SOFTWARE DESIGN DOCUMENT

2D SWORD & SHIELD FIGHTING GAME

1. INTRODUCTION

1.1 Purpose

The purpose of this Software Design Document (SDD) is to describe the **overall design**, **architecture**, and **implementation approach** of the *2D Sword & Shield Fighting Game*.

This document explains **how the system will be structured in the Godot Engine**, how the main components will interact, and which design choices guide the development process.

The document helps the development team follow the same structure while building and testing the game.

The document is also written for the course instructor, who will check if the design matches the requirements in the SRS.

1.2 Project Scope

In Scope:

- 1v1 local multiplayer gameplay with sword and shield combat mechanics.
- Player actions: movement (right or left), dash (right or left), attacking (upward, horizontal, downward), and defending (upward, horizontal, downward).
- Game menus: Main Menu, Options, Tutorial, Credits, and Pause Menu.
- Game results: match end screen with options to rematch or return to the main menu.
- Audio and resolution settings adjustable in the Options menu.

Out of Scope:

- AI-controlled opponents.
- Multiple playable characters.
- Online multiplayer functionality.

1.3 Intended Audience

- **Course Instructor:** The document helps the instructor review the assignment and verify that the design is consistent with the requirements.
- **Developers:** The document guides the development team during design, implementation, and testing by explaining the system structure and the main components.

1.4 References

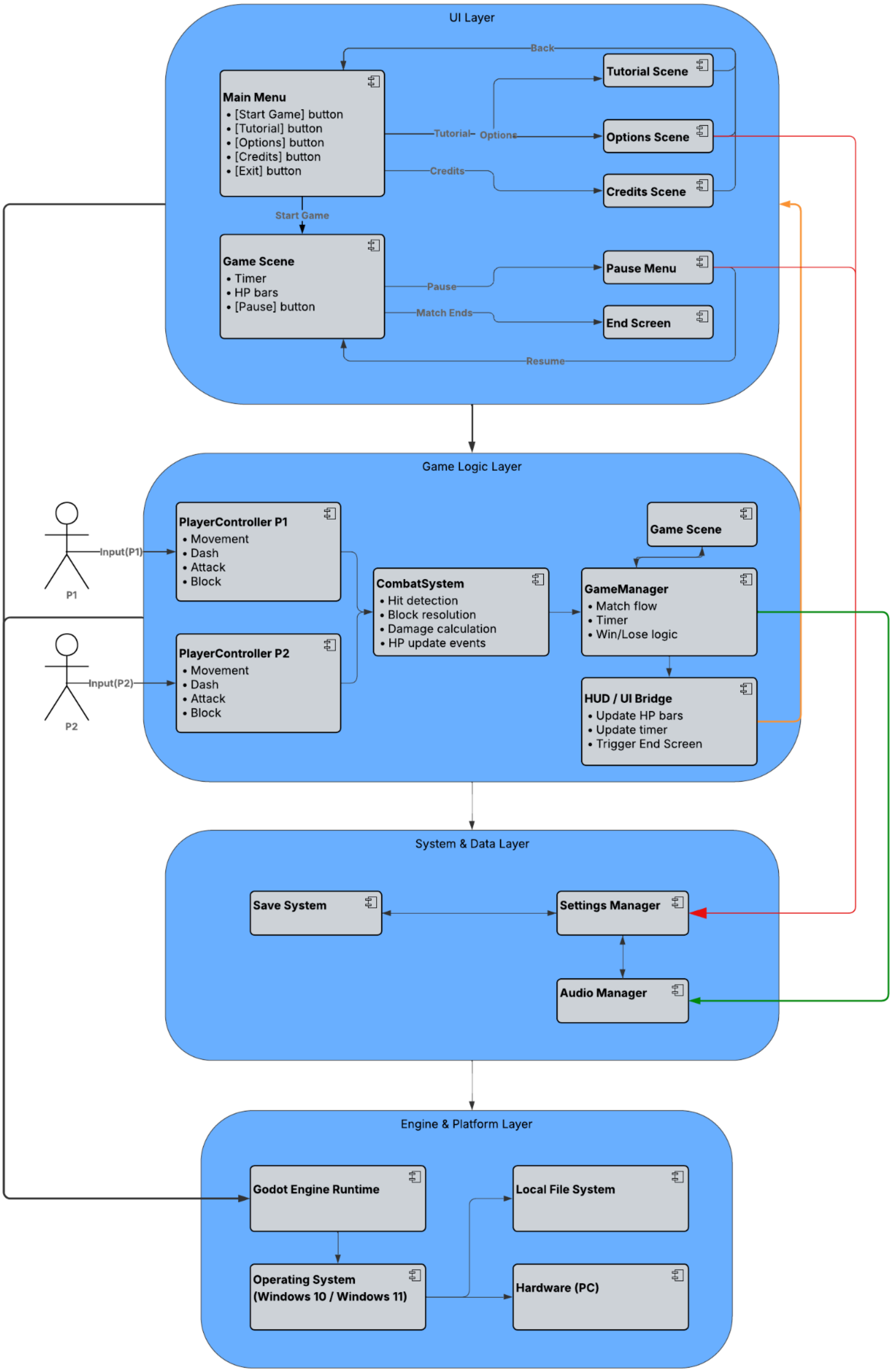
- **GitHub Repository Link:**https://github.com/OguzhanCel/YZM2021-Project
- **GitHub Project Board Link:** github.com/users/OguzhanCel/projects/3
- Use Case names (like UC-04) are referenced from the previously submitted SRS document.

2. SYSTEM OVERVIEW

2.1 Project Summary

The *2D Sword & Shield Fighting Game* is a standalone local 1v1 desktop application built using the Godot Engine and GDScript. Designed for Windows 10 and 11, the game provides fast-paced combat with core mechanics such as directional attacking, defending, and dashing. It includes a complete gameplay loop supported by a main menu, interactive tutorial, pause functionality, and a match results screen. The system runs fully offline without relying on external servers or APIs, and uses Godot's built-in physics, rendering, and input systems to handle player actions, collisions, and combat interactions. User preferences, such as audio and display settings, are stored locally using Godot's file management tools.

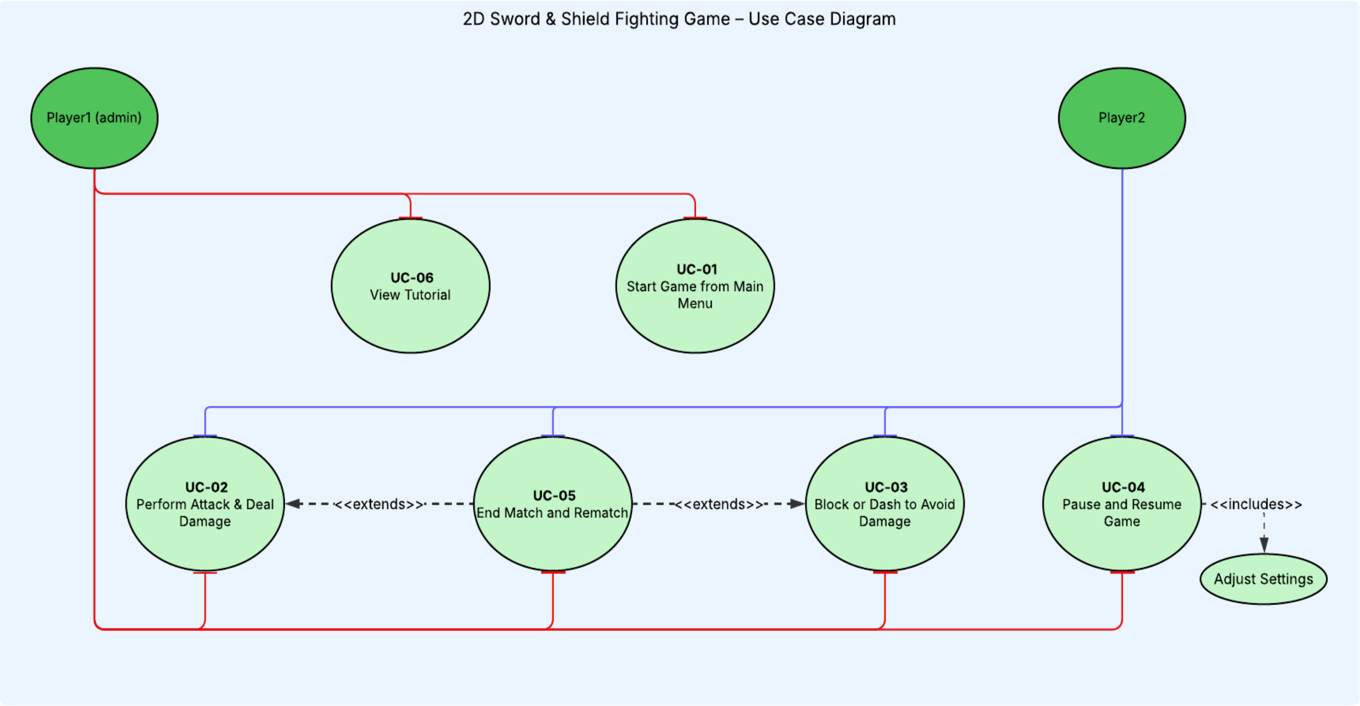
2.2 System Diagram



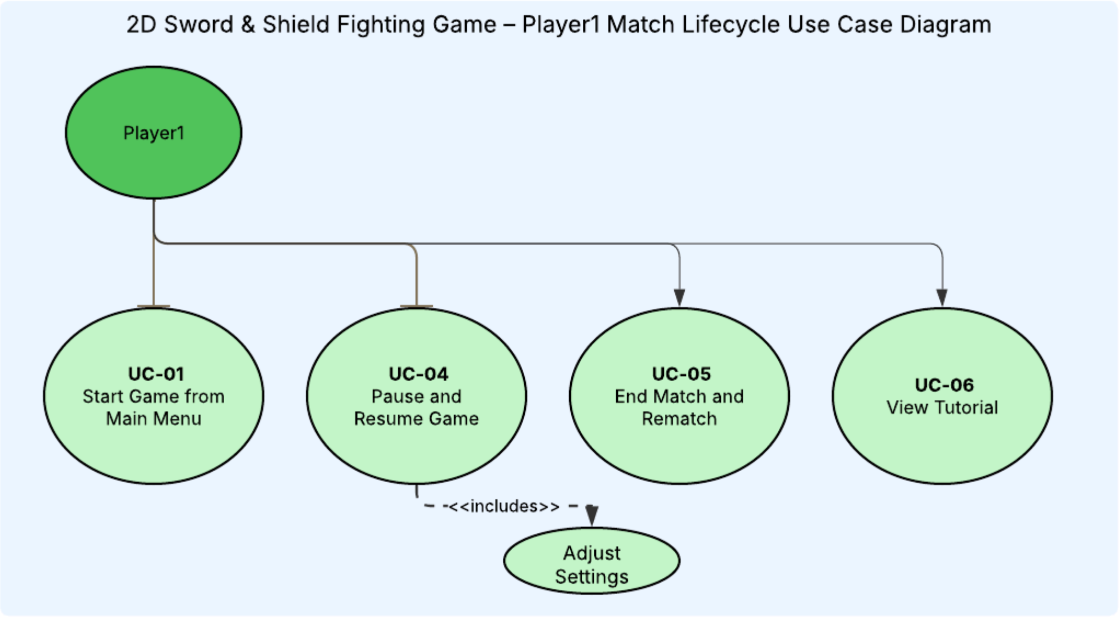
3. UML DIAGRAMS

3.1 Use Case Diagrams

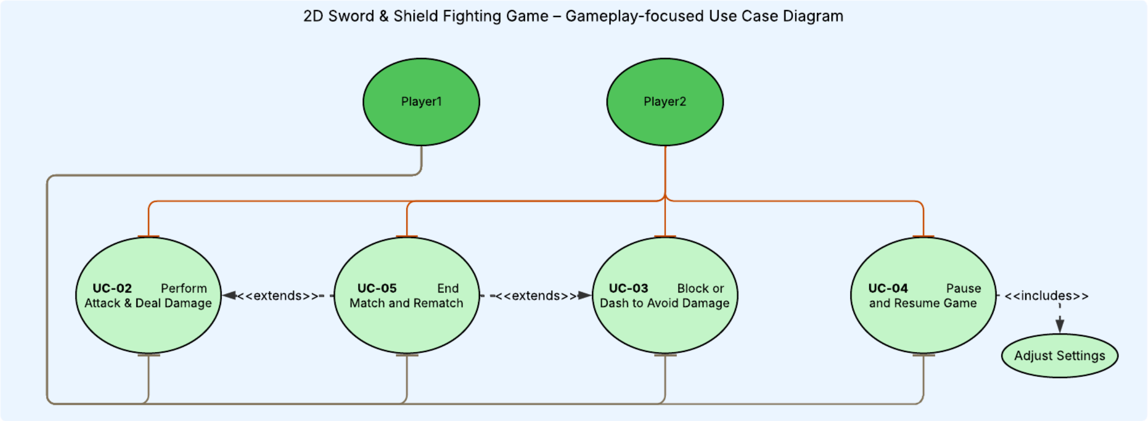
3.1.1 Overall Use Case Diagram



3.1.2 Match Lifecycle Use Case Diagram

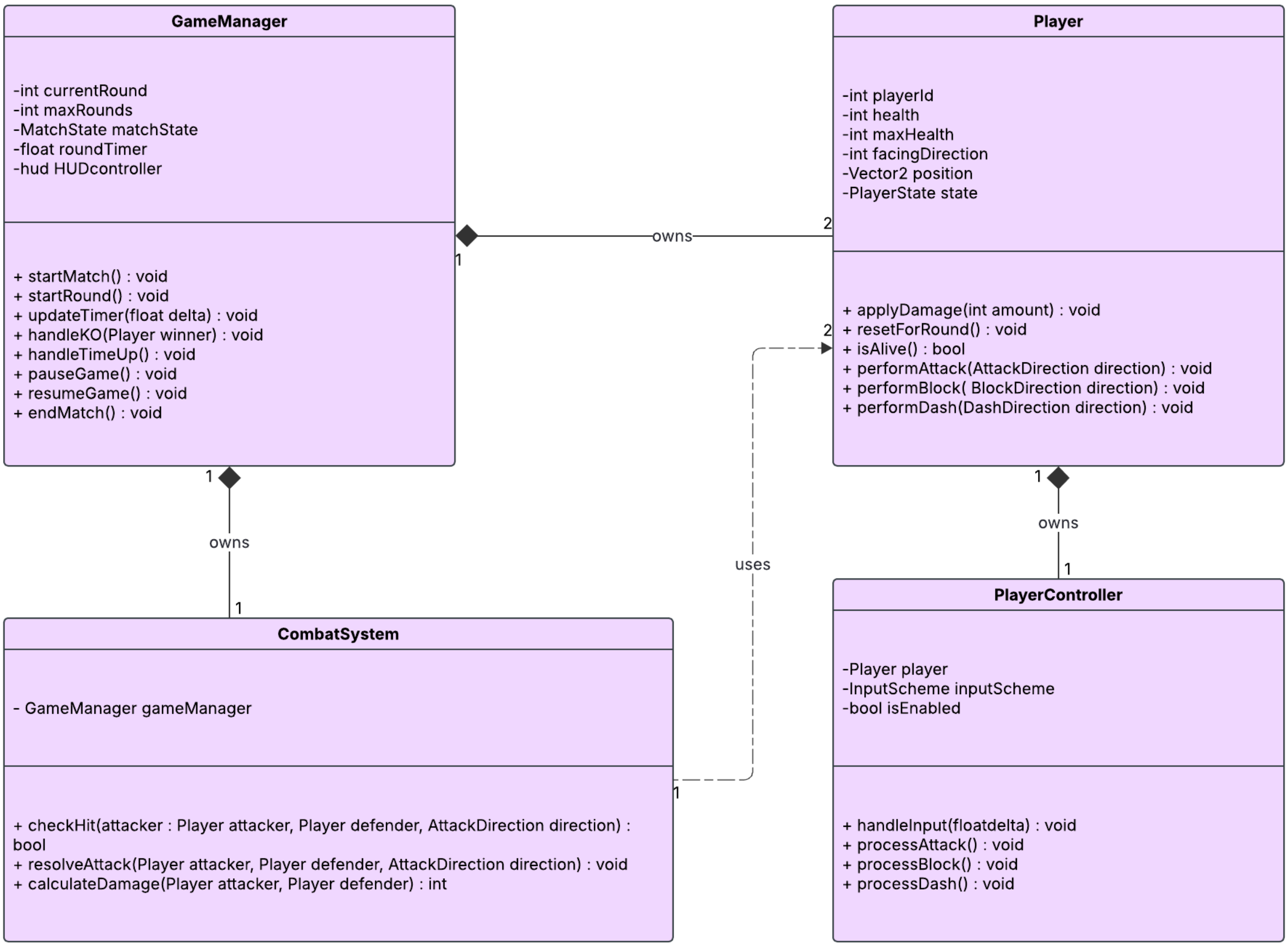

2D Sword & Shield Fighting Game – Player1 Match Lifecycle Use Case Diagram
Player1
UC-01 Start Game from Main Menu
UC-04 Pause and Resume Game
UC-05 End Match and Rematch
UC-06 View Tutorial
<<includes>>
Adjust Settings

3.1.3 Gameplay-focused Use Case Diagram


2D Sword & Shield Fighting Game – Gameplay-focused Use Case Diagram
Player1
Player2
UC-02 Perform Attack & Deal Damage
<<extends>>
UC-05 End Match and Rematch
<<extends>>
UC-03 Block or Dash to Avoid Damage
UC-04 Pause and Resume Game
<<includes>>
Adjust Settings

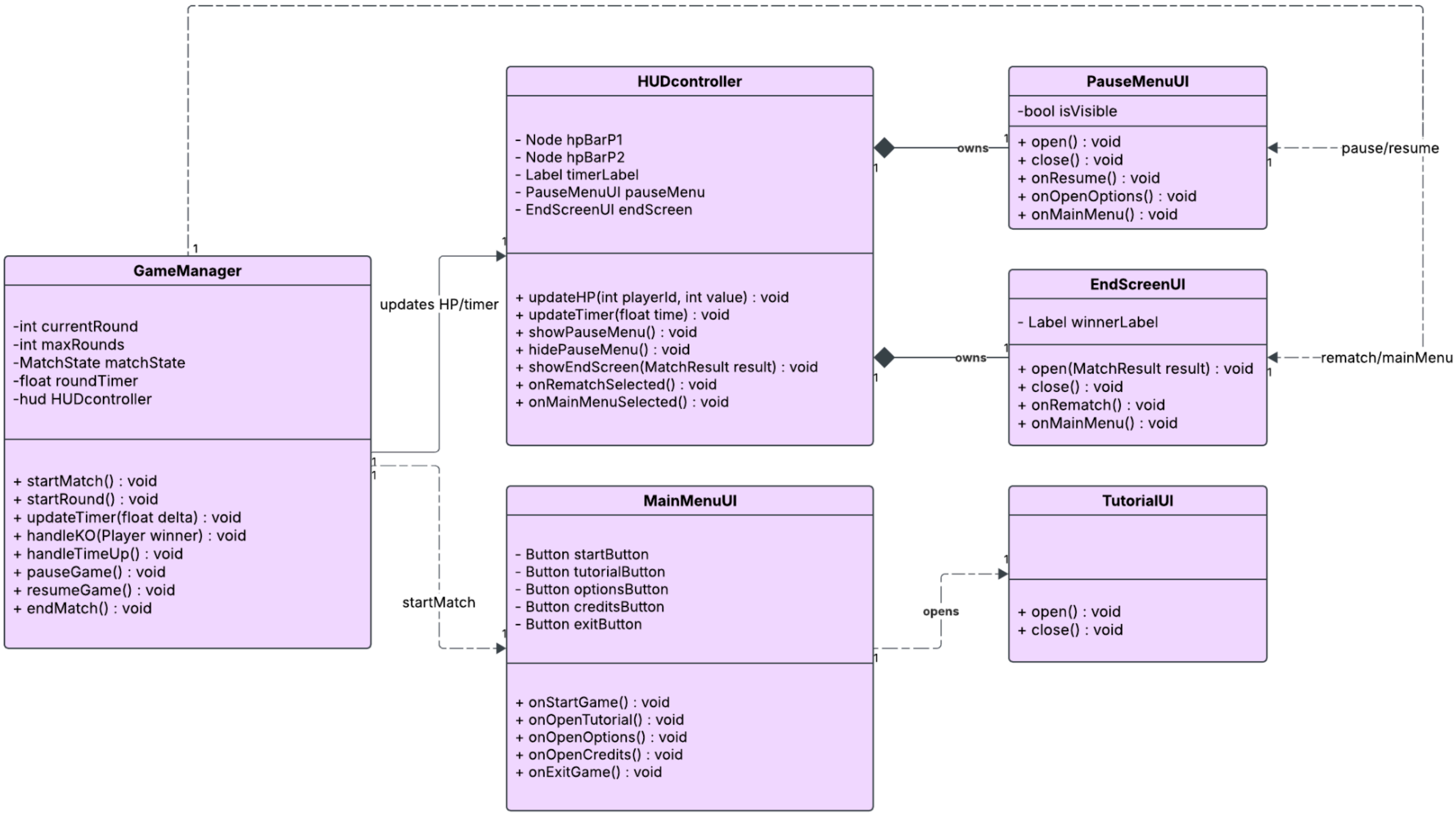
3.2 Class Diagrams

3.2.1 UI Class Diagram



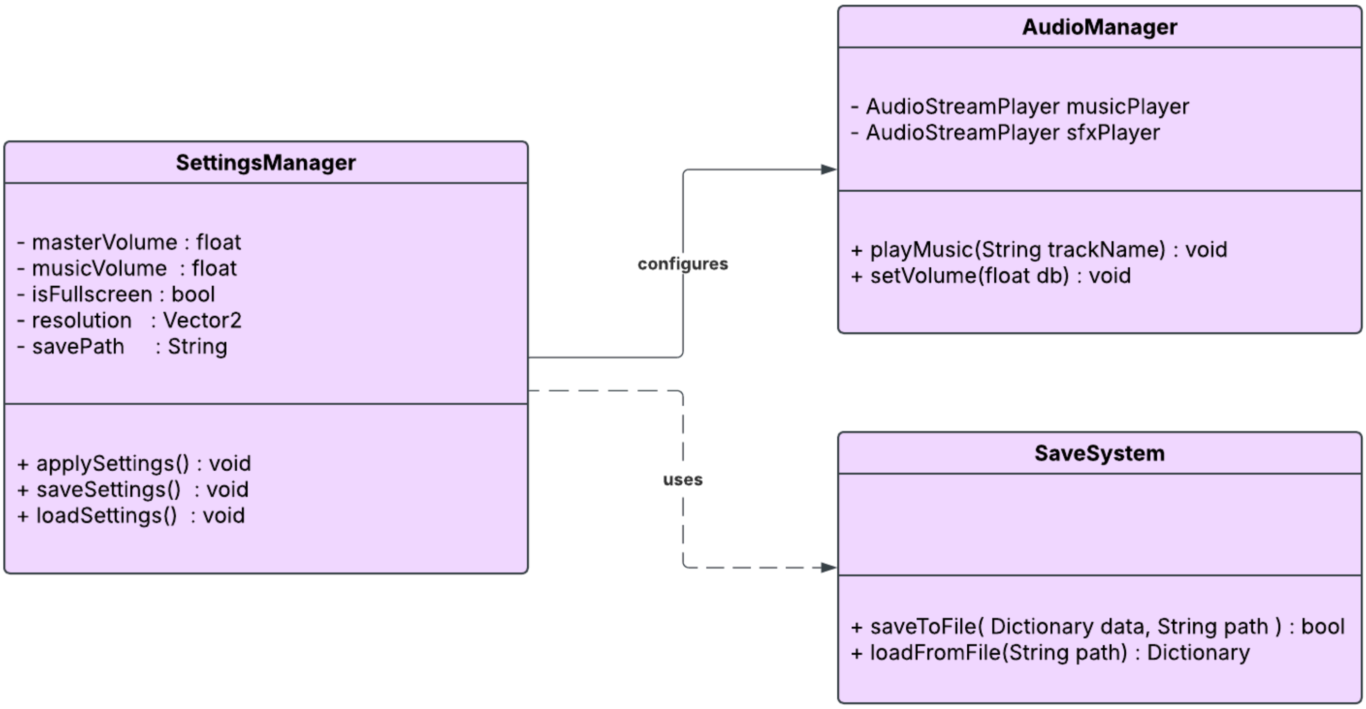
- This diagram describes the structure and interactions of the user interface components.
- HUDController owns the in-game UI elements (PauseMenuUI and EndScreenUI) and updates health bars, the timer, and menu visibility.
- MainMenuUI handles main menu navigation and opens the tutorial scene through TutorialUI.
- GameManager communicates with UI components to start matches, update the HUD, pause or resume gameplay, and trigger end-of-match screens.
- All associations reflect one-way dependencies, showing that UI elements rely on GameManager for game flow but are not part of the game logic layer.

3.2.2 Core Gameplay Class Diagram



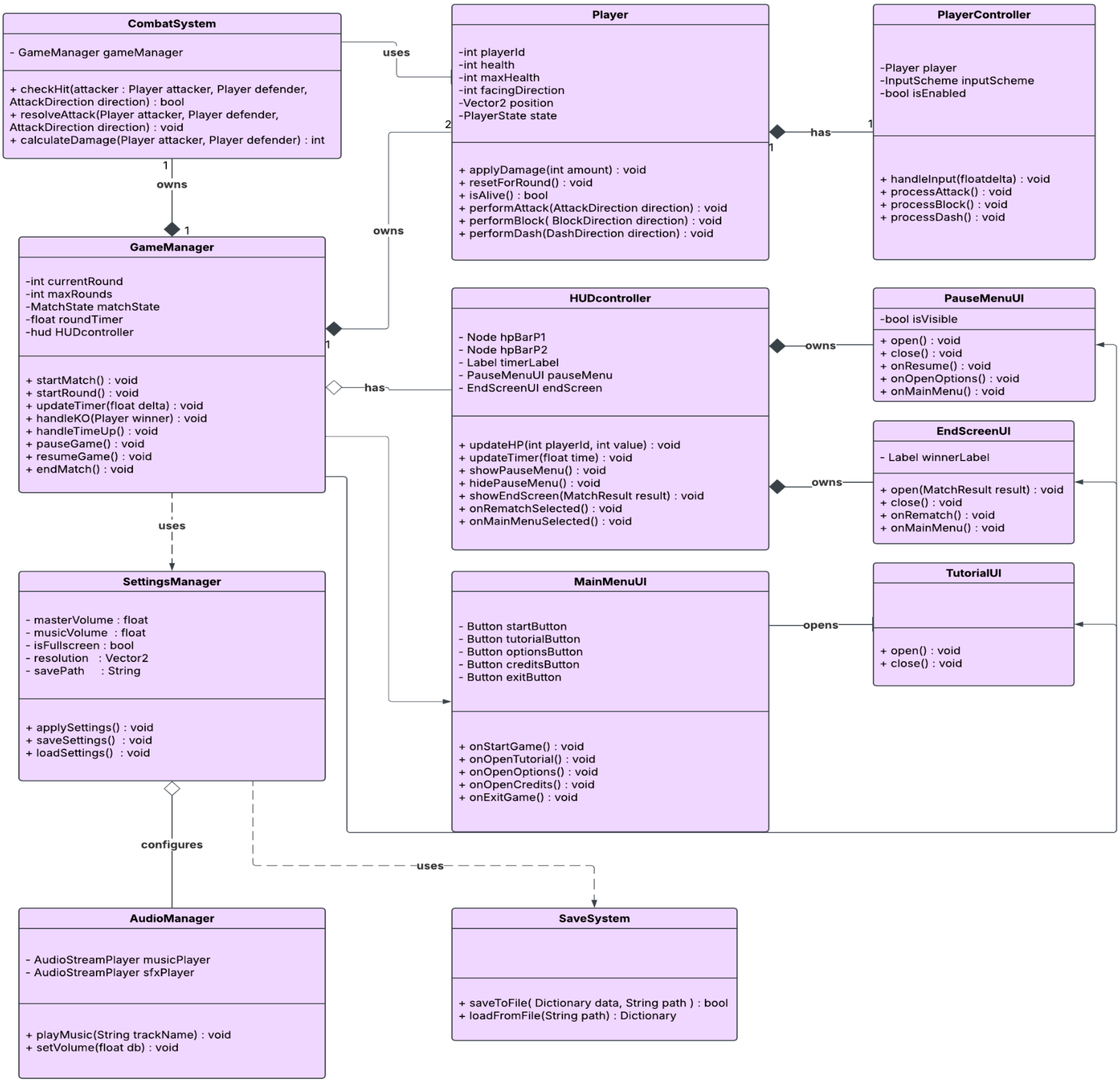
- This diagram describes the structure and interactions of the user interface components.
- HUDController owns the in-game UI elements (PauseMenuUI and EndScreenUI) and updates health bars, the timer, and menu visibility.
- MainMenuUI handles main menu navigation and opens the tutorial scene through TutorialUI.
- GameManager communicates with UI components to start matches, update the HUD, pause or resume gameplay, and trigger end-of-match screens.
- All associations reflect one-way dependencies, showing that UI elements rely on GameManager for game flow but are not part of the game logic layer.

3.2.3 Settings & Data Class Diagram



- This diagram models the settings and persistence subsystem of the game.
- SettingsManager keeps the current user preferences such as volume levels, resolution, and fullscreen mode, and provides operations to apply, load, and save these settings.
- AudioManager is configured by SettingsManager to update runtime audio levels and control background music playback.
- SaveSystem is a utility component that SettingsManager uses to serialize and deserialize settings to a local file, satisfying the requirements for storing user preferences and audio configuration.

3.2.4 Overall System Class Overview Diagram

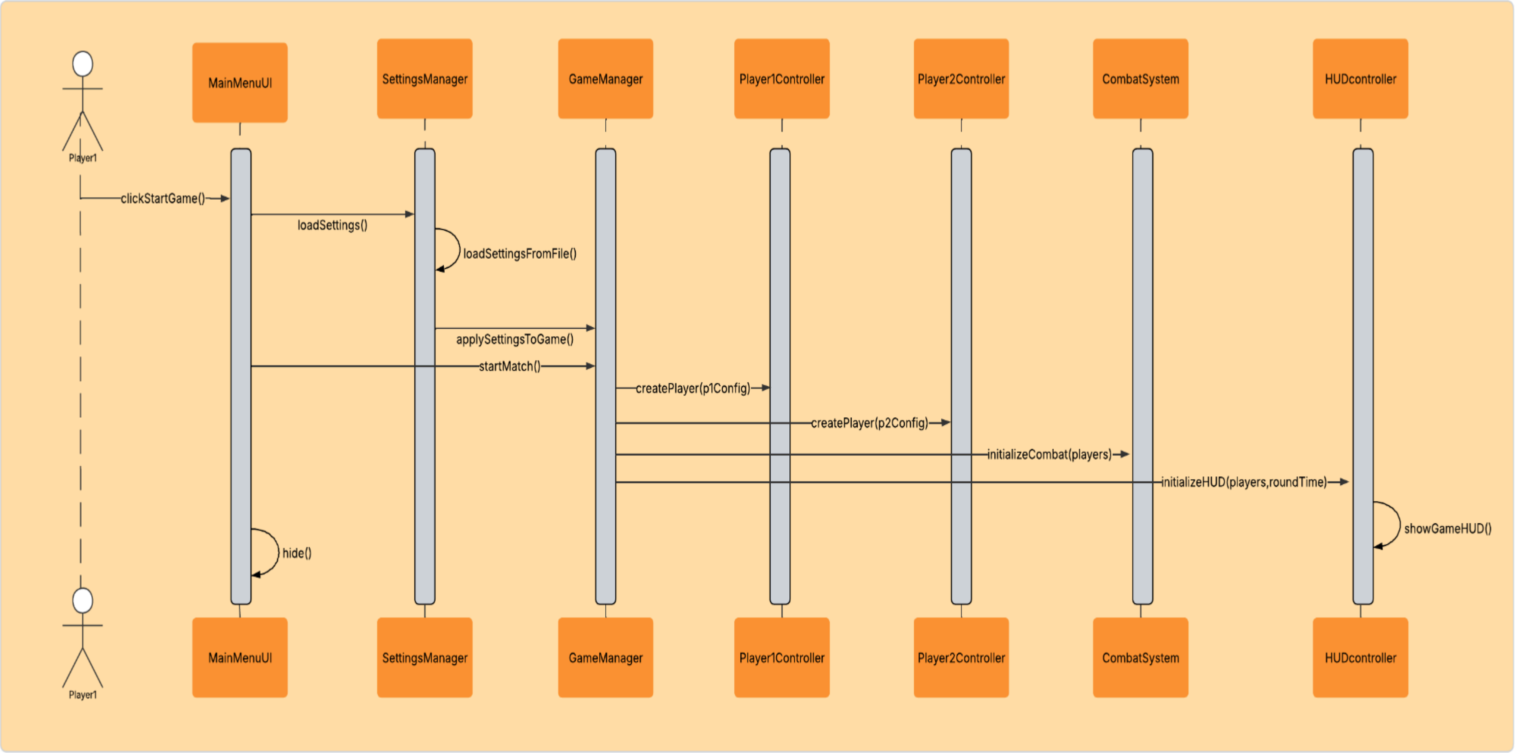


- This diagram provides an overall view of the main classes and how the subsystems are connected.
- **GameManager** coordinates the core gameplay flow and interacts with the player entities, combat logic, and UI controllers. The UI layer (**HUDController, MainMenuUI**) communicates with **GameManager** to reflect game state and receive user input.
- **SettingsManager** manages user configuration and collaborates with **AudioManager** and **SaveSystem** to apply and persist settings.

3.3 Sequence Diagrams

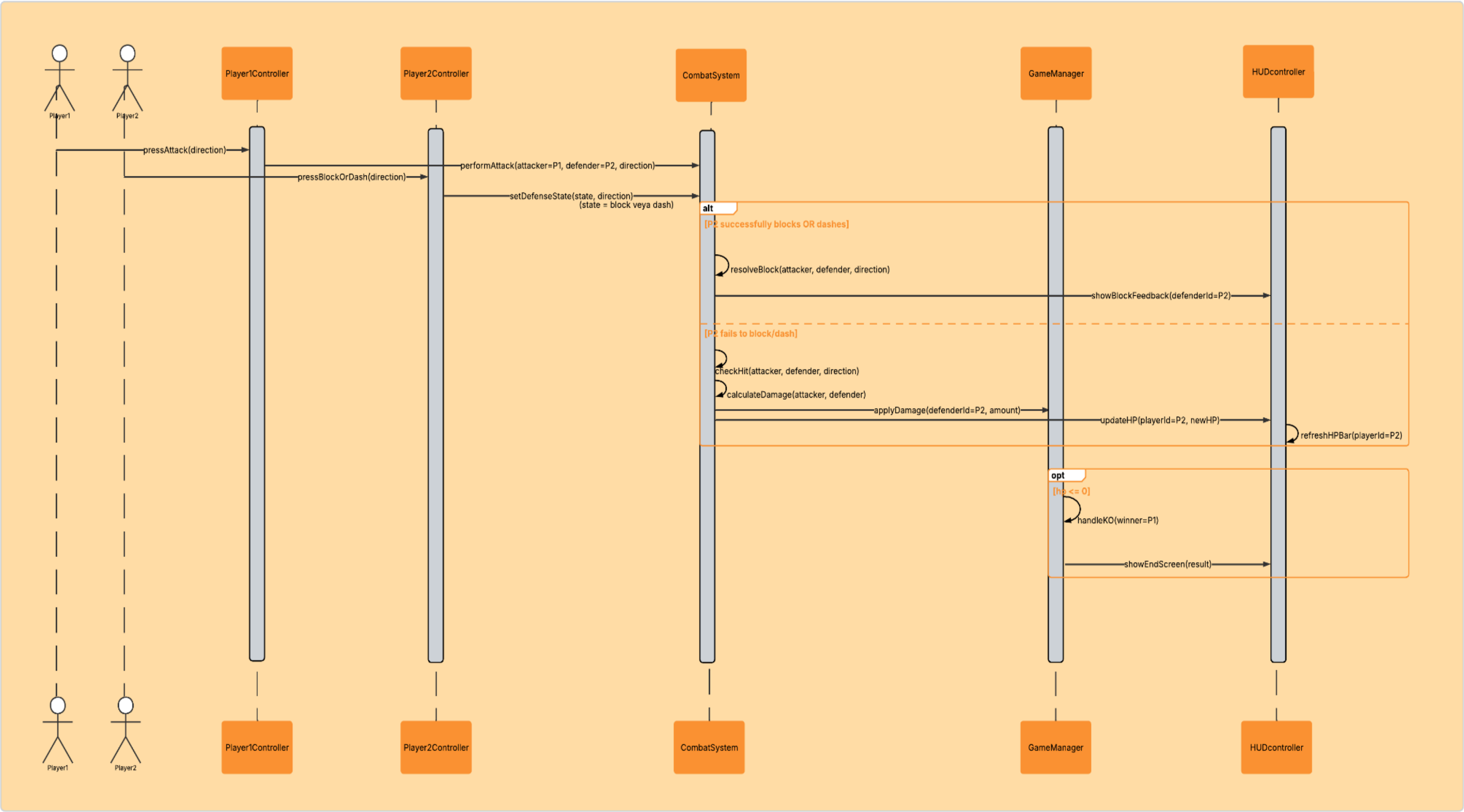
3.3.1 Sequence Diagram: Start Game from the Main Menu

UC-01 Sequence Diagram(Start Game from Main Menu)



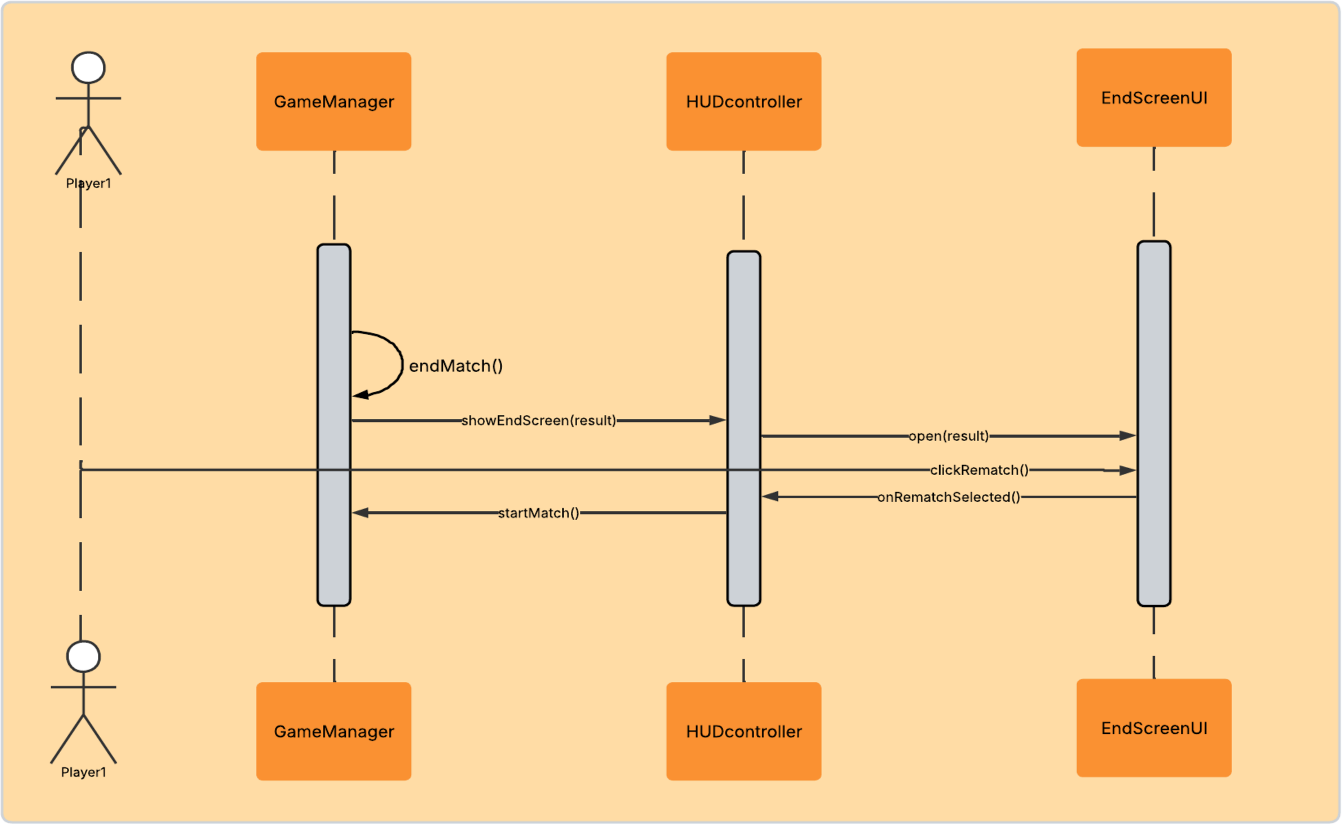
3.3.2 Sequence Diagram: Perform Attack and Deal Damage

UC-02/UC-03 Sequence Diagram(Perform Attack and Deal Damage)


Player1
Player2
Player1Controller
Player2Controller
CombatSystem
GameManager
HUDcontroller
pressAttack(direction)
performAttack(attacker=P1, defender=P2, direction)
pressBlockOrDash(direction)
setDefenseState(state, direction)
(state = block veya dash)
alt
[P2 successfully blocks OR dashes]
resolveBlock(attacker, defender, direction)
showBlockFeedback(defenderId=P2)
[P2 fails to block/dash]
checkHit(attacker, defender, direction)
calculateDamage(attacker, defender)
applyDamage(defenderId=P2, amount)
updateHP(playerId=P2, newHP)
refreshHPBar(playerId=P2)
opt
[hp <= 0]
handleKO(winner=P1)
showEndScreen(result)
Player1
Player2
Player1Controller
Player2Controller
CombatSystem
GameManager
HUDcontroller

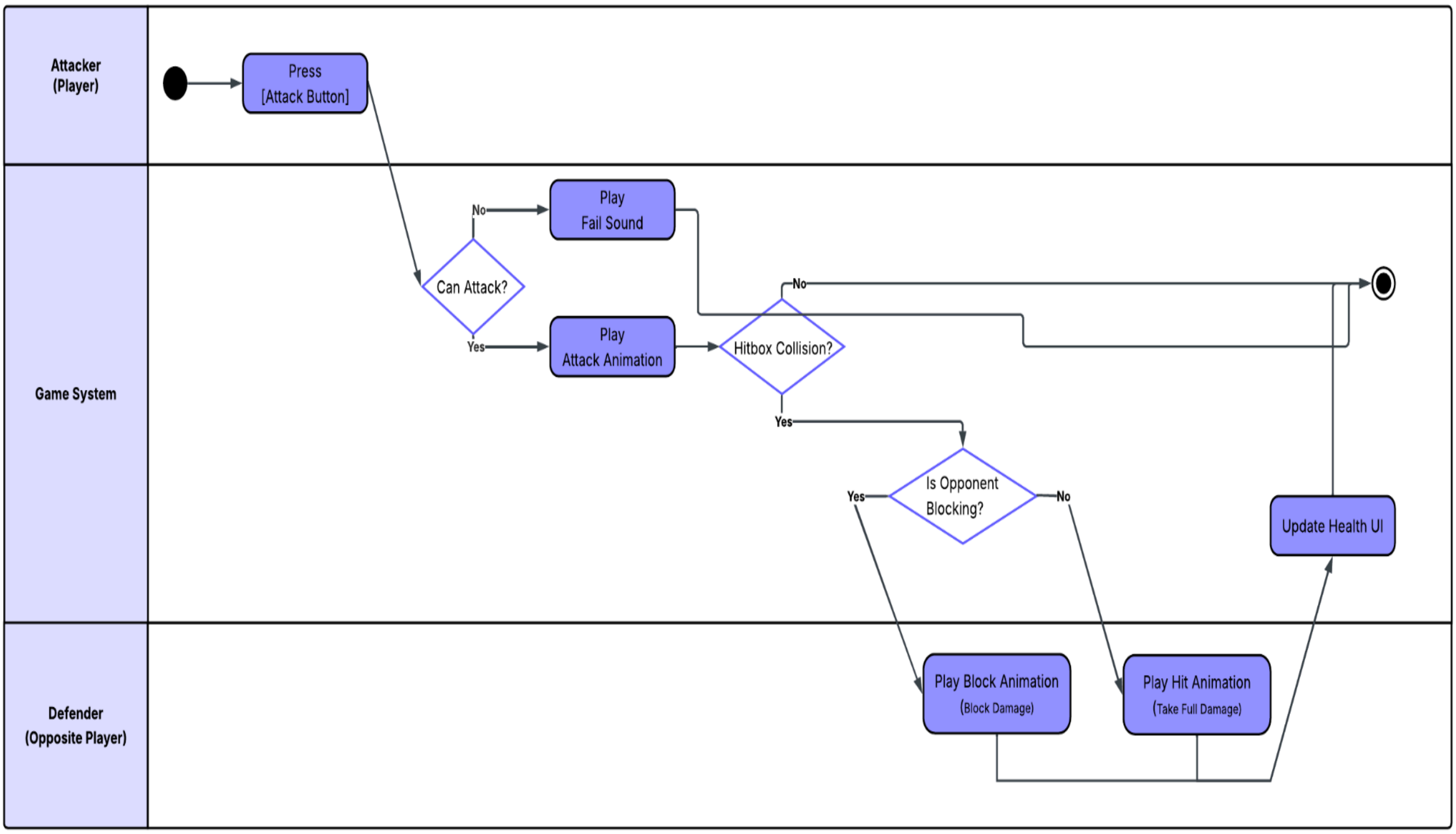
3.3.3 Sequence Diagram: End Game & Rematch

UC-05 Sequence Diagram(End Game & Rematch)

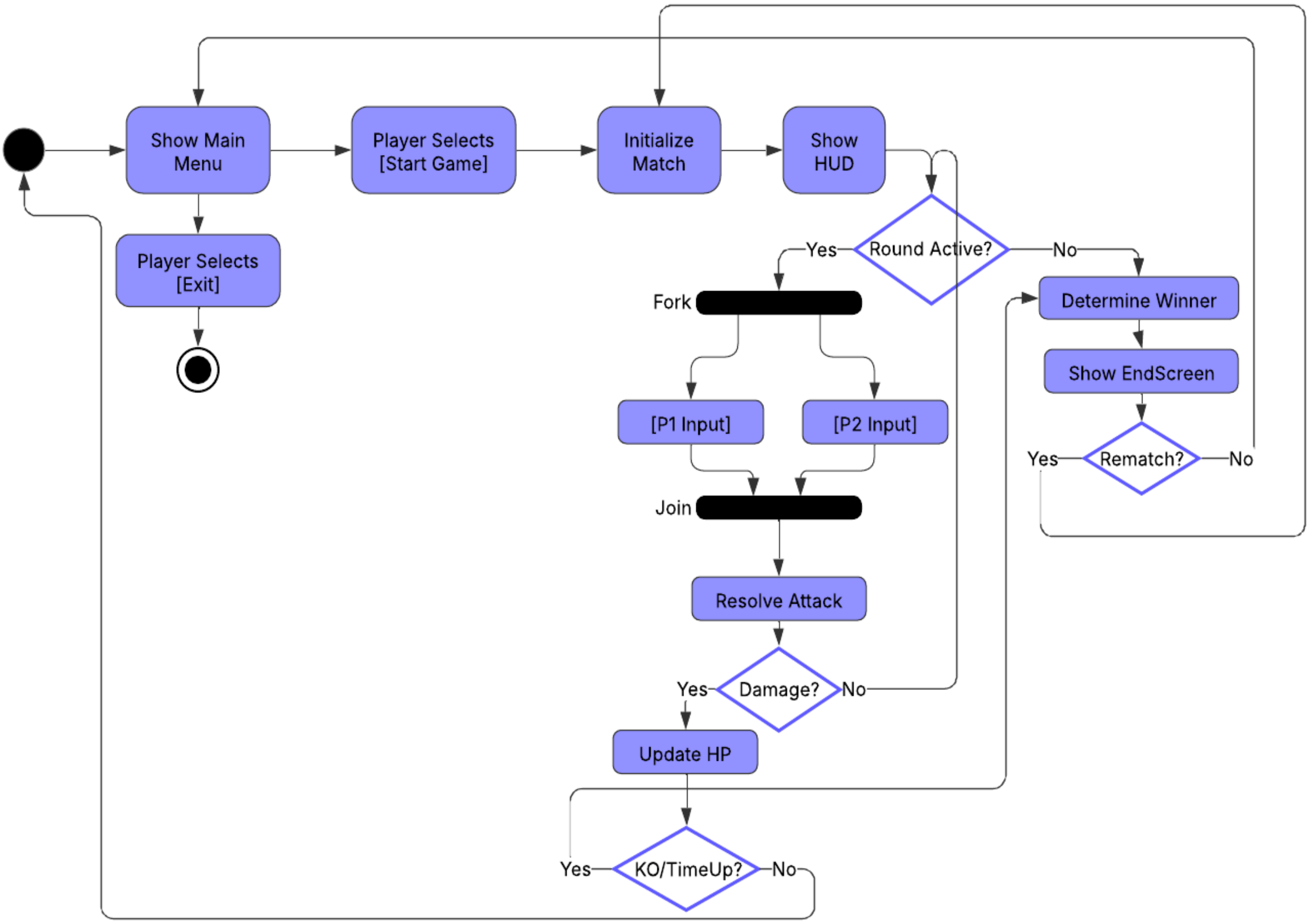


3.4 Activity Diagrams

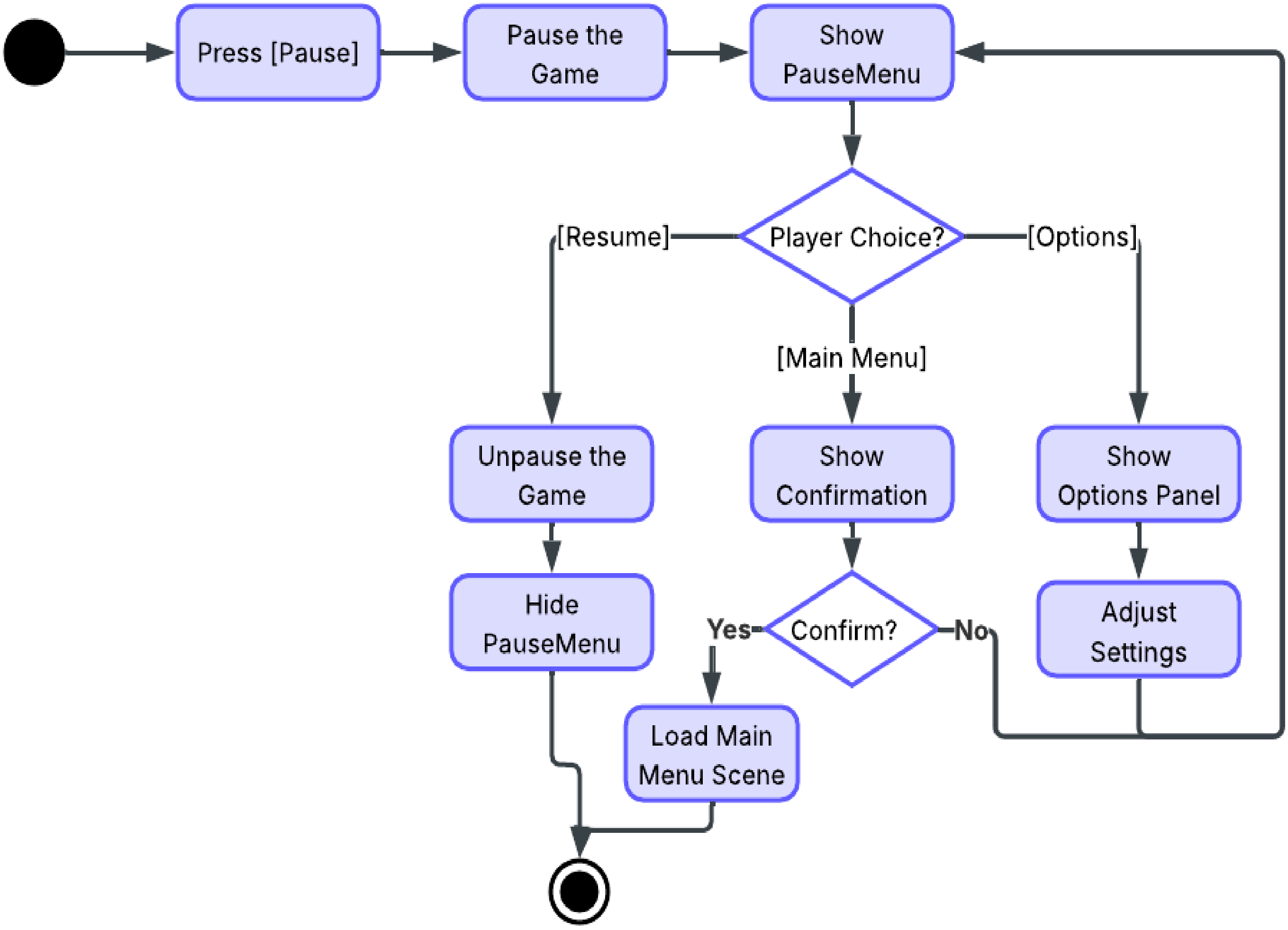
3.4.1 Activity Diagram: Match Flow



3.4.2 Activity Diagram: Attack Resolution

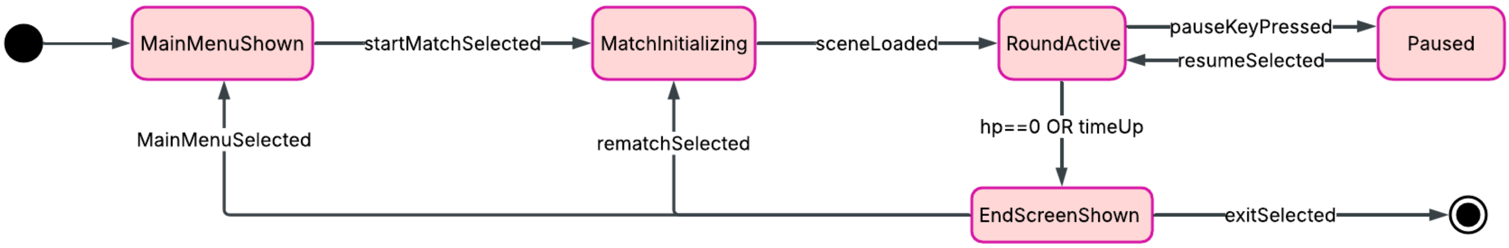


3.4.3 Activity Diagram: PauseMenu

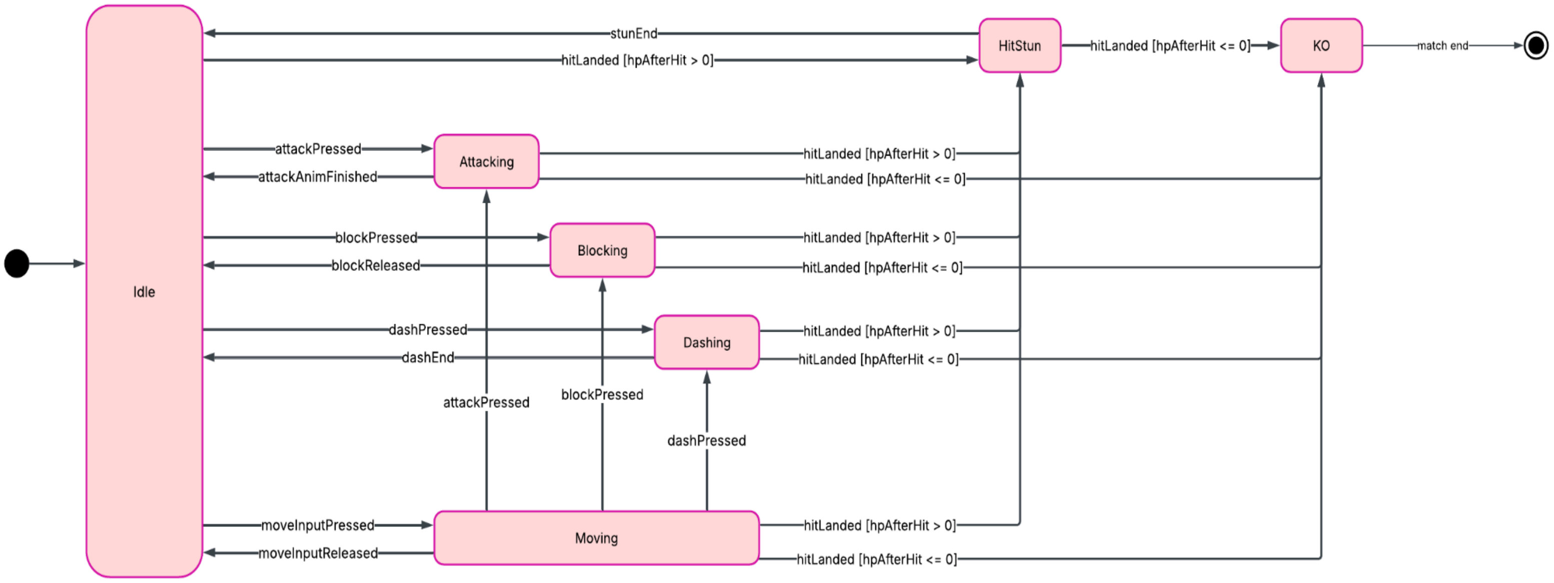


3.5 State Machine Diagrams

3.5.1 GameManager State Machine Diagram(Match Lifecycle)



3.5.2 Player Combat State Machine Diagram


Idle
stunEnd
hitLanded [hpAfterHit > 0]
HitStun
hitLanded [hpAfterHit <= 0]
KO
match end
attackPressed
attackAnimFinished
Attacking
hitLanded [hpAfterHit > 0]
hitLanded [hpAfterHit <= 0]
blockPressed
blockReleased
Blocking
hitLanded [hpAfterHit > 0]
hitLanded [hpAfterHit <= 0]
dashPressed
dashEnd
Dashing
hitLanded [hpAfterHit > 0]
hitLanded [hpAfterHit <= 0]
attackPressed
blockPressed
dashPressed
moveInputPressed
moveInputReleased
Moving
hitLanded [hpAfterHit > 0]
hitLanded [hpAfterHit <= 0]

Team Members

Project Manager and Core Systems Engineer:

Oğuzhan Çelik

Gameplay Engineer:

Yunus Emre Yılmaz

Menu and UI Engineer:

Tuğana Öykü Yıldız