

2025 SE Term Project (윷놀이) Final Report

소프트웨어공학 01분반 Team 2

이름	학번
정인혁	20215431
권재민	20214521
김진하	20210623
이민규	20215143
최정우	20216620

목차

1. 프로젝트 개요

2. 요구 사항 분석

3. 설계 및 구현

4. 테스트

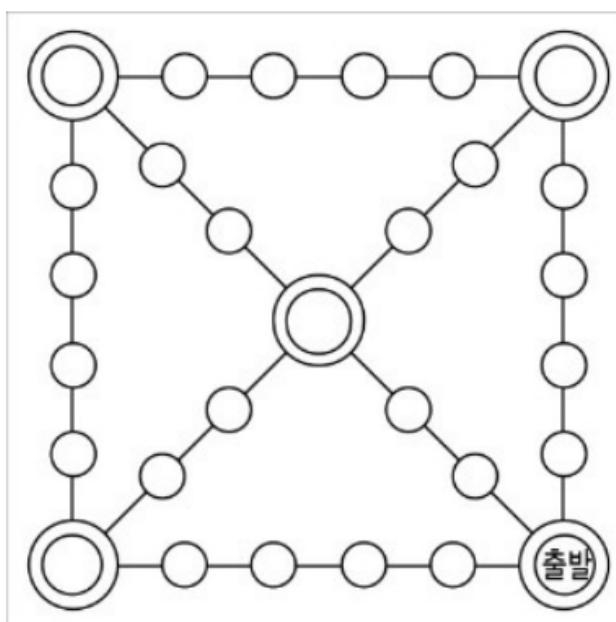
5. GitHub 프로젝트

1. 프로젝트 개요

해당 프로젝트는 윷놀이를 OOAD 기법, MVC Architecture 패턴을 사용하여 소프트웨어로 개발하는 프로젝트이다. 주요 기능은 다음과 같다.

구현 기능

- 게임 시작 시 참여자의 명수 (최소 2명, 최대 4명)와 게임 말 갯수 (최소 2개, 최대 5개)를 지정할 수 있다.
 - 참여자가 늘어도, 개인전으로 진행. 즉 4명이 참여하면 말의 종류는 4가지가 됨



- 표준 윷놀이 판은 다음 그림과 같게 하며, 각 참여자의 말의 현재 위치가 표시되어야 한다.
- 각 턴의 진행을 위해 <랜덤 윷 던지기> 버튼과 <지정 윷 던지기> 버튼이 표시된다.

- <지정 윷 던지기> 버튼을 누르면 빼도, 도, 개, 걸, 윷, 모 중 선택한 결과로 진행된다. (테스트용)
- <랜덤 윷 던지기> 버튼을 누르면 랜덤한 윷의 결과가 나타난다.
- 사용자는 윷 던지기 결과를 적용할 게임 말을 선택할 수 있으며, 그에 따라 진행이 자동으로 되어야 한다. 단, 사용자의 선택이 필요한 순간에는 사용자에게 선택권을 주어야 한다. (e.g., 개 위치에 말이 있는 상황에서, 윷을 던졌는데 모가 나오고 잇달아 걸이 나오면, 어떤 말에 모/걸을 적용할지 판단 의뢰)
- 게임 말을 업는 (grouping) 기능을 지원해야 한다.
- 다른 사용자의 말을 잡는 기능을 지원해야 한다.
- 게임 말들이 출발에서 시작해서 먼저 모든 말을 내보내는 팀이 게임에서 승리하며, 이때 어느 팀이 승리했는지 표시한다.
- 한 게임이 끝났을 때 게임 재시작 혹은 종료가 가능해야 한다.

주요 사항

- OOAD 기법을 적극적으로 사용하며, 그 결과를 문서화해야 한다.
- MVC Architecture 패턴을 사용하여 UI와 Model을 분리하여 구현해야 한다.
- 두 개 이상의 UIToolKit을 이용해서 두 개 이상의 UI를 구현한다. 이때 UI를 제외한 나머지 코드들이 거의 수정 없이 재사용 되는 것을 보여야 한다.
- 테스트 용이한 설계를 하여 JUnit으로 모델 테스트를 수행한다.
- 윷놀이 판을 customize 할 수 있어야 한다.
 - 사각형, 오각형, 육각형까지 선택
 - 오각형, 육각형일 때 path 설정 규칙 : 중심점에서 멈추는 경우 종료지점과 가장 가까운 path로 진행하고, 그렇지 않은 경우에는 종료지점과 두 번째로 가까운 path로 진행한다.

개발 프로세스

윷놀이 소프트웨어를 체계적으로 개발하기 위하여 우리는 수업 시간에 배운 **Waterfall model**과 **Agile model**을 고려하였다. 각 process의 장단점, 해당 model의 적용이

타당한 이유는 다음과 같다.

- **Waterfall model**

- Advantages

- 개념적으로 단순하며, 주어진 문제를 서로 다른 phases로 분명하게 나눌 수 있다.
 - 각 phase가 이정표 역할을 하여 관리하기 쉽다.

- Disadvantages

- 요구 사항이 이른 시점에 결정된다.
 - 모든 개발이 끝나야 결과를 알 수 있다.
 - 사용되지 않는 요구 사항이 구현될 수 있다.

- The reason why it was reasonable

- 윗줄이에 대한 정보는 과거에서부터 현재까지 꾸준히 누적되어 왔으며, 해당 정보를 인터넷에서 쉽게 얻을 수 있다.
 - 윗줄이 개발에 대한 요구 사항이 변화하기 않기 때문에 Waterfall model 을 사용했을 때 효율적인 개발이 가능하다.

- **Agile model**

- Advantages

- 위험성이 낮음.
 - 수정에 대한 유연성을 가지고 있음.

- Disadvantages

- 설계가 최선이 아닐 수 있음.
 - 재작업이 증가하여 total cost가 커질 수 있음.

- The reason why it was reasonable

- 요구 사항 분석, 디자인의 오류 및 수정 사항이 발생했을 때 빠른 대응이 가능하다.

이를 고려한 결과 최종적으로 선택한 개발 프로세스는 **Agile model**이다. 요구 사항 분석 단계에서 우리가 놓치는 예외 사항이 발생할 가능성을 배제할 수 없으며, 오각형 및 육각형 게임판에 대한 구현은 새롭게 만들어야 하는 부분이기 때문에 점진적 개발이 유리하다고 판단하였기 때문이다.

2. 요구 사항 분석

요구 사항을 설립하기 위해 Use case를 다음과 같이 정리하였다.

UC1 : 게임 시작

- **Scope** : Yutnori Game Application
- **Level** : User Goal
- **Actors**
 - **Primary Actor** : Player
 - **Offstage Actor** : Game System
- **Stakeholders and Interests**
 - **Player** : 입력한 게임 정보에 따라 게임이 시작되기를 원함
 - **Game System** : Player가 입력한 정보에 맞는 게임 시작
- **Success Guarantee**
 - Player가 입력한 정보에 맞는 게임이 시작됨
- **Main Success Scenario**
 1. Player가 프로그램을 실행한다.
 2. Game System이 게임 정보 입력을 위한 창을 띄운다.
 3. Player는 게임 시작을 위한 정보를 차례로 입력한다.
 - a. 게임판 종류 (사각형, 오각형, 육각형)
 - b. 참여 인원 (2~4)

c. 1인당 말 개수 (2~5)

4. Game System은 입력된 정보를 바탕으로 게임판을 초기화하고 디스플레이함.

UC2 : 웃 던지기

- **Scope** : Yutnori Game Application
- **Level** : User Goal
- **Actors**
 - **Primary Actor** : Player
 - **Offstage Actor** : Game System
- **Stakeholders and Interests**
 - **Player** : 현재 자신의 말이 최적의 경로로 이동할 수 있는 결과가 나오기를 원함.
 - **Game System** : 주어진 확률에 따라 Player가 던진 웃 결과를 제공하기를 원함.
- **Preconditions**
 - Player가 게임 시작을 위한 정보를 성공적으로 입력하고 Game System이 해당 정보에 맞는 게임판을 디스플레이함
- **Success Guarantee**
 - Player가 던진 웃 결과를 성공적으로 저장함.
- **Main Success Scenario**
 1. Player가 웃을 던진다.
 2. Game System은 해당 결과를 표시 및 저장한다.
 3. 웃 또는 모가 나온 경우, Player에게 추가 턴을 부여하기 위해 1로 돌아간다.

UC3 : 말 이동하기

- **Scope** : Yutnori Game Application
- **Level** : User Goal
- **Actors**
 - **Primary Actor** : Player
 - **Offstage Actor** : Game System
- **Stakeholders and Interests**
 - **Player** : 이동하고자 하는 말에 자신이 던진 윷 결과가 정확히 반영되기를 원함
 - **Game System** : Player가 희망하는 대로 말을 이동하기를 원함
- **Preconditions**
 - Player가 던진 윷 결과를 저장하고 있어야 함
- **Success Guarantee**
 - Game System이 Player가 선택한 이동 결과를 반영하여 게임판을 업데이트함
- **Main Success Scenario**
 1. Game System은 저장한 결과를 디스플레이한다.
 2. Player는 이동할 칸 수, 이동할 말을 선택한다.
 3. Game System은 현재 게임판의 종류 (사각형, 오각형, 육각형 중 1)을 확인하고 그에 맞는 이동 규칙을 적용하여 말을 이동한다.
 4. Game System은 특수 규칙 (업기, 잡기)을 적용하여 게임판을 업데이트한다.
 5. Player가 추가로 말을 이동할 수 있는 경우 (저장된 이동 결과가 남아 있는 경우) 2로 이동한다.

UC4 : 이동 규칙 (사각형) 적용하기

- **Scope** : Yutnori Game Application
- **Level** : Subfunction
- **Actors**
 - **Primary Actor** : Game System
 - **Offstage Actor** : Player
- **Stakeholders and Interests**
 - **Player** : 룰이 공정하게 적용되기를 원함
 - **Game System** : 이동 규칙을 정확히 처리해야 함
- **Preconditions**
 - Player는 이동할 말, 이동할 칸 수를 모두 선택함
- **Success Guarantee**
 - 이동 규칙을 적용하여 Player가 이동하고자 한 말의 현재 위치를 변경함
- **Main Success Scenario**
 1. Game System은 Player가 선택한 이동 칸 수, 이동하고자 하는 말의 현재 위치를 확인한다.
 2. 아래 규칙을 차례로 적용하여 해당하는 경우의 맞게 Player의 말을 이동한다.
 - a. 빽도의 경우 이전에 해당 말이 있었던 방향으로 되돌아간다
 - b. 그 외의 경우
 - i. 게임판에 존재하지 않는 경우 대각선이 아닌 외곽을 따라 이동한다.
 - ii. 시작점, 꼭짓점을 제외한 외곽에서 출발하는 경우 외곽을 따라 도착 지점에 가까워지는 방향으로 이동한다.
 - iii. 상단 꼭짓점에서 출발하는 경우 대각선 방향으로 이동한다. 단 좌측 하단 꼭짓점에서 이동하는 경우 외곽을 따라 이동한다.
 - iv. 중심점에서 출발하는 경우 도착 지점을 향해 이동한다.

v. 중심점을 제외한 내부에서 출발하는 경우 대각선 방향으로 이동한다.

- **Alternative Scenarios**

- 2aa. Player가 이동하고자 하는 말이 게임판에 존재하지 않는 경우 말이 이동하지 않는다
- 2ab. 해당 말이 게임판에 올라갔다가 빼도로 시작 지점에서 도달한 상태에서 빼도 가 나온 경우 완주 처리한다.

UC5 : 이동 규칙 (오각형, 육각형) 적용하기

- **Scope** : Yutnori Game Application

- **Level** : Subfunction

- **Actors**

- **Primary Actor** : Game System

- **Offstage Actor** : Player

- **Stakeholders and Interests**

- **Player** : 룰이 공정하게 적용되기를 원함

- **Game System** : 이동 규칙을 정확히 처리해야 함

- **Preconditions**

- Player는 이동할 말, 이동할 칸 수를 모두 선택함

- **Success Guarantee**

- 이동 규칙을 적용하여 Player가 이동하고자 한 말의 현재 위치를 변경함

- **Main Success Scenario**

1. Game System은 Player가 선택한 이동 칸 수, 이동하고자 하는 말의 현재 위치를 확인한다.

2. 아래 규칙을 차례로 적용하여 해당하는 경우의 맞게 Player의 말을 이동한다.
 - a. 빽도의 경우 이전에 해당 말이 있었던 방향으로 1칸 되돌아간다
 - b. 그 외의 경우
 - i. 게임판에 존재하지 않는 경우 대각선이 아닌 외곽을 따라 이동한다.
 - ii. 출발 지점, 꼭짓점을 제외한 외곽에서 출발하는 경우 외곽을 따라 도착 지점에 가까워지는 방향으로 이동한다.
 - iii. 꼭짓점에서 출발하는 경우 중심점 방향으로 이동한다. 중심점을 지나치는 경우 종료지점과 두 번째로 가까운 경로를 따른다. 단 중심점을 경유하는 것보다 종료 지점을 빠르게 도달하는 경로가 존재하는 경우 해당 경로를 따른다.
 - iv. 중심점에서 출발하는 경우 종료 지점과 가장 가까운 경로를 따라 이동한다.
 - v. 중심점을 제외한 내부에서 출발하는 경우 종료지점과 두 번째로 가까운 경로로 이동한다.

- **Alternative Scenarios**

- 2aa. Player가 이동하고자 하는 말이 게임판에 존재하지 않는 경우 말이 이동하지 않는다.
- 2ab. 해당 말이 게임판에 올라갔다가 빽도로 시작 지점에서 도달한 상태에서 빽도가 나온 경우 완주 처리한다.

UC6 : 이동 후 규칙 (완주, 업기, 잡기) 적용하기

- **Scope** : Yutnori Game Application
- **Level** : Subfunction
- **Actors**
 - **Primary Actor** : Game System
 - **Offstage Actor** : Player
- **Stakeholders and Interests**
 - **Player** : 룰이 공정하게 적용되기를 원함

- **Game System** : 잡기 업기, 추가 턴 등의 룰을 정확히 처리해야 함
- **Preconditions**
 - Player가 말을 이동하였음
- **Success Guarantee**
 - 특수 규칙이 적용되어 Players의 말 위치를 조정한다. 필요한 경우 특정 Player에게 추가 턴을 부여한다.
- **Main Success Scenario**
 1. Game System은 Player가 이동한 말의 위치를 확인한다.
 2. 도착 지점인 경우 Player의 점수를 올리고 해당 말을 게임판에서 삭제한다.
 3. 해당 위치에 상대편 말이 있으면 잡기 (상대편 말을 탈락시키고, 말을 잡은 Player에게 추가 턴 부여)로 처리한다.
 4. 해당 위치에 자신의 말이 있으면 업기 (말을 겹쳐서 함께 움직이는 규칙, 탈락, 이동 등의 행동 공유)로 처리한다.
 5. 탈락한 말은 시스템이 강제로 시작 위치로 이동시킨다.
- **Alternative Scenario**
 - 2a. 도에서 빠도로 도착 지점에 도착한 경우 해당 말을 완주로 처리하지 않는다.

UC7 : 턴 넘기기

- **Scope** : Yutnori Game Application
- **Level** : Subfunction
- **Actors**
 - **Primary Actor** : Game System
 - **Offstage Actor** : Player

- **Stakeholders and Interests**

- **Player** : 턴이 공정하게 바뀌어야 함
- **Game System** : Player의 턴 순서를 정확하게 유지해야 함

- **Preconditions**

- Player가 던진 육 결과를 자신의 말에게 모두 부여하고 더 이상 자신의 턴에서 움직일 수 있는 말이 존재하지 않음

- **Success Guarantee**

- 다음 Player에게 턴이 넘어감

- **Main Success Scenario**

1. Game System은 턴을 다음 Player에게 넘김
 2. Player 턴을 표시하는 상태 메세지를 업데이트한다.
-

UC8 : 승리 조건 확인

- **Scope** : Yutnori Game Application

- **Level** : subfunction

- **Actors**

- **Primary Actor** : Game System
- **Offstage Actor** : Player

- **Stakeholders and Interests**

- **Player** : 시스템이 자신이 승리 조건을 달성했는지 정확하게 판단하는 것을 원함.
- **Game System** : 승리 조건을 달성한 Player를 정확하게 감지함.

- **Preconditions:**

- 특정 Player의 턴이 종료됨.
 - **Success Guarantee**
 - 승리 조건을 달성한 Player가 존재하는 경우 해당 플레이어를 명시하고 게임을 종료함
 - 승리 조건을 달성한 Player가 없는 경우 게임을 계속 진행
 - **Main Success Scenario**
 1. 시스템이 Player의 모든 말이 완주했는지를 확인함.
 2. 특정 Player의 모든 말이 도착 지점에 도달한 경우, 해당 Player를 승자로 지정함.
 3. 모든 말이 도착 지점에 도달한 Player가 존재하지 않는 경우 게임을 계속 진행함.
-

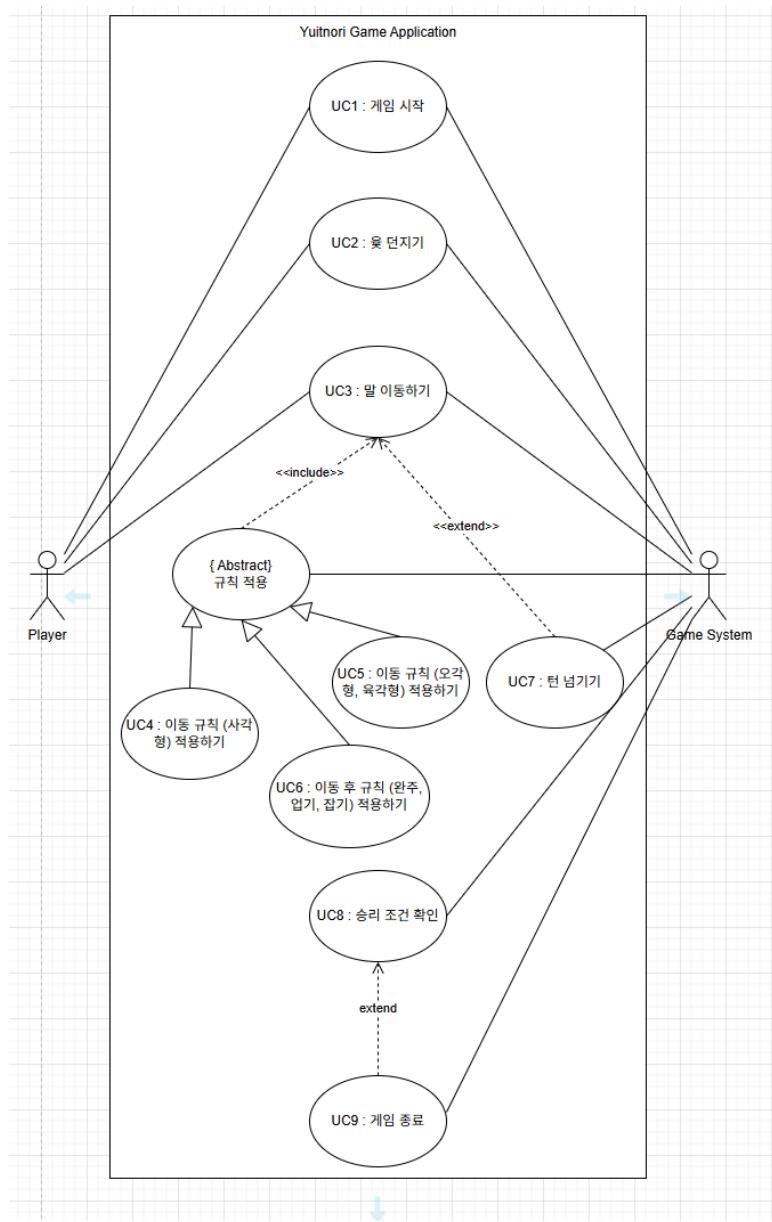
UC9 : 게임 종료

- **Scope :** Yutnori Game Application
- **Level :** user goal
- **Actors**
 - **Primary Actor :** Player
 - **Supporting Actor :** Game System
- **Stakeholders and Interests**
 - Player, Game System : 승리 조건을 달성한 Player가 정확하게 명시되기를 원함
- **Preconditions :**
 - 승리 조건 (모든 말이 게임판을 완주함)을 달성한 Player가 존재함
- **Success Guarantee:**
 - 승리 Player를 정확히 명시하고 게임을 종료함.

- **Main Success Scenario**

1. Game System이 승리 Player를 화면에 표시함
 2. Game System은 게임을 종료하고, Player에게 다시 시작 또는 종료 선택 인터페이스를 제공함
 3. Player는 게임을 다시 시작할지, 종료할지 결정함.
-

Use Case Diagram



Domain Model

윷놀이 소프트웨어에 OOAD 적용을 용이하게 하고 이후 Class diagram 설계를 위해 Domain model을 설계하였다. 우리는 Domain model을 설계하기 위하여 다음 과정을 거쳤다.

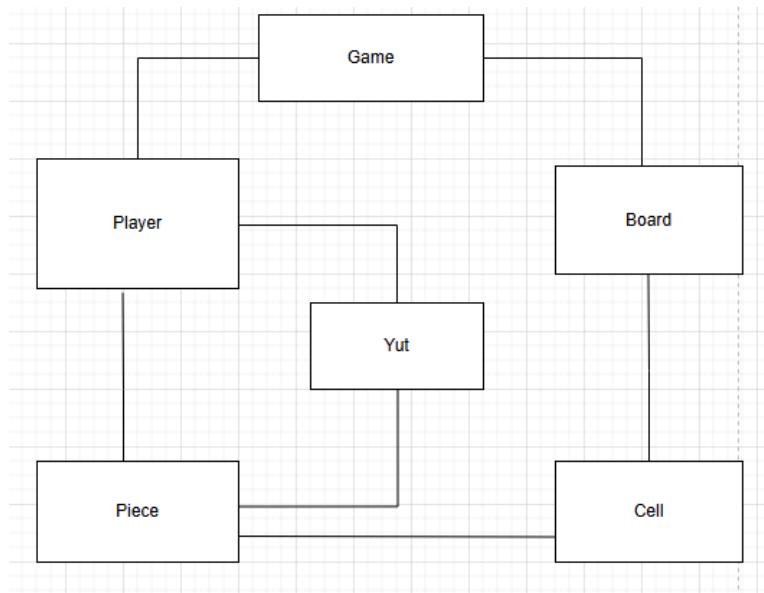
1. Find conceptual classes

Conceptual classes를 찾기 위하여 우리는 Use case에서 Noun phrases를 참고하였다. Use case에서 반복적으로 등장하는 단어들을 선정하였으며 그 결과는 다음과 같다.

- Game System → Game
- 플레이어 → Player
- 게임판 → Board
- 윷 → Yut
- 말 → Piece
- 말의 위치 → Cell

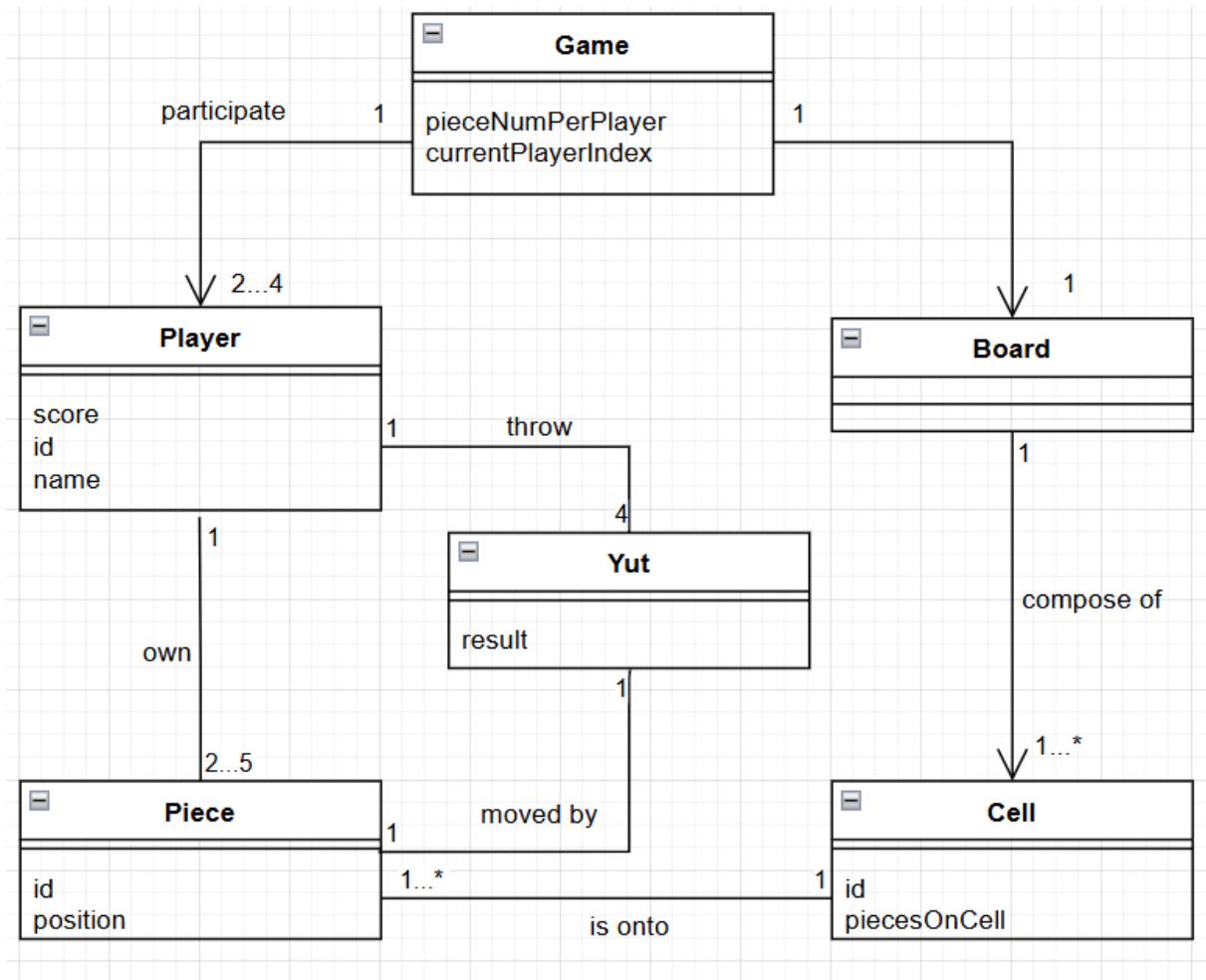
2. Draw the identified conceptual classes in a UML class diagram

위 Use case를 기반으로 앞에서 정의한 conceptual class 사이의 relationship 유무를 분석하였으며 그 결과는 다음과 같다.



3. Add associations and attributes to conceptual classes

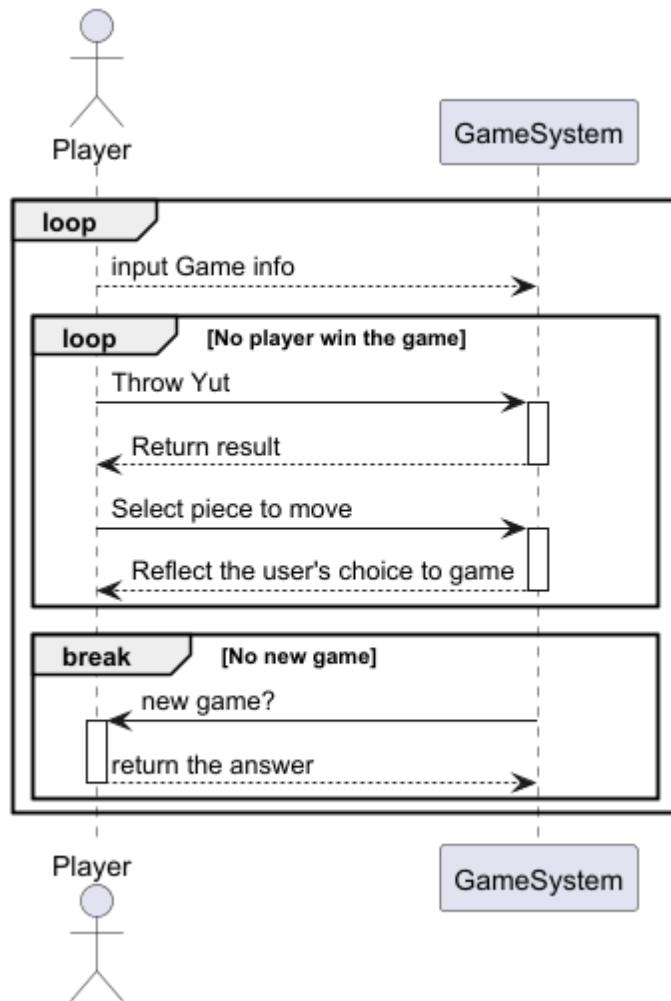
2의 결과를 바탕으로 association과 attribute를 추가한 domain model은 다음과 같다.



- 한 Game에는 2~4명의 Player가 참여할 수 있다.
- 한 Game에서 사용하는 Board는 하나이다.
- Board는 1개 이상의 Cells로 이루어져 있으며, 각 Cell 위에는 1개 이상의 Pieces가 위치할 수 있다.
- 각 Player는 2~5개의 Piece를 소유하고 있다.
- Player가 Yut을 던지면 그 결과만큼 하나의 Piece를 이동한다.

System Sequence Diagram

위 Use case와 Domain model에서 게임에 참여하는 플레이어는 Conceptual class의 Player에 대응되며, 나머지 Conceptual classes는 System을 구성하는 요소이다. 이러한 측면에서 System sequence diagram은 다음과 같이 Player와 GameSystem, 둘 사이의 상호작용으로 구성된다.



- 게임이 시작되면 플레이어는 게임에 대한 정보 (참여 인원, 게임판 종류, 플레이어 당 말 개수)를 입력한다. GameSystem은 해당 정보를 바탕으로 새로운 게임을 생성한다.
- 각 플레이어들은 최종 승자가 나오기 전까지 윷을 던져 말을 이동하는 작업을 반복한다.
- 최종 승자가 결정되면 GameSystem은 플레이어에게 새로운 게임을 시작할지 물어본다. 해당 질문에 대한 플레이어의 대답을 가지고 GameSystem은 새로운 게임을 시작하거나, 프로그램을 종료할지 결정한다.

Operation Contract

위 System Sequence Diagram을 보면 알 수 있듯이 사용자의 행동은 다음과 같다.

- 게임 시작을 위한 정보 입력
- 윷 던지기
- 이동할 말 선택

- 새로운 게임을 시작할지 결정

위 4가지 행동에 대한 Operation Contract는 다음과 같다.

1. 게임 시작을 위한 정보 입력

Operation:	Input Game Info
Cross References	Use Case 1. 게임 시작
Preconditions:	
PostConditions:	사용자가 입력한 정보를 기반으로 게임이 생성됨

2. 윷 던지기

Operation:	Throw Yut
Cross References	Use Case 2. 윷 던지기
Preconditions:	게임이 시작됨
Postconditions:	결과를 플레이어에게 반환하고 UI에 명시

3. 이동할 말 선택

Operation:	Select Piece to move
Cross References	Use Case 3. 말 이동하기, Use Case 4,5,6 규칙 적용
Preconditions	플레이어가 윷을 던지고 GameSystem은 해당 결과를 반환하고 UI에 명시
Postconditions	GameSystem이 이동 결과를 UI에 명시

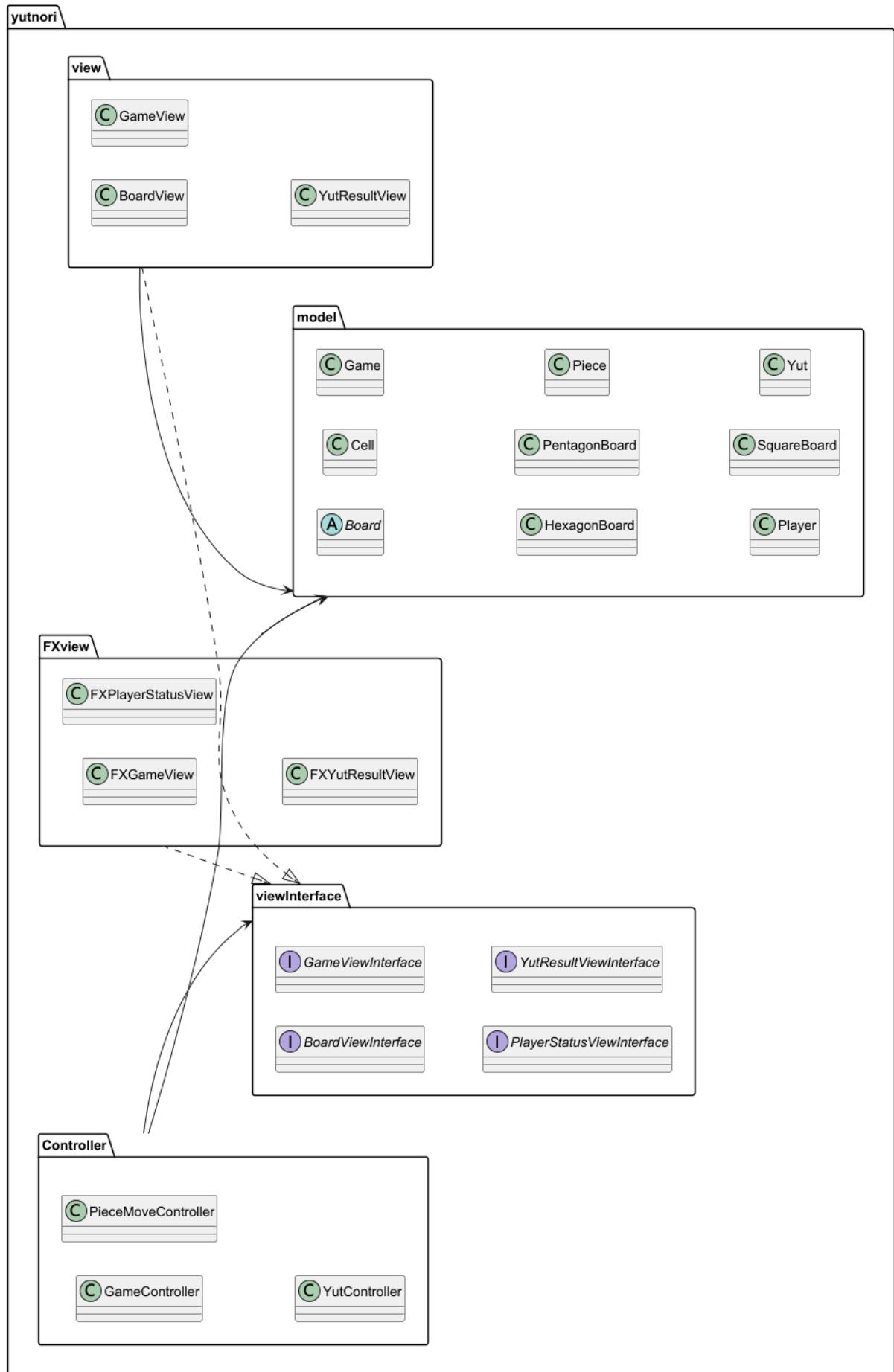
4. 새로운 게임을 시작할지 결정

Operation:	Determine whether new game or not
Cross References	Use case 9. 게임 종료
Preconditions	게임의 승자가 결정되어 게임이 종료됨
Postconditions	플레이어의 입력에 따라 새로운 게임이 시작되거나, 프로그램이 종료됨

3. 설계 및 구현

MVC Model

위 Domain Model, System sequence diagram, Operation contract를 기반으로 MVC Pattern을 적용하였다. 당시 가장 중요하게 생각한 요소는 UI 추가에 따른 기존 코드의 수정 최소화이다. 만약 Model, Controller가 특정 UI toolkit에 의존한다면 새로운 UI가 추가될 때마다 기존 코드의 전반적인 수정이 요구되며, 이는 좋은 설계라고 보기 어렵다. 우리는 이러한 상황을 피하기 위하여 Interface 기반 설계를 추가로 도입하였으며 그 결과는 다음과 같다.



- **Model**

- 게임을 구성하는 객체들의 정보를 저장한다.

- **viewInterface**

- view, FXview 계층의 추상화
 - 각각의 UI toolkit에 대응되는 view layer는 viewInterface를 통하여 Controller와 연결된다.

- **view, FXview**

- 각각 Swing, JavaFX 기반 GUI를 구현한다.
 - Model을 참조하여 UI를 rendering한다.
 - User gestures (Input game info, Throw Yut, etc)를 Controller에게 전달한다.

- **Controller**

- 사용자 입력 처리 및 Model/view, FXview 연결
 - UI layer가 User gestures를 전달하면, 이에 대한 적절한 Action (Move Piece, check end condition, etc)을 Model에게 전달한다.

UI 추가에 따른 기존 코드 수정 여부 검증

1. Interface 기반 설계의 필요성

Interface를 도입하지 않고 Model, view, Controller package를 이용하여 프로젝트를 설계하였다고 가정해 보자. 이러한 설계에서는 Controller가 직접 view package를 참조한다 (Controller 내 class가 view 내 class 객체를 멤버로 가진다). 위 설계에서 새로운 UI Toolkit을 사용하여 view package를 추가하는 경우, Controller의 수정이 불가피하다 (추가된 view package의 객체를 멤버로 가져야 한다). 따라서 Controller가 특정 view를 직접적으로 참조하는 설계는 확장성이 떨어진다.

2. 실제 적용 방식

```
public class GameController
{
    private Game game;
    private GameViewInterface view;
    ...
}
```

Controller를 구성하는 클래스들은 view 내의 클래스 객체를 멤버로 가지지 않고 viewInterface 내의 인터페이스 객체들을 멤버로 가지며, 각 view 구현체는 해당 인터페이스 객체를 통해 주입된다. 이를 통해 Controller는 특정 UI에 의존하지 않고 동일한 방식으로 동작한다.

```
GameViewInterface View = new GameView(); //FXGameView 객체로 대체 가능
GameController controller = new GameController(View);
```

3. 기존 코드의 유지 확인

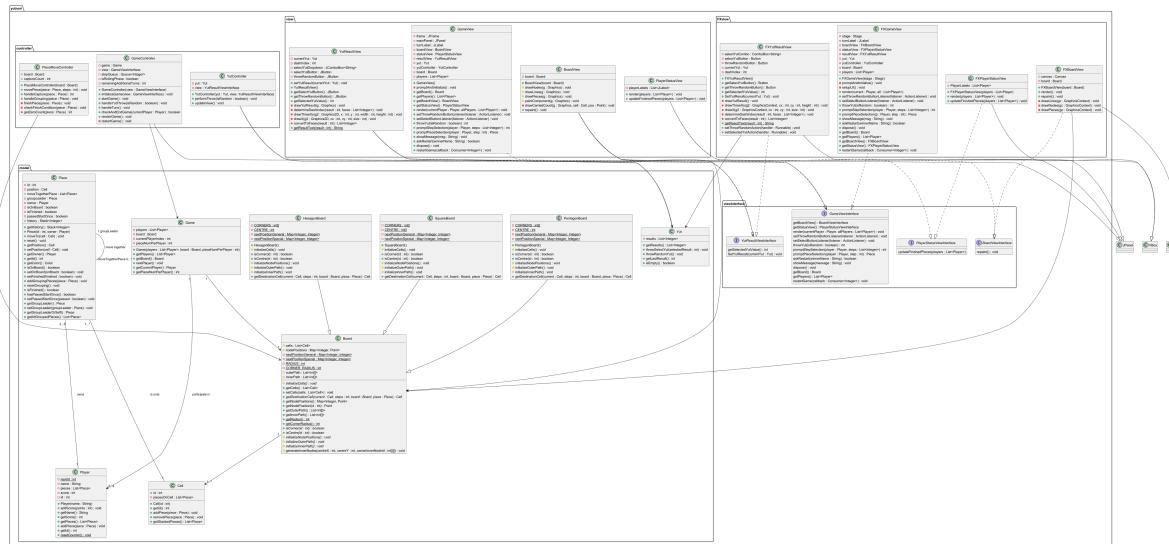
- Controller 패키지 내 클래스는 viewInterface를 통하여 각 view 패키지를 참조 함.
- Model은 다른 패키지를 참조하지 않으므로 수정 필요 X

4. 전략적 결과

- MVC 패턴과 인터페이스를 활용한 Polymorphism 패턴을 적용하여 기존 코드의 수정 없이 JavaFX 기반 UI를 적용함.
→ 코드 재사용성 및 확장성 증가

Class diagram

위 MVC Model을 기반으로 SOLID principle을 반영하여 다음과 같이 Class diagram을 설계하였다.



Model

- **Game** : 게임 진행에 대한 전반적인 정보를 저장하는 클래스

- **Attribute**

- `players` : 게임에 참여하는 Player 객체 리스트
 - `board` : 게임이 진행되는 Board 객체
 - `currentPlayerIndex` : 현재 차례의 Player index
 - `pieceNumPerPlayer` : Player마다 할당된 Piece의 개수

- **Method**

- `void nextPlayer()` : `currentPlayerIndex`를 다음 차례의 Player index로 바꿈

- **Player** : 게임에 참여하는 플레이어에 대한 정보를 저장하는 클래스

- **Attribute**

- `name` : Player의 이름
 - `pieces` : Player가 가지고 있는 Piece 객체 리스트
 - `score` : Player가 도착점까지 이동한 Piece 개수에 따른 점수
 - `id` : Player를 구분하기 위해 Player 객체 각각에게 할당하는 id

- **Method**

- `void addScore(int points)` : Player가 소유하는 Piece가 도착점에 도착했을 때 Player의 score를 증가시킴
 - `void addPiece(Piece piece)` : Player가 가지고 있는 Piece List에 매개변수로 전달된 Piece 객체를 추가함
 - `static void resetCounter()` : Game 재시작 시, game 내의 player의 카운트를 1부터 다시 시작하게 함 (Player1, Player2, ... 가 만들어진 이후 재시작 시, Player3, Player4, ... 로 자동화되지 않고 Player1부터 다시 생성하게 만들어 줌)
- **Board** : 게임이 진행되는 게임판에 대한 정보를 가진 abstract 클래스, 실제 게임판 객체를 생성하는 클래스(SquareBoard, PentagonBoard, HexagonBoard)가 해당 클래스를 상속하고 있음
 - **Attribute**
 - `cells` : 게임판 각각의 위치를 나타내는 Cell 객체 리스트
 - `nodePositions` : 각 Cell의 위치를 저장
 - `nextPositionGeneral` : 시작 위치가 분기를 만드는 Cell이 아닐 때의 이동 경로
 - `nextPositionSpecial` : 시작 위치가 분기를 만드는 중앙/코너 Cell일 때의 이동 경로
 - `RADIUS` : 일반 Cell의 반지름 (flexible)
 - `CORNER_RADIUS` : 코너 및 센터 Cell의 반지름 (flexible)
 - `outerPath` : Cell 사이의 연결을 저장하는 리스트 (각 연결은 int[]로 표현)
 - `innerPath` : Cell 사이의 연결을 저장하는 리스트 (각 연결은 int[]로 표현)
 - **Method**
 - `void initializeCells()` : 각 4,5,6각형에 cells를 만드는 역할.
 - 각 모양에 필요한 cell들의 갯수가 다름 (예컨대 사각형에서는 30개),
 - `Cell getDestinationCell(Cell current, int steps, Board board, Piece piece)` : current에서 steps만큼 이동할 때 도착 Cell을 반환
 - `Point getNodePosition(int id)` : Cell id에 해당하는 위치를 반환

- `boolean isCorner(int id)` : 게임판에서 코너 Cell인지 판단하는 method. 게임판 종류에 따라 코너 Cell이 다르기 때문에 abstract method로 선언
 - `boolean isCentre(int id)` : 게임판에서 중심 Cell인지 판단하는 method. 게임판 종류에 따라 중심 Cell이 다르기 때문에 abstract method로 선언
 - `void initializeNodePositions()` : 각 node의 position들을 define.
 - `void initializeOuterPath()` : outermost path (대각선 루트를 타지 않는, 즉 worst case의 path)를 나타냄.
 - `void initializeInnerPath()` : inner path (대각선 루트)를 나타냄. 미리 define해둔 뒤, 이 2개 (`initializeOuterPath()`, `initializeInnerPath()`)는 `drawLines()`라는 `boardView`의 선을 그리는 method에 쓰일 예정. 즉 윷놀이 판에서 선을 그리기 위해서 path를 define하는 method임.)
 - `void generateInnerNodes(int centreX, int centreY, int[][] cornerInnerNodeId)` : linear interpolation을 사용하여 안쪽 노드 (outermost path에 있지 않은 inner node, 즉 대각선 노드들)들을 생성.
- **SquareBoard** : 사각형 게임판에 대한 정보를 가진 클래스, Board 상속
 - **Attribute**
 - `CORNERS` : 사각형 윷판의 코너 ID
 - `CENTRE` : 사각형 윷판의 중앙 ID

Board 클래스에서 상속받은 Attribute, Method는 설명에서 제외하였다.
- **PentagonBoard** : 오각형 게임판에 대한 정보를 가진 클래스, Board 상속
 - **Attribute**
 - `CORNERS` : 오각형 윷판의 코너 ID
 - `CENTRE` : 오각형 윷판의 중앙 ID

Board 클래스에서 상속받은 Attribute, Method는 설명에서 제외하였다.
- **HexagonBoard** : 육각형 게임판에 대한 정보를 가진 클래스, Board 상속

- **Attribute**

- `CORNERS` : 육각형 윷판의 코너 ID
- `CENTRE` : 육각형 윷판의 중앙 ID

Board 클래스에서 상속받은 Attribute, Method는 설명에서 제외하였다.

- **Piece** : Player가 움직이는 말에 대한 정보를 담은 클래스

- **Attribute**

- `id` : Piece 객체끼리의 구분을 위해 각각에게 할당되는 정수
- `position` : 게임판에서 Piece의 현재 위치
- `moveTogetherPiece` : 해당 객체와 함께 움직이는 Piece 객체들 (업기 기능을 위한 속성)
- `groupLeader` : 업힌 말들의 leader piece. (대표적으로 움직이는 개체. 맨 밑에 있는 말을 기준으로 함. 말3이 말1에 업혔다면 말1이 leader piece가 되는 등)
- `owner` : 해당 말을 조종하는 Player
- `isOnBoard` : 현재 게임판 위에 있는지 여부를 확인하기 위한 변수
- `isFinished` : 한 바퀴 다 돈 말인지 여부를 판별하기 위한 변수
- `passedStartOnce` : 도착지점 (출발지점)에 도착했는지 여부를 판별하기 위한 변수
- `history` : 말이 지나왔던 history. 빽도 등의 특수한 case에서 어디로 가야 할지 판단하기 위한 변수.

- **Method**

- `void moveTo(Cell cell)` : cell로 Piece 객체 이동
- `void reset()` : 보드에서 해당 말 제거
- `Color getColor()` : 게임판에 표시되는 해당 말의 색 반환 (Player1은 빨간색, Player2는 하늘색, Player3은 노란색, Player4는 초록색)
- `void addGroupingPiece(Piece piece)` : 매개변수로 주어진 말이 해당 객체와 함께 움직이도록 moveTogetherPiece에 추가

- `resetGrouping()` : 말이 잡힐 때 기존 grouping을 해제하는 method
- **Cell** : 게임판을 구성하는 눈에 대한 클래스
 - **Attribute**
 - `id` : 게임판을 구성하는 각각의 Cell 객체를 구분하기 위한 정수
 - `piecesOnCell` : Cell 위에 있는 Piece 객체들
 - **Method**
 - `void addPiece(Piece piece)` : Cell 위에 Piece 추가
 - `removePiece(Piece piece)` : Cell 위에 있는 Piece 제거
- **Yut** : 윷에 대한 클래스
 - **Attribute**
 - `List <Integer> results` : 플레이어가 던진 윷의 결과값들을 전부 저장하는 리스트
 - **Method**
 - `void throwSelectYut(int selectedResult)` : 선택한 윷 결과값을 리스트에 추가
 - `void throwRandomYut()` : 무작위로 윷의 결과값을 결정하여 해당 값을 리스트에 추가
 - `int getLastResult()` : 마지막으로 던진 윷의 결과값을 반환
 - `List<Integer> getResults()` : 윷 결과값들을 전부 반환
 - `boolean isEmpty()` : results가 비어있으면 true 반환

ViewInterface

- **GameViewInterface** : GameView, FXGameView에 대한 interface
 - **Method**
 - `void render(Player currentPlayer, List<Player> allPlayers)` : 현재 차례 플레이어 및 전체 플레이어 상태 업데이트 (rendering)

- `void setThrowRandomButtonListener(ActionListener listener)` : 랜덤 윷 던지기 버튼
 - `void setSelectButtonListener(ActionListener listener)` : 선택 윷 던지기 버튼
 - `int throwYut(boolean isRandom)` : 윷을 던져서 그 결과를 받는 method
 - `int promptStepSelection(Player player, List<Integer> steps)` : 윷 혹은 모가 나와서, 윷의 결과가 하나가 아닌 여러 개가 있을 때 어떤 윷의 결과를 사용할 것인지 묻는 method
 - `Piece promptPieceSelection(Player player, int step)` : 어떤 말에 윷의 결과를 반영시킬 것인지 묻는 method
 - `boolean askRestart(String winnerName)` : 게임이 종료되었을 때 다시 시작할 것인가 묻는 method
 - `void showMessage(String message)` : 사용자에게 메시지를 보여줌. UI에 따라 message를 보여주는 방식이 다르기 때문에 (Swing은 JOptionPane, JavaFX는 alert 사용)
 - `void dispose()` : 재시작 시 기존 창을 닫는 method

- **BoardViewInterface** : BoardView, FXBoardView에 대한 interface
 - **Method**
 - `void repaint()` : 재렌더링하는 함수. Swing에서는 repaint() 함수이고, JavaFX는 render() 함수를 씀. 이것에 대한 통일성을 유지하고자 repaint()라는 함수를 만들어 view의 type과 관계없이 (UI toolkit에 관계없이) repaint를 하고자 함.

- **PlayerStatusViewInterface** : PlayerStatusView, FXPlayerStatusView에 대한 interface
 - **Method**
 - `void updateFinishedPieces(List<Player> players)` : 최종지점에 도착하여 점수가 된 말들을 계산, 플레이어별로 획득한 점수를 각각의 라벨로 저장한 후 repaint한다

- **YutResultViewInterface** : YutResultView, FXYutResultView에 대한 interface
 - **Method**
 - `int getSelectedYutValue()` : 플레이어가 선택한 윷 결과를 해당하는 정수로 반환

- `void setYutResult(Yut currentYut)` : 입력된 윷의 결과값을 시각적으로 출력

View (Swing)

- GameView

- Attribute

- `board`
 - `boardView`
 - `statusView`
 - `resultView`
 - `yut`
 - `yutController`
 - `players`

- Method

- `void promptAndInitialize()` : 게임판 형태, 플레이어 수 및 말 개수 선택, 내부에서 prompt 처리를 위하여 선언된 메소드

- BoardView : 게임판을 시각화하는 클래스

- Attribute

- `board`

- Method

- `void drawNodes(Graphics g)` : 노드의 위치를 그림
 - `void drawLines(Graphics g)` : 노드 간의 연결을 그림
 - `void drawPieces(Graphics g)` : 게임판 위 말들을 그림
 - `void paintComponent(Graphics g)` : `drawNodes`, `drawLines`, `drawPiece`를 호출하여 게임판을 구성하는 요소를 그림

- `void drawCarriedCount(Graphics g, Cell cell, Point pos)` : 말 위에 얼마나 업혀있는지 (말 1 위에 말2 가 업혀있으면 x2로 나타내는 등) 나타내줌
- **PlayerStatusView** : 플레이어의 점수, 게임판을 완주한 말 등의 정보를 시각화하는 클래스
 - **Attribute**
 - `playerLabels` : 플레이어 와 그 플레이어의 점수를 라벨로 변환하여 저장한 리스트
 - **Method**
 - `void render(List<Player> players)` : 게임이 시작될 때 정보들을 시각화 한다, 이후의 업데이트는 `updateFinishedPieces()`에서 진행된다
- **YutResultView** : 윷의 결과를 시각화하는 클래스
 - **Attribute**
 - `currentYut` : 시각화 하려는 yut의 객체
 - `dashIndex` : 빽도가 들어갈 위치
 - **Method**
 - `void drawYutResult(Graphics g)` : 아래의 method들을 사용해 YutResultView()에서 생성한 공간에 윷 결과값을 시각화
 - `void determineDashIndex(int result, List<Integer> faces)` : 윷 중에 뒷면인 것들 중에서 빽도 표시를 할 위치를 무작위로 결정
 - `void drawThreeXs(Graphics2D g2, int x, int y, int width, int height)` : `drawX()`를 사용해서 3개의 x를 그리는 method
 - `void drawX(Graphics2D g2, int cx, int cy, int size)` : x자를 그리는 method
 - `List<Integer> convertToFaces(int result)` : 윷 결과에 따라 4개의 윷이 앞면일지 뒷면 일지 결정
 - `String getResultText(int result)` : 정수로 되어있는 윷의 결과값을 문자로 변환

`viewInterface`에서 설명한 메소드, 시각화에 관련된 속성 (*e.g.*, `JPanel`)은 제외하였다.

FXView (JavaFX)

- FXGameView

- Attribute

- stage
 - turnLabel
 - boardView
 - statusView
 - resultView
 - yut
 - yutController
 - board
 - players

- Method

- void setUpUI()

- FXBoardView : 게임판을 시각화하는 클래스

- Attribute

- board

- Method

- void drawNodes(GraphicsContext gc) : 노드의 위치를 그림
 - void drawLines(GraphicsContext gc) : 노드 간의 연결을 그림
 - void drawPieces(GraphicsContext gc) : 게임판 위 말들을 그림
 - void render() : drawNodes, drawLines, drawPiece를 호출하여 게임판을 구성하는 요소를 그림

- **FXPlayerStatusView** : 플레이어의 점수, 게임판을 완주한 말 등의 정보를 시각화하는 클래스
 - **Attribute**
 - `playerLabels` : 플레이어 와 그 플레이어의 점수를 라벨로 변환하여 저장한 리스트
 - **Method**
 - `void render(List<Player> players)` : 게임이 시작될 때 정보들을 시각화 한다, 이후의 업데이트는 `updateFinishedPieces()`에서 진행된다
- **FXYutResultView** : 윷의 결과를 시각화하는 클래스
 - **Attribute**
 - `currentYut`
 - `dashIndex`
 - **Method**
 - `void drawYutResult()` : 아래의 method들을 사용해 `YutResultView()`에서 윷 결과값을 시각화
 - `void determineDashIndex(int result, List<Integer> faces)` : 윷 중에 뒷면인 것들 중에서 빼도 표시를 할 위치를 무작위로 결정
 - `void drawThreeXs(GraphicsContext g2, int x, int y, int width, int height)` : `drawX()`를 사용해서 3개의 x를 그리는 method
 - `void drawX(GraphicsContext g2, int cx, int cy, int size)` : x자를 그리는 method
 - `List<Integer> convertToFaces(int result)` : 윷 결과에 따라 4개의 윷이 앞면일지 뒷면일지 결정
 - `String getResultText(int result)` : 정수로 되어있는 윷의 결과값을 문자로 변환

`viewInterface`에서 설명한 메소드, 시각화에 관련된 속성 (*e.g.*, `JPanel`)은 제외하였다.

Controller

- **GameController**

- **Attribute**

- `game`
- `view`
- `stepQueue`
- `isRollingPhase : true`
- `remainingAdditionalTurns` : 잡은 말 수에 따라 추가 턴 수를 저장하는 변수

- **Method**

- `void initializeGame(GameViewInterface view)`
- `void startGame()` : 게임 시작 시 초기화 및 이벤트 리스너 설정
- `void handleYutThrow(boolean isRandom)` : 윷 던지기 이벤트 처리
- `void handleTurn()` : 턴을 넘길 수 있는지 확인하고 넘길 수 있다면 턴을 넘김
- `boolean checkAndEndGame(Player currentPlayer)`
- `void renderGame()` :
- `void restartGame()` :

- **PieceMoveController** : 게임판 위 말의 행동 (이동, 업기, 잡기)을 반영하는 클래스

- **Attribute**

- `board`
- `captureCount`

- **Method**

- `void movePiece(Piece piece, int steps)` : 말 이동 로직 (목적지 Cell 계산 후 이동 + 업기 처리)
- `int handleCapture(Piece piece)` : 말 잡기 로직 (다른 플레이어 말이 있으면 잡고 원 위치)
- `void handleGrouping(Piece piece)` : 말 업기 로직 (같은 플레이어의 다른 말이 해당 칸에 있으면 업기 수행)

- `void finishPiece(Piece piece)` : 말이 윷판을 다돌고 들어왔으면 끝내기 처리
 - `void checkFinishCondition(Piece piece)` : 말이 윷판을 다돌아서 finish될 수 있는지 체크하는 method
 - `int getZeroCount(Piece piece)` : 해당 말이 출발점을 몇 번 밟았는지 확인
- **YutController** : 윷 던지기를 UI에 반영하는 클래스
 - **Attribute**
 - `yut`
 - `view`
 - **Method**
 - `void performThrow(boolean isRandom)` : 입력된 값에 따라 선택 윷던지기 또는 랜덤 윷던지기를 수행하는 method
 - `void updateView()` : view 업데이트

SOLID principle 적용 여부

1. Single-Responsibility Principle (SRP)

각 클래스는 하나의 명확한 책임을 가지도록 설계하였다. 적용 예시는 다음과 같다.

- `Cell` : 게임판의 한 칸을 나타내고, 그 위의 말들을 관리함
- `Piece` : 말의 상태와 자신과 함께 이동하는 말을 관함
- `Player` : 플레이어 정보와 소유한 말을 관리
- `GameController` : 게임 흐름을 제어
- `PieceMoveController` : 말의 이동 로직만 담당
- `YutController` : 윷 던지기 제어

2. Open-Closed Principle (OCP)

확장은 가능하되, 수정은 불가하도록 설계되어 있다. 적용 예시는 다음과 같다.

- **Board**

- 새로운 보드 타입을 추가할 때 기존 코드를 수정하지 않고 확장 가능
- `SquareBoard`, `PentagonBoard`, `HexagonBoard`로 확장
- 새로운 보드 형태(예: 원형 보드)를 추가하려면 `Board`를 상속받아 구현하면 됨

- **viewInterface**

- 새로운 UI toolkit 추가 시 기존 코드 수정 없이 확장이 가능하다
- Swing 기반 View와 JavaFX 기반 FXView로 분리
- 새로운 UI 기술(예: Web UI)을 추가하려면 인터페이스만 구현하면 됨

3. Liskov Substitution Principle (LSP)

하위 타입은 상위 타입을 완전히 대체할 수 있도록 설계되어 있다. 적용 예시는 다음과 같다.

- `SquareBoard`, `PentagonBoard`, `HexagonBoard`는 상위 abstract class인 `board`를 대체 할 수 있다.
- Swing 기반 view (*e.g.*, `YutResultView`)와 JavaFX 기반 view (*e.g.*, `FXYutResultView`)는 `YutResultViewInterface`를 구현하여 상호 교체가 가능하다.

4. Interface Segregation Principle (ISP)

클라이언트가 사용하지 않는 인터페이스에 의존하지 않도록 인터페이스를 세분화하였다. 적용 예시는 다음과 같다.

- `BoardViewInterface` : 보드 뷰에 특화된 기능만 포함 (`repaint()`)
- `PlayerStatusViewInterface` : 플레이어 상태 뷰에 특화된 기능만 포함
- `YutResultViewInterface` : 윷 결과 뷰에 특화된 기능만 포함
- `GameViewInterface` : 게임 전체 뷰 관리에 필요한 기능들만 포함

5. Dependency Inversion Principle (DIP)

고수준 모듈이 저수준 모듈에 의존하지 않고, 추상화에 의존하도록 설계되어 있다. 적용 예시는 다음과 같다.

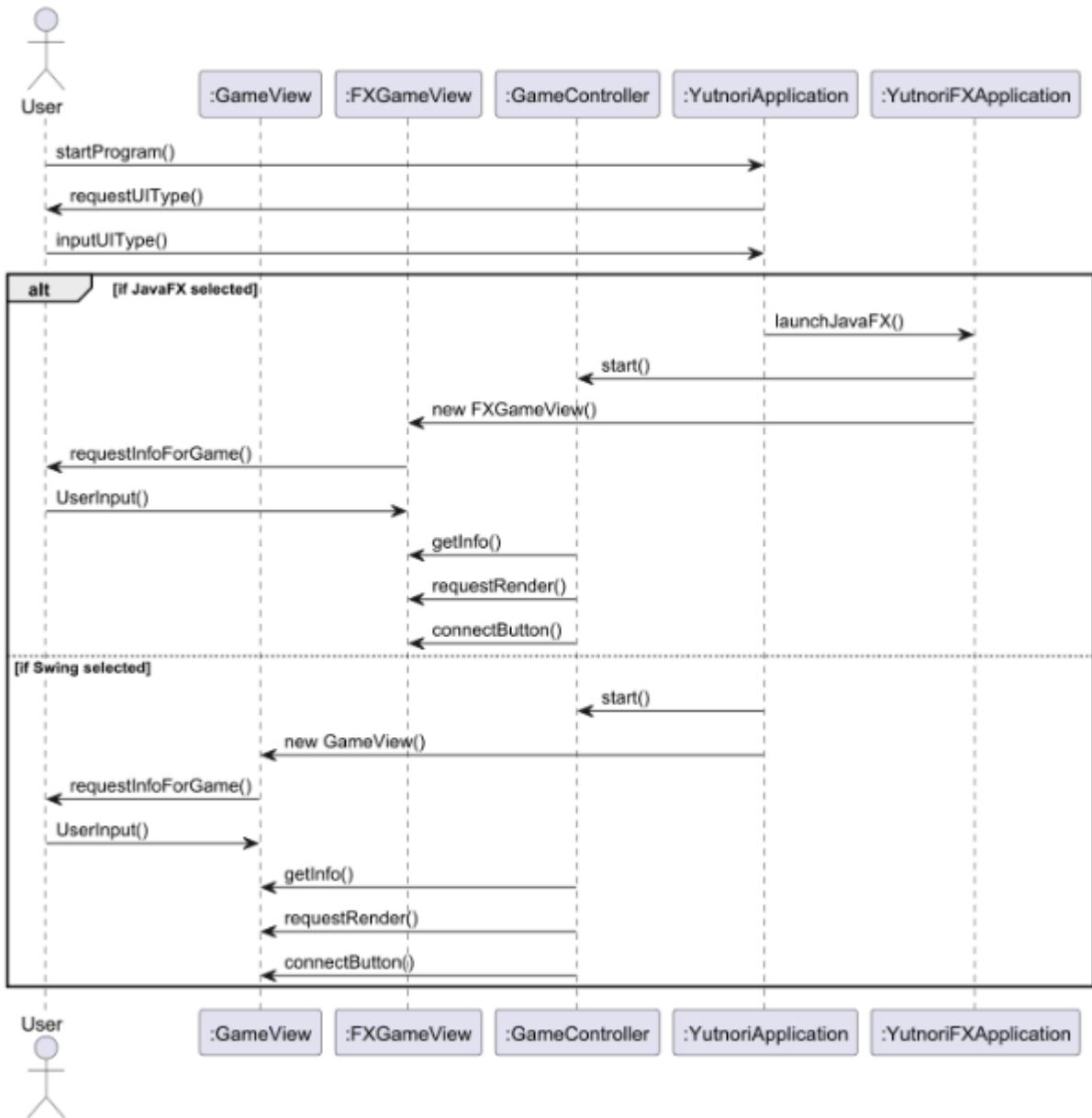
- Controller 내 클래스들은 구체적인 view 클래스에 의존하지 않고 인터페이스 객체에 의존한다.
 - `PieceMoveController` 는 구체적인 게임판 클래스 (*e.g.*, `SquareBoard`)에 의존하지 않고 `Board` 에 의존한다.
-

Sequence Diagram

구현 과정에서 소프트웨어의 flow를 보다 명확히 이해하기 위하여 다음과 같이 각각의 Use case에 대하여 Sequence diagram을 작성하였다.

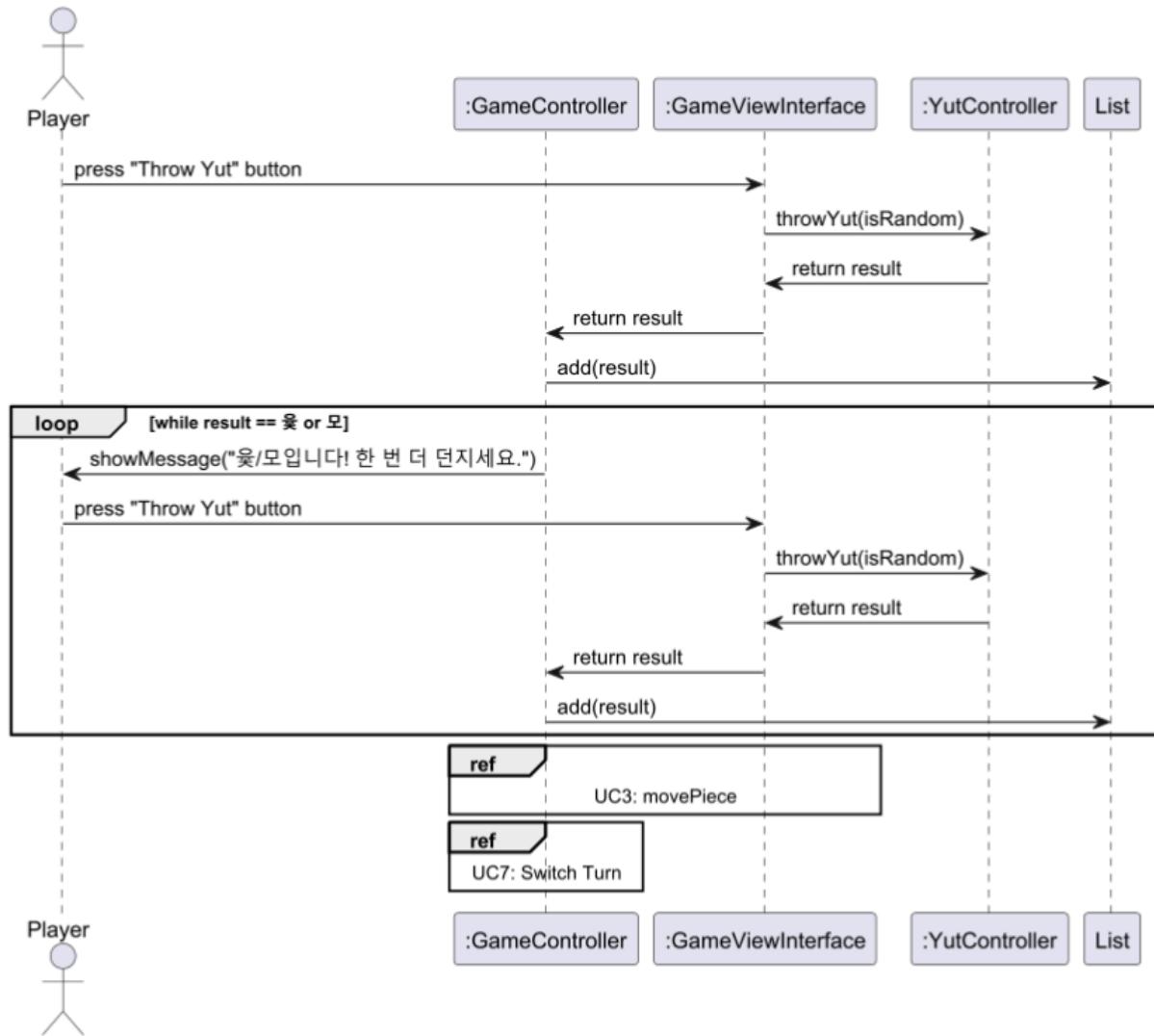
Use case 1 : Game initialization

UC1: Game Initialization Sequence Diagram



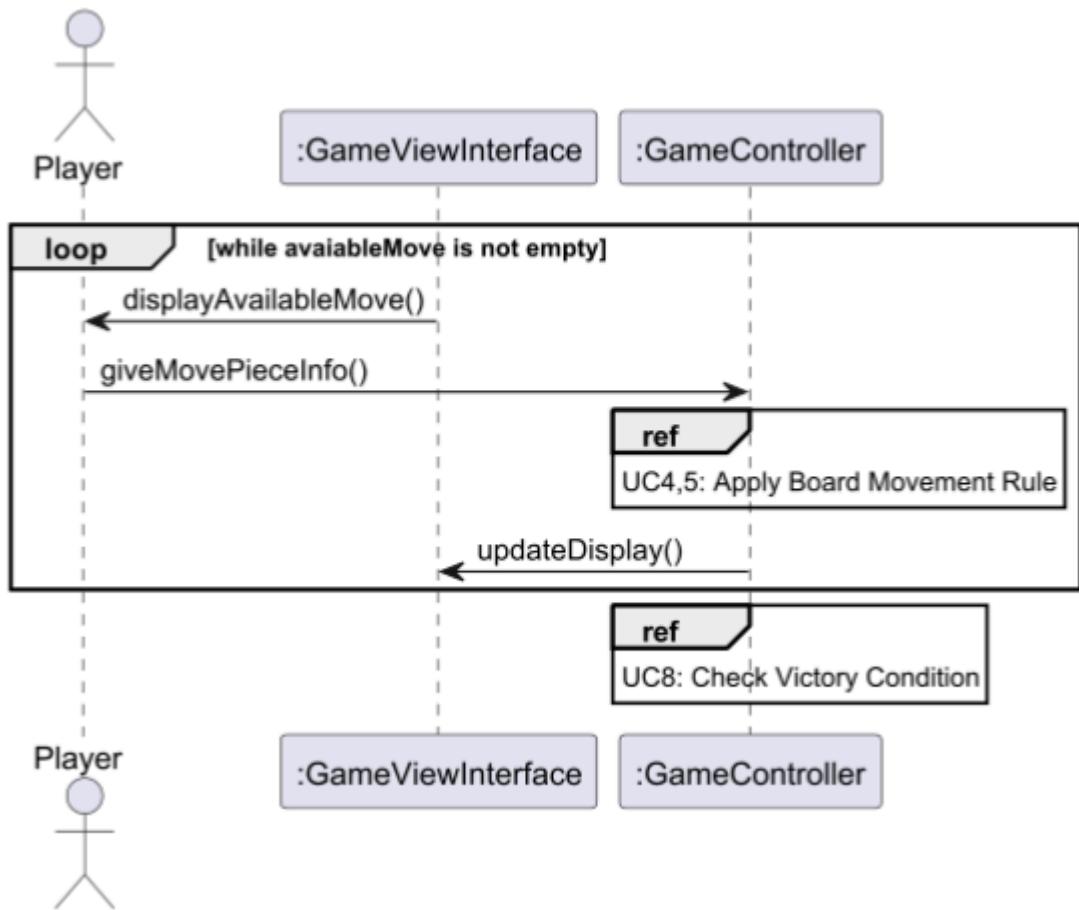
Use case 2 : Throw Yut

UC2: Throw Yut

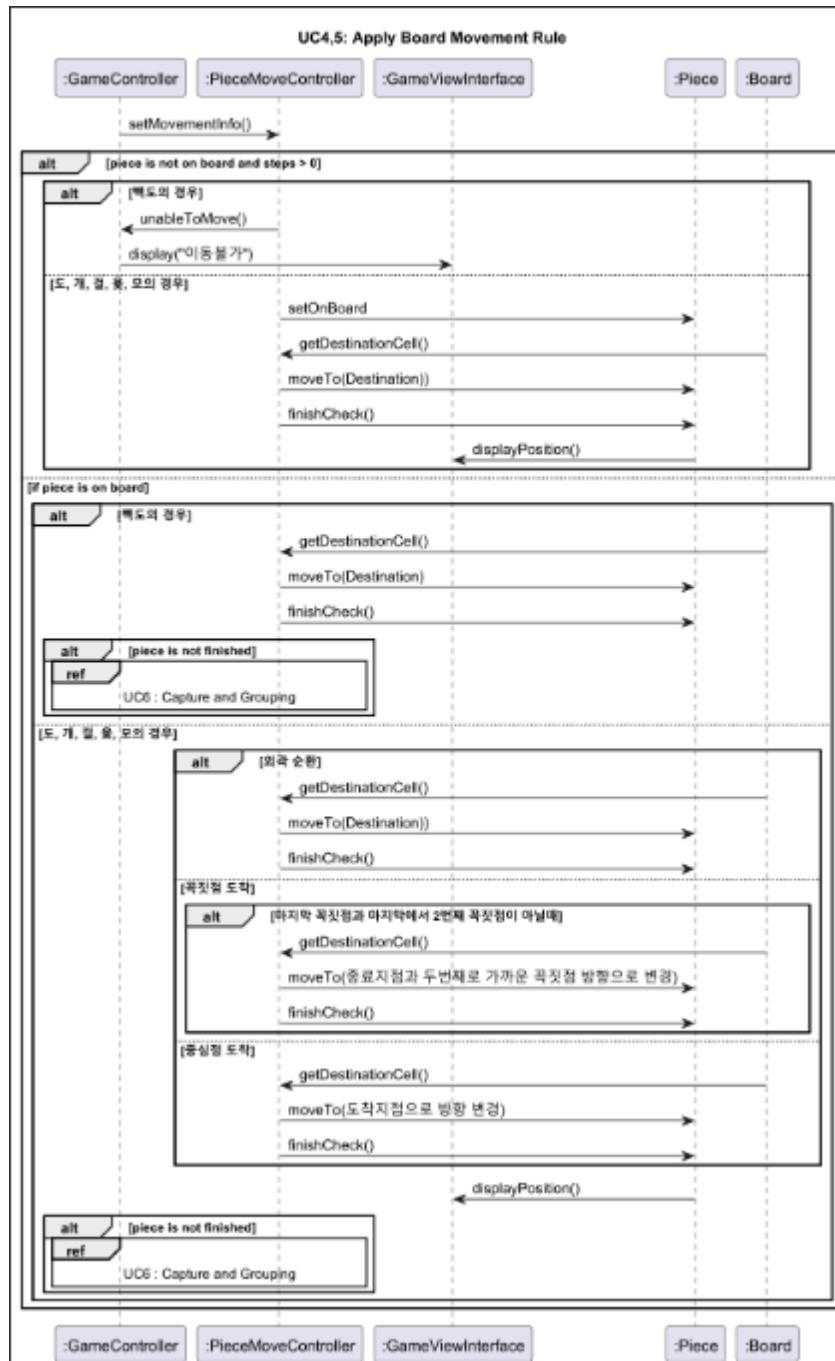


Use case 3 : Move Piece

UC3: Move Piece



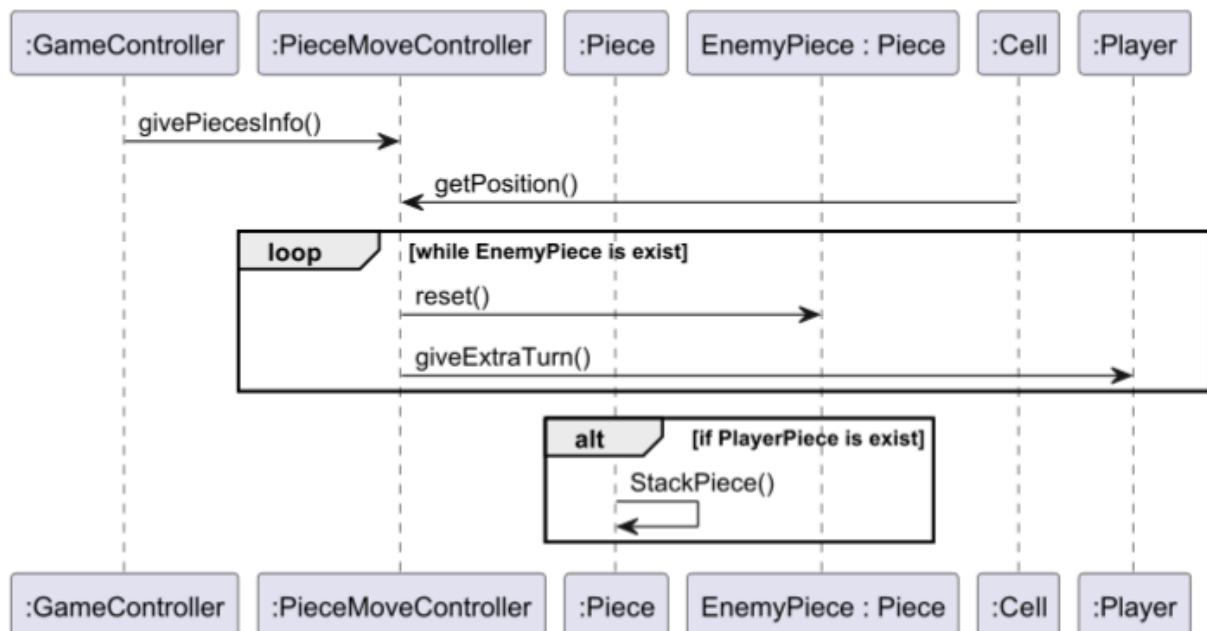
Use case 4, 5 : Apply Board Movement Rule



Use case 4,5는 게임판에 따라 이동 규칙을 정의하였다. 구체적인 이동 규칙은 다르더라도, 프로그램 진행 방향은 동일하다고 판단하고 하나의 Sequence diagram으로 작성하였다.

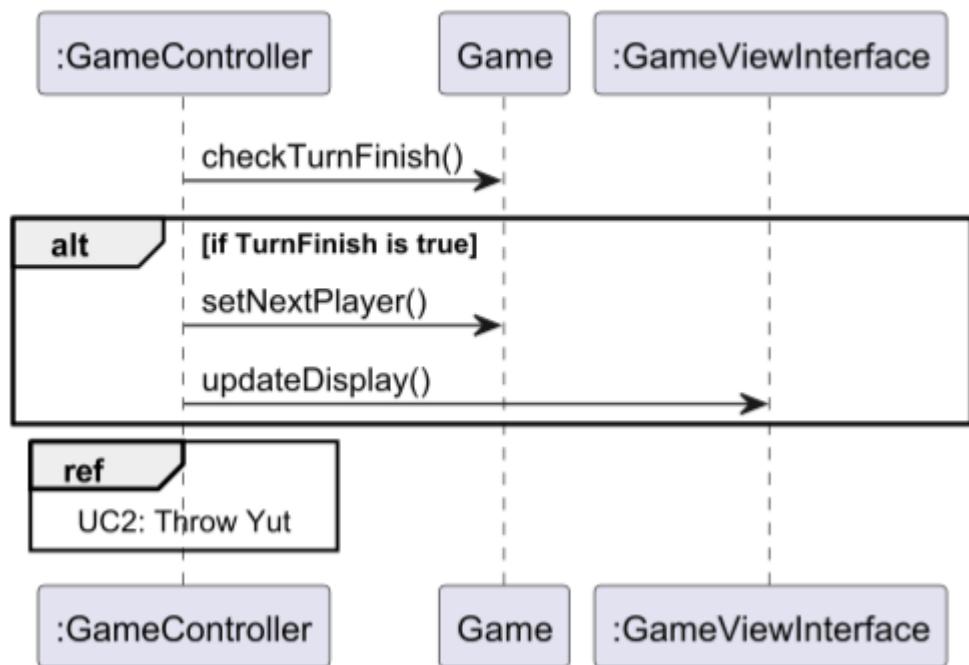
Use case 6 : Apply Special Rules - Capture and Grouping

UC6: Apply Special Rules - Capture and Grouping



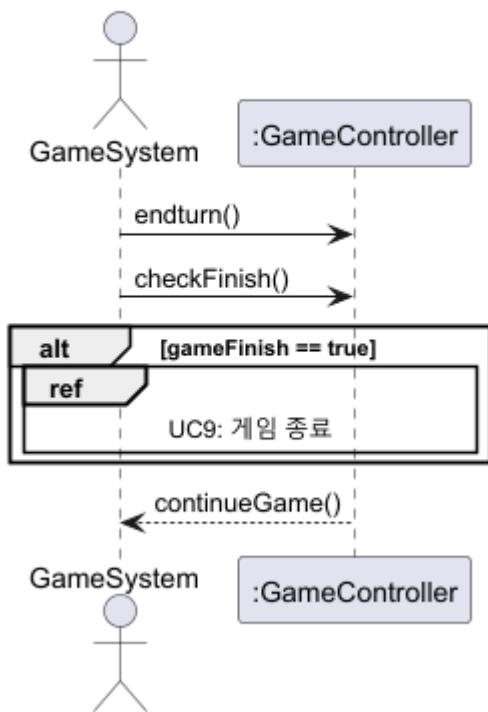
Use case 7 : Switch Turn

UC7: Switch Turn



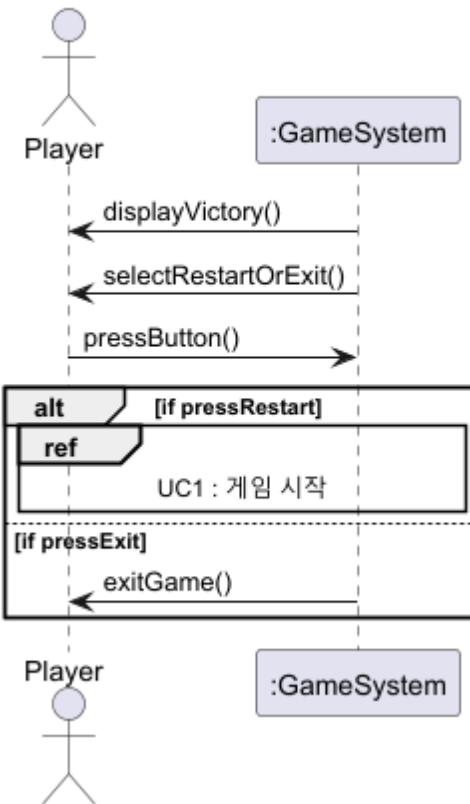
Use case 8 : Check Victory Condition

UC8: Check Victory Condition

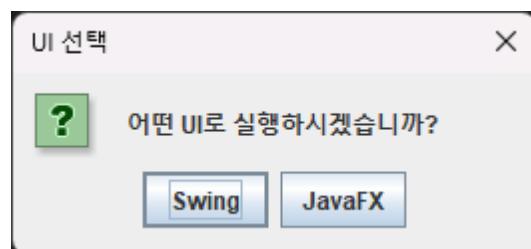


Use case 9 : Restart or Exit Game

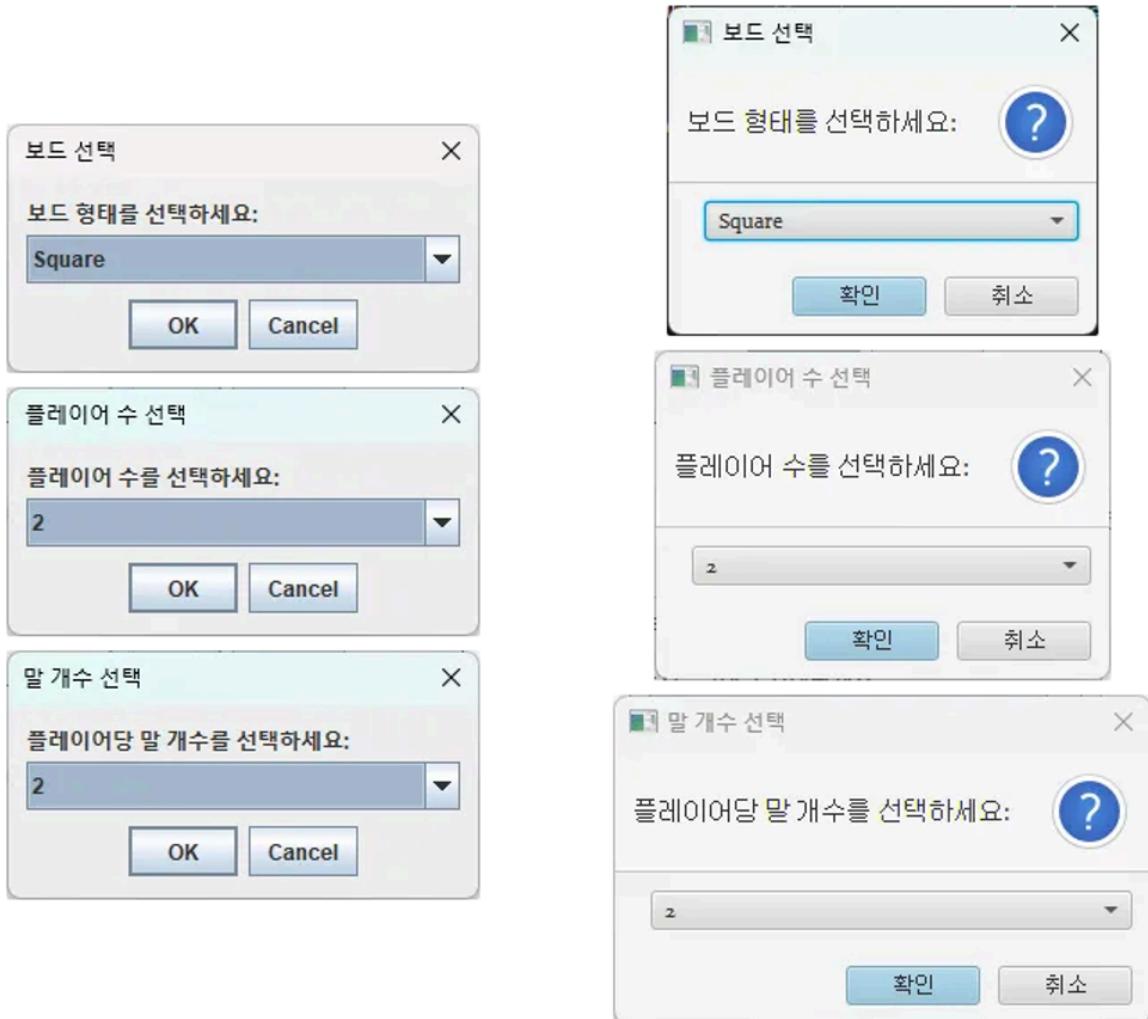
UC9: Restart or Exit Game



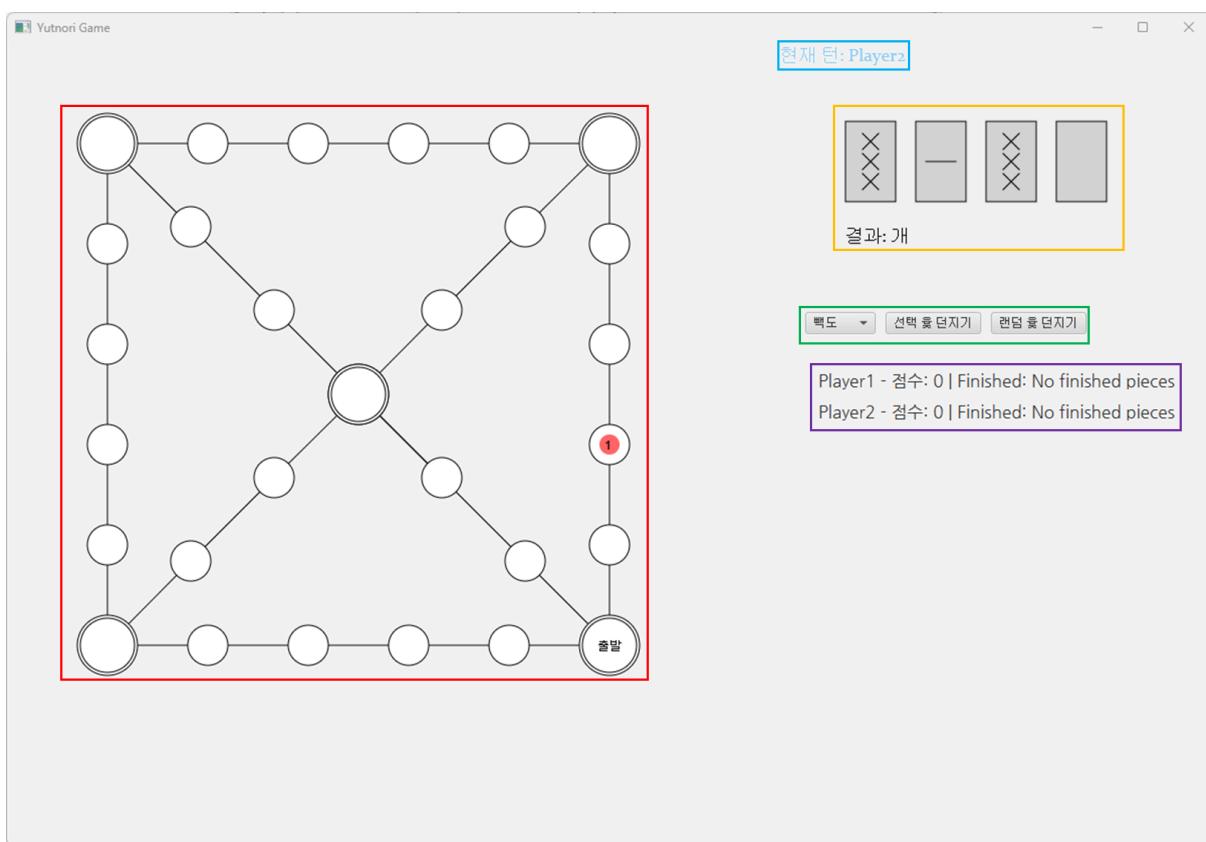
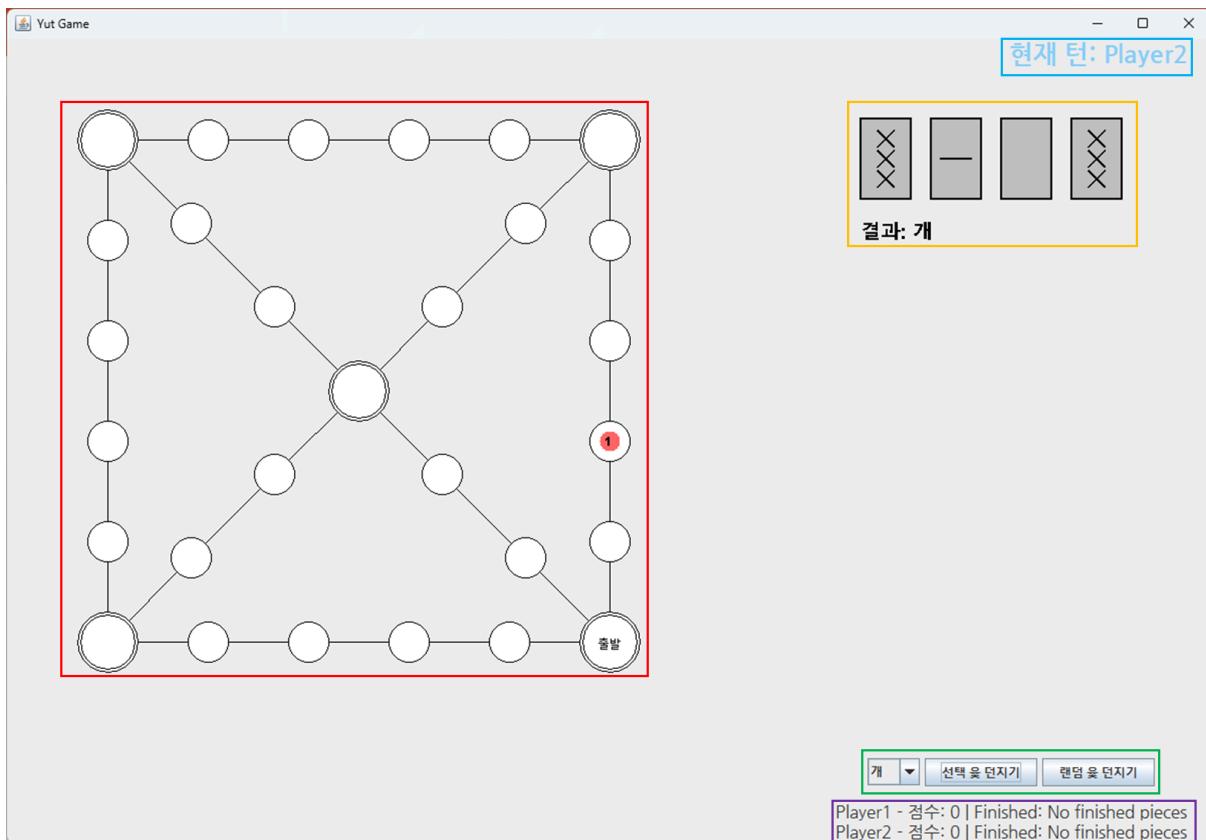
프로그램 시연



프로그램을 실행하면 사용자에게 UI 선택 화면을 제공한다. 사용자가 원하는 UI를 선택하면 해당 UI를 이용하여 게임이 만들어진다.



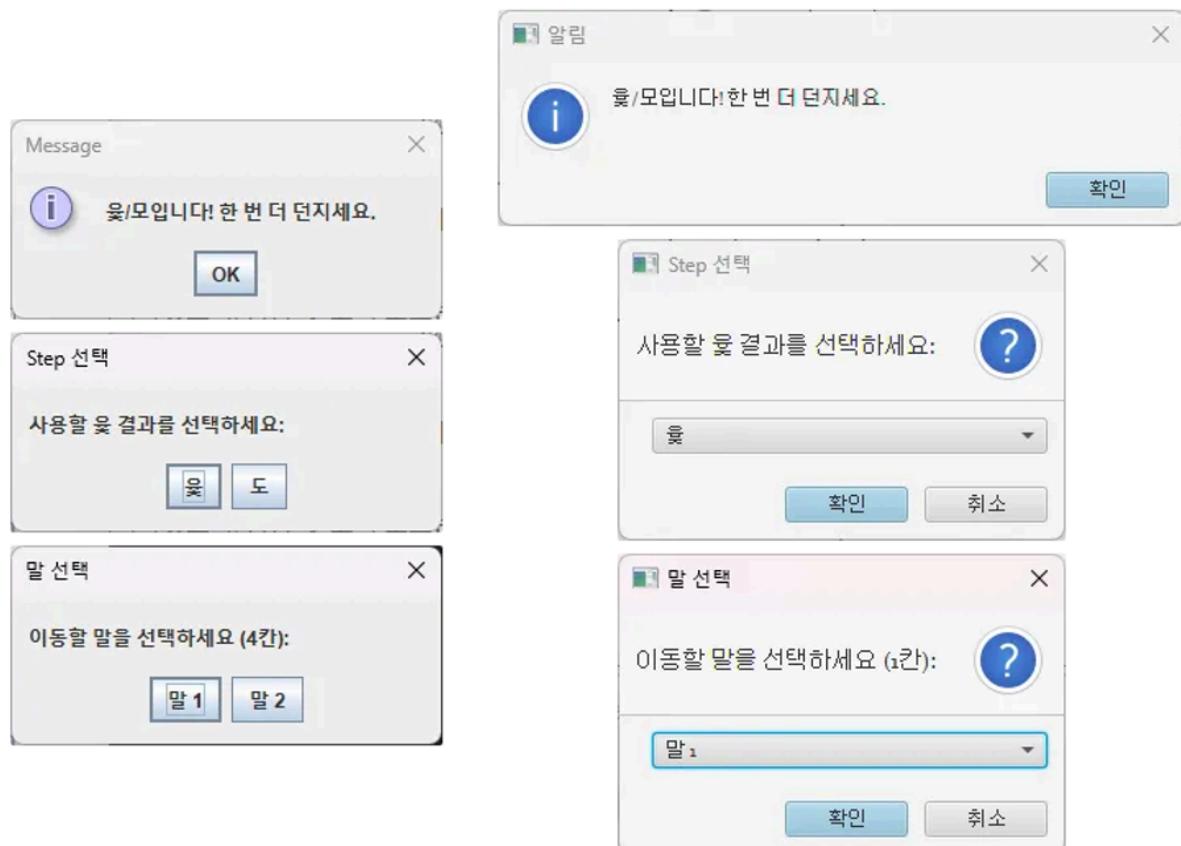
- 사용자에게 위 화면을 차례로 제공한다. 사용자는 각각의 화면에서 게임 시작을 위한 정보를 차례로 입력하고, OK를 누르면 다음 화면으로 이동하고, Cancel을 누르면 프로그램을 종료한다. 말 개수를 선택하고 OK를 누르면 게임이 시작된다.



게임 시작 후 랜덤 윷 던지기 버튼을 이용하여 Player1의 말을 1회 이동한 후의 UI이다. UI를 구성하는 요소는 다음과 같다.

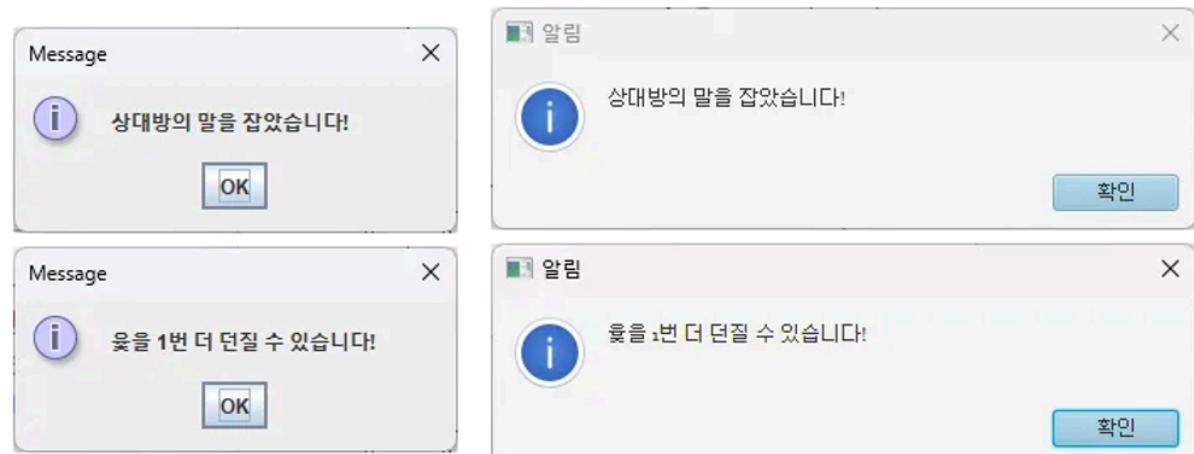
- **빨간색** : 게임판의 출발점을 텍스트를 이용하여 명시하였으며, 중앙 및 코너 노드는 두 개의 원을 겹쳐 표현하였다.
- **하늘색** : 현재 차례의 플레이어를 명시하였다. Player1은 빨간색, Player2는 하늘색, Player3는 노란색, Player4는 녹색 글씨로 표현하여 가독성을 높였다. 각각의 Player가 조종하는 말의 색도 동일하다.
- **녹색** : 현재 차례의 플레이어는 두 버튼을 이용하여 윷을 던진다.
- **노란색** : 던진 윷의 결과를 명시한다.
- **보라색** : 게임에 참여하는 플레이어가 도착점까지 이동시킨 말의 개수와 종류를 알 수 있다. 도착점까지 이동시킨 말 하나당 100점을 획득한다.

다음은 플레이어가 윷 던지기 버튼을 눌렀을 때 새롭게 등장하는 UI를 차례로 나열한 것이다.



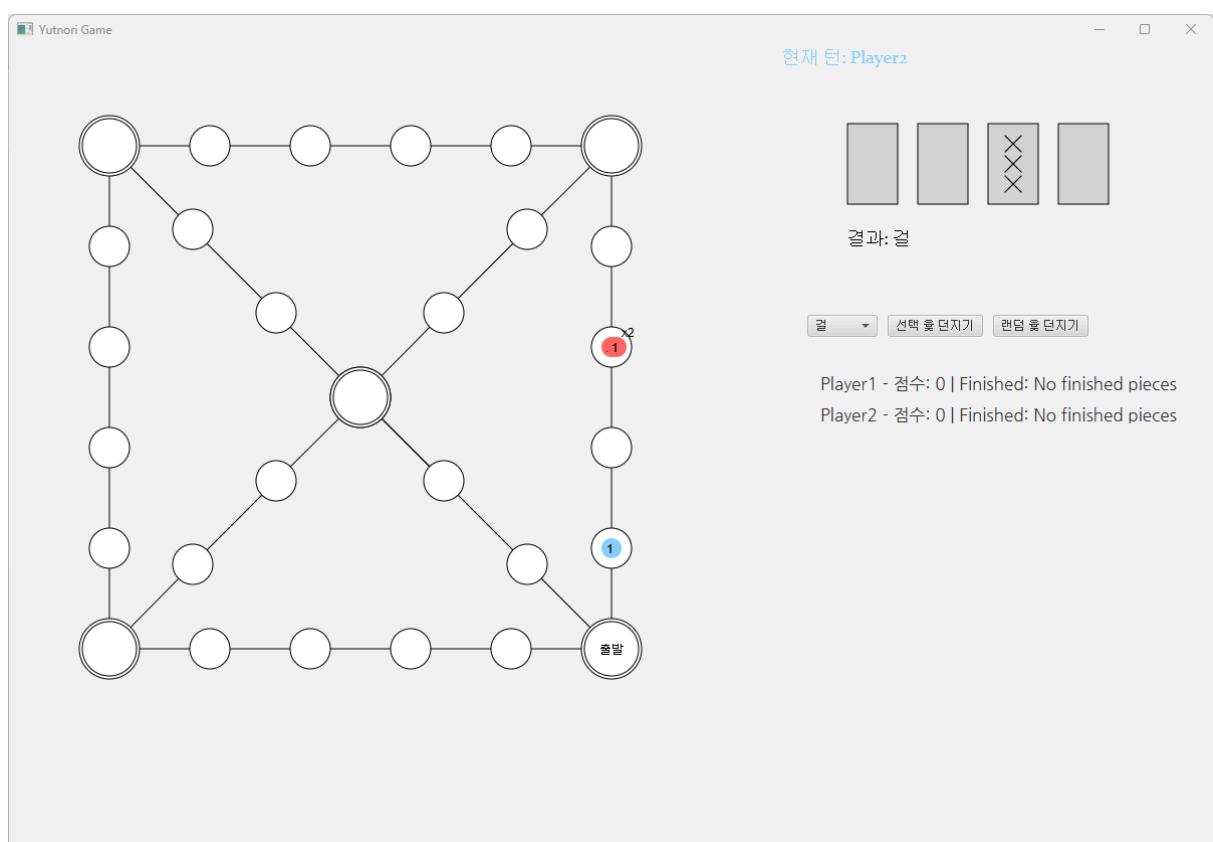
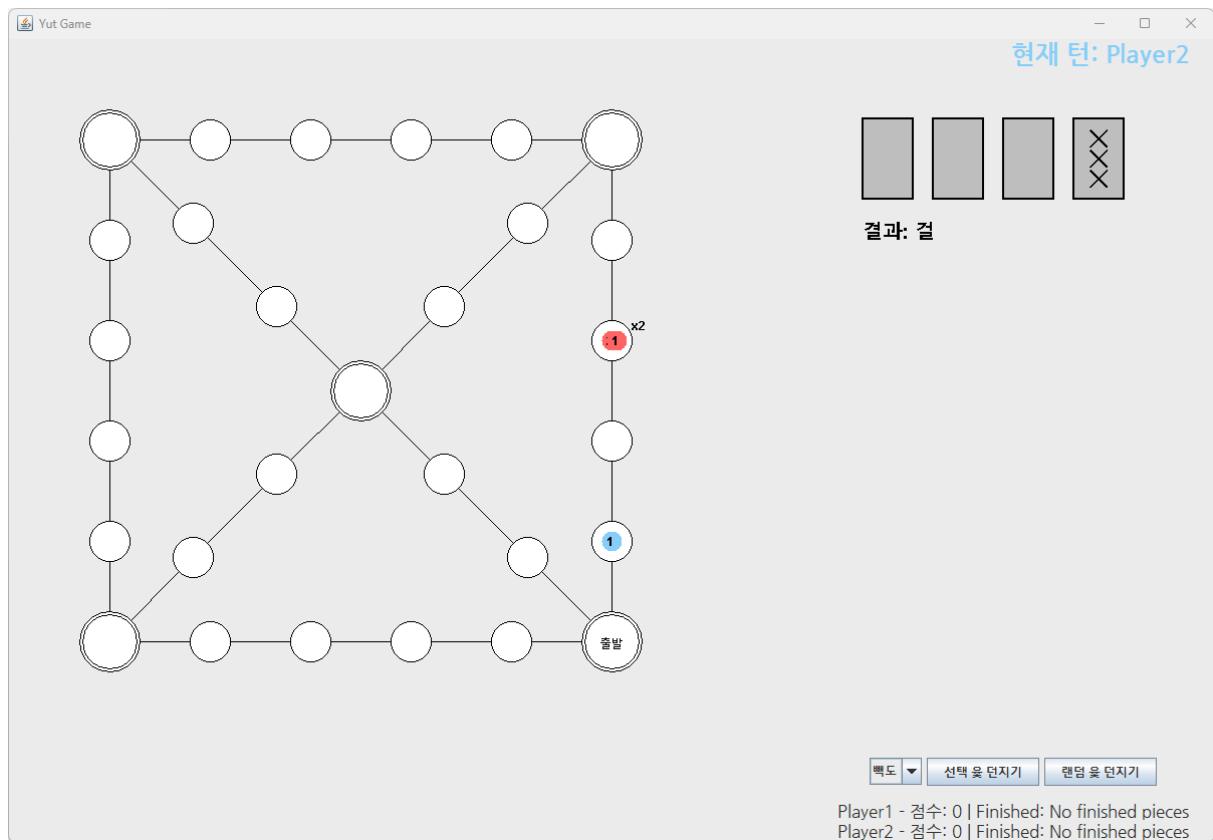
- 첫 번째 화면은 플레이어가 던진 윷의 결과가 윷/모인 경우 추가로 윷을 던질 수 있도록 안내한다.
- Step 선택 화면은 플레이어가 윷을 여러 번 던진 경우 어떠한 결과를 사용할지 선택하도록 안내한다. 만약 플레이어가 사용할 수 있는 결과가 1개인 경우 해당 UI는 등장하지 않는다.
- 말 선택 화면은 두 번째 화면에서 선택한 결과를 어떠한 말에게 적용할지 선택하도록 안내한다. 특정 말을 선택하면 게임판에 그 결과가 반영된다. 만약 게임판 위에 해당 플레이어의 말이 존재하지 않는 경우 해당 UI는 등장하지 않고 자동으로 1번 말이 이동한다. 남은 윷 결과가 존재하지 않으면 다음 플레이어에게 차례가 넘어간다. 남은 윷 결과가 1회이면 해당 UI가 다시 등장하고, 2개 이상인 경우 두 번째 화면이 다시 나타난다.

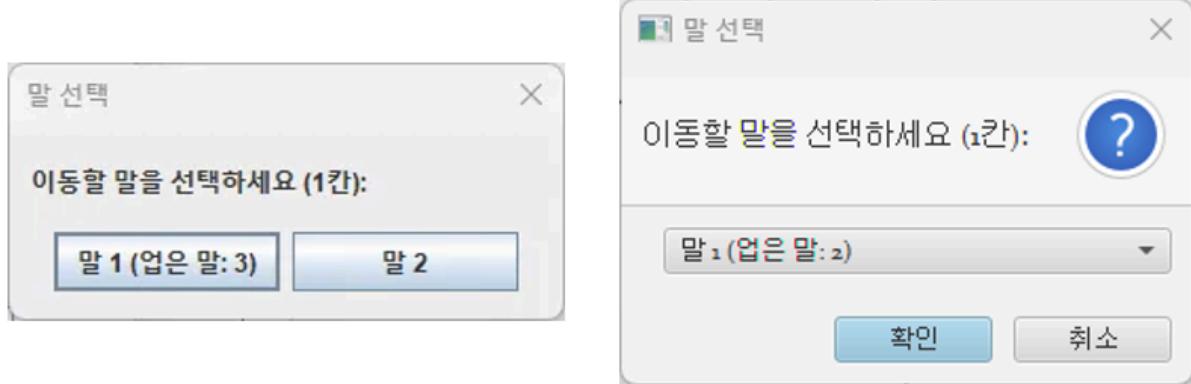
다음은 다른 플레이어의 말을 잡은 경우 새롭게 등장하는 UI를 차례로 나열한 것이다.



- 첫 번째 화면은 플레이어에게 상대방의 말을 잡았음을 안내한다.
- 두 번째 화면은 위 플레이어에게 추가 턴을 부여함을 안내한다. 추가 턴은 잡은 상대방의 말 개수만큼 부여된다. (e.g., 상대방의 말을 한 개 잡은 경우 추가 턴 1회 부여, 상대방의 말을 두 개 잡은 경우 추가 턴 2회 부여)

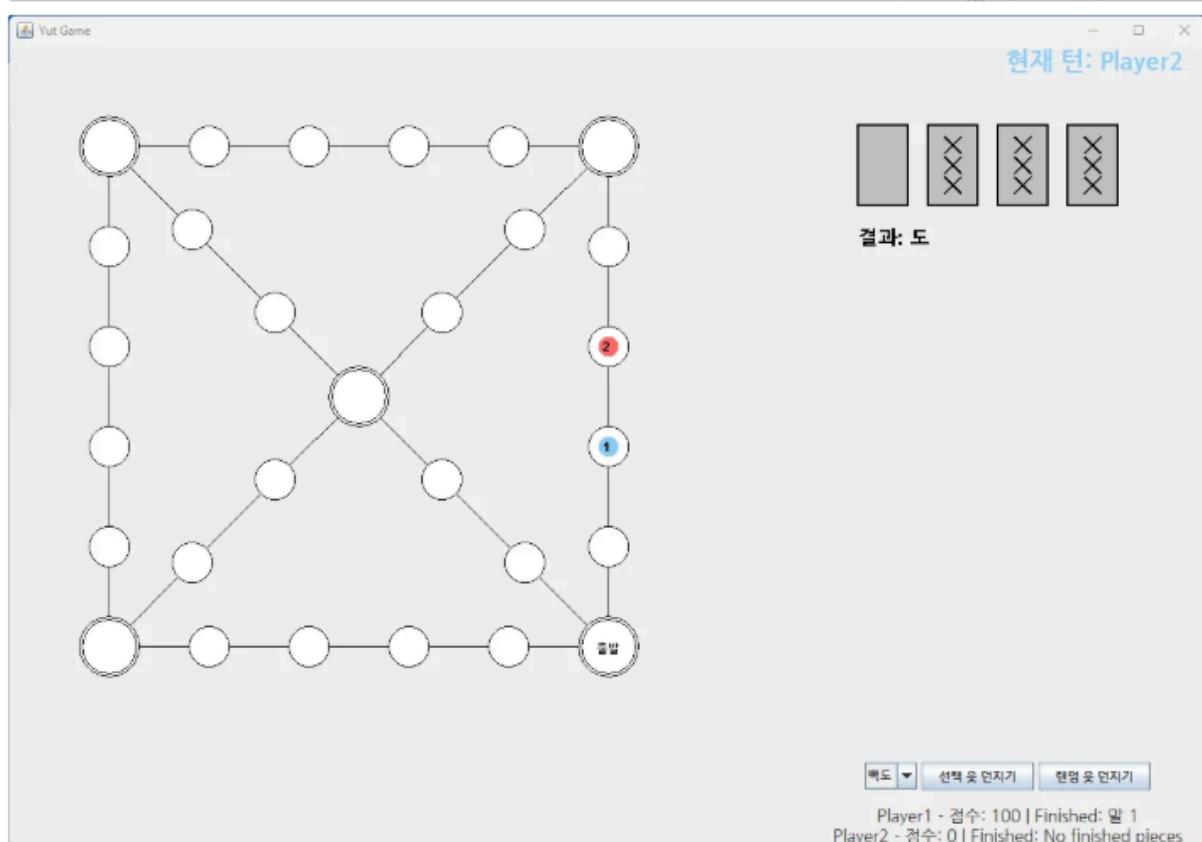
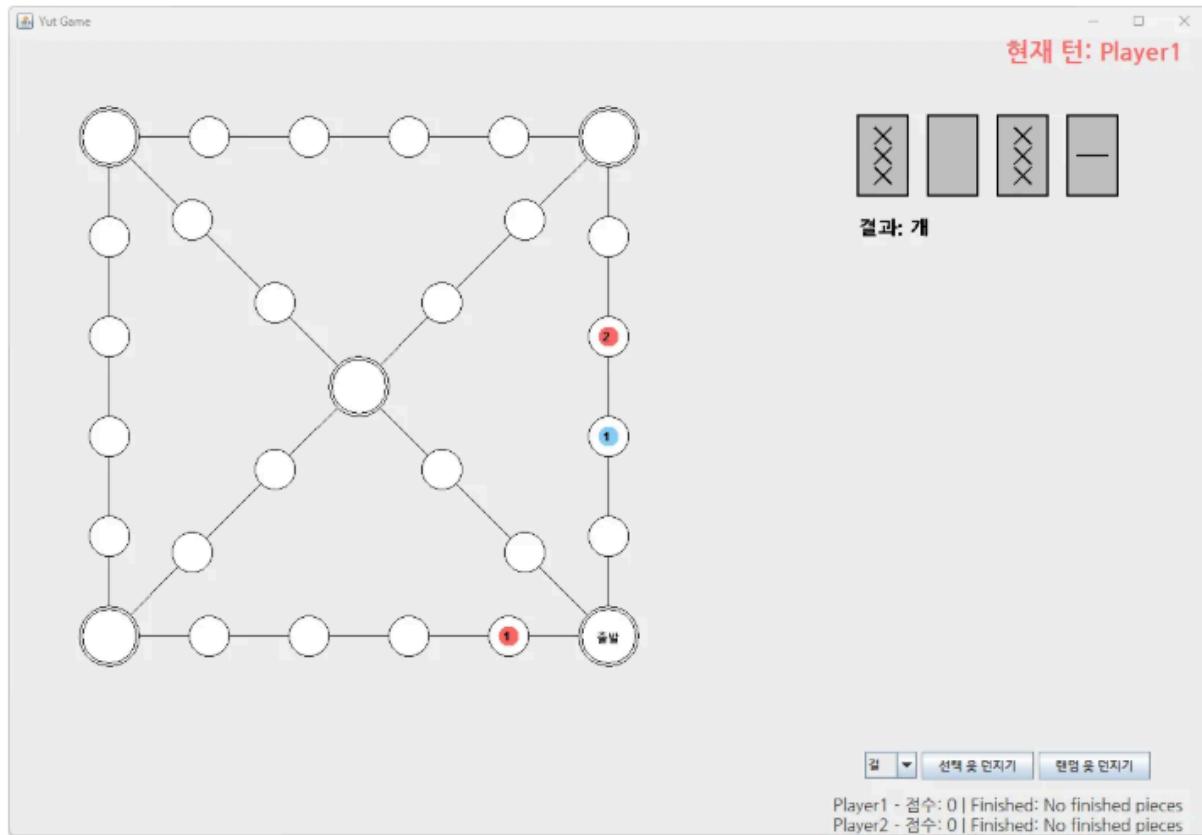
다음은 말을 업은 경우의 UI이다.

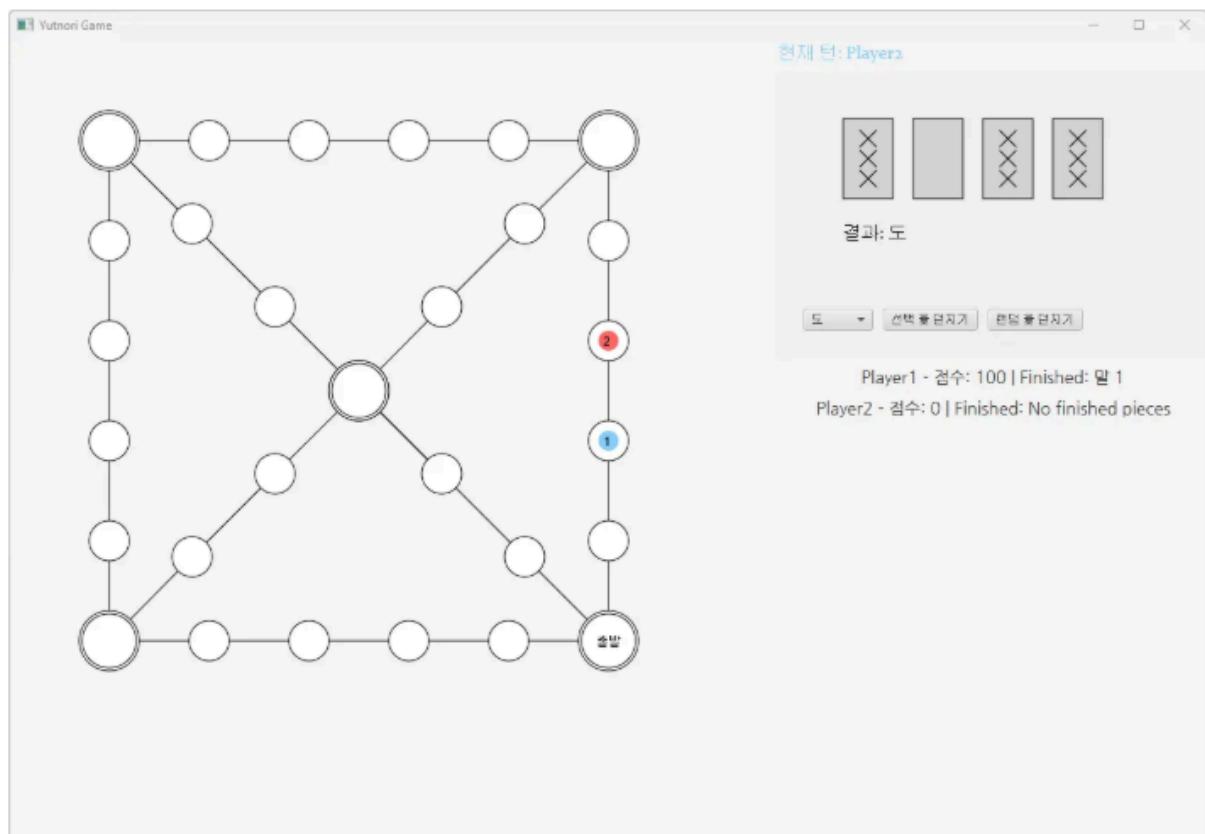
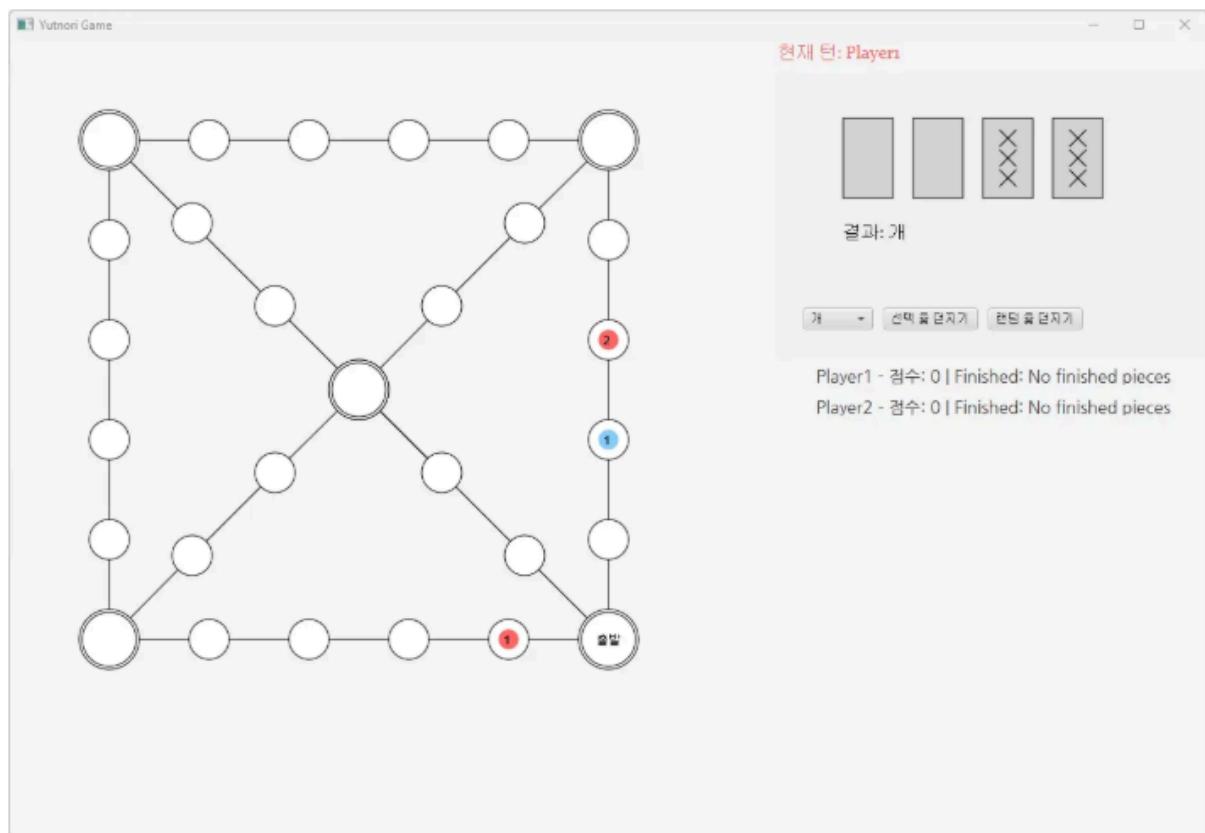




- Player1 (빨간색)의 1번 말과 2번 말이 같이 위치해 있다. 플레이어의 이해를 돋기 위해 해당 노드에 자신의 말이 몇 개 위치하는지 ' $\times k$ ' 형태로 노드 우측 상단에 명시하였다.
- 플레이어에게 말 선택을 안내할 때 다음과 같이 업기를 반영하여 명시하였다.

다음은 말이 결승점을 통과할 때의 UI 변화이다.





- Player1의 1번 말이 결승점을 통과하였다. 우측 하단 Player1의 점수가 통과한 말의 개수 (1개) × 100만큼 증가하였으며, 어떤 말이 통과하였는지 명시하였다.

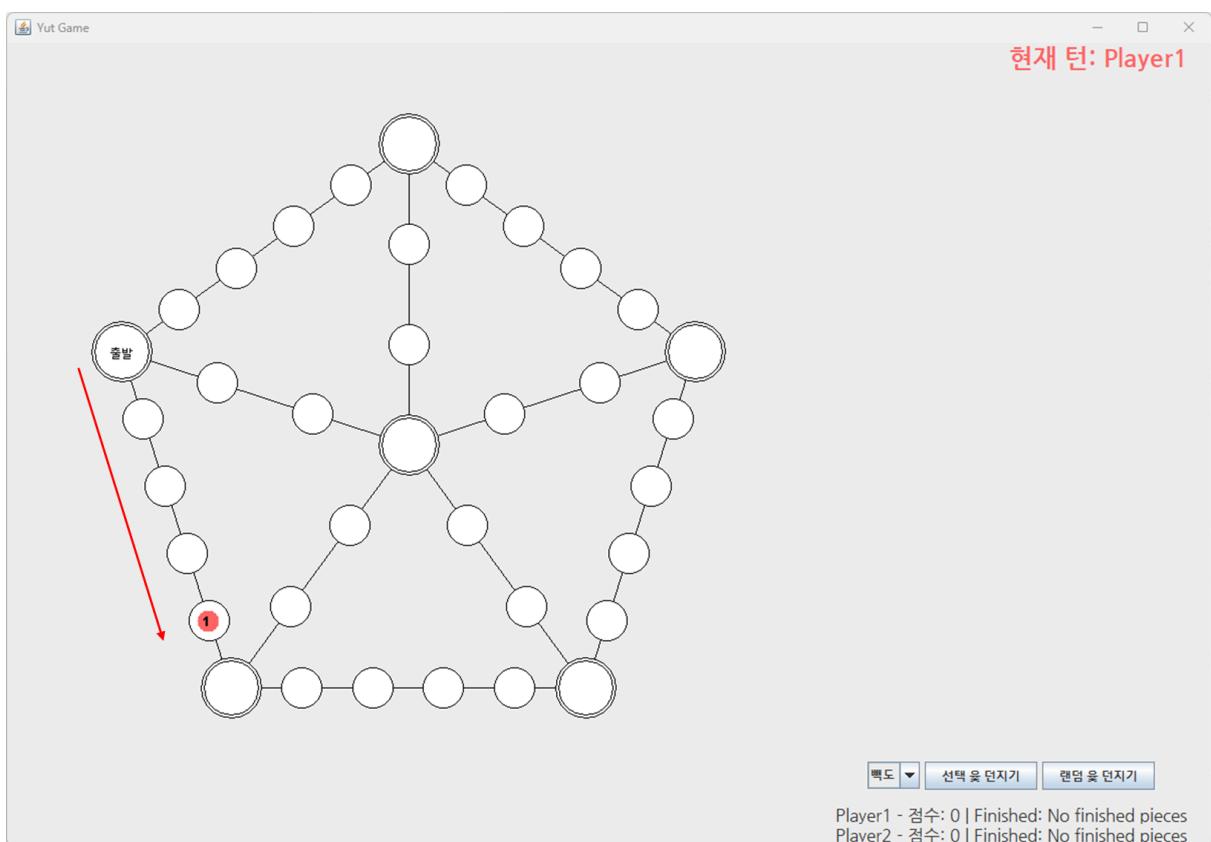
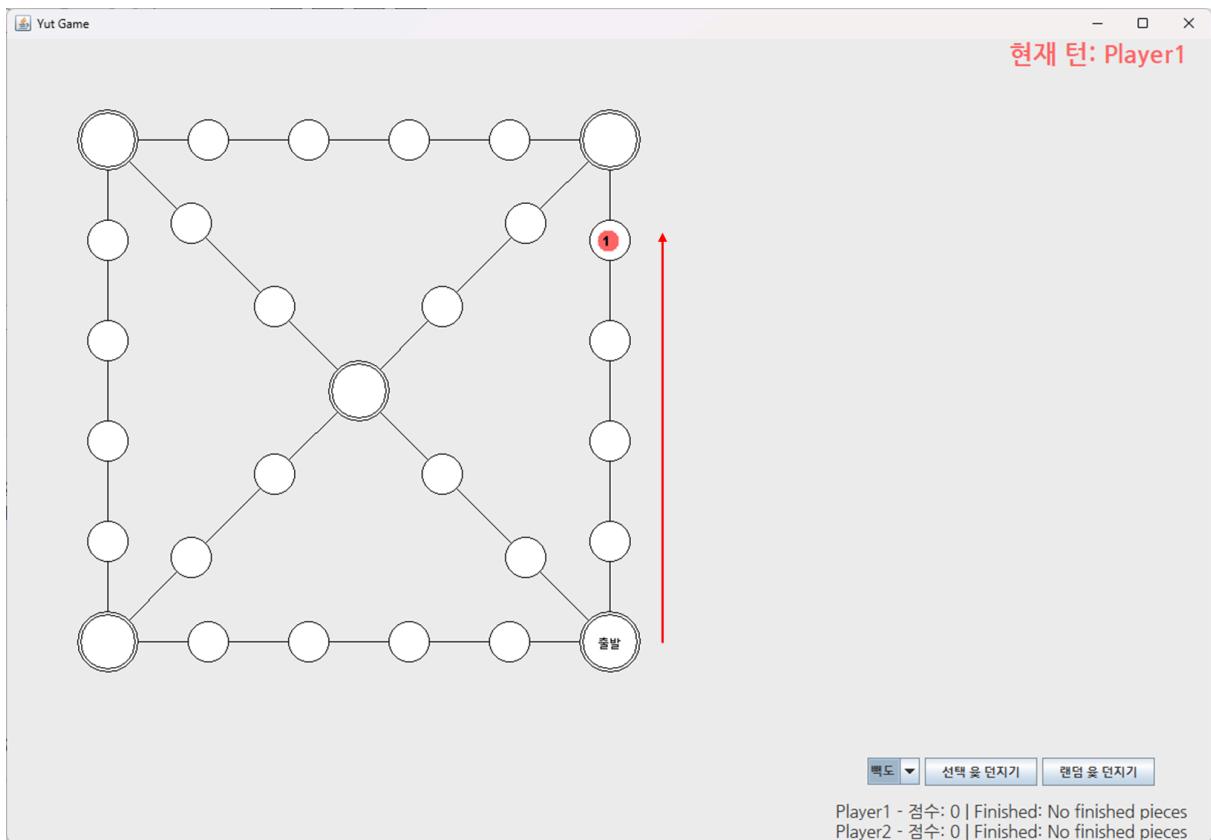
다음은 특정 플레이어가 모든 말을 완주하여 게임이 종료되는 경우 등장하는 UI이다.

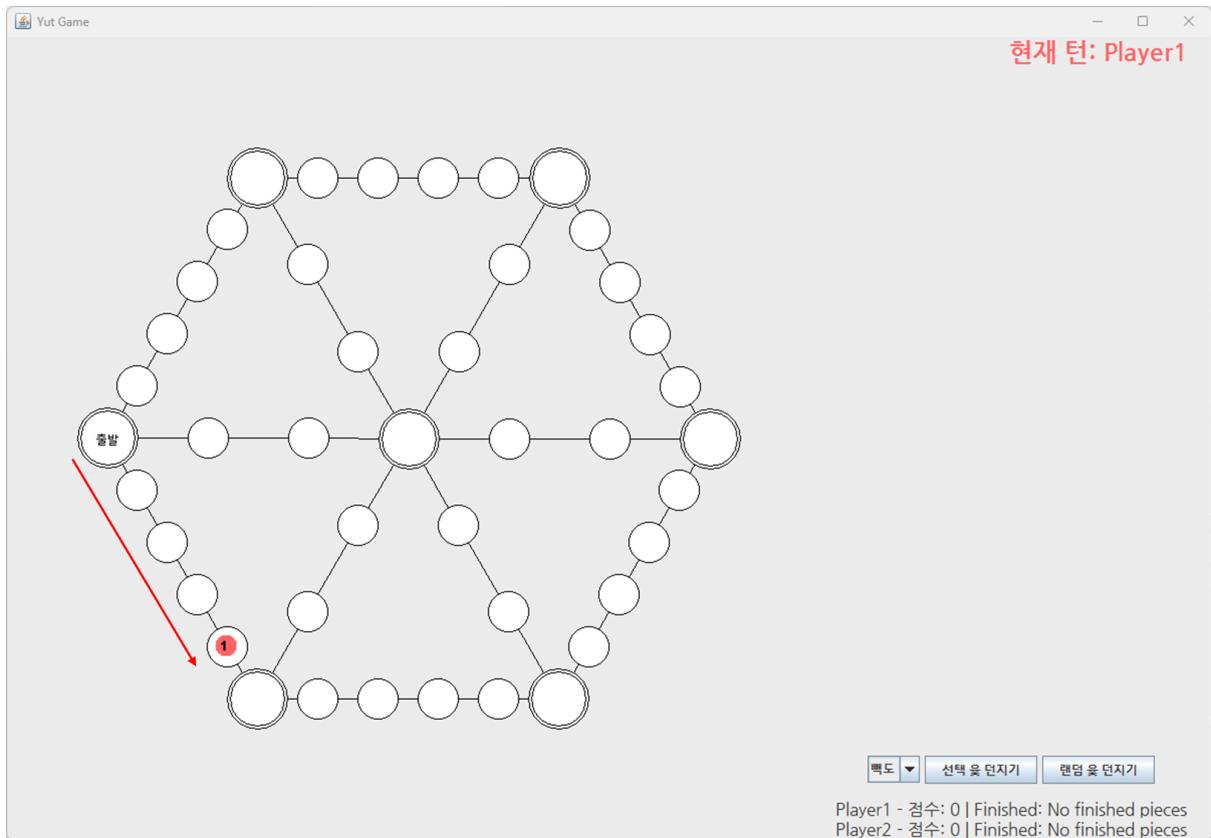


- 승리 플레이어를 다음과 같이 명시하고 사용자에게 게임 재시작 여부를 안내한다. 재시작을 선택한 경우 위의 보드 선택 UI로 돌아간다. 종료를 선택한 경우 프로그램을 종료 한다.

다음은 말 이동 로직을 Case별로 나열한 것이다.

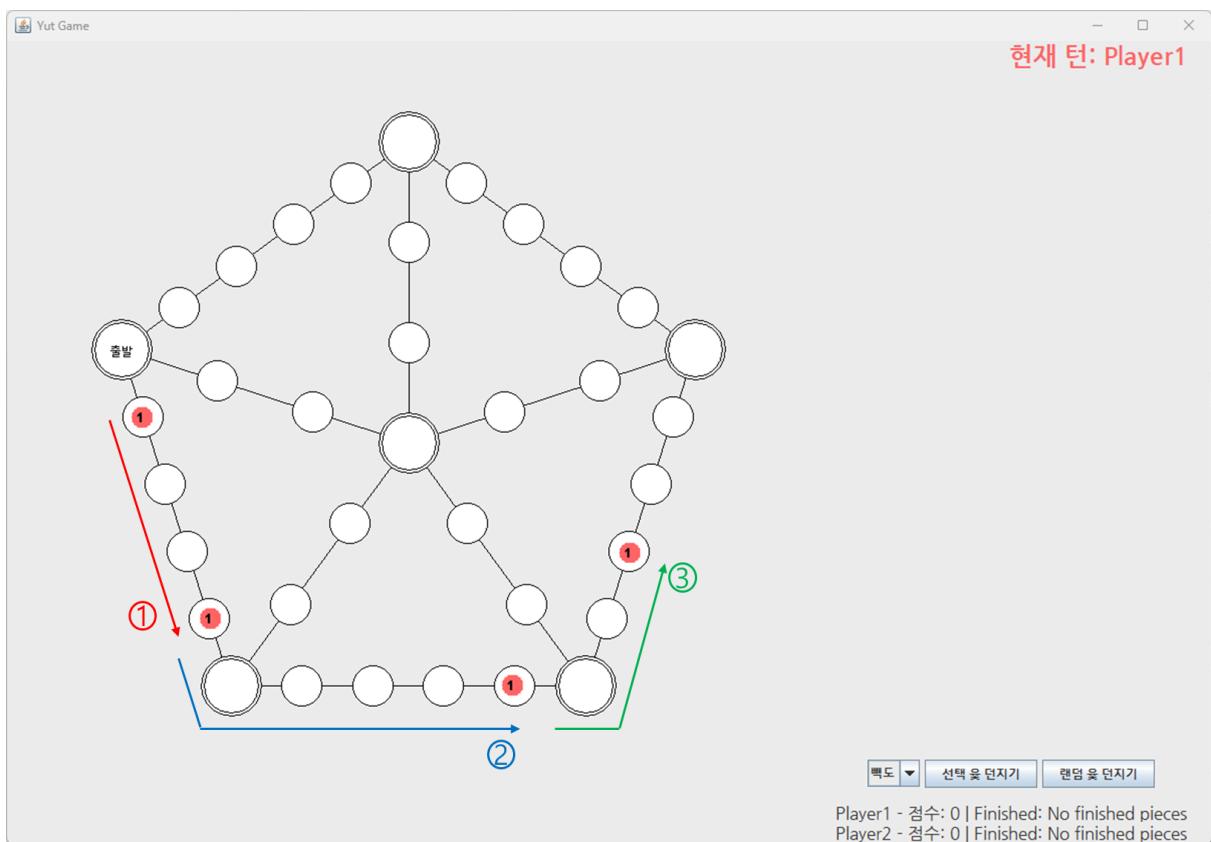
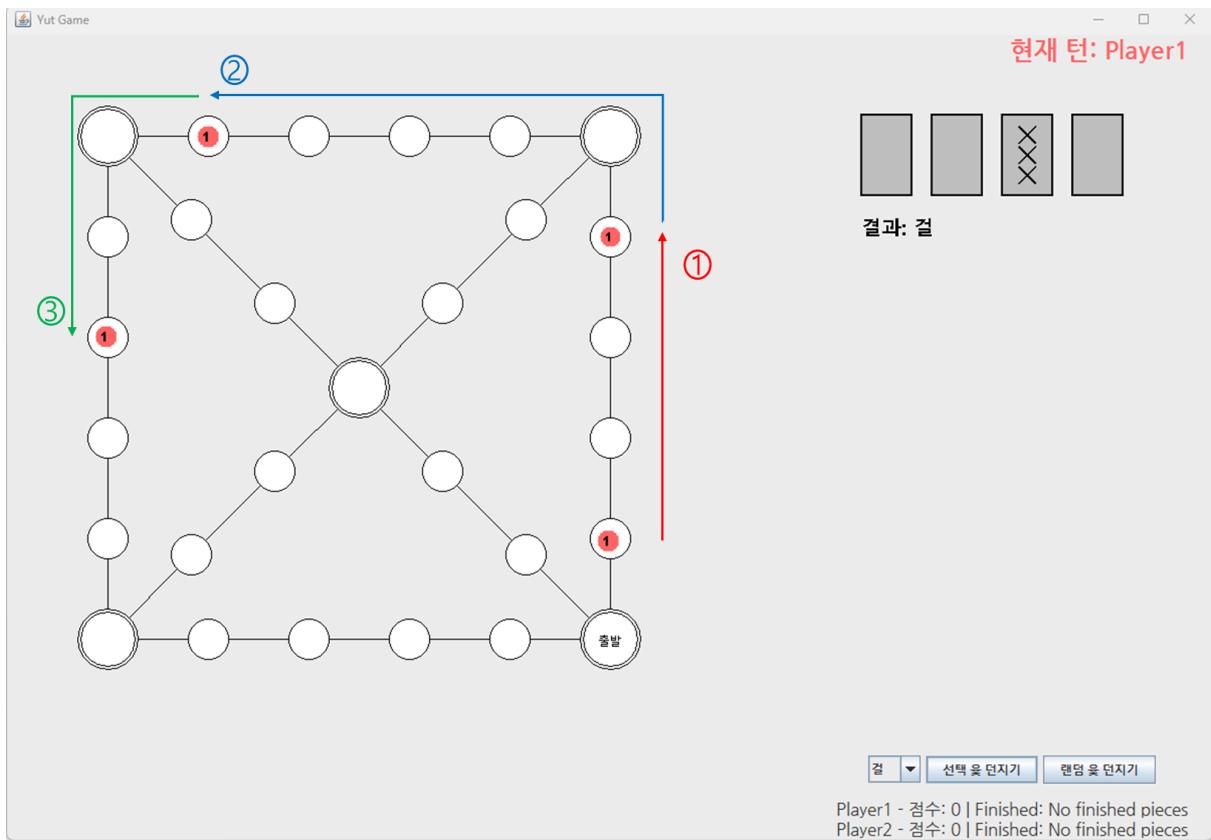
- **Case 1: 게임판에 말이 존재하지 않는 경우**

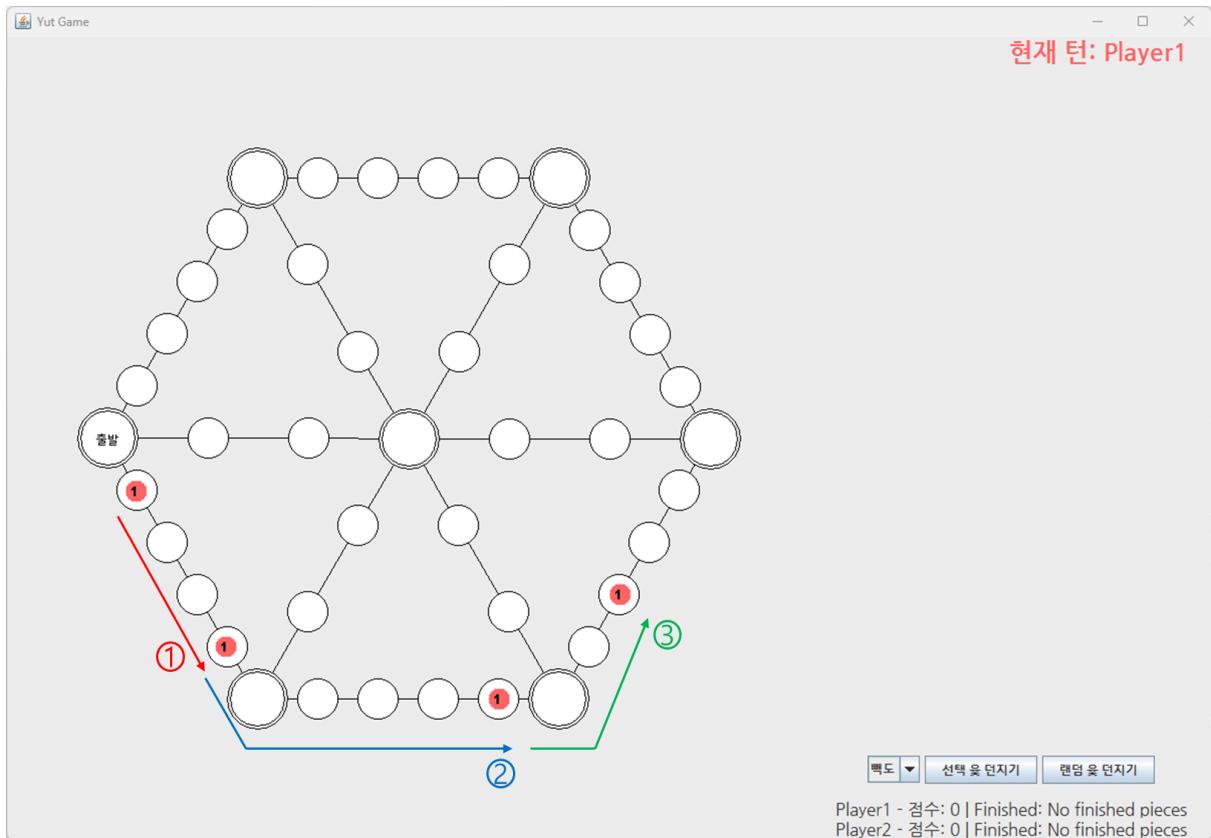




윷(4칸)만큼 이동할 때의 위치 변화와 방향 (외곽을 따라 이동)이다.

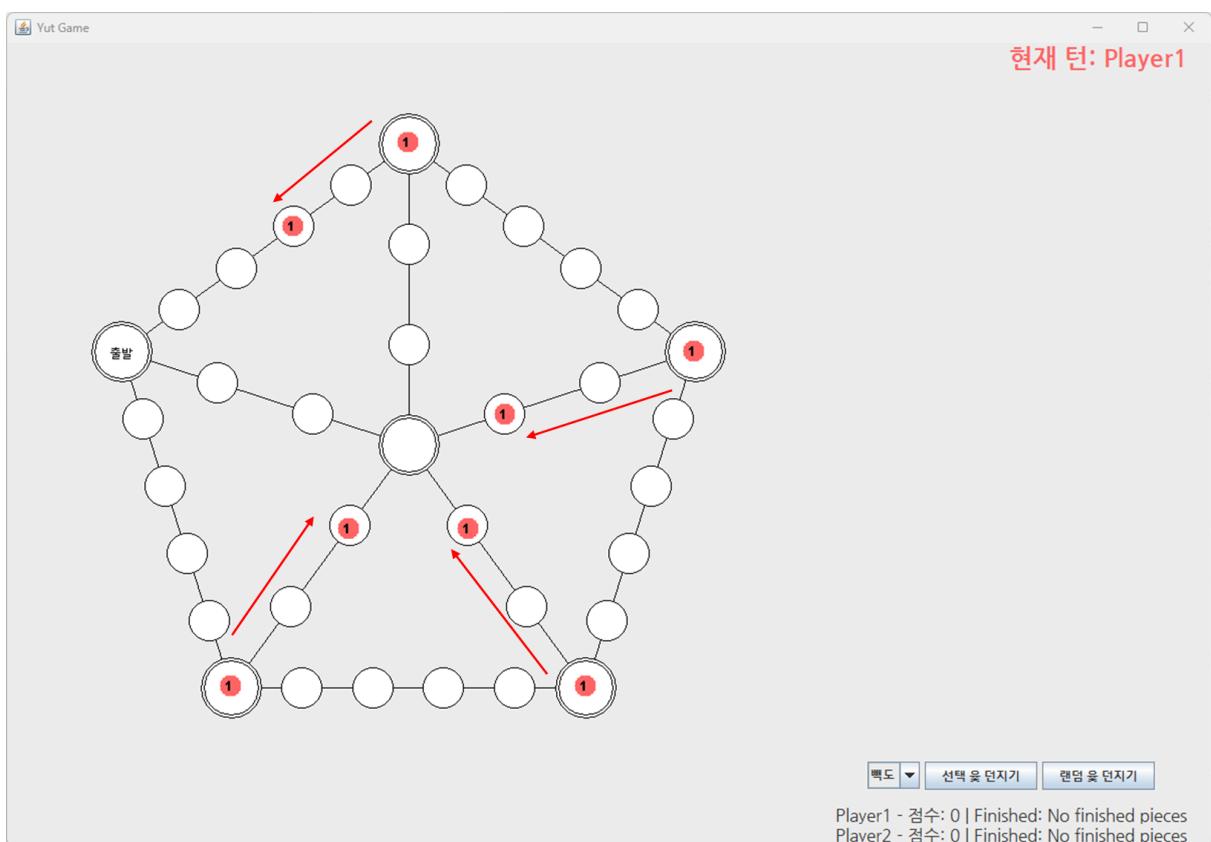
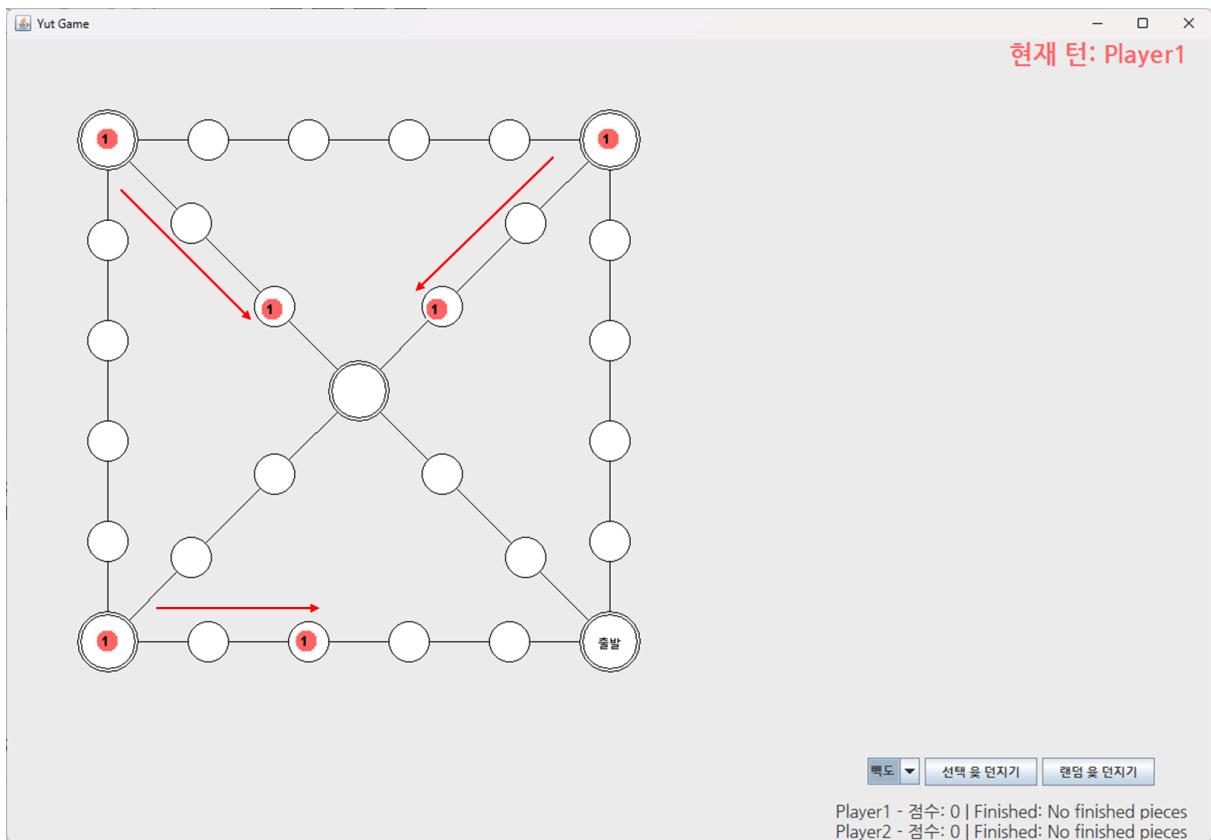
- **Case 2 : 외곽 (꼭짓점 제외) 노드에서 출발**

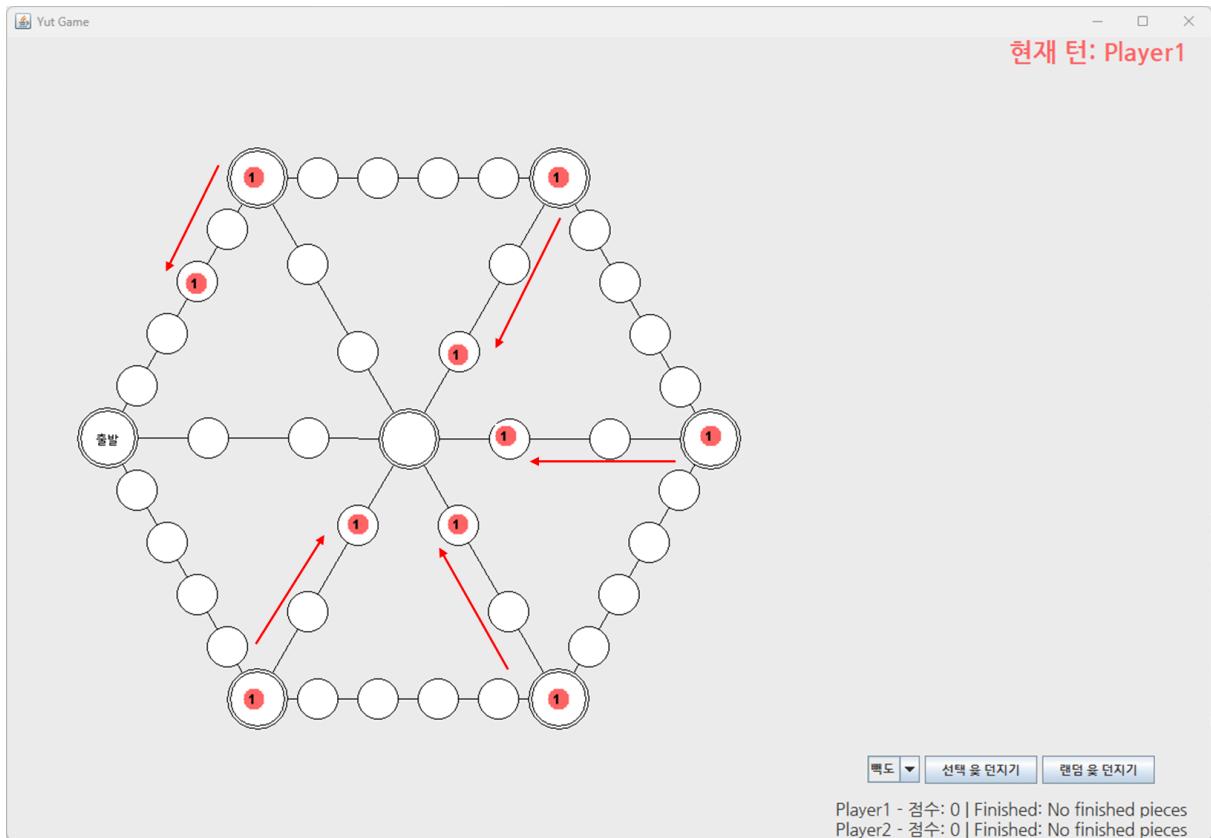




1번 말을 걸 → 모 → 걸 순서대로 이동시켰을 때의 위치 변화와 방향이다. 중심 노드, 꼭짓점 노드를 제외한 노드에서 출발하는 경우 위와 같이 외곽 경로를 따라 이동한다.

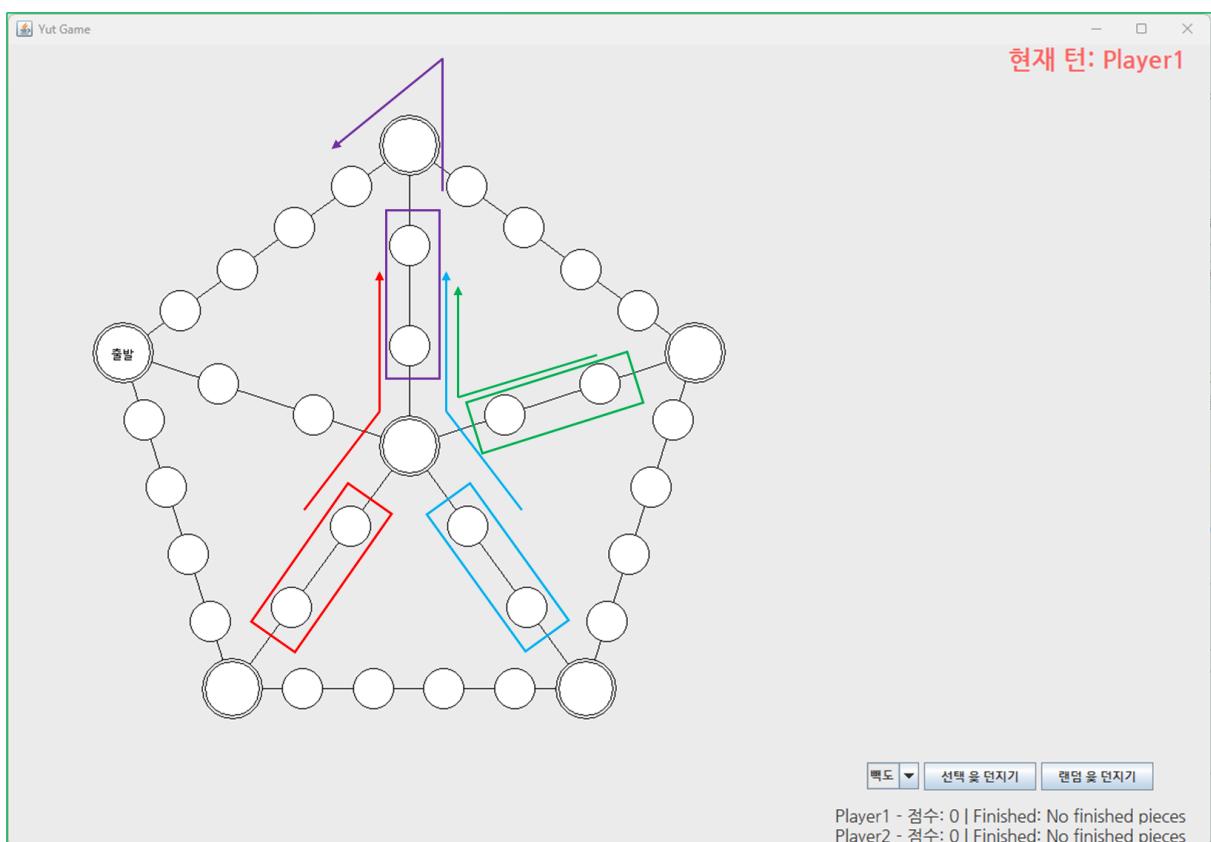
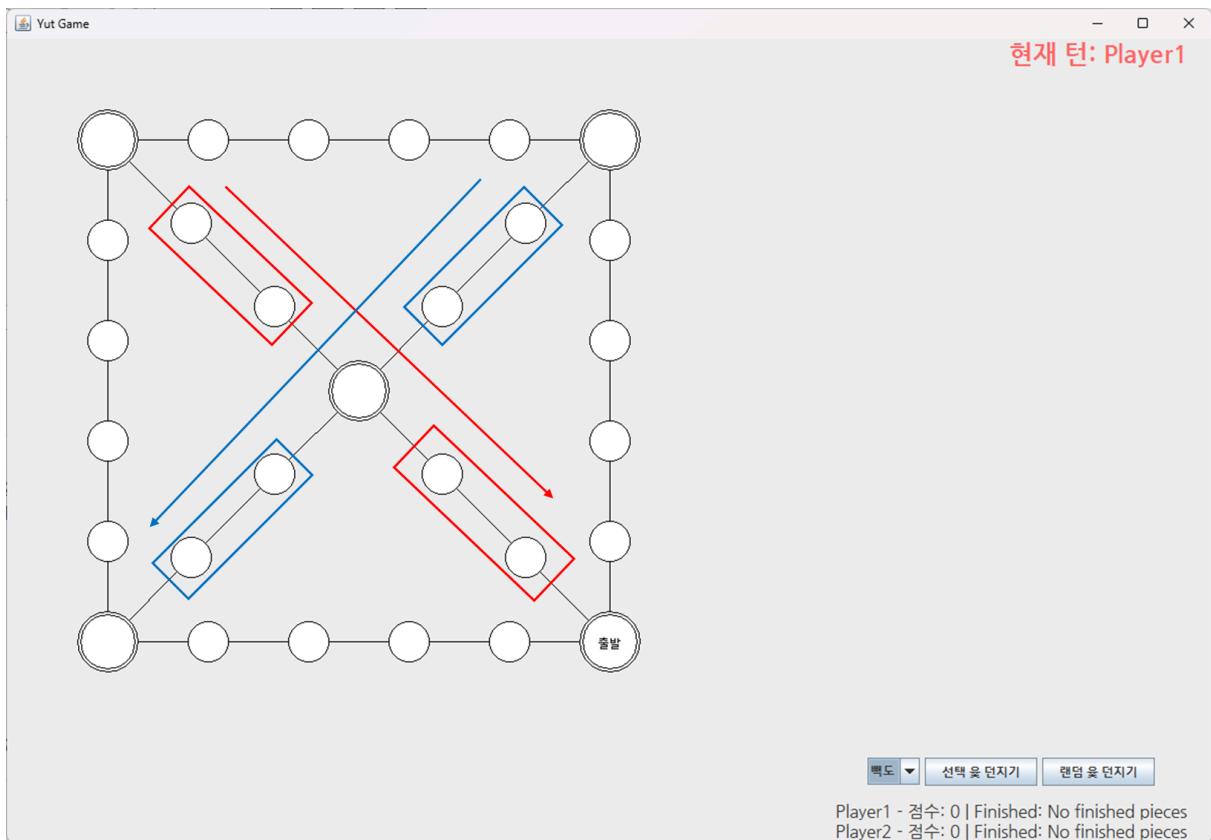
- **Case 3 : 꼭짓점 노드에서 출발**

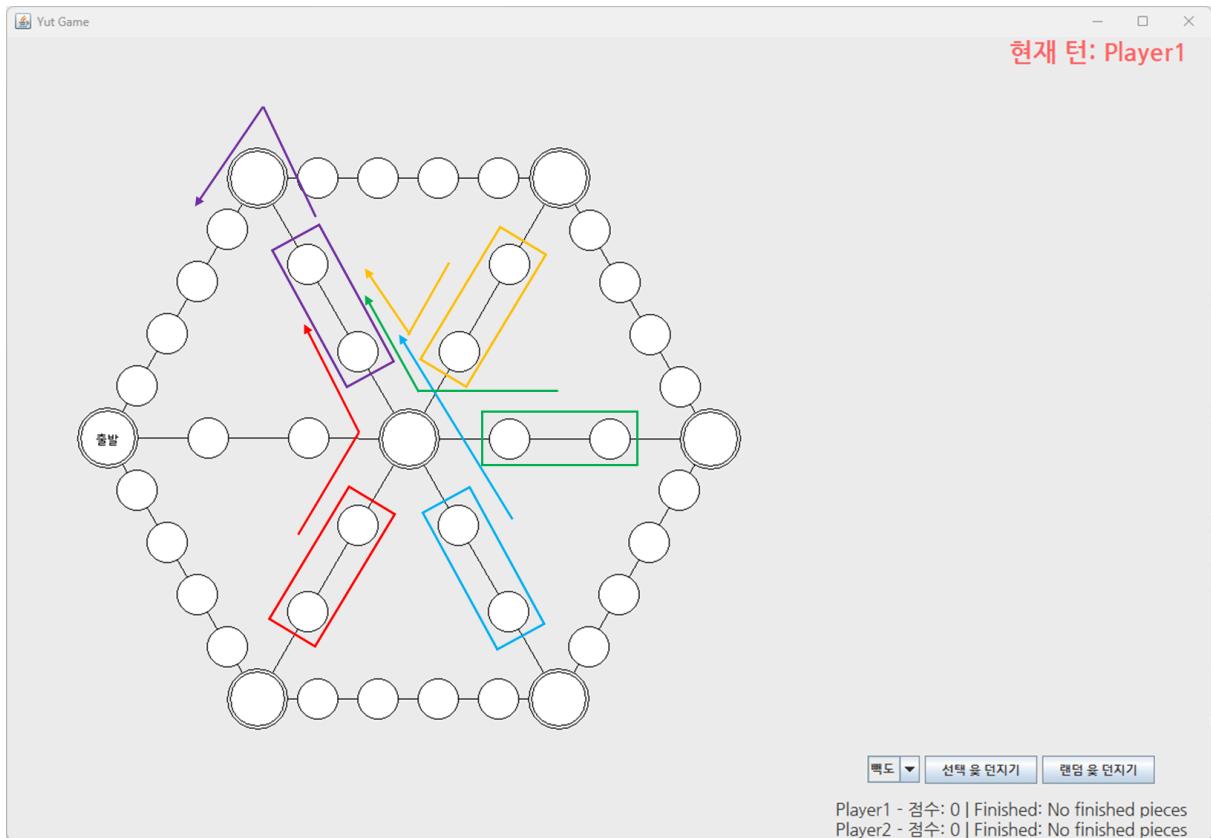




각 꼭짓점 노드에서 개(2칸)만큼 이동할 때의 위치 변화와 방향이다.

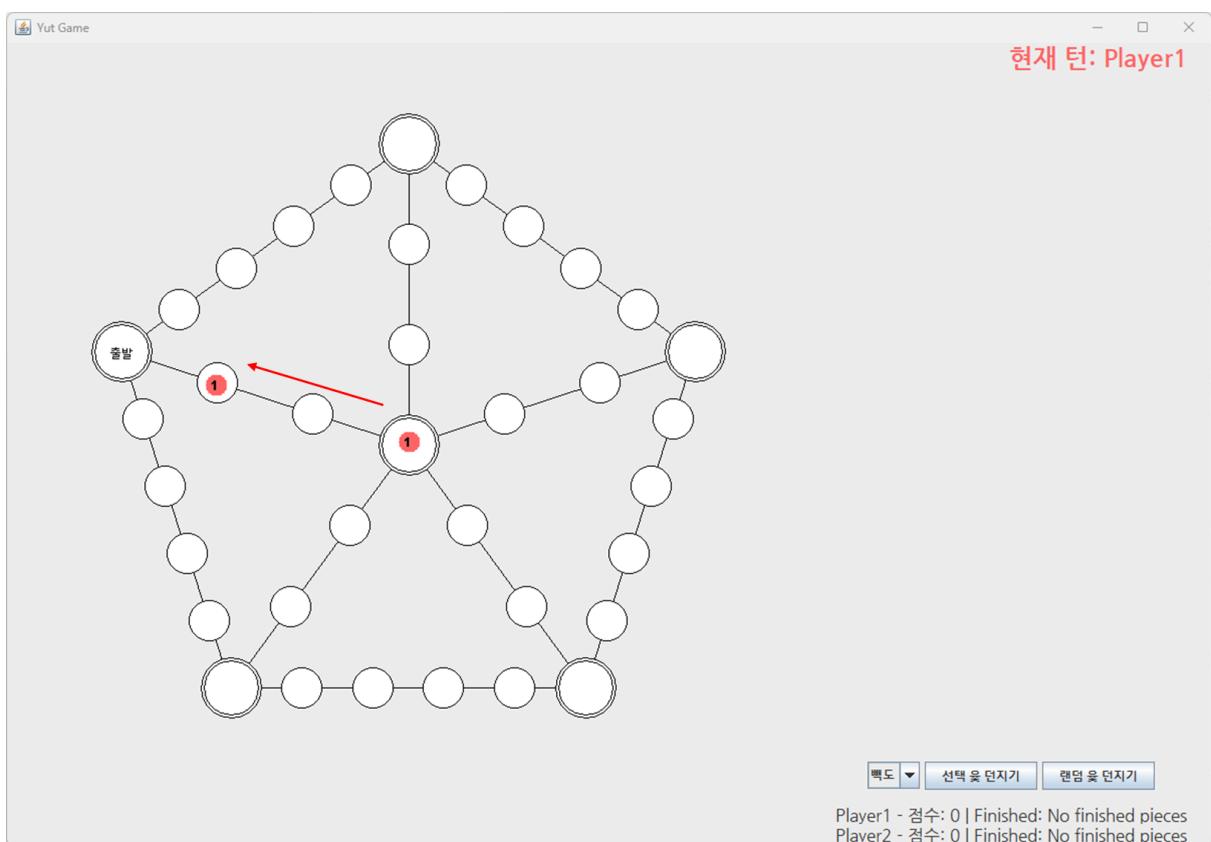
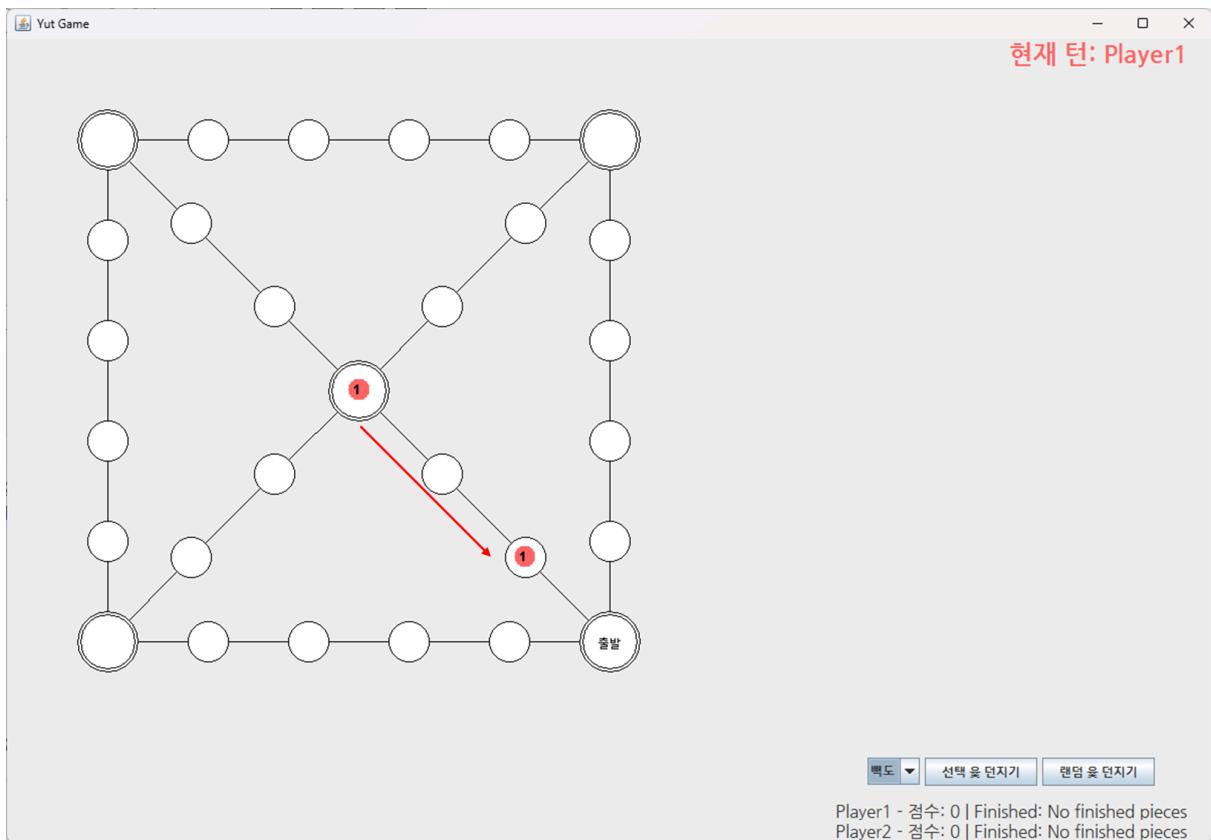
- SquareBoard : 상단 꼭짓점에서 출발할 때는 대각선 방향으로 이동하나, 좌측 하단 꼭짓점에서 이동할 때는 외곽을 따라 이동한다.
- PentagonBoard, HexagonBoard : 중심점 방향으로 이동하나, 종료 지점까지 더 가까운 경로가 존재하는 각각의 case에는 해당 경로를 따라 이동한다.
- **Case 4 : 내부 노드 (중심 제외)에서 출발**

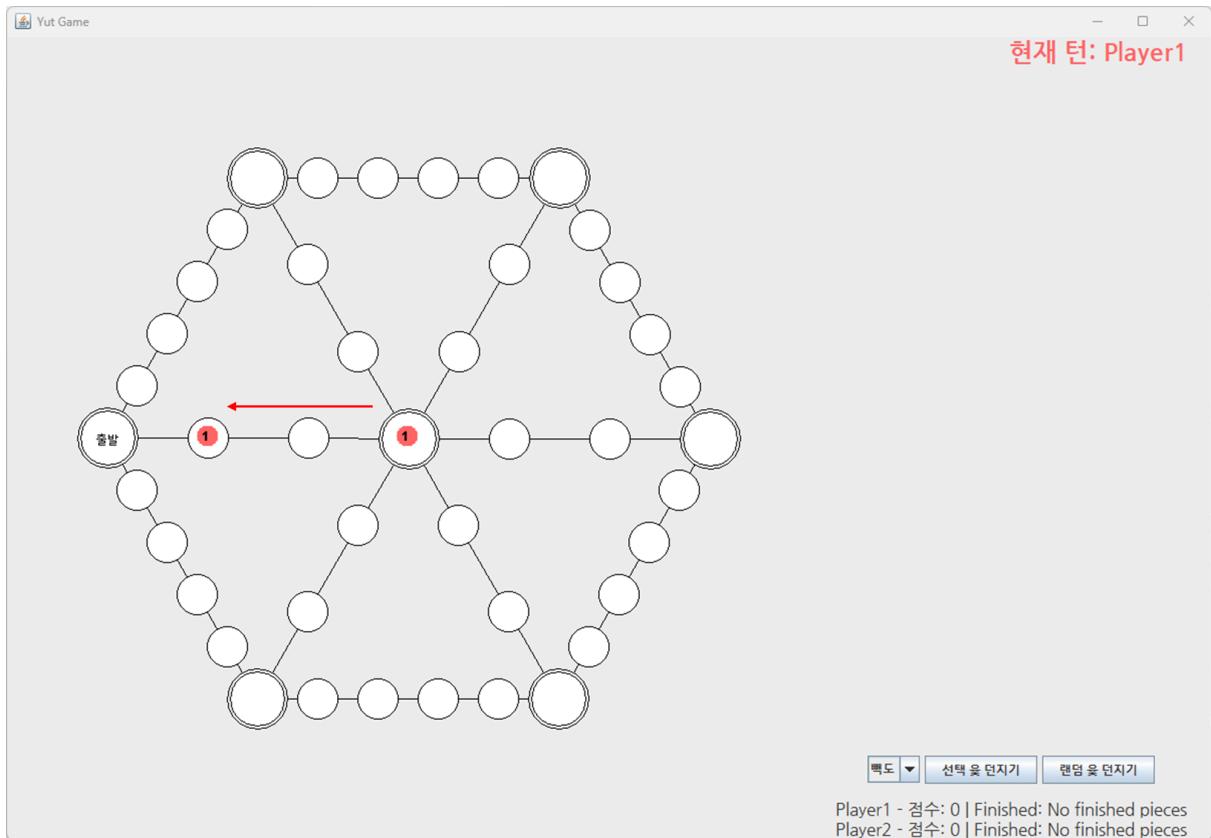




중심 노드를 제외한 내부 노드에서 출발할 때의 이동 방향이다.

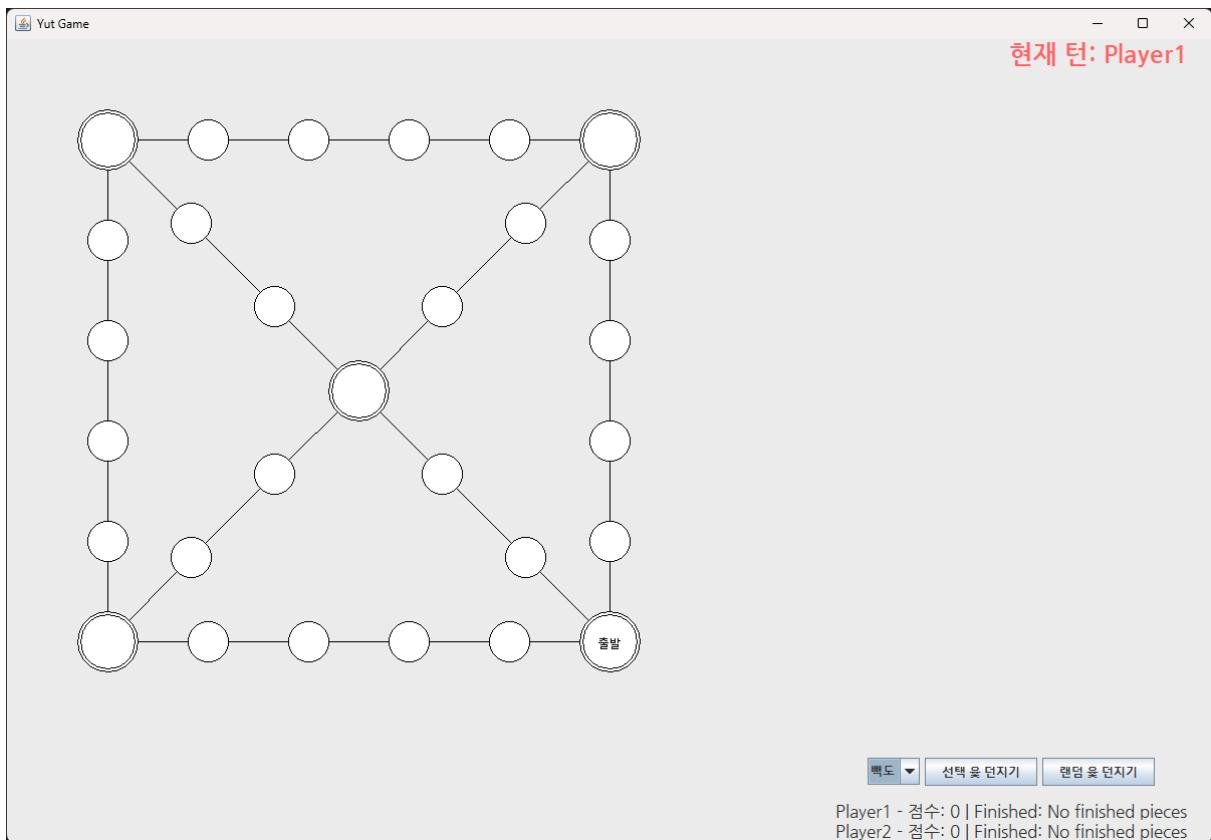
- SquareBoard : 대각선 방향으로 이동한다.
- PentagonBoard, HexagonBoard : 중심점 방향으로 이동한다. 중심점을 지나치는 경우 종료지점과 두 번째로 가까운 경로를 따른다. 단 중심점을 경유하는 것보다 종료 지점을 빠르게 도달하는 경로가 존재하는 경우 해당 경로를 따른다.
- **Case 5 : 중심 노드에서 출발**





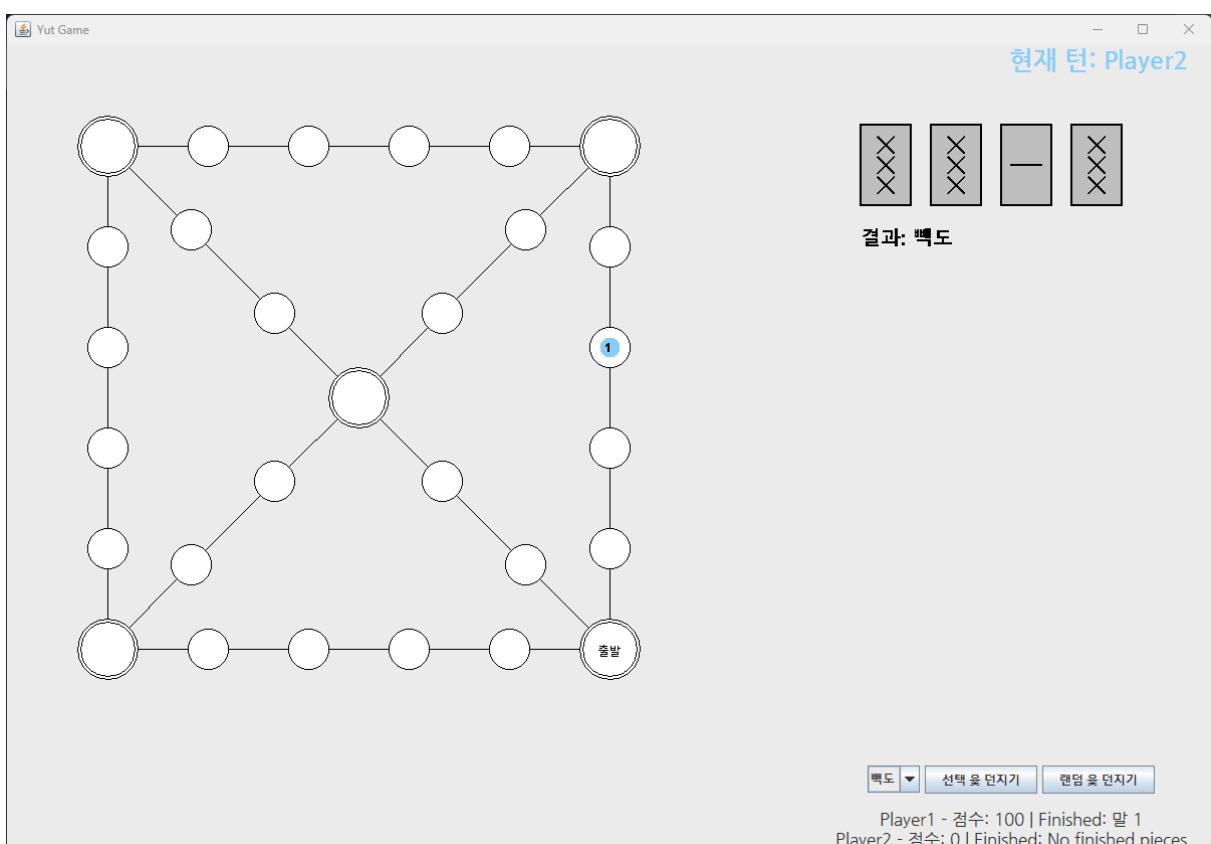
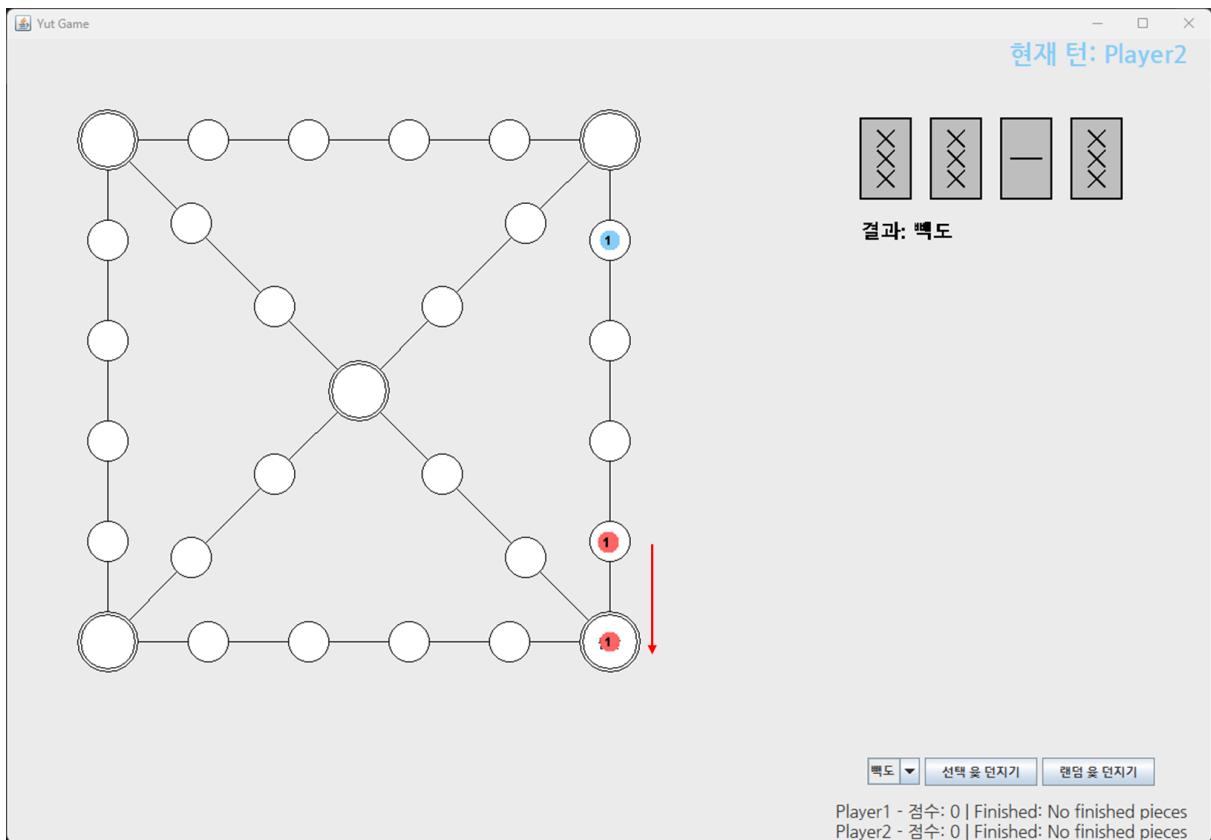
중심 노드에서 개(2칸)만큼 이동할 때의 위치 변화와 방향 (도착 지점을 향함)이다.

- **Case 6 : 이동하고자 하는 말이 게임판에 존재하지 않는 경우에서의 빼도**



위와 같이 게임판에 플레이어의 말이 없는 상황에서 빼도가 나오는 경우 말의 이동이 불가하다. 해당 내용을 새로운 UI를 통해 플레이어에게 안내한다.

- **Case 7 : 시작 지점에서 빼도가 나온 경우**



다음과 같이 Player1의 1번 말이 게임판 위에 올라갔다가 빽도를 통하여 출발점으로 돌아온 상황에서 빽도로 1번 말을 이동시키는 경우 해당 말은 완주 처리된다.

4. 테스트

이 테스트는 Yutnori 게임 프로젝트의 주요 로직이 안정적으로 동작하는지 검증하고, 핵심 기능에 대한 분기 조건들을 체계적으로 확인하여 신뢰성을 높이기 위해 수행되었습니다.

테스트 범위

- 모델 계층 (model)
 - Board, Cell, Player, Piece, SquareBoard, HexagonBoard, PentagonBoard
- 컨트롤러 계층 (controller)
 - GameController: 게임 흐름 전체
 - PieceMoveController: 말 이동, 그룹핑, 말 잡기 등 핵심 로직
- 뷰 계층은 (view): UI는 Swing 및 JavaFX 기반으로 구성되어 있어 직접 실행을 통해 수동 검증하였습니다. 각 버튼과 상호작용은 게임 흐름을 따라 실행해보며 정상 작동을 확인했고, 단위 테스트는 도메인 및 로직 중심으로 한정하였습니다.

테스트 환경

- 운영체제(OS): macOS (Apple Silicon 기반)
- Java 버전: Java 23.0.1 (2024-10-15), Java™ SE Runtime Environment (build 23.0.1+11-39)
- 빌드 도구: Gradle
- Gradle 설정: build.gradle 파일에 JavaFX, JUnit 5, Mockito 의존성 명시

- **테스트 프레임워크:** JUnit 5 (org.junit.jupiter:junit-jupiter), Mockito 5
- **JavaFX 버전:** JavaFX 21 (javafx-controls, javafx-fxml 모듈 사용)
- **개발 환경(IDE):** IntelliJ IDEA (테스트 실행 및 커버리지 측정에 사용)
- **테스트 실행 방법:** ./gradlew test 명령어 또는 IntelliJ의 테스트 실행 기능 사용
- **커버리지 도구:** IntelliJ IDEA의 기본 내장 커버리지 도구 사용 (JaCoCo는 별도 사용하지 않음)

테스트 코드 설명

Model



MockBoard

테스트 코드 전반에서 사용된 MockBoard는 실제 게임에서 사용하는 SquareBoard, PentagonBoard, HexagonBoard 등의 복잡한 보드 구현체를 대신하여, 테스트 목적에 최적화된 간소화된 보드입니다.

이 클래스는 Board 추상 클래스를 상속하여 구현되었으며, 오직 기능 검증만을 위한 구조로 구성되었습니다. 테스트 환경에서는 복잡한 경로 분기나 도착 셀 계산 로직이 오히려 변수로 작용할 수 있으므로, MockBoard는 5개의 셀만 생성하고, 단순히 현재 위치에서 정해진 step만큼 이동하는 구조를 통해 이동 경로를 예측 가능하게 만듭니다.

또한 isCorner()와 isCentre() 등의 판단 메서드도 실제 조건이 아닌 테스트 전용 조건(예: 짹수 ID는 Corner, ID 2는 Centre 등)으로 구현되어 있으며, nodePositions, outerPath, innerPath 또한 최소한의 값만 구성되어 있어 테스트 통과에 필요한 기본 요건만 충족합니다.

이처럼 MockBoard는 테스트의 신뢰성과 반복성을 높이기 위한 도구로, 실제 게임 진행용 보드가 아님을 명시합니다. 이를 통해 로직 자체의 기능 테스트와 예상 결과 확인이 정확하게 이루어질 수 있습니다.

GameTest

이 테스트 클래스는 Game 클래스의 기본 상태, 플레이어 순환 로직, 설정된 보드 및 말 개수 관련 기능을 검증합니다. Game은 게임 전반을 관리하는 중심 객체로, 초기 상태와 흐름이 정확히 유지되는지를 확인하는 것이 핵심입니다.

테스트 메서드 설명

- **testInitialState()**
 - 목적: Game 객체 생성 직후의 상태가 기대한 대로 초기화되었는지 확인
 - 확인 항목:
 - 플레이어 수, 순서
 - 보드 객체의 정확성
 - 플레이어당 말 개수 설정 (pieceNumPerPlayer)
- **testGetCurrentPlayer()**
 - 목적: 현재 플레이어가 올바르게 반환되는지 확인
 - 방식: 초기 상태에서 getCurrentPlayer() 호출 → 이후 nextPlayer() 호출 후 상태 변화 확인
- **testNextPlayerCyclesCorrectly()**
 - 목적: 플레이어가 두 명일 때 nextPlayer() 호출이 올바르게 순환하는지 확인
 - 이유: 턴제 게임에서 순환 로직은 게임의 핵심 흐름을 제어하는 요소이기 때문에 정확성이 중요
- **testGetPieceNumPerPlayer()**
 - 목적: 생성 시 설정한 말 개수가 내부적으로 정확히 저장되었는지 확인

테스트 전략

- 단일 책임 원칙에 따라 Game 클래스의 핵심 기능만 집중 테스트
- 실제 보드 로직은 MockBoard를 통해 추상화하여 외부 의존성 제거
- Player.resetCounter() 호출을 통해 일관된 테스트 환경 보장

BoardTest

이 테스트 클래스는 Board의 추상 클래스를 상속한 MockBoard를 통해 공통 동작을 검증합니다. 실제 게임 보드(SquareBoard, HexagonBoard, PentagonBoard)의 기반이 되는 핵심 기능들에 대한 테스트입니다.

테스트 메서드 설명

- **testGetAndSetCells()**
 - 목적: Board가 setCells()로 셀 리스트를 설정한 후 getCells()를 통해 동일한 리스트를 반환하는지 확인
 - 이유: 내부 cells 필드의 세터/게터 정상 작동 여부 검증
- **testgetNodePositions()**
 - 목적: 노드 ID와 위치를 저장하는 nodePositions 맵이 제대로 초기화되었는지 확인
 - 이유: 보드 위에 말을 그릴 때 위치 정보가 필수이므로, 맵이 null이 아니고 적절한 key 값을 포함하는지 체크
- **testgetNodePositionById()**
 - 목적: 특정 노드 ID(예: 0)에 대한 위치 정보가 null이 아닌지 확인
 - 이유: 보드 좌표계 기반 렌더링 로직에서 null 좌표가 발생하지 않아야 하므로 방어적으로 검증
- **testGetOuterAndInnerPath()**
 - 목적: outerPath, innerPath가 비어 있지 않고 초기화된 상태인지 확인
 - 이유: 각 보드 타입에서 정의한 경로 정보가 게임 로직에 영향을 주므로 정상 존재 여부 확인
- **testRadiusConstants()**
 - 목적: RADIUS와 CORNER_RADIUS 상수가 정상적으로 설정되어 있는지 검증
 - 이유: 그래픽 표시 및 충돌 판정 등에 사용되는 반지를 값의 일관성 유지 확인

테스트 전략

- 추상 클래스를 직접 테스트할 수 없기 때문에 테스트용 MockBoard를 활용
- 보드의 위치, 경로, 셀 설정 등 **공통 기반 기능을 중심으로 테스트**

- 좌표, 경로, 셀 목록이 null이 아닌지, 값이 정확한지 등 **기본적인 방어 코드 역할을 테스트 코드로 수행**
-

CellTest

이 테스트 클래스는 Cell 객체가 말(Piece)을 올바르게 추가하거나 제거하는지, 특히 그룹핑된 말의 동작을 정상적으로 처리하는지를 중점적으로 검증합니다. Cell은 게임 보드의 핵심 단위로, 상태 유지와 그룹 처리의 정확성이 게임 로직 전반에 영향을 미칩니다.

테스트 메서드 설명

- **testGetId()**
 - 목적: 셀의 ID가 생성 시 올바르게 설정되었는지 확인
- **testAddSinglePiece()**
 - 목적: 단일 말이 정상적으로 셀에 추가되는지 확인
 - 검증: stackedPieces 리스트에 정확히 포함되는지
- **testAddGroupedPiecesTogether()**
 - 목적: 그룹핑된 말(리더 + 팔로워)을 추가할 때 자동으로 전체 그룹이 함께 추가되는지 검증
 - 포인트: addPiece() 호출 시 그룹 전체가 스택에 포함되어야 함
- **testAddGroupedPieceAlreadyInCell_doesNotDuplicate()**
 - 목적: 팔로워 말이 이미 셀에 있을 경우, 리더 말 추가 시 중복 없이 처리되는지 확인
- **testRemoveSinglePiece()**
 - 목적: 셀에 존재하는 단일 말을 정상적으로 제거하는지 확인
- **testRemoveGroupedPiecesTogether()**
 - 목적: 그룹 리더 제거 시 팔로워도 함께 제거되는지 확인
- **testRemoveGroupedPieceThatIsNotInCell_doesNotThrowOrChange()**
 - 목적: 셀에 존재하지 않는 그룹 구성원을 제거해도 예외 없이 안정적으로 동작하는지 검증
- **testRemoveGroupedPieceNotInCell_skipsRemovalGracefully()**

- 목적: 내부적으로 그룹은 구성되어 있지만 셀에는 실제 존재하지 않는 말을 제거할 때, 예외 없이 graceful하게 처리되는지 확인

테스트 전략

- 그룹핑된 말의 동시 처리 여부와 중복 추가/제거 회피가 핵심 포인트
 - 말의 추가/제거 작업은 Cell과 Piece 간의 연계성을 반영하여 테스트 구성
 - 예외 발생 여부와 상태 유지에 초점을 맞춘 안정성 테스트 포함
-

PieceTest

이 테스트 클래스는 게임 말(Piece) 객체의 주요 속성과 동작을 검증합니다. 이동 상태, 말의 완료 상태(finished), 그룹핑 로직, 색상 반환, 초기화(reset) 등 핵심적인 기능들이 포함되어 있으며, 게임 흐름에서 자주 변경되는 객체인 만큼 **상태 변화의 신뢰성** 확보가 핵심 목표입니다.

테스트 메서드 설명

- **testInitialState()**
 - 생성 직후 Piece의 초기 상태(위치, 보드 탑승 여부, 완료 여부, 색상 등)를 검증합니다.
 - ID에 따라 색상이 결정되는 로직도 포함되어 있어 기본 색상도 확인합니다.
- **testMoveToUpdatePositionAndOnBoard()**
 - moveTo() 호출 시 위치와 isOnBoard() 상태가 올바르게 갱신되는지 확인합니다.
- **testMoveToNull_whenNoPreviousPosition_doesNotThrow()**
 - 말이 아직 보드에 올라가지 않은 상태에서 null로 이동 시도할 경우 예외 없이 작동하는지 테스트합니다.
- **testFinishClearsPositionAndSetsFinished()**
 - 말이 완료 처리되면 위치가 초기화되고 보드 위에서 내려오는지 확인합니다.
- **testResetRestoresInitialState()**
 - reset() 호출 시 말의 위치, 상태, 기록 등 모든 필드가 초기화되는지 확인합니다.
- **testAddGroupingPieceCreatesBidirectionalLink()**

- 그룹핑된 말이 서로 연결되었는지(특히 리더-팔로워 구조) 확인합니다.
- **testAddGroupingPiece_whenListIsNull_initializesList()**
 - 내부 그룹 리스트(moveTogetherPiece)가 null일 때도 안전하게 동작하는지 리플렉션으로 검증합니다.
- **testAddGroupingPiece_whenAlreadyGrouped_doesNotDuplicate()**
 - 중복 그룹핑이 발생하지 않도록 제어하는지 확인합니다.
- **testResetGrouping_withEmptyGroup_doesNotThrow()**
 - 그룹핑이 없는 상태에서 resetGrouping()을 호출했을 때 예외 없이 처리되는지 확인합니다.
- **testGetGroupingPieces_whenListIsNull_returnsEmptyList()**
 - 그룹 리스트가 null일 경우에도 안전하게 빈 리스트를 반환하는지 확인합니다.
- **testResetGroupingBreaksLinks()**
 - 리더-팔로워 구조를 가진 말에서 그룹을 리셋할 경우 링크가 해제되는지 확인합니다.
- **testResetGrouping_resetsChildrenGroupLeader()**
 - 그룹 리셋 시 팔로워의 groupLeader도 null로 초기화되는지 확인합니다.
- **test GetAllGroupedPieces_whenAlone_containsSelfOnly()**
 - 그룹에 속하지 않은 말은 getAllGroupedPieces() 호출 시 자기 자신만 포함하는지 확인합니다.
- **testReset_whenPositionIsNull_doesNotThrow()**
 - 말의 위치가 null일 때 reset()을 호출해도 예외가 발생하지 않아야 함을 검증합니다.
- **testGetColor_returnsDefaultWhenPlayerIdOutOfRange()**
 - 정의되지 않은 플레이어 ID에 대해 말 색상이 기본값(GRAY)으로 설정되는지 확인합니다.

테스트 전략

- 객체 생성부터 상태 변경, 예외 처리까지 **객체 상태의 생명주기 전반을 포괄적으로 테스트**
- 리플렉션을 활용해 null 상태의 예외적 상황까지 커버

- 게임 중 가장 자주 상태가 바뀌는 도메인인 만큼 **안정성과 견고함을 확보**하는 데 중점
-

PlayerTest

이 테스트 클래스는 Yutnori 게임의 핵심 참가자인 Player 객체의 기본 정보, 점수, 말 (Piece) 관리 기능의 정확성과 예외 상황을 검증합니다. ID 부여, 점수 누적, 말 추가 기능이 모두 게임 로직에 필수적이므로, **기초적이지만 매우 중요한** 검증 단위입니다.

테스트 메서드 설명

- **testPlayerNameAndId()**
 - 각 플레이어의 이름과 자동 부여되는 ID가 순서대로 잘 설정되는지 확인합니다.
 - Player.resetCounter() 호출 후 ID가 1부터 시작됨을 보장합니다.
- **testAddScoreAndGetScore()**
 - addScore() 메서드로 점수를 누적한 결과가 정확하게 반영되는지 확인합니다.
- **testAddPieceAndGetPieces()**
 - 플레이어가 보유한 말(Piece) 리스트에 대해 추가 및 검색이 정확히 동작하는지 검증합니다.
- **testResetCounter()**
 - 정적 ID 카운터를 초기화하고 새로운 플레이어 생성 시 ID가 다시 1부터 시작하는지 확인합니다.

테스트 전략

- Player 클래스는 비교적 단순한 데이터 보관 역할이지만, ID 관리와 점수, 말 소유 상태 등 게임 전체 흐름에 **직접적 영향을 미치는 요소들을** 포함하고 있습니다.
 - 테스트에서는 **상태 초기화 → 행위 수행 → 결과 확인**의 3단계로 테스트 구성
 - 여러 테스트 간 ID 충돌을 방지하기 위해 **매 테스트마다 ID 초기화 수행** (@BeforeEach에서 Player.resetCounter() 호출)
-

YutTest

이 테스트 클래스는 Yut 객체의 핵심 기능인 윷 던지기 결과 처리, 예외 처리, 상태 조회에 대한 모든 시나리오를 포괄적으로 검증합니다. 윷놀이라는 게임의 본질이 “말의 움직임을 결정하는 주사위(윷)“에 달려 있기 때문에, 해당 클래스는 게임 전반의 무작위성과 결과 누적 처리에 있어 핵심적인 역할을 합니다.

테스트 메서드 설명

- **testThrowSelectYut_validInput_shouldBeAddedToResult()**
 - throwSelectYut() 메서드가 유효한 값(-1, 1~5)을 정확히 results에 누적하는지 확인합니다.
- **testThrowSelectYut_invalidInput_shouldThrowException()**
 - 비정상 입력값(0, -2, 6)에 대해 예외(IllegalArgumentException)가 발생하는지 테스트합니다.
- **testThrowSelectYut_boundaryValidInputs()**
 - 경계값(-1, 1, 5)에 대해서는 정상적으로 동작함을 확인합니다. (명시적 예외 방지 검증)
- **testThrowRandomYut_distribution()**
 - throwRandomYut()이 랜덤 값을 생성할 때 유효 범위 내에서 고르게 결과가 나오는지 간단히 확률 분포 테스트를 합니다. (10,000회 반복 → 모든 값이 최소 1회 이상 등장하는지 확인)
- **testGetResults_returnsSameList()**
 - 결과 리스트가 정확히 누적되며 반환되는지 확인합니다.
- **testGetLastResult_shouldReturnLastResult()**
 - 가장 마지막 윷 결과가 정확히 반환되는지 확인합니다.
- **testGetLastResult_whenEmpty_shouldThrowException()**
 - 결과가 없을 때 getLastResult() 호출 시 IllegalStateException이 발생하는지 확인합니다.
- **testIsEmpty_initialState_shouldReturnTrue()**
 - 객체 생성 직후에는 isEmpty()가 true를 반환해야 함을 확인합니다.
- **testIsEmpty_afterThrow_shouldReturnFalse()**
 - 하나라도 윷 결과가 누적되면 isEmpty()는 false를 반환해야 함을 테스트합니다.

테스트 전략

- 이 테스트는 예외 처리, 상태 조회, 누적 결과 확인, 무작위성 검증 등 Yut 클래스의 기능적 경계값과 확률적 요소까지 포함하여 체계적으로 커버합니다.
- 특히 throwRandomYut()의 경우 테스트에서 확률 분포의 왜곡 여부를 최소화하기 위해 반복 횟수(10,000회)를 설정해 검증 범위를 넓혔습니다.
- 예외가 발생하는 조건과 발생하지 않아야 하는 조건을 명확히 분리하여 작성함으로써 안정성 및 오류 회피 능력도 함께 검증하고 있습니다.

SquareBoardTest

이 테스트 클래스는 SquareBoard가 올바른 경로 탐색 로직과 구조적 정의(예: 코너/센터 노드 식별 등)를 따르고 있는지 검증합니다. SquareBoard는 윷놀이판의 경로를 구성하며, 말의 분기 이동, 중앙 진입 조건 등 게임 로직의 핵심 동작 경로를 결정합니다.

테스트 메서드 설명

- **testInitialCellWhenNull()**
 - 현재 위치가 null인 경우(말이 아직 보드에 진입하지 않은 상태), 기본 시작 셀(0번 셀)을 반환해야 함을 검증합니다.
- **testOneStepFrom0()**
 - 0번 셀에서 한 칸 이동 시 1번 셀로 정상 이동하는지 확인합니다.
- **testSpecialRouteFrom5()**
 - 5번 셀은 분기점입니다. 여기에 도달한 후 1칸 이동 시 일반 경로가 아닌 특수 분기 경로(20번 셀)로 이동하는지 확인합니다.
- **testHistoryStacked()**
 - 경로를 따라 이동할 때 Piece 객체의 history가 정상적으로 누적되는지 검증합니다.
예: 0번 셀에서 3칸 이동 시 → 1, 2, 3번이 순서대로 기록되고, history.peek()는 2번 셀이어야 함.
- **testMultiStepMovement()**
 - 여러 칸(3칸) 이동 시 최종 도착지가 예상대로 3번 셀인지 확인합니다.

- **testIsCorner_returnsTrueForCornerIds()**
 - 0, 5, 10, 15번 셀은 코너 노드로 정의되어 있어 isCorner()가 true를 반환해야 함을 검증합니다.
- **testIsCorner_returnsFalseForNonCornerIds()**
 - 1번, 6번 셀은 코너가 아니므로 isCorner()가 false를 반환해야 함을 확인합니다.
- **testIsCentre_returnsTrueForCentreIds()**
 - 중앙 셀로 정의된 22번, 27번 셀에 대해 isCentre()가 true를 반환하는지 검증합니다.
- **testIsCentre_returnsFalseForNonCentreIds()**
 - 중앙 셀이 아닌 0번, 21번 셀에 대해 isCentre()가 false를 반환하는지 확인합니다.

테스트 전략

- 경로 계산과 노드 속성 판단이 제대로 구현되었는지를 확인하는 구조적 테스트입니다.
 - 분기점 로직(5번 셀 → 20번 셀) 및 이력 저장 로직을 포함하여 말 이동의 일관성과 보드 구조의 신뢰성을 동시에 보장합니다.
 - 특히 getDestinationCell()에 대한 다양한 조건 테스트를 통해 실제 게임 상황의 움직임을 정밀하게 시뮬레이션하고 있습니다.
-

PentagonBoardTest

PentagonBoard는 오각형 기반의 특수 경로 구조를 가진 윷놀이판입니다. 이 테스트는 분기 경로, 중심 노드, 코너 노드 판단 등이 정확히 구현되었는지 확인하기 위한 목적을 갖습니다.

테스트 메서드 설명

- **testInitialCellWhenNull()**
 - 말의 현재 위치가 null인 경우(보드에 아직 올라가지 않은 상태), 시작점인 0번 셀로 이동하는지 확인합니다.
- **testOneStepFrom0()**
 - 0번 셀에서 한 칸 이동하면 1번 셀로 정상 도달해야 합니다.

- **testSpecialRouteFrom5()**
 - 5번 셀은 오각형 보드의 특수 분기점입니다. 여기서 한 칸 이동 시, 25번 셀(중앙 또는 내부 루트)로 분기되어야 함을 검증합니다.
- **testMultiStepFrom0To3()**
 - 0번에서 3칸 이동 시 3번 셀로 도달하는지 확인하여, 일반 경로 탐색이 잘 작동하는지 검증합니다.
- **testHistoryRecordedCorrectly()**
 - 3칸 이동 시 각 셀 방문 기록이 Piece.history에 올바르게 누적되는지 확인합니다. peek() 값이 2인지 확인합니다.
- **testIsCorner_withCornerIds_returnsTrue()**
 - 오각형 보드에서 코너로 정의된 셀 ID(0, 5, 10, 15, 20)에 대해 isCorner()가 true를 반환해야 합니다.
- **testIsCorner_withNonCornerId_returnsFalse()**
 - 코너가 아닌 셀 ID(1, 6, 27)에 대해 isCorner()가 false를 반환하는지 확인합니다.
- **testIsCentre_withCentreId_returnsTrue()**
 - 중앙 셀(27번)에 대해 isCentre()가 true를 반환하는지 확인합니다.
- **testIsCentre_withNonCentreId_returnsFalse()**
 - 0, 5, 30번 셀처럼 중심이 아닌 경우 false 반환을 검증합니다.

테스트 전략

- **분기 경로 탐색, 코너/센터 셀 식별, 이동 이력 기록과 같은 보드의 핵심 역할에 대해 전반적인 테스트를 수행하고 있습니다.**
- PentagonBoard가 일반 경로와 특수 경로 모두를 안정적으로 처리할 수 있는지 검증하는 데 중점을 두고 있으며, 중앙 셀이나 코너 셀 식별을 통해 **게임 전개 흐름 제어**에 있어서의 정확성도 함께 확인됩니다.

HexagonBoardTest

HexagonBoard는 육각형 구조의 윷놀이판으로, 일반 경로 외에도 특수 분기 지점과 중앙 지점을 포함합니다. 이 테스트는 보드의 이동 경로 계산, 분기 판단, 코너 및 중심 노드 판별 기능이 정상적으로 동작하는지를 확인하기 위해 작성되었습니다.

테스트 메서드 설명

- **testInitialCellWhenNull()**

초기 위치가 null인 경우, 첫 셀(0번)로 이동하는 기본 동작을 검증합니다.

- **testOneStepFrom0()**

0번 셀에서 한 칸 이동 시 1번 셀로 정확히 이동하는지 확인합니다.

- **testSpecialRouteFrom5()**

5번 셀은 특수 경로 분기 지점입니다. 한 칸 이동 시 30번 셀로 분기되는 동작을 테스트 합니다.

- **testMultiStepFrom0To3()**

일반 경로에서 다중 스텝 이동이 정확히 수행되는지 검증합니다 ($0 \rightarrow 3$).

- **testHistoryStackedProperly()**

이동 중 방문한 셀들이 Piece 객체의 history 스택에 올바르게 누적되는지 확인합니다. 마지막 위치는 2번 셀이 되어야 합니다.

- **testIsCorner_returnsTrueForCornerIds()**

육각형 보드에서 코너 셀로 정의된 ID들(0, 5, 10, 15, 20, 25)에 대해 isCorner()가 true를 반환하는지 반복 확인합니다.

- **testIsCorner_returnsFalseForNonCornerId()**

일반 셀 ID(3)에 대해 isCorner()가 false를 반환해야 함을 검증합니다.

- **testIsCentre_returnsTrueForCentreId()**

중앙 셀(32번 셀)에 대해 isCentre()가 true를 반환하는지 확인합니다.

- **testIsCentre_returnsFalseForNonCentreId()**

중심이 아닌 셀(예: 15)에 대해 isCentre()가 false를 반환하는지 확인합니다.

테스트 전략

- **경로 이동, 분기 조건, 셀 식별(Corner & Centre)**을 중심으로 보드 내부 로직의 정확성과 완전성을 검증합니다.
- HexagonBoard는 다양한 특수 경로와 셀 구조를 포함하므로, 이 테스트는 보드 기능이 예상된 게임 흐름을 방해하지 않고 작동하는지 확인하는 데 매우 중요합니다.

Controller

GameControllerTest

이 테스트는 GameController 클래스의 핵심 기능, 즉 윷 던지기, 턴 처리, 말 이동 및 게임 종료 조건 확인 등의 로직이 올바르게 작동하는지를 검증합니다. 모의 객체(Mock)를 활용해 UI 인터페이스와의 상호작용도 점검합니다.

테스트 메서드 설명

- **testRenderGame_callsRender**
renderGame() 호출 시 뷰의 render() 메서드가 정상적으로 호출되는지 확인합니다.
- **testCheckAndEndGame_returnsFalseWhenNotFinished**
모든 말이 종료되지 않은 상태에서는 게임이 끝나지 않아야 함을 검증합니다.
- **testHandleTurn_turnEndsIfNoPieceSelected**
사용자가 말을 선택하지 않으면 턴이 종료되고 UI가 다시 렌더링되는지 확인합니다.
- **testHandleTurn_movesPieceAndRendersUI**
말이 실제로 이동하고, UI가 렌더링되는지, 이동한 셀 ID가 예상대로인지 검증합니다.
- **testHandleTurn_noCapture_noExtraTurn**
이동 중 말 잡기가 발생하지 않은 경우, 추가 메시지나 로직이 작동하지 않아야 함을 확인합니다.
- **testHandleYutThrow_normalResult_triggersHandleTurn**
윷 결과가 도~걸일 경우 handleTurn()이 이어서 실행되는지 확인합니다.
- **testHandleYutThrow_addsResultAndSkipsHandleTurn_onYutOrMo**
윷(4) 또는 모(5)가 나온 경우, 추가 턴을 위해 큐에 값만 넣고 handleTurn()은 생략되는 흐름을 검증합니다.
- **testHandleTurn_multipleCaptures_showsCorrectMessage_realistic**
동일 셀에 위치한 적 말 여러 개를 잡은 경우, 정확한 메시지가 출력되는지 확인합니다.
- **testHandleTurn_additionalTurns_granted**

`remainingAdditionalTurns`가 설정된 상태에서, 사용자에게 추가 턴 가능 메시지가 정상 출력되는지 검증합니다.

- **testHandleYutThrow_earlyReturn_whenNotRollingPhase**

현재 턴이 윷 던지기 단계가 아닌 경우, `throwYut()` 자체가 호출되지 않아야 함을 테스트합니다.

- **testCheckAndEndGame_restartFlow**

모든 말이 완료된 후 사용자가 재시작을 선택하면 게임이 종료되고 뷰가 `dispose()` 되는지 확인합니다.

테스트 전략

- 각 테스트는 **게임 진행 흐름의 다양한 분기와 상황**을 커버합니다.
- Mockito를 통해 `GameViewInterface` 및 하위 뷰 인터페이스들을 **Mock 객체**로 처리함으로써, **UI와의 의존성 제거** 및 **단위 로직 집중 테스트**를 구현했습니다.
- `stepQueue`, `isRollingPhase` 등의 private 필드는 리플렉션을 사용해 내부 상태를 제어하며 테스트 신뢰도를 높였습니다.

PieceMoveControllerTest

이 테스트 클래스는 `Yutnori` 게임의 핵심 로직 중 하나인 말 이동, 그룹핑(업기), 캡처(잡기), 그리고 도착 처리 등을 담당하는 `PieceMoveController`의 동작을 포괄적으로 검증합니다.

테스트 메서드 설명

- **testFirstMove_startsCorrectly**

게임에서 말이 처음 보드에 진입할 때 정상적으로 시작되고 히스토리도 기록되는지 확인합니다.

- **testMoveToStartTwice_triggersImmediateFinish**

말이 출발점을 두 번 통과한 경우, 즉시 완료 처리되는지 검증합니다.

- **testMoveWithBackDo_whenNotStarted_doesNothing**

아직 보드에 올라가지 않은 말에 빼도가 들어오면 무시되는지 확인합니다.

- **testMoveWithBackDo_whenOnBoard_goesBack**

이미 보드 위에 있는 말이 빽도를 받았을 때, 이전 위치로 되돌아가는지 테스트합니다.

- **testMovePiece_doesNothingWhenAlreadyFinished**

완료된 말은 어떤 이동도 수행하지 않아야 함을 검증합니다.

- **testMovePiece_doesNothingWhenNextIsNull**

유효하지 않은 이동 거리(-100 등)로 인해 목적지가 계산되지 않으면 아무 이동도 하지 않는지 확인합니다.

- **testMovePiece_withGroupedPiece_movesLeader**

업한 말이 이동할 경우, 그룹 리더가 이동하는지 확인합니다.

- **testMovePiece_withGroupLeader_movesLeader**

그룹 리더가 실제 이동하고, 업한 말의 리더도 그대로 유지되는지 테스트합니다.

- **testHandleGrouping_sameGroupLeader_skipsMerge**

이미 같은 그룹 리더를 공유하고 있다면 추가 병합이 발생하지 않는지 확인합니다.

- **testBackDo_finishesWhenHistoryEmptyAndPassedStartOnce**

말이 출발점을 한 번 통과한 후 history가 비어 있을 때 빽도를 받아면 말이 완료되는지 확인합니다.

- **testBackDo_whenHistoryEmptyAndNotPassedStart_doesNothing**

아직 한 바퀴를 돌지 않았는데 history가 비어 있는 경우, 빽도를 받아도 완료되지 않는지 확인합니다.

- **testCapture_enemyPiecesReset**

같은 셀에 적군 말이 있을 때, 캡처가 발생하고 해당 말이 보드에서 사라지는지 확인합니다.

- **testCapture_whenNoEnemyPieces_returnsZero**

같은 칸에 있는 말이 모두 같은 플레이어라면 캡처가 발생하지 않아야 함을 검증합니다.

- **testFinishCondition_pieceGetsFinished**

출발점을 여러 번 지난 말이 도착하면 자동으로 완료되는지 확인합니다.

- **testCheckFinishCondition_triggersFinishWhenAtStartWithHistory**

출발점에 도달했을 때 히스토리가 충분하면 완료 처리되는지 검증합니다.

- **testFinishTrigger_whenOnSpecialCellAfterTwoZeros**

특수 셀에 도착했더라도 출발점을 두 번 방문했다면 완료되는지 확인합니다.

- **testGrouping_whenSamePlayerExistsOnSameCell**

같은 플레이어의 다른 말이 같은 셀에 있으면 그룹핑이 발생하는지 확인합니다.

- **testHandleGrouping_centralCellPairMergedInSquareBoard**

SquareBoard의 특성상 22번 셀과 27번 셀이 동일 셀로 간주되며, 업기가 제대로 동작하는지 검증합니다.

- **testGrouping_skipsWhenAlreadyGrouped**

이미 그룹에 포함된 말을 다시 그룹핑할 경우 아무런 변화가 없어야 합니다.

- **testGrouping_whenNoCandidates_doesNothing**

셀에 그룹핑할 말이 없다면 아무 일도 일어나지 않아야 합니다.

- **testBackDo_setsPassedStartOnce_whenLandingOnStart**

빡도 이동 결과 출발점에 도달하면 setPassedStartOnce()가 호출되는지 확인합니다.

테스트 전략

- 말의 이동 조건에 따라 다양한 경로 분기를 테스트합니다.
- groupLeader, history, position, isFinished 등 상태 필드를 중심으로 테스트 케이스를 분리했습니다.
- 도착 조건, 그룹 병합, 캡처 등 핵심 게임 로직의 **엣지 케이스를 집중적으로 커버**합니다.
- SquareBoard를 기반으로 테스트하며, 다른 보드 유형은 별도의 보드 테스트에서 검증합니다.

YutControllerTest

이 테스트 클래스는 YutController가 사용자의 웃 던지기 액션을 받아 적절한 로직(Yut)을 수행하고 결과를 뷰(YutResultView)에 반영하는지 검증합니다.
특히 무작위 던지기와 선택 던지기, 예외 처리 흐름까지 포함합니다.

테스트 메서드 설명

- **testPerformThrowRandom_callsThrowRandomYutAndUpdatesView**

isRandom이 true인 경우, 내부적으로 Yut.throwRandomYut()가 호출되고 view.setYutResult(yut)로 결과가 전달되는지 확인합니다.

- `testPerformThrowSelect_callsThrowSelectYutWithCorrectValue`

사용자가 선택한 윷 값(view.getSelectedYutValue() 반환값)을 기반으로 Yut.throwSelectYut(int)가 정확히 한 번 호출되고, 이후 결과가 뷰에 반영되는지 검증합니다.

- `testPerformThrowSelect_invalidValue_doesNotThrow`

유효하지 않은 웃 값을 선택했을 때 예외가 발생하더라도 테스트가 중단되지 않으며, `setYutResult()` 호출 없이 조용히 종료되는 흐름을 테스트합니다.

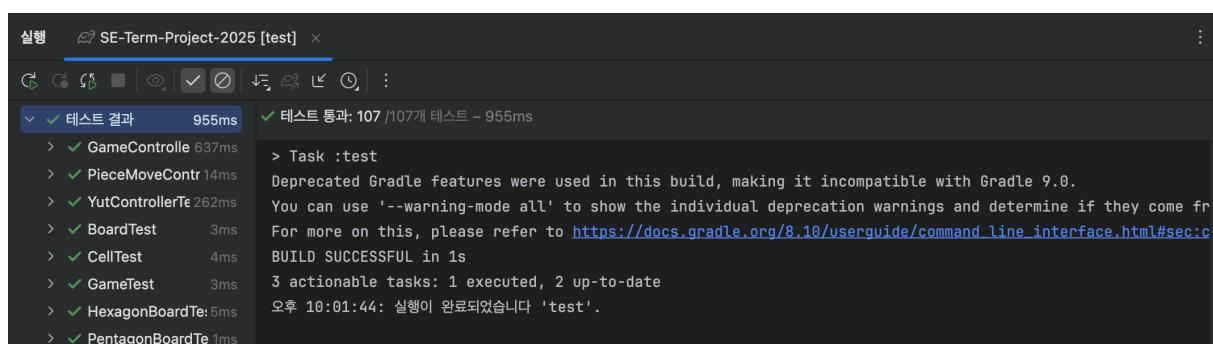
실제 UI에서의 견고한 예외 처리 전략을 반영합니다.

테스트 전략

- 테스트는 **Mock 객체** 기반으로 구성되어 실제 로직 수행 여부와 UI 업데이트 여부를 정밀하게 검증합니다.
 - 유효하지 않은 값 입력 시 예외 발생을 시뮬레이션하여 **예외 안전성**도 확인합니다.
 - 단위 테스트로서 YutController의 **역할 수행 여부**만을 독립적으로 검증하며, Yut이나 View의 내부 로직에는 의존하지 않습니다.

테스트 결과 및 커버리지

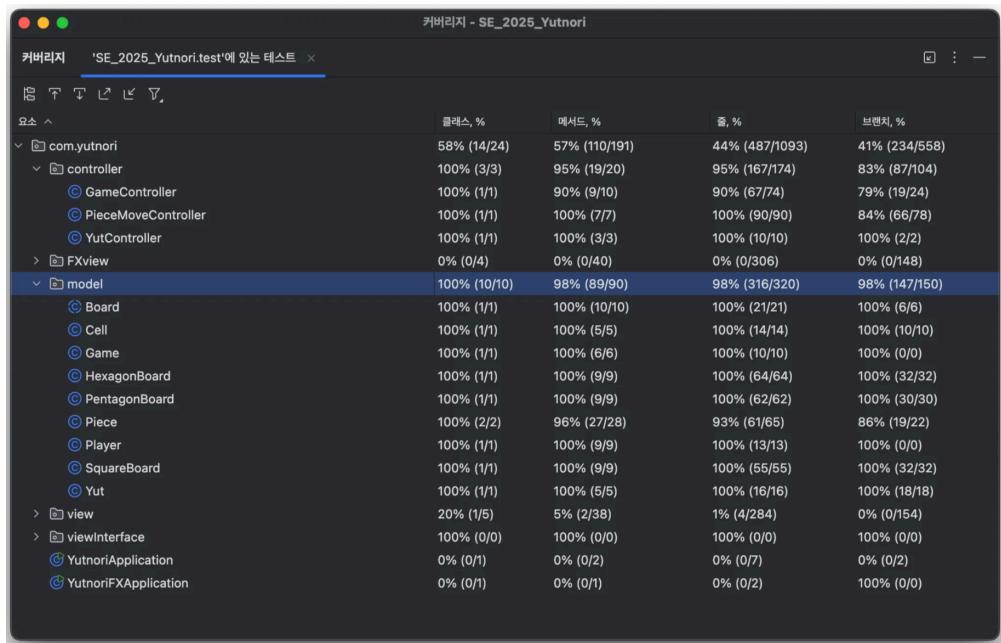
테스트 실행 결과



```
> Task :compileJava UP-TO-DATE  
> Task :processResources NO-SOURCE  
> Task :classes UP-TO-DATE  
> Task :compileTestJava UP-TO-DATE
```

```
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
WARNING: A Java agent has been loaded dynamically (/Users/jaemin/.gradle/caches/modules-2/files-2.1/net.bytebuddy/byte-buddy-agent/1.14.12/be4984cb6fd1ef1d11f218a648889dfda44b8a15/byte-buddy-agent-1.14.12.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
선택 오류: 유효하지 않은 육 결과입니다.
> Task :test
Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.
For more on this, please refer to https://docs.gradle.org/8.10/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the Gradle documentation.
BUILD SUCCESSFUL in 1s
3 actionable tasks: 1 executed, 2 up-to-date
오후 10:01:44: 실행이 완료되었습니다 'test'.
```

커버리지 측정 결과



Total

- 전체 클래스 기준으로 **58%**, 메서드 기준 **57%**, 줄(line) 커버리지는 **44%**, 분기(branch) 커버리지는 **41%**입니다.
- 그러나 이는 view 및 application 패키지를 포함한 값이며, UI 관련 클래스는 자동화 테스트에서 제외되어 있기 때문에 실제 핵심 로직을 담은 controller, model 패키지 기준으로는 커버리지가 매우 높습니다.

model	100% (10/10)	98% (89/90)	98% (316/320)	98% (147/150)
Board	100% (1/1)	100% (10/10)	100% (21/21)	100% (6/6)
Cell	100% (1/1)	100% (5/5)	100% (14/14)	100% (10/10)
Game	100% (1/1)	100% (6/6)	100% (10/10)	100% (0/0)
HexagonBoard	100% (1/1)	100% (9/9)	100% (64/64)	100% (32/32)
PentagonBoard	100% (1/1)	100% (9/9)	100% (62/62)	100% (30/30)
Piece	100% (2/2)	96% (27/28)	93% (61/65)	86% (19/22)
Player	100% (1/1)	100% (9/9)	100% (13/13)	100% (0/0)
SquareBoard	100% (1/1)	100% (9/9)	100% (55/55)	100% (32/32)
Yut	100% (1/1)	100% (5/5)	100% (16/16)	100% (18/18)

Model

- 총 10개의 모델 클래스 모두 테스트 대상에 포함되어 **클래스 커버리지 100%, 메서드 98%, 라인 98%, 분기 98%**라는 매우 높은 수준을 달성했습니다.
- 특히 말 이동 기록(Piece), 플레이어 점수 누적(Player), 보드 타입별 경로 분기 (SquareBoard, HexagonBoard, PentagonBoard) 등 다양한 조건을 테스트해 실제 게임 로직의 안정성을 확보했습니다.

controller	100% (3/3)	95% (19/20)	95% (167/174)	83% (87/104)
GameController	100% (1/1)	90% (9/10)	90% (67/74)	79% (19/24)
PieceMoveController	100% (1/1)	100% (7/7)	100% (90/90)	84% (66/78)
YutController	100% (1/1)	100% (3/3)	100% (10/10)	100% (2/2)

Controller

- 모든 컨트롤러 클래스가 100% 클래스 커버리지를 기록했으며, 메서드 및 줄 커버리지도 90% 이상을 유지하고 있습니다.
- GameController는 윷 던지기, 말 선택, 게임 종료, 추가 턴 부여 등 실제 사용자 흐름을 중심으로 테스트되었습니다.
- PieceMoveController는 말 이동, 그룹핑, 잡기, 종료 처리 등 도메인 로직을 담당하는 핵심 메서드 커버리지가 골고루 높게 측정되었습니다.
- YutController는 JavaFX UI와 윷 데이터를 연결하는 인터페이스로, Mock 객체를 활용해 주요 흐름이 검증되었습니다.

커버리지 한계 및 예외 처리

다음 항목은 테스트 커버리지 통계에 포함되지 않거나, 일부만 측정되었습니다. 각 항목에 대한 설명과 테스트 제외 이유는 아래와 같습니다.

1. View 패키지(JavaFX 기반 UI)

- 제외 이유:** view 및 viewInterface 패키지는 JavaFX를 기반으로 한 사용자 인터페이스(UI) 요소를 포함합니다.
- 설명:** JavaFX 컴포넌트는 이벤트 루프 및 시각적 인터랙션에 의존하기 때문에, 일반적인 단위 테스트(JUnit)로는 검증이 어렵습니다.
- 대안:** 주요 UI 동작은 프로젝트 실행 후 수동으로 점검하였고, 게임 흐름에 맞춘 인터랙션이 정상 작동함을 확인했습니다. 별도 UI 테스트 프레임워크(TestFX 등)를 도입할 경우 추후 자동화가 가능하지만, 본 프로젝트에서는 범위를 벗어나므로 제외하였습니다.

2. 예외 흐름 테스트 제한

- 설명:** 사용자의 잘못된 입력이나 엣지 케이스(예: 유효하지 않은 윷 값, 입력 취소 등)는 주 흐름을 벗어나기 때문에 제한적으로만 테스트되었습니다.

- **대응:** 가능한 경우 `IllegalArgumentException`, `IllegalStateException` 발생 여부는 테스트 코드로 검증했고, 나머지는 예외 없이 무시되거나 UI로 대응되는 설계입니다.
- **예시:** `YutControllerTest.testPerformThrowSelect_invalidValue_doesNotThrow` 등

3. MockBoard 등 테스트 유틸리티

- **설명:** `MockBoard` 클래스는 테스트를 위한 간이 `Board` 구현으로, 실제 게임 로직에서 사용되지 않습니다.
- **테스트 목적:** 복잡한 분기 없이 단일 경로로만 움직이는 보드 구조를 통해, `Board` 추상 클래스의 메서드 동작을 검증하기 위함입니다.
- **커버리지 제외 이유:** 실제 게임에는 영향을 주지 않으므로 별도 커버리지 항목으로 평가하지 않음.

결론

본 테스트는 `Yutnori` 게임 프로젝트의 핵심 로직이 의도대로 동작하는지를 체계적으로 검증하고, 주요 기능에 대한 안정성과 신뢰성을 확보하는 것을 목표로 수행되었습니다.

테스트 성과

- 모델 및 컨트롤러 계층의 클래스, 메서드, 줄, 분기 커버리지 모두 **90% 이상**을 기록하였습니다.
- **Game, Piece, Board, GameController** 등 게임 로직의 중심이 되는 주요 클래스에 대한 검증을 완료하였습니다.
- **단위 테스트 범위** 내에서 말의 이동, 그룹핑, 잡기, 도착 처리 등 다양한 상황을 커버하며, 복잡한 조건 분기와 엣지 케이스까지 포함하였습니다.
- **MockBoard**를 활용하여 테스트의 예측 가능성과 반복성을 확보하고, UI 로직으로부터 독립된 순수 로직 검증을 수행하였습니다.

커버리지 한계와 보완점

- JavaFX 기반의 UI(view 패키지)는 자동화 테스트가 어려워 수동 검증으로 대체하였으며, 커버리지 통계에는 포함되지 않았습니다.
- **사용자 입력 오류나 잘못된 흐름 등 예외 처리 로직은 일부 주요 흐름에 한정하여 테스트되었고, 보완의 여지가 있습니다.**
- **통합 테스트 또는 시나리오 기반 테스트가 추가로 구성되면 실제 사용자 흐름 전반에 대한 검증이 보다 완전해질 수 있습니다.**

종합 평가

이번 테스트는 단순히 코드의 기능을 확인하는 수준을 넘어, 윷놀이 게임의 전체 흐름과 로직이 예상대로 작동하는지를 깊이 있게 검증하는 과정이었습니다. 특히 말의 이동, 그룹핑, 잡기, 도착 처리 등 게임 진행의 핵심 기능에 대해 다양한 시나리오를 직접 구성하고, 엣지 케이스까지 고려한 테스트를 수행함으로써 로직의 견고함을 점검할 수 있었습니다.

테스트 도중 예상하지 못했던 예외 상황이나 조건 분기에서도 로직이 잘 견디는지 확인하면서, 게임 설계가 상당히 구조화되어 있다는 점을 체감할 수 있었고, 일부 로직은 테스트를 통해 더욱 명확하게 정리되는 계기가 되었습니다. 또한 테스트 커버리지를 측정하면서 실제로 어떤 로직이 사용자 흐름의 핵심을 이루고 있는지를 구체적으로 파악할 수 있었고, 테스트 코드가 단순 확인을 넘어 **프로젝트 이해의 도구**가 된다는 점도 직접 느낄 수 있었습니다.

전반적으로 model 및 controller 계층의 테스트 커버리지는 클래스/메서드/라인/분기 기준으로 대부분 90% 이상을 달성하며, **핵심 기능에 대해 충분한 신뢰 수준을 확보한 것으로** 판단됩니다. 특히 PieceMoveController와 GameController는 실제 게임 상황을 거의 완전하게 시뮬레이션하며, 다양한 조건 속에서도 일관된 동작을 보여주었습니다.

결과적으로, 이번 테스트는 단위 기능의 검증을 넘어서 **프로젝트 전체의 안정성과 완성도를 높이는 데 기여한 중요한 과정이었으며**, 테스트가 코드의 마지막 단계가 아닌 설계와 구조에 대한 피드백 도구로도 매우 유효하다는 점을 실감할 수 있었습니다.

5. GitHub 프로젝트

이름	학번	GitHub ID	역할
정인혁	20215431	wjddlsgur0928	설계 및 문서화
권재민	20214521	jack0928	설계 및 테스트
김진하	20210623	jinha0907	개발
이민규	20215143	mingyulee327	개발

최정우	20216620	justine1118	개발
-----	----------	-------------	----

LINK : <https://github.com/jack0928/SE-Term-Project-2025>

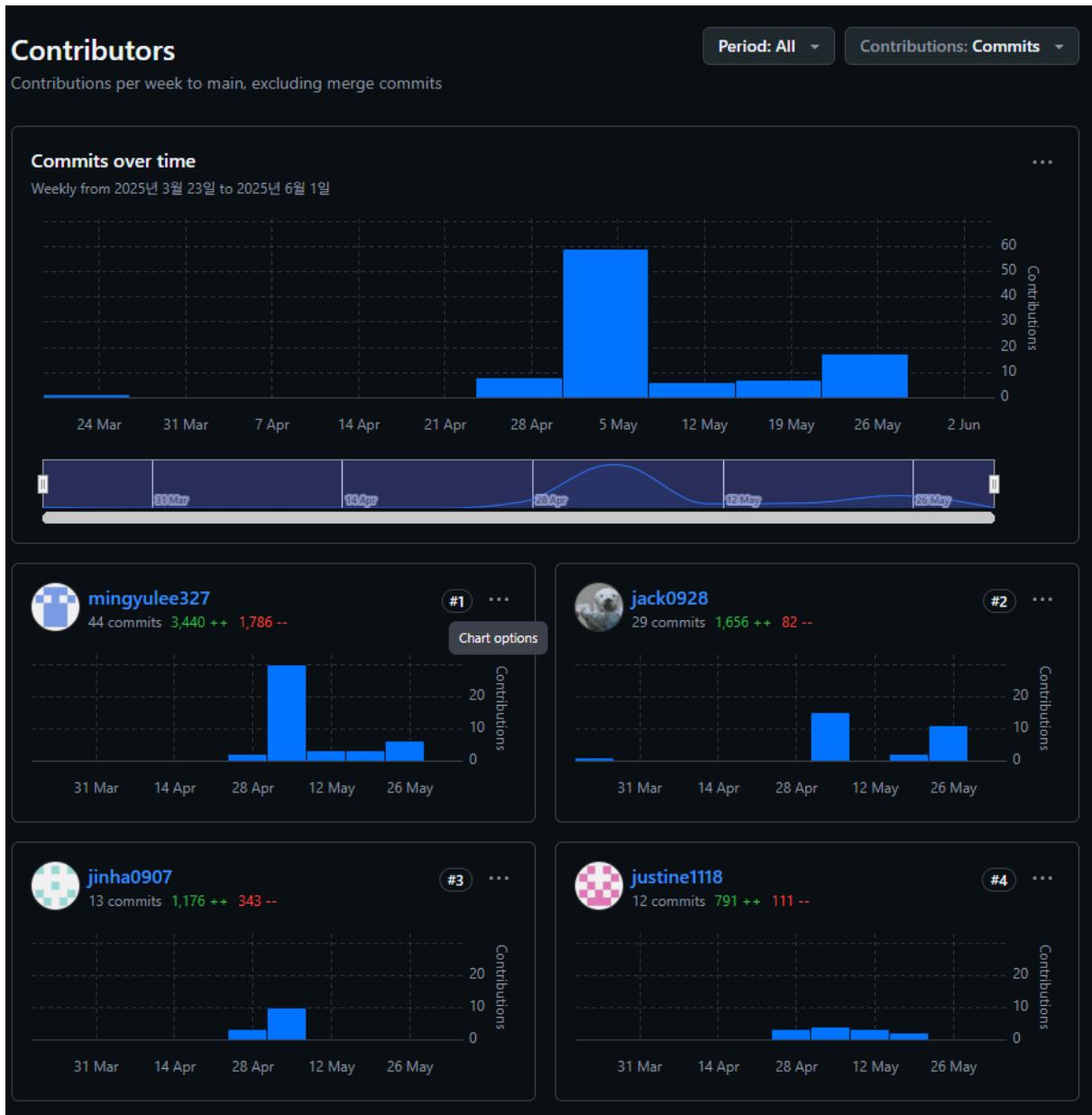
Notion LINK : <https://jaemink.notion.site/2025-SE-Term-Project-1ceffa740e3d80a198e0fef6739fad4>

GitHub만을 이용하여 프로젝트 진행 상황 및 회의 기록 정리는 가독성이 떨어진다고 판단하고 Notion을 추가로 이용하였다. 해당 Notion 링크는 GitHub 사이트의 Wiki에 다음과 같이 추가하였다.

The screenshot shows the GitHub Wiki interface for the repository 'SE-Term-Project-2025'. The main content area displays a simple 'Welcome to the SE-Term-Project-2025 wiki!' message. Below it is a dashed box containing a '+ Add a custom footer' button. To the right, there is a sidebar with a 'Pages' section showing 2 pages: 'Home' and 'Team Notion Page'. At the bottom of the sidebar, there is another dashed box with a '+ Add a custom sidebar' button. The URL of the page is visible at the bottom: <https://github.com/jack0928/SE-Term-Project-2025/wiki/Home>.

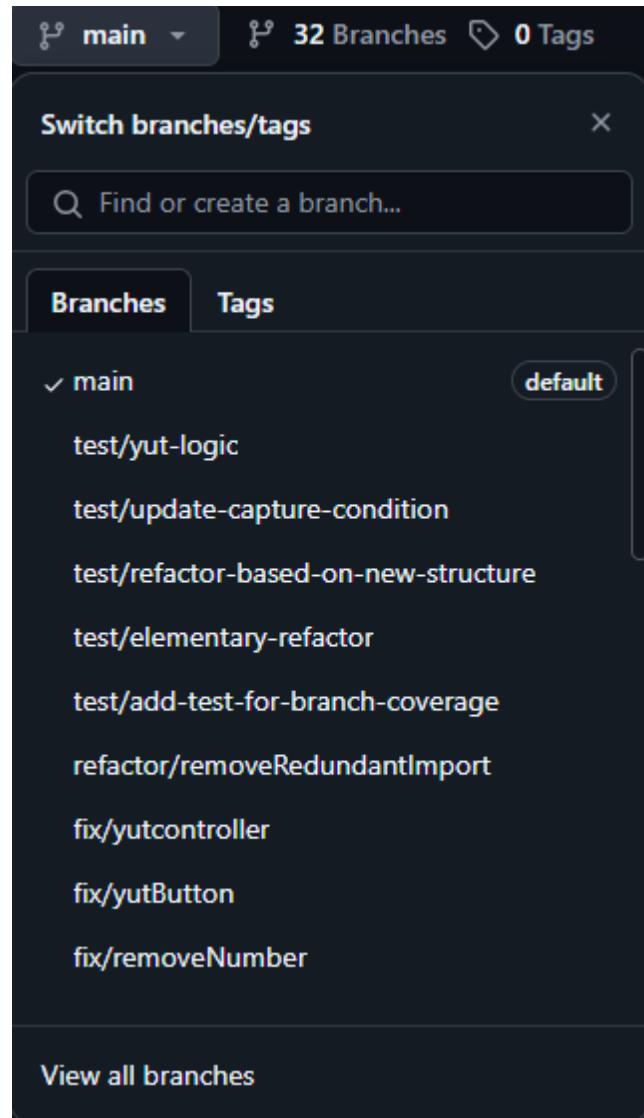
Progress history & Contribution

개발 및 테스트 인원의 commit 기록은 다음과 같다.



Branch

Structure



- **main**
 - 최종 제출용 브랜치
 - 직접 커밋하지 않는다.
- **develop**
 - 전체 개발 브랜치
 - 기능 개발이 끝나면 이곳으로 병합한다.
- **feature/기능명**
 - 개인 또는 특정 기능 단위의 작업 공간
 - 예시: feature/yut-logic, feature/ui-swing, feature/game-controller
- **fix/기능명**

- 버그 수정 브랜치
- 예시 : fix/yutcontroller
- **test/기능명**
 - 테스트 코드 개발용 브랜치
 - 예시 : test/update-capture-condition
- **refactor/기능명**
 - 기능 변화 없이 코드 구조 개선 담당 브랜치
 - 예시 : refactor/removeRedundantImport

Flow

1. 기능 시작 시 develop에서 브랜치 분기
2. 각자 feature/* 브랜치에서 작업
3. 작업 완료 후 develop에 Pull Request
4. 1명 이상 리뷰 → Merge

Commit convention

Commit rule

- **Action: Description**
 - Action
 - feat : 새로운 기능 추가
 - fix : 버그 수정
 - docs : 문서 작성/수정
 - refactor : 코드 구조 개선 (기능 변화 X)
 - test : 테스트 코드 추가
 - Description
 - 구체적인 변경 사항을 쉽게 이해할 수 있도록 요약

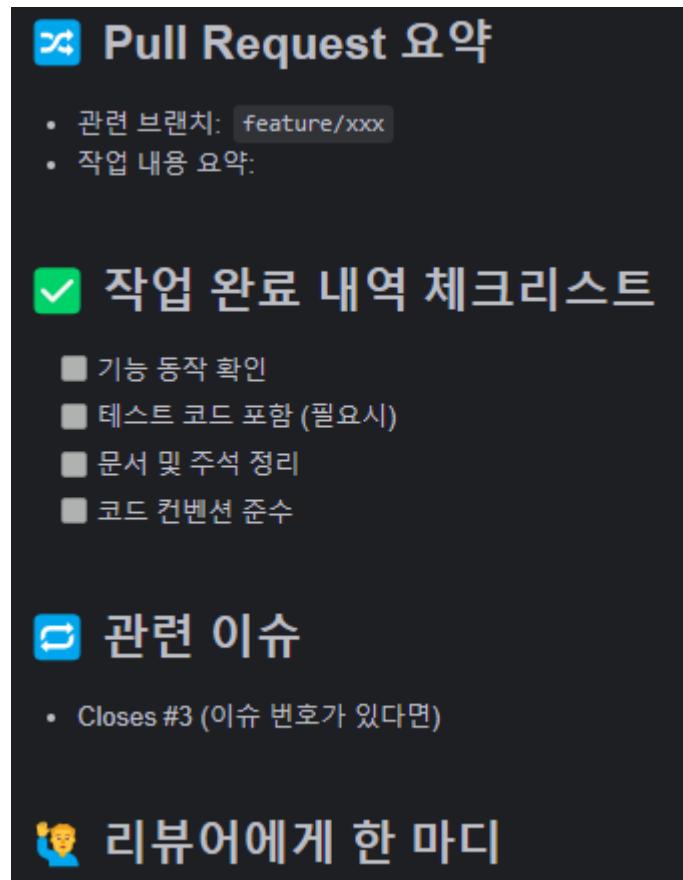
Commit example

Commits on May 9, 2025

feat: victory, restart, exit feature has been added. UI has been improved as well and all the UI now works correctly. mingyulee327 committed 3 weeks ago	8fb15d3
fix: player's turn alert will now follow the player's colour. Also, there was a bug where the turn doesn't change when a player catches another player's piece. It has been fixed. mingyulee327 committed 3 weeks ago	a299b92
refactor: deleted Test classes mingyulee327 committed 3 weeks ago	0102797
refactor: YutnoriApplication에서의 restartGame() function을 refactor하여 YutnoriApplication.java에서는 main() function만 있도록 refactor. Also, annotations were added as needed. mingyulee327 committed 3 weeks ago	3bd0b31

Pull Request

다음은 PR의 가독성을 높이기 위해 자체적으로 정한 PR Template이다.



- **Pull Request 요약**
 - 작업을 진행한 branch를 명시하여 Git 흐름을 명확히 한다.
 - PR에서 무엇을 변경했는지 간단하게 요약하여 리뷰어가 빠르게 작업 범위를 파악 할 수 있도록 도와준다.
- **작업 완료 내역 체크리스트**
 - 기능 동작 확인
 - 내가 만든 기능이 실제로 잘 작동하는지 직접 확인했는지를 체크한다.
 - 테스트 코드 포함
 - 단위 테스트, 통합 테스트 등을 작성했는지 체크한다.
 - 문서 및 주석 처리
 - 코드 내 주석, 또는 프로젝트 문서에 변경 사항을 반영했는지 확인한다.
 - 코드 컨벤션 준수
 - 사전에 정한 코드 스타일을 따랐는지 확인한다.
- **관련 이슈**
 - 연결되거나 참고할 만한 이슈를 명시한다.
- **리뷰어에게 한 마디**
 - 리뷰어가 리뷰하기 전에 알아야 할 사항이나, 유의사항 등을 자유롭게 작성한다.

다음은 PR Template에 의거하여 진행한 PR 중 하나이다.

[Fix] 백도 상황 정리 및 main() refactoring #38

Merged jinha0907 merged 3 commits into develop from feature/Integrated-UI 3 weeks ago

Conversation 1 Commits 3 Checks 0 Files changed 5

mingyulee327 commented 3 weeks ago • edited

Pull Request 요약

- 관련 브랜치: feature/Integrated-UI
- 작업 내용 요약:

- 백도 상황에서, 말이 보드에 없을 경우 백도 이동이 불가하나 시스템에서 보드에 없는 말까지 백도 선택권을 주는 issue 해결 (무조건 보드에 있어야만 말 선택 가능, 없을 경우 그냥 턴만 넘어감)
- main() 코드 refactoring. 대부분의 경우 GameController 내부로 보내. MVC 구조에 더욱 충실히 만들었음.

작업 완료 내역 체크리스트

- 기능 동작 확인
- 테스트 코드 포함 (필요시)
- 문서 및 주석 정리
- 코드 컨벤션 준수

관련 이슈

- Closes [Fix] 백도 상황에서의 말 choice 관련 fix #35

리뷰어에게 한 마디

백도 상황 수정 및 main() refactoring complete.

mingyulee327 added 3 commits 3 weeks ago

- fix: 백도상할 시 piece는 무조건 board에 있어야만 하게 변경 86c4e95
- refactor: debugging messages and useless methods were removed. f8e9307
- fix: merged with develop and resolved conflict, added logic for 백도, r... 7babc62

mingyulee327 requested a review from jack0928 3 weeks ago

mingyulee327 self-assigned this 3 weeks ago

mingyulee327 added fix, 민규 labels 3 weeks ago

mingyulee327 requested a review from jinha0907 3 weeks ago

jinha0907 approved these changes 3 weeks ago

jinha0907 left a comment

확인했습니다.

jinha0907 merged commit e2f6d22 into develop 3 weeks ago

Pull request successfully merged and closed

You're all set — the branch has been merged.

다음은 개발 과정에서의 PR 기록이다.

Author	Label	Projects	Milestones	Reviews	Assignee	Sort
mingyulee327	[Refactor] remove redundant import	민규	#70 by mingyulee327 was merged yesterday • Approved	4 tasks	1	
jack0928	[Test] Coverage 향상을 위한 테스트 코드 추가 및 수정		#69 by jack0928 was merged yesterday • Approved	4 tasks done	1	
jinha0907	fix: removed numberings in the nodes in javafx version	fix 민규 전하	#67 by jinha0907 was merged 3 days ago • Approved	4 tasks	1	
jack0928	Test/refactor based on new structure	test 재민	#65 by jack0928 was merged last week • Approved	4 tasks done	4	
mingyulee327	[Feat] GameView 및 View 관련 MVC 구조 개선	feat fix 민규	#63 by mingyulee327 was merged last week • Approved	3 of 4 tasks	2	
justine1118	feat: FXPlayerStatusView 생성, TestFXPlayerStatusView로 테스트 가능	feat 정우	#60 by justine1118 was merged last week • Approved	3 of 4 tasks	4	
mingyulee327	[Feat] FXBoardView (BoardView but in JavaFX instead of Swing)	feat 민규	#58 by mingyulee327 was merged 2 weeks ago • Approved	4 tasks done	1	
justine1118	Feat: JavaFX 버전 YutResultView 제작	feat 정우	#56 by justine1118 was merged 2 weeks ago • Approved	4 tasks	1	
jinha0907	fix: grouping's grouping error 2	fix 전하	#54 by jinha0907 was merged 3 weeks ago • Approved	4 tasks		
jinha0907	Test/update capture condition	test 전하	#53 by jinha0907 was merged 3 weeks ago • Approved	4 tasks done	1	
mingyulee327	[Fix] finish and grouping/capturing order modified	fix 민규	#52 by mingyulee327 was merged 3 weeks ago • Approved	4 tasks done	3	
mingyulee327	[Fix] SquareBoard Exception Handling	fix 민규	#51 by mingyulee327 was merged 3 weeks ago • Approved	4 tasks	1	
mingyulee327	[Fix] UI and 백도 fixation	fix 민규	#50 by mingyulee327 was merged 3 weeks ago • Approved	4 tasks	1	
jinha0907	fix: grouping in center error	fix 전하	#49 by jinha0907 was merged 3 weeks ago • Approved	4 tasks		
jack0928	[Fix] Test 코드 1차 리팩토링		#48 by jack0928 was merged 3 weeks ago	4 tasks		
mingyulee327	[Fix] 백도 문제 수정 및 노드 ID 숫자 제거	fix 민규	#47 by mingyulee327 was merged 3 weeks ago • Approved	4 tasks		
mingyulee327	[Fix] fixed finishing logic	fix 민규	#45 by mingyulee327 was merged 3 weeks ago • Approved	4 tasks		
mingyulee327	[Fix] 백도 관련 issue resolve (백도 적용 방지)	fix 민규	#44 by mingyulee327 was merged 3 weeks ago • Approved	4 tasks	1	
jack0928	[Test] 1차 테스트 코드 작성		#42 by jack0928 was merged 3 weeks ago	3 of 4 tasks		
mingyulee327	[Feat] 업은 말에 대한 정보 표시	feat 민규	#41 by mingyulee327 was merged 3 weeks ago • Approved	3 of 4 tasks	2	
mingyulee327	[Fix]: Grouping's Grouping Issue fixed	fix 민규	#40 by mingyulee327 was merged 3 weeks ago • Approved	3 of 4 tasks	1	
mingyulee327	[Fix] 백도 상황 정리 및 main() refactoring	fix 민규	#38 by mingyulee327 was merged 3 weeks ago • Approved	3 of 4 tasks	1	
jinha0907	Feature/code refactoring	feat 전하	#37 by jinha0907 was merged 3 weeks ago • Approved	3 of 4 tasks		
mingyulee327	[Fix] 폰트 Printing 오류 수정	fix 민규	#34 by mingyulee327 was merged 3 weeks ago • Approved	4 tasks		
mingyulee327	[Feat] 승리, 제시작, 게임 종료 기능 구현	feat 민규	#32 by mingyulee327 was merged 3 weeks ago • Approved	3 of 4 tasks		

		Author	Label	Projects	Milestones	Reviews	Assignee	Sort
0 Open	✓ 41 Closed							
↳ Feature/add grouping (feat) 진화	#29 by jinha0907 was merged 3 weeks ago • Approved	0 4 tasks done			⌚ 1			
↳ Feat: added total UI and options for users (feat) 민규	#28 by mingyulee327 was merged 3 weeks ago • Approved	⌚ 8 tasks				⌚ 1		
↳ Fix: YutController 수정 (fix) 정우	#26 by justine1118 was merged 3 weeks ago	⌚ 4 tasks				⌚ 1		
↳ fix: 선택 옻 던지기와 랜덤 옻 던지기 버튼 추가, 선택 옻 던지기의 값 선택을 위한 드롭다운 추가, YutController... (fix) 정우	#24 by justine1118 was merged 3 weeks ago	⌚ 4 tasks				⌚ 1		
↳ Fix: fixed movement logic add (fix) 진화	#23 by jinha0907 was merged 3 weeks ago • Approved	⌚ 4 tasks				⌚ 1		
↳ Feature/integrated UI (fix) 민규	#21 by mingyulee327 was merged 3 weeks ago • Approved	0 4 tasks done				⌚ 1		
↳ Fix: movement logic repair - 1 (fix) 진화	#20 by jinha0907 was merged 3 weeks ago • Approved	⌚ 3 of 4 tasks				⌚ 1		
↳ fix: throwRandomYut을 확률에 따라 던지게 수정 (fix) 정우	#19 by justine1118 was merged 3 weeks ago • Approved	⌚ 4 tasks				⌚ 1		
↳ fix: fixed MVC Structure of board class and TestPieceView. Also, usel... (fix) 민규	#16 by mingyulee327 was merged 3 weeks ago • Approved	⌚ 3 of 4 tasks				⌚ 1		
↳ Feature/add entire logic (feat) 진화	#14 by jinha0907 was merged 3 weeks ago • Approved	⌚ 3 of 4 tasks				⌚ 2		
↳ feat: PlayerStatusView 구현 (feat) 정우	#13 by justine1118 was merged 3 weeks ago • Approved	⌚ 4 tasks				⌚ 1		
↳ feat: added Piece UI. It now appears on the board and moves around by... (feat) 민규	#9 by mingyulee327 was merged last month	⌚ 4 tasks done				⌚ 1		
↳ Feat: 옻 기능, UI 및 controller 구현 (feat) 정우	#8 by justine1118 was merged last month • Approved	⌚ 4 tasks done				⌚ 1		
↳ feat: Added basic structure (Square, Pentagon, Hexagon) for Yutnori b... (feat) 민규	#4 by mingyulee327 was merged 27 days ago	⌚ 4 tasks done				⌚ 3		
↳ feat: Add template for Issue and PR (docs) 진화	#2 by jinha0907 was merged 28 days ago • Approved					⌚ 1		
↳ feat: 프로젝트 초기구조 및 기본클래스 추가 (feat) 진화	#1 by jinha0907 was merged 28 days ago • Approved					⌚ 1		