**Software Engineering**

**Team Project**

**- 윷놀이 -**

**TEAM 7**

**20153283 이정원**

**20173198 이승찬**

**20172774 송정우**

**목차**

1. **Team Members & Roles**
2. **Vision**
   * **Short Introduction to the system**
   * **Main features**
3. **Use Case Model**
   * **Use case diagrams**
   * **Use case descriptions**
   * **System Sequence Diagrams**
   * **Operation Contracts**
4. **Analysis & Design, Implementation, and Test Document**
   * **Domain model**
   * **Software Architecture + Design Model**
5. **Project Management Report**
   * [**https://github.com/bluayer/yut/**](https://github.com/bluayer/yut/)
   * **Project progress history**
   * **Brief experience**
6. **Team Members & Roles**

**20153283 이정원**

**: Controller, Requirements Analysis, State Chart Diagram 작성**

**20173198 이승찬**

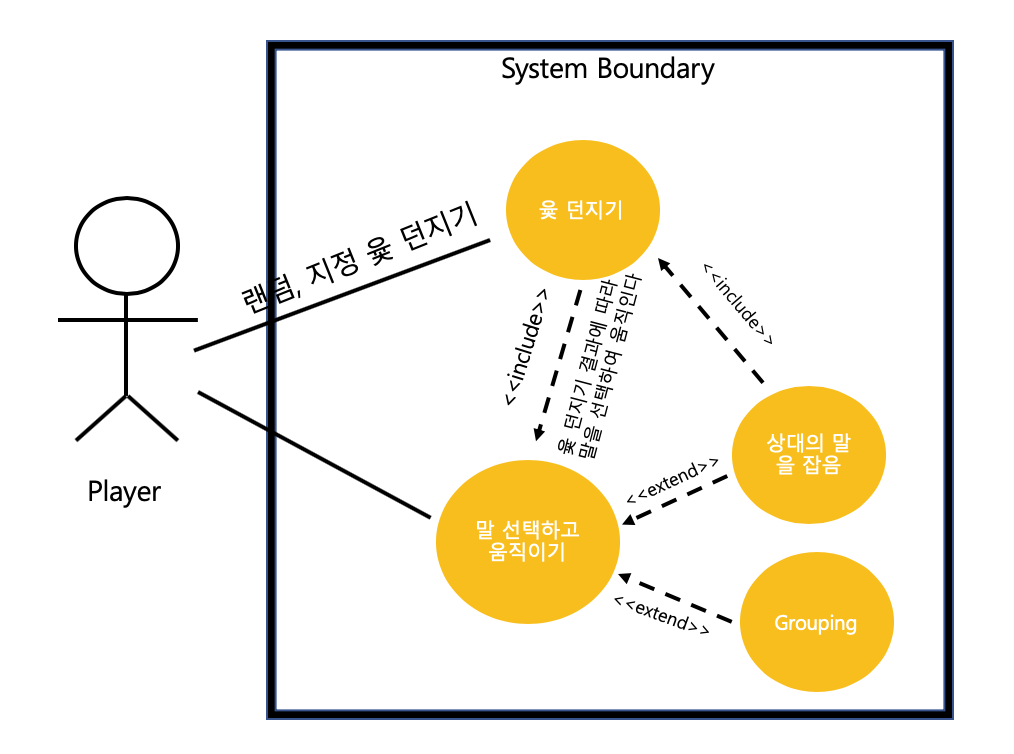
**: Model, Use case description, Class Diagram 작성**

**20172774 송정우**

**: View, Use case Diagram, System Sequence Diagram 작성**

1. **Vision**
   * **Short introduction to the system**

* Java Swing을 이용하여 2~4명의 플레이어가 함께 할 수 있는 윷놀이 게임을 OOAD 기법과 MVC 패턴을 이용하여 개발
  + **Main Features**
* 사용자는 게임이 시작하기 전 게임에 참여할 플레이어 수와 각 플레이어당 가질 수 있는 말의 수를 설정하고 게임을 시작 한다.
* 게임이 시작되면 사용자는 윷 던지기, 말 선택하기, 말 움직이기 등의 동작을 할 수 있으며 각 동작의 결과에 따른 이벤트들이 발생하게 된다.
* 게임이 종료되면 사용자는 게임을 재 시작 할지 그대로 게임을 종료시킬지 결정 할 수 있다.

1. **Use Case Model**
   * **Use Case Diagram**

윷놀이 게임은 Primary Actor인 게임 player가 윷을 던지는 것과 던져진 결과에 의해 말을 선택하고 움직인다. 따라서 말 선택하고 움직이기를 하기 위해서는 말 선택하고 움직이기가 필요하기 때문에 include 관계가 생기고, Base는 윷 던지기가 되며 말 선택하고 움직이기가 included가 된다. 또한 말 선택하고 움직이는 과정은 Grouping이 필요할 수도 있지만, 꼭 그래야 할 필요는 없기 때문에 extend 관계가 성립하게 되고 말 선택하고 움직이기는 Base가 되며 Grouping은 Extending이 된다. 또한 말 선택하고 움직이는 과정에서 상대방의 말을 잡을 수도 있고 아닐 수도 있기 때문에 extend 관계라고 판단하였으며, 상대방의 말을 잡게 되면 윷 던지기가 필연적으로 일어나야 하므로, include 관계가 성립하게 된다.

* + **Use Case Desrciption**

**1. Scope :** 윷놀이 게임

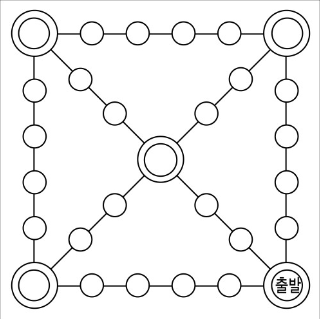
**2. Level :** user goal

**3. Primary Actor :** Player

**4. Stakeholders and Interests:**

**- Player :** 게임 시작, 윷 던지기, 말 이동하기, 말 업기, 게임 종료 및 재시작을 원한다.

**4. Preconditions:**

****Player는 2~4인이다. 말의 개수는 2~5개이다. 또한 윷놀이 판은 다음 그림과 같다.

**5. Success Guarantee ( or Post conditions)**

- 적어도 한 명 이상의 Player의 말이 모두 내보내졌다.

- 게임 승리 시, 승리 Player를 화면에 명시한다.

- 게임 종료와 재시작 중 선택할 수 있게 한다.

**6. Main Success Scenario(or Basic Flow):**

1) Player가 윷놀이 게임 시작 버튼을 누른다.

2) Player의 수를 정한 후, 주어질 말의 개수를 정한다.

3) random하게 생성된 턴 순서대로 지정 윷 던지기와 랜덤 윷 던지기 중 하나를 눌러 윷을 던진다.

* 윷이나 모가 나왔을 경우, 한 번 더 윷을 던질 수 있다.

4) Player는 나온 윷의 결과(빽도, 도, 개, 걸, 윷, 모)를 적용할 말을 선택한다.

5) Player는 선택한 말을 이동시킨다.

* 말을 옮길 위치에 다른 말이 있는 경우
  + 본인의 말이라면, 무조건 grouping한다.
  + 상대방의 말이 있다면 상대방의 말을 잡고 3)과정부터 다시 실행한다.
* 이동 가능 위치가 여러 개일 경우 먼저 하나를 선택한다.
* 이동 횟수가 남은 경우 4)과정부터 다시 실행한다.

시스템은 각 턴에 Step 3, 4, 5를 실행하고, 이를 승리자가 나올 때까지 반복한다.

6) 한 Player의 모든 말이 나오게 되면 승리했음을 화면에 띄운다.

7) 게임을 종료할지, 재 시작할지 Player가 선택한다.

**7. Extensions (or Alternative Flows)**

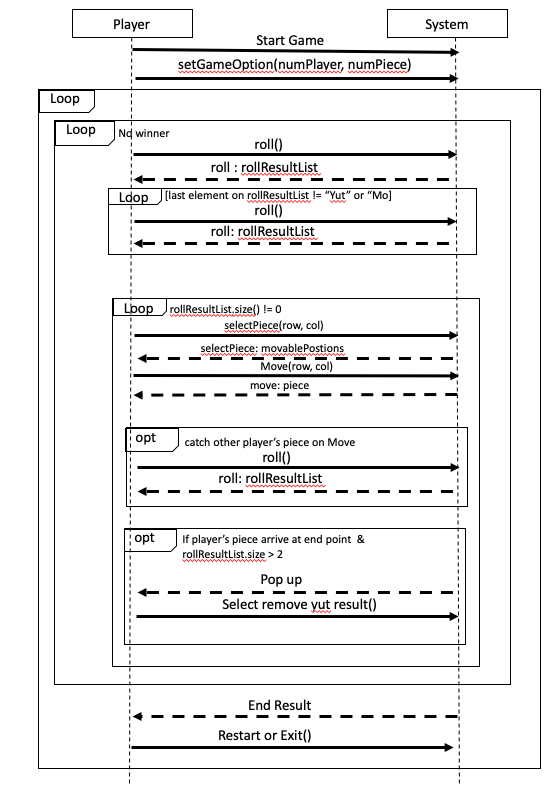
**\*a. At any time, 판 위에 말이 없을 때 ‘빽도’가 나온 경우**

- 그냥 ‘도’로 적용한다.

**\*b. At playing time, 모나 윷이 전체 말을 goal로 옮길 정도로 많이 나온 경우**.

- 해당 플레이어의 승리로 처리하고 6)단계로 점프한다.

* + **System Sequence Diagram**

****

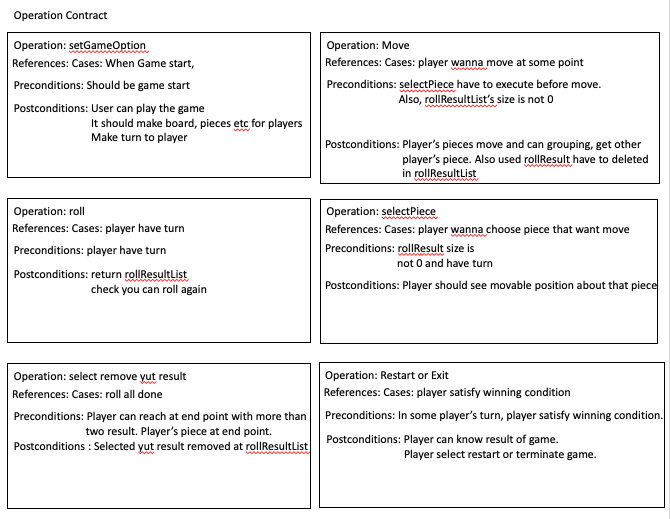
* **SSD**

Player가 첫 화면에서 옵션을 입력하고, 그 옵션은 플레이어 수와 피스의 수가 된다. 그리고 나서 게임을 시작하게 되는데, Loop 안에서 게임을 시작하게 된다. 윷을 던지고 말을 움직이는 일련의 과정을 승리자가 나오기 전까지 하나의 Loop에서 하게 되고 밖의 Loop에서는 승리자가 나오면 결과에 따라 플레이어가 재 시작 혹은 종료 여부를 결정할 수 있게 된다.

윷놀이 게임 내부에서는 roll을 하는데 윷이나 모가 나오게 되면 윷을 한 번 더 던지게 되며 결과가 윷이나 모면 계속 던지게 된다. 윷을 다 던진 후에는 윷의 결과가 0개가 아니라면 selectPiece를 통해 피스를 선택할 수 있게 되고, 그러면 피스가 움직일 수 있는 곳을 보여준다. 움직일 수 있는 곳 중에 하나를 선택하여 움직이게 되면 움직인 결과를 보여준다. 만약 그 과정에서 다른 플레이어의 피스를 잡게 된다면, 윷을 한 번 더 던질 수 있게 된다. 만약 이 과정에서 피스가 끝 지점에 도달하게 됐으며, 움직일 수 있는 결과가 2개 이상이라면 Pop up 창을 띄우고 플레이어는 끝 지점에 도달 가능한 결과들 중 하나를 선택하여 지울 수 있게 한다.

이런 일련의 과정들이 끝나게 되면 결과를 보여주고, 재 시작 할지 종료할지 선택하게 하여 게임을 진행한다.

* + **Operation Contracts**

****

* **Operation Contract**

다음은 Operation contract에 대한 간략한 설명이다.

setGameOption은 플레이어가 piece와 player 개수에 대한 수를 넘겨주고 게임을 시작하도록 하는 함수이다.

roll은 플레이어가 턴을 가지고 있을 때, 윷 결과를 받는 것으로 rollResultList라는 던진 결과들이 있는 리스트를 받게 되고 만약 결과가 윷이나 모인지에 따라 다시 던질 수 있어야 한다.

selectPiece는 플레이어의 피스를 선택하게 되면 피스가 갈 수 있는 곳을 보여줄 수 있도록 하는 함수이다.

Move는 피스가 이미 선택됐으며, 윷을 던진 결과가 있을 때 실행 가능한 것으로 실행하고 나면 어디로 움직였는지 사용자에게 보여주고 움직이는데 사용한 윷 던진 결과값을 리스트에서 삭제한다.

select remove yut result는 유저의 피스가 엔드 포인트에 도착했으며, 던진 윷의 결과들 중에 엔드 포인트에 도달 가능한 결과가 2개 이상일 때, 유저가 결과들 중 하나를 선택하여 삭제할 수 있게 한다.

Restart or Exit은 어떤 플레이어가 이기는 조건을 만족했을 경우, 게임의 결과를 알려준 다음 플레이어가 재 시작 혹은 종료를 선택할 수 있도록 하는 기능이다

1. **Analysis & Design, implementation**
   * **Requirements analysis**

Gathered Requirements

* FR
  1. 플레이어의 수를 직접 지정 할 수 있다.(2~4인)
  2. 한 플레이어당 말의 개수를 직접 지정 할 수 있다.(2~5개)
  3. 각 플레이어는 게임 진행을 위한 랜덤 윷 던지기와 테스트 진행을 위한 지정 윷 던지기를 할 수 있다.
  4. 플레이어는 윷의 결과를 적용할 말을 선택할 수 있다.
  5. 말들은 서로 업어 갈 수 있다.(Grouping)
  6. 한 플레이어의 모든 말들이 윷놀이 판에서 나가면 승리가 결정되고 게임이 종료된다.
  7. 게임이 끝난다면 게임을 재 시작할 지 그대로 끝낼지 결정할 수 있다.
* NFR
  1. 수업시간에 다룬 OOAD 기법을 사용하여 소프트웨어를 개발해야 한다.
  2. MVC 아키텍처 패턴을 사용하여 UI 와 Model을 분리하여 개발해야 한다.
  3. 유닛 테스트를 진행 할 수 있어야 한다.

Specified Requirements

* FR
  1. 게임이 시작되기 전에 플레이어의 수를 미리 정해준다.
  2. 게임이 시작되기 전에 각 플레이어가 가질 말의 수를 미리 정해준다.
  3. 게임이 시작되면 플레이어는 정해진 순서에 따라 윷을 던지는데 실제 게임 진행을 위한 랜덤 윷 던지기 버튼과 테스트를 위한 지정 윷 던지기 중 선택을 할 수 있다.
  4. 플레이어 생성 순서가 턴 순서가 된다.
  5. 자기 차례에 윷을 던지면 나온 결과(빽도, 도, 개, 걸, 윷, 모)에 따라 플레이어는 아직 판에 올려지지 않은 말들과 판에 있는 말들 중 선택하여 그 결과를 적용 할 수 있다.
  6. 던져진 윷의 결과가 ‘윷’ 또는 ‘모’라면 윷을 한번 더 던질 수 있다.

6.1 윷을 한 번 더 던진 후, 나온 결과를 선택하여 원하는 말을 움직일 수 있다.

* 1. 플레이어는 자기 턴에 윷을 던지고 말이 이동 한 후, 다른 플레이어의 말을 잡았다면 한번 더 윷을 던질 수 있다.
  2. (Grouping 조건) 만약 말을 움직였을 때 자신의 말과 같은 칸에 가게 된다면 말들은 서로 업혀 다음 차례부터는 함께 이동할 수 있다.

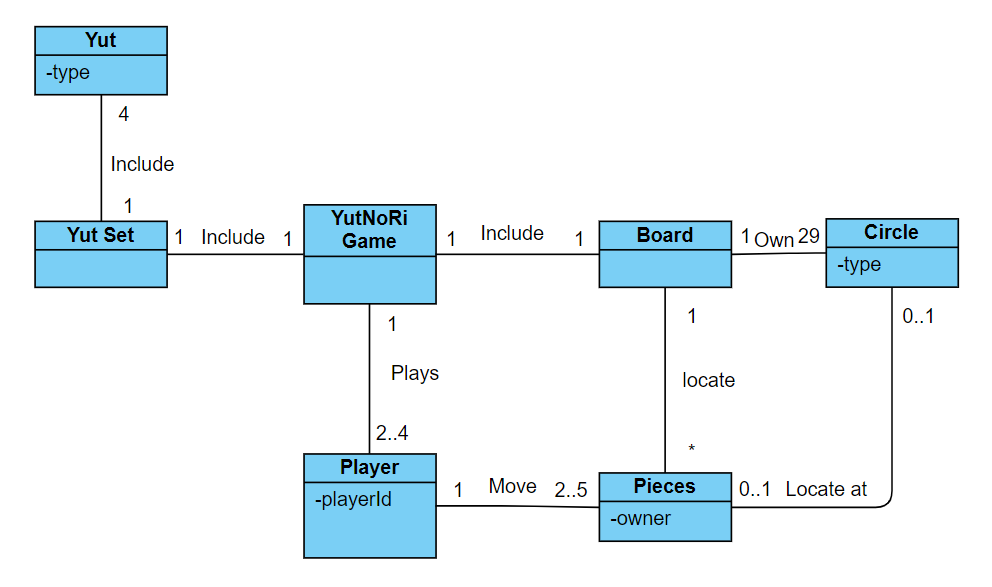
8.1 같은 칸에 같은 팀의 말이 있는 경우, 업히거나 업히지 않은 상태를 선택할 수 있다.

* 1. 한 플레이어가 모든 행동을 마치면, 차례가 다음 플레이어로 넘기게 된다.
  2. 말들이 윷판의 각 모서리 칸 혹은 중앙 칸에 들어가면, 다음 이동 시에 어디로 이동할지 결정 할 수 있다.
  3. (완주 조건) 말이 마지막 칸(시작점)에 도착한다면 완주한 것으로 간주한다.

11.1 시작점에서 한 칸 움직인 말이 ‘빽도’로 움직여 시작점으로 가게 된 경우, 완주한 것으로 판단하지 않는다.

* 1. 한 플레이어의 모든 말들이 윷놀이 판에서 나가면 그 플레이어로 승리자가 결정되고 게임이 종료된다.
  2. 게임이 끝난다면 게임을 재 시작할지 그대로 끝낼지 결정할 수 있다.

* NFR
  1. OOAD 기법을 사용하기 위해 객체지향 언어인 JAVA 사용해 개발한다.
  2. JAVA Swing을 이용해 인터페이스를 구성한다.
  3. JAVA언어로 유닛테스트를 하기 위해 JUnit을 사용한다.
  4. 협업 을 통해 github을 이용하고 모든 활동(source & document 등)을 github에 기록한다.
  5. Use case 모델, 분석.설계.구현 문서 등의 프로젝트 문서 작성.
  + **Domain Model**

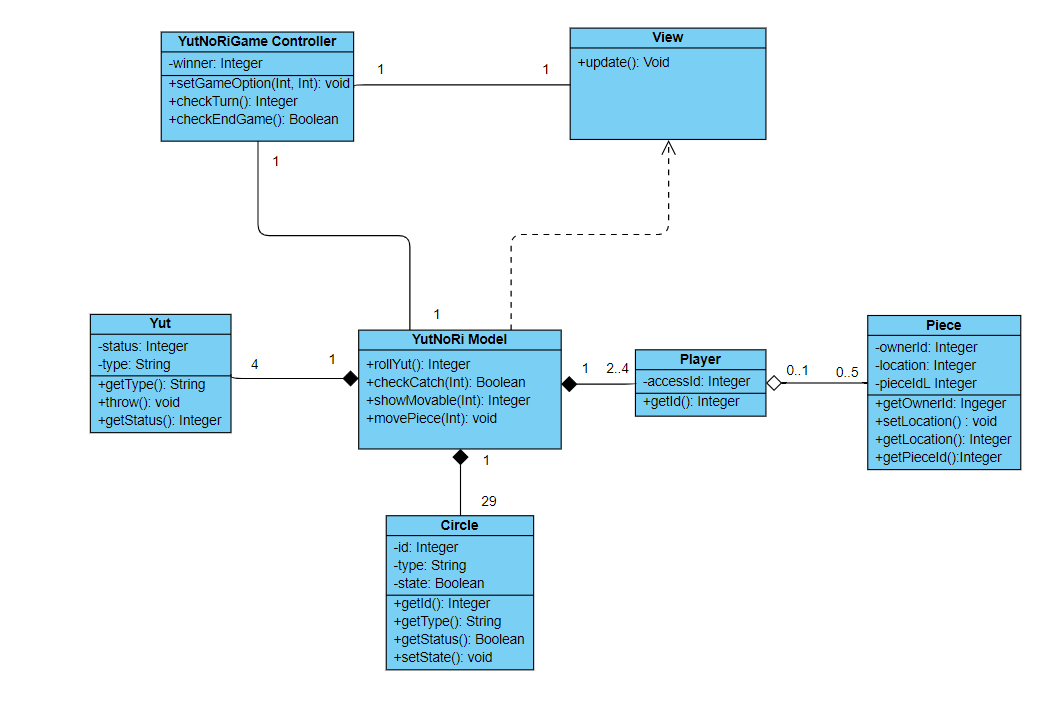
****

Domain Model에는 YutNoRi Game이라는 컨트롤러와 Player, Yut Set, Board등 윷노리 게임에 필요한 모델들이 존재 한다. YutNoRi Game은 각 모델들의 정보를 조회, 갱신하여 게임을 진행한다.

먼저 YutNoRi Game도메인은 Yut Set과 Player, Board 도메인을 각각 1개, 2~4개, 1개 가진다. Yut Set은 4개의 Yut을 가지고, 각각의 Yut은 앞면과 뒷면을 나타내는 type을 가진다. Board는 윷놀이 판에서 각 노드에 해당하는 29개의 Circle을 가진다. Player는 각각 2~5개의 Piece를 가진다. Piece는 Board에 여럿 올라갈 수 있고, 하나의 Circle 위에 올라간다.

이러한 도메인 간의 관계, 도메인에 대한 개념을 설계 이전에 확실하게 잡으면서 윷놀이 게임에 필요한 요구사항들을 좀 더 명확히 할 수 있었고, 그렇기 때문에 이후 class diagram, sequence diagram등을 조금 더 쉽게 작성할 수 있었다.

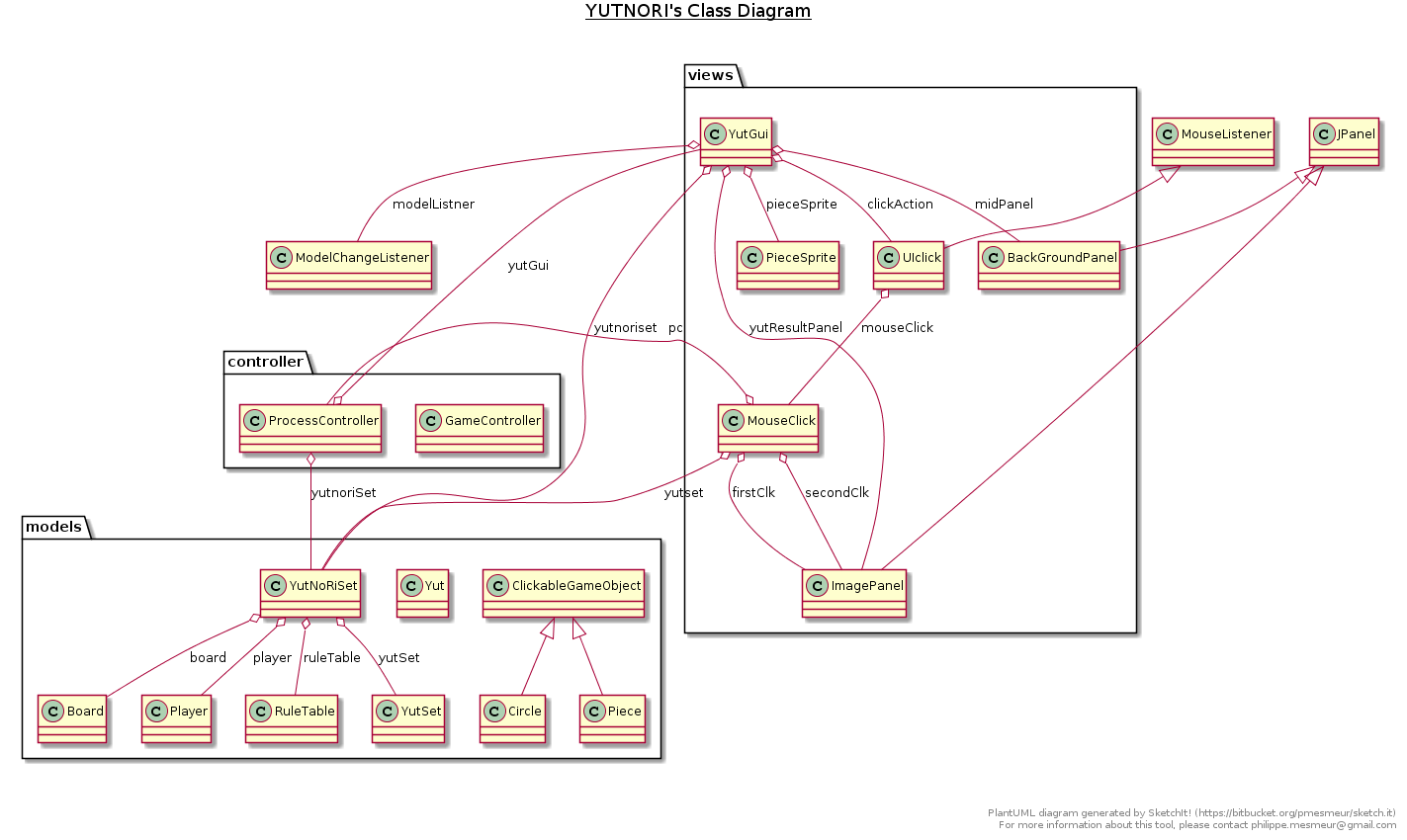
* + **Class Diagram**
* **기존**



MVC 패턴 적용을 좀더 용이하게 하기 위해 위와 같은 모델을 만들었다. View와 YutNoRi Game Controller는 YutNoRi Model을 통해 각 모델에 접근이 가능하다. View는 getter를 이용해 View에 필요한 모델 정보들을 가지고 온다. Controller는 move 등 YutNoRi Model에 정의된 상태 변환 함수들을 호출하여 모델의 데이터를 변화시킬 수 있다. Yut과 Circle, Player는 각 모델들에게 투명하여 각각의 구조가 변하더라도 크게 바뀌는 것이 없게 만들었다.

위 class diagram에서 YutNoRi Model은 다른 모델들의 접근을 통제하고, 각 모델들에 대한 getter와 setter를 재정의 하여 사용하였다. 하지만 구조가 Controller를 연상시켰고, 또한 개발 과정에서 위 다이어그램을 따라 개발하는데 있어 어려움이 있음을 깨닫고 이후 자료조사와 회의를 거쳐, 위 diagram을 수정하였다.

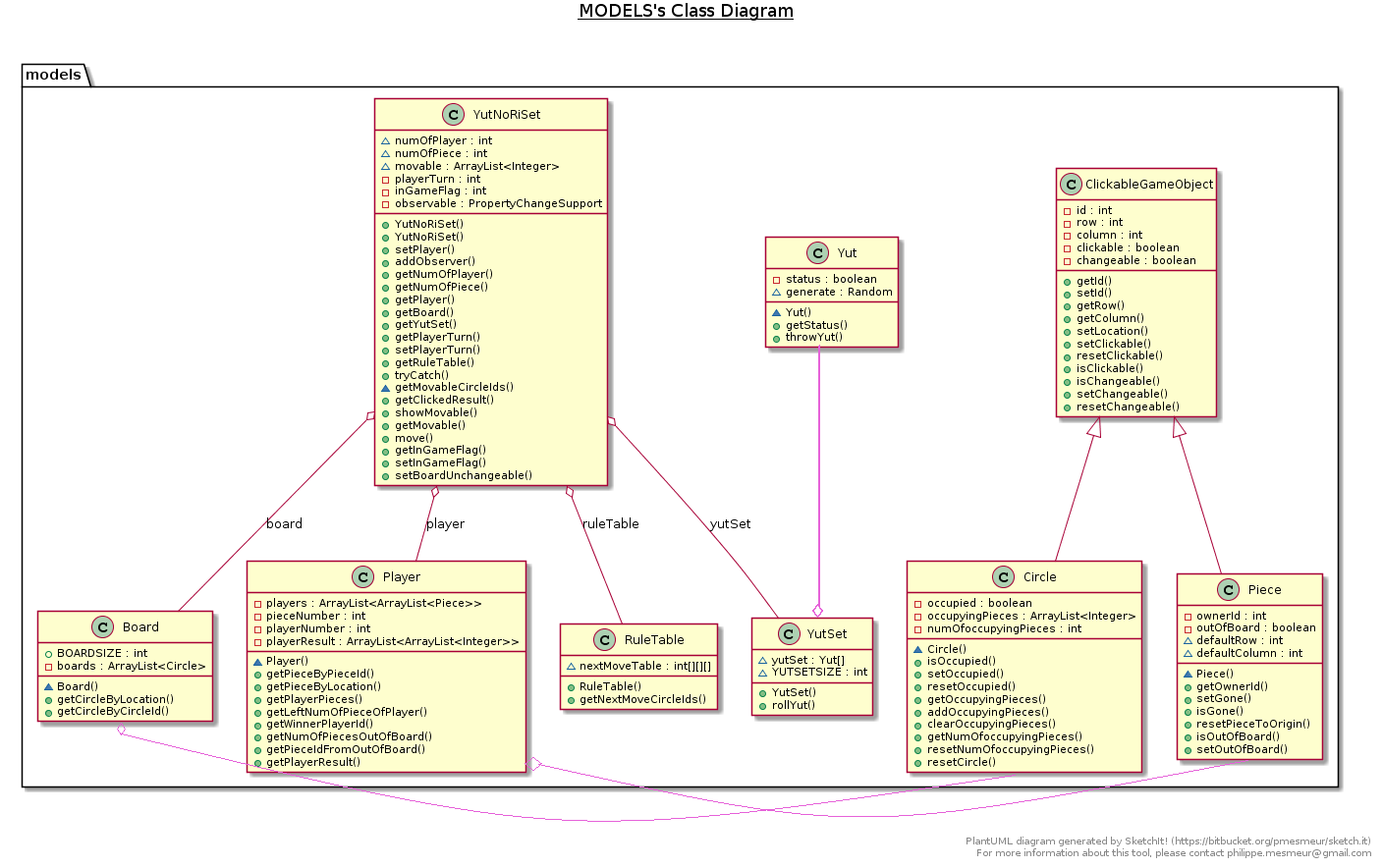
* **변화**
  + 전체 구조

****

본 프로젝트의 클래스들은 세 분류로 나뉘는데 MVC 패턴을 따르는 model, view, controller로 나뉘게 된다. 처음 프로그램을 시작하면 유저에게 View가 보이게 되고 View에서 이벤트를 입력 받으면 입력 값이 Controller에 전송되고 Controller는 그 결과를 이용해 model의 값을 갱신해준다. 갱신된 model은 view에게 자신이 갱신됨을 알리고 view는 model의 데이터에 접근해서 자신을 갱신시킨다. 이와 같이 MVC패턴을 최대한 활용 할 수 있도록 노력하였다.

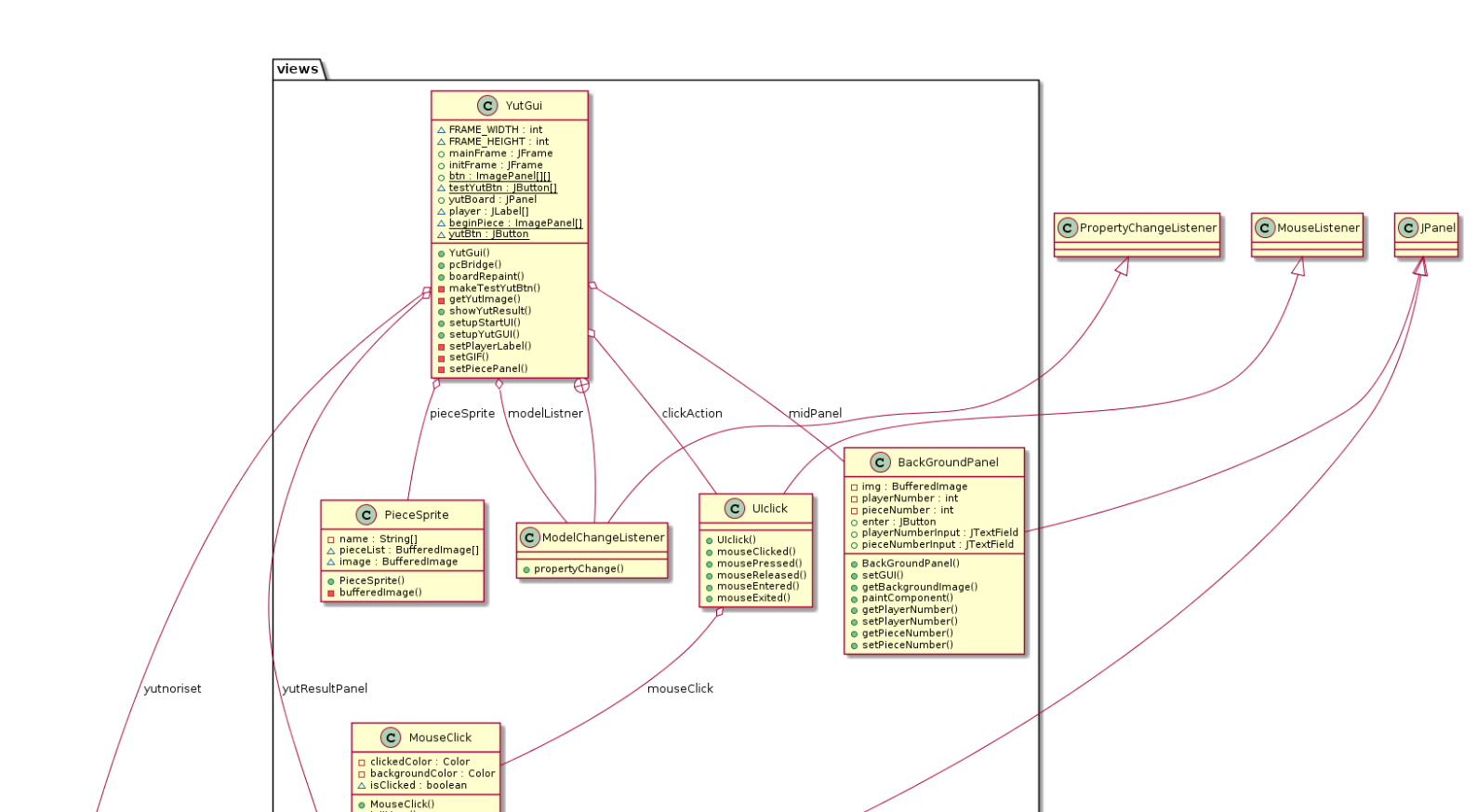
원래는 모델이 바뀌면 View에 이를 알리기 위해 java에 있는 observable을 사용하려 했으나, 본 팀이 사용한 java 11에서는 deprecated 되어 있는 기능이었다. 따라서 다른 방법을 찾아야 했는데 본 구조에서 보면 알 수 있듯, View에 modelChangeListener라는 propertyChangeListener를 사용하게 되었다. propertyChangeListener는 Model의 YutNoRiSet에도 추가되어 있으며, 모델에서 특정 change가 일어났을 때 modelChangeListener를 통해 change가 일어났음을 Listen할 수 있다. 이를 통해 Observable pattern을 활용한 MVC패턴을 구현하고자 노력하였다.

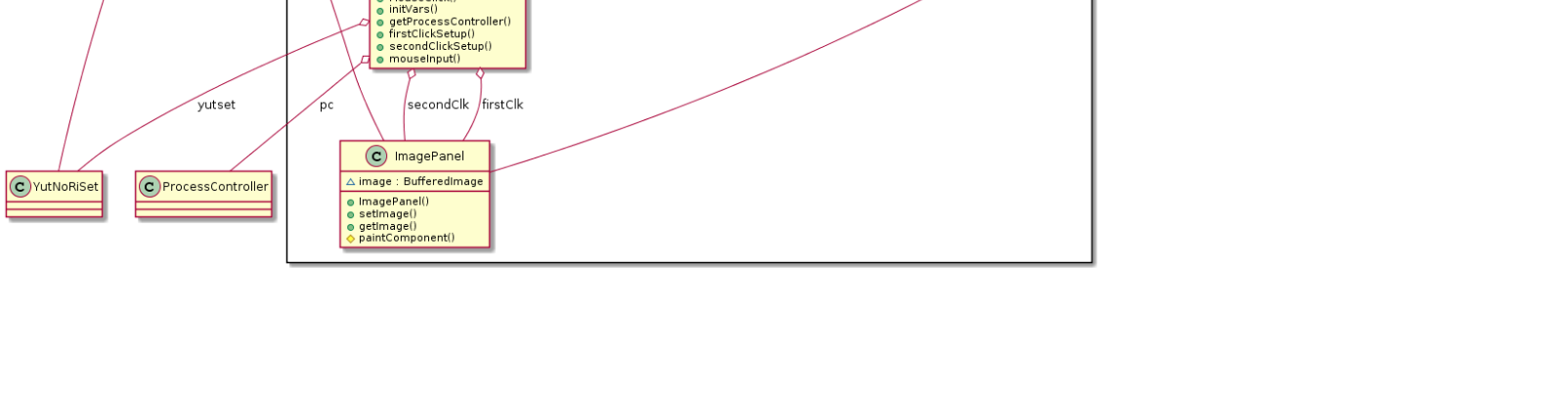
* + Model



다이어그램의 크기가 매우 커 위와 같이 좌우로 나누어 그려졌다. 왼쪽의 가장 큰 class는 나머지 모든 모델에 접근하기 위한 YutNoRiSet class다. 이 클래스는 4개의 Yut instance를 객체로 지닌 YutSet instance와 29개의 Circle instance를 가진 Board instance, 각 Circle에서 나온 결과들에 따라 갈 수 있는 Circle을 가지고 있는 RuleTable instance를 각 1개씩 가지고 있다. 또한 최소 2개에서 5개의 Piece instance를 가진 Player instance를 최소 2개에서 4개 가지고 있다. 이 YutNoRoSet을 통해 외부 클래스들은 기타 모델들에 접근이 가능하다.

* + View

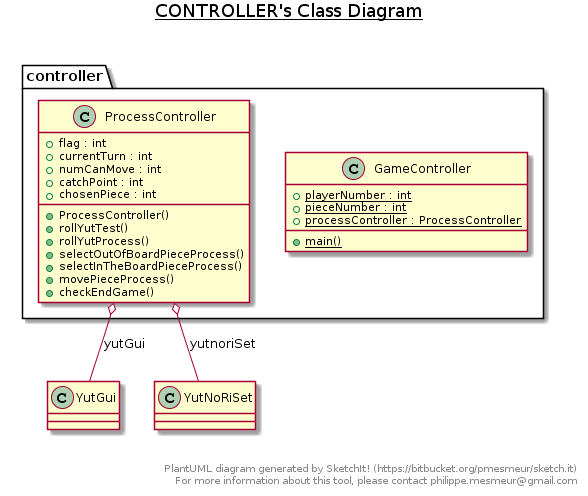




게임이 시작되면 가장 먼저 View가 보이게 되는데 이때 보이는 것은 Main Frame인 YutGui에 있는 BackGround Panel 이다. BackGround Panel은 처음에 플레이어 수, 플레이어당 말의 수를 입력 받아 버튼을 누르면 컨트롤러에 해당 정보를 넘겨주고 컨트롤러는 그 정보를 받아 Model에게 그에 맞는 플레이어, 말 객체를 생성하게 한다. 그리고 그 결과가 다시 YutGui에 그려지게 된다.

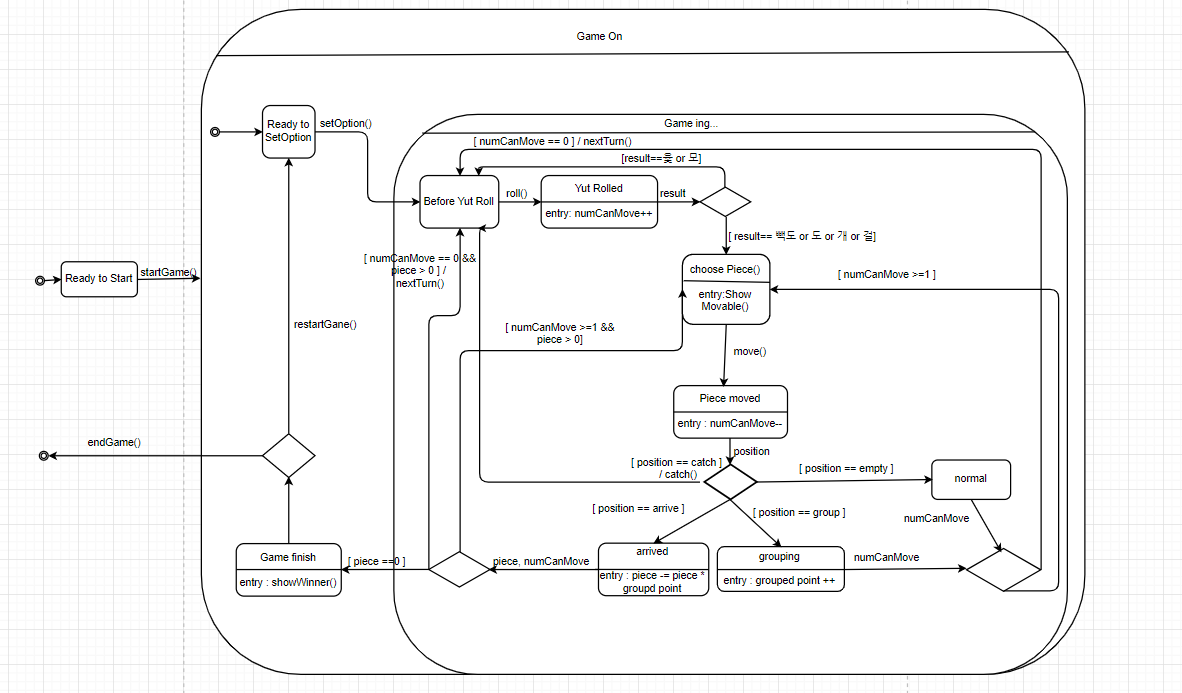
이후 YutGui에 있는 여러 버튼들(‘윷 던지기’, ‘테스트용 윷 던지기’, 각 말들과 윷놀이 판 등)이 눌리게 되면 Controller의 함수를 호출해 Controller의 함수가 Model을 업데이트 시켜주게 된다. 그 후 업데이트 된 정보가 View에 나타난다.

* + Controller

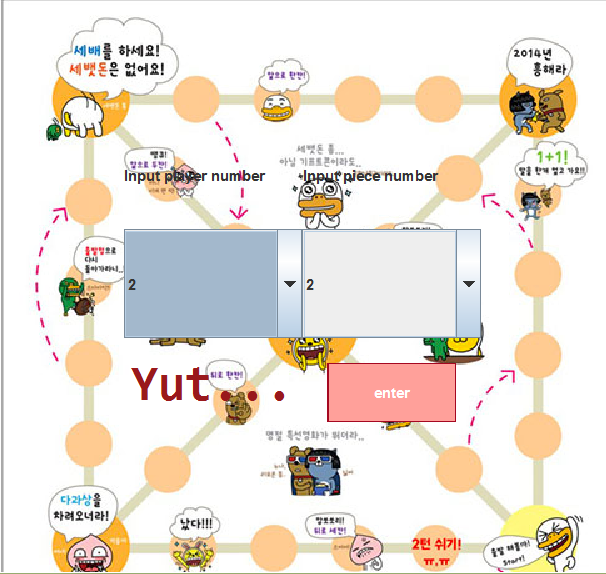


컨트롤러는의 클래스는 GameController와 ProcessContoller로 나눠지게 된다. GameController는 main을 포함하고 있으며 게임의 시작, 재 시작에 관여한다. 게임이 실행되고 플레이어 수와 말의 수를 입력 받아 그에 해당하는 ProcessController를 생성해준다. ProcessController는 게임 진행중에 플레이에 관한 logic을 담당한다. 예를들어 view에서 ‘윷 던지기’ 버튼이 눌리면 ProcessController에 있는 rollYutProcess() 함수가 불려 윷을 던지고 그 결과를 Model에 있는 Player에 저장 해준다. 이처럼 Controller는 view 에서 받은 입력에 따른 결과를 생성하고 Model에 저장해주는 역할을 한다.

* + **Software architecture + Design Model**
* **State chart diagram**

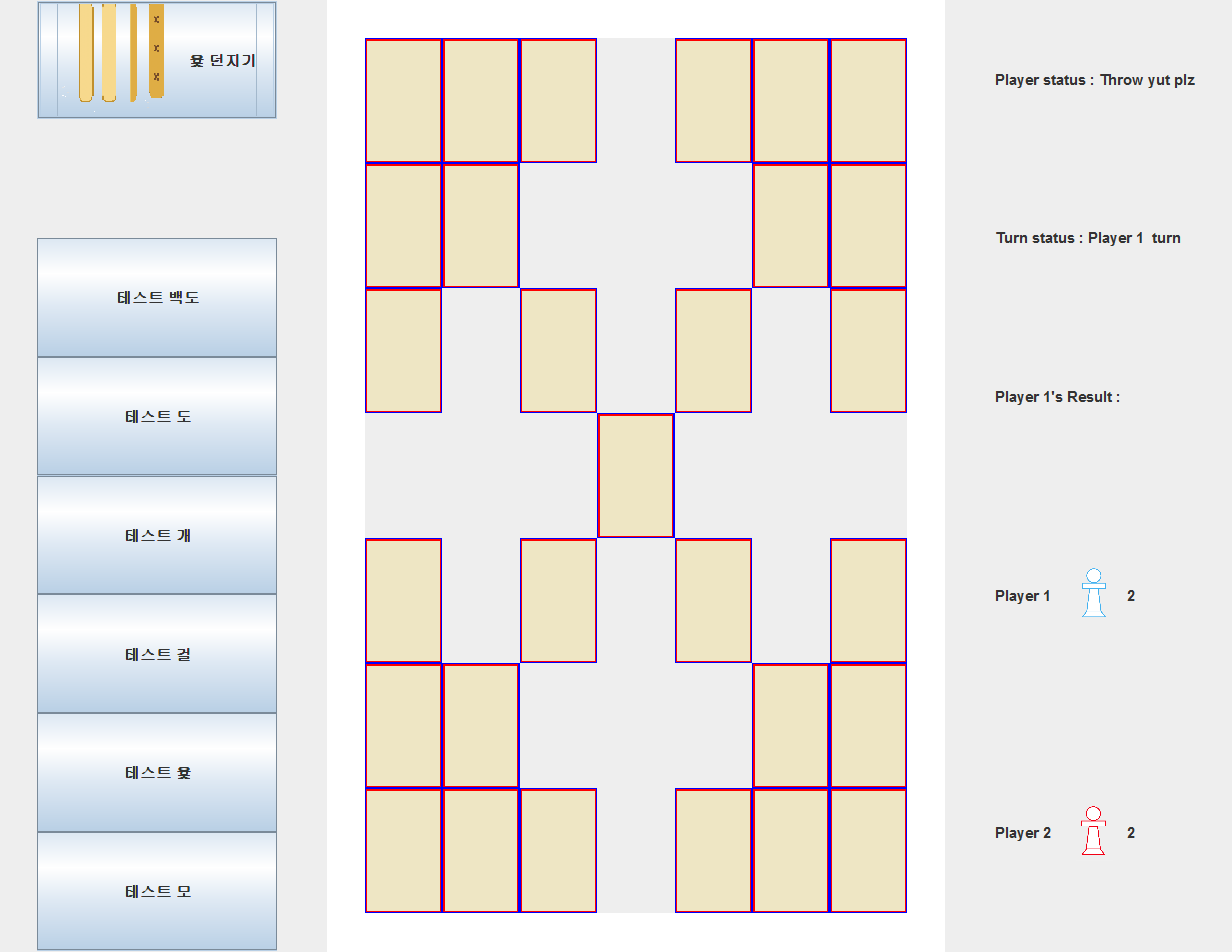
****

1. 먼저 게임이 켜지게 되면 Ready to SetOption state로 진입
2. 사용자에게 플레이어 수, 말 수를 입력(setOption)받으면 게임 진행 상태로 진입
3. 게임이 시작 되면 가장 먼저 Before Roll Yut State가 되는데 이때는 윷 던지기 버튼만 누를 수 있음
4. 윷을 던지면 결과에 따라 분기하게 되는데 윷의 결과가 윷이나 모가 나오면 결과를 저장하고 다시 윷을 던질 수 있음(Roll\_Yut())
5. 윷의 결과가 윷이나 모가 아니라면 다음 State로 진입함
6. ChosePiece state에서는 움직일 말을 선택함, 이 때 사용자는 윷놀이 판 위에 있는 말 혹은 아직 올려지지 않은 말 중 하나를 선택 할 수 있음
7. 말을 선택하면 미리 저장된 윷 던지기의 결과를 참고하여 말이 이동할 수 있는 범위를 나타냄(Show Movable())
8. 위에서 나타낸 판의 위치 중 하나를 선택하게 되면 말이 이동하게됨 (PieceMoved State)
9. 말이 이동했다면 몇 가지 조건을 통해 어떻게 분기할지를 결정하게 되는데
   1. 만약 상대 말을 잡았다면 바로 윷을 한 번 더 던짐
   2. 도착한 위치에 본인의 말이 있으면 말들을 Grouping한 후 움직일 수 있는 횟수가 남았다면 ChosePiece state로 이동, 그게 아니면 턴을 넘긴 후 BeforeYutRoll state로 이동
   3. 아무것도 없을 때도 남은 움직일 수 있는 횟수를 보고 ChosePiece state로 갈건지 BeforeYutRoll state로 갈건지 선택
   4. 만약 움직인 말이 도착했다면 먼저 게임이 종료되었는지를 판단하고(최근에 나간 말이 해당 플레이어의 마지막 말인지를 검사) 끝났다면 그대로 해당 게임을 종료시키고 게임을 그대로 종료시킬지 게임을 재 시작 할 지 결정.
10. 만약 게임이 재 시작 되었다면 Ready to Set Option state로 이동하여 다시 게임을 시작함
    * **Usage of Our Program**
11. **시작 화면**

****

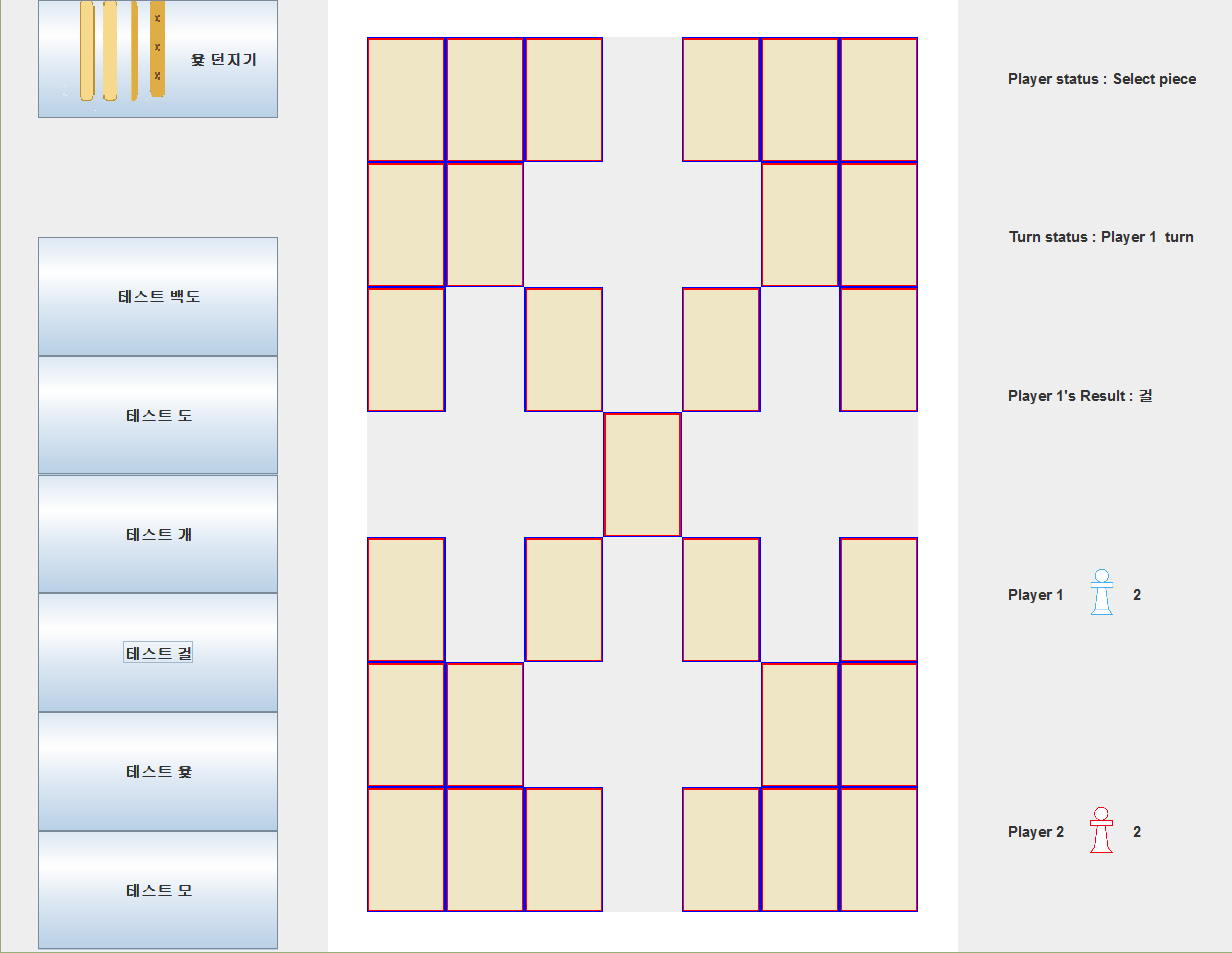
프로그램을 실행시키면 플레이어 수와 말 수를 입력할 수 있는 창이 나타난다.

1. **초기화면 (BRFORE ROLL YUT STATE)**

****

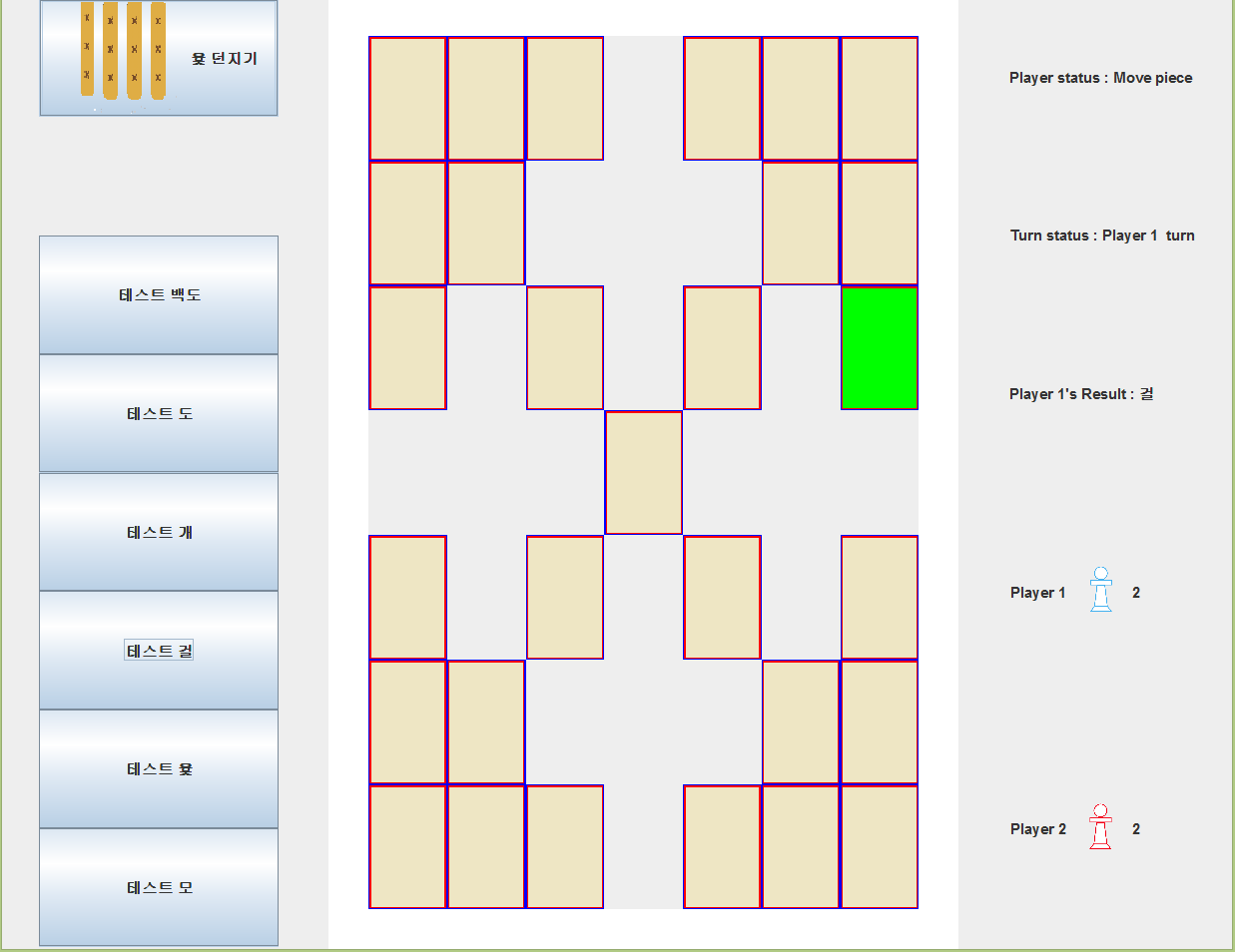
위의 화면은 윷놀이 게임을 처음 시작했을 때 화면이다. 패널의 1시 방향에 플레이어가 취해야 할 동작과 현재 차례인 플레이어를 나타내준다. 그 아래에는 각 플레이어의 말 중 아직 판에 올려지지 않은 말의 수를 나타내준다.

1. **윷을 던지고 난 후(YUT ROLLED / BEFORE SELECT PIECE STATE)**

****

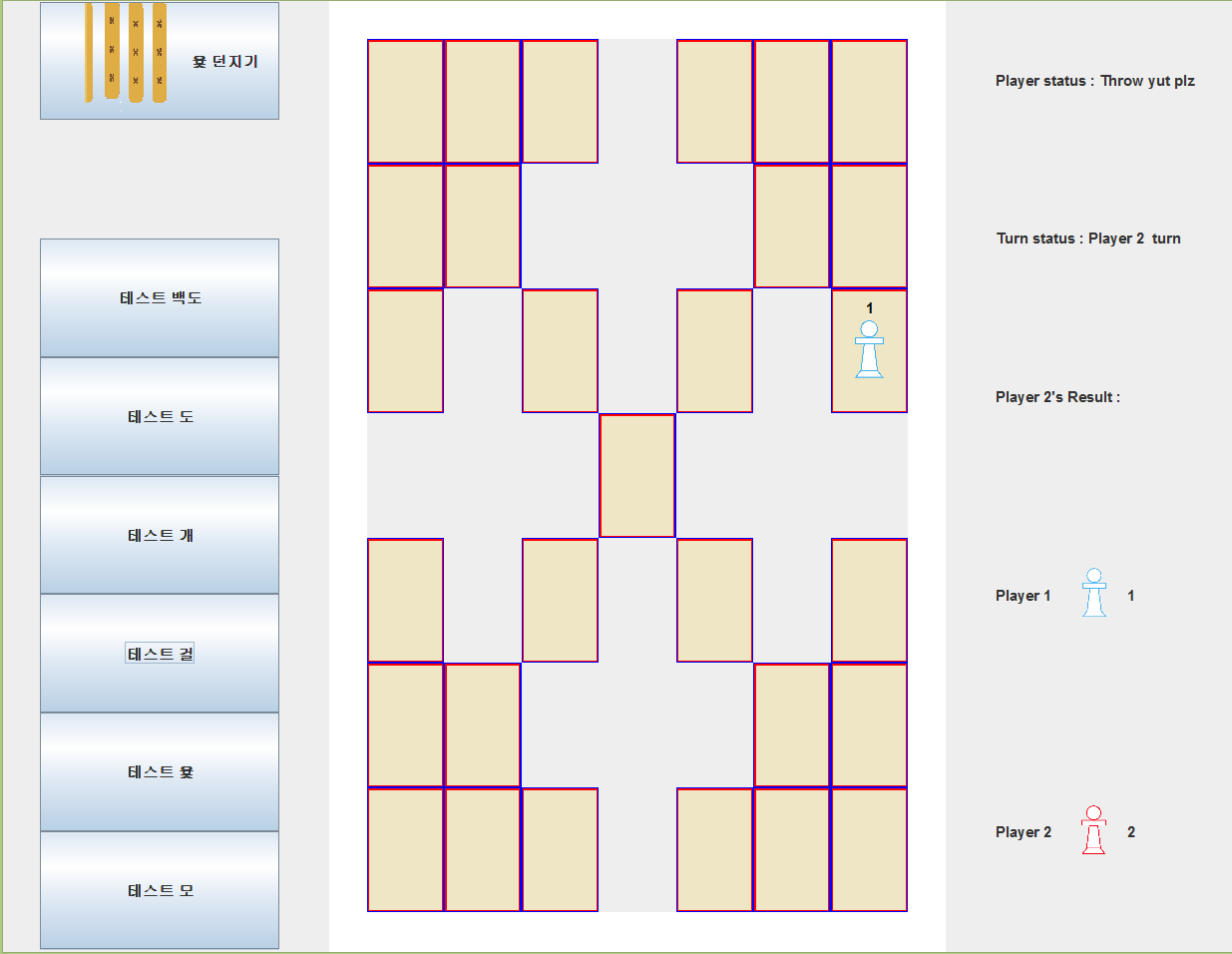
플레이어가 ‘윷 던지기’ 혹은 테스트용 윷을 던지면 패널의 상태가 바뀌는데 status는 select piece가 되고 윷을 던져 나온 결과를 나타내준다.

1. **움직일 말을 선택**

****

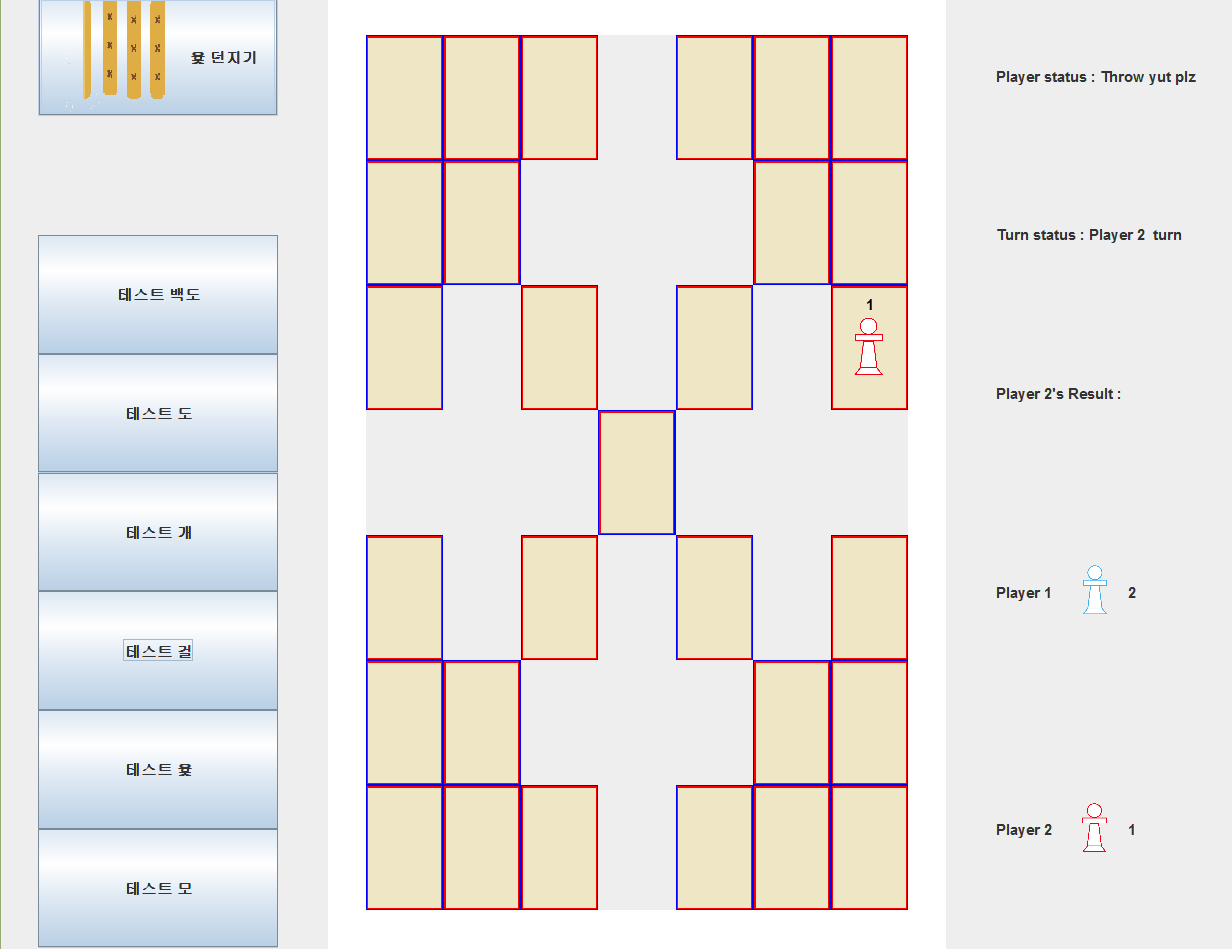
움직일 말을 선택 하고 나면 그 말이 갈 수 있는 위치를 초록색으로 표시해준다.

1. **말을 움직이고 난 후**

****

위의 화면은 Player 1의 말을 움직이고 난 후의 화면이다. 이전에 결과가 걸이 나와 걸만큼 움직였고(본 윷놀이 게임의 시작 위치는 맨 오른쪽 아래칸이다.) Status 패널을 Player 2의 것을 보여준다. Player1의 말 중 한 개가 판 위로 올라갔기 때문에 패널에 있는 말의 수는 줄여준다.

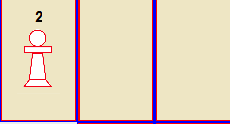
1. **상대 말을 잡았을 경우**

****

위의 시나리오는 Player1의 말이 걸 위치에 있었고, Player2가 걸이 나와 Player1의 말을 잡은 경우이다. 플레이어 턴은 여전히 Player2로 유지되고 Player2는 윷을 던질 수 있다.

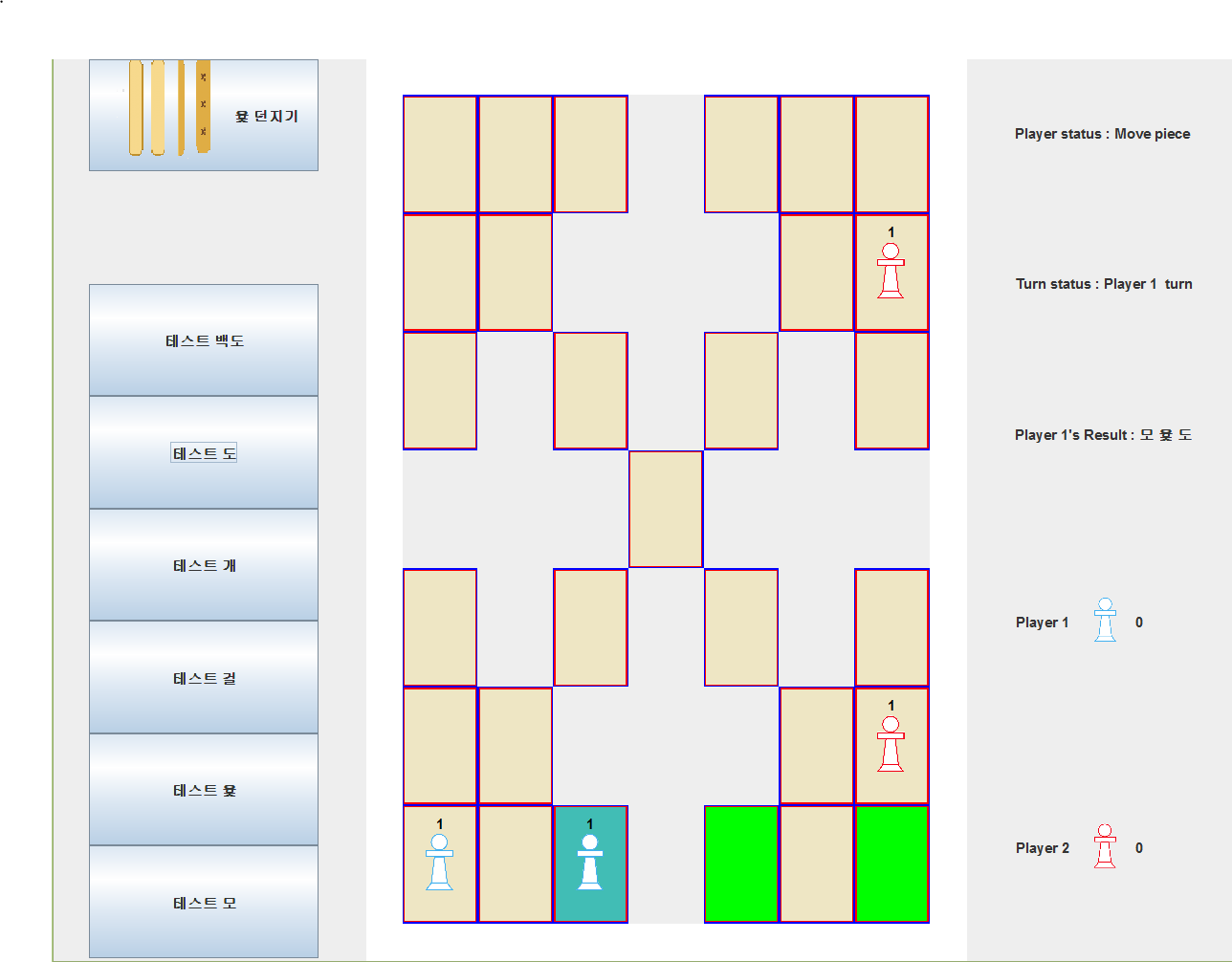
Player1의 말이 판 밖으로 나왔고 Player2의 말이 판 안으로 들어간 결과를 오른쪽 아래 패널을 참고하여 알 수 있다.

1. **Grouping 된 말이 있을 경우**

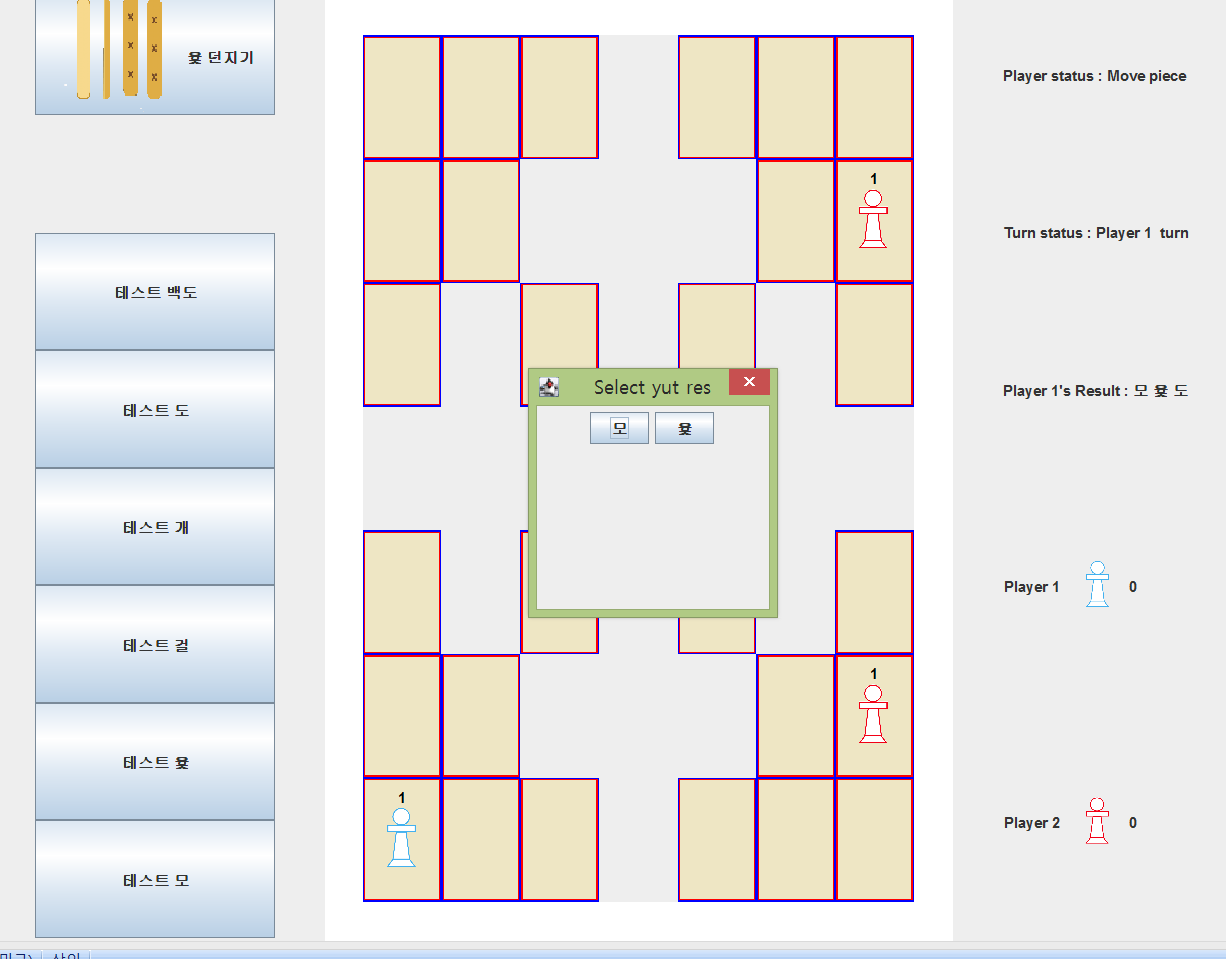
****

위의 사진과 같이 칸 하나에 몇 개의 말이 올려져 있는지 나타내준다

1. **도착하기 직전의 경우**

****

위의 시나리오는 Player1의 말이 도착 직전에 있고 결과로 모, 윷, 도를 가지고 있는 상태이다.



이 때 말을 도착 위치로 옮기게 되면 선택된 말은 ‘모’, ‘윷’ 으로는 도착 할 수 있으나 ‘도’로는 도착 할 수 없었기 때문에 ‘모’, ‘윷’만 나타내주게 되고 어떤 결과를 소비하여 도착을 할 지 결정 할 수 있다.

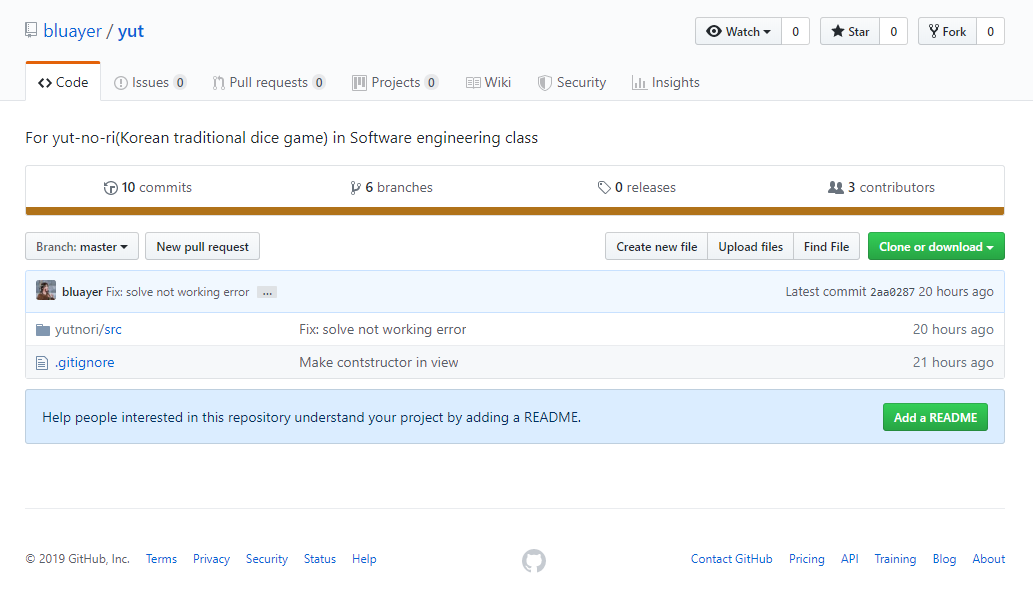
1. **게임이 끝났을 경우**

****

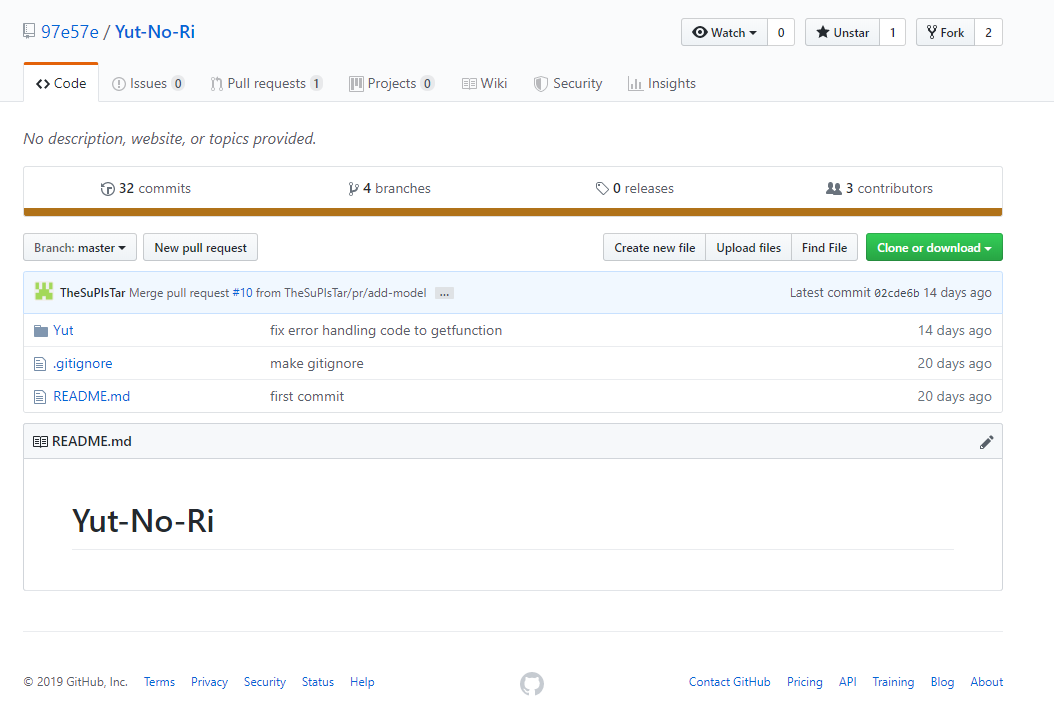
게임이 끝났다면 승자를 표시해주고 게임을 다시 시작 할 건지 끝낼 것인지 결정 할 수 있다.

1. **Project Management Report**
   * **Git repository**

저희는 repository 이전을 2 번 실시하여 2개의 repository기록을 가지고 있습니다. 각각 아래의 repository다.



처음/현재 repository <https://github.com/bluayer/yut>

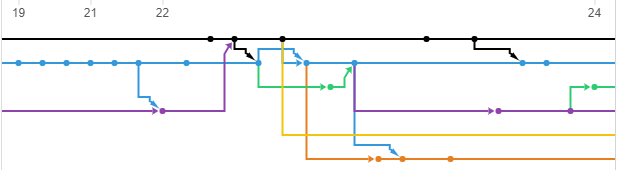


중간 repository. <https://github.com/97e57e/Yut-No-Ri>

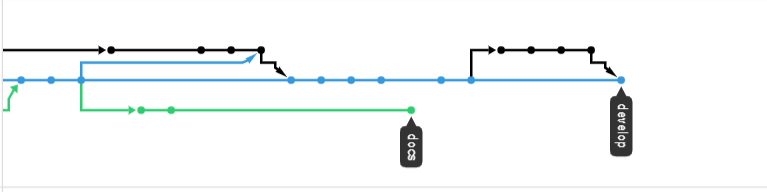
처음 우리들은 python web framework인 Django를 사용하여 mvc패턴에 맞는 게임을 개발하려 했다. 하지만 Django를 이용했을 때, Django의 모델에서 class를 이용하여 method를 생성하는 과정에서 개념상의 혼란이 와 repository를 이전하게 되었다.

두번째 repository에서는 java를 이용한 개발을 했다. JAVA를 이용한 개발 과정 중 모델과 컨트롤러의 구분, 뷰와 모델의 상호작용 등, 익숙하지 못해 발생하는 이슈들이 발생했다. 하지만 이미 class diagram과 sequence diagram이 어느 정도 정리된 이후였기 때문에 그래도 꽤나 개발과정은 순조로웠다.

하지만 git repository를 관리함에 있어, 팀원 모두가 repository의 관리에 능숙하지 못해 working branch들이 서로 얽히고 설켜 프로젝트의 구조를 그것을 보고 파악하지 못할 지경에 이르렀다. 고심 끝에 다시 repository를 처음 것으로 돌리고 해당 repository를 비우자고 이야기했다. 바꾸게 된 중간 repository의 working branch의 일부이다



다시 처음 repository에 돌아와서는 commit까지 신경 쓰며 관리했다. 물론 중간중간 실수 도 있긴 하였지만 꽤나 깔끔한 working branch가 생성되었습니다. 해당 branch는 다음과 같다.



저희는 처음 web frame work를 이용하여 개발을 하려고 했던 만큼, MVC적용에 대해 깊게 고민하였다. 특히 인터넷에 자료가 많은 .net frame work 등의 자료에서는 수직적 MVC모델이 등장하는 등 어떤 MVC모델을 적용하고, 어떻게 MVC가 상호작용해야 하는지에 대하여 많은 고민을 하였다. 그 결과 source package 구성을 다음과 같이 정리하고, 개발하여 각각이 서로의 함수를 호출하는 방식으로 개발을 마쳤다.

* + **Project Progress history**
* 05.03.2019

팀 협업을 위한 슬랙 구성 및 깃 repository setting

* 05.07.2019

일정 조율 및 개발 계획

-유즈케이스 작성 후 다이어 그램 그리기

-Django와 Pytest를 이용하여 개발하기로 협의

* 05.10.2019

슬랙을 이용한 회의로 대략적인 역할 분담.

* 05.12.2019 ~ 05.19.2019

지속적으로 학교에서 만나 각자 맡은 문서 조사 및 작성. 지속적 피드백으로 개선. 다이어그램 또한 리뷰를 통해 필요 요소들 점검.

* 05.20.2019

Django를 이용한 개발 시작

개발 과정 중 모델의 권한 관련 이슈 발생

슬랙 회의를 통해 Django에서 java로 개발 계획 수정

* 05.21.2019 ~ 05.26.2019

모델 및 뷰, 컨트롤러 스켈레톤 코드 추가.

모델 리펙토링 완료.

* 06.06.2019 ~ 06.07.2019

깃허브 워킹트리 관리 부족으로 인한 권한 혼란.

그에 따른 레포지토리 이전. 이후 지속적으로 개발 작업.