



0507...첫 회의...

▼ Use Cases 정의

<플레이어가 4개의 윷을 랜덤하게 던지고 결과에 따라 말을 이동시키며, 가장 먼저 모든 말을 도착시키면 승리하는 윷놀이 게임>

UC1: 게임 설정 및 시작

- Step1: 플레이어가 플레이어 수, 말 수, 보드 형태를 선택한다.
- Step2: 플레이어가 "게임 시작" 버튼을 클릭한다.
- Step3: 시스템이 입력값을 검증하고 Game, Board, Player 객체를 초기화한 후 첫 번째 턴을 할당한다.

UC2: 랜덤 윷 던지기

- Step1: 플레이어가 "윷 던지기" 버튼을 클릭한다.
- Step2: 시스템이 윷 결과(뽕도, 도, 개, 걸, 윷, 모) 중 하나를 랜덤으로 생성한다.
- Step3: 시스템이 결과를 표시하고 말 선택을 활성화한다.

UC3: 지정 윷 결과 설정

- Step1: 플레이어가 "윷 결과 지정"을 클릭한다.
- Step2: 플레이어가 테스트용으로 원하는 결과를 선택한다.
- Step3: 시스템이 선택된 결과를 표시하고 말 선택을 활성화한다.

UC4: 말 선택 및 이동

- Step1: 플레이어가 이동할 말을 선택한다.
- Step2: 시스템이 윷 결과에 따라 목표 칸을 계산한다.
- Step3: 시스템이 해당 칸으로 말을 이동시키고 보드를 갱신한다.

UC5: 말 집단화

- Step1: 이동 후, 시스템이 동일 플레이어의 말이 두 개 이상 같은 칸에 있는지 감지한다.
- Step2: 시스템이 말 그룹 객체를 생성하고 해당 말들을 연결한다.
- Step3: 그룹화된 말이 이후 턴에 함께 이동하도록 한다.

UC6: 상대 말 포획

- Step1: 이동 후, 시스템이 목표 칸에 상대 플레이어의 말이 있는지 감지한다.
- Step2: 시스템이 해당 상대 말을 제거하고 시작 위치로 돌려보낸다.
- Step3: 시스템이 보드를 갱신하여 변화를 반영한다.

UC7: 승리 판정

- Step1: 턴 종료 시 시스템이 각 플레이어의 모든 말이 도착했는지 확인한다.
- Step2: 조건을 만족하는 플레이어를 승자로 선언한다.
- Step3: 시스템이 게임 종료 화면을 표시하고 재시작/종료 옵션을 제공한다.

UC8: 게임 재시작 및 종료

- Step1: 플레이어가 "재시작" 또는 "종료"를 선택한다.
- Step2: 시스템이 게임 상태를 재초기화하거나 애플리케이션을 종료한다.
- Step3: 시스템이 선택된 작업을 확인하고 메인 메뉴로 돌아가거나 애플리케이션을 닫는다.

김세진 김나린 주예린

▼ Domain model

• 게임(Game)

◦ 속성

- `players: List<Player>` : 플레이어 담는 배열
- `board: Board` : 게임 판 하나
- `currentPlayer: Player` : 현재 플레이어
- `status: GameStatus {READY, IN_PROGRESS, FINISHED}` : 게임의 현재 상태 (준비, 게임 중, 끝)

◦ 책임

- `start(config: GameConfig)` : 게임 초기화
- `nextTurn()` : 다음 플레이어에게 턴 전환
- `checkVictory(): Player?` : 승리 조건 확인 및 승자 반환

• 게임설정(GameConfig)

◦ 속성

- `numPlayers: int` : 플레이어 수(몇명인지) 인듯
- `piecesPerPlayer: int` : 플레이어의 말
- `boardShape: BoardShape {SQUARE, PENTAGON, HEXAGON}` : 게임 판 모양 (사각, 오각 육각)

- 책임
 - `validate(): Boolean` : 설정값 검증
- 플레이어(Player)
 - 속성
 - `id: String` : 플레이어 고유 식별 ID
 - `name: String` : 플레이어 이름
 - `pieces: List<Piece>` : 플레이어의 말 배열(플레이어 당 4개의 말을 보통 가지고 있으니까?)
 - 책임
 - `requestThrow(): YutThrow` : 윷던지기 request보내는 듯
 - `selectPiece(result: YutThrow): Piece` : 윷던지기를 어느 말에 적용할지 고름
- 윷던지기(YutThrow)
 - 속성
 - `result: ThrowResult {BACK_DO, DO, GAE, GEOL, YUT, MO}` : 윷던지기 결과 (뱃도, 도, 개, 걸, 윷, 모)
 - 책임
 - `generateRandom()` : 랜덤 결과 만들어내기
 - `setResult(r: ThrowResult)` : 나온 랜덤 결과를 set함.(사용자의 말에게 저장시켜 주는 거겠지?)
- 보드(Board)
 - 속성
 - `cells: List<Cell>` : cell들의 집합
 - `shape: BoardShape` : 보드의 모양 (4,5,6)
 - 책임
 - `getNextCell(from: Cell, steps: int): Cell`
 - `initialize(shape: BoardShape)` : 보드판 초기화
- 칸(Cell)
 - 속성

- `id: String` : 칸 고유식별자?
- `nextCells: List<Cell>` 이동 가능한 cell
- `occupants: List<Piece>` 하나의 cell에 있을 수 있는 (같은?)팀의 말 개수 (같은 팀 말이 한 cell에 있으면 하나로 취급??? 맞나)
- 책임
 - `addPiece(p: Piece)` : 말이 업히는 거
 - `removePiece(p: Piece)` : 말이 잡히는 거
- 말(Piece)
 - 속성
 - `id: String` : 말 고유 ID
 - `owner: Player` : 이 말을 가진 플레이어
 - `position: Cell` : 말의 현재 위치(현재 cell)
 - `grouped: Boolean` : 업힌거??그래서 group?인듯?
 - 책임
 - `moveTo(dest: Cell)` : 어디로 움직일건지 (어느 cell로)
 - `joinGroup(g: PieceGroup)` : 말 업는 기능 인듯
- 말그룹(PieceGroup)
 - 속성
 - `members: List<Piece>` : 같은 팀의 말이 한 cell에 있으면 말 그룹으로 취급
 - 책임
 - `moveGroup(steps: int)` : 같은 말 끼리 함께 움직이는 것
- 이동액션(MoveAction)
 - 속성
 - `piece: Piece` : 이동 액션 대상 말
 - `fromCell: Cell` : 현재 cell
 - `toCell: Cell` : 이동할 목적지 cell
 - `throwResult: YutThrow` : 윷던진 결과 저장된 곳
 - 책임

- `execute()` : 말 이동시키기
 - `applyRules()` : 말 업기, 말 잡기, 승리판정에 대한 룰..의 적용?
- 룰엔진(RuleEngine)
 - 책임
 - `applyGrouping(cell: Cell)` : 한 칸에 같은 플레이어의 말 2개 이상 모였을 때 PieceGroup으로 묶기
 - `applyCapture(cell: Cell)` : 상대방 말 잡기
 - `applyVictoryCheck(game: Game)` : 매 턴이 끝난 뒤 각 플레이어가 모든 말을 도착지점에 올려놓았는지 검사 후 승패 결정

클래스 간 주요 연관관계

- **Game 1 — * Player**
- **Game 1 — 1 Board**
- **Board 1 — * Cell**
- **Cell 1 — * Piece**
- **Player 1 — * Piece**
- **Piece 0..1 — 1 PieceGroup**
- **MoveAction * — 1 Piece, 1 YutThrow**
- **RuleEngine** — 의존 — 모든 도메인 클래스

▼ □ ○ □ □ □ □ □ □ □ □ □

참여자의 말

- field
 - 말의 현재 옷 결과
 - 옷 결과 종류
 - int 뺑도 = -1;
 - int 도 = 1;
 - int 개 = 2;

- int 걸 = 3;
- int 윷 = 4;
- int 모 = 5;

- method

-

윷놀이 판

- 사각형
- 오각형
- 육각형

윷놀이 진행시키는 관리자? 뭐 그런 거 같은거;;

- field
- method
 - 게임 시작

사용자 말 관리

- field
- method

▼ 번개 게임;;; 실전으로 깨달은 것들ㄷㄷㄷㄷ

같은 유저가 보드판에 이미 있는 게임말을 옮길건지, 아니면 새로운 말을 옮길건지 선택해야 함 → 공식적인 윷놀이 규칙 맞나여ㄷㄷㄷㄷ

- 지 패키지 구조

```
yutnori/
└─ model/
```

```

|   |— Game.java
|   |— Player.java
|   |— Piece.java
|   |— Board.java
|   |— YutThrow.java
|— ui/
|   |— MainFrame.java // Swing 기반 화면
|   |— GamePanel.java
|— controller/
|   |— GameController.java

```

GameController 역할

- 버튼 클릭 등 UI 이벤트를 받아 Game 객체에 전달
- 말 이동, 턴 전환, 윷 결과 등 관리

유지보수를 위한 팁

- UI(View)에는 로직을 절대 넣지 말고, 모두 `GameController` 를 통해 처리하게 해.
- Board와 말 위치는 int index로 표현하거나 `Position` 클래스로 일반화하면, 다른 보드에 서도 재사용 가능해.

필요한 오브젝트 (클래스)



Game

- 게임 전체를 관리하는 클래스
- 턴 관리, 플레이어 관리, 윷 던지기 수행



Player

- 각 팀(플레이어)을 나타냄
- 보유 말 목록 관리
- 플레이어 이름, 차례 여부 등 포함



Piece

- 실제 말(토큰) 하나
- 위치, 소속 플레이어, 합류 여부 등을 포함



YutThrow

- 윷 던지기 결과 계산
- 결과 이름 및 이동 칸 수 반환



Board

- 말의 경로와 칸들을 정의
- 각 칸에 어떤 말이 있는지 저장



Position

- 경로 내 위치를 식별하기 위한 구조
- 일반 경로, 중앙 경로, 지름길 등 분기점 구분 가능