A

PROJECT REPORT

ON

URI – A Surgical Strike Game

SUBMITTED BY

HIMANSHU KHAIRNAR	**FS22CO032**
PRITHVIRAJ JADHAV	**FS22CO024**
VAIBHAV SAPAKALE	**FS22CO035**
SAHIL TADAVI	**FS22CO047**
VANSH PAWAR	**FS22CO021**

In The Partial Fulfillment for The Requirements of Diploma in Computer Engineering (Sandwich Pattern)

2024-2025

Under the guidance of

MS. JIJNASA S. PATIL

DEPARTMENT OF COMPUTER ENGINEERING

GOVERNMENT POLYTECHNIC, MUMBAI - 400051

DEPARTMENT OF COMPUTER ENGINEERING (SANDWICH PATTERN)

GOVERNMENT POLYTECHNIC, MUMBAI

MUMBAI – 400051

CERTIFICATE

This is to certify that following students have successfully and satisfactorily completed the project on **URI – A Surgical Strike Game** presented their report in the partial fulfillment of requirement for Diploma in Computer Engineering from Government Polytechnic, Mumbai under the guidance of Ms. Jijnasa S. Patil in the academic year 2023-2024.

HIMANSHU KHAIRNAR	**FS22CO032**
PRITHVIRAJ JADHAV	**FS22CO024**
VAIBHAV SAPAKALE	**FS22CO035**
SAHIL TADAVI	**FS22CO047**
VANSH PAWAR	**FS22CO021**

Project Guide

External Examiner

Head of Department

Principal

ACKNOWLEDGEMENT

This project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the journey of our project. All that we have done is only due to such supervision and assistance and we would not forget to thank them.

We would first and foremost like to thank our principal Dr. N. N. Mhala and HOD Mr.Anat Dhulshete, for supporting our project idea from the beginning and motivating us to work on our project without any worries. We owe our deep gratitude to our Guide, Ms. Jijnasa S. Patil, who helped us on our project work and guided us all along, by providing all the necessary information for developing a good system.

We respect her for providing us an opportunity to do the project work in Government Polytechnic Mumbai and giving us support and guidance which made us complete the project duly. We are extremely thankful to her for providing such a nice support and guidance.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all teaching staff of Computer Department, GPM who helped us in successfully progressing with our project work.

INDEX

TABLE INDEX

FIGURE INDEX

ABSTRACT

"URI" is a 3D third-person shooter (TPS) game that immerses players in a high-intensity military mission against terrorist forces. Designed to offer an adrenaline-pumping experience, *"URI"* combines tactical objectives, realistic combat environments, and suspenseful scenarios to keep players on the edge of their seats.

The game leverages cutting-edge technology to create a fully immersive battlefield where players must navigate challenging terrain, evade drones, and strategically eliminate enemies. High-quality 3D graphics and immersive sound design enhance each mission, from the faint hum of drones overhead to the tactical silence of enemy ambushes, aiming to heighten the player's sense of urgency and immersion.

Players are tasked with completing a series of six strategic objectives: collecting a key to unlock the weapons warehouse, disabling a computer system, shutting down a generator, securing a time bomb, destroying a second weapons warehouse, and finally, escaping via helicopter. Each objective builds on the next, creating a seamless flow of tension and accomplishment as players progress through the mission.

URI executes its tactical encounters with precision, using visual and auditory cues to keep players alert and engaged. The game aims to offer more than just intense action; it seeks to provide a realistic military experience, rewarding players who think strategically and act decisively in hostile environments.

URI is designed to push the boundaries of tactical shooter gaming, delivering a unique and memorable experience for players who seek the thrill of realistic military missions and strategic objectives in a high-stakes environment.

In summary, *"URI"* offers an intense, immersive experience combining tactical combat with high-stakes objectives. Players are challenged to complete key mission tasks while navigating hostile environments filled with danger. The game emphasizes strategic thinking and precise action, providing a thrilling and engaging adventure. With its compelling gameplay and realistic settings, *"URI"* promises to deliver an unforgettable gaming experience.

CHAPTER 1

INTRODUCTION

Introduction:

The gaming industry has evolved significantly over the years, with players constantly seeking fresh and exhilarating experiences. One genre that consistently captivates audiences is the tactical shooter, known for its adrenaline-fueled action and strategic depth. In response to the growing demand for realistic and engaging military shooters, we introduce *"URI"*, a 3D third-person shooter game that combines high-intensity combat with immersive mission objectives in a hostile environment.

1.1 Objectives:

The *"URI"* project is driven by a set of key objectives aimed at creating a groundbreaking tactical shooter experience that pushes the boundaries of realism and player engagement:

1.1.1 Tactical Combat: The core aim of *"URI"* is to immerse players in intense, combat-driven gameplay where every decision can determine survival. By integrating realistic shooting mechanics and enemy AI, players will face authentic combat scenarios that require strategy, precision, and quick thinking..

1.1.2 Immersive Environments: *"URI"* seeks to create a vivid and hostile world where players navigate through dangerous terrains. From urban settings to terrorist-controlled facilities, the environments are designed with attention to detail, providing a fully immersive backdrop for the military operations that unfold within the game.

1.1.3 Dynamic Gameplay: The gameplay of *"URI"* will adapt to the player's choices, offering dynamic missions that respond to their strategies. Whether stealthily completing objectives or engaging in all-out combat, the game will provide multiple approaches to each mission, ensuring a fresh and engaging experience every time..

1.1.4 Strategic Objectives: The central objectives of *"URI"*—from disabling systems to escaping via helicopter—are crafted to provide both tactical depth and narrative progression. Players will need to approach each objective with careful planning and execution, making each mission more challenging and rewarding.

1.1.5 Realistic Audio and Visual Design: In *"URI"*, sound and visuals play an integral role in creating an atmosphere of tension and immersion. From the sounds of approaching drones to the visual cues signaling enemy presence, the game uses audio and graphics to heighten the realism of every encounter.

This project report will delve into the methods by which these objectives will be realized, highlighting the technical, creative, and gameplay elements that make *"URI"* a unique and exciting tactical shooter experience.

CHAPTER 2
Feasibility Study

Feasibility Study

A feasibility study is an assessment conducted to determine the practicality and potential success of a project or business venture by evaluating various factors such as financial, technical, and operational aspects. It helps stakeholders make informed decisions about the project's viability.

2.1 Project Scope

The scope of *"URI"* revolves around the creation of a 3D third-person shooter game where players, assuming the role of a military soldier, must complete a series of high-stakes objectives while combating terrorist threats. The game will feature dynamic environments, realistic combat scenarios, and strategic gameplay elements, offering a mix of action and narrative-driven tasks.

To ensure the project's success, we plan to utilize a range of development tools and resources. The game will be developed using the Unity game engine, which offers a comprehensive suite of features for creating 3D games. We will make use of assets from the Unity Asset Store to enhance visual quality and speed up the development process, ensuring that the game meets its visual and gameplay goals within the available budget and time frame. Additionally, custom assets will be created to meet specific gameplay needs, ensuring a unique gaming experience.

2.2 Development Environment

The development environment for *"URI"* will leverage cutting-edge technology and industry-standard tools to create an immersive and engaging game experience.

Unity Game Engine: Utilizing the Unity game engine provides us with a solid foundation for game development. It offers a user-friendly interface and extensive resources for 3D game creation.

Unity Asset Store: By accessing the Unity Asset Store for free models, we not only streamline development but also gain access to a vast library of assets that enhance the game's visual quality. This approach significantly accelerates the development process, making it feasible to meet project timelines.

Freesound.org: We will source sound effects from Freesound.org, ensuring high-quality audio components while adhering to budget constraints.

Mega Cloud Hosting: Collaborative development across teams will be made efficient and cost-effective through Mega cloud hosting. This approach facilitates real-time collaboration, version control, and project management, enabling our diverse team to work together seamlessly.

Resource Allocation: Resource allocation will be managed judiciously, ensuring that financial and human resources are effectively distributed to support the project's objectives.

2.3 Existing System

In developing *"URI"*, we have drawn inspiration from several existing games and films known for their engaging tactical shooter gameplay and military themes. These references have provided valuable insights into creating a dynamic, high-pressure gaming experience.

1. URI: The Surgical Strike (Movie):

The film *URI: The Surgical Strike*, directed by Aditya Dhar, is a patriotic military action movie based on real-life events. It portrays the Indian Army's surgical strikes on terrorist camps in Pakistan, highlighting themes of national security, military precision, and courage under fire. The movie's realistic depiction of military strategy, the intensity of combat operations, and the emotional drive behind the soldiers' mission has heavily influenced the design of *"URI"*. We aim to mirror the tactical execution and high-stakes drama of the film in our game, where the player will take on the role of a military soldier carrying out critical operations against terrorist threats. The sense of urgency and heroism portrayed in *URI: The Surgical Strike* is integral to the atmosphere and narrative of our game.

2. Call of Duty: Modern Warfare (Game):

The *Call of Duty: Modern Warfare* series has set high standards for realistic military combat and immersive narratives. The game's attention to detail in combat realism, weapon mechanics, and mission structure has inspired key aspects of *"URI"*. We've studied the game's pacing and the way it balances intense action with narrative elements, ensuring that *"URI"* offers a gripping and action-packed experience that appeals to fans of tactical shooters.of our project.

CHAPTER 3
Project Planning

Project Planning

Project planning is part of project management, which uses schedules and subsequently report progress within the project environment. Project planning can be done manually. The project plan clearly defines how the project is created, monitored and completed.

3.1 SCOPE AND SCHEDULE

Now, let's delve into the project's potential, its scale, and the expected timeframe for its completion. To achieve this, we will establish a well-structured timeline, specifying the start date and the projected completion date.

SCHEDULING OF THE PROJECT GIVEN BELOW:

Date	**Day**	**Subject**
11/07/2024	Thursday	Introduction to Third Year Project
16/07/2024	Tuesday	Group Selection
25/07/2024	Thursday	Guide Allocation
19/08/2024	Monday	Project selected
26/08/2024	Monday	Discussion with guide about project
29/08/2024	Thursday	Planning Story For the game
20/09/2024	Friday	Designing Environments and level
05/10/2024	Saturday	Implementing Functionalities in the game
15/10/2024	Tuesday	Designing User Interface
30/10/2024	Wednesday	Game tested successfully
07/11/2024	Thursday	Exported The Final Build

Table 3.1

3.2 Resources and Roles

1. Unity:

Unity is a powerful and popular cross-platform game development engine and framework. It's used to create 2D, 3D, augmented reality (AR), and virtual reality (VR) applications. Unity allows developers to build interactive experiences for various platforms, including PC, mobile, consoles, and more.

2. Unity Engine:

The Unity engine is the core software that provides the framework for creating games and interactive applications. It includes a wide range of features, such as graphics rendering, physics simulation, audio, scripting, and more. Unity is known for its user-friendly interface and the ability to export projects to multiple platforms.

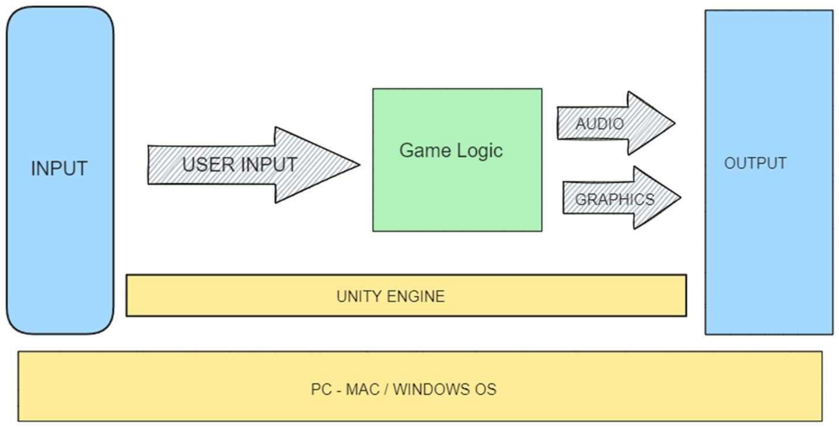


Figure 3.1

3. Unity Hub:

Unity Hub is a management tool that simplifies the process of working with multiple Unity projects and versions. It allows developers to create, open, and organize projects, install different versions of the Unity engine, and manage assets and packages. Unity Hub streamlines development workflows by providing a centralized hub for all Unity-related tasks.

4. C# (C Sharp):

C# is a high-level, object-oriented programming language developed by Microsoft. It's widely used in Unity for scripting game logic and creating interactive elements. C# is known for its ease of use, strong typing, and extensive libraries, making it a popular choice for Unity game development.

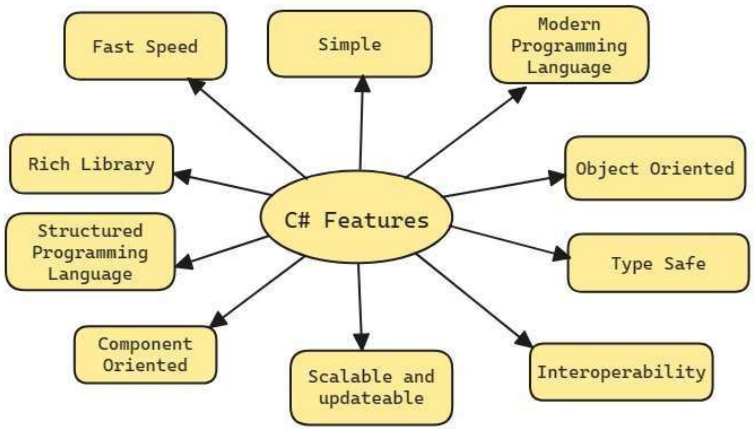


Figure 3.2

5. **Visual Studio:**

Microsoft Visual Studio is an integrated development environment (IDE) developed by Microsoft. It is used primarily for developing computer programs, web applications, and mobile apps. Visual Studio provides a range of features and tools for various programming languages, including C#, C++, F#, Visual Basic, and more.

3.3 Change in Planning

Change management was critical to adapt to evolving circumstances. As the project progressed, changes in scope, technology, or resources arose. We were committed to handle changes efficiently by evaluating their impact on the project timeline, costs, and objectives. We regularly reviewed and adapted to user feedback to ensure that "Insanity" remained aligned with the dynamic nature of the gaming industry and player expectations. Our iterative development approach was designed to incorporate changes seamlessly, providing flexibility without compromising project quality.

CHAPTER 4
Requirement Analysis

Requirement Analysis

"URI – A Surgical Strike Game" stands out from others by offering meticulously designed environments, an emotionally resonant narrative, and dynamic gameplay that keeps players engaged and on edge, creating an unparalleled horror experience that lingers long after the game is over.

The requirement analysis phase of "URI – A Surgical Strike Game" involves a thorough examination of both functional and non-functional requirements to ensure the successful development of the game.

Requirement Specifications:

- **Functional Requirements**
- **Non-Functional Requirements**

4.1 Functional Requirements

Functional requirements define specific features and interfaces crucial for the game's functionality:

User Interface (UI): The UI in "URI" must provide players with an intuitive and visually appealing means of navigating the game. It includes menus for game options, character customization, and inventory management. The UI should be aesthetically consistent with the horror theme, ensuring that players can easily access and adjust game settings, enhancing the overall user experience.

Hardware Interface: The game efficiently utilizes hardware interfaces to ensure smooth performance on a variety of hardware configurations. Optimization for different screen sizes, resolutions, and input devices, such as keyboard and mouse provided an optimal gaming experience for all players. Compatibility with a range of hardware setups was maintained to maximize accessibility.

Software Interfaces: "URI" seamlessly integrates with the Unity game engine, Unity Asset Store, and Freesound.org for asset utilization. This involves smooth asset importing, management, and integration to streamline the development process. Additionally, the game is compatible with popular gaming platforms, including PC, gaming consoles enabling easy distribution to a wide audience of players while adhering to platform-specific requirements and guidelines.

User Experience (UX): User experience encompasses the holistic interaction between players and the game. Beyond the UI and hardware interface, UX in "URI" emphasizes the emotional and psychological aspects of gameplay. It is about creating a sense of fear, tension, and immersion. The UX includes the design of dynamic horror elements, player choices, and the overall atmosphere that keeps players engaged, scared, and excited. It also considers responsiveness, efficient navigation, and an optimal balance between challenge and enjoyment to ensure a deeply engaging and satisfying experience for players.

4.2 Non-Functional Requirements

Non-functional requirements ensure the quality and performance of "Insanity":

Performance: The game runs smoothly and efficiently on various hardware configurations, including lower-end and high-end systems, ensuring an optimal player experience without significant lag, frame drops, or performance issues. High-quality 3D graphics and audio do not compromise performance.

Scalability: The game's architecture allows for potential updates and expansions, ensuring scalability for future improvements. The development team can add new content, features, and levels without causing a significant impact on the game's performance or stability, allowing for long-term growth and player engagement.

Maintainability: Maintenance and updates are manageable and cost-effective. The game is designed with clean and well-documented code, facilitating the efficient resolution of issues and the addition of new content or features. This keeps the game fresh and engaging for players while minimizing the development team's workload in the long term. Regular updates and bug fixes are conducted seamlessly to ensure a consistently high-quality player experience.

CHAPTER 5
System Design

System Design

Methodology

The methodology employed in the development of "Insanity - A 3D Horror Game" is structured to ensure efficient project management, asset utilization, and quality control. The following elements comprise the methodology:

- **Agile Development:** We have adopted an Agile development approach, emphasizing iterative cycles and frequent collaboration between team members. This methodology allows for flexibility, facilitating the accommodation of changing requirements and the integration of user feedback into the development process.

- **Collaboration Through Cloud Tools:** To enable seamless collaboration across geographically dispersed teams, we use cloud-based tools like Mega. This ensures real-time communication, version control, and centralized project management. It has proven to be instrumental in enhancing team coordination and productivity.

- **Quality Assurance:** Quality assurance and testing are integral to our methodology. We employ a rigorous testing process that includes functional, performance, and compatibility testing. This phase ensures that the game meets defined requirements, runs smoothly, and provides an engaging player experience.

- **User Feedback Integration:** Throughout the development process, we actively gather and integrate user feedback. This iterative approach ensures that the game aligns with player expectations and evolving industry standards. Player feedback guides refinements and improvements in subsequent development cycles.

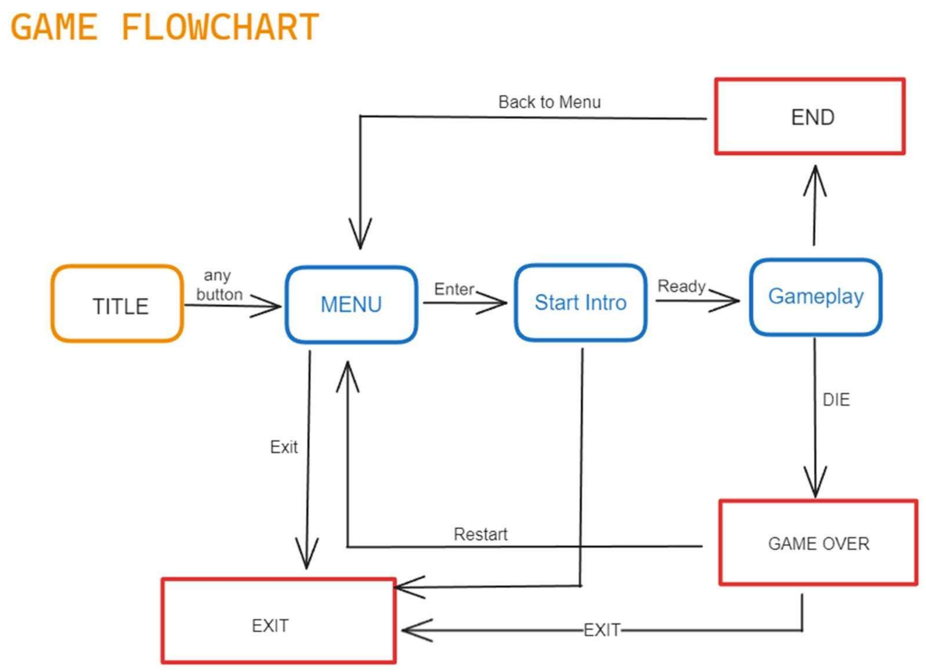
- **Resource Allocation:** Resource allocation is managed judiciously, ensuring that financial and human resources are effectively distributed to support project objectives. This approach enables us to optimize costs while maintaining the desired quality.

Game Control

- W – Forward
- A – Left
- S – Backward
- D – Right
- N – Objective Menu
- F – Pick Up items
- Q – Shut Down Systems
- E – Drop items
- Space – Jump
- Esc - Menu

Game Flow

Understanding the flow of the game is critical to delivering a suspenseful and engaging player experience:



Basic Game Model Flowchart:

Figure 5.1

This flowchart is a schematic representation of the game's timeline. Once the player starts the game, he/she will be directed to the Title screen. By pressing any button there, then the player will be directed to the Menu screen.

Here the player can start the game or quit. If the player chooses to play, then he will be shown the intro scene. Once the intro scene is over, then the gameplay starts. If the player dies during the game, then he will be redirected to the menu screen. If the player succeeds in defeating the terrorist, then it will end the game and the desired ending is shown. Then he will be directed to the menu screen again, where he can quit or play the game again.

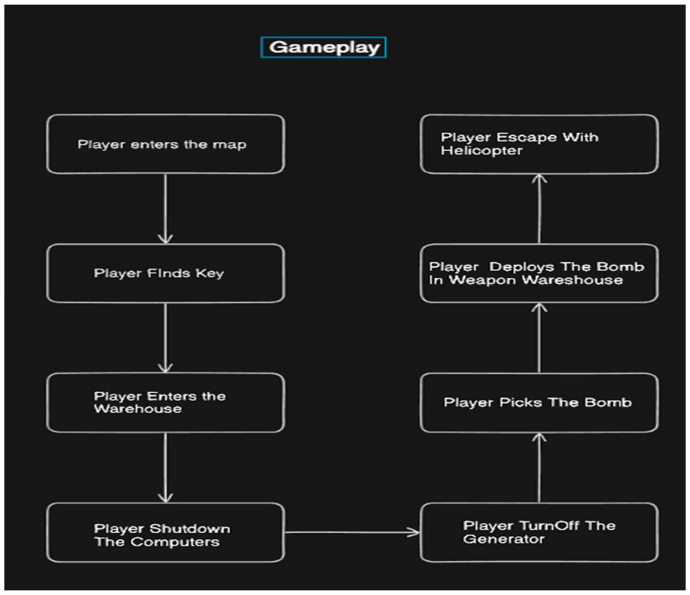

Gameplay
Player enters the map
Player Finds Key
Player Enters the Warehouse
Player Shutdown The Computers
Player TurnOff The Generator
Player Picks The Bomb
Player Deploys The Bomb In Weapon Wareshouse
Player Escape With Helicopter

Figure 5.2

The gameplay starts by the player entering the main map. The player has to explore the area in search of the keys of warehouse. As it the first Objective of the , it is still an important mission and objective for the player. After finding the key player has to find warehouse and open the door . Once the player encounters the enemy and drone, then it has to kill the enemy whilst completing six objectives. The last objective is to escape the area with helicopter

If the player cannot successfully complete the game or gets kill by the enemies , then it would lead to the defeat of the player thus ending the game prematurely.
Once the player completes the objectives dolls, the game ends the game.

ENEMY AI

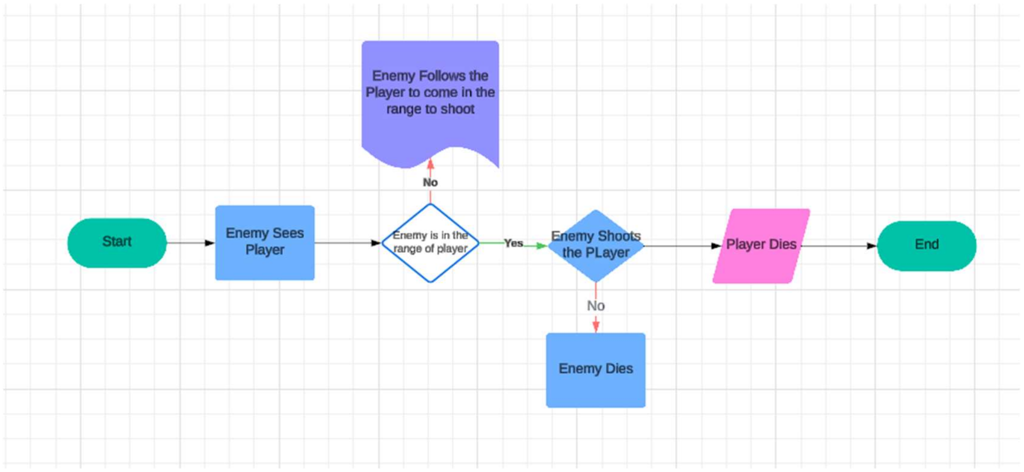

Enemy Follows the Player to come in the range to shoot
No
Start
Enemy Sees Player
Enemy is in the range of player
Yes
Enemy Shoots the PLayer
Player Dies
End
No
Enemy Dies

Figure 5.3

This is a flowchart for depicting the implementation of enemy ai.
If the enemy sees the player, (i.e. their colliders intersect) then the system will check if the player is in range of the enemy to start shooting. If the player is not in range, then he has limited time to move out of its range before it starts chasing and shooting.
If the player is in range, then the ghost starts to shoot the player for some time. If the player gets hit multiple times in total during that time, then the player dies.
If the player hit the enemy and the enemy health decreases to zero then enemy dies .

SOUND

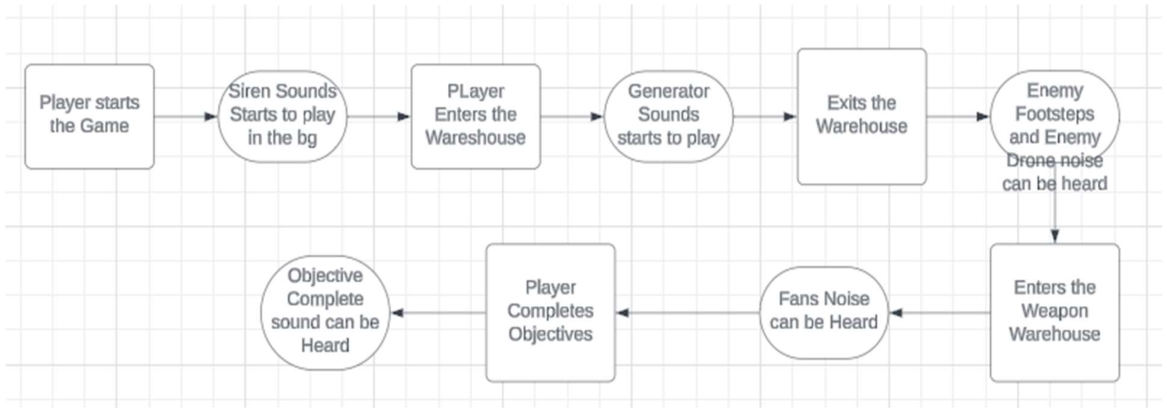

Player starts the Game
Siren Sounds Starts to play in the bg
PLayer Enters the Wareshouse
Generator Sounds starts to play
Exits the Warehouse
Enemy Footsteps and Enemy Drone noise can be heard
Enters the Weapon Warehouse
Fans Noise can be Heard
Player Completes Objectives
Objective Complete sound can be Heard

Figure 5.4

This is a flowchart depicting the sound system in the game.

Once the player starts the game, then the default siren environmental sound will start playing. After entering the warehouse, then generator sound will start playing. And during Enemy battle enemy gun footstep sound will start playing. This will be more intense and heart thumping music. Along with the music, there will be different sound effects. If the player completes the objective then objective completion sound will start to play. And Player footsteps and rifle sound will also start to play if player starts to run, walk and shoot respectively.

CHAPTER 6
Coding

Source Code (C# Scripts)

PlayerScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UIElements;
public class PlayerScript : MonoBehaviour
{
   [Header("Player Movement")]
   public float playerSpeed = 1.9f;
   public float playerSprint = 3f;
   public GameObject prefab;
   [Header("Player Health Things")]
   private float playerHealth = 300f;
   private float presentHealth;
   public HealthScripy healthScripy;
   public AudioSource audioSource;
   [Header("Player Script Cameras")]
   public Transform playerCamera;
   public GameObject deathCamera;
   public GameObject EndGameMenuUI;
   [Header("Player Animator and Gravity")]
   public CharacterController
characterController;
   public float gravity = -9.81f;
   public Animator animator;
   [Header("Player Jumping and Velocity")]
   float turnCalmVelocity;
   public float turnCalmTime = 0.1f;
   public float jumpRange = 1f;
   Vector3 velocity;
   public Transform surfaceCheck;
   public bool onSurface;
   public float surfaceDistance = .4f;
   public LayerMask surfaceMask;
   [Header("Aiming")]
   private bool isAiming;
   private bool hasRotated = false;
   private Quaternion originalRotation;
   private void Start()
   {
      UnityEngine.Cursor.lockState =
CursorLockMode.Locked;
      presentHealth = playerHealth;

healthScripy.GetfullHealth(playerHealth);
      originalRotation =
prefab.transform.rotation;
   }
   private void Update()
   {
   onSurface =
Physics.CheckSphere(surfaceCheck.position,
surfaceDistance, surfaceMask);

   if (onSurface && velocity.y < 0)
   {
      velocity.y = -2f;
   }
   velocity.y += gravity * Time.deltaTime;
   characterController.Move(velocity * Time.deltaTime);
   if(Input.GetButton("Fire1") || Input.GetButton("Fire2"))
   {
      isAiming = true;
   }
   else
   {
      isAiming = false;
   }
   if (isAiming )
   {
      AimAtCenterOfScreen();
      if (!hasRotated)
      {
         prefab.transform.rotation = Quaternion.Euler(0,
30f, 0) * prefab.transform.rotation;
         hasRotated = true;
      }
      Debug.Log("Is Aiming: " + isAiming);
   }
   else
   {
      Debug.Log("Is not    Aiming: " + isAiming);
      if (hasRotated)
      {
         Debug.Log(originalRotation);
         prefab.transform.rotation = Quaternion.Euler(0, -
30f, 0) * prefab.transform.rotation;
         hasRotated = false;
         Debug.Log("p"+prefab.transform.rotation);
      }
      playerMove();
      Jump();
      Sprint();
   }
 }
```

```
void AimAtCenterOfScreen()
   {
      animator.SetBool("AimWalk", true);
      animator.SetBool("Idle", false);
      Vector3 aimDirection =
playerCamera.forward;
      aimDirection.y = 0;
      Quaternion lookRotation =
Quaternion.LookRotation(aimDirection);
      transform.rotation =
Quaternion.Slerp(transform.rotation,
lookRotation, Time.deltaTime * 5f);
      animator.SetBool("AimWalk", true);
      animator.SetBool("Idle", false);
   }
   void playerMove()
   {
      float horizontal_axis =
Input.GetAxisRaw("Horizontal");
      float vertical_axis =
Input.GetAxisRaw("Vertical");
      Vector3 direction = new
Vector3(horizontal_axis, 0f,
vertical_axis).normalized;
      if (direction.magnitude >= 0.1f)
      {
         animator.SetBool("Idle", false);
         animator.SetBool("Walk", true);
         animator.SetBool("Running", false);
         animator.SetBool("AimWalk", false);
         animator.SetBool("IdleAim", false);
         animator.SetBool("Fire", false);
         float targetAngle =
Mathf.Atan2(direction.x, direction.z) *
Mathf.Rad2Deg +
playerCamera.eulerAngles.y;
         float angle =
Mathf.SmoothDampAngle(transform.eulerAn
gles.y, targetAngle, ref turnCalmVelocity,
turnCalmTime);
         transform.rotation =
Quaternion.Euler(0f, angle, 0f);
         Vector3 moveDirection =
Quaternion.Euler(0f, targetAngle, 0f) *
Vector3.forward;

characterController.Move(moveDirection.nor
malized * playerSpeed * Time.deltaTime);
      }
      else
      {
         animator.SetBool("Idle", true);
         animator.SetBool("Walk", false);
      }
   }
   void Sprint()
   {
      if (Input.GetButton("Sprint") &&
Input.GetKey(KeyCode.W) ||
Input.GetKey(KeyCode.UpArrow) && onSurface)
      {
         float horizontal_axis =
Input.GetAxisRaw("Horizontal");
         float vertical_axis = Input.GetAxisRaw("Vertical");
         Vector3 direction = new Vector3(horizontal_axis, 0f,
vertical_axis).normalized;
         if (direction.magnitude >= 0.1f)
         {
            animator.SetBool("Walk", false);
            animator.SetBool("Running", true);
            animator.SetBool("IdleAim", false);
            animator.SetBool("Idle", false);
            float targetAngle = Mathf.Atan2(direction.x,
direction.z) * Mathf.Rad2Deg +
playerCamera.eulerAngles.y;
            float angle =
Mathf.SmoothDampAngle(transform.eulerAngles.y,
targetAngle, ref turnCalmVelocity, turnCalmTime);
            transform.rotation = Quaternion.Euler(0f, angle,
0f);
            Vector3 moveDirection = Quaternion.Euler(0f,
targetAngle, 0f) * Vector3.forward;

characterController.Move(moveDirection.normalized *
playerSprint * Time.deltaTime);
         }
         else
         {
            animator.SetBool("Walk", false);
            animator.SetBool("Idle", false);
         }
      }
      else
      {
         animator.SetBool("Running", false);
      }
   }
   void Jump()
   {
      if (Input.GetButtonDown("Jump") && onSurface)
{
        animator.SetTrigger("Jump");
            animator.SetBool("Idle",
false);
            velocity.y =
Mathf.Sqrt(jumpRange * -2 *
gravity);
        }
        else
        {

animator.ResetTrigger("Jump");
            animator.SetBool("Idle",
true);
        }
    }

    public void playerHitDamage(float
takeDamage)
    {
        presentHealth -= takeDamage;

healthScripy.SetHealth(presentHealth
);
        if (presentHealth <= 0)
        {
            playerDie();
        }
    }
    private void playerDie()
    {

EndGameMenuUI.SetActive(true);
        UnityEngine.Cursor.lockState =
CursorLockMode.Locked;
        deathCamera.SetActive(true);
        UnityEngine.Cursor.lockState =
CursorLockMode.None;
        Object.Destroy(gameObject,1f);
    }
}
```

Rifle.cs

```
using System;
using System.Collections;
using UnityEditor;
using UnityEngine;

public class Rifle : MonoBehaviour
{
    [Header("Rifle Settings")]
    public Camera cam;
    public Animator animator;
    public float damage = 10f;
    public float range = 100f;
    public float fireRate = 15f;
    private float nextTimeToShoot = 0f;
    public PlayerScript player;
    public Transform hand;
    [Header("Ammunition Settings")]
    public int maxAmmo = 32;
    public int mags = 10;
    private int currentAmmo;
    public float reloadTime = 2f;
    private bool isReloading = false;
    [Header("Rifle Effects")]
    public GameObject impactEffect;
    public GameObject goreEffect;
    public GameObject droneEffect;
    [Header("Sound and UI")]
    [SerializeField] private GameObject AmmoOutUi;
    [SerializeField] private int timeToShowUI = 1;
    public AudioClip shootingSound;
    public AudioClip reloadingSound;
    public AudioSource audioSource;
    public ParticleSystem muzzleFlash;
    private void Awake()
    {
        transform.SetParent(hand);
        currentAmmo = maxAmmo;
    }
    private void OnEnable()
    {
        isReloading = false;
        animator.SetBool("Reloading", false);
    }
    private void Update()
    {
        if (isReloading) return;
        if (currentAmmo <= 0 && mags > 0)
        {
```

```
StartCoroutine(Reload());
            return;
        }
        if (Input.GetButton("Fire1") &&
Time.time >= nextTimeToShoot)
        {
            if (Input.GetKey(KeyCode.W) ||
Input.GetKey(KeyCode.UpArrow))
            {
                // Fire while walking
                animator.SetBool("Idle", false);
                animator.SetBool("FireWalk",
true);
            }
            else
            {
                animator.SetBool("Fire", true);
            }
            nextTimeToShoot = Time.time + 1f /
fireRate;
            Shoot();
        }
        else
        {
            ResetFireAnimation();
        }
    }
    private void Shoot()
    {
        if (mags == 0)
        {
            StartCoroutine(ShowAmmoOut());
            return;
        }
        if (currentAmmo <= 0)
        {
            Debug.Log("Out of ammo");
            return;
        }

        currentAmmo--;
        if(currentAmmo ==0)
        {
            mags--;
        }

AmmoCount.occurence.UpdateAmmoText(c
urrentAmmo);
            AmmoCount.occurence.UpdateMagText(mags);
                muzzleFlash.Play();
        audioSource.PlayOneShot(shootingSound);
        if (Physics.Raycast(cam.transform.position,
cam.transform.forward, out RaycastHit hit, range))
        {
            Enemy enemy =
hit.transform.GetComponent<Enemy>();
            EnemyDrone enemyDrone =
hit.transform.GetComponent<EnemyDrone>();
            Debug.Log(hit.transform.name);
            ObjectToHit target =
hit.transform.GetComponent<ObjectToHit>();
            target?.ObjectHitDamage(damage);
            if (target != null)
            {
                target.ObjectHitDamage(damage);
                GameObject impactGO = Instantiate(impactEffect,
hit.point, Quaternion.LookRotation(hit.normal));
                Destroy(impactGO, 1f);
            }
            else if (enemy != null)
            {
                enemy.enemyHitDamage(damage);
                GameObject impactGO = Instantiate(goreEffect,
hit.point, Quaternion.LookRotation(hit.normal));
                Destroy(impactGO, 2f);

            }
            else if (enemyDrone != null)
            {
                enemyDrone.enemyHitDamage(damage);
                GameObject impactGO = Instantiate(droneEffect,
hit.point, Quaternion.LookRotation(hit.normal));
                Destroy(impactGO, 2f);
            }
        }
    }
    private void ResetFireAnimation()
    {
        animator.SetBool("Fire", false);
        animator.SetBool("FireWalk", false);
    }
    IEnumerator Reload()
    {
        isReloading = true;
        SetPlayerMovement(0f, 0f);
        animator.SetBool("Reloading", true);
        audioSource.PlayOneShot(reloadingSound);
        yield return new WaitForSeconds(reloadTime);
```

```
currentAmmo = maxAmmo;
SetPlayerMovement(1.9f, 3f);
  isReloading = false;
    animator.SetBool("Reloading", false);
  }
 private void SetPlayerMovement(float speed,
float sprint)
   {
      player.playerSpeed = speed;
      player.playerSprint = sprint;
   }
   IEnumerator ShowAmmoOut()
   {
      AmmoOutUi.SetActive(true);
      yield return new
WaitForSeconds(timeToShowUI);
      AmmoOutUi.SetActive(false);
   }
}
```

Computer.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.Build;
using UnityEngine;

public class Computer : MonoBehaviour
{
   [Header("Computer On/Off")]
   public bool lightsOn = true;
   private float radius = 3f;
   public Light ligths;

   [Header("Computer Assign Things")]
   public PlayerScript Player;
   [SerializeField] private GameObject
ComputerUI;
   [SerializeField] private int
showComputerUIfor = 5;

   [Header("Sounds")]
   public AudioClip
objectiveCompletedSound;
public AudioSource audioSource;
   private void Awake()
   {
      ligths = GetComponent<Light>();
   }

   private void Update()
   {
      if(Vector3.Distance(transform.position,Player.tra
nsform.position)   < radius)
      {
         Debug.Log("Player is within radius.");
         if (Input.GetKeyDown(KeyCode.Q))
         {
            Debug.Log("Q key pressed.");

            StartCoroutine(ShowComputerUI());
            lightsOn = false;
            ligths.intensity = 0;
            ObjectivesCompleted.occurence.GetObjecti
ves(true, true, false, false,false,false);
            //sound effect
            audioSource.PlayOneShot(objectiveComple
tedSound);
         }
      }
   }

   IEnumerator ShowComputerUI()
   {
      ComputerUI.SetActive(true);
      yield return new
WaitForSeconds(showComputerUIfor);
      ComputerUI.SetActive(false);
   }
}
```

GeneratorTurnOff.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GeneratorTurnOff :
MonoBehaviour
{
   [Header("Generator Lights and
button")]
   public GameObject greenight;
   public GameObject redLight;
   public bool button;

   [Header("Generator Sound
Effects and Radius")]
   private float radius = 2f;
   public PlayerScript player;
   public Animator anim;
   public AudioSource
audioSource;
   [Header("Sounds")]
   public AudioClip
objectiveCompletedSound;


   private void Awake()
   {       button= false;
      audioSource =
GetComponent<AudioSource>();
   }

   private void Update()
   {
      if
(Input.GetKeyDown(KeyCode.Q)
&&
Vector3.Distance(transform.positio
n, player.transform.position) <
radius)
      {
         button = true;
         anim.enabled = false;
         greenight.SetActive(false);
         redLight.SetActive(true);
         audioSource.Stop();
         audioSource.PlayOneShot(objectiveComplete
dSound);
            ObjectivesCompleted.occurence.GetObjecti
ves(true, true, true, false,false,false);
      }
      else if (button==false)
      {
         greenight.SetActive(true);
         redLight.SetActive(false);
      }
   }
}
```

HealthScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HealthScripy : MonoBehaviour
{
   public Slider healthsliderbar;

   public void GetfullHealth(float health)
   {
      healthsliderbar.maxValue = health;
      healthsliderbar.value = health;
   }

   public void SetHealth(float health)
   {
      healthsliderbar.value = health;
   }

}
```

MainMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
public void OnPlayButton()
   {
      SceneManager.LoadScene("Scene_A");
```

```
 }
       public void onQuitButton()
       {
          Application.Quit();
          Debug.Log("Quiting
Game...");
       }
    }
```
Menu.cs
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using
UnityEngine.SceneManagement;

public class Menus :
MonoBehaviour
{
   [Header("All Menu's")]
   public GameObject
pauseMenuUi;
   public GameObject
EndGameMenuUi;
   public GameObject
ObjectiveMenuUI;

   public static bool
GameIsStopped = false;

   private void Update()
   {
      if
(Input.GetKeyUp(KeyCode.Escape
))
      {
         if (GameIsStopped)
         {
            Resume();
            Cursor.lockState =
CursorLockMode.Locked;
         }
         else
         {
            Pause();
            Cursor.lockState =
CursorLockMode.None;
         }
      }
      else if
(Input.GetKeyDown(KeyCode.N))
      {
         if (GameIsStopped)
         {
            removeObjectives();
         }
         else
         {
            showObjectives();
            Cursor.lockState = CursorLockMode.None
         }
      }
   }

   public void showObjectives()
   {
      ObjectiveMenuUI.SetActive(true );
      Time.timeScale = 0f;
      GameIsStopped = true;
   }

   public void removeObjectives()
   {
      ObjectiveMenuUI.SetActive(false );
      Time.timeScale = 1f;
      Cursor.lockState = CursorLockMode.Locked ;
      GameIsStopped = false;
   }
   public void Resume()
   {
      pauseMenuUi.SetActive(false);
      Time.timeScale = 1f;
      Cursor.lockState = CursorLockMode.Locked;
      GameIsStopped = false;
   }
   public void Restart()
   {
      SceneManager.LoadScene("Scene_A");
   }

   public void LoadMenu()
   {
      Debug.Log("Loading Main Menu...");
      Time.timeScale = 1f;
      SceneManager.LoadScene("MainMenu");

   }
   public void QuitGame()
   {
      Debug.Log("Quiting Game");
      Application.Quit();
```

```
    }
    void Pause()
    {
        pauseMenuUi.SetActive(true);
        Time.timeScale = 0f;
        GameIsStopped = true;
    }
}
```

ObjectiveCompleted.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ObjectivesCompleted :
MonoBehaviour
{
    [Header("Objectives to
Complete")]
    public Text objectives1;
    public Text objectives2;
    public Text objectives3;
    public Text objectives4;
    public Text objectives5; // added
    public Text objectives6; // added

    public static
ObjectivesCompleted occurence;

    private void Awake()
    {
        if (occurence == null)
        {
            occurence = this;
        }
        else
        {
            Destroy(gameObject);
        }
    }

    public void GetObjectives(bool
objt1, bool objt2, bool objt3, bool
objt4, bool objt5, bool objt6)
    {
        Debug.Log($"Objectives -
Key: {objt1}, Computer: {objt2},
Generator: {objt3}, Bomb: {objt4},
Destroy: {objt5}, Helicopter:
{objt6}");

        if (objt1)
        {
            objectives1.text = "1. Key picked up";
            objectives1.color = Color.green;
        }
        else
        {
            objectives1.text = "1. Find Key to open the door";
            objectives1.color = Color.white;
        }

        // Objective 2: Central Computer Shutdown
        if (objt2)
        {
            objectives2.text = "2. Central Computer is
disconnected";
            objectives2.color = Color.green;
        }
        else
        {
            objectives2.text = "2. Shutdown the Central
Computer";
            objectives2.color = Color.white;
        }
        if (objt3)
        {
            objectives3.text = "3. Generator is offline";
            objectives3.color = Color.green;
        }
        else
        {
            objectives3.text = "3. Shutdown the Generator";
            objectives3.color = Color.white;
        }
        if (objt4)
        {
            objectives4.text = "4. Bomb picked up";
            objectives4.color = Color.green;
        }
        else
        {
            objectives4.text = "4. Pick up the Time-Bomb";
            objectives4.color = Color.white;
        }

        // Objective 5: Warehouse Destroyed
        if (objt5)
        {
            objectives5.text = "5. Warehouse destroyed";
            objectives5.color = Color.green;
```

```
  }
      else
      {
         objectives5.text = "5.
Destroy the Weapon Warehouse";
         objectives5.color =
Color.white;
      }

      // Objective 6: Escape via
Helicopter
      if (objt6)
      {
         objectives6.text = "6. Game
Completed";
         objectives6.color =
Color.green;
      }
      else
      {
         objectives6.text = "6.
Escape with Helicopter";
         objectives6.color =
Color.white;
      }
   }
}
```

RotateUIHealthBar.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RotateUIHealthBar :
MonoBehaviour
{
   public Transform MainCamera;

   private void LateUpdate()
   {
      transform.LookAt(transform.p
osition+ MainCamera.forward);
   }
}
```

SwitchCamera.cs

```
using UnityEngine;

public class SwitchCamera : MonoBehaviour
{
   [Header("Camera to Assign")]
   public GameObject AimCam;
   public GameObject AimCanvas;
   public GameObject ThirdPersonCam;
   public GameObject ThirdPersonCanvas;

   [Header("Camera Animator")]
   public Animator animator;
   public void Update()
   {
      if (Input.GetButton("Fire2") &&
Input.GetKey(KeyCode.W)||
Input.GetKey(KeyCode.UpArrow))
      {
         animator.SetBool("Idle", false);
         animator.SetBool("IdleAim", true);
         animator.SetBool("AimWalk", true);
         animator.SetBool("Walk",true );
         ThirdPersonCam.SetActive(false);
         ThirdPersonCanvas.SetActive(false);
         AimCam.SetActive(true);
         AimCanvas.SetActive(true);
      }
      else if(Input.GetButton("Fire2"))
      {
         animator.SetBool("Idle", false);
         animator.SetBool("IdleAim", true);
         animator.SetBool("AimWalk", false);
         animator.SetBool("Walk", false);

         ThirdPersonCam.SetActive(false);
         ThirdPersonCanvas.SetActive(false);
         AimCam.SetActive(true);
         AimCanvas.SetActive(true);
      }
      else
      {
         animator.SetBool("Idle", true);
         animator.SetBool("IdleAim", false);
         animator.SetBool("AimWalk", false);
         ThirdPersonCam.SetActive(true);
         ThirdPersonCanvas.SetActive(true);
         AimCam.SetActive(false);
         AimCanvas.SetActive(false);

   }
   }
```

KeyGateRegulator.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.Build;
using UnityEngine;

namespace KeyNetwork
{
   public class KeyGateRegular :
MonoBehaviour
   {
      [Header("Animations")]
      private Animator
gateAnimator;
      private bool OpenGate = false;
      [SerializeField] private string
openAnimationName =
"GateOpen";
      [SerializeField] private string
closeAnimationName =
"GateClose";

      [Header("Time and UI")]
      [SerializeField] private int
timeToShowUI = 1;
      [SerializeField] private
GameObject showGateLockedUI =
null;
      [SerializeField] private
KeyList keylist = null;
      [SerializeField] private int
waitTimer = 1;
      [SerializeField] private bool
pauseInteraction = false;

      [Header("Sound Effect")]
      public AudioClip gateSound;
      public AudioSource
audioSource;
      private void Awake()
      {
         gateAnimator =
GetComponent<Animator>();
      }

      public void StartAnimation()
      {
         if (keylist.hasKey)
         {
            Open_Gate();
         }
         else
         {
            StartCoroutine(showGateLoked());
         }
      }

      private IEnumerator StopGateInterconnection()
      {
         pauseInteraction = true;
         yield return new WaitForSeconds(waitTimer);
         pauseInteraction = false;
      }
      void Open_Gate()
      {
         if (!OpenGate && !pauseInteraction)
         {
            gateAnimator.Play(openAnimationName,
0, 0.0f);
            audioSource.PlayOneShot(gateSound);
            OpenGate = true;
            ObjectivesCompleted.occurence.GetObjecti
ves(true,false,false,false,false,false);
            StartCoroutine(StopGateInterconnection());

         }
         else if (OpenGate && !pauseInteraction)
         {
            gateAnimator.Play(closeAnimationName,
0, 0.0f);
            audioSource.PlayOneShot(gateSound);
            OpenGate = false;
            StartCoroutine(StopGateInterconnection());
         }
      }

      IEnumerator showGateLoked()
      {
         showGateLockedUI.SetActive(true);
         yield return new
WaitForSeconds(timeToShowUI);
         showGateLockedUI.SetActive(false);
      }
   }

}
```

KeyObjectRegulator.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
namespace KeyNetwork
{
   public class KeyObjectRegulator :
MonoBehaviour
   {
       [SerializeField] private bool key = false;
       [SerializeField] private bool gate =
false;
       [SerializeField] private KeyList keylist;
      private KeyGateRegular gateObject;
       private void Start()
       {
           if (gate)
               gateObject =
GetComponent<KeyGateRegular>();
       }
 public void foundObject()
       {
           if (key)
           {
               keylist.hasKey = true;
               gameObject.SetActive(false);
           }
           else if (gate)
           {

  gateObject.StartAnimation();

           }

}
```

KeyList.cs

```
   using System.Collections;
   using System.Collections.Generic;
   using UnityEngine;
namespace KeyNetwork
{
      public class KeyList : MonoBehaviour
      {
          public bool hasKey = false}}
```

KeyRayCast.cs

```
using Bomb;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;

namespace KeyNetwork
{
  public class KeyRayCast : MonoBehaviour
  {
    [Header("Raycast Radius and Layer")]
    [SerializeField] private int rayRadius = 6;
    [SerializeField] private LayerMask LayerMaskCollective;
    [SerializeField] private string banLayerName = null;

    private KeyObjectRegulator raycastedObject;
    [SerializeField] private KeyCode openGateButton = KeyCode.F;
    [SerializeField] private Image crosshair = null;

    private bool checkCrosshair;
    private bool Onetime;

    private string collectiveTag = "collectiveObject";

    [Header("Raycast Radius and Layer for bomb")]
    [SerializeField] private int rayRadiusBomb = 6;
    [SerializeField] private LayerMask LayerMaskCollectiveBomb;
    [SerializeField] private string banLayerNameBomb = null;

    private BombRegulator raycastedObjectBomb;
    [SerializeField] private KeyCode PickUpButton = KeyCode.F;

    private string bombTag = "Bombobject";

    private void Update()
    {
      PickBomb();
      PickKey();


    }

    void ChangeCrosshair(bool changeCh)
    {
      if (changeCh && !Onetime)
      {
        crosshair.color = Color.blue;
      }
      else
      {
        crosshair.color = Color.white;
        checkCrosshair = false;
      }
    }

    void PickKey()
    {
      RaycastHit hitinfo;

      Vector3 forwardDirection = transform.TransformDirection(Vector3.forward);
      int mask = 1 <<
LayerMask.NameToLayer(banLayerName) |
LayerMaskCollective.value;


      if
(Physics.Raycast(transform.position,
forwardDirection, out hitinfo, rayRadius,
mask))
      {
        if
(hitinfo.collider.CompareTag(collectiveTag))
        {
          if (!Onetime)
          {
            raycastedObject =
hitinfo.collider.gameObject.GetComponent<
KeyObjectRegulator>();
            ChangeCrosshair(true);
          }
          checkCrosshair = true;
          Onetime = true;


          if
(Input.GetKeyDown(openGateButton))
          {
            raycastedObject.foundObject(
);
          }

        }
      }
      else
      {
        if (checkCrosshair)
        {
          ChangeCrosshair(false);
          Onetime = false;
        }

      }
```

```
}


    void PickBomb()
    {
      Debug.Log("In the PickBomb");
      RaycastHit hitInfo;
      Vector3 forwardDirection =
transform.TransformDirection(Vector3.forward);


      // Use the LayerMask specific to the bomb
      int mask = 1 <<
LayerMask.NameToLayer(banLayerNameBomb) |
LayerMaskCollectiveBomb.value;


      if (Physics.Raycast(transform.position,
forwardDirection, out hitInfo, rayRadiusBomb,
mask))
      {
        // If bomb object is detected within range
        if (hitInfo.collider.CompareTag(bombTag))
        {
          if (!Onetime)  // Only change crosshair
and set up the bomb once
          {
            // Cache the bomb object so it can be
used for pickup
            raycastedObjectBomb =
hitInfo.collider.gameObject.GetComponent<BombRe
gulator>();
            ChangeCrosshair(true);  // Change
crosshair to show interact action
          }
          checkCrosshair = true;
          Onetime = true;
    if (Input.GetKeyDown(PickUpButton) )
          {
            raycastedObjectBomb.foundObject();
// Call bomb pickup method from BombRegulator
            Onetime = false;
```

```
                }
              }
            }
            else
            {
              // Reset crosshair and one-time flag
if no bomb is detected
              if (checkCrosshair)
              {
                ChangeCrosshair(false);
                Onetime = false;
              }
            }
          }


    }
}
```

BombObject.cs

```
using UnityEngine;
using System.Collections;

namespace Bomb
{
    public class BombWarehouse :
MonoBehaviour
    {
        public HasBomb hasbomb;
        public GameObject warehouse;
        public GameObject bombPrefab;
        private bool bombPlaced = false;
        private bool playerInsideWarehouse =
false;
        public ParticleSystem destroyEffect;

        private Transform playerTransform;

        private void Start()
        {
            playerTransform =
GameObject.FindWithTag("Player").transform;
        }

        void Update()
        {
            if ( !bombPlaced && hasbomb.hasBomb &&
Input.GetKeyDown(KeyCode.E))
            {
                PlaceBomb();
            }
            else
            {
                Debug.Log("PLayer is outside the
warehouse");
            }
        }
        void PlaceBomb()
        {
            bombPlaced = true;
            Vector3 warehousePosition =
playerTransform.position + new Vector3(2, 0, 0);
            GameObject bomb = Instantiate(bombPrefab,
warehousePosition, Quaternion.identity);
            Debug.Log(bomb.name);
            bomb.transform.SetParent(warehouse.transfor
m);
            StartCoroutine(DestroyWarehouseAfterDelay(
10f));
        }
        IEnumerator DestroyWarehouseAfterDelay(float
delay)
        {
            destroyEffect.Play();
            yield return new WaitForSeconds(delay);
            Destroy(warehouse);
            ObjectivesCompleted.occurence.GetObjective
s(true, true, true, true, true, false);
            Debug.Log("Warehouse destroyed after bomb
explosion!");
        }
        private void OnTriggerEnter(Collider other)
```

```
 {
          if (other.CompareTag("Player"))
          {
             playerInsideWarehouse = true;
             Debug.Log("Player entered the
warehouse.");
          }
       }
       private void OnTriggerExit(Collider
other)
       {
          if (other.CompareTag("Player"))
          {
             playerInsideWarehouse = false;
             Debug.Log("Player exited the
warehouse.");
          }
       }
   }
}
```

BonbRegulator.cs

```
using KeyNetwork;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace Bomb
{
   public class BombRegulator :
MonoBehaviour
   {
       [SerializeField] private bool isBomb =
false;
       [SerializeField] private bool
isWarehouse = false;

       [SerializeField] private HasBomb
hasBomb;

       private BombWarehouse
warehouseObject;

       private void Start()
       {
          if (isWarehouse)
          {
             warehouseObject =
GetComponent<BombWarehouse>();
          }
       }

       public void foundObject()
       {
          if (isBomb)
          {
             if (hasBomb != null)
             {
                hasBomb.hasBomb = true;


             }
             gameObject.SetActive(false);
             ObjectivesCompleted.occurence.GetObjecti
ves(true, true, true, true, false, false);
          }
          else if (isWarehouse && warehouseObject !=
null)
          {
             DestroyWarehouse();
          }
       }

       private void DestroyWarehouse()
       {
          // Logic to destroy the warehouse, like
deactivating the GameObject or playing an animation
          Debug.Log("Warehouse destroyed!");
          Destroy(warehouseObject.gameObject); //
Assuming warehouseObject is the GameObject you
want to destroy
       }
```

```
    }


}
```

HasBomb.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HasBomb : MonoBehaviour
{
    public bool hasBomb = false;
}
```

CHAPTER 7
Testing

Functional Test Cases:

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result
TC-011	Player Damage System	Functional	Player can and have hit from guns	Get damage and give damage from rifle	Player gets the damage and gives the damage	Player gets the damage and gives the damage
TC-012	Character Interaction	Functional	Character encounters an NPC	Initiate a conversation with an NPC	Dialogue options are displayed, and conversations progress logically	Dialogue options are displayed, and conversations progress logically
TC-013	Object Physics	Functional	Player interacts with physics-based objects	Push, throw, or manipulate physics objects	Objects respond to interactions realistically	Objects respond to interactions realistically
TC-014	Game Save Load	Functional	Player is in the game and has previously saved progress	Save the game, exit, and then load the saved game	The game is saved and loaded accurately, and the player resumes from the saved state	The game is saved and loaded accurately, and the player resumes from the saved state
TC-015	Puzzle Solving	Functional	Player encounters a puzzle	Attempt to solve the puzzle using the correct approach	Solving the puzzle correctly progresses the game	Solving the puzzle correctly progresses the game

Table 7.1

Performance Test Cases:

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result
TC-016	Frame Rate Stability	Performance	Game is running	Play the game, monitor frame rates, and check for any noticeable drops	The game maintains a stable frame rate without significant drops	The game maintains a stable frame rate without significant drops
TC-017	Loading Time	Performance	Loading a new game level	Measure the time it takes to load a level	Loading times are within acceptable limits	Loading times are within acceptable limits
TC-018	Memory Usage	Performance	Game is running	Monitor memory usage during gameplay	Memory usage remains within acceptable limits	Memory usage remains within acceptable limits
TC-019	Network Performance	Performance	Game includes online features	Test online gameplay, check for lag or disconnections	Online gameplay is smooth with minimal lag or disconnections	Online gameplay is smooth with minimal lag or disconnections
TC-020	Resource Utilization	Performance	Game is running	Monitor CPU and GPU utilization	CPU and GPU usage are within acceptable limits	CPU and GPU usage are within acceptable limits

Table 7.2

User Experience (UX) Test Cases:

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result
TC-021	User Interface Clarity	User Experience	Player is in the game	Navigate through the game menu and UI	Game menus and UI elements are clear and intuitive	Game menus and UI elements are clear and intuitive
TC-022	Controls Intuitiveness	User Experience	Player starts the game	Attempt to move, interact, and use in-game controls	Controls are intuitive and easy to use	Controls are intuitive and easy to use
TC-023	HUD Information	User Experience	Player in gameplay	Observe the heads-up display (HUD)	Relevant information on the HUD is easily readable and helpful	Relevant information on the HUD is easily readable and helpful
TC-024	Audio Balance	User Experience	Game is launched	Play the game with different audio settings	Audio levels are balanced, allowing for clear dialogue and sound effects	Audio levels are balanced, allowing for clear dialogue and sound effects
TC-025	Navigation Guidance	User Experience	Player exploring the house	Try to find the way to the next objective	Clear visual or audio cues guide the player to the next objective	Clear visual or audio cues guide the player to the next objective

Table 7.3

Requirement Traceability Matrix

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result	Status
TC-001	Player Movement	Functional	Player starts the game	Move the player character forward	Player character moves forward smoothly	Player character moves forward smoothly	Passed
TC-002	Graphics Quality	Performance	Game is launched	Adjust graphics settings to the highest quality	High-quality graphics with no lag	High-quality graphics with no lag	Passed
TC-003	Audio Effects	Functional	Game is launched	Test game audio with headphones	Immersive and chilling audio experience	Immersive and chilling audio experience	Passed
TC-004	Jump Scares	User Experience	Player exploring the house	Enter a room with a jump scare event	Player experiences a jump scare	Player experiences a jump scare	Passed
TC-005	Lighting Effects	Visual	Game is launched	Enter a dark room	Realistic and eerie lighting effects	Realistic and eerie lighting effects	Passed
TC-006	Ghost Encounters	Functional	Player exploring the house	Encounter a ghost in a specific location	Player encounters a ghostly apparition	Player encounters a ghostly apparition	Passed
TC-007	Saving Progress	Functional	Player in the middle of gameplay	Use the save feature	Game saves the progress correctly	Game saves the progress correctly	Passed
TC-008	Game Performance	Performance	Game is running	Play the game for an extended period	Game doesn't crash or lag	Game doesn't crash or lag	Passed
TC-009	Puzzles and Objectives	Functional	Player exploring the house	Solve a game puzzle or objective	Objective is completed with correct input	Objective is completed with correct input	Passed

Test Case ID	Requirement	Type	Precondition	Steps	Expected Result	Actual Result	Status
TC-010	Storyline Progression	Functional	Player completes certain objectives	Continue with the storyline	Game progresses to the next part of the story	Game progresses to the next part of the story	Passed
TC-021	User Interface Clarity	User Experience	Player is in the game	Navigate through the game menu and UI	Game menus and UI elements are clear and intuitive	Game menus and UI elements are clear and intuitive	Passed
TC-022	Controls Intuitiveness	User Experience	Player starts the game	Attempt to move, interact, and use in-game controls	Controls are intuitive and easy to use	Controls are intuitive and easy to use	Passed
TC-023	HUD Information	User Experience	Player in gameplay	Observe the heads-up display (HUD)	Relevant information on the HUD is easily readable and helpful	Relevant information on the HUD is easily readable and helpful	Passed
TC-024	Audio Balance	User Experience	Game is launched	Play the game with different audio settings	Audio levels are balanced, allowing for clear dialogue and sound effects	Audio levels are balanced, allowing for clear dialogue and sound effects	Passed
TC-025	Navigation Guidance	User Experience	Player exploring the house	Try to find the way to the next objective	Clear visual or audio cues guide the player to the next objective	Clear visual or audio cues guide the player to the next objective	Passed

Table 7.4

CHAPTER 8
Output/Screenshots

Menu Screen:

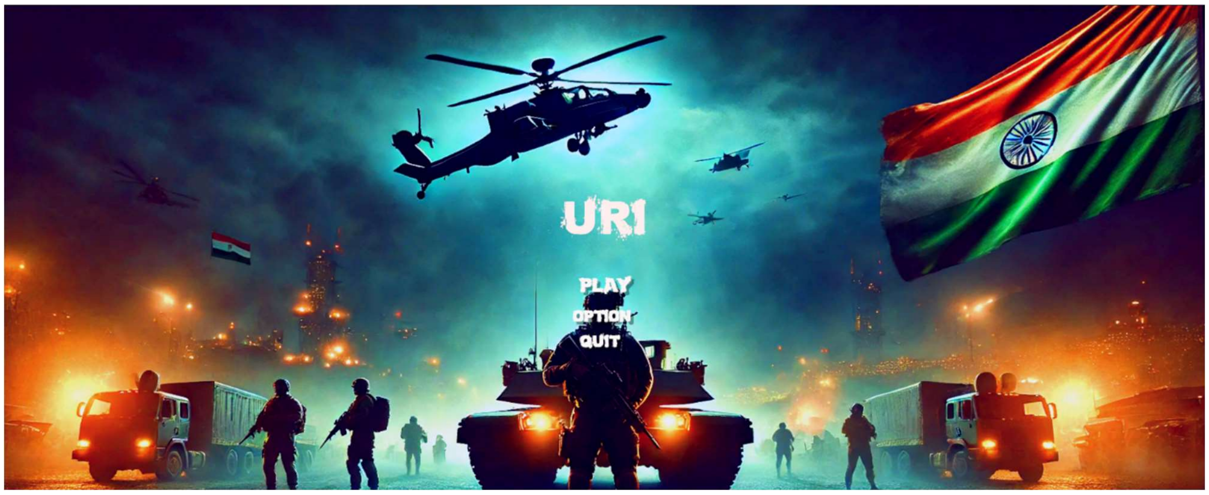


Figure 8.1

Opening Scene:

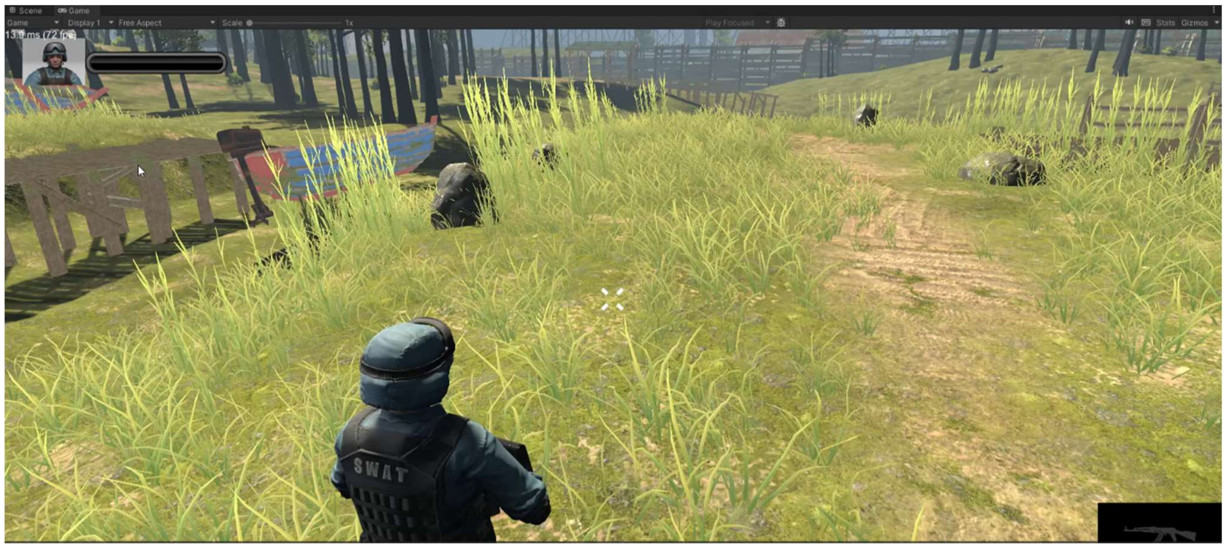


Figure 8.2

Player Shooting Enemy Drone:

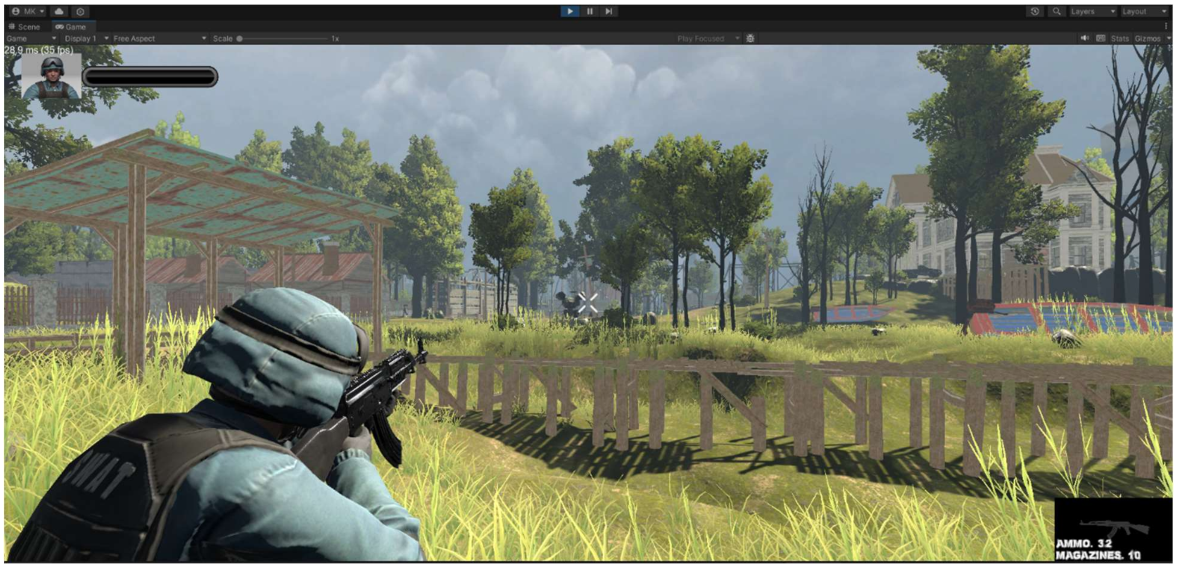


Figure 8.3

Player Shooting Enemy:

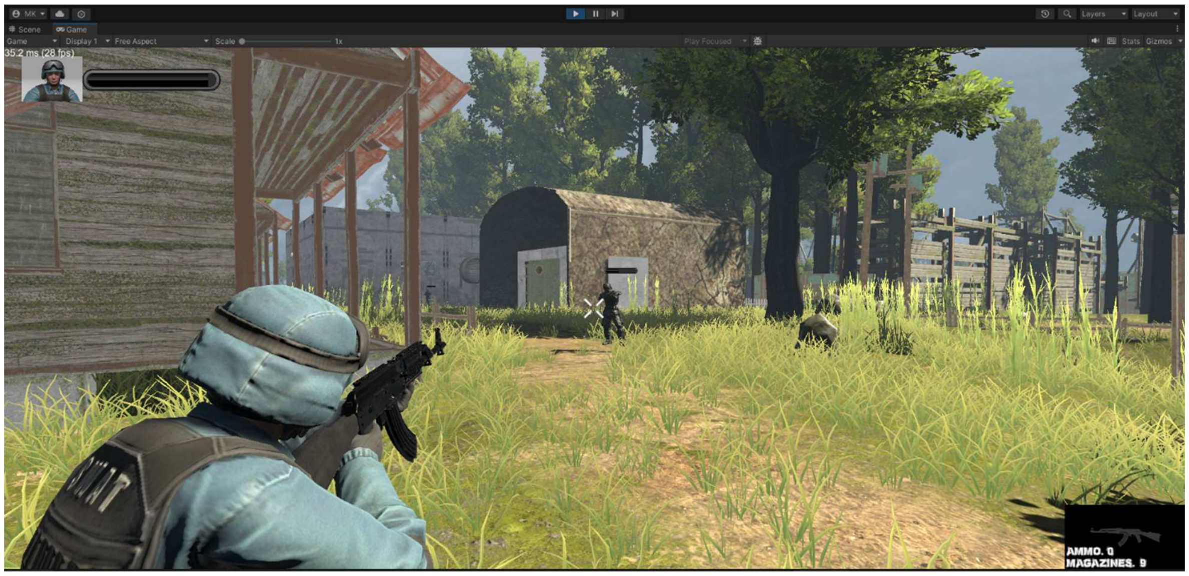


Figure 8.4

Warehouse

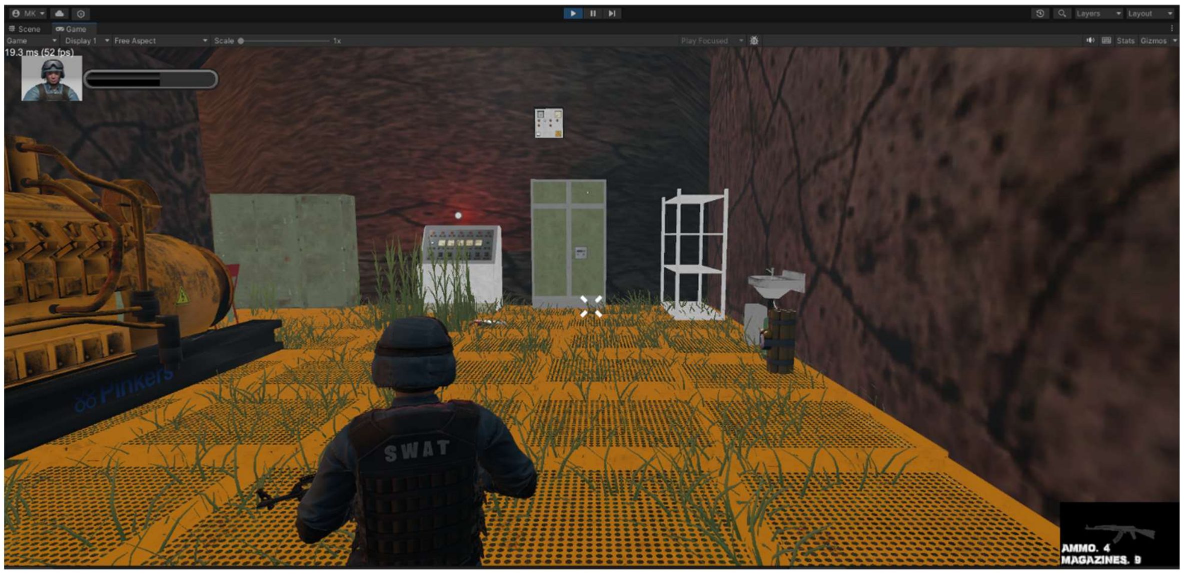


Figure 8.5

Computer Disable Objective

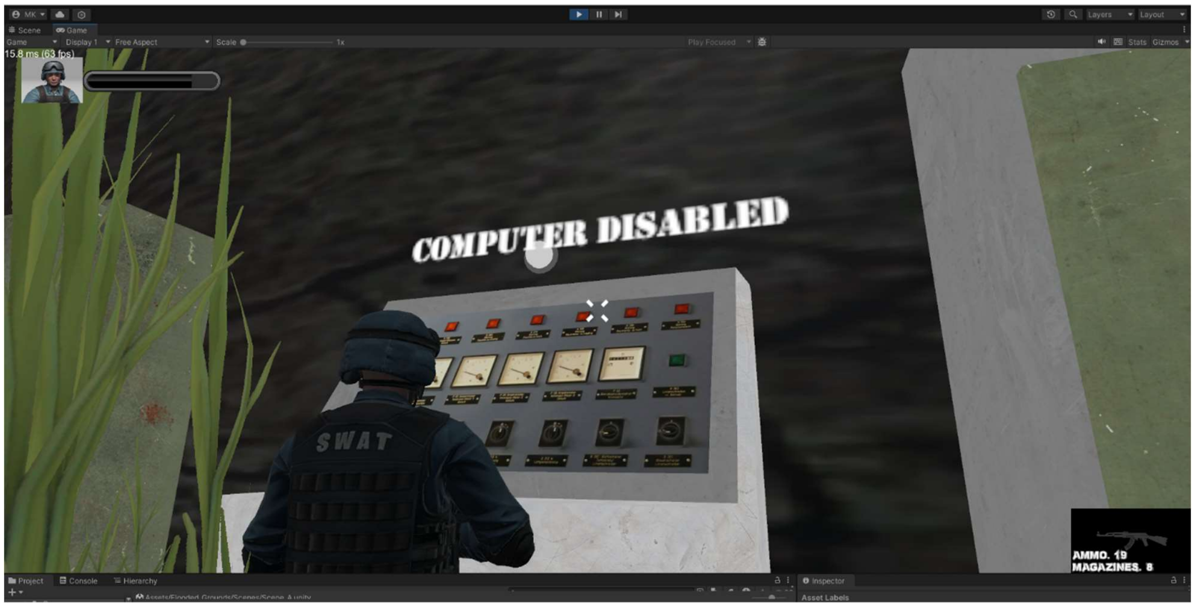


Figure 8.6

Objective Menu

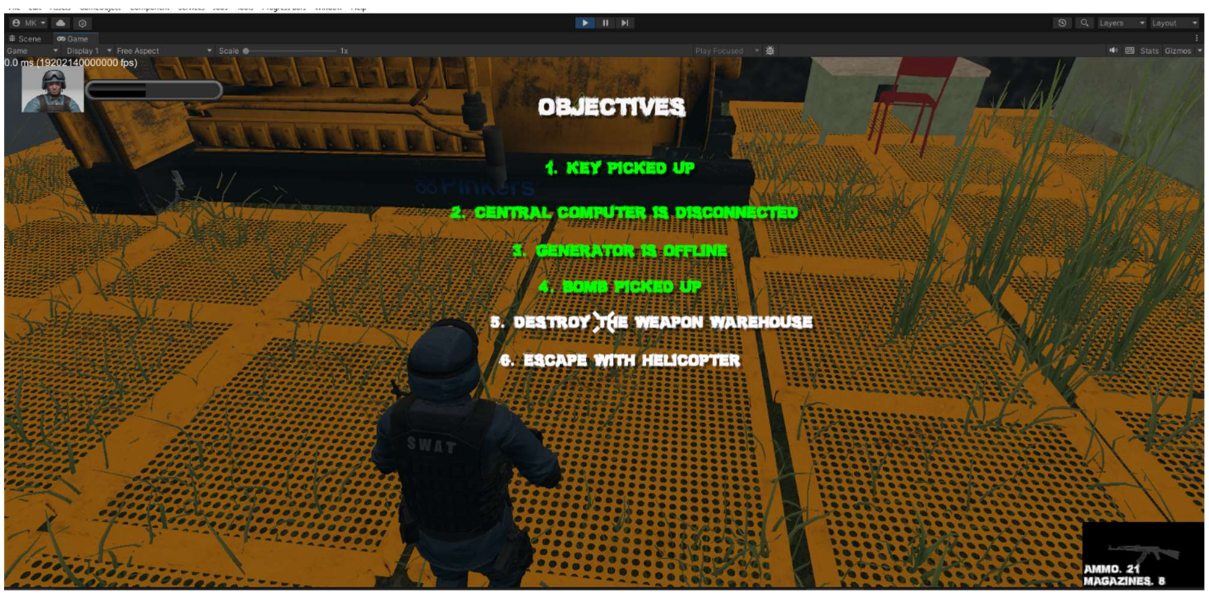
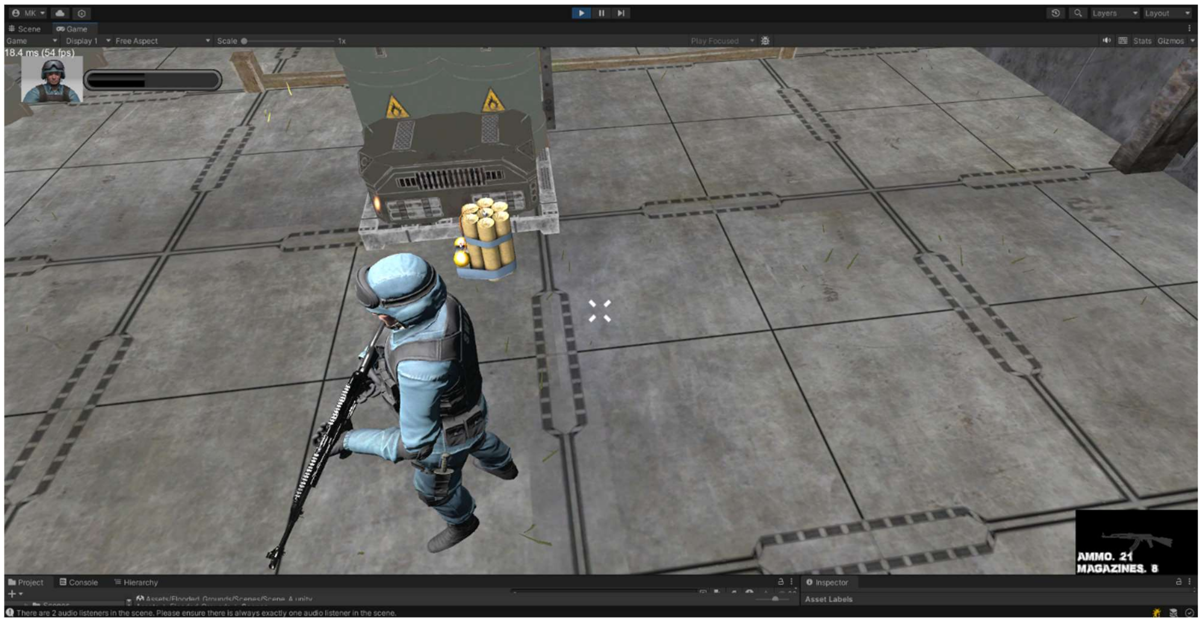

OBJECTIVES
1. KEY PICKED UP
2. CENTRAL COMPUTER IS DISCONNECTED
3. GENERATOR IS OFFLINE
4. BOMB PICKED UP
5. DESTROY THE WEAPON WAREHOUSE
6. ESCAPE WITH HELICOPTER
AMMO. 21
MAGAZINES. 8

Figure 8.7

Deploying bomb in weapon warehouse



There are 2 audio listeners in the scene. Please ensure there is always exactly one audio listener in the scene.

Last Objective

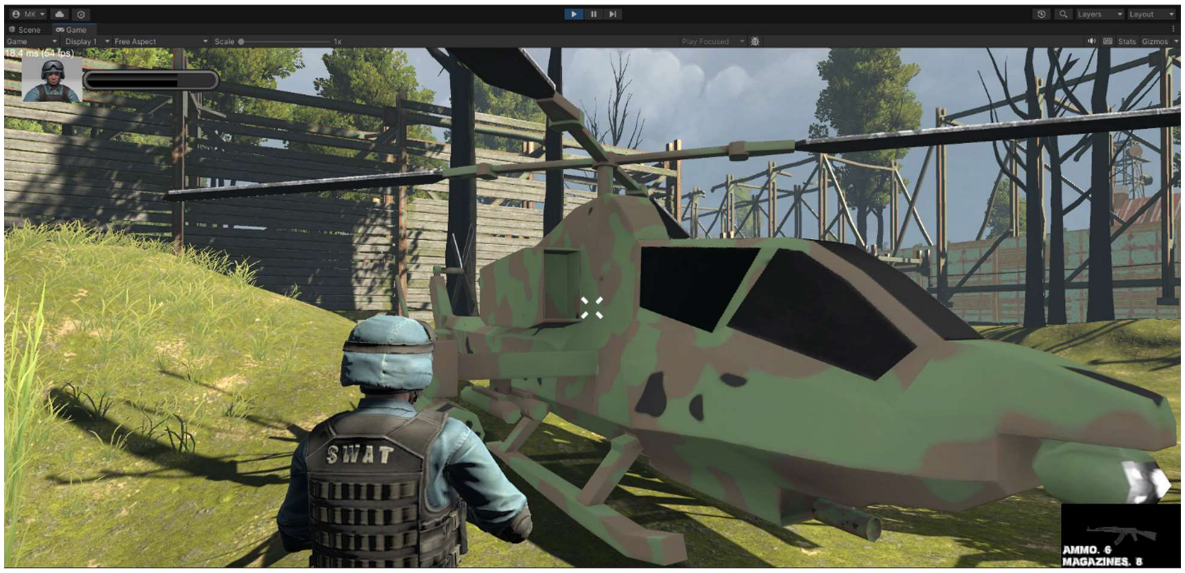


Figure 8.9

Death Menu

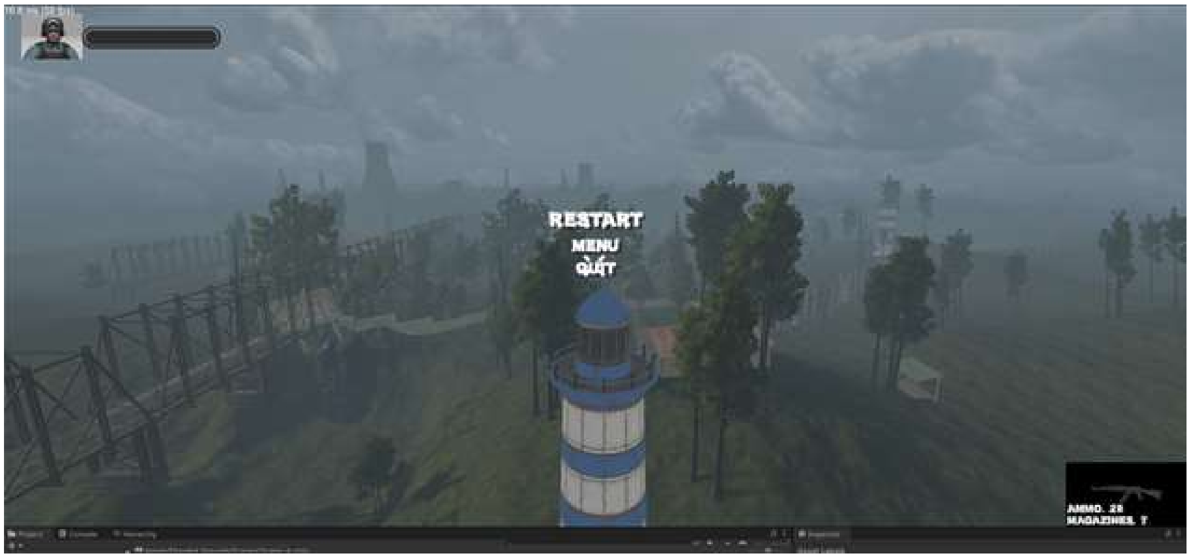


Figure 8.10

CHAPTER 9
Future Scope and Conclusion

Future Scope

As *"URI - A 3D Tactical Shooter Game"* approaches completion, exploring its future development potential is crucial. Here are some exciting avenues for enhancing and expanding the game:

New Missions and Environments: *"URI"* could be expanded with additional missions, enemy types, and diverse environments. Adding new locations and mission objectives would continue to challenge players and provide fresh experiences, increasing replay ability and engagement for both current and new players.

Multiplayer Mode: Integrating a multiplayer mode could open opportunities for cooperative or competitive gameplay. Players could work as a team, tackling high-stakes missions together, or face off against each other in strategic, team-based scenarios. This would introduce a social and tactical layer, enriching the gameplay.

Enhanced Narrative and Characters: Expanding the game's narrative depth with more character backstories, cutscenes, and dialogues could further immerse players. Building on the hero's motivations and the story's geopolitical themes would allow players to connect emotionally with the characters, enhancing the game's impact.

VR Adaptation: Adapting *"URI"* for virtual reality would offer players an even more immersive experience. VR integration could bring players closer to the action, enabling realistic movement and interactions within the game environment. This approach would leverage the game's tactical elements and first-person perspective to create an intense, immersive experience.

Mobile Version: Creating a mobile adaptation of *"URI"* could broaden its audience, allowing fans to engage with the game on-the-go. A mobile version would focus on optimizing controls, graphics, and mission lengths for portable play without sacrificing core gameplay elements.

Conclusion:

In conclusion, *"URI - A 3D Tactical Shooter Game"* has been a challenging and rewarding project that pushes the boundaries of tactical shooter games. From detailed environments and realistic combat mechanics to an engaging storyline and high-stakes mission structure, the game provides an intense and immersive experience that keeps players engaged and invested.

The project's success was made possible through our methodical approach, leveraging Agile development and Scrum practices to adapt to changing requirements and player feedback effectively. Cloud-based collaboration tools also facilitated teamwork and efficient project management.

The game’s core features—including tactical gameplay, detailed environments, and responsiv mechanics—combine to create a compelling, action-filled experience. The dynamic mission structure an player-driven choices ensure that players remain fully engaged in *"URI"*'s intense and suspenseful world.

Looking to the future, *"URI"* has great potential for further innovation and expansion throug additional content, multiplayer options, enriched storytelling, VR adaptation, and a mobile version. Thes developments will enhance player engagement, longevity, and accessibility of the game.

In summary, *"URI - A 3D Tactical Shooter Game"* is a testament to the exciting potential within th tactical shooter genre, delivering an intense and unforgettable experience for players while holding promis for continued evolution and growth in the future.

CHAPTER 10
REFERENCES

REFERENCES

-Unity Game Engine:

- Unity Technologies. (2023). *Unity Real-Time Development Platform* [Online]. Available: https://unity3d.com

- Unity Technologies. (2023). *Real-Time 3D Development Software for Games & More* [Online]. Available: https://unity3d.com/unity

C# Scripting:

- Unity Technologies. (2023-11-01). *Unity – Manual: Scripting* [Online]. Available: https://docs.unity3d.com/Manual/ScriptingSection.html

- Vishnu Sivan. (2022-04-30). *Unity 3D C# scripting cheatsheet for beginners* [Online]. Available: https://blog.devgenius.io/unity-3d-c-scripting-cheatsheet-for-beginners-be6030b5a9ed

Websites:

- Unity Technologies. (2023-11-01). *Unity – Manual: Working in Unity* [Online]. Available: https://docs.unity3d.com/Manual/UnityOverview.html

- Unity Technologies. (2023-11-01). *Unity – Manual: Physics* [Online]. Available: https://docs.unity3d.com/Manual/PhysicsSection.html

- Unity Technologies. (2023-11-01). *Unity – Manual: Audio* [Online]. Available: https://docs.unity3d.com/Manual/Audio.html

- Maximo Characters and Animations https:// www.mixamo.com

Unity Asset Store:

- Unity Technologies. (2024). *Unity Asset Store - The Best Assets for Game Making* [Online]. Available: https://assetstore.unity.com

- Flooded Grounds. (2024). *Unity Asset Store- Best 3d Environments*: https://assetstore.unity.com/packages/3d/environments/flooded-grounds/-