15 조 Project Document

김영기, 김성윤, 오준석, 일리야

(1) Team Members & Roles

20171768 김영기 : Model & Controller & View 및 Document 작성

20173046 김성윤 : Model & Controller & View 및 Document 작성

20174612 오준석 : Prototype 및 디자인

20172357 일리야 : Quality Assurance

(2) Vision

- Introduction

휴먼 ICT 소프트웨어 공학 팀 프로젝트로 제작된 윷놀이 게임.

윷놀이 규칙이 지방마다 조금씩 다르기 때문에 가장 표준적인 규칙으로 제작하였다.

윷놀이의 플레이어 수를 2 ~ 4 명, 각 플레이어 당 말의 개수를 2 ~ 5 개 선택할 수 있다.

- Main Features

- 윷놀이 규칙

윷놀이 규칙은 최대한 표준을 지키기 위해 온라인 위키에 작성된 룰을 참고하였다.

https://namu.wiki/w/%EC%9C%B7%EB%86%80%EC%9D%B4

- 플레이어 수 선택

윷놀이를 하는 플레이어의 수를 2 ~ 4 명으로 선택할 수 있다.

- 말의 개수 선택

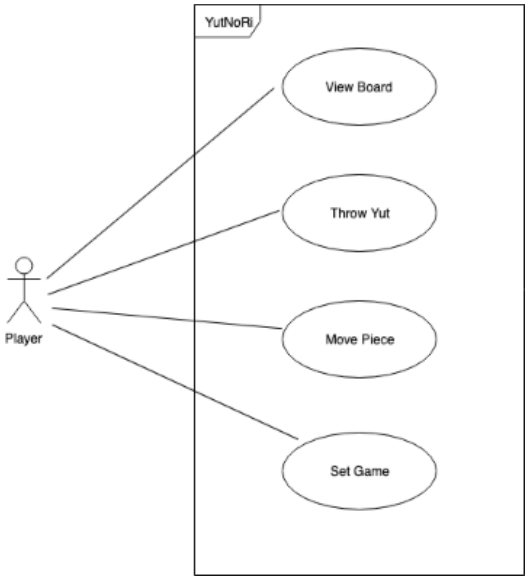
각 플레이어 당 말의 개수를 2 ~ 5 개로 선택할 수 있다.

- 윷 던지기

일반적인 윷놀이처럼 윷을 네 개 던지면 백도, 도 ~ 모가 각 확률에 따라 나오도록 구현하였으며 테스트를 위해 각 결과를 강제로 나오게 할 수 있는 버튼을 만들었다.

(3) Use Case Model

- Use Case Diagram



- Use Case Diagram -

- Set Game (Actor : Player)

플레이어가 게임을 시작하기 전에 게임의 플레이어 수와 각 플레이어 당 말의 개수를 설정한다.

- Throw Yut (Actor : Player)

플레이어가 윷놀이를 진행하기 위해 윷을 던진다. 결과는 확률에 맞추어 백도, 도, 개, 걸, 윷, 모가 나와야 한다.

- Move Piece (Actor : Player)

Throw Yut 의 결과를 토대로 플레이어가 자신의 말을 움직인다.

- View Board (Actor : Player)

플레이어가 윷놀이 게임의 상태를 본다. 즉 게임의 현재 상태를 화면에 그린다. 이때 게임의 상태는 윷놀이 판과 판 위의 말들, 누구의 차례인가, 윷 던지기 기회가 몇 번 남았는가 등을 의미한다.

- Use Case Description

Scope : 윷놀이 게임

Level : User Goal

Primary Actor : Player

Stakeholders and interests:

- Player : 윷놀이의 규칙을 따르며 자신의 말들을 최대한 빨리 도착하게 한다.

Precondition : 윷놀이 규칙

Success guarantee : 윷놀이 규칙에 따라 프로그램이 진행되다 보면 최소 한 명의 플레이어는 모든 말이 도착하게 되며 프로그램은 종료되게 된다.

Main success scenario:

1. 플레이어가 플레이어 수와 말의 수를 선택한다.
2. 윷을 던진다.
3. 윷의 결과에 따라 말을 움직인다.
4. 자신의 차례를 기다린다.
5. 한 명의 플레이어가 승리할 때까지 2~4 를 반복한다.

Alternative flows :

2 – 1) 윷의 결과가 윷, 모인 경우 윷을 한번 더 던진다.

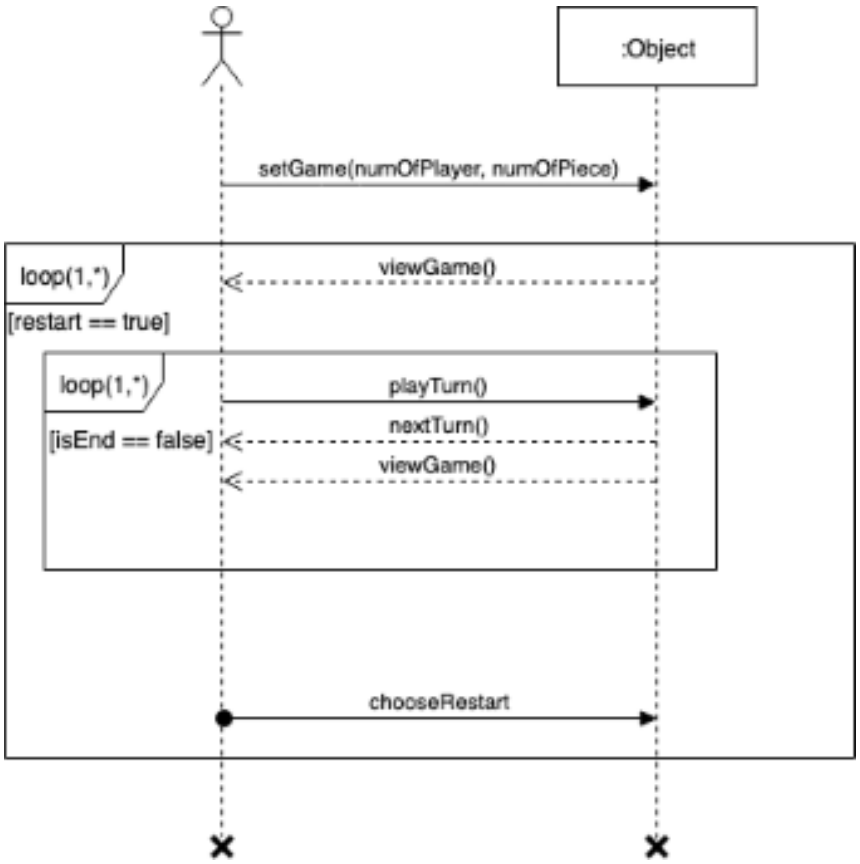
3 – 1) 말이 움직인 위치에 상대 말이 있는 경우 그 말을 잡고 윷을 한번 더 던진다.

3 – 2) 말이 움직인 위치에 자신의 말이 있는 경우 말을 업는다.

3 – 3) 말이 시작점을 넘는 경우(도착하는 경우) 말이 나오며 어떤 윷 결과를 사용할 지 선택한다.

4 – 1) 상대편이 나의 말을 잡으면 윷놀이 판 밖으로 옮긴다.

- System Sequence Diagram



- System Sequence Diagram -

게임을 시작하면 플레이어의 수와 플레이어 당 말의 개수를 설정하는 화면이 나온다. 이들을 설정하면 윷놀이 게임이 시작된다. 윷을 던지고 말을 옮기는 행위를 자신의 차례마다 윷놀이 게임이 종료될 때까지 반복한다. 게임이 끝나면 Actor 에게 게임 재시작 여부를 물어보는 팝업창이 나온다. Actor 가 Exit 을 선택하면 프로그램이 종료되고 Restart 를 선택하면 윷놀이 게임이 재시작 된다.

- Operation Contract

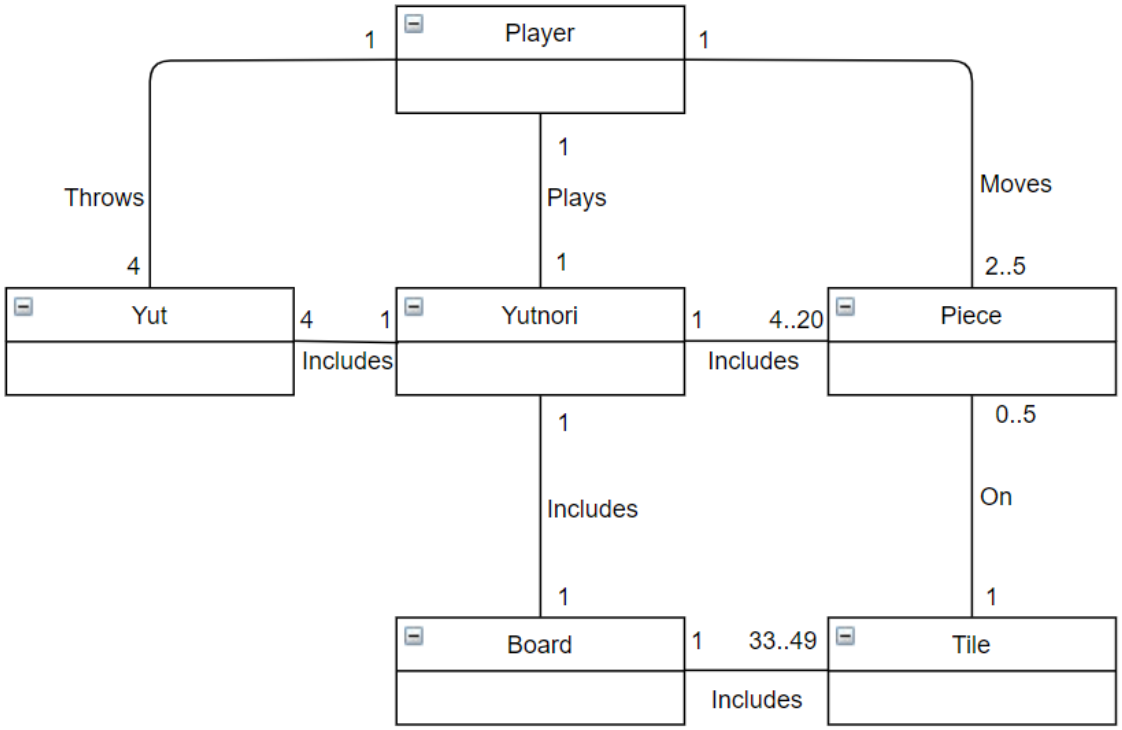
Operation:	throwYut
CrossReferences:	Use Case : Throw Yut
Preconditions:	게임 시작 후 특정 플레이어의 순서에 (특정 플레이어가 윷 던지기 기회가 남아 있을 때)
Postconditions:	윷을 4 개 던진 뒤 결과를 벡터에다 저장하고 게임 상태를 플레이어가 말을 움직이기 기다리는 상태로 바꿨다.

Operation:	movePiece
CrossReferences:	Use Case : Move Piece
Preconditions:	윷을 던진 뒤 (결과 벡터에 움직임이 남아 있을 때)
Postconditions:	윷놀이 규칙에 맞게 말을 움직인 뒤 움직인 거리를 윷 결과 저장 벡터에서 삭제했다.

Operation:	startGame
CrossReferences:	Use Case : set Game
Preconditions:	사용자가 플레이어의 수와 말의 개수를 선택한 뒤
Postconditions:	게임을 해당 설정으로 시작했다.

(4) Analysis, Design, Implementation

- Domain Model

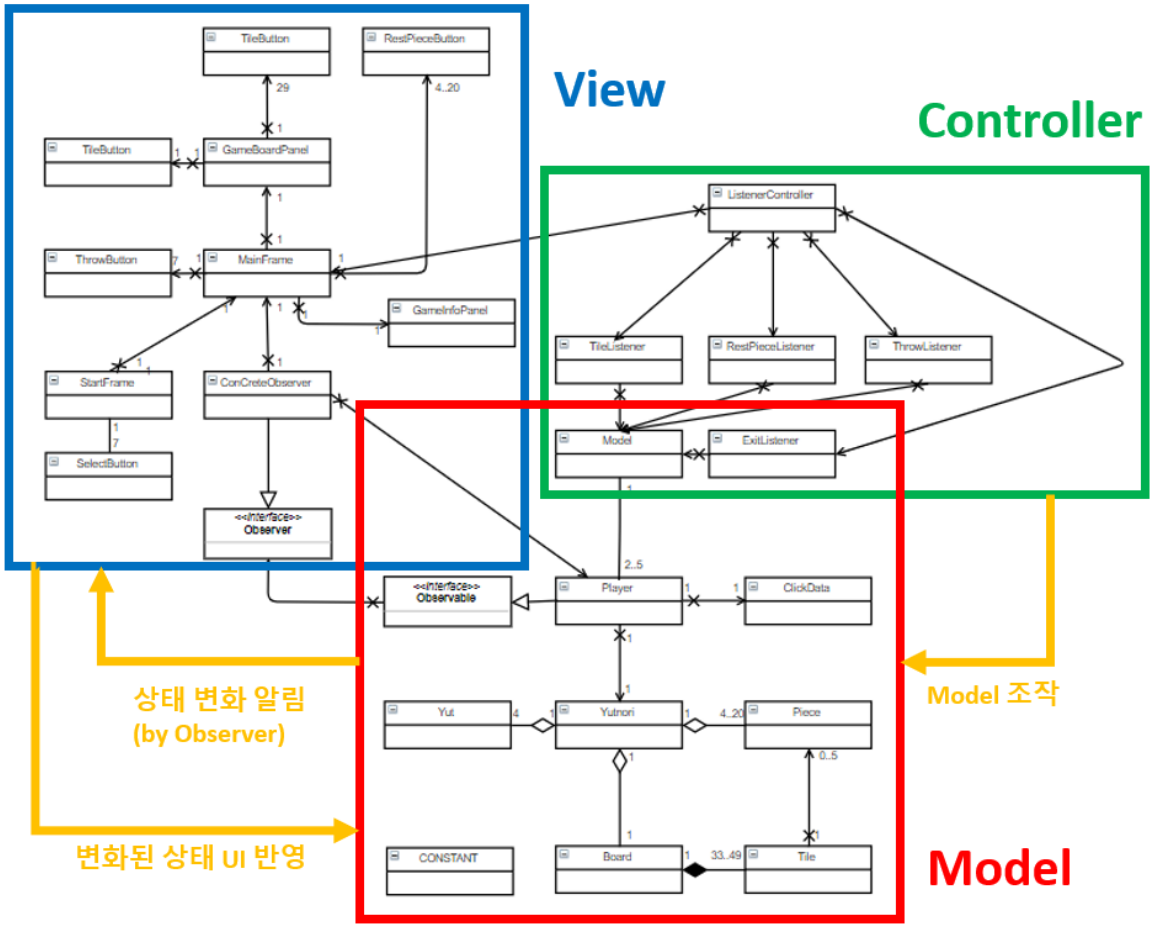


- Domain Model -

Class Diagram 보다 조금 더 Real World Domain 에 가깝게 모델링한 Domain Model 이다. 각 객체의 속성, 상속 관계 등은 생략하였다. Player 는 Yunori 를 하며 Yutnori 는 4 개의 Yut, 플레이어 수와 플레이어당 말의 개수에 따라 최소 4 개 최대 20 개의 Piece, 한 개의 Board 로 구성되어 있다. Board 는 여러 개의 Tile(각 칸 하나하나)로 구성되어 있으며 Player 는 4 개의 윷을 던지고 각 말들을 움직일 수 있다.

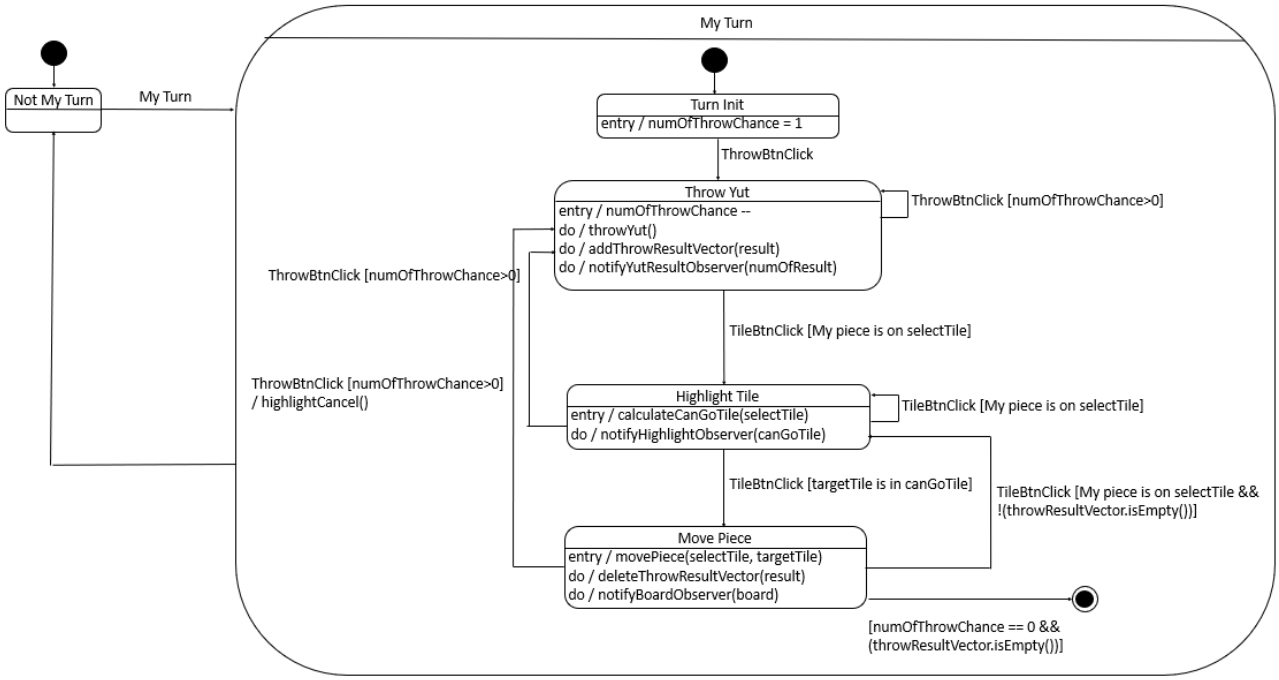
- Software Architecture & Design Model

- Class Diagram



- 간략화 한 Class Diagram -

- State Chart



- Player Object Start Chart –

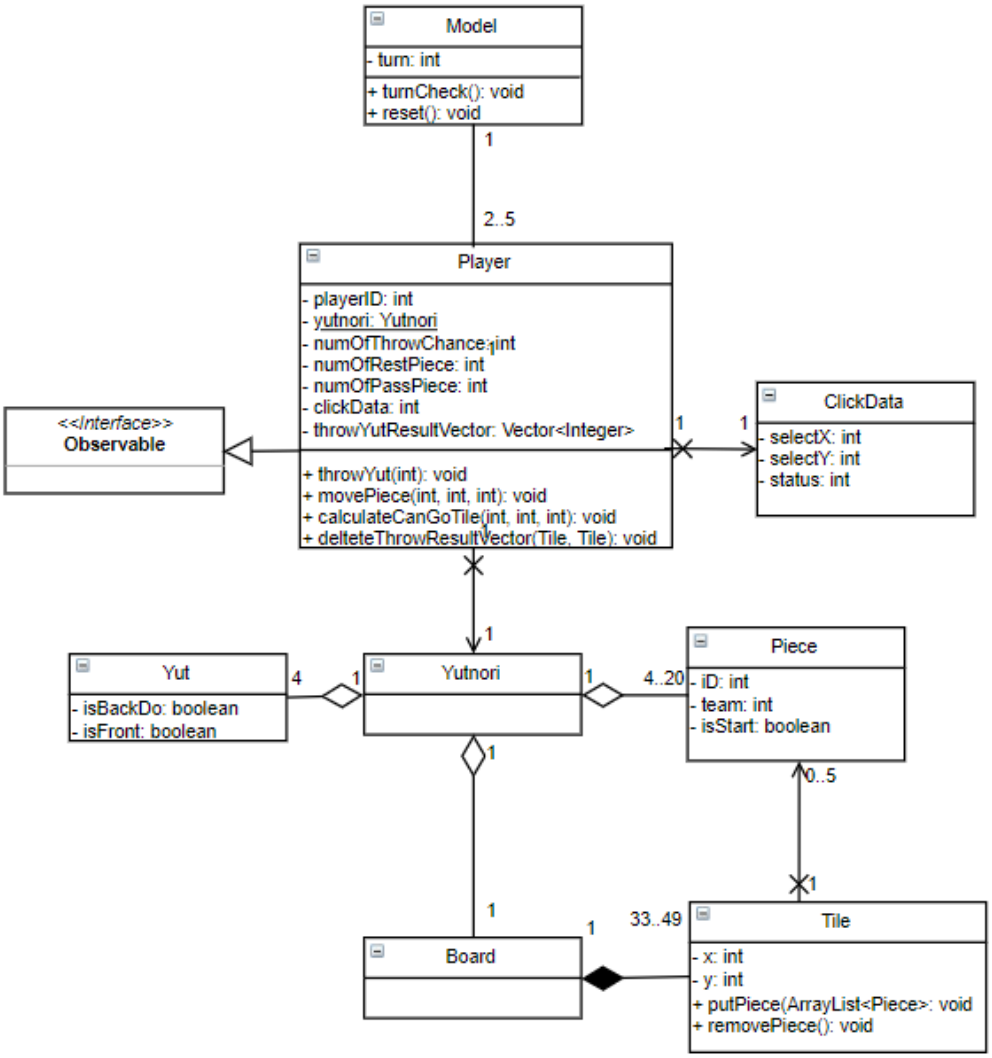
Player 객체는 크게 내 차례(**My Turn**)와 내 차례가 아님(**Not My Turn**) 두가지 상태로 나뉜다. **My Turn** 상태로 들어감과 동시에 **Turn Init** 상태로 들어가며 윷 던지기 기회(numOfThrowChance)가 1 로 초기화된다. ThrowBtnClick 이벤트가 발생하면 **Throw Yut** 상태로 들어가며, 윷을 던졌으므로 numOfThrowChance 를 1 만큼 감소시키고 throwYut 의 결과를 throwResultVector 에 추가한다. 이때 throwYut 의 결과가 윷 또는 모라면 numOfThrowChance 를 1 만큼 증가시킨다. (해당 증가는 throwYut 메소드 내부에 구현되어 있다.) 이때 부여된 추가 윷 던지기 기회는 Throw Yut 상태 뿐 아니라 추후, **Highlight Tile**, **Move Piece** 상태에서도 ThrowBtnClick 이벤트를 통해 사용 가능하다. Throw Yut 상태에서 TileBtnClick 이벤트가 발생하고 이때 Tile 위에 자신의 말이 존재한다면 **Highlight Tile** 상태로 들어간다. **Highlight Tile** 상태에 들어오면서 선택된 타일에서 이동 가능한 타일들을 계산하고 이를 notifyHighlightObserver 메소드를 통해 윷놀이 판 위에 하이라이트 한다. Highlight Tile 상태에서 TileBtnClick 이벤트가 발생하고 선택된 타일이 전 단계에서 계산한 이동 가능한 타일 중 하나라면 **Move Piece** 상태로

들어간다. **Move Piece** 상태에서는 movePiece 메소드를 통해 말을 움직이고 움직인 거리를 throwResultVector 에서 제거해준다. 또한, notifyBoardObserver 메소드를 통해 말의 움직임을 윷놀이 판으로 보여준다. **MovePiece** 상태에서 윷 던지기 기회가 남아있거나 (numOfThrowChance > 0), 아직 사용하지 않은 윷 던지기 결과가 남아 있다면 (!throwResultVector.isEmpty()) Actor 의 ThrowBtnClick 또는 TileBtnClick 이벤트를 기다리며, 그렇지 않다면 My Turn 상태에서 빠져나와 Not My Turn 상태에 들어가고 다시 My Turn 이 돌아오기를 기다린다.

- MVC Concept

Model

UI, 표시형식에 의존하지 않고 기능적 동작을 하도록 구현되었다. 특정 상태가 변화할 때마다 View 에 이를 통보하여 (이와 관련해서는 아래 Observer Pattern 에서 자세히 설명할 것이다.) UI 를 최신 상태로 업데이트 한다.



- Model Class Diagram –

각 클래스의 Setter/Getter, 인터페이스 구현 메소드 등을 제외하고 주요 속성(attribute), 메소드를 표현한 Model 의 Class Diagram 이다. 각 클래스에 대해 자세히 알아보자.

① Model: 윷놀이 전반의 진행을 담당하는 클래스이다. 누구의 차례인지 나타내는 turn 과 관련된 메소드를 가지고 있으며 승자 결정 후 재시작시 게임을 초기화하는 메소드 reset 을 가지고 있다.

② Player: 윷놀이를 하는 플레이어를 나타내는 클래스이다. 출발하지 않은 말의 개수를 나태내는 numOfRestPiece, 이미 도착한 말의 개수를 나타내는 numOfPassPiece, 남은 윷 던지기 기회를 나타내는 numOfThrowChance, 아직 사용하지 않은 윷 던지기 결과를 저장하는 throwYutResultVector 등이 속성으로 존재한다. 또한 윷놀이 세트를 나타내는 Yutnori 객체 속성은 모든 플레이어가 공유하도록 static 으로 선언되었다. throwYut 메소드는 윷을 던지고 결과를 throwYutResultVector 에 결과를 저장하는 역할을, movePiece 메소드는 Piece 를 기존 타일에서 새 타일로 이동시키는 역할을, calculateCanGoTile 메소드는 이동 가능해 판 위에 하이라이트 할 좌표를 계산하는 역할을 한다.

③ ClickData: State Chart 에서 설명한 Player 의 상태를 저장하는 클래스이다.

④ Yutnori: Player 들이 공유하는 윷놀이 세트를 나타내는 클래스로 Yut, Piece, Board 클래스와 HAS-A 관계를 가진다.

⑤ Yut: 윷놀이의 윷을 나타내는 클래스로 앞 뒷면 정보를 저장하는 isFront, 백도인 윷을 나타내는 isBackDo 속성을 가지고 있다. Yutnori 객체 한 개는 Yut 객체 4 개와 관계를 맺는다.

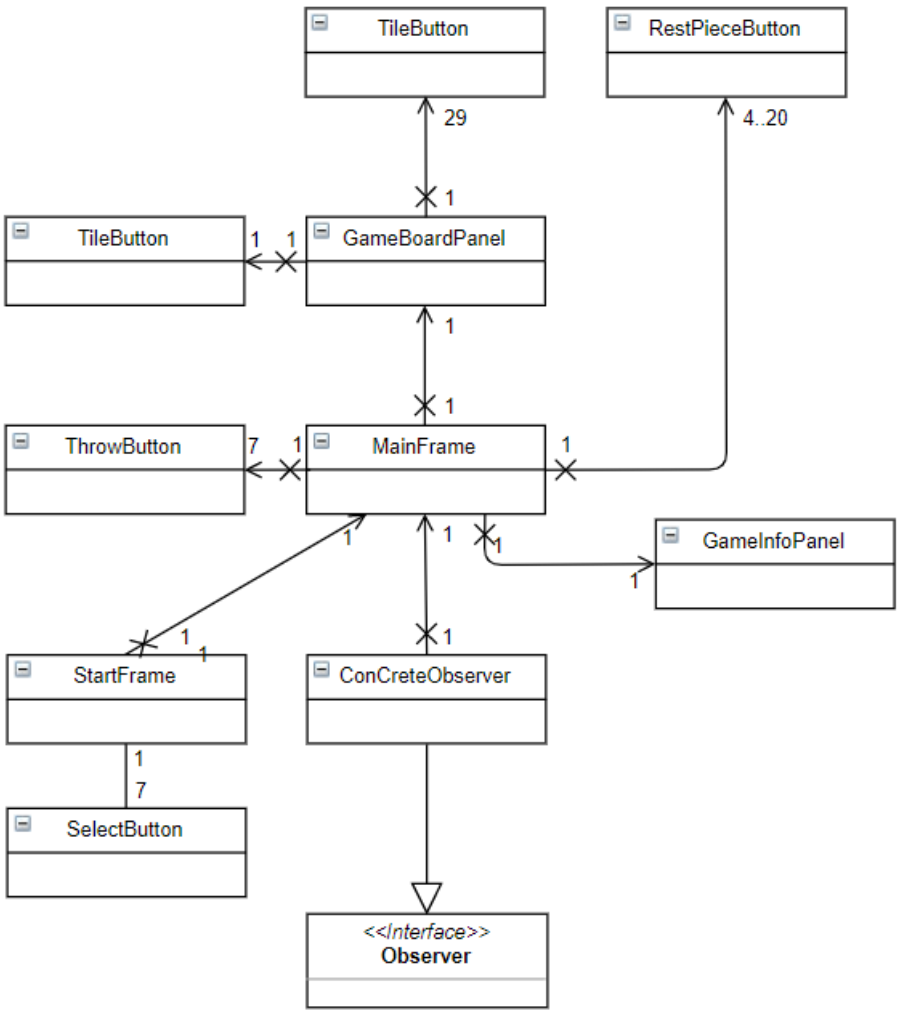
⑥ Piece: 윷놀이의 말을 나타내는 클래스로 팀 정보, 출발 여부를 저장하는 등의 속성을 가지고 있다. Yutnori 객체 한 개는 Piece 4 ~ 20 개와 관계를 맺는다.

⑦ Board: 윷놀이의 판을 나타내는 클래스로 33 ~ 49 개의 Tile 객체와 Composition 관계를 가진다.

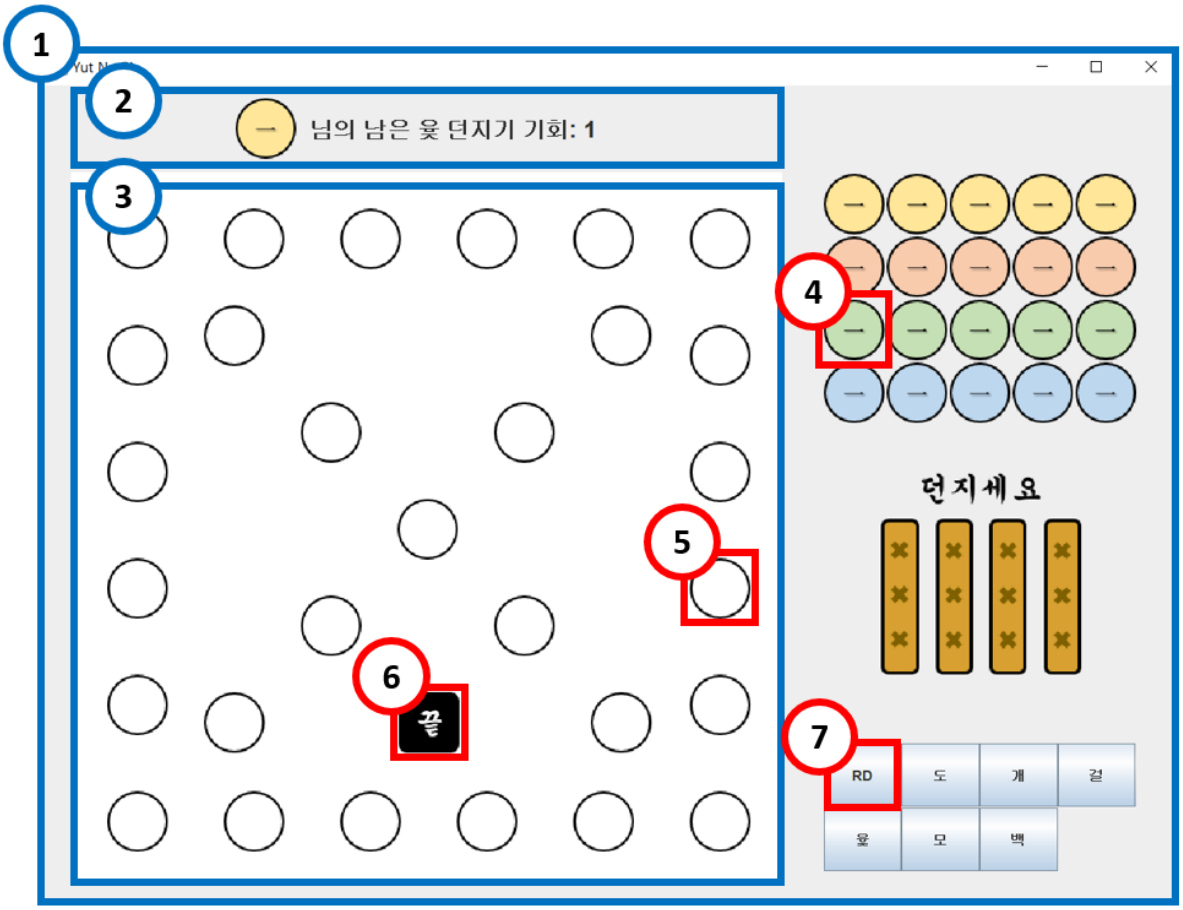
⑧ Tile: 윷놀이 판의 각 칸을 나타내는 클래스로 한 개의 Tile 에는 최대 5 개의 Piece 까지 ArrayList 형태로 저장할 수 있다. putPiece 메소드는 Tile 위에 말을 올리는 역할을, removePiece 메소드는 Tile 위의 말을 제거하는 역할을 한다.

View (UI Component)

각 UI Component 를 초기화 하며 사용자가 볼 결과물을 생성하기 위해 Model 로부터 정보를 얻어온다. 이때 모델의 상태 변화를 감지하기 위해 Observer Pattern 을 사용하였다.



- 간략화 한 View Class Diagram -



- Main Frame UI Component-

① MainFrame: 윷놀이 게임의 메인 Frame

② GameInfoPanel: 누구의 턴인지, 윷 던지기 기회는 몇 번 남았는지 등 게임 진행 관련 정보를 보여주는 Panel

③ GameBoardPanel: 윷 놀이 판 Panel 로 29 개의 TileButton 과 한 개의 ExitButton 을 가지고 있는 Panel

④ RestPieceButton: 윷 놀이 판 밖에서 대기하고 있는 말들의 Button

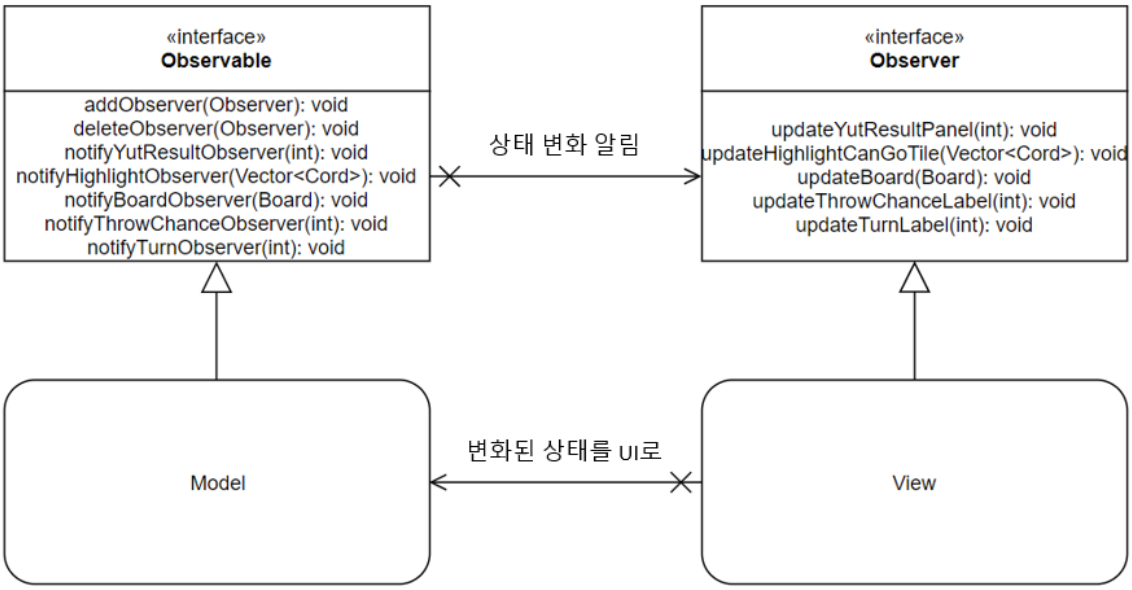
⑤ TileButton: 윷 놀이 판 위에 위치하고 있는 (Tile)칸들의 Button

⑥ ExitButton: 윷 놀이 판 위에 위치하며 도착을 위해 사용되는 Button

⑦ ThrowButton: RD(랜덤)하게 또는 백도, 도 ~ 모 원하는 윷의 결과를 반환해 주는 윷 던지기 Button

Observer Pattern

Model 의 상태가 변하였을 때 이를 View 에 알려주기 위해 Observer Pattern 을 사용하였다. 이를 통해 Model 과 View 사이의 결합을 더욱 느슨하게 하여 Model 과 UI 를 분리하여 구현할 수 있었다. Controller 에서의 메소드 호출로 인해 Model 에서 이벤트가 발생하고 특정 상태가 바뀌면 Model 은 Observable 인터페이스에 선언된 notify~~() 메소드를 호출한다. 이는 View 에서 Observer 인터페이스에 선언된 update~~() 메소드를 콜백함수로 호출하여 Model 상태에 맞게 View 를 변경해 준다. java.util API 에서 제공하는 Observable, Observer 인터페이스를 사용하지 않고 직접 구현하여 사용하였다.



- Observer Pattern -

- addObserver 와 deleteObserver 메소드는 Model 의 상태변화를 전달받을 Observer 를 등록하고 제거하는 메소드이다.

- notifyYutResultObserver 메소드는 Model 의 throwYut 메소드가 호출되어 윷의 결과가 변화할 때 마다 호출되며 Observer 의 updateYutResultPanel 메소드를 콜백함수로 호출하여 윷 결과 표시 라벨의 사진을 업데이트한다.

- notifyHighlightObserver 메소드는 Model 의 calculateCanGoTile 메소드가 호출되어 선택된 말이 갈 수 있는 좌표가 변화할 때 마다 호출되며 Observer 의 updateHighlightCanGoTile 메소드를 콜백함수로 호출하여 갈 수 있는 좌표에 해당하는 TileButton 을 하이라이트 한다.
- notifyBoardObserver 메소드는 Model 의 movePiece 메소드가 호출되어 말이 움직이고 Board 의 상태가 변화할 때 마다 호출되며 Observer 의 updateBoard 메소드를 콜백함수로 호출하여 각 칸마다 위치한 말의 팀과 개수에 맞게 GameBoardPanel 을 업데이트 한다.
- notifyThrowChanceObserver 와 notifyTurnObserver 메소드는 Model 의 윷 던지기 기회와 차례 상태가 변화할 때 마다 호출되며 Observer 의 콜백함수를 각각 호출하여 GameInfoPanel 을 업데이트 한다.

Controller

위 UI Component 의 ④ ~ ⑦ 버튼에 대한 Listener 가 구현되어 있으며 이들은 사용자의 입력을 감지하여 알맞게 Model 의 상태를 변화시킨다. ListenerController 클래스는 View 의 각 컴포넌트에 Controller 에서 정의한 Listener 를 심어주는 역할을 하며 Listener Class 들은 Model 의 메소드를 호출하여 Model 상태를 변경한다.

```
public class ListenerController {

    public ListenerController(Model model, MainFrame mainFrame) {

        for(int i=0; i<mainFrame.getThrowButton().length; i++) {
            mainFrame.getThrowButton()[i].addThrowListener(new ThrowListener(model, i));
        }

        for(int i=0; i<CONSTANT.PLAYERNUM; i++) {
            for(int j=0; j<CONSTANT.PIECENUM; j++) {
                mainFrame.getRestPieceButton()[i][j].addRestPieceListener(new RestPieceListener(model, i, j));
            }
        }
```

- ListnerController Class 중 일부-

```java
public class RestPieceListener implements ActionListener {
    Model model;
    int x, y;

    public RestPieceListener(Model model, int x, int y) {
        this.x = x;
        this.y = y;
        this.model = model;
    }

    public void actionPerformed(ActionEvent e) {
        model.getTurnPlayer().calculateCanGoTile(x, y, 0);
    }
}
```

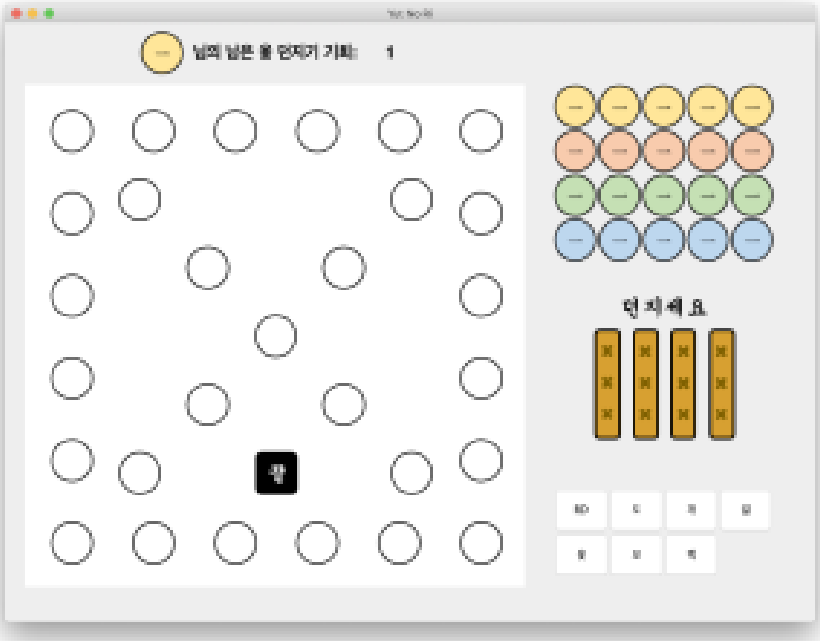
-RestPieceListener Class-

코드를 통해 살펴보면, Controller 에 구현된 Listener 중 하나인 RestPieceListener 는 ListenerController 클래스에 의해 UI Component 중 ④ RestPieceButton 에 심어진다. 또한 Listener 는 Model 의 내부와 상관없이 간단하게 Model 의 메소드를 호출하도록 구현되어 있다.

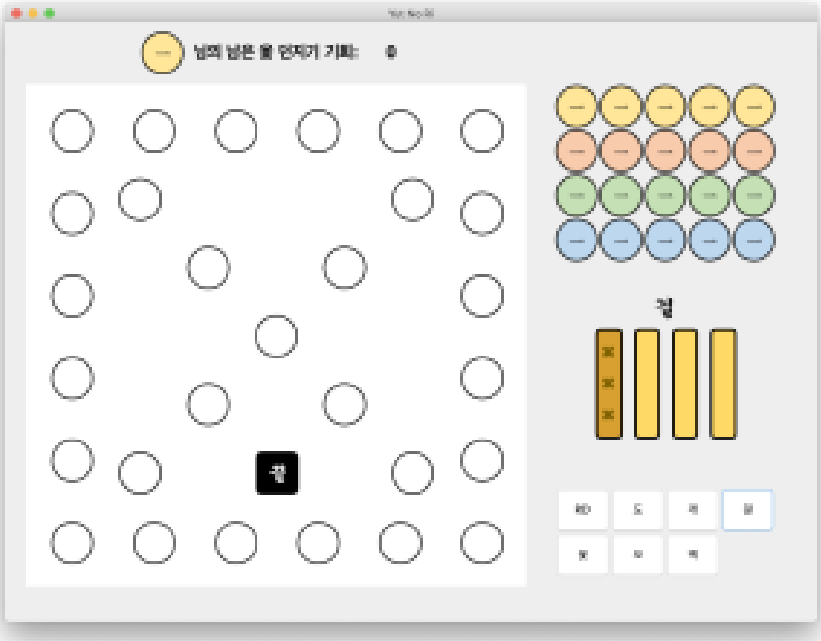
- Usage of Program & Screen Shot

- 게임 설정(플레이어 수, 말 수 선택)

- 게임 기본 화면

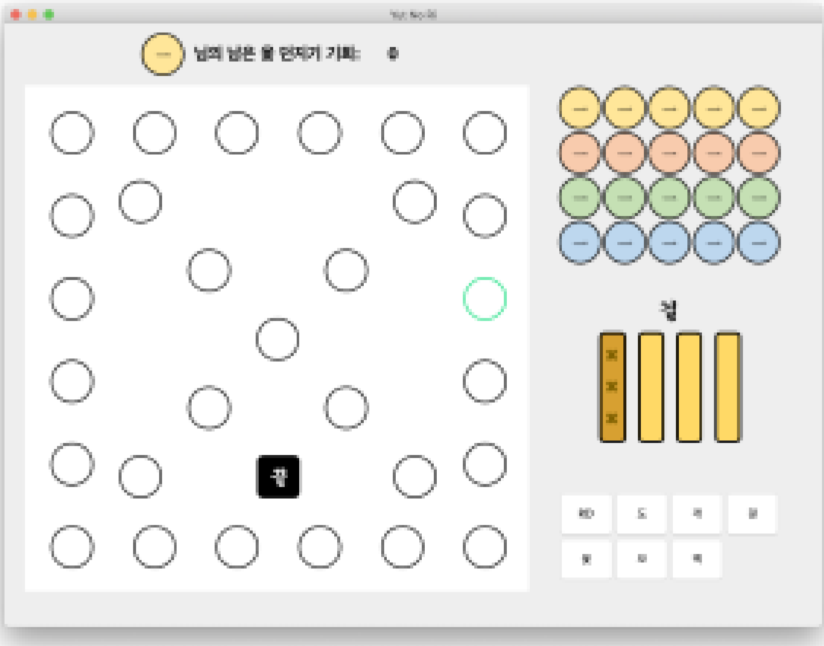


- 윷 던지기 버튼 클릭 시(결과를 윷 사진으로 표시)

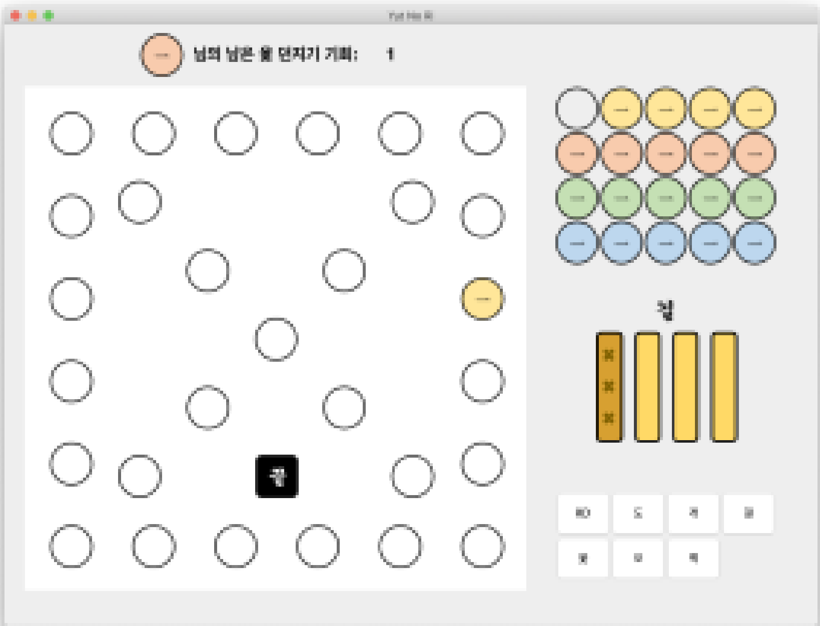


- 말 움직이기

1) 움직일 말을 선택하면 초록색 원으로 이동 가능한 칸을 하이라이트

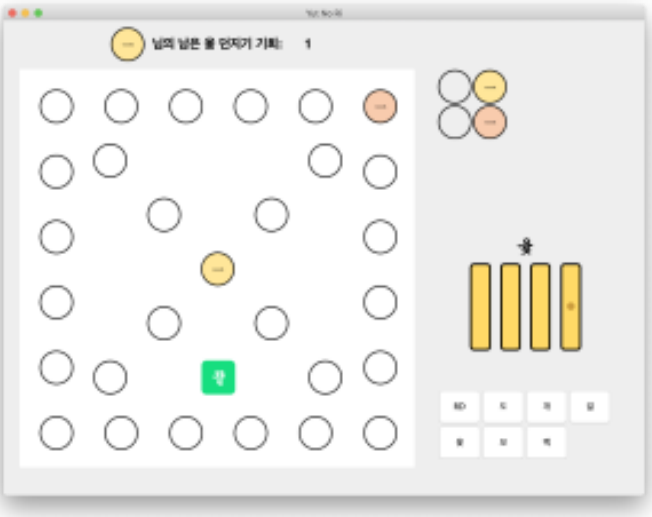


2) 이동가능한 칸 중 이동할 칸을 선택하면 말이 이동

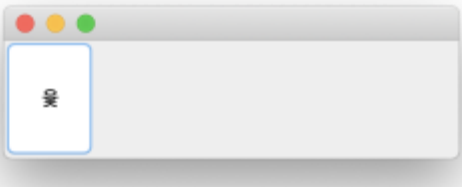


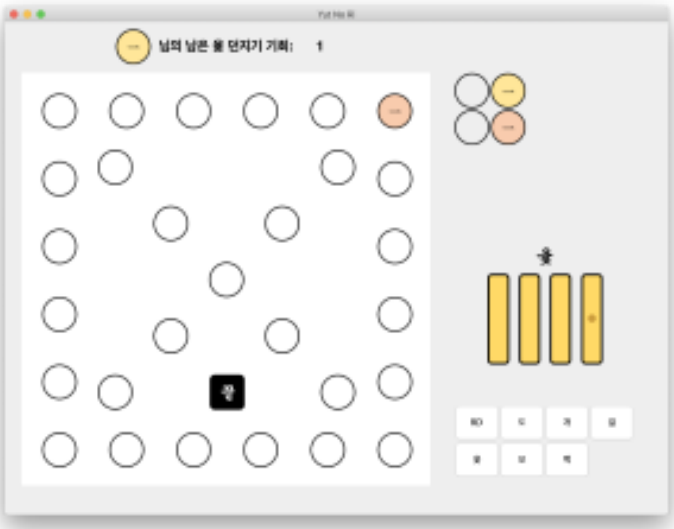
- 말 나가기 (어떤 결과를 사용하여 나갈 것인지 선택)

1) 도착 가능한 말을 클릭하면 초록색으로 '끝' 칸 하이라이트

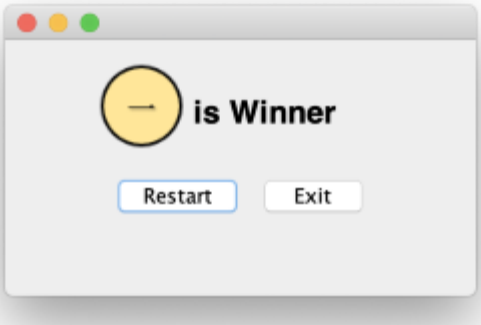


2) '끝' 칸을 클릭하면 사용할 윷 결과 선택 팝업창 표시 & 선택한 말을 판 위에서 제거





- 게임 종료(승자 표시, 재시작/종료 선택 가능)



(5) Project Management Report

- Github Repo

https://github.com/kimyoungi99/yutnori

- Project Progress History

1 일차 (5.23) : 요구사항 정립 및 MVC Concept 기반 Architecture Design

6 일차 (5.28) : Prototype 개발 완료 및 Prototype Review

7 일차 (5.29) : 윷, 말, 플레이어 등 Model 일부 1 차 구현 완료

9 일차 (5.31) : 윷 던지기 버튼, 말 버튼 등 Controller, View 1 차 구현 완료

10 일차 (6.1) : Model 이 View 에 상태 변화를 알려주기 위해 Observer 패턴 적용

14 일차 (6.5) : Model, View, Controller 2 차 구현 완료

15 일차 (6.6) : 플레이어 수 및 플레이어 당 말 개수 선택 구현

16 일차 (6.7) : 재시작 구현, Trivial Bug 수정, 문서 작성

- Experience

과목이 소프트웨어 공학에 대한 내용을 다루기 때문에 저희 조는 윷놀이 자체보다는 MVC Concept과 객체지향적 분석과 디자인에 초점을 맞춰 공부하고 구현하는 것을 목표로 하였다. Github Commit History가 보여주듯 물론 기능적 요구사항 만족을 위해서도 많은 노력을 하였지만 MVC Concept을 지키는 등 비기능적 요구사항 만족을 위해 더 큰 노력을 하였다. MVC Concept을 직접 적용해 본 적이 처음이라, 초반에는 각 클래스가 Model, View, Controller 중 어디에 들어가야 하고 어떤 기능을 해야 하는지 많이 헷갈렸으나, 직접 적용을 시켜보며 MVC Concept에 대한 확신을 가지게 되었다. MVC Concept으로 UI와 Model을 분리하여 구현하니 확실히 오류가 생겼을 때 원인을 찾기 쉽고 유지보수가 쉬웠다.