
소프트웨어 공학

- OOAD를 이용한 Bridge 게임 개발 -



과목명	소프트웨어공학
교수명	이찬근
제출일	2022.06.10
학 번	20175371
학 과	컴퓨터공학과
이 름	강명석

목차

요구 정의 및 분석 산출물	3
UseCase Diagram	3
UseCase Text.....	3
Domain Model	4
System Sequence Diagram	4
Business Rules(Domain Rules).....	5
설계 산출물	7
Package Diagram	7
Sequence Diagram	8
Class Diagram	12
UI Prototype	14
구현 산출물	15
소스코드 설명	15
프로그램 사용법	21
프로젝트 요구사항 비교표	34

요구 정의 및 분석 산출물

UseCase Diagram

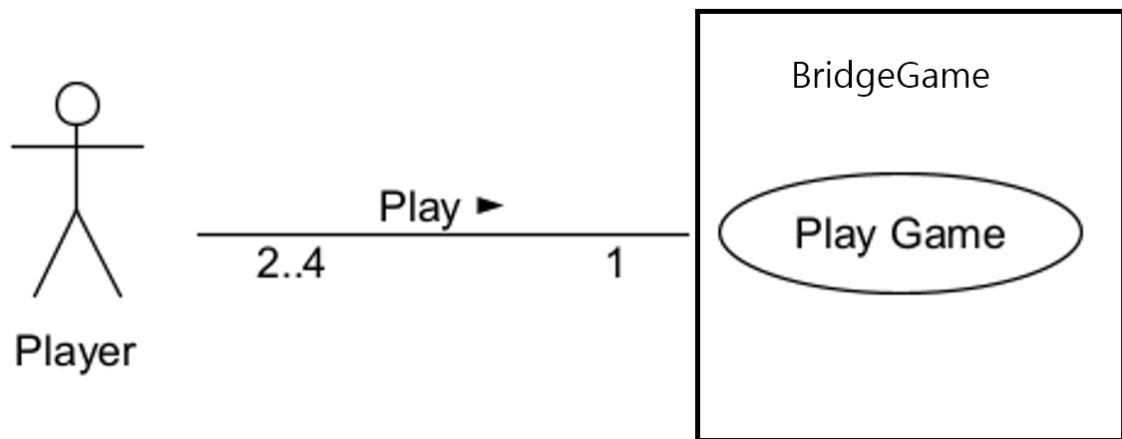


그림 1 - UseCase Diagram

본 프로젝트에서는 요구사항이 명확하고, 시스템이 작기 때문에 단 하나의 유스케이스로 요구사항을 기술할 수 있다. 외부 User(Player) 입장에서 해당 프로그램으로부터 하고자 하는 것은 "Play Game"이다. 특히, 프로젝트 요구사항 중 '맵을 만들기', '주사위를 굴리기', '이동하기' 등은 하나의 유스케이스로 보는 것이 아니라, 게임하기 내에서의 하나의 시나리오에 포함되는 행위로 보는 것이 적절하다. 또한 '다리에서 건널지 선택하기', '매 턴마다 쉴 수 있기' 등의 모든 게임을 또한 UC에서 나열하는 것 보다, Business Rules(Domain Rules)에서 기술하는 것이 더 자연스럽다.

UseCase Text

Scope : BridgeGame Application

Level : user goal

Primary Actor : Player

Stakeholders and interests

- Player : Bridge게임을 콘솔화면, GUI로 플레이 하고 싶음.

Main success Scenario

1. Player가 게임을 플레이할 인원 수와, 맵을 선택한다.
2. Player가 게임을 시작한다.
3. Player는 자신의 턴에 주사위를 던진다.
4. Player는 자신의 주사위 수 내에서 갈 수 있는 만큼 이동한다.
5. Player는 자신의 턴을 다음 Player에게 넘겨준다.

플레이 하고 있는 사용자가 한 명만 남을 때까지 3,4,5번 과정을 반복한다.

Special Requirements

- Application은 콘솔 모드와, GUI 모드 2가지를 모두 제공해야 한다.

Domain Model

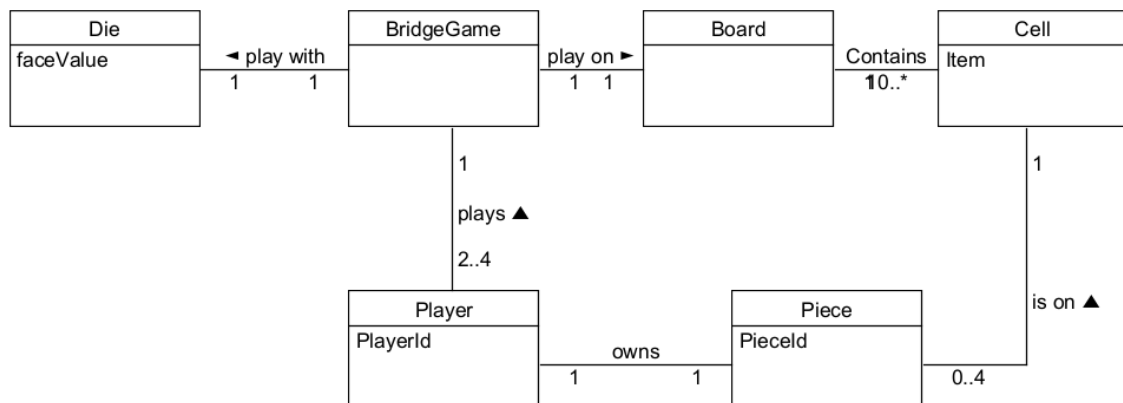


그림 2 Domain Model

도메인 모델은 현실 세계를 개념적인 클래스로 표현한 것이다. 2명에서 4명의 플레이어는 BridgeGame을 플레이한다. 하나의 Game안에는 하나의 주사위(Die)와, 하나의 Board가 존재한다. 하나의 보드는 또한 최소 10개 이상의 셀들로 이루어져 있으며, 몇 개의 셀은 아이템을 가지고 있다. 그리고 Player는 보드판 위의 말들을 소유하고 있다.

System Sequence Diagram

SSD는 외부인 입장에서 시스템 내부 로직을 고려하지 않고, 시스템이 무엇을 하는지에 초점을 맞춰서 그린 다이어그램이다. UseCase에서 시나리오를 SSD로 표현하면 다음과 같다.

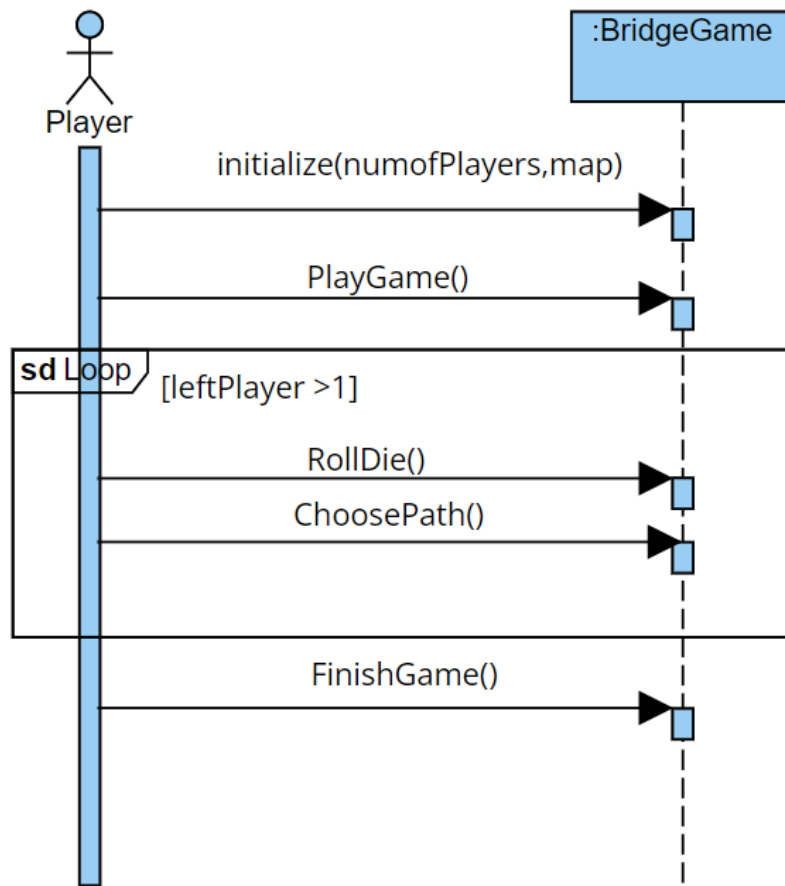


그림 3 System Sequence Diagram

Business Rules(Domain Rules)

어느정도 규모의 프로젝트라면, 기능적 요구사항 뿐만 아니라 비기능적 요구사항을 모두 충족시키기 위해서 Supplementary Specification을 작성한다. 이전 UseCase에서 언급했던 게임의 룰도 Supplementary Specification의 세부 항목중 Application Specific Business Rule로써 존재한다. 하지만 본 프로젝트에서는 Supplementary Specification을 따로 작성하는 것은 불필요한 문서만 작성하는 것으로 판단되어, 그 중 Application Specific Business Rule만 따로 작성한다.

ID	Rule
Rule1	플레이어는 자신의 턴이 되면, 쉴지(Stay) 혹은 주사위를 굴릴지를 결정할 수 있다. 쉬기로 결정한 경우 다리 카드 1장을 시스템으로 반납한다. 이때, 만약 다리카드가 0장인 플레이어는 다리 카드의 반납 없이 턴을 넘기도록 한다.
Rule2	플레이어는 자신의 턴이 되면 주사위를 굴린다. 이때, 플레이어가 이동 가능한 칸은 주사위 수에서 현재 플레이어가 가진 다리 카드 개수 만큼을 뺀 만큼 이동 가능하다. 만약 그 값이 음수라면 해당 플레이어는 그 턴을 종료한

	다.
RULE3	플레이어는 이동 가능한 칸 수 만큼 앞 혹은 뒤로 이동하도록 선택할 수 있다. 시스템은 사용자의 이동 가능한 칸 수 범위 내에서 사용자의 입력이 타당한지 검증해야 한다.
RULE4	플레이어는 자신의 턴에 이동할 때, 현재 셀의 위치가 다리의 시작 지점이라면, 다리를 건널지 아닌지를 선택할 수 있다. 이때, 다리는 하나의 셀로 취급되어 칸 수 하나를 소모한다. 또한 다리를 건너면 사용자는 다리 카드 한장을 받게된다. 다리를 건너고 나서 이동 가능한 방향은 정방향으로 원래 맵의 정방향으로 제한 한다.
RULE5	한명의 플레이어라도 END에 도착하게 되면, 남은 플레이어들은 뒤로 가는것이 제한된다. 즉, (주사위 수 - 다리카드 수) 만큼 무조건 이동해야 한다.
RULE6	시스템은 사용자로부터 파일시스템 내에 존재하는 임의의 맵파일을 로드할 수 있어야 한다.
RULE7	시스템은 게임을 플레이할 사용자 수를 2-4명 중에서 선택할 수 있어야 한다.

- Operation Contact는 그 목적상 System Operation 의 결과가 명확하지 않을 때 PostCondition, Precondition을 정의하여 추가적인 정보를 제공할 때 작성한다. 본 프로젝트에서는 System Operation들(rollDie, ChoosePath 등)이 그 결과가 명확하다고 판단하여 추가적인 정보를 제공하지 않는 것이 낫다고 판단하였다.

설계 산출물

Package Diagram

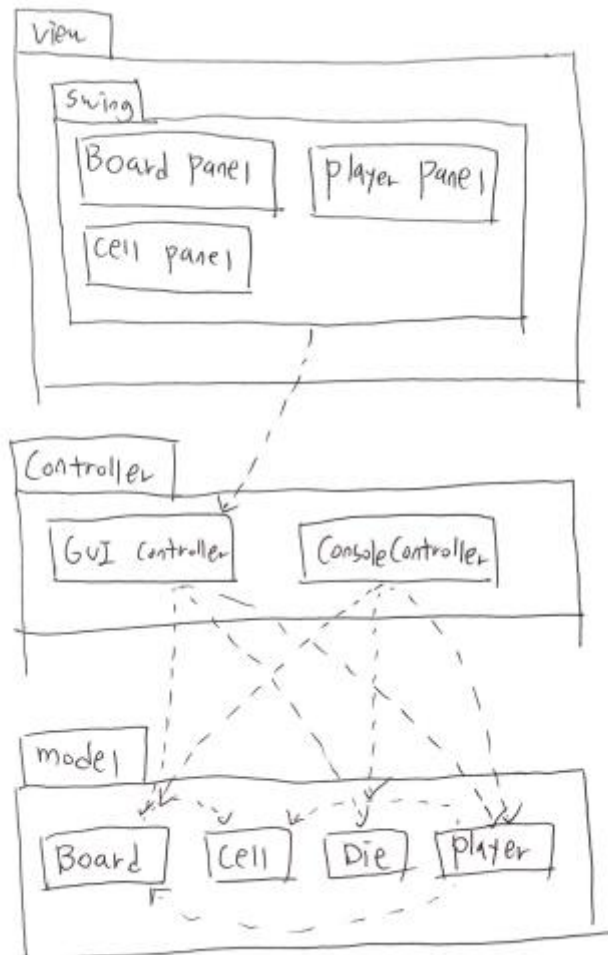


그림 4 - Package Diagram

우선 패키지 다이어그램을 통해 전체적인 아키텍처를 설계한다. MVC 패턴을 최대한 지키며 설계하였다. GUI, Console의 컨트롤러를 각자 구성하되, Model은 동일하게 재사용하도록 Model 부분에서는 Controller와 View에 대한 어떠한 참조도 존재하지 않도록 설계한다. 프로젝트를 진행하며 Console을 먼저 개발하고, 그 이후로 GUI 개발을 수행하였지만 Model의 어떠한 수정도 하지 않고 개발을 진행하였다. 추가적으로 요구사항 분석과 다른 점은, Package Diagram에서의 Player 클래스는 Domain Model에서의 Piece이다. 실제 게임 내에서는 Piece가 Player 그 자체 이므로 설계 단계에서 Model영역에 있는 Player는 Domain Model에서의 Piece를 의미한다.

Sequence Diagram

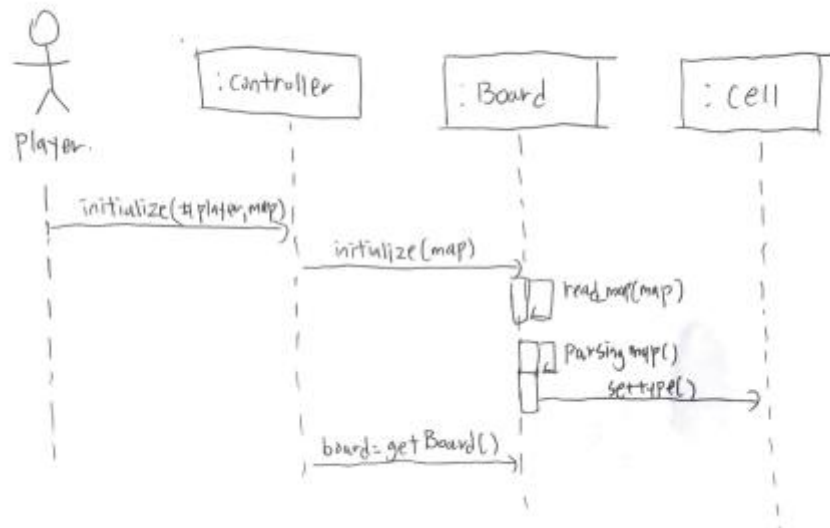


그림 5 - Sequence Diagram(initialization)

[그림 3]의 SSD에서 `initialize()`에 대한 시스템 내부 동작 로직이다. Controller는 사용자로부터 플레이어 수, 맵의 이름을 받는다. 컨트롤러는 이를 받아서 Board 클래스로 넘겨준다. Board 클래스는 해당 파일을 읽고(`readMap`), 읽어온 파일을 파싱하여(`ParsingMap`) Cell들을 만들어 낸다. 그리고 이에 대한 정보를 Controller가 참조하도록 한다.

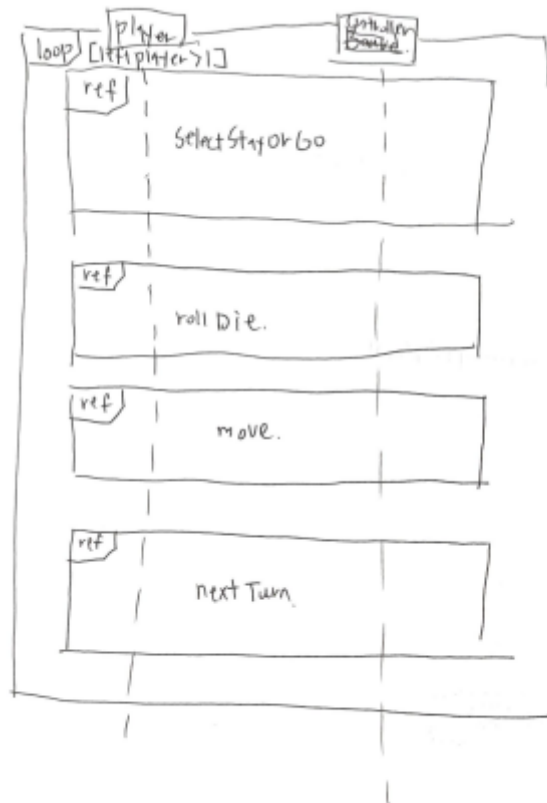


그림 6 - Sequence Diagram(Play Game)

PlayGame()함수에 대한 시스템 동작 로직이다. 크게 4가지 스텝으로 구분하였으며, 세부 내용은 추가적인 Sequence Diagram으로 표현하였다. 첫 번째 단계는 사용자로부터 해당 턴에 Stay할지, 이동을 할지를 결정하는 기능에 대한 설계이다. 두 번째 단계는 사용자가 진행하기로 하였을 때, 주사위를 던지는 기능이다. 이때, 설계 단계에서 주요한 변경사항으로 사용자로부터의 입력을 받는 방법을 변경하였다. 기존요구사항으로는 사용자로부터 RRUD와 같은 방식으로 사용자에게 맵 칸에 대한 입력을 요구하였지만, 설계과정에서 사용자의 입력을 검증하기 위해서 시스템에서 내부적으로 입력가능한 모든 목적지를 사전에 계산할 수 있다고 판단되었다. 따라서 사용자가 맵 칸에 대한 방향 입력이 아닌, 현재 칸으로부터 목적지 칸으로까지의 거리만 입력하도록 변경함으로써 사용자의 게임 편의성을 높였다. 세 번째 단계는 두 번째 단계에서 계산한 사용자가 이동 가능한 칸 수를 입력받아 실제 model이 이동하는 과정이다. 마지막은 현재 턴에서의 플레이어가 턴을 끝내는 로직이다. 그리고 해당 게임은 마지막에 남은 플레이어(leftPlayer)가 한명이 될 때까지 이 4가지 process를 반복하도록 한다.

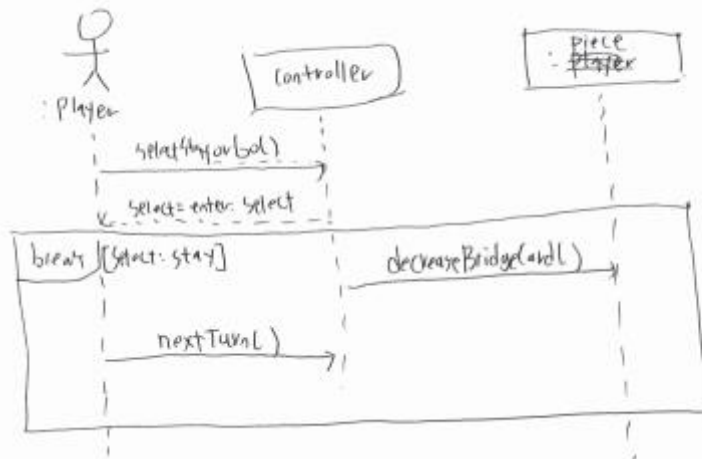


그림 7 - Sequence Diagram(Play Game : selectStayOrGo)

사용자는 Controller로 select 값을 입력한다. 사용자가 Stay를 입력하였다면, BridgeCard를 하나 줄이고, 턴을 넘겨준다. Go를 입력하면 추가적인 과정 없이 끝내서 다음 Roll Die가 실행된다.

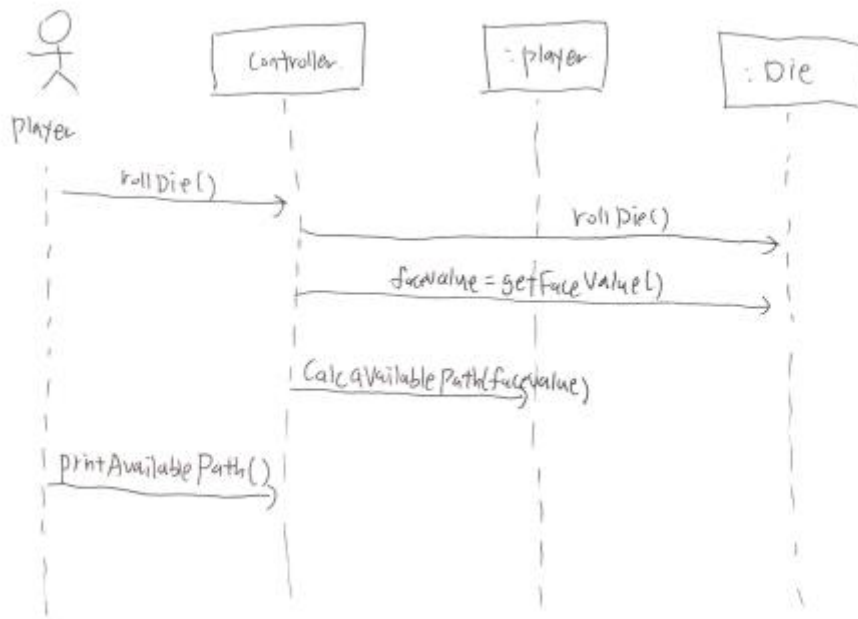


그림 8 - Sequence Diagram(Play Game : rollDie)

사용자는 Controller로 rollDie() 함수를 호출한다. Controller는 이에 대한 처리를 Die 객체의 RollDie()로 넘겨주고, 그 결과값을 받아온다. 그리고 설명했듯이 player 객체에서 calcAvailablePath() 함수를 통해서 해당 사용자가 이동 가능한 칸 수를 계산하고, PrintAvailablePath() 함수를 통해서 이를 출력하도록 설계하였다.

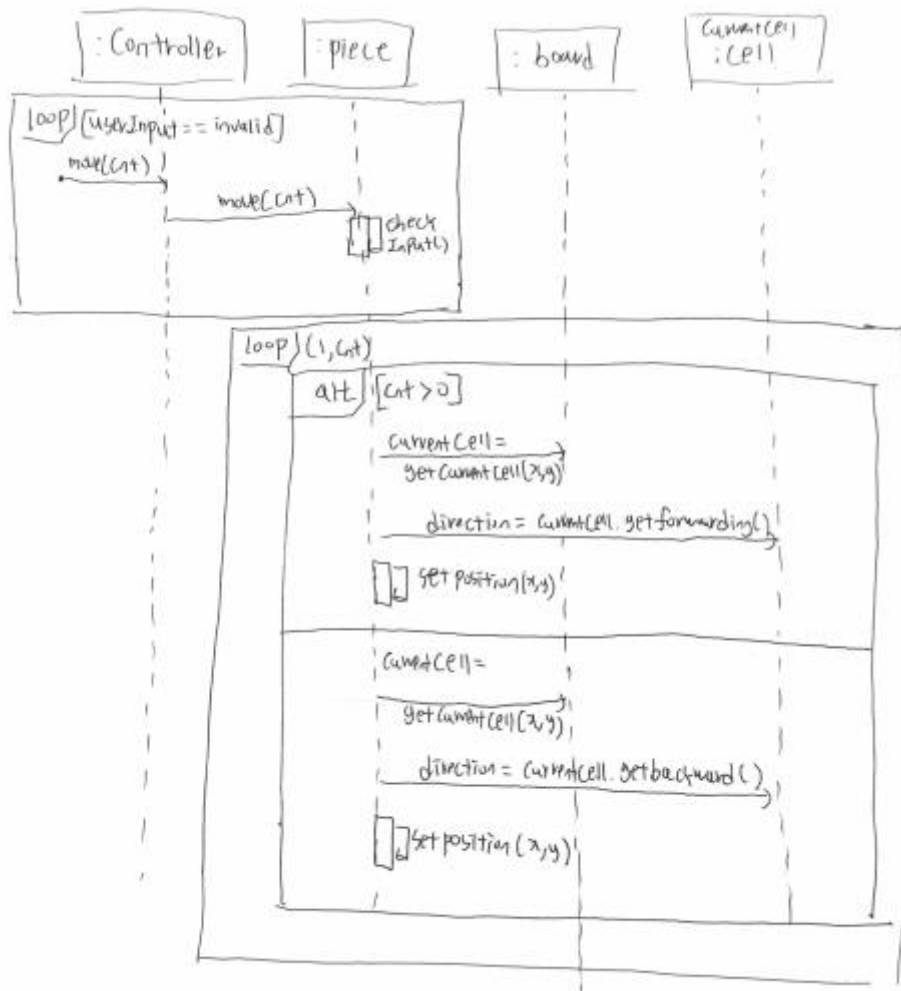


그림 9 - Sequence Diagram(Play Game : move)

이동 가능한 칸 수를 출력한 뒤에, 사용자로부터 입력을 받는다. 이때 사용자의 입력값이, 출력된 정보와 다르면 다시 입력을 받도록 요구한다. 사용자의 입력이 타당한 입력이면, piece 객체가 이동을 한다. 크게 앞으로 가는 경우와, 뒤로 가는 경우로 나누고 Piece 객체는 board 객체로부터 자신의 위치에 해당하는 셀을 불러온다. 셀에는 forward, backward 정보를 저장해놔서 사용자가 해당 정보를 가져오고, 방향에 맞도록 포지션을 변경한다. 이 과정을 이동하려는 칸의 개수만큼 반복하여 piece가 이동한다.

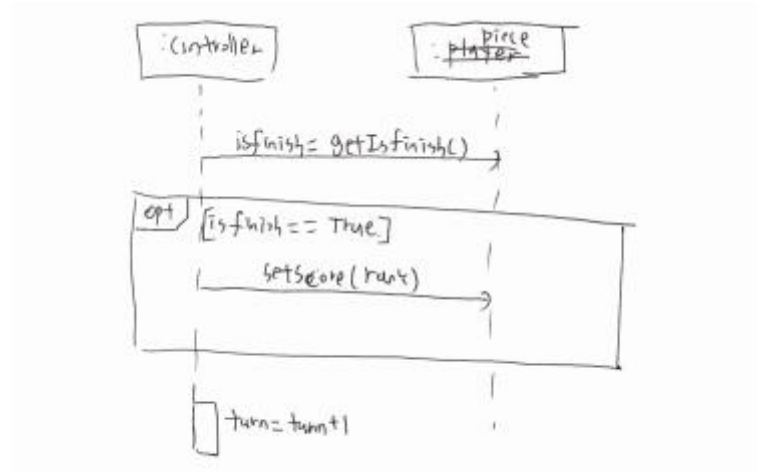
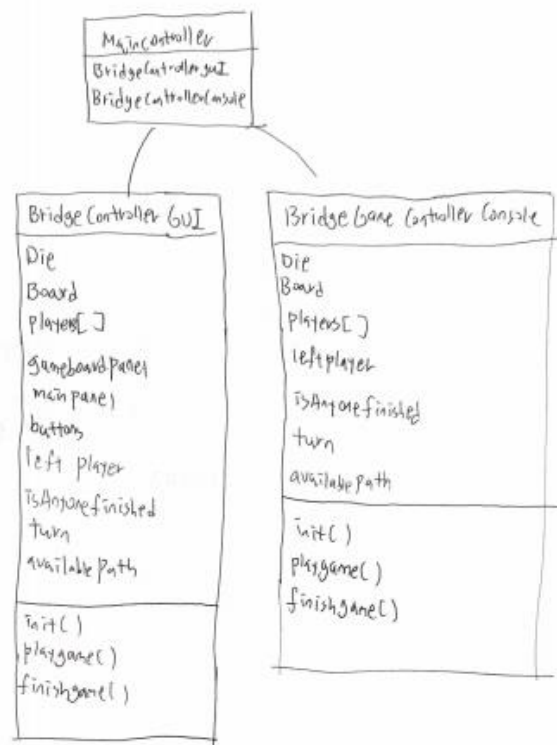


그림 10 - Sequence Diagram(Play Game: nextTurn)

마지막으로 턴을 끝내는 로직이다. 매 턴을 끝내기 전에 플레이어가 목적지에 도착하였는지 확인한 뒤에, 만약 목적지에 도착했다면 끝내는 프로세스를 추가해주고 턴을 넘겨준다.

Class Diagram

Controll의 클래스 다이어그램은 다음과 같다.



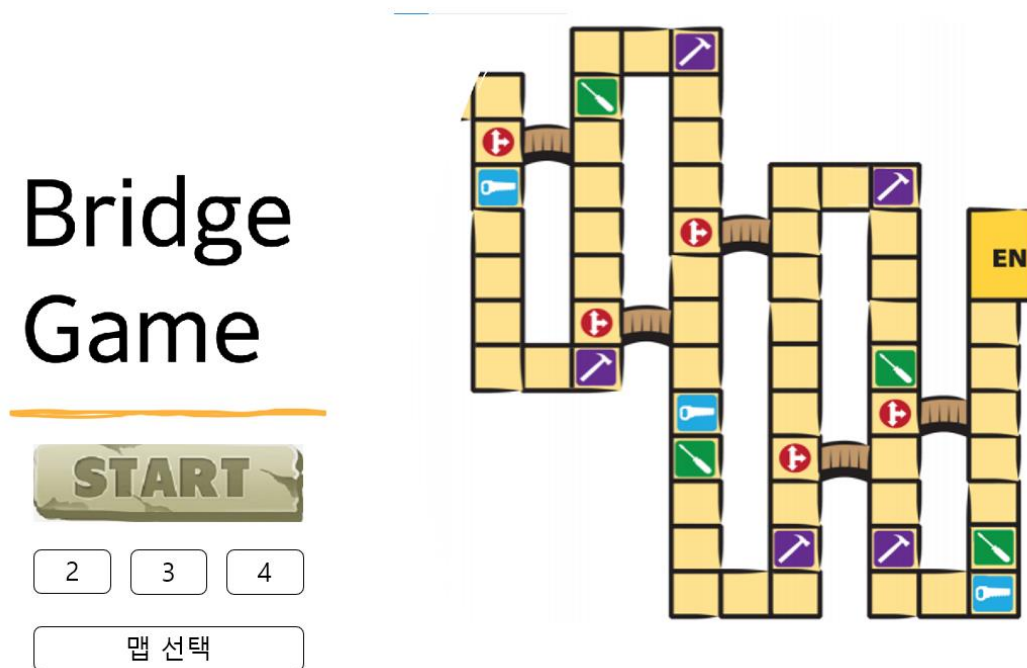
MainController에서 GUI, Console 컨트롤러를 가지며 사용자 Input에 따라 알맞은 컨트롤러를 호출한다. 그리고 GUI, Console 컨트롤러는 기본적으로 가지고 있는 정보가 동일하다. 유일한 차이점은 Gui 컨트롤러의 경우 View 객체들을 가지기 Panel, Button같은 변수들이 포함되지만, Console은 Model 변수들만 가진다.

다음은 Model의 클래스 다이어그램이다.



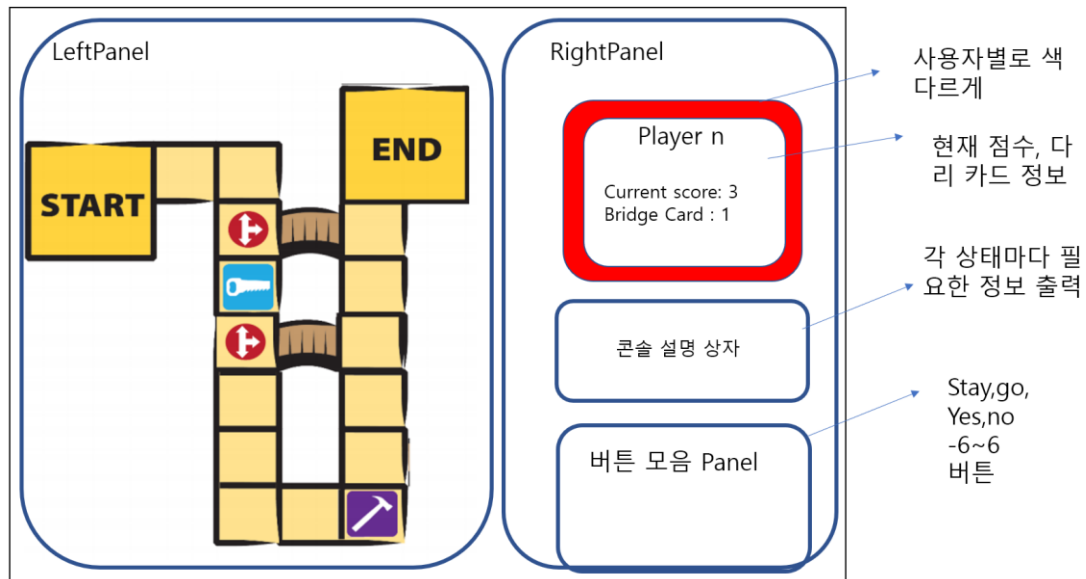
Model 패키지에 포함되는 최종 클래스들은 총 4개로 Player, Die, Board, Cell이다. 각각의 연관 관계를 보면 우선 가장 중요한 내용은 Controller는 Model들을 참조하지만, Model에는 Controller와 View에서의 어떠한 연결 점도 없다. 그리고 Model내에서의 참조 관계로는 꼭 필요한 참조 관계를 고려하여, Player는 Board, Cell을 참조하고 Board는 Cell을 참조한다.

UI Prototype



시작화면 UI Prototype이다. 사용자 수를 선택하고, 맵을 파일시스템에서 선택한 뒤에 Start 버튼을 누르면 게임 화면이 나오도록 한다.

JFrame

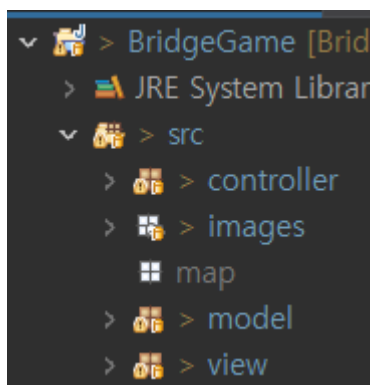


게임 실행시 prototype이다. 전체 프레임을 좌 우로, 나눈다음 왼쪽에는 보드와 플레이어가 출력 되도록 하고, 오른쪽에 현재 턴의 사용자 정보, 콘솔 설명 상자, 그리고 눌러야 하는 버튼들을 넣어준다.

구현 산출물

소스코드 설명

전체 프로젝트 패키지 구조이다. 패키지 다이어그램에서 설계한 대로 패키지를 구성하였으며 MVC 패턴을 고려하여 구현하였다.



1 Controller 패키지

1.1 MainController

Controller 패키지는 MainController 파일이 메인인 컨트롤러이며, GUI, Console 각각의 컨트롤러를 만들어 주었다. MainController에서 입력을 받아 Gui모드로 실행을 할지, 아니면 Console 모드로 실행을 할지를 결정할 수 있다.

1.2 BridgeGameControllerConsole

Console모드의 컨트롤러이다. 컨트롤러의 역할을 생각하며 그 기능을 세분화 했으며 주요 중점사항은 다음과 같다. 첫 째로 사용자의 Input / Output은 무조건 Controller에 배치하였다. 따라서 Model에는 어떠한 출력문도 존재하지 않는다. 두 번째로 Controller는 어플리케이션의 로직을 가지지 않는다. Controller의 책임성을 고려하여 사용자의 Input을 적절한 Model로 연결하도록 해줬고, Model의 결과를 사용자에게 출력해줄도록 구현하였다. 해당 컨트롤러의 생성자는 다음과 같다.

```
public BridgeGameControllerConsole() {  
    init();  
    playGame();  
    finishGame();  
}
```

useCase 및 설계에서 만들었던 대로 게임을 시작(init)하고 게임을 진행(playGame)하고, 마지막에 게임을 끝내는(finishGame)과정으로 수행된다. 가장 핵심이 되는 playGame()의 함수 프로토타입을 보면 다음과 같다.


```

public void playGame() {
    while(leftPlayer > 1) {
        currentPlayer = players[turn];
        if(isPlayerFinished(currentPlayer)) { continue; }

        // 현재 보드 상태 출력
        printBoard();

        // 현재 플레이어 정보 출력
        printCurrentPlayerInfo(currentPlayer);

        // 1. 해당 턴에 주사위를 굴릴지, 멈추고 다리카드를 지울지 선택
        if(selectStayOrGo(currentPlayer)) { continue; }

        // 2. 현재 칸이 다리 시작지점이면 다리를 건널지 여부를 판단.
        if(currentPlayer.checkBridge()) { determineBridge(currentPlayer); }

        // 3. 주사위 굴리기
        faceValue = rollDie();

        // 4. 현재 플레이어가 갈 수 있는 경로 계산 및 출력
        availablePath = currentPlayer.calcAvailablePath(faceValue);
        printAvailablePath(availablePath, currentPlayer);
        if(currentPlayer.getAvailableCount() == 0) {
            turn = (turn + 1) % numPlayer;
            continue;
        }

        // 5. 사용자가 선택한 숫자 만큼 이동
        movePlayer(currentPlayer);

        // 6. 현재 플레이어가 끝났는지 확인 후, 1-5번 반복
        if(currentPlayer.getIsFinish()) {
            finishPlayer(currentPlayer);
        }

        turn = (turn + 1) % numPlayer;
    }
    /*마지막 한명이 남으면 플레이어들의 등수/점수를 화면에 출력하면서 종료*/
}

```

Sequence Diagram을 통해 얻은 프로그램 로직에 따라서 Controller의 로직을 구현하였다.

1.3 BridgeGameControllerGUI

BridgeGameController도 정확히 동일하게 Console방식으로 구현하였다. 다만 GUI 특성상 이벤트를 통해서 코드가 진행되기 때문에 Console컨트롤러 처럼 순차적으로 코드가 구성 되어 있지는 않다. 하지만 ConsoleController에서 주로 고려한 내용은 컨트롤러에는 어플리케이션의 로직이 들어있지 않도록 하는 내용과, 모든 사용자와의 상호작용은 컨트롤러

에서 수행되도록 구현하였다.

2 View 패키지

2.1 BoardPanel

```
public class BoardPanel extends JPanel {  
    private Board board;  
    private CellPanel[][] cellPanelArray;  
  
    public BoardPanel(int xCoord, int yCoord, int width, int height, Board board) {}  
  
    private void initializeCells() {}  
  
    public void paintComponent(Graphics g) {}  
}
```

View에서는 해당하는 Model의 정보를 출력하는 기능만 가지고 있다. 그 이외의 사용자의 Input 처리 및 어플리케이션의 로직은 존재하지 않는다.

2.2 CellPanel

```
public class CellPanel extends JPanel {  
  
    Cell cell;  
    private int x;  
    private int y;  
  
    public final static Image START_IMG = new ImageIcon("src/images/start.png").getImage();  
    public final static Image BRIDGE_IMG = new ImageIcon("src/images/bridge.png").getImage();  
    public final static Image CELL_IMG = new ImageIcon("src/images/cell.png").getImage();  
    public final static Image DRIVER_IMG = new ImageIcon("src/images/driver.png").getImage();  
    public final static Image END_IMG = new ImageIcon("src/images/end.png").getImage();  
    public final static Image HAMMER_IMG = new ImageIcon("src/images/hammer.png").getImage();  
    public final static Image INTERSECTION_IMG = new ImageIcon("src/images/intersection.png").getImage();  
    public final static Image SAW_IMG = new ImageIcon("src/images/saw.png").getImage();  
  
    public CellPanel(int x, int y, int width, int height) {}  
  
    public Cell getCell() {}  
  
    public void setCell(Cell cell) {}  
  
    public void paintComponent(Graphics g) {}  
}
```

Cell Panel도 마찬가지이다. 해당하는 Cell 변수를 가지고 paintComponent에 의해서 자신의 Cell Type에 따라 다른 이미지를 화면에 출력한다

2.3 PlayerPanel

```

public class PlayerPanel extends JPanel {
    public static final int PLAYER_IMG_WIDTH = 20;
    public static final int PLAYER_IMG_HEIGHT = 33;

    public final static Image PLAYER1_IMG = new ImageIcon("src/images/player1.png").getImage();
    public final static Image PLAYER2_IMG = new ImageIcon("src/images/player2.png").getImage();
    public final static Image PLAYER3_IMG = new ImageIcon("src/images/player3.png").getImage();
    public final static Image PLAYER4_IMG = new ImageIcon("src/images/player4.png").getImage();

    private Player player;

    public PlayerPanel(Player player) {}

    public Player getPlayer() {}

    public void setPlayer(Player player) {}

    public void paintComponent(Graphics g) {}
}

```

3 Model 패키지

3.1 Board

```

public class Board {
    public static final int MAX_MAP_SIZE = 16;

    private char[][] bufferMap;
    private int cellCount;
    private Cell[][] cellArray;
    private Cell startCell;

    public Board(String s) {}

    public void loadMap(String path) {}

    public void parsingMap() {}

    public int[] calcMapBoundary() {}

    public Cell[][] getCellArray() {}

    public Cell getStartCell() {}

    public Cell getCell(int x, int y) {}
}

```

Class Diagram에서 명시한 함수들을 그대로 구현하였으며, 일부 서브 프로세스들이 추가되었다. CalcMapBoundary 함수는 맵을 파싱할 때, 첫 시작 지점을 보정하기 위한 함수이다. 예를 들어 default map 같은 경우는 시작 지점이 0,0이 아닌 1,0에서 시작해야 전체 맵이 array index안에 들어올 수 있다.

3.2 Cell

Cell은 기본적으로 특별히 하는 기능이 존재 하지 않기 때문에 자신의 Cell 위치를 반환하거나 Type 정보를 반환하는 Getter, Setter함수만 존재한다.

```
public Cell(int x,int y,int type) {  
  
    public void setBackwardDirection(int backwardDirection) {  
    public void setForwardDirection(int forwardDirection) {  
    public void setType(int type) {  
  
    public int getBackwardDirection() {  
    public int getForwardDirection() {  
    public int getType() {  
    public int getX() {  
    public int getY() {  
        return y;  
    }  
}
```

3.3 Player

Player 클래스가 가장 많은 기능을 가지고 있다. 게임에서 가장 중요한 로직이 플레이어의 이동이기 때문에 move()함수를 구현하는 과정에서 몇 개의 서브 함수를 구현하였다. Getter/setter를 제외한 주요 함수들의 프로토타입은 다음과 같다.

```
public Player(int x, int y, int playerId) {  
  
    /* 주사위 값을 인자로 받아, 현재 다리 카드 갯수를 고려하여 해당 사용자가  
    public int[] calcAvailablePath(int faceValue) {  
  
    /* 인자로 주어진 수 만큼 앞으로 이동 */  
    public void moveForward(int moveCnt) {  
|  
    /* 인자로 주어진 수 만큼 뒤로 이동 */  
    public void moveBackward(int moveCnt) {  
  
    /*다리를 건너는 경우 호출하는 함수*/  
    public void crossBridge(int moveCnt) {  
  
    /*사용자의 입력값에 해당하는 만큼 이동하는 함수*/  
    public boolean move(int userInput) {  
    /*사용자의 입력값이 정상적인 값인지 판단하는 함수*/  
    public boolean checkUserInput(int userInput) {  
    /*현재 플레이어의 위치가 다리 시작점인지 판단하는 함수 */  
    public boolean checkBridge() {
```

Move 함수 내에서 moveForward, moveBackward, crossBridge가 state에 따라서 호출이 된다.

그리고 calcAvaiablePath()함수는 사용자가 진행 가능한 경로에 대해서 계산을 하고 그 값을 변수에 저장해 놓는다.

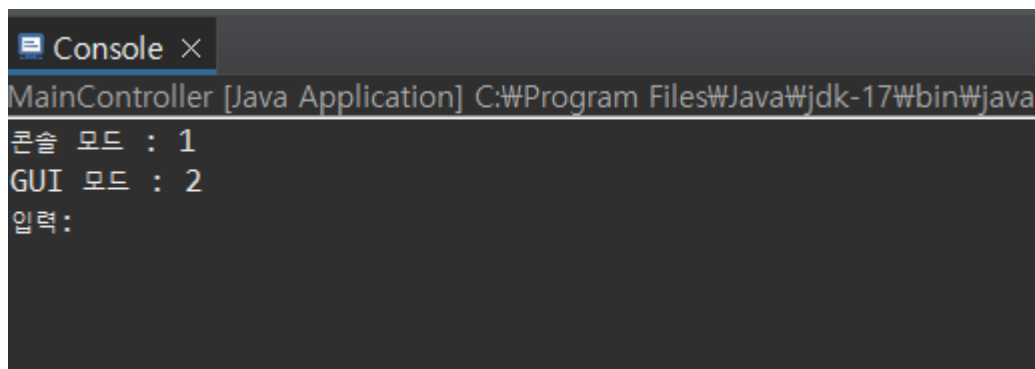
3.4 Die

Die 클래스는 주사위를 굴리고, 해당하는 주사위 값을 가져오는 함수로 이루어져 있다.

```
public class Die {  
    private int faceValue;  
  
    public int rollDie() {  
        faceValue = (int)(Math.random()*6)+1;  
        return faceValue;  
    }  
  
    public int getFaceValue() {  
        return faceValue;  
    }  
}
```

프로그램 사용법

프로그램을 시작하면 MainController에서 다음과 같이 2가지 모드를 제공한다.



The screenshot shows a console window titled "Console" with a close button. The text inside the window reads: "MainController [Java Application] C:\Program Files\Java\jdk-17\bin\java", "콘솔 모드 : 1", "GUI 모드 : 2", and "입력:". The text is displayed in a monospaced font on a dark background.

1. 콘솔 모드

```
콘솔 모드 : 1
GUI 모드 : 2
입력:
1
게임을 플레이 할 사용자 수를 입력해 주세요.
>>> 3
플레이 할 맵의 이름을 입력해 주세요.
[default.map another.map]
>>> default.map
```

콘솔모드를 실행하면 사용자 수와, 맵의 이름을 선택하여 프로그램을 실행 가능하다.

```
===== printBoard() =====
3S  C  C  P  C
    B  =  b  C
    S  C  C  C  C  H
    C  C  B  =  b  C
    C  C  C  C  C  E
    C  B  =  b  C  C
    C  C  H  C  C  P  C
          S  C  B  =  b
          P  B  =  b  C
          C  C  C  C
          C  H  H  P
          C  C  C  C  C  S

=====
===== PlayerInfo() =====
현재 플레이어 : player1
다리 카드 갯수 : 0
현재 점수 : 0
===== 선택하기 =====
1. 이동하기
2. 이번턴을 끝내고 다리 카드 한장 제거
>>>
```

매 턴에 위의 그림과 같이 board 정보와 현재 플레이어 정보(다리 카드,점수) 그리고 현재 어떠한 입력을 해야 하는지에 대한 콘솔정보를 출력한다. 사용자의 가독성을 위해서 특정 위치들에 색을 입혔으며, 사용자의 입력 콘솔에 '>>>' 표시를 하였다. 콘솔의 특성상 한 칸에 여러 플레이어가 있다면 가장 높은 순서의 플레이어가 그 아래 플레이어를 덮어쓴다. 다음은 이동하기를 선택한 뒤에 이동을 하는 과정이다.

```

===== PlayerInfo() =====
현재 플레이어 : player1
다리 카드 갯수 : 0
현재 점수 : 0
===== 선택하기 =====
1. 이동하기
2. 이번턴을 끝내고 다리 카드 한장 제거
>>> 1
주사위 결과 : 2
다리 카드 갯수 : 0
현재 플레이어가 가진 다리카드 갯수를 제외하여 총 2칸 이동 가능합니다.
이동 가능한 칸 수는 다음과 같습니다. 이중 하나를 입력하세요.
    2  0
>>> 2

```

Player 1이 주사위를 굴려서 2가 나왔고, 맨 처음 시작시에는 뒤로 가는 것이 불가능 하기 때문에 오른쪽으로 2칸을 이동하거나, (앞,뒤) 이동을 하여 0칸을 이동하는 경우가 있다. 2를 입력하여 다음과 같이 이동 가능하다.

```

===== printBoard() =====
3S  C 1C      C  C  H
      B  =  b  C
      S  =  C  C  C  C  H
      C  C  B  =  b  C  C  E
      C  C  B  =  b  C  C  C
      C  C  H  C  C  P  C
          S  C  B  =  b
          P  B  =  b  C
          C  C  C  C
          C  H  H  P
          C  C  C  C  C  S

=====
===== PlayerInfo() =====
현재 플레이어 : player2
다리 카드 갯수 : 0
현재 점수 : 0
===== 선택하기 =====
1. 이동하기
2. 이번턴을 끝내고 다리 카드 한장 제거
>>>

```

플레이어 1이 2칸 이동하여 화면에 표시된 것을 볼 수 있다. 이동 후에 자동으로 턴은 다음 플레이어에게 넘어간다.

```

===== printBoard() =====
      C C H
S C 1C P C
  2B = b C
  S C C C C H
3C C B = b C
C C C C C E
C C B = b C C C
C C H C C P C
      S C B = b
      P B = b C
      C C C C
      C H H P
      C C C C C S

=====
===== PlayerInfo() =====
현재 플레이어 : player2
다리 카드 갯수 : 0
현재 점수 : 0
===== 선택하기 =====
1. 이동하기
2. 이번턴을 끝내고 다리 카드 한장 제거
>>> 1
현재 위치에서 다리를 건널 수 있습니다. 다리를 건너시겠습니까? [Y/N]

```

Player2가 다리 시작 위치에 있는 경우에 대한 처리과정이다. 이 경우에는 사용자에게 다리를 건널지 아니면 그냥 이동할지에 대한 선택을 하도록 한다. Y/N로 선택할 수 있으며 Y를 선택한 경우에 다음과 같이 다리를 이동할 수 있다.

```

현재 위치에서 다리를 건널 수 있습니다. 다리를 건너시겠습니까? [Y/N]
Y
주사위 결과 : 5
다리 카드 갯수 : 0
현재 플레이어가 가진 다리카드 갯수를 제외하여 총 5칸 이동 가능합니다.
이동 가능한 칸 수는 다음과 같습니다. 이중 하나를 입력하세요.
  5 3 1
>>> 3
===== printBoard() =====
      C C H
S C 1C 2P C
  B = b C
  S C C C C H
3C C B = b C
C C C C C E
C C B = b C C C
C C H C C P C
      S C B = b
      P B = b C
      C C C C
      C H H P
      C C C C C S

=====

```

이동한 뒤에 player2의 정보를 보면 다음과 같다.


```

===== printBoard() =====
      C C H
S C 1C 2P C
      B = b C
      S C C C C H
      3C C B = b C
      C C C C C E
      C B = b C C
      C C H C C P C
      S C B = b
      P B = b C
      C C C C
      C H H P
      C C C C C S

=====
===== PlayerInfo() =====
현재 플레이어 : player2
다리 카드 갯수 : 1
현재 점수 : 1
===== 선택하기 =====
1. 이동하기
2. 이벤트를 끝내고 다리 카드 한장 제거
>>>

```

다리를 건너왔기 때문에 다리 카드 개수가 1개 추가되었으며, P에 도착하였기 때문에 해당 하는 점수 1점을 얻은 것을 볼 수 있다. 그 상태에서 주사위를 굴리면 다음과 같이 다리카드 개수를 제외한 만큼 이동가능하다.

```

===== PlayerInfo() =====
현재 플레이어 : player2
다리 카드 갯수 : 1
현재 점수 : 1
===== 선택하기 =====
1. 이동하기
2. 이벤트를 끝내고 다리 카드 한장 제거
>>> 1
주사위 결과 : 2
다리 카드 갯수 : 1
현재 플레이어가 가진 다리카드 갯수를 제외하여 총 1칸 이동 가능합니다.
이동 가능한 칸 수는 다음과 같습니다. 이중 하나를 입력하세요.
1 -1
>>>

```

만약 사용자가 1, -1 이외의 정보를 입력하면 다음과 같이 다시 입력을 받도록 한다.

```

>>> 1
주사위 결과 : 2
다리 카드 갯수 : 1
현재 플레이어가 가진 다리카드 갯수를 제외하여 총 1칸 이동 가능합니다.
이동 가능한 칸 수는 다음과 같습니다. 이중 하나를 입력하세요.
  1 -1
>>> 2
유효하지 않은 경로를 입력했습니다. 다시 입력하세요.
>>> 0
유효하지 않은 경로를 입력했습니다. 다시 입력하세요.
>>> 5
유효하지 않은 경로를 입력했습니다. 다시 입력하세요.
>>>

```

다음은 플레이어가 목적지에 도착했을 때의 처리이다.

```

현재 플레이어가 가진 다리카드 갯수를 제외하여 총 5칸 이동 가능합니다.
이동 가능한 칸 수는 다음과 같습니다. 이중 하나를 입력하세요.
  5 3 1
>>> 3
player1가 1등으로 도착했습니다.
player1의 총 획득 점수 : 7
===== printBoard() =====
  S  C  C    1E
      B  =  b
      S    C
      B  =  b
      3C   C
      2C   C
      C  C  H

```

위의 그림과 같이 플레이어1이 목적지에 도착한 경우 총 획득 점수를 출력한다. 그리고 한 명 이상의 플레이어가 도착했기 때문에 그 다음 턴 부터는 남은 플레이어는 뒤로 이동하는 것이 제한된다. 아래 그림은 Player2가 주사위를 던졌을 때 나오는 메시지이다. '한명 이상의 플레이어가 목적지에 도착했습니다. 뒤로 가기가 제한 됩니다.'라는 메시지와 함께 이동 가능한 칸 수가 제한된 것을 볼 수 있다.

```

===== PlayerInfo() =====
현재 플레이어 : player2
다리 카드 갯수 : 0
현재 점수 : 0
===== 선택하기 =====
1. 이동하기
2. 이벤트를 끝내고 다리 카드 한장 제거
>>>
1
주사위 결과 : 1
다리 카드 갯수 : 0
현재 플레이어가 가진 다리카드 갯수를 제외하여 총 1칸 이동 가능합니다.
한명 이상의 플레이어가 목적지에 도착했습니다. 뒤로 가기가 제한 됩니다.
이동 가능한 칸 수는 다음과 같습니다. 이를 입력하세요.
1
>>> 1

```

마지막으로 남은 보드에 남은 플레이어가 한 명이 되면 다음과 같이 총 획득 점수를 출력하고 게임을 끝낸다.

```

===== 선택하기 =====
현재 플레이어가 가진 다리카드 갯수를 제외하여 총 5칸 이동 가능합니다.
한명 이상의 플레이어가 목적지에 도착했습니다. 뒤로 가기가 제한 됩니다.
이동 가능한 칸 수는 다음과 같습니다. 이를 입력하세요.
5
>>> 5
player3가 2등으로 도착했습니다.
player3의 총 획득 점수 : 3
===== 게임 종료 =====
===== 플레이어 점수 =====
player1 : 7
player2 : 2
player3 : 3

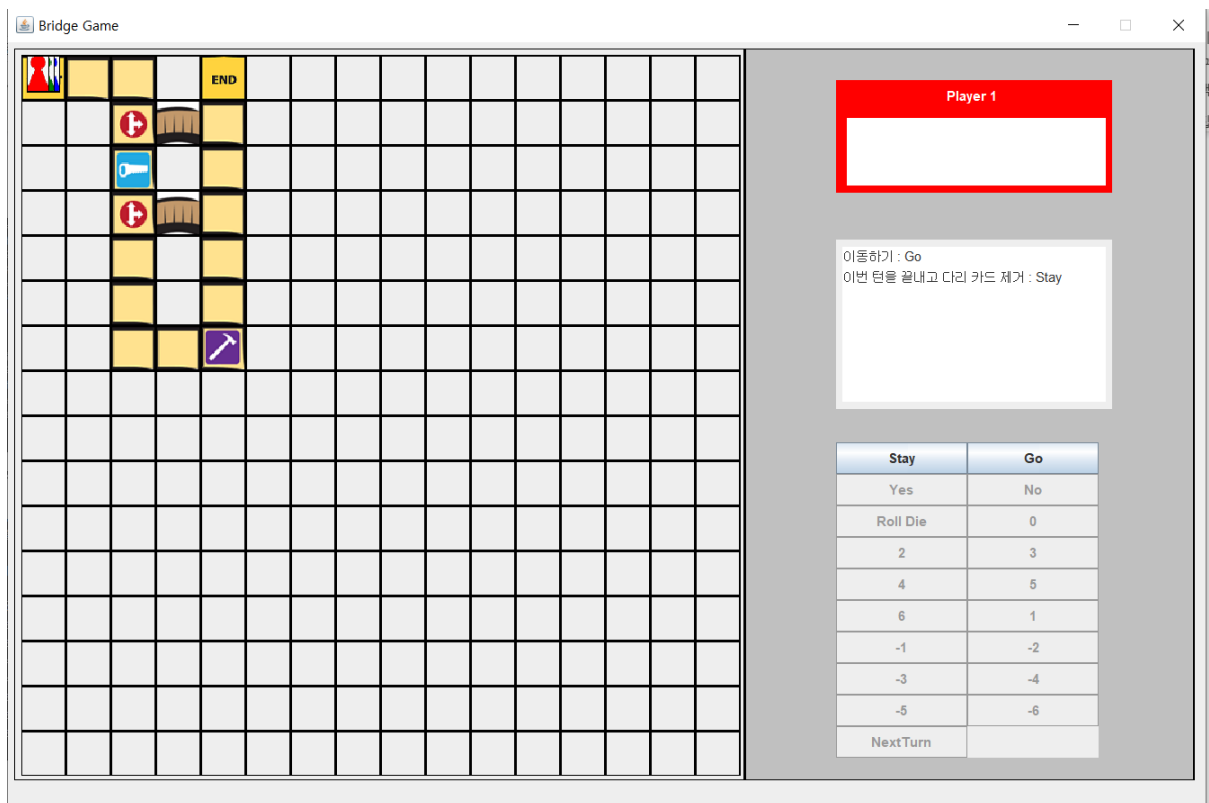
```

2. GUI 모드

GUI모드의 경우 일부 요구사항을 구현하지 못했다. 우선 UI Prototype처럼 시작 화면과 게임 화면을 분리하는 작업을 하지 못했다. 처음 맵의 선택과 플레이어 수는 다음과 같이 콘솔을 통해 입력을 받는다.

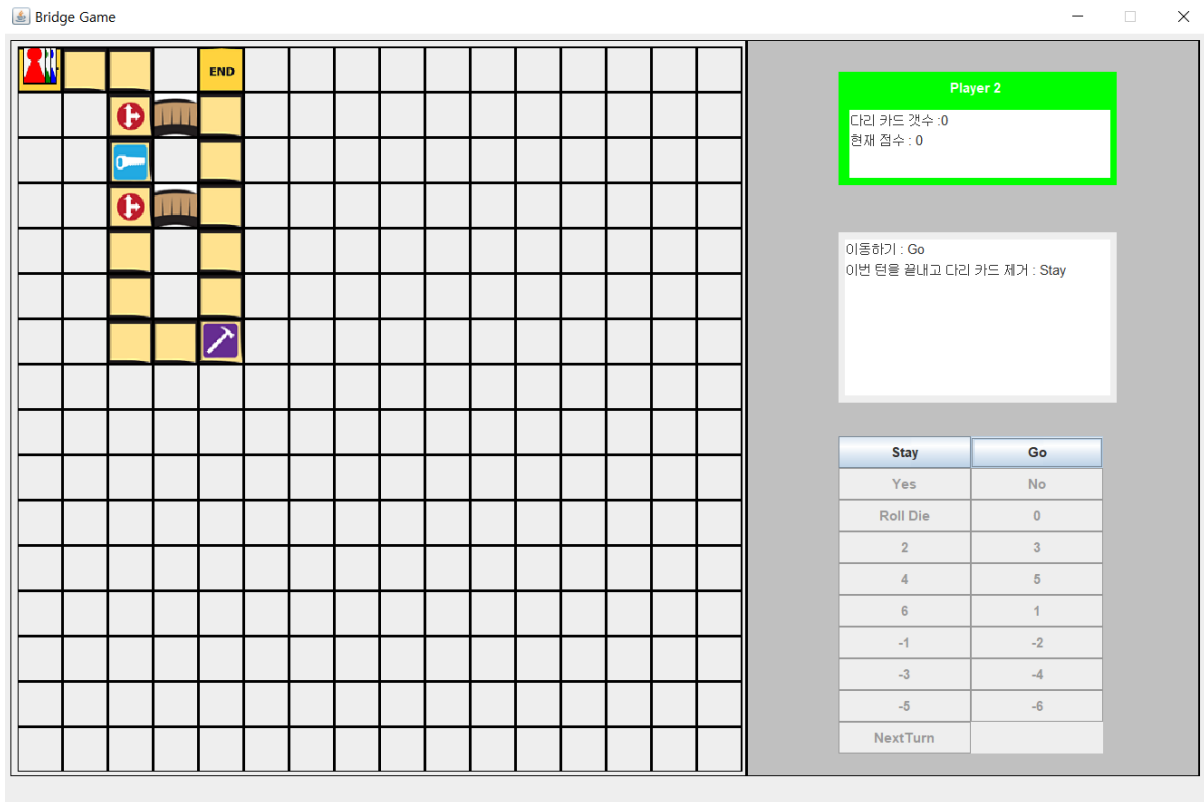
```
Console X
MainController [Java Application] C:\Program
콘솔 모드 : 1
GUI 모드 : 2
입력:
2
맵 이름을 입력하세요.
default.map another.map
another.map
사용자 수를 입력하세요
3
```

게임 시작시의 UI는 Prototype의 내용대로 구현하였다.

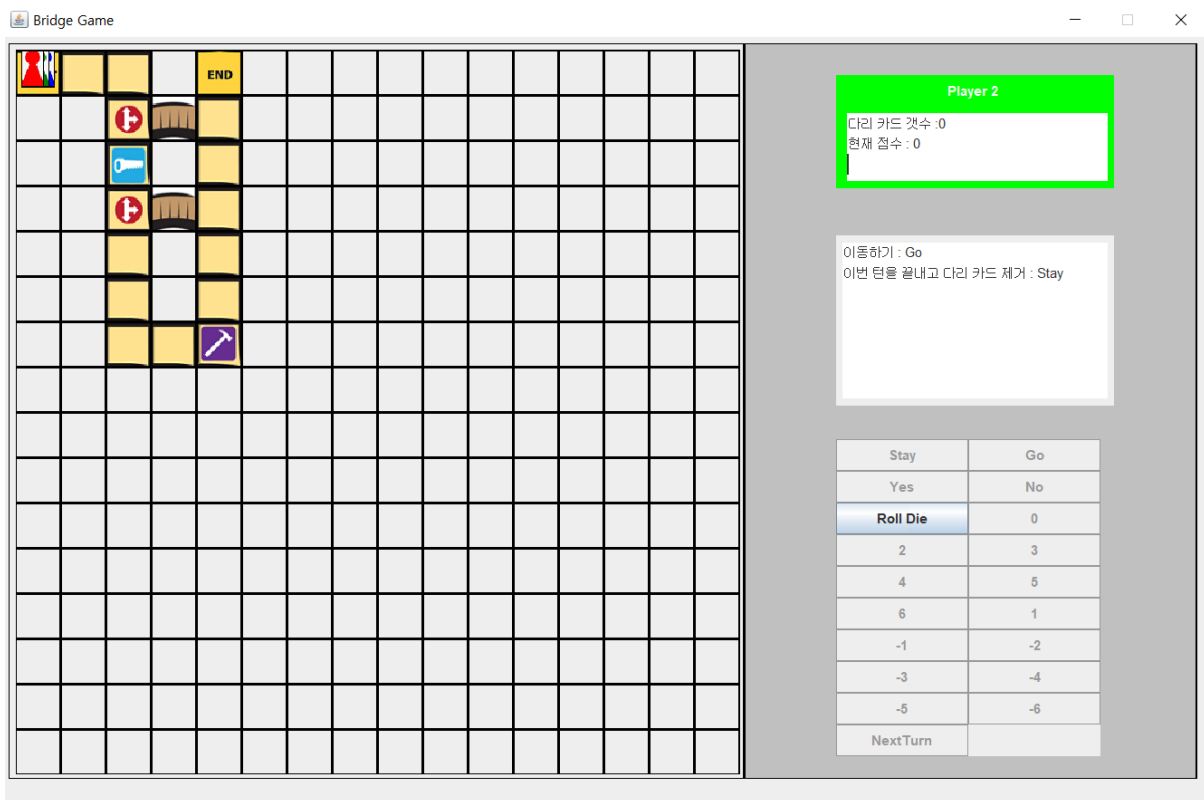


왼쪽 패널에 보드와 플레이어가 그려지고, 오른쪽에는 위에서부터 플레이어의 정보, 콘솔정보, 버튼들을 넣어주었다. Console과 동일한 Model을 사용했기 때문에 기능적인 요구사항은 모두 충족시켰다.

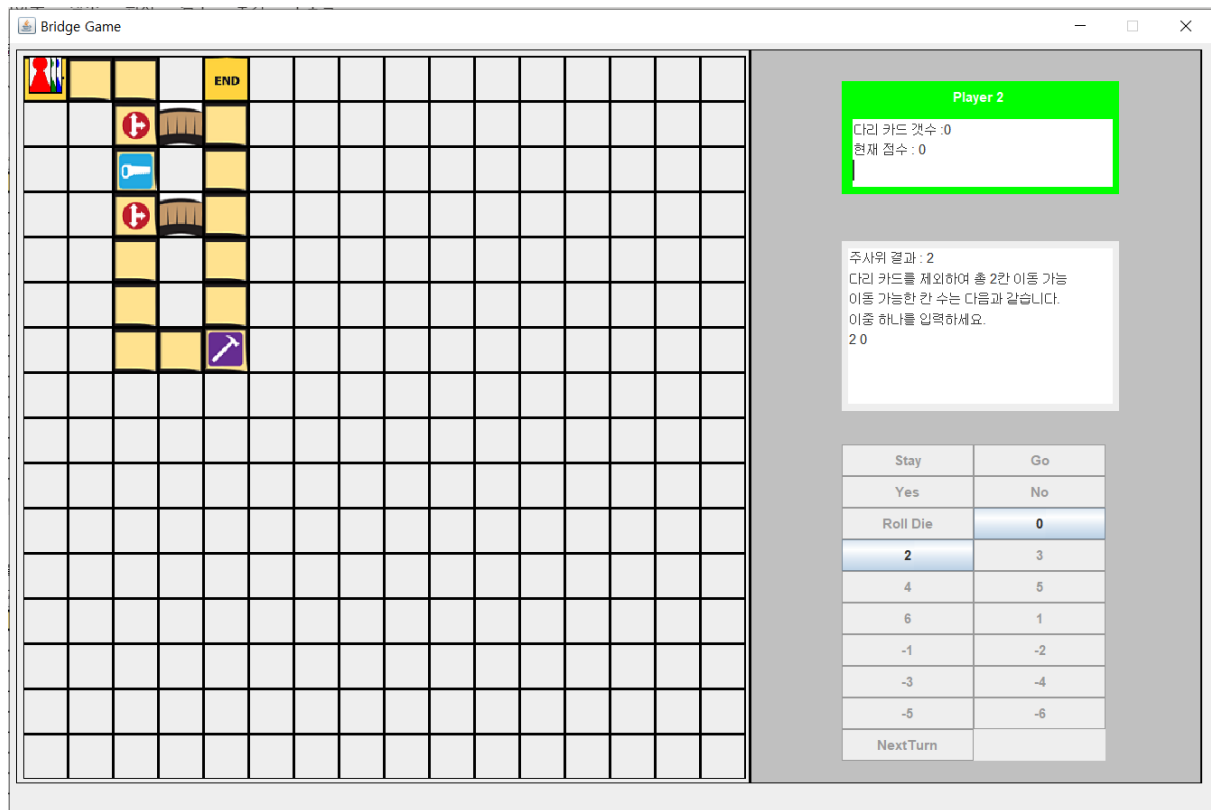
다음과 같이 각각의 플레이어 턴마다 누구의 턴인지 인식할 수 있도록 색을 다르게 해주었다.



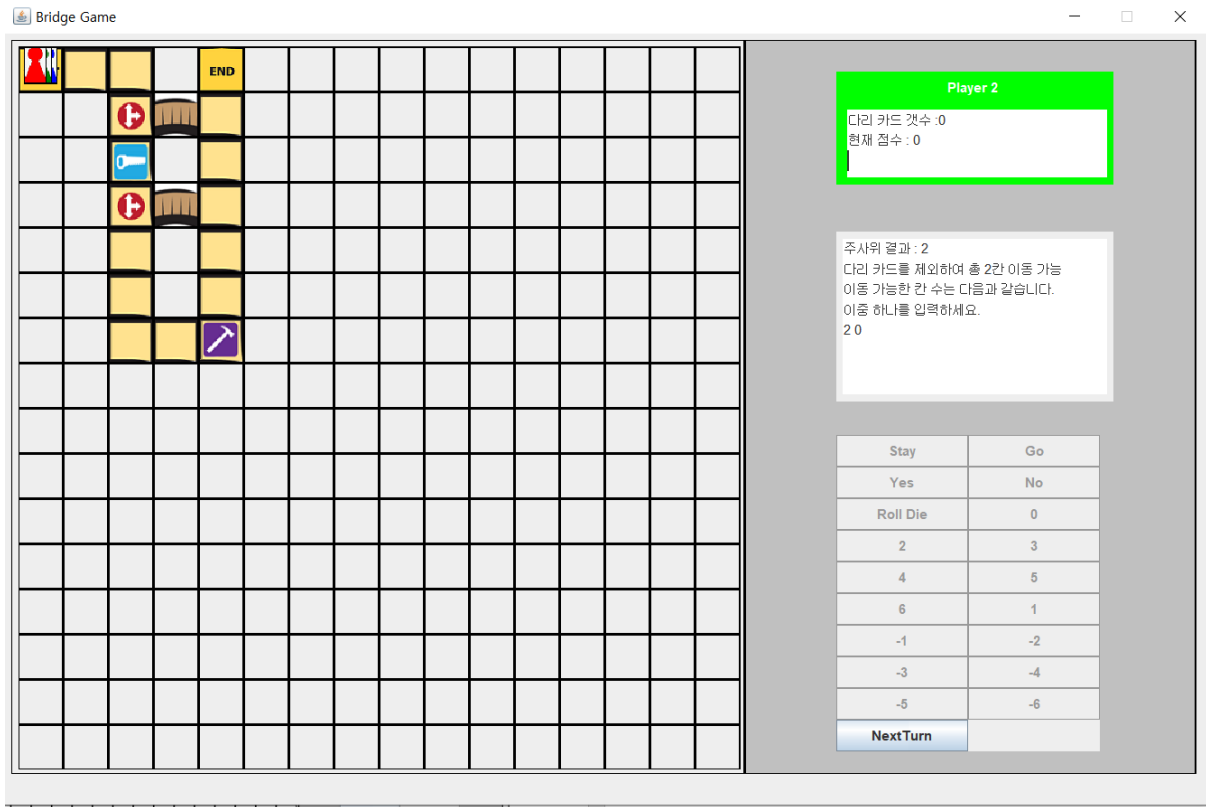
Go를 클릭하면 다음과 같이 화면이 전환된다.



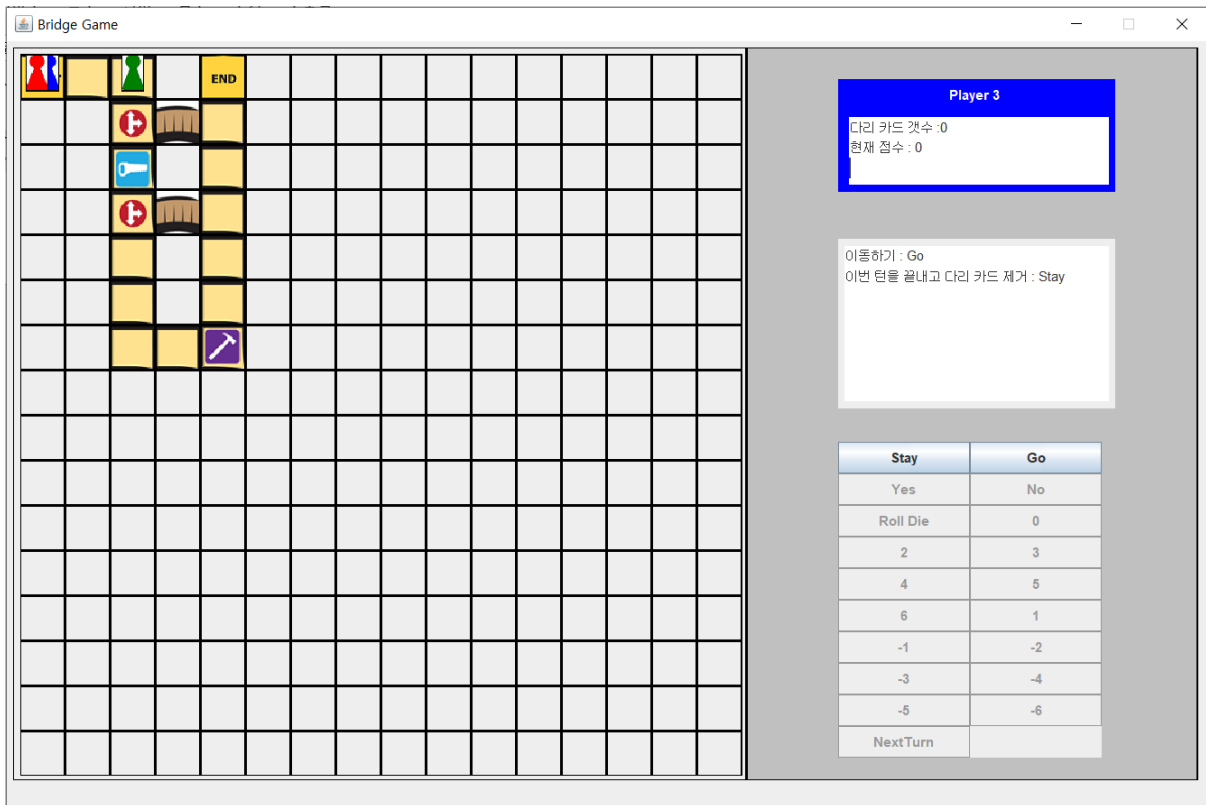
Roll Die 버튼을 클릭하면 Model에서 주사위를 굴리고, 이동 가능한 경로를 계산하여 사용자의 버튼이 활성화 된다.



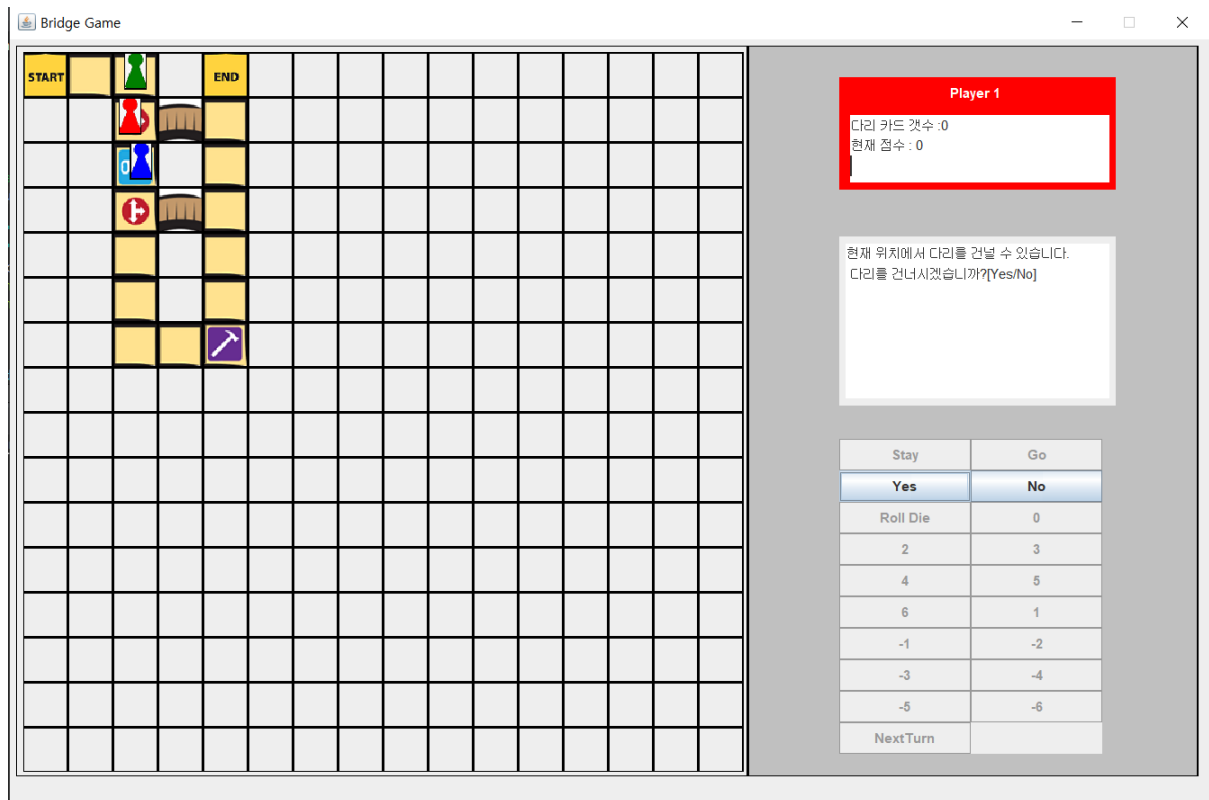
이동하고 싶은 칸 만큼의 버튼을 클릭하면 다음과 같이 NextTurn 버튼이 활성화 된다.



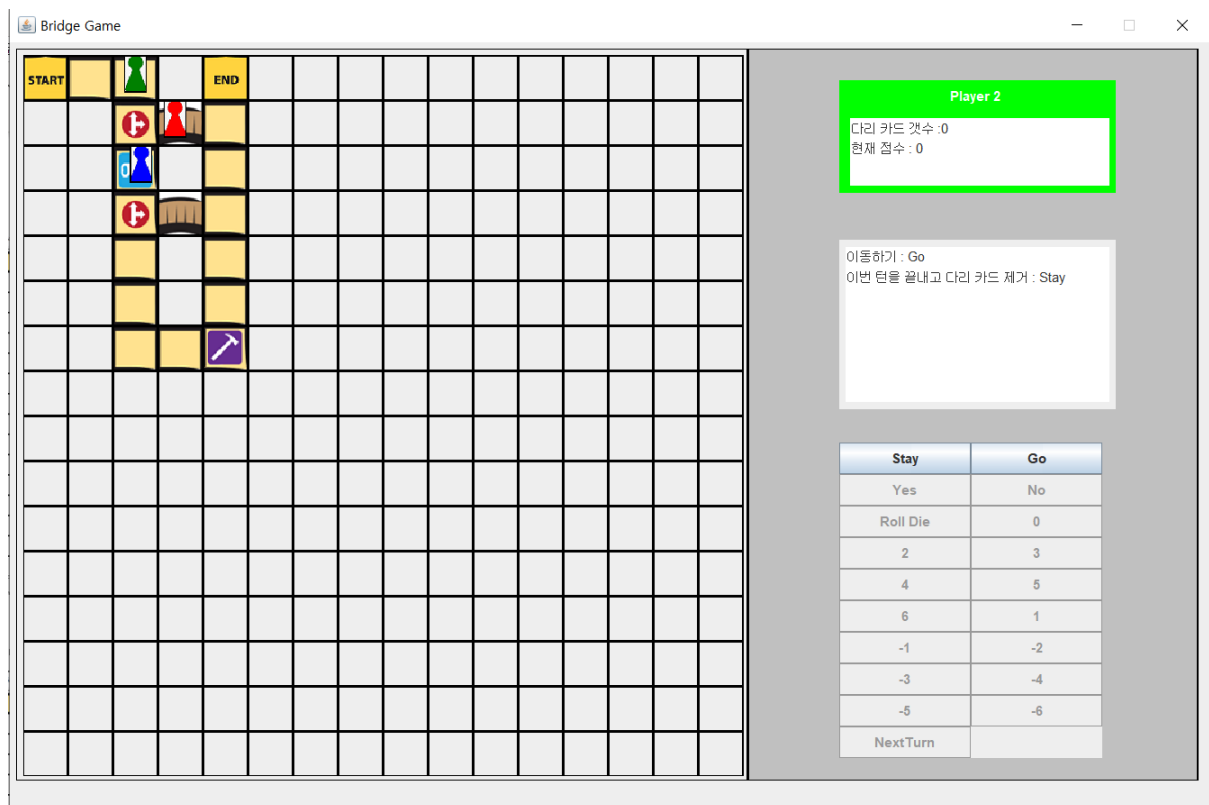
NextTurn을 누르면 플레이어의 턴이 넘어가고 이 과정이 반복되도록 구현하였다.



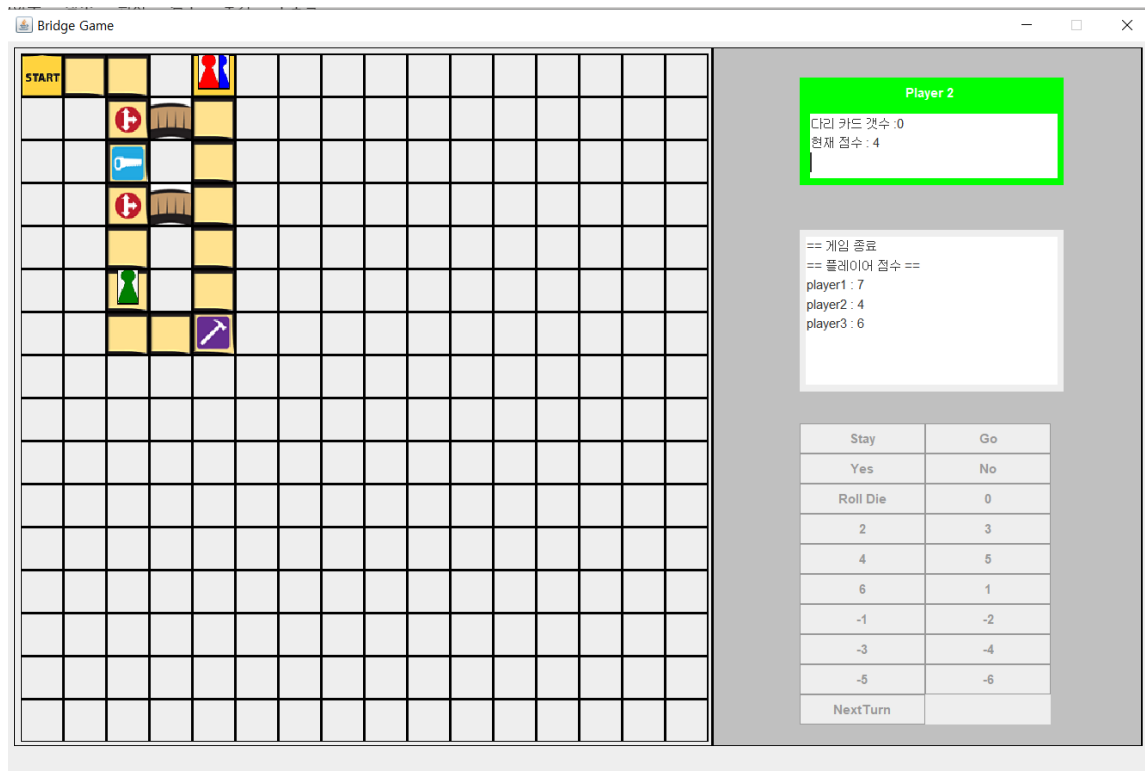
다음은 다리에 있을 때 다리를 건널지 선택하는 화면이다.



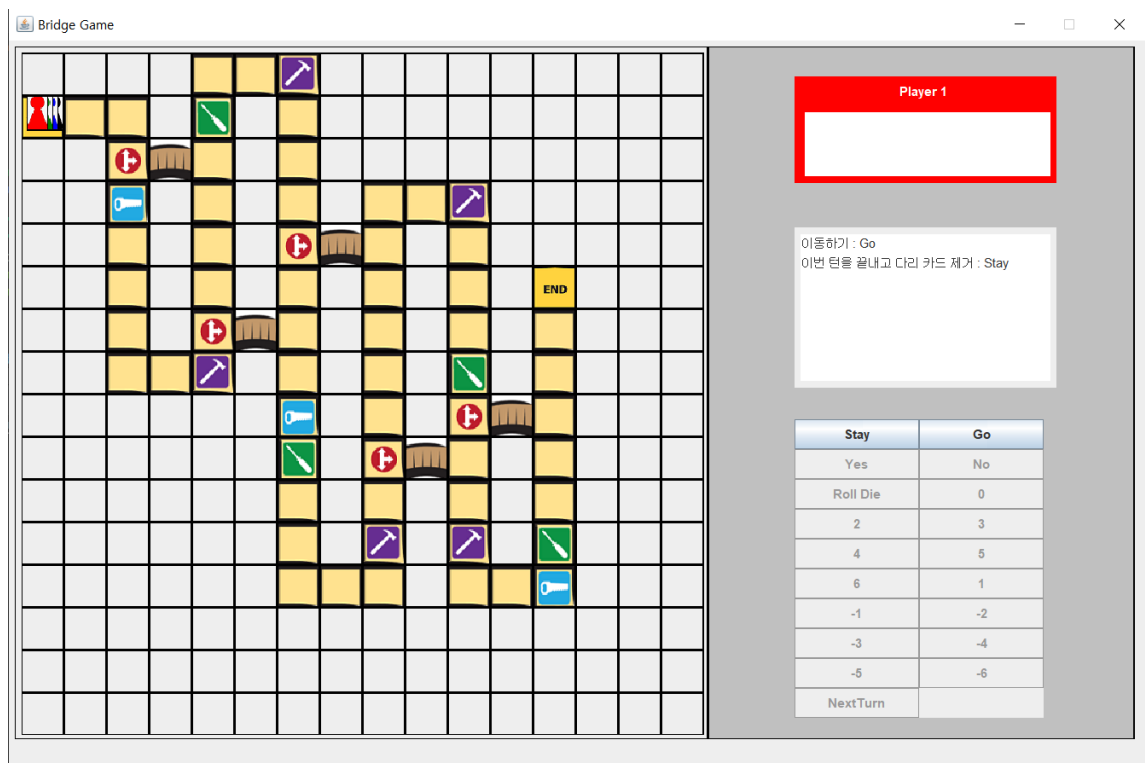
Yes를 눌러서 다리를 건너기를 선택할 수 있고 다음과 같이 다리를 건널 수 있다.



마지막으로 한 명의 플레이어가 맵에 남았을 때 다음과 같이 모든 button을 비활성화 시킨 뒤에 총 결과를 출력하면서 게임은 종료된다.



다음은 default.map을 호출 했을 때의 화면이다.



프로젝트 요구사항 비교표

프로젝트 명세서에 나와있는 요구사항들을 만족했는지에 대한 점검표이다.

요구사항	Console	GUI	비고
임의의 지도 파일 로드	O	X	GUI 화면 전환까지 구현하지 못하여 Console을 이용하여 맵을 받는다.
플레이어 수 입력	O	X	콘솔로 처리
플레이어 위치 출력	O	O	
플레이어가 가진 카드 수 출력	O	O	
종료 시 점수 출력	O	O	
매 턴마다 쉬는 기능 구현(다리 카드 감소)	O	O	
플레이어 입력 및 Invalid Input 검증	O	O	기존 요구사항에서는 사용자의 입력값을 검증하고 이동하였지만, 이를 미리 계산하고 사용자에게 제공하여 편의성을 향상시킴. 당연히 사용자의 Invalid Input은 다시 받도록 구현함
UI와 로직의 분리	O	O	
GUI / 콘솔 모두 구현	O	O	
아이템 점수 획득	O	O	
한명의 플레이어가 끝나면 뒤로가기 제한	O	O	
다리 건너기 선택	O	O	
다리 카드 증가	O	O	
플레이어 이동 시 다리 카드 고려	O	O	
등수에 따른 점수 차등 부여	O	O	