불끈카우

다빈치 sw 공모전

20163128 민지원

20176830 이주희

20173958 양진경

20154098 염승윤

**a. Team Members & Roles**

민지원 : 문서화, 게임 규칙 제작

이주희 : 아이템 제작, 맵 디자인

양진경 : 맵 제작, 게임 규칙 제작

염승윤 : 멘토

**b. Vision**

**Short Introduction to the System**

게임 이름은 <불꺼카우!>로 스마트폰에서 할 수 있는 게임이다. 화재를 진압하고 화재 현장을 대피하는 시뮬레이션 게임으로 실제 건물을 배경으로 하여 대피 요령을 익히는 것을 목표로 한다.

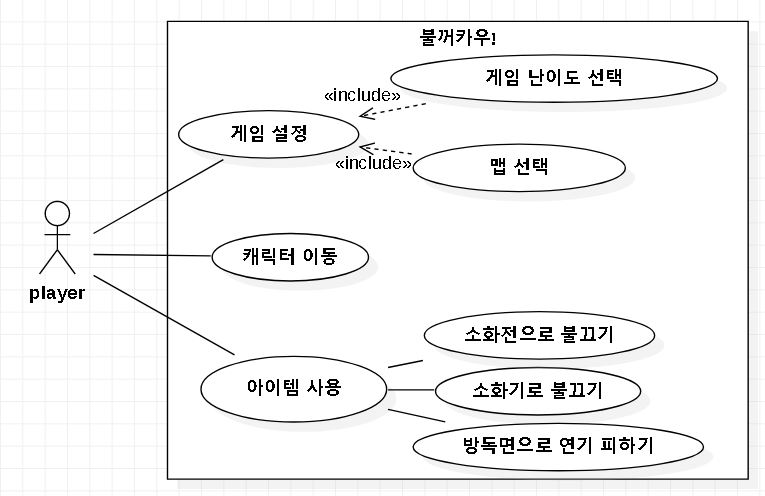
**Main Feature**

캐릭터는 각종 아이템을 사용하여 쓰러진 친구를 구하고 안전하게 건물을 탈출하는 것을 목표로 하고 다음의 세부적인 규칙을 추가한다.

* 소화전은 다수의 불을 끌 수 있다.
* 화재가 난 곳에 진입할 수 없다.
* 연기가 계속 퍼져서 제한 시간내에 탈출해야 한다.
* 소화기 아이템을 사용하면 불을 끌 수 있다.
* 쓰러진 친구와 함께 탈출해야 한다.
* 화제 시 엘리베이터는 이용은 할 수 없다.
* 에스컬레이터는 계단으로 처리한다.
* 튜토리얼에서 탈출 경로를 알려준다. (게임이 아니라 사진으로 탈출구가 어디에 있는지 알려주는 정도로 가능할 거 같아요???)

**c. Use Case Model**

**Use Case Diagrams**



불꺼카우! 게임의 기본적인 기능에는 초기 게임 설정과 캐릭터 이동, 아이템 사용 기능이 있다. 이 세가지 필수 기능을 중점으로 Use Case Diagram을 완성했다.

(탈출 경로를 알려주는 튜토리얼, 랭킹 기능, 게임 일시정지 기능, 게임 현재상태 저장 기능, 맵 만들기 기능 등등을 추가할 수 있을 것 같아요)

**Use Case Description**

**<UC1>**

|  |  |
| --- | --- |
| 유스케이스 명 | 게임 설정 |
| 개 요 | 게임의 난이도와 맵을 선택한다. |
| 관련 액터 | 플레이어 |
| 우선 순위 |  |
| 선행 조건 | 게임이 실행되고 있다. |
| 이벤트 흐름 | 1. 게임 난이도(쉬움, 보통, 어려움)를 선택한다.  2. 맵을 선택한다. |
| 후행 조건 | 1. 선택한 게임 난이도에 따라 탈출 경로 제한, 탈출 제한 시간, 구조해야 하는 친구 명 수가 정해진다.  2. 선택한 맵에 따라 배경이 달라진다. |
| 기타 요구사항 | 난이도가 어려움일지라도 탈출할 수 있다. |

**<UC2>**

|  |  |
| --- | --- |
| 유스케이스명 | 게임 진행 |
| 개 요 | 제한 시간내에 쓰러져 있는 친구를 구하고 탈출해야 한다. |
| 관련 액터 | 플레이어 |
| 우선 순위 |  |
| 선행 조건 | 게임 설정을 완료된 상태 |
| 이벤트 흐름 | **<기본 흐름>**  1. 선택한 맵에 따라 배경이 바뀐다.  2. 캐릭터 위치는 랜덤으로 생성된다.  3. 화제가 발생한 지역은 통과할 수 없다.  4. 연기 때문에 제한 시간 내에 탈출해야 한다.  5. 쓰러진 친구를 발견했다면 구출해야 한다.  6. 1층에 도착하면 탈출할 수 있다.  **<대안 흐름>**  3-1. 소화기 아이템이 있다면 불을 끌 수 있다.  3-2. 불이 꺼진 곳은 지나갈 수 있다.  4-1. 방독면 아이템이 있다면 제한 시간을 늘릴 수 있다.  4-2. 제한 시간내에 탈출하지 못했다면 목표 달성에 실패하고 게임이 종료된다.  5-1. 쓰러진 친구를 무시하고 탈출하면 목표 달성에 실패하고 게임이 종료된다. |
| 후행 조건 | 쓰러진 친구를 구하고 제한 시간 내에 1층에 도착하여 탈출하면 목표 달성과 함께 게임이 종료된다. |
| 기타 요구사항 |  |

**Operation Contract**

1. “start” button click

|  |  |
| --- | --- |
| Operation | Clicking “start” button |
| Cross References | Use Case 게임 설정 |
| Preconditions | 1. Map 객체가 생성된다. 2. Player객체가 생성된다. 3. GameControl 객체가 생성된다. |
| Postconditions | 게임의 맵(배경), 게임의 난이도가 결정된다. |

2. “Joystick” button click

|  |  |
| --- | --- |
| Operation | “캐릭터 이동” button click |
| Cross References | Use case 캐릭터 이동 |
| Preconditions | 게임이 시작됨 |
| Postconditions | 해당 방향으로 캐릭터가 이동한다. |

3. “소화기 사용” button click

|  |  |
| --- | --- |
| Operation | “소화기 사용” button click |
| Cross References | Use case 소화기 사용 |
| Preconditions | 소화기 아이템을 찾아서 인벤토리에 보관하고 있어야한다. |
| Postconditions | 1. 소화기를 사용하면 길을 막고 있는 불이 사라진다. 2. 만약 소화기 개수가 0이면 사용할 수 없다. |

4. “방독면 사용” button click

|  |  |
| --- | --- |
| Operation | “방독면 사용” button click |
| Cross References | Use case 방독면 사용 |
| Preconditions | 방독면 아이템을 찾아서 인벤토리에 보관하고 있어야한다. |
| Postconditions | 1. 방독면을 사용하면 제한 시간을 늘릴 수 있다. 2. 만약 방독면 개수가 0이면 사용할 수 없다. |

5. “소화전 사용” button click

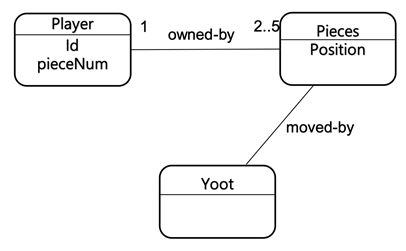
|  |  |
| --- | --- |
| Operation | “소화전 사용” button click |
| Cross References | Use case 소화전 사용 |
| Preconditions | 소화전이 있는 장소에서만 사용이 가능하다. |
| Postconditions | 1. 소화전을 사용할 수 있는 시간이 정해져 있다. 2. 소화전 사용 시간 동안 불을 끌 수 있다. |

5. “X” button click

|  |  |
| --- | --- |
| Operation | Clicking “X” button click button. |
| Cross References | Use Case: Shut down |
| Preconditions |  |
| Postconditions | 생성된 모든 객체가 소멸된다. |

**d. Analysis & Design and Implementation**

**Domain model**



게임 진행에서 **Main Success Scenario**를 생각해보면 다음과 같다.

1. 플레이어는 불을 피해 탈출 지점으로 이동한다.

2. 쓰러진 친구 구하기 이벤트가 있으면 친구를 구해야 한다.

3. 소화기와 방독면을 이용해서 탈출을 더 수월하게 할 수 있다.

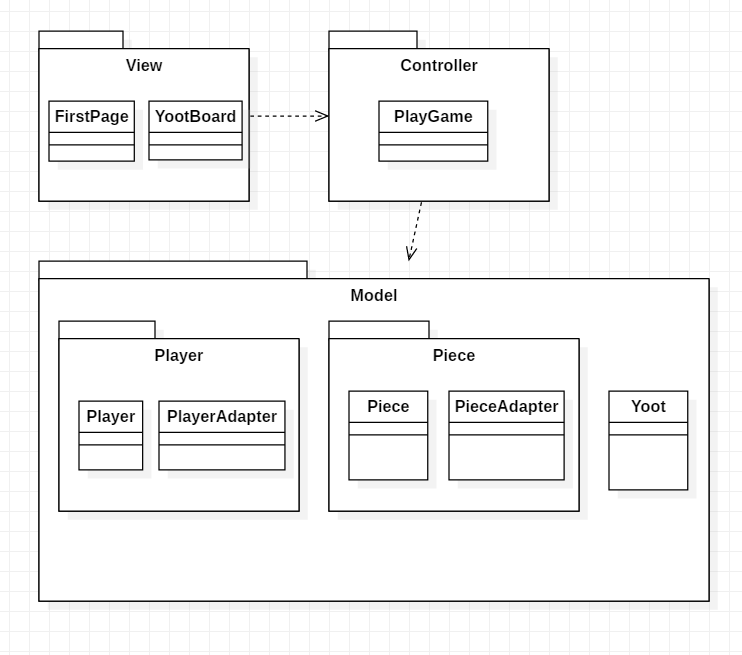
Main Success Scenario에서 핵심이 되는 Noun들을 뽑아내면 플레이어, 윷, 말이 된다. Noun들간의 관계를 고려하면 위의 Domain Model과 같은 관계가 형성된다.

- Player는 2~5개의 Pieces를 소유할 수 있고 각각의 Piece는 하나의 Player에만 속한다.

- Piece는 Yoot이 반환한 값만큼 position을 움직여준다

**Software Architecture + Design Model**

**MVC Model(Architecture)**

****

**(MVC 패턴 구현 방식)**

Model은 Player와 Piece, Yoot으로 나뉘어 있다.

1. Player는 Piece들을 효율적으로 관리하기 위해 Piece 생성, Piece 업혀 있는지 확인, Piece 잡혔는지 확인, Piece 골인 지점으로 들어갔는지 확인하는 게임 규칙과 관련된 기능이 있다. 만약 Player class 없이 Piece만 있었다면 각 Piece 별로 따로 확인을 해야 하지만 Player class로 묶여 있어 Player 객체 하나만 확인하면 된다.
2. Piece는 윷놀이 게임 중에 실제로 움직이는 실체이다. Piece는 윷 말의 위치를 나타내는 x,y 좌표와 얼만큼 업혀 있는지 나타내는 point를 가진다.
3. Yoot은 윷 객체이며 윷을 던지는 기능이 있다. 윷을 던진 결과에 따라 Piece가 움직인다.

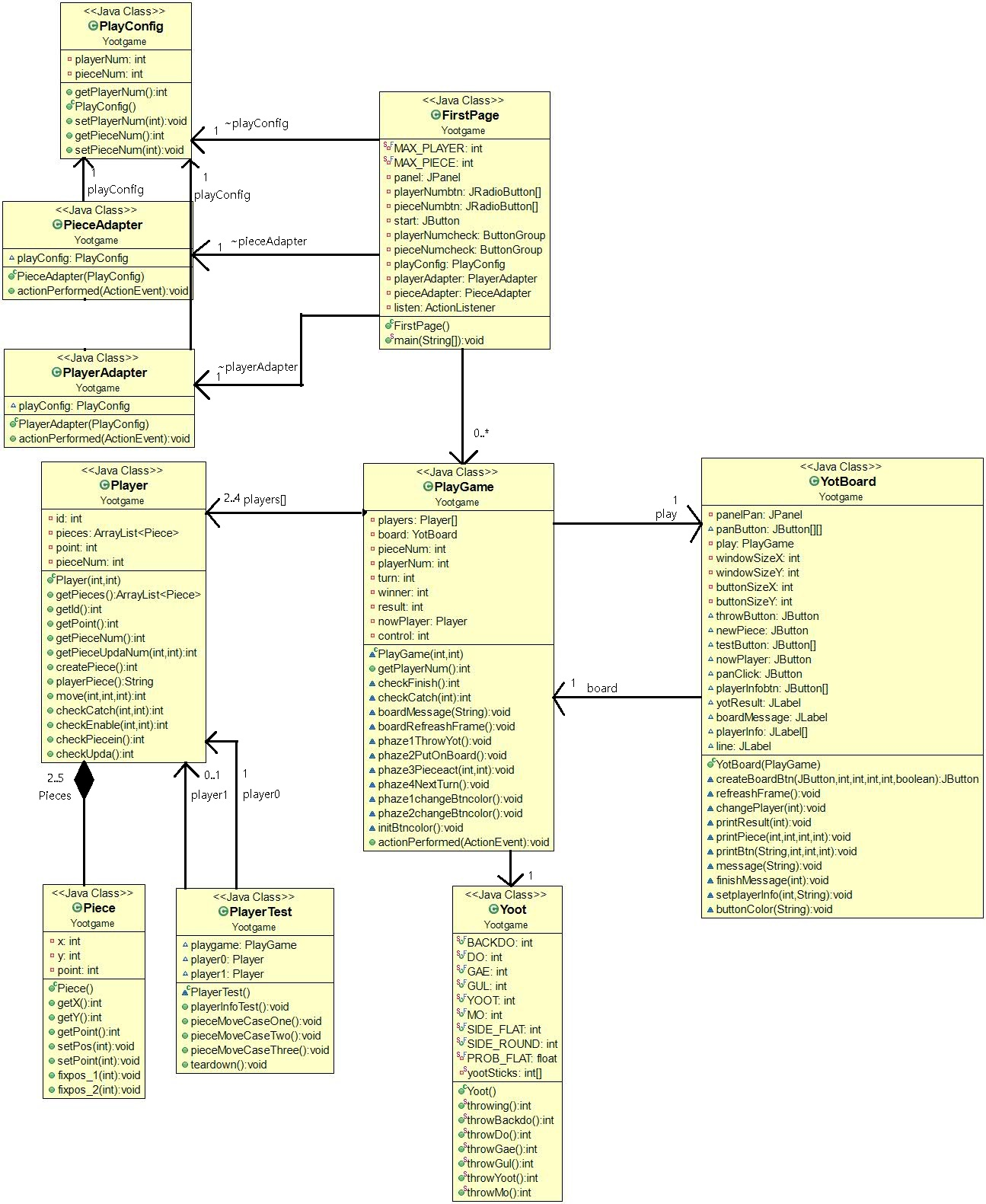
Controller는 PlayGame, PieceAdapter, PlayerAdapter으로 이뤄져 있다.

1. PlayGame은 전체 게임의 진행을 관장하는 객체이다. 이를테면 게임 진행 순서를 지키기 위해 Player 생성, Player의 모든 말이 골인했는지 확인, 다른 Player의 piece를 잡았는지 확인, 현재 Player의 턴이 끝나 다음 Player로 넘어가야 하는지 확인, 윷 다시 던져야 하는지 확인하는 게임 진행 순서와 관련된 기능이 있다.
2. PlayGame 컨트롤러를 통해서 Player, Piece의 모델과 YotBoard UI가 연결된다. 만약 PlayGame 컨트롤러가 없다면 Player 객체 마다 게임 실행 순서로 실행해야 하지만 PlayGame으로 묶여 있기 때문에, PlayGame 객체 하나만 있으면 게임 진행 순서대로 실행할 수 있다. 게임 수행 중 발생할 수 있는 여러 상황을 대처하기 위해서 많은 기능들이 포함되어 있어서 응집도가 낮다는 단점이 있다.
3. PieceAdapter와 PlayerAdapter는 FirstPage에서 받은 Player와 Piece 데이터를 PlayConfig 객체를 통하여 PlayGame 객체에 전달한다.

View는 YotBoard로 이뤄져 있다.

YotBoard는 Controller로부터 Model에 대한 정보를 받아 화면에 출력하는 역할을 한다. 따라서 각종 버튼 생성과 Player들의 Piece 위치와 정보를 화면에 출력하고, 게임 진행하면서 생기는 상황에 따른 메시지 출력하는 기능이 있다.

JUnit은 Model에 테스트를 위해 만든 또 다른 하나의 Controller이다.

**Class Diagram**

**Model**

- Piece

(개요)

판 위에 놓일 말을 생성하는 클래스

역할

현재 말의 x, y 좌표와 업힌 말이 있는 지 여부를 나타내는 point를 저장한다.

주요 기능 및 메소드

fixpos\_1(int act): 윷 판이 규칙적이지 않아 외곽에서 대각선으로 진행 방향을 바꿔준다.

fixpos\_2(int act): 대각선에서 진행 방향을 바꿔준다.

* Player

개요

윷놀이에 참여하는 플레이어들을 만들어주는 클래스

역할

생성된 플레이어 인스턴스마다 구분을 하고 윷놀이에 사용할 말 객체들을 갖고 있다. 판에 나오지 않고 대기 중인 말의 수와 결승점을 통과한 말의 수를 저장하고 있는다.

주요 기능 및 메소드

createPiece(): 미리 말을 생성하지 않고 남은 말의 수만 갖고 있다가 생성 요청이 들어올 경우 말을 미리 만들어 놓은 array에 담아준다.

move(int x, int y, int active): 말의 위치와 윷의 값을 전달해서 선택한 위치에 해당하는 말의 위치를 윷의 값만큼 이동시킨다.

checkCatch(int positionx, int positiony): 말이 저장된 array를 돌면서 잡힌 말이 있으면 그 말을 판 위에서 제거해준다.

checkEnable(int posX, int posY): 선택한 위치에 있는 말을 확인하고 그 말을 활성화해준다.

checkPieceint(): 말이 결승점을 통과했는지 확인한다. 말이 통과했으면 판에서 제거하고 플레이어의 point를 증가시킨다.

checkUpda():

* Yoot

개요

윷을 던져, 값을 반환해주는 클래스

역할

윷가락이 둥근 면과 평평한 면이 나올 확률을 0.4, 0.6으로 다르게 해주어서 값이 랜덤하게 나오도록 한다. 나온 값은 Player에게 전달되어 선택한 말을 해당 값만큼 움직여줄 수 있도록 한다.

주요 기능 및 메소드

도, 개, 걸, 윷, 모, 빽도에 해당하는 값을 미리 정해준다.

throwing(): 윷가락 4개를 던져 나온 면을 랜덤하게 설정한 후 확인하여 값을 반환해준다.

View

* FirstPage

개요

게임 진행 전 설정을 담당하는 View 클래스이다.

역할

윷놀이 게임 인원 수를 설정한다.

윷놀이에 필요한 인당 말의 수를 설정한다.

게임을 시작한다.

주요 기능 및 메소드

FirstPage(): 사용자 수, 말 개수 선택하는 버튼을 RadioButton으로 만들어서 중복 선택 불가하도록 해준다. 선택되어 있는 RadioButton의 값을 PlayerAdapter와 PieceAdapter에서 알고 있다가 start 버튼이 눌리면 PlayGame에 값을 전달하여 플레이어와 말의 수를 정해준다.

* YotBoard

개요

View에 해당하는 클래스

역할

윷판, 각종 버튼, 상태메시지 출력 등을 하면서 플레이어에게 그래픽으로 윷놀이 진행 상황을 보여준다. 플레이어는 YotBoard를 통해 게임과 상호작용한다.

주요 기능 및 메소드

YotBoard는 Java의 GUI 관련 라이브러리인 swing의 JFrame을 상속받아 구현되었다.

YotBoard(PlayGame playObject): 판의 각 칸은 버튼으로 만들어져있다. 판의 외곽에 있는 칸과 안쪽 대각선에 존재하는 칸이 딱 들어맞지 않아서 버튼을 3\*21의 배열에 저장해서 사용한다.

panButton[0][1~20]: 외곽에 해당하는 버튼을 관리한다.

panButton[1][0~5]: Right-Top에 해당되는 대각선에 있는 버튼을 관리한다. 인덱스 3에 해당되는 버튼은 중간 교차점에 해당되는 버튼이다. 이 버튼은 Left-Top 대각선에 해당 되는 부분에서 관리한다. 중간이나 초과해서 사용하지 않은 인덱스는 비워둔다.

panButton[2][0~5]: Left-Top에 해당되는 대각선에 있는 버튼을 관리한다. 초과된 인덱스에 해당되는 부분은 비워둔다.

버튼 이외에 상태메시지, 플레이어 상태 등과 관련된 기능은 따로 메소드로 구현해서 불러와 View를 출력한다.

refreshFrame(): 플레이어가 턴을 넘길 때 나온 윷의 값과 현재 플레이어에 관한 출력을 모두 지워 다음 플레이어에 해당되는 메시지를 출력할 수 있도록 준비하는 메소드

Controller

* PlayGame

개요

게임 전반적인 진행을 담당하는 Controller에 해당하는 클래스

역할

플레이어가 누른 버튼에 해당되는 메소드를 Call한다. 메소드를 Call해서 필요한 Model에 해당하는 값을 반환하도록 하거나 변경하도록 전달하는 역할을 한다.

주요 기능 및 메소드

게임 진행은 총 4개의 단계로 나눠서 각 단계에 해당하는 메소드를 구현해서 게임을 진행한다. 각 단계는 선행되는 조건이 있기 때문에 이를 제어해주기 위해서 control 변수를 만들어서 관리하게 했다. 각 메소드들은 자신에게 해당되는 control 변수 값이 아닐 경우 동작하지 않고, 자신의 기능을 적절히 수행한 다음에는 다음 단계에 해당하는 control 걊으로 변경한다.

phaze1ThrowYut(): 플레이어가 수행할 가장 첫번째 단계인 윷 던지기에 해당되는 메소드. Yoot 클래스의 throwing 메소드를 호출해서 값을 받아온다.

phaze2PutOnBoard(): 판 위에 새로운 말을 생성하는 메소드.

phaze3Pieceact(int posx, int posy): 플레이어가 클릭한 말의 좌표를 받아서 해당 말을 움직여주는 메소드

phaze4NextTurn(): 다음 플레이어에게 턴을 넘기거나 플레이어가 승리했다면 게임을 중지하는 메소드.

* PlayConfig

개요

게임 설정 전체를 총괄하는 클래스

역할

PlayerAdapter와 PieceAdapter에서 파라미터로 사용되어 게임 설정을 돕는다.

주요 기능 및 메소드

PlayConfig(): 플레이어 수, 말의 수 모두 최소값인 2로 초기화한다.

getPlayerNum(): 플레이어 수 반환

setPlayerNum(int playerNum): 플레이어 수 설정

getPieceNum(): 말의 수 반환

setPieceNum(int pieceNum): 말의 수 설정

* PlayerAdapter

개요

PlayConfig와 현재 선택된 RadioButton을 전달받아 플레이어 수를 저장하는 클래스

역할

현재 선택된 플레이어 수를 start 버튼이 눌리면 PlayGame으로 전달한다.

주요 기능 및 메소드

actionPerformed(ActionEvent e): 선택된 RadioButton이 변경될 때마다 그 값을 플레이어 수에 저장한다.

* PieceAdapter

개요

PlayConfig와 현재 선택된 RadioButton을 전달받아 말의 개수를 저장하는 클래스

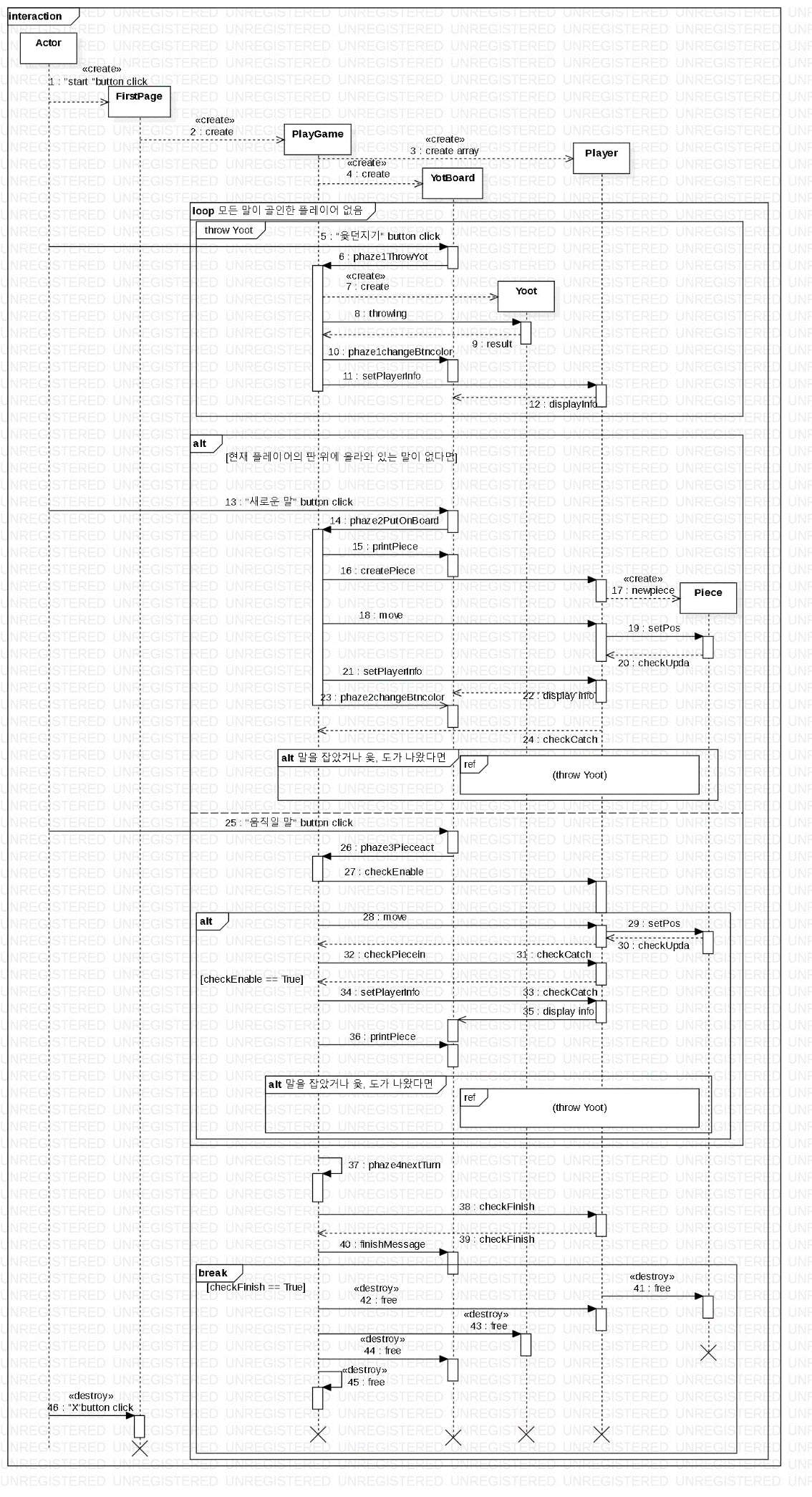
역할

현재 선택된 말의 개수를 start 버튼이 눌리면 PlayGame으로 전달한다.

주요 기능 및 메소드

actionPerformed(ActionEvent e): 선택된 RadioButton이 변경될 때마다 그 값을 말의 개수에 저장한다.

checkUpda(): 한 플레이어가 갖고 있는 말들을 비교하면서 동일한 위치에 있는 말이 존재할 경우 업힌 상태를 변경해주고 업힌 말을 제거한다.

**Sequence Diagram**

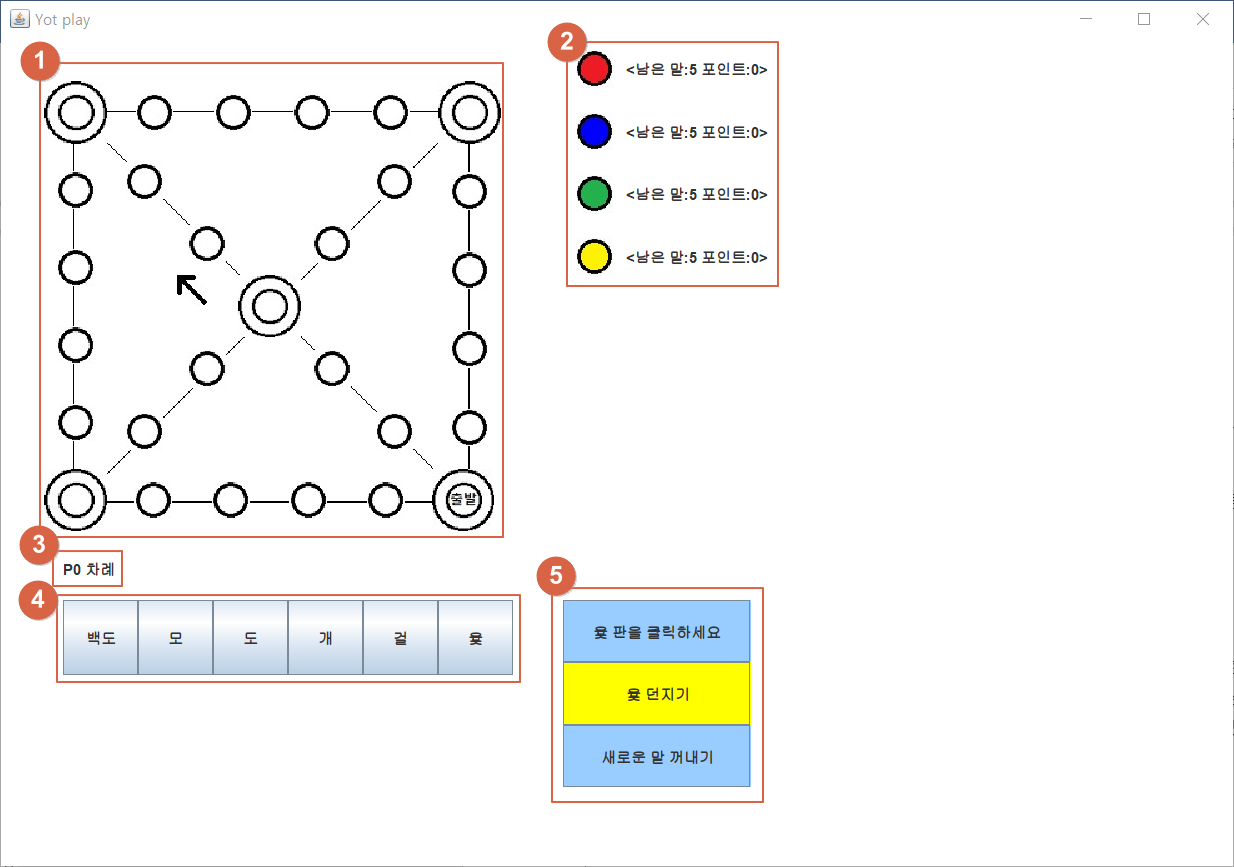
**e. 게임 실행**

- 첫번째 페이지



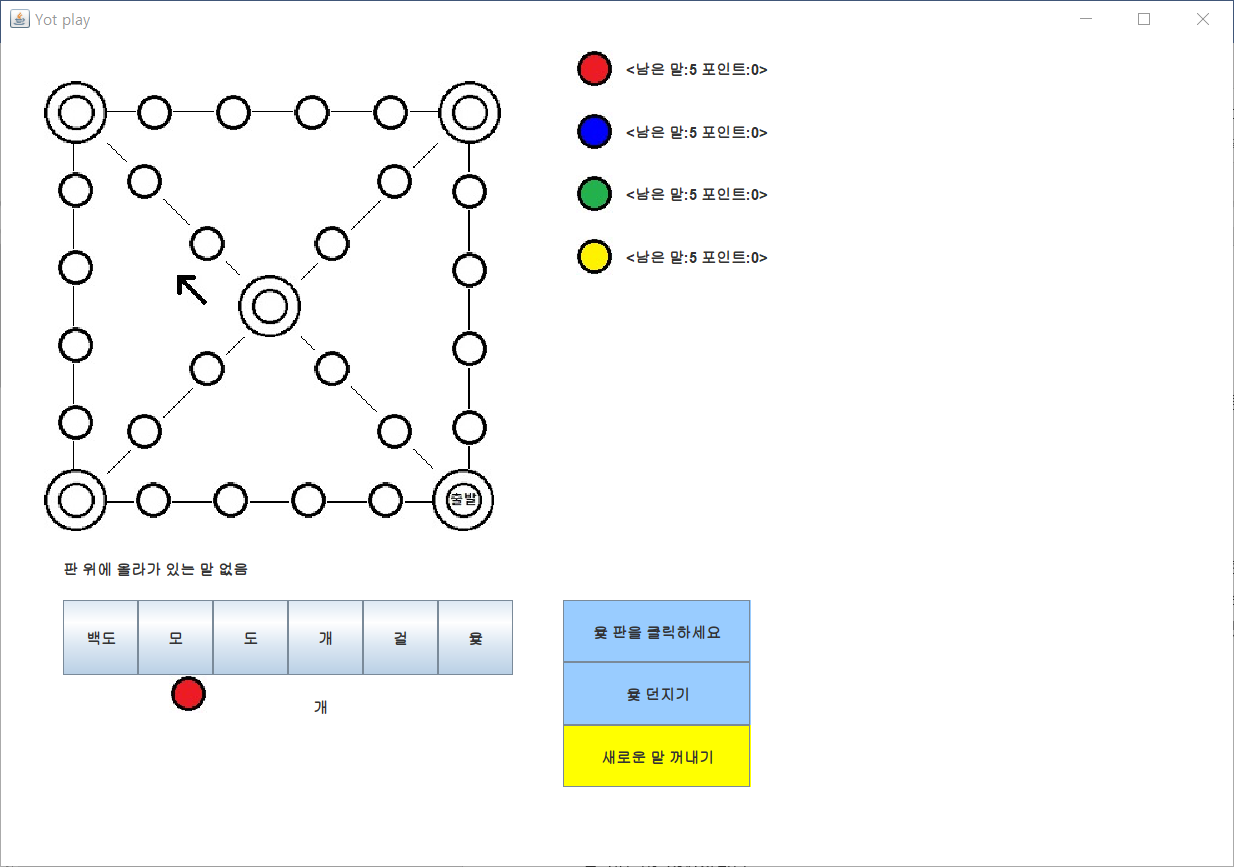
사용자 수, 말 개수를 선택할 수 있게 한다. 각각의 설정을 중복해서 선택할 수 없도록 Radio 버튼을 사용해주었다. 선택이 끝난 후 시작 버튼을 누르면 본격적인 게임이 시작된다.

- 첫번째 플레이어부터 시작



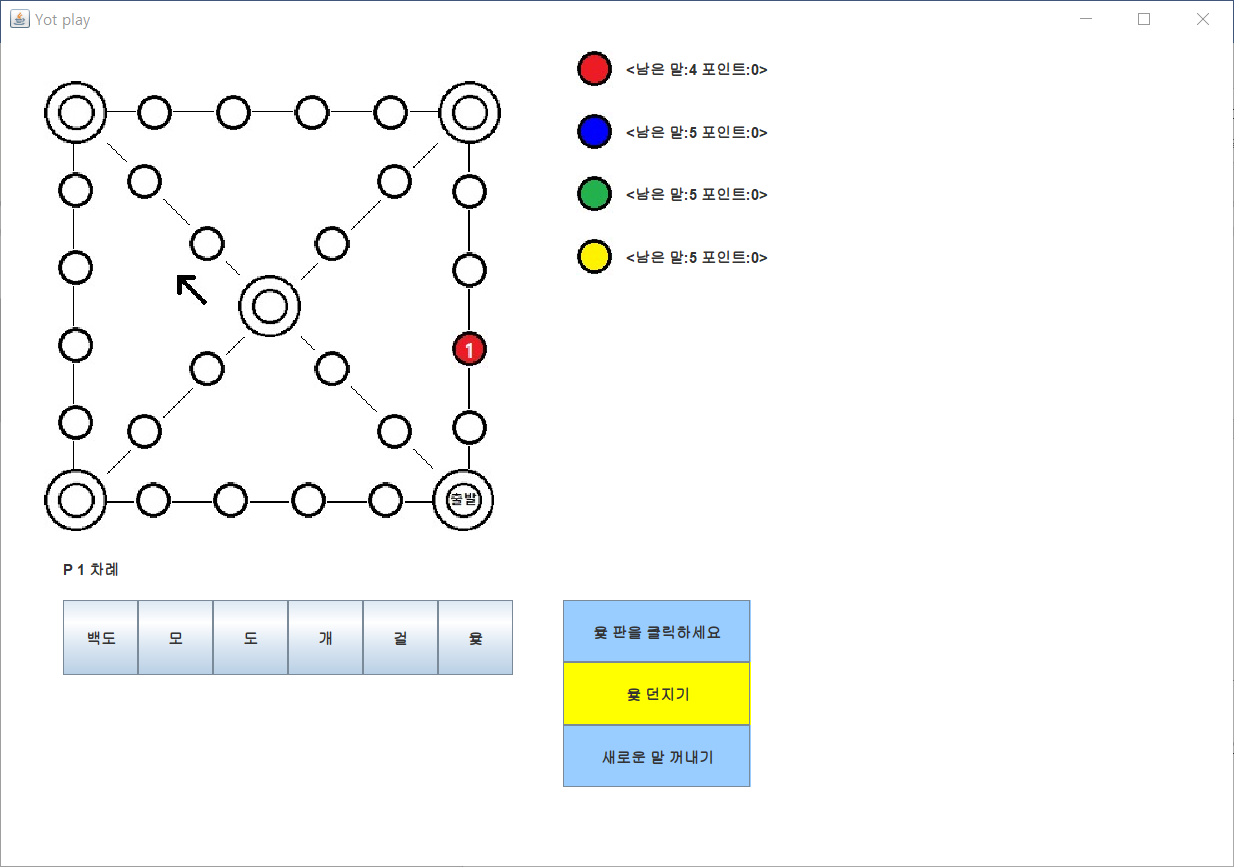
1. 윷판: 출발(도착)점, 교차점은 이중 원으로 표현해서 가독성을 높여주었다. 화살표는 중앙 교차점에서 빽도가 나왔을 경우 가게 될 방향을 나타내준다.
2. 플레이어 정보: 플레이어 말의 색을 다르게 하여 플레이어를 구분해준다. 말 오른쪽에 플레이어의 남은 말과 점수를 숫자로 나타내준다.
3. 상태 메시지: 누구의 차례인지와 같은 현재 상태 정보를 나타내준다.
4. 지정 윷 던지기 버튼
5. 플레이어가 할 수 있는 행동과 관련된 버튼: 누를 수 있는 버튼을 노란색으로 표현해서 플레이어가 해당 버튼을 누를 수 있도록 유도한다. 만약 노란색 버튼을 누르지 않았다면 상태메시지 창에 잘못 눌렀습니다, 다시 누르세요를 나타내서 눌러야 하는 버튼을 다시 누를 수 있도록 한다.

- 윷을 던진 직후



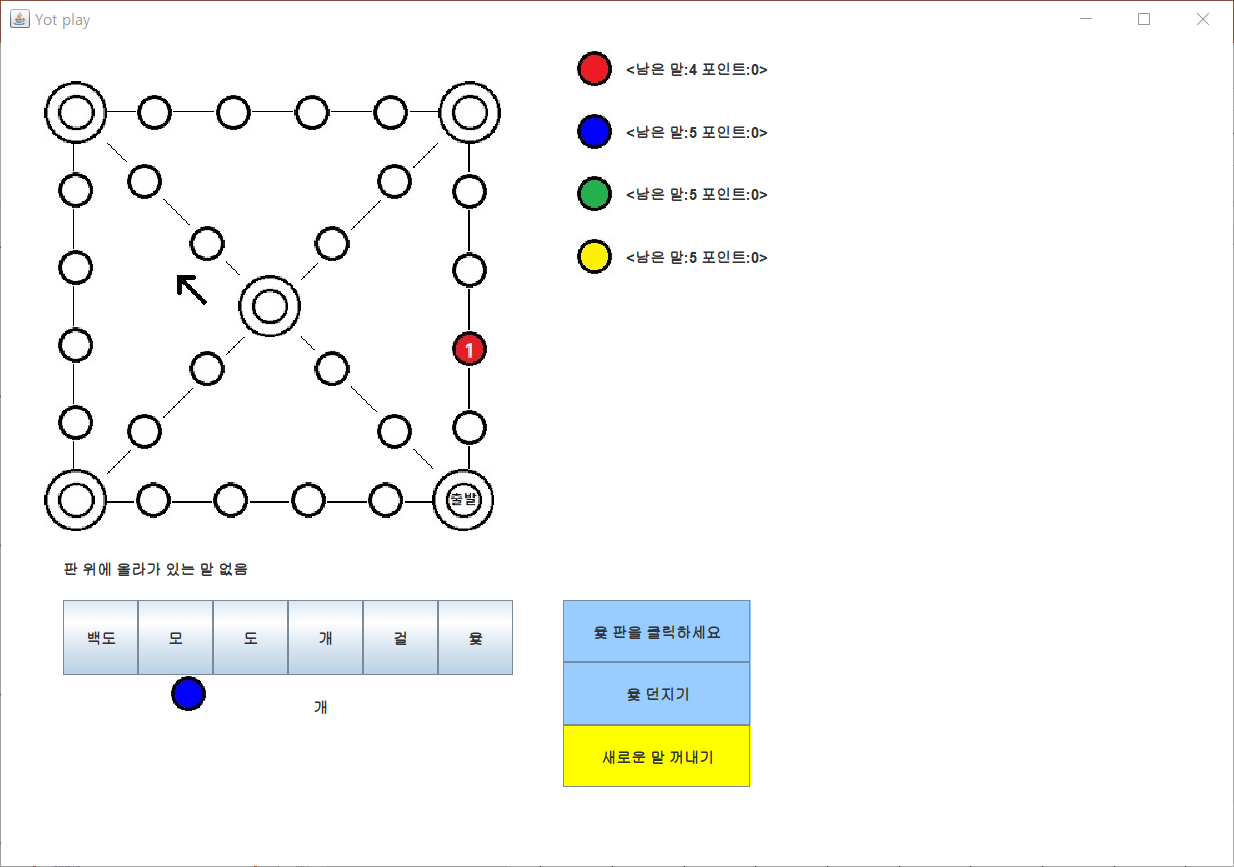
윷을 던졌기 때문에 다음 행동으로 말을 선택해서 움직여주어야 한다. 하지만 아직 판 위에 현재 플레이어의 말이 없기 때문에 새로운 말을 꺼내는 행동만 가능하다. 따라서 새로운 말에 해당되는 버튼만 활성화된다.

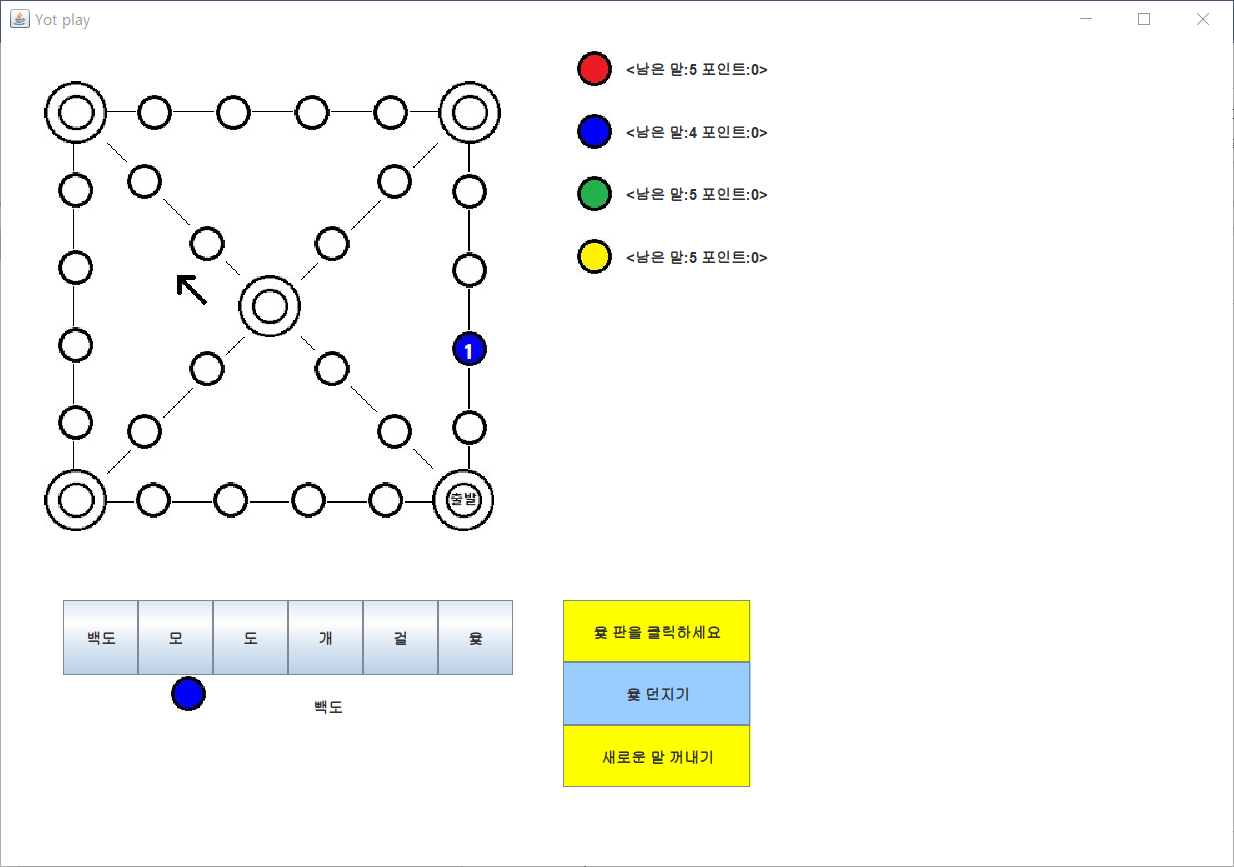
- 새로운 말 꺼내기를 선택한 직후



새로운 말을 정상적으로 꺼냈다면 판 위해 해당 플레이어의 말이 윷의 값만큼 이동해 나온 위치에 나타난다. 아직 말을 업지 않았기 때문에 플레이어 말 위에 숫자 1로 아무것도 업지 않고 하나의 말만 움직이는 것을 나타낸다. 만약 하나의 말을 업어서 두개가 한번에 움직인다면 숫자 2가 나타날 것이다. 그리고 첫번째 플레이어의 말이 생성되었으므로 첫번째 플레이어의 정보를 업데이트한다.

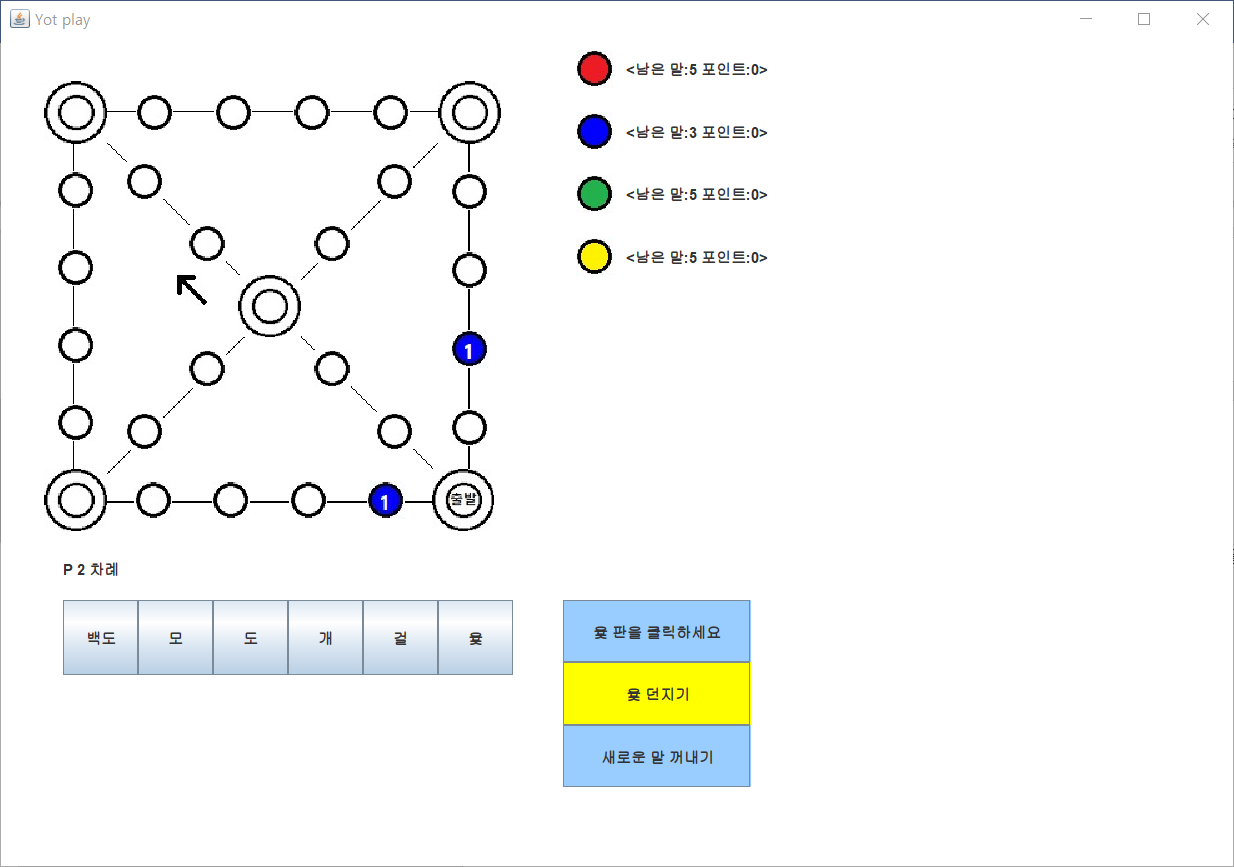
- 말을 잡을 수 있는 경우





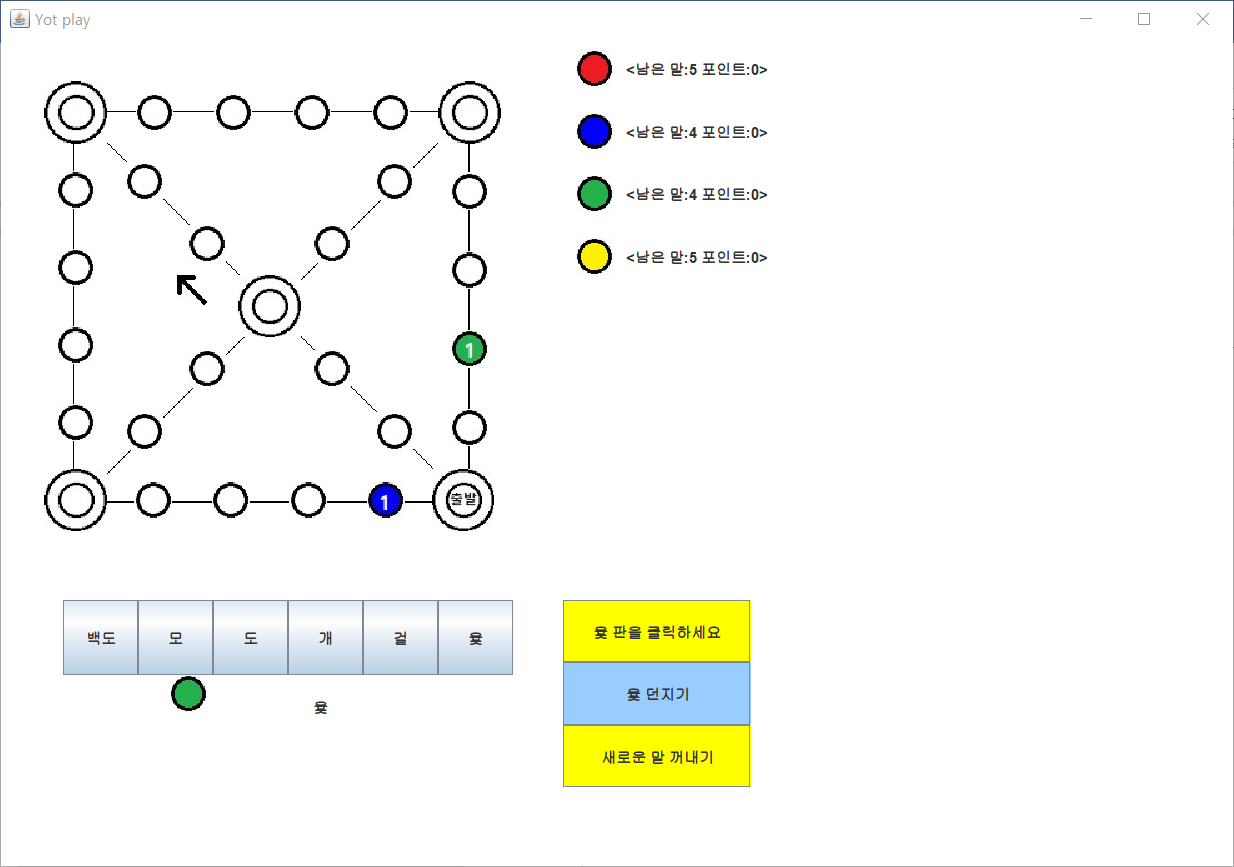
말을 선택해서 이동시킨 다음 말을 잡으면 윷 던지기를 누르지 않아도 자동으로 윷을 던진다.

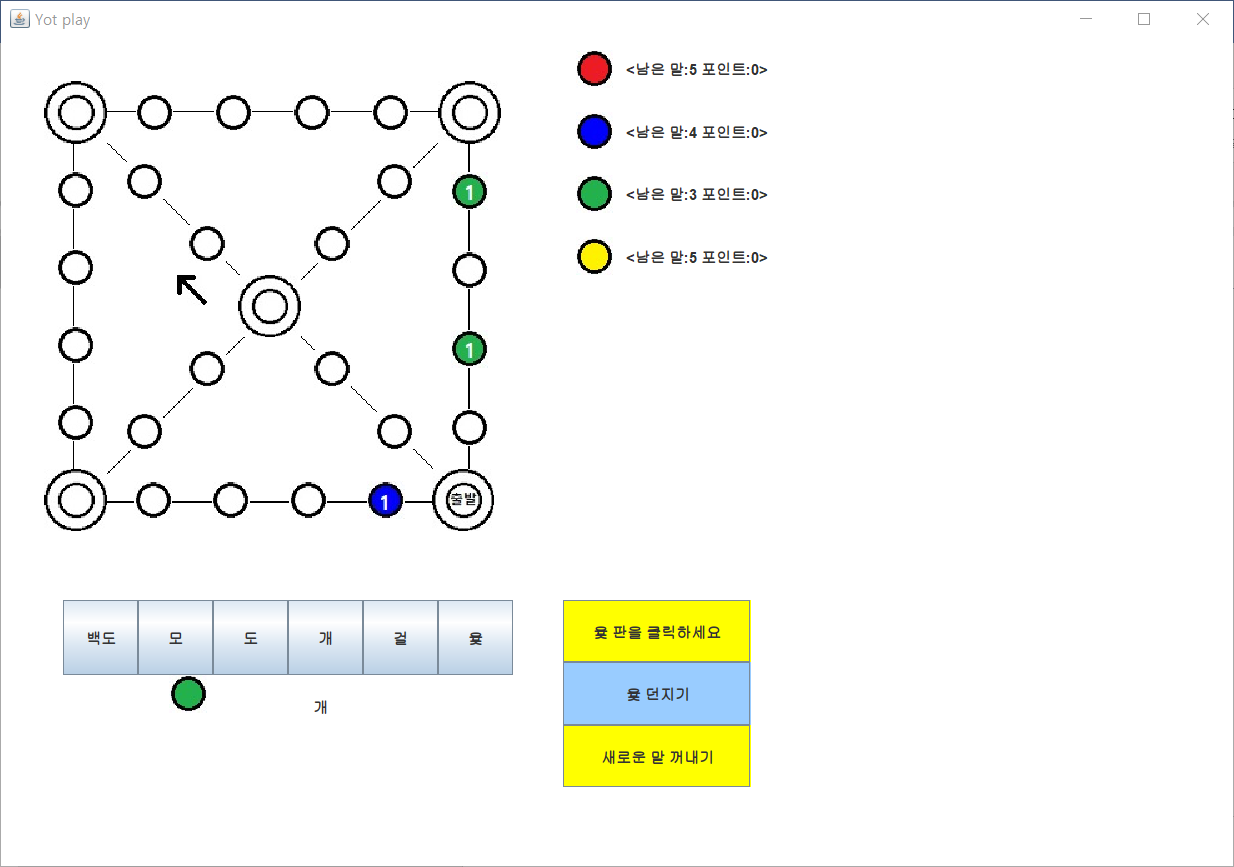
- 빽도가 나온 경우



빽도가 나온 경우에는 판 위에 있는 말을 뒤로 옮길 수도 있지만 새로운 말을 꺼내게 되면 결승점 바로 직전에 놓을 수 있다.

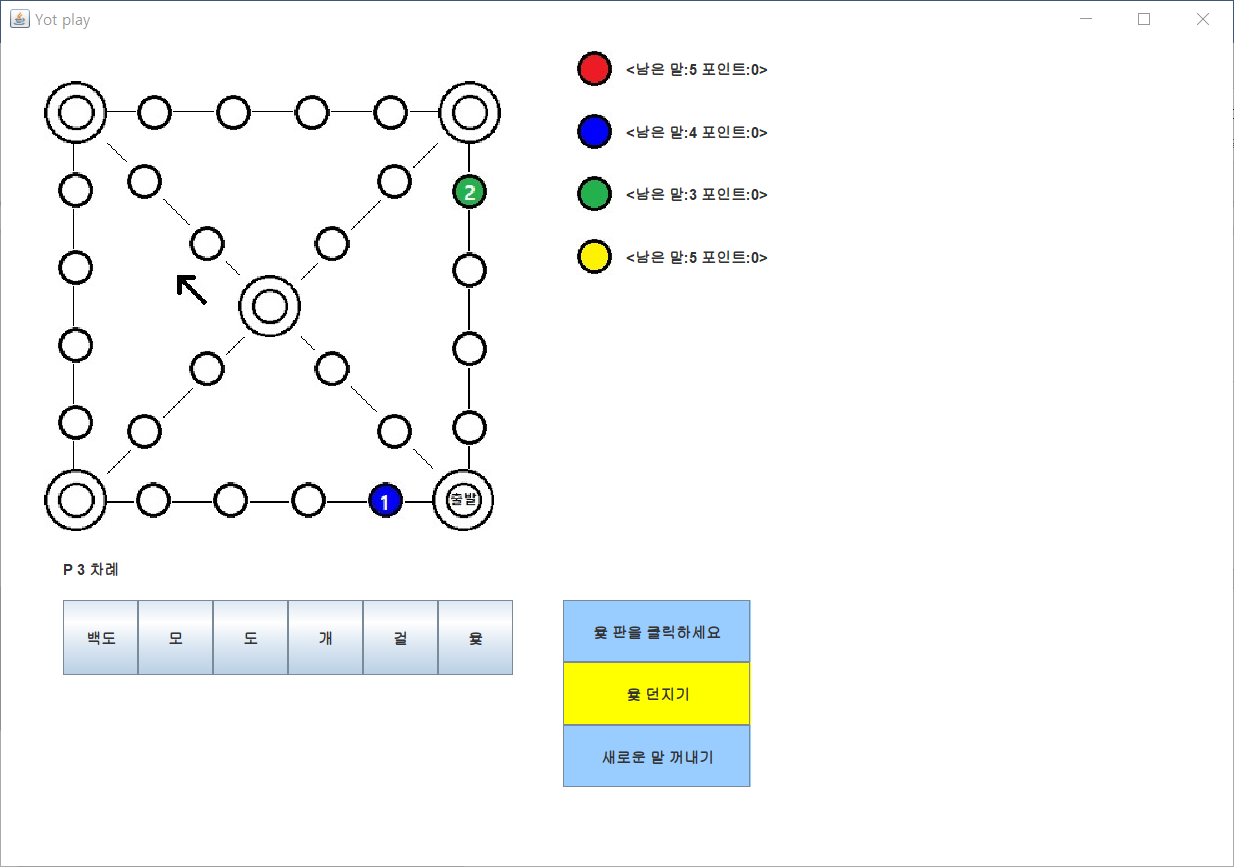
- 윷(또는 모)가 나온 경우





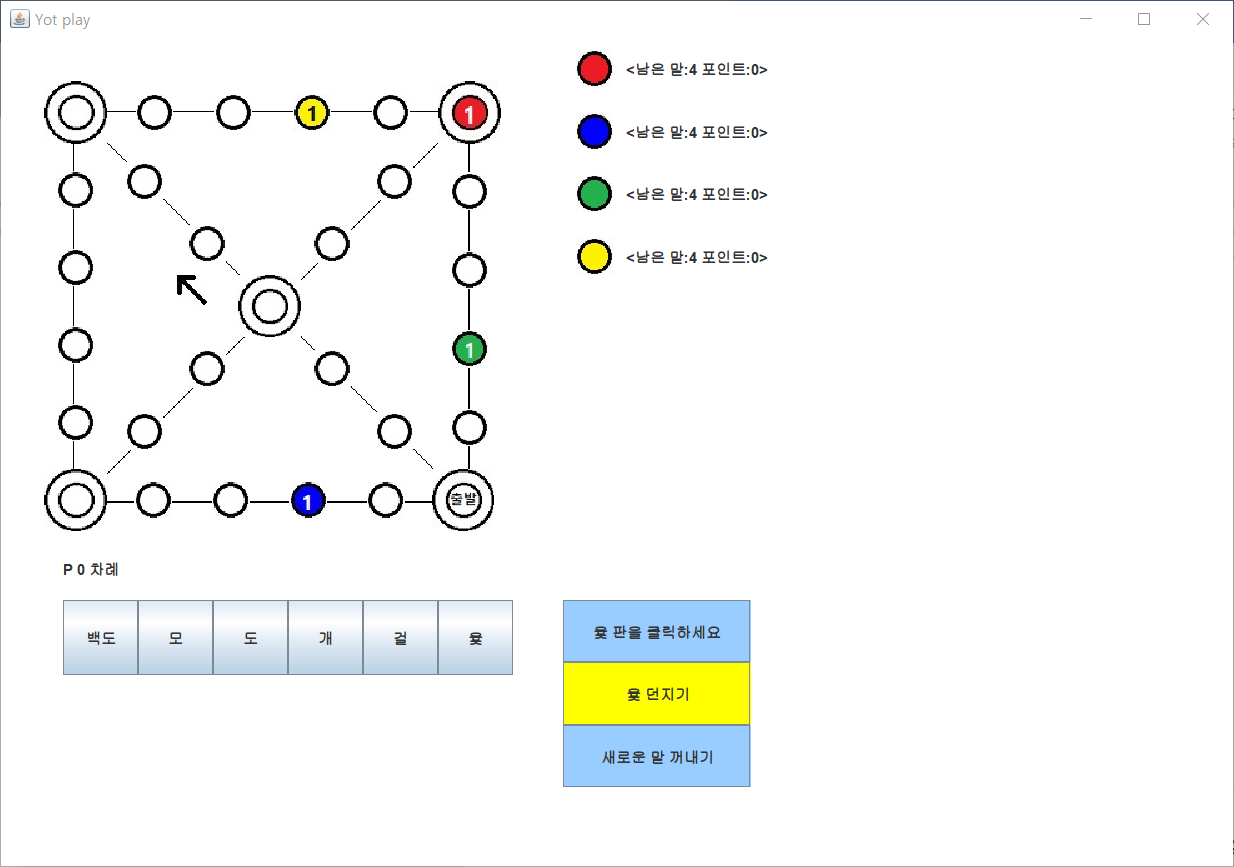
다시 한번 더 던질 수 있지만 윷을 순서대로 적용하기로 했기 때문에 먼저 말을 옮긴 다음에 자동으로 윷을 던진다.

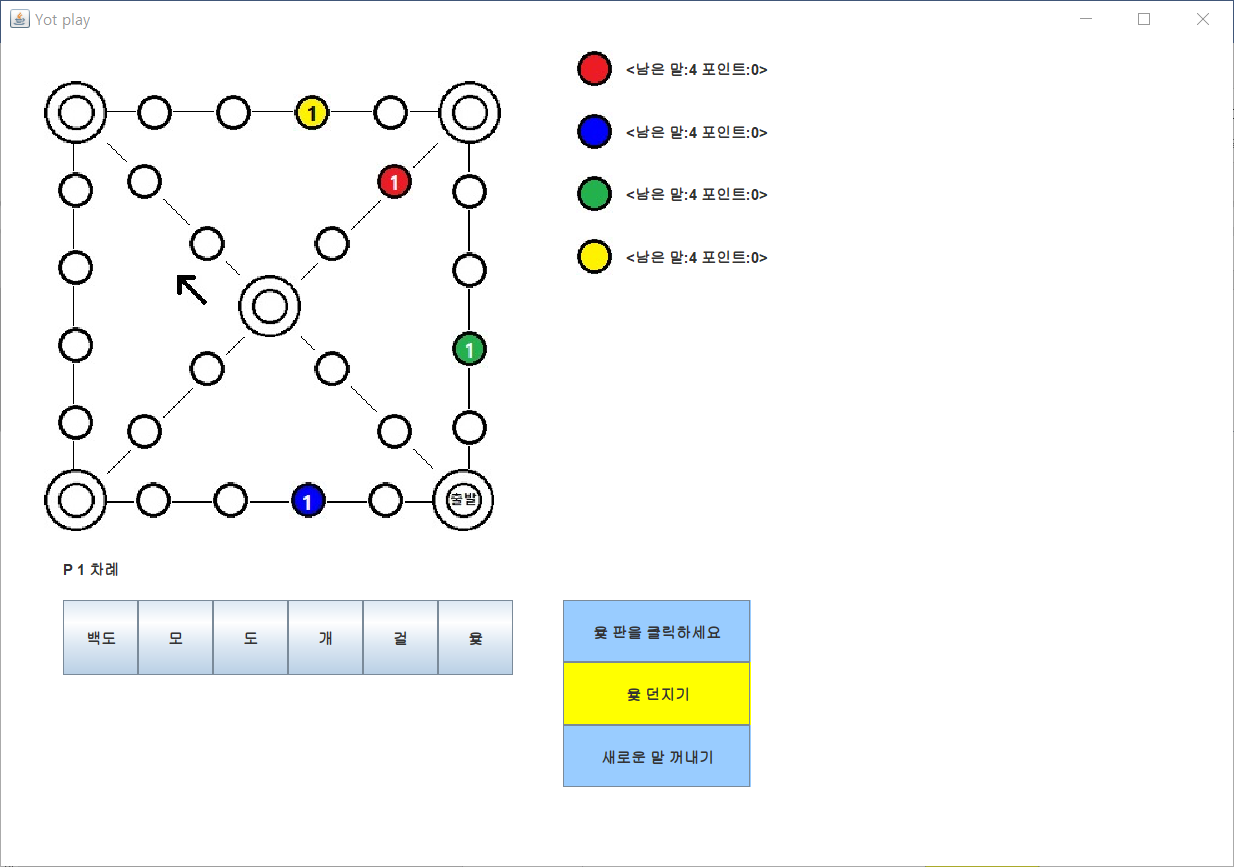
- 말을 업을 수 있는 경우



이전 상황에서 3번째 플레이어는 개가 나와서 말을 업을 수 있었다. 말을 업게 되면 해당 플레이어가 소유하고 있는 말 중에 업은 말의 개수를 판 위에 표시한다. 업은 말의 수는 숫자로 표현된다.

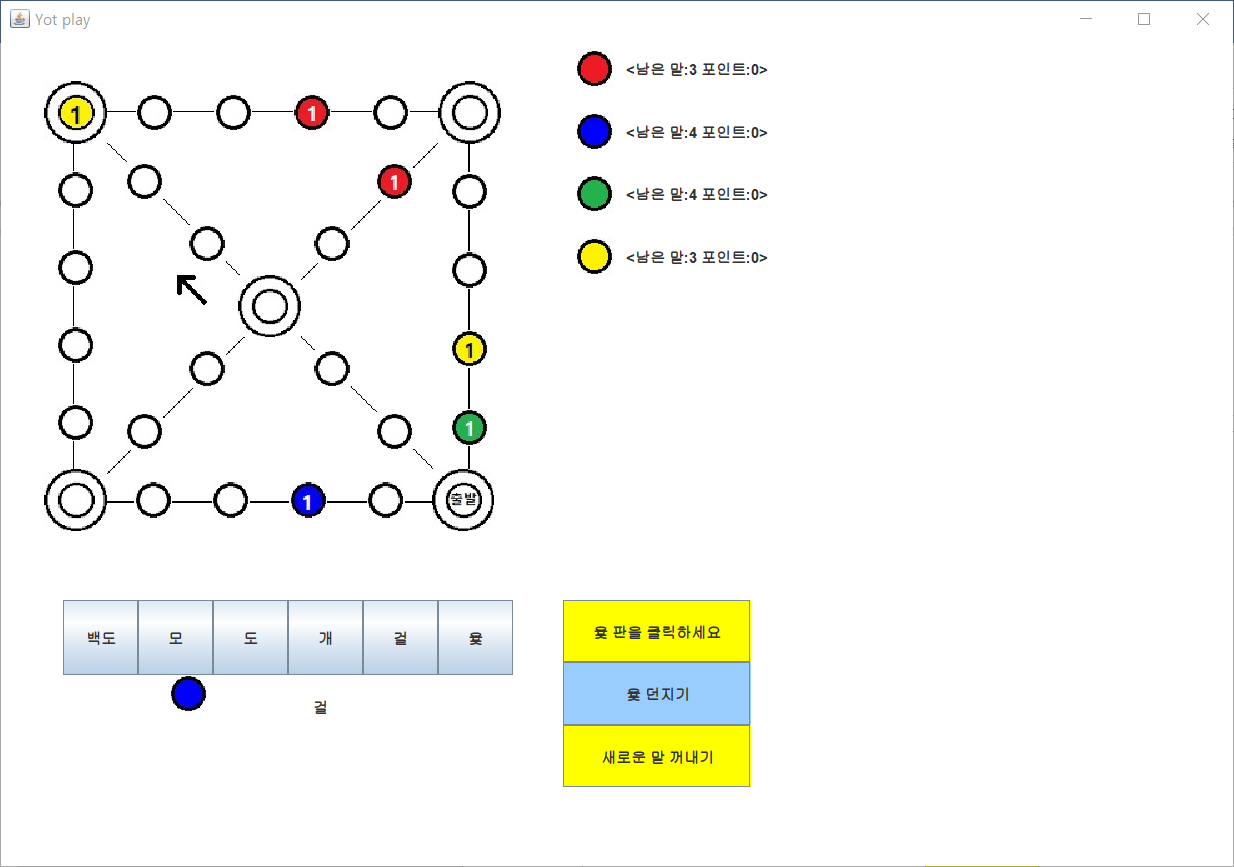
- 교차점에 있는 경우

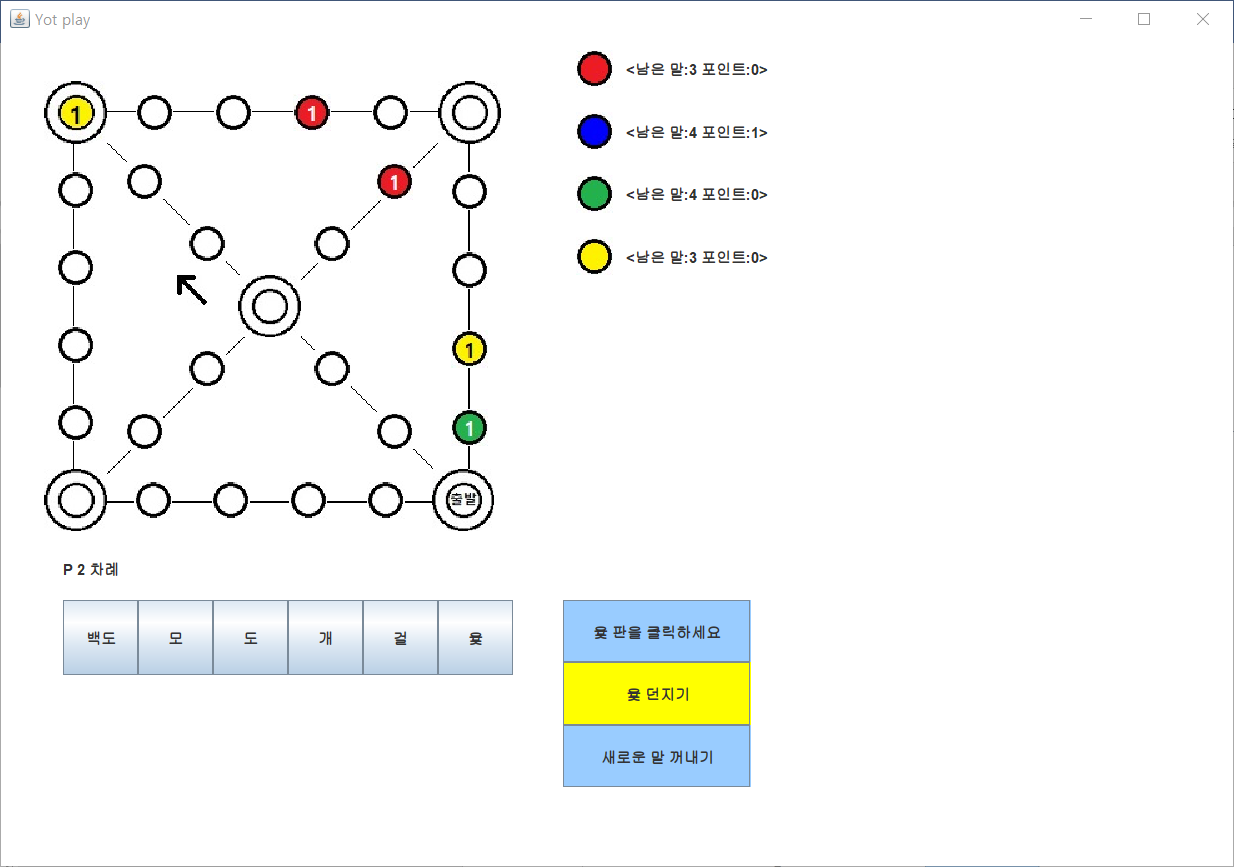




교차점에 있는 경우에는 반드시 안쪽 지름길로만 진행할 수 있다

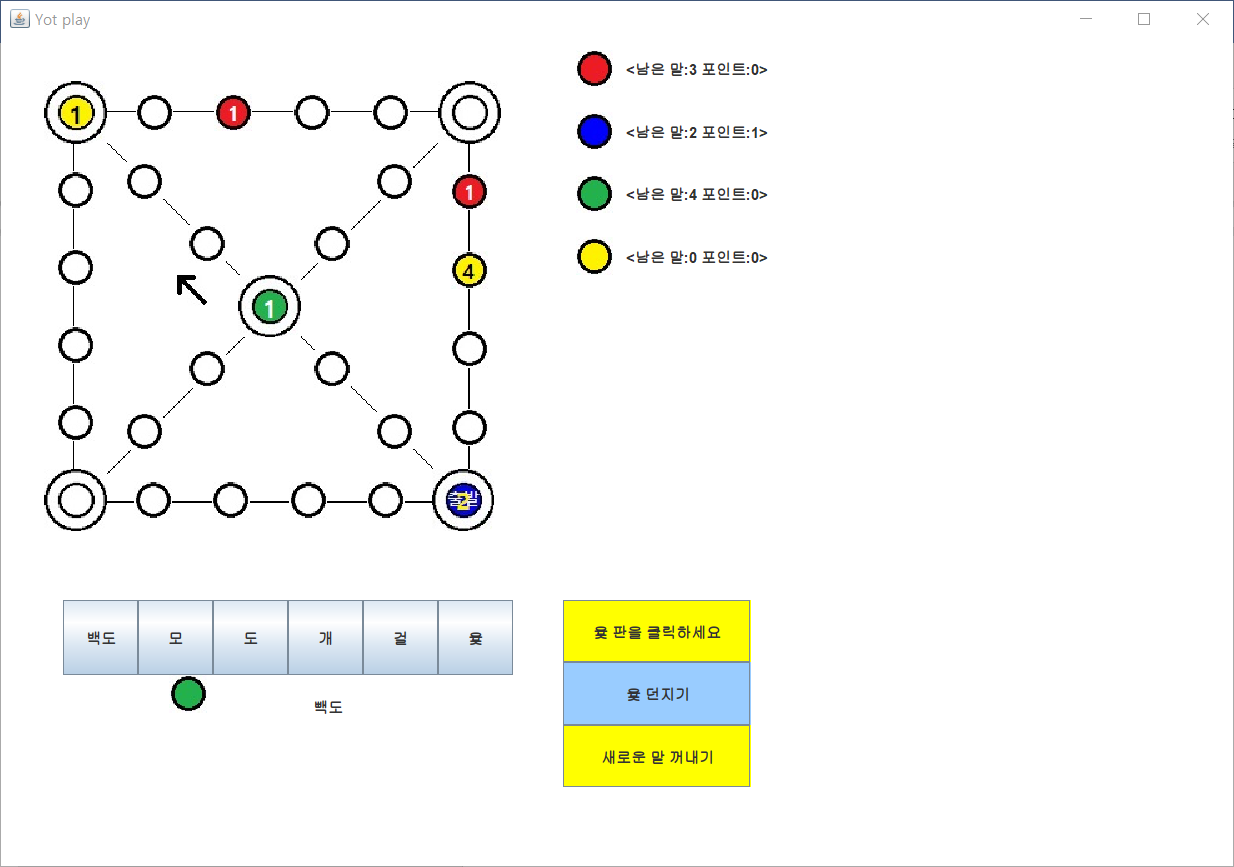
- 결승점을 통과하는 경우

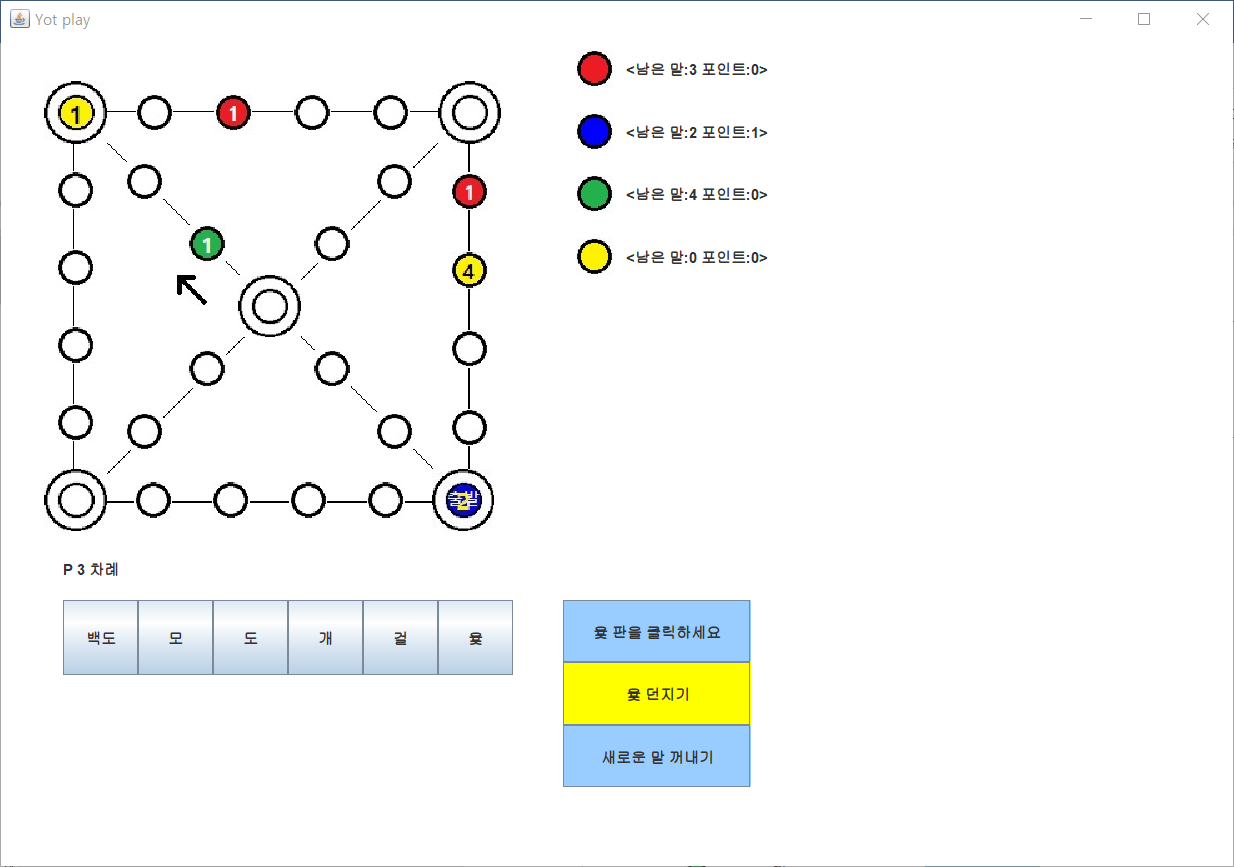




두번째 플레이어의 말이 결승점을 통과하면 통과한 말의 수만큼 포인트가 올라간다.

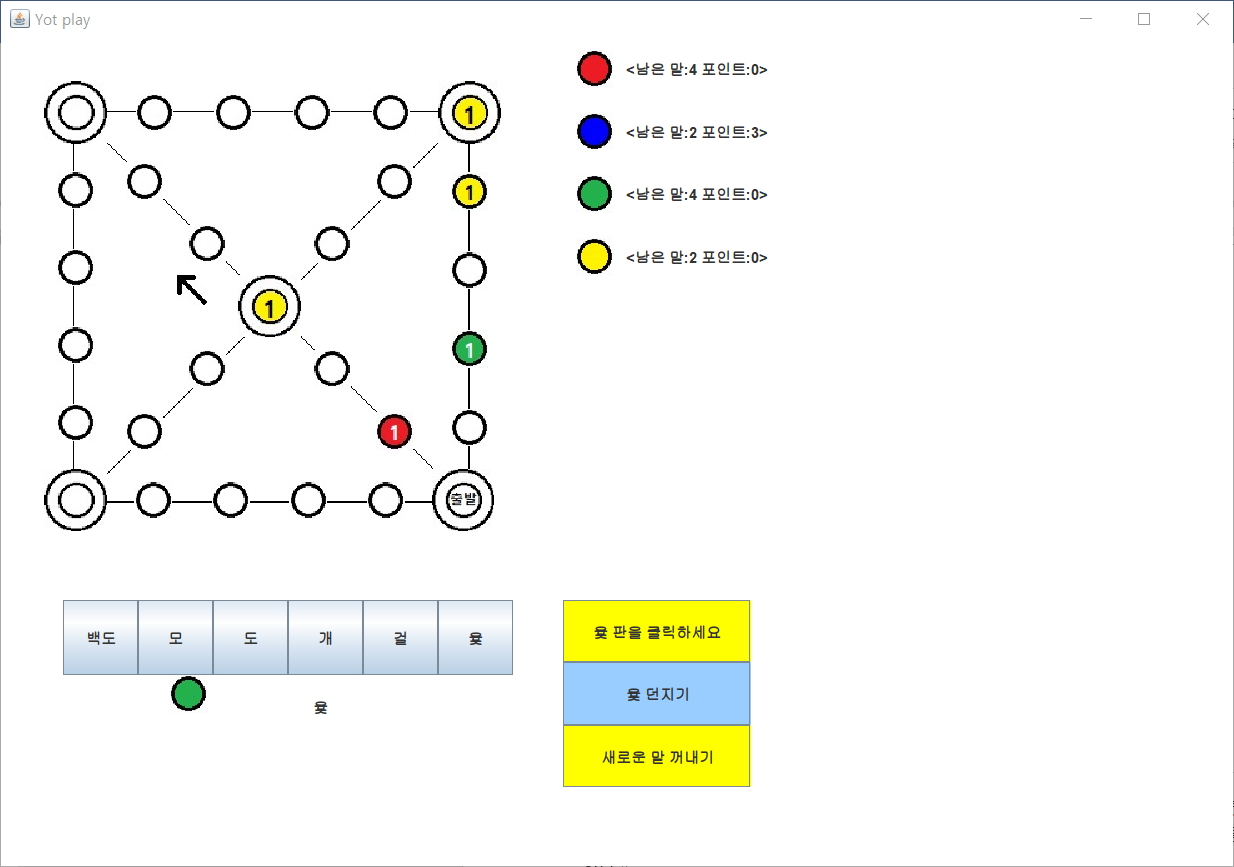
- 중간 교차점에서 빽도가 나온 경우

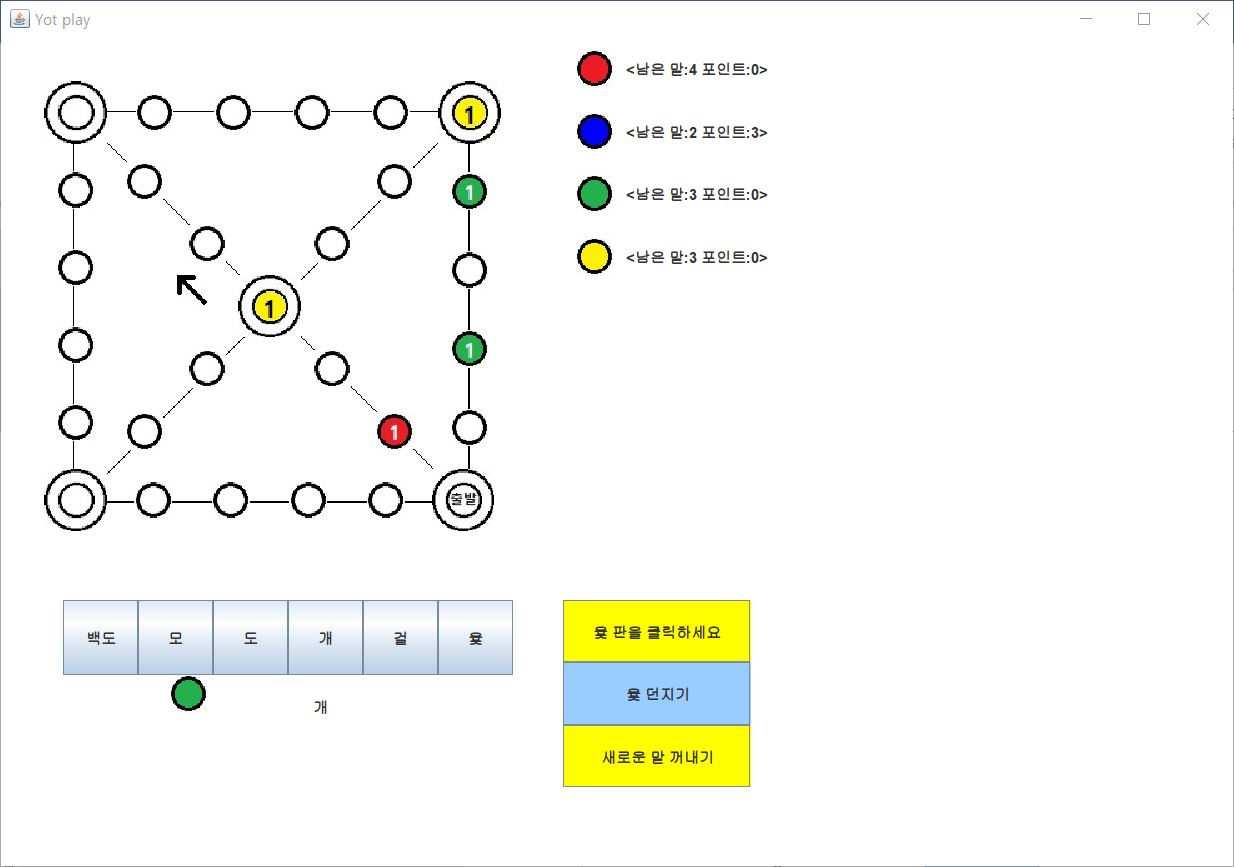


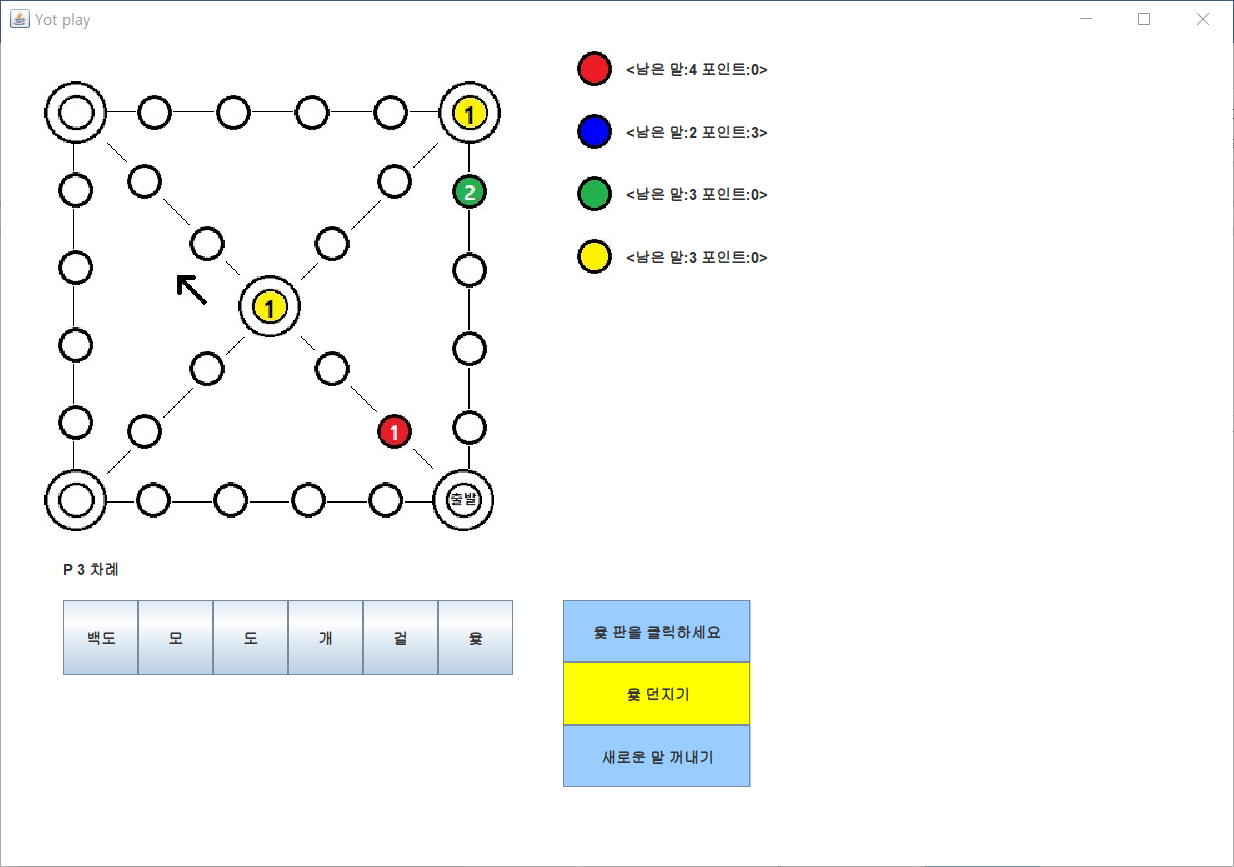


중간 과정을 생략했지만 초록색 말은 오른쪽 지름길에서 중간 교차로에 도달했다. 빽도가 나왔을 때 진행 방향의 반대로 가는 것이지만 중간 교차점에서는 화살표 방향대로 왼쪽 지름길로만 움직인다.

- 윷(또는 모)로 말을 잡는 경우

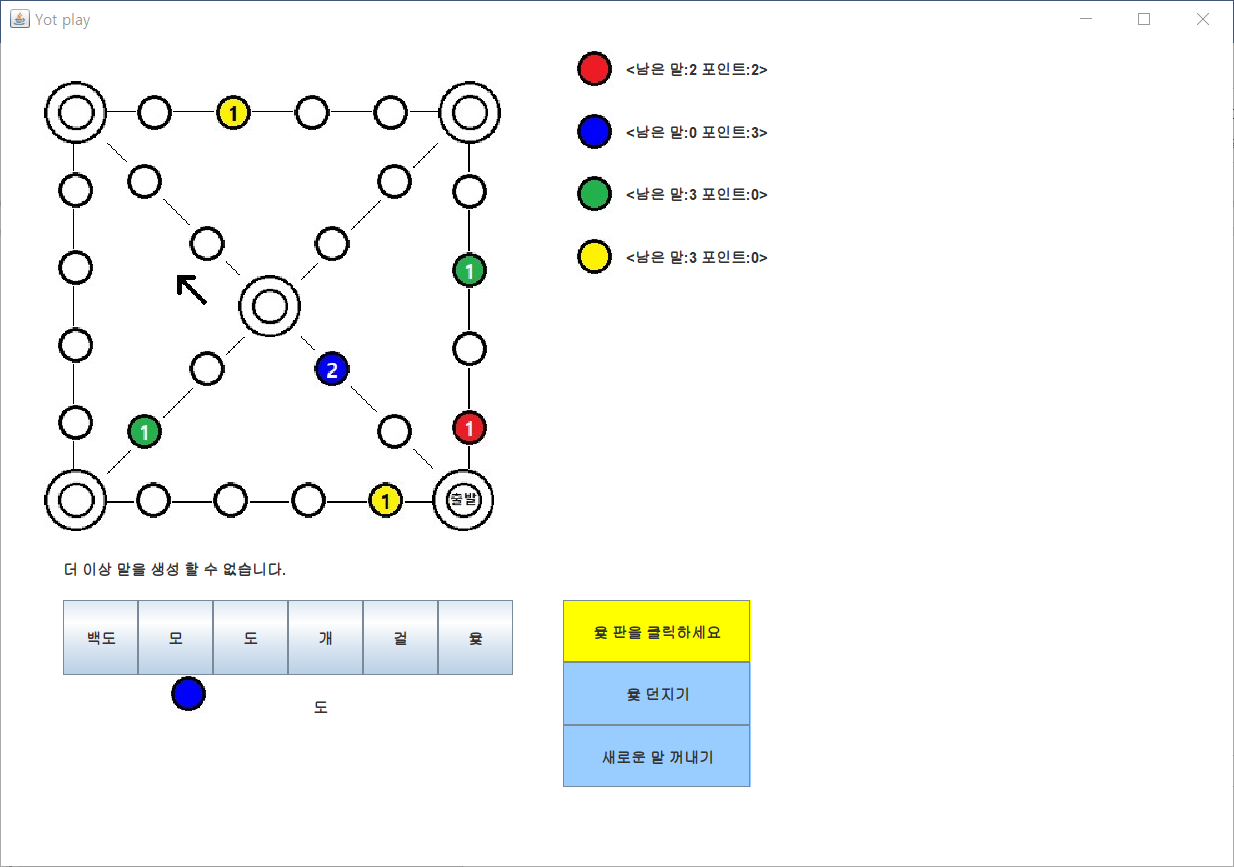






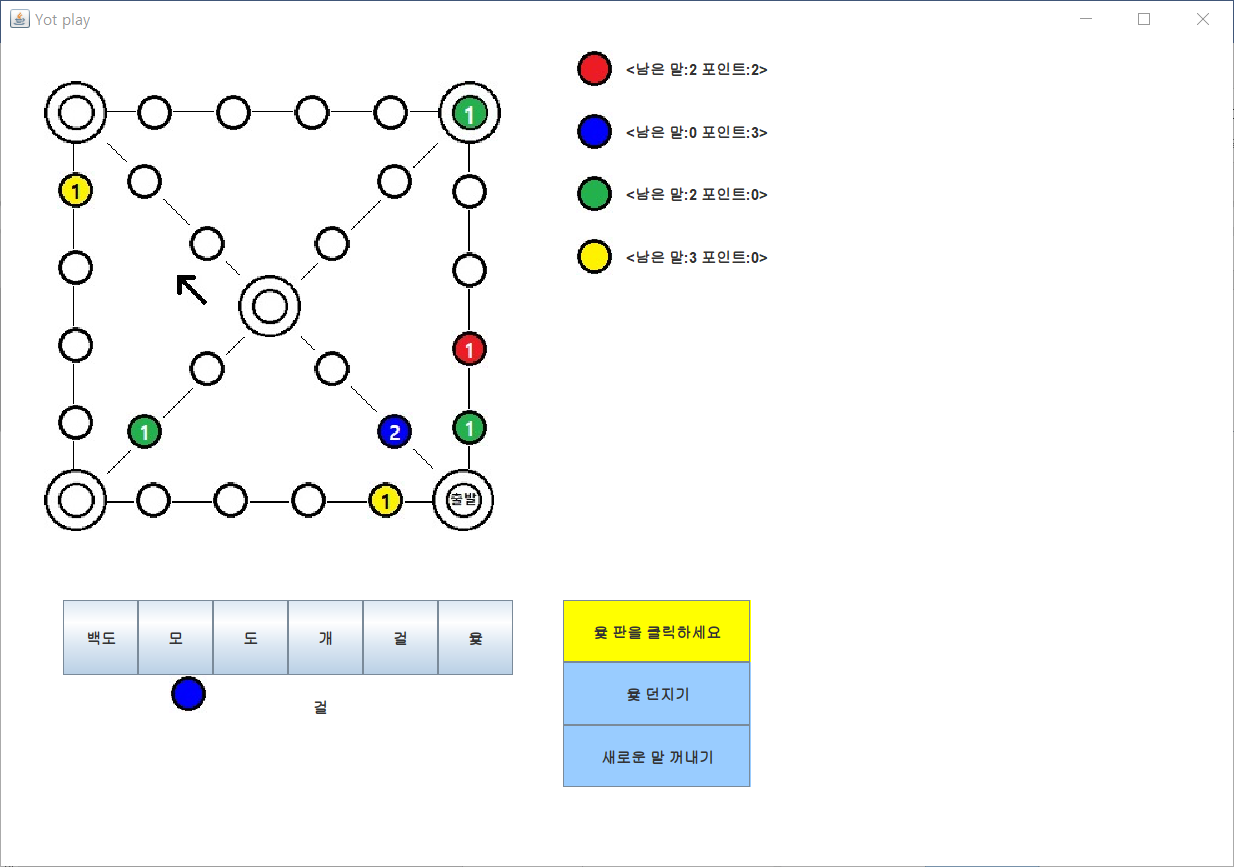
윷(또는 모)로 말을 잡는 경우에는 윷을 두번 더 던지는 것이 아닌 한번만 더 던져준다.

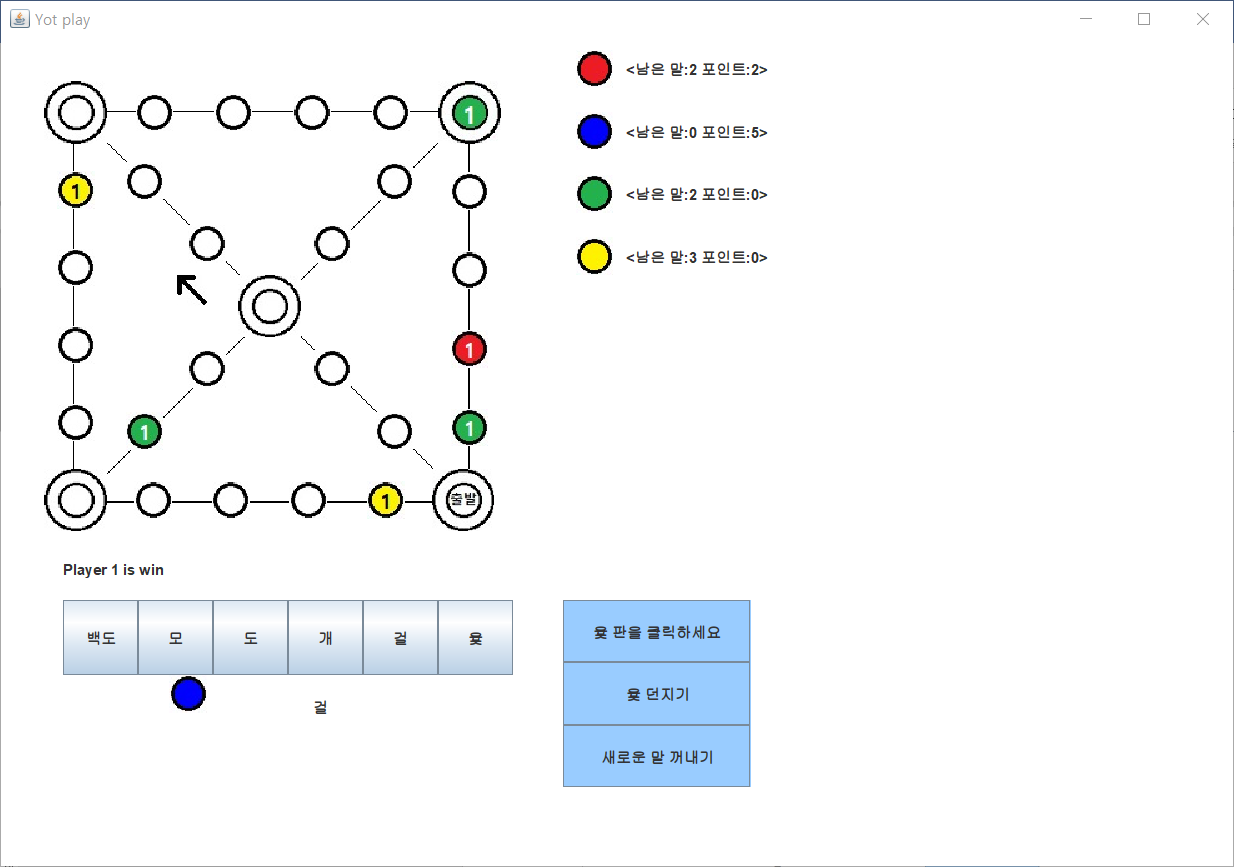
- 더 이상 꺼낼 말이 없을 때



새로운 말 버튼이 비활성화된 것을 볼 수 있다.

- 승자가 나오는 경우





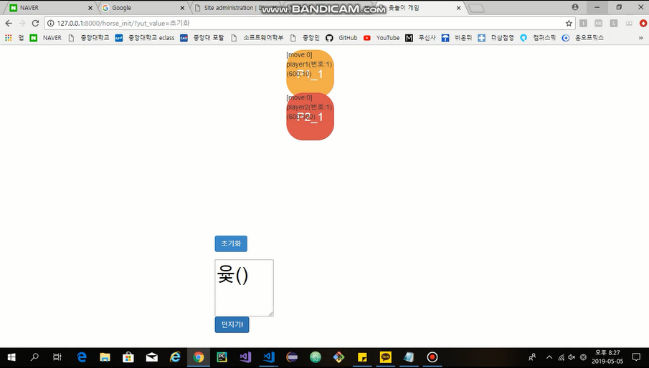
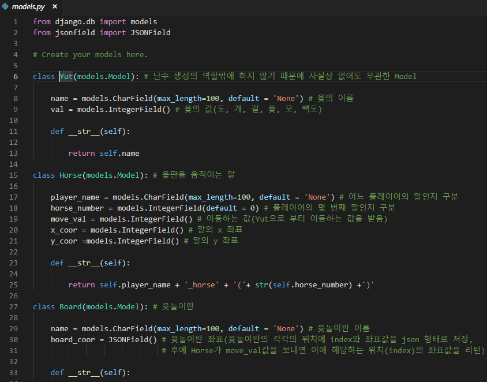
승자를 상태 메시지에 표시하고 모든 버튼을 비활성화한다.

- 데모 영상

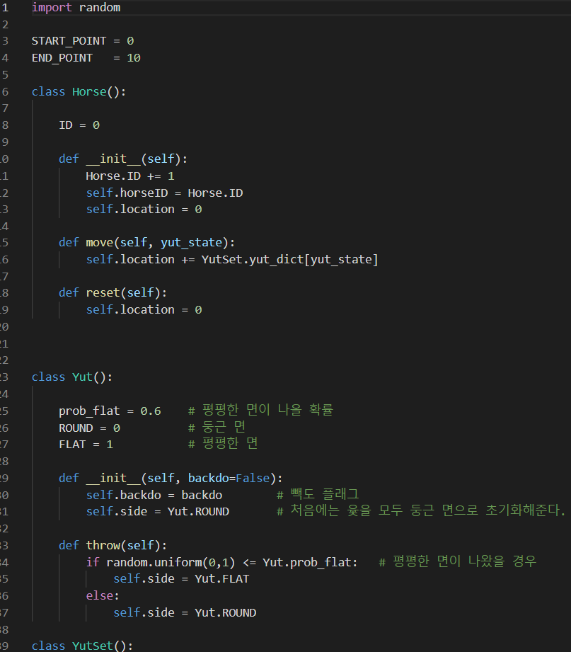
[](file:///C:\Users\KJK\AppData\Roaming\Microsoft\Word\윷놀이%20데모.mp4)

**f. History & Experience**

**History)**

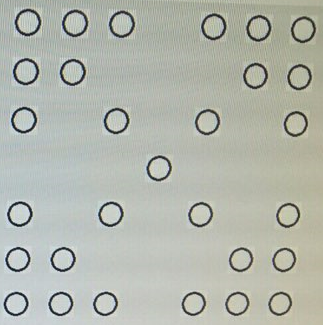
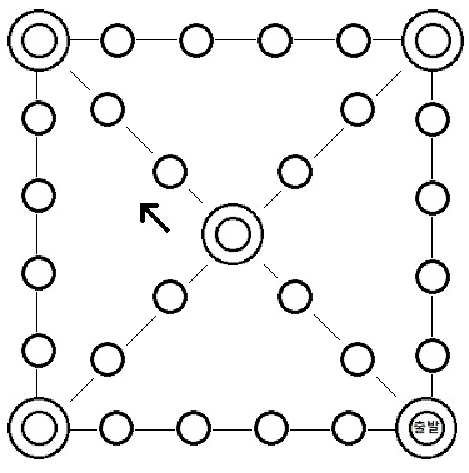
1. 프로토타입  
   윷 판과 윷 던지기 부분을 각자 프로토타입으로 만들어서 윷놀이 진행에 필요한 기능을 이해하고 어떤 Requirement가 있는지 알아보았다.

▲ 파이썬 + 쟝고 프로토타입



▲ 자바스크립트 + pug 프로토타입

▲ 파이썬 윷 던지기 프로토타입

1. 쟝고 프레임워크에 따라 MVT 모델로 나눠서 개발 결정
2. 자바, Swing 라이브러리 활용하는 것으로 변경
3. 자바 식별자 명명 관례를 이용해서 클래스명, 메소드, 변수명 등 통일하는 규칙 결정
4. Use Case 결정  
    FirstPage, Player, Piece, YotBoard, PlayGame 클래스가 필요함을 명세하고 Design 시작
5. Player과 Yoot 클래스 분리  
   Player가 윷 값을 단순히 받아오는 것이라 Yoot 클래스를 따로 만들어줄 필요를 못 느껴서 따로 Yoot 클래스를 두지 않았지만 윷 또는 모가 너무 많이 나오는 버그를 확인하고 논리상 Yoot 클래스를 따로 두는 것이 맞다는 회의 결과 Yoot을 클래스로 분리시킴
6. 보드판 UI 개선  
     

▶

1. 사용자 정보 출력 UI 개선  
   Piece의 좌표 값 출력 삭제하고, 사용자 정보 즉각 업데이트되지 않는 문제 해결, 플레이하는 수만큼만 플레이어 정보 출력하도록 변경
2. JUnit test 추가
3. PlayConfig, PlayerAdapter, PieceAdapter 추가
4. 중앙 교차점 버그 개선  
   중앙 교차점에서 업힘이 발생하면 오류를 발생하는 문제 해결

**Experience)**

**유인근:** 이번 Term Project를 통해 OOAD의 중요성을 많이 느꼈고, 팀원들과 협력해 소프트웨어를 디자인하고 개발한 경험이 실제 개발 업무에서 큰 도움이 될 거라고 생각합니다. 그리고 Unit Test가 무엇인지 경험할 수 있었고, 실제 서비스를 만들 때 Unit Test가 굉장히 중요한 작업이라는 것을 체감할 수 있었습니다.

**김진관:** 객체지향프로그래밍에 대한 개념은 현재 프로그래밍에서 가장 중요한 부분이라 이미 알고 있던 부분이었다. 하지만 실제 구현할 때 각 클래스의 관계를 체계적으로 짜기가 힘든 부분이 많았다. 이번에 소프트웨어 공학을 통해 OOAD를 적용해서 프로젝트를 진행해보니까 객체지향 설계에 대한 감을 잡았고 실제 프로젝트를 어떤 식으로 진행해야 하는지에 대한 이해가 깊어졌다. 이후 계획한 다른 프로젝트를 진행할 때도 배운 것을 활용해 볼 수 있을 것 같아서 기대가 된다.

**김병창:** 팀프로젝트의 경험이 적어, 프로젝트의 초기에는 어려움을 겪었습니다. 각자의 스타일이 다르고, 경험과 생각이 다르기 때문에 어려웠습니다. 하지만 각자 프로토타입을 구현하며 프로젝트 전반에 대한 이해를 높이고, 프로세스를 따라 진행하다 보니 이러한 어려움의 상당 부분이 해소되었습니다. 그리고 완벽하지는 않지만 MVC패턴으로 코드를 구현하니, 코드의 수정이 쉬워졌고, 에러의 발생이 줄어든다는 것을 몸소 느끼게 되었습니다.

**염승윤:** UI가 있는 버전의 완성된 게임을 만든 것이 이번이 처음입니다. 이렇게 완성도 높은 프로그램을 만들 수 있었던 것은 팀원과 OOAD 설계 덕분이라고 생각합니다. 먼저 전체적인 큰 틀을 프로토타입 하고 팀원들 각자 역할을 나누어 개발한 결과 맡은 부분은 완벽하게 알고 있고 버그도 적었습니다. 또한 수업 때 배운 각종 다이아그램과 프로세스들을 이용해 개발하다 보니 전체적인 구조를 한눈에 알아볼 수 있어서 프로그래밍 하기도 수월 했습니다. 이 경험을 통해 프로그램 개발에 자신감을 얻었고 팀원과 협력하는 방법을 배우게 되었습니다.