소프트웨어공학 팀프로젝트 보고서
윷놀이 게임

Team17

팀원

배민웅 유민성 위종인 희건춘 향원원

목차

A. 프로젝트 개요

본 프로젝트는 전통 보드게임인 윷놀이를 소프트웨어로 구현한 것이다.

개발은 Java 기반의 MVC 아키텍처로 진행되었으며, 사용자 명수(최소 2 명, 최대 4 명)와 게임 말 갯수(최소 2 개, 최대 5 개)를 지정할 수 있다. 윷던지기 방식(랜덤 및 수동 선택), 게임 판 커스터마이징 사각형 을 지원한다.

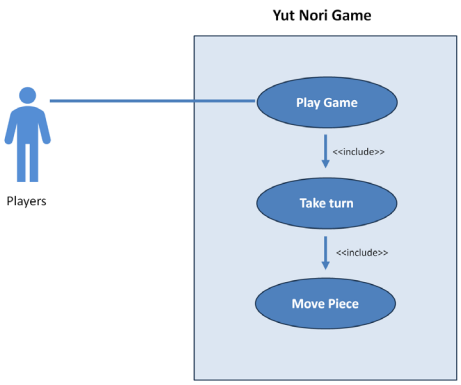
게임의 승리 조건은 한 플레이어가 모든 말을 먼저 도착지점까지 이동시키는 것이다.

한 컴퓨터에서 여러 명이 돌아가며 플레이하며, 통신 기능은 요구되지 않는다.

본 프로젝트는 UI 분리 및 테스트 용이성을 고려하여 설계되었고, 두 가지 서로 다른 UI 를 지원한다 (Swing, CLI 등).

B. 유스케이스 모델

i. 유스케이스



ii. 유스케이스 텍스트

Use Case 1: Move Piece

- **Scope**: Game Board
- **Level**: Subfunction
- **Primary Actor**: Player
- **Stakeholders and Interests**:
  - **Player**: Wants to select and move a piece to maximize advantage.
  - **Other Players**: Want their own pieces to remain safe.
- **Preconditions**:
  - The game is in progress.
  - The player has at least one movable piece.
- **Postconditions**:
  - The selected piece is moved according to the game rules.
  - Possible capture, grouping, or scoring is applied.
- **Main Success Scenario**:
- The player selects a piece.
- The piece moves along the appropriate path.
- If it lands on an opponent’s piece, it is captured.
- If it lands on a friendly piece, they are grouped.
- If it reaches the end, it scores and is removed from the board.
- **Extensions**:
  - **2a.** If the piece starts from home, it enters the board.
  - **2b.** If the result is Back-Do, the piece moves backward.
  - **3a.** If no movable piece exists, the turn ends immediately.

Use Case 2: 턴을 진행한다

- **Scope**: Yut Nori Game
- **Level**: user-goal
- **Primary Actor**: Player (whose turn it is)
- **Stakeholders and Interests**:
  - **Current Player**: Wants to advance the game and eventually win.
  - **Other Players**: Want the game to be fair and not lose.
- **Preconditions**:
  - The game is in progress.
  - It is the player’s turn.
- **Postconditions**:
  - The player has taken all throws and moved pieces.
  - The turn passes to the next player, or the player wins the game.

- **Main Success Scenario**:
- The player throws the Yut sticks.
- The result is displayed.
- If Yut or Mo, the player throws again.
- The player selects a piece and moves it (see Use Case: Move Piece).
- If the player captures an opponent’s piece, they get an extra throw.
- If a piece scores, it is removed.
- If all pieces are scored, the player wins.
- Otherwise, the turn ends.
- **Extensions**:
    - **1a.** The player uses a designated Yut result (for testing).
    - **4a.** The player has no valid piece to move → the turn ends.

Use Case 3: Play Game

- **Scope**: Yut Nori Game
- **Level**: user-goal
- **Primary Actor**: Players
- **Stakeholders and Interests**:
    - **Players**: Want to play the game and win.
    - **Game System**: Manages turns, rules, and outcome display.
- **Preconditions**:
    - At least two players are present.
    - The game has started with proper settings.
- **Postconditions**:
    - One player has won the game.
- **Main Success Scenario**:
- Players set the number of participants and pieces.
- The first player begins the turn.
- Players alternate turns using the "Take Turn" use case.
- The game continues until a player wins.
- The winner is displayed.
- Players choose to restart or exit.

C.기타 요구사항 문서

UC – 01: 말을 움직인다.

Operation Contract : movePiece(Piece piece)

항목	설명
연산 이름	movePiece(Piece piece)
입력값	Piece: 현재 플레이어가 선택한 말 객체
사전 조건 (Preconditions)	– 게임이 진행 중이어야 함– 플레이어가 이미 윷을 던졌어야 함 (hasRolled == true) – 이동하려는 말이 현재 플레이어 소속이어야 함 – 해당 말이 현재 주사위(윷) 결과로 이동 가능해야 함
사후 조건 (Postconditions)	● 말이 현재 주사위(윷) 결과에 따라 새로운 위치로 이동됨 ● 해당 말이 리드 피스일 경우, 스택된 말들도 함께 이동함 ● 이동 위치에 상대 말이 있을 경우, ■ 해당 상대 말과 그 아래에 스택된 말들은 모두 위치 -1 (home)으로 이동됨 ■ 기존 보드 위치 ↔ 말 관계는 해제됨 ■ reThrowAllowed == true, captureOccurred == true로 설정됨 ■ 해당 말을 잡고 홈(-1)으로 보냄 ● 이동 위치에 아군 말이 있을 경우

	■ 해당 말들은 piece.stackPiece()를 통해 piece 밑으로 스택됨 ■ 스택된 말들은 더 이상 독립적인 위치를 갖지 않으며, 리드 피스에 종속됨 ● 도착 위치가 목표 지점일 경우 ■ piece 및 스택된 말들은 모두 보드에서 제거됨 (position == -2 등으로 표시) ■ 현재 플레이어의 완료된 말 수 (finishedCount)가 증가함 ■ 모든 말이 완료되었을 경우, gameOver == true, winner == currentPlayer로 설정됨 ● 모든 말이 점수 처리되었을 경우, 게임이 종료되고 승자가 설정됨 ● 이동이 유효하고 완료되었을 경우 true를 반환하며, 실패 시 false 반환 ● reThrowAllowed가 true이면 hasRolled는 false로 리셋됨 (재던지기 허용) ● 그렇지 않으면 hasRolled는 true 상태로 유지됨
관련 항목	– Use Case 1: 말 이동– Use Case 2: 턴 진행 (Step 4~6)

Operation Contract : handlePieceInteraction(Piece piece, int position)

Field	Description
Operation Name	handlePieceInteraction(Piece piece, int position)
Inputs	- piece: 이동을 완료한 말 객체 (리드 피스여야 함)- position: 말이 도착한 위치 인덱스
Preconditions	- 게임이 진행 중이어야 함 (gameOver == false)- piece 는 이미 position 위치로 이동한 상태여야 함- piece 는 스택된 말이 아닌, 리드 피스여야 함 (piece.isStacked() == false)- 도착한 위치에는 다른 말이 없거나 존재할 수 있음
Postconditions	- position 에 **상대방의 말**이 있을 경우: • 해당 말과 그 하위 스택 말들은 모두 position = -1 (홈)으로 이동됨 • 캡처된 말들과 리드 피스 간의 연관 관계가 모두 해제됨 • reThrowAllowed = true, captureOccurred = true 로 설정됨 - position 에 **같은 팀의 말**이 있을 경우: • 해당 말은 piece.stackPiece()를 통해 piece 아래로 스택됨 • 스택된 말은 독립된 위치 정보가 제거되고, 리드 피스의 하위로 종속됨 - 그 외에는 아무런 상호작용 없이 종료됨

	- 반환값은 없으며, 모든 상태 변화는 side effect 로 발생함
Cross References	- movePiece(Piece, int, PathType)에서 호출됨 - reThrowAllowed, captureOccurred 는 이후 로직 제어에 사용됨 - Use Case 1: Move Piece (Step 3 – 캡처, Step 4 – 그룹핑) - Use Case 2: Take Turn (Step 5 – 캡처 시 재던지기 가능)

Operation Contract: getNextPosition(int currentPosition, int steps, PathType pathType)

항목	설명
연산 이름	getNextPosition(int currentPosition, int steps, PathType pathType)
입력값	– currentPosition: 현재 말이 위치한 인덱스 또는 노드 ID – steps: 앞으로 이동할 칸 수 (일반적으로 1~5) – pathType: 사용할 경로 전략 (MAIN_PATH, DIAGONAL_LEFT, DIAGONAL_RIGHT 등)
사전 조건 (Preconditions)	– currentPosition 은 지정된 pathType 경로 내의 유효한 노드여야 함 – steps 는 1 이상이어야 함 – 해당 pathType 에 해당하는 경로가 보드 내에 정의되어 있어야 함 – 게임 보드는 선택된 BoardType 에 따라 적절한 경로들이 초기화되어 있어야 함
사후 조건 (Postconditions)	– 입력된 steps 만큼 지정된 pathType 경로를 따라 이동한 후의 **도착 위치**가 반환됨 – 만약 이동이 경로의 마지막 노드를 넘어서게 되면, 도착 위치는 목표 지점(예: 30 번)으로 간주하여 반환됨 – 만약 currentPosition 이 해당 경로 상에 없을 경우(예: 집 또는 시작 지점), 경로의 첫 번째 인덱스부터 시작해서 이동함

	– 보드 내부의 경로 로직(path list)은 변경되지 않고 그대로 유지됨 (immutable)
관련 항목 (Cross References)	– Game.movePiece()에서 새로운 위치 계산 시 호출됨 – Board.getPathByType(PathType)에서 내부 경로 리스트를 제공함 – Use Case 1: 말 이동 (Step 2 – 경로 따라 이동)

Operation Contract: finishMove()

Field	Description
Operation Name	finishMove()
Preconditions	- 게임이 진행 중이어야 함 (gameOver == false) - 현재 플레이어가 말을 이동시킨 후거나, 윷을 던진 직후 상태여야 함 -availableRolls 또는 reThrowAllowed 중 하나는 true 일 수 있음 - 현재 플레이어가 존재해야 하며, 최소 1 명의 플레이어가 게임에 참여 중이어야 함
Postconditions	- 만약 availableRolls 가 비어 있지 않다면: • 아무 상태 변화 없이 반환됨 (턴은 계속됨) - 만약 availableRolls 가 비어 있고 reThrowAllowed == true 라면: • hasRolled 가 false 로 설정됨 (→ 플레이어가 다시 던질 수 있도록 함) • 턴 종료 없이 반환됨 - 위 조건이 모두 false 인 경우: • endTurn()이 호출되어 다음 상태 변화가 발생함: – 현재 플레이어 인덱스(currentPlayerIndex)가 다음 플레이어로 변경됨 – availableRolls 는 초기화됨 (clear)

	– hasRolled, reThrowAllowed, captureOccurred 는 모두 false 로 초기화됨
Cross References	- movePiece() 이후 마지막에 호출되어 턴 종료 여부를 판별함 - reThrowAllowed, availableRolls 플래그 기반으로 흐름 분기- endTurn() → 플레이어 순환 및 턴 리셋 처리 담당 - Use Case 2: Take Turn (Step 8 – 턴 종료 조건 판단 및 전환)

UC – 02 : 턴을 진행한다.

Operation Contract: handleThrowResult(int result)

Field	**Description**
Operation Name	handleThrowResult(int result)
Inputs	- result: 윷 던지기의 결과값. 0~5 사이의 정수 (BACKDO = 0, DO = 1, ..., MO = 5)
Preconditions	- 게임이 진행 중이어야 함 (gameOver == false) - result 는 0 이상 5 이하의 유효한 윷 결과여야 함 - currentPlayer 가 존재해야 하며, 윷 던질 수 있는 시점이어야 함 (hasRolled == false)

Postconditions	- currentRoll 이 result 로 설정됨 → 게임의 현재 윷 결과가 갱신됨 - hasRolled == true 로 설정됨 → 현재 턴에서 던졌음을 명시 - availableRolls 리스트에 result 값이 추가됨 - reThrowAllowed 플래그는 다음 조건에 따라 설정됨: • result 가 YUT(4) 또는 MO(5)일 경우 → true • 그렇지 않을 경우 → false
Cross References	- throwYut() 또는 throwSpecifiedYut(int) → 이 메서드를 호출하여 결과 처리 - Game.getCurrentRoll() 및 getAvailableRolls() → 이후 말 이동 시 사용 - Use Case 2: Take Turn (Step 2 – 윷 결과 확인, Step 3 – 재던지기 여부 판별)

Operation Contract 4: endTurn()

Field	Description
Operation Name	endTurn()
Inputs	없음 (void method)
Preconditions	- 게임이 진행 중이어야 함 (gameOver == false) - 현재 플레이어의 턴이 진행 중이었고, 해당 턴의 처리가 완료된 상태여야 함 - 최소 2명의 플레이어가 게임에 참여하고 있어야 함 (players.size() ≥ 2)
Postconditions	- 현재 턴을 가진 플레이어의 상태가 초기화됨: • hasRolled = false • currentRoll = 0 • availableRolls 리스트는 clear됨 • reThrowAllowed = false • captureOccurred = false – currentPlayerIndex가 (currentPlayerIndex + 1) % players.size()로 업데이트됨– 그에 따라 getCurrentPlayer()의 반환 대상이 변경됨 → **턴 주도권이 다음 플레이어로 이동됨** – Game 객체 내부의 다음 필드가 변경됨: • currentPlayerIndex: 증가됨 (참조 관계 변경) • hasRolled: false

	• currentRoll: 0 • availableRolls: clear() • reThrowAllowed: false • captureOccurred: false – 이전 턴 플레이어와의 주도권 참조 관계가 해지됨 (내부적으로는 유지되지만, 주도권은 새로운 플레이어에게 이동됨)– 반환값 없음 (side effect로 게임 상태를 변경함)
Cross References	- finishMove() → 턴 종료 조건을 만족할 때 호출함 - YutNoriApp의 End Turn 버튼 → 이 메서드에 직접 연결되어 있음 - Use Case 2: Take Turn (Step 8 – 턴 종료 및 다음 플레이어 전환) - Game.getCurrentPlayer() → 이후의 모든 행동 주체가 변경됨

UC – 03 : 게임을 플레이한다.

Operation Contract: getPlayerCount() (플레이어 수 설정)

Field	Description
Operation Name	getPlayerCount()
Inputs	없음 (파라미터 없음)
Preconditions	- Game 또는 PlayerSelectionDialog 인스턴스가 생성되어 있어야 함 - players 혹은 playerNames 배열이 초기화된 상태여야 함 (null 아님)
Postconditions	- 현재 게임 또는 UI에서 설정된 플레이어 수를 정수 값으로 반환함 - 반환된 값은 내부적으로 보유한 컬렉션(List<Player> 또는 String[] playerNames)의 길이 또는 유효값의 개수임 - 반환값은 외부에 복사된 int 값으로 전달되며, 참조 관계를 생성하지 않음 - 반환값은 Game 내부의 players.size() 또는 PlayerSelectionDialog.selectedPlayerCount에 의존함.
Cross References	- initializeGame(int playerCount, ...) 등에서 플레이어 수 체크에 사용 가능 - UI 구성 시 (예: 칸 크기 계산, 턴 순환 등) 로직의 조건

	으로 사용 - Use Case 3: Play Game (Step 1 – 참가자 수 설정 및 확인)

Operation Contract: checkGameOver() (게임 종료 판단)

Field	Description
Operation Name	checkGameOver()
Inputs	없음
Preconditions	- Game 객체가 초기화되어 있어야 함 - players 리스트가 null이 아니고, 각 Player 객체가 hasFinished() 상태를 반환할 수 있어야 함 - 말 이동 등의 행동 이후 상태에서 호출되는 것이 일반적
Postconditions	- 만약 한 명의 플레이어라도 hasFinished() == true이면: • gameOver = true로 설정됨 • winner 참조가 해당 플레이어 객체로 설정됨 • 이후 모든 턴/이동 로직은 gameOver == true 조건에 의해 차단 가능 - 그렇지 않으면: • gameOver는 기존 상태 유지 (false) • winner == null 상태 유지 - 반환값 true는 게임 종료 여부를 의미하며, 호출자는 이를 바탕으로 추가 UI 처리를 수행할 수 있음
Cross References	- movePiece(), finishMove() 등 말 이동 이후 호출됨 - getWinner()는 winner 필드에 접근하므로 이 메서드의

	실행 여부에 의존함 - Use Case 2: Take Turn (Step 7 – 모든 말이 완주 시 승리 판정)

추가적으로 구현되어야 할 메소드

1. 말이 한 바퀴를 다 돌았을 때, 말을 없애주기

```
<예시 코드>
private void handleGoalArrival(Piece piece) {
    final int GOAL_POSITION = 30;  // 말이 도달해야 할 완주 위치
    if (piece.getPosition() == GOAL_POSITION) {
        // 말 제거 처리
        piece.setPosition(-2); // -2: 완주한 말로 간주
        // 말이 스택된 상태면, 함께 있는 말도 제거
        for (Piece stacked : piece.getStackedPieces()) {
            stacked.setPosition(-2);
        }
        // 플레이어의 완주 말 수 증가
        getCurrentPlayer().incrementFinishedCount(); // Player 클래스에서 구현 필요
        System.out.println("Piece reached goal and was removed. Finished count: "
                + getCurrentPlayer().getFinishedCount());
        // 게임 종료 조건 확인
        if (getCurrentPlayer().hasFinished()) {
            gameOver = true;
            winner = getCurrentPlayer();
            System.out.println("Game over! Winner: " + winner.getName());
        }
    }
```

Field	Description
Operation Name	handleGoalArrival(Piece piece)
Inputs	- piece: 목표 지점에 도달한 말 객체 (리드 피스일 수 있음)
Preconditions	- piece 는 현재 플레이어 소속이어야 함 (piece.getOwner() == getCurrentPlayer()) - piece.getPosition()이 목표 지점 위치(예: 30 번)이어야 함 - Game 객체가 초기화되어 있고, 게임이 종료되지 않은 상태여야 함 (gameOver == false)
Postconditions	- piece.setPosition(-2) 또는 removePieceFromBoard(piece) 호출됨으로써 말이 보드에서 제거됨

	- 해당 플레이어의 완주 말 수(finishedCount) 가 1 증가함 - 만약 해당 말이 스택된 말들을 보유하고 있다면: • 모든 하위 말들도 함께 보드에서 제거됨 • 하위 말들과의 스택 연관 관계가 해제됨 -만약 player.hasFinished() == true 가 되었을 경우: – gameOver = true – winner = player - piece 는 Board 상에서 위치 참조가 제거됨 - stackedPieces 를 보유한 경우: • 각 하위 말은 piece 와의 연결이 끊기고, 보드에서 제거됨 • 스택 해제 및 위치 제거 발생
Cross References	- movePiece() – 도착 위치가 목표 지점일 경우 이 메서드를 호출함 - checkGameOver() – 내부에서 호출되거나 이어서 호출됨 - Use Case 1: Move Piece (Step 5 – 골 도달 및 제거 처리) - Use Case 2: Take Turn (Step 7 – 모든 말이 도달하면 승리 처리)

2. 랜덤 윷 던지기 or 지정 윷 던지기 지정 (과제물 포함 조건)

```
<예시 코드 - UI 에서 지정 윷 던지기 버튼 구현은 별도로 필요>
public void throwSpecifiedYut(int result) {
    if (result < 0 || result > 5) throw new
IllegalArgumentException("Invalid Yut result");
    handleThrowResult(result);
}
```

항목	설명
연산 이름 (Operation Name)	throwSpecifiedYut(int result) – 지정된 윷 결과로 던지기
입력값 (Inputs)	- result: 윷 결과를 나타내는 정수값 (0 = 백도, 1 = 도, 2 = 개, 3 = 걸, 4 = 윷, 5 = 모)
사전 조건 (Preconditions)	- 입력값 result 는 0 이상 5 이하의 유효한 값이어야 함- 게임이 진행 중이어야 함 (gameOver == false) - 현재 플레이어가 아직 윷을 던지지 않은 상태여야 함 (hasRolled == false)
사후 조건 (Postconditions)	- 유효한 입력값인 경우, 내부적으로 handleThrowResult(result)가 호출됨- 그 결과, 다음 필드들이 업데이트됨: • currentRoll ← result 값 • hasRolled ← true • availableRolls ← result 추가 • reThrowAllowed ← 윷(4) 또는 모(5)일 경우 true 로 설정

	- 입력값이 유효하지 않은 경우 (0~5 범위를 벗어날 경우): • IllegalArgumentException 예외가 발생 • 게임의 어떤 상태도 변경되지 않음 (side effect 없음)
관련 항목 (Cross References)	- handleThrowResult(int result) – 이 메서드를 내부에서 직접 호출하여 상태를 설정함 - Use Case 2: 턴 진행하기 (1a 단계 – 플레이어가 지정된 윷 결과를 사용할 때) - UI 구성 요소: YutThrowPanel 의 수동 윷 던지기 버튼과 직접 연결됨

3. 윷 판 커스터마이징 (과제물 포함 조건 – 사각형, 오각형, 육각형 중 선택)

```
<예시 코드 – 오각형, 육각형 board는 별도 구현 필요>
public class Board {
    private final BoardType type;

    public Board(BoardType type) {
        this.type = type;
        // 각 타입에 따른 경로 초기화
        if (type == BoardType.SQUARE) {
            initSquarePath();
        } else if (type == BoardType.PENTAGON) {
            initPentagonPath();
        } else if (type == BoardType.HEXAGON) {
            initHexagonPath();
        }
    }

    private void initSquarePath() {
        // 기존 경로
    }
    private void initPentagonPath() {
        // 오각형 경로 정의
    }
    private void initHexagonPath() {
        // 육각형 경로 정의
    }
    public BoardType getType() {
        return type;
    }
}
```

항목	설명
연산 이름 (Operation Name)	Board(BoardType type) – 보드 객체 생성자
입력값 (Inputs)	- type: 사용자가 선택한 보드 형태를 나타내는 열거형 값 (SQUARE, PENTAGON, HEXAGON 중 하나)
사전 조건 (Preconditions)	- type은 null이 아니어야 함- 게임이 아직 시작되지 않은 상태여야 함 (보드가 초기화되기 전) - 해당 BoardType에 대응하는 경로 초기화 메서드 (initSquarePath(), initPentagonPath(), initHexagonPath())가 구현되어 있어야 함

사후 조건 (Postconditions)	- 입력된 type에 따라 보드 내부 구조(경로 노드, 위치 매핑 등)가 초기화됨 - 내부 필드 this.type은 입력된 BoardType으로 설정됨- 세 가지 중 **해당 보드 형태에 대응되는 경로 초기화 메서드만** 실행됨: • SQUARE → initSquarePath() • PENTAGON → initPentagonPath() • HEXAGON → initHexagonPath() - 이후 보드는 말의 이동 경로 계산 (getNextPosition(), getPreviousPosition())에 사용할 준비가 완료된 상태가 됨
관련 항목 (Cross References)	- Use Case 3: 게임 플레이 시작 (Step 1 – 보드 형태 선택)- getNextPosition(), getPreviousPosition() – 이동 계산 시 이 보드 구조를 기반으로 작동함

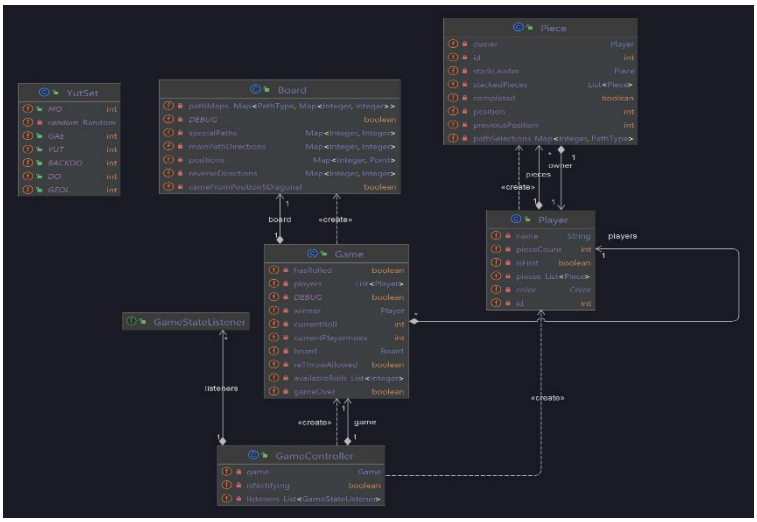
D.설계 및 구현 리포트

i. 설계 주안점

MVC 아키텍처 적용:

Model: Player, Piece, Board, Game, YutSet
View: GameBoard, UISelectionScreen 등
Controller: GameController

클래스 다이어그램:

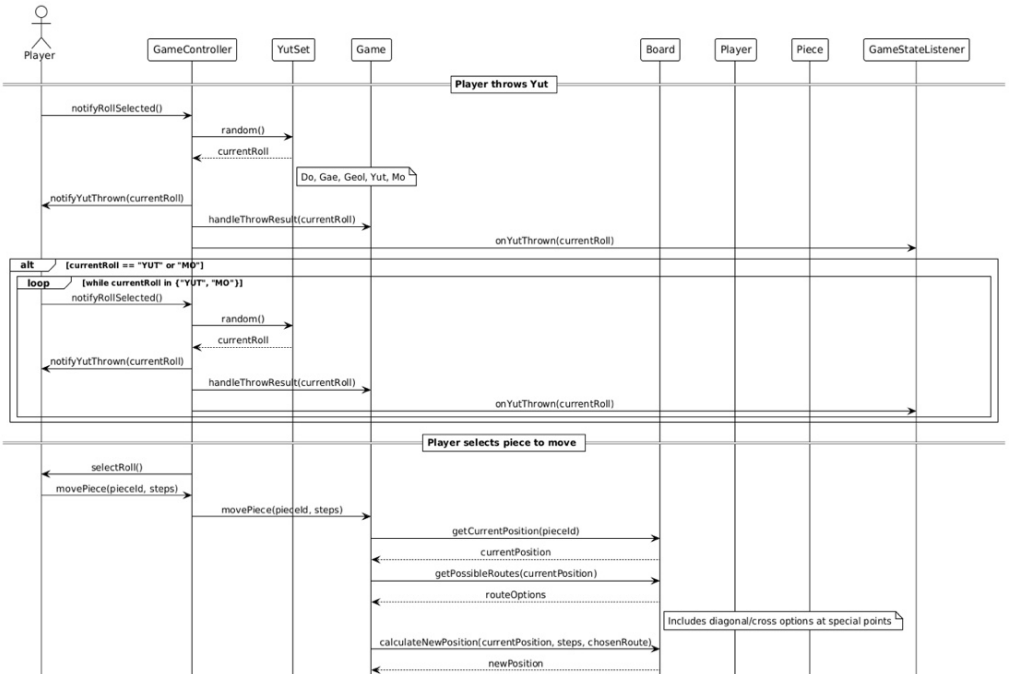


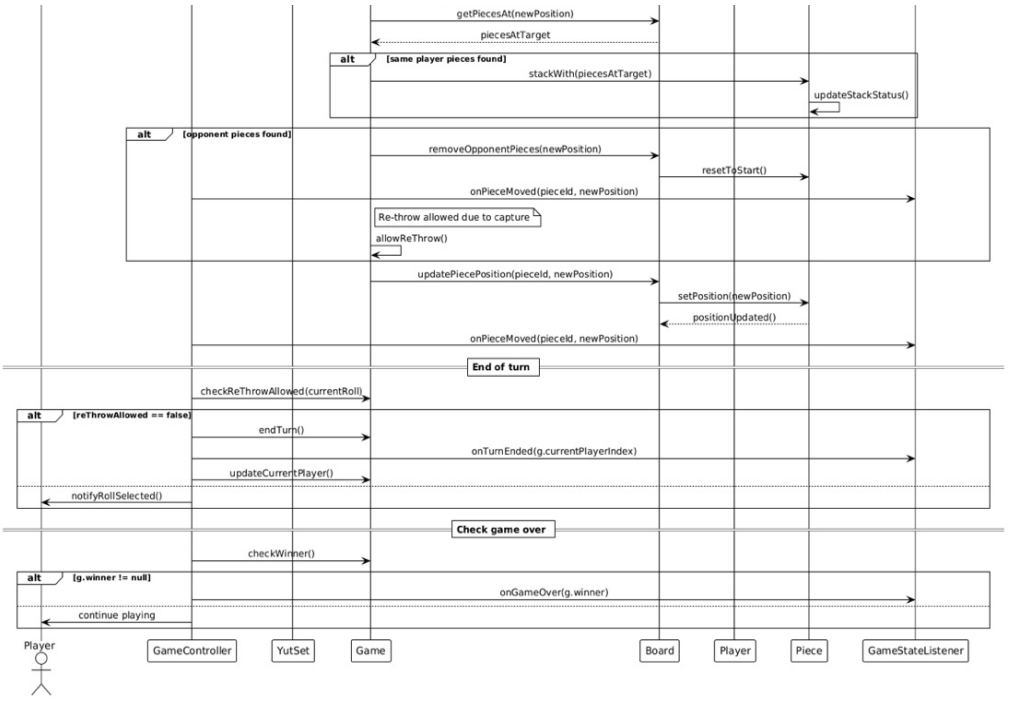
Graphic: 1 ModelClassDiagram

GameController 가 Game 모델을 조작하고 상태 변화를 GameStateListener 에게 알림

Game 은 Board 와 Players 를 포함, Player 는 여러 Piece 를 소유

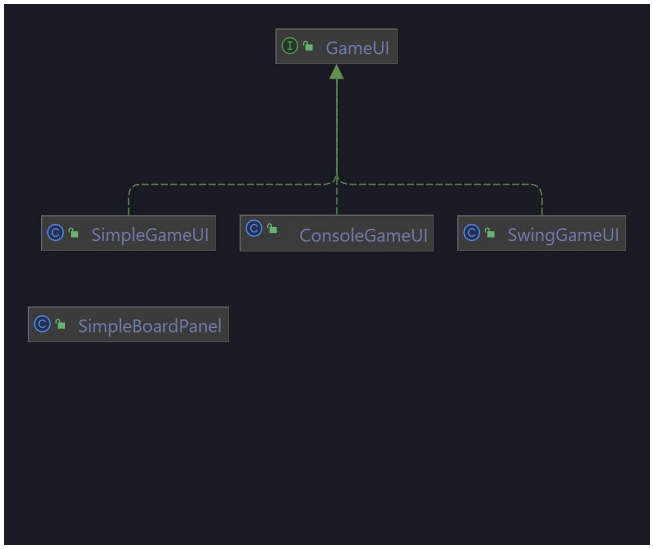
시퀀스 다이어그램





턴 시작 → 윷 던지기 → 윷 결과 → 사용자 말 선택 → movePiece()

ii. UI 교체시 변경 최소화 설명



Graphic: 2 UISelectionScreen_CD

UI 계층과 Model/Controller 계층을 Observer 패턴과 Listener 인터페이스(GameStateListener)로 분리하여 결합도를 낮췄음

새로운 UI 도입 시, GameController 를 직접 수정할 필요 없이 GameStateListener 인터페이스만 구현하면 동일 로직 재사용 가능

E. 테스트 리포트

i. JUnit 테스트 케이스

PieceTest 클래스 테스트

테스트 메소드	테스트 목적	결과
testThrowYut()	윷 던지기 결과가 유효한 6가지 중 하나인지 검증	PASS
testIsRethrowAllowed()	윷/모는 다시 던질 수 있고 나머지는 불가함 확인	PASS

테스트 메소드	테스트 목적	결과
testGetRollName()	윷 결과에 대한 한글 이름 반환값이 올바른지 확인	PASS

PlayerTest 클래스 테스트

테스트 메소드	테스트 목적	결과
testPlayerInitialization()	플레이어 생성시 이름, 색상, 말 4 개 생성 여부 검증	PASS
testGetPiece()	ID 로 말 조회 기능 확인, 없는 ID 는 null 반환 확인	PASS
testHasFinished()	모든 말이 완주했을 때 완료 여부 확인	PASS
testGetFinishedPieceCount()	완료된 말 수 집계 정확성 테스트	PASS
testResetPieces()	말 위치 초기화, 겹침 초기화가 제대로 수행되는지 확인	PASS
testGetMovablePiece()	이동 가능한 말 선택 기능 확인	PASS

YutSetTest 클래스 테스트

테스트 메소드	테스트 목적	결과
testInitialState()	말 생성시 초기 상태(ID, 소유자, 위치 등)가 올바른지 확인	PASS
testSetPosition()	위치 변경시 현재/이전 위치 업데이트 동작 확인	PASS
testHasFinished()	완주 여부 판별 정확성 확인	PASS
testSetCompleted()	완료 상태 직접 설정 기능 확인	PASS
testStackPieces()	말 겹치기 및 겹침 이동/해제 동작 확인	PASS
testClearStackedPieces()	겹쳐진 말 전체 해제 기능 테스트	PASS

BoardTest 클래스 테스트

테스트 메소드	테스트 목적	결과
testGameInitialization()	게임 초기 상태 확인 (턴, 승자 없음, 윷 미던짐 등)	PASS
testHandleThrowResult()	윷 던진 결과 저장 및 재던지기 여부 확인	PASS
testMovePiece()	윷 결과 기반으로 말 이동 가능 여부, 위치	PASS

테스트 메소드	테스트 목적	결과
	이동 확인	
testEndTurn()	턴 종료시 다음 플레이어로 전환되는지 확인	PASS
testSelectRoll()	윷 결과 선택 기능 및 허용되지 않는 결과 거부 동작 확인	PASS
testCanMove()	윷 던지기 전/후 말 이동 가능 여부 및 상대 말 접근 제한 확인	PASS
testDiagonalPathMovement()	대각선 경로 이동 기능 검증	PASS
testGameOver()	모든 말 완주 후 게임 종료 및 승자 설정 기능 확인	PASS

GameTest 클래스 테스트

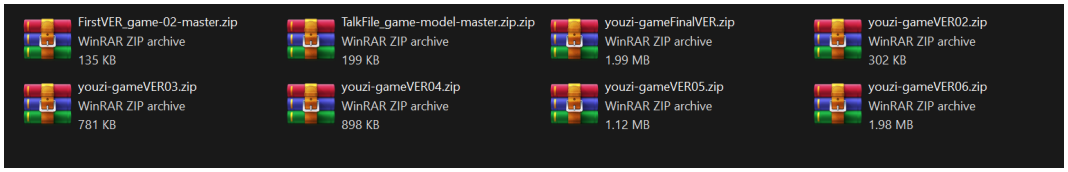
테스트 메소드	테스트 목적	결과
testInitialization()	주요 보드 위치 초기화 여부 및 존재 여부 확인	PASS
testMainPathMovement()	주 경로 이동 테스트 (예: -1 → 1, 1 → 3 등)	PASS
testBackwardMovement()	백도 시 뒤로 이동 로직 및 경계값 동작 확인	PASS
testJunctionPoints()	교차점 여부 판별 기능 확인	PASS
testDiagonalPathMovement()	대각선 경로 이동 정확성 확인	PASS
testCenterPosition()	중앙 위치(23 번) 도달 여부 테스트	PASS
testPathTypeSelection()	교차점에서 가능한 경로 유형 반환 확인	PASS
testCompletingCircuit()	원형 경로 완주 로직 및 마지막 점 도달 후 재시작 여부 확인	PASS

F. GitHub 프로젝트 리포트

i. GitHub 주소

[TeamProject17_YutNori-Game]

ii. 프로젝트 progress history 스크린샷



iii. 팀원별 기여도

팀원 이름: **희건춘**
담당 역할: 프로젝트 총괄 / 코드 리뷰
상세 기여:
프로젝트 설계 방향 결정 및 일정 조율
모델 및 컨트롤러 설계 참여
Git 커밋 리뷰 및 병합 관리
사용자 인터페이스(UI) 개발

팀원 이름: 향원원
담당 역할: 설계 다이어그램 작성 담당
상세 기여:
문서 & PPT 제작
Diagram 피드백

팀원 이름: 배민웅
담당 역할: 문서 정리 및 보고서 편집
상세 기여:
문서 & PPT 제작
UI 색상 견의
테스트 및 diagram 편집

팀원 이름: 유민성
담당 역할: 문서 작성 및 테스트
상세 기여:
use case diagram 작성
use case text 작성
프로그램 테스트 및 피드백

팀원 이름: 위종인
담당 역할: 문서 작성
상세 기여:
-Operation Contarct 작성
-데모 비디오 녹화