SE Project Document

윷놀이 게임 설계 및 구현 리포트



**SE\_Team\_08**

서규민(20225679)

윤지훈(20225889)

이재민(20235464)

박건희(20214897)

# 목차

1. **프로젝트 개요**
2. **유스케이스 모델**
3. **기타 요구사항**
4. **상세 설계**
5. **UI 교체설계 핵심**
6. **UI 교체 시나리오 분석**
7. **설계 패턴 적용**
8. **설계의 이점**
9. **성능 및 메모리 최적화**
10. **테스트 리포트**
11. **향후 확장 가능성**
12. **결론**
13. **Github 프로젝트 리포트**

**1. 프로젝트 개요**

본 프로젝트는 객체지향 분석 및 설계(OOAD) 원칙을 바탕으로 한국의 전통 윷놀이 게임을 소프트웨어로 구현하는 것을 목표로 합니다. 사용자 정의 기능과 실제 윷놀이 규칙을 충실히 반영하여, 직관적인 인터페이스와 체계적인 게임 흐름을 제공하는 데 중점을 두고 있습니다.

게임은 한 대의 컴퓨터에서 2명에서 4명까지 참여할 수 있으며, 모든 플레이어는 개인전으로 진행하게 됩니다. 각 플레이어는 2개에서 5개까지 게임 말을 설정할 수 있고, 서로 다른 종류의 말을 사용합니다. 플레이어들은 같은 기기를 사용해 번갈아 가며 게임을 진행하기 때문에 별도의 네트워크 통신 기능은 필요하지 않습니다.

여러 말이 동시에 이동할 수 있거나 전략적 판단이 필요한 상황에서는 사용자에게 선택지를 제공해야 합니다. 또한 윷놀이의 고유한 규칙인 말 업기 기능과 다른 플레이어의 말을 잡는 기능도 지원됩니다. 업기 기능은 같은 팀의 말이 함께 이동할 수 있도록 하며, 상대의 말을 잡을 경우 해당 말은 시작 지점으로 되돌아갑니다.

게임은 모든 말을 먼저 완주한 플레이어가 승리하게 되며, 게임 종료 시 승리자를 화면에 명확히 표시해야 합니다. 이후 사용자에게 게임을 재시작할지, 종료할지를 선택할 수 있는 기능도 제공해야 합니다.

이 프로젝트는 OOAD 원칙에 따라 역할 분리와 책임 분산을 고려한 구조로 설계되며, GUI는 Java Swing 혹은 JavaFX 등을 활용해 구현해야 합니다. 게임의 흐름 제어, 사용자 인터랙션 처리, 상태 관리 등의 핵심 기능을 포함하여 전체 시스템의 완성도를 높이는 것이 목표입니다.

**2. 유스케이스 모델**

* 1. **기능적 요구사항**

FR1. 사용자는 게임을 시작할 수 있다

FR2. 사용자는 윷놀이 판 형태를 선택할 수 있다(정사각형, 오각형 등)

FR3. 사용자는 게임 시작 전 플레이어 수(2-4)와 말 개수(2-5)를 설정할 수 있다

FR4. 사용자는 랜덤 혹은 지정 방식으로 윷을 던져 결과를 생성할 수 있다

FR5. 사용자는 윷 결과가 특정되면 선택한 말을 움직인다

FR6. 윷이나 모가 나오면 윷을 한번 더 던진다

FR7. 동일 칸에 같은 팀 말이 존재할 경우 말 업기를 선택할 수 있다

FR8. 동일 칸에 상대 팀 말이 존재할 경우 말 잡기가 발생하고 윷을 한번 더 던진다

FR9. 말이 골인지점에 도달하면 골인처리되고 사라진다

FR10. 모든 말이 골인처리되면 해당 사용자는 승리한다

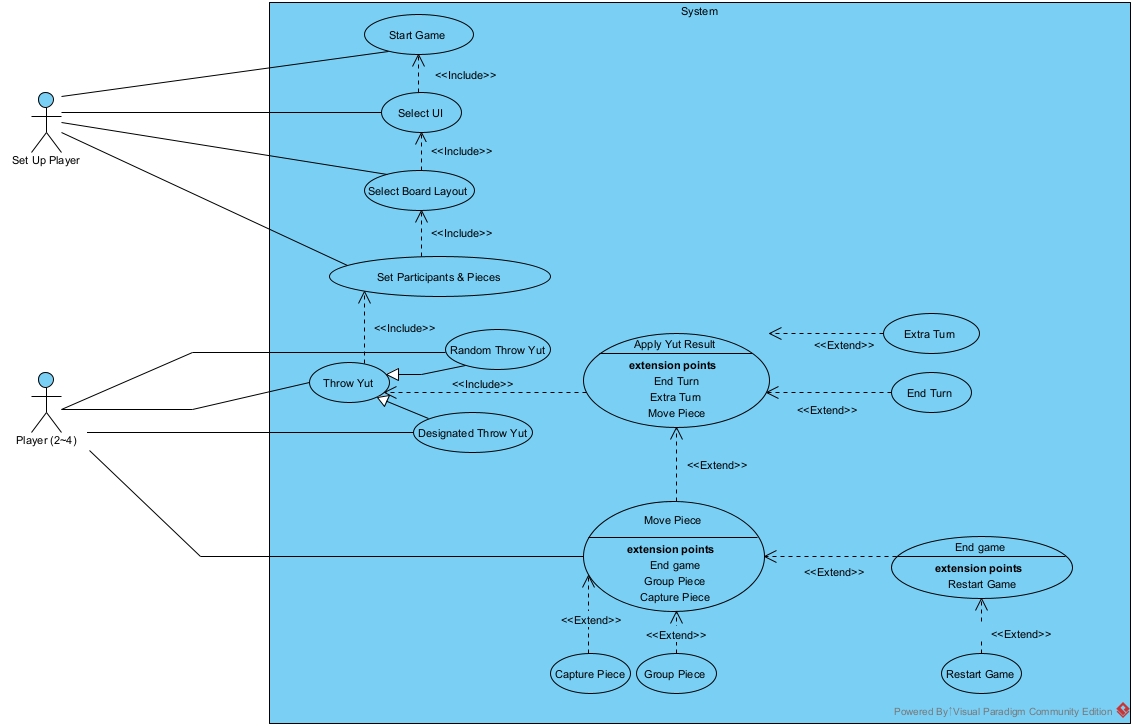
* 1. **비기능적 요구사항**

NFR1. MVC 아키텍처 패턴을 사용해 UI와 모델을 분리 구현해야 한다

NFR2. 2개 이상의 UI를 구현해야 한다. 이때 UI를 제외한 나머지 코드들이 거의 수정없이 재사용 되어야 한다.

NFR3. 테스트 용이한 설계로 Junit 모델 테스트를 수행해야 한다.

* 1. **유스케이스**



**UC1. 게임 시작하기**

Main Flow : 윷놀이 게임이 시작된다

**UC2. UI 선택하기**

Precondition : 사용자가 게임을 실행한 상태이다

Main Flow : UI 선택창에서 Swing과 JavaFX중 하나를 선택한다

**UC3. 윷 놀이 판 형태 선택하기**

Precondition : 사용자가 UI를 선택한 상태이다

Main Flow : 윷 놀이 판을 사각형, 오각형, 육각형 중 하나를 선택한다

**UC4. 참가자 수와 말의 개수 선택하기**

Precondition : 사용자가 윷 놀이 판 형태를 선택한 상태이다

Main Flow : 참가자 수를 먼저 숫자로 입력하고 말의 개수를 입력한다

Alternative Flows :

만약 입력 값이 비정상적이면 오류를 출력한다

**UC5. 윷 던지기**

Precondition : 플레이어 수와 말 개수가 입력되었다

Main Flow : 랜덤으로 (-1~5) 혹은 지정하여 윷 결과를 생성한다.

Alternative Flows : 만약 윷 던저기를 취소한다면 차례를 상대방에게 넘어간다

빽도는 음수처리이다

**UC6. 윷 결과 적용하기**

Precondition : 윷 던지기로 결과 생성 완료

Main Flow : 윷 던지기 결과를 적용할 말을 선택한다. 만약 추가 턴이 발생하면 이 상태가 유지된다. 그렇지 않다면 턴이 종료되고 상대방에게 차례가 넘어간다

Alternative Flows : 만약 윷 던지기 결과 적용을 취소하면 차례가 상대방에게 넘어간다

**UC7. 말 이동하기**

Precondition : 윷 던지기 결과를 적용할 말을 선택하였다

Main Flow : 윷 던지기 결과를 적용할 말을 선택한다. 선택한 말을 지정한 만큼 이동한다.

Alternative Flows : 출발하지 않은 말은 빽도(-1)를 적용할 수 없다

**UC8. 말 업기**

Precondition : UC7에서 동일한 칸에 아군 말이 2개 이상 존재한다

Main Flow : 말 업기를 할지 물어보고 수락하면 말 업기가 되어 함께 이동한다. 말 업기를 하지 않으면 계속 따로 이동한다.

Alternative Flows : 다른 팀이라면 말 잡기 기능이 호출된다

**UC9. 말 잡기**

Precondition : UC7에서 이동한 말의 동일한 칸에 적군 말이 1개 이상 존재한다

Main Flow : 적군 말을 윷 놀이 판에서 제거하여 출발 전 상태로 만든다.

Alternative Flows : 만약 적군이 말 업기 상태였다면 해당 말 전부를 말 잡기처리한다.

**UC10. 턴 종료하기**

Precondition : 윷, 모, 말 잡기 같은 추가 턴 발생 이벤트가 발생하지 않고 말 이동이 끝났다.

Main Flow : 나의 차례가 종료되고 상대방에게 차례가 넘어간다. 차례를 넘겨받은 상대방은 윷 던지기부터 시작한다.

Alternative Flows : 윷, 모가 나오거나 말 잡기가 발생하면 추가 턴이 발생한다

**UC11. 추가 턴 진행하기**

Precondition : 윷, 모가 나오거나 말 잡기 같은 추가 턴 발생 이벤트가 발생했다.

Main Flow : UC5부터 다시 시작할 수 있다.

Alternative Flows : 추가 턴 이벤트가 연속해서 발생하면 연속해서 추가 턴이 생긴다.

**UC12. 게임 종료하기**

Precondition : 플레이어 중 한명의 모든 말이 골인하였다

Main Flow : 승자가 표시되며 게임이 종료된다. 다시하고 싶다면 게임을 리셋할 수 있다

**UC13. 게임 리셋하기**

Precondition : 게임이 종료되고 게임 리셋을 선택하였다

Main Flow : 게임이 처음부터 다시 시작된다

**처음부터 흐름 이해해보기**

사용자는 프로그램을 실행하여 게임을 시작한다.(Start Game) 그러면 UI 선택창이 뜨게 되고(Select UI), Swing과 JavaFX중 하나를 선택한다. Board Layout을 선택하여 사각, 오각, 육각형 판 중 플레이할 판을 선택한다(Select Board Layout). 사용자는 플레이어 수와 말 수를 선택한다. 여기까지의 과정은 필수적으로 1회 진행되어야 이후 게임 플레이(윷 던지기)가 가능하기에 include 관계를 통해 해당 과정이 선행되어야 함을 표현했다. 또한 모든 플레이어가 해당 설정을 하는 것이 아니라 Set Up Player 한 사람만이 이를 설정하기에 Actor를 구분하여 표기했다. 사용자(Player 2~4)는 윷을 랜덤 윷 던지기, 지정 윷 던지기 중 하나를 선택하여 윷을 던진다. 둘 다 윷을 던진다는 개념은 동일하기에 Throw Yut 으로 일반화한다. 윷 결과가 특정되면 해당 윷을 말에 적용한다 윷을 던져야 말에 적용할 수 있기에 Throw Yut과 Apply Yut Result는 include 관계로 설정했다. 윷이나 모가 나오면 플레이어가 윷을 한번 더 던지기에 extend로 Extra Turn을 제공한다. 혹은 아무 말도 출발하지 않은 상태에서 빽도가 나온다면 적용할 수 있는 말이 없기에 End Turn하게 된다(extend). 아니면 윷을 적용하여 말을 이동하며 게임은 진행되고(extend), 게임 상황에 따라 선택적으로 말 업기(Group Piece), 말 잡기(Capture Piece), 말 골인(End Piece) 기능이 수행되고 모든 말이 End하게 되면 그 플레이어는 게임에서 승리한다.

**3. 기타 요구사항**

**3.1 Operation Contract**

**1. startGame()**

유스케이스: Start Game

사전조건: 프로그램이 실행됨

사후조건: UI 선택 화면으로 진입

주요 흐름:

사용자가 시스템 실행 후 “게임 시작” 선택

시스템이 초기화 루틴을 실행하고 다음 화면으로 이동

**2. selectUI(mode)**

유스케이스: Select UI

사전조건: 게임이 시작된 상태

사후조건: 선택한 UI(Swing/JavaFX)가 로드됨

주요 흐름:

시스템이 UI 목록을 제시

사용자가 원하는 모드를 선택

시스템이 해당 UI 모듈을 로딩

**3. selectBoardLayout(layoutType)**

유스케이스: Select Board Layout

사전조건: UI가 설정됨

사후조건: 선택된 보드 레이아웃이 게임에 적용됨

주요 흐름:

시스템이 보드 레이아웃 목록을 보여줌

사용자가 하나를 선택

시스템이 선택값을 저장하고 해당 보드를 준비함

**4. setupParticipants(playerNum, pieceNum)**

유스케이스: Set Participants & Pieces

사전조건: 보드가 설정됨

사후조건: 참가자 수, 이름, 색상, 말 개수 등이 등록됨

주요 흐름:

시스템이 인원수 및 이름 입력을 요구

사용자가 정보 입력

시스템이 각 플레이어와 말(Token) 초기화

예외 흐름:

사용자가 적절한 값[참가자(2~4 사이의 값), 말(2~5 사이의 값)]을 선택하지 않으면 다시 입력받음

**5. throwYut(player)**

유스케이스: Throw Yut

사전조건: 현재 플레이어의 턴이며, 게임이 종료되지 않음

사후조건: 도/개/걸/윷/모 중 하나의 결과 생성

주요 흐름:

플레이어가 윷 던지기 요청

시스템이 무작위 결과를 생성

해당 결과가 사용자에게 표시되고 저장됨

**6. movePiece(yutResult, piece)**

유스케이스: Move Piece

사전조건: 플레이어가 윷 결과를 얻고 말을 선택함

사후조건: 말이 이동하고 다음 상태(잡기, 묶기, 종료 등)가 처리됨

주요 흐름:

시스템이 선택된 말과 윷 값을 바탕으로 이동 경로 계산

이동 중 도착지에 따라 잡기/묶기/종료 등 처리

UI에 반영됨

예외 흐름:

아무 말도 출발하지 않았을 때 빽도가 나오면 piece를 move하지 않고 종료처리.

**7. checkExtraTurn(result)**

유스케이스: Extra Turn

사전조건: 윷 결과가 존재함

사후조건: 윷 또는 모인 경우 extraTurn 값 증가

주요 흐름:

시스템이 결과를 확인

윷 또는 모일 경우 extraTurn += 1

그 외는 다음 턴으로 넘어갈 준비

**8. checkGameEnd()**

유스케이스: End Game

사전조건: 말 이동 이후

사후조건: 모든 말이 도착하면 게임 종료됨

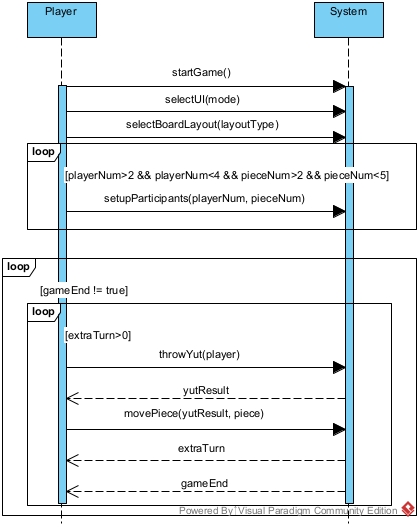
주요 흐름:

시스템이 현재 플레이어의 말 상태를 모두 확인

모든 말이 도착 상태라면 승리 처리

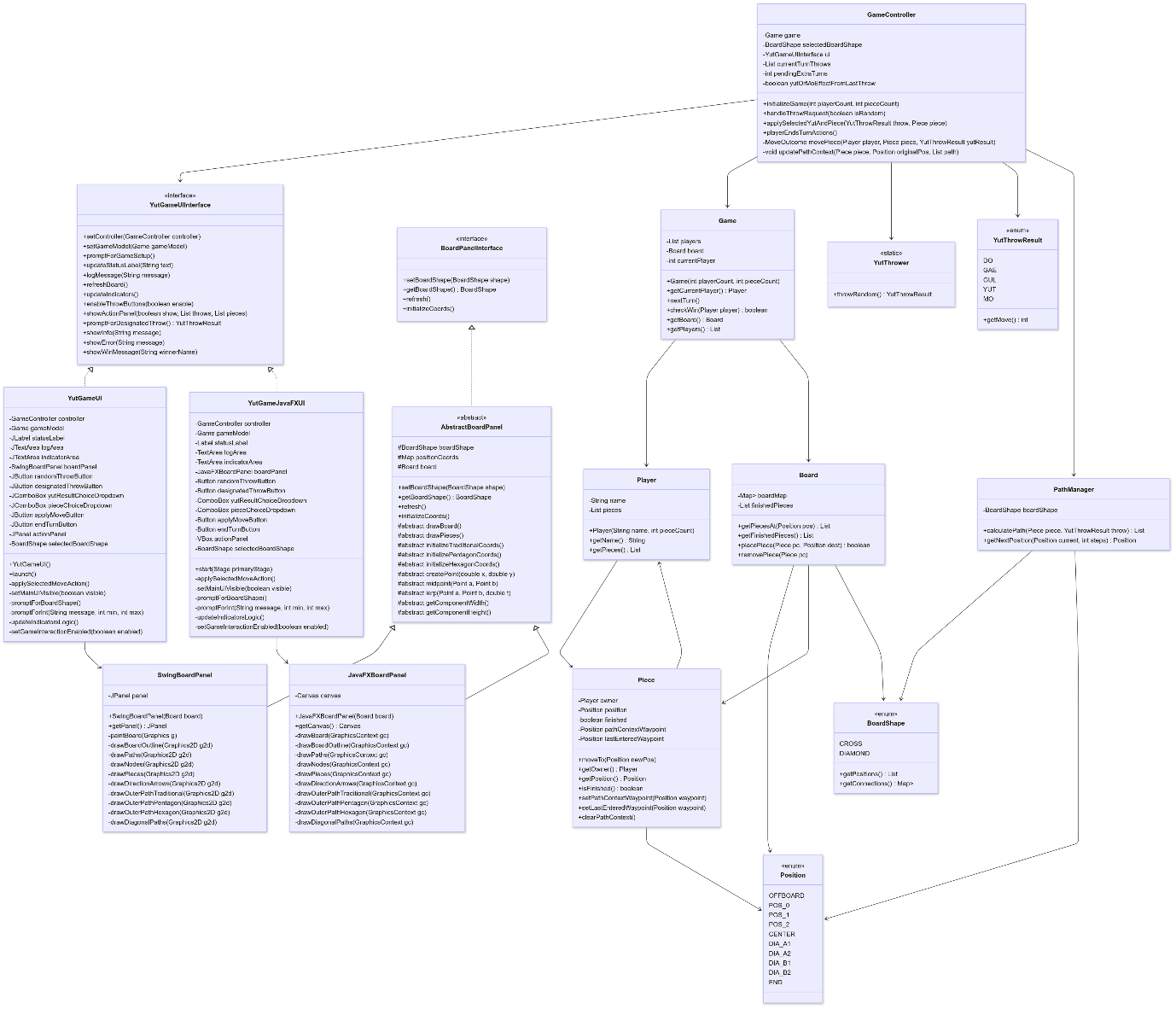
종료 화면 출력

**3.2 System Sequence Diagram**

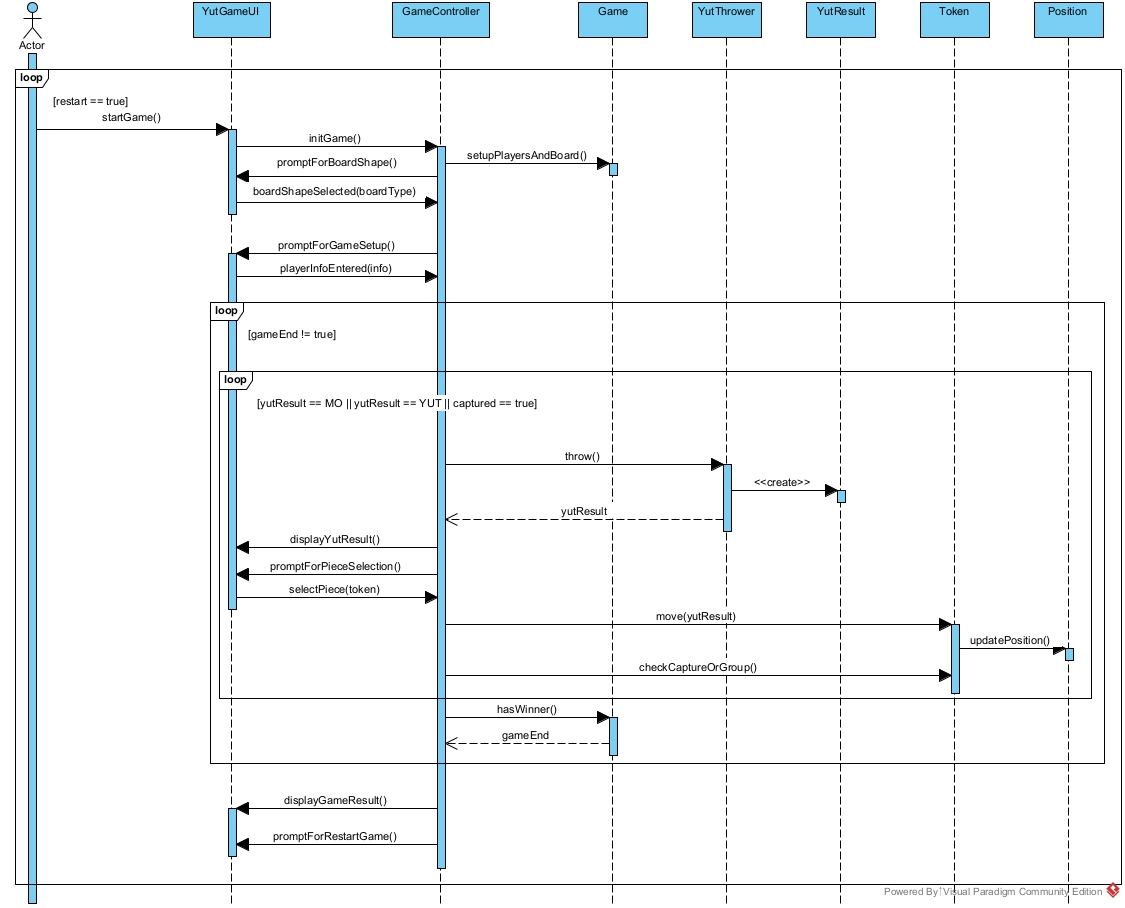


**4. 상세 설계**

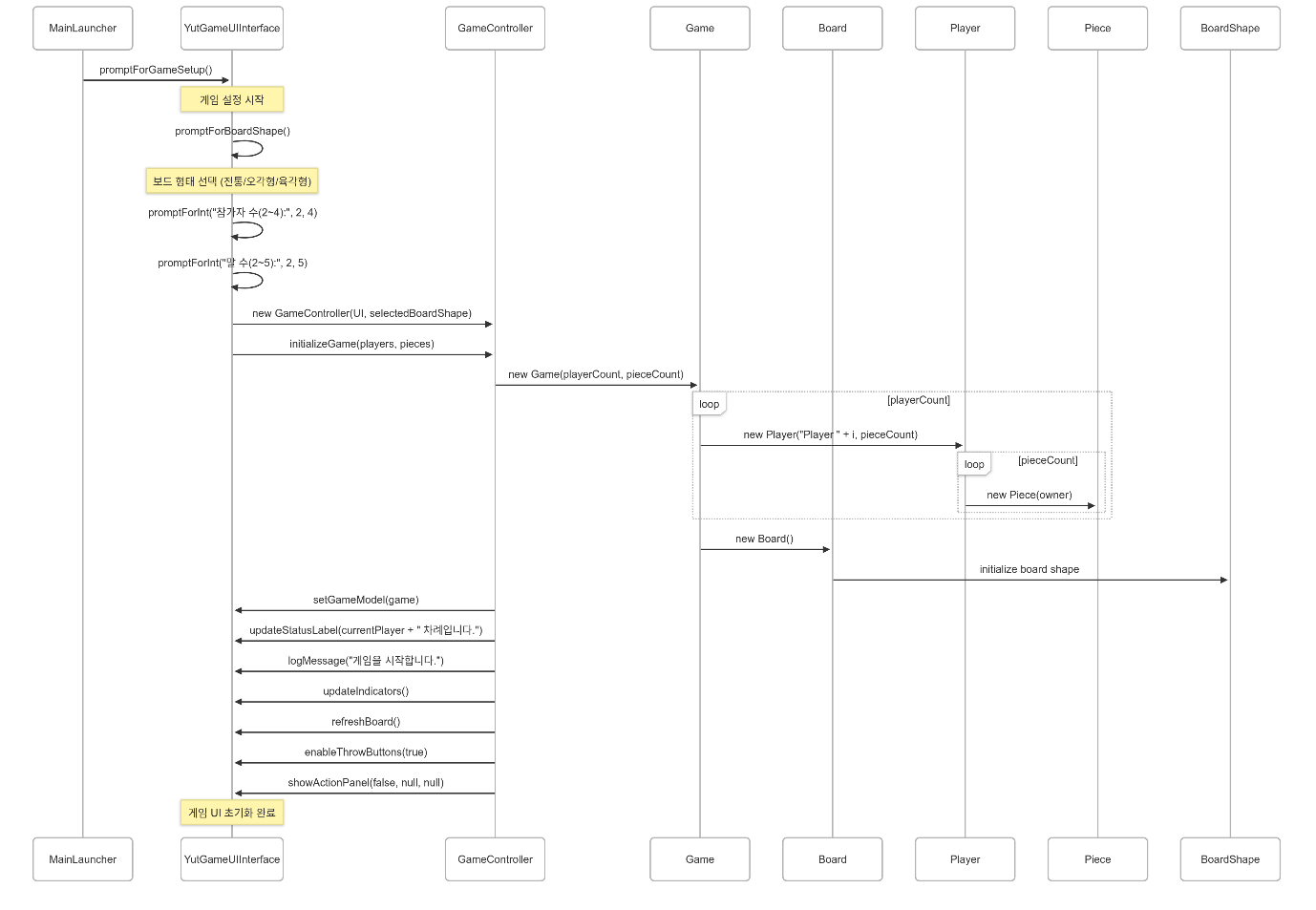
**4.1 Class Diagram**



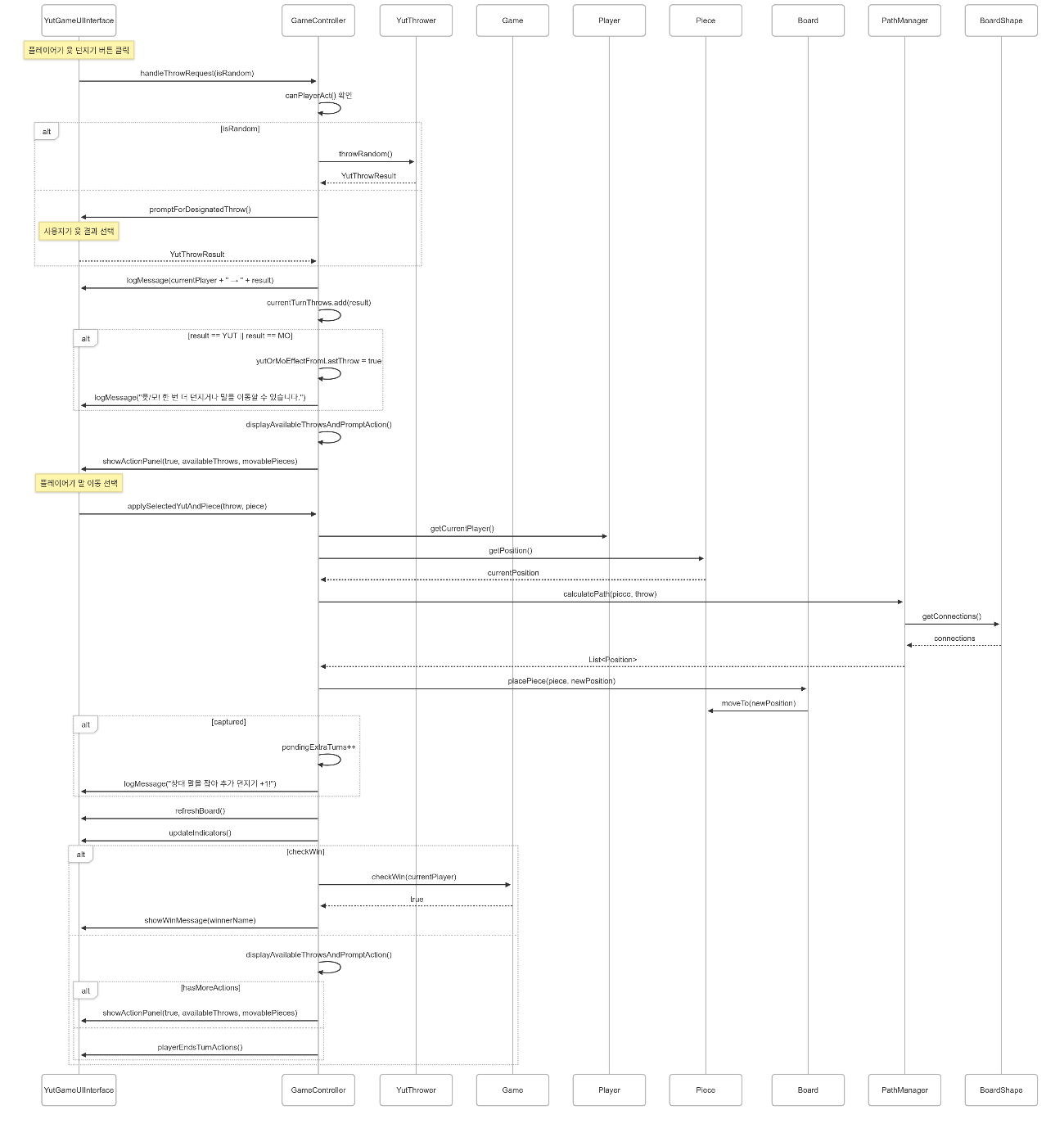
**4.2 Sequence Diagram – 전체 게임 진행**

****

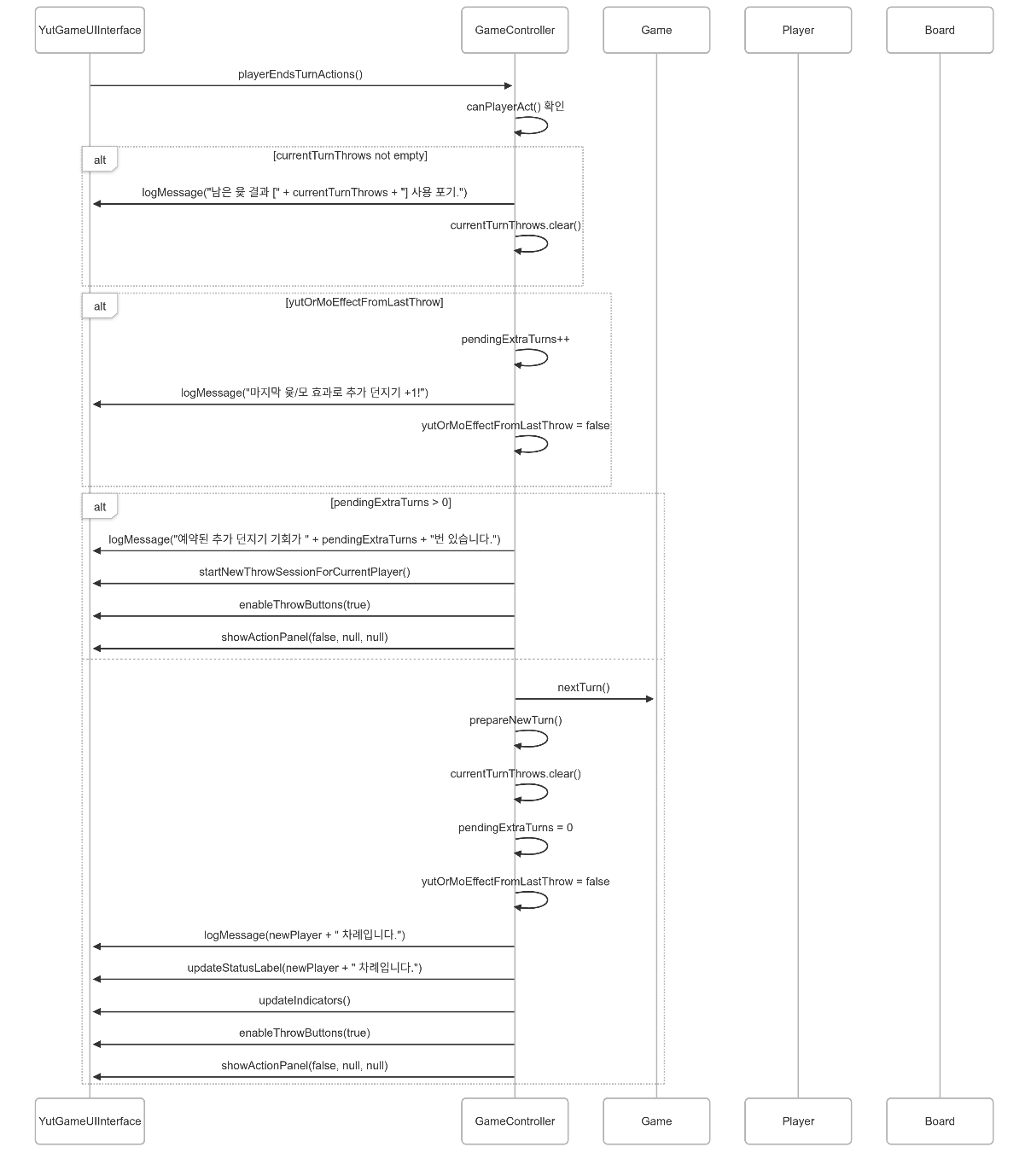
**4.3 Sequence Diagram - 게임 초기화**



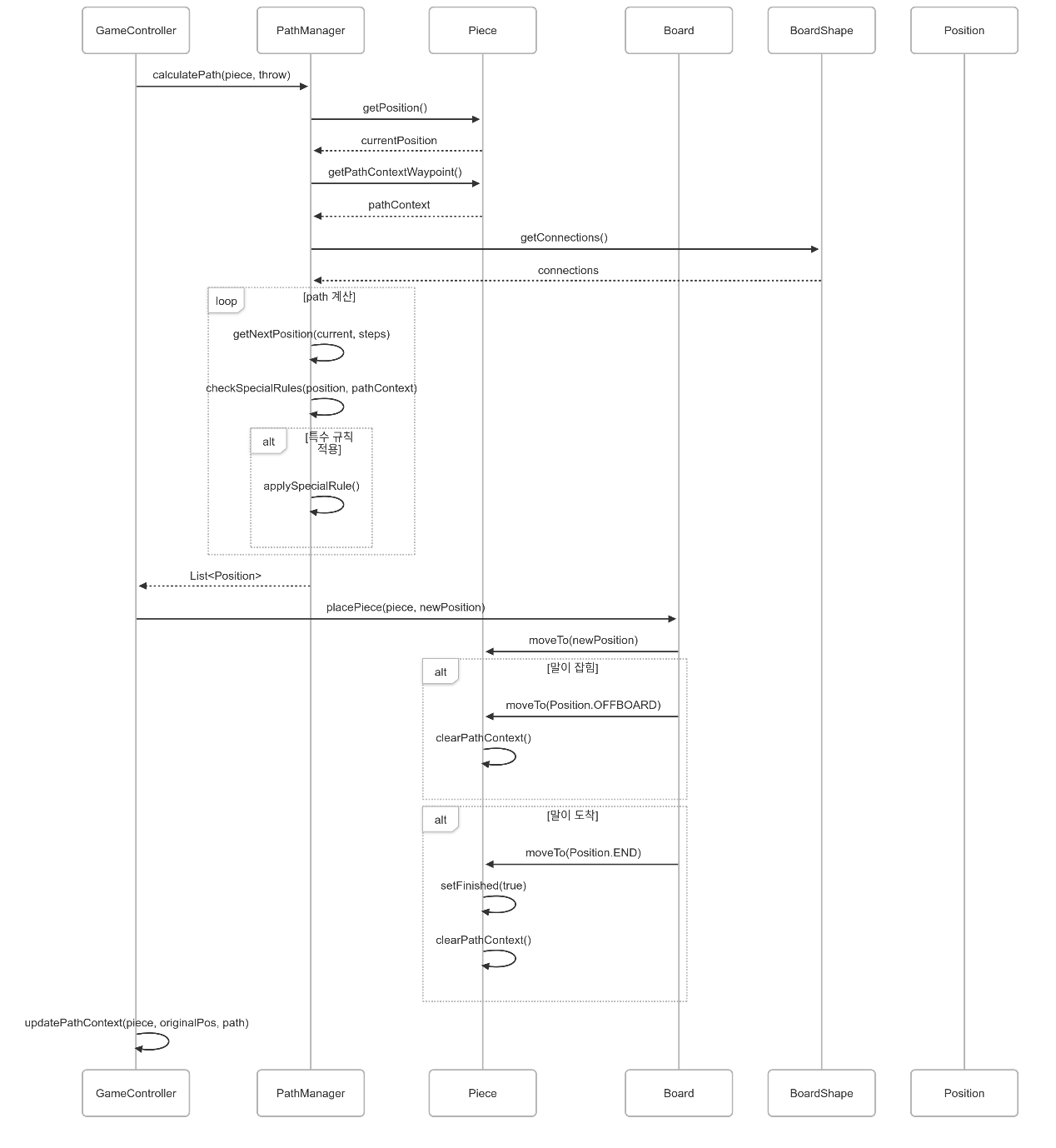
**4.4 Sequence Diagram - 윷 던지기 및 말 이동**



**4.5 Sequence Diagram - 턴 종료 및 다음 턴 시작**



**4.6 Sequence Diagram - 말 이동 및 경로 계산**



**5. UI 교체설계 핵심**

**5.1 시스템 구조**

본 윷놀이 게임은 계층형 아키텍처를 기반으로 설계되어 UI 프레임워크 변경에 대한 유연성을 극대화했습니다.

[Presentation Layer - UI]

├── YutGameUIInterface (인터페이스)

│ ├── YutGameUI (Swing 구현체)

│ └── YutGameJavaFXUI (JavaFX 구현체)

├── BoardPanelInterface (인터페이스)

│ ├── AbstractBoardPanel (추상 클래스)

│ │ ├── SwingBoardPanel (Swing 구현체)

│ │ └── JavaFXBoardPanel (JavaFX 구현체)

└── MainLauncher (UI 실행 관리)

[Business Logic Layer - 게임 로직]

├── GameController (컨트롤러)

├── Game (게임 상태 관리)

├── YutThrower (윷 던지기 로직)

└── YutThrowResult (윷 결과)

[Domain Layer - 도메인 모델]

├── Board (게임판)

├── BoardShape (게임판 형태)

├── Player (플레이어)

├── Piece (말)

└── Position (위치)

[Infrastructure Layer - 기반 시설]

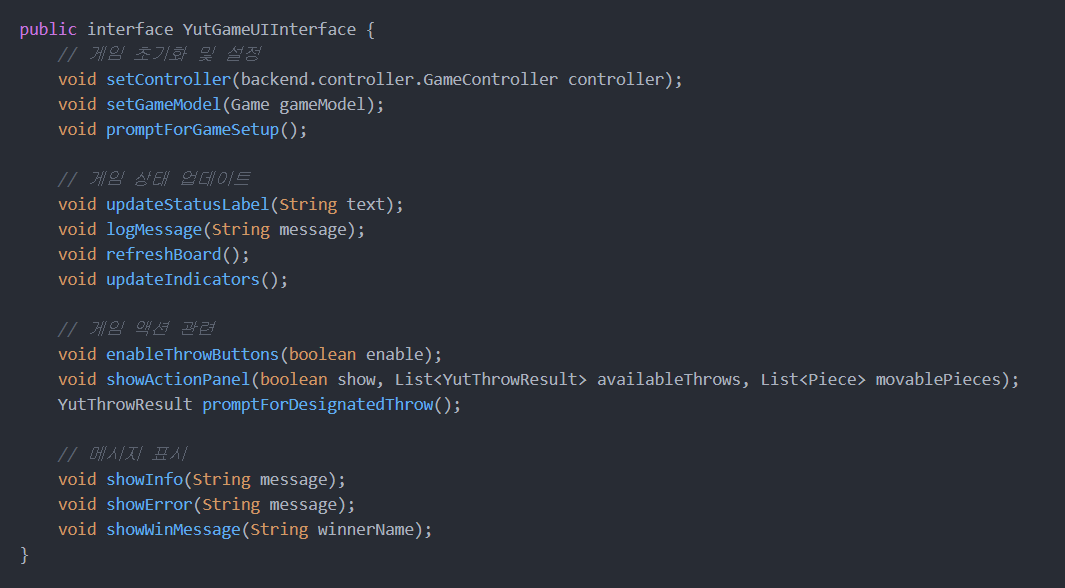
└── PathManager (경로 관리)

**5.2 핵심 설계 원칙**

* **관심사 분리 (Separation of Concerns)**
  + UI 로직과 게임 로직의 완전한 분리
  + 각 계층의 독립적인 책임
  + 프레임워크별 UI 구현체 분
* **의존성 역전 원칙 (Dependency Inversion Principle)**
  + UI 컴포넌트들이 인터페이스에 의존
  + YutGameUIInterface 와 BoardPanelInterface를 통한 추상화
  + 구체적인 UI 구현체와의 결합도 최소화
* **인터페이스 분리 원칙 (Interface Segregation Principle)**
  + 각 인터페이스가 단일 책임을 가짐
* **개방-폐쇄 원칙 (Open-Closed Principle)**
  + 새로운 UI 프레임워크 추가 시 기존 코드 수정 불필요
  + AbstractBoardPanel을 통한 공통 기능 확장
  + 인터페이스 구현을 통한 새로운 기능 추가
* **단일 책임 원칙 (Single Responsibility Principle)**
  + PathManager: 경로 계산만 담당
  + YutThrower: 윷 던지기 로직만 담당
  + GameController: 게임 흐름 제어만 담당

**5.3 인터페이스 기반 추상화**

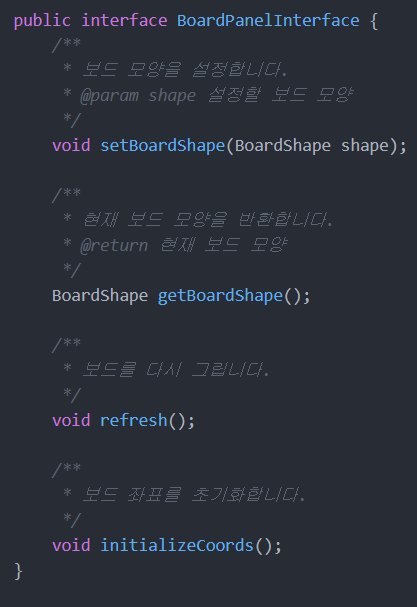
YutGameUIInterface는 UI 프레임워크에 독립적인 계약을 정의합니다.



* UI 관련 모든 기능을 인터페이스로 정의
* 구체적인 UI 구현체와의 결합도 최소화
* 새로운 UI 프레임워크 추가 시 인터페이스만 구현하면 됨

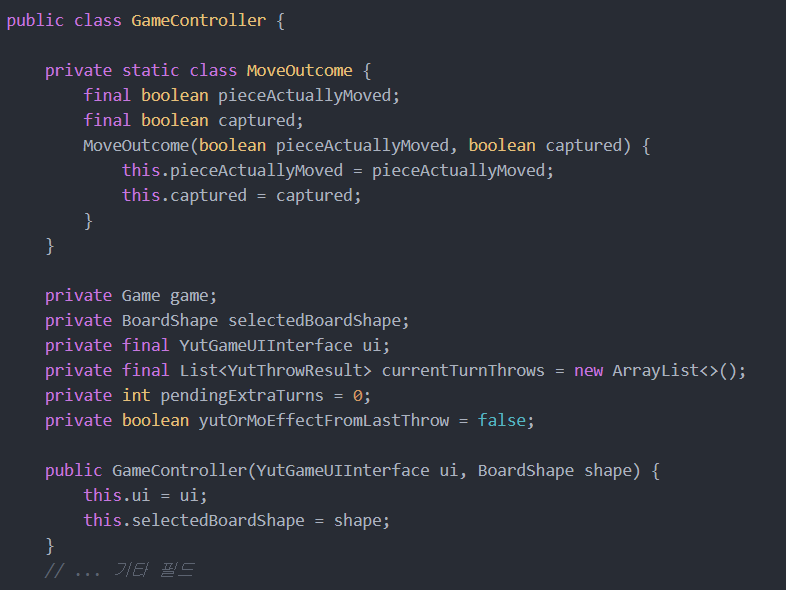
**5.4 보드 패널 추상화**

**BoardPanel** 추상 클래스는 UI 프레임워크별 구현체들의 공통 기능을 제공합니다.



* 게임 보드 렌더링 로직을 추상화
* 프레임워크별 구현체 분리 (Swing/JavaFX)
* 공통 기능은 추상 클래스에서 구현

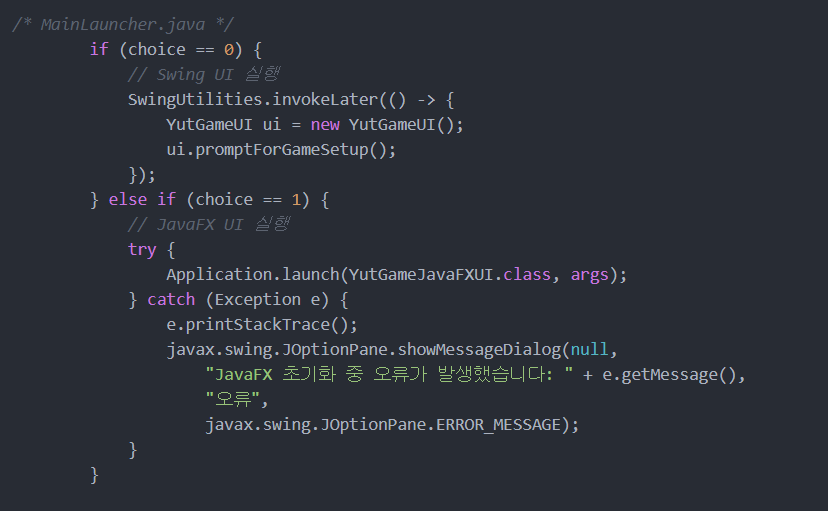
**5.5 컨트롤러 분리**



* UI 로직과 게임 로직의 완전한 분리
* 컨트롤러가 UI 인터페이스에만 의존

**6. UI 교체 시나리오 분석**

**6.1 Swing에서 JavaFX로 전환**



* 런타임에 UI 선택 가능
* 기존 코드 수정 없이 새로운 UI 추가

**6.2 새로운 UI 프레임워크 추가 시**

* YutGameUIInterface 구현
* BoardPanelInterface 구현
* AbstractBoardPanel 상속
* 기존 코드 수정 불필요

**7. 설계 패턴 적용**

**7.1 MVC (Model-View-Controller) 패턴**

* **Model**: 게임 도메인 모델 (Game, Board, Player, Piece)
* **View**: UI 구현체 (YutGameUI, YutGameJavaFXUI, ... )
* **Controller**: 게임 컨트롤러 (GameController)

**7.2 전략 패턴 (Strategy Pattern)**

* 다양한 보드 렌더링 전략 구현
* 런타임에 전략 교체 가능

**7.3 템플릿 메서드 패턴 (Template Method Pattern)**

* 공통 알고리즘 구조 정의
* 하위 클래스에서 구체적인 구현

**7.4 의존성 주입 패턴 (Dependency Injection)**

* 의존성을 외부에서 주입
* 결합도 감소 및 테스트 용이성 향상

**8. 설계의 이점**

**8.1 유지보수성 향상**

* UI 로직과 게임 로직의 명확한 분리
* 각 컴포넌트의 단일 책임 원칙 준수
* 코드 재사용성 향상

**8.2 확장성 극대화**

* 새로운 UI 프레임워크 추가 용이
* 기존 코드 수정 최소화
* 인터페이스 기반 설계로 유연성 확보

**8.3 테스트성 증대**

* UI와 게임 로직의 독립적 테스트 가능
* 목(Mock) 객체를 통한 테스트 용이
* 단위 테스트 작성 용이

**9. 성능 및 메모리 최적화**

**9.1 메모리 관리**

* 효율적인 데이터 구조 사용
* 불필요한 객체 생성 최소화

**9.2 렌더링 최적화**

* 더블 버퍼링을 통한 화면 깜빡임 방지
* 부분 갱신을 통한 성능 향상

**10. 테스트 리포트(Junit 테스트)**

**1.GameControllerTest** 클래스

**1.1 testGameInitialization()**

목적: 게임 컨트롤러의 초기화가 올바르게 수행되는지 검증

**테스트 내용:**

게임 객체가 null이 아닌지 확인

플레이어 수가 설정된 값(2명)과 일치하는지 확인

각 플레이어의 말 개수가 설정된 값(4개)과 일치하는지 확인

**테스트 결과:** 통과 모든 초기화 값이 정확히 설정됨

**1.2 testHandleThrowRequest()**

**목적:** 윷 던지기 요청 처리 기능이 정상 작동하는지 확인

**테스트 내용**:

랜덤 윷 던지기 실행

던진 결과가 사용 가능한 윷 목록에 추가되는지 확인

**테스트 결과:** 통과 - 윷 던지기 후 사용 가능한 윷 결과 목록이 정상적으로 생성됨

**1.3 testApplySelectedYutAndPiece()**

**목적**: 선택된 윷과 말을 이용한 이동 기능이 정상 작동하는지 확인

**테스트** 내용:

현재 플레이어의 첫 번째 말 선택

윷 던지기 후 첫 번째 결과 사용

말 이동 실행

말의 위치가 OFFBOARD에서 변경되었는지 확인

**테스트 결과:** 통과 - 말이 초기 위치(OFFBOARD)에서 정상적으로 이동됨

2. **GameTest 클래스**

**2.1 testGameInitialization()**

**목적**: 게임 객체의 기본 초기화가 올바르게 수행되는지 검증

**테스트 내용:**

플레이어 목록이 null이 아닌지 확인

플레이어 수가 2명인지 확인

플레이어 이름이 "Player 1", "Player 2"로 설정되었는지 확인

각 플레이어가 4개의 말을 보유하는지 확인

모든 말이 OFFBOARD 위치에서 시작하는지 확인

모든 말이 완주(finished) 상태가 아닌지 확인

**테스트 결과:** 통과 - 모든 초기 상태가 올바르게 설정됨

**2.2 testCurrentPlayer()**

**목적:** 현재 플레이어 반환 기능 검증

**테스트 내용:**

게임 시작 시 현재 플레이어가 "Player 1"인지 확인

**테스트 결과:** 통과 - 첫 번째 플레이어가 현재 플레이어로 정상 설정됨

**2.3 testNextTurn()**

**목적**: 턴 순환 기능이 정상 작동하는지 확인

**테스트 내용:**

초기 현재 플레이어가 "Player 1"인지 확인

nextTurn() 호출 후 "Player 2"로 변경되는지 확인

다시 nextTurn() 호출 후 "Player 1"로 돌아오는지 확인

**테스트 결과:** 통과 - 플레이어 턴이 순환적으로 정상 변경됨

**2.4 testCheckWin()**

**목적**: 승리 조건 확인 기능이 정상 작동하는지 검증

**테스트 내용**:

초기 상태에서 승리하지 않았음을 확인

모든 말을 END 위치로 이동

승리 조건이 충족되었는지 확인

**테스트 결과**: 통과 - 모든 말이 도착했을 때만 승리로 정확히 판정됨

**2.5 testBoardInitialization()**

**목적**: 게임 보드가 올바르게 초기화되는지 확인

**테스트 내용**:

게임 보드 객체가 null이 아닌지 확인

**테스트 결과**: 통과 보드 객체가 정상적으로 생성됨

**3. YutThrowerTest 클래스**

**3.1 testThrowSpecified()**

**목적**: 지정된 윷 결과 반환 기능이 정확히 작동하는지 확인

**테스트 내용:**

모든 윷 결과(BACKDO, DO, GAE, GEOL, YUT, MO)에 대해

throwSpecified()메서드가 입력값과 동일한 결과를 반환하는지 확인

**테스트 결과:** 통과 - 입력된 윷 결과가 그대로 정확히 반환됨

**3.2 testThrowRandom**(**)**

**목적**: 랜덤 윷 던지기가 모든 가능한 결과를 생성할 수 있는지 확인

**테스트 내용:**

1000번의 랜덤 던지기 실행

6가지 윷 결과가 모두 최소 한 번은 나오는지 확인

**테스트 결과:** 통과 - 충분한 시행 후 모든 윷 결과가 정상적으로 출현함

**3.3 testThrowRandomDistribution()**

**목적**: 랜덤 윷 던지기의 확률 분포가 설계된 대로 작동하는지 검증

**테스트 내용:**

10,000번의 랜덤 던지기 실행

각 윷 결과의 출현 빈도가 설정된 확률 범위 내에 있는지 확인

BACKDO: 5% (3-7% 범위)

DO: 25% (20-30% 범위)

GAE: 25% (20-30% 범위)

GEOL: 20% (15-25% 범위)

YUT: 15% (10-20% 범위)

MO: 10% (5-15% 범위)

**테스트 결과:** 통과 - 각 윷 결과의 출현 확률이 설계된 범위 내에

서 정상 분포됨

4. **YutThrowResultTest 클래스**

**4.1 testMoveValues(**)

**목적**: 각 윷 결과의 이동 값이 올바르게 설정되었는지 확인

**테스트 내용:**

각 윷 결과의 getMove() 반환값 확인

BACKDO: -1, DO: 1, GAE: 2, GEOL: 3, YUT: 4, MO: 5

**테스트 결과:** 통과 모든 윷 결과가 정의된 이동 값을 정확히 반환함

**4.2 testFromString()**

목적: 문자열로부터 윷 결과 변환 기능이 정상 작동하는지 확인

**테스트 내용:**

소문자 문자열("backdo", "do", "gae", "geol", "yut", "mo")을 해당하는 윷 결과 열거형으로 변환하는지 확인

**테스트 결과:** 통과 - 문자열이 정확한 윷 결과로 완벽하게 변환됨

**4.3 testFromStringCaselnsensitive()**

**목적**: 문자열 변환이 대소문자를 구분하지 않는지 확인

**테스트 내용:**

대문자 문자열("BACKDO", "DO", "GAE")이 올바른 윷 결과로 변환되는지 확인

**테스트 결과:** 통과 대소문자 구분 없이 정확한 변환이 성공적으

로 수행

**4.4 testFromStringInvalidInput**(**)**

**목적:** 잘못된 입력에 대한 예외 처리가 정상 작동하는지 확인

**테스트 내용**:

유효하지 않은 문자열("invalid") 입력 시 IllegalArgumentException이 발생하는지 확인

**테스트** 결과: 통과 - 잘못된 입력에 대해 적절한 예외가 정상 발생함

**5. PathManagerTest 클래스**

**5.1 testGetNextPositionsFromStart**()

**목적**: POS\_0에서 전진할 때 특수 규칙이 올바르게 적용되는지 검증

**테스트 내용**:

POS\_0에서 1칸 전진 시 바로 END로 이동하는 특수 규칙 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 Position.END인지 확인

**테스트 결과**: 통과 - 특수 규칙이 정확히 적용됨

**5.2 testGetNextPositionsFromStartBackward**()

**목적**: POS\_0에서 후진할 때 순환 구조가 올바르게 작동하는지 검증

**테스트 내용**:

POS\_0에서 -1칸 이동 시 POS\_19로 이동하는 순환 구조 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 Position.POS\_19인지 확인

**테스트 결과:** 통과 - 순환 구조가 정확히 구현됨

**5.3 testGetNextPositionsFromOffboard()**

**목적**: OFFBOARD에서 시작하는 말의 이동 경로가 올바른지 검증

**테스트 내용:**

OFFBOARD에서 1칸 이동 시 외곽 경로의 첫 번째 위치로 이동

확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 null이 아닌지 확인

**테스트 결과:** 통과 - 게임 시작 시 말의 진입이 정상적으로 처리됨

**5.4 testGetNextPositionsWith Diagonal()**

**목적**: 대각선 입구에서 대각선 경로로 진입하는지 검증

**테스트 내용:**

POS\_5(대각선 A 입구)에서 1칸 이동 시 대각선 경로 진입 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 대각선 경로 또는 CENTER인지 확인

**테스트 결과**: 통과 - 대각선 지름길 진입이 정확히 처리됨

**5.5 testGetNextPositionsTo Center(**)

**목적:** 대각선 경로에서 CENTER로 이동하는 로직 검증

**테스트 내용:**

DIA\_A2에서 1칸 이동 시 다음 위치 계산 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 null이 아닌지 확인

**테스트 결과:** 통과 - 대각선 경로 내 이동이 정상적으로 처리됨

**5.6 testGetNextPositionsFromCenter()**

목적: CENTER에서 기본 출구 경로를 사용한 이동 검증

**테스트 내용:**

CENTER에서 1칸 이동 시 기본 출구 경로 사용 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 null이 아닌지 확인

**테스트 결과:** 통과 - CENTER에서의 기본 출구 처리가 정확함

**5.7 testGetNextPositionsFromCenterWithContext()**

**목적**: CENTER에서 컨텍스트 정보를 활용한 이동 검증

**테스트 내용:**

CENTER에서 경로 컨텍스트(DIA\_A2) 설정 후 1칸 이동

반환된 경로의 크기가 1인지 확인

반환된 위치가 null이 아닌지 확인

**테스트 결과**: 통과 - 컨텍스트 기반 경로 선택이 정확히 작동함

**5.8 testGetNextPositionsToEnd()**

**목적:** 특정 위치에서 END 지점 도달 로직 검증

**테스트 내용:**

POS\_19에서 2칸 이동 시 END 도달 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 Position.END인지 확인

**테스트 결과:** 통과 - 게임 완주 처리가 정확히 구현됨

**5.9 testGetNextPositionsCircular(**)

**목적:** 외곽 경로의 순환 이동 로직 검증

**테스트 내용**:

POS\_19에서 1칸 이동 시 POS\_0으로 순환 이동 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 Position.POS\_0인지 확인

**테스트 결과**: 통과 - 외곽 경로 순환이 정확히 구현됨

**5.10 testGetNextPositionsBackward()**

**목적**: 일반 위치에서 후진 이동 로직 검증

**테스트 내용:**

POS\_5에서 -1칸 이동 시 POS\_4로 이동 확인

반환된 경로의 크기가 1인지 확인

반환된 위치가 Position.POS\_4인지 확인

테스트 **결과:** 통과 - 후진 이동이 정확히 처리됨

**5.11 testGetNextPositionsBackwardFromCenter**()

**목적**: CENTER에서 컨텍스트 기반 후진 이동 검증

**테스트 내용:**

CENTER에서 경로 컨텍스트 설정 후 -1칸 이동

반환된 경로의 크기가 1인지 확인

반환된 위치가 null이 아닌지 확인

**테스트 결과**: 통과 - CENTER에서의 후진 처리가 정확함

**5.12 testGetNextPositionsWithContext()**

**목적**: 대각선 경로에서 컨텍스트를 활용한 이동 검증

**테스트 내용:**

DIA\_A2에서 이전 위치 컨텍스트 설정 후 1칸 이동

반환된 경로의 크기가 1인지 확인

반환된 위치가 null이 아닌지 확인

**테스트 결과:** 통과 - 컨텍스트 기반 경로 선택이 정상 작동함

**5.13 testGetNextPositionsZeroSteps(**)

**목적**: 0칸 이동 시 예외 처리 검증

**테스트 내용:**

POS\_5에서 0칸 이동 시 빈 리스트 반환 확인

반환된 경로가 비어있는지 확인

**테스트 결과:** 통과 - 0칸 이동 예외 처리가 정확함

**5.14 testGetNextPositionsWithDifferentBoardShapes()**

**목적**: 다양한 보드 형태에서 이동 로직 호환성 검증

**테스트 내용:**

POS\_1에서 전통, 오각형, 육각형 보드에서 1칸 이동 확인

각 보드 형태별로 반환된 경로가 null이 아닌지 확인

각 보드 형태별로 반환된 경로의 크기가 1인지 확인

**테스트 결과:** 통과 모든 보드 형태에서 정상 작동함

**5.15 testGetNextPositionsMultipleStepsToEnd()**

**목적**: 여러 위치에서 충분한 거리로 END 도달 검증

**테스트 내용:**

POS\_18, POS\_17, POS\_16에서 5칸 이동 시 END 도달 확인

각 시작 위치별로 반환된 경로의 크기가 1인지 확인

반환된 위치가 Position.END인지 확인

**테스트 결과**: 통과 - 다양한 위치에서 END 도달이 정확히 계산됨

**5.16 testCircularMovement()**

**목적:** 연속적인 순환 이동 동작 검증

**테스트 내용:**

POS\_18에서 POS\_19로, POS\_19에서 POS\_0으로 연속 이동 확인

각 단계별 반환된 경로의 크기가 1인지 확인

각 단계별 반환된 위치가 예상 위치와 일치하는지 확인

**테스트 결과**: 통과 - 연속적인 순환 이동이 정확히 처리됨

**5.17 debugPathManager()**

**목적**: 다양한 위치에서의 이동 결과 종합 확인

**테스트 내용:**

OFFBOARD, POS\_0, POS\_1, POS\_5, POS\_10, CENTER, DIA\_A2에서 1칸 이동 결과 출력

각 위치별 이동 결과가 오류 없이 반환되는지 확인

**테스트 결과**: 통과 - 모든 테스트 위치에서 정상적인 결과 반환

**6. BoardShapeTest** 클래스

**6.1 testTraditionalBoardShape(**)

**목적:** 전통 보드 형태의 구조가 올바르게 정의되어 있는지 검증

**테스트 내용:**

외곽 경로가 21개 위치(0~19+ POS\_0)로 구성되는지 확인

대각선 이름이 A, B로 설정되어 있는지 확인

기본 중앙 출구 경로가 B로 설정되어 있는지 확인

**테스트 결과:** 통과 - 전통 보드의 모든 속성이 정확히 설정됨

**6.2 testPentagonBoardShape**()

**목적**: 오각형 보드 형태의 구조 검증

**테스트 내용:**

외곽 경로가 26개 위치(0~24+ POS\_0)로 구성되는지 확인

대각선 이름이 A, B, C로 설정되어 있는지 확인

기본 중앙 출구 경로가 B로 설정되어 있는지 확인

**테스트 결과:** 통과 - 오각형 보드의 모든 속성이 정확히 설정됨

**6.3 testHexagonBoardShape**()

**목적**: 육각형 보드 형태의 구조 검증

**테스트 내용**:

외곽 경로가 31개 위치(0~29+ POS\_0)로 구성되는지 확인

대각선 이름이 A, B, C로 설정되어 있는지 확인

기본 중앙 출구 경로가 C로 설정되어 있는지 확인

**테스트 결과:** 통과 - 육각형 보드의 모든 속성이 정확히 설정됨

**6.4 testTraditionalDiagPaths(**)

**목적:** 전통 보드의 대각선 경로 구성 검증

**테스트 내용**:

A 대각선 경로가 7개 위치로 구성되고 CENTER를 통과하는지 확인

B 대각선 경로가 7개 위치로 구성되고 CENTER를 통과하는지 확인

각 대각선의 시작점과 끝점이 올바른지 확인

**테스트 결과**: 통과 - 모든 대각선 경로가 정확히 구성됨

**6.5 testDistanceToEnd(**)

**목적:** 각 대각선에서 END까지의 거리 계산 정확성 검증

**테스트 내용**:

B 대각선에서 END까지 거리가 20인지 확인

A 대각선에서 END까지 거리가 5인지 확인

테스트 **결과:** 통과 - 거리 계산이 정확히 수행됨

**6.6 testCache DistanceToEnd(**)

**목적**: 거리 계산 결과 캐싱 기능 동작 검증

**테스트 내용:**

동일한 대각선에 대해 두 번 거리 계산 수행

두 결과가 동일한지 확인

캐싱된 결과가 정확한 값인지 확인

**테스트** 결과**:** 통과 - 캐싱 기능이 정상적으로 작동함

**7. BoardTest 클래스**

**7.1 testInitialBoardState(**)

**목적**: 게임 보드의 초기 상태가 올바르게 설정되는지 검증

**테스트 내용:**

모든 위치(OFFBOARD, END 제외)에 빈 리스트가 있는지 확인

완주한 말 목록이 비어있는지 확인

**테스트 결과**: 통과 보드 초기화가 정확히 수행됨

**7.2 testGetPiecesAtSpecialPositions(**)

**목적:** 특수 위치에 대한 조회 처리 검증

**테스트 내용:**

OFFBOARD 위치 조회 시 빈 리스트 반환 확인

END 위치 조회 시 빈 리스트 반환 확인

**테스트 결과:** 통과 - 특수 위치 처리가 정확함

**7.3 testPlacePieceNormalPosition()**

**목적:** 일반 위치에 말 배치 기능 검증

**테스트 내용:**

POS\_O에 말 배치 시 잡기 발생하지 않음 확인 말의 위치가 POS\_0으로 변경되는지 확인 해당 위치에 말이 포함되어 있는지 확인

**테스트 결과:** 통과 - 일반 위치 말 배치가 정상 작동함

**7.4 testPlacePieceToEnd**()

**목적**: END 위치에 말 배치 시 완주 처리 검증

**테스트 내용**:

END 위치 배치 시 잡기 발생하지 않음 확인

말의 위치가 END로 변경되는지 확인

말이 완주 상태로 변경되는지 확인

완주 목록에 말이 추가되는지 확인

**테스트 결과:** 통과 - 완주 처리가 정확히 수행됨

**7.5 testCapturePiece(**)

**목적:** 상대방 말을 잡는 기능 검증

**테스트 내용**:

상대방 말이 있는 위치에 말 배치 시 잡기 발생 확인

잡힌 말이 OFFBOARD로 이동하는지 확인

잡은 말이 해당 위치에 배치되는지 확인

**테스트 결과**: 통과 - 말 잡기 기능이 정확히 구현됨

**7.6 testSamePlayerPiecesNotCaptured()**

**목적**: 같은 플레이어의 말은 잡히지 않는 규칙 검증

**테스트 내용:**

같은 플레이어의 두 말을 같은 위치에 배치

잡기가 발생하지 않음 확인

해당 위치에 두 말이 모두 배치되는지 확인

**테스트 결과**: 통과 - 같은 플레이어 말 중복 배치가 정상 처리됨

**7.7 testRemovePiece(**)

**목적**: 보드에서 말을 제거하는 기능 검증

**테스트 내용**:

POS\_15에 말 배치 후 해당 위치에 포함되는지 확인

말 제거 후 해당 위치에서 제거되는지 확인

**테스트 결과:** 통과 - 말 제거 기능이 정상 작동함

**8. PieceTest 클래스**

**8.1 testInitialState()**

목적: 게임말의 초기 상태가 올바르게 설정되는지 검증

**테스트 내용**:

말의 소유자가 올바르게 설정되는지 확인

초기 위치가 OFFBOARD인지 확인

완주 상태가 false인지 확인

경로 컨텍스트와 마지막 진입점이 null인지 확인

**테스트 결과:** 통과 - 모든 초기 상태가 정확히 설정됨

**8.2 testMoveTo()**

**목적**: 일반 위치로 이동하는 기능 검증

**테스트 내용:**

POS\_5로 이동 시 위치 변경 확인

완주 상태가 false로 유지되는지 확인

**테스트 결과:** 통과 - 일반 위치 이동이 정상 작동함

**8.3 testMoveToEnd**()

**목적**: END 위치 이동 시 완주 처리 검증

**테스트 내용:**

END 위치로 이동 시 위치 변경 확인

완주 상태가 true로 변경되는지 확인

경로 컨텍스트가 초기화되는지 확인

**테스트 결과:** 통과 - 완주 처리가 정확히 수행됨

**8.4 testMoveToOffboard()**

**목적:** OFFBOARD 이동 시 컨텍스트 초기화 검증

**테스트 내용:**

경로 컨텍스트 설정 후 다른 위치로 이동

OFFBOARD로 이동 시 위치 변경 확인

경로 컨텍스트가 초기화되는지 확인

**테스트 결과:** 통과 OFFBOARD 이동 시 컨텍스트 초기화가 정상 처리됨

**8.5 testPathContextWaypoint()**

**목적**: 경로 컨텍스트 관리 기능 검증

**테스트 내용:**

경로 컨텍스트 설정 시 올바르게 저장되는지 확인

경로 컨텍스트 초기화 시 null로 변경되는지 확인

**테스트 결과**: 통과 - 경로 컨텍스트 관리가 정확히 구현됨

**8.6 testLastEnteredWaypoint()**

**목적**: 마지막 진입 지점 추적 기능 검증

**테스트 내용:**

마지막 진입 지점 설정 시 올바르게 저장되는지 확인

**테스트** 결과: 통과 - 마지막 진입 지점 추적이 정상 작동함

**9. PlayerTest** 클래스

**9.1 testPlayerCreation()**

**목적:** 플레이어 객체 생성이 올바르게 수행되는지 검증

**테스트 내용:**

플레이어 이름이 설정된 값과 일치하는지 확인

생성된 말의 개수가 설정된 값(4개)과 일치하는지 확인

**테스트 결과**: 통과 - 플레이어 생성이 정확히 수행됨

**9.2 testPiecesOwnership(**)

**목적:** 생성된 말들의 소유권이 올바르게 설정되는지 검증

**테스트** 내용**:**

플레이어가 소유한 모든 말의 소유자가 해당 플레이어인지 확인

**테스트 결과**: 통과 - 모든 말의 소유권이 정확히 설정됨

**9.3 testHasFinishedAllPieces()**

**목적:** 게임 완료 판단 로직 검증

**테스트** 내용:

초기 상태에서 게임 완료 상태가 false인지 확인

모든 말을 END로 이동 후 게임 완료 상태가 true인지 확인

**테스트 결과**: 통과 - 게임 완료 판단이 정확히 구현됨

**9.4 testPiecesInitialPosition()**

목적: 플레이어의 모든 말이 올바른 초기 상태로 설정되는지 검증

**테스트 내용:**

모든 말의 초기 위치가 OFFBOARD인지 확인

모든 말의 완주 상태가 false인지 확인

테스트 **결과:** 통과 - 모든 말의 초기 상태가 정확히 설정됨

**10. PositionTest** 클래스

**10.1 testPositionValues(**)

**목적**: 게임에서 사용되는 모든 위치 상수가 올바르게 정의되어 있는지 검증

**테스트 내용:**

기본 위치(OFFBOARD, END, CENTER)가 존재하는지 확인

외곽 위치(POS\_0, POS\_30)가 존재하는지 확인

대각선 위치(DIA\_A1, DIA\_A4, DIA\_B1, DIA\_B4)가 존재하는지 확인

**테스트 결과**: 통과 - 모든 필요한 위치 상수가 정의됨

**10.2 testPositionOrder()**

**목적:** Position 열거형의 순서가 올바르게 정의되어 있는지 검증

**테스트 내용**:

전체 위치 배열의 길이가 0보다 큰지 확인

첫 번째 위치가 OFFBOARD인지 확인

마지막 위치가 END인지 확인

**테스트 결과**: 통과 - 위치 열거형 순서가 정확히 정의됨

**11. 향후 확장 가능성**

**11.1 새로운 UI 프레임워크 지원**

* **Web UI**: React, Vue.js 기반 웹 인터페이스
* **Mobile UI**: Android, iOS 네이티브 앱
* **Desktop UI**: Electron, WPF 등

**11.2 게임 기능 확장**

* **네트워크 멀티플레이**: 온라인 대전 기능
* **AI 플레이어**: 컴퓨터 상대방 추가
* **게임 기록**: 플레이 기록 저장 및 분석

**11.3 접근성 개선**

* **다국어 지원**: i18n 인터페이스 추가
* **접근성 지원**: 시각/청각 장애인을 위한 UI
* **커스터마이징**: 사용자 정의 테마 및 설정

**12. 결론**

**12.1 설계의 성공 요인**

* 인터페이스 기반 추상화
* 명확한 계층 분리
* 확장 가능한 구조

**12.2 향후 개선 방향**

* 더 많은 UI 프레임워크 지원
* 성능 최적화
* 사용자 경험 개선

**12.3 프로젝트의 의의**

* 객체지향 설계 원칙의 실천
* 유지보수 가능한 코드 구조
* 확장 가능한 아키텍처

이러한 설계를 통해 UI 프레임워크 변경에 대한 유연성을 극대화하고, 향후 확장성과 유지보수성을 보장할 수 있도록 했습니다.

**13. Github 프로젝트 리포트**

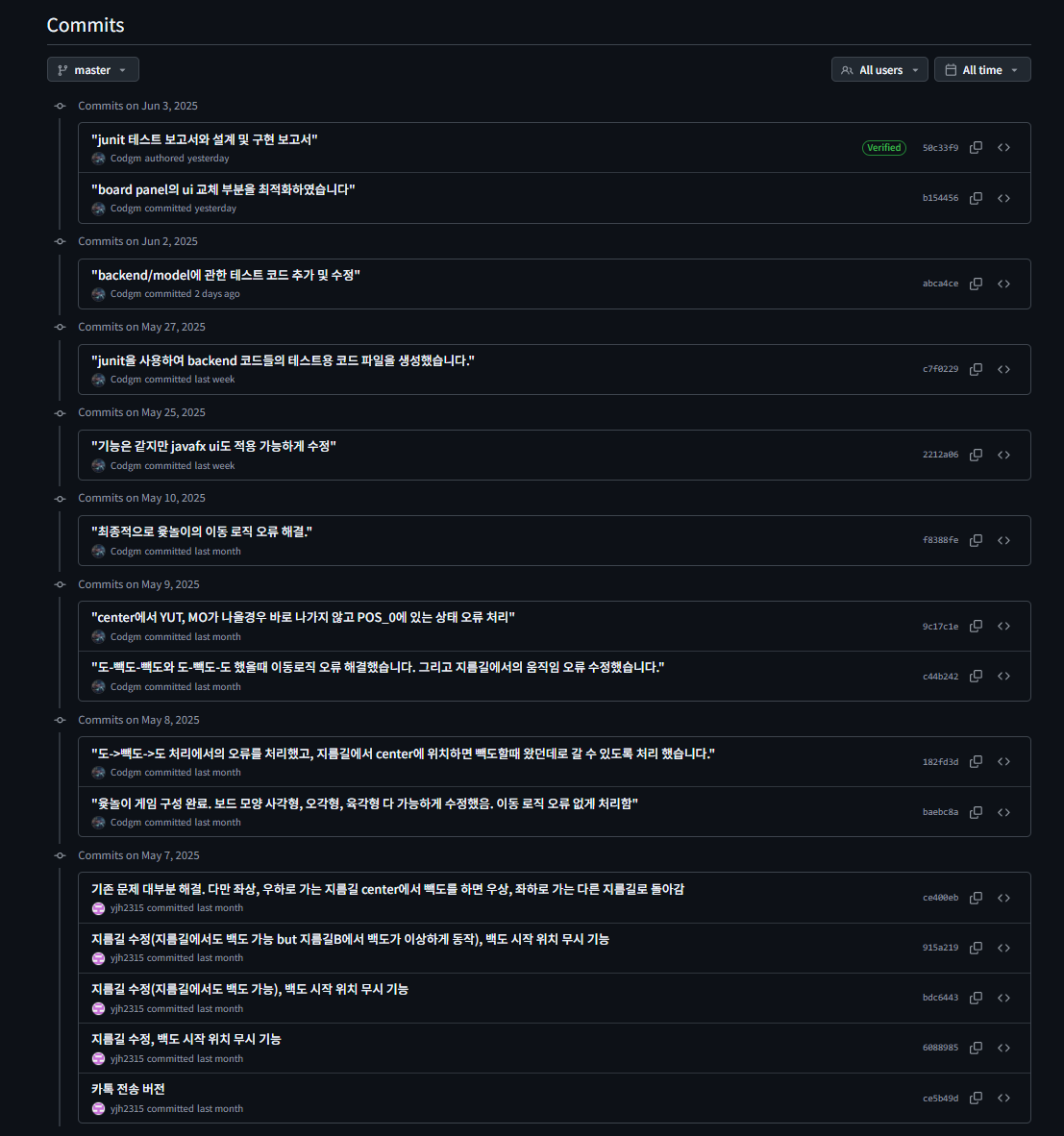
* 1. **팀의 Github 프로젝트 주소**

<https://github.com/SE-Project-Yutnori/Yutnori-Team8>

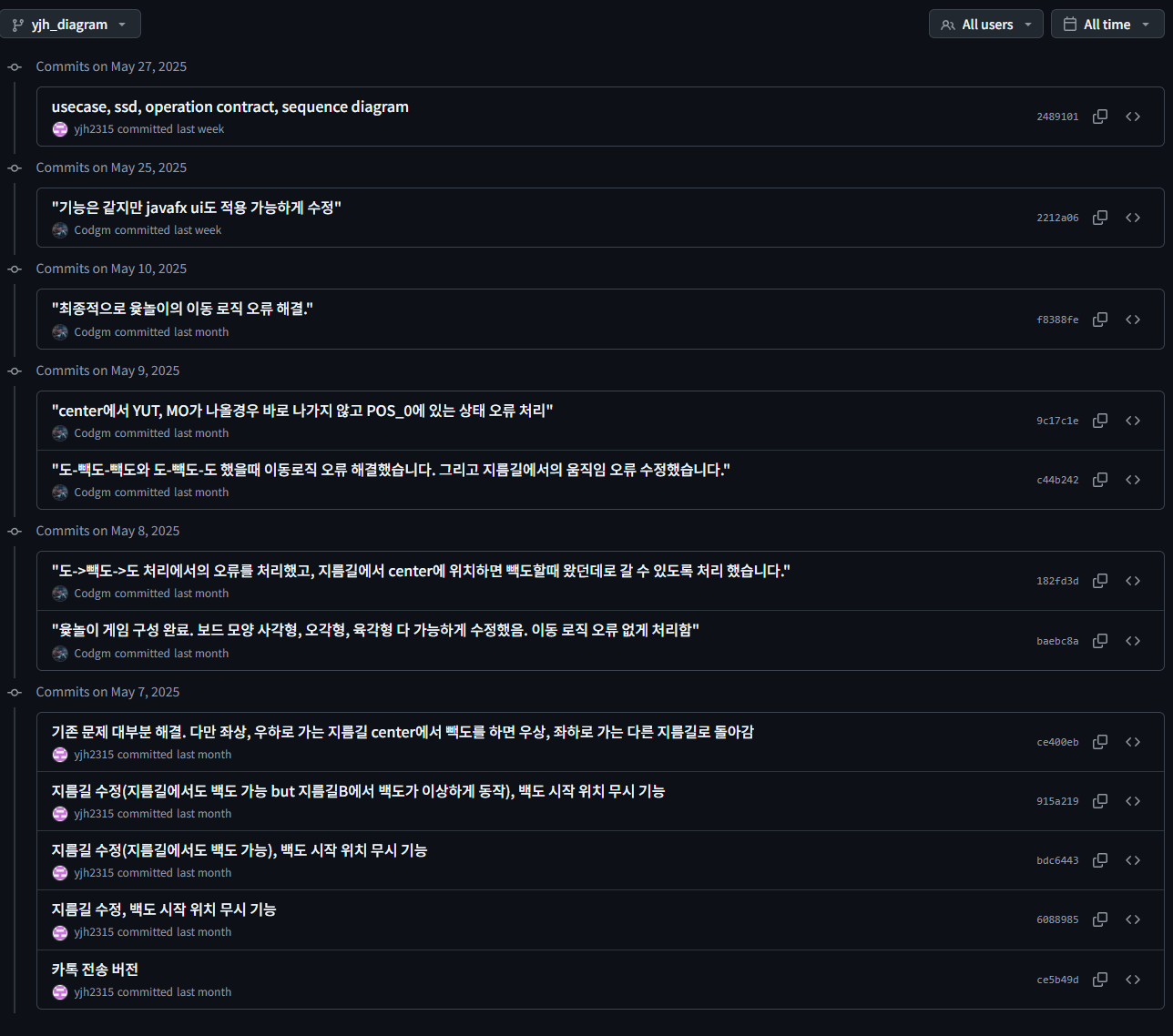
**13.2** **progress history 스크린샷**

****

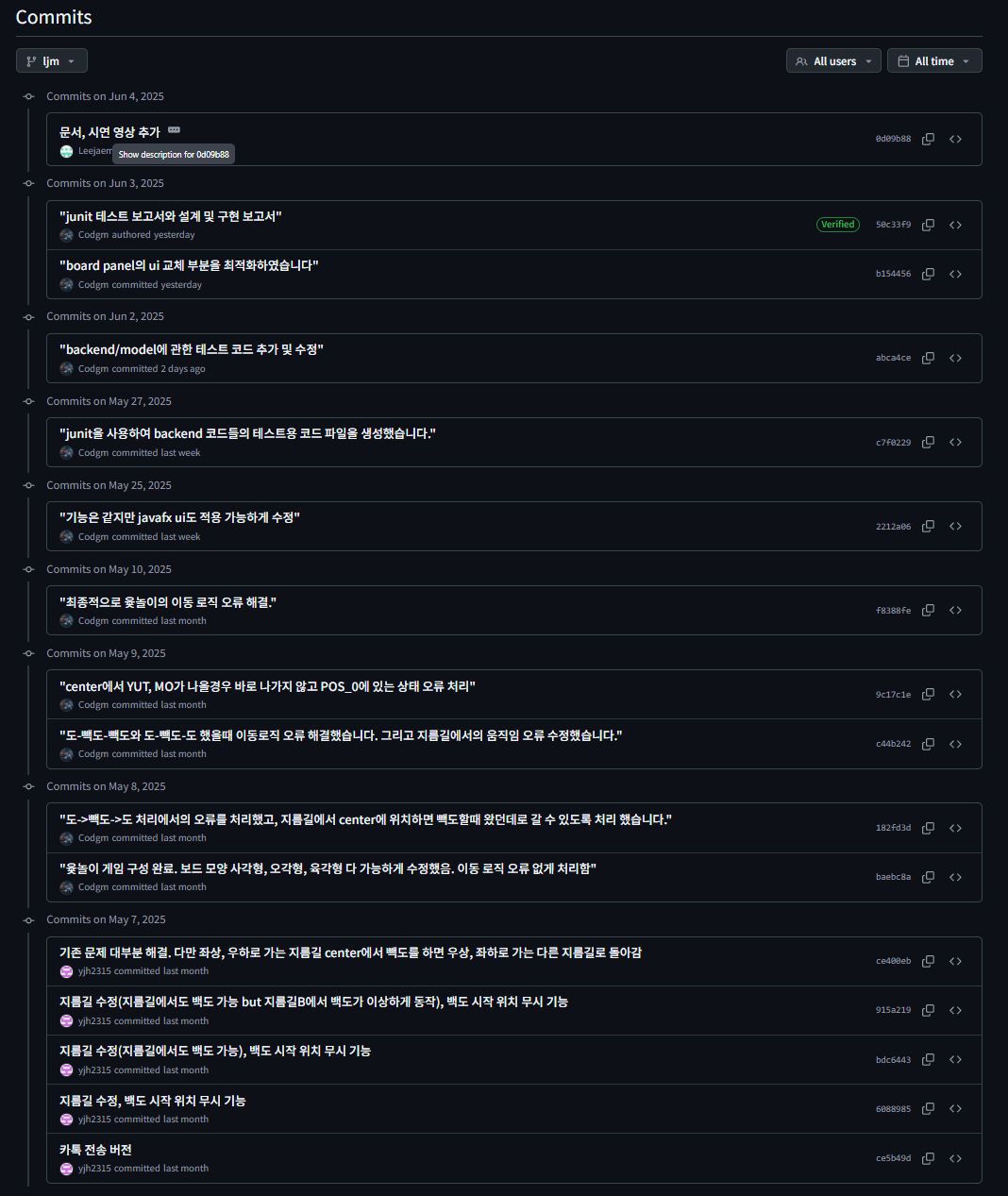
서규민(20225679)



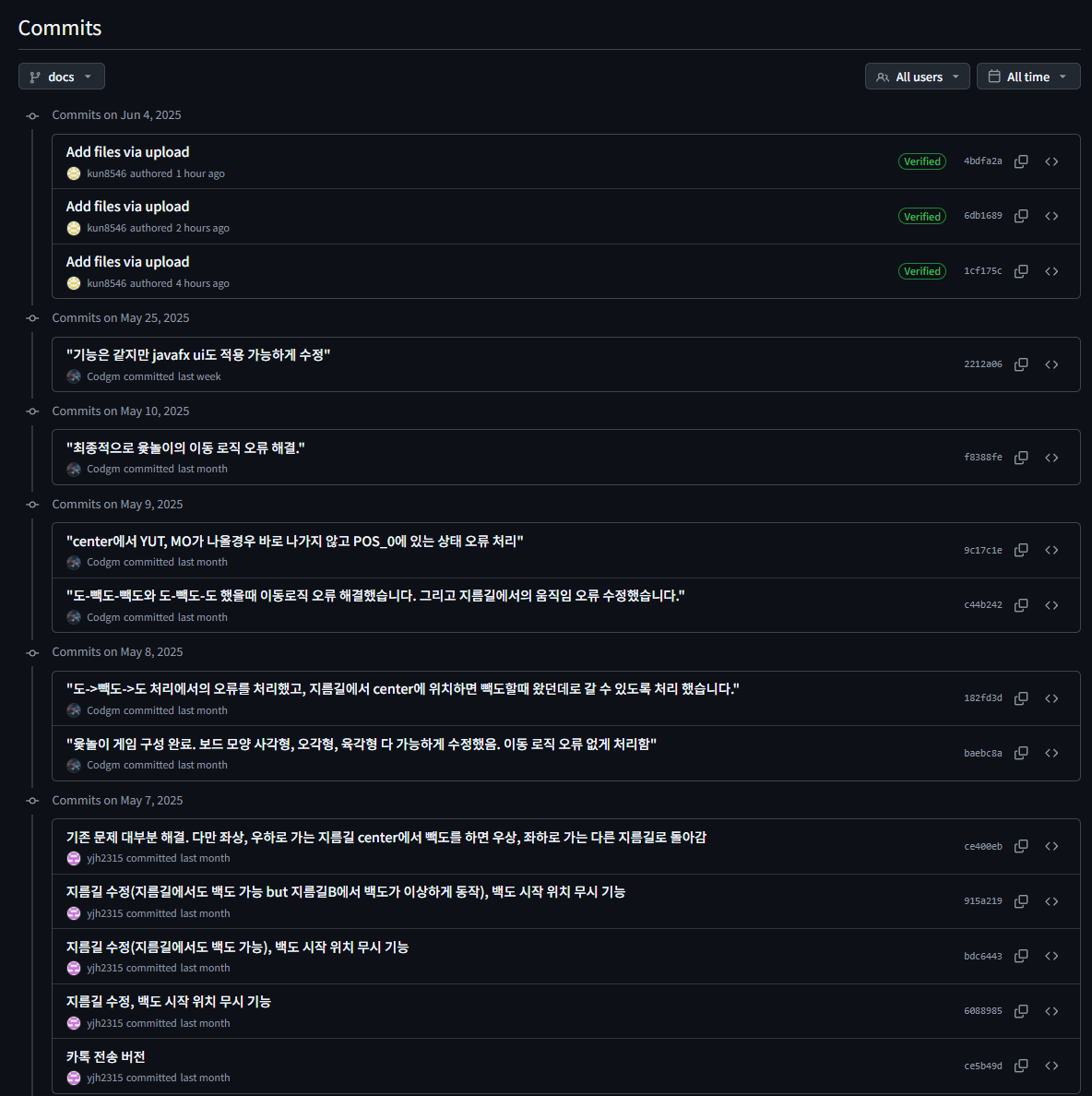
윤지훈(20225889)



이재민(20235464)

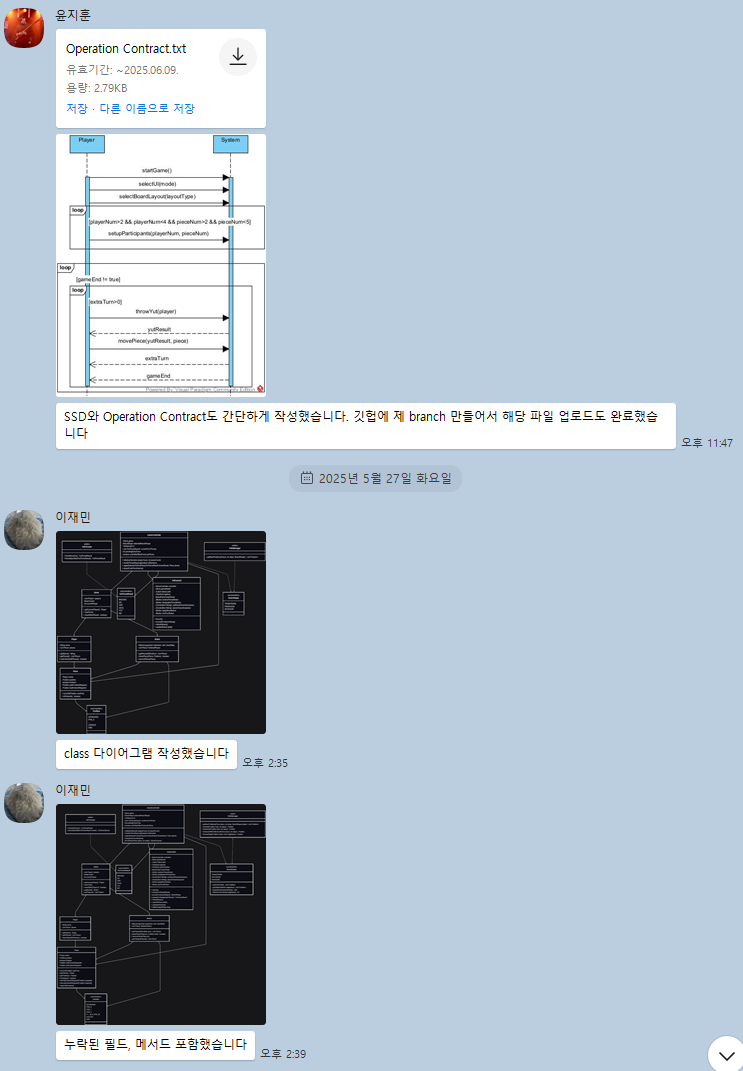


박건희(20214897)

****

* 1. **기타 작업 기록**

****

****