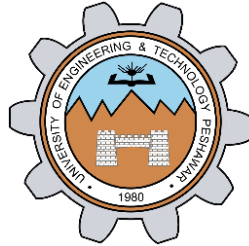


Project Report



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

Introduction.....	4
Gameplay Features	4
Top-Down Gameplay	4
Player Character - The Assassin.....	4
1. Knife.....	4
2. Silenced Gun	4
Stealth and Assassination	5
Detection System.....	5
Level Objectives and Approaches.....	5
1. Stealth Kill (Silent Approach)	5
2. Discreet Gunshots (Silenced Approach)	5
3. Avoid Combat (Non-violent Approach)	5
4. Failed Stealth	6
3D Environments	6
Input Controls	6
Combat Mechanics:	6
Contribution of Each Group Member.....	7
Muhammad Sadeeq's Contribution as a 3D Artist	7
1. Environment Assets.....	7
2. Character Models.....	8
3. Weapons & Items.....	8
Tools & Software Used	9
Challenges & Solutions.....	9
Ali Asghar's Contribution as Main Logic Designer	10
1. Gameplay Mechanics	10
2. AI & Game Systems	10
Challenges & Solutions.....	11

Suleman Shah's Contribution as Level and Game Designer	12
Level Design & Environment Development	12
Challenges & Solutions.....	12
Muhammad Shahab's Contribution as UI Designer	13
User Interface Design	13
Challenges & Solutions.....	13
Code Screenshots	14
PlayerController Script	14
Enemy AI Script.....	19
GameManager Script	23
Bullet Script.....	25
CameraFollow Script	25
CutsScene Script	26
LaserCollider Script	27
SettingsController Script	27
Game Screenshots	29
Topics Covered from the Course.....	31
Conclusion	31

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.

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:

1. 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.

2. 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 and standing guard

Detection System

Enemies have a field of vision. If the player enters this field, 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:

1. 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.

2. 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.

3. 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.

4. 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 takes place in a detailed urban scene that features both exterior and interior spaces. The environment includes intricately designed buildings, furnished interiors, and various props that enhance immersion. Players can navigate through open streets, enter buildings, and utilize the surroundings for strategic gameplay. The level design encourages stealth, offering hiding spots, interactive objects, and multiple ways to approach objectives.

Input Controls

There are two types of controls: Touch-Based (Point & Click) and Joystick.

- **Touch-Based (Point & Click):**

Players can tap on the screen to move their character to the selected location. This control scheme is ideal for a more tactical and calculated playstyle.

- **Joystick:**

A virtual joystick allows for full manual movement, giving players precise control over their character's positioning and actions. The right side of the screen can be used for camera movement, while additional on-screen buttons enable interactions such as crouching, sprinting, and using special abilities. This setup is suited for players who prefer a more hands-on and immersive experience.

Combat Mechanics:

- When a player is near an enemy, they can use a **melee knife** for a silent takedown.
- If the enemy is within the **laser sight** of the pistol, players can take a ranged shot for a quick elimination.

Contribution of Each Group Member

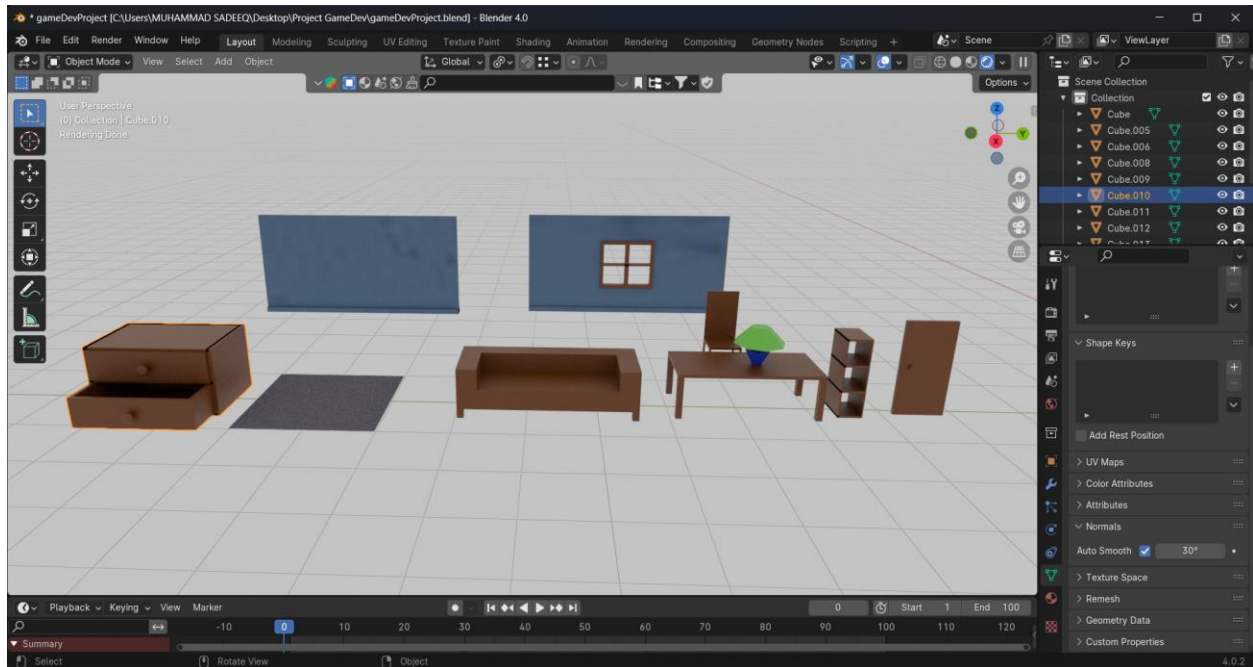
The contribution of each group member is described as follows.

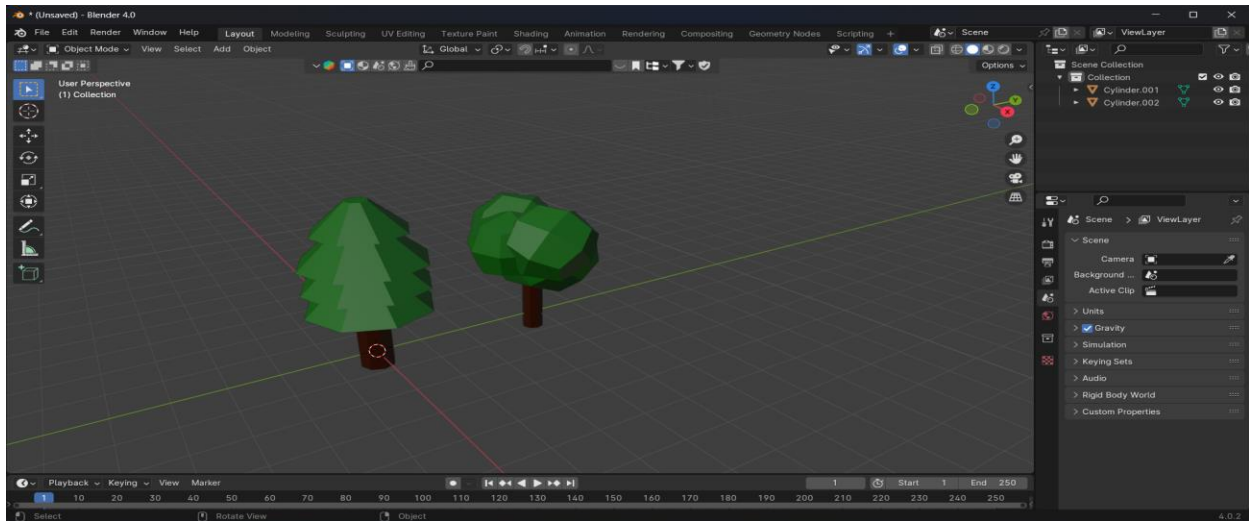
Muhammad Sadeeq's Contribution as a 3D Artist

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

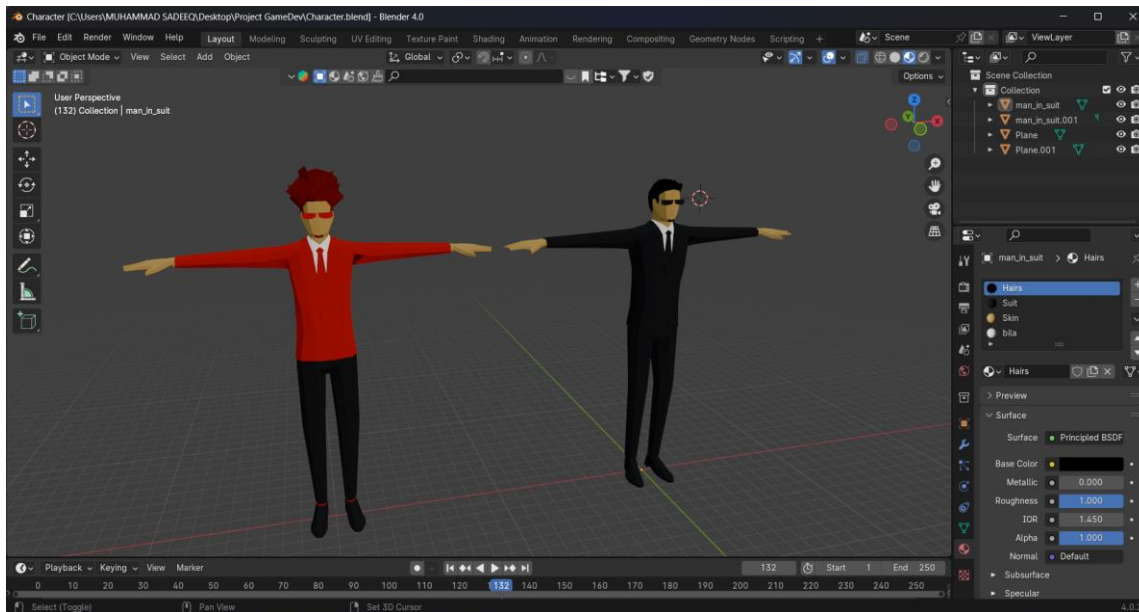
- **Walls:** Designed low-poly walls with textures to create an immersive game world.
- **Props:** Created objects such as table, chairs, doors, and lamp to enhance the game environment.
- **Terrain Features:** Modeled, trees, and grass to add realism to outdoor areas.





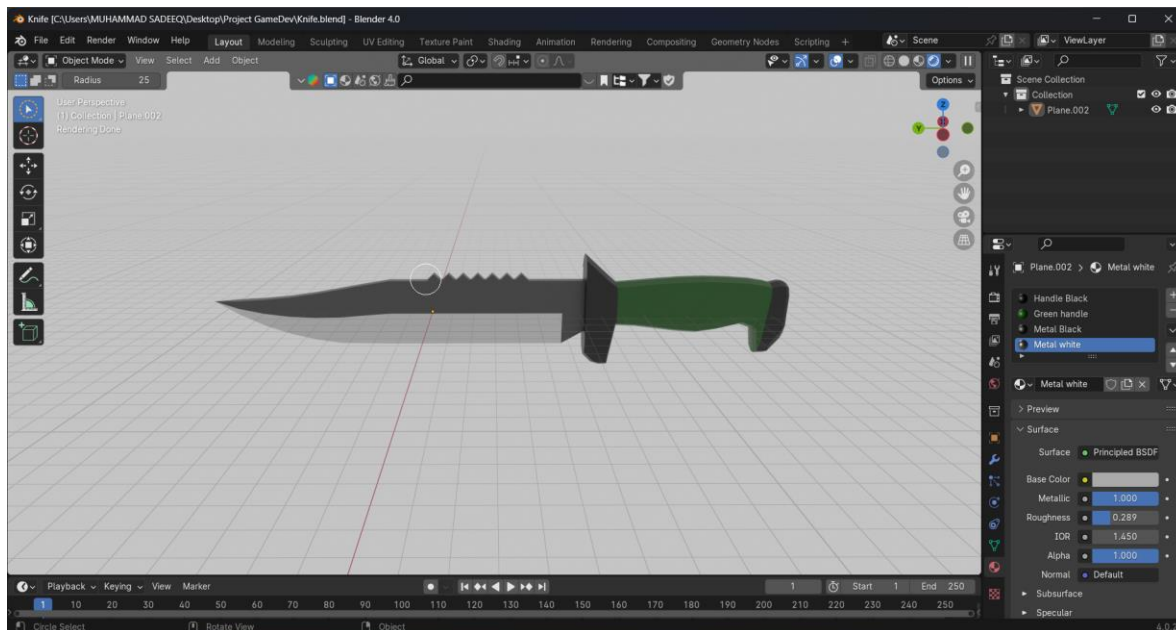
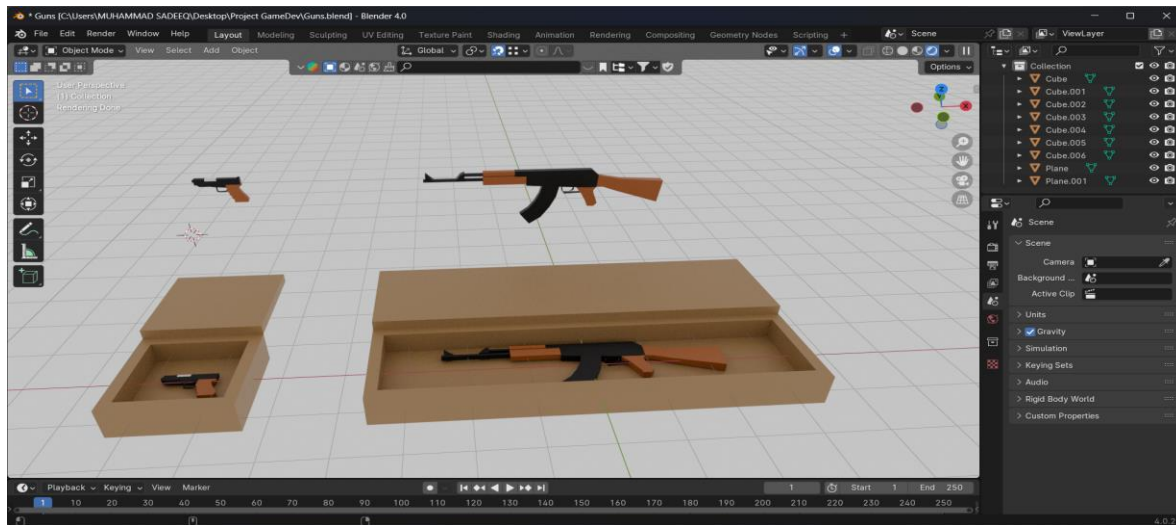
2. Character Models

- **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.

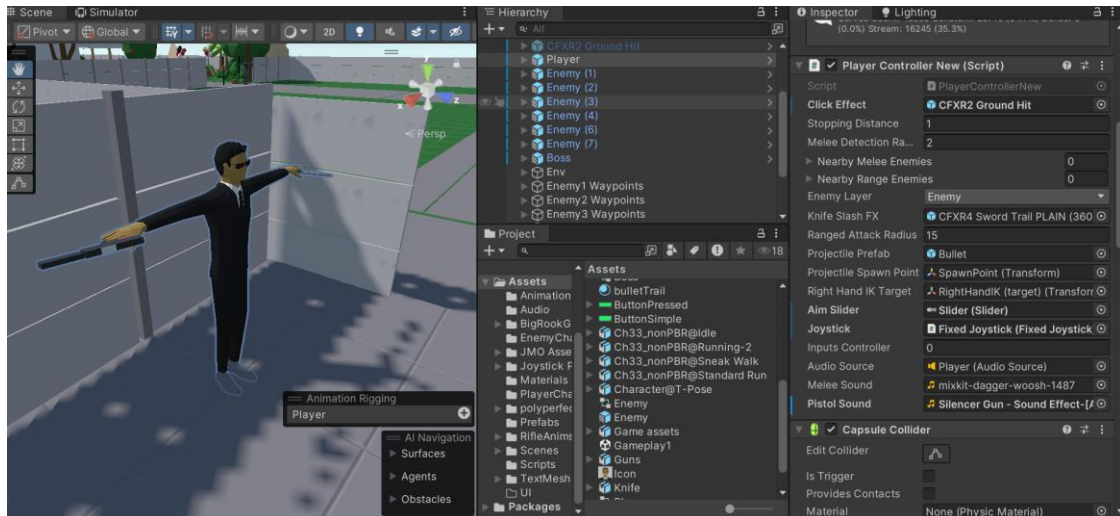
- **Solution:** Ensured correct rigging and exported models in FBX format for seamless Unity integration.

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:

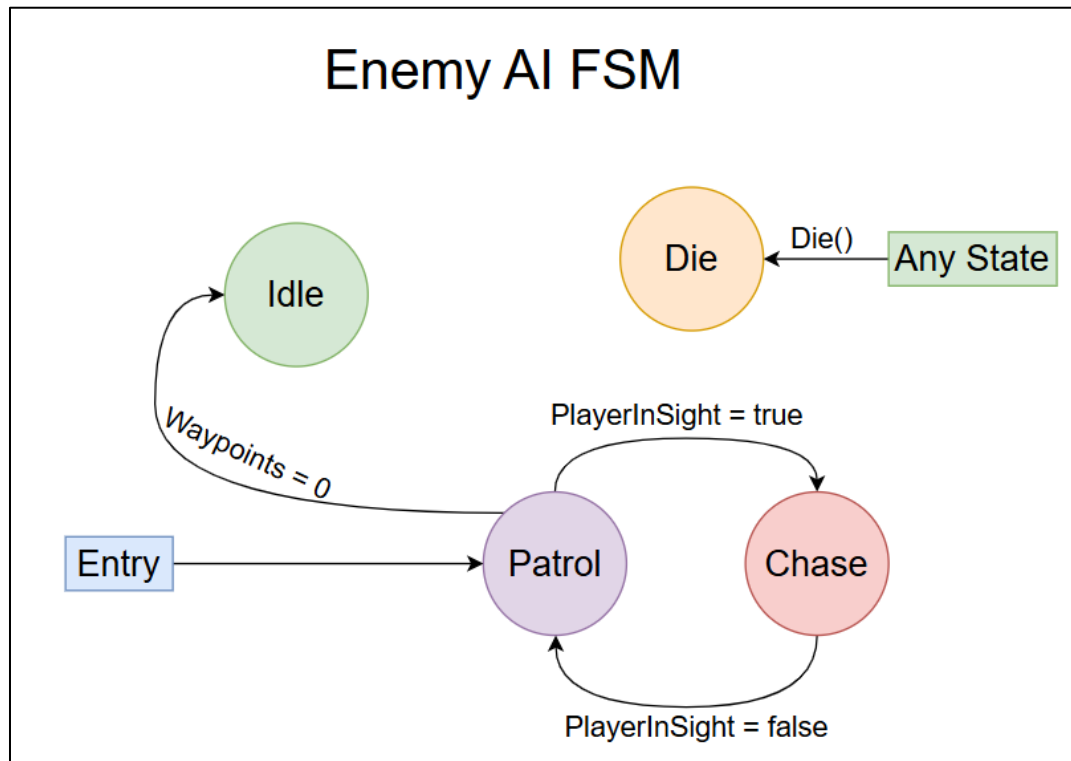
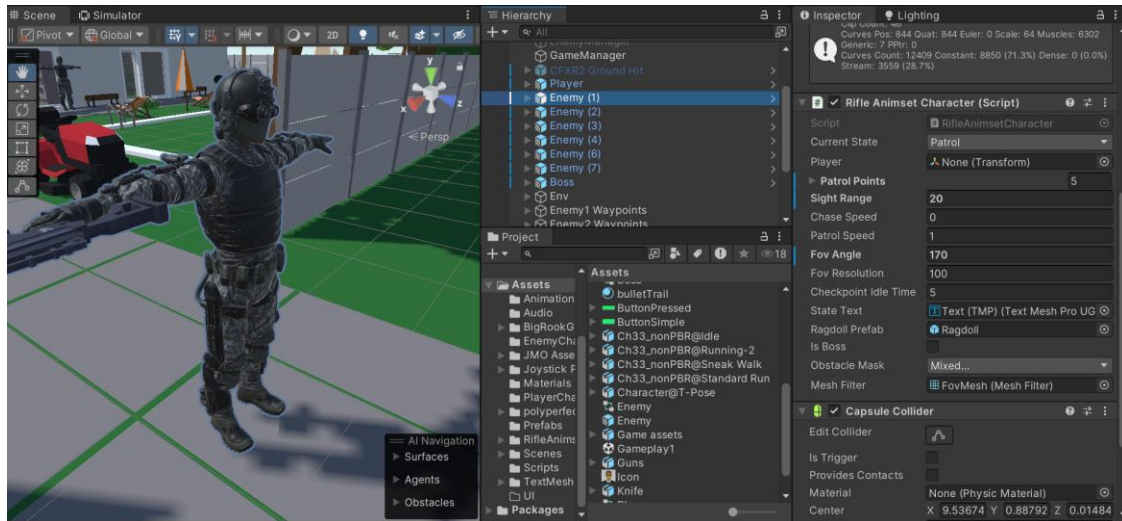
1. Gameplay Mechanics

- **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

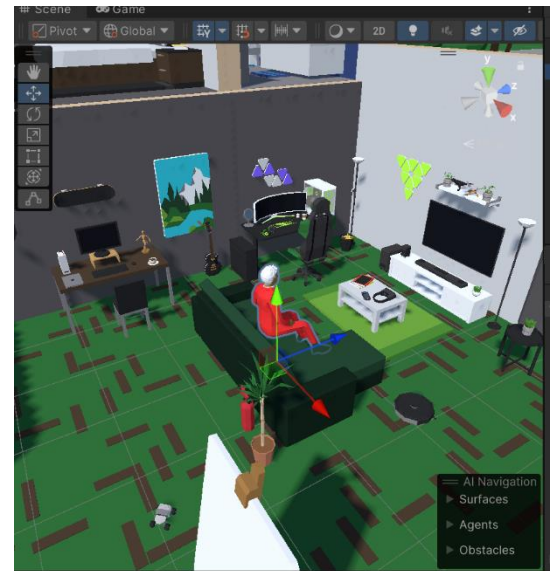
- **Challenge:** AI pathfinding was inefficient in complex environments.
 - **Solution:** Implemented NavMesh for dynamic navigation and obstacle avoidance.
- **Challenge:** Player inputs felt unresponsive.
 - **Solution:** Refined input handling with interpolation and better physics calculations.

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:

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

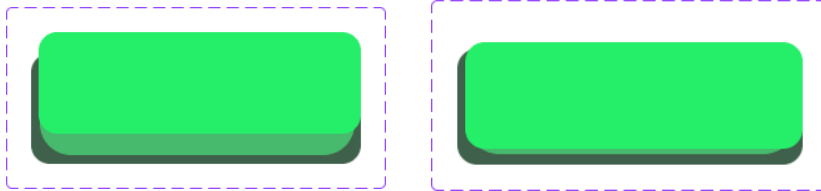
- **Challenge:** Levels felt too linear, reducing player engagement.
 - **Solution:** Introduced multiple pathways and hidden areas to encourage exploration.
- **Challenge:** Players struggled with navigation due to poor visibility of objectives.
 - **Solution:** Enhanced visual cues and lighting to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

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 X
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 X
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 |   }
36 | }
```

```

0 references
37 void Update(){
38
39     inputsController = PlayerPrefs.GetInt("InputsControl", 0);
40
41     if (inputsController == 0){
42         JoystickInputs();
43         joystick.gameObject.SetActive(true);
44
45         joystick.gameObject.SetActive(true);
46     }
47     else{
48         PointClickInputs();
49         joystick.gameObject.SetActive(false);
50     }
51 }
52
1 reference
52 void MoveAlongPath(){
53     if (path.corners.Length == 0 || currentWaypoint >= path.corners.Length){
54         GetComponent<Rigidbody>().freezeRotation = true;
55         return;
56     }
57
58     Vector3 direction = (target - transform.position).normalized;
59     transform.rotation = Quaternion.LookRotation(direction); // Rotate towards next
60
61     float distanceToTarget = Vector3.Distance(transform.position, target);
62     if (distanceToTarget < stoppingDistance) {
63         currentWaypoint++;
64         if (currentWaypoint < path.corners.Length){
65             target = path.corners[currentWaypoint];
66         }
67         else{
68             StopWalking(); // Reached final destination
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109 private Collider GetClosestMeleeEnemy(){
110     Collider closest = null;
111     float closestDistance = float.MaxValue;
112
113     foreach (Collider enemy in nearbyMeleeEnemies){
114         float distanceToEnemy = Vector3.Distance(transform.position, enemy);
115         if (distanceToEnemy < closestDistance){
116             closest = enemy;
117             closestDistance = distanceToEnemy;
118         }
119     }
120
121     return closest;
122 }
123

```



```

124     1 reference
125     private Collider GetClosestEnemyInRange(){
126         Collider closest = null;
127         float closestDistance = float.MaxValue;
128
129         foreach (Collider enemy in nearbyRangeEnemies){
130             float distanceToEnemy = Vector3.Distance(transform.position, enemy.transform.position);
131             if (distanceToEnemy < closestDistance){
132                 closest = enemy;
133                 closestDistance = distanceToEnemy;
134             }
135         }
136     }

```

```

139     public void PerformMelee(){
140         var enemy = GetClosestMeleeEnemy();
141         if (NavMesh.CalculatePath(transform.position, enemy.transform.position, NavMesh.AllAreas) > 1){
142             if (path.corners.Length > 1){
143                 currentWaypoint = 0;
144                 target = path.corners[currentWaypoint]; // Set first waypoint
145                 StartWalking();
146             }
147         }
148         Vector3 direction = (enemy.transform.position - transform.position).normalized;
149         transform.rotation = Quaternion.LookRotation(direction); // Rotate towards next point
150
151         // Play melee attack sound
152         if (audioSource != null && meleeSound != null)
153             audioSource.PlayOneShot(meleeSound);
154
155         animator.SetBool("IsMelee", true);
156         StartCoroutine(DisableIsMeleeParam());
157         RifleAnimsetCharacter enemyChar = enemy.GetComponent<RifleAnimsetCharacter>();
158
159         enemyChar.Die();
160         knifeSlashFX.SetActive(true);
161
162         if (enemyChar.isBoss)
163             GameManager.Instance.GameWin();
164     }

```

```

166     IEnumerator DisableIsMeleeParam(){
167         yield return new WaitForSeconds(1f);
168         animator.SetBool("IsMelee", false);
169     }
170
171     1 reference
172     public void PerformRangeAttack(){
173         var enemy = GetClosestEnemyInRange();
174         if (enemy == null) return;
175
176         Vector3 direction = (enemy.transform.position - transform.position).normalized;
177         if (audioSource != null && pistolSound != null)
178             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);

```

```

217 // Convert joystick input to world direction
218 Vector3 inputDirection = new Vector3(horizontal, 0, vertical);
219 if (inputDirection.magnitude > 0.1f) {
220     // Align input with the camera's forward direction
221     Vector3 cameraForward = Camera.main.transform.forward;
222     cameraForward.y = 0; // Ignore vertical tilt
223     Vector3 cameraRight = Camera.main.transform.right;
224     cameraRight.y = 0;
225
226     // Compute movement direction relative to the camera
227     Vector3 moveDirection = (cameraRight * horizontal + cameraForward * vertical);
228     // Set animator parameters
229     animator.SetFloat("InputMagnitude", 1);
230     animator.SetFloat("Vertical", inputDirection.magnitude);
231     animator.SetBool("IsStopRU", false);
232     animator.SetBool("IsStopLU", false);
233
234     // Rotate character exactly toward movement direction
235     transform.rotation = Quaternion.LookRotation(moveDirection);
236 }
237 else{
238     // Idle state
239     animator.SetFloat("InputMagnitude", 0);
240     animator.SetFloat("Vertical", 0);
241     animator.SetBool("IsStopRU", true);
242     animator.SetBool("IsStopLU", true);
243 }
244 }

```

```

246 void PointClickInputs(){
247     Rect allowedScreenArea = new Rect(100, 100, Screen.width, Screen.height - 300);
248     if (Input.GetMouseButtonDown(0) && allowedScreenArea.Contains(Input.mousePosition)) {
249         Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
250         int layerMask = LayerMask.GetMask("Ground");
251
252         if (Physics.Raycast(ray, out RaycastHit hit, Mathf.Infinity, layerMask)){
253             //Debug.Log(hit.transform.name);
254             // Debug.Log("DISTANCE" + Vector3.Distance(hit.transform.position, transform.
255             if (Vector3.Distance(hit.transform.position, transform.position) < 2.2f){
256                 Debug.Log("RETURNED");
257                 return;
258             }
259
260             clickEffect.SetActive(true);
261             clickEffect.transform.position = hit.point;
262             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     MoveAlongPath();
273

```

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;

```

```

22 | 2 references
    | private bool gameOverTriggered = false;
23 |
    | 0 references
24 | public TMP_Text stateText;
    | 1 reference
25 | public GameObject ragdollPrefab;
    | 1 reference
26 | public bool isBoss = false;
    | 6 references
27 | private Mesh fovMesh;
    | 1 reference
28 | public LayerMask obstacleMask; // To detect obstacles
    | 3 references
29 | public MeshFilter meshFilter;
30 |
    | 0 references
31 | void Start(){
32 |     animator = GetComponent<Animator>();
33 |     animator.SetBool("IsAware", true);
34 |     if (player == null) player = GameObject.FindGameObjectWithTag("Player").transform;
35 |     currentState = State.Patrol;
36 |
37 |     fovMesh = new Mesh();
38 |     if (meshFilter != null) meshFilter.mesh = fovMesh;
39 | }
    | 0 references
40 | void LateUpdate(){
41 |     if (meshFilter != null) DrawFOV();
42 | }

```

```

    | 0 references
44 | void Update(){
45 |     if (!gameOverTriggered && playerSpotted){
46 |         gameOverTriggered = true;
47 |         GameManager.Instance.GameOver();
48 |         Destroy(this);
49 |         return;
50 |     }
51 |
52 |     switch (currentState){
53 |         case State.Patrol:
54 |             Patrol();
55 |             break;
56 |         case State.Chase:
57 |             Chase();
58 |             break;
59 |         case State.Idle:
60 |             Idle();
61 |             break;
62 |         case State.Die:
63 |             Die();
64 |             break;
65 |     }
66 |
67 | }
    | 1 reference
69 | void Patrol(){
70 |     if (patrolPoints.Length == 0){
71 |         currentState = State.Idle;

```

```

71         currentState = State.Idle;
72         animator.SetFloat("Speed", 0);
73         return;
74     }
75
76     animator.SetFloat("Speed", patrolSpeed);
77     currentWaypoint = patrolPoints[currentPatrolIndex];
78     Vector3 direction = (currentWaypoint.position - transform.position).normalized;
79
80     // Smoothly rotate towards the waypoint using Quaternion.Slerp
81     Quaternion targetRotation = Quaternion.LookRotation(direction);
82     transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime);
83
84     if (Vector3.Distance(transform.position, currentWaypoint.position) < 0.5f){
85         // Increment and wrap around if needed
86         currentPatrolIndex = (currentPatrolIndex + 1) % patrolPoints.Length;
87         StartCoroutine(IdleAtCheckpoint()); // Idle at checkpoint for a while
88     }
89
90     if (PlayerInSight()){
91         currentState = State.Chase;
92         animator.SetFloat("Speed", chaseSpeed);
93         playerSpotted = true;
94         return;
95     }
96 }
97
98 1 reference
99 void Chase(){

```

```

98 void Chase(){
99     if (!PlayerInSight()){
100         currentState = State.Patrol;
101         animator.SetFloat("Speed", patrolSpeed);
102         return;
103     }
104
105     // Smoothly rotate towards the player using Quaternion.Slerp
106     Vector3 direction = (player.position - transform.position).normalized;
107     Quaternion targetRotation = Quaternion.LookRotation(direction);
108     transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime);
109     animator.SetFloat("Speed", chaseSpeed);
110 }
111
112 1 reference
113 void Idle(){
114     playerSpotted = PlayerInSight();
115 }
116
117 3 references
118 public void Die(){
119     Instantiate(ragdollPrefab, transform.position, transform.rotation);
120     Destroy(gameObject);
121 }
122
123 3 references
124 bool PlayerInSight(){
125     float distanceToPlayer = Vector3.Distance(transform.position, player.position);
126     if (distanceToPlayer <= sightRange){
127         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{
6         18 references
7         public static GameManager Instance { get; private set; }
8         1 reference
9         public GameOverPanel gameOverPanel;
10        1 reference
11        public GameWinPanel gameWinPanel;
12        9 references
13        public InGamePanel inGamePanel;
14        5 references
15        public PlayerControllerNew player;
16        2 references
17        public GameObject cutscenePanel;
18        2 references
19        public Slider aimSlider;
20
21        0 references
22        void Awake(){
23            if (Instance == null){
24                Instance = this;
25            }
26            else{
27                Destroy(gameObject);
28                return;
29            }
30
31            inGamePanel.useKnifeButton.onClick.AddListener(player.PerformMelee);
32            inGamePanel.usePistolButton.onClick.AddListener(player.PerformRangeAttack);
33        }

```

```

27 public void GameOver(){
28     Debug.Log("Game Over!");
29     gameOverPanel.gameObject.SetActive(true);
30     inGamePanel.gameObject.SetActive(false);
31 }
32
33 1 reference
34 public void GameWin(){
35     Debug.Log("Game Win!");
36     gameWinPanel.gameObject.SetActive(true);
37 }
38
39 1 reference
40 public void EnableKnifeButton(){
41     inGamePanel.EnableKnifeButton();
42 }
43
44 1 reference
45 public void DisableKnifeButton(){
46     inGamePanel.DisableKnifeButton();
47 }
48
49 1 reference
50 public void EnablePistolButton(){
51     inGamePanel.EnablePistolButton();
52 }
53
54 2 references
55 public void DisablePistolButton(){
56     inGamePanel.DisablePistolButton();
57 }
58

```

```

59 1 reference
60 public void EnableCutscenePanel(){
61     cutscenePanel.SetActive(true);
62 }
63
64 2 references
65 public void DisableCutscenePanel(){
66     cutscenePanel.SetActive(false);
67 }
68
69 2 references
70 public void EnableInGameUI(){
71     inGamePanel.gameObject.SetActive(true);
72 }
73
74 1 reference
75 public void DisableInGameUI(){
76     inGamePanel.gameObject.SetActive(false);
77 }
78
79 }

```


Bullet Script

```
5 public class bullet : MonoBehaviour
6 {
7     0 references
8     void OnCollisionEnter(Collision other){
9         if (other.gameObject.CompareTag("Enemy"))
10        {
11            other.gameObject.GetComponent<RifleAnimsetCharacter>().Die();
12            Destroy(gameObject);
13        }
14    }
```

CameraFollow Script

```
5  ✓ public class CameraFollow : MonoBehaviour{
6      5 references
7      public Transform target;
8      1 reference
9      public Vector3 offsetVector;
10
11     0 references
12     void Start(){
13         if (target == null)
14             target = GameObject.FindGameObjectWithTag("Player").transform;
15     }
16
17     0 references
18     void Update(){
19         transform.position = target.position + offsetVector;
20         transform.LookAt(target);
21     }
22 }
```

Cutscene Script

```
4 public class CutsceneScript : MonoBehaviour{
5     7 references
6     public Transform[] targets; // Array of target transforms
7     1 reference
8     public float switchDelay = 3f; // Delay before switching target
9     5 references
10    private int currentTargetIndex = 0;
11    3 references
12    private CameraFollow cameraFollow;
13
14    0 references
15    void Start(){
16        if (cameraFollow == null)
17            cameraFollow = FindObjectOfType<CameraFollow>();
18
19        if (targets.Length > 0){
20            GameManager.Instance.EnableCutscenePanel();
21            GameManager.Instance.DisableInGameUI();
22            GameManager.Instance.player.enabled = false;
23            StartCoroutine(SwitchTargetRoutine());
24        }
25    }
26
27    1 reference
28    IEnumerator SwitchTargetRoutine()
29    {
30        do{
```

```
31        do{
32            yield return new WaitForSeconds(switchDelay);
33            SetTarget(targets[currentTargetIndex]);
34            currentTargetIndex = (currentTargetIndex + 1) % targets.Length;
35        } while (currentTargetIndex != targets.Length - 1);
36        SetTarget(targets[currentTargetIndex]);
37        GameManager.Instance.player.enabled = true;
38        GameManager.Instance.DisableCutscenePanel();
39        GameManager.Instance.EnableInGameUI();
40        Destroy(gameObject);
41    }
42
43    3 references
44    void SetTarget(Transform target){
45        cameraFollow.target = target;
46    }
47
48    0 references
49    public void SkipCutscene(){
50        StopAllCoroutines();
51        SetTarget(targets[targets.Length - 1]);
52        GameManager.Instance.player.enabled = true;
53        GameManager.Instance.DisableCutscenePanel();
54        GameManager.Instance.EnableInGameUI();
55        Destroy(gameObject);
56    }
57 }
```

LaserCollider Script

```
0 references
5 public class LaserCollider : MonoBehaviour
6 {
7     0 references
8     void OnTriggerEnter(Collider other)
9     {
10         if (other.gameObject.CompareTag("Enemy"))
11         {
12             Debug.Log("ENEMY ENTER SIGHT");
13             GameManager.Instance.EnablePistolButton();
14         }
15     }
16     0 references
17     void OnTriggerExit(Collider other)
18     {
19         if (other.gameObject.CompareTag("Enemy"))
20         {
21             Debug.Log("ENEMY EXIT SIGHT");
22             GameManager.Instance.DisablePistolButton();
23         }
24     }
25 }
```

SettingsController Script

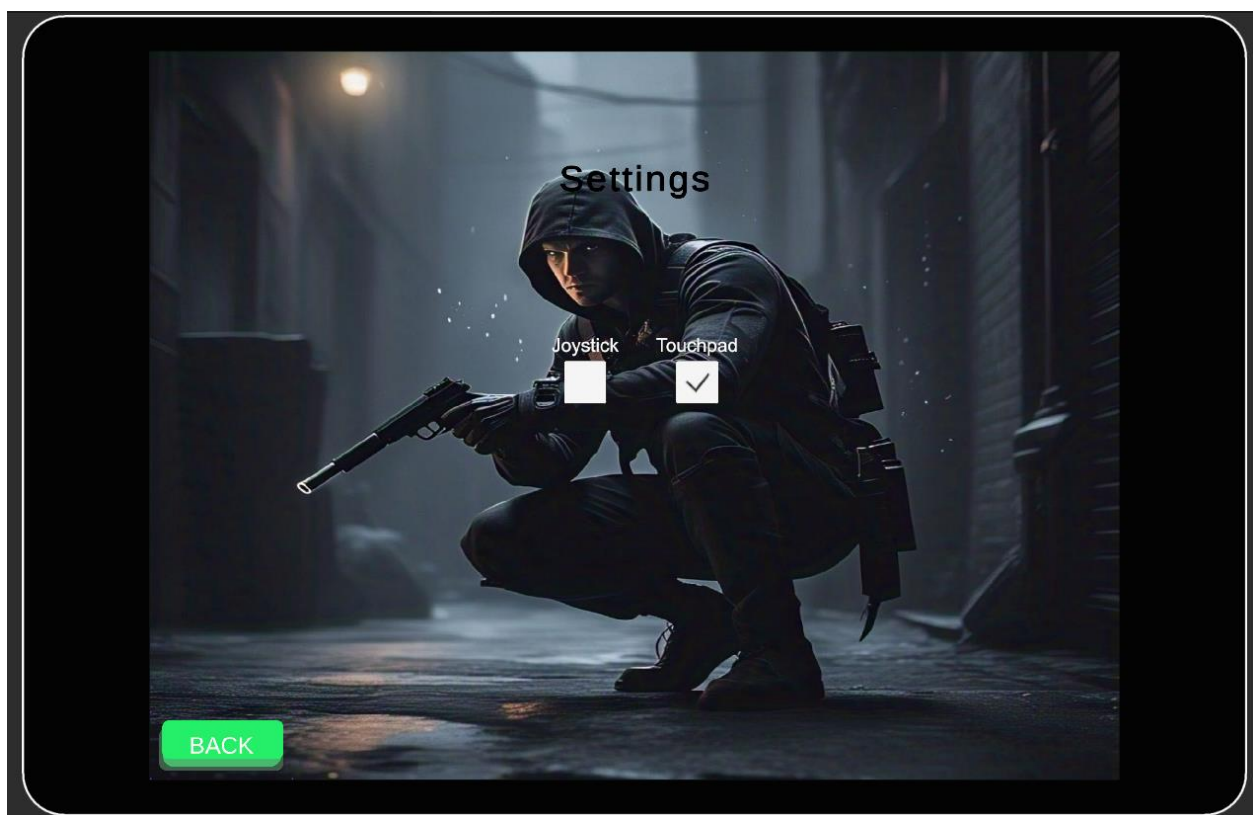
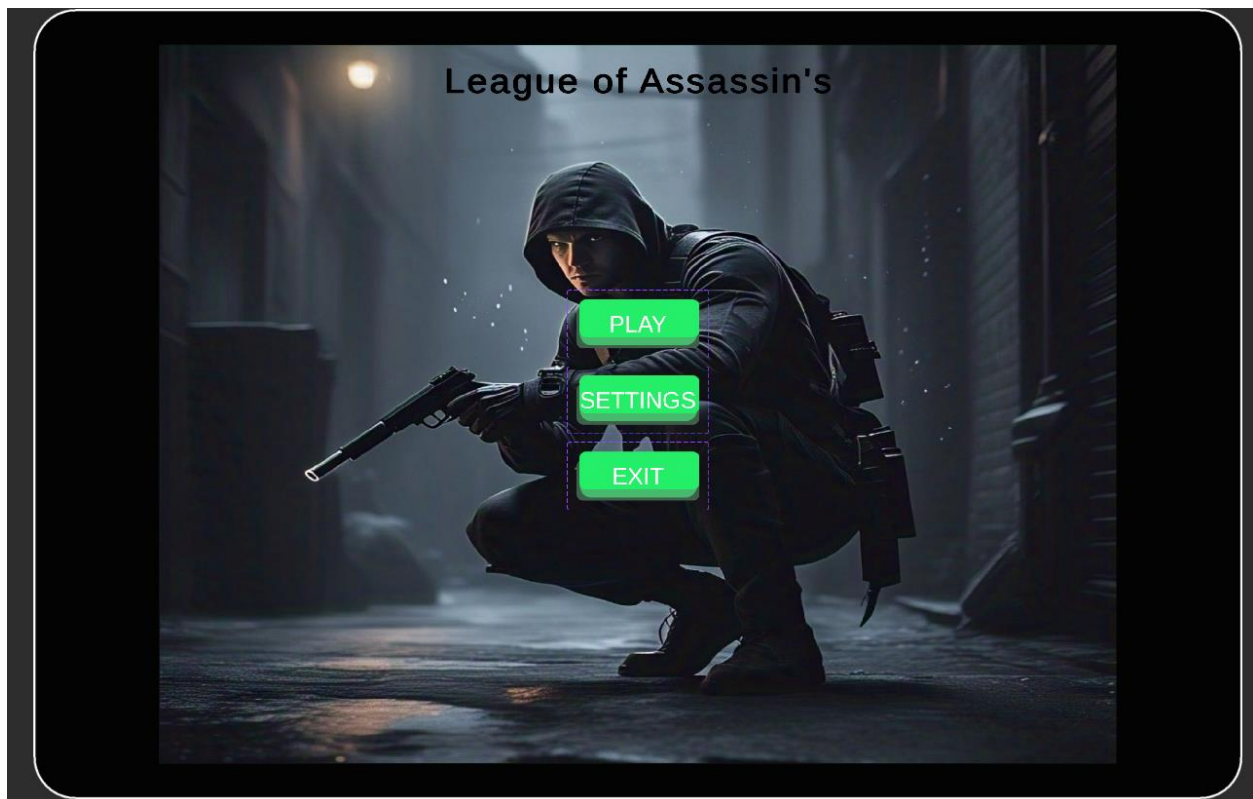
```
0 references
4 public class SettingsController : MonoBehaviour{
5     1 reference
6     public ToggleGroup toggleGroup;
7     2 references
8     public Toggle toggleOption1;
9     1 reference
10    public Toggle toggleOption2;
11    2 references
12    private string playerPrefKey = "InputsControl";
13    1 reference
14    public GameObject settingsPanel;
15    1 reference
16    public GameObject mainMenuPanel;
17    1 reference
18    public Button backButton;
19
20    0 references
21    void Start(){
22        LoadToggleState();
23        backButton.onClick.AddListener(BackPressed);
24    }
25
26    0 references
27    public void OnToggleChanged(){
28        // Get the active toggle
29        Toggle activeToggle = toggleGroup.GetFirstActiveToggle();
30        if (activeToggle != null){
31            int toggleIndex = activeToggle == toggleOption1 ? 0 : 1;
32            PlayerPrefs.SetInt(playerPrefKey, toggleIndex);
33        }
34    }
35 }
```

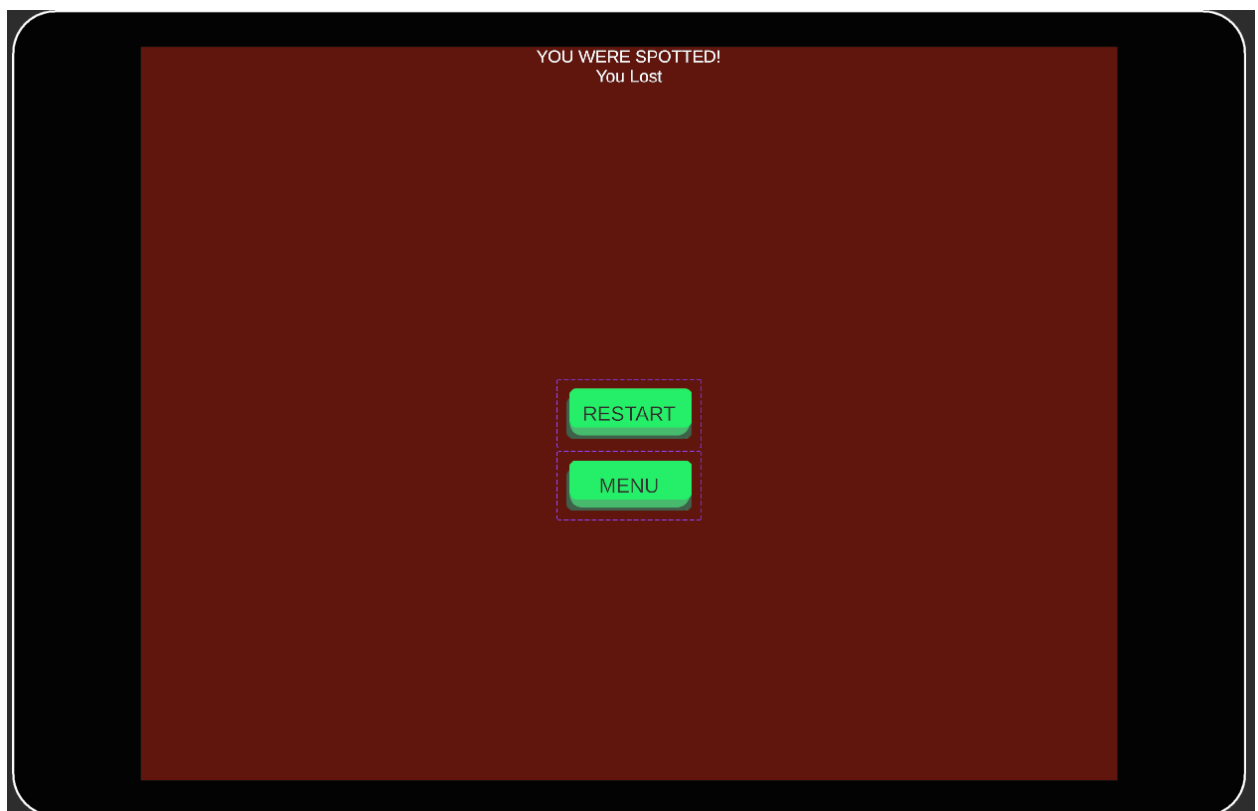
```

23         PlayerPrefs.SetInt(playerPrefKey, toggleIndex);
24         PlayerPrefs.Save();
25     }
26 }
27
28 1 reference
28 void LoadToggleState(){
29     int savedValue = PlayerPrefs.GetInt(playerPrefKey, 1); // Default to t
30
31     // Set the correct toggle active based on saved value
32     if (savedValue == 0)
33         toggleOption1.isOn = true;
34
35     else
36         toggleOption2.isOn = true;
37
38 }
39
40 1 reference
40 void BackPressed(){
41     settingsPanel.SetActive(false);
42     mainMenuPanel.SetActive(true);
43 }
44 }

```

Game Screenshots





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.