**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**

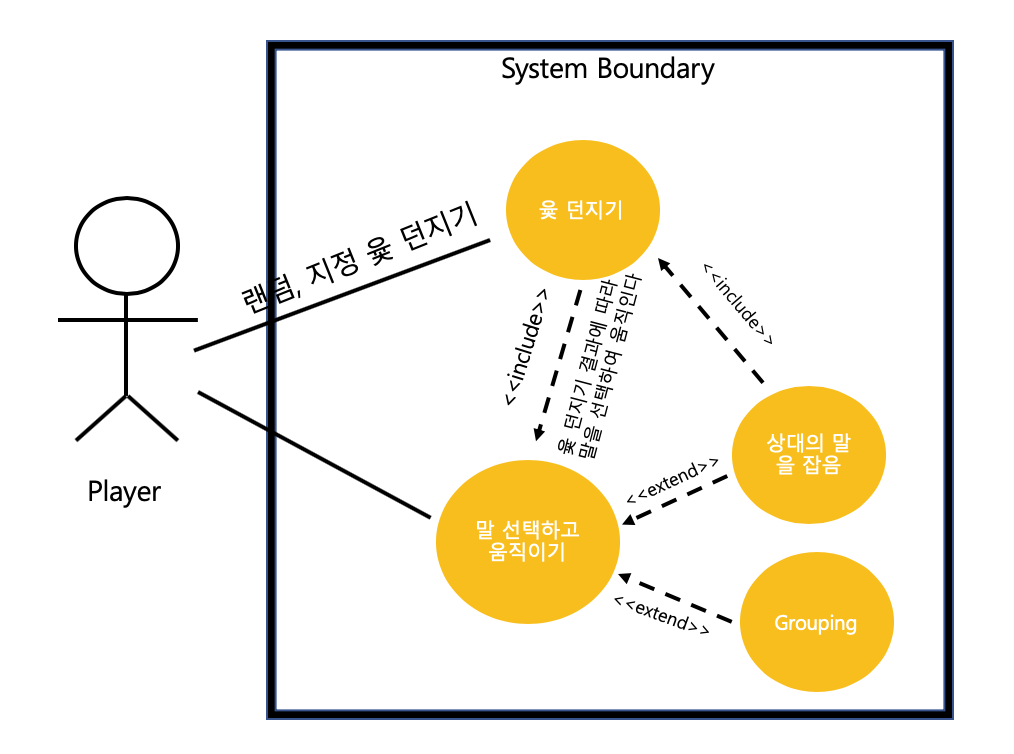
- 사용자는 게임이 시작하기 전 게임에 참여할 플레이어 수와 각 플레이어당 가질 수 있는 말의 수를 설정하고 게임을 시작 한다.

- 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

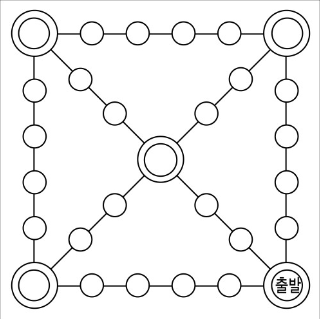
**2. Level :** user goal

**3. Primary Actor :** Player

**4. Stakeholders and Interests:**

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

**5. 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)**

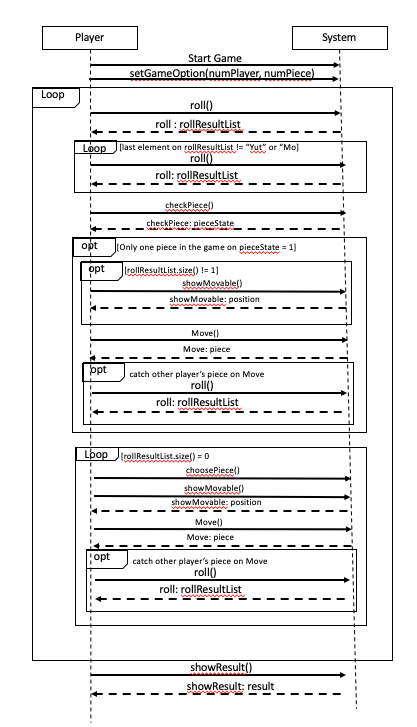
**\*a. At any time, 판 위에 말이 없을 때 ‘빽도’가 나온 경우**

- 그냥 ‘도’로 적용한다.

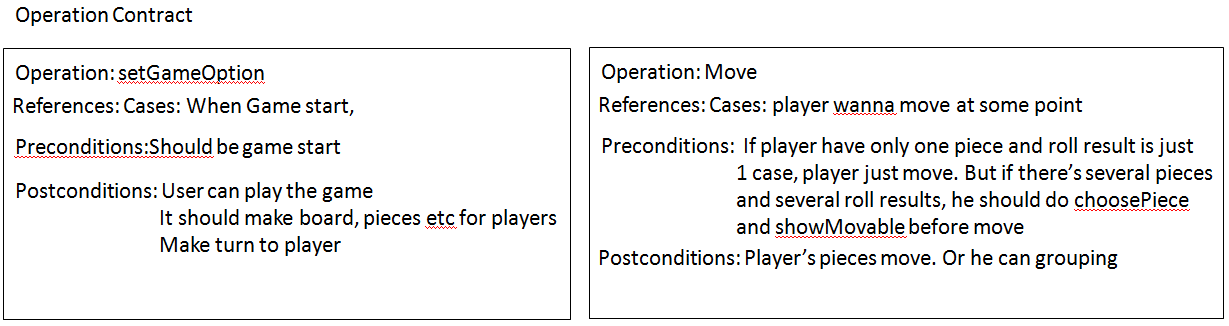
**\*b. At playing time, 모나 윷이 전체 말을 goal로 옮길 정도로 많이 나온 경우**.

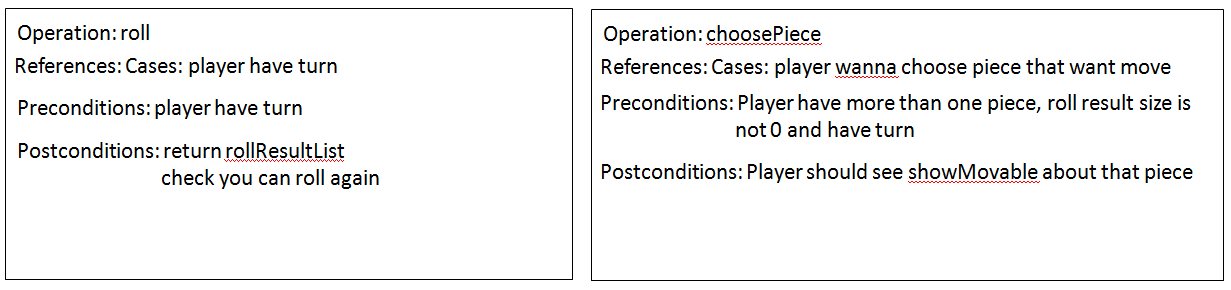
- 해당 플레이어의 승리로 처리하고 6)단계로 점프한다.

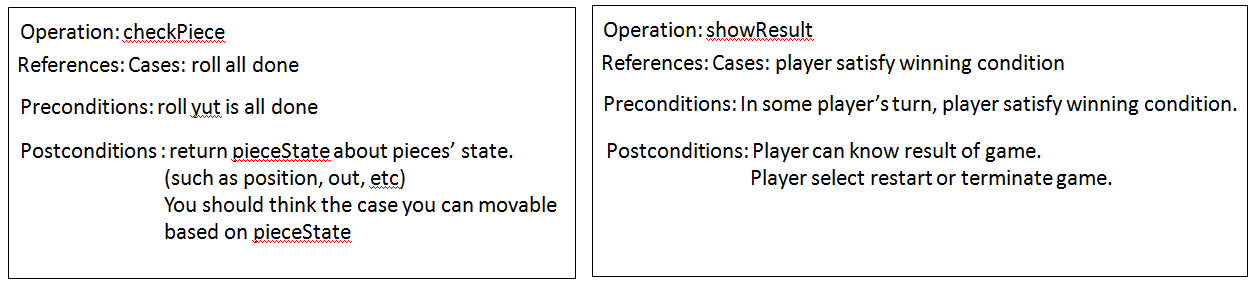
**◼︎ System Sequence Diagram**

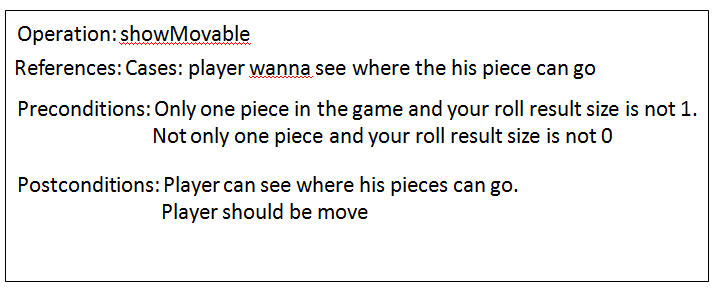
****

**◼︎ Operation Contracts**



****

****

****

위는 usecase diagram을 통해 추출한 operation contract들이다.

SetGameOption은 게임을 시작하기전, 플레이어 수와 피스 수를 받는 함수이다. Move는 선택된 piece를 circle에 옮기는 함수이다.

Roll은 yut들을 던져 결과를 얻는 함수이다.

ChoosePiece는 게임에서 선택가능한 piece들을 선택하면 그 피스의 아이디 혹은 그 피스의 위치를 반환하는 함수이다.

Show result는 게임의 player 한명이 종료 조건을 충족하였을 때 해당 플레이어가 승리했음을 보여주고 재시작 혹은 종료를 하는 화면을 보여주는 함수이다.

Show movable 함수는 특정 piece의 위치에서 roll의 결과에 대해 갈 수 있는 위치의 circle의 상태를 바꾸는 함수이다.

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.

7. 플레이어는 자기 턴에 윷을 던지고 말이 이동 한 후, 다른 플레이어의 말을 잡았다면 한번 더 윷을 던질 수 있다.

8. (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.

9. 한 플레이어가 모든 행동을 마치면, 차례가 다음 플레이어로 넘기게 된다.

10. 말들이 윷판의 각 모서리 칸 혹은 중앙 칸에 들어가면, 다음 이동 시에 어디로 이동할지 결정 할 수 있다.

11. (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.

12. 한 플레이어의 모든 말들이 윷놀이 판에서 나가면 그 플레이어로 승리자가 결정되고 게임이 끝나게 된다.

13. 게임이 끝난다면 게임을 재 시작할지 그대로 끝낼지 결정할 수 있다.

**● NFR**

1. OOAD 기법을 사용하기 위해 객체지향 언어인 Java을 사용해 개발한다.

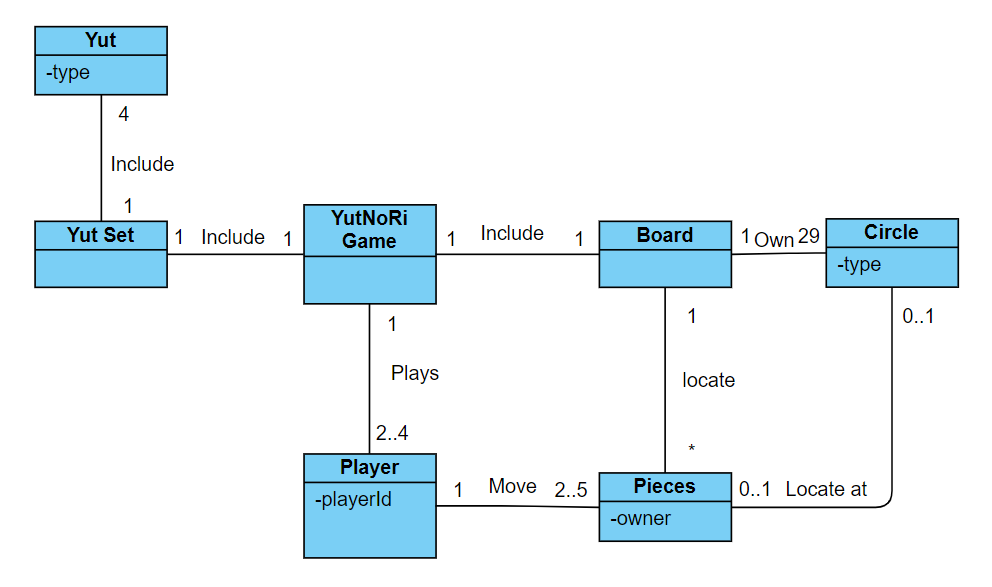
2. Java를 이용해 Model, Controller를 개발하고 Swing 라이브러리를 이용해 유저에게 보일 화면을 만든다.

3. Java를 사용하여 유닛테스트를 하기 위해 JUnit을 사용한다.

4. 협업 을 통해 github을 이용하고 모든 활동(source & document 등)을 github에 기록한다.

5. Use case 모델 및 분석, 설계, 구현과 관련된 프로젝트 문서들을 작성한다.

**◼︎ Domain Model**

****

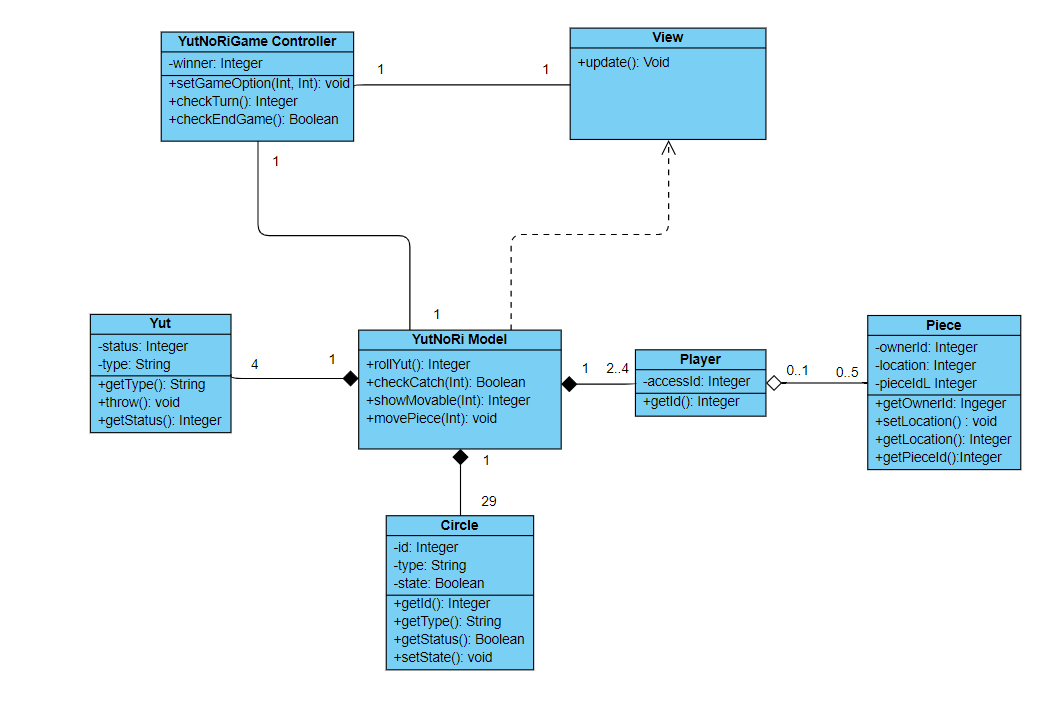
Domain Model에는 YutNoRi Game이라는 컨트롤러와 Player, Yut Set, Board등 윷놀이 게임에 필요한 모델들이 존재 한다. YutNoRi Game은 각 모델들의 정보를 조회, 갱신하여 게임을 진행한다.

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**

**- 프로젝트 구현 진행 전**

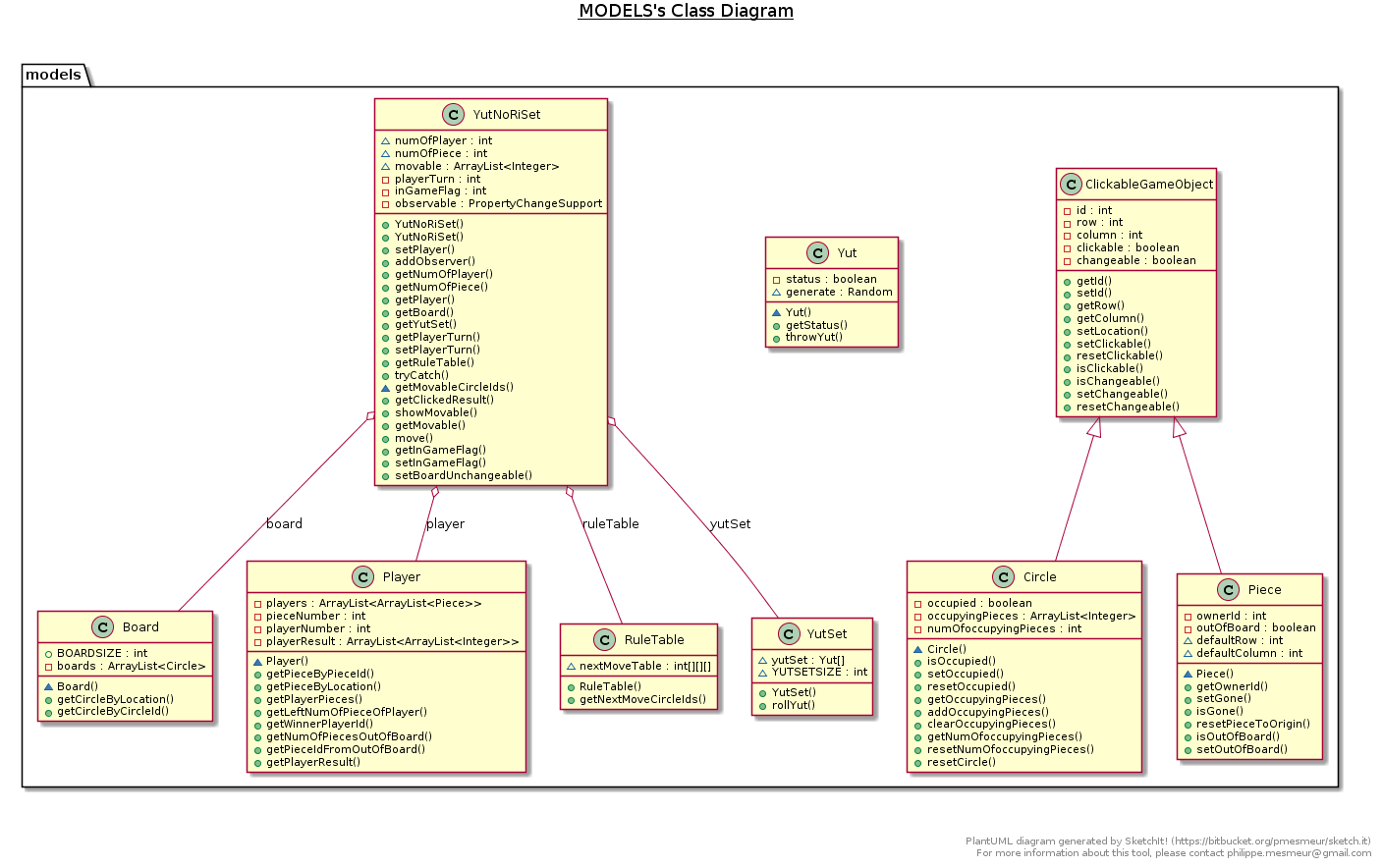


MVC 패턴 적용을 좀더 용이하게 하기 위해 위와 같은 모델을 만들었다. View와 YutNoRi Game Controller는 YutNoRi Model을 통해 각 모델에 접근이 가능하다. View는 getter를 이용해 View에 필요한 모델 정보들을 가지고 온다. Controller는 move 등 YutNoRi Model에 정의된 상태 변환 함수들을 호출하여 모델의 데이터를 변화시킬 수 있다. Yut과 Circle, Player는 각 모델들에게 투명하여 각각의 구조가 변하더라도 크게 바뀌는 것이 없게 만들었다.

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.

**- 프로젝트 구현 후**

**◼︎ model**

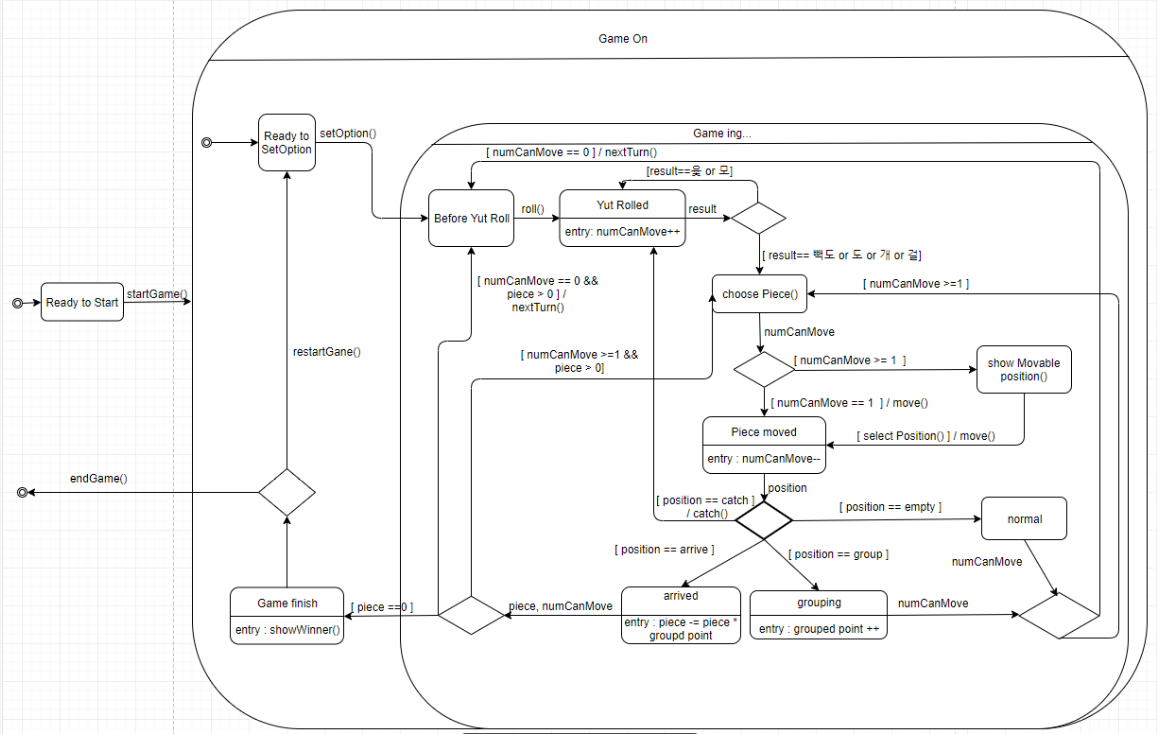


다이어그램의 크기가 매우 커 위와 같이 좌우로 나누어 그려졌다. 왼쪽의 가장 큰 class는 나머지 모든 모델에 접근하기 위한 YutNoRiSet class다. 이 클래스는 4개의 Yut instance를 객체로 지닌 YutSet instance와 29개의 Circle instance를 가진 Board instance, 각 Circle에서 나온 결과들에 따라 갈 수 있는 Circle을 가지고 있는 RuleTable instance를 각 1개씩 가지고 있다. 또한 최소 2개에서 5개의 Piece instance를 가진 Player instance를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

컨트롤러의 클래스는 GameController와 ProcessContoller로 나눠지게 된다. GameController는

**◼︎ 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 이동하게 됨 (PieceMoved State)

9. 말이 이동했다면 몇 가지 조건을 통해 어떻게 분기할지를 결정하게 되는데

9.1. 만약 상대 말을 잡았다면 바로 윷을 한 번 더 던짐

9.2. 도착한 위치에 본인의 말이 있으면 말들을 Grouping한 후 움직일 수 있는 횟수가 남았다면 ChosePiece state로 이동, 그게 아니면 턴을 넘긴 후 BeforeYutRoll state로 이동

9.3. 아무것도 없을 때도 남은 움직일 수 있는 횟수를 보고 ChosePiece state로 갈지 BeforeYutRoll state로 갈지 선택

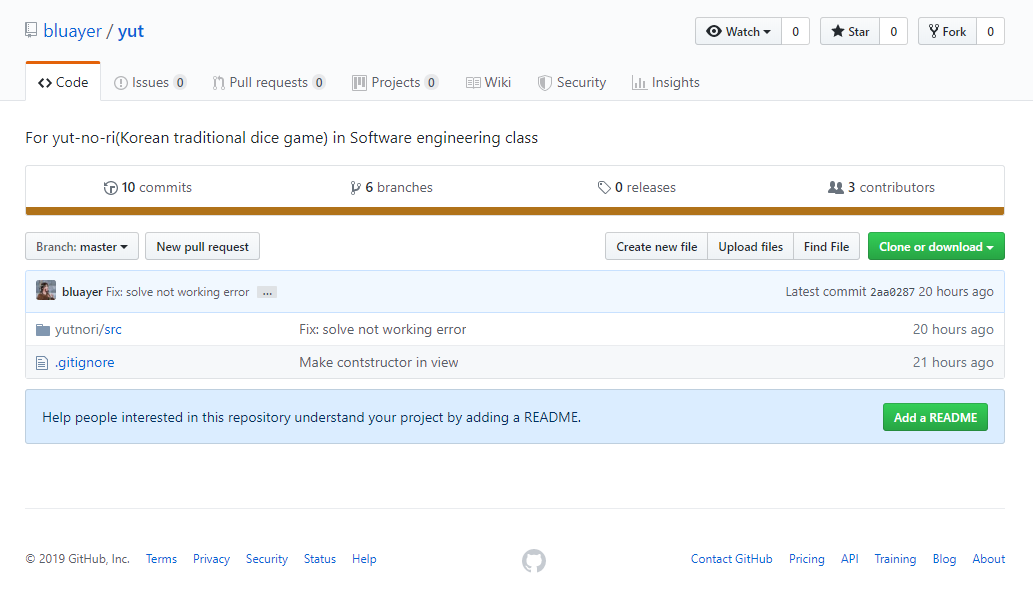
9.4. 만약 움직인 말이 도착했다면 먼저 게임이 종료 여부를 판단하고(최근에 나간 말이 해당 플레이어의 마지막 말인지를 검사) 끝났다면 그대로 해당 게임을 종료시키고 게임을 그대로 종료 할지 게임을 재 시작 할 지 결정.

10. 만약 게임이 재 시작 되었다면 Ready to Set Option state로 이동하여 다시 게임을 시작함

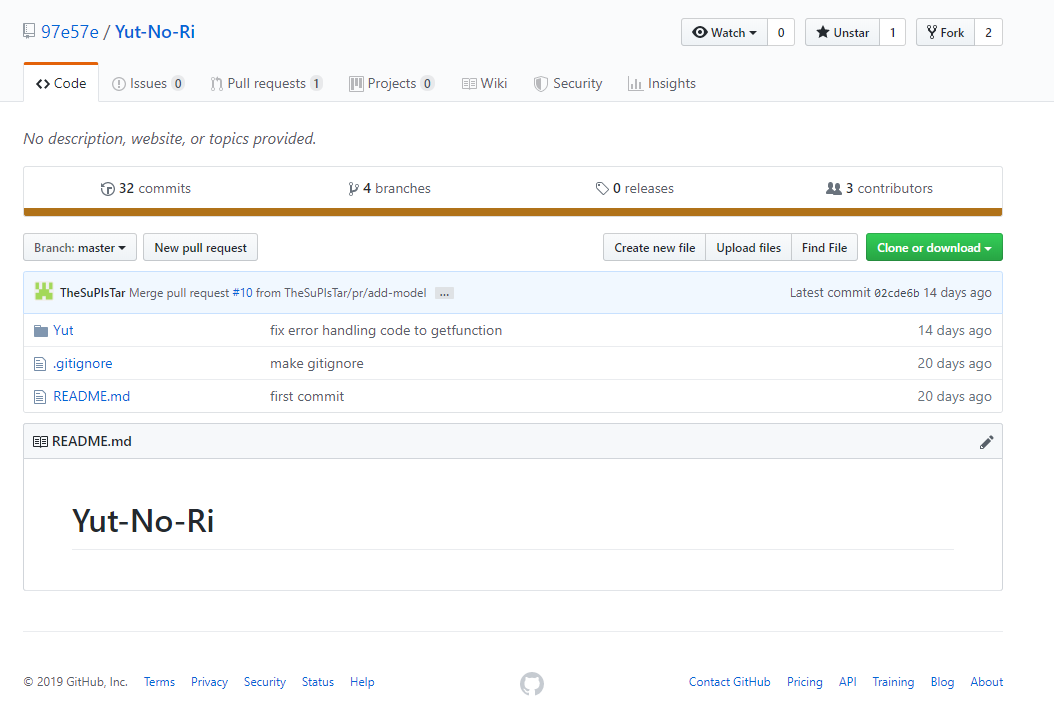
1. **Project Management Report**

**◼︎ Git repository**

저희는 repository 이전을 1 번 실시하여 2개의 repository기록을 가지고 있습니다. 각각 아래의 repository다.



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

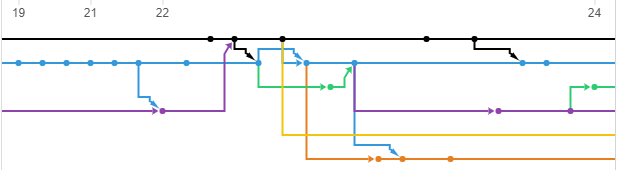


중간 repository. <https://github.com/97e57e/Yut-No-Ri>

처음에는 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.

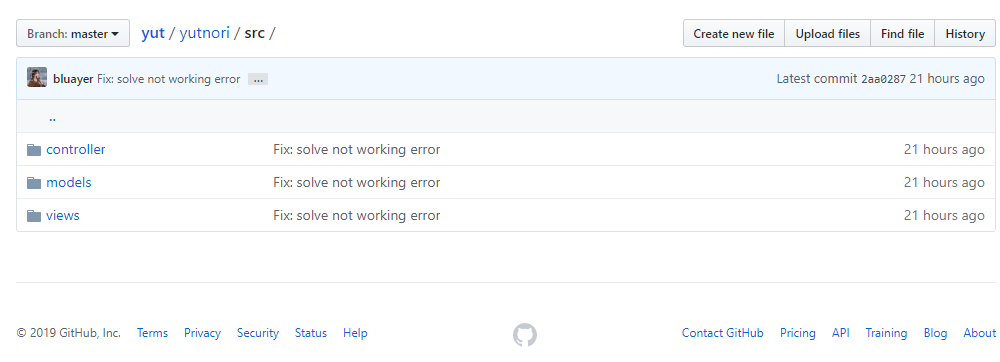
두번째 repository에서는 java를 이용한 개발을 했다. JAVA를 이용한 개발 과정 중 모델과 컨트롤러의 구분, 뷰와 모델의 상호작용 등, 익숙하지 못해 발생하는 이슈들이 발생했다. 하지만 이미 class diagram과 sequence diagram이 어느 정도 정리된 이후였기 때문에 그래도 꽤나 개발과정은 순조로웠다.

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.

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**

**본 프로젝트 진행은 2019년에 진행되었습니다. 진행 사항은 다음과 같습니다.**

**• 5월 3일**

팀 협업을 위한 슬랙 구성 및 깃 repository setting

**• 5월 7일 (1주차)**

일정 조율 및 개발 계획

- 유즈케이스 작성 후 다이어 그램 그리기

- Django와 Pytest를 이용하여 개발하기로 협의

**• 5월 10일(1주차)**

슬랙을 이용한 회의로 대략적인 역할 분담.

**• 05.12일 ~ 05.19일 (2주차)**

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

**• 5월 20일(3주차)**

Django를 이용한 개발 시작

개발 과정 중 모델의 권한 관련 이슈 발생

슬랙 회의를 통해 Django에서 java로 개발 계획 전환

**• 5월 21일 ~ 5월 26일 (3주차)**

모델 및 뷰, 컨트롤러 스켈레톤 코드 추가.

모델 리팩토링 완료.

**• 6월 6일 ~ 6월 7일 (4주차)**

깃허브 워킹트리 관리 부족으로 인한 권한 혼란.

그에 따른 레포지토리 이전. 이후 지속적으로 개발 작업.

**• 6월 7일 ~ 6월 8일 (4주차)**

보고서 작성 및 프로젝트 구현 마무리.