SE Term – Project Document

2019-1

과목	휴먼 ICT 소프트웨어 공학
교수님	이 찬 근
Team	Project_Team_11
이름	선승엽 20144753 양명철 20122776 정여민 20174374 최은정 20156033

0. Team member & Roll

- 선승엽 20144753 (Algorithm & Development)
- 양명철 20122776 (Document & Presentation)
- 정여민 20174374 (UI Design)
- 최은정 20156033 (Documentation)

1. Vision

1.1 Introduction to the System

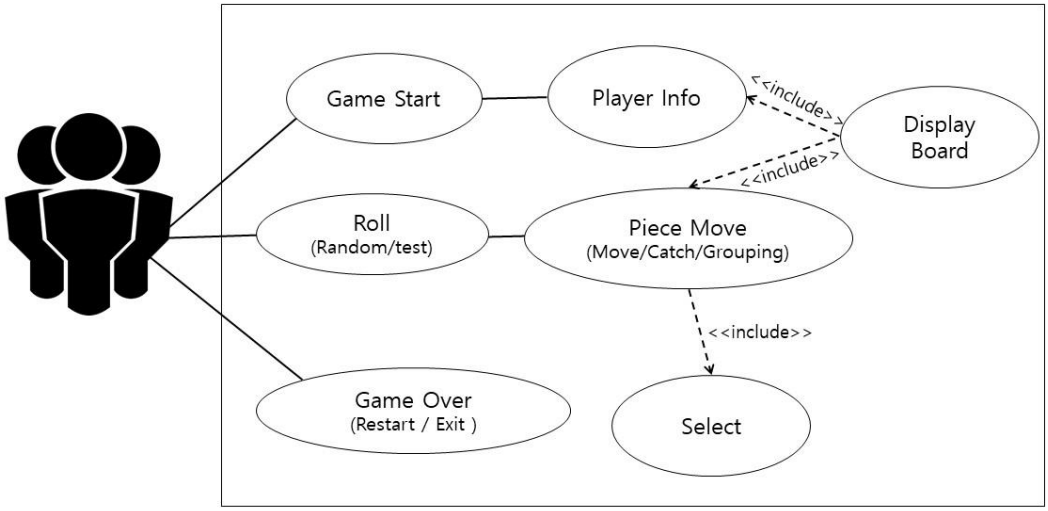
이 프로그램은 한국의 전통놀이인 윷놀이를 바탕으로 제작되었다. 게임을 하는 방법은 비교적 쉬우나 게임에서 이기기 위해서는 주어진 규칙을 익혀 최대한으로 활용해야 한다. 게임 시작 전 최소 2 명에서 최대 4 명의 유저를 설정할 수 있으며, 말판 위에서 움직이는 말은 최소 2 개에서 5 개까지 설정할 수 있다. 순서를 정하기 위한 윷을 던져서 유저의 게임 순서를 정하게 되며 중복되지 않도록 한다. 유저 설정 이후에 윷을 던지면서 게임이 본격적으로 시작된다. 윷을 던진 후에는 도,개,걸,윷,모,빽도 중 하나의 결과를 확인하고 새로운 말을 추가하거나 기존의 말을 이동한다. 말이 이동하여 상대방의 말을 잡으면 한 번 더 기회가 주어지고 잡힌 말은 출발점으로 이동해 처음부터 다시 시작한다. 자신의 말을 업을 경우 말을 묶어 같이 이동시킨다. 또한, 윷이나 모가 나온 경우에는 윷을 한 번 더 던질 기회가 주어지고 모든 말이 이동하고 윷을 던질 기회가 없다면 다음 유저에게 턴이 넘어간다. 한 유저의 모든 말이 출발하여 말판을 한 바퀴 돌아 다시 출발점으로 도착한 경우에 승자가 정해지고 게임이 종료된다.

1.2 Main features

윷놀이 프로그램은 기존의 전통적인 윷놀이 규칙을 따르고 있다. OOAD 를 적용하여 Python 언어를 사용하여 소프트웨어 개발하였다. 프로젝트 요구 사항대로 게임 시작 시 참여자의 명수(최소 2 명, 최대 4 명)와 게임 말 개수(최소 2 개, 최대 5 개)를 지정할 수 있다 그리고 Pyqt5 UI 를 사용하여 표준 윷놀이 판을 사용하여 각 참여자의 말의 현재 위치가 표시하였다. 각 턴의 진행을 위해 <랜덤 윷 던지기> 버튼과 <지정 윷 던지기> 버튼을 구현하였다. 사용자가 윷 던지기 결과를 적용할 게임 말을 선택할 수 있으며, 그에 따라 진행된다. 게임 말을 업는(grouping) 기능을 지원하고 다른 사용자의 말을 잡는 기능을 지원한다. 게임 말들이 출발에서 시작해서 먼저 모든 말을 내보내는 팀이 게임에 승리하며, 이때 어느 팀이 승리했는지 표시하는 기능도 구현을 하였다. OOAD 기법을 적극적으로 사용하여 개발하였다.

MVC 아키텍처 패턴을 사용하여 UI 와 모델을 분리하여 구현하였다. 테스트 용이한 설계를 하여 Unit 으로 모델 테스트를 수행하였다.

1.3 UseCase Diagram



[UC1] 게임 시작 후 플레이어의 정보를 입력하는 Use-case

[UC2] 윷을 던지는 Use-case

[UC3] 말을 선택하는 Use-case

[UC4] 말을 움직이는 Use-case

[UC5] 교차점에서의 플레이어의 움직임에 대한 Use-case

[UC6] 같은 팀의 말을 업는(Grouping) Use-case

[UC7] 다른 팀의 게임의 말을 잡는 Use-case

[UC8] 말이 골인하는 Use-case

[UC9] 게임 종료 Use-case

1.3 UseCase Discription

[UC1] 게임 시작 후 플레이어의 정보를 입력하는 Use-case

1.1 Preconditions : X

1.2 Main Flow

게임 시작 시 플레이어의 명수(최소 2 명, 최대 4 명), 플레이어의 이름 및 게임 말 갯수(최소 2 개, 최대 5 개)를 입력한다.

1.3 Subflows : X

1.4 Alternative Flows:

[E1] 플레이어 수는 2 에서 4 사이의 정수. 플레이어가 정수를 입력하지 않거나 숫자가 2 와 4 사이가 아닌 경우 게임은 플레이어에게 다시 플레이어 수를 다시 입력하도록 요청한다.

[E2] 말의 개수는 2 에서 5 사이의 정수. 플레이어가 정수를 입력하지 않거나 숫자가 2 와 5 사이가 아닌 경우 게임은 플레이어에게 다시 말의 개수를 다시 입력하도록 요청한다.

[E3] 이름은 빈 문자열은 불가. 플레이어가 빈 문자열을 입력하면 게임은 플레이어에게 이름을 다시 입력하라고 요청한다.

[E4] 취소 버튼을 누르면 플레이어 정보 창이 닫히고 게임이 종료된다.

[UC2] 윷을 던지는 Use-case

2.1 Preconditions :

1. 플레이어의 턴 순서는 UC1 에서 유저정보를 설정하면서 순서대로 결정된다.

2.2 Main Flow : 게임은 차례로 진행되는 턴 기반이다. 플레이어 정보가 입력되면 첫 번째 플레이어의 차례가 시작된다. 각 플레이어는 자신의 차례에 윷 4 개를 한번에 던지도록 하는 '던지기' 버튼을 누른다.

2.3 Subflows :

[S1] 테스트용 버튼을 누르면 빽도/도/개/걸/윷/모를 선택할 수 있고 선택한대로 윷이 던져져 선택한 윷던지기 결과를 확인할 수 있다.

[S2] 랜덤 윷던지기 버튼을 누르면 윷이 랜덤으로 던져진다.

[S3] 윷/모가 나온 경우는 윷을 한 번 더 던진다.

[S4] 윷을 한 번 더 던질 경우, 정해진 시간 내에 윷을 던지지 않으면 턴이 종료된다.

2.4 Alternative Flows : X

[UC3] 말을 선택하는 Use-case

3.1 Preconditions : 플레이어가 자기 턴에 윷을 던져 결과를 확인한 상태이다.

3.2 Main Flow : 말의 움직임은 플레이어가 던진 윷의 결과에 따라 달라진다. 플레이어는 새로운 말을 추가하거나 이전의 말을 선택해 이동시킬 수 있다.

3.3 Subflows :

[S1] 플레이어는 이동하기 위해 새로운 말을 선택해 추가할 수 있다.

[S2] 플레이어는 이동하기 위해 기존의 말을 선택할 수 있다.

3.4 Alternative Flows :

[E1] 추가된 말의 개수가 최대치일 때 더 이상 말을 추가할 수 없다는 문구가 뜬다.

[E2] 이전에 움직인 말이 보드에 없을 시, 새로운 말을 추가한다.

[E3] 자신의 말이 아닌 말을 선택한 경우에는 에러메시지를 보여준다.

[E4] 자신의 말이 없는 빈 맵을 선택한 경우에는 아무 일도 일어나지 않게 한다.

[UC4] 말을 움직이는 Use-case

4.1 Preconditions : UC3 에서 플레이어가 이동할 말을 선택한 상태이다. .

4.2 Main Flow : 도 → 한 칸, 개 → 두 칸, 걸 → 세 칸, 윷 → 네 칸, 모 → 다섯 칸 앞으로 움직인다. 빽도가 나온 경우에는 뒤로 한 칸 이동한다. 플레이어가 던진 윷의 결과에 따라 플레이어가 선택한 말이 이동하며 말의 이동이 끝나면 턴이 끝난다.

4.3 Subflows :

4.4 Alternative Flows :

[E1] 처음에 빽도가 나오면 무시하고 턴을 종료한다.

[UC5] 교차점에서의 플레이어의 움직임에 대한 Use-case

5.1 Preconditions :

1. 플레이어의 차례.

2. 플레이어가 윷을 던진 후여야 한다.

3. 플레이어가 이동할 말을 선택한다.

5.2 Main Flow : 플레이어의 기존 위치가 모서리인 경우, 플레이어는 사각형 테두리가 아닌 시작점으로 가장 빠르게 갈 수 있는 대각선 방향으로 말을 움직인다. 말판의 정가운데 교차점의 경우에는 시작점과 가까운 방향으로 움직인다.

5.3 Subflows : X

5.4 Alternative Flows : X

[UC6] 같은 팀의 말을 업는(Grouping) Use-case

6.1 Preconditions :

1. 플레이어의 차례.

2. 플레이어가 윷을 던진 후여야 한다.

6.2 Main Flow : 플레이어가 움직인 위치에 같은 팀의 말이 이미 있다면, 해당 말을 업는다(grouping).

6.3 Subflows :

[S1] 업힌 말과 업은 말은 그룹으로 묶여 하나의 말처럼 같이 움직이게 된다.

6.4 Alternative Flows : X

[UC7] 다른 팀의 게임의 말을 잡는 Use-case

7.1 Preconditions :

1. 플레이어의 차례.

2. 플레이어가 윷을 던진 후여야 한다.

7.2 Main Flow : 플레이어가 움직인 위치에 다른 팀의 말이 이미 있다면, 해당 말을 잡을 수 있다.

7.3 Subflows :

[S1] 잡힌 말은 처음부터 다시 출발해야 한다.

[S2] 상대의 말을 잡으면 한 번 더 윷을 던진다.

7.4 Alternative Flows : X

[UC8] 말이 골인하는 Use-case

8.1 Preconditions :

1. 플레이어의 차례.

2. 플레이어가 윷을 던진 후여야 한다.

8.2 Main Flow : 말이 말판을 한 바퀴 돌아 도착점으로 올 때, 도착점을 지나친 경우에 도착으로 인정한다.

8.3 Subflows :

[S1] 골인 한 말 수만큼 유저의 말 개수가 수정된다.

8.4 Alternative Flows : X

[UC9] 게임 종료 Use-case

9.1 Preconditions :

1. 플레이어의 차례.

2. 플레이어가 윷을 던진 후여야 한다.

9.2 Main Flow : 출발점에서 시작하여 한 바퀴 돌아 상대편 말보다 먼저 모든 말이 모두 도착하면 게임이 종료된다.

9.3 Subflows :

[S1] 이긴 플레이어의 이름이 화면에 표시된다.

[S2] 게임 재시작 및 종료를 선택할 수 있다.

9.4 Alternative Flows : X

1.5 Operation Contract

Operation Contract 는 Preconditons 와 Postconditions 를 사용하여 시스템 조작 결과로 도메인 모델의 오브젝트가 변경된 부분을 상세하게 설명해주는 것이다. 이 프로그램의 경우 플레이어 정보를 입력하는 경우, 윷을 던지는 경우, 말을 선택하는 경우, 정보를 변경하는 경우 이렇게 4 가지에 대한 Operation Contract 를 작성하였다.

Operation	Init()
Cross Reference	**게임 시작 후 플레이어의 정보를 입력하는 Use-case(UC1)**
Preconditons	1. 게임 스타트 버튼이 눌린 상태여야한다.
Postconditions	· 플레이어의 이름이 기록되었다. · 플레이어 명수가 기록되었다. (2-4 명) · 플레이 할 말의 개수가 정해졌다. (2-5 개) · 플레이어 순서가 정해졌다. · 설정된 값을 System 에 전달했다. . (Association)

Operation	Roll()

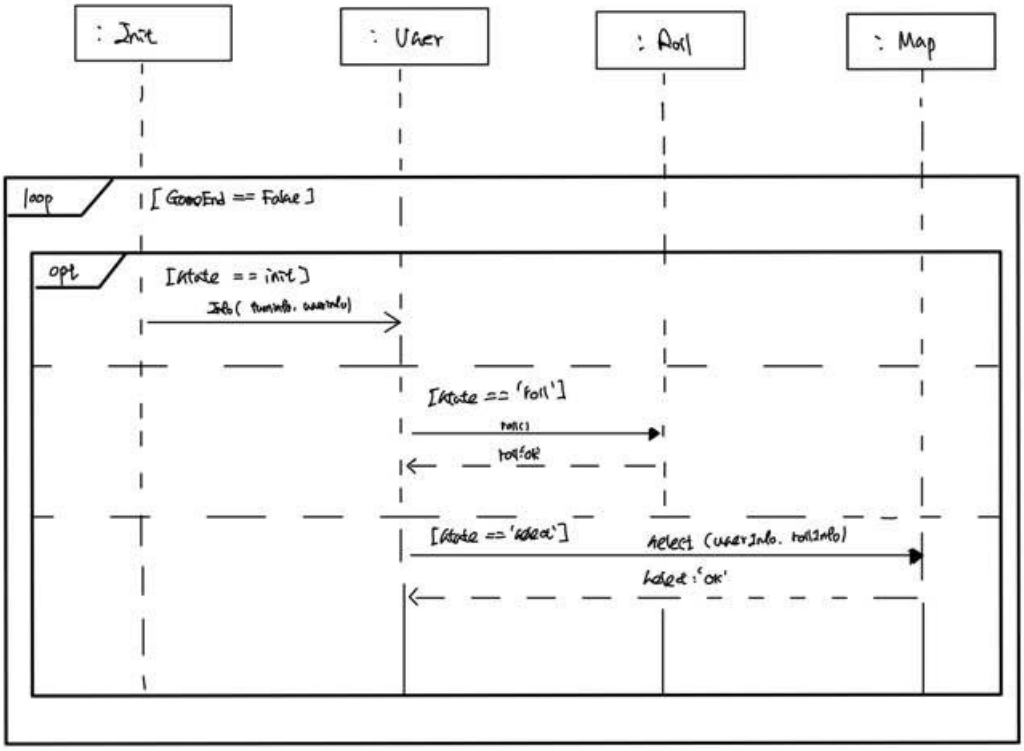
Cross Reference	윷 던지는 UseCase(UC2)
Preconditons	1. 플레이어의 정보가 입력되고 게임이 시작된 상황이어야한다. 2. 플레이어가 자신의 턴에서 윷던지기 버튼을 누르다.
Postconditions	· 랜덤함수를 사용해서 4 개의 윷을 임의의 0 과 1 로 표현되게 하고 그 결과를 확인했다. · 1 의 개수에 따라 도, 개, 걸, 윷, 모가 결정된다. · 결과값을 시스템에 전달한다. (Association)

Operation	change_info(self, departure, destination)
Cross Reference	말이 이동하는 UseCase(UC4)
Preconditons	1. 윷을 던지고 결과를 확인한다. .
Postconditions	· 말판의 인덱스를 통해 현재위치에서 결과값 만큼 이동하고 도착점으로 다시 현재위치를 변경했다. . 턴 정보가 바뀌었다 · (도착점에 우리팀이나 상대편의 말이 있다면 말을 잡거나 업는 UseCase 로 이동한다.) · 변경된 정보를 전달했다.

Operation	select(self, n_roll, map_index, user_info):
Cross Reference	말을 선택하는 UseCase(UC3)
Preconditons	1. 윷을 던지고 결과를 확인한다. .

Postconditions	· 플레이어가 새로운 말을 추가했거나 기존의 말을 이동하기 위해 말이 있는 맵 위치를 누른다. (Composition) · 말판의 교차점에서 출발하게 된다면 정해진 방향(up, down, None)에 따라 말을 이동시킬 위치를 확인했다.

1.6 SSD(System Sequence Diagram)

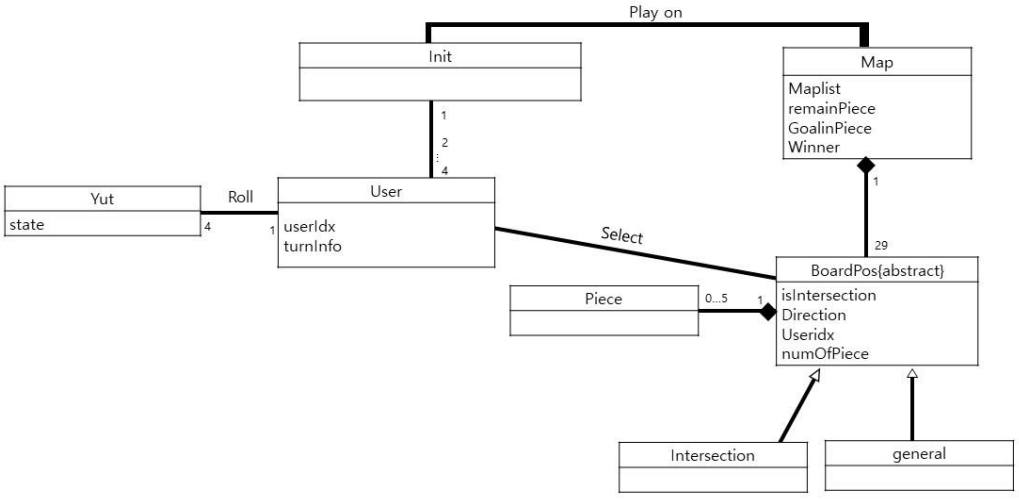


프로젝트를 개발할 때 시스템 시퀀스 다이어그램(System Sequence Diagram, SSD)을 사용하여 사용자와 시스템간에 특정 작업이 수행되는 방법을 설명할 수 있다. 이 시스템 시퀀스다이어그램의 목적은 시각적 형식으로 유스케이스 를 설명하는 것이다. 이 모델은 행위자(시스템에 영향을 미치는 사람)와 시스템이 작업을 수행 할 때의 논리를 보여준다. SSD 에서는 개발중인 시스템이 어떻게 동작하는지는 기술하지 않고 무엇을 하는지를 기술함으로써 시스템 행위(행동)을 보여준다.

여기서 행위자는 User 이고, System 은 Map 이다. 게임 시작 화면에서 유저가 시스템에 유저 정보를 설정하고 전달해서 게임을 시작할 수 있다. 이후 게임이 시작되면 행위자는 시스템에 윷

던지기(roll)를 요청할 수 있고 그 결과값을 확인한다. 그리고 말판 위에서 말을 선택(select)해 새로운 말을 선택하거나 이동할 수 있다. 선택된 말이 움직여서 이동한 후 결과값을 유저에게 전달한다.

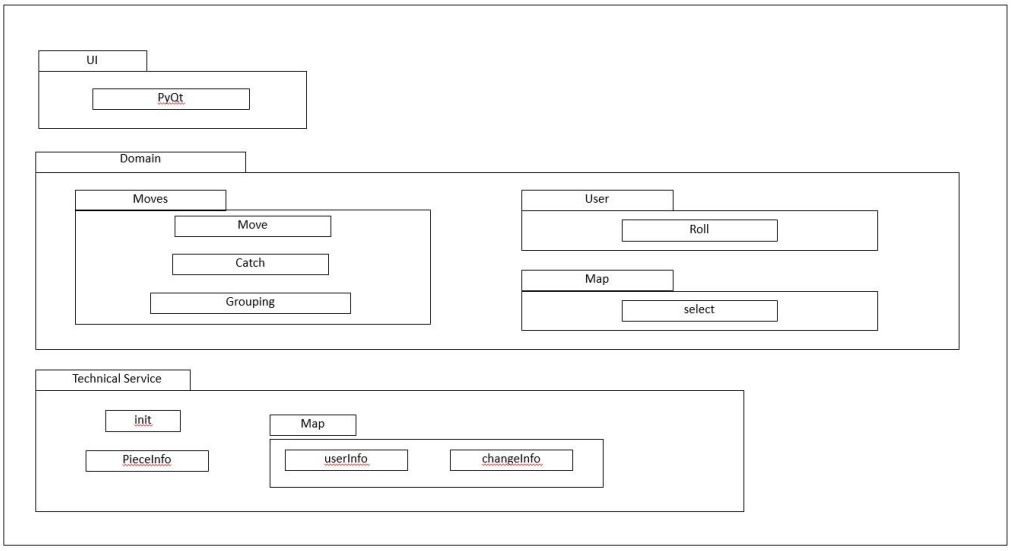
2. Domain Model



도메인 모델이란 도메인 안에 존재하는 개념적 클래스나 실제 상황의 객체를 시각적으로 표현한 것이다. 1 명의 유저는 4 개의 윷을 가지고 Roll 하게 된다. 유저의 말은 BoardPos 안에서 포함되어 나타난다. 하나의 말 판 자리에는 0 개부터 최대 5 개까지 포함된다. BoardPos 는 abstract 으로 나타내어 교차점이 아닌 지점(General)과 교차점 지점(Intersection)에서 이를 상속받아 구현할 수 있게 된다. 그리고 이렇게 구현된 말 판은 29 개가 모여 Map 이 된다.

3. Software Architecture & Design model

3.1 Layer Architecture



논리 아키텍쳐(Logical Architecture)는 시스템의 각종 기능 및 서비스를 구분하여 나타내고, 개념적인 정보의 흐름을 나타내는 골격이다. 각 서비스의 구현을 위해서는 사용자와 시스템 간의 정보교환 및 전체 시스템의 절차를 명확히 설정한 후, 절차 안에서 이루어지는 정보와 기능을 추출하여 이 정보와 기능간의 관계를 모델화하는 작업이 필요하다.

Layered Architecture 는 효율적인 개발과 유지보수를 위해 계층화하는 것이다. 계층은 시스템의 주요 측면에 대한 일관된 책임을 가진 클래스, 패키지 또는 하위 시스템을 간단하게 보여준다. 상위 레이어가 하위 레이어의 서비스를 호출하도록 구성되지만 일반적으로 하위 레이어가 상위 레이어를 호출하도록 구성되지 않는다.

사용자 인터페이스(User Interface)는 파이썬을 활용하여 GUI 클래스를 구성할 수 있는 PyQT 를 사용해서 사용자가 게임 화면을 보고 게임을 할 수 있도록 제작했다.

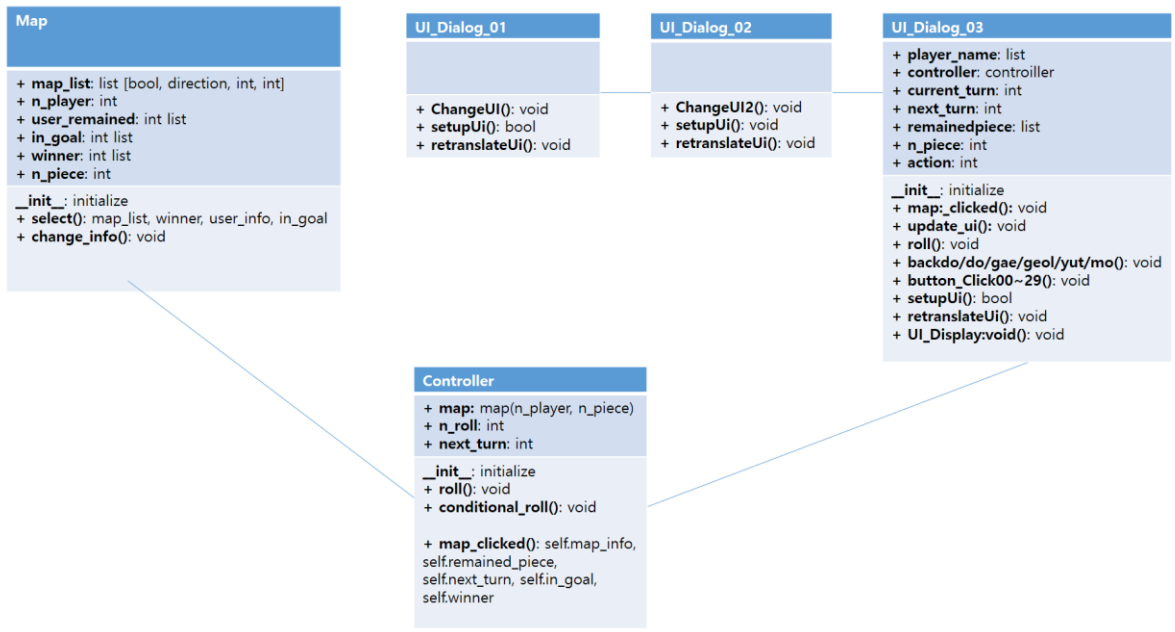
도메인(Domain)층에서는 응용 프로그램 논리 및 도메인 개체와 같이 응용 프로그램 요구 사항을 충족하는 도메인 개념을 나타내는 소프트웨어 개체를 다룬다. 여기서는 유저, 맵 이동기능 등 게임을 하는데 필요로 하는 개념들을 넣어 도메인 레이어를 구성하였다.

기술 서비스(Technical Service)층에서는 데이터베이스 또는 인터페이스와 같은 지원 기술 서비스를 제공하는 객체나 하위시스템은 다룬다. 이 윷놀이 게임의 경우 게임시작 시 유저 정보를 초기화하는 Init 이 이 레이어에 포함되어 있다. 플레이어의 수와 그 플레이어의 순서 그리고 말의 개수를 가지고 있는 User Info 와 사용자가 게임 할 때 사용하는 말의 정보를 가진 Map info, 그리고 게임을 하면서 변하는 정보들을 관리하는 Change info 또한 이 기술

서비스층에 포함된다. 이 레이어에 포함된 서비스는 대개 독립적이며 여러 시스템에서 다시 사용할 수 있다..

이렇게 Layer Architecture를 구성하면 관련 복잡성은 캡슐화되고 분해 가능하게 된다. 또한, 일부 레이어는 새로운 구현으로 대체 될 수 있고, 하위 레이어에는 재사용 가능한 기능이 포함되어 있다. 일부 계층 (주로 도메인 및 기술 서비스)을 배포 할 수 있게 되며, 논리적 인 세분화 때문에 팀 별 개발이 도움이 될 수 있다. 하지만 몇몇 상황에서는 계층을 추가하는 것은 성능 문제를 초래할 수 있으며, 이 패턴이 모든 문제에 다 적용될 수 있는 것이 아니라는 점을 염두에 두어야 한다.

3.2 Class Diagram



소프트웨어 엔지니어링에서 통합 모델링 언어(UML)의 클래스 다이어그램(class diagram)은 시스템의 클래스, 그 속성(attribute), 운영(또는 방법)/(operation(or method)) 및 객체 간의 관계(relationship between objects)를 보여줌으로써 시스템의 구조를 설명하는 정적 구조 다이어그램의 일종이다.

윷놀이 게임 프로그램의 클래스는 Map, UI_Dialog_01, UI_Dialog_02, UI_Dialog_03, Controller 로 구성되어있다. UI_Dialog_01 클래스와 UI_Dialog_02 클래스는 화면이 순차적으로 전환이 되어 있어 관련이 있다. UI_Dialog_02 클래스와 UI_Dialog_03 클래스

역시 순차적인 전환을 하므로 관련이 되어있다. UI_Dialog_03 클래스는 윷놀이 게임 보드 디스플레이이다. 화면 정보를 Controller 를 통해서 전달이 된다. Controller 클래스는 이를 Map 클래스에 전달을 한다. Map 클래스는 윷놀이 게임결과를 Controller 클래스를 통해서. UI_Dialog_03 로 전달을 한다. Map class 와 UI_Dialog_03 는 각각 Map 클래스와 연관을 맺고 있다.

1) Map Class

Map 클래스는 map_list, user_remained, in_goal, winner 를 속성(attribute)로 가지고 있고, __init__으로 초기화를 하고 select(), change_info()의 도구(method)를 갖고 있다. map_list 는 list 의 형태를 갖고 있다.

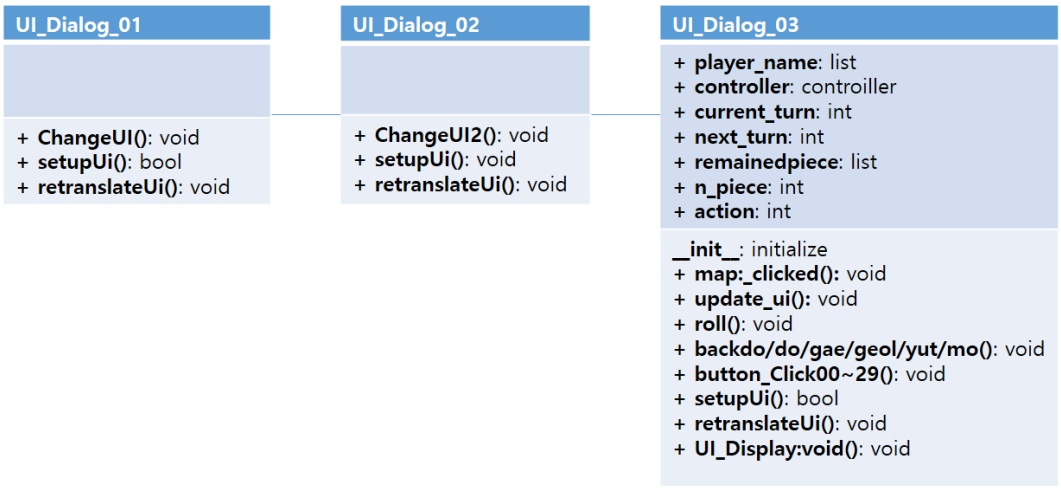
속성(attribute)는 다음과 같다. 첫 번째 리스트의 값은 bool 타입으로 True 이면 윷놀이 말판의 방향이 바뀌는 곳이고, False 이면 윷놀이 말판의 방향이 바뀌지 않는 곳이다. 두 번째 리스트의 값은 direction 으로 “up”으로 갈 것인지 “down”으로 갈 것인지, None 인지를 표시해준다. 세 번째 리스트의 값은 윷놀이 말판에 몇 개의 말이 들어 있는지를 표시한다. 네 번째 리스트의 값은 윷놀이 말판에 말 개수가 몇 개가 있는지를 표시한다. 윷놀이의 말이 업힌 경우(grouping) 표시가 된다. n_player 는 어떤 플레이어인지를 설정을 한다.

user_remained 은 list 의 형태로 각 플레이어당 몇 개의 말이 출발하지 않고 남아있는지를 표시한다. in_goal 은 list 의 형태로 각 플레이어별의 말의 개수가 goal 에 들어갔는지를 표시한다. Winner 는 초기값은 -1 로 설정이 되어있다. 승리한 플레이어의 user_info 의 값이 들어가 게임에서 이긴 플레이어를 표시한다.

도구(method)는 다음과 같다. __init__은 초기화를 한다. select()는 윷놀이 결과 몇 칸 움직일지를 나타내는 n_roll, 그리고 현재 말의 위치 map_index, 어떤 플레이어인지를 나타내주는 user_info, 말을 생성시킬지 안 생성시킬지 boolean 타입 generate 을 인자로

받는다. n_roll 의 결과를 user_info 와 map_index 를 반영하여 말을 이동시키는 역할을 한다. change_info 는 말이 출발하는 departure, 말이 도착하는 destination, 그리고 말 생성 여부를 나타내는 boolean 타입 generate 를 인자로 받는다. Depature 와 destination 값에 따라 map 에서 말의 위치를 변경한다.

2) UI Class



UI 를 담당하는 3 개의 class 에 대해서 살펴보겠다. UI Class 에는 UI_Dialog_01 클래스, UI_Dialog_02 클래스, UI_Dialog_03 클래스가 있다.

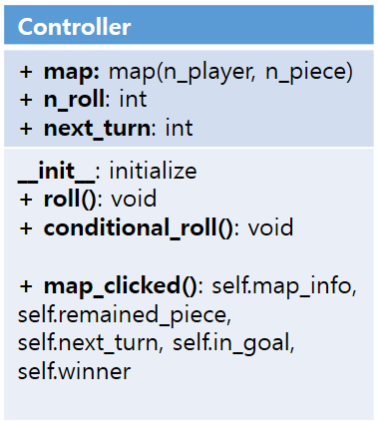
UI_Diaglog 01 클래스와 UI Diaglog_02 클래스에서 ChangeUI 함수는 한 윈도우창에서 다른 윈도우창을 띄울 때 사용한다. setupUi 함수는 UI 의 필요한 label 및 버튼, 그림, 그래픽 뷰, 위젯 등을 설정한다. retranslateUi 함수는 Dialog 의 위젯들(label, 버튼 등)에 글씨 및 그림을 넣는다. UI Diaglog_03 클래스는 __init__에서 특징들을 초기화 해준다. player_name 은 name1~4 의 list 자료를 갖는다. controller 는 controller 클래스의 controller 를 가져와서 설정한다. current_turn 은 0 으로 초기화를 해준다. next_turn 은 0 으로 초기화를 해준다. remainedpiece 는 list 의 형태로 각 플레이어의 n_piece 값을 갖는다. action 은 0 으로 초기화를 해준다. Action 의 변수에 따라서 윷을 던지거나(action=0), 윷의 결과가 화면에 출력이 되거나(action=1), 화면 상 말이 이동을 하고, 승자가 있으면 승자를 표시한다.(action=3)

map_clicked 함수는 화면과 남아 있는 말의 정보를 알려주고, UI 를 업데이트하고 승자가 있다면 승자를 표시하고 그렇지 않다면 어떤 플레이어의 턴인지를 표시한다. update_ui 함수는 윷놀이 모델 결과를 새롭게 UI 에 업데이트 기능을 수행한다. roll 함수는

controller 에서 roll 을 가져와서 도, 개, 걸, 윷, 모, 빽도를 수행하고 그 결과를 알려준다. 그리고 빽도, 도, 개, 걸, 윷, 모 함수는 controller 로부터 결과값을 받아와서 action 으로 구분한다. action 이 0 일 때는 윷을 던지는 것이고 던지고 나면 acton 은 1 로 바뀐다.

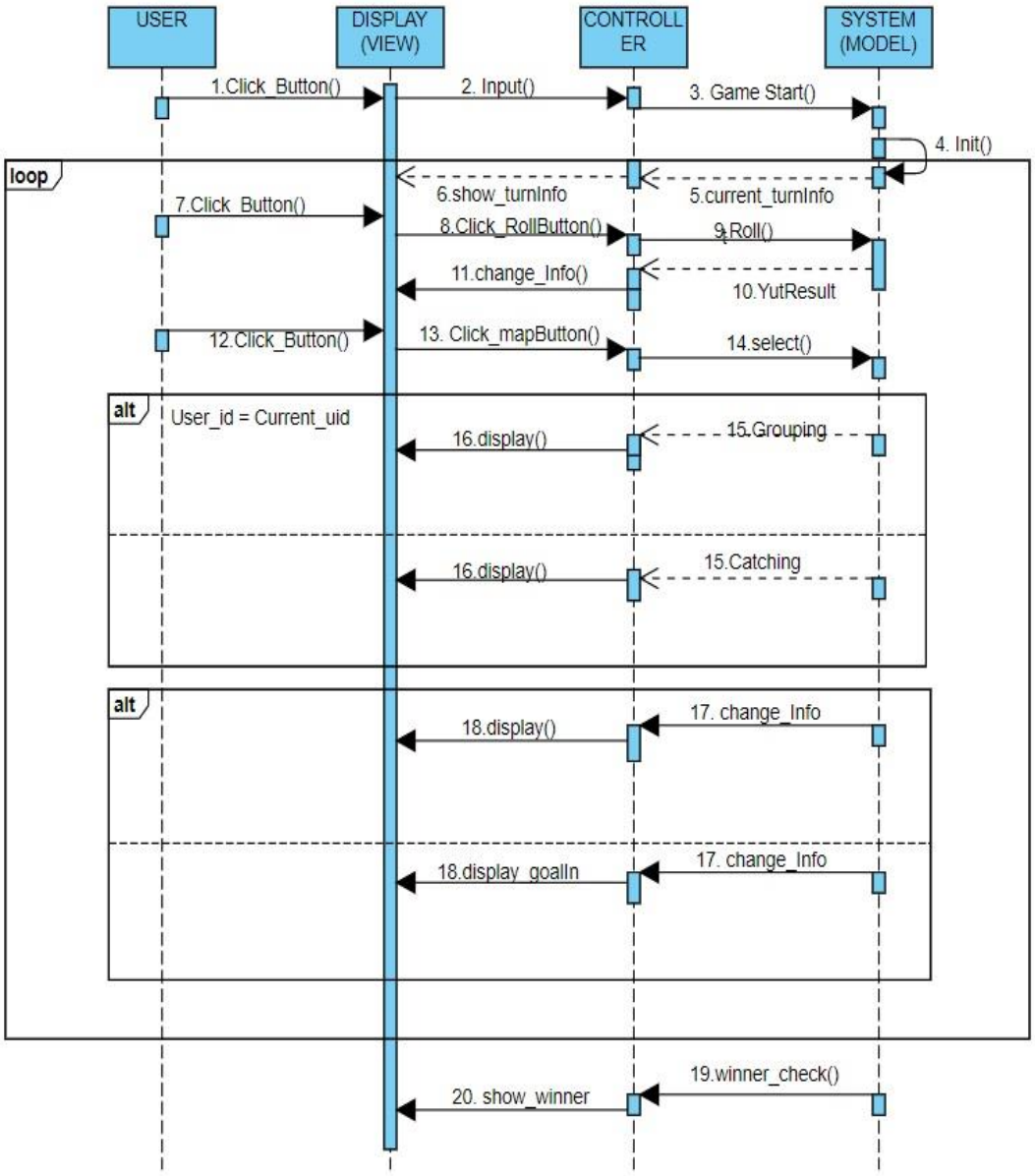
Def button_Click 은 말을 추가할 때 클릭이 되면 수행이 된다. button_Click00 ~ button_Click28 은 윷놀이 말판에 각 지점이 클릭이 될 때 수행이 된다. setupUI 는 그래픽 뷰, 말 추가 버튼, 윷 던지기 버튼, 지정 윷 던지기(도, 개, 얼, 윷, 모, 빽도), 윷놀이 보드판 각 플레이어의 말 5 개를 설정한다. 그리고 각 보드의 버튼 0 ~28 까지 있으며, 그 순서는 윷놀이 보드판의 인덱스와 일치한다. 버튼을 생성하고, 버튼 위치 및 사이즈, 버튼 이름을 설정한다. retranslateUi 함수는 각 버튼 및 레이블의 글자를 설정한다.

3) Controller Class



Controller 클래스에서 __init__함수는 변수들을 초기화한다. Map 은 Ymap 에서 Map(n_player, n_piece)를 가져온다. n_roll 은 윷놀이 결과값으로 0 으로 초기화를 한다. next_turn 은 다음 누가 플레이어를 나타낼지 0 으로 초기화를 한다. roll 함수는 윷놀이를 던지는 함수이다. yut 이라는 4 개의 list 에서 randint 를 사용하여 0 과 1 사이의 값이 나온다. 0 번째 배열에서만 1(도)가 나온 경우 빽도로 생각하여 -1 의 값을 반환한다. Yut 의 모든 값의 합이 2 가 나오면 개, 3 이 나오면 걸, 4 개 나오면 윷, 5 개 나오면 모를 반환한다. conditional_roll 함수는 지정 윷 던지기를 수행한다. map_clicked 함수는 윷 던지는 결과값인 n_roll, 현재 맵의 map_index, 다음 턴인 next_turn 을 출력한다. 그리고 다시 n_roll 을 0 으로 초기화한다. map_clicked 은 map_info, remained_piece, next_turn, in_goal, winner 를 반환한다.

3.3 Sequence Diagram



시퀀스 다이어그램은 문제 해결을 위한 객체를 정의하고 객체간의 상호작용 메시지 시퀀스를 시간의 흐름에 따라 나타내는 다이어그램이다.

이 프로그램에서는 사용자의 버튼 클릭 이벤트로 오브젝트간 상호작용을 시작한다고 보았다. 사용자가 게임시작 버튼을 누르면 버튼 클릭 이벤트가 전달되고 플레이어의 숫자와 말의 개수를 설정할 수 있는 화면으로 넘어간다. 그리고 유저가 값을 입력한 후에 게임 시작 버튼을 누르면 이벤트가 전달되어 Model, View, Controller 가 어떻게 상호작용하는지에 대해서 이 시퀀스 다이어그램을 통해 알 수 있다.

사용자가 View 에서 윷 던지기(Roll)버튼을 누르면 View 는 버튼 클릭 이벤트를 Controller 에 전달한다. Controller 는 이 이벤트로 Model 에 있는 Roll() 함수를 호출하게

되고, 윷 던지기 결과 YutResult 를 반환한다. 그리고 Controller 는 바뀐 정보를 View 에 전달한다.

사용자가 View 에서 말이 위치한 말판을 선택하면 버튼 클릭 이벤트를 다시 Controller 에 전달하고 Controller 는 이 이벤트로 Model 의 Select() 함수를 실행한다. 만약 도착지점에 다른 말이 있다면 유저 정보를 확인해 같다면 자동으로 합쳐지고 같지 않다면 그 말을 잡게 된다. 그리고 바뀐 정보는 Controller 에 전달되고 이를 통해 View 에 보여지게 된다.

사용자의 말이 도착지점에 도착하게 되면 사용자별로 도착한 말의 개수를 업데이트하고 바뀐 정보를 Controller 에 전달한다. Controller 는 바뀐 정보를 View 에 전달하여 사용자에게 보여지게 한다.

이 후, winner 가 정해졌는지 도착한 말의 개수를 통해 확인하고, 정해졌다면 loop 를 빠져나와 승자를 확인하고 View 를 통해 보여준다. 아니라면 loop 를 계속 돌아 승자가 정해질 때까지 현재 플레이어의 턴 정보를 Model 로부터 받아와 과정을 반복한다.

4. MVC Model

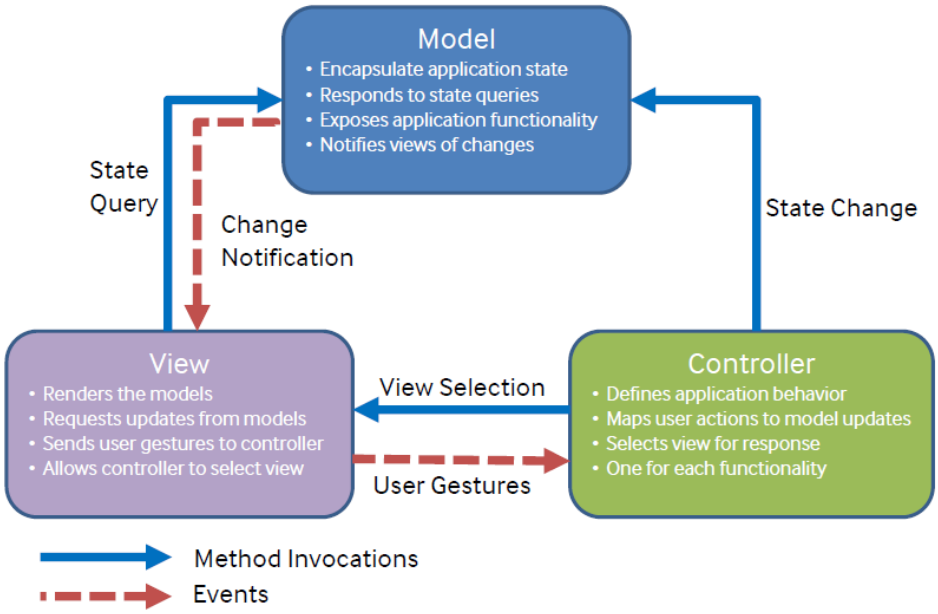
거시적 관점에서의 MVC architecture pattern 은 UI 가 빈번하게 변경된다는 사실을 인지하고 다른 요소들이 이에 영향을 받지 않게 하기 위한 것이다. 서로 다른 상호작용을 통해 업데이트하고 공유되는 데이터를 변화시키기 때문에 데이터에 대한 접근을 잘 관리해야 한다. 그렇기 때문에 중심기능을 UI 에서 분리시킬 필요가 있다.

이러한 잦은 변화에 영향을 받지 않기 위해서 각각 Model, View, Controller 로 책임을 나눠 핵심 기능을 분리하여 제어한다. 모델은 소프트웨어 내에서 데이터를 의미하고, 뷰는 사용자에게 보이는 화면 내용을 의미한다. 이러한 모델과 뷰의 상호 작용을 관리하는 것이 컨트롤러이다.

4.1 Major design decisions

우선 윷놀이 게임에서 요구하는 사항들을 파악하기 위해서 넷마블의 윷놀이 온라인 게임의 규칙과 플레이를 보고 제공된 윷놀이 게임의 규칙을 참고해 Usecase 를 만들어보았다. 이 후, 이를 바탕으로 UseCase Model (Use case diagrams/ Use case descriptions/ System Sequence Diagrams (SSD)/ Operation Contracts) 을 작성하였다. 각각의 기능들을 구현하는 데 있어 독립적으로, 효과적으로 제작하기 위해 OOP 적 design 을 고려하게 되었다. 윷놀이의 기능을 Model, View, Controller 로 나눠서 윷놀이 기능을 효율적으로 제어하고자 했다.

4.2 MVC Model의 적용



Model : Model 에서는 윷놀이 게임 진행을 담당한다. Map 클래스가 이 프로그램에서 Model 을 담당하고 있다. Model 은 사용자가 선택하는 말 판의 위치를 알고 있어야 하며, 말의 움직임을 제어한다. 모델에서 모든 유저의 말을 제어하고, 말의 위치 비교를 통해 발생하는 이벤트를 확인해 발생하는 변화를 처리한다. Model 은 Controller 를 통해서만 접근이 가능하고, Model 에서 변경된 정보는 모두 Controller 를 통해서 View 전달되어 사용자에게 보여지게 된다.

View : View 에서는 Model 에서 진행중인 작업을 표시하는 것이고, 현재 상태를 오른쪽 상단에 메시지를 통해 보여준다. View 에서는 유저의 말이 말판의 어느 위치에 있는지에 대한 위치 정보 등을 필요로 한다. View 에서는 전체 말판을 그리고, 그 위에 사용자의 말이 있다면 말을 그리고 아니라면 말판 옆에 말을 그려 말에 대한 정보를 표시해주어야 한다. View 에서 윷 던지기 버튼을 누르면 Clicked_Button 을 통해서 그 버튼을 누르는 마우스 이벤트 자체만 Controller 에 전달되고 이를 통해서 윷 던지기라는 기능이 실행되는 것이다.

Controller : .본 프로그램의 중심 기능이 되는 Model 과 View 를 상호작용을 관리하기 위한 것이다.여기서 View 과 Controller 를 Composition 관계로 연결해 기능이 작동하도록 한다. 사용자의 입력이 뷰를 통해서 컨트롤러로 들어오면 컨트롤러는 모델에서 데이터를 불러오게 된다. 모델에서 해당 데이터에 접근해 컨트롤러에 전달해주고 컨트롤러는 다시 이 정보를 뷰에 전달해 화면으로 출력하는 것이다. 이 프로그램에서는 UI 에서 말을 누르면 Controller 에 마우스버튼 이벤트가 전달 되고 Controller 는 이를 받아서 모델에서 맵의 현재 인덱스랑 다음 턴이 누군지 던진 윷의 결과를 전달한다. 모델은 정보를 수정하고 맵 정보와 남아있는 말의

개수 다음 턴이 누군지 도착한 말의 개수와 승자가 있는지를 반환한다. 그리고 이 정보를 다시 Controller 가 받아서 View 에 전달한다.

4.3 How to apply OOAD (diagram, code)

1. Encapsulation(캡슐화)

밖에서는 보이지 않도록 내부의 내용, 정보를 숨기는 과정이다. 외부에서 이 객체에 어떤 상태가 있는지 행위는 어떻게 하는지 알 필요가 없다는 것이다.

윷을 던지는 과정(roll)은 밖에서는 보이지 않는다. 윷을 던지는 과정을 Controller 클래스만 알고 있고 밖으로 보이지 않아도 된다. 또한, 비슷한 기능을 하는 roll(랜덤 윷 던지기)와 conditional_roll(지정 윷 던지기)이 함께 있어 cohesion 을 높이는 역할을 한다.


```python
class Contoller():

    def __init__(self, n_player, n_piece):
        self.map = ymap.Map(n_player, n_piece)
        self.n_roll = 0
        self.next_turn = 0

    def roll(self):
        yut = [random.randint(0, 1), random.randint(0, 1), random.randint(0, 1), random.randint(0, 1)]
        if sum(yut) == 1:
            if yut[0] == 1:
                self.n_roll = -1
            else:
                self.n_roll = 1
        elif sum(yut) == 2:
            self.n_roll = 2
        elif sum(yut) == 3:
            self.n_roll = 3
        elif sum(yut) == 4:
            self.n_roll = 4
        else:
            self.n_roll = 5

    def conditional_roll(self, num):
        self.n_roll = num
```

2. Inheritance(상속)

상속은 기존의 클래스를 재사용해 새로운 클래스를 만드는 것이다. 이러한 특징으로 적은 양의 코드로 새로운 클래스를 만들 수 있으며 코드를 한번에 관리할 수 있다.

Unit-test 를 하기 위해 unittest.py python 파일의 TestMap 클래스는 ymap 의 Map 클래스를 상속받는다. ymap 의 함수를 가져와서 기능이 잘 구현되었는지 확인한다.

```python
import ymap

class TestMap(ymap.Map):
    """
    Map의 child class
    Map list의 element를 수정할 수 있는 method가 추가됨.
    """
    def modify(self, map_index, user_info, n_piece):
        self.map_list[map_index][2] = user_info
        self.map_list[map_index][3] = n_piece
        self.user_remained[user_info] = self.user_remained[user_info] - n_piece
```

3. Composition(컴포지션)

다른 클래스의 일부 기능을 그대로 이용하고 싶으나, 전체 기능 상속은 피하고 싶을 때 사용한다. 그 예시들은 다음과 같이 코드 예제를 첨부하여 설명한다.

1) UI_Dialog_01 에서 UI_Dialog_02 의 객체 pi 를 선언하여 UI_Dialog_02 와 연결을 한다.

```python
from PlayerInfoScreen import UI_Dialog_02 as pi    # 띄워진 창을 바꾸기 위해 두번째 씬인 PlayerInfoScreen을 임포트

class UI_Dialog_01(object):

    def ChangeUI(self):                            # 한 윈도우창에서 다른 윈도우창을 띄울 때 사용할 함수
        self.app = QtWidgets.QApplication(sys.argv)        #윈도우 창을 띄우기 위함
        self.Dialog = QtWidgets.QDialog()                  # Dialog(윈도우 창이라고 생각하면 됨)을 사용하기 위함
        t = pi()                                           # PlayerInfoScreen 객체 선언
        self.ui = t                                        # ui에 해당 클래스(PlayerInfoScreen의 UI_Dialog_02)를 연결
        self.ui.setupUi(self.Dialog) # 해당 ui의 setupUi함수를 실행 -
        #Dialog.hide()                                     # 현재 창을 지움 -> 이건 현재 클래스에서 프로그램을 돌릴 때는 실행되는데, 없으면 실행이 안되어서 막아둠(Controller에서 돌릴 것이기 때문)
        self.Dialog.show()                                 # 새로운 창을 띄움
```

2) UI_Dialog_02 에서 UI_Dialog_03 의 gp 객체를 생성하고 인스턴스를 보낸다.

```
from GamePlayingScreen import UI_Dialog_03 as gp  # GamePlayingScreen을 임포트

class UI_Dialog_02(object):

    def ChangeUI2(self):  # 한 UI에서 다른 UI띄울 때 사용할 함수
        self.app = QtWidgets.QApplication(sys.argv)  # 윈도우 창을 띄우기 위함
        self.Dialog = QtWidgets.QDialog()  # Dialog(윈도우 창이라고 생각하면 됨)을 사용하기 위함
        a = gp(self.PlayerNum.value(), self.Player1_PieceNum.value(), self.lineEdit.text(), self.lineEdit_2.text(),
               self.lineEdit_3.text(), self.lineEdit_4.text())  # GamePlayingScreen 객체 선언
        self.ui = a  # ui에 해당 클래스( GamePlayingScreend의 UI_Dialog_023)를 연결
        self.ui.setupUi(self.Dialog)  # 해당 ui의 setupUi함수를 실행 -
        # Dialog.hide()
        self.Dialog.show()
```

3) UI_Dialog_03 에서 Controller 객체 controller 생성을 하고 인스턴스를 보낸다.

```
import Controller
import sys

class UI_Dialog_03(object):

    # 이 아래의 함수 3개와 temp_list는 만들다가 못 만든 것...T^T 지우셔도 됩니다

    # UI 설정
    def __init__(self, n_player, n_piece, name1, name2, name3, name4):
        self.player_name = [name1, name2, name3, name4]
        self.controller = Controller.Contoller(n_player, n_piece)
        self.current_turn = 0
        self.next_turn = 0
        self.remainedpiece = [n_piece, n_piece, n_piece, n_piece]
        self.n_piece = n_piece
        self.action = 0
```

4) Controller의 인스턴스를 UI_Dialog_03가 참조하여 업데이트 할 수 있도록 한다.

```
import Controller
import sys

class UI_Dialog_03(object):

    # 이 아래의 함수 3개와 temp_list는 만들다가 못 만든 것...T^T 지우셔도 됩니다

    # UI 설정
    def __init__(self, n_player, n_piece, name1, name2, name3, name4):
        self.player_name = [name1, name2, name3, name4]
        self.controller = Controller.Contoller(n_player, n_piece)
        self.current_turn = 0
        self.next_turn = 0
        self.remainedpiece = [n_piece, n_piece, n_piece, n_piece]
        self.n_piece = n_piece
        self.action = 0
```

5) Controller는 모델의 map의 객체를 생성하여 map의 인스턴스를 참조한다.

```
class Contoller():

    def __init__(self, n_player, n_piece):
        self.map = ymap.Map(n_player, n_piece)
        self.n_roll = 0
        self.next_turn = 0


    def map_clicked(self, map_index, generate=False):
        if self.n_roll != 0:
            print(self.n_roll, map_index, self.next_turn)
            self.map_info, self.remained_piece, self.next_turn, self.in_goal, self.winner = self.map.select(self.n_roll,
                                                                                                             map_index,
                                                                                                             self.next_turn,
                                                                                                             generate)
            self.n_roll = 0
            return self.map_info, self.remained_piece, self.next_turn, self.in_goal, self.winner
```

4. Polymorphism(다형성)

Polymorphism(다형성)은 프로그램에서 하나의 클래스나 함수가 다양한 방식으로 동작이 가능한 것을 말합니다.

Python 언어 특성상 타입 선언이 별도로 필요 없다. Python의 함수는 여러 가지 방법으로 작동이 가능하기 때문에 다형성이 지켜진다.

5. Abstraction(추상화) / Interface(인터페이스) / Polymorphism(다형성)

Abstraction(추상화)은 기존의 클래스에서 공통된 부분을 추상화하여 상속하는 클래스에게 구현을 강제화하는 것이다. 메소드의 동작을 구현하는 자식클래스로 책임을 위임을 한다. 공유의 목적으로 추상화를 한다.

Interface(인터페이스)는 인터페이스를 상속받는 클래스에서는 반드시 인터페이스에 있는 메소드를 다 구현해야 한다.

추상화 또는 인터페이스를 사용하게 되면, 내부에 자주 사용하는 매소드들이 있어야 한다. 하지만 각각의 말판이 variable 의 역할만을 하고 어떠한 객체 안의 메소드도 필요로 하지 않기 때문에 클래스를 쓰는 것이 오히려 비효율적이라 생각하여 사용하지 않았다.

5. Program Screen Shot

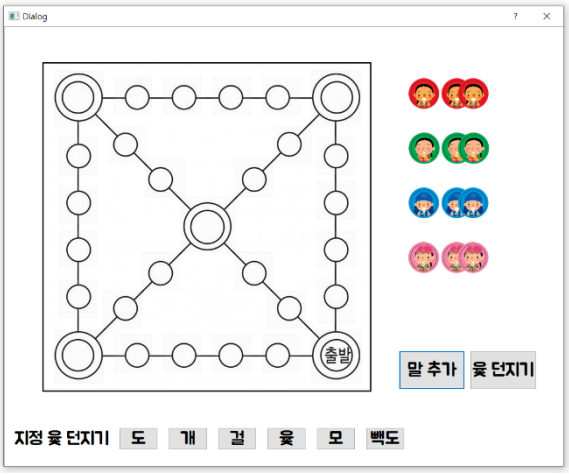
1. 게임 시작 화면

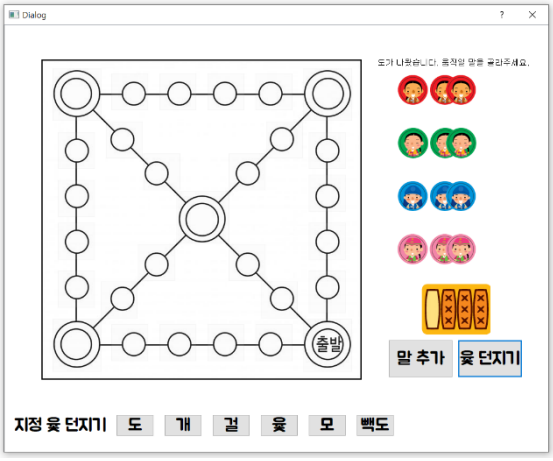
프로그램을 시작하면 가장 먼저 떠오르는 화면이다. ‘START’버튼을 누르면 플레이어의 정보를 입력하는 화면으로 전환되고, ‘EXIT’버튼을 누르면 프로그램이 종료된다.

2. 플레이어의 정보 입력

플레이어의 정보를 입력할 수 있는 화면으로 플레이어의 명수 및 말의 개수를 선택한 후 각각의 플레이어의 이름을 입력할 수 있다. 2~4 까지 플레이가 가능하고, 플레이어당 말의 개수는 2~5 개까지 선택 가능하다. 'Game Start'버튼을 누르면 윷놀이 게임을 플레이하는 화면으로 전환되고, 'EXIT'버튼을 누르면 프로그램이 종료된다.

3. 윷 던지기

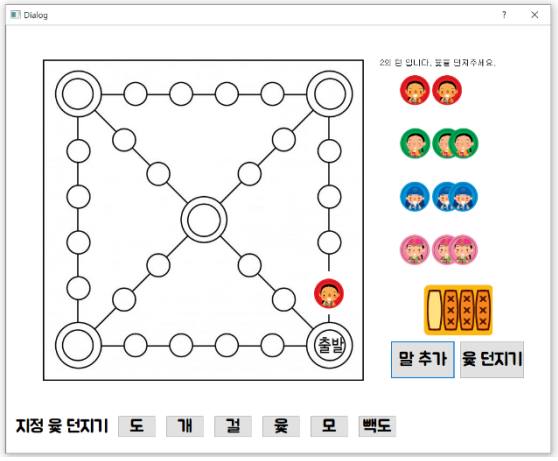




플레이어 정보를 입력한 후 윷놀이 게임이 시작되면 플레이어는 '윷 던지기'버튼을 눌러서 윷을 던질 수 있다. 첫 번째 플레이어(빨간색 말), 두 번째 플레이어(초록색 말), 세 번째 플레이어(파란색 말), 네 번째 플레이어(분홍색 말)까지 있으며 첫 번째 플레이어부터 차례대로 윷을 던진다. 윷이 던져지면 윷 던지기 버튼 위에 윷의 결과 그림이 떠오르고, 화면의 오른쪽 윗부분에 결과값이 써진다.

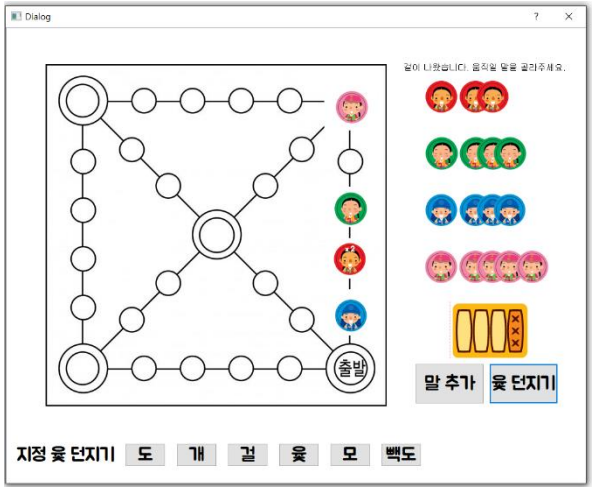
4. 말 이동하기

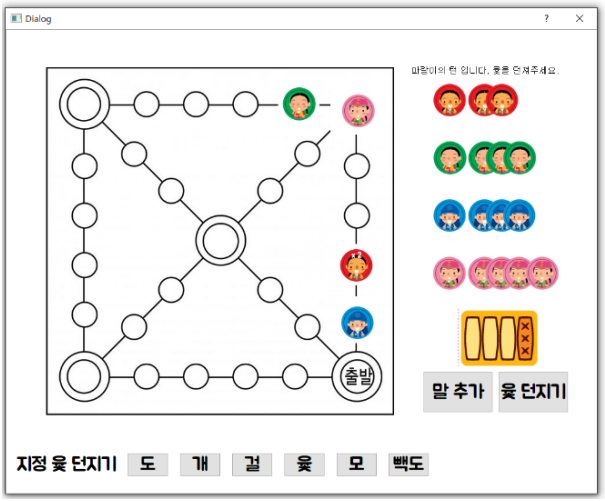
4.1 새로운 말 추가



플레이어는 윷을 던진 후에 결과값을 보고 '말 추가'버튼을 눌러 새로운 말을 윷놀이 보드판 위에 추가할 수 있다. 이 '말 추가' 버튼은 플레이어 정보 입력 단계에서 입력받은 말의 개수까지 추가할 수 있으며, 그 이상 추가하는 것은 불가능하다. 추가된 말은 플레이어가 던진 윷의 결과에 따라 도,개,걸,윷,모의 규칙에 따라 이동한다. 단, 빽도가 나온 경우 새로운 말을 추가하는 것은 불가능하며 이미 보드판 위에 추가된 기존의 말만 움직일 수 있다. 사진은 던져진 윷의 결과 '도'에 따라 빨간색 말이 한 칸 움직인 것이다.

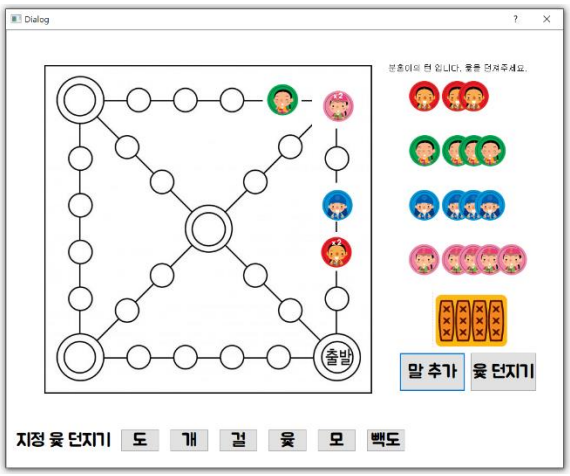
4.2 기존 말 이동하기

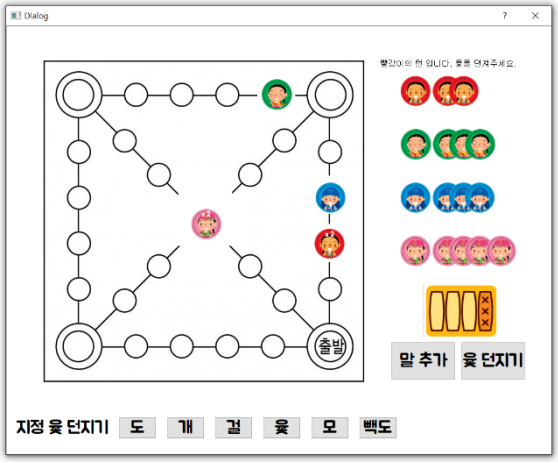




플레이어는 윷을 던진 후 보드판 위에 있는 말을 선택하여 던진 윷의 결과에 따라 해당 말을 이동할 수 있다. 사진은 기존에 있던 초록색 말이 던져진 윷의 결과 '걸'에 따라 3 칸을 이동한 것이다.

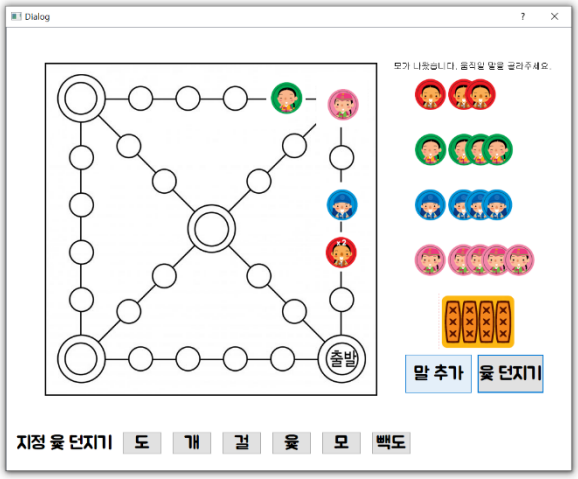
4.3 모서리에서의 말 이동

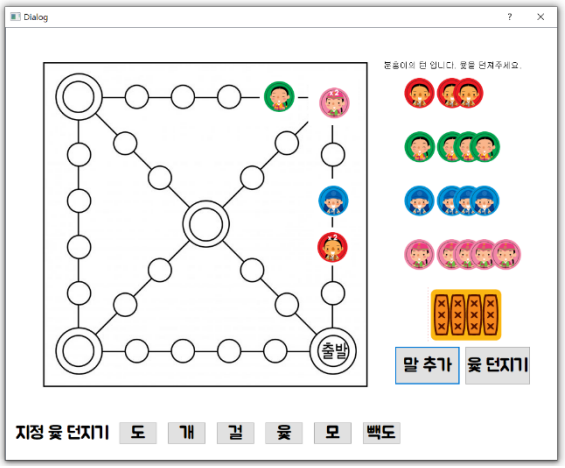




모서리에서는 말들이 기본적으로 골인 지점(원점) 방향을 향해 움직인다. 사진에서는 분홍색 말이 던져진 윷의 결과인 '걸'에 따라 대각선으로 3 칸 움직였다.

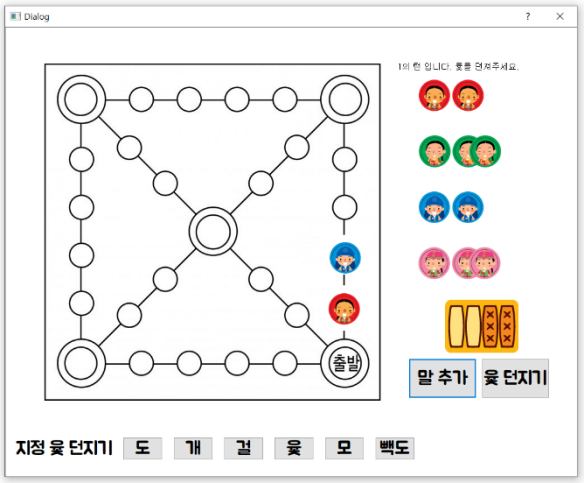
5. 같은 팀의 말 업기

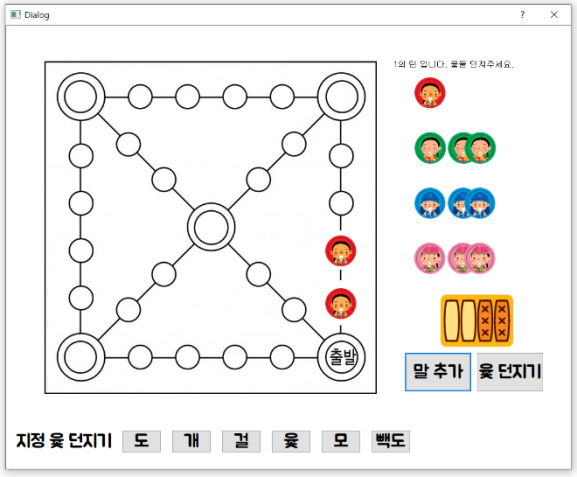




만약 플레이어가 이동시킨 말의 위치에 자신의 말이 있는 경우 말을 업을 수 있다. 말이 업힌 개수대로 말판 위에 숫자가 바뀌고(예 : 1 개의 말이 없힌 경우 x1 , 2 개의 말이 업힌 경우 x2) 이후 이 말들은 함께 움직이게 된다. 사진은 분홍색 말이 던져진 윷의 결과인 '모'에 따라 시작 위치로부터 5 칸 이동하여 기존에 이미 위치한 분홍색 말을 업었다. (업힌 후 분홍색 말 위에 x2 글씨가 생긴다.)

6. 상대 팀의 말 잡기





만약 플레이어가 이동시킨 말의 위치에 다른 사람의 말이 있는 경우 상대의 말을 잡을 수 있다. 잡힌 말은 시작점으로 다시 돌아가며 만약 잡힌 말에 여러 말들이 업혀있는 경우 이 역시 전부 시작점으로 돌아간다.

6. Unit Test Screen Shot

유닛 테스트(unit test)는 컴퓨터 프로그래밍에서 소스 코드의 특정 모듈이 의도된 대로 정확히 작동하는지 확인하는 절차이다. 즉, 모든 함수와 메소드에 대한 테스트 케이스(Test case)를 작성하고 이를 실행해 작동여부와 결과를 확인하는 것이다. 그렇기 때문에 코드가 변경되어 문제가 생기거나 의도치 않은 조작이 발생할 경우, 빠른 시간 내에 이를 파악하고 수정하여 올바르게 작동할 수 있도록 하는 것이다. 이상적인 유닛 테스트에서는 각 테스트 케이스가 서로 분리되어 확인하는 동안 서로 영향을 미치지 않아야 한다. 유닛 테스트에는 추가 프로그래밍이 많고, Test case(Test suite)를 구축하는데 많은 시간이 소요된다는 단점이 있지만 이 후, 프로그램의 버그를 줄이고, 유지보수 및 수정을 더 쉽게 만들어준다.

여기서는 TestMap 에서 Map 정보를 나타내는 ymap class 를 상속받아서 ymap 의 element 를 가져오고 modify() 메소드를 추가해 element 를 수정할 수 있도록 했다. Unit Test 를 하기 위해 assertEqual 이라는 함수를 정의하고 x, y 를 매개변수로 준다. 이 x 와 y 를 비교해 다르면 에러메시지를 보여주고, 아니라면 문제없이 실행된다.

크게 갈림길에서 올바른 길로 가는지, 말이 도착지점에 도착하면 정보가 올바르게 변경이 되는지, 도착지점에 상대편 말이 있다면 말을 잡고 하위 이벤트들이 올바르게 발생하는지,

빽도가 나와 뒤로 한 칸 이동할 때 올바른 위치로 가는지 등에 대해서 유닛 테스트를 진행했다.

1. 5 번 갈림길에서 윷을 던졌을 때, 제대로 말이 도착하는지 확인하는 Test Case

ts.parting_path1(1,1,1)를 실행하면, tamp 을 말판에서 5 번에 위치시키고 매개변수로 받은 유저 정보와 말 정보를 수정한다. 테스트가 문제없이 실행되었고, 윷 결과에 따라 위치를 수정하고 좌측에서 적절하게 이동 된 것을 볼 수 있다.

```
def parting_path1(self, user_info, n_roll, n_piece):
    """
    5번 갈림길에서 윷판을 던졌을 때, 제대로 말이 도착하는지 확인.
    """
    tmap = TestMap()
    tmap.modify(5, user_info, n_piece)
    map_result , _ , _ , _ = tmap.select(n_roll, 5, user_info)
    if n_roll != 3:
        if n_roll in [4,5]:
            n_roll = n_roll - 1
        self.assertEqual(map_result[19+n_roll][2], user_info)
        self.assertEqual(map_result[19+n_roll][3], n_piece)
    elif n_roll == 3:
        self.assertEqual(map_result[28][2], user_info)
        self.assertEqual(map_result[28][3], n_piece)
    print(map_result)
```

```
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', 1, 1],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]

In [15]:
```

2. 10 번 갈림길에서 윷을 던졌을 때, 제대로 말이 도착하는지 확인하는 Test Case

ts.parting_path2(1,2,1) 를 실행하면, tamp 을 말판에서 10 번에 위치시키고 매개변수로 받은 유저 정보와 말 정보를 수정한다. 테스트가 문제없이 실행되었고, 윷 결과에 따라 위치를 수정하고 좌측에서 적절하게 이동 된 것을 볼 수 있다

```
def parting_path2(self, user_info, n_roll, n_piece):
    """
    10번 갈림길에서 윷판을 던졌을 때, 제대로 말이 도착하는지 확인.
    """
    tmap = TestMap()
    tmap.modify(10, user_info, n_piece)
    map_result , _ , _ , _ = tmap.select(n_roll, 10, user_info)
    if n_roll != 3:
        if n_roll in [4,5]:
            n_roll = n_roll - 1
        self.assertEqual(map_result[23+n_roll][2], user_info)
        self.assertEqual(map_result[23+n_roll][3], n_piece)
    elif n_roll == 3:
        self.assertEqual(map_result[28][2], user_info)
        self.assertEqual(map_result[28][3], n_piece)
    print(map_result)
```

```
In [16]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', 1, 1], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]

In [17]:
```

3. 22 번에서 15 번을 넘어가는 갈림길에서, 제대로 말이 도착하는지 확인하는 Test Case

ts.parting_path3(1,3,1)를 실행하면, tamp 을 말판에서 22 번에 위치시키고 매개변수로 받은 유저 정보와 말 정보를 수정한다. 테스트가 문제없이 실행되었고, 윷 결과에 따라 위치를 수정하고 좌측에서 3 칸이동 된 것을 볼 수 있다

```
def parting_path3(self, user_info, n_roll, n_piece):
    """
    22번에서 15번을 넘어가는 갈림길에서, 제대로 말이 도착하는지 확인.
    """
    tmap = TestMap()
    tmap.modify(22, user_info, n_piece)
    map_result , _ , _ , _ = tmap.select(n_roll, 22, user_info)
    if n_roll == 1:
        self.assertEqual(map_result[23][2], user_info)
        self.assertEqual(map_result[23][3], n_piece)
    else:
        self.assertEqual(map_result[13 + n_roll][2], user_info)
        self.assertEqual(map_result[13 + n_roll][3], n_piece)
    print(map_result)
```

```
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, 1, 1], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]

In [18]:
```

4. 28 번에 있을 때, 제대로 말이 도착하는지 확인하는 Test Case

```
def parting_path4(self, user_info, n_roll, n_piece):
    """
    28번에 있을 때, 제대로 말이 도착하는지 확인.
    """
    tmap = TestMap()
    tmap.modify(28, user_info, n_piece)
    map_result , _ , _ , in_goal = tmap.select(n_roll, 28, user_info)
    if n_roll < 3:
        self.assertEqual(map_result[25 + n_roll][2], user_info)
        self.assertEqual(map_result[25 + n_roll][3], n_piece)
    if n_roll == 3:
        self.assertEqual(map_result[0][2], user_info)
        self.assertEqual(map_result[0][3], n_piece)
    else:
        self.assertEqual(in_goal[user_info], n_piece)
        self.assertEqual(in_goal[user_info], n_piece)
    print(map_result)
    print(in_goal)
```

ts.parting_path4(1,3,2)를 실행하면 말이 도착지점에 도착만하고 골인하지 않는 상황이기 때문에, 말은 0 번 위치에 위치하게 되고 골인한 말의 개수를 나타내는 배열에서는 정보가 수정되지 않은 것을 확인할 수 있다.

```
In [18]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, '27', 1, 2], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]
[0, 0, 0, 0]
```

하지만 ts.parting_path4(1,4,2)를 실행하게 되면 말이 도착지점을 지나 골인을 하게 되는 상황이다. 골인한 말의 개수를 나타내는 배열에서 도착한 유저의 인덱스의 숫자가 도착한 말의 개수만큼 변경된 것을 확인 할 수 있다.

```
In [20]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]
[0, 2, 0, 0]
```

5. goal 에 윷판이 제대로 들어가는지 확인하는 Test Case

이 테스트 케이스에서 assertEqual 메소드로 비교했을 때, 문제 없이 실행된다.

```
def is_goal(self, user_info, n_piece):
    """
    goal에 윷판이 제대로 들어가는지 확인.
    """
    tmap = TestMap()
    tmap.modify(0, user_info, n_piece)
    self.assertEqual(tmap.select(1, 0, user_info)[3][user_info], n_piece)
```

```
In [21]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap

In [22]:
```

6. 말이 상대방의 윷판을 제대로 잡는지, 말을 잡고 윷을 한번 더 던지는지, 잡힌 상대방 말은 시작점으로 돌아가는지 확인하는 Test Case

이 테스트 케이스에서 assertEqual 메소드로 4 개의 변수를 비교했을 때, 에러가 발생하지 않고 문제 없이 실행된다.

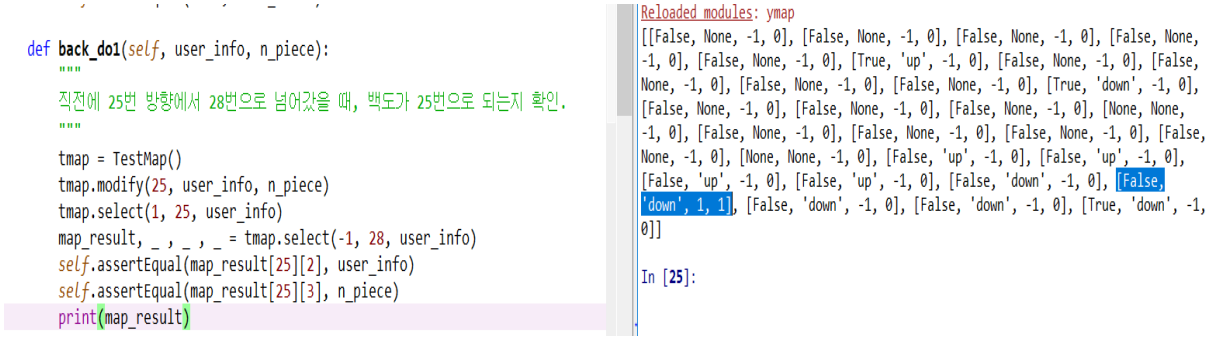
```
def catch(self, user_info1, n_piece1, user_info2, n_piece2):
    """
    말이 상대방의 윷판을 제대로 잡는지 확인.
    말을 잡고 윷을 한번 더 던지는지 확인.
    잡힌 상대방 말은 시작점으로 돌아가는지 확인.
    """
    tmap = TestMap()
    tmap.modify(5, user_info2, n_piece2)
    tmap.modify(4, user_info1, n_piece1)
    map_result, remained, turn, _ = tmap.select(1, 4, user_info1)
    self.assertEqual(map_result[5][2], user_info1)
    self.assertEqual(map_result[5][3], n_piece1)
    self.assertEqual(remained[user_info2], 4)
    self.assertEqual(turn, user_info1)
```

```
In [22]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap

In [23]:
```

7. 직전에 25 번 방향에서 28 번으로 넘어갔을 때, 백도가 25 번으로 되는지 확인하는 Test Case

ts.back_do1(1,1)을 실행했을 때, 말의 인덱스를 25 로 변경하고, assertEqual 함수를 통해서 비교한다. Map_result 를 print 했을 때 올바르게 25 번 인덱스가 변경된 것을 확인할 수 있다.



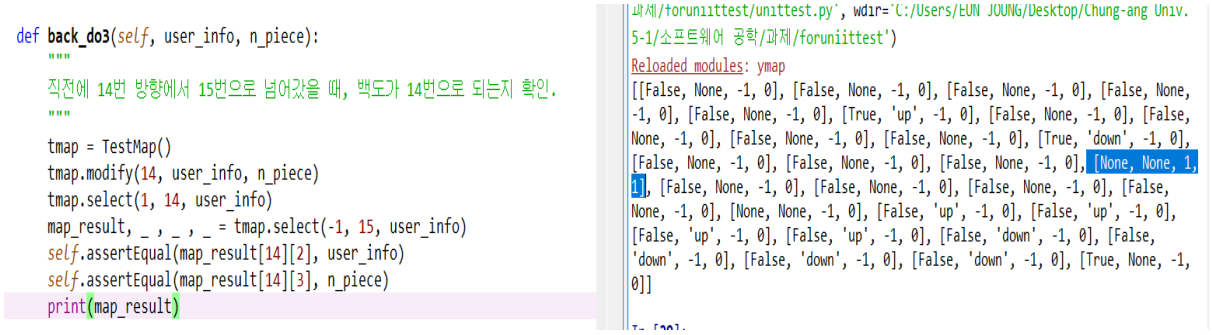
```
def back_do1(self, user_info, n_piece):
    """
    직전에 25번 방향에서 28번으로 넘어갔을 때, 백도가 25번으로 되는지 확인.
    """
    tmap = TestMap()
    tmap.modify(25, user_info, n_piece)
    tmap.select(1, 25, user_info)
    map_result, _ , _ , _ = tmap.select(-1, 28, user_info)
    self.assertEqual(map_result[25][2], user_info)
    self.assertEqual(map_result[25][3], n_piece)
    print(map_result)
```

```
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', 1, 1], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, 'down', -1,
0]]

In [25]:
```

8. 직전에 14 번 방향에서 15 번으로 넘어갔을 때, 백도가 14 번으로 되는지 확인하는 Test Case

back_do3(1,1)을 실행했을 때, 말의 인덱스를 14 로 변경하고 15 번으로 넘어간 상황을 만들어준다. assertEqual 함수를 통해서 비교한 후, Map_result 를 print 했을 때 올바르게 14 번 인덱스가 변경된 것을 확인 할 수 있다.



```
def back_do3(self, user_info, n_piece):
    """
    직전에 14번 방향에서 15번으로 넘어갔을 때, 백도가 14번으로 되는지 확인.
    """
    tmap = TestMap()
    tmap.modify(14, user_info, n_piece)
    tmap.select(1, 14, user_info)
    map_result, _ , _ , _ = tmap.select(-1, 15, user_info)
    self.assertEqual(map_result[14][2], user_info)
    self.assertEqual(map_result[14][3], n_piece)
    print(map_result)
```

```
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None, 1,
1], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]
```

9. 직전에 23 번 방향에서 15 번으로 넘어갔을 때, 백도가 23 번으로 되는지 확인하는 Test Case

back_do4(1,2)을 실행했을 때, 말의 인덱스를 23 으로 변경하고 15 번으로 넘어간 상황을 만들어준다. assertEqual 함수를 통해서 비교한 후, Map_result 를 print 했을 때 올바르게 23 번 인덱스가 변경된 것을 확인 할 수 있다.

```
def back_do4(self, user_info, n_piece):
    """
    직전에 23번 방향에서 15번으로 넘어갔을 때, 백도가 23번으로 되는지 확인.
    """
    tmap = TestMap()
    tmap.modify(23, user_info, n_piece)
    tmap.select(1, 23, user_info)
    map_result, _ , _ , _ = tmap.select(-1, 15, user_info)
    self.assertEqual(map_result[23][2], user_info)
    self.assertEqual(map_result[23][3], n_piece)
    print(map_result)
```

```
In [30]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, 'up', -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', 1, 2], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]
```

10. 24 번에서 백도를 했을 때, 10 번으로 돌아가는지 확인하는 Test Case

back_do5(1,2)을 실행했을 때, 말의 인덱스를 24 로 변경하고 윷 결과를 빽도인 -1 로 준다. assertEqual 함수를 통해서 비교했을 때, 에러가 발생하지 않고, Map_result를 print한 결과 올바르게 10 번 인덱스가 변경된 것을 확인 할 수 있다.

```
def back_do5(self, user_info, n_piece):
    """
    24번에서 백도를 했을 때, 10번으로 돌아가는지 확인.
    """
    tmap = TestMap()
    tmap.modify(24, user_info, n_piece)
    map_result, _ , _ , _ = tmap.select(-1, 24, user_info)
    self.assertEqual(map_result[10][2], user_info)
    self.assertEqual(map_result[10][3], n_piece)
    print(map_result)
```

```
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', 1, 2],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]
```

11. 20 번에서 백도를 했을 때, 5 번으로 돌아가는지 확인하는 Test Case

back_do6(1,2)을 실행했을 때, 말의 인덱스를 20 로 변경하고 윷 결과를 빽도인 -1 로 준다. assertEqual 함수를 통해서 비교한 후, Map_result 를 print 했을 때 올바르게 5 번 인덱스가 변경된 것을 확인 할 수 있다.

```
def back_do6(self, user_info, n_piece):
    """
    20번에서 백도를 했을 때, 5번으로 돌아가는지 확인.
    """
    tmap = TestMap()
    tmap.modify(20, user_info, n_piece)
    map_result, _ , _ , _ = tmap.select(-1, 20, user_info)
    self.assertEqual(map_result[5][2], user_info)
    self.assertEqual(map_result[5][3], n_piece)
    print(map_result)
```

```
과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ.
5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None,
-1, 0], [False, None, -1, 0], [True, 'up', 1, 2], [False, None, -1, 0], [False,
None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0],
[False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None,
-1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False,
None, -1, 0], [None, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0],
[False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False,
'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1,
0]]
```

12. 직전에 19 번 방향에서 0 번으로 넘어갔을 때, 백도가 19 번으로 되는지 확인하는 Test Case

back_do7(1,2)을 실행했을 때, 말의 인덱스를 19 로 변경하고 윷 결과를 빽도인 -1 로 준다. assertEqual 함수를 통해서 비교했을 때, 에러가 발생하지 않고, Map_result 를 print 한 결과 올바르게 10 번 인덱스가 변경된 것을 확인 할 수 있다.

```
def back_do7(self, user_info, n_piece):
    """
    직전에 19번 방향에서 0번으로 넘어갔을 때, 백도가 19번으로 되는지 확인.
    """
    tmap = TestMap()
    tmap.modify(19, user_info, n_piece)
    tmap.select(1, 19, user_info)
    map_result, _ , _ , _ = tmap.select(-1, 0, user_info)
    self.assertEqual(map_result[19][2], user_info)
    self.assertEqual(map_result[19][3], n_piece)
    print(map_result)
```

```
In [35]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, '19', -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None, 1, 2], [False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [True, None, -1, 0]]
```

13. 직전에 27 번 방향에서 0 번으로 넘어갔을 때, 백도가 27 번으로 되는지 확인하는 Test Case

back_do8(1,2)을 실행했을 때, 말의 인덱스를 27 로 변경하고 윷 결과를 빽도인 -1 로 준다. assertEqual 함수를 통해서 비교했을 때, 에러가 발생하지 않고, Map_result 를 print 한 결과 올바르게 0 번 인덱스가 변경된 것을 확인 할 수 있다.

```
    def back_do8(self, user_info, n_piece):
        """
        직전에 27번 방향에서 0번으로 넘어갔을 때, 백도가 27번으로 되는지 확인.
        """
        tmap = TestMap()
        tmap.modify(27, user_info, n_piece)
        tmap.select(1, 27, user_info)
        map_result, _ , _ , _ = tmap.select(-1, 0, user_info)
        self.assertEqual(map_result[27][2], user_info)
        self.assertEqual(map_result[27][3], n_piece)
        print(map_result)


ts = TestLogic()

ts.back_do8(1,2)
```

```
In [36]: runfile('C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/과제/foruniittest/unittest.py', wdir='C:/Users/EUN JOUNG/Desktop/Chung-ang Univ. 5-1/소프트웨어 공학/과제/foruniittest')
Reloaded modules: ymap
[[False, '27', -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'up', -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [True, 'down', -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [None, None, -1, 0], [False, None, -1, 0], [False, None, -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'up', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [False, 'down', -1, 0], [False, 'down', 1, 2], [True, None, -1, 0]]

In [37]:
```

7. Project Management Report

1) Github Project Repository

https://github.com/josephy93/SWE-2019-1

2) Project Progress History

Date	Project Progress
5/1	- Github Project Repository 생성
5/8	- 윷놀이 게임 비전 및 UseCase Diagram과 User Scenario 작성 - System Sequence Diagram을 작성 - Programming Language를 Python으로 결정
5/15	- Domain Model을 작성 - Sequence Diagram을 완성 - Major Design Decisions을 간단하게 작성 - MVC Model을 사용하여 분석 - UseCase에서 정의한 기능들을 토대로 Python으로 Model 개발 시작 - PyQT를 사용해 View를 구현하기로 결정
5/22	- PyQT로 시작화면과 플레이어 설정화면을 완성 - Model에서 의도한 규칙에 맞게 작동하는지 Algorithm을 확인하고 이를 수정 (Select) - Operation Contract를 추가해 기존에 작성하던 UseCase Model Document를 완성.
5/29	- 버튼 클릭으로 화면전환 구현 및 PyQT로 게임 플레이 화면을 완성 (UI Design complete) - Controller 구현 시작 - Model에서 빽도 관련 부분 에러 수정 - MVC 모델 분석 수정 - Class Diagram작성
6/3	- View Button를 Controller의 기능과 연결 - Model 및 Controller 완성 (Algorithm & Logic Complete) - Controller 와 Model에서 발생하는 에러 해결 - Model, View, Controller를 통합 - Sequence Diagram 작성 - OOAD 문서화
6/5	- Unit Test 작성 및 실행 - Layered Architecture 작성

6/7	- 문서화 작업을 완료 - 프로그램 구현 완료
6/8	- Program 및 Report 제출
6/10	- Presentation & Feedback
6/11	- Final Demonstration

3) Experience & Realization

이번 프로젝트를 하면서 실제로 현업에서 어떤 식으로 프로젝트가 진행되는지에 관해 알 수 있었다. 또한, 처음 설계했던 소프트웨어 구조가 개발을 하면서 변동 될 수 있기 때문에 개발 상황이나 프로젝트에 따라 잘 설계 되어야 함을 느꼈다.

Use case Model 을 구현하면서 전체적인 시스템을 파악할 수 있었고, 어떤 기능을 구현해야 하는지 알 수 있었다. 이를 통해, 개발에 있어서 좀 더 체계적인 접근이 가능하게 되었다. 또한, MVC 를 개발하면서, Model 과 View 를 분리하여 Controller 을 통해 제어함으로써 더 효율적으로 프로젝트를 관리할 수 있었다.

그리고 unit test 를 적용하여 Unit test 의 사용으로 프로그램의 코드의 안정성을 보장하고 유지보수를 할 수 있다는 것을 이해할 수 있었다. 그에 맞는 프로그램을 설계하는 과정을 통해 테스트 기반 프로그래밍 방법의 장점을 알게 되었다. 실제로 프로젝트를 완성하고, 의도한 대로 프로그램이 잘 작동하게 되어 성취감을 느꼈다.