THAKUR DEGREE COLLEGE OF SCIENCE AND COMMERCE

KANDIVALI (EAST) MUMBAI

A PROJECT REPORT ON

NightWars (A Unity 3d based Desktop Game)

For

Thakur College of Science and Commerce

By

Deep G Patel

Submitted in partial fulfilment of Master of Science (Computer Science)

[UNIVERSITY OF MUMBAI]

Thakur Degree College of Science and Commerce Kandivali (East),Mumbai.

ACADEMIC YEAR 2023 – 2024

Guided by Dr. Girish Tere & Dr. Sujitha Mohan

Thakur Educational Trust's (Regd.)
THAKUR COLLEGE OF SCIENCE & COMMERCE
AUTONOMOUS COLLEGE, PERMANENTLY AFFILIATED TO UNIVERSITY OF MUMBAI
NAAC Accredited Grade 'A' (3rd Cycle) & ISO 9001: 2015 (Certified)
Best College Award by University of Mumbai for the Year 2018-2019
CELEBRATING 25 YEARS OF GLORY

Department of Computer Science

Certificate

Class: M.Sc. Computer Science – Part II (Semester: IV)

Roll No: 4715

Academic Year: 2023-2024

This is to confirm & certify that Mr. Deep G Patel, MSc (Computer Science) student from "Thakur College of Science & Commerce [Autonomous], Kandivali (E)", with Roll No. 4715 has successfully submitted his/her project titled "NightWars (A Unity 3d based Desktop Game)". It is an original research-based study work conducted in the field of Game Development..

Title: NightWars (A Unity 3d based Desktop Game).

PROJECT GUIDE	HEAD OF DEPARTMENT	EXTERNAL EXAMINER

Date:

ACKNOWLEDGEMENT

Achievement is finding out what you would be doing rather than what you have to do. It is not until you undertake such a project that you realize how much effort and hard work it really is, what are your capabilities and how well you can present yourself or other things. It tells us how much we rely on the efforts and goodwill of others. It gives me immense pleasure to present this report towards the fulfilment of my project.

It has been rightly said that we are built on the shoulder of others. For everything. I have achieved, the credit goes to all those who had helped me to complete his project successfully.

I take this opportunity to express my profound gratitude to management of Thakur Degree College of Science & Commerce for giving me this opportunity to accomplish this project work.

I am very much thankful to **Mrs. C. T. Chakraborty** Principal of Thakur College for their kind co-operation in the completion of my project.

A special vote of thanks to my faculty, **Mr. Ashish Trivedi** who is our HOD & also project guide **Dr. Girish Tere & Dr. Sujitha Mohan** for their most sincere, useful and encouraging contribution throughout the project span, without them we couldn't start and complete the project on time.

Finally, I would like to thank all my friends & entire Computer Science Department who directly or indirectly helped me in completion of this project & to my family without whose support, motivation & encouragement this would not have been possible.

(Deep G Patel)

DECLARATION

I / We hereby declare that the project entitled, **"NightWars (A Unity 3d based Desktop Game)"** done at THAKUR DEGREE COLLEGE OF SCIENCE AND COMMERCE , has not been in any case duplicated to submit to any other university for the award of any degree.To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirement for the award of degree of MASTER OF SCIENCE (COMPUTER SCIENCE) to be submitted as final semester project as part of our curriculum.

(Deep G Patel)

Table of Contents

Title: NightWars (A Unity3d based Desktop Game)

ABSTRACT

In an era marked by a heightened emphasis on Gaming Evolution, the "NightWars" Unity3D desktop game emerges as an immersive solution to cater to the evolving preferences of gamers. With Unity3D as its foundation, this project aims to redefine the traditional gaming experience by offering players a captivating survival shooter adventure without compromising on safety or convenience. Drawing inspiration from the growing demand for immersive gaming experiences, "NightWars" leverages Unity3D's robust capabilities to create a dynamic and challenging gameplay environment.[1]

The abstract concept of "NightWars" revolves around the integration of engaging gameplay mechanics, dynamic level design, and strategic shooting elements. Players navigate through progressively difficult levels, tasked with eliminating enemies and prioritizing survival in intense combat scenarios. Developed with a focus on player engagement and satisfaction, "NightWars" seeks to push the boundaries of interactive entertainment, offering players an opportunity to test their skills, strategize, and emerge victorious in the face of relentless opposition.[3]

As a Unity3D desktop game project, "NightWars" not only aims to provide an immersive gaming experience but also to showcase the potential of game development as a creative and innovative medium. By fostering creativity, problem-solving skills, and a sense of accomplishment, "NightWars" endeavors to leave a lasting impression on players, contributing to the vibrant tapestry of the gaming industry. In a time when safety and convenience are paramount, "NightWars" stands as a testament to the power of technology to redefine commonplace interactions and offer players a truly engaging and captivating gaming experience.[4]

1] INTRODUCTION TO THE PROJECT

1.1 Introduction

In recent years, the demand for game development has surged exponentially, positioning it as a leading force in the digital entertainment industry. Unity3D, an integral part of this growth, has democratized game creation, making it accessible to a diverse pool of developers. This dynamic and vibrant field is responsible for crafting interactive and immersive experiences across a multitude of platforms. Game development serves various purposes, including entertainment, education, and simulation, offering creative outlets and fostering problem-solving skills. Its far-reaching benefits span personal and artistic growth, entrepreneurial opportunities, and technological advancements, cementing game development as a pivotal and influential domain in today's digital landscape.[1]

The scope of the "Night Wars" game development project involves creating an immersive Unity 3D desktop game in the survival shooter genre. Players will face progressively challenging levels as they eliminate enemies and prioritize their survival. Developed with C#, the project focuses on delivering an engaging and captivating gaming experience that encourages players to strategize and demonstrate their shooting skills for survival and success within the game.[3]

1.2 Background

The surge in demand for game development in recent years has propelled it to the forefront of the digital entertainment industry. Unity3D, a key player in this growth, has democratized game creation, making it accessible to a wide range of developers. Game development encompasses a diverse array of purposes, from entertainment and education to simulation, offering avenues for creativity and problem-solving skills development. Its benefits extend to personal growth, artistic expression, entrepreneurial opportunities, and technological advancements, solidifying game development as a pivotal domain in today's digital landscape.[2]

In response to the burgeoning demand for immersive gaming experiences, particularly within the survival shooter genre, the "Night Wars" project emerged as a creative endeavor to deliver an engaging Unity 3D desktop game. Recognizing Unity3D's pivotal role in democratizing game development, the project aims to leverage its capabilities to craft a dynamic and challenging gaming experience.[1]

With an emphasis on C# programming language, the project team intends to harness the power of Unity3D to create a captivating gameplay environment where players are thrust into intense combat scenarios against increasingly formidable adversaries. The project draws inspiration from the rich tapestry of survival shooter games while striving to introduce innovative mechanics and level design to keep players engaged.[5]

Through meticulous planning and iterative development cycles, the "Night Wars" project seeks to push the boundaries of interactive

entertainment, providing players with an opportunity to test their skills, strategize, and ultimately emerge victorious in the face of relentless opposition. By fostering creativity, problem-solving, and a sense of accomplishment, the game aims to leave a lasting impression on players, contributing to the vibrant tapestry of the gaming industry.[6]

1.3 Objectives

"Night Wars" is an immersive and challenging Unity 3D desktop game, featuring a survival shooter gameplay. Developed with C# programming language, players are tasked with eliminating enemies and prolonging their survival as much as possible. As the player successfully eliminates more enemies, the game dynamically increases in difficulty with each progressing level.

The primary objective of "Night Wars" game development is to create an immersive Unity 3D desktop game in the survival shooter genre. The game challenges players to eliminate enemies while emphasizing survival, dynamically increasing in difficulty as they progress. Developed in C#, the goal is to provide an engaging and captivating gaming experience, requiring players to strategize and showcase their shooting skills to endure and excel within the game.[6]

2] LITERATURE REVIEW

In the realm of Unity 3D game development, a plethora of literature exists to guide developers through the intricate process of creating immersive gaming experiences. Beginning with foundational texts such as "Unity in Action" by Joseph Hocking and "Learning C# by Developing Games with Unity 2020" by Harrison Ferrone, developers can grasp the fundamentals of Unity's development environment and the principles of object-oriented programming crucial for scripting in C#. These resources lay the groundwork for understanding Unity's robust features, including its physics engine, asset importing capabilities, and scene management tools, essential for creating dynamic and visually captivating games. [7][8]

Furthermore, literature on game design principles, such as "The Art of Game Design: A Book of Lenses" by Jesse Schell and "Level Up! The Guide to Great Video Game Design" by Scott Rogers, provides invaluable insights into crafting compelling gameplay experiences. By exploring topics such as player motivation, game mechanics, level design, and user interface design, developers can refine their creative vision and ensure that their Unity 3D projects resonate with players on a profound level. Through a synthesis of these foundational texts and supplementary resources, developers can navigate the complexities of Unity 3D game development with confidence, paving the way for the creation of engaging and memorable gaming experiences. [9]

3] PROBLEM STATEMENT

In a world increasingly characterized by digital entertainment, the "Night Wars" game development project addresses a fundamental challenge: the need for engaging and immersive experiences that simultaneously entertain and educate. By leveraging Unity 3D's democratized game creation tools, the project aims to provide a solution that not only satisfies the demand for entertainment but also fosters valuable skills such as strategic thinking, problem-solving, and hand-eye coordination.

Through the creation of a survival shooter game, "Night Wars" offers players an opportunity to engage in intense combat scenarios while navigating dynamically challenging environments. This not only provides an outlet for entertainment but also serves as a platform for players to hone their decision-making abilities under pressure and improve their reflexes.

Moreover, the project's emphasis on C# programming language and Unity 3D development demonstrates the practical application of coding skills in a creative context. Aspiring developers and enthusiasts can learn valuable programming concepts while contributing to the gaming experience.

Ultimately, "Night Wars" contributes to addressing the need for accessible and engaging educational tools in the digital age. By providing a platform for entertainment, skill development, and learning, the game project offers tangible benefits to individuals seeking meaningful engagement in their leisure time while also nurturing valuable skills that extend beyond the virtual realm.

The "Night Wars" game development project addresses the need for an engaging and immersive gaming experience that not only entertains players but also fosters skills applicable to real-life scenarios. In an increasingly digital world, where individuals seek entertainment that offers more than mere distraction, this game provides an opportunity for players to enhance their problem-solving abilities, strategic thinking, and hand-eye coordination within a dynamic and challenging environment. By simulating survival scenarios and requiring players to navigate through progressively difficult levels, "Night Wars" equips players with skills that are transferrable to real-life situations, such as decision-making under pressure, resource management, and adaptability. Moreover, the game serves as a platform for players to unwind and relieve stress while engaging in a rewarding and intellectually stimulating activity. Ultimately, "Night Wars" aims to contribute positively to players' personal growth and well-being by providing a fun and educational gaming experience.

4] SYSTEM IMPLEMENTATION

4.1 Development Environment and Tools

In the development of our Unity 3D game project, we utilized a robust set of tools and technologies to facilitate the creation process and ensure efficient workflow. This section provides an overview of the key components of our development environment.

Unity 3D Engine

Unity 3D served as the cornerstone of our game development process. This powerful game engine provided a comprehensive suite of features and tools to create immersive and visually stunning gaming experiences. With Unity, we were able to leverage its advanced graphics rendering capabilities, physics engine, and asset importing functionalities to bring our creative vision to life. Additionally, Unity's cross-platform compatibility enabled us to deploy our game across multiple platforms, reaching a wider audience.

C# Programming Language

C# emerged as the primary programming language for scripting within the Unity environment. Its object-oriented nature and extensive libraries made it an ideal choice for implementing game logic, handling user input, and orchestrating various gameplay mechanics. Through C#, we were able to develop modular and maintainable code, enabling seamless integration with Unity's APIs and systems.

Visual Studio IDE

For coding and script development, we relied on Visual Studio, a feature-rich integrated development environment (IDE). Visual Studio offered powerful tools for code editing, debugging, and project management, enhancing our productivity and streamlining our development workflow. Its seamless integration with Unity allowed for smooth collaboration among team members and facilitated rapid iteration and testing of game features.

Unity Asset Store

The Unity Asset Store played a crucial role in our development process by providing access to a vast library of assets, including 3D models, textures, animations, audio files, and plugins. Leveraging the Asset Store, we were able to enhance our game's visuals, audio, and functionality, saving valuable development time and resources. From pre-made assets to custom scripts and tools, the Unity Asset Store offered a diverse range of resources to support our game development efforts.

By leveraging the Unity 3D engine, C# programming language, Visual Studio IDE, and Unity Asset Store, we were able to create a cohesive and efficient development environment conducive to the realization of our game project's objectives. These tools provided the foundation upon which we built our game, empowering us to innovate, iterate, and deliver a compelling gaming experience to our audience.

4.2 Methodology

Our project has been executed in the following phases:

1) Conceptualization and Planning:

- The game concept and core mechanics were defined.
- A game design document outlining gameplay, levels, characters, and other essential elements was created.
- The development timeline and resource allocation were planned.

2) Asset Creation:

- 2D/3D models for characters, objects, and environments were developed or acquired.
- Textures, materials, and visual effects for the game were created.
- Audio assets, including music, sound effects, and voiceovers, were designed and created.
- Animations for characters and objects were gathered or created.

3) Project Setup:

- A new Unity project was created.
- Assets were imported into the project.
- The project hierarchy and folder structure were organized.

4) Level and Scene Design:

- The game levels and environments were constructed using the assets.
- Interactive elements and gameplay mechanics were implemented within the scenes.

5) Scripting:

- C# or other supported scripts were written to define the game logic and behavior.
- Player controls, AI behaviors, game rules, and interactions were implemented.
- Assets and scripts were integrated for seamless gameplay.

6) User Interface (UI) Design:

- The user interface, including menus, HUD, and in-game UI elements, was designed and implemented.
- UI scripts were created to manage interactions and transitions.

7) Audio Integration:

- Audio sources were added to appropriate game objects for sound effects and background music.
- Audio scripts were implemented to control volume, playback, and other audio-related aspects.

8) Animation Integration:

- Animation controllers and animator components were set up for characters and objects.
- Animations were triggered based on player input, events, or game logic.

9) Testing and Debugging:

- Thorough testing was conducted to identify and fix bugs, performance issues, and gameplay imbalances.
- Feedback from playtesting was gathered to make improvements.

10) Optimization:

- The game was optimized to ensure smooth performance on target platforms.
- Techniques such as reducing poly count, texture compression, and other optimization methods were considered and implemented to improve efficiency.

4.3 Modules

The game contains various modules as follows:

1)Game Menu:

- A game menu with options for controlling brightness, music and effect volumes
- There us data persistence of the volumes so the settings do not have to be set repeatedly
- There are 2 buttons to Start/Resume and Exit
- Can be reached by pressing 'Esc' key to Pause the game

2) Player(Character):

- A 3D Animated character with running, idle and dying animations
- Has a gun which fires only 1 bullet at a time (initially)
- Moves with movement keys and fires on *Click* or pressing *Left Ctrl* key.
- Mouse Pointer to be used for setting firing direction
- Starts with 100 health

3)Enemies:

- Different types of enemies with different specifications
- Each type has different kill points
- Each has their own set of animations
- They burn up in flames when killed and drop pickup items randomly
- **Zombear:** Can attack when in proximity and runs to player **if** in its line of sight
- **Zombunny:** Functionally similar to *Zombear*, but looks different
- **Hellephant:** Can fly around, attacking with bullets

4)Pickups:

- Dropped on random when enemies killed
- Different types of pickups will give different powers to the player
- **Health:** Adds 25 points to health
- **Bounce:** Bullets become green and bounce 4 times when they hit other game objects (30 secs)
- **Pierce:** A single bullet can pierce through enemies but not game objects (20 secs)
- **Extra bullet:** Adds an extra bullet to be fired

5)Levels / Waves:

- Level increases as the game progresses
- Fixed set of enemies per wave
- Next wave triggered if all enemies killed or none killed in the last 20 seconds
- Difficulty increases with increasing levels
- With each level - the number of enemies, enemy health, score points increases

4.4 Flow Chart

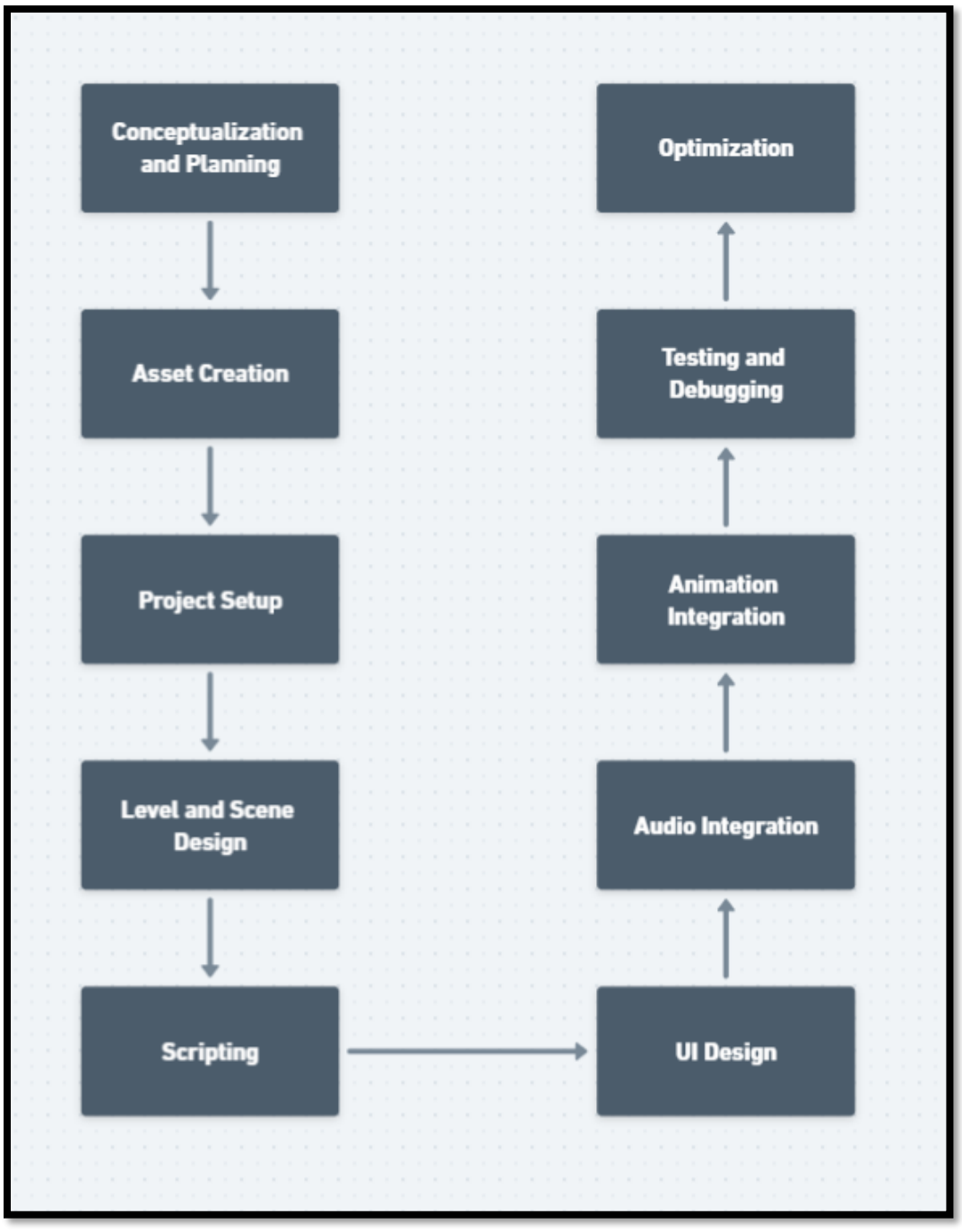


Fig.1. Flow Chart

5] SYSTEM REQUIREMENTS

5.1 Software Requirements

Software Requirements	
Operating System	Windows 7 or higher
Programming Language	C#
Software Application	Unity 3D
Code Editor	Visual studio code or any other suitable code editor

Table No 1. Software Requirements

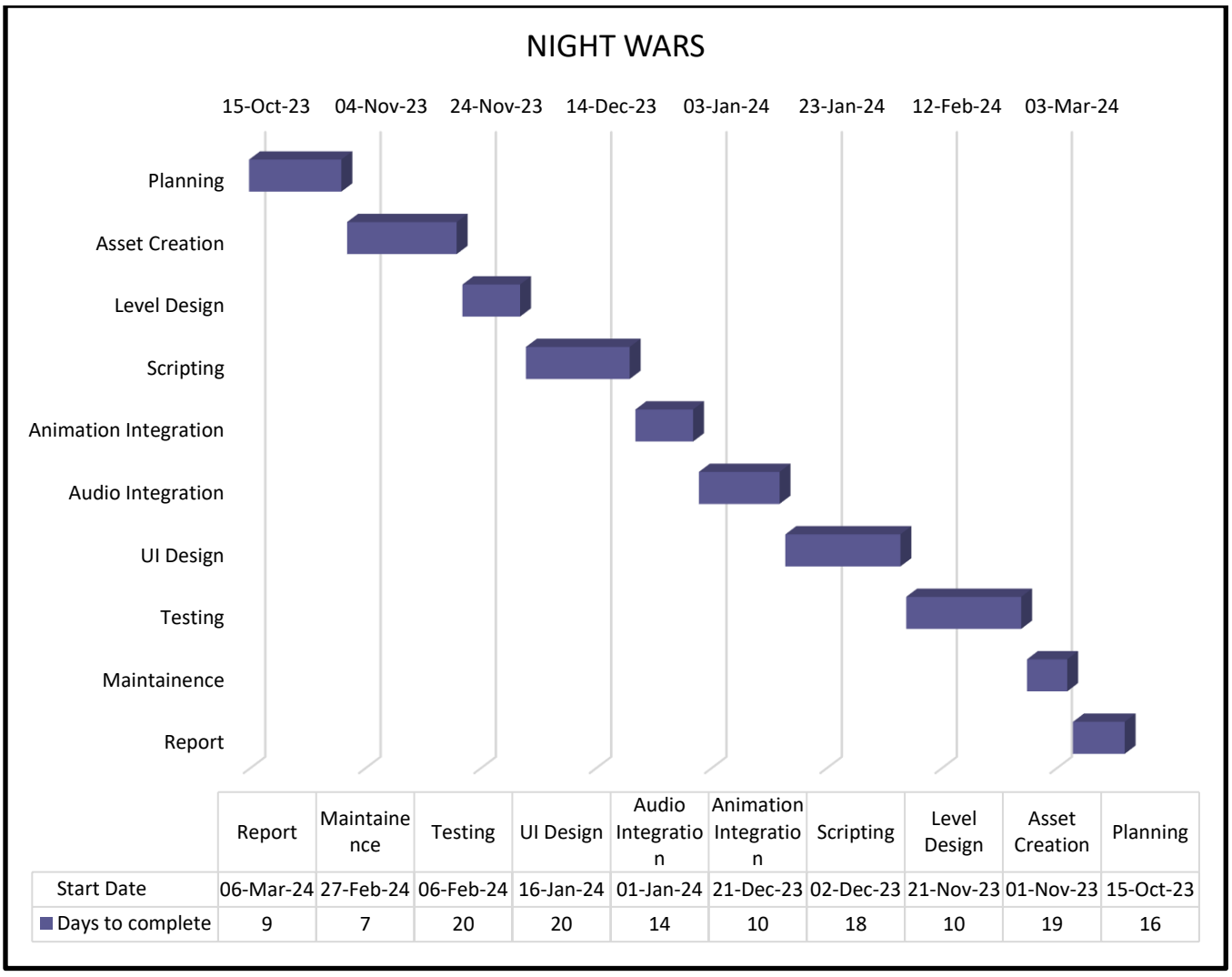
5.2 Hardware Requirements

Hardware Requirements	
System	Computer/Laptop with minimum 8GB RAM
Internet Connection	Stable Internet Connection

Table No 2. Hardware Requirements

6] SYSTEM PLANNING

6.1 Gantt Chart



	Report	Maintainence	Testing	UI Design	Audio Integration	Animation Integration	Scripting	Level Design	Asset Creation	Planning
Start Date	06-Mar-24	27-Feb-24	06-Feb-24	16-Jan-24	01-Jan-24	21-Dec-23	02-Dec-23	21-Nov-23	01-Nov-23	15-Oct-23
Days to complete	9	7	20	20	14	10	18	10	19	16

Fig.2. Gantt Chart

7] SYSTEM DESIGN

7.1 Use-Case Diagram

Use case diagrams are the diagrammatic representation depicting interaction with the system user's

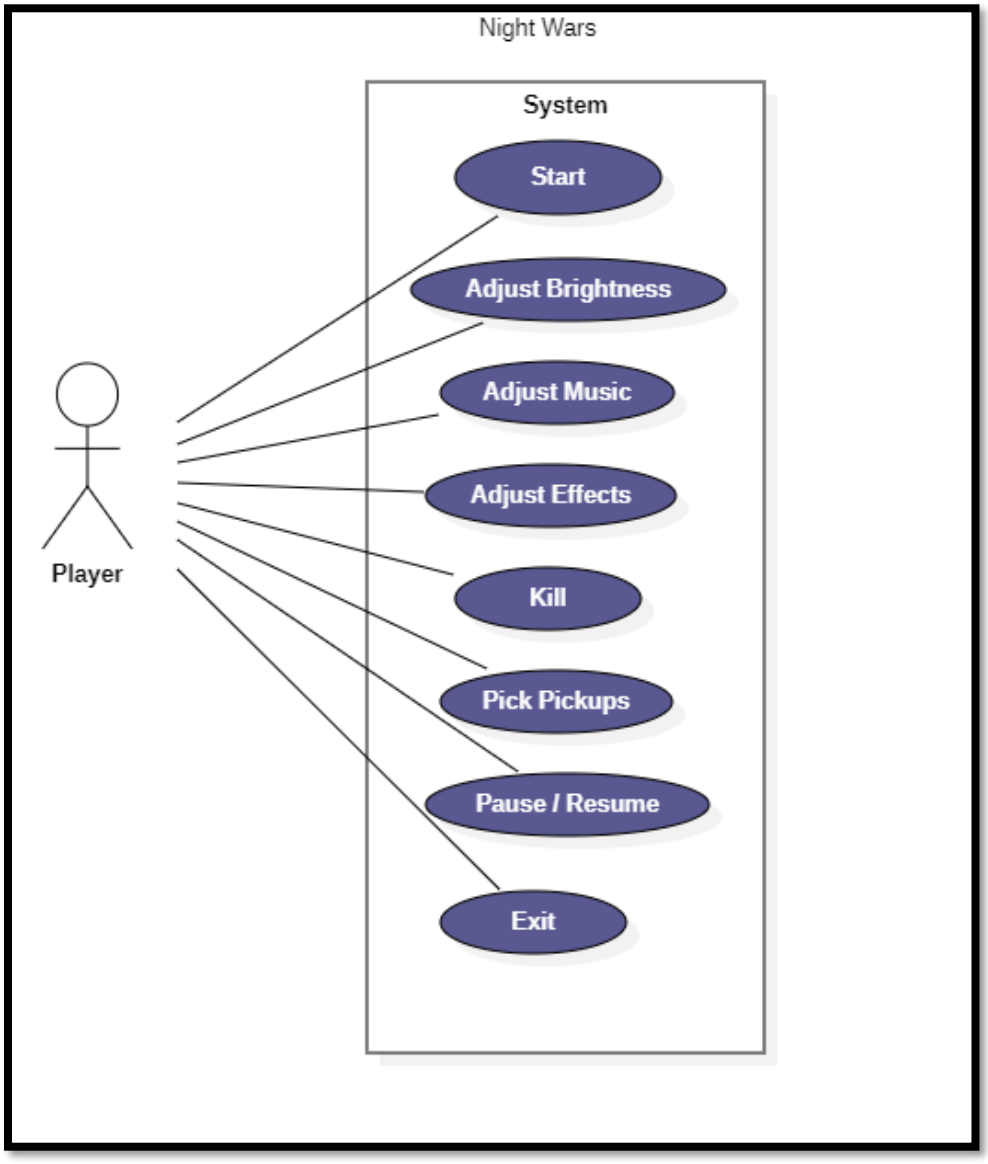


Fig.3. Use-Case Diagram

7.2 Activity Diagram

An activity diagram is a **behavioral diagram** i.e. it depicts the behavior of a system.
An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

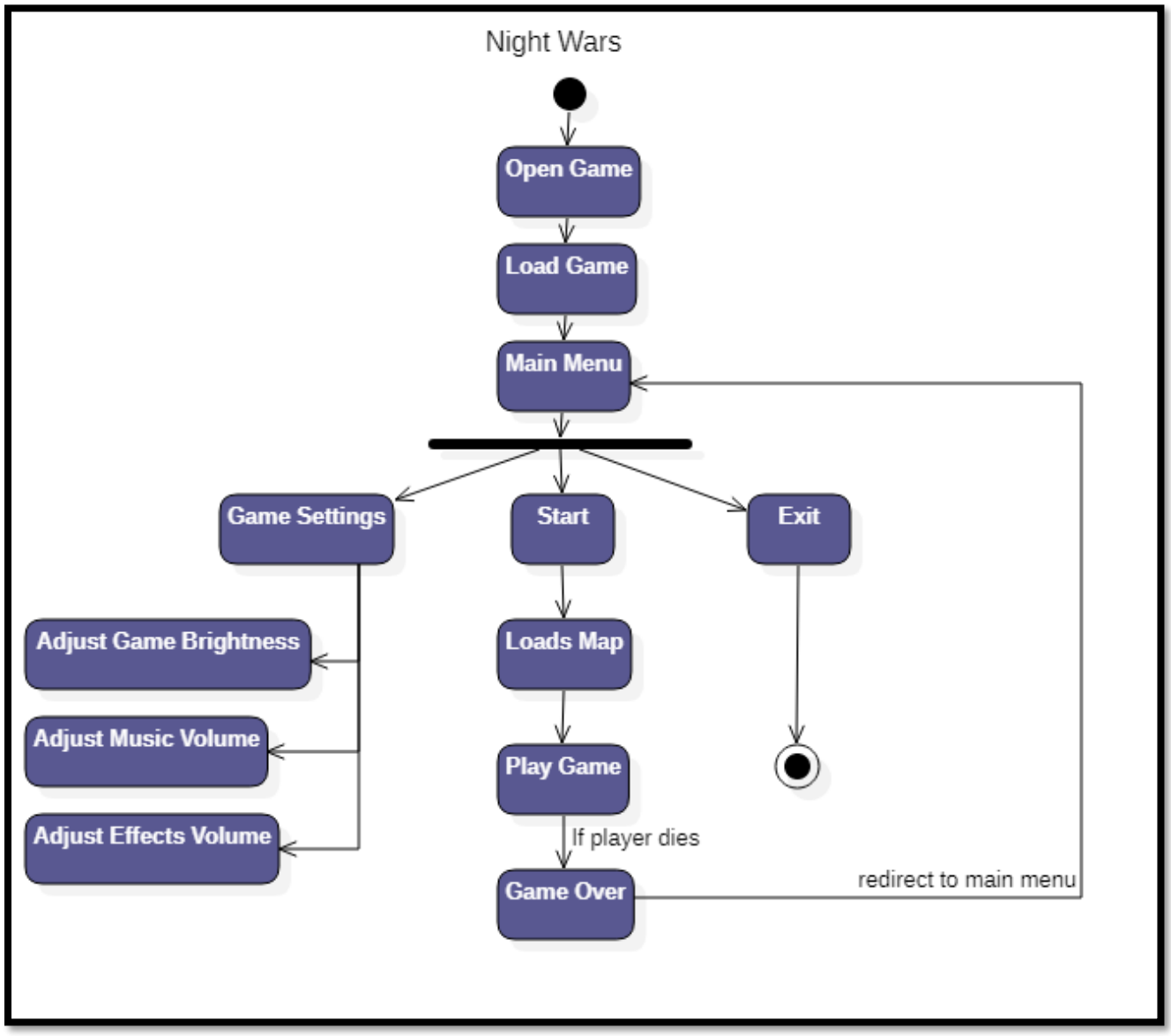


Fig.4. Activity Diagram

7.3 Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.

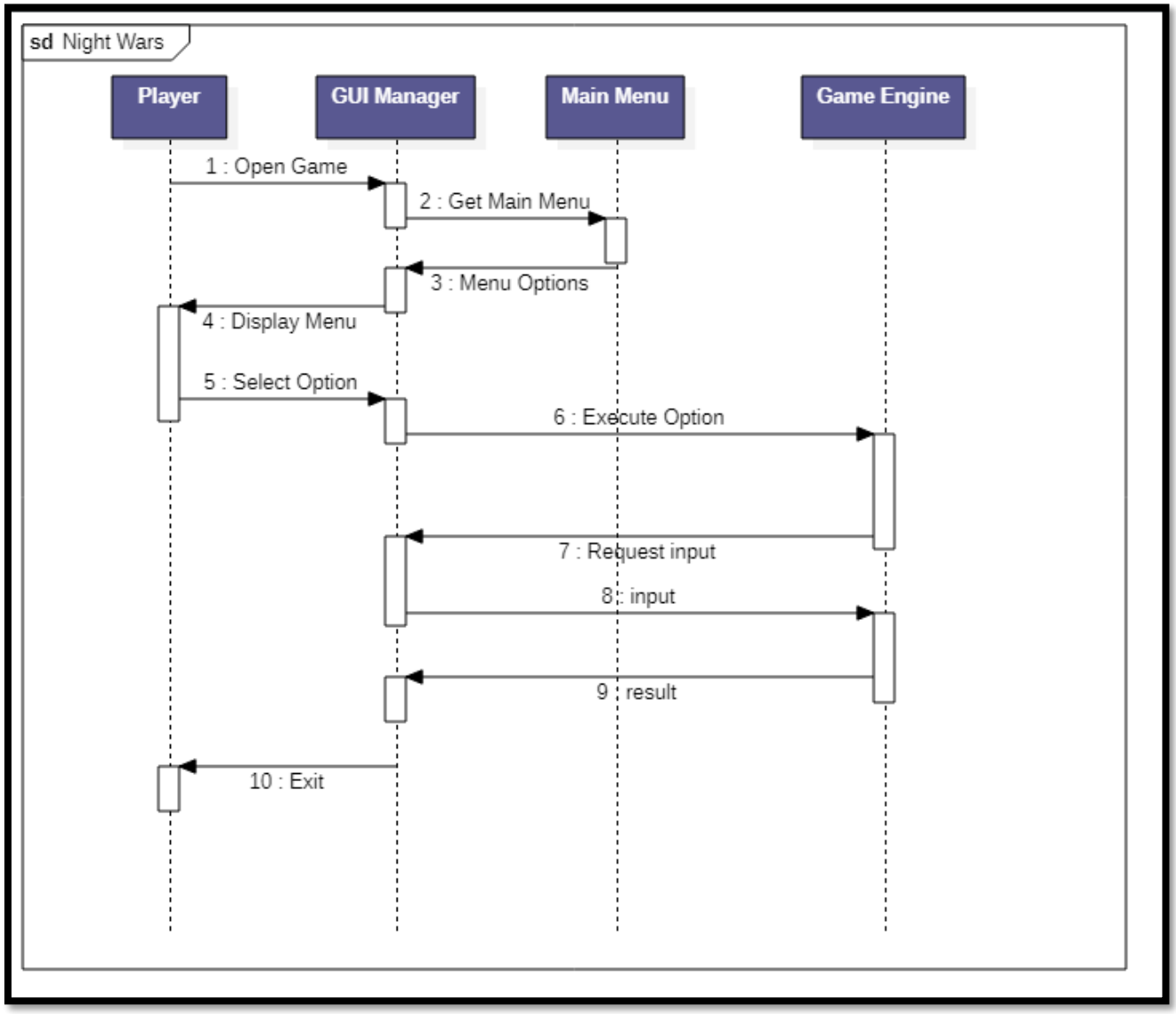


Fig.5. Sequence Diagram

7.4 Class Diagram

Class diagrams are the main building block of any object-oriented solution. It shows the classes in a system, attributes, and operations of each class and the relationship between each class. In most modeling tools, a class has three parts. Name at the top, attributes in the middle and operations or methods at the bottom. In a large system with many related classes, classes are grouped together to create class diagrams. Different relationships between classes are shown by different types of arrows.

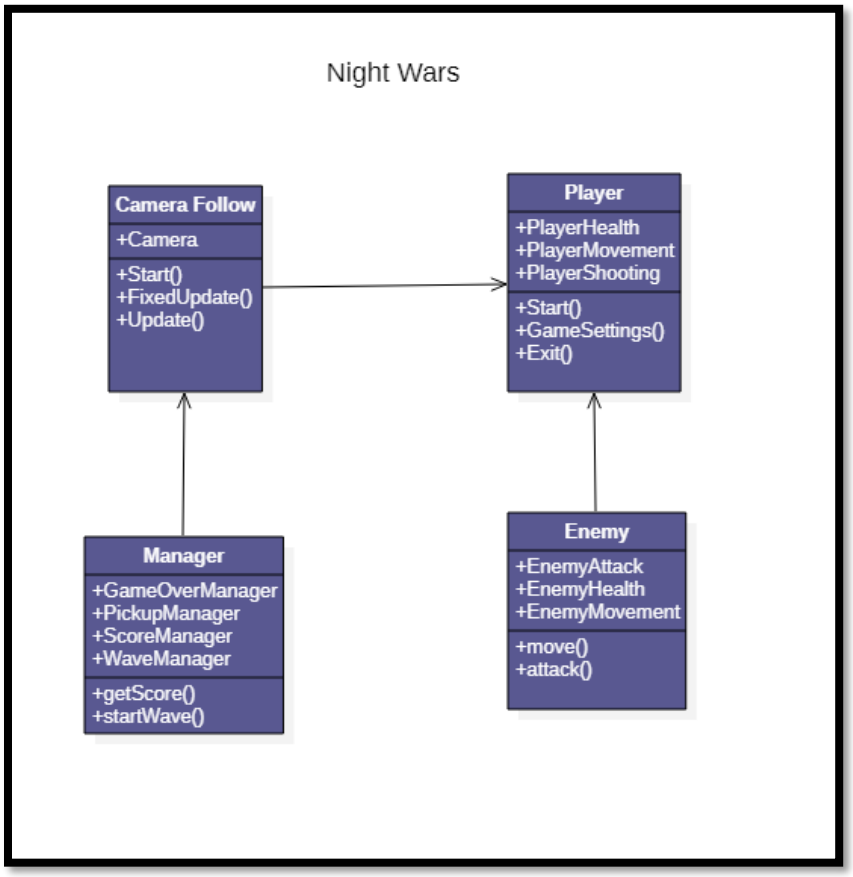


Fig.6. Class Diagram

8] Experimental Results and Analysis

8.1 Analysis

During the implementation , we have conducted various testing measures to assess the performance and user experience of our desktop game. This can include the following:

Gameplay Testing:
This involves thoroughly testing all aspects of the game's core mechanics, including character movement, combat systems, enemy AI behavior, and level design.
The goal is to ensure that gameplay is smooth, balanced, and engaging, providing players with a satisfying experience.
Testers may play through various levels or scenarios multiple times, trying different strategies and approaches to identify any issues or areas for improvement.

User Interface Testing:
User interface testing focuses on evaluating the usability and functionality of all interface elements, including menus, buttons, HUD elements, and in-game displays.
Testers assess whether the interface is intuitive and easy to navigate, whether all elements are properly aligned and displayed, and whether any text or icons are legible.
Feedback from UI testing helps developers refine and optimize the interface to enhance the overall user experience.

Bug Testing:

Bug testing involves identifying and addressing any glitches, bugs, or errors that may occur during gameplay.

Testers actively seek out issues such as game crashes, graphical glitches, scripting errors, or gameplay inconsistencies.

Bugs are assigned to developers for resolution, ensuring that the game is stable and functional across different platforms and hardware configurations.

Performance Testing:

Performance testing evaluates the game's performance metrics, including frame rate, loading times, memory usage, and overall optimization.

Optimization techniques such as code refactoring, asset optimization, and memory management are applied to improve performance and ensure a smooth gaming experience on a wide range of devices.

Performance testing is especially crucial for desktop games, where hardware variations can significantly impact gameplay performance.

All these measures help in testing, maintenance and optimization of the game.

8.2 Test Cases

Sr no	Test Cases	Expected Result	Status	Output	Test Result
1	Game Execution Check	Game should start and run correctly without any error	Performed	Working	Success
2	Main Menu Functionality Test.	All the main menu components should work properly	Performed	Working	Success
3	Character Movement Test	Character should move according to keys pressed	Performed	Working	Success
4	Character Rotation Test	Character should rotate according to mouse movement	Performed	Working	Success
5	Character and Enemy Animation Validation	Animations for character and enemies should function properly	Performed	Working	Success
6	Audio Audibility Test	Game music and Audio effects should be clearly audible	Performed	Working	Success
7	Graphics Rendering Validation	All Enemies and game objects are displayed accurately and without visual glitches	Performed	Working	Success
8	Game Termination Test	Game over screen should be visible when game ends	Performed	Working	Success

Table No 3. Test Cases

8.3 Results

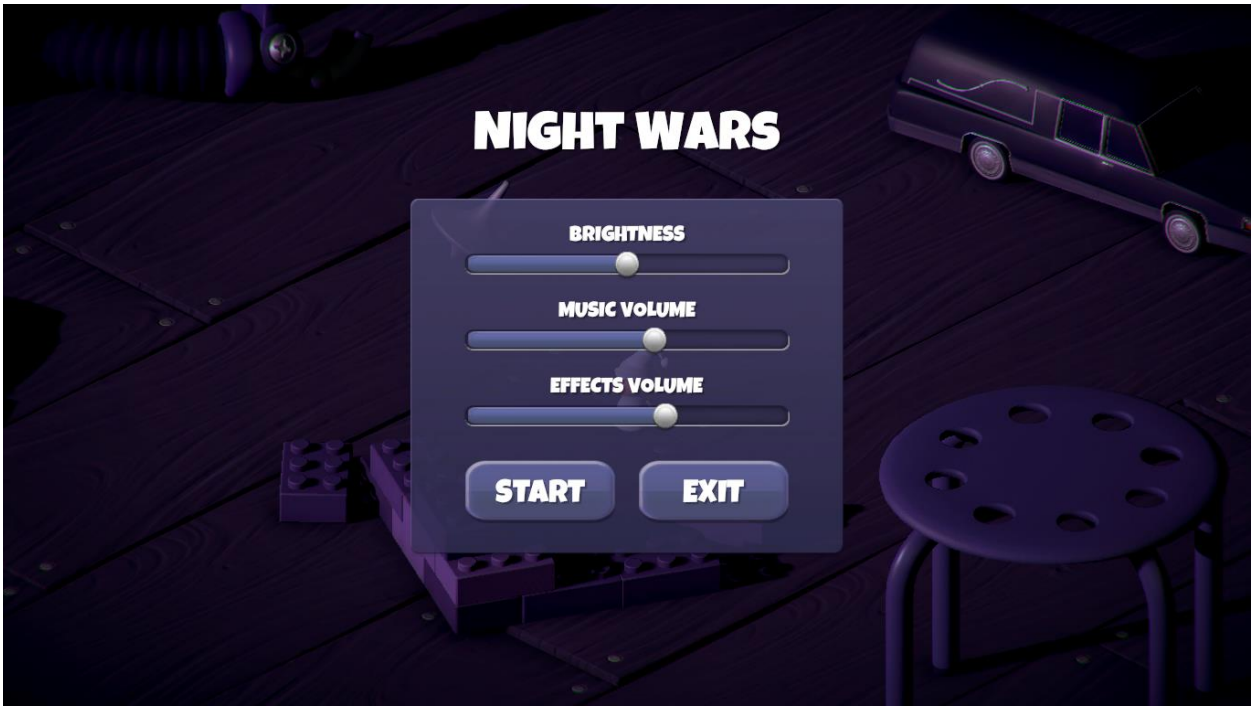


Fig.7. Main Menu

Fig.8. Player (3D Character)

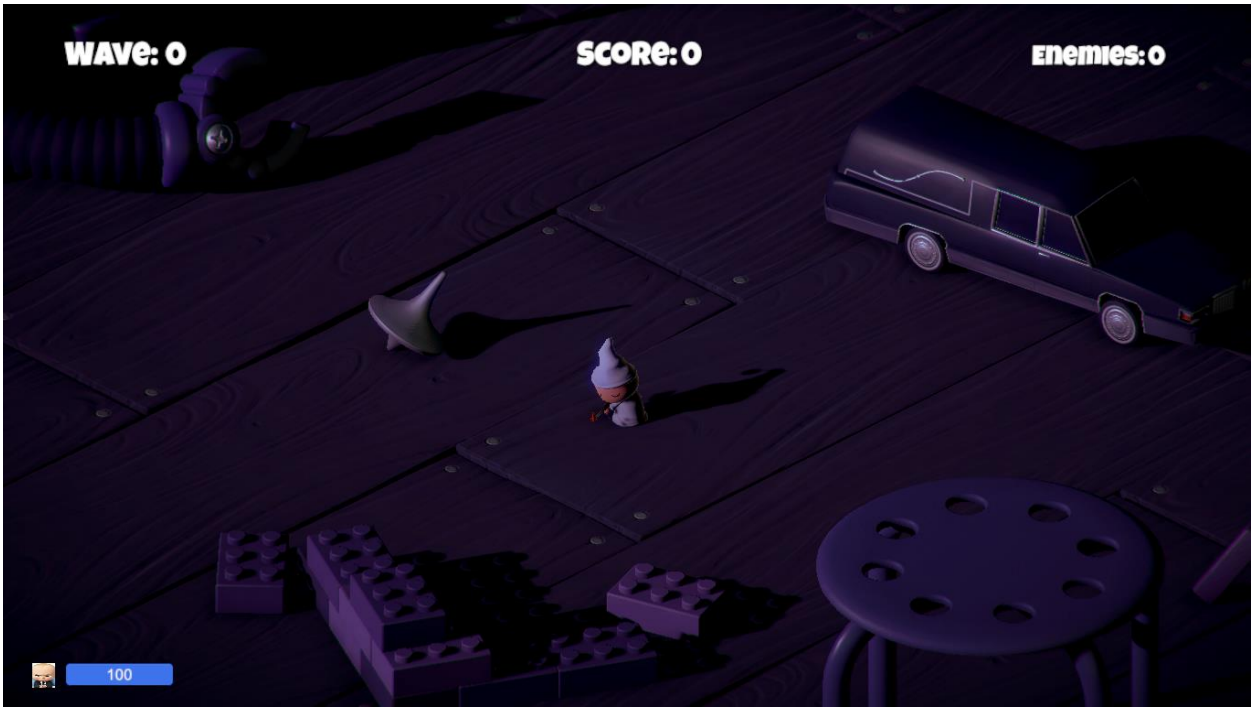


Fig.9. Game Enviroment

Fig.10. Shooting Enemies

Fig.11.Bounce Pickup

Fig.12.Bullet Pickup

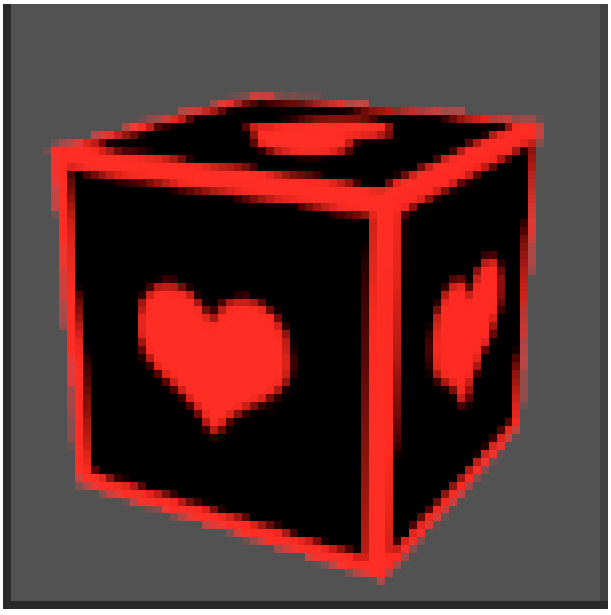


Fig.13.Health Pickup

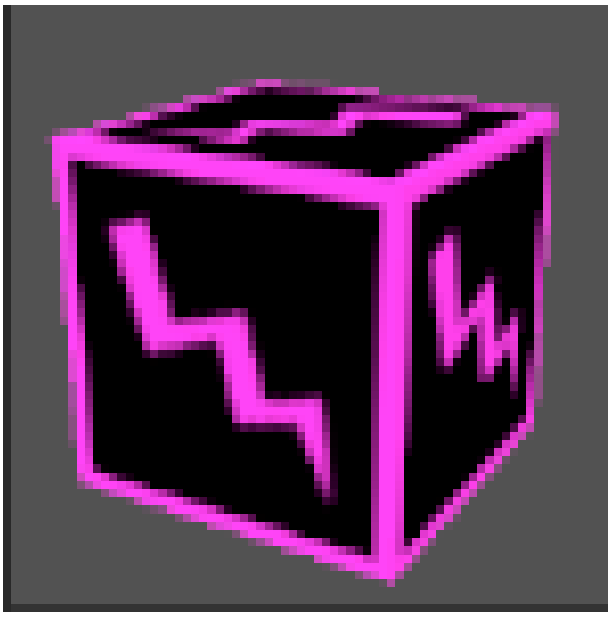


Fig.14.Pierce Pickup

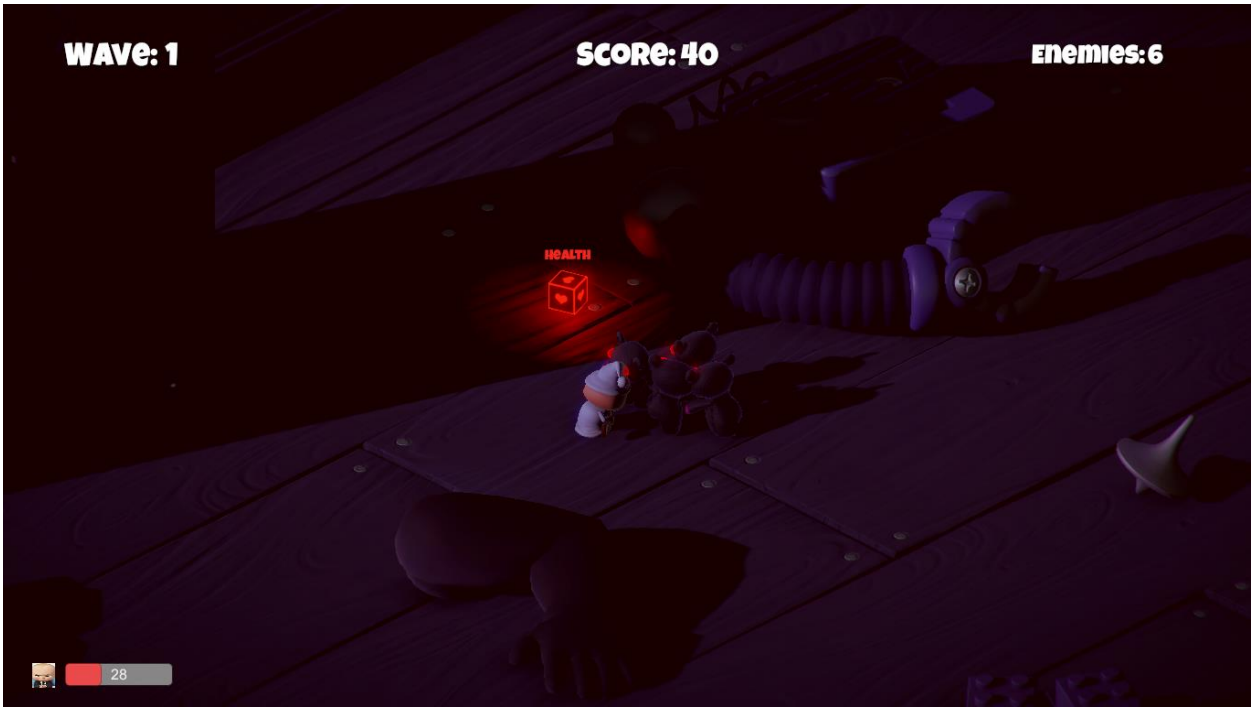


Fig.15.Spawning Pickups

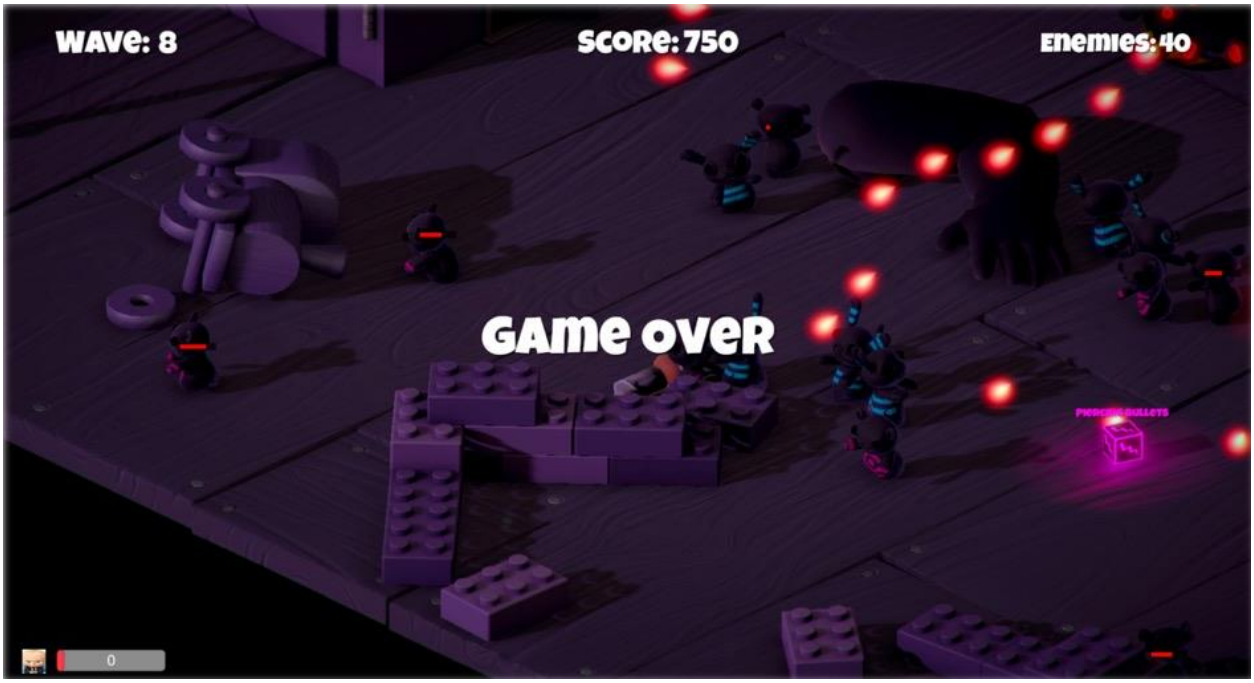


Fig.16. Game Over

9] Conclusion

9.1 Conclusion

In Conclusion, the aim of the "Night Wars" project is to develop an immersive and captivating Unity 3D desktop game in the survival shooter genre. The primary goal is to create a gaming experience that challenges players to eliminate enemies while emphasizing the crucial aspect of survival. Developed with C#, the project aims to provide an engaging and dynamic gameplay environment where players must strategically showcase their shooting skills to excel and endure within the game. The game's progressive difficulty, responsive character controls, and a variety of enemy types aim to ensure players are continually tested and rewarded for their strategies and skills. Ultimately, the project seeks to deliver a high-quality and enjoyable gaming experience that showcases the capabilities of Unity 3D in the development of interactive and challenging games.

10] Future Scope

10.1 Future Scope

Here are some potential future enhancements for our Unity 3D game development project:

Multiplayer Support:
This enhancement would involve implementing networking functionality to allow multiple players to connect and interact within the game world simultaneously. Depending on the game's design, multiplayer modes could include competitive modes like deathmatches or team-based gameplay, as well as cooperative modes where players work together to achieve objectives or overcome challenges.

Expanded Level Design:
Expanding the game's level design entails creating new environments or extending existing ones to offer players more areas to explore and conquer. This could involve designing new terrain, adding additional obstacles or enemies, and incorporating new mechanics or challenges to keep gameplay fresh and engaging.

Customization Options:
Introducing character customization options empowers players to personalize their in-game avatars according to their preferences. This could include choosing different outfits, accessories, or even abilities that affect gameplay. Providing a wide range of customization options enhances player immersion and encourages them to invest more time in the game.

Advanced AI:

Implementing more sophisticated artificial intelligence for enemies and non-player characters (NPCs) elevates the overall gameplay experience. Advanced AI can make opponents more strategic and challenging to defeat, creating more dynamic and immersive encounters for players. NPCs with realistic behaviors and responses can also enhance the game's narrative and immersion.

Enhanced Visual Effects:

Improving visual effects such as particle systems, shaders, and lighting can significantly enhance the game's aesthetics and immersion. This could involve adding more detailed and realistic effects to environmental elements, character animations, and special abilities, creating a visually stunning and immersive game environment that captivates players.

Additional Game Modes:

Introducing new game modes offers players different ways to experience and enjoy your game. Time trials can test players' speed and agility, survival challenges can push their endurance and resource management skills, while puzzle modes can stimulate their problem-solving abilities. Adding diverse game modes expands the game's replay value and keeps players engaged for longer periods.

11] References

[1] Unity Game Development Engine: A Technical Survey Afzal Hussain1 , Haad Shakeel1 , Faizan Hussain1 , Nasir Uddin1 , and Turab Latif Ghouri1

[2] Haas, John K. "A history of the unity game engine." (2014).

[3] Canossa, Alessandro. "Interview with nicholas francis and thomas hagen from unity technologies." In-Game Analytics, pp. 137-142. Springer, London, 2013.

[4] Kim, Sung Lee, Hae Jung Suk, Jeong Hwa Kang, Jun Mo Jung, Teemu H. Laine, and Joonas Westlin. "Using Unity 3D to facilitate mobile augmented reality game development." In 2014 IEEE World Forum on the Internet of Things (WF-IoT), pp. 21-26. IEEE, 2014.

[5] Patil, Pratik P., and Ronald Alvares. "Cross-platform Application Development using Unity Game Engine." Int. J 3, no. 4 (2015).

[6] Blackman, Sue. Beginning 3D Game Development with Unity 4: All-in-one, multi-platform game development. Apress, 2013.

[7] Hocking, Joseph. "Unity in Action." Manning Publications, 2018.

[8] Ferrone, Harrison. "Learning C# by Developing Games with Unity 2020." Packt Publishing, 2020.

[9] Schell, Jesse. "The Art of Game Design: A Book of Lenses." CRC Press, 2008.

[10] Rogers, Scott. "Level Up! The Guide to Great Video Game Design." Wiley, 2010.

12] Appendix

Player Movement Script

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour {

        public float speed = 6f;
        Vector3 movement;
        Animator anim;
        Rigidbody playerRigidbody;
        int floorMask;
        float camRayLength = 100f;

        void Awake() {
            floorMask = LayerMask.GetMask("Floor");
            anim = GetComponent<Animator>();
            playerRigidbody = GetComponent<Rigidbody>();
        }

        void FixedUpdate() {
            float h = Input.GetAxisRaw("Horizontal");
            float v = Input.GetAxisRaw("Vertical");
            Move (h, v);
            Turning ();
            Animating (h, v);
        }

        void Move(float h, float v) {
            movement.Set (h, 0f, v);
            movement = movement.normalized * speed * Time.deltaTime;
            playerRigidbody.MovePosition(transform.position + movement);
        }

        void Turning() {
            Ray camRay = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit floorHit;

            if (Physics.Raycast (camRay, out floorHit, camRayLength, floorMask)) {
                Vector3 playerToMouse = floorHit.point - transform.position;
                playerToMouse.y = 0f;

                Quaternion newRotation = Quaternion.LookRotation(playerToMouse);

                playerRigidbody.MoveRotation(newRotation);
            }
        }

        void Animating(float h, float v) {
            bool walking = h != 0f || v != 0f;
            anim.SetBool("IsWalking", walking);
        }
}
```

Player Shooting Script

```
using UnityEngine;
using UnityEngine.UI;

public class PlayerShooting : MonoBehaviour {

    public Color[] bulletColors;
    public float bounceDuration = 10;
    public float pierceDuration = 10;
    public int damagePerShot = 20;
    public int numberOfBullets = 1;
    public float timeBetweenBullets = 0.15f;
    public float angleBetweenBullets = 10f;
    public float range = 100f;
    public LayerMask shootableMask;
    public Image bounceImage;
    public Image pierceImage;
    public GameObject bullet;
    public Transform bulletSpawnAnchor;
       public GameObject pierceTimerObj;
       public GameObject bounceTimerObj;

    float timer;
    Ray shootRay;
    RaycastHit shootHit;
    ParticleSystem gunParticles;
    LineRenderer gunLine;
    AudioSource gunAudio;
    Light gunLight;
    float effectsDisplayTime = 0.2f;
    float bounceTimer;
    float pierceTimer;
    bool bounce;
    bool piercing;
    Color bulletColor;

    public float BounceTimer {
        get { return bounceTimer; }
        set { bounceTimer = value; }
    }

    public float PierceTimer {
        get { return pierceTimer; }
        set { pierceTimer = value; }
    }

    void Awake() {
        gunParticles = GetComponent<ParticleSystem>();
        gunAudio = GetComponent<AudioSource>();
        gunLight = GetComponentInChildren<Light>();
        bounceTimer = bounceDuration;
        pierceTimer = pierceDuration;
    }

    void Update() {
              bounceTimerObj.SetActive(false);
```

```
            pierceTimerObj.SetActive(false);

        if (bounceTimer < bounceDuration) {
            bounce = true;
        }
        else {
            bounce = false;
        }

        if (pierceTimer < pierceDuration) {
            piercing = true;
        }
        else {
            piercing = false;
        }

        bulletColor = bulletColors[0];
        if (bounce) {
                    bounceTimerObj.SetActive(true);
                    Text bounceTime = bounceTimerObj.GetComponent<Text> ();
                    float floatVal = bounceDuration - bounceTimer;
                    int val = Mathf.CeilToInt(floatVal);
                    bounceTime.text = val.ToString ();
            bulletColor = bulletColors[1];
            bounceImage.color = bulletColors[1];
        }
        bounceImage.gameObject.SetActive(bounce);

        if (piercing) {
                    pierceTimerObj.SetActive(true);
                    Text pierceTime = pierceTimerObj.GetComponent<Text> ();
                    float floatVal = pierceDuration - pierceTimer;
                    int val = Mathf.CeilToInt(floatVal);
                    pierceTime.text = val.ToString ();
            bulletColor = bulletColors[2];
            pierceImage.color = bulletColors[2];
        }
        pierceImage.gameObject.SetActive(piercing);

        if (piercing & bounce) {
            bulletColor = bulletColors[3];
            bounceImage.color = bulletColors[3];
            pierceImage.color = bulletColors[3];
        }

            var main = gunParticles.main;
            main.startColor = bulletColor;
        gunLight.color = (piercing & bounce) ? new Color(1, 140f / 255f, 30f / 255f, 1) :
bulletColor;
        bounceTimer += Time.deltaTime;
        pierceTimer += Time.deltaTime;
        timer += Time.deltaTime;


        if (Input.GetButton("Fire1") && timer >= timeBetweenBullets && Time.timeScale !=
0) {
            Shoot();
        }
```

```
        if (timer >= timeBetweenBullets * effectsDisplayTime) {
            DisableEffects();
        }
    }

    public void DisableEffects() {
        gunLight.enabled = false;
    }

    void Shoot() {
        timer = 0f;
        gunAudio.pitch = Random.Range(1.2f, 1.3f);
        if (bounce) {
            gunAudio.pitch = Random.Range(1.1f, 1.2f);
        }
        if (piercing) {
            gunAudio.pitch = Random.Range(1.0f, 1.1f);
        }
        if (piercing & bounce) {
            gunAudio.pitch = Random.Range(0.9f, 1.0f);
        }
        gunAudio.Play();

        gunLight.intensity = 2 + (0.25f * (numberOfBullets - 1));
        gunLight.enabled = true;
        gunParticles.Stop();
              var main = gunParticles.main;
        main.startSize = 1 + (0.1f * (numberOfBullets - 1));
        gunParticles.Play();
        shootRay.origin = transform.position;
        shootRay.direction = transform.forward;

        for (int i = 0; i < numberOfBullets; i++) {
            float angle = i * angleBetweenBullets - ((angleBetweenBullets / 2) *
(numberOfBullets - 1));
            Quaternion rot = transform.rotation * Quaternion.AngleAxis(angle,
Vector3.up);
            GameObject instantiatedBullet = Instantiate(bullet,
bulletSpawnAnchor.transform.position, rot) as GameObject;
            instantiatedBullet.GetComponent<Bullet>().piercing = piercing;
            instantiatedBullet.GetComponent<Bullet>().bounce = bounce;
            instantiatedBullet.GetComponent<Bullet>().bulletColor = bulletColor;
        }
    }
}
```

Enemy Attack Script

```
using UnityEngine;
using System.Collections;

public class EnemyAttack : MonoBehaviour {

	public float timeBetweenAttacks = 0.5f;
	public int attackDamage = 10;
	GameObject player;
	PlayerHealth playerHealth;
	EnemyHealth enemyHealth;
	bool playerInRange;
	float timer;

	void Awake() {
		player = GameObject.FindGameObjectWithTag("Player");
		playerHealth = player.GetComponent<PlayerHealth>();
		enemyHealth = GetComponent<EnemyHealth>();
	}

	void OnTriggerEnter(Collider other) {
		if (other.gameObject == player) {
			playerInRange = true;
			timer = 0.2f;
		}
	}


	void OnTriggerExit(Collider other) {
		if (other.gameObject == player) {
			playerInRange = false;
		}
	}


	void Update() {
		timer += Time.deltaTime;

		if (timer >= timeBetweenAttacks && playerInRange &&
enemyHealth.currentHealth > 0 && playerHealth.currentHealth > 0) {
			Attack();
		}
	}


	void Attack() {
		timer = 0f;
		playerHealth.TakeDamage(attackDamage);
	}
}
```

Score Manager Script

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

public class ScoreManager : MonoBehaviour {

	int score;
	public Text number;

	void Awake() {
		score = 0;
	}

	void Update() {
		number.text = score.ToString();
	}

	public void AddScore(int toAdd) {
		score += toAdd;
		number.GetComponent<Animation>().Stop();
		number.GetComponent<Animation>().Play();
	}

	public int GetScore() {
		return score;
	}
}
```