character Setup

Player Model

- A humanoid character from **Mixamo**
- Rig set to **Humanoid**

Player Animations (Mixamo)

The following animations were imported:

- Idle
- Walk
- Run
- Crouch

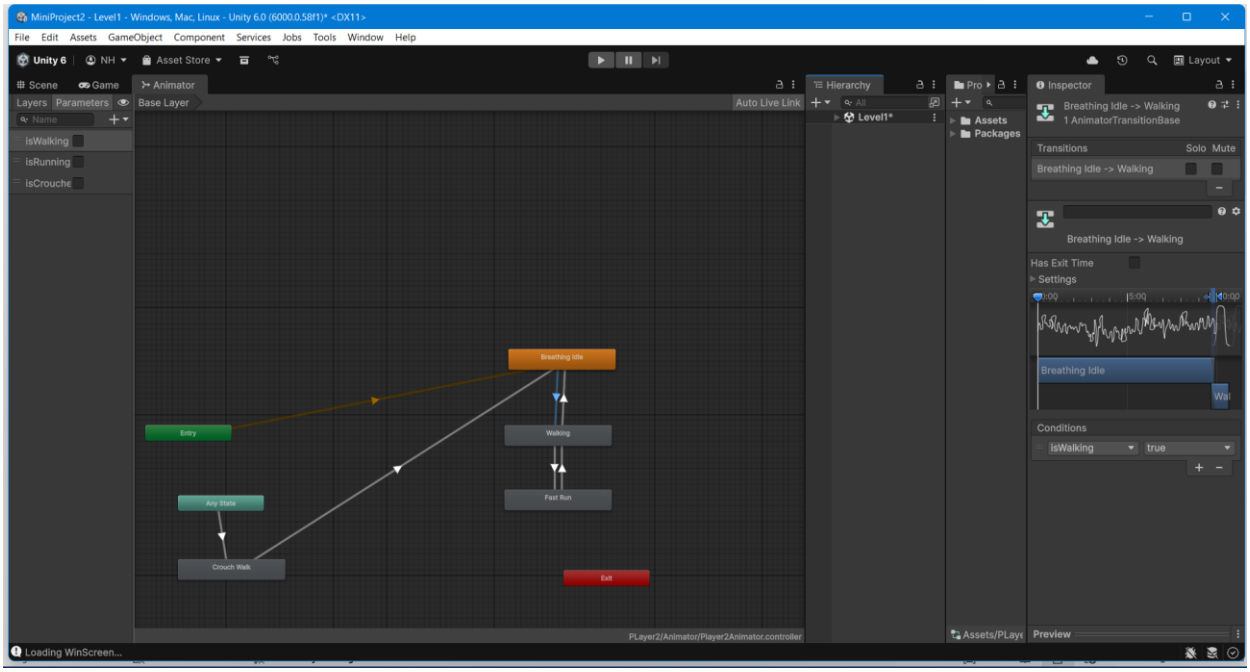
Animator Controller (Boolean-Based)

The Animator Controller uses **Boolean parameters** instead of blend trees:

Parameter	Type
isWalking	Bool

Parameter	Type
isRunning	Bool
isCrouched	Bool

Transitions were created between states with **Has Exit Time disabled** to ensure instant animation response.



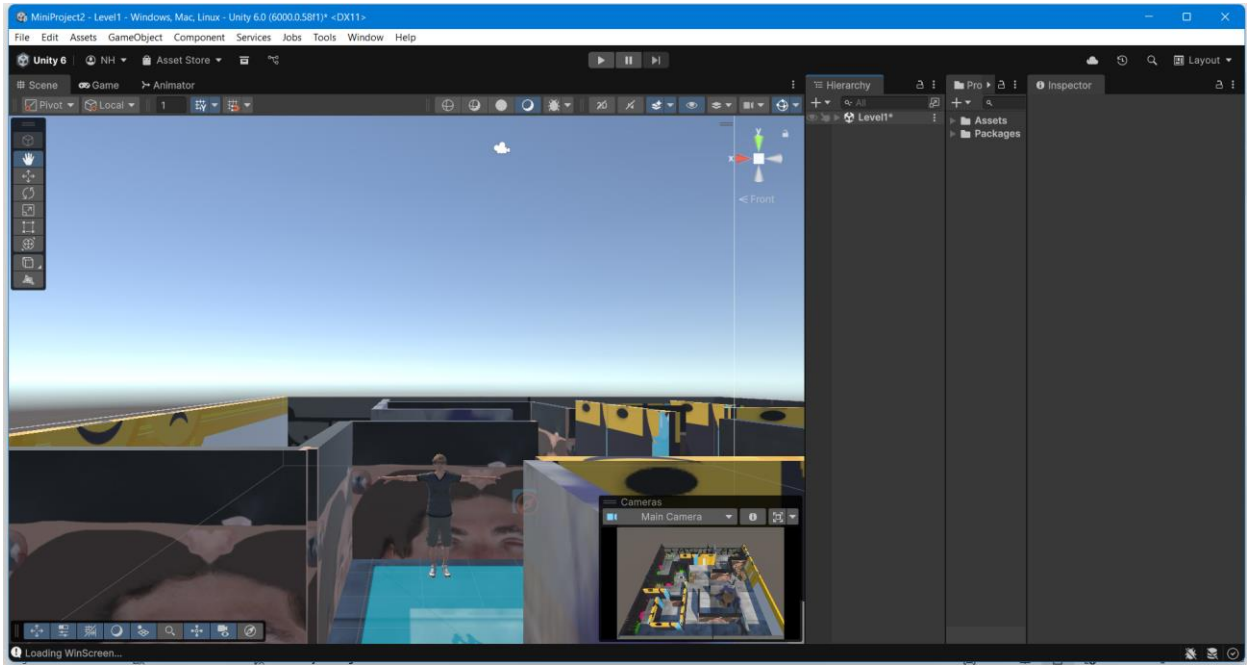
Player Movement Script

The player movement is implemented using a **CharacterController**.

Features:

- WASD for movement
- Left Shift for running
- C for crouching
- Gravity handling
- Camera-relative movement
- Smooth character rotation
- Animator parameter control

The script reads input using Input.GetAxis and applies movement via CharacterController.Move().



Enemy AI (Guards)

Guard Setup

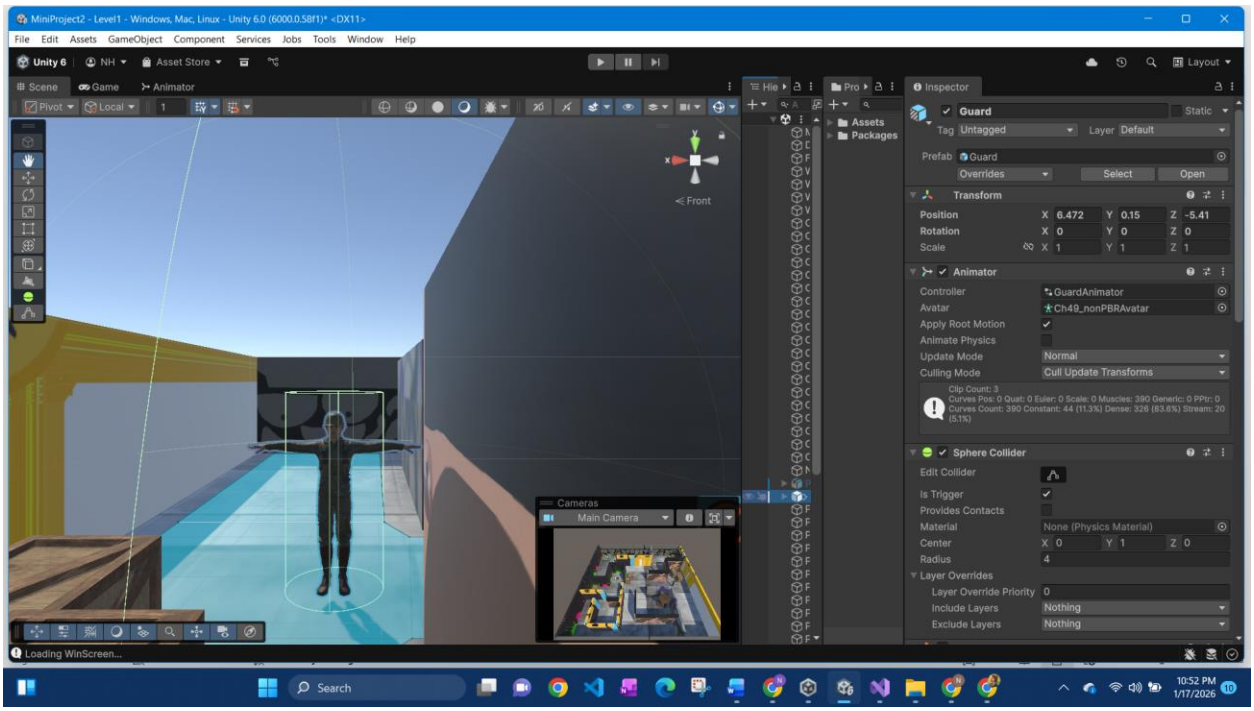
Each guard includes:

- NavMeshAgent
- Animator
- Capsule Collider (Trigger for detection)
- GuardAI script

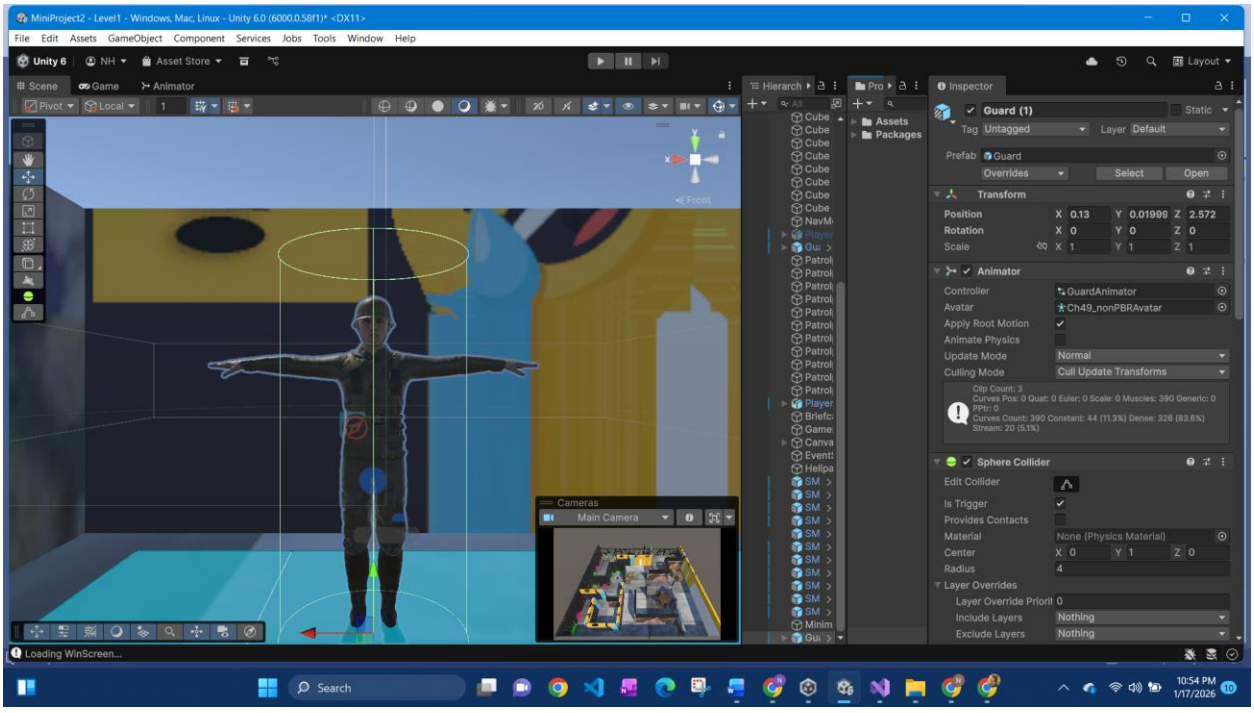
Two guards were placed in the level.

Patrol System

- Each guard has **independent patrol points**
- Total patrol points: **11**
- Guards move between points using NavMeshAgent
- Guards wait briefly at each patrol point



Guard 1



Guard 2

Guard Detection and Chase System

Detection:

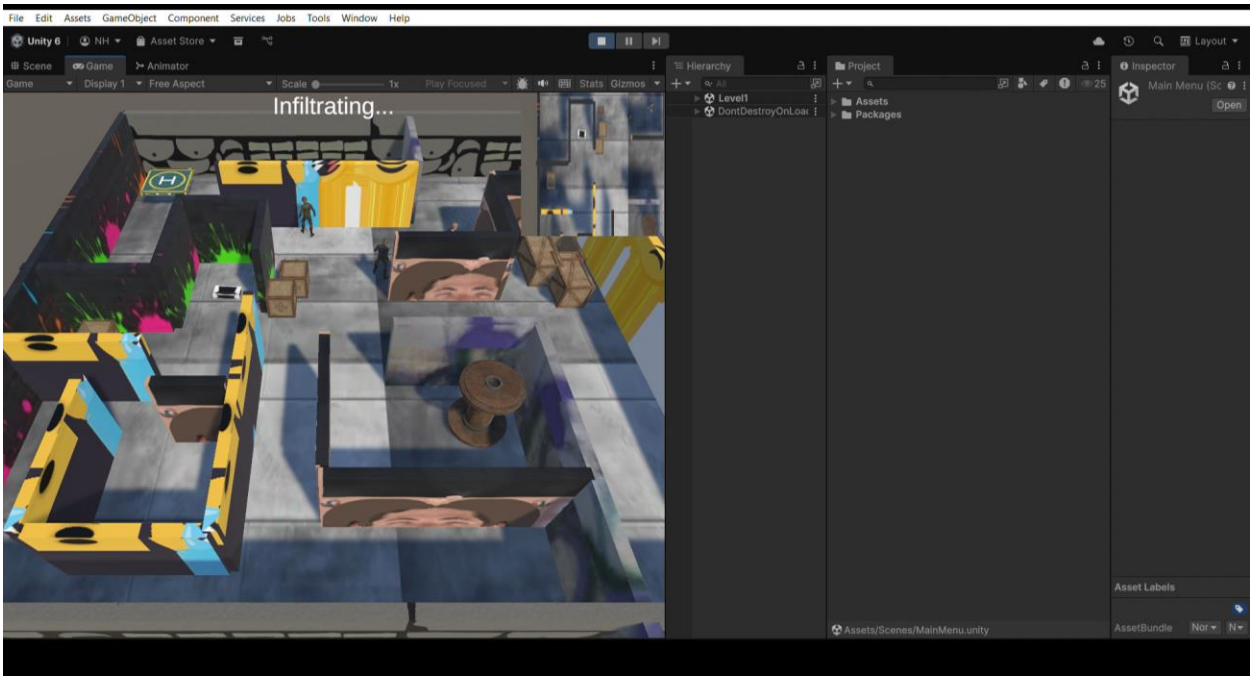
- Sphere/Capsule trigger collider
- When player enters trigger → guard starts chasing

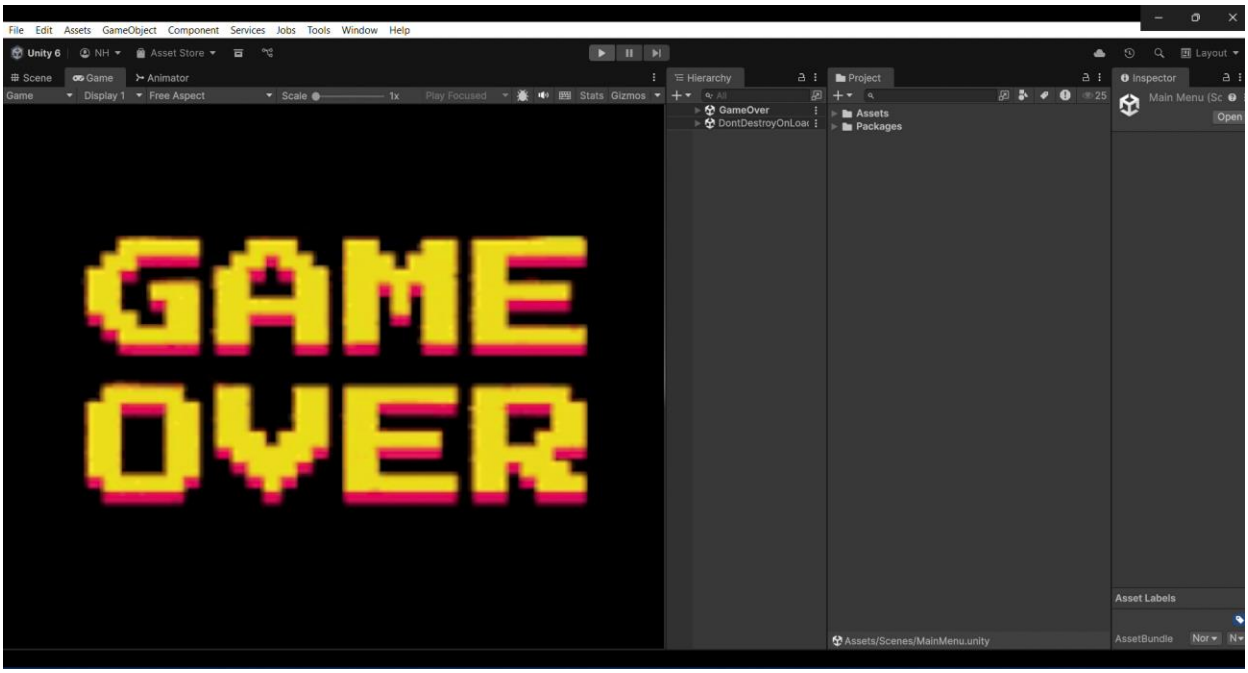
Chase:

- Guard speed increases
- Running animation plays
- Guard continuously sets destination to player position

Lose Condition:

- If distance between guard and player < **1.5 meters**
- Player is considered caught



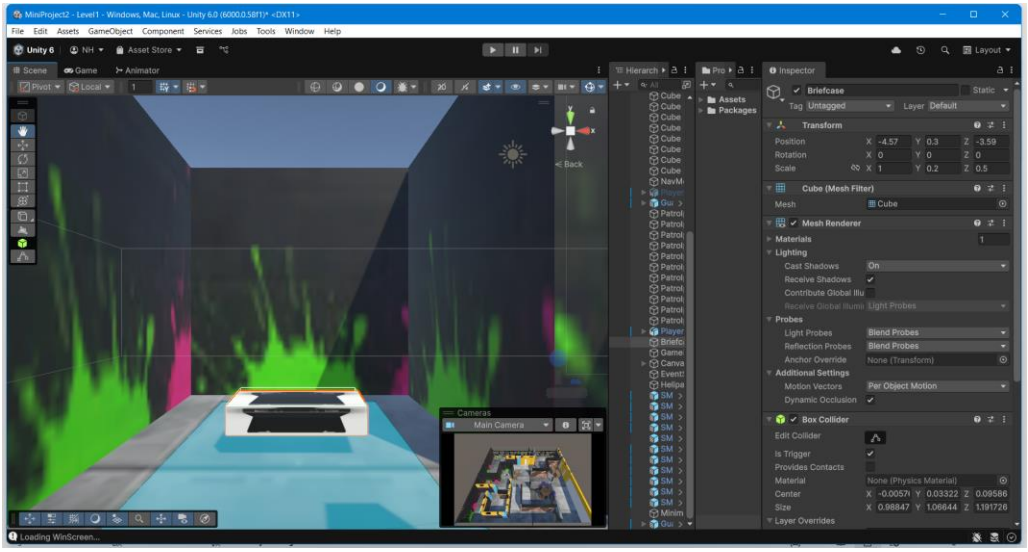


Briefcase Pickup System

A **Data Briefcase** object is placed in a secure room.

Functionality:

- Uses OnTriggerEnter
- Detects player collision
- Updates game status
- Displays "Objective Secured"
- Briefcase disappears after pickup

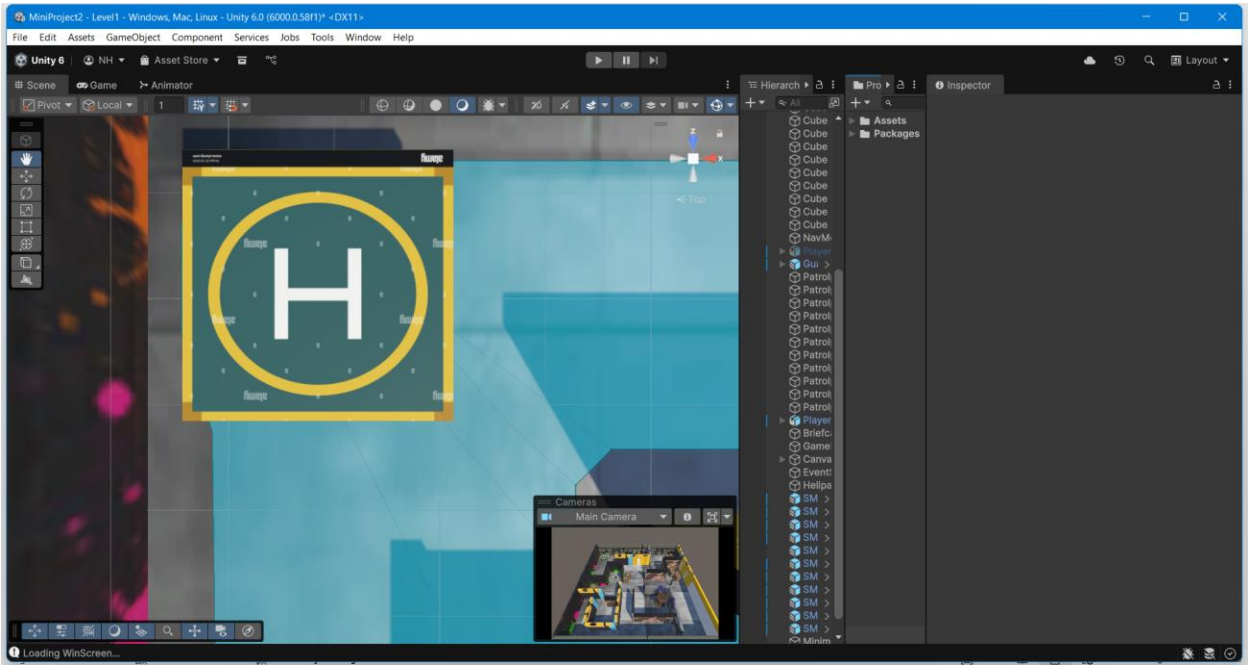


Helipad & Win Condition

The helipad is a trigger zone placed at the end of the level.

Functionality:

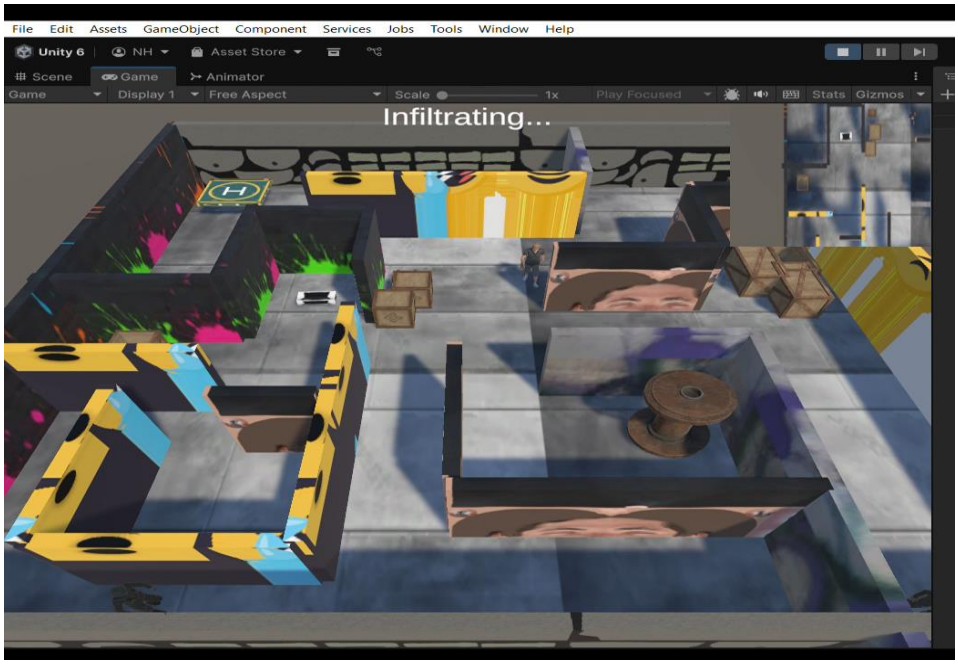
- Detects player entry
- Loads **WinScreen** scene
- Displays “MISSION COMPLETE”

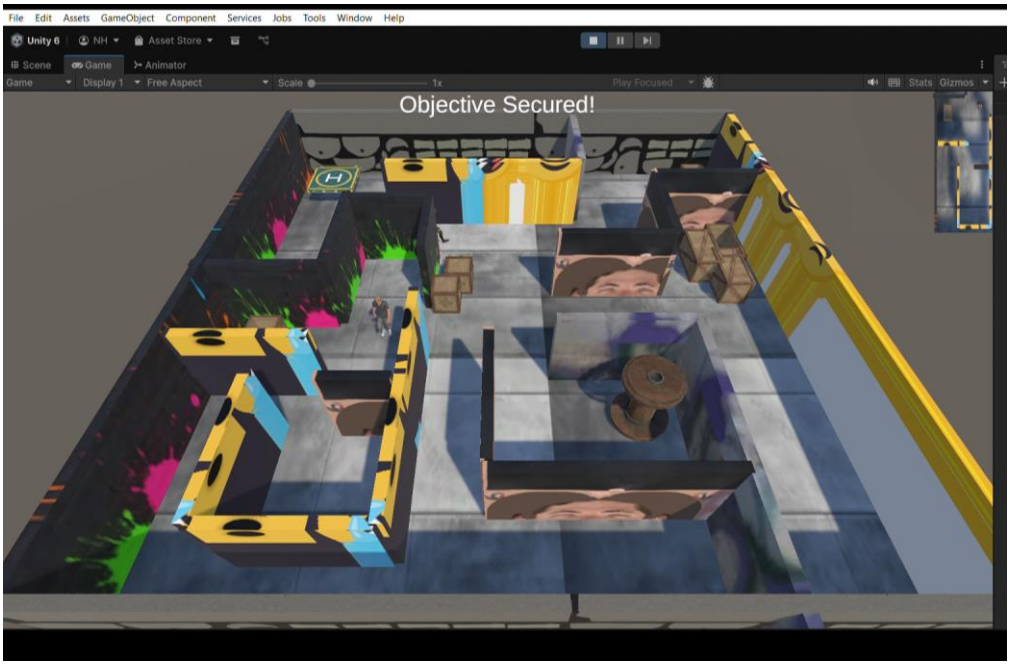


User Interface (UI)

UI Elements:

- Status Text:
  - “Infiltrating”
  - “SPOTTED!”
  - “Objective Secured”
- Main Menu buttons
- Win Screen text
- Mini-map





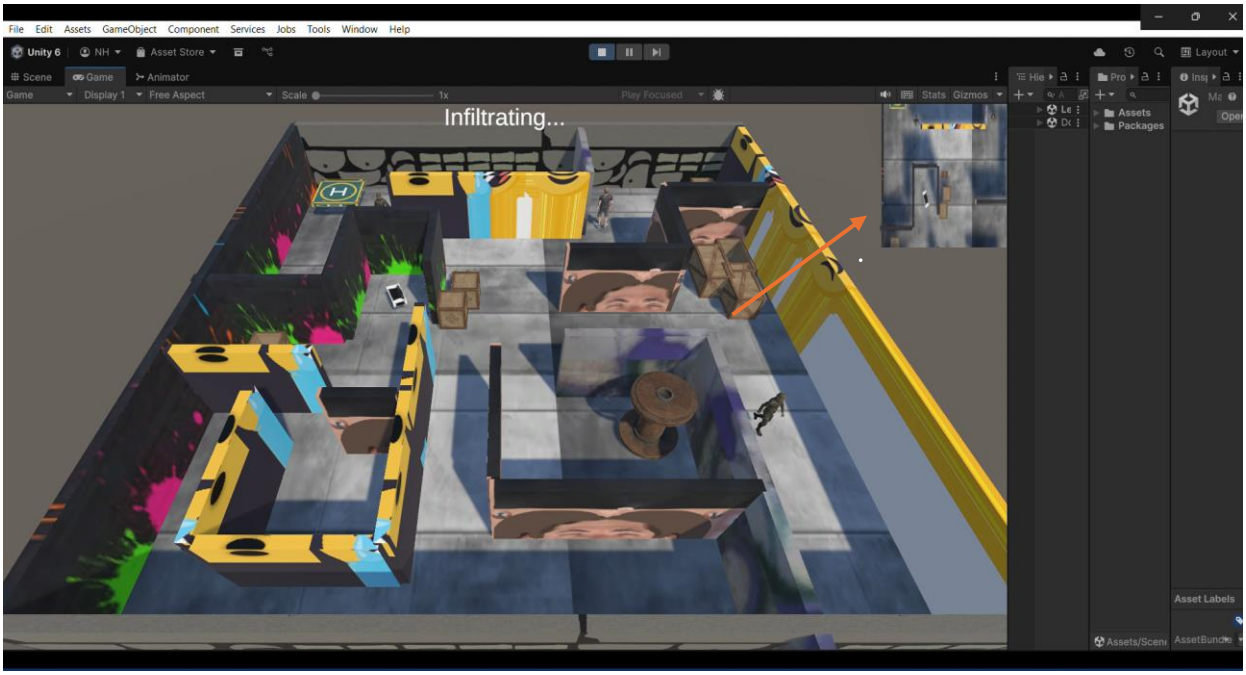
Mini-Map System

A mini-map was implemented using:

- Secondary top-down camera
- RenderTexture
- UI RawImage

Features:

- Displays player position
- Shows guard locations
- Helps navigation within the maze

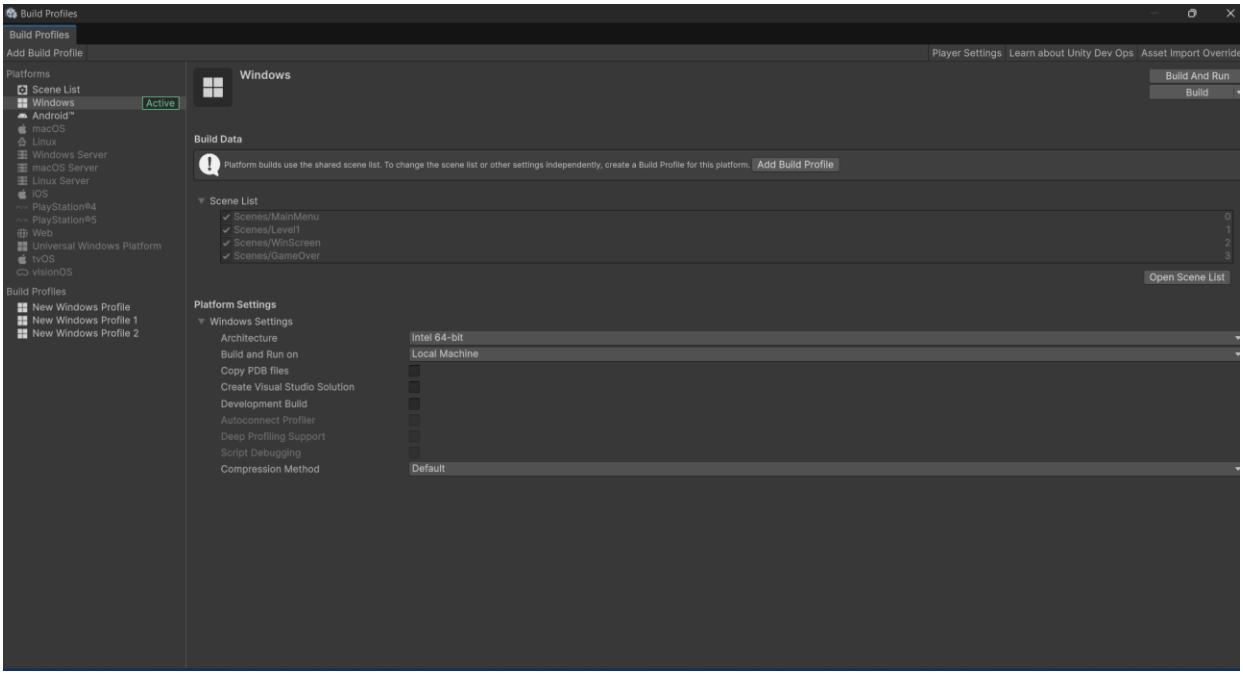


Scene Management

A **GameManager/MainMenuManager** script handles:

- Scene transitions
- Start Game
- Quit Game
- Win and Lose logic

Unity Build Profiles were used to ensure scenes load correctly.



Conclusion

The project successfully meets all requirements of **GD Mini Project 2**:

- Functional stealth gameplay
- Responsive animations
- Intelligent enemy AI
- Clear win/loss conditions
- Polished UI and mini-map

SCRIPTS

PLAYERCONTROLLER.CS

```
using UnityEngine;

public class PlayerMovement1 : MonoBehaviour

{

   public float walkSpeed = 3f;

   public float runSpeed = 6f;

   public float crouchSpeed = 1.5f;
```

```
public float mouseSensitivity = 150f;
public float turnSpeed = 10f;
public Transform cameraPivot;
public Transform playerModel;
private CharacterController controller;
private Animator animator;
private float gravity = -9.8f;
private float yVelocity;
private float xRotation = 0f;
void Start()
{
   controller = GetComponent<CharacterController>();
   animator = GetComponent<Animator>();
   Cursor.lockState = CursorLockMode.Locked;
   Cursor.visible = false;
}
void Update()
{
   HandleMouseLook();
   HandleMovement();
}
void HandleMouseLook()
{
   float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
   float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;
   xRotation -= mouseY;
   xRotation = Mathf.Clamp(xRotation, -80f, 80f);
   cameraPivot.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
}
void HandleMovement()
{
```

```
        float h = Input.GetAxisRaw("Horizontal");
        float v = Input.GetAxisRaw("Vertical");
        bool isMoving = (h != 0 || v != 0);
        bool isRunning = Input.GetKey(KeyCode.LeftShift);
        bool isCrouched = Input.GetKey(KeyCode.C);
        float speed = walkSpeed;
        if (isCrouched)
            speed = crouchSpeed;
        else if (isRunning)
            speed = runSpeed;
        Vector3 camForward = Camera.main.transform.forward;
        Vector3 camRight = Camera.main.transform.right;
        camForward.y = 0;
        camRight.y = 0;
        camForward.Normalize();
        camRight.Normalize();
        Vector3 moveDir = (camForward * v + camRight * h).normalized;
        Vector3 movement = moveDir * speed;
        if (controller.isGrounded)
            yVelocity = -2f;
        else
            yVelocity += gravity * Time.deltaTime;
        movement.y = yVelocity;
        controller.Move(movement * Time.deltaTime);
        if (isMoving)
        {
            Quaternion targetRotation = Quaternion.LookRotation(moveDir);
            playerModel.rotation = Quaternion.Slerp(playerModel.rotation, targetRotation, turnSpeed *
Time.deltaTime);
        }
        // Animator
```

```
        animator.SetBool("isWalking", isMoving);
        animator.SetBool("isRunning", isRunning && !isCrouched && isMoving);
        animator.SetBool("isCrouched", isCrouched);
    }
}
```

GUARDAI.CS

```
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.SceneManagement;
public class GuardAI : MonoBehaviour
{
    [Header("Patrol Settings")]
    public Transform[] patrolPoints; // 13 points
    public float patrolSpeed = 3f;
    public float waitTimeAtPoint = 2f; // time to pause at patrol points
    [Header("Chase Settings")]
    public Transform player;
    public float chaseSpeed = 4f;
    public float loseDistance = 0.5f; // distance at which player is caught
    [Header("Animation")]
    public Animator animator; // Guard Animator
    private string walkParam = "isWalking";
    private string runParam = "isRunning";
    private NavMeshAgent agent;
    private int currentPatrolIndex = 0;
    private bool chasing = false;
    private float waitTimer = 0f;
    void Start()
    {
```

```
    agent = GetComponent<NavMeshAgent>();
    if (agent == null) Debug.LogError("NavMeshAgent missing on Guard!");
    if (patrolPoints.Length == 0) Debug.LogError("No patrol points set!");
    agent.speed = patrolSpeed;
    GoToNextPatrolPoint();
}
void Update()
{
    if (chasing)
    {
        ChasePlayer();
    }
    else
    {
        Patrol();
    }
    // Animator speed
    if (chasing)
    {
        animator.SetBool(walkParam, false);
        animator.SetBool(runParam, true);
    }
    else
    {
        if (agent.velocity.magnitude > 0.1f)
        {
            animator.SetBool(walkParam, true);
            animator.SetBool(runParam, false);
        }
```

```
        else
        {
            animator.SetBool(walkParam, false);
            animator.SetBool(runParam, false);
        }
    }
}
#region Patrol
void Patrol()
{
    if (patrolPoints.Length == 0) return;
    if (!agent.pathPending && agent.remainingDistance < 0.5f)
    {
        if (waitTimer <= 0f)
        {
            waitTimer = waitTimeAtPoint;
        }
        else
        {
            waitTimer -= Time.deltaTime;
            if (waitTimer <= 0f)
            {
                GoToNextPatrolPoint();
            }
        }
    }
}
void GoToNextPatrolPoint()
{
```

```
    if (patrolPoints.Length == 0) return;
    agent.speed = patrolSpeed;
    agent.SetDestination(patrolPoints[currentPatrolIndex].position);
    currentPatrolIndex = (currentPatrolIndex + 1) % patrolPoints.Length;
}
#endregion
#region Chase
void ChasePlayer()
{
    if (player == null) return;
    agent.speed = chaseSpeed;
    agent.SetDestination(player.position);
    // Check lose condition
    float distance = Vector3.Distance(transform.position, player.position);
    if (distance <= loseDistance)
    {
        Debug.Log("Player Caught! Game Over!");
        SceneManager.LoadScene("GameOver");
    }
}
#endregion
#region Detection
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        chasing = true;
    }
}
```

```
    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            chasing = false;
            agent.SetDestination(patrolPoints[currentPatrolIndex].position);
        }
    }
    #endregion
}
```