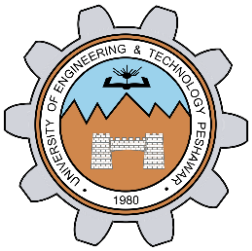
Project Report



Fall 2024

CSE-411L Intro to Game Development Lab

Submitted by:

Ali Asghar(21PWCSE2059)

Muhammad Sadeeq(21PWCSE2028)

Suleman Shah(21PWCSE1983)

Muhammad Shahab(21PWCSE2074)

Submitted to:

Engr. Abdullah Hamid

Date:

9th February 2025

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Contents

Project Report: League of Assassin's

Introduction

League of Assassins is a top-down stealth action game set in meticulously crafted 3D environments where players assume the role of an elite assassin. With a knife for silent kills and a gun equipped with a silencer for discreet ranged eliminations, players must infiltrate various locations to eliminate high-profile targets while staying undetected. Every level offers multiple approaches to completing the mission, with different outcomes depending on the player's tactics. A dynamic fame system tracks the player's performance and showcases their reputation as an assassin, rewarding stealth and precision while penalizing reckless actions.

Gameplay Features

The following are the main gameplay features of this game.

Top-Down Gameplay

The game features a top-down perspective, providing players with a strategic view of the environment and allowing them to plan their movements and stealth tactics effectively. The 3D environments are rich and detailed, offering multiple pathways, hiding spots, and opportunities for environmental interaction, which are crucial for completing stealth-based missions.

Player Character - The Assassin

Players take on the role of a skilled assassin belonging to the prestigious League of Assassins. The character is equipped with two primary weapons:

Knife

A silent, close-range weapon that allows the player to eliminate targets quietly and without alerting others. It is ideal for stealth kills when the player is up close to enemies.

Silenced Gun

A range weapon with a silencer attached, ensuring that the gunshots do not alert nearby enemies. However, the gun has limited ammunition, so players must use it strategically, balancing between silent knife kills and careful gunshots when needed.

The key challenge is to carefully use both weapons in combination, determining when to rely on stealthy knife kills or when the silenced gun is necessary to eliminate distant or harder-to-reach targets.

Stealth and Assassination
At the heart of the gameplay is the need for stealth. The environment is full of patrolling enemies and obstacles, and players must navigate these spaces without being detected. Enemy AI is programmed with dynamic behavior patterns, including patrolling, standing guard, and reacting to noises or visual cues

Detection System
Enemies have a field of vision and a hearing range. If the player enters this field or makes noise, they risk being detected. The player's goal is to avoid detection by using cover, staying out of sight, and using the silenced gun or knife when appropriate. The player must plan their approach carefully to avoid triggering alarms or being seen.

Level Objectives and Approaches
Each mission has a central target the player must eliminate. There are multiple ways to complete the objectives:

Stealth Kill (Silent Approach)
The player avoids detection by using the knife and silenced gun to take out enemies quietly. The target is killed without alerting anyone, and no chaos is caused. This approach reflects a true assassin's skill and rewards players with high fame.

Discreet Gunshots (Silenced Approach)
The player uses the silenced gun to eliminate the target or any necessary guards. Although this still maintains a level of discretion, it's riskier than relying on the knife. Players must be cautious with the gun's limited ammo.

Avoid Combat (Non-violent Approach)
The player may decide to avoid killing other enemies entirely, focusing only on the target. By bypassing most guards and using stealth to reach the target undetected, the player avoids unnecessary bloodshed, which can impact the ending.

Failed Stealth
If the player is detected during the mission, the game forces a shift to combat mode. The player may then choose to fight their way out or attempt a hasty escape, but they will suffer a loss in reputation, and the mission ends in failure.

3D Environments

The game features expansive 3D environments that vary from urban settings and sleek high-rise buildings to more intimate, confined spaces like dark alleyways and hidden underground facilities. Each environment is designed to encourage strategic thinking, with ample opportunities for stealth, hiding spots, and creative ways to approach the target.

Contribution of Each Group Member

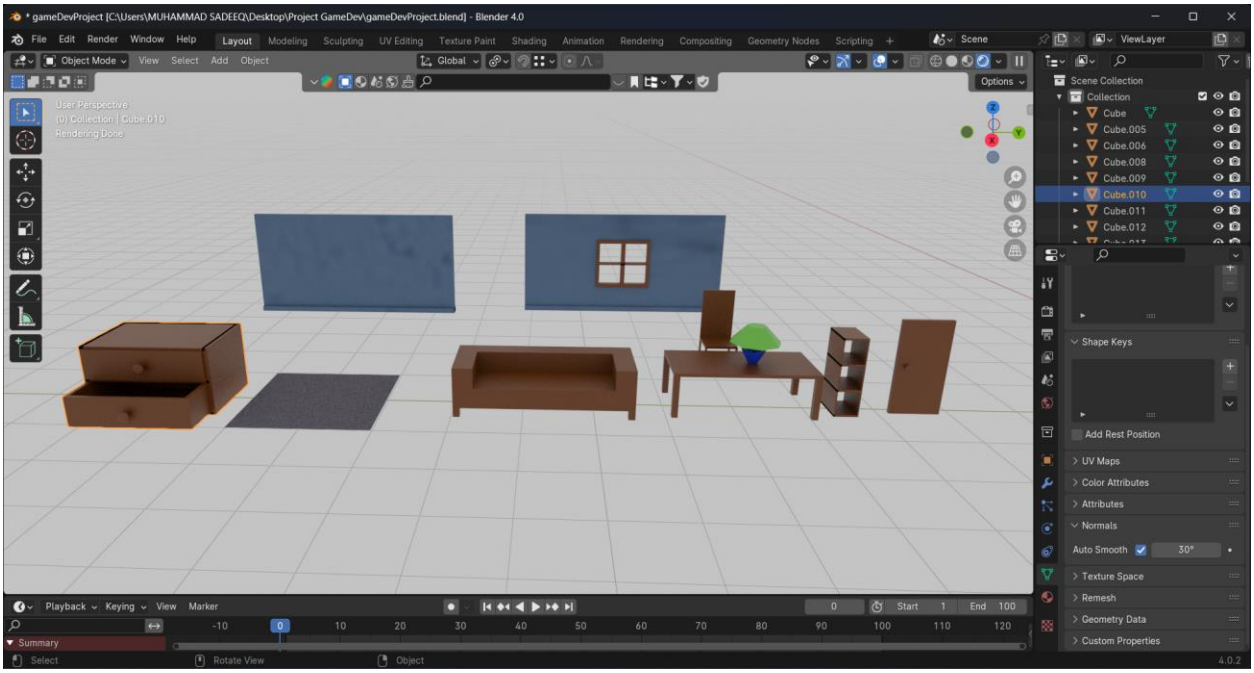
The contribution of each group member is described as follows.

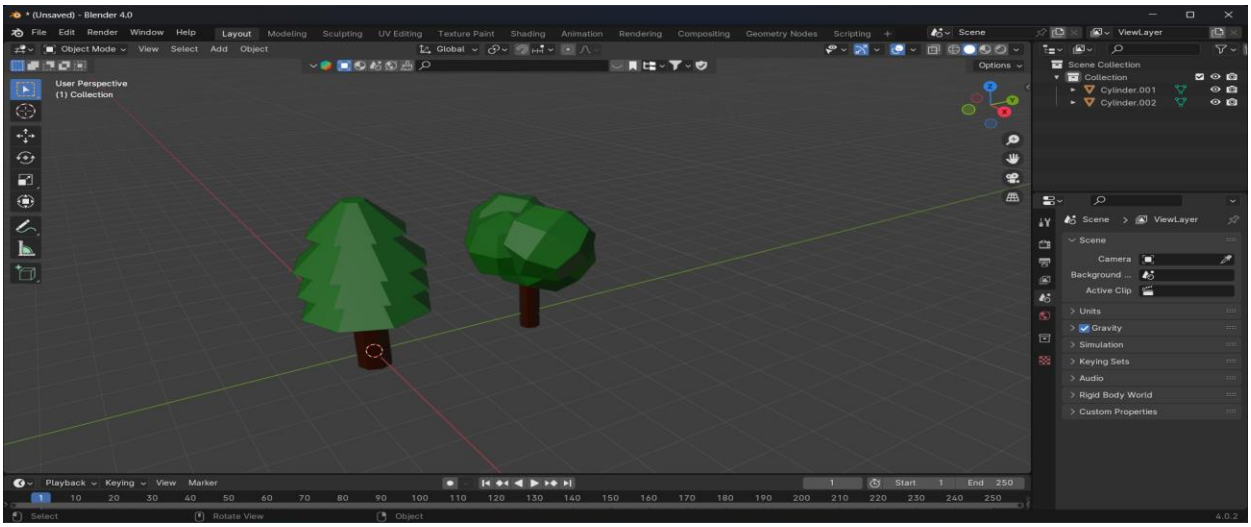
Muhammad Sadeeq's Contribution as a 3D Artist

As the 3D Artist, Sadeeq was responsible for creating various models essential to the game. These models were designed, textured, and optimized for performance in Unity. Below are the key models he created:

1. Environment Assets

- **Walls:** Designed low-poly walls with textures to create an immersive game world.
- **Props:** Created objects such as table, chairs, doors, and lamp to enhance the game environment.
- **Terrain Features:** Modeled, trees, and grass to add realism to outdoor areas.



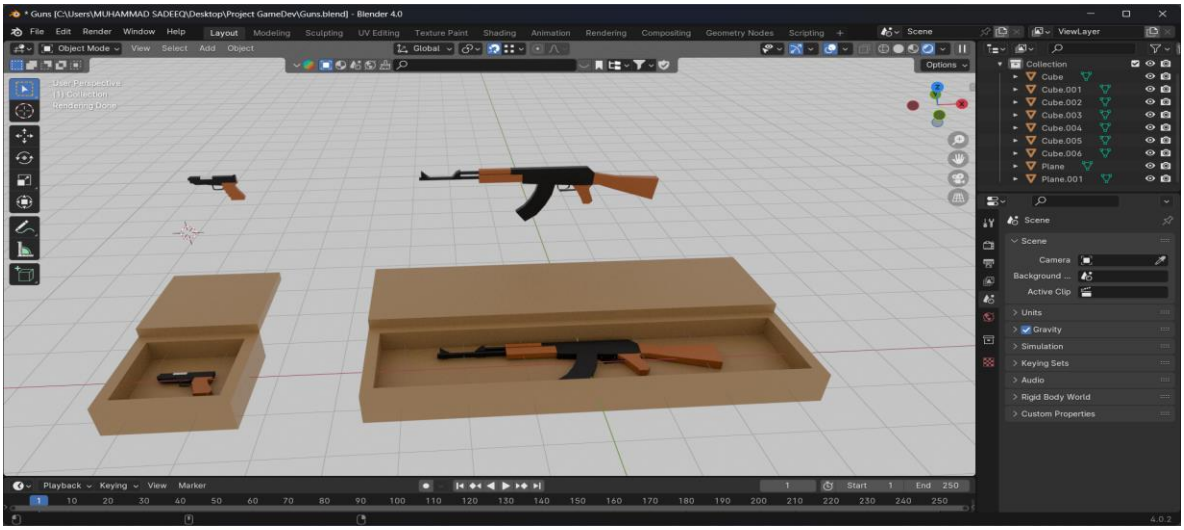


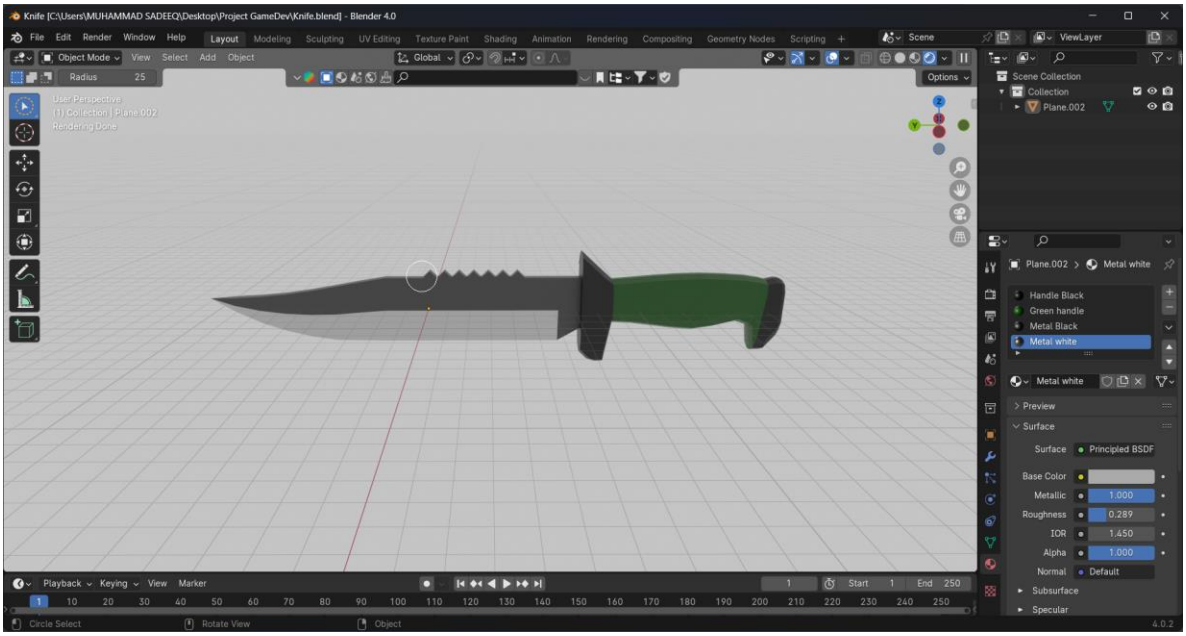
2. Character Models

- **Main Character:** Downloaded a 3D model for playable character, fixing and optimization of it mesh, ensuring proper rigging and animation compatibility.
- **NPCs (Non-Playable Characters):** Created enemy NPC from the main character by making changes in it mesh and materials.

3. Weapons & Items

- **Weapons:** Modeled guns, other weapons based on game requirements.
- **Collectibles:** Designed gun boxes, and other collectible items.





Tools & Software Used

- **Blender:** 3D modeling, texturing, and UV unwrapping.
- **Unity 3D:** Integration of 3D assets into the game.
- **Mixamo:** For rigging the characters

Challenges & Solutions

- **Challenge:** High-poly models affected performance.
    - **Solution:** made low-poly asset
- **Challenge:** Texture misalignment issues.
    - **Solution:** Proper UV unwrapping and texture mapping.
- **Challenge:** Importing models with animations.

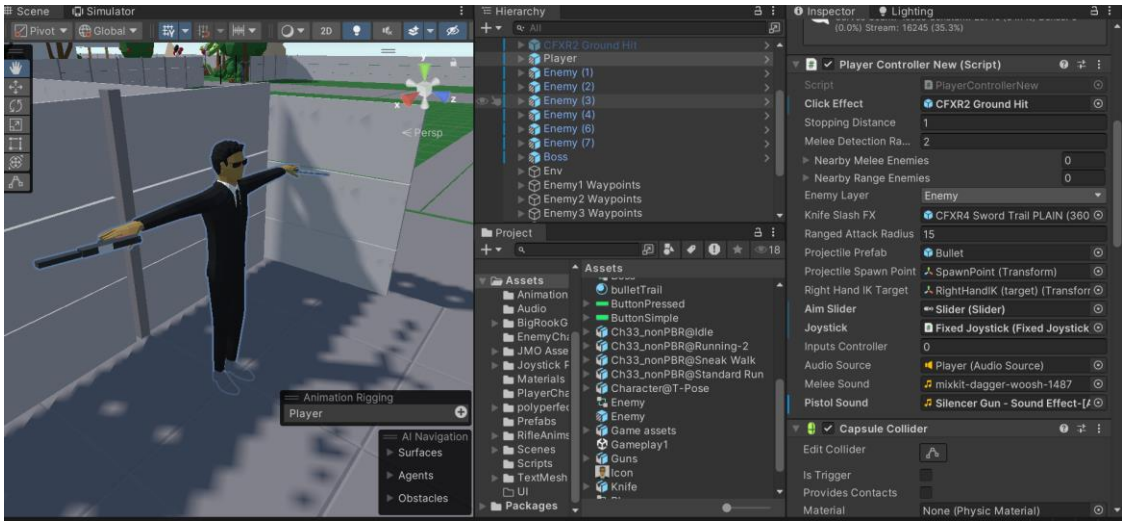
- **Solution:** Ensured correct rigging and exported models in FBX format for seamless Unity integration.

Ali Asghar's Contribution as Main Logic Designer

As the Main Logic Designer, Ali Asghar was responsible for implementing core gameplay mechanics and system logic. He designed and optimized game interactions, ensuring smooth and efficient execution in Unity. Below are the key areas he contributed to:

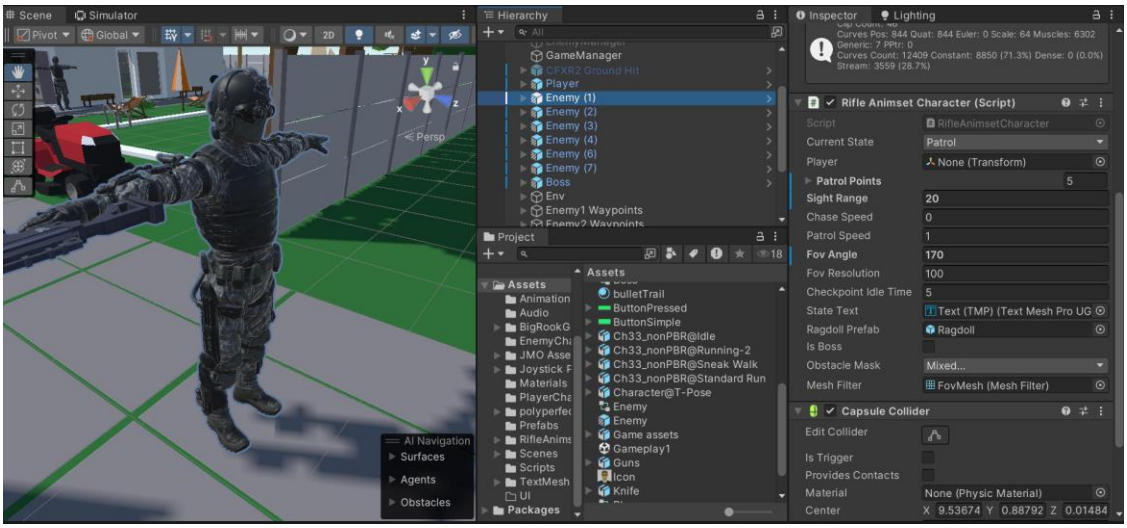
1. Gameplay Mechanics

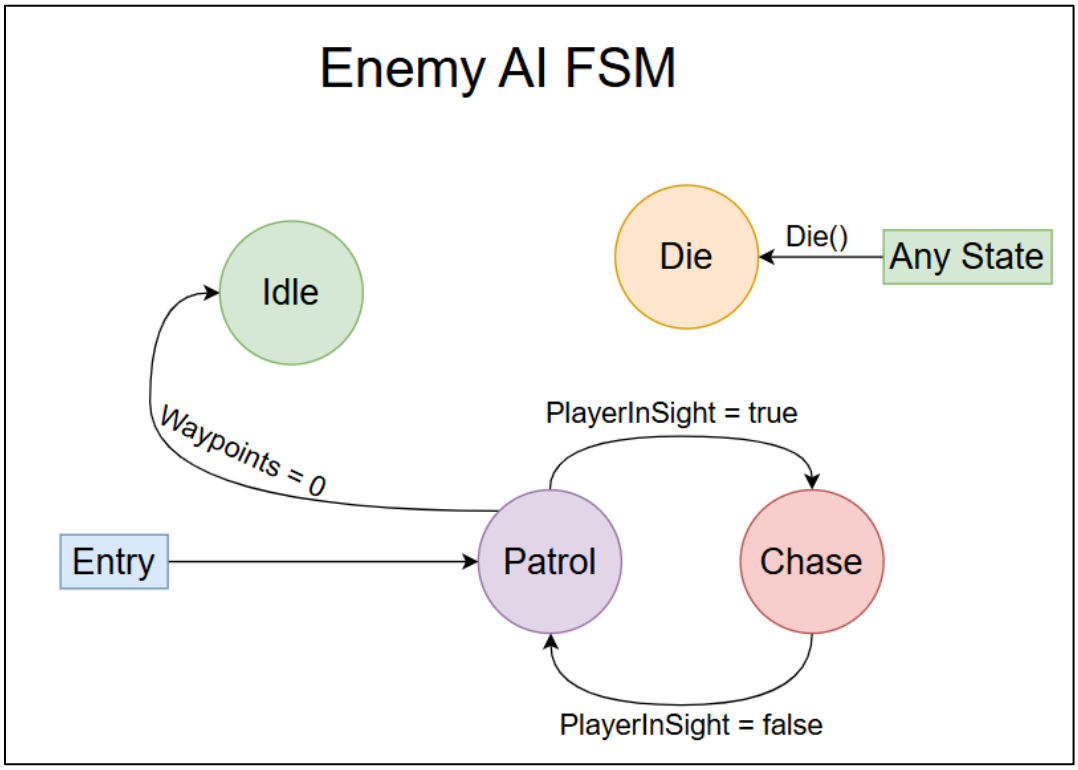
- **Player Input Controls:** Developed responsive player movement, handling inputs and physics interactions.
- **Combat System:** Implemented shooting mechanics, enemy AI behavior, and damage calculations.



2. AI & Game Systems

- **Enemy AI:** Programmed NPC behaviors, including patrolling, attacking, and reacting to the player.
- **State Management:** Utilized state machines to manage different game states such as idle, attack, and chase.





Challenges & Solutions

- **Challenge:** AI pathfinding was inefficient in complex environments.
  - **Solution:** Implemented NavMesh for dynamic navigation and obstacle avoidance.
- **Challenge:** Player inputs felt unresponsive.
  - **Solution:** Refined input handling with interpolation and better physics calculations.

Suleman Shah's Contribution as Level and Game Designer

As the Level and Game Designer, Suleman Shah was responsible for designing engaging game environments and ensuring a seamless player experience. He crafted immersive levels, balanced game mechanics, and optimized player progression. Below are the key areas he contributed to:

1. Level Design & Environment Development

- **Map Layouts:** Designed well-structured game levels, ensuring balanced pacing and exploration opportunities.
- **Environmental Storytelling:** Created immersive environments that enhanced narrative and player engagement.
- **Player Navigation:** Designed clear pathways and objectives to ensure intuitive movement and interaction.

Challenges & Solutions

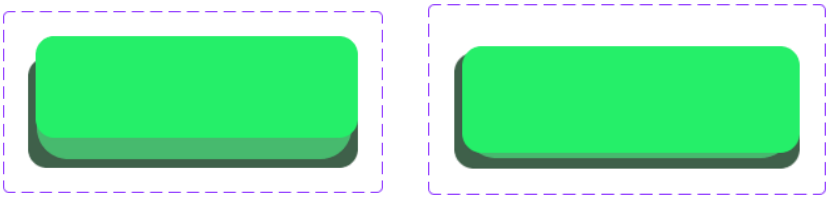
- **Challenge:** Levels felt too linear, reducing player engagement.
  - **Solution:** Introduced multiple pathways and hidden areas to encourage exploration.
- **Challenge:** Players struggled with navigation due to poor visibility of objectives.
  - **Solution:** Enhanced visual cues and lighting to guide players more effectively

Muhammad Shahab's Contribution as UI Designer

As the UI Designer, Muhammad Shahab was responsible for creating an intuitive and visually appealing user interface. He focused on usability, accessibility, and aesthetic consistency to enhance the player's experience. Below are the key areas he contributed to:

1. User Interface Design

Menu & HUD Design: Developed user-friendly menus, heads-up displays (HUDs), and interactive elements.

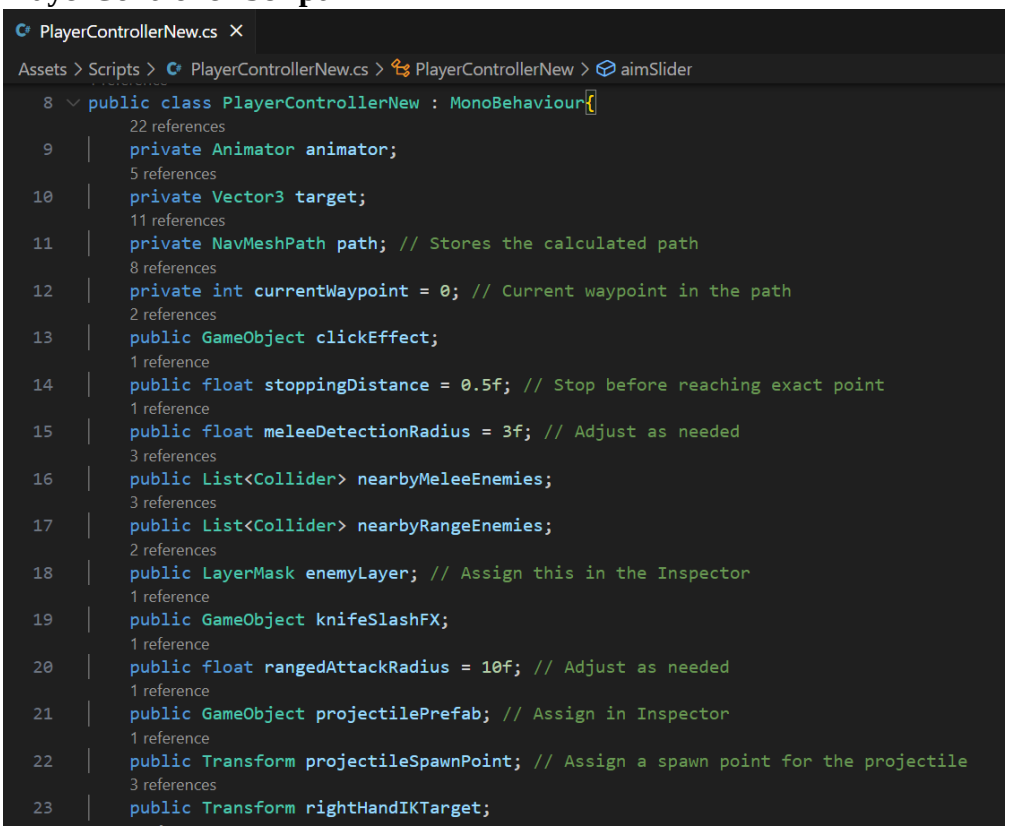


Challenges & Solutions

- **Challenge:** UI elements were cluttered, reducing usability.
  - **Solution:** Simplified layouts and implemented clear visual hierarchies.
- **Challenge:** Inconsistent UI responsiveness across different devices.
  - **Solution:** Used scalable design techniques and adaptive UI components.

Code Screenshots

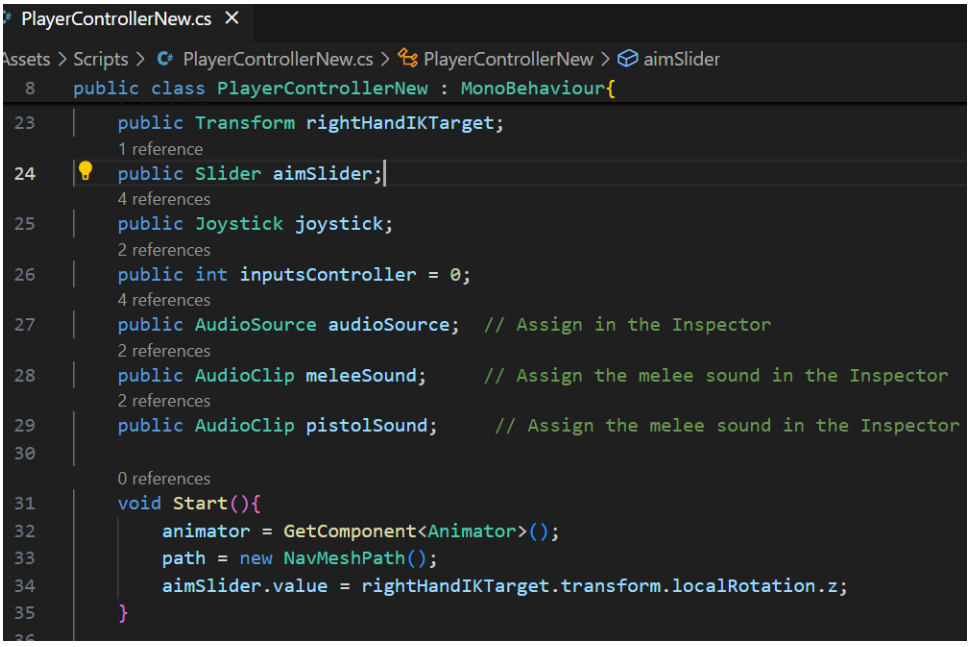
PlayerController Script



PlayerControllerNew.cs

Assets > Scripts > PlayerControllerNew.cs > PlayerControllerNew > aimSlider

```
8   public class PlayerControllerNew : MonoBehaviour{
        22 references
9       private Animator animator;
        5 references
10      private Vector3 target;
        11 references
11      private NavMeshPath path; // Stores the calculated path
        8 references
12      private int currentWaypoint = 0; // Current waypoint in the path
        2 references
13      public GameObject clickEffect;
        1 reference
14      public float stoppingDistance = 0.5f; // Stop before reaching exact point
        1 reference
15      public float meleeDetectionRadius = 3f; // Adjust as needed
        3 references
16      public List<Collider> nearbyMeleeEnemies;
        3 references
17      public List<Collider> nearbyRangeEnemies;
        2 references
18      public LayerMask enemyLayer; // Assign this in the Inspector
        1 reference
19      public GameObject knifeSlashFX;
        1 reference
20      public float rangedAttackRadius = 10f; // Adjust as needed
        1 reference
21      public GameObject projectilePrefab; // Assign in Inspector
        1 reference
22      public Transform projectileSpawnPoint; // Assign a spawn point for the projectile
        3 references
23      public Transform rightHandIKTarget;
```



PlayerControllerNew.cs

Assets > Scripts > PlayerControllerNew.cs > PlayerControllerNew > aimSlider

```
8   public class PlayerControllerNew : MonoBehaviour{
23      public Transform rightHandIKTarget;
        1 reference
24      public Slider aimSlider;
        4 references
25      public Joystick joystick;
        2 references
26      public int inputsController = 0;
        4 references
27      public AudioSource audioSource;  // Assign in the Inspector
        2 references
28      public AudioClip meleeSound;     // Assign the melee sound in the Inspector
        2 references
29      public AudioClip pistolSound;     // Assign the melee sound in the Inspector
30
        0 references
31      void Start(){
32          animator = GetComponent<Animator>();
33          path = new NavMeshPath();
34          aimSlider.value = rightHandIKTarget.transform.localRotation.z;
35      }
```

```
0 references
void Update(){

    inputsController = PlayerPrefs.GetInt("InputsControl", 0);

    if (inputsController == 0){
        JoystickInputs();
        joystick.gameObject.SetActive(true);
        joystick.gameObject.SetActive(true);
    }
    else{
        PointClickInputs();
        joystick.gameObject.SetActive(false);
    }

}

1 reference
void MoveAlongPath(){
    if (path.corners.Length == 0 || currentWaypoint >= path.corners.Length){
        GetComponent<Rigidbody>().freezeRotation = true;
        return;
    }

    Vector3 direction = (target - transform.position).normalized;
    transform.rotation = Quaternion.LookRotation(direction); // Rotate towards next

    float distanceToTarget = Vector3.Distance(transform.position, target);
    if (distanceToTarget < stoppingDistance) {
        currentWaypoint++;
        if (currentWaypoint < path.corners.Length){
            target = path.corners[currentWaypoint];
        }
        else{
            StopWalking(); // Reached final destination
```

```
private Collider GetClosestMeleeEnemy(){
    Collider closest = null;
    float closestDistance = float.MaxValue;

    foreach (Collider enemy in nearbyMeleeEnemies){
        float distanceToEnemy = Vector3.Distance(transform.position, enem
        if (distanceToEnemy < closestDistance){
            closest = enemy;
            closestDistance = distanceToEnemy;
        }
    }

    return closest;
}
```

```
1 reference
private Collider GetClosestEnemyInRange(){
    Collider closest = null;
    float closestDistance = float.MaxValue;

    foreach (Collider enemy in nearbyRangeEnemies){
        float distanceToEnemy = Vector3.Distance(transform.position, enemy
        if (distanceToEnemy < closestDistance){
            closest = enemy;
            closestDistance = distanceToEnemy;
        }
    }
```

```
public void PerformMelee(){
    var enemy = GetClosestMeleeEnemy();
    if (NavMesh.CalculatePath(transform.position, enemy.transform.position, NavMesh.All
        if (path.corners.Length > 1){
            currentWaypoint = 0;
            target = path.corners[currentWaypoint]; // Set first waypoint
            StartWalking();
        }
    }
    Vector3 direction = (enemy.transform.position - transform.position).normalized;
    transform.rotation = Quaternion.LookRotation(direction); // Rotate towards next poi

    // Play melee attack sound
    if (audioSource != null && meleeSound != null)
        audioSource.PlayOneShot(meleeSound);

    animator.SetBool("IsMelee", true);
    StartCoroutine(DisableIsMeleeParam());
    RifleAnimsetCharacter enemyChar = enemy.GetComponent<RifleAnimsetCharacter>();

    enemyChar.Die();
    knifeSlashFX.SetActive(true);

    if (enemyChar.isBoss)
        GameManager.Instance.GameWin();
}
```

```
IEnumerator DisableIsMeleeParam(){
    yield return new WaitForSeconds(1f);
    animator.SetBool("IsMelee", false);
}

1 reference
public void PerformRangeAttack(){
    var enemy = GetClosestEnemyInRange();
    if (enemy == null) return;

    Vector3 direction = (enemy.transform.position - transform.position).normalized;
    if (audioSource != null && pistolSound != null)
        audioSource.PlayOneShot(pistolSound);
```

```
178
179         // Instantiate and launch projectile
180         GameObject projectile = Instantiate(projectilePrefab, projectileSpawnPoint.position
181         projectile.GetComponent<Rigidbody>().velocity = direction * 50f; // Adjust speed as
182
183         animator.SetBool("IsShoot", true);
184         animator.SetInteger("WeaponNumber", 2);
185         StartCoroutine(DisableIsRangeAttackParam());
186     }
187
        1 reference
188     IEnumerator DisableIsRangeAttackParam(){
189         yield return new WaitForSeconds(0.5f);
190         animator.SetBool("IsShoot", false);
```

```
193     void StartWalking(){
194         animator.SetFloat("InputMagnitude", 1f);
195         animator.SetFloat("Vertical", 1f);
196         animator.SetBool("IsStopRU", false);
197         animator.SetBool("IsStopLU", false);
198     }
199
        1 reference
200     void StopWalking(){
201         animator.SetFloat("InputMagnitude", 0f);
202         animator.SetFloat("Vertical", 0f);
203         animator.SetBool("IsStopRU", true);
204         animator.SetBool("IsStopLU", true);
205     }
206
        0 references
207     public void UpdateAimRotation(float value){
208         if (rightHandIKTarget != null){
209             rightHandIKTarget.localRotation = Quaternion.Euler(0, 0, value);
210         }
211     }
212
        1 reference
213     void JoystickInputs(){
214         float horizontal = joystick.Horizontal;
215         float vertical = joystick.Vertical;
216
217         // Convert joystick input to world direction
218         Vector3 inputDirection = new Vector3(horizontal, 0, vertical);
```

```
        // Convert joystick input to world direction
        Vector3 inputDirection = new Vector3(horizontal, 0, vertical);
        if (inputDirection.magnitude > 0.1f) {
            // Align input with the camera's forward direction
            Vector3 cameraForward = Camera.main.transform.forward;
            cameraForward.y = 0; // Ignore vertical tilt
            Vector3 cameraRight = Camera.main.transform.right;
            cameraRight.y = 0;

            // Compute movement direction relative to the camera
            Vector3 moveDirection = (cameraRight * horizontal + cameraForward * vertical
            // Set animator parameters
            animator.SetFloat("InputMagnitude", 1);
            animator.SetFloat("Vertical", inputDirection.magnitude);
            animator.SetBool("IsStopRU", false);
            animator.SetBool("IsStopLU", false);

            // Rotate character exactly toward movement direction
            transform.rotation = Quaternion.LookRotation(moveDirection);
        }
        else{
            // Idle state
            animator.SetFloat("InputMagnitude", 0);
            animator.SetFloat("Vertical", 0);
            animator.SetBool("IsStopRU", true);
            animator.SetBool("IsStopLU", true);
        }
    }
```

```
    void PointClickInputs(){
        Rect allowedScreenArea = new Rect(100, 100, Screen.width, Screen.height - 300);
        if (Input.GetMouseButtonDown(0) && allowedScreenArea.Contains(Input.mousePosition)){
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            int layerMask = LayerMask.GetMask("Ground");

            if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)){
                //Debug.Log(hit.transform.name);
                // Debug.Log("DISTANCE" + Vector3.Distance(hit.transform.position, transform.
                if (Vector3.Distance(hit.transform.position, transform.position) < 2.2f){
                    Debug.Log("RETURNNED");
                    return;
                }

                clickEffect.SetActive(true);
                clickEffect.transform.position = hit.point;
                Debug.Log("Hit Position: " + hit.point);
```

```
263
264                 if (NavMesh.CalculatePath(transform.position, hit.point, NavMesh.AllAreas, pa
265                     if (path.corners.Length > 1){
266                         currentWaypoint = 0;
267                         target = path.corners[currentWaypoint];
268                         StartWalking();
269                     }
270                 }
271             }
272         }
273         MoveAlongPath();
```

Enemy AI Script

Assets > Scripts > RifleAnimsetCharacter.cs > RifleAnimsetCharacter

```
5   public class RifleAnimsetCharacter : MonoBehaviour{
        12 references | 5 references | 2 references | 3 references | 1 reference
6       public enum State { Patrol, Chase, Idle, Die }
        7 references
7       public State currentState = State.Patrol;
8
        6 references
9       public Transform player;
        3 references
10      public Transform[] patrolPoints;
        3 references
11      private Transform currentWaypoint;
        4 references
12      public float sightRange = 10f;
        2 references
13      public float chaseSpeed = 1f;
        2 references
14      public float patrolSpeed = 0.5f;
        3 references
15      public float fovAngle = 90f;
        2 references
16      public int fovResolution = 20;
        1 reference
17      public float checkpointIdleTime = 5f; // Time to idle at a ch
18
        8 references
19      private Animator animator;
        3 references
20      private int currentPatrolIndex = 0;
        3 references
21      private bool playerSpotted = false;
```

```
2 references
private bool gameOverTriggered = false;

0 references
public TMP_Text stateText;
1 reference
public GameObject ragdollPrefab;
1 reference
public bool isBoss = false;
6 references
private Mesh fovMesh;
1 reference
public LayerMask obstacleMask; // To detect obstacles
3 references
public MeshFilter meshFilter;

0 references
void Start(){
    animator = GetComponent<Animator>();
    animator.SetBool("IsAware", true);
    if (player == null) player = GameObject.FindGameObjectWithTag("Player").transform;
    currentState = State.Patrol;

    fovMesh = new Mesh();
    if (meshFilter != null) meshFilter.mesh = fovMesh;
}
0 references
void LateUpdate(){
    if (meshFilter != null) DrawFOV();
}
```

```
0 references
void Update(){
    if (!gameOverTriggered && playerSpotted){
        gameOverTriggered = true;
        GameManager.Instance.GameOver();
        Destroy(this);
        return;
    }

    switch (currentState){
        case State.Patrol:
            Patrol();
            break;
        case State.Chase:
            Chase();
            break;
        case State.Idle:
            Idle();
            break;
        case State.Die:
            Die();
            break;
    }

}

1 reference
void Patrol(){
    if (patrolPoints.Length == 0){
        currentState = State.Idle;
```

```
            currentState = State.Idle;
            animator.SetFloat("Speed", 0);
            return;
        }

        animator.SetFloat("Speed", patrolSpeed);
        currentWaypoint = patrolPoints[currentPatrolIndex];
        Vector3 direction = (currentWaypoint.position - transform.position).normalized;

        // Smoothly rotate towards the waypoint using Quaternion.Slerp
        Quaternion targetRotation = Quaternion.LookRotation(direction);
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.de

        if (Vector3.Distance(transform.position, currentWaypoint.position) < 0.5f){
            // Increment and wrap around if needed
            currentPatrolIndex = (currentPatrolIndex + 1) % patrolPoints.Length;
            StartCoroutine(IdleAtCheckpoint()); // Idle at checkpoint for a while
        }

        if (PlayerInSight()){
            currentState = State.Chase;
            animator.SetFloat("Speed", chaseSpeed);
            playerSpotted = true;
            return;
        }
    }

    1 reference
    void Chase(){
```

```
    void Chase(){
        if (!PlayerInSight()){
            currentState = State.Patrol;
            animator.SetFloat("Speed", patrolSpeed);
            return;
        }

        // Smoothly rotate towards the player using Quaternion.Slerp
        Vector3 direction = (player.position - transform.position).normalized;
        Quaternion targetRotation = Quaternion.LookRotation(direction);
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.delt
        animator.SetFloat("Speed", chaseSpeed);
    }

    1 reference
    void Idle(){
        playerSpotted = PlayerInSight();
    }

    3 references
    public void Die(){
        Instantiate(ragdollPrefab, transform.position, transform.rotation);
        Destroy(gameObject);
    }

    3 references
    bool PlayerInSight(){
        float distanceToPlayer = Vector3.Distance(transform.position, player.position);
        if (distanceToPlayer <= sightRange){
            Vector3 directionToPlayer = (player.position - transform.position).normalized;
```

```
124             Vector3 directionToPlayer = (player.position - transform.position).normalized;
125             float angleToPlayer = Vector3.Angle(transform.forward, directionToPlayer);
126 
127             if (angleToPlayer < fovAngle / 2){
128                 if (Physics.Raycast(transform.position, directionToPlayer, out RaycastHit h
129                     if (hit.transform == player){
130                         return true;
131                     }
132                 }
133             }
134         }
135         return false;
136     }
137 
        1 reference
138     void DrawFOV(){
139         // Lists to store local-space vertices and triangle indices.
140         List<Vector3> vertices = new List<Vector3>();
141         List<int> triangles = new List<int>();
142 
143         vertices.Add(Vector3.zero);
144 
145         float stepAngle = fovAngle / fovResolution;
146         for (int i = 0; i <= fovResolution; i++){
147             // Calculate the current angle relative to the forward direction.
148             float angle = -fovAngle / 2 + stepAngle * i;
149 
150             // Compute the world-space direction.
151             // Use the object's forward vector so that the FOV rotates with the character.
152             Vector3 worldDirection = Quaternion.Euler(0, angle, 0) * transform.forward;
```

```
153 
154             // Calculate the default end point in world space.
155             Vector3 worldVertex = transform.position + worldDirection * sightRange;
156 
157             // If an obstacle is hit, update the endpoint.
158             if (Physics.Raycast(transform.position, worldDirection, out RaycastHit hit, sig
159                 worldVertex = hit.point;
160             }
161 
162             // Convert the world-space vertex to local space relative to this transform.
163             Vector3 localVertex = transform.InverseTransformPoint(worldVertex);
164             vertices.Add(localVertex);
165         }
166 
167         // Create triangles for the mesh using a triangle fan.
168         // Starting from the center (vertex 0), each pair of consecutive vertices forms a t
169         for (int i = 1; i < vertices.Count - 1; i++){
170             triangles.Add(0);       // Center of the fan.
171             triangles.Add(i);       // Current boundary point.
172             triangles.Add(i + 1);   // Next boundary point.
173         }
```

```
174
175         // Update the mesh.
176         fovMesh.Clear();
177         fovMesh.vertices = vertices.ToArray();
178         fovMesh.triangles = triangles.ToArray();
179         fovMesh.RecalculateNormals();
180     }
```

```
183     private IEnumerator IdleAtCheckpoint(){
184         currentState = State.Idle;
185         animator.SetFloat("Speed", 0);
186         yield return new WaitForSeconds(checkpointIdleTime); // Wait for the checkpoint
187         currentState = State.Patrol; // Resume patrolling after idle time
188     }
189 }
```

GameManager Script

```
5   public class GameManager : MonoBehaviour{
        18 references
6       public static GameManager Instance { get; private set; }
        1 reference
7       public GameOverPanel gameOverPanel;
        1 reference
8       public GameWinPanel gameWinPanel;
        9 references
9       public InGamePanel inGamePanel;
        5 references
10      public PlayerControllerNew player;
        2 references
11      public GameObject cutscenePanel;
        2 references
12      public Slider aimSlider;
13
        0 references
14      void Awake(){
15          if (Instance == null){
16              Instance = this;
17          }
18          else{
19              Destroy(gameObject);
20              return;
21          }
22
23          inGamePanel.useKnifeButton.onClick.AddListener(player.PerformMelee);
24          inGamePanel.usePistolButton.onClick.AddListener(player.PerformRangeAttack);
25      }
```

```
27      public void GameOver(){
28          Debug.Log("Game Over!");
29          gameOverPanel.gameObject.SetActive(true);
30          inGamePanel.gameObject.SetActive(false);
31      }
32
        1 reference
33      public void GameWin(){
34          Debug.Log("Game Win!");
35          gameWinPanel.gameObject.SetActive(true);
36      }
37
        1 reference
38      public void EnableKnifeButton(){
39          inGamePanel.EnableKnifeButton();
40      }
        1 reference
41      public void DisableKnifeButton(){
42          inGamePanel.DisableKnifeButton();
43      }
44
        1 reference
45      public void EnablePistolButton(){
46          inGamePanel.EnablePistolButton();
47      }
        2 references
48      public void DisablePistolButton(){
49          inGamePanel.DisablePistolButton();
50      }
```

```
52      public void EnableCutscenePanel(){
53          cutscenePanel.SetActive(true);
54      }
        2 references
55      public void DisableCutscenePanel(){
56          cutscenePanel.SetActive(false);
57      }
58
        2 references
59      public void EnableInGameUI(){
60          inGamePanel.gameObject.SetActive(true);
61      }
        1 reference
62      public void DisableInGameUI(){
63          inGamePanel.gameObject.SetActive(false);
64      }
65  }
```

Bullet Script

```
public class bullet : MonoBehaviour
{
    0 references
    void OnCollisionEnter(Collision other){
        if (other.gameObject.CompareTag("Enemy"))
        {
            other.gameObject.GetComponent<RifleAnimsetCharacter>().Die();
            Destroy(gameObject);
        }
    }
}
```

CameraFollow Script

```
public class CameraFollow : MonoBehaviour{
    5 references
    public Transform target;
    1 reference
    public Vector3 offsetVector;

    0 references
    void Start(){
        if (target == null)
            target = GameObject.FindGameObjectWithTag("Player").transform;
    }

    0 references
    void Update(){
        transform.position = target.position + offsetVector;
        transform.LookAt(target);
    }
}
```

Cutscene Script

```
public class CutsceneScript : MonoBehaviour{
    7 references
    public Transform[] targets; // Array of target transforms
    1 reference
    public float switchDelay = 3f; // Delay before switching targe
    5 references
    private int currentTargetIndex = 0;
    3 references
    private CameraFollow cameraFollow;

    0 references
    void Start(){
        if (cameraFollow == null)
            cameraFollow = FindObjectOfType<CameraFollow>();

        if (targets.Length > 0){
            GameManager.Instance.EnableCutscenePanel();
            GameManager.Instance.DisableInGameUI();
            GameManager.Instance.player.enabled = false;
            StartCoroutine(SwitchTargetRoutine());
        }
    }

    1 reference
    IEnumerator SwitchTargetRoutine()
    {
        do{
```

```
        do{
            yield return new WaitForSeconds(switchDelay);
            SetTarget(targets[currentTargetIndex]);
            currentTargetIndex = (currentTargetIndex + 1) % targets.Length;

        } while (currentTargetIndex != targets.Length - 1);
        SetTarget(targets[currentTargetIndex]);
        GameManager.Instance.player.enabled = true;
        GameManager.Instance.DisableCutscenePanel();
        GameManager.Instance.EnableInGameUI();
        Destroy(gameObject);
    }

    3 references
    void SetTarget(Transform target){
        cameraFollow.target = target;
    }

    0 references
    public void SkipCutscene(){
        StopAllCoroutines();
        SetTarget(targets[targets.Length - 1]);
        GameManager.Instance.player.enabled = true;
        GameManager.Instance.DisableCutscenePanel();
        GameManager.Instance.EnableInGameUI();
        Destroy(gameObject);
    }
}
```

LaserCollider Script

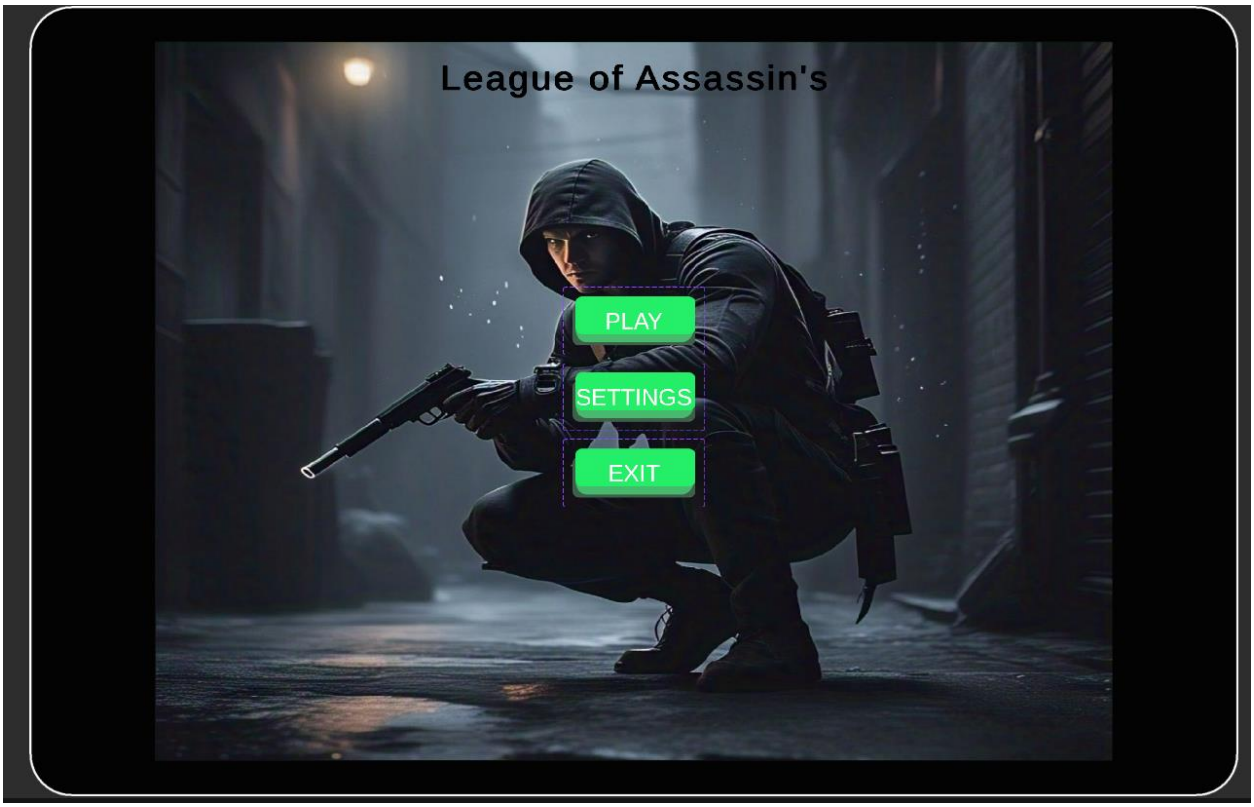
```
0 references
public class LaserCollider : MonoBehaviour
{
    0 references
    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Enemy"))
        {
            Debug.Log("ENEMY ENTER SIGHT");
            GameManager.Instance.EnablePistolButton();

        }
    }
    0 references
    void OnTriggerExit(Collider other)
    {
        if (other.gameObject.CompareTag("Enemy"))
        {
            Debug.Log("ENEMY EXIT SIGHT");
            GameManager.Instance.DisablePistolButton();
        }
    }
}
```

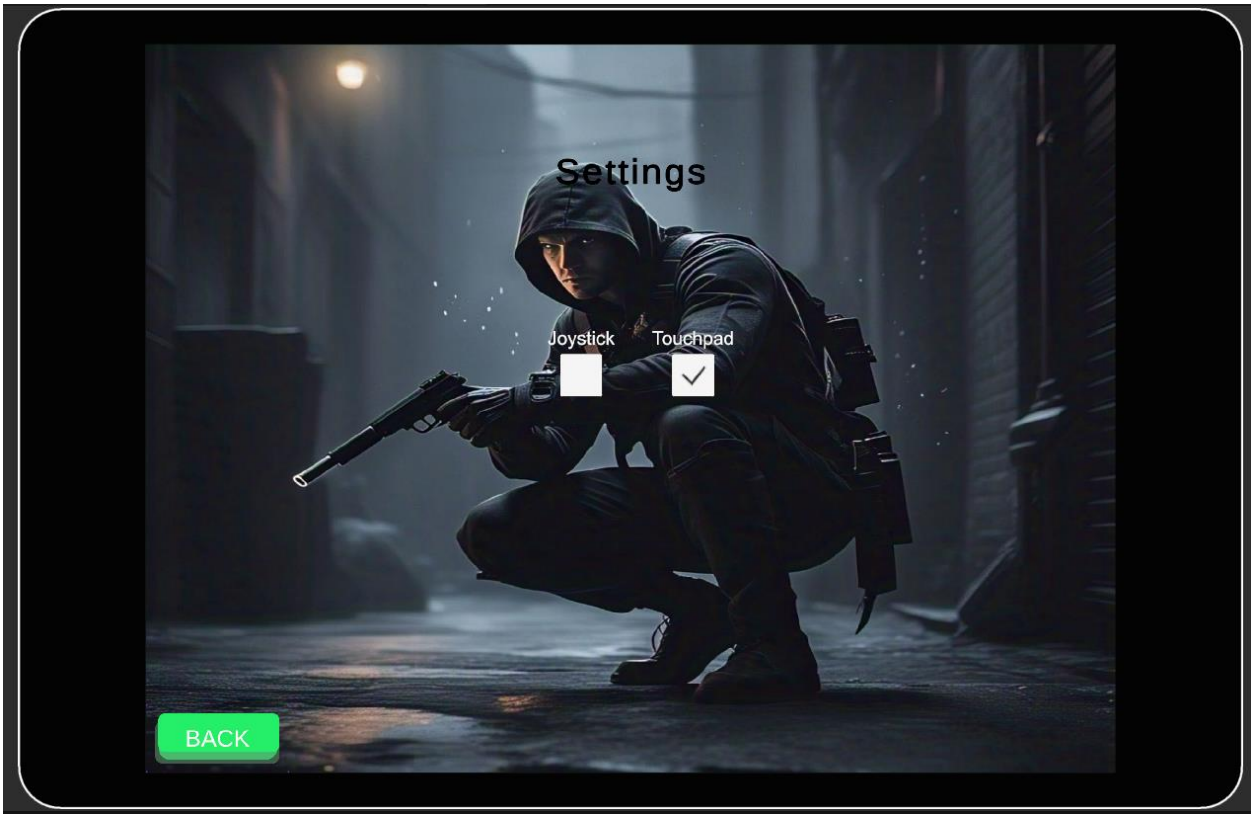
SettingsController Script

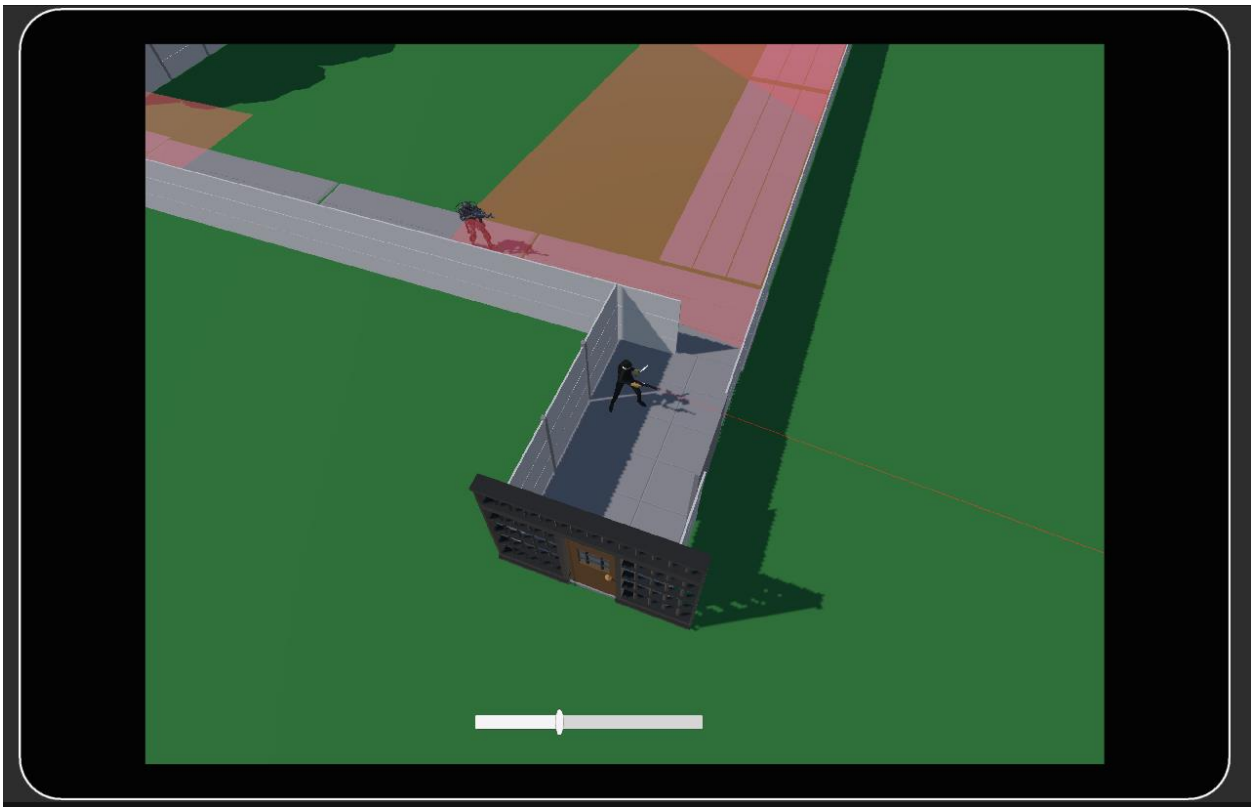
```
public class SettingsController : MonoBehaviour{
    1 reference
    public ToggleGroup toggleGroup;
    2 references
    public Toggle toggleOption1;
    1 reference
    public Toggle toggleOption2;
    2 references
    private string playerPrefKey = "InputsControl";
    1 reference
    public GameObject settingsPanel;
    1 reference
    public GameObject mainMenuPanel;
    1 reference
    public Button backButton;

    0 references
    void Start(){
        LoadToggleState();
        backButton.onClick.AddListener(BackPressed);
    }

    0 references
    public void OnToggleChanged(){
        // Get the active toggle
        Toggle activeToggle = toggleGroup.GetFirstActiveToggle();
        if (activeToggle != null){
            int toggleIndex = activeToggle == toggleOption1 ? 0 : 1;
            PlayerPrefs.SetInt(playerPrefKey, toggleIndex);
```

```
            PlayerPrefs.SetInt(playerPrefKey, toggleIndex);
            PlayerPrefs.Save();
        }
    }

    1 reference
    void LoadToggleState(){
        int savedValue = PlayerPrefs.GetInt(playerPrefKey, 1); // Default to t

        // Set the correct toggle active based on saved value
        if (savedValue == 0)
            toggleOption1.isOn = true;

        else
            toggleOption2.isOn = true;

    }

    1 reference
    void BackPressed(){
        settingsPanel.SetActive(false);
        mainMenuPanel.SetActive(true);
    }
}
```

Game Screenshots








YOU WERE SPOTTED!
You Lost
RESTART
MENU

Topics Covered from the Course

The following topics from the course were covered during the project.

1. Level Designing
2. PlayerPrefs
3. C# Scripting
4. Built-In Unity Methods (OnTriggerEnter, OnTriggerExit etc)
5. Animator Controllers and Animations
6. Input System
7. Unity UI
8. Unity Physics

Conclusion

League of Assassins offers rich and immersive stealth experience, where careful planning, precise execution, and tactical thinking are essential to success. Whether you prefer silent knife kills, discreet gunshots, or non-violent approaches, the game rewards those who can navigate its dangerous world with finesse and precision. Your reputation as an assassin will shape your journey in the League of Assassins, and only the most skilled will achieve the highest fame and respect.