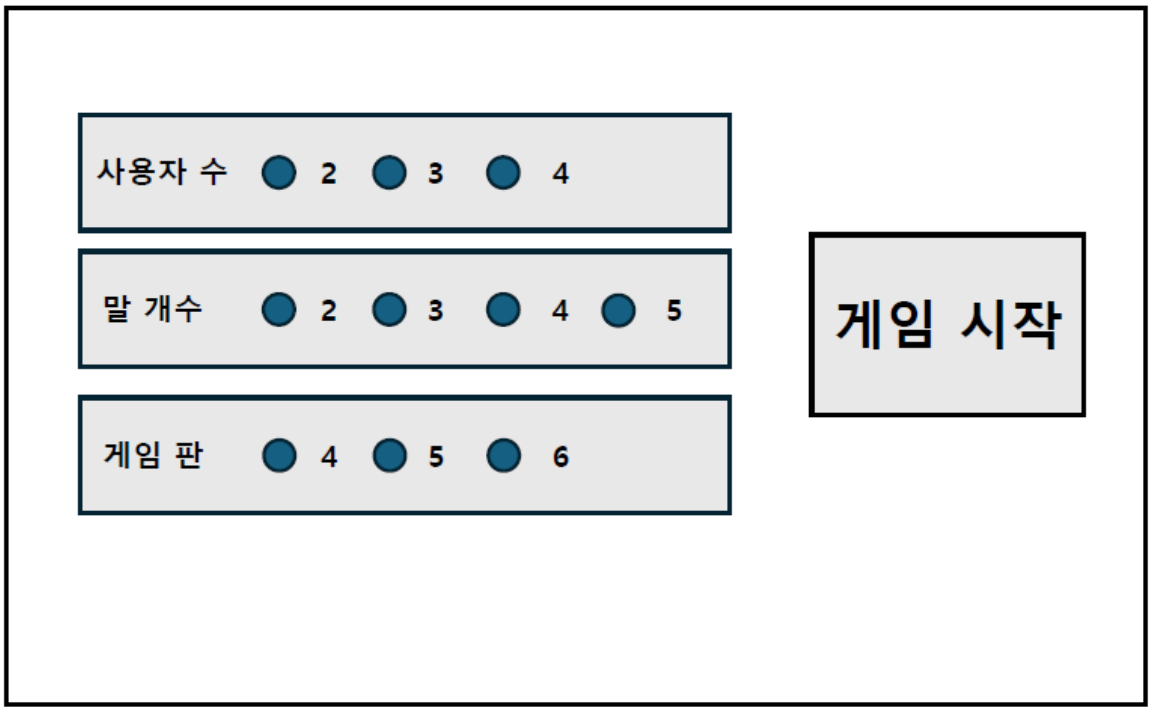
Use Case 개요
(소프트웨어공학 윷놀이 게임)

1) 초기 화면

Precondition: 유저가 Java 프로젝트를 실행

현재 상황: 유저가 Player 수, 말의 개수, Board 모양을 선택하고 '시작'버튼을 눌러야 함.

Postcondition: 유저가 선택한 옵션을 옵션 Handler가 받고 메인 게임 UI의 정보로 넘긴다. 유저는 Game 설정 판을 보게 된다.



2) 메인 화면 (윷놀이 판)

Precondition: 유저가 초기 화면에서 option을 선택해 게임이 시작됨.

현재 상황: 좌측 상단 위에 윷놀이 판이 있고 우측 상단에는 특정 플레이어의 차례, 각 플레이어의 남은 말의 개수가 표시되어 있다. 화면 하단에는 게임을 테스트해보기 위한 옵션형태의 움직임 및 본 게임을 위한 랜덤 선택 버튼이 존재한다. 그 외의 화면에는 진출해 있는 말을 선택할 것인지 아니면 새로운 말을 화면에 추가적으로 배치할 것인지를 선택하는 버튼도 존재한다. '모', '윷'과 같이 보너스를 가지는 말의 경우 추가적인 처리를 위해 우측에 버튼 형태로 존재한다. 유저는 이 버튼을 누름으로써 말의 움직임을 선택할 수 있게 된다.

Postcondition:

1. 유저가 윷을 던져서 결과가 나오게 된다.

2-1. 유저가 던진 윷이 '도', '개' '걸'인 경우 진출한 말 혹은 새로운 말 중에서 나아갈 말을 선택할 수 있다.

2-2. 유저가 던진 윷이 '윷', '모'인 경우 다시 던질 준비를 한다. '1.'의 과정으로 다시 돌아가고 2-1로 분기할 때까지 계속해서 반복한다. 이때 반복하는 동안의 던진 결과는 arraylist에 보관되며 우측 화면의 UI로 표기가 된다.

3-1. 말을 움직일 때 움직일 곳에 말이 없다면 기존의 위치의 칸의 색깔을 다시 흰색으로 바꾼다. 도착할 위치의 경우 현재 플레이어의 색깔 및 말의 개수를 같이 표기한다.

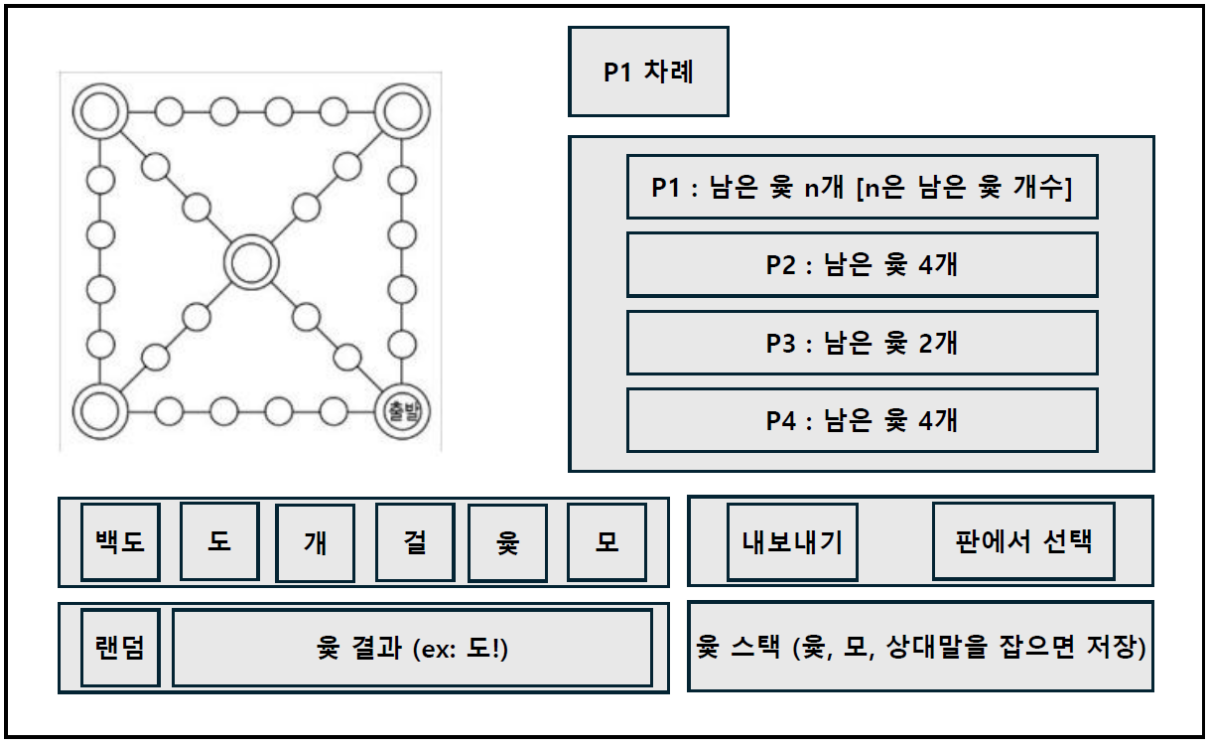
3-2. 말을 움직일 곳에 자신의 말이 존재하는 경우 기존의 위치의 칸의 색깔을 다시 흰색으로 바꾼다. 도착할 위치의 버튼의 숫자를 기존의 위치하던 숫자 + 추가될 숫자를 적용해서 다시 화면에 표현한다.

3-3 말을 움직일 곳에 상대 말이 있는 경우 기존의 위치의 칸의 색깔을 다시 흰색으로 바꾼다. 도착할 위치의 버튼의 색깔을 현재의 플레이어의 색깔로 바꾸고 숫자도 마찬가지로 바꾼다. 다른 사람의 말은 다시 회수되며 우측 상단에 잡힌 말의 플레이어에 대한 정보를 다시 갱신한다. 잡은 사람은 다시 1.의 과정으로 돌아가 윷을 던지게 된다.

3-4. 말을 움직일 곳이 원점 혹은 그 이후를 지나는 경우 기존에 위치하던 버튼의 색깔을 다시 흰색으로 바꾸어 주고 현재 플레이어의 말의 개수를 조정해준다. 만약 조정한 말의 남은 개수가 0개라면 5.의 과정으로 넘어간다.

4. 위의 모든 과정을 반복하고 아무런 추가적인 이벤트가 없다면 현재 플레이어의 순서를 종료하고 다음 플레이어로 턴을 넘긴다.

5. 승자가 나왔다. 엔딩 화면으로 분기한다.

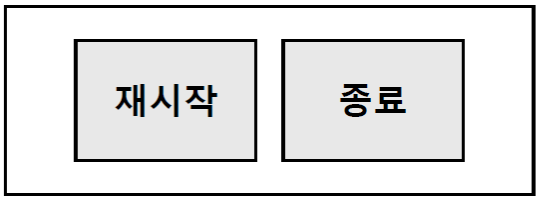


3) 엔딩 화면

Precondition: 특정 유저가 자신이 가진 모든 말을 해방시켰다.

현재 상황: 화면에는 게임을 다시 시작하는 것 혹은 종료하는 것 두 개의 버튼으로 나뉘어 플레이어의 선택을 기다린다.

Postcondition: 만약 유저가 다시하기를 누른다면 1)의 화면으로 다시 넘어가서 위의 과정을 다시 반복하게 된다. 1)의 화면으로 돌아가기전 동적 할당을 했던 객체를 모두 free 해주어야한다(Java의 경우 전역 변수 혹은 static class에서 참조한 instance를 garbage collector가 수거해 가지 않기 때문에 free를 안하고 다시 게임을 돌리는 경우 메모리 가용 용량이 줄어들게 된다). 만약 유저가 종료를 한다면 화면을 닫는다.



- **(a) 재시작 시 게임 시작 UI로 회귀**
- **(b) 종료 시 모든 창을 닫기**