**Software Engineering**

* **Term Project : 윷놀이 게임 -**

‘



**2019.06.08**

**20174507김다빈**

**20174470전희수**

**20161344허정우**

**20163453현도연**

* **목차 및 작성내용**

|  |  |
| --- | --- |
| **목차** | **작성 내용** |
| 1. 팀원 및 역할 분배 | 1. 팀 구성 2. 역할 분배 |
| 1. 비전 | 1. Introduction to the system 2. Main features |
| 1. Use Case Model | 1. Use case Diagram 2. Use case Descriptions 3. System Sequence Diagram 4. Operation Contracts |
| 1. 설계 및 구현 | 1. 설계 2. 구현 3. 보완점 |
| 1. 사용법 및 테스트 결과 | 1. 사용법 2. 테스트 결과(스크린샷) |
| 1. Project Management | 1. github repository주소 2. Progress History |

1. **팀원 및 역할 분배**
2. **팀 구성**

해당 팀은 김다빈, 전희수, 허정우, 현도연 4명의 팀원으로 구성되어 있다.

1. **역할 분배**

김다빈 –

전희수 –

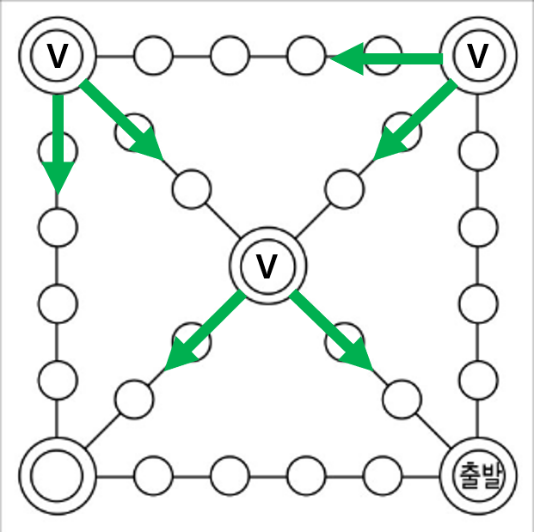
허정우 –

현도연 -

1. **비전**
2. **Introduction to the System**

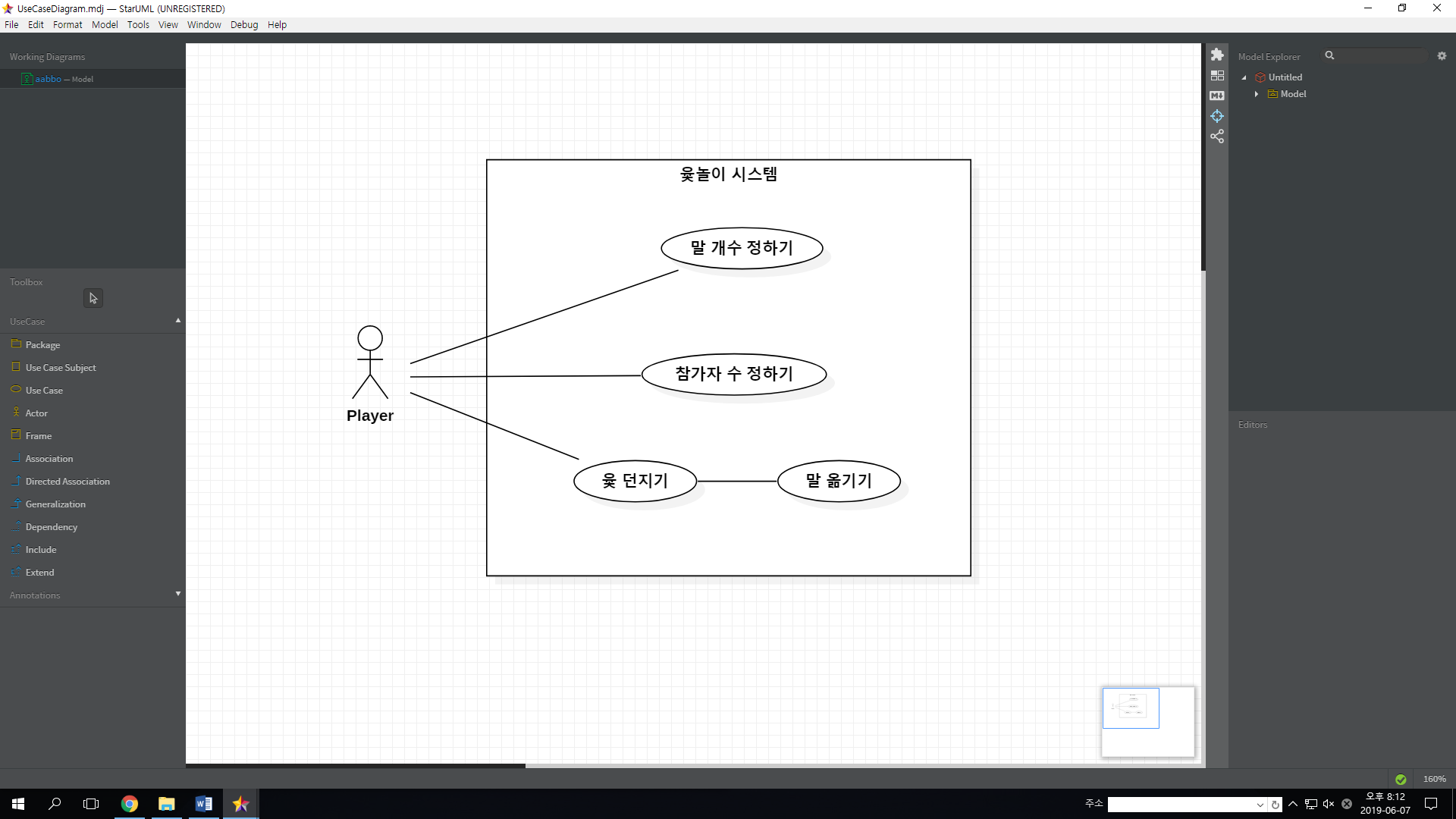
윷놀이 게임이란 ‘윷판’과 4개의 ‘윷’을 이용한 일종의 보드게임으로 한국의 전통놀이이다. 4개의 ‘윷’을 던지면 던져진 모양에 따라 ‘도, 개, 걸, 윷, 모, 빽도’와 같이 총 5가지의 결과가 나오는데 이에 따라 말을 움직이며 한 팀에 배정된 모든 말을 도착지점까지 도달시키면 승자가 된다. 사용자는 본 시스템을 이용하여 ‘윷판’과 ‘윷’이 없더라도 윷놀이 게임을 즐길 수 있다.

1. **Main Features**

* 본 시스템은 Qt 라이브러리를 이용해 제작된다.
* 사용자는 참여자의 인원수(2~4)와 게임 말 개수(2~5)를 슬라이드바를 이용해 조절하여 선택할 수 있다.
* 일정 순서에 따라 참여자는 돌아가면서 윷을 던질 수 있다.
* 참여자는 랜덤 혹은 지정버튼을 클릭하여 윷의 결과를 얻는다.
* ‘도-개-걸-윷-모’ 가 나오면 사용자는 각각 1,2,3,4,5칸씩 이동할 수 있다.
* 사용자가 자신이 움직이려는 말을 클릭하면 그 말이 이동할 수 있는 위치가 표시되고 그 중에 하나를 선택하여 말을 움직인다.
* ‘윷’ 또는 ‘모’가 나올 경우 윷을 한번 더 던질 수 있으며 이 경우 윷을 두 번 던진 후에 [이동할 말 선택 – 이동 – 이동할 말 선택 – 이동]의 순서로 진행되며 윷을 던져서 나온 결과의 순서대로 말을 이동한다.
* 말이 ‘윷판’의 꼭지점(<Figure1>의 ‘V’ 표시가 된 위치)에 정확히 위치했을 때만 갈림길로 들어갈 수 있는 기회가 생긴다.
*  ‘윷판’의 꼭지점(그림 넣고 해당 위치 표시 등으로 수정) 위치했을 때 ‘빽도’가 나올 같은 칸에 같은 팀의 말이 동시에 위치하게 되면 말을 업을 수 있고 다른 팀의 말이 있으면 다른 말을 잡을 수 있다.
* 참여자 중에 가장 먼저 모든 말을 골인지점을 거쳐 내보내는 팀이 게임에 승리하며 승자를 표시하는 화면이 나타난다.
* 같은 말을 이동시키는데 게임 시작 후 처음부터 [도 – 빽도]의 순서로 이동하여 출발지점에 되돌아오는 경우는 말이 골인지점으로 온 것이 아니다. 말이 윷판을 한 바퀴 돌아 출발 지점으로 되돌아오면 말이 골인지점으로 이동한 것이다.

<Figure 1>

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



1. **Use Case Description**

* Scope : 윷놀이 application
* Level : user goal
* Primary Actor – Player
* Stakeholders and Interests
* Preconditions: Player is identified and authenticated.
* Success Guarantee(or Postconditions)
* 사용자는 해당 판의 결과(승자)를 알 수 있다.
* 윷을 던지면 해당 수만큼 말이 정확히 이동한다.
* 한 플레이어의 순서가 지날 때마다 던진 윷의 결과가 바로 보드에 적용된다.
* 한 플레이어의 순서가 지날 때마다 게임이 종료되는지 확인된다.
* 랜덤 윷을 선택한 경우 랜덤하게 윷의 결과가 나온다.
* Main Success Scenario

1. 플레이어가 총 몇 명의 플레이어와 참여할지, 한 팀당 몇 개의 말로 진행할지 결정한다.

2.     게임을 시작(시작 버튼을 클릭)한다.

3.     플레이어는 자기 차례에 랜덤 윷 버튼(?)을 클릭해 윷을 던진다.

4. 이동 가능한 말 중 플레이어가 이동할 말을 클릭한다.

5.     클릭한 말이 이동 가능한 위치가 보드에 하이라이팅 된다.

6.     하이라이팅 된 위치 중 이동할 위치를 클릭한다.

7.     말이 해당 위치로 이동하여 보드에 나타난다.

3-7의 과정을 게임이 종료될 때까지 반복한다.

8.     한 플레이어의 모든 말이 먼저 다 돌아온 경우(모든 말을 보드 밖으로 뺀 경우) 해당 플레이어가 승리한다.

9.     승리한 플레이어의 정보를 팝업창으로 띄운다.

10.     팝업창에서 종료 버튼을 누르면 게임을 종료한다.

* Extensions:

**\*a. 윷을 던질 때 자신이 원하는 윷의 결과를 지정하여 윷을 던질 수 있다**

  1. 지정 윷 버튼을 클릭한다.

2. 도, 개, 걸, 윷, 모, 빽도 중 원하는 버튼을 클릭하여 던진다.

3. 지정 윷의 결과에 따라 해당 팀의 말이 이동 가능한 위치가 보드에 하이라이팅된다.

4. 하이라이팅 된 위치 중 이동할 위치를 클릭한다.

5. 말이 해당 위치로 이동하여 보드에 나타난다.

**\*b. 언제라도 시스템을 종료할 수 있다.**

임의로 시스템을 종료할 시에 해당 게임의 진행 상황을 저장하지 않고 종료된다.

해당 진행상황을 복구할 방법은 없다.

**\*c. 모든 플레이어는 해당 차례에 이동 가능한 위치에 자신의 다른 말이 있을 때 업을 수 있다.**

1. 플레이어가 버튼을 클릭해 윷을 던진다.
2. 플레이어는 이동할 말을 선택한다.
3. 해당 말이 이동 가능한 위치가 하이라이팅 된다.
4. 하이라이팅 되어있는 위치 중 같은 팀의 말이 존재하는 경우, 그 위치를 클릭하여 말을 업는다.
5. 플레이어의 말 두 개가 보드의 같은 위치에 나타나고 이후 함께 이동한다.

**\*d. 이동 가능한 위치에 상대편의 말이 있을 때 상대편의 말을 잡을 수 있다.**

1. 플레이어가 윷을 던진다.
2. 플레이어는 이동할 말을 선택한다.
3. 이동 가능한 위치가 하이라이팅된다.
4. 하이라이팅 되어 있는 위치 중 상대편의 말이 있는 경우, 플레이어가 상대편의 말이 있는 곳으로 이동한다.
5. 해당 위치에 있던 상대편의 말은 플레이어의 말에게 잡힌다.
6. 상대편의 말이 출발점으로 되돌아가고 해당 말은 다시 출발점에서 시작해야 한다.
7. 플레이어는 윷을 또 던진다.

**\*e. 이동 불가능한 위치(하이라이팅 되어 있지 않은 위치)를 클릭할 수 없다.**

1. 플레이어가 윷을 던진다.
2. 플레이어는 이동할 말을 선택한다.
3. 이동 가능한 위치가 하이라이팅 된다.
4. 하이라이팅 되어있지 않은 위치를 클릭한다.
5. 변화가 없고 하이라이팅 되어있는 위치를 클릭하는 이벤트를 기다린다.

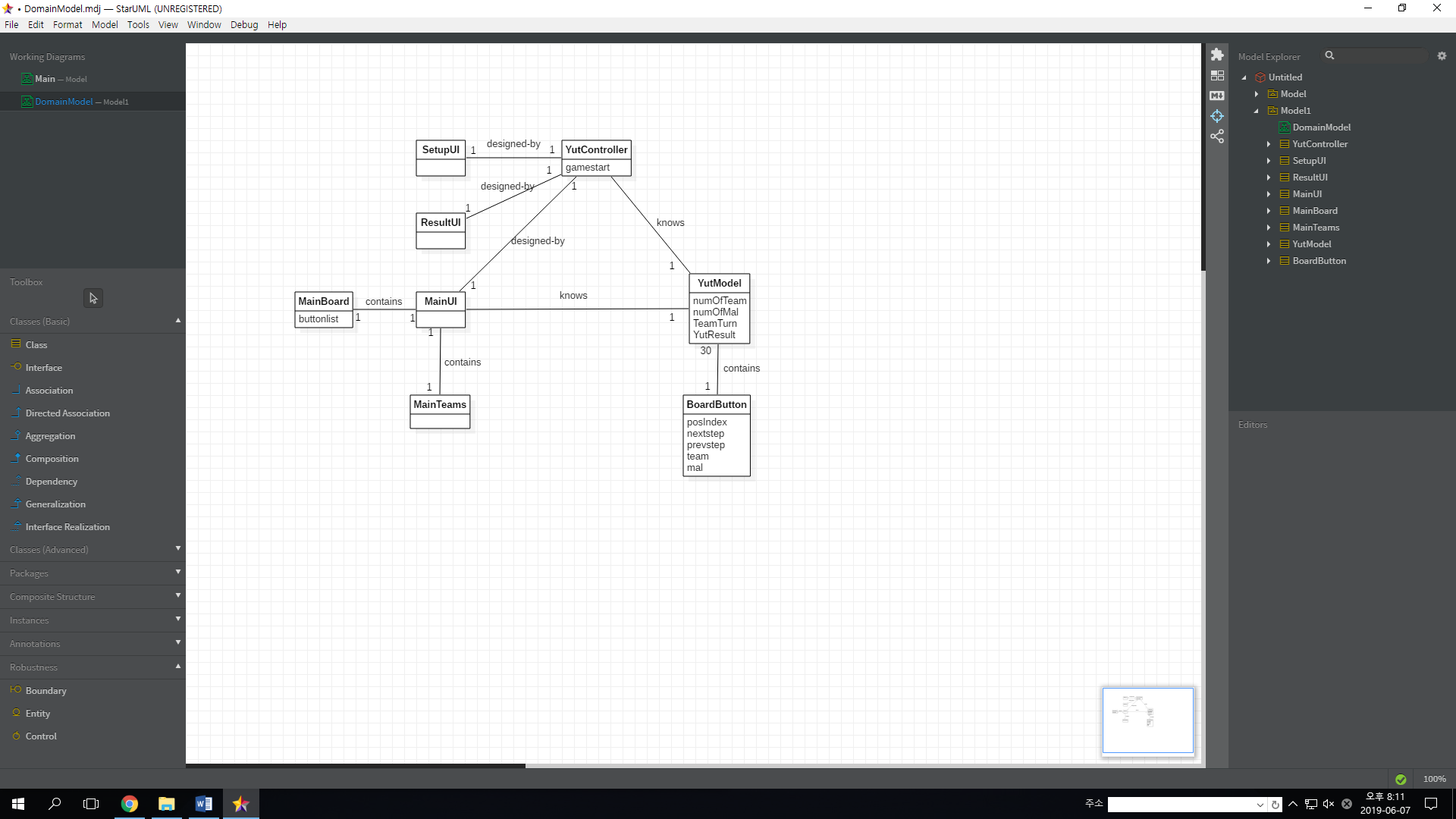
**\*f. 윷을 던져 윷 혹은 모가 나오면 윷을 한 번 더 던진다.**

1. 플레이어가 윷을 던진다.
2. 윷 혹은 모가 나오면 한 번 더 윷을 던진다.
3. 첫 번째 던진 결과에 대해 이동할 말을 선택한다.
4. 해당 말이 이동 가능한 위치가 하이라이팅 된다.
5. 하이라이팅 된 위치 중 이동할 위치를 선택한다
6. 해당 위치로 이동한다.
7. 두 번째 결과에 대해 이동할 말을 선택한다.

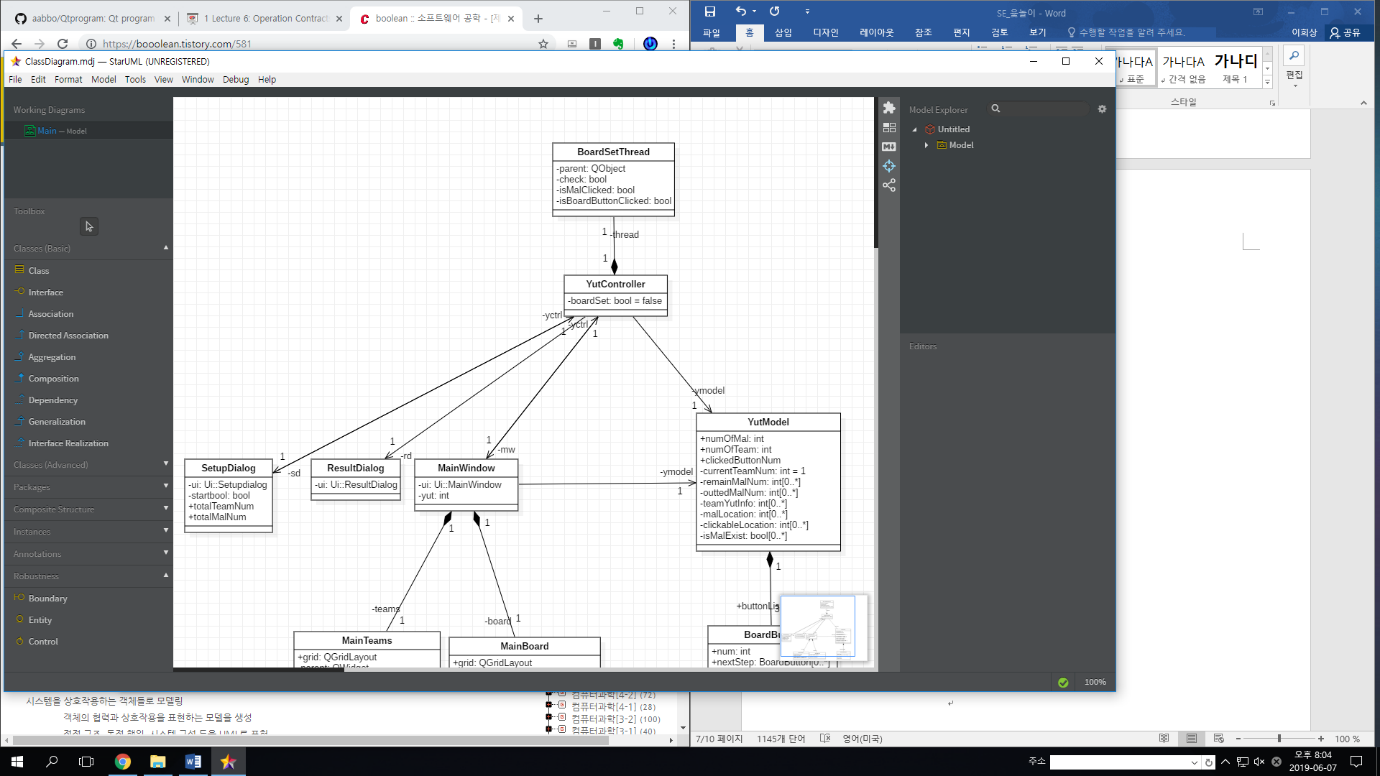
4- 6 과정을 한 번 더 반복한다.

**\*g. 윷판 위에서 우클릭 시 (기존에 선택했던) 이동하고자 하는 말을 변경 가능하다.**

1. 플레이어가 윷을 던진다.
2. 이동 가능한 위치가 하이라이팅 된다.
3. 하이라이팅 되어있지 않은 위치를 클릭한다.
4. 변화가 없고 하이라이팅 되어있는 위치를 클릭하는 이벤트를 기다린다.
5. **System Sequence Diagram**
6. **Operation Contracts**
7. **설계 및 구현**
8. **설계**
   1. **Domain Model**



* 1. Class Diagram



1. **구현** 
   1. Sequence Diagram
   2. State chart
   3. MVC model

본 프로그램은 java의 observable interface를 사용하지 않고 c++언어 기반인 Qt GUI 프로그램을 사용했다. java를 이용하면 model이 java.util.Observable을 extends하여 바로 view를 업데이트 할 수 있으나 Qt에서는 해당 기능이 없어서 아래와 같이 코드를 구현했다.

* + 1. 사용자가 게임을 조작한다.
    2. view에서 이벤트를 인식한다.
    3. view가 이벤트를 controller에게 알린다.
    4. controller는 원하는 결과를 얻기 위해 model을 호출한다.
    5. model은 계산 후에 결과를 return 한다.
    6. controller는 model이 return한 결과를 view를 호출하여 넘겨준다.
    7. view는 새로운 데이터를 이용해 사용자에게 보여준다.
* **Controller** : YutController클래스와 BoardSetThread 클래스에서 사용자가 게임을 조작하는 전반적인 순서를 제한한다. Qt의 특성상 각종 이벤트를 view의 class에서 받을 수 밖에 없다.(slot, signal) 대신 이벤트의 결과를 받고, 모델에 보내서 각종 처리를 한 후 그 결과를 받아 view에 보낸다.
* **Model** : YutModel클래스와 BoardButon클래스가 해당된다. 참여 팀의 수, 말의 개수, 말이 어디에 위치하는지, 던진 윷의 결과 등등을 모두 저장하고 있다. BoardButton클래스는 따로 윷판에 대한 정보를 가지고 있다. (현재 칸의 이전 칸, 다음 칸, 여기에 위치한 말의 수 등등) Model은 View와 Controller에 대한 것을 아무것도 모르지만 View와 Controller는 Model을 알고있다.
* **View** : SetupDialog, ResultDialog, MainWindow, MainTeams, MainBoard 클래스가 View에 해당된다. View는 버튼을 어디에 위치할지, 어떤 이미지를 보여줄지, 등등 보여주는 것에 대한 모든 것을 처리한다. 우리가 만든 화면 구조 말고 다른 방식으로 만들고 싶다면 뷰 코드를 수정하면 된다. 뷰는 크게 3가지가 있는데 처음에 게임을 시작할 때 참여 팀, 말의 개수를 정하는 SetupDialog.cpp, 게임을 진행하는 MainWindow.cpp, 게임 결과를 보여주는 ResultDialog.cpp가 있다. 이때 SetupDialog와 ResultDialog는 동작이 단순하여 model을 알 필요가 없다고 판단, controller만 알 수 있도록 설계했다. MainTeams는 게임화면 상에서 팀 별 말의 현황을 보여주는 View를 구현하는 클래스이고 MainBoard는 윷판에 관련된 View를 구현하는 클래스이다. 둘다 MainWindow에 속해있다.

1. **보완점** --- Qt의 한계로 인한
2. **사용법 및 테스트 결과**
3. **사용법**
4. **테스트 결과**
5. **Project Management**
6. **Github Repository 주소**

개발 및 문서작업을 진행한 깃헙 레포지토리 주소는 <https://github.com/aabbo> 이다. 개발은 Qtprogram 디렉토리에서, 문서작업 및 회의 기록은 doc 디렉토리에서 진행하였다.

1. **Progress History**

* Operation Contracts

precondition – assumptions about the state of the system or objects in the Domain Model before execution of the operation. non-trivial assumptions the reader should be told

**postcondition – state of objects in the Domain Model after completion of the operation**

Express postconditions in the past tense. To emphasize that they are observations about state changes that arose from an operation, not an action to happen. That’s why they are called postconditions. 🡪 Ex> A SalesLineItem was created.

system blackbox로 들어가는 **인풋에 대해서만 하면 되는지**를 모르겠다??!

--- postconditions는 **instance creation / association formed / attribute modification 세 개로 분리해서 하면 될듯…**

Operation contracts **use a pre and post-condition form to describe detailed changes to objects in a domain model** as the result of a system operation.

[Main Scenario]

**Contract : 1. 팀, 말 개수 정하기**

Operation: 팀,말 개수 정하기(Team: integer, Mal: integer)

Cross References: Use Cases: 팀 수 정하기, 말 수 정하기

Preconditions: 게임 플레이 이전에 일어난다.

Postconditions: 각각의 팀 객체(?)가 생성되었다. (instance created)

팀별로 말 객체(?)가 생성되었다. (instance created)

팀별로 자신의 말과 연결되었다. (associations formed)

**Contract : 2. 보드 초기화**

Operation: initializeBoard(Team: integer, Mal: integer)

Cross References: Use Cases: 팀 수 정하기, 말 수 정하기

Preconditions: 게임 플레이 이전에 일어난다.

Postconditions: 보드 객체가 생성되었다. (instance created)

보드랑 뭐가 연관관계 생성되었다…(association formed)

(attribute modification)

**Contract : 3. 윷 던지기**

Operation: throwYut()

Cross References: Use Cases: 윷 던지기

Preconditions: 자기 팀의 차례일 때 일어난다.

Postconditions: (instance created)

(association formed)

(attribute modification)

**Contract : 4. 이동할 말 선택**

Operation: chooseMal()

Cross References: Use Cases: 말 이동하기

Preconditions: 자신의 팀의 차례에 윷을 던진 이후에 일어난다.

Postconditions: (instance created)

(association formed)

(attribute modification)

**Contract : 5. *갈 수 있는 곳 표시***

Operation: highlightPossiblePoint(Mal: Mal)

Cross References: Use Cases: 말 이동하기

Preconditions:

Postconditions: (instance created)

(association formed)

(attribute modification)

**Contract : 6. 말 이동할 곳 선택**

Operation: setPoint(Mal: Mal, Cell: Cell)

Cross References: Use Cases: 말 이동하기

Preconditions: 이동할 곳은 하이라이팅 되어있는 위치 중에서만 선택 가능하다.

Postconditions:(instance created)

(association formed)

(attribute modification)

**Contract : 7. *보드 업데이트***

Operation: updateBoard(Board:Board)

Cross References: Use Cases: 말 이동하기

Preconditions: 팀의 차례가 넘어가기 전에(? 동시에인가,,,) 보드 업데이트가 일어난다.

Postconditions: (instance created)

(association formed)

(attribute modification)

**Contract : 8. 승자 표시**

Operation: displayWinner(Team:Team)

Cross References: Use Cases: *게임 종료*

Preconditions: 가장 먼저 자신의 팀의 말을 모두 보드 밖으로 뺀 팀이 승자이다.

Postconditions: (instance created)

(association formed)

(attribute modification)

**Contract : 9. 게임 종료**

Operation: gameOver()

Cross References: Use Cases: 게임 종료

Preconditions: 한 팀의 모든 말이 보드 밖으로 나와 승자가 결정된 상태이다.

Postconditions: (instance created)

(association formed)

(attribute modification)

[Extension – a ]

**Contract : 1. 팀, 말 개수 정하기**

Operation: 지정 윷integer, Mal: integer)

Cross References: Use Cases: 팀 수 정하기, 말 수 정하기

Preconditions: 게임 플레이 이전에 일어난다.

Postconditions: 각각의 팀 객체(?)가 생성되었다. (instance created)

팀별로 말 객체(?)가 생성되었다. (instance created)

팀별로 자신의 말과 연결되었다. (associations formed)