SE_Term_Project_2025

Team8

서규민(20225679)

윤지훈(20225889)

이재민(20235464)

박건희(20214897)

목차

- Requirements Elicitation
- Analysis
- Architecture / Detailed Design
- Testing
- Q&A

Requirements Elicitation(Use Case Model)

Functional-Requirements

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 던다

FR9. 말이 골인지점에 도달하면 골인처리되고 사라진다

FR10. 모든 말이 골인처리되면 해당 사용자는 승리한다

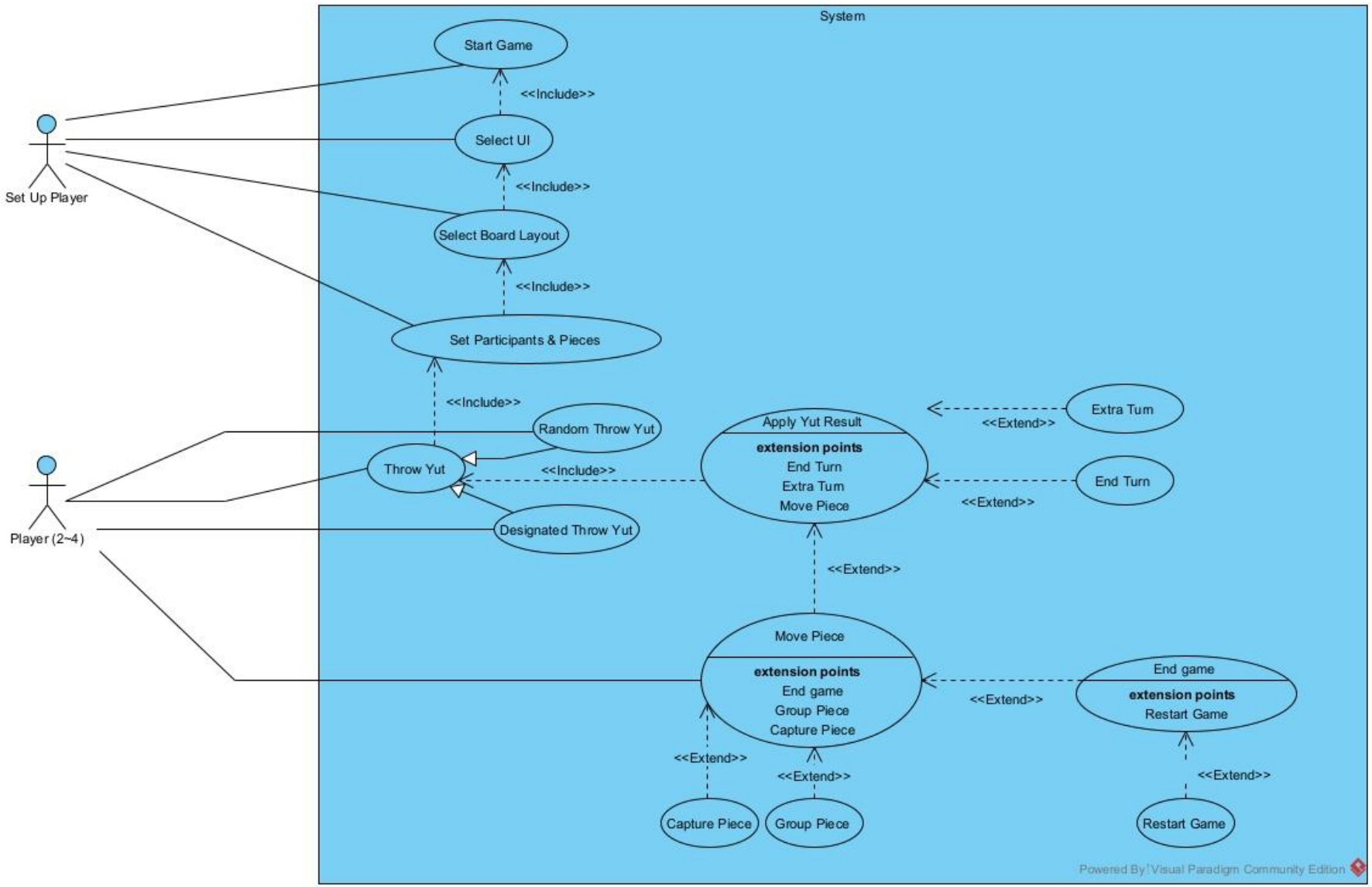
Non-Functional-Requirements

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.

Requirements Elicitation(Use Case Diagram)



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)

개방-폐쇄 원칙 (Open-Closed Principle)

- 새로운 UI 프레임워크 추가 시 기존 코드 수정 불필요
- AbstractBoardPanel을 통한 공통 기능 확장

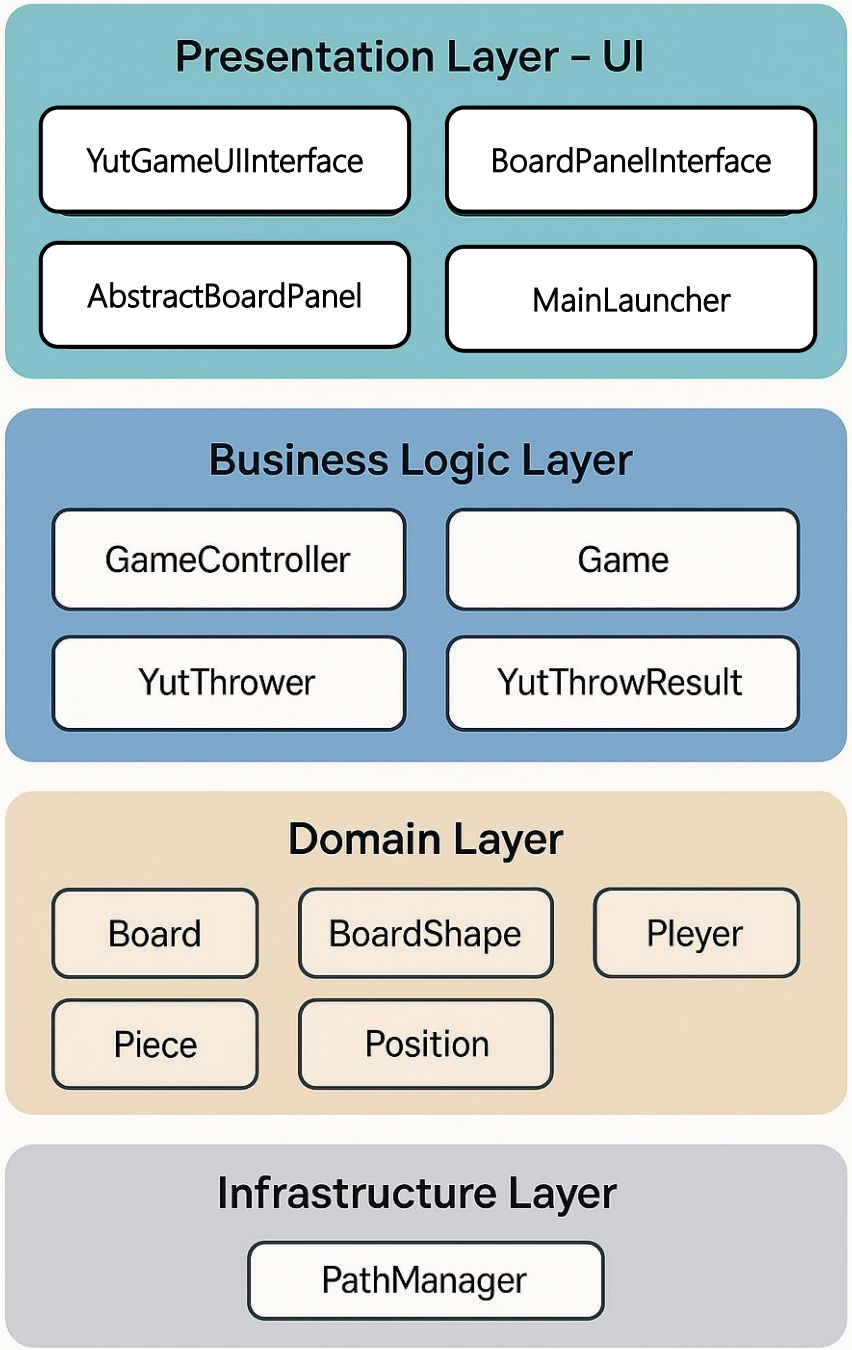
단일 책임 원칙 (Single Responsibility Principle)

- PathManager: 경로 계산만 담당
- YutThrower: 윷 던지기 로직만 담당
- GameController: 게임 흐름 제어만 담당

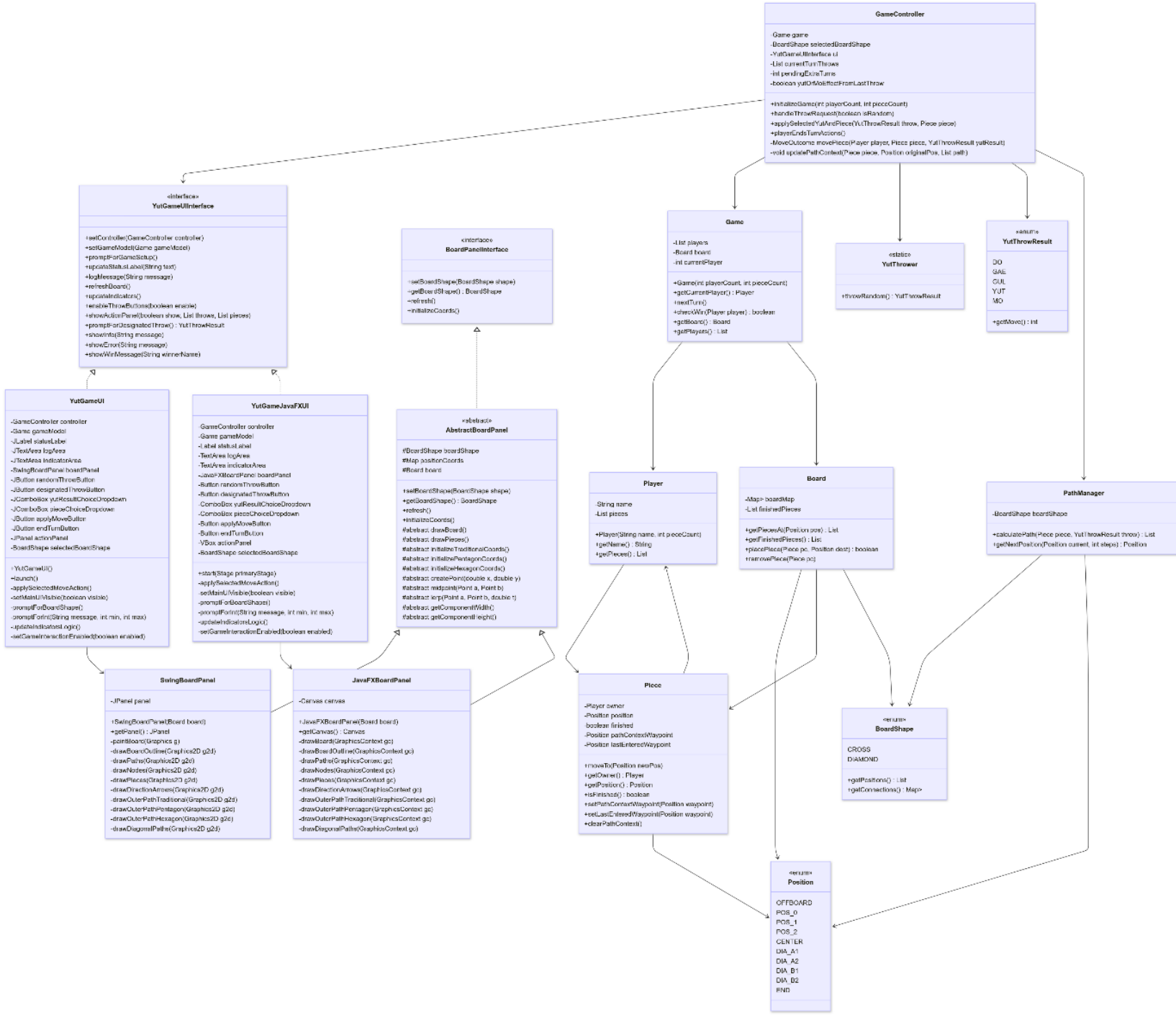
프로젝트 설계

MVC (Model-View-Controller) 패턴

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

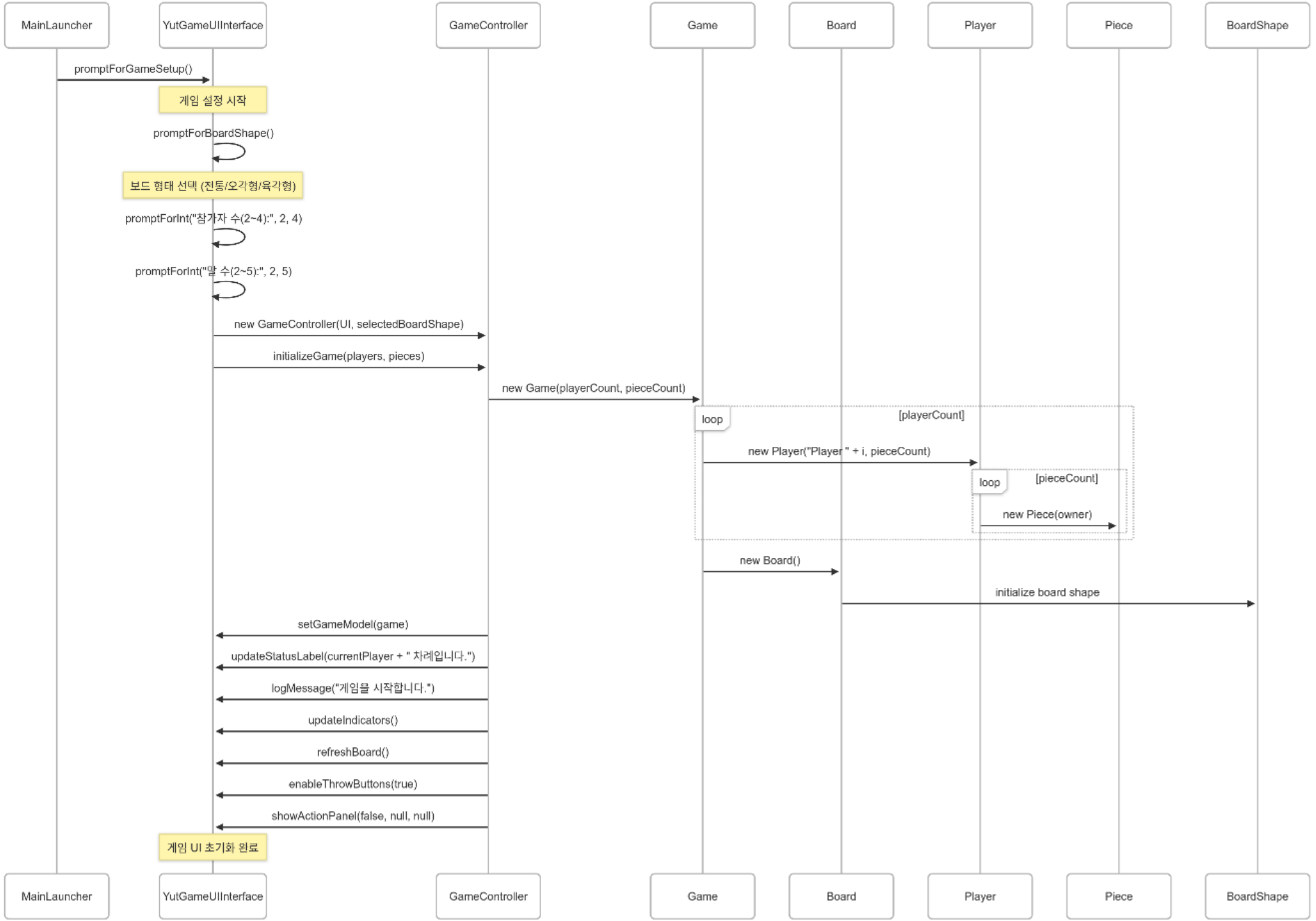


Analysis
(Class-diagram)



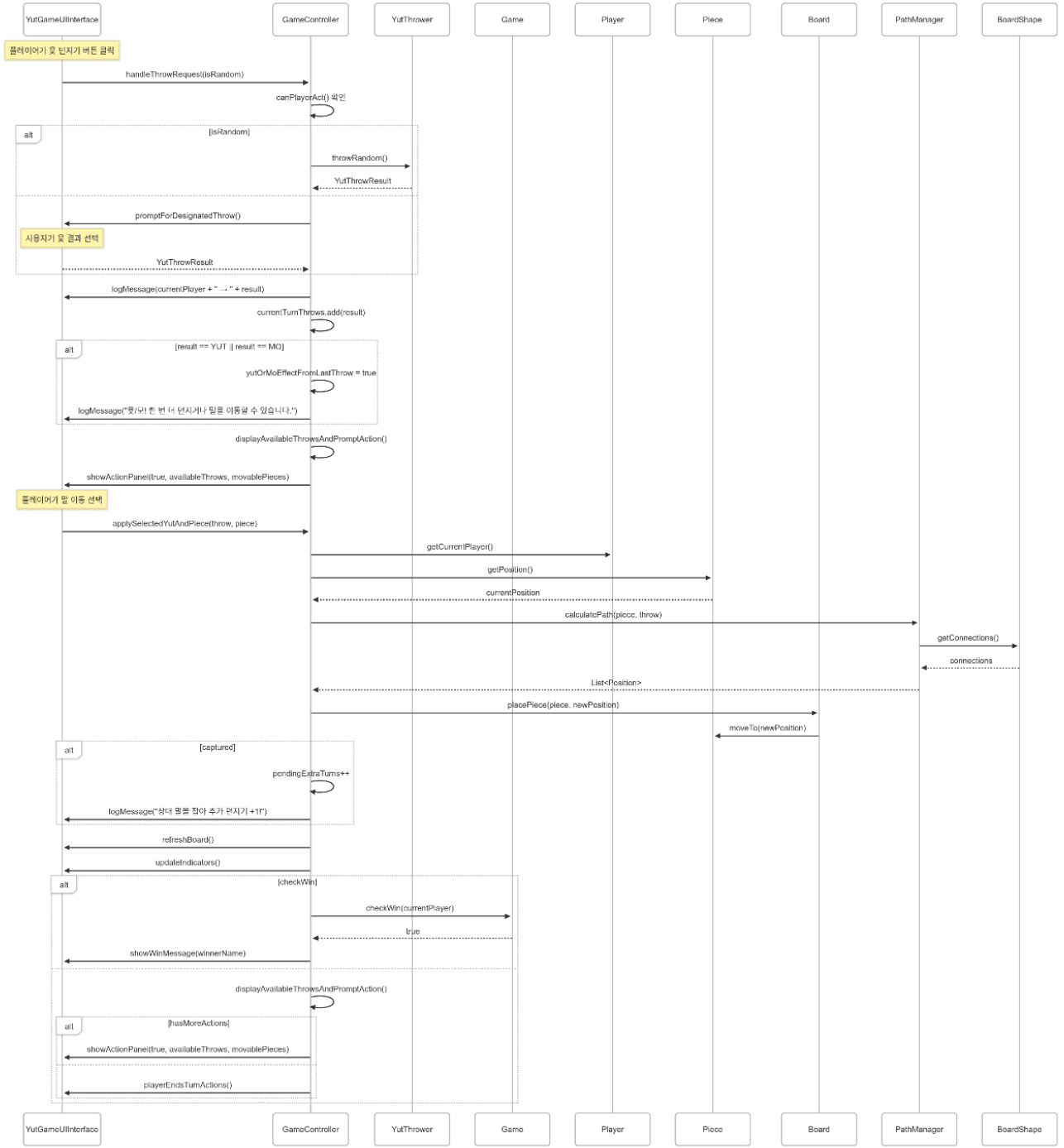
Sequence Diagram

- **게임 초기화**



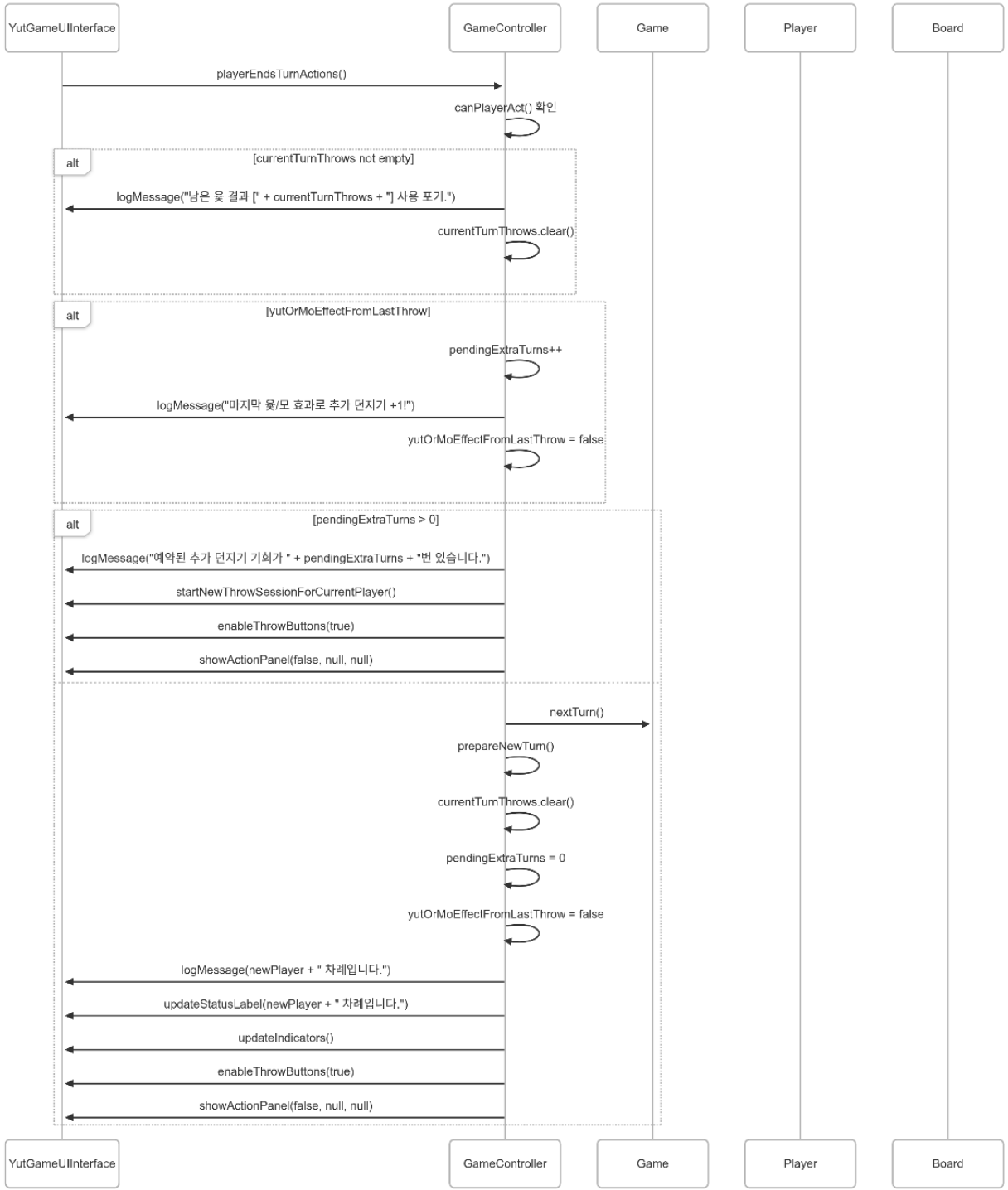
Sequence Diagram

- **윷 던지기 및 말 이동**



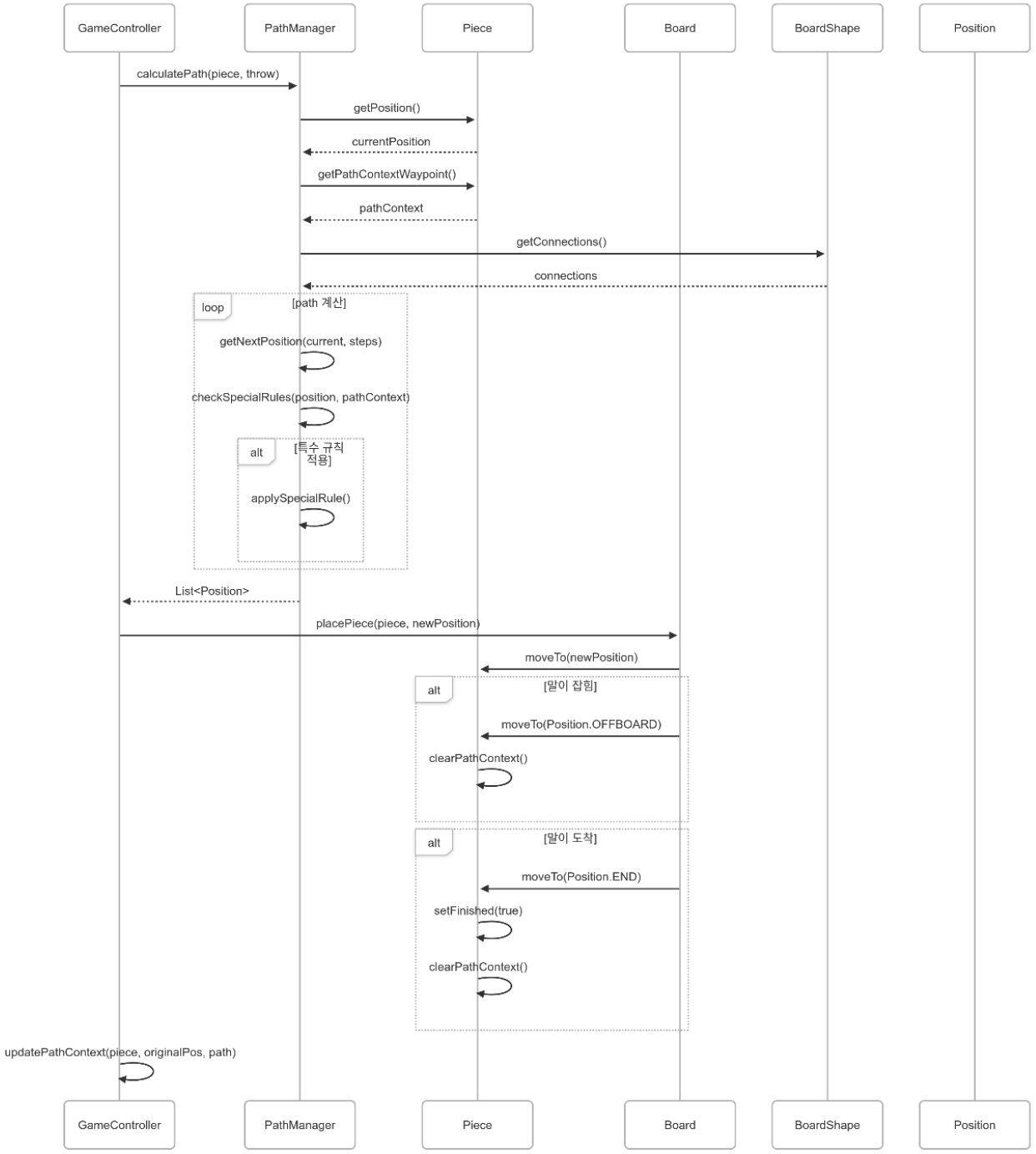
Sequence Diagram

- **턴 종료 및 다음 턴 시작**



Sequence Diagram

- **말 경로 계산 및 이동**



Junit test

```java
@Test
void testGameInitialization() {
    Game game = controller.getGame();
    assertNotNull(game);
    assertEquals(PLAYER_COUNT, game.getPlayers().size());
    assertEquals(PIECE_COUNT, game.getPlayers().get(0).getPieces().size());
}

@Test
void testHandleThrowRequest() {
    // 랜덤 던지기 테스트
    controller.handleThrowRequest(true);
    List<YutThrowResult> availableThrows = controller.getCurrentAvailableThrows();
    assertNotNull(availableThrows);
}

@Test
void testApplySelectedYutAndPiece() {
    // 초기 말 선택
    Player currentPlayer = controller.getGame().getCurrentPlayer();
    Piece piece = currentPlayer.getPieces().get(0);

    // 윷 결과 설정
    controller.handleThrowRequest(true);
    YutThrowResult result = controller.getCurrentAvailableThrows().get(0);

    // 말 이동
    controller.applySelectedYutAndPiece(result, piece);

    // 말이 이동했는지 확인
    assertNotEquals(Position.OFFBOARD, piece.getPosition());
}
```

```java
@Test
void testThrowSpecified() {
    // 지정된 결과가 정확히 반환되는지 확인
    assertEquals(YutThrowResult.BACKDO, YutThrower.throwSpecified(YutThrowResult.BACKDO));
    assertEquals(YutThrowResult.DO, YutThrower.throwSpecified(YutThrowResult.DO));
    assertEquals(YutThrowResult.GAE, YutThrower.throwSpecified(YutThrowResult.GAE));
    assertEquals(YutThrowResult.GEOL, YutThrower.throwSpecified(YutThrowResult.GEOL));
    assertEquals(YutThrowResult.YUT, YutThrower.throwSpecified(YutThrowResult.YUT));
    assertEquals(YutThrowResult.MO, YutThrower.throwSpecified(YutThrowResult.MO));
}

@Test
void testThrowRandom() {
    // 여러 번 던져서 모든 가능한 결과가 나올 수 있는지 확인
    boolean[] results = new boolean[6];
    int attempts = 1000;

    for (int i = 0; i < attempts; i++) {
        YutThrowResult result = YutThrower.throwRandom();
        results[result.ordinal()] = true;
    }

    // 모든 결과가 최소 한 번은 나왔는지 확인
    for (boolean result : results) {
        assertTrue(result, "모든 윷 결과가 최소 한 번은 나와야 합니다.");
    }
}
```

```java
@Test
void testThrowRandomDistribution() {
    // 확률 분포가 예상대로인지 확인
    int[] counts = new int[6];
    int attempts = 10000;

    for (int i = 0; i < attempts; i++) {
        YutThrowResult result = YutThrower.throwRandom();
        counts[result.ordinal()]++;
    }

    // 각 결과의 확률이 예상 범위 내에 있는지 확인
    // BACKDO: 5%
    assertTrue(counts[YutThrowResult.BACKDO.ordinal()] > attempts * 0.03);
    assertTrue(counts[YutThrowResult.BACKDO.ordinal()] < attempts * 0.07);

    // DO: 25%
    assertTrue(counts[YutThrowResult.DO.ordinal()] > attempts * 0.20);
    assertTrue(counts[YutThrowResult.DO.ordinal()] < attempts * 0.30);

    // GAE: 25%
    assertTrue(counts[YutThrowResult.GAE.ordinal()] > attempts * 0.20);
    assertTrue(counts[YutThrowResult.GAE.ordinal()] < attempts * 0.30);

    // GEOL: 20%
    assertTrue(counts[YutThrowResult.GEOL.ordinal()] > attempts * 0.15);
    assertTrue(counts[YutThrowResult.GEOL.ordinal()] < attempts * 0.25);

    // YUT: 15%
    assertTrue(counts[YutThrowResult.YUT.ordinal()] > attempts * 0.10);
    assertTrue(counts[YutThrowResult.YUT.ordinal()] < attempts * 0.20);

    // MO: 10%
    assertTrue(counts[YutThrowResult.MO.ordinal()] > attempts * 0.05);
    assertTrue(counts[YutThrowResult.MO.ordinal()] < attempts * 0.15);
}
```

성능 및 메모리 최적화

메모리 관리

- 효율적인 데이터 구조 사용
- 불필요한 객체 생성 최소화

렌더링 최적화

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

향후 확장 가능성

새로운 UI 프레임워크 지원

게임 기능 확장

접근성 개선

Q&A