**Table of Contents**

Contents

[1 Introduction 5](#_Toc177126194)

[1.1 Vision Statement 5](#_Toc177126195)

[1.2 Project Deliverables 5](#_Toc177126196)

[1.3 System Limitations/Constraints 6](#_Toc177126197)

[1.4 Tools and Technologies 6](#_Toc177126198)

[1.5 Relevance to Course Modules 6](#_Toc177126199)

[2 Problem Definition 7](#_Toc177126200)

[2.1 Problem Statement 7](#_Toc177126201)

[2.2 Problem Solution 7](#_Toc177126202)

[2.3 Objectives of the Proposed System 7](#_Toc177126203)

[2.4 Scope 7](#_Toc177126204)

[2.5 Modules 8](#_Toc177126205)

[2.5.1 Module1: User Interface Module 8](#_Toc177126206)

[2.5.2 Module 2: Game Logic Module 8](#_Toc177126207)

[2.5.3 Module 3: Multiplayer Module 8](#_Toc177126208)

[2.5.4 Module 4: BOT Module 8](#_Toc177126209)

[3 Requirement Analysis 9](#_Toc177126210)

[3.1 User classes and characteristics 9](#_Toc177126211)

[3.2 Requirement Identifying Technique 10](#_Toc177126212)

[3.3 Functional Requirements 13](#_Toc177126213)

[3.3.1 Functional Requirement 1: Game Play 13](#_Toc177126214)

[3.3.2 Functional Requirement 2: Player Movement on Terrain 14](#_Toc177126215)

[3.3.3 Functional Requirement 3: Combat Mechanics 14](#_Toc177126216)

[3.3.4 Functional Requirement 4: Player’s Health and Interaction with Enemy 15](#_Toc177126217)

[3.3.5 Functional Requirement 5: Game over Screen Display 15](#_Toc177126218)

[3.3.6 Functional Requirement 6: Multiplayer 16](#_Toc177126219)

[3.4 Non-Functional Requirements 16](#_Toc177126220)

[3.4.1 Reliability 16](#_Toc177126221)

[3.4.2 Usability 16](#_Toc177126222)

[3.4.3 Performance 17](#_Toc177126223)

[3.4.4 Security 17](#_Toc177126224)

[3.5 External Interface Requirements 17](#_Toc177126225)

[3.5.1 User Interfaces Requirements 17](#_Toc177126226)

[3.5.2 Software interfaces 18](#_Toc177126227)

[3.5.3 Hardware interfaces 18](#_Toc177126228)

[3.5.4 Communication Interfaces 19](#_Toc177126229)

[4 Design and Architecture 20](#_Toc177126230)

[4.1 Architectural Design 20](#_Toc177126231)

[4.2 Design Models 20](#_Toc177126232)

[4.2.1 Activity Diagram 20](#_Toc177126233)

[4.2.2 Sequence Diagram: 21](#_Toc177126234)

[4.2.3 State Transition Diagram: 21](#_Toc177126235)

[4.2.4 Class Diagram: 22](#_Toc177126236)

[4.3 Data Design 23](#_Toc177126237)

[4.3.1 Data Dictionary 23](#_Toc177126238)

[4.4 Human Interface Design 23](#_Toc177126239)

[4.4.1 Screen Images 24](#_Toc177126240)

[5 Implementation 25](#_Toc177126241)

[5.1 Algorithm 25](#_Toc177126242)

[5.2 External APIs/SDKs 29](#_Toc177126243)

[5.3 User Interface 30](#_Toc177126244)

[5.4 Deployment 31](#_Toc177126245)

[5.4.1 Unity Engine 31](#_Toc177126246)

[5.4.2 Itch.io 31](#_Toc177126247)

[6 Testing and Evaluation 32](#_Toc177126248)

[6.1 Unit Testing 32](#_Toc177126249)

[6.1.1 Unit Testing 1: User Interface 32](#_Toc177126250)

[6.1.2 Unit Testing 2: Game Logic 32](#_Toc177126251)

[6.1.3 Unit Testing 5: Multiplayer 33](#_Toc177126252)

[6.1.4 Unit Testing 6: Enemy AI 33](#_Toc177126253)

[6.2 Functional Testing 33](#_Toc177126254)

[6.2.1 Functional Testing 1: Game Play 33](#_Toc177126255)

[6.2.2 Functional Testing 2: Game Logic 34](#_Toc177126256)

[6.2.3 Functional Testing 3: Combat Mechanics 34](#_Toc177126257)

[6.2.4 Functional Testing 4: Player's Health and Interaction with Enemy 34](#_Toc177126258)

[6.2.5 Functional Testing 5: Game Over Screen Display 35](#_Toc177126259)

[7 Conclusion 35](#_Toc177126260)

[7.1 Conclusion 35](#_Toc177126261)

[8 References 36](#_Toc177126262)

**List of Figures**

[Figure 1 Main Menu 8](#_bookmark24)

[Figure 2 Story Menu 8](#_bookmark25)

[Figure 3 Multiplayer Scene 8](#_bookmark26)

[Figure 4 Tribe Area 9](#_bookmark27)

[Figure 5: Architectural Design 16](#_bookmark53)

[Figure 6: Activity Diagram 16](#_bookmark56)

[Figure 7: Sequence Diagram 17](#_bookmark58)

[Figure 8: State Transition Diagram 17](#_bookmark60)

[Figure 9: Class Diagram 18](#_bookmark62)

[Figure 10: Player Image 20](#_bookmark67)

[Figure 11: Enemy Image 20](#_bookmark68)

**List of Tables**

[Table 1 Related System Analysis with proposed project solution 1](#_bookmark3)

[Table 2 Tools and Technologies for Proposed Project 3](#_bookmark7)

[Table 3: User classes and Characteristics 5](#_bookmark21)

[Table 4: Event Response Table 7](#_bookmark23)

[Table 5: FR-1 Game Plays 9](#_bookmark30)

[Table 6: FR-2 Player Movement on Terrain 10](#_bookmark32)

[Table 7: FR-3 Combat Mechanics 10](#_bookmark34)

[Table 8: FR-4 Player’s Health and Interaction with Enemy 11](#_bookmark36)

[Table 9: FR: 5 Game Over Screen Display 11](#_bookmark38)

[Table 10: FR-6 Multiplayer 12](#_bookmark40)

[Table 11: Example of Algorithm 21](#_bookmark71)

[Table 12: Details of APIs used in the project 26](#_bookmark73)

# Introduction

**"The Conquest of Mongolia: The Game,"** which provides a distinct take on the realm of Mongolian conquest. You will have the opportunity to guide your group through the difficulties of forging alliances, bringing tribes together, and touring Mongolia's enormous plains in this game. The goal of "The Conquest of Mongolia: The Game" is to provide a simple yet historically engaging gaming experience, even though it may not have elaborate gameplay or gorgeous graphics.

As a player, you take on the role of a champion in Mongolia's cloud fields, where the choices you make will determine how your team performs. As you work to establish your empire, you must make strategic decisions, manage your resources, and fight conflicts.

## Vision Statement

Our goal is to provide "The Conquest of Mongolia: The Game," a distinctive and approachable introduction to the grand strategy genre for gamers and history buffs. Through strategically designed gameplay that emphasizes alliance formation and resource management, the game seeks to immerse players in the historical adventure of uniting Mongolian tribes and extending their lands. We offer a more straightforward yet historically rich option to the graphically heavy strategy games that are now taking over the market by encapsulating the spirit of Mongolian history without overwhelming players with complicated mechanics. Our goal is to provide a fun and instructive experience that makes history of Mongolia interesting and approachable, particularly for individuals who have a strong desire to construct strategically sound empires with a historical flare.

## Project Deliverables

The listing of undertaking deliverables is as follows.

* + - **Source Code:** Using the language to recreate the world, and the code written to put into effect the sport.
    - **User Interface Design:** Articles and illustrations, menu schematics, HUD (Head-Up Display) and different elements associated with gameplay user interface.
    - **Game Executable**: The compiled model of the sport that may be run on Windows.
    - **Game Design Document:** A comprehensive file outlining the game's concept, mechanics, functions, together with fight mechanics, PvP multiplayer capabilities, and the overarching storyline set in the Mongolian landscape.
    - **Prototype:** An preliminary playable prototype showcasing the middle gameplay mechanics, inclusive of combat machine mechanics and the capacity to interact in PvP multiplayer suits on the equal device, with a glimpse of the storyline factors.
    - **Art Assets:** Visual property depicting characters, environments, and consumer interface elements related to fight situations, multiplayer interactions, and key story elements within the game.
    - **Game Development:** The initial new release of the game executable, demonstrating the combat gadget, multiplayer abilities, and narrative factors, offering gamers with an immersive gaming revel in.

## System Limitations/Constraints

* + - Budget, time, and talent constraints brought forth by scarce resources may affect the breadth of development.
    - Performance, multiplayer capabilities, and graphics may all be impacted by technical constraints.
    - Risks of scope creep are associated with ambitious features like plot integration, PvP, and combat.

Why Creating compelling mechanics and tales for content creation takes a significant amount of time and work.

* + - Issues with platform compatibility could occur and impact device performance.
    - It takes significant consideration and resources to localize content for a variety of languages and geographical areas.
    - It can be difficult to balance playability with historical accuracy.

## Tools and Technologies

**Table 2 Tools and Technologies for Proposed Project**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tools And**  **Technologies** | **Tools** | **Version** | **Rationale** |
| Unity | 2022.3.4f1 | Game Development  Platform |
| Visual Studio 2022 | 2022 | IDE |
| MS Word | 2013 | Documentation |
| MS PowerPoint | 2013 | Presentation |
| **Technology** | **Version** | **Rationale** |
| Unity | 2022.3.4f1 | Front-end Development |
| C# | Latest | Back-end Development |
| PlantUML |  | Diagrams |

## Relevance to Course Modules

Although our BCS program does not provide any classes specifically on game development, we do understand programming and the fundamentals of software engineering, which are relevant to game development. Our Visual Programming (CSC412) foundation gave us the necessary C# language abilities, which inspired us to use Unity, a C#-based game engine, for game design.

# Problem Definition

## Problem Statement

There aren't many easily accessible, interesting historical strategy games out on the market right now that provide a simple but deep experience. Current games frequently place a premium on large budgets and many features, underserving a certain niche market. Furthermore, the creation of personalized and engaging gaming experiences is hampered by resource limitations and technical limitations. "The Conquest of Mongolia: The Game" aims to bridge this gap by offering players who want a more approachable and immersive strategy game a distinctive and entertaining gameplay experience set against the historical backdrop of Mongolia.

## Problem Solution

The lack of easily accessible and captivating historical strategy games in the gaming industry is filled by "The Conquest of Mongolia: The Game." Without overwhelming players with intricate mechanics, the game maintains the spirit of Mongolian history by concentrating on strategic gameplay features like resource management and alliance formation. This strategy makes the game playable by a larger audience by guaranteeing a simple but engaging experience. We overcome restricted financial and technical limitations by utilizing efficient development approaches and concentrating on essential gameplay components instead of comprehensive features. Our game offers a distinct, instructive, and entertaining experience, targeting players who are looking for a simpler, historically accurate strategy game.

## Objectives of the Proposed System

* + - **Diverse Gameplay Modes:** Provide a feature-rich online gaming platform with a range of gameplay options for both single- and multiplayer players to suit their varying tastes.
    - **User-Friendly Interface:** Create an interface that is simple to use and intuitive so that players can interact and traverse the game with ease. This will make sure that every player has a good and enjoyable experience.
    - **Complex Matchmaking Algorithms:** Use sophisticated matchmaking algorithms to pair players according to skill levels, guaranteeing impartial and equitable gameplay in multiplayer mode.
    - **Skill Development and Progression:** AI-controlled opponents, or BOTs, are introduced to provide players the chance to advance through the game at their own speed and develop their skills.

## Scope

A straightforward yet engrossing role-playing game (RPG) set against the historical backdrop of the Mongolian peninsula is provided by "The Conquest of Mongolia: The Game." In addition to creating and customizing their avatar, players will explore the open world and take part in real-time combat and tactical negotiations. In this stressful historical backdrop, players can forge alliances, command warriors in combat, and eventually strive to establish a strong empire as their character develops. The game's controls are kept straightforward and user-friendly, making it ideal for anyone seeking a straightforward role-playing adventure that uncovers the historical reality of the Mongolian plateau.

## Modules

### Module1: User Interface Module

The visual and interactive part of the game is the User Interface Module, which uses graphic displays, complex images, and intuitive controls to provide in-game statistics, high quality graphics, and interface control. The following elements are included in this module as the *Main Menu Interface* and *the In-Game HUD (Heads-Up Display).*

### Module 2: Game Logic Module

The Game Logic Module, which connects the core gameplay components and systems and generates the primary game engine, is another focal point of "The Conquest of Mongolia: The Game." Including the general activities of our Player and interacting with the environment and terrain. *Combat System and Player Controls* are included in this.

### Module 3: Multiplayer Module

This module is intended to provide multiplayer functionality to "The Conquest of Mongolia: The Game," allowing players to engage in head-to-head combat.

### Module 4: BOT Module

This module adds AI opponents and allies to our game, enabling dynamic gaming experiences that change based on the decisions and actions of the player. This was made possible by the use of Nav Mesh Agent technology, which gives our AI characters the ability to intelligently navigate the game environment and react in real time to player actions and decisions.

# Requirement Analysis

This document details our goals for the "The Conquest Of Mongolia: The Game" project and offers a thorough analysis to make sure our target audience will find the game engaging. To determine the essential components and features that our game should have, we have carried out market research. Our goal is to create a multiplayer experience that is both captivating and immersive. This study addresses a number of topics, such as the essential components of the game, its features, and the requirements that must be met, both functional and non-functional. In addition to enabling interactive gaming, the interface will highlight the development features. This requirement analysis's objective is to make sure that "The Conquest of Mongolia: The Game," the finished product, lives up to user expectations and provides an enjoyable gaming experience.

## User classes and characteristics

**User classes and characteristics:**

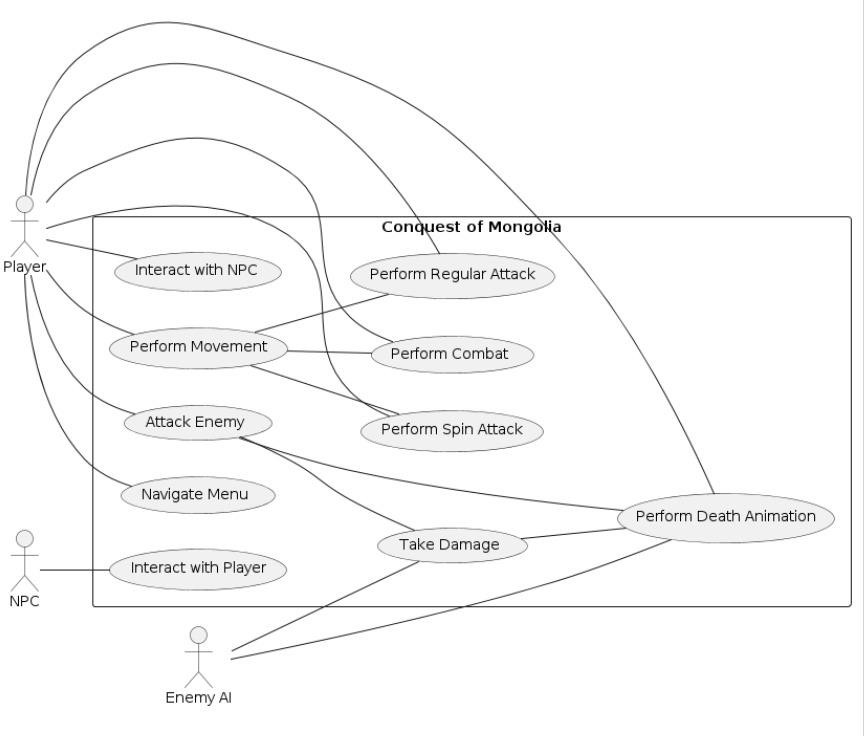
**Table 3: User classes and Characteristics**

|  |  |
| --- | --- |
| **User**  **class** | **Description** |
| **Casual Players** | These viewers merely play the game for enjoyment and to decompress. These players can be new to the genre in this instance and are searching for an engaging game without a convoluted plot. Such games could be endorsed by this audience as laid-back options with an obvious plot. This may be the case for time-pressed players who would rather play brief games with a ton of experience and involvement than passive games that only display silent  movements on the screen. |
| **Younger Players** | These gamers are too young to handle complex plots and user interfaces. These are the kinds of games that new players could be accustomed to playing, and  they would be more likely to play games that they find entertaining and |

|  |  |
| --- | --- |
|  | engaging than ones that are more precise and survival-focused. These players might like to play games with vibrant graphics and straightforward gameplay mechanics. Since they might not have a lot of gaming expertise, they would  prefer titles that are easy to pick up and don't require a lot of understanding. |

## Requirement Identifying Technique

**Use Case Diagram:**



**Figure 1: Use Case Diagram**

**Event Response Table:**

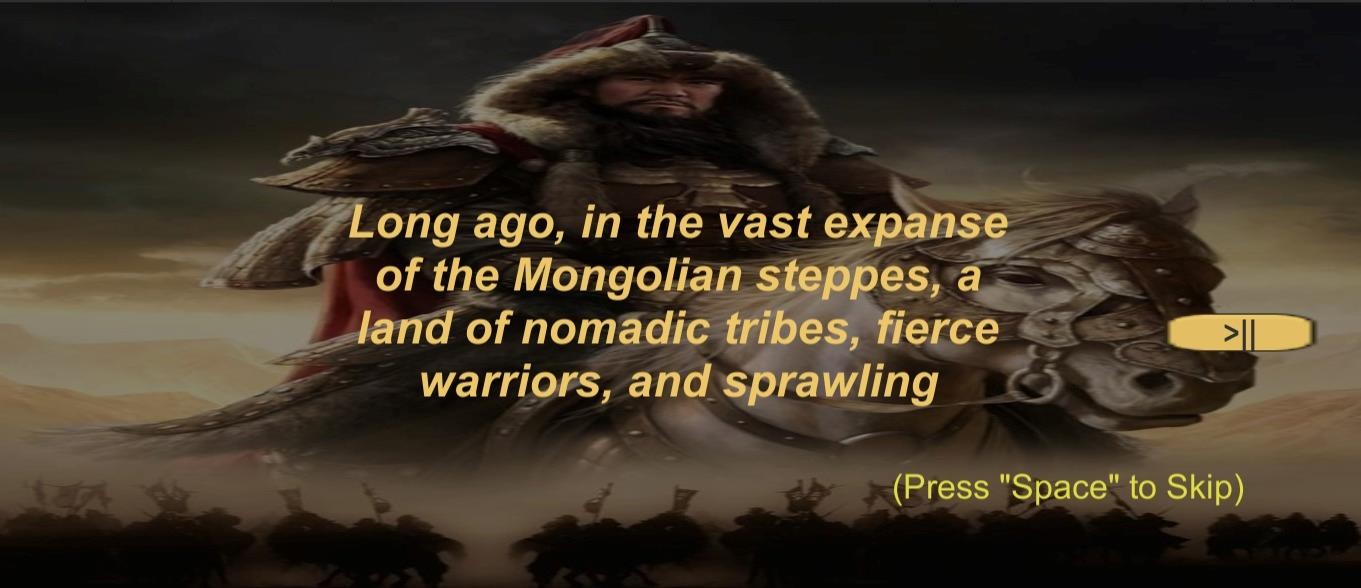
**Table 4: Event Response Table**

|  |  |  |
| --- | --- | --- |
| **Event** | **System State** | **Response** |
| Player encounters enemy tribe | Player navigating through Mongolian terrain | Initiate combat sequence with enemy AI |
| Player discovers hidden treasure | Player exploring uncharted territory | Gain valuable item or currency |
| Opponent tribe launches an attack | Player's journey through Mongolian plains | Engage in battle, adapt combat strategy |
| Player's scout reports enemy movements | Player preparing for potential conflict | Adjust tactics, prepare for upcoming battle |
| Player finds ancient artifact | Player exploring historical sites | Unlock knowledge or historical insight |
| Player attacks enemy tribe | Player confronting enemy forces | Initiate combat sequence, damage enemy |
| Enemy tribe retaliates | Player under attack by enemy forces | Engage in battle, adapt defensive strategy |
| Player suffers defeat | Player's health reaches zero | Display Game Over screen, reset to previous state |
| Enemy tribe defeated | Enemy forces defeated by player | Gain resources or territory, continue exploration |
| Player explores uncharted terrain | Player traversing unknown areas | Trigger random event or encounter |

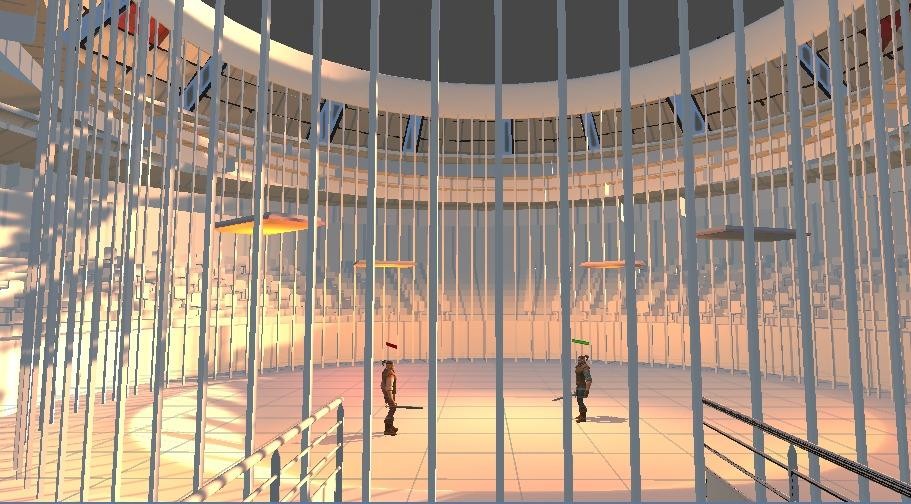
**Storyboarding:**



**Figure 1 Main Menu**



**Figure 2 Story Menu**



**Figure 3 Multiplayer Scene**



**Figure 4 Tribe Area**

## Functional Requirements

The functional requirements for "The Conquest of Mongolia: The Game" outline the specific traits and talents required for the system to function properly. This section outlines the capabilities that provide a strong foundation for the development and operation of the system. These functional specifications are necessary to provide gamers with an excellent and captivating multiplayer experience.

### Functional Requirement 1: Game Play

**Table 5: FR-1 Game Play**

|  |  |
| --- | --- |
| **Identifier** | FR-1 |
| **Title** | Game Play |
| **Requirement** | **User Perspective:** The user will run game. Play the game by interacting with the controls and features of the game.  **System Perspective:** The system allows players to start a new game. The system allows players to control the character movement. |
| **Source** | User |

|  |  |
| --- | --- |
| **Rationale** | The core gameplay mechanics of our game require accurate player control, NPC controls and score feedback. |
| **Dependencies** | System |
| **Priority** | High |

### Functional Requirement 2: Player Movement on Terrain

**Table 6: FR-2 Player Movement on Terrain**

|  |  |
| --- | --- |
| **Identifier** | FR-2 |
| **Title** | Player movement on Terrain. |
| **Requirement** | **User Perspective:** The Player must perform movements on the terrain it is supposed to walk on.  **System Perspective:** The system should follow what the player controls inputs by moving the character on the center of steppe of Mongolia, using present position as a base. |
| **Source** | Game Design Team |
| **Rationale** | To give an amazing and engaging games for the players who get full control of movement of characters on the steppe. |
| **Dependencies** | Keyboard |
| **Priority** | High |

### Functional Requirement 3: Combat Mechanics

**Table 7: FR-3 Combat Mechanics**

|  |  |
| --- | --- |
| **Identifier** | FR-3 |
| **Title** | Combat Mechanics |
| **Requirement** | **User Perspective:** The Player should perform combat against enemies NPC. And provides damage to them  **System Perspective:** The figure will interpret user input so the enemy will respond properly. |
| **Source** | Game Design Team |
| **Rationale** | For a more active gameplay, we should design our game in such a way that it would give a player the opportunity to closely control the game outcome with a variety of options. |
| **Dependencies** | Keyboard, NPC AI System, Collision Detection, Damage Calculation System |

|  |  |
| --- | --- |
| **Priority** | High |

### Functional Requirement 4: Player’s Health and Interaction with Enemy

**Table 8: FR-4 Player’s Health and Interaction with Enemy**

|  |  |
| --- | --- |
| **Identifier** | FR-5 |
| **Title** | Player’s Health and obstacle Interaction |
| **Requirement** | **User Perspective:** The player will respond to the enemy attack as player’s health would drop.  **System Perspective:** The system will disallow player to move anymore after player dies. |
| **Source** | Game Design Team |
| **Rationale** | Linking player health to enemy interactions adds realism and intensity to gameplay, enhancing immersion. |
| **Dependencies** | Health Management System, Enemy Attack System, Collision Detection,  Movement Control System |
| **Priority** | High |

### Functional Requirement 5: Game over Screen Display

**Table 9 FR: 5 Game Over Screen Display**

|  |  |
| --- | --- |
| **Identifier** | FR-5 |
| **Title** | Game Over Screen Display |
| **Requirement** | **User Perspective**: The software will trigger the Game Over screen when the player's health reaches zero, indicating the end of the game.  **System Perspective:** Upon the player's health depleting entirely, the game system will transition to the Game Over screen, reflecting the player's defeat. |
| **Source** | Game Design Team |
| **Rationale** | To inform players of the game's conclusion and outcome, enhancing the overall gaming experience |
| **Dependencies** | Health Management System, Screen Transition System, User Interface  System |
| **Priority** | Medium |

### Functional Requirement 6: Multiplayer

**Table 10: FR-6 Multiplayer**

|  |  |
| --- | --- |
| **Identifier** | FR-6 |
| **Title** | Multiplayer |
| **Requirement** | **User Perspective:** The software will trigger the Game Over screen when any player's health reaches zero, indicating the end of the game for all participants.  S**ystem Perspective:** Upon any player's health depleting entirely, the game system will transition to the Game Over screen for all players, reflecting the collective defeat in the multiplayer session. |
| **Source** | Game Design Team |
| **Rationale** | Ensuring a synchronized and coherent Game Over screen experience across all players in local multiplayer |
| **Dependencies** | Multiplayer Synchronization System |
| **Priority** | High |

## Non-Functional Requirements

The qualities of the game that are crucial to its success but unrelated to its functioning are referred to as non-functional requirements. The following are some of the game's non-functional requirements:

### Reliability

* When the user loses progress or data due to a sudden crash or termination caused by a severely broken game, it is referred to as an error.
* Accepting a software failure implies that the player will most likely lose a lot of progress, that the user may become unhappy or frustrated, and that the reputation of the game may be in jeopardy.
* We will be able to guarantee error detection by completing comprehensive testing using a variety of testing methodologies, including functional testing, unit testing, and integration testing.
* New software upgrades to address issues found in log files, automatic recovery mechanisms, and the installation of corrective measures that must be made right away and then reported.

### Usability

* The gameplay and instructions for the game should be straightforward and simple enough for new players to follow, making it easier for them to comprehend how the game works.
* This will be made possible by the users' seamless navigation of video game menus and interfaces, together with unambiguous visual cues and recurrent design patterns serving the same function.
* Error avoidance and recovery procedures must be implemented in order to assist users with common issues such as erroneous inputs and connection failures.
* The goal of optimizing interaction efficiency is to minimize the amount of time the player must spend on routine operations, such as creating an avatar or joining an online session.

### Performance

* It will be ensured that the play level loads in less than 10 seconds, allowing players to move between scenes smoothly and quickly.
* Weather multiplayer latency will be minimized, resulting in a seamless online experience. Matchmaking and interaction response times average no more than 100 milliseconds apiece.

### Security

* The game will protect user data saved locally on the device, such as player identity and game progress, by utilizing industry-standard security techniques.
* The game environment will employ user authentication and permission procedures to control access to elements that are restricted to gameplay or administrative settings.
* To guarantee the integrity and security of the game software and its data, both internally and externally, regular security audits and vulnerability assessments will be carried out.

## External Interface Requirements

Interactions between users and other systems, such as other games, are necessary for an auxiliary interface. The following are a few examples of external interface needs for the game: The following are a few examples of external interface needs for the game.

**Platform Compatibility:** This feature allows the game to be played on the majority of contemporary gaming platforms, such as PCs and mobile phones, which means a large user-base.

By satisfying these external interface specifications, we can provide players with a seamless and essential experience, enabling them to engage with the game and other players on many platforms and systems.

### User Interfaces Requirements

* **Main Menu:** Along with other features like settings and game exit, the main menu layout will prominently display options for both single-player and multiplayer modes. For players of all skill levels, there will be distinct graphic backdrops, buttons, and text labels to visually highlight each option and ensure intuitive navigation. Simple labels and icons will

make it easier for users to comprehend and utilize the game, especially for those with limited gaming expertise. In order to maintain consistency and player involvement without detracting from gameplay, the design will blend in with the general theme and style of the game.

* **Game Interface:** A dynamic setting depicting the Mongolian terrain will be included in the game interface, along with player-controlled soldiers, opponents, barriers, power-ups, and strategic features. Players will be able to control their units, move about the game area, perform maneuvers, and participate in combat through an extensive controls menu. Throughout gaming, key details like the player's health, score, available resources, and mission goals will be easily visible at the top of the screen. Important events and actions will be conveyed to the player through both visual and audio signals, guaranteeing an immersive and captivating game experience.
* **Game Over Interface**: The game over screen will provide information about the final score, earned bonuses, and reached milestones after the player's character is defeated. There will be a number of alternatives available to players, enabling them to go back to the main menu, pick up where they left off in the quest, or restart the game from the beginning. The emotions of success or defeat will be contextualized through the skillful use of music and images, improving player immersion and engagement by giving a clear feeling of development and consequence inside the game environment.

### Software interfaces

* **Unity Engine Interface:** The Unity game engine, which offers an underlying structure for the game and a set of application programming interfaces (APIs) that are interfaced with the engine and enable the game components to interact with it, such as managing game objects, processing input events, and rendering graphics, will be used in the development of the game.
* **Input Device Interface:** The gamepad, keyboard, mouse, and touch screen, to name a few, will not be necessary for the games' interface to function at all. The input controllers in the game will be used to move the player, interact with the objects in the game, and other tasks.
* **Network Interface:** There is requirement for an external network because the game implements Multiplayer through room management.

### Hardware interfaces

* **Display Interface:** To display the images and animations in the game, a display interface—such as a monitor or projector—will be absolutely necessary. It is preferable for the display's interface to have a high refresh rate and resolution.
* **Input Device Interface:** To enable players to carry out the various actions, the game will require a variety of input devices, such as a keyboard, mouse, gamepad, or touch screen.

The game's control system must be exact, and players must react quickly. If not, the gamer can experience game lag.

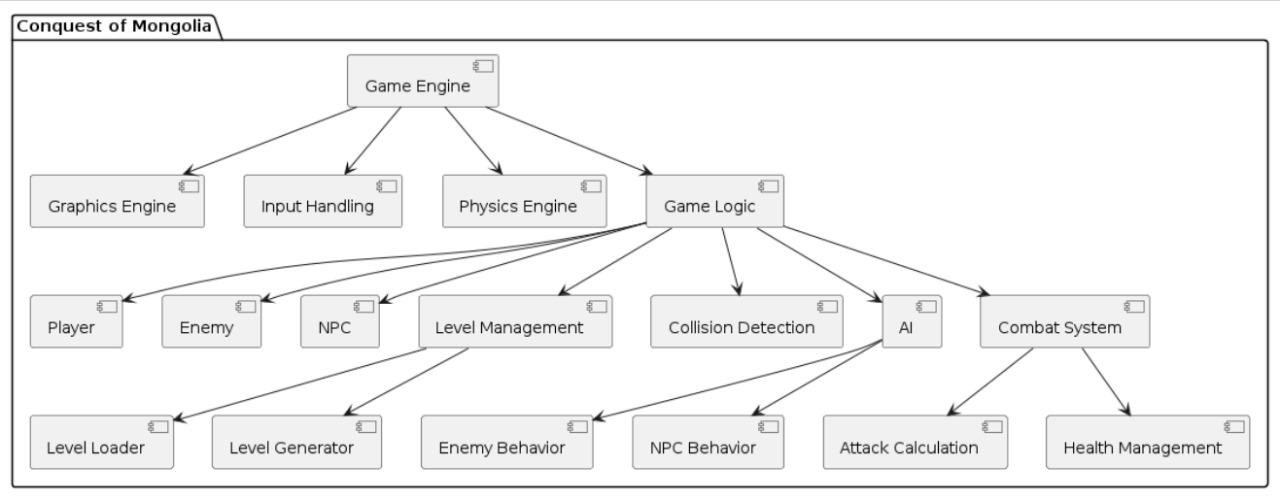
* **Storage Interface:** A system for writing and rewriting data, such as game settings, player profiles, and game progress (persistence), may be required for the game. This mechanism will also require storage components (hard disk drive or solid-state drive). The storage interface's writing and reading speeds must be fast enough to enable effective and seamless data management, and its capacity must be sufficient to accommodate the high input/output demands.

### Communication Interfaces

* **Input Device Interface:** The game required to get the information from many kinds of equipment which they could be keyboard, mouse or game pad. Input device interface has to work with multiple input methods like joysticks, directional buttons and pace while providing dedicated and immediate response upon user's command.
* **Audio API Interface:** The game would be able to invoke audio through unity built in AudioSource play sounds, and music. The audio module offer different audio formats for the interface and ensure the smoothness and speediness of the audio playback.
* **Network Interface:** A network interface is needed in order for connecting with other players in the event that the game supports online multiplayer, which links users to other players via the Internet. The interface should support various network topologies and protocols and be equipped with security and unlucky data transmission features.

# Design and Architecture

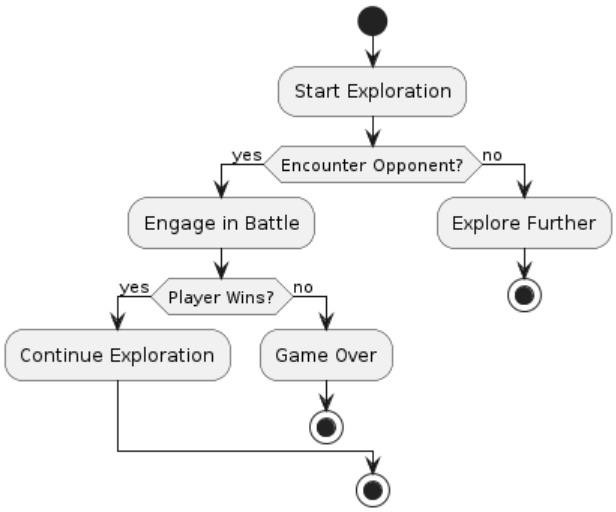
## Architectural Design



**Figure 5: Architectural Design**

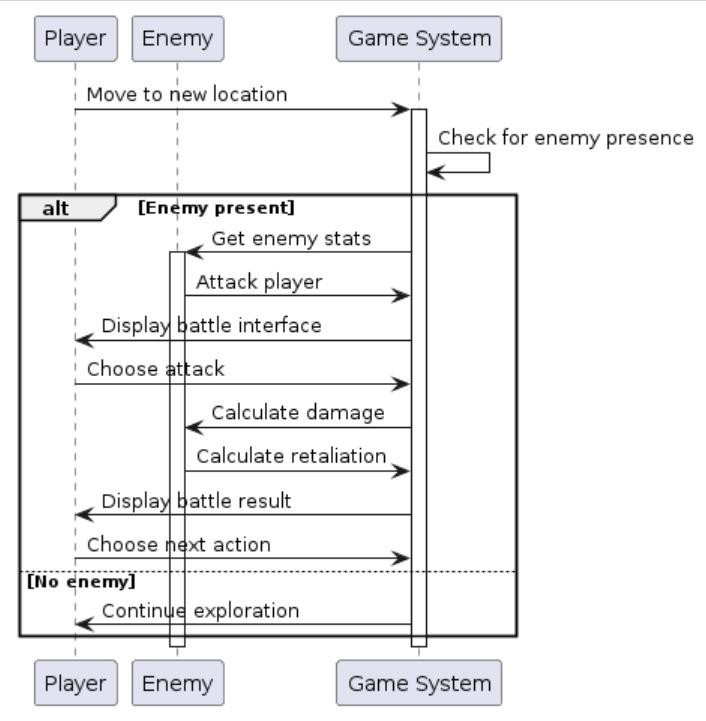
## Design Models

### Activity Diagram



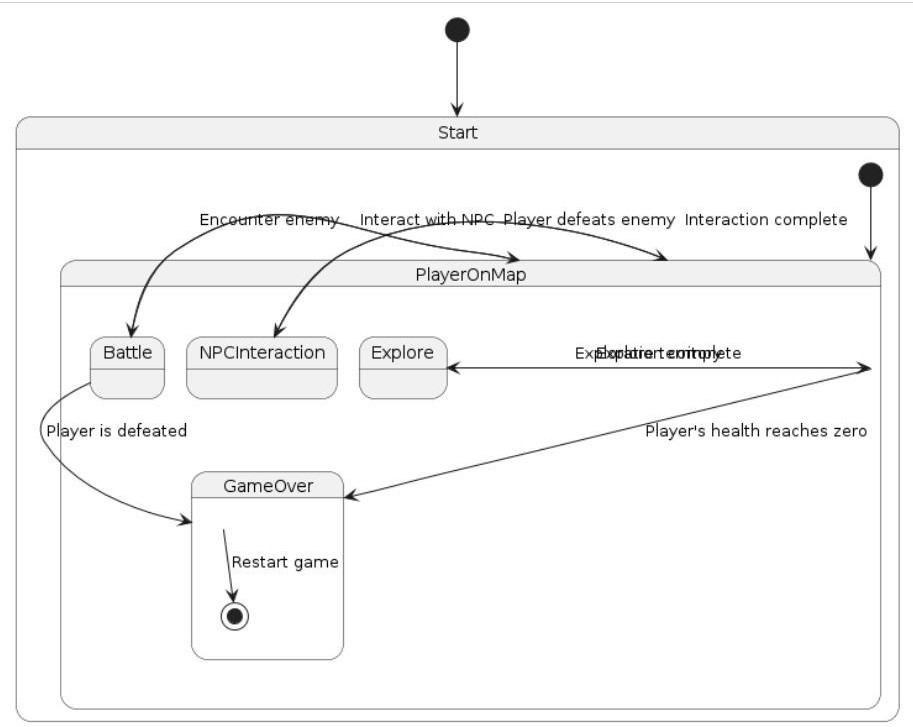
**Figure 6: Activity Diagram**

### Sequence Diagram:



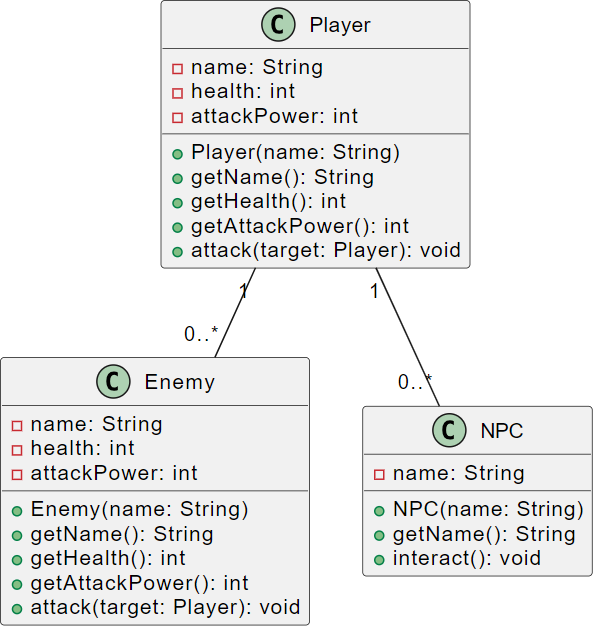
**Figure 7: Sequence Diagram**

### State Transition Diagram:



**Figure 8: State Transition Diagram**

### Class Diagram:



**Figure 9: Class Diagram**

* **Player:** Has name, health, attackPower; can attack(target); interacts with multiple Enemy and NPC.
* **Enemy:** Has name, health, attackPower; can attack(target).
* **NPC:** Has name; can interact() with players.
* **Player to Enemy**: One player can encounter multiple enemies.
* **Player to NPC:** One player can interact with multiple NPCs.

## Data Design

**Game Assets**: Multimedia components that are crucial to the game, including character models, animations, textures, sound effects, and music tracks, are referred to as game assets. The game's file system contains organized directories where these assets are kept. For instance,

* The "Animations" folder might contain character models and animations.
* The "Textures" folder might have textures for environments and objects.
* You can save sound effects and music files in the "Sounds" folder. (Details optional) Within these folders, every asset is properly arranged and labeled to facilitate easy access and administration by the game engine during runtime. Processing game assets entails putting them into memory when needed for gameplay, displaying them on screen, and controlling playback (for audio assets only).

.

### 4.3.1 Data Dictionary

* **Mongolian Steppe/Terrain:** An object having characteristics including terrain type, height, and environmental dangers that symbolizes the vastness of Mongolia's landscapes..
* **Adventure:** An item that symbolizes the actual game, complete with features like player advancement, ongoing tasks, and rules. There are two methods: exit game, and start game.
* **Player**: An item that embodies a player character, complete with name, level of expertise, and favorite fighting style. There are three methods: move, attack, and die.
* **Enemy:** An object having characteristics such as combat skill, agility, and degree of hostility that symbolizes an enemy character. Techniques include attacking, spin, and dying.
* **Battle System**: An apparatus used to control battle mechanics, such as tactics, strategies, and damage estimations.
* **Weaponry:** An object with characteristics like weight, damage potential, and range that represents the different tools and weapons that warriors can use.
* **Interface:** An item with features like menus, buttons, and HUD components that symbolizes the user interface. There are other methods, such as display, input, and visual updating.

## Human Interface Design

The system will include a number of features so that the player can have a genuine game experience, such as:

**Single-player and multiplayer modes:** Both single-player and multiplayer options are available to players; they can choose to play against another player or not.

**Game controls:** The player can use their mouse or keyboard to communicate with the enemy.

### 4.4.1 Screen Images



**Figure 10: Player Image**



**Figure 11: Enemy Image**

# Implementation

## Algorithm

**Table 11 Algorithms**

|  |
| --- |
| **Algorithm1: PlayerController** |
| // Class to control player character Class PlayerController  // Variables Declare animator  Declare animation hashes: isWalkingHash, isRunningHash, isWalkingLeftHash, isWalkingRightHash, isWalkingBackwardsHash  Declare walkSpeed as 3 Declare runSpeed as 6 Declare HealthBar as Slider  // Start method Method Start  Get animator component  Initialize animation hashes for walking, running, etc. End Method  // FixedUpdate method Method FixedUpdate  Get input for forward, backward, left, right, run, attack, and spin  // Move the character based on input  Call MoveCharacter with input parameters  // Set animation booleans based on input  Call SetAnimationBools with input parameters  // Handle combat if attack or spin input is detected If attack input or spin input  Call Combat with input parameters End If  // Check if health is zero or below  If HealthBar value is less than or equal to 0 Call Die  End If End Method  // Function to move the character  Method MoveCharacter(forwardPressed, backwardPressed, leftPressed, rightPressed, runPressed) Initialize movementDirection as zero vector  // Update movementDirection based on input If forwardPressed  Add forward direction to movementDirection End If  If backwardPressed  Subtract forward direction from movementDirection End If  If leftPressed  Subtract right direction from movementDirection |

|  |
| --- |
| End If  If rightPressed  Add right direction to movementDirection End If  // Normalize movement direction Normalize movementDirection  // Set speed based on run input If runPressed  Set speed to runSpeed Else  Set speed to walkSpeed End If  // Move character  Translate character in the direction of movementDirection with the calculated speed End Method  // Function to set animation booleans  Method SetAnimationBools(forwardPressed, backwardPressed, leftPressed, rightPressed, runPressed) Set animator boolean isWalking to true if any movement key is pressed  Set animator boolean isRunning to true if run key is pressed  Set animator boolean isWalkingLeft to true if only left key is pressed Set animator boolean isWalkingRight to true if only right key is pressed  Set animator boolean isWalkingBackwards to true if only backward key is pressed End Method  // Function to handle combat  Method Combat(attackPressed, spinPressed) If attack input is detected  Trigger attack animation  // Add attack functionality here End If  If spin input is detected Trigger spin animation  // Add spin functionality here End If  End Method  // Function to handle player's death Method Die  Trigger death animation Disable player control  // Add game over or additional functionality here End Method  End Class |
| **Algorithm 2: AI Enemy** |
| Class Enemy\_NPC  Declare playerTransform as Transform Declare followDistance as 2  Declare runDistance as 5 Declare stopDistance as 1.5 Declare runSpeed as 5 Declare attackDistance as 1 Declare counterInterval as 5  Declare agent as NavMeshAgent  Declare animator as Animator Declare isFollowing as false |

|  |
| --- |
| Declare isRun as false  Method Start  Get agent component Get animator component  If playerTransform is null  Log error: "Player Transform is not assigned" End If  End Method  Method Update  If playerTransform is null or animator is null or agent is null or agent is not active and enabled Return  End If  Calculate distanceToPlayer as distance between NPC and player If distanceToPlayer <= followDistance  If agent is on NavMesh  Set agent destination to player position End If  Set isRun based on distanceToPlayer between stopDistance and runDistance Set animator "isRun" parameter based on isRun  Set agent speed to runSpeed if isRun  If distanceToPlayer <= attackDistance Trigger "Attack" animation  Stop agent movement Else  Set isFollowing to true End If  Decrease counterInterval by deltaTime If counterInterval <= 0  Trigger "Counter" animation Reset counterInterval to 5  End If Else  If isFollowing Reset agent path  Set animator "isRun" parameter to false Set isFollowing to false  End If End If  If not isRun and distanceToPlayer > followDistance Stop agent movement  End If End Method  Method Die  Destroy NPC GameObject End Method  End Class |
| **Algorithm 3: PlayerHealth** |
| Class PlayerHealth  Declare maxHealth as 100  Declare currentHealth Declare healthSlider as Slider |

|  |
| --- |
| Method Start  Set currentHealth to maxHealth Call UpdateHealthUI  End Method  Method UpdateHealthUI  Set healthSlider value to currentHealth End Method  Method TakeDamage(damageAmount) Decrease currentHealth by damageAmount Call UpdateHealthUI  If currentHealth <= 0 Call Die  End If End Method  Method Die  Log "Player has died!"  // Implement additional death behavior End Method  End Class |
| **Algorithm 4: EnemyHealth** |
| Class EnemyHealth  Declare maxHealth as 100 Declare currentHealth  Declare damageCooldown as 1 Declare lastDamageTime Declare healthSlider as Slider  Method Start  Set currentHealth to maxHealth Call UpdateHealthUI  End Method  Method UpdateHealthUI  Set healthSlider value to currentHealth End Method  Method TakeDamage1(damageAmount)  If Time.time - lastDamageTime > damageCooldown Decrease currentHealth by damageAmount  Call UpdateHealthUI  If currentHealth <= 0 Call Die  End If  Set lastDamageTime to Time.time End If  End Method  Method Die  Log "Enemy has died!" Destroy gameObject  End Method End Class |

|  |
| --- |
| **Algorithm 5: PlayerCmbat** |
| Class PlayerCombat  Declare damageAmount as 10 Declare attackRange as 2  Declare attackKey as RightControl Declare spinKey as RightAlt  Method Update  If attackKey is pressed Call Attack  If spinKey is pressed Call Spin  End Method  Method Attack  Get colliders within attackRange For each collider in hitColliders  If collider is tagged as "Player" and not the current player's collider Get PlayerHealth component of the collider as enemyHealth  If enemyHealth is not null  Call enemyHealth.TakeDamage with damageAmount End If  End If End For  End Method  Method Spin  Get colliders within attackRange For each collider in hitColliders  If collider is tagged as "Player" and not the current player's collider Get PlayerHealth component of the collider as enemyHealth  If enemyHealth is not null  Call enemyHealth.TakeDamage with damageAmount End If  End If End For  End Method End Class |

## External APIs/SDKs

The presented Alteruna Multiplayer API is a very strong tool for creating online multiplayer games with Unity 3D. The API supports easy integration with other systems and applications, providing the following features:

* **Room Management:** Establish, assign, and delete modes, and control who is in the game.
* **Real-time Networking:** Inform players immediately, ensuring they are aware they are playing against each other during the match.
* **Synchronization:** Ensure all players are in the same world with identical object states.

A Photo Pun2 API that is well documented and easy to operate is the best choice for anyone including the creator that is a beginner in the field.

**Table 12: Details of APIs used in the project**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of API and version** | **Description of API** | **Purpose of usage** | **List down the API**  **endpoint/function/class in which it is used** |
| Alteruna Multiplayer API | A powerful tool for Unity 3D online multiplayer game creation. | To facilitate player engagement and real-time networking in a multiplayer setting. | -Room Management: Features for adding, removing, and altering game modes as well as managing player access.  -Real-time networking: Assures that participants are aware of one other's whereabouts during matches and promptly notifies them of events occurring in the game.  -Synchronization: The purpose of synchronization is to guarantee that every player is in the same game environment and that all object states are the same. |

## User Interface



**Figure 11: GamePlay Screen**



**Figure 12: Environment and Game Mode Selection**

## Deployment

We continued working on the game using Unity, using C# as the main programming language for the gameplay and game physics.

### Unity Engine

I tackle this by putting my expertise with Unity's whole toolkit to use, which includes the Unity Editor, C#, and Unity Physics system. The project pipeline of Unity facilitates the incorporation of 3D models, textures, and animations from several sources into the gaming environment, enhancing it with captivating visuals and dynamic effects.

### Itch.io

Publishing a game made with the Unity engine on itch.io might be seen as a way to take use of this virtual marketplace and social networking site, which connects independent game developers and allows them to interact, market, and make money off of it. A variety of tools are available on itch.io for creators to exhibit their game, monitor analytics, participate in game jams and other community-focused activities, and more. The layer of the platform would be accessible to independent developers who aim for originality and diversity in their works.

# Testing and Evaluation

The game play system is subjected to both function and unit testing. Unit testing is used to verify each tiny, discrete component of the project. System testing is used to assist uncover faults that might be concealed from the user. Before it is made available for usage, the testing needs to be finished.

## Unit Testing

### Unit Testing 1: User Interface

**Testing Objective:** Validating that main menu interface is displaying all the options correctly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1 | User navigates to the main menu interface. | Main menu interface displays options for starting the game, single-player, multiplayer, etc. | Player successfully interacts with all the options of user interface. | Pass |
| 2 | Player is in game. | User interface  displays health, weapons and combat. | All the objectives are displayed clearly and accurately. | Pass |

### Unit Testing 2: Game Logic

**Testing Objective:** Validating the player controls are responsive and intuitive and combat system is providing engaging encounters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1 | Player interacts with game controls. | Player movement, interaction with objects is smooth and  responsive | Player successfully interacts with all the objects. | Pass |
| 2 | Player engage in combat with enemies. | Combat Mechanics including Weapon handling and enemy AI provide dynamic and challenging  encounters. | Player successfully engage with the enemies. | Pass |

### Unit Testing 5: Multiplayer

**Testing Objective:** Validating the multiplayer interaction during multiplayer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1 | Player joins a multiplayer mode. | Player can sycn against opponent  player. | Successfully synch against opponent  player. | Pass |
| 2 | Player interaction during multiplayer session works as expected | Players does take and receive damage against opponent. | Player can  successfully send and receive damage. | Pass |

### Unit Testing 6: Enemy AI

**Testing Objective:** Validating the skill level variance of AI opponents.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1. | Player skill improvement against BOT | Player engages in combat against AI. | Player does successfully engage in combat against AI.  (Partially) | Pass |

## Functional Testing

### Functional Testing 1: Game Play

**Testing Objective:** To ensure that players can start a new game and control character movement accurately.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1. | Player selects "Start New Game" option | New game starts, player is placed in  game world | User successfully interact with the main menu page. | Pass |
| 2. | Player uses controls to  move character on the game terrain | Character moves  accurately according to inputs | User successfully adjust settings | Pass |
| 3. | Player interacts with game features and  objects | Character interacts with objects as  intended | User successfully interacts. | Pass |

### Functional Testing 2: Game Logic

**Testing Objective:** To ensure that player controls are functioning correctly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1. | Player presses arrow keys to move character | Character moves in corresponding  direction | Character is moving according to desired direction | Pass |
| 2. | Player uses gamepad to control character  movement | Character responds to gamepad inputs | Does Interacts with inputs. | Pass |
| 3. | Player navigates  character through different terrain types | Character movement  adjusts to terrain features | Does adjust to terrain movement. | Pass |

### Functional Testing 3: Combat Mechanics

**Testing Objective:** To ensure that combat mechanics function properly during player interactions with enemy NPCs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1. | Player initiates combat with enemy NPC | Combat sequence  begins, damage is dealt to enemy | True. | Pass |
| 2. | Player uses combat skills and abilities | Skills activate,  affecting enemy NPC | False | Fail |
| 3. | Enemy NPC responds  to player attacks | Enemy NPC reacts to  player actions | Partially True | Pass |

### Functional Testing 4: Player's Health and Interaction with Enemy

**Testing Objective:** To ensure that player's health responds correctly to enemy interactions and obstacles.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1. | Player sustains damage from enemy attack | Player's health  decreases accordingly | True | Pass |
| 2. | Player collides with  obstacle | Player's health  decreases or | Partially True | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | movement is  hindered |  |  |
| 3. | Player health reaches zero | Game Over screen is triggered | False | Pass |

### Functional Testing 5: Game Over Screen Display

**Testing Objective:** To ensure that the Game Over screen is displayed correctly when the player's health reaches zero.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test case/Test script** | **Expected result** | **Actual result** | **Result** |
| 1. | Player's health reaches  zero | Game Over screen is  displayed | False | Pass |
| 2. | Player's health is depleted entirely in multiplayer game | Game Over screen is displayed for all players | True | Pass |

# Conclusion

## Conclusion

The Conquest of Mongolia: The Game" represents the culmination of my final year project. This game, though , showcases the results of my development efforts and learning throughout the project.

While the game may be brief, it serves as a demonstration of the skills and concepts I applied during its creation. The focus was on crafting an engaging, though compact, experience centered around Mongolia's historical context.

Please note that this project is intended to remain as-is on GitHub, serving as a showcase of my work with no further development planned. I appreciate any feedback and encourage you to explore this snapshot of my final year project.

# References

It’s common to refer to various resources for a comprehensive Software Requirement Specification that will influence the development process. Here are some references that will be relevant.

* Unity will provide a comprehensive documentation on its game development engine, including the guidelines for development in C#. ( https://docs.unity3d.com/)
* IDE Visual Studio 2022: <https://visualstudio.microsoft.com/vs/>
* Assets:
* Unity Asset Store Link: <https://assetstore.unity.com/>
* Assets from : <https://devassets.com/>
* Animations from: <https://www.mixamo.com/>