3D Game Development in UNITY

Prof. (Ms). Arti Sondawale[1], Mrunal Wankhede[2]
[1, 2] Dept of Robotics & AI
[1, 2] Priyadarshini College of Engineering, Nagpur

Abstract- *This project looks at how to create an engaging single-player game stressing smooth controls and real-time interaction developed in Unity game Engine. A responsive gaming experience is possible for players who can move smoothly between various states including walking, running, crouching, and standing still. The game features a countdown timer indicating the conclusion of a match and a restart choice, as well as systems for managing enemy spawning and scoring. Weapons make fight to seem dynamic and varied by allowing both semi-automatic and full-automatic shooting modes.*

An Easy to understand and friendly User Interfae lets players know with live updates on their score and left time. All things considered, the game offers an engaging and immersive experience totally created in Unity by combining good mechanics with a simple UI.

Keywords- 3D Unity, UNITY, Game Development, Blender, C-Sharp, Game Development in Unity

I. INTRODUCTION

Not just in terms of players and income but also in the tools accessible for developers, the gaming sector has seen extraordinary expansion in recent years. Among the most remarkable engines propelling this growth is Unity. Known for its flexibility, cross-platform compatibility, and straightforward design, Unity has become a well-liked choice for both experienced and new game creators. Its large asset ecosystem and C# scripting let complicated 2D and 3D games be built with reasonable simplicity in a strong development environment.

The focus of this research project is creating a 3D single-player game that is entirely powered by the Unity engine.The goal of the project is to design a useful and playable gaming environment made up of required game mechanics including character movement, animations, a shooting system, enemy spawning, score tracking, timing management, and a user-friendly interface. Though the game concept is simple, it catches key elements required to understand how a whole game comes together—from the programming of behaviours to UI integration and player interaction.Using a state machine approach, the player character is meant to move in different states: walking, running, crouching, and idle. Implemented is a shooting mechanism whereby bullets are created and driven towards targets. Randomly generated within a specified area, foes interact with the shooting system to raise the player's score upon being hit. With clear transitions between the main menu, gameplay, and restart screen, the user interface system displays the score and remaining time.

This paper describes the structure, logic, and flow of the game's parts by looking at and linking a number of scripts that describe how they work. More than graphics or advanced AI, the attention is on how useful it is and how people can interact with it. With this project's method, you can learn how to make and handle user interfaces, enemy behavior, shooting, input, movement, and other separate systems all within a single Unity project.

The study wants to give would-be developers a clear way to learn how to make Unity games, understand how to control code design, and see how different systems work together to make a single play experience.

II. LITERATURE REVIEW

3D game development has become a vibrant sector combining creative design with technical accuracy as interactive entertainment and immersive experiences gain traction. Building a viable 3D shooting game using Unity not only allows developers to explore these evolving technologies but also offers a way to understand industry-grade processes. The main focus of this work is real-time shooting mechanics, score-based feedback systems, and time-bound gameplay—all of which are smoothly integrated using user-friendly UI elements.

Its total end-to-end functionality distinguishes this project: from asset interaction to responsive UI, all done in a standalone offline mode without networking or multiplayer features. This means a concentrated investigation of single-player game design. Tracking performance in real time—bridging technical programming with interesting game mechanics—the game lets users aim, shoot, and get instant visual and audio feedback.

Creating such a project uses various fundamental ideas and tools, all of which have been extensively researched and honed in prior game development literature and research. These consist of 3D modelling, animation systems, Unity scripting, and UI development. The following parts discuss the main elements used to create this kind of game.

A. 3D Modelling:

Any visual game element is built on 3D modelling. Current studies stress programs like Blender for producing low-poly, optimized assets fit for real-time engines like Unity. Game-ready asset design is all about mesh topology, poly count, and export formats including FBX.

B. Rigging:

Rigging gives models a skeleton structure, therefore allowing animation. Efficient rigging lets realistic object or character movement. Rigging is still crucial for any future growth into animated targets or player characters even if your game emphasizes target shooting over character animation.

C. UV Unwrapping and Texturing:

Correct UV unwrapping guarantees that textures map properly on 3D surfaces. Researches underline this stage for effective rendering and visual consistency. Your project targets well-defined surfaces, so allowing the use of different materials and shaders visible during play.

D. Animation and State Machines:

Animator Controllers and state machines drive Unity animations. These control things like gun recoil or target movements. Although minimal animation is used in this project, understanding this system enables scalable development for future enhancements
.

E. Scripting and Game Logic:

Shooting logic, audio feedback, game states, scoring, and timers are controlled in this project using modular C# scripts. Literature emphasizes the need of systematic scripting, and your work adheres to organized code distribution across managers, therefore improving readability and efficiency.

F. User Interface (UI) and UX Design:

Score, time, and control of game states (start, quit, restart) are shown using Unity's UI system. Studies indicate that user involvement is directly influenced by UI clarity. This project combines simple, engaging user interface components to enable seamless player interaction without stressing the user.

G. Real-Time Feedback and Optimization:

User experience studies back up the method shown by the addition of audio clips for gunfire and visual score updates, which offer instant feedback. Unity's engine is perfect for standalone projects like this since it allows for lightweight yet reactive performance for all these systems.

H. Physics and Collision Systems:

Handling realistic motion and interactions depends much on Unity's built-in physics engine. Projectile behavior, impact detection, and development of a responsive shooting experience all depend on rigidbody components and colliders. This guarantees that interactions with targets are significant and that bullets act physically correctly.

I. Audio Integration:

Good sound design increases immersion. Your work makes use of Unity's AudioSource and AudioClip system for gunfire sounds. Especially in fast-paced games, literature underlines how audio feedback supports player actions and helps to create a more interesting atmosphere.

J. Camera and Aiming Mechanics:

Your shooter game is mostly about exact aiming. Studies on first- and third-person shooter mechanics back up ray casting’s use for targeting and camera alignment. Often, this part of the literature investigates field of view optimization, aim assist, and camera smoothing—all of which shape the fundamental interaction loop of your game.

K. Game State Management:

It's vital to handle various game states—starting, playing, pausing, and restarting. Researches underline the need of building a responsive state machine architecture. Your project demonstrates this via UI interactions that reveal/hide panels, reset score, and control timers depending on player input.

L. Performance Efficiency:

Making Optimized code helps even tiniest of game. By avoiding needless instantiations and dividing logic into modular parts, your scripts maximize performance.

III. PROPOSED METHOD

We will now discuss the way adopted to construct the 3D game using Unity, which makes use of various essential tools and techniques. The suggested approach emphasizes integrating all the assets into Unity, modelling, texturing, rigging, animation, and so forth. Creating an interactive 3D game with fluid character movements, realistic textures, and interesting animations all while guaranteeing smooth play is a main feature of this project. The following subtopics investigate all relevant aspects of this development process.

Technology Stack and Implementation:

The technology stack for this project comprises several potent tools and platforms, each fulfilling a particular function in the game development pipeline:

Fig 1: Technology Stack and Implementation

- **Blender**: For texturing, rigging, and 3D modelling. It enables thorough design of both functional and aesthetically pleasing characters and assets.
- **Unity**: The game development engine integrating all assets. Unity offers player interactivity, physics simulations, and real-time rendering.
- **C#**: Scripting in Unity to manage player movement, interactions, and game logic uses this.
- **Adobe Illustrator or GIMP**: To produce 2D assets and textures to be mapped on 3D models.
- **Visual Studio**: To debug C# scripts to develop games

Each component in this stack plays a vital role in developing a seamless and enjoyable 3D game. The approach seeks to integrate all these technologies to produce a consistent game with an interesting frontend and strong backend.

A. Modelling in Blender:

Perfect for creating models for games, Blender is a free and open-source 3D creation suite. Blender was used often in this project to produce 3D models including people, objects, and settings. Starting with simple forms, the modelling process is sculpted to fit the design specifications. When striving for realistic in-game items, Blender's strong mesh-editing tools enable great accuracy, which is essential.

Blender's key features are its capacity to handle complicated meshes, which enables it to produce characters with detailed complexity. Blender also supports several export formats, therefore guaranteeing that models could easily fit into Unity. Blender's capacity to manage both hard and soft surface modelling guarantees that all from weapons to environmental items may be designed properly.

The modelling process calls for a balance between aesthetics and performance. Designed for real-time rendering, the models guarantee good Unity performance without performance bottlenecks.

B. UV Unwrapping:

UV Unwrapping is method used in 3D modelling to surface the 3D object into 2D plane for painting the object. This is done by unwrapping the 3D mesh into a 2D plan that can then be painted with textures. UV unwrapping's significance is in its capacity to guarantee that all areas of the model get correct and high-quality application of textures.

Created in Illustrator (or GIMP as an alternative), the textures for this project were applied to the Blender models. High-quality textures help the models to be more realistic, therefore enhancing the general look of the game. Essential for immersive gaming, texturing provides visual detail to the game and may include surface characteristics including skin, fabric, and metallic finishes.

Especially in a 3D game where many models are rendered concurrently, the main difficulty in this stage was optimizing textures to guarantee they do not harm game performance. Efficient UV unwrapping also enables better performance since less texture maps have to be loaded into memory at once.

C. Rigging and Importance of Anatomy:

Rigging is the technique of building a skeletal framework for three-dimensional models so they may be animated. Character rigging—the process of building a skeleton with bones and joints that can be controlled during animation—is a key component of this work. Blender's rigging system enables the creation of both human and non-human characters with realistic motions.

Especially when working with humanoic characters, the significance of anatomy in rigging cannot be emphasized. Realistic movement of characters depends on bone placement following their anatomy. Wrong rigging can cause unnatural motions that could ruin the game. This project paid particular attention to the precise placement of joints including the elbow, knee, and shoulder as well as facial rigging for expressions.

Furthermore, correct rigging raises the caliber of the animation. A smoother rig makes the animations more natural. Blender offers great weight painting tools to make sure the mesh deforms properly as the bones move. Animation is built on this stage; thus it is absolutely vital.

D. Animation:

Game development relies on animation to bring people and objects to life. Animation comes after rigging. Blender was used in this project to produce keyframe animations for people and objects. Among these animations are shooting, running, walking, and others. The animation technique uses Blender's interpolation system to fill in the frames between keyframes by first specifying positions for the character or object at particular time intervals.

Later animation are brough inside Unity and run under an Animator Controller. To guarantee that the player's actions, like pressing movement keys, directly correspond with the character's movements, the animations have to be smooth and responsive. Proper animation methods, including blend trees in Unity, let you smoothly transition between various animations, such as changing from walking to running.

E. Integrating Everything in Unity:

Unity is the tool used to combine the 3D models, textures, rigging, and animations into a running game. Assets are imported into Unity following their creation and export from Blender. Unity provides a simple interface for placing models in 3D space and changing their properties.

Game mechanics—including player movement and enemy interaction, also lsoshooting—are controlled by C# scripting. C#'s implementation in Unity offers dynamic and flexible gaming. Player input drives several animations via Unity's Animator Controller, therefore guaranteeing interactive and responsive gameplay.

Unity also guarantees optimal display of the visual components of the game across several devices by means of real-time rendering and lighting. The integration process guarantees that all assets interact smoothly and that the game logic—player actions, UI updates, and scoring—operates perfectly.

F. UI and Its Importance:

The User Interface (UI) provides vital information like score, health, and time left, so significantly influencing the player's experience. Designed to be simple and intuitive, the UI in this project guarantees that users may concentrate on the actual game without extraneous distractions. The main elements of the UI are:

a. **Score Display**: Continuously updates as players shoot enemies.
b. **Time Remaining**: A countdown timer that adds pressure to the gameplay.
c. **Restart Panel**: Displays the final score and allows players to restart the game.

Unity's built-in UI system, combined with **TextMeshPro** for text rendering, was used to create the UI elements. The UI is also responsive, adjusting based on screen resolution to ensure the game looks great on different devices.

IV. IMPLEMENTATION

The implementation phase is when the ideas, methods, and tools detailed in the Proposed Method are actually used to produce a functional 3D game. This project's implementation creates an interactive and immersive 3D environment by means of a systematic workflow integrating modelling, texturing, rigging, animation, and game mechanics under Unity. We will investigate the processes included in the execution below, together with important obstacles and answers.

A. Asset Creation and Import

The first step in the implementation was creating 3D assets in Blender. This included modelling characters, enemies, and environmental objects. The process was divided into two main phases: asset creation and optimization.

a. **Asset Creation**: While keeping an ideal polygon count to guarantee smooth performance in Unity, characters and objects were modelled with a focus on detail. Using programs like Illustrator and Photoshop, the models were then UV unwrapped and textured to provide realistic surface detail.
b. **Optimization**: After being built and textured, the models were optimized to run effectively inside the Unity engine. This meant making sure no unneeded details were included that would tax the system's resources, baking textures to lower-

resolution maps, and lowering the polygon count as needed.

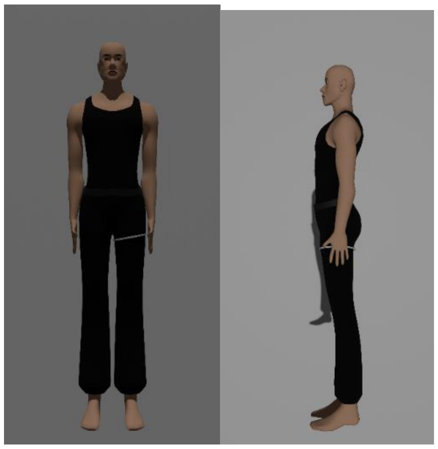


Fig 2: 3D Model Creating in Blender

These assets were then brought into Unity and their settings changed to make sure they fit right in the game environment. Correctly scaled and positioned in the 3D space, the models were ready for the next stage: animation and rigging.

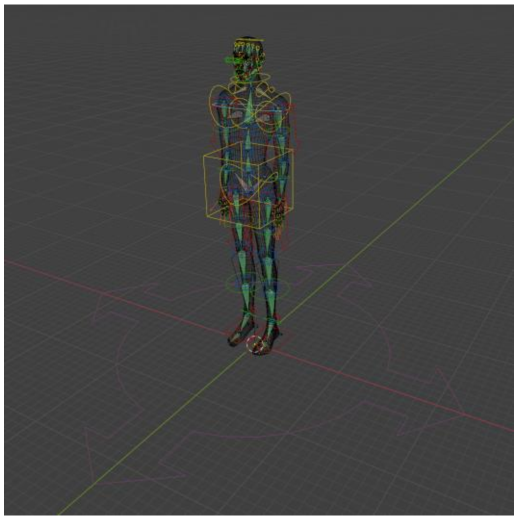
B. Rigging and Animation Integration

The next stage was to combine the rigging and animations once the assets were imported into Unity. Characters' skeletons were made using Blender's rigging system, which comprises bones and weight painting. Rigging completed, animations for fundamental activities like walking, running, shooting, and idle poses were produced.

a) **Rigging in Blender**: Blender's armature system let users exactly place bones and joints, therefore rigging the characters. The characters were then painted with weight to guarantee proper mesh deformation during animations.

b) **Animation Integration in Unity**: Blender's FBX files were exported as animations and brought into Unity. Based on player input—such as running or jumping—Unity's Animator Controller was used to arrange and manage the transitions between animations.

The integration of animations into the Unity project allowed for seamless character movements, where actions like shooting or jumping directly correlated with the character’s in-game behavior.



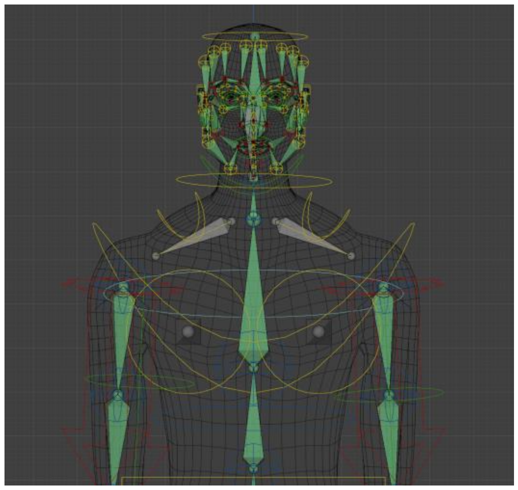


Fig 3: Rigging of Model

C. Game Logic and Mechanics with C#

Game mechanics—programmed in C# in Unity—implementation defines the heart of the play. The UI, shooting mechanics, enemy interactions, and player movements were all controlled by C# scripts. These scripts handled key game mechanics including scoring, health, and time.

a. **Movement and Shooting**: Scripts were created to identify player input for movement and firing. A projectile is shot toward the target and the appropriate animation runs when the player clicks the mouse or presses a key.

b. **Collision Detection**: A script was also run to find collisions between enemy objects and projectiles. A collision causes the enemy to die completely or their health to drop.

c. **Game Over and Restart Logic**: The game goes into "game over" mode when the timer hits zero or the player runs out of health. A restart button shows up on the screen for the user to start a new round.

d. **Scoring System**: The player's capacity to strike foes determined the scoring system. As foes are destroyed, the player's score changes in real time. The UI shows this score; it resets when the game starts again.

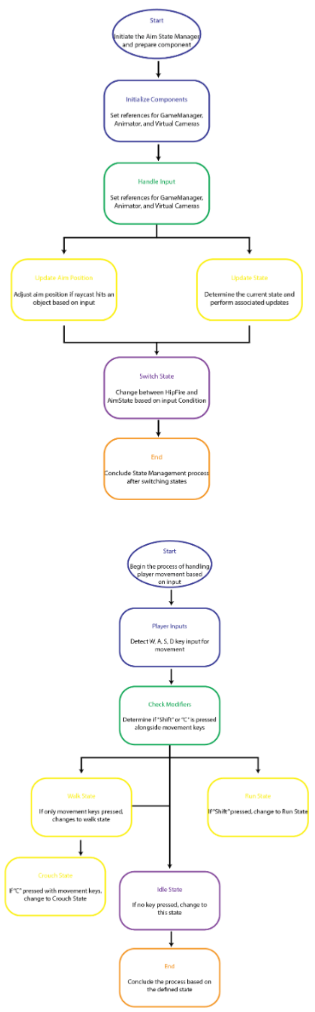

Start
Initiate the Aim State Manager and prepare component
Initialize Components
Set references for GameManager, Animator, and Virtual Cameras
Handle Input
Set references for GameManager, Animator, and Virtual Cameras
Update Aim Position
Adjust aim position if raycast hits an object based on input
Update State
Determine the current state and perform associated updates
Switch State
Change between HipFire and AimState based on input Condition
End
Conclude State Management process after switching states
Start
Begin the process of handling player movement based on input
Player Inputs
Detect W, A, S, D key input for movement
Check Modifiers
Determine if "Shift" or "C" is pressed alongside movement keys
Walk State
If only movement keys pressed, changes to walk state
Run State
If "Shift" pressed, change to Run State
Crouch State
If "C" pressed with movement keys, change to Crouch State
Idle State
If no key pressed, change to this state
End
Conclude the process based on the defined state

Fig 4: Flowchart for Aiming and Movement

D. *UI Development and Integration*

Providing player feedback depends much on the User Interface (UI). Key components like the score display, timer, and restart buttons were created using Unity's built-in UI tools.

a. **Score and Timer**: The UI shows the time remaining on the timer and the player's present score. Updated in real time, these features show the player's score rising with each enemy hit and the timer counting down as the game advances.

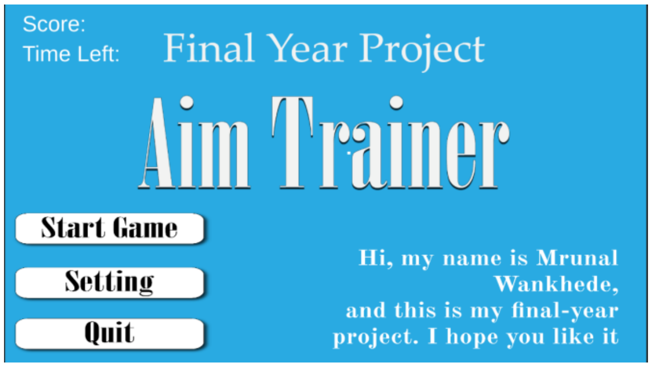

Score:
Time Left:
Final Year Project
Aim Trainer
Start Game
Setting
Quit
Hi, my name is Mrunal Wankhede, and this is my final-year project. I hope you like it

Fig 5: UI

b. **Restart Panel**: When the game ends, a restart panel with the player's last score and a choice to start the game again shows up. This enables a fast restart without closing the game or starting the whole procedure again.

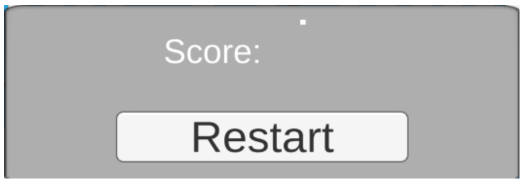


Fig 6: UI: Panel to Restart the game

Through Canvas, TextMeshPro, and Button components, Unity's UI engine makes sure the interface works well and looks good.

Final Testing And Optimisation

The game was run for bugs and performance problems after all the assets, animations, and gameplay mechanics had been combined. Testing was done in several phases to guarantee every component functioned as planned:

- **Playtesting**: Making sure the UI components, animations, and game mechanics were operating as planned
- **Performance Optimization**: Verifying the frame rates and memory consumption to guarantee the game's smooth operation on several hardware setups. This meant using Unity's built-in Occlusion Culling and Level of Detail (LOD) capabilities, lowering texture sizes, and optimizing models.

Once the game mechanics were stable, the final version of the game was ready for release.

V. RESULT

Developing the 3D game in Unity produced a completely interactive and functional experience that combined real-time gameplay mechanics, enemy interactions, and a responsive scoring system. This section outlines the project's main outcomes, emphasizing how the different stages—from asset creation to final implementation—came together to create a cohesive and well-executed game.

1. Game Efficiency and Stability

One of our key objectives was to make sure the game ran consistently and smoothly; we are happy to report that this was achieved. Even under concurrent operation of many objects and animations, the game ran steadily with consistent rates across several platforms.

Maintaining performance without sacrificing visual quality was greatly aided by optimization methods such polygon count reduction, texture simplification, and occlusion culling implementation. It ran dependably all around on devices with many different hardware setups. This guaranteed a smooth gaming experience for players without a need for good system

2. Gameplay Mechanics and Interaction

The basic mechanics—shooting, movement, and enemy interactions—functioned exactly as planned. Based on mouse input, bullets fired toward foes, making shooting both responsive and accurate. Working in real time, the collision detection system consistently triggered events such enemy destruction.
Enemy AI also worked properly, responding to player actions and showing suitable animations upon hit. These animations increased realism and made the game more engaging. Awarding points for every enemy killed, the scoring system monitored player progress well. Located in the top-right corner of the display, a countdown clock let players know how much time they had left, so heightening excitement and urgency.

3. UI Functionality and Usability

The whole experience was made efficient by the user interface. While still clean and simple to use, it obviously showed vital information including the score and time remaining. The UI components were simple and quick, allowing users to monitor progress unobtrusively.
From a functional standpoint, everything ran well. All of the buttons for starting, quitting, and restarting were completely interactive; the restart panel showed exactly when players could fast return to the game following a session end. Player input underlined how the interface supported a smooth and pleasant experience.

4. Animation and Asset Integration

Bringing Blender assets into Unity went fairly well; characters, objects, and environments imported properly. Every model was correctly animated, textured, and rigged. Player movements—including walking, running, shooting, and idle stances—were directly linked to inputs, so generating natural and fluid animations.

The game had seamless transitions between different states because to the effective blending of Blender's animation tools and Unity's Animator Controller. Everything seemed and felt consistent whether the character was moving, attacking, or standing still. This greatly improved the polish and immersion of the last game.

5. Challenges Faced and Solutions

Although the overall development process went well, we did face a few issues:

- ***Performance Issues:***In the beginning, high-poly models created lags. Reducing the engine load by simplifying the models and baking textures solved this problem without compromising too much visual quality.
- ***Animation Transitions***: Initially, blending between activities like running and shooting was uneven. Using Unity's Animator Controller, we improved this to produce more realistic, seamless transitions.
- ***Collision Detection***: Accurate hit detection between projectiles and foes required some trial and error. Eventually, changes to Unity's built-in collision system let us get consistent outcomes.

Despite these challenges, all major features were successfully implemented, and each problem encountered helped improve the final product.

6. Player Feedback and Testing

Polishing the game depended on playtesting. Testers provided insightful comments on the UI, controls, and difficulty. Small adjustments depending on their feedback resulted in a far more pleasant and smoother experience.

Players especially liked the clear user interface, responsive shooting, and fluid animations. All things considered, the comments were good, verifying that the game was interesting and simple to interact with.

VI. CONCLUSIONS

In this project I successfully utilized Unity, Blender, C-Sharp and Illustrator to create a working real time 3d game which is fun to play. The code is created in a way to reduce extra computational cost. The controls are very simple to understand and get use to them. All component from animation to user interface, they work together to give us a complete and optimised experience. My game met all of its primary requirements and stand as an example for game development using modern tools

REFERENCES

[1] United We Stand: Platforms, Tools and Innovation With the Unity Game Engine[1]
[2] Advanced AI Mechanics in Unity 3D for Immersive Gameplay: A Study on Finite State Machines & Artificial Intelligence [2]
[3] The Path of UNITY or the Path of UNREAL? [3]
[4] Rules of Play: Game Design Fundamentals[4]
[5] Mr. Maxwell Foxman, School of Journalism and Communication, University of Oregon, Eugene, OR 97403, USA
[6] Mr. Arslan Akram**,** Department of Computer Science, University of Central Punjab, Lahore, Pakistan,Dec. 2024
[7] Al Lawati, H. A. J. . (2020). The Path of UNITY or the Path of UNREAL? A Comparative Study on Suitability for Game Development. *Journal of Student Research*. https://doi.org/10.47611/jsr.vi.976
[8] Katie Salen Tekinbas, Eric ZimmermanMIT Press, 25 Sept 2003