|  |
| --- |
| **Term Project: 윷놀이** |

- 소프트웨어공학 -

시계, 폰트, 그래픽, 상징이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

|  |  |
| --- | --- |
| 과목명 | 소프트웨어공학 |
| 교수명 | 이찬근 |
| 제출일 | 2025.06.04. |
| Team | Team 4 |
| 팀 원 | 이다빈, 안선우, 민진설, 이레뒤, 정도진 |

**A. 프로젝트 개요**

본 프로젝트는 한국의 전통 보드게임인 '윷놀이'를 소프트웨어로 개발하는 것을 목표로 합니다. 객체지향 분석 및 설계(OOAD) 기법을 적용하여, 유스케이스 모델링, 클래스 설계, MVC 아키텍처 구성 등 수업에서 학습한 내용을 실제 프로젝트에 구현하는 데 중점을 두었습니다. 개발 언어는 Java를 사용했으며, UI 툴킷으로는 Java Swing을 필수로 하고 JavaFX로 추가 UI를 제작하여 UI 계층의 교체 가능성을 실현하고자 했습니다. 버전 관리는 GitHub, 테스트는 JUnit을 사용했습니다.

주요 기능은 다음과 같습니다:

* 2~4명의 플레이어 수 및 2~5개의 말 수 설정 기능
* 랜덤 및 지정 윷 던지기 기능
* 말 이동, 잡기, 업기 기능
* 모든 말을 먼저 완주시키는 승리 조건
* 게임 재시작 및 종료 기능
* UI 툴킷 교체가 용이한 구조

**B. 유스케이스 모델**

프로젝트의 요구사항은 유스케이스 다이어그램과 각 유스케이스에 대한 설명을 통해 정의되었습니다.

텍스트, 도표, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**주요 유스케이스 설명**

|  |  |  |  |
| --- | --- | --- | --- |
| **유스케이스 이름** | **액터** | **목적** | **기본 흐름** |
| **UC1. 게임 설정** | 사용자 (Player) | 플레이어 수와 말 수를 선택하여 게임 시작 조건을 설정합니다. | 1. 사용자가 플레이어 수(2~4명)와 말 수(2~5개를 선택합니다. 2. "시작" 버튼을 클릭하면 GameController가 PlayConfig 값을 읽어 게임을 초기화합니다. 3. GameState와 YutBoard가 초기화되며 게임이 시작됩니다. |
| **UC2. 윷 던지기** | 사용자 (Player) | 자신의 차례에 윷을 던져 이동할 수를 결정합니다. | 1. 사용자가 "윷 던지기" 버튼을 클릭합니다. 2. Controller가 윷 결과를 생성하고 GameState에 저장합니다. 3. Controller는 View에 윷 결과 표시를 요청합니다. |
| **UC3. 말 선택 및 이동** | 사용자 (Player) | 윷 결과에 따라 이동시킬 말을 선택하고 이동시킵니다. | 1. 사용자가 보드 위의 말을 선택합니다. 2. Controller는 해당 위치의 말이 유효한지 확인하고, Player.move()를 호출하여 말을 이동시킵니다. 3. 이동 후 업기 또는 골인 여부를 판단하며, View에 말 위치 변경을 요청합니다. |
| **UC4. 말 업기** | 시스템 | 같은 위치에 있는 아군 말을 하나의 말로 묶어 함께 이동시킵니다. | 1. 말 이동 후, 같은 위치에 있는 아군 말을 탐색합니다. 2. 조건이 충족되면 말을 그룹으로 묶습니다. |
| **UC5. 말 잡기** | 시스템 | 이동된 말이 상대방 말과 위치가 겹칠 경우 상대 말을 판에서 제거합니다. | 1. 말 이동 후, 같은 위치에 있는 상대 말을 탐색합니다. 2. 상대 말을 잡으면, 해당 말은 시작점으로 돌아가고 잡은 플레이어는 한 번 더 윷을 던질 기회를 얻습니다. |
| **UC6. 승리 판정** | 시스템 | 한 플레이어가 모든 말을 완주시키면 승리 상태를 표시합니다. | 1. 말이 골인될 때마다 GameState의 점수 상태가 업데이트됩니다. 2. 특정 플레이어의 점수가 총 말 수와 같아지면 승리로 처리하고 게임을 종료합니다. |

**C. 기타 요구사항 (Operation Contracts, SSD 등)**

**Operation Contracts**

오퍼레이션: startGame(playerCount, pieceCount)

* Responsibilities: 사용자로부터 입력받은 설정으로 게임의 상태와 보드를 초기화합니다.
* 관련 유스케이스 (Cross Reference): UC1. 게임 설정
* Preconditions
  + 시스템이 초기 상태이며, 설정 화면이 활성화되어 있어야 합니다.
* Postconditions
  + PlayConfig 객체에 플레이어 수와 말 수가 설정됩니다.
  + GameState 인스턴스가 생성되고, 플레이어 수만큼 Player 객체가 생성됩니다.
  + 게임 상태가 '초기화 완료' 상태가 됩니다.
  + 게임 보드가 사용자에게 표시됩니다.

오퍼레이션: throwYut()

* Responsibilities: 윷 던지기 결과를 생성하고 게임 상태에 기록하며, 결과를 사용자에게 표시합니다.
* 관련 유스케이스 (Cross Reference): UC2. 윷 던지기
* Preconditions
  + 게임이 시작되어 있고, 현재 사용자의 턴이어야 합니다.
  + 게임 단계(Phase)가 'THROW' 상태여야 합니다.
* Postconditions
  + 이동할 수(도, 개, 걸, 윷, 모, 빽도)가 결정되어 GameState의 lastThrow 리스트에 추가됩니다.
  + 결정된 결과가 사용자에게 표시됩니다.
  + 게임 단계가 'MOVE'로 전환될 수 있습니다.
  + '윷'이나 '모'가 나온 경우, throwCount가 증가하여 한 번 더 던질 기회가 부여됩니다.

오퍼레이션: selectPiece(pathIndex, stepIndex)

* Responsibilities: 선택된 말을 윷 결과에 따라 이동시키고, 잡기(Capture) 또는 업기(Grouping)와 같은 게임 규칙을 적용합니다.
* 관련 유스케이스 (Cross Reference): UC3. 말 이동, UC4. 말 업기, UC5. 말 잡기
* Preconditions
  + throwYut 오퍼레이션이 호출된 후, 이동할 수 있는 윷 결과가 하나 이상 존재해야 합니다.
  + pathIndex, stepIndex 위치에 현재 플레이어의 말이 존재해야 합니다.
* Postconditions
  + 선택된 Piece 객체의 위치 정보(pathIndex, stepIndex)가 갱신됩니다.
  + (말 잡기 발생 시) 이동한 위치에 있던 상대방 Piece는 pathIndex가 -1로 변경되어 시작점으로 돌아가고, 현재 플레이어의 throwCount가 1 증가합니다.
  + (말 업기 발생 시) 이동한 위치에 있던 아군 Piece들의 isGrouped 상태가 true로 변경됩니다.
  + (완주 발생 시) Piece의 isFinished 상태가 true로 변경되고, 해당 플레이어의 점수가 오릅니다.
  + GameState의 lastThrow 리스트에서 사용된 윷 결과가 제거됩니다.
  + 모든 윷 결과를 사용했다면, 다음 플레이어의 턴으로 넘어가거나 추가 턴이 없다면 턴이 종료됩니다.

오퍼레이션: restartGame()

* Responsibilities: 현재 게임을 종료하고, 초기 설정 화면으로 되돌아가 새로운 게임을 시작할 수 있도록 준비합니다.
* 관련 유스케이스 (Cross Reference): UC6. 승리 판정 (확장 관계인 '게임 재시작')
* Preconditions
  + 게임이 종료된 상태(한 플레이어가 승리한 상태)여야 합니다.
* Postconditions
  + 현재 게임 화면(YutBoard)이 닫힙니다.
  + 초기 설정 화면(FirstPage)이 사용자에게 표시됩니다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**SSD**

**D. 설계 및 구현 리포트**

본 프로젝트는 **MVC 아키텍처**를 기반으로 설계되었습니다. 이는 UI(View)와 비즈니스 로직(Model)을 분리하여 유지보수성과 확장성을 높였습니다.

**클래스 구조 (MVC 기반)**

텍스트, 도표, 라인, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **Model**: 게임의 데이터와 비즈니스 로직을 담당합니다. UI나 컨트롤러에 대해 알지 못합니다.
  + PlayConfig: 플레이어 수, 말 수, 보드 종류 등 게임 초기 설정값을 저장합니다.
  + PathConfig: 사각형, 오각형 등 보드 형태에 따른 경로 정보를 관리합니다.
  + GameState: 현재 턴, 플레이어 정보, 윷 던지기 결과 등 게임의 현재 상태를 총괄합니다.
  + Player: 플레이어의 ID, 보유한 말 리스트 등의 정보를 가집니다.
  + Piece: 각 말의 위치, 완주 여부, 업힘 상태 등을 관리합니다.
  + Yut: 윷 던지기 결과를 생성하는 로직을 포함합니다.
* **View**: 사용자 인터페이스를 담당하며, 게임 상태를 시각적으로 표현합니다.
  + GameView (Interface): updateBoard, showWinner 등 UI가 구현해야 할 메소드를 정의합니다. 이를 통해 Controller와 View 간의 결합도를 낮춥니다.
  + FirstPage: 게임 시작 전 플레이어 수, 말 개수 등을 설정하는 초기 화면입니다. (Swing 및 JavaFX 버전으로 각각 구현됨)
  + YutBoard: 실제 윷놀이 게임이 진행되는 메인 보드 화면입니다. (Swing 및 JavaFX 버전으로 각각 구현됨)
* **Controller**: 사용자의 입력을 받아 Model의 상태를 변경하고, 변경된 내용을 View에 업데이트하도록 지시합니다.
  + GameController: "윷 던지기", "말 선택"과 같은 사용자 이벤트에 따라 GameState를 변경하고, GameView 인터페이스를 통해 UI 갱신을 요청합니다.
  + PlayerAdapter, PieceAdapter: FirstPage에서 사용자의 선택을 PlayConfig 모델에 반영하는 어댑터 클래스입니다. (Swing UI에 사용됨)

**Sequence Diagram**

**1. 게임 설정**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**2. 윷 던지기**

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**3. 말 선택 및 이동**

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**UI 교체 용이성**

이 프로젝트의 설계는 **UI 교체 용이성**을 핵심 원칙으로 삼았습니다. 이는 MVC 아키텍처를 통해 달성되었습니다.

* **GameView 인터페이스**: Swing UI와 JavaFX UI는 모두 GameView 인터페이스를 구현합니다. 이 인터페이스는 updateBoard(GameState state), showWinner(Player winner) 등 UI가 수행해야 할 동작을 추상 메소드로 정의합니다.
* **Loose Coupling:** GameController는 구체적인 UI 클래스를 직접 참조하지 않고, GameView 인터페이스에만 의존합니다.
* **UI 교체 방법**: 따라서 UI를 Swing에서 JavaFX로 교체할 때, Model(GameState, Player 등)이나 GameController의 코드는 전혀 수정할 필요 없이 GameController를 생성할 때 GameView 구현체만 바꾸어 주입하면 됩니다.
  + **Swing 사용 시:** new GameController(config, names, new YutBoard(config, names));
  + **JavaFX 사용 시:** new GameController(config, names, new YutBoard(config, names, stage));

이처럼 비즈니스 로직과 UI가 분리되어 있기 때문에, 새로운 UI 툴킷을 추가하더라도 GameView 인터페이스만 구현하면 기존 로직을 그대로 재사용하여 쉽게 확장할 수 있습니다.

**E. 테스트 리포트**

JUnit을 사용하여 게임의 핵심 로직을 테스트했습니다. junitTest.java 파일에 주요 테스트 케이스가 정의되어 있습니다.

|  |  |  |
| --- | --- | --- |
| **테스트 케이스** | **목적** | **내용 및 검증** |
| **playerInfoTest** | 플레이어 정보 초기화 확인 | GameState 생성 후, 각 플레이어의 ID와 보유한 말의 개수가 PlayConfig에 설정된 값(4개)과 일치하는지 assertEquals로 검증합니다. |
| **pieceMoveShortcutTest1** | 말 이동 및 첫 번째 지름길 진입 | '윷'(4칸)과 '도'(1칸)를 던진 후, 말이 외곽 경로를 따라 4칸, 1칸을 순서대로 이동하여 첫 번째 분기점(P0, S5)에 정확히 도달하는지 assertEquals로 검증합니다. |
| **pieceMoveShortcutTest2** | 말 이동 및 두 번째 지름길 진입 | '윷', '윷', '개'를 던져 외곽 경로를 10칸 이동시킨 후, 말이 두 번째 분기점(P0, S10)에 정확히 도달하는지 assertEquals로 검증합니다. |
| **pieceGroupTest** | 말 업기 기능 | 한 말이 특정 위치(P0, S2)에 있을 때, 다른 말이 같은 위치로 이동하면 두 말의 isGrouped()가 true가 되는지 assertTrue로 검증합니다. |
| **pieceCatchTest** | 말 잡기 기능 | 내 말이 상대방 말이 있는 위치(P0, S2)로 이동했을 때, 상대방 말의 위치가 시작점(-1)으로 리셋되고, 나의 throwCount가 1 증가하는지 assertEquals로 검증합니다. CAPTURE\_OCCURRED 이벤트가 발생했는지도 확인합니다. |
| **pieceFinishAndPointTest** | 말 완주 및 점수 획득 | 완주 직전(P0, S18)에 있는 말이 '개'(2칸)만큼 이동했을 때, 해당 말의 isFinished() 상태가 true가 되고, 플레이어의 완주한 말 개수가 1이 되는지 검증합니다. |

**Junit test 결과**

텍스트, 스크린샷, 소프트웨어, 컴퓨터이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**F. GitHub 프로젝트 리포트**

**i. 팀의 github 프로젝트 주소**

<https://github.com/CAU-SoftwareEngineering/YutBoardGame.git>

**ii. 프로젝트 progress history 스크린샷**

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**iii. 팀원 별 역할**

**이다빈: 전반적인 코드 구현 및 설계, junit test, 보고서 작성**

**안선우: 코드 구현 보조, 영상제작**