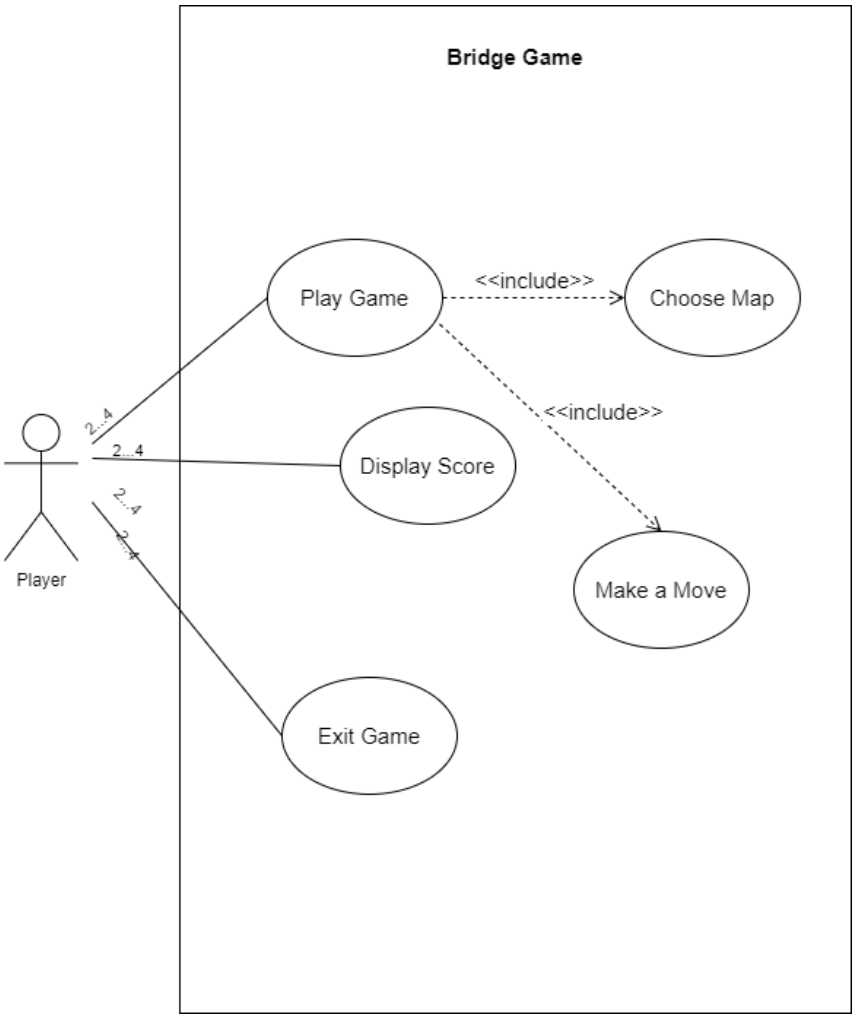
SE 2022 봄학기 텀프로젝트: OOAD를 이용한 Bridge 게임 개발

20182345 이건호

레포트 목차

[요구 정의 및 분석 산출물]

Bridge게임 개발을 위해서 먼저 UseCase분석을 했습니다.



외부 Actor는 Player가 있고 이 bridge game과 액터가 상호작용하는 방식은 크게 Play Game, Display Score, Exit Games로 나눠 봤습니다. 그리고 Play Game은 Choose Map과 Make a Move를 include하고 있는데 이는 사실 Play Game안에 다 포함시켜도 되지만, 명세를 작성할 때 Play Game이 너무 복잡해 지기 때문에 include를 이용하여 분리하였습니다. 플레이어는 문서에 나온 대로 2~4명이 가능한 것으로 생각하였습니다.

Play Game의 대한 명세는 다음과 같습니다.

-Use Case Name: Play Game

-Scope: Bridge Game

-Level: User Goal

-Primary Actor: Player

-Stakeholders and Interests:

*Player: 게임을 시작하기를 원함, 플레이어수를 선택하고 싶어 함, 맵을 고르고자 함

-Preconditions: 플레이어의 수는 2~4명 사이여야 함.

-Success Guarantee: 플레이어가 입력한 수 만큼의 말(piece)가 생성됨. 고른 맵이 생성됨. 각종 기본 설정들이 완료됨.

-Main Success Scenario:

1. 플레이어가 게임을 실행함

2. 플레이어수를 입력함

3. 맵을 고름

4. 고른 맵이 표시되고, 입력한 플레이어 수 만큼의 말(piece)이 생성됨.

5. 플레이어마다 턴이 돌아가고 움직임을 선택함.

게임 시스템은 5번을 게임이 종료되거나 한 명을 제외한 모두가 END 지점에 도달할 때까지 반복함.

-Extensions:

3.a 맵 고르기를 분리해서 명세함 :Include Choose Map

5.a 플레이어의 움직임 선택에 대한 명세 :Include Make a Move

3.a의 맵고르기 같은 경우는 또 길어지고 좀 더 자세한 설명이 필요할 것 같아서 **include관계**를 이용하여 따로 빼서 명세를 만들었습니다.

5.a의 움직임 선택에 관한 것도 마찬가지입니다.

그럼 먼저 3.a의 Choose Map에 대한 명세는 다음과 같습니다.

-Use Case Name: Choose Map

-Scope: Bridge Game

-Level: Subfunction

-Primary Actor: Player

-Stakeholders and Interests:

*Player: 플레이하고 싶은 맵을 골라서 게임을 플레이하려함.

-Preconditions: 게임 시작버튼을 누른 후 발생 함.

-Success Guarantee: 선택된 맵이 생성됨.

-Main Success Scenario:

1. 게임 프로그램은 플레이어에게 선택 가능한 맵을 보여줍니다.

2. 플레이어는 그 중 하나를 선택합니다.

3. 시스템은 해당 맵을 바탕으로 한칸한칸을 맵으로 만들며 게임을

생성합니다.

-Extensions:

2.a 만약 해당 맵이 실제로는 존재하지 않으면

1. 시스템은 플레이어에게 다른 맵을 선택해야 한다고 알려줌

2. 다시 선택 가능한 맵을 알려줌.

그 다음은 Make a move에 대한 명세입니다.

-Use Case Name: Make a Move

-Scope: Bridge Game

-Level: Subfunction

-Primary Actor: Player

-Stakeholders and Interests:

*Player: 주사위를 굴려서 움직일지, 쉴지를 정하고 싶음.

-Preconditions: 해당 플레이어의 턴이여야 함.

-Success Guarantee: 플레이어의 말(piece)이 플레이어의 의도를 따라 map상에서 위치가 갱신됨. 도구 카드를 집으면 점수에 합산함.

-Main Success Scenario:

1. 플레이어에게 턴(turn)이 넘어옴.

2. 플레이어는 주사위를 굴림.

3. 결과로 나온 주사위 값(1~6)을 이용해서 어떻게 움직일 건지 정함.

4. 다리를 건넜다면 다리 카드를 얻고, 마지막 위치에 도구가 있다면

해당하는 점수를 더하고 카드를 얻음.

5. 턴이 넘어감.

게임 시스템은 5번을 게임이 종료되거나 한 명을 제외한 모두가 END 지점에 도달할 때까지 반복함

-Extensions:

2-4a. 만약 플레이어가 다리 카드를 갖고 있고 움직이지 않고 싶다면

1. 다리카드 1장을 소모하고 움직이지 않음.

Use case에 나타난 Display Score는 점수를 확인하는 행위에 관한 것입니다. 이에 관한 명세는 다음과 같습니다.

-Use Case Name: Display Score

-Main Success Scenario:

1. 플레이어가 스코어 보기를 원함.

2. 모든 플레이어들의 점수를 한번에 보여줌.

3. END를 넘어간 플레이어를 따로 표시를 해줌.

4. 현재 순위를 보여줌.

그리고 use case의 마지막인 말 그대로의 Exit Game,게임 종료에 관한 명세는 다음과 같습니다.

-Use Case Name: Exit Game

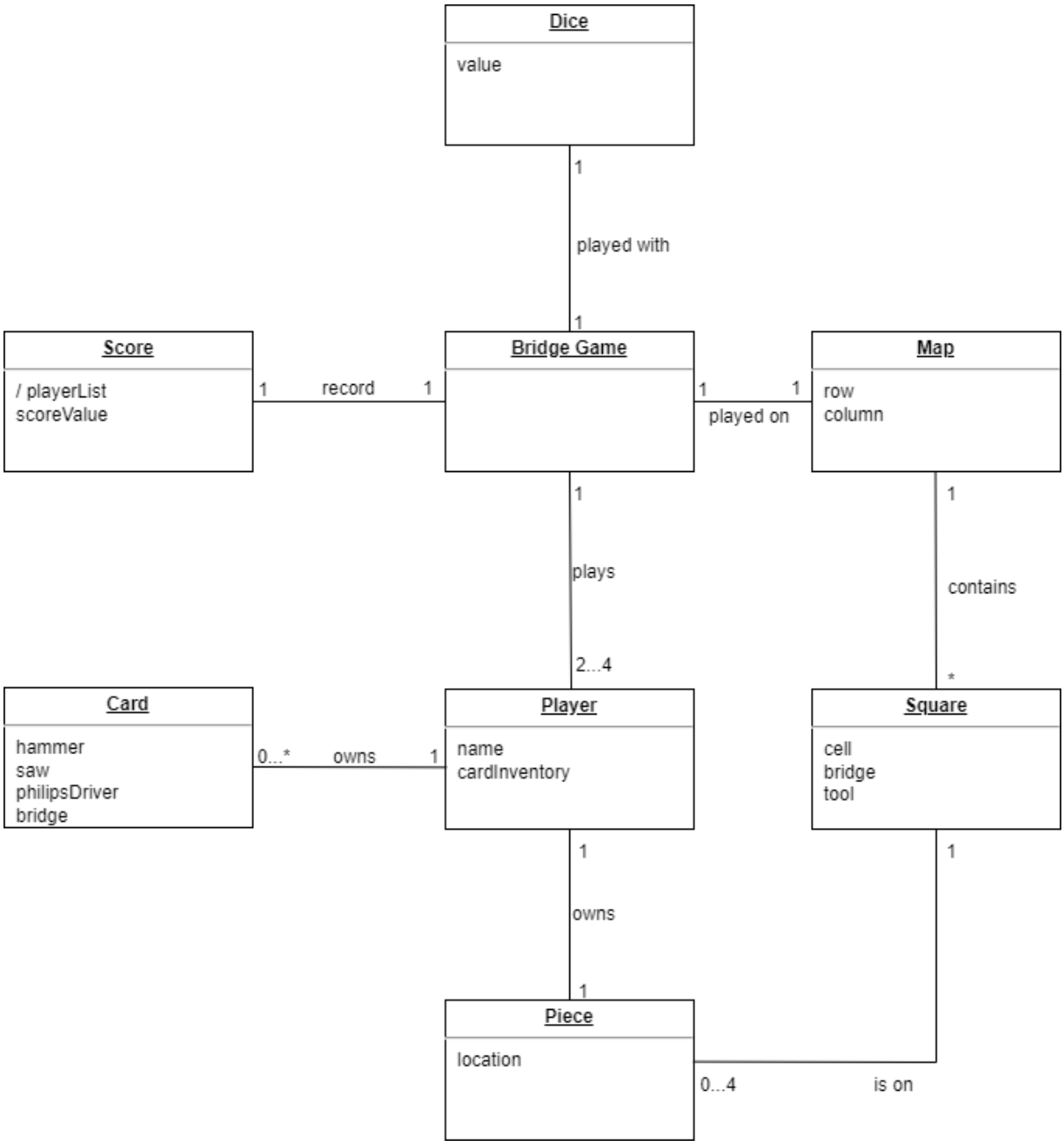
-Main Success Scenario:

1. 플레이어가 종료를 원함

2. 종료버튼을 누르던가 창닫기 버튼이 눌림.

3. 앞서 진행중이던 모든 과정이 멈추고 프로그램이 꺼짐.

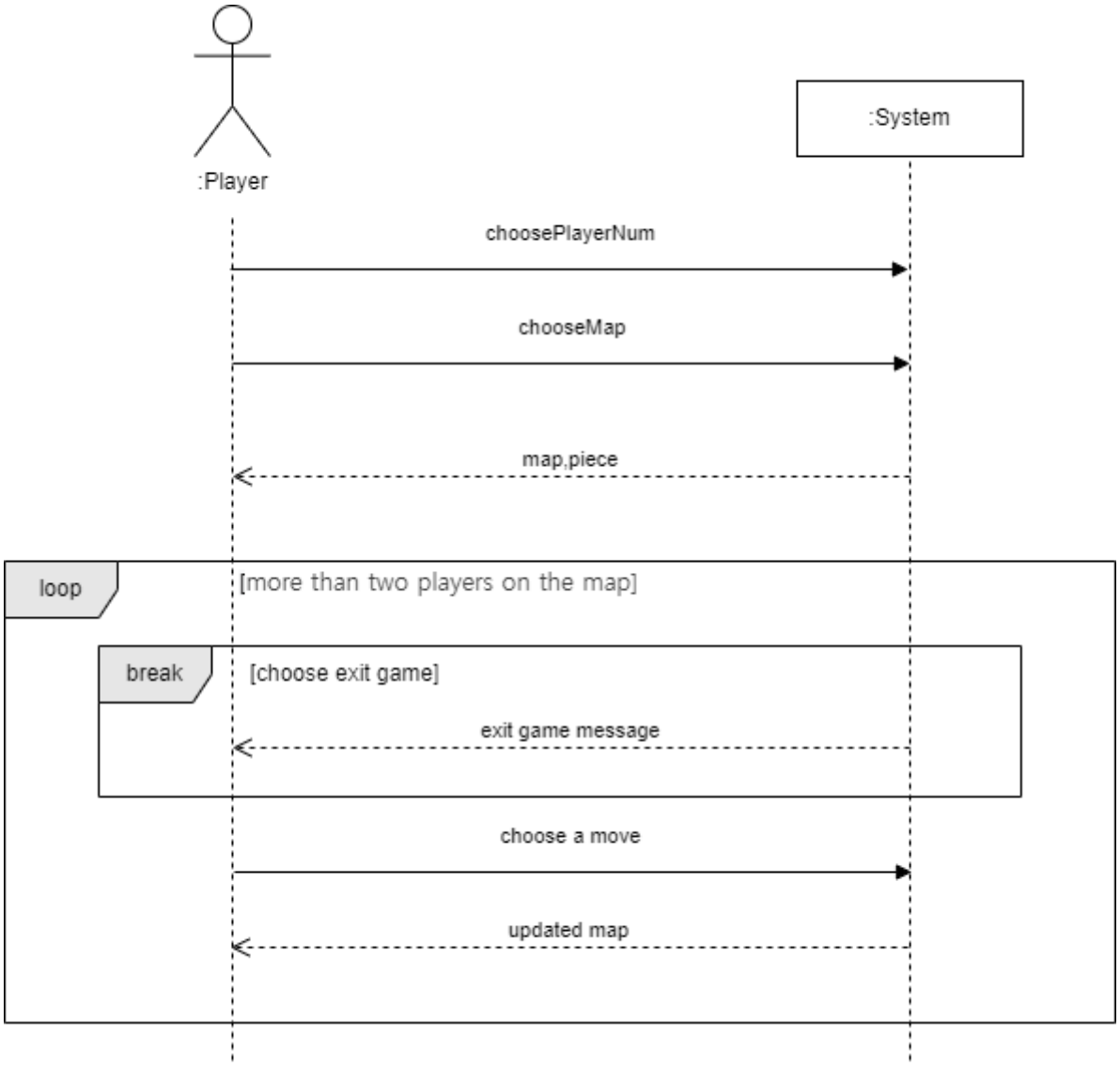
위에서 적은 use case명세를 바탕으로 주요 Domain들을 뽑아내고 그것들을 사용해서 Domain Model을 그렸습니다.



실제로 코드를 만들 때 위 도메인들 중 Score를 제외한 나머지 전부가 클래스로 만들어지게 됩니

다.

그리고 이번에는 use case명세를 바탕으로 이 게임의 Player와 Game이 어떤 식으로 소통하는지에 대한 System Sequence Diagram을 그렸습니다. 여기서 나온 모든 명령은 코드로 구현할 때 일종의 사용자가 시스템에 입력하는 인터페이스 같은 느낌으로 사용되었습니다.



플레이어는 먼저 인원수를 선택하고, 맵을 선택한 후, 그 다음 플레이어가 맵에 1명만 남을때까지 움직임을 선택하고, 게임은 그 움직임 입력을 바탕으로 맵을 업데이트하고를 반복하는 것이 시스템의 큰 동작 로직입니다.

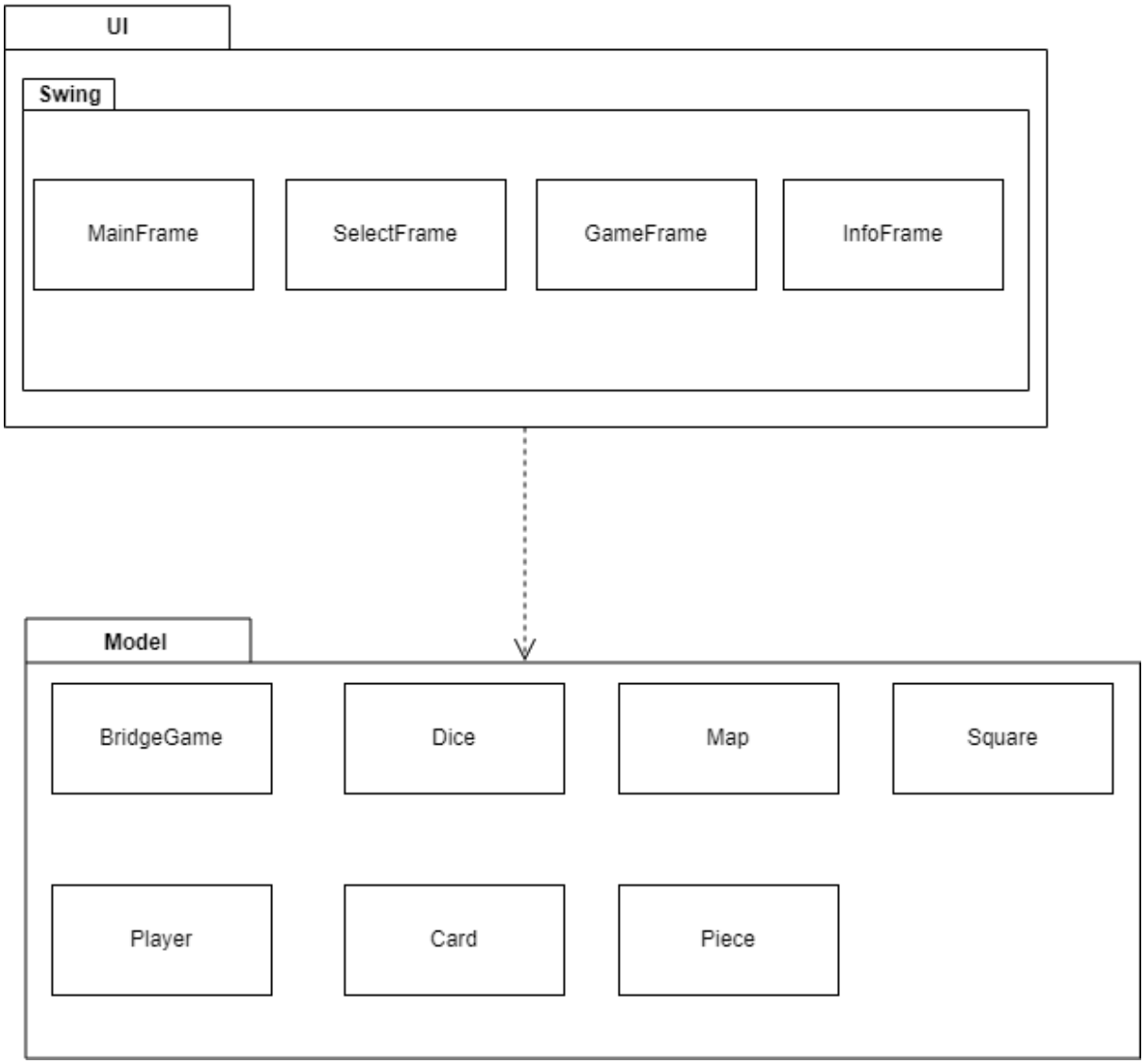
그리고 위 SSD에서 Player가 게임시스템에 전달해주는 명령 중 choosePlayerNum(플레이인원 고르기)는 너무 간단한 기능이라 제외하고 나머지 둘인, chooseMap, chooseMove명령에 대한 Operation Contracts를 작성하였습니다. 이때 적은 operation contracts가 나중에 코딩으로 구현을 할 때 조건과 결과를 빼먹지 않고 프로그램을 만드는데 큰 도움을 주었습니다.

Operation:	chooseMap(mapName: String)
Cross References:	Use Cases: Choose Map
Preconditions:	게임 시작 버튼을 누른 후에 일어남. 플레이어수를 입력 받은 후여야 함. 맵을 고를 수 있도록 목록이 보여져야 함. 이 과정을 거치지 않으면 게임 시작이 되지 않음.
Postconditions:	-Map의 instance인 map이 매개변수로 받은 mapName과 동일한 파일을 바탕으로 생성됨. -map과 관련된 Square의 instance인 square들이 생성됨. -map과 현재 BridgeGame의 association이 발생함. -map과 square의 association이 발생함.

Operation:	chooseMove( )
Cross References:	Use Cases: Make a Move
Preconditions:	map과 piece가 생성된 상태여야함. Card의 인스턴스인 card가 종류별로 생성된 상태여야함. Dice의 인스턴스인 dice가 생성된 상태여야함. Score의 인스턴스인 score가 생성된 상태여야함. 해당 플레이어의 턴이여야함.
Postconditions:	-움직이지 않기를 원한면: 해당 플레이어의 piece의 상태와 score의 상태는 그대로임. -움직이기를 원하면: -dice를 굴리고 그 값은 dice.value에 저장됨 -dice.value의 값만큼 u,d,l,r을 조합하여 입력받은 값을 토대로 piece의 위치가 변경됨. -갱신된 piece위치에 따라서 해당 플레이어의 cardInventory를 갱신함. -마지막으로 score를 갱신함.

[설계 산출물]

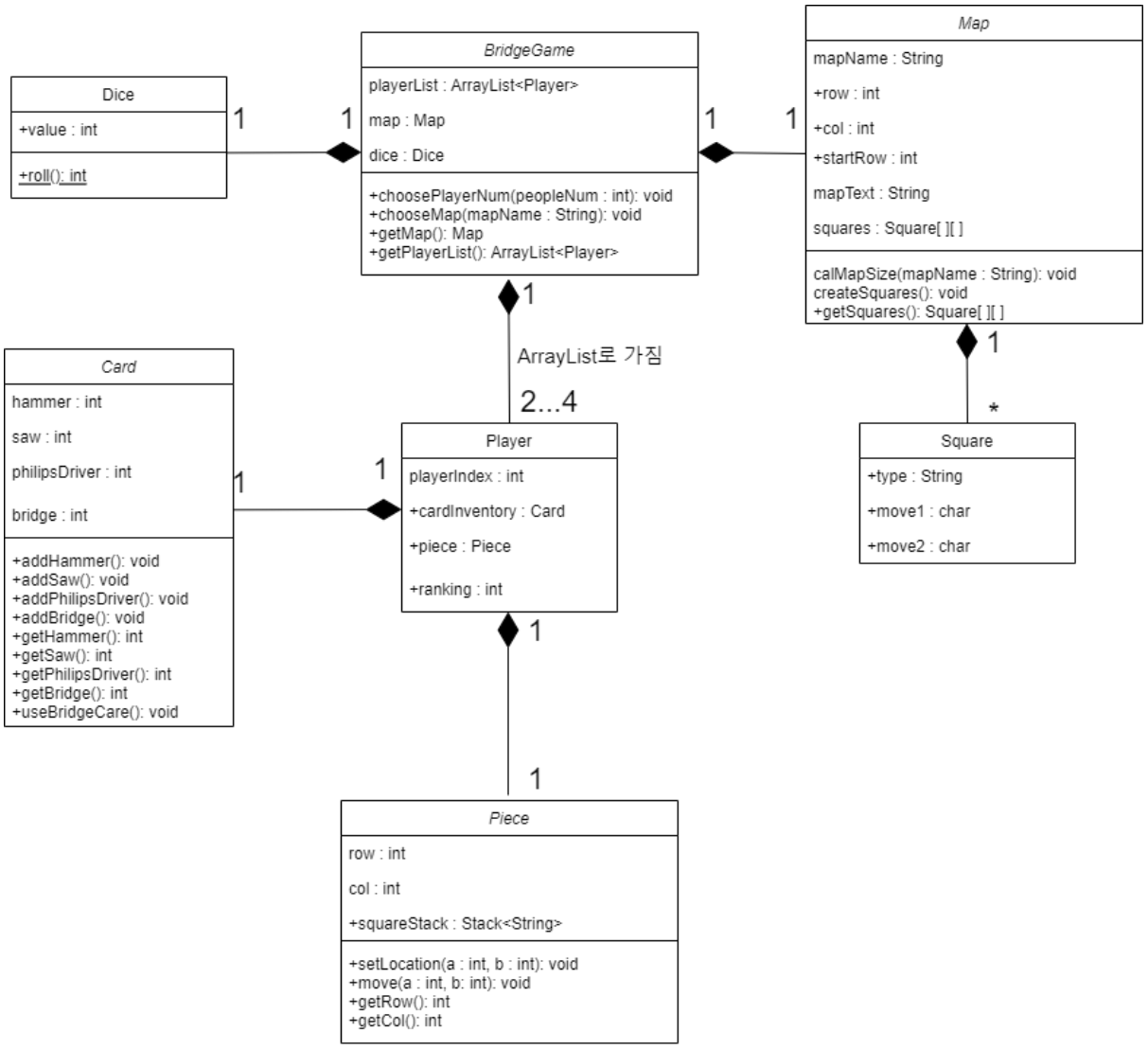
디자인 단계에선 가장 먼저 **Package Diagram을 그려서 UI레이어와 Non-UI(Model)레이어를 나누었습니다.** UI레이어의 변경이 Non-UI(Model)레이어에게 영향을 미치지 않게 하기 위해서 입니다.



UI레이어는 먼저 Java의 Swing을 이용한 4개의 클래스들로 이루어져 있고, MainFrame은 첫 화면을 나타내기 위해서, SelectFrame은 플레이어 인원과 맵 선택, GameFrame을 실제 게임을 위한 화면, InfoFrame은 플레이어의 점수와 카드 보유상황, 게임진행 여부를 나타내기 위한 것입니다.

Model레이어는 Non-UIi레이어와 같은 의미로, 해당 레이어의 클래스들은 UI레이어의 클래스들의 정보를 전혀 갖고 있지 않습니다. 그럼 먼저 Model 레이어의 클래스들에 대해 설명하겠습니다.

앞서 생성된 다양한 산출물들을 중 **Domain Model의 Domain들을 가지고 Design Class Diagram을 만들었습니다.** 해당 클래스들은 전부 Model레이어에 속합니다. 도메인 모델에는 있던 Score 도메인은 구현할 때 클래스로 만들 필요가 없고 UI부분에서 controller역할을 해주는 곳에서 처리해주는 것이 맞다고 판단하여서 제외하였습니다.



코드를 구현할 때 크게 Model과 UI부분 둘로 나눴는데 위 클래스다이어그램은 Model에 해당하는 클래스들을 나타낸 다이어그램입니다. 설명을 해보자면 먼저 중심적인 역할을 해주는 class인 **BridgeGame 클래스**가 있습니다. 이것은 필드로 갖는 playerList(ArrayList타입)에 입력받은 플레이어 수 만큼의 Player 인스턴스를 생성해주고, map에는 사용자가 고른 map을 로드합니다. dice는 말 그대로 주사위 클래스의 인스턴스 입니다. 첫번째 메소드인 choosePlayerNum(peopleNum)은 입력받은 플레이어 수 만큼 playerList에 Player인스턴스를 추가해 주는 역할을 합니다. chooseMap(mapName)메소드는 맵을 플레이어가 고르면 해당 맵을 로드하는 역할을 합니다. getMap( )은 생성된 map인스턴스를 반환해주는 기능이고, getPlayerList( )는 생성된 플레이어들의 목록(ArrayList)을 반환해주는 기능입니다. 그리고 결과적으로 이 BridgeGame클래스는 Dice클래스

의 인스턴스를 생성하고 Map의 인스턴스도 생성하고 Player의 인스턴스도 생성하는, 중추적인 역할을 하는 클래스입니다.

그럼 이번엔 **Dice클래스**에 대해서 알아보면, 먼저 필드로 int value를 하나 갖고 있습니다. 이는 무엇이냐면 Dice클래스의 메소드인 int roll( )을 실행시켰을 때 나온 1~6의 랜덤값을 저장하는 변수 입니다. 비교적 간단한 클래스입니다.

이번엔 **Map클래스**에 대해서 알아보겠습니다. Map클래스는 BridgeGame클래스에 의해서 생성되는데, 생성될때 map=new Map(mapName); 의 형식으로 인스턴스가 만들어집니다. 즉 생성자의 매개변수로 플레이어가 선택한 맵의 이름을 받습니다. 그러면 해당 맵의 이름을 가진 파일을 찾아서 그 속 내용을 읽어옵니다. 그리고 이 읽은 내용은 필드인 String mapText에 저장됩니다. 그 후, 이 mapText를 파싱하여서 2차원배열로 맵을 생성할 때 row와 column이 몇 개여야 하는지 알아내고 그 값을 필드인 row와 col에 넣어줍니다. 그리고 Start셀의 위치가 몇번째 row인지도 알아내서 필드변수인 startRow에 넣어줍니다. 추후에 row와 col과 startRow값이 UI부분의 2차원 배열 모양의 GUI를 생성하는데 사용이됩니다. 여기까지는 calMapSize(mapName) 메소드의 내용입니다.

그다음은 createSquares( )메소드를 실행하는데 이는 맵을 실제로 구성하는 2차원 배열의 각 칸인 Square클래스의 인스턴스를 만들어주는 기능을 합니다. 여기서 생성된 square인스턴스는 Map클래스의 필드인 Square[ ][ ] squares에 2차원 배열로 저장됩니다. 추후에 이 2차원 배열을 사용해서 해당 칸이 어떤 타입(일반 셀,다리,톱,드라이버,망치,시작,끝,벽)인지 또 어떤 움직임이 가능한지 분석하는데 사용됩니다. 여기까지 나온 메소드는 모두 Map클래스의 생성자속에서 실행되는 메소드들입니다. 즉, Map 클래스의 인스턴스가 생성됨과 동시에 기본적으로 Map에 관한 정보가 세팅이 되는 것입니다. 그리고 마지막 남은 getSquares( )메소드는 위에서 말한 square[ ][ ] 2차원 배열 인스턴스를 반환해 주는 역할을 합니다. UI부분에서 움직임을 구현하기 위해 필요해서 만들었습니다.

이번엔 **Square클래스**에 대해서 알아보겠습니다. Square클래스는 Map클래스의 인스턴스가 생성되는 과정에서 인스턴스로 만들어지게됩니다. 메소드는 없고, 맵에서 어느 특정 칸의 속성을 나타내는 String type필드(S,E,C,B,b,H,S,P,bridge,blank 이렇게 총 9개가 옴. S는 시작 셀, E는 end셀,C는 일반 셀,B는 다리 왼쪽 셀,b는 다리 오른쪽 셀,H는 hammer셀,S는 saw셀,P는 philipsdriver 셀,bridge는 다리 셀,blank는 길이없는 빈 셀을 의미함)와 해당 칸의 가능한 움직임을 알려주는 char move1,move2(U,P,L,R,u,p,l,r 또는 움직일수 없을 때 x값을 갖음)로 구성되어 있습니다.

이번엔 **Player클래스**에 대해서 알아보겠습니다. 이 클래스의 인스턴스는 BridgeGame클래스의 choosePlayerNum(int peopleNum)메소드에 의해서 ArrayList의 타입으로 저장됩니다. 즉, 입력받은 플레이어 수만큼 생성되는 것입니다. 해당 클래스는 메소드는 없고 필드로만 구성되어있는데, int playerIndex는 몇번째 플레이어인지 저장하기 위해 존재하고, Card cardInventory는 해당 플레이어의 보유중인 카드를 저장하기 위해 존재합니다. Piece piece는 해당 플레이어의 말을 저장하기 위해 존재하고 int ranking은 해당 플레이어가 end지점에 도달했을 때 몇 번째로 도달하였는가를 저장하기 위해서 존재합니다.(추후 스코어 계산 때 쓰임). 그리고 제한 조건으로 최대 2~4명만 생성

할 수 있고 이 제한은 UI레이어에서 제한을 해줍니다.

이번엔 **Card클래스**에 대해서 알아보겠습니다. Card클래스의 인스턴스는 Player클래스의 인스턴스가 생성되는 과정에서, 즉 Player클래스의 생성자가 실행될 때 만들어지게 됩니다. 왜냐하면 한명의 player와 Card cardInventory는 1:1로 매칭되어야 하기 때문입니다. 이렇게 해서 만들어진 인스턴스인 cardInventory는 카드를 종류별로 몇 장 갖고 있는지를 저장합니다. 그래서 필드로 int hammer, int saw, int philipsDriver, int bridge를 갖고 있습니다. 여기에 저장된 값들은 나중에 score를 보여주거나 움직이지 않음을 선택했을 때 bridge카드를 소모하기 위해 사용됩니다. 이 클래스에 있는 메소드는 각 카드를 add해주거나(카드를 획득 했을 때 실행하여서 카드 개수를 1장 늘려줌) get(해당 카드가 몇 장 있는지 반환해주는)메소드들과 움직이지 않음을 선택했을 때 다리 카드를 사용하기 위한 useBridgeCard( )로 이루어져 있습니다.

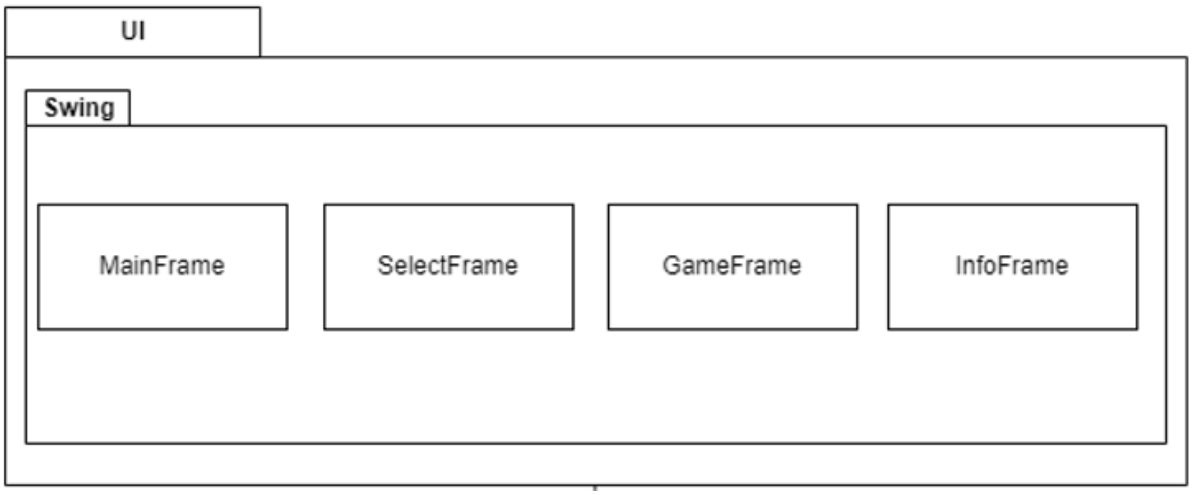
마지막으로 **Piece클래스**에 대해서 알아보겠습니다. Piece클래스는 각 플레이어의 말을 나타내는데, 그렇기 때문에 해당 클래스의 인스턴스도 마찬가지로 Player와 1:1로 매칭되어야 합니다. 그래서 Player클래스의 생성자가 실행될 때 만들어 지게 됩니다. 필드로는 해당 말이 지금 2차원 셀들(클래스로는 Square)로 구성된 맵에서 어느 위치에 있냐를 저장하기 위해 int row, int col을 갖습니다. 그리고 Stack<String> squareStack도 필드로 갖는데 이는 해당 말의 최근 움직임을 지속적으로 기록하여 다리를 건넜는지 안건넜는지를 체크 하기 위해 스택을 이용하였습니다. 예를 들어서 스택에 담긴 최근 3개의 값이 B,bridge,b 이거나 b,bridge,B이면 다리를 정상적으로 건넌 것이라 판단할 수 있고, 그렇지 않은 경우에는 전부 다리를 건넌 것이 아니라고 판별할 수 있게 됩니다. 메소드로는 해당 말의 위치를 지정해주는 setLocation(a,b), 해당 말을 움직여서 row와 col의 값에 연산을 해주는 move(a,b), 현재 말의 row값을 알려주는 getRow( ), 현재 말의 column값을 알려주는 getCol( )로 이루어져 있습니다.

여기까지가 해당 프로그램의 Model부분, 즉 NON-UI부분의 클래스에 대한 설명이었습니다.

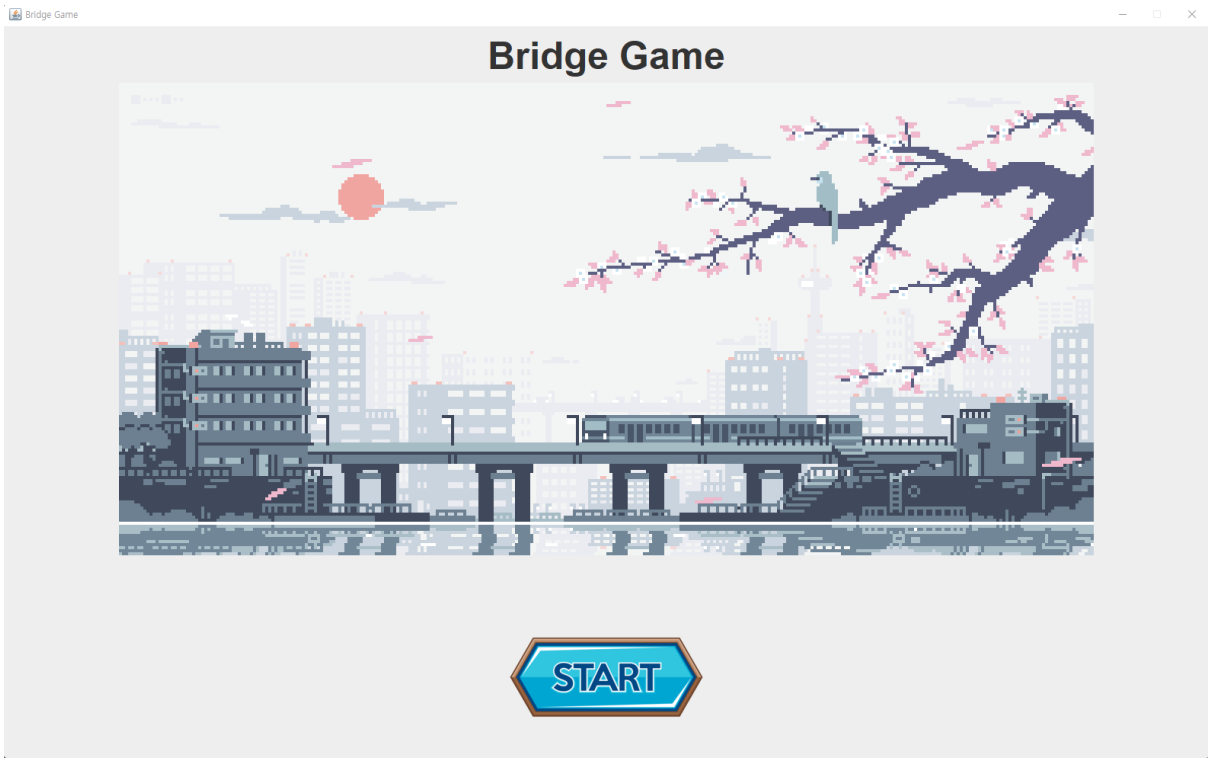
UI부분에서는 버튼 및 값 입력을 받아서 이 Model레이어의 클래스들의 값을 업데이트 시켜주고, 업데이트 된 값을 UI레이어에서 받아서 다시 GUI로 표시해주게 됩니다. 즉, UI레이어와 Model레이어의 분리를 지키고자 위와 같이 구분하여 설계하였습니다. Model레이어의 모든 클래스는 UI레이어의 그 어떤 클래스에 대한 정보도 갖고 있지 않습니다.

이번엔 UI 레이어에 속하는 클래스들의 간단한 설명과 함께 작동 로직을 설명하겠습니다.

앞서 설명했듯이 UI레이어의 클래스는 다음과 같은 4개의 클래스로 이루어져 있습니다. 모두 자바의 Swing을 사용하고 있습니다.



먼저 MainFrame을 설명하겠습니다.



다음과 같이 구성되어 있으며, 가운데 이미지는 gif파일로 움직이는 그림입니다.

START버튼을 누르면 단순히 SelectFrame으로 넘어가는 기능만 하고 있습니다. Model과의 연관성은 없습니다. 그럼 이제 START버튼을 누르고 다음 프레임으로 넘어가겠습니다.

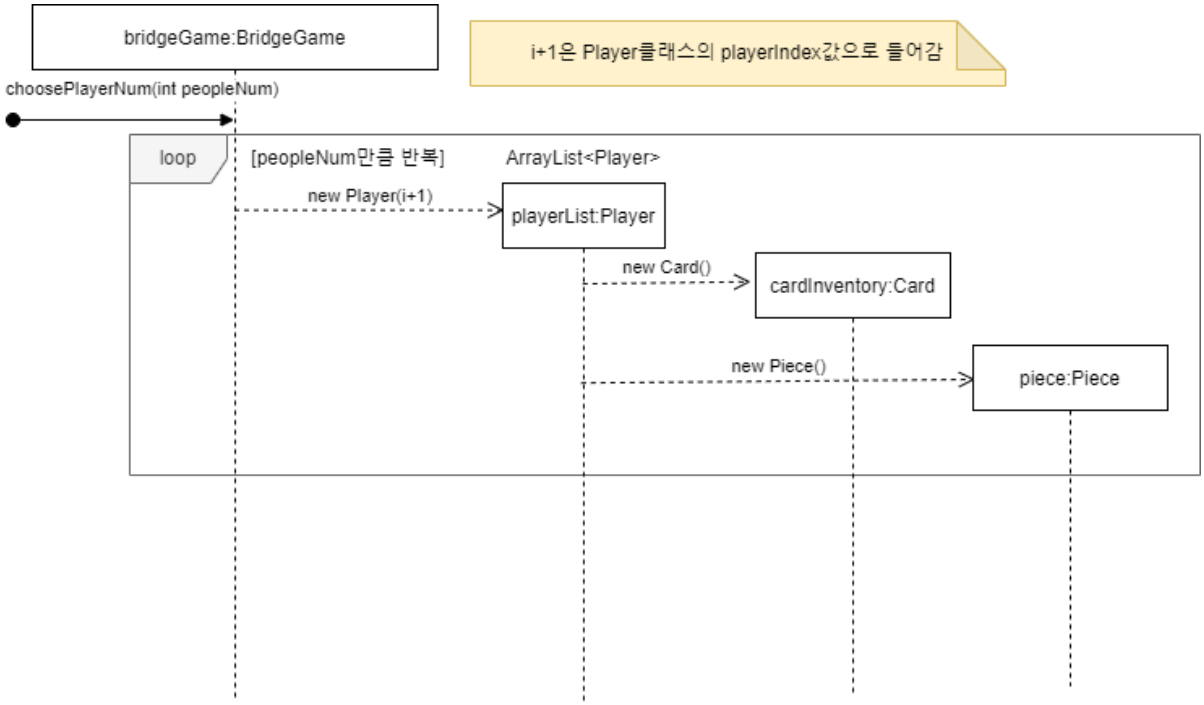
이번엔 SelectFrame입니다.

위와 같이 플레이어 수와 맵을 JComboBox를 통해서 선택할 수 있도록 구현해 놨습니다.

플레이어 수는 2~4명으로 제한되고, 맵은 프로젝트 폴더 속 map폴더에 위치한 파일들을 자동으로 읽어와서 목록을 보여줍니다. 선택한 후 아래 체크버튼을 누르면 SelectFrame클래스가 가지고 있는 Model레이어의 BridgeGame클래스의 인스턴스인 bridgeGame의 메소드인 bridgeGame.choosePlayerNum(peopleNum); 과 bridgeGame.chooseMap(mapName);을 부르게 됩니다. UI레이어에 속한 클래스이기 때문에 모델의 인스턴스를 갖을 수 있게 설계했습니다.

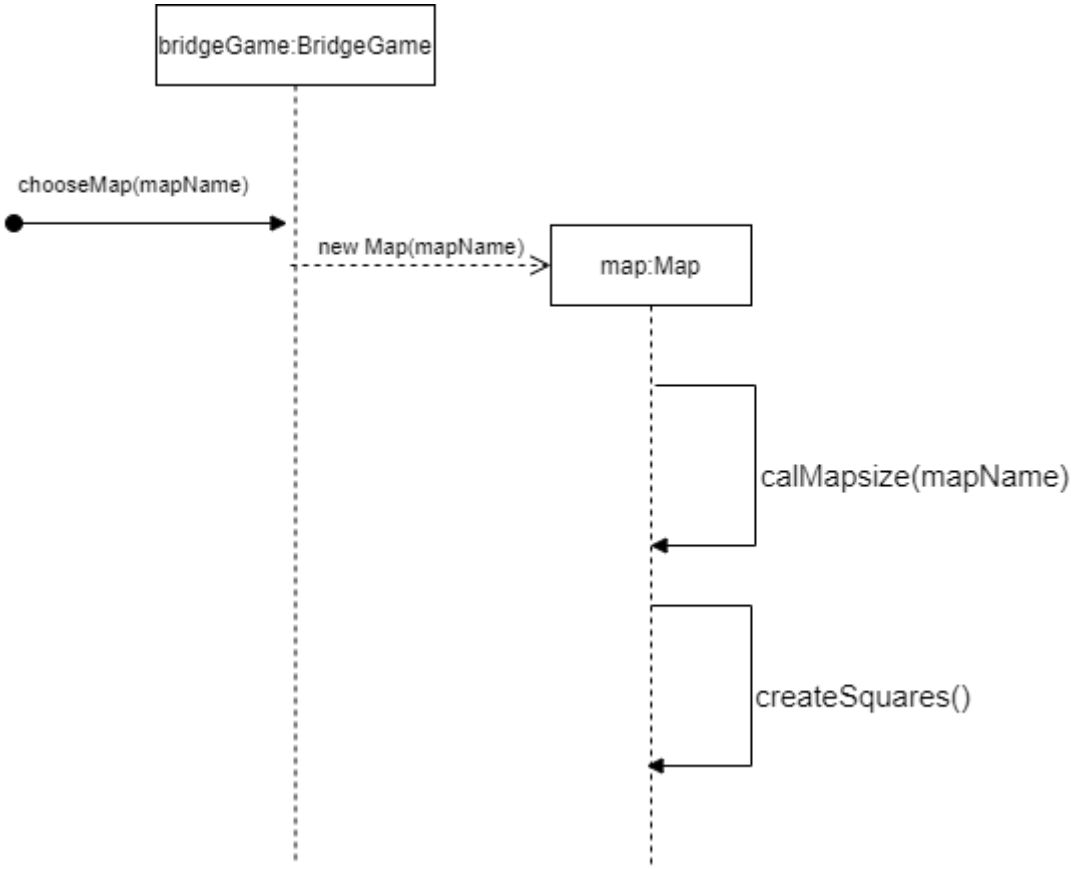
그럼 이제 체크버튼을 눌렀을 때 실행되는 위의 두 메소드가 어떤 식으로 진행되는지 Sequence Diagram을 통해서 알아보겠습니다.

먼저 choosePlayerNum(peopleNum)메소드입니다.



체크 버튼을 누르면 실행되는 메소드이며, 콤보박스를 통해 입력 받은 플레이어의 수가 peopleNum값으로 들어가게 됩니다. 메소드가 실행되면 bridgeGame에서 for문을 통해 Player클래스 인스턴스를 생성하여 ArrayList타입 변수인 playerList에 add해줍니다. 이때 player인스턴스를 만들 때 마다 new Card()와 new Piece()를 통해서 해당 player인스턴스의 필드인 cardInvetory와 piece를 초기화 해줍니다. 이렇게 peopleNum만큼 반복하면 결과적으로 입력받은 플레이어 수 만큼의 플레이어가 생성되고 각 플레이어객체는 카드보관함과 자신의 말을 갖게 됩니다.

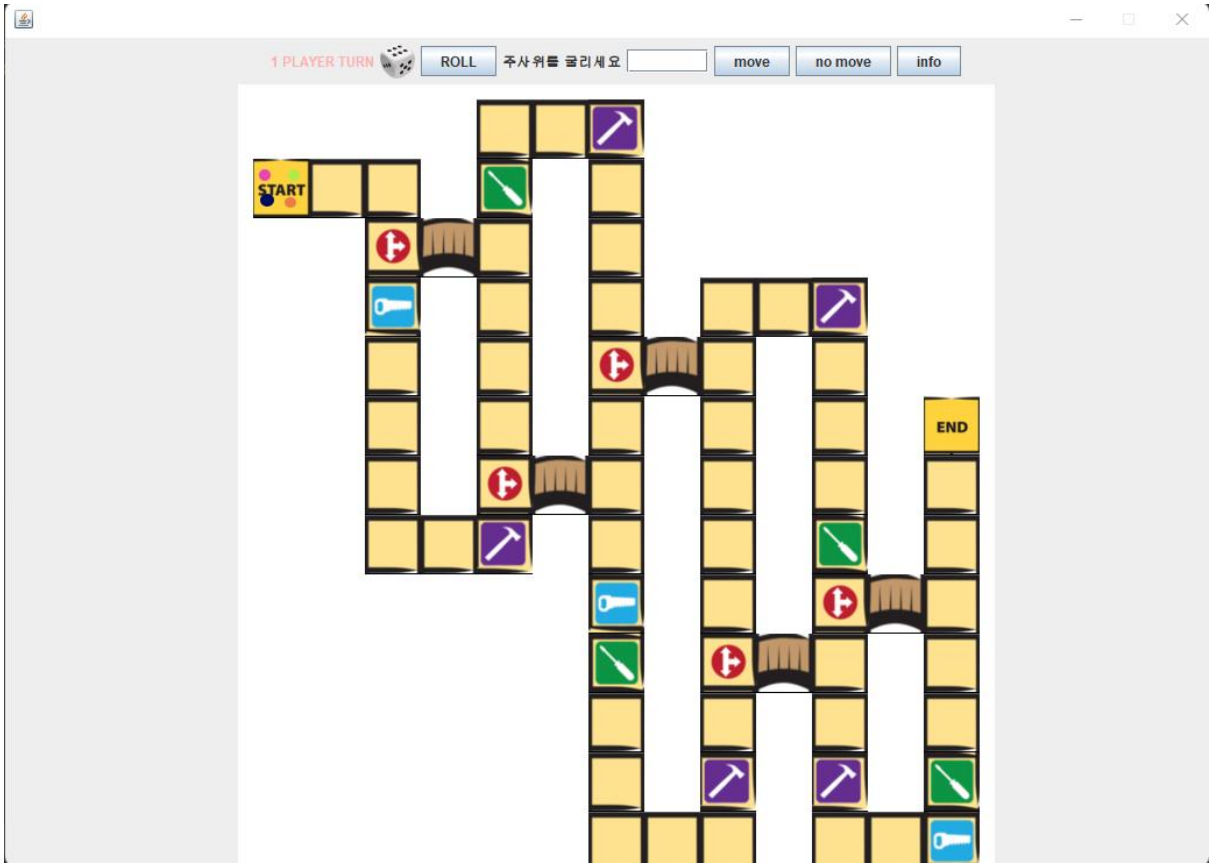
그 다음은 chooseMap(mapName)메소드입니다.



마찬가지로 체크버튼을 누르면 실행되는 메소드이며, 콤보박스를 통해 입력받은 map이름이 mapName값으로 들어가게 됩니다. 그럼 bridgeGame인스턴스에서 new Map(mapName)을 통해 map 인스턴스를 생성하는데 먼저,calMapsize(mapName)메소드를 통해서 해당 이름을 가진 맵 파일의 내용을 읽어오고 파싱해서 맵의 행의 개수와 열의 갯수와 시작점 인덱스를 얻습니다. 그 후 createSquares()메소드를 사용해서 2차원 배열의 squares변수를 만들어서 맵의 한칸한칸에 대한 정보를 저장합니다.

여기까지의 과정을 마치면 맵 생성이 완료되고, 플레이어의 기본 설정이 완료됩니다. 그럼 이제 GameFrame으로 넘어가게 됩니다.

이번엔 GameFrame입니다.



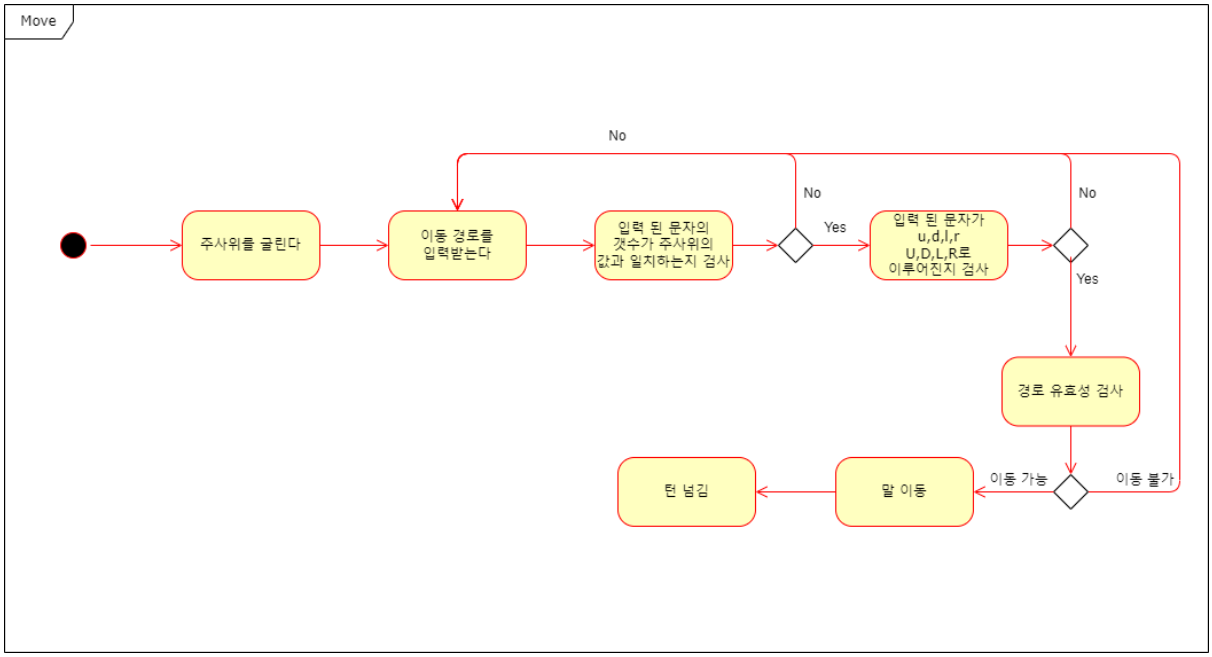
map파일의 내용에 따라 동적으로 맵을 생성해줍니다. 해당 게임의 맵 형식에만 맞는다면 미리 저장된 맵 이미지가 아닌, 각 칸마다 2차원배열에 따라서 동적으로 생성하는 방식입니다.

이 프레임에서 사용자가 할 수 있는 행동은 크게 3가지 입니다.

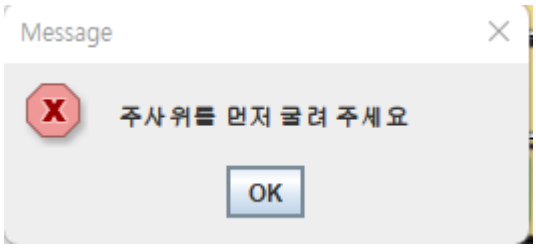
1. 움직인다
2. 움직이지 않는다
3. 현재 종합 정보를 본다.

이 3가지 행동에 대한 Activity Diagram을 각각 그려서 어떤 로직으로 게임을 구현하였는지 설명하겠습니다.

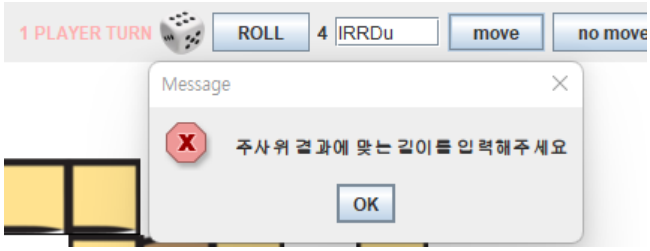
먼저 움직인다는 행동을 수행할 때의 로직입니다.



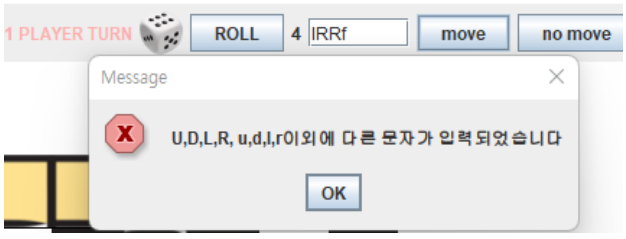
이때 주사위를 굴리지 않고 move버튼을 누르면 경고창이 뜹니다.



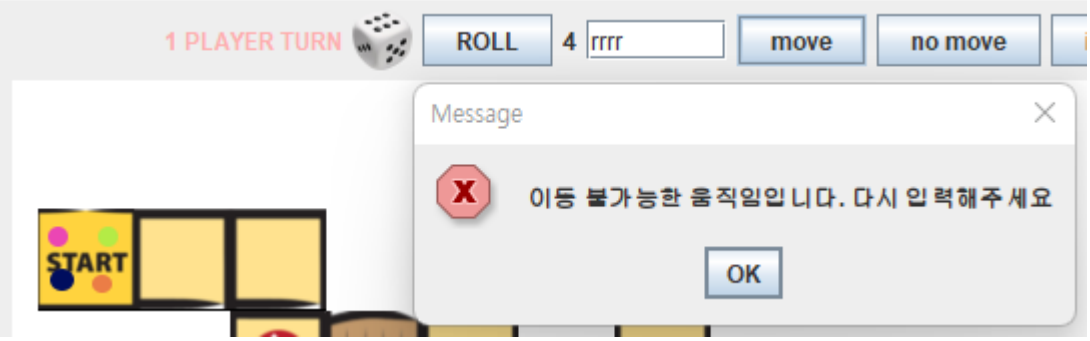
입력된 문자의 길이가 주사위의 값과 다를 경우 경고창이 뜹니다.



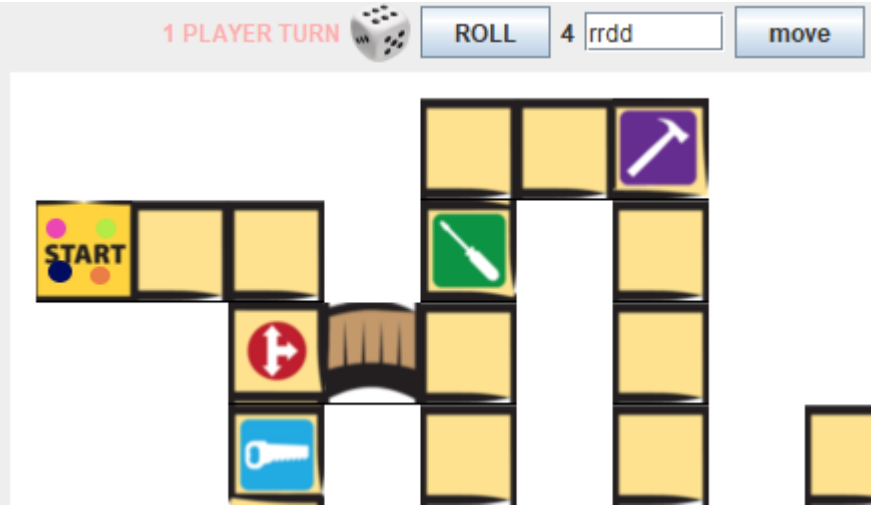
입력된 문자가 U,D,L,R,u,d,l,r이 아닐 경우 경고창이 뜹니다.



갈 수 없는 경로일 때 경고창을 띄웁니다. 아래 사진의 경우 rr후 r을 한번 더 하는 순간 빈칸이기때문에 움직일 수 없기 때문입니다.



모든 것이 정상적이라고 판별될 경우 움직이게 됩니다.



이 경우 입력 갯수, 입력 문자, 경로 유효성 모두 정상이기 때문에 move를 누르면

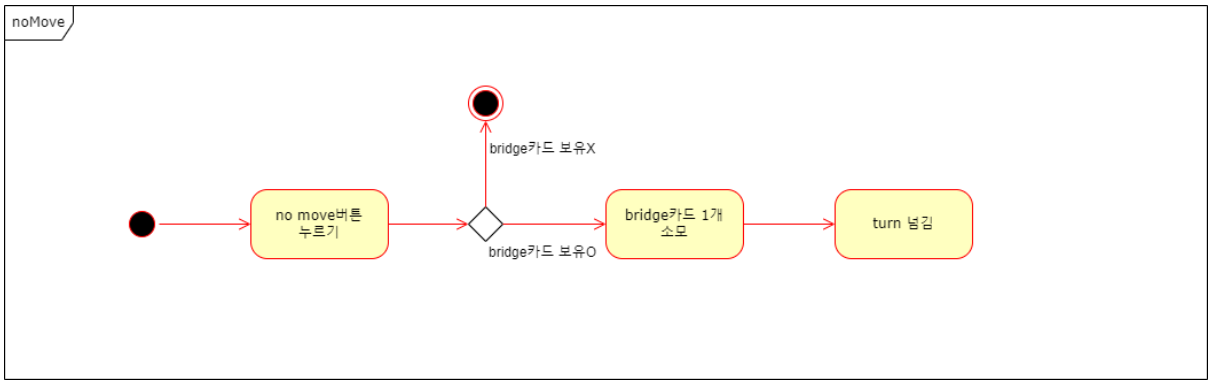
이렇게 1player의 말이 움직인 것을 확인할 수 있습니다. 그리고 위쪽 턴을 보면 2Player의 턴으로 턴이 이동한걸 볼 수 있습니다. 그리고 위 사진 같은 경우 player1이 Saw셀에 도착했으므로

info버튼을 눌러서 확인해보면

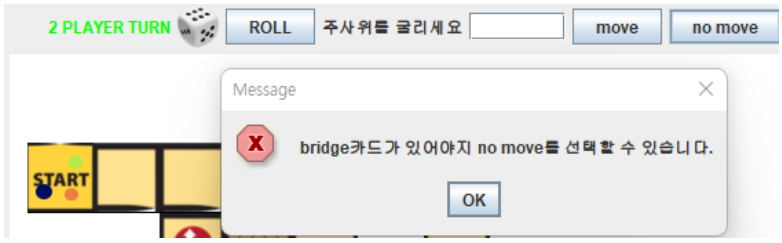
톱 카드1개를 정상적으로 획득하였고, 총 점수가 톱1개에 해당하는 점수인 3점만큼 획득한 걸 볼 수 있습니다.

이러한 움직임 과정의 적합성을 검사하고 실행하는 과정에서 Model레이어에 있는 Dice클래스를 통해 주사위 값을 얻고, Map클래스와 Square클래스를 통해서 맵 정보를 얻고, Player클래스,Piece클래스를 통해서 말을 이동시킨 값을 저장하고, Card클래스에 얻은 카드를 저장하는 등 UI레이어에서 Model레이어의 클래스들을 활용하여 게임을 구동하고 있습니다.

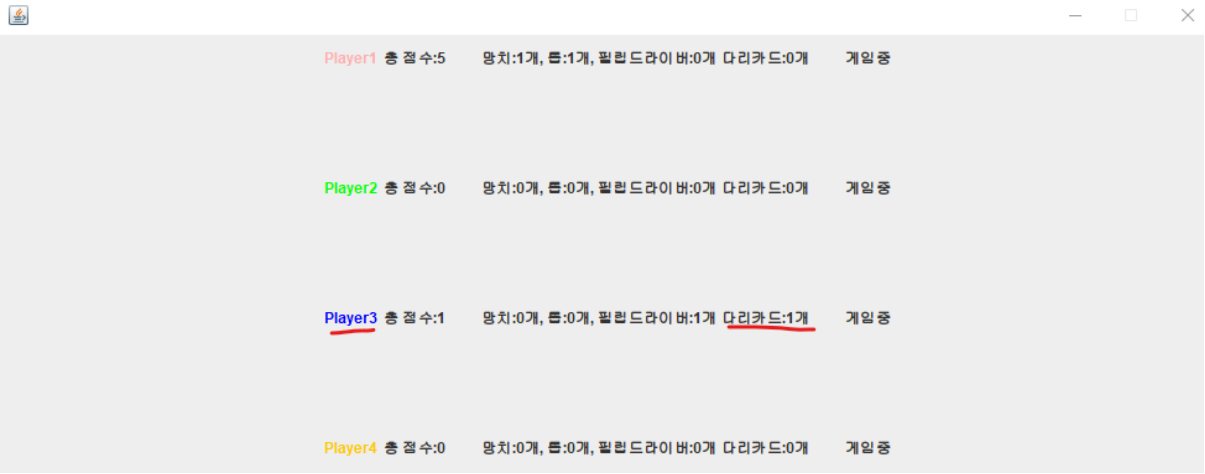
이번엔 움직이지 않는다는 행동을 수행할 때의 로직입니다.



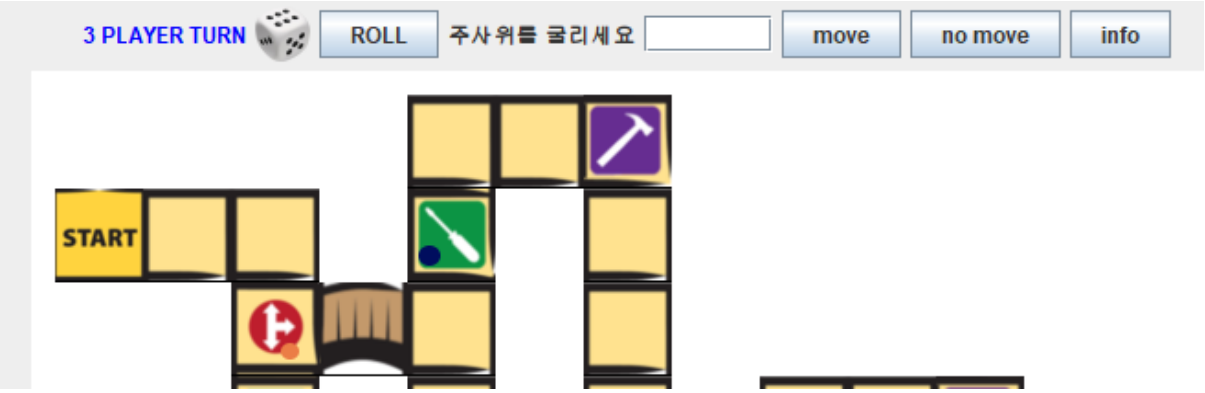
No move버튼을 누르면 먼저 해당 플레이어가 bridge카드를 보유중인지 검사하고 없다면 경고창이 나타납니다.



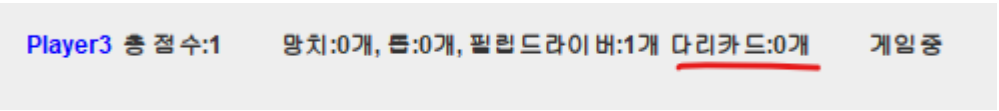
bridge카드가 있다면 보유중인 bridge카드를 1개 소모하고 턴을 넘깁니다.



Ex)현재 Player3은 다리카드 1개 보유중

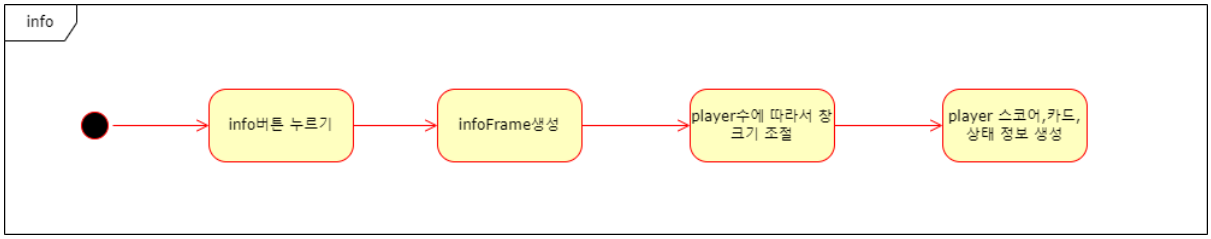


이 상태에서 no move를 누르면 player3(파란색 동그라미)는 움직이지 않고 다리카드를 1개 소모하고 턴이 넘어감.



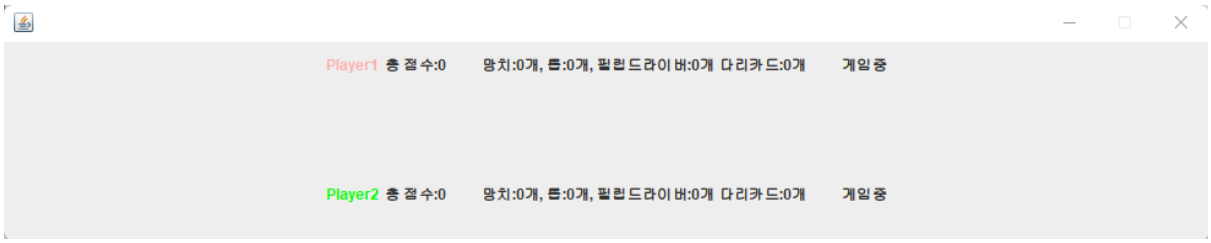
이 경우도 Model레이어의 클래스인 Player클래스의 cardInventory 필드를 통해서 bridge 카드가 있는지 확인하고 처리를 하기 때문에 UI레이어에서 Model레이어의 정보를 사용한 것이 되겠습니다.

마지막으로 현재 종합 정보를 본다의 로직입니다.



info버튼을 누르면 새로운 Frame인 infoFrame클래스가 생성이됩니다. 이때 현재 플레이중인 player수에 따라서 창의 크기가 동적으로 설정됩니다. player수는 bridgeGame인스턴스의 playerList.size( )을 통해 얻을 수 있습니다. 그리고 player와 piece와 cardInventory를 통해서 총 스코어와 갖고 있는 카드에 대한 정보, 현재 게임중인지 end셀에 도달했는지에 대한 정보를 표시해 줍니다. 이것 또한 UI레이어에서 Model레이어의 메소드 및 필드를 사용한 예 입니다.

Ex)플레이어가 2명일때의 info창



이 외 구현에 대한 더 자세한 설명은 영상으로 설명하도록 하겠습니다.

감사합니다.