

Software Engineering FT2 Report

Project Title: Chains of Valor

Prepared by:

- Eyad Metwally El-Nakib – 22100757
- Muhammad Sherif Muhammad – 22100638
- Mazen Ahmed Samir – 22100369
- Fares Hassan Hamid – 22101096
- Ahmed Hazem Attia – 22100194
- Abdallah Bassem Zain – 22100848

Date: September 6th, 2025 [06/09/2025]

Github Repo Link: <https://github.com/mezo04/AI-Unity-training>

Abstract

Chains of Valor is a third-person action game developed in Unity where the player assumes the role of a gladiator fighting waves of enemies in a Roman-style colosseum. The game features dynamic AI-driven combat, health management, and a final boss encounter. This report documents the system's business, functional, and technical requirements, design, implementation, testing, and potential future improvements.

Table of Contents

1. Introduction
2. System Overview
3. Requirements Analysis
 - 3.1 Business Requirements
 - 3.2 Functional Requirements
 - 3.3 Non-Functional Requirements
 - 3.4 Requirement Traceability Matrix (RTM)
4. System Design
5. Implementation
6. Testing
 - 6.1 Test Case Table
 - 6.2 Traceability of Requirements → Test Cases
7. Project Management

8. Deployment & Maintenance
9. Conclusion & Recommendations
10. References
11. Appendices

1. Introduction

Problem Statement: Develop an engaging, dynamic gladiator combat game with wave progression, AI opponents, and a boss challenge.

Scope: Single-player arena combat game with progressive enemy waves and a final boss.

Objectives:

- Provide smooth, responsive controls for player movement and attacks.
- Implement AI-driven enemy and boss combat using reinforcement learning.
- Include health kits, score tracking, and a functional game menu.
- Ensure engaging gameplay with victory conditions.

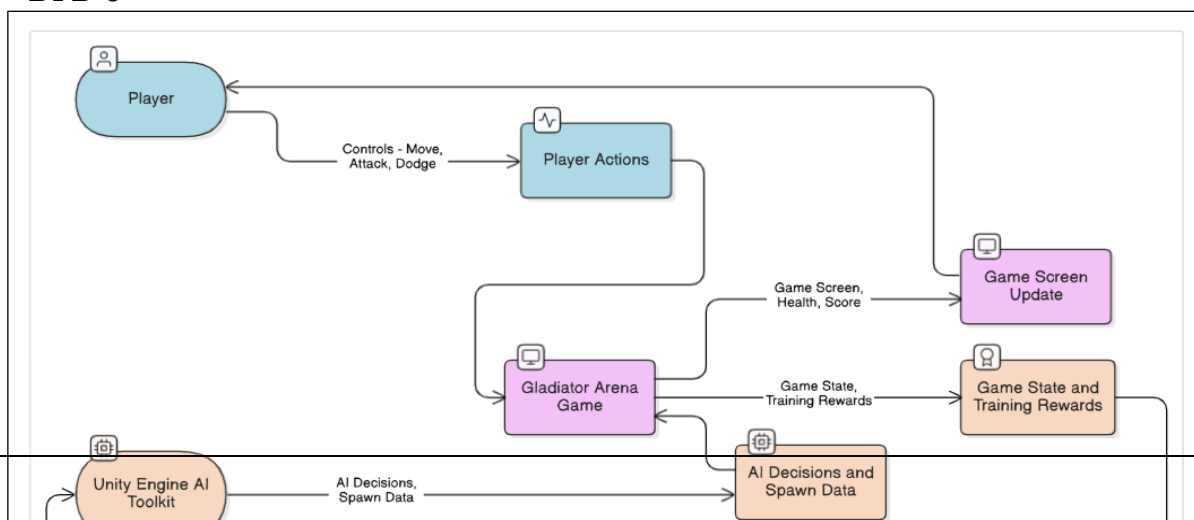
Constraints:

- Developed using Unity and C#.
 - Limited to free Unity Asset Store assets.
 - Single arena and single-player mode initially.
-

2. System Overview

Chains of Valor consists of the player character, AI enemies, boss, health kits, and UI components, all managed by the Unity engine. The system tracks inputs, spawns enemies, manages combat and health, and displays game state.

DFD 0



3. Requirements Analysis

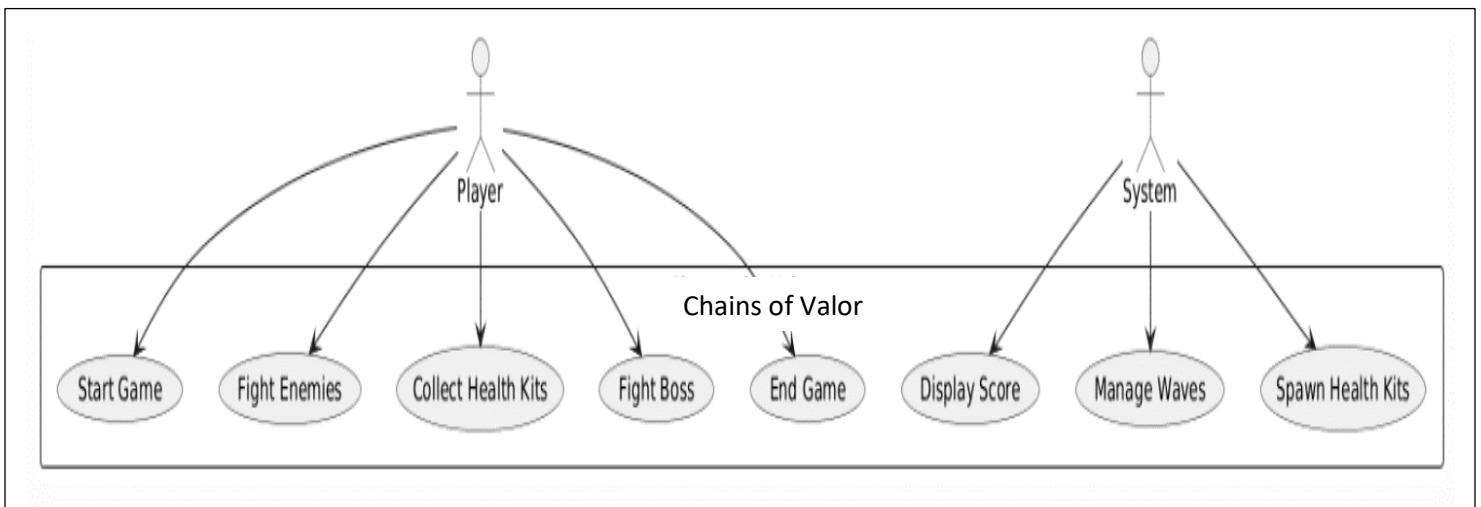
3.1 Business Requirements

- **BR1:** Create an immersive gladiator combat experience in a Roman-style colosseum.
 - **BR2:** Provide progressively challenging AI opponents to increase player engagement.
 - **BR3:** Implement a scoring and health system to track player progress and reward performance.
 - **BR4:** Include a functional main menu and user interface for smooth game navigation.
 - **BR5:** Ensure the game is stable and performs well on a standard PC configuration.
-

3.2 Functional Requirements

- Player can move, attack, and dodge.
- Enemies spawn one at a time; waves progress upon defeat.
- Boss appears after defeating a threshold number of enemies.
- Health kits spawn periodically and restore player health.
- Score updates after each enemy defeat and is displayed on the HUD.
- Game ends upon boss defeat, displaying final results.
- Main menu allows starting the game.

Use case diagram:



3.3 Non-Functional Requirements

- Game must run smoothly with low input lag (<100ms).
 - AI must behave reliably and challenge the player without crashes.
 - UI must update in real-time (health bars, score).
 - Game assets and performance should not exceed standard PC capabilities.
-

3.4 Requirement Traceability Matrix (RTM)

Purpose: Ensure all business and functional requirements are covered by design and testing.

Requirement Traceability Matrix (RTM) – Gladiator Arena					
Req. ID	Requirement Description	Type	Design Element	Implementation Module	Test Case(s)
FR1	Player can move, attack, and dodge	Functional	Use Case UC1, Class Diagram (PlayerController)	PlayerController.cs	TC 2, TC 3
FR2	Enemies spawn one at a time	Functional	Sequence Diagram (Spawn System), DFD Level 1	EnemyManager.cs	TC 5
FR3	Boss spawns after defeating threshold enemies	Functional	Activity Diagram (Game Flow), GameManager logic	BossManager.cs / GameManager.cs	TC 6
FR4	Player can collect health kits to restore health	Functional	Use Case UC3, Class Diagram (HealthKit)	HealthKit.cs, HealthSystem.cs	TC 4
FR5	Game ends upon boss defeat	Functional	Activity Diagram (Game Loop), Use Case UC4	GameManager.cs, UIManager.cs	TC 7

FR6	Score increases when enemy defeated	Functional	Use Case UC5, Sequence Diagram (Combat)	ScoreManager.cs	TC 8
FR7	Player health bar is visible and updates	Functional	UI Wireframe, Class Diagram (HealthSystem)	UIManager.cs, HealthSystem.cs	TC 4, TC 9, TC 10
FR8	Main Menu with Play option	Functional	Use Case UC1, UI Wireframe	MenuManager.cs	TC 1
NFR1	Game should respond to input with minimal delay (<100ms)	Non-Functional	System Architecture, Unity Input System	PlayerController.cs	TC 2, TC 3
NFR2	UI elements update in real-time	Non-Functional	UI Wireframe, Activity Diagram	UIManager.cs	TC 10
NFR3	Stable performance during wave progression	Non-Functional	System Architecture, Deployment Diagram	EnemyManager.cs, GameManager.cs	Stress Test
NFR4	AI must provide dynamic challenge using RL	Non-Functional	Sequence Diagram (AI decisions), Class Diagram	EnemyAI.cs, BossAI.cs	AI Test

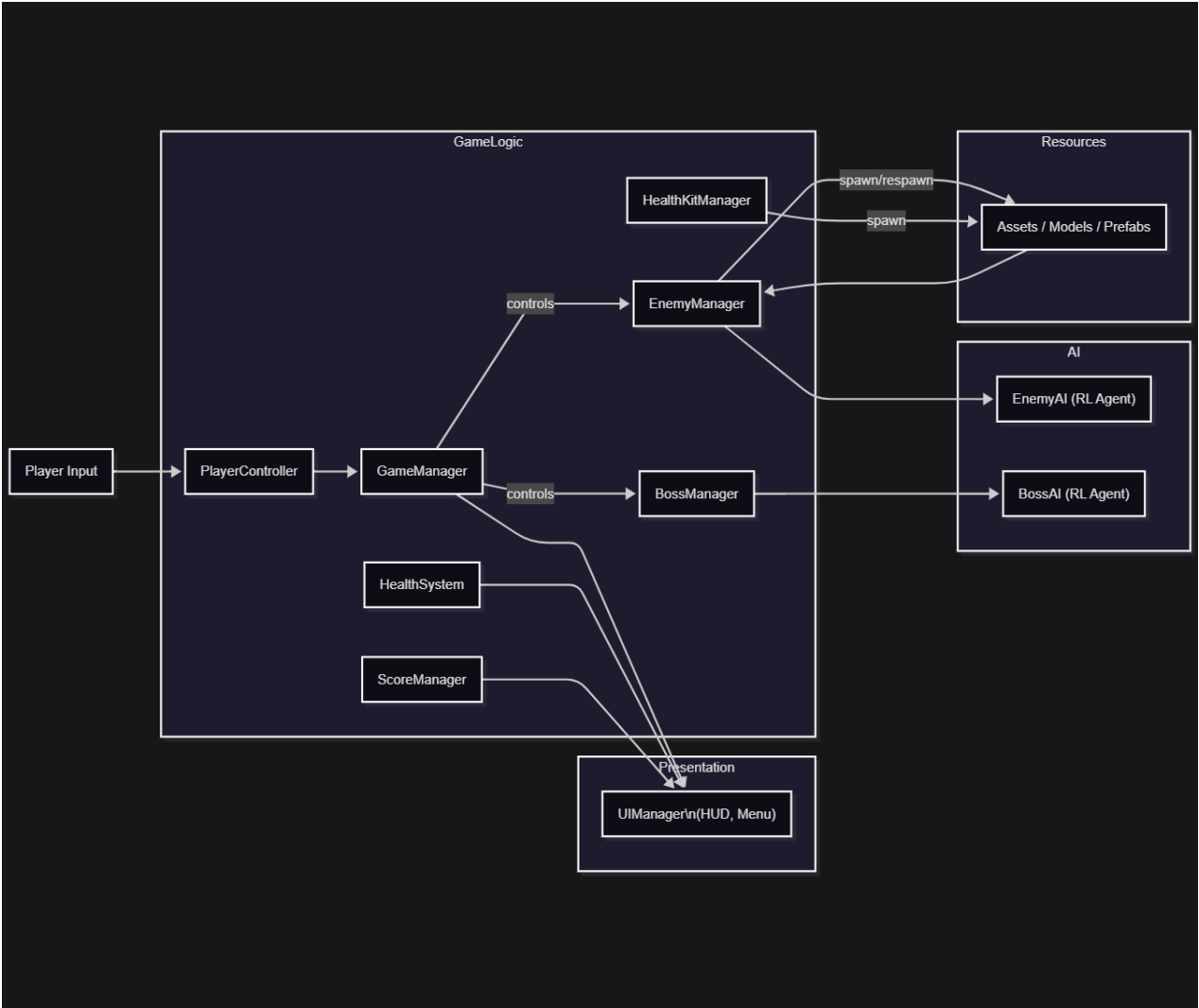
4. System Design

Architectural Overview:

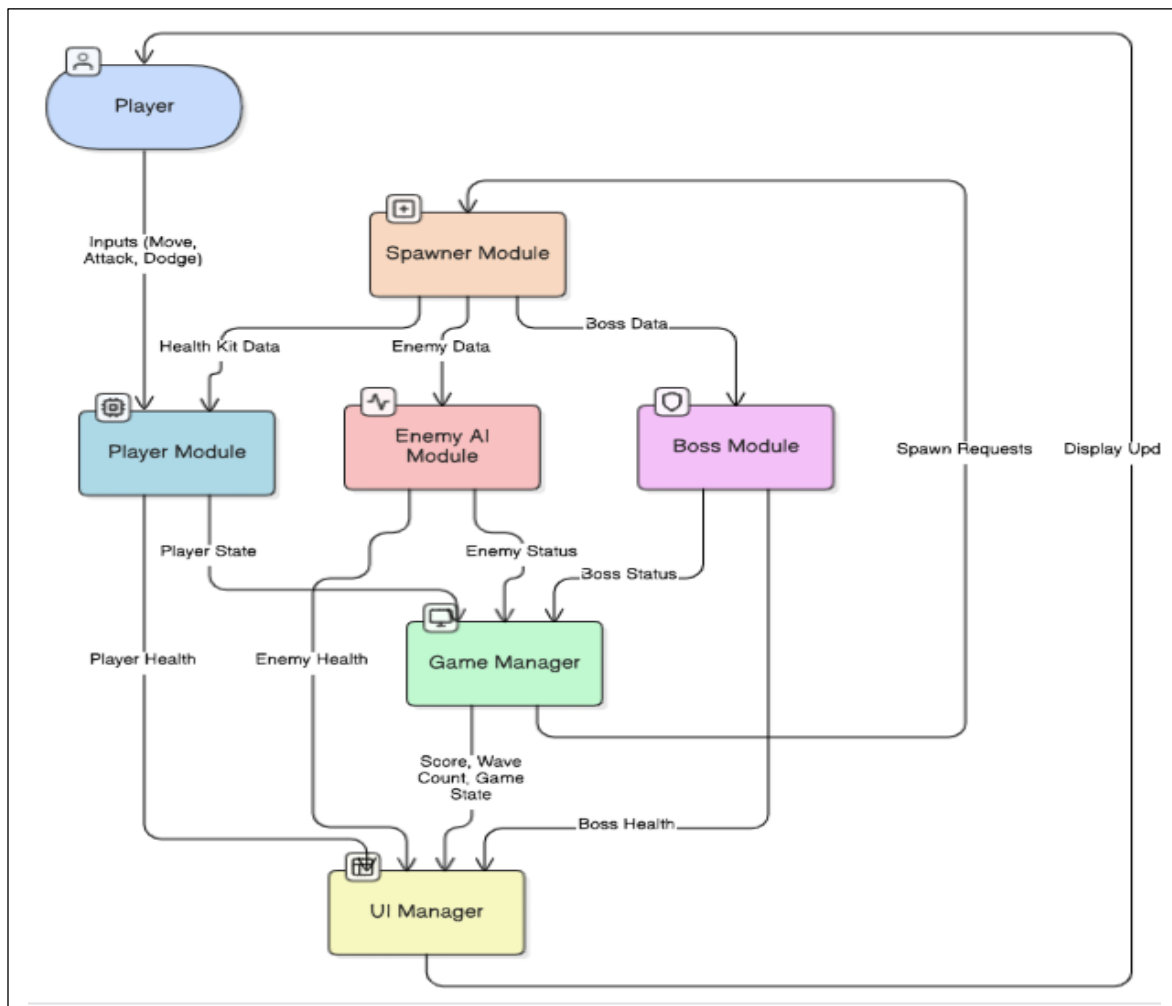
- Unity Engine core manages all game logic.
- **PlayerController:** Handles input, movement, and attacks.
- **EnemyManager:** Controls AI spawning and combat behavior.
- **BossManager:** Advanced AI for boss encounter.
- **GameManager:** Controls game loop, wave progression, and victory conditions.
- **UIManager:** Manages main menu, health bars, and score display.
- **HealthKitManager:** Handles spawning and collection of health kits.

Diagram Placeholders:

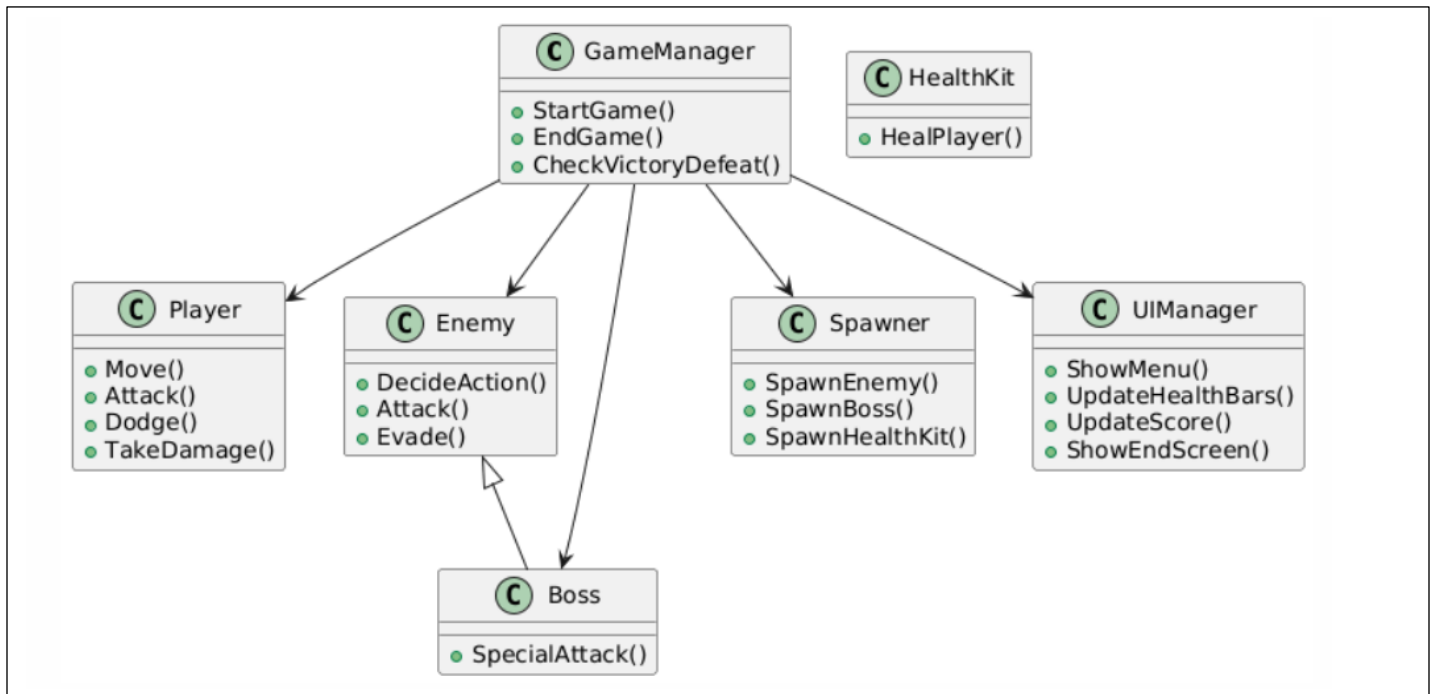
- [System Architecture Diagram]
-



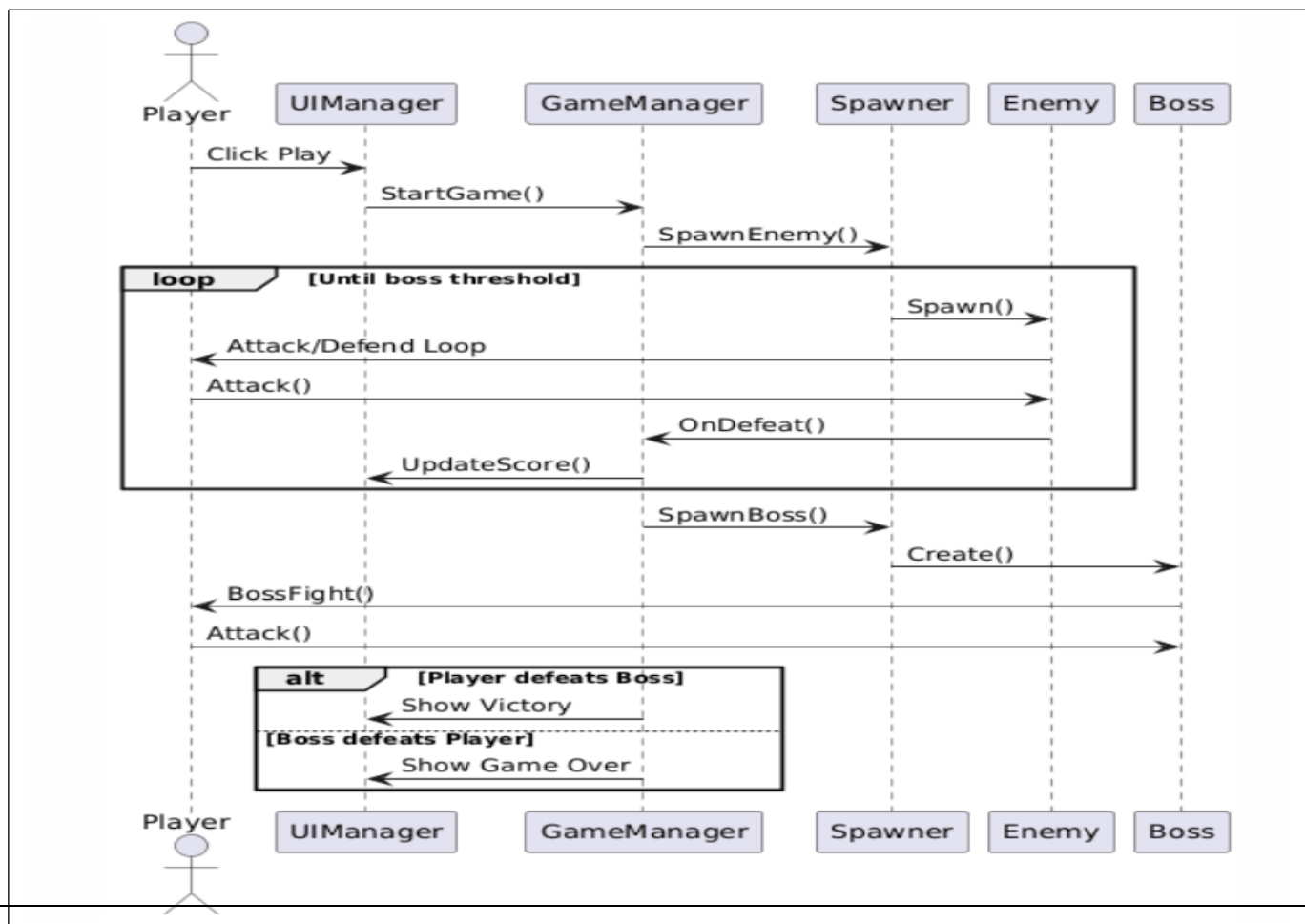
- [DFD Level 1]



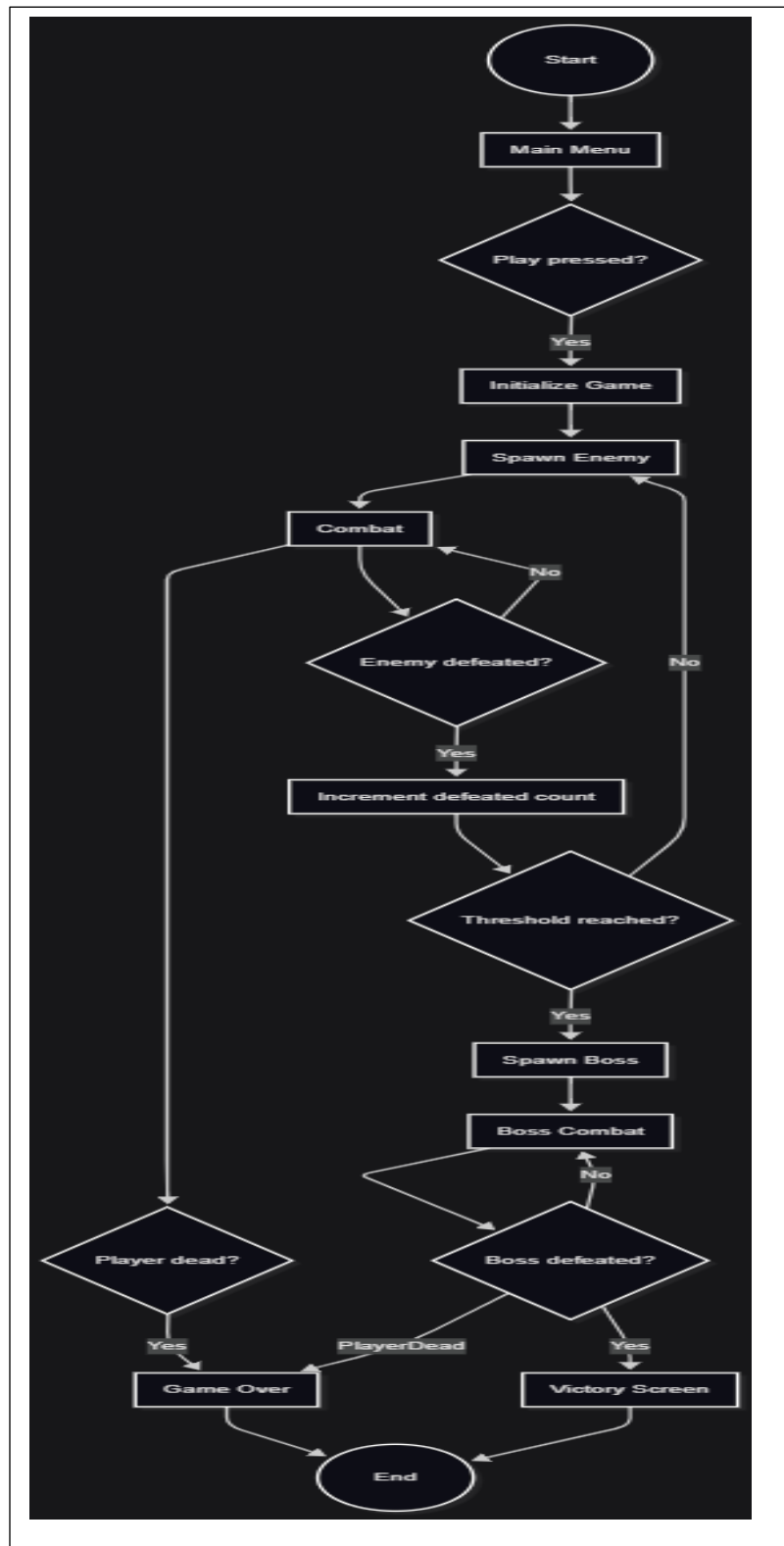
- [Class Diagram]



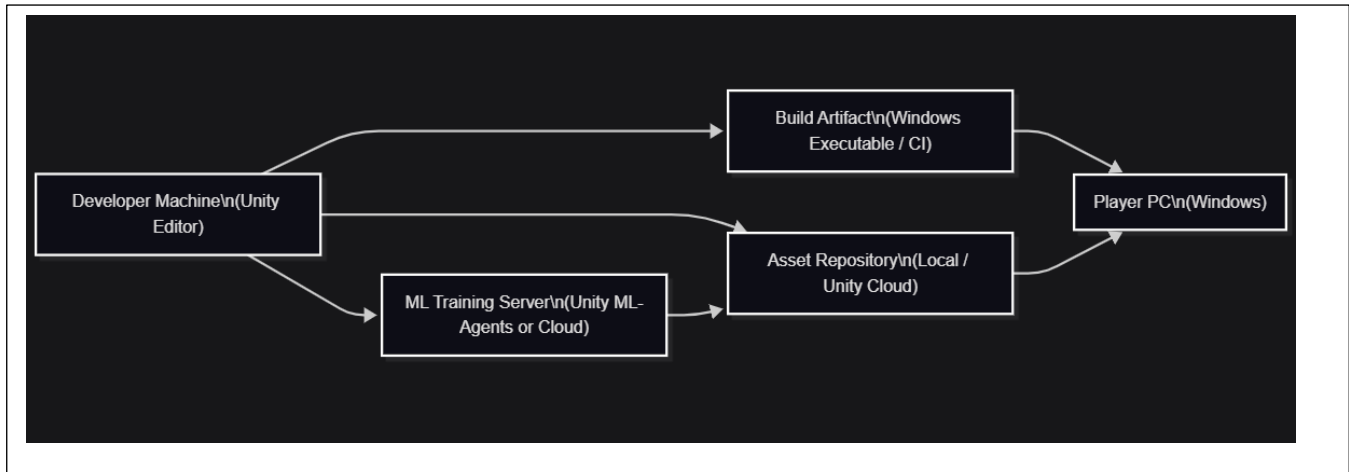
- [Sequence Diagram – Combat Interaction]



- **UI:** Main Menu, HUD (health bars, score counter).
- **Game Loop:** Menu → Spawn Enemy → Wave Progression → Boss → Victory.
- **[Activity Diagram]**



- [Deployment Diagram]



6. Testing


6.1 Test Case Table

Test Case Table					
ID	Test Case Description	Input/Action	Expected Output	Result	Test Case(s)
TC1	Start Game from Main Menu	Click "Play" button	Arena loads, player spawns	Pass	TC2, TC3
TC2	Player Movement	Press WASD keys	Player moves in respective direction	Pass	TC5
TC3	Player Attack	Press attack key	Attack animation plays, enemy health decreases	Pass	TC6
TC4	Health Kit Pickup	Player collides with health kit	Health bar increases	Pass	TC4
TC5	Enemy Defeat → Next Enemy Spawns	Kill enemy	Next enemy appears	Pass	TC7
TC6	Boss Spawn	Defeat 10 enemies	Boss spawns in arena	Pass	TC8

TC7	Boss Defeat	Reduce boss health to 0	Victory screen appears	Pass	TC4, TC9, TC10
TC8	Score Update	Kill enemy	Score counter increases	Pass	TC1
TC9	Player Death	Reduce player health to 0	Game Over message	Pass	TC2, TC3
TC10	UI Elements	During gameplay	Health bar & score update in real-time	Pass	TC10
NFR3	Stable performance during wave progression	Non-Functional	System Architecture, Deployment Diagram	EnemyManager.cs, GameManager.cs	Stress Test
NFR4	AI must provide dynamic challenge using RL	Non-Functional	Sequence Diagram (AI decisions), Class Diagram	EnemyAI.cs, BossAI.cs	AI Test

6.2 Traceability of Requirements → Test Cases

Purpose: Map requirements to corresponding test cases to ensure full coverage.

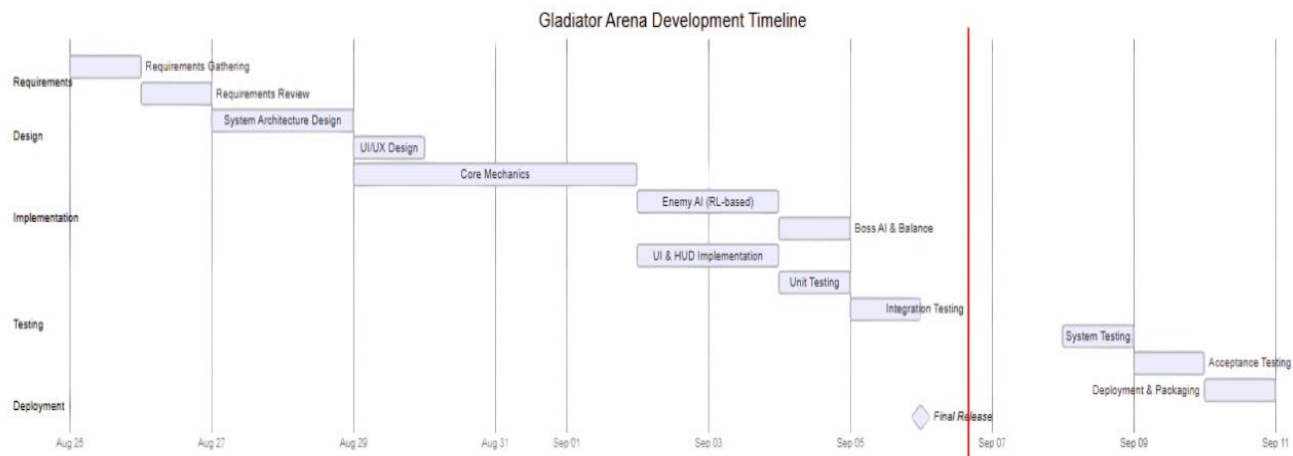
 Traceability Matrix (Requirements → Test Cases)		
Requirement ID	Requirement Description	Linked Test Cases
FR1	Player can move, attack, and dodge	TC2, TC3
FR2	Enemies spawn one at a time	TC5
FR3	Boss spawns after defeating threshold enemies	TC6
FR4	Player can collect health kits to restore health	TC4
FR5	Game ends upon boss defeat	TC7
FR6	Score increases when enemy defeated	TC8
FR7	Player health bar is visible and updates	TC4, TC9, TC10
FR8	Main Menu with Play option	TC1

NFR1	Game should respond to input with minimal delay	TC2, TC3
NFR2	UI elements update in real-time	TC10

7. Project Management

- **Methodology:** Agile (Scrum Sprints)
- **Timeline:** Prototype → Core Mechanics → AI → UI → Testing
- **Risks:**
 - AI difficulty imbalance
 - Performance issues with RL
 - Limited assets causing repetitive gameplay

Development Timeline:



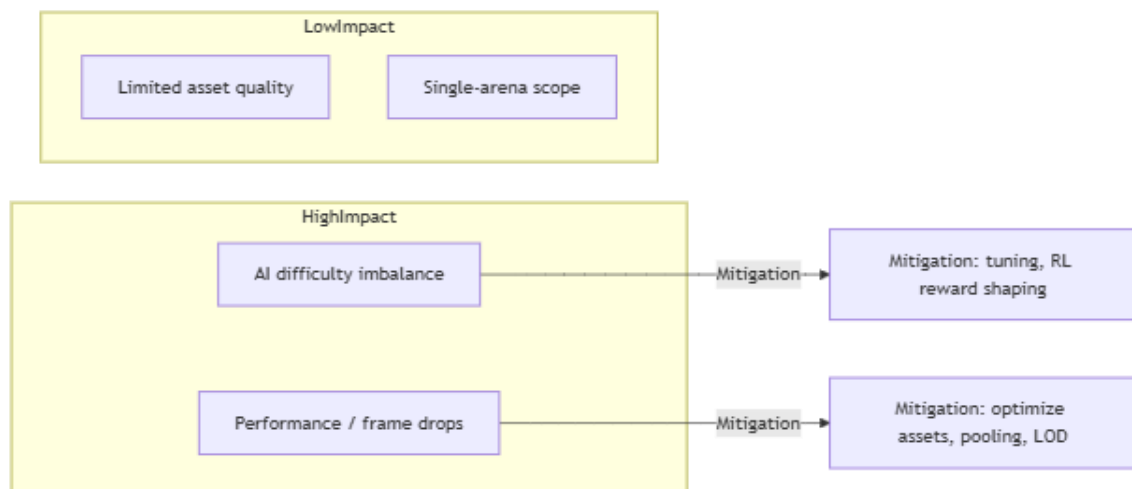
• [Risk Matrix]

- Below is a concise risk matrix for the Gladiator Arena project. The table lists key risks, likelihood, impact, and mitigations. A small Mermaid visualization groups risks by impact.

Risk ID	Risk	Likelihood	Impact	Mitigation	Owner
R1	AI difficulty imbalance (RL behaviour unexpected)	High	High	Reward shaping, staged testing, curriculum	AI Lead

				learning, manual overrides	
R2	Performance drops during wave progression	Medium-High	High	Optimize assets, use LOD, object pooling, performance profiling	Tech Lead
R3	Limited asset quality / repetitiveness	Medium	Medium	Curate assets, tweak materials, reuse with variation	Art Lead
R4	Timeline / scope creep	Medium	Medium	Prioritise MVP features, sprint reviews, strict RTM	PM

- **[Risk Matrix Diagram]**



8. Deployment & Maintenance

- **Deployment:** Windows executable via Unity build.
- **Requirements:** PC with 8GB RAM, GPU support.
- **Maintenance:** Bug fixes, balancing, gameplay improvements.

- **Future Enhancements:** Multiplayer, additional enemy types, progressive difficulty, new arenas, power-ups.
-

9. Conclusion & Recommendations

Chains of Valor demonstrates dynamic combat and wave-based challenges using AI-driven enemies. Future improvements could include co-op multiplayer, more enemy types, level progression, and power-ups for enhanced engagement.

10. References

- Unity Documentation: <https://docs.unity3d.com>
 - Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*.
 - Unity Asset Store Packages
-

11. Appendices

- Gameplay Screenshots
 - UI Mockups
 - Key Code Snippets
-

Diagram Placeholders Summary

1. Context Diagram / DFD Level 0
2. Use Case Diagram
3. System Architecture Diagram
4. DFD Level 1
5. Class Diagram
6. Sequence Diagram
7. Activity Diagram
8. Deployment Diagram
9. Gantt Chart / Timeline
10. Risk Matrix
11. Requirement Traceability Matrix (RTM)
12. Test Case Table

13. Traceability of Requirements → Test Cases
