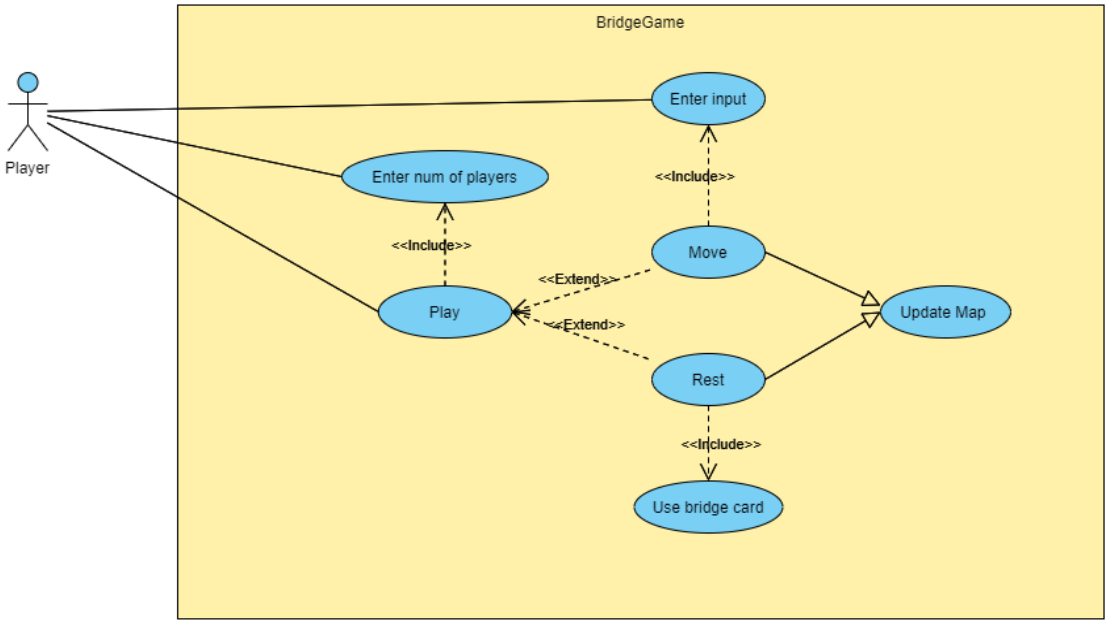
소프트웨어공학 Term Project

20186091 이동건

1. 요구 정의 및 분석

(1) 유스케이스 다이어그램



1) 플레이어 수 입력(Enter num of players) 유스케이스 명세서

: 게임을 시작하기 전 게임을 플레이 하는 사용자가 플레이어의 숫자를 게임에 입력하는 유스케이스이다. 게임을 시작 하기 전에 입력한다.

- 성공 시나리오

  1) 입력할 공간에 플레이어가 사용자 수를 입력한다.

  2) 입력한 숫자가 2~4 사이인지 확인한다.

  3) 2~4 사이가 맞으면 숫자만큼 플레이어 객체를 생성하고 맵 객체를 생성한다.

- 예외 흐름

  : 플레이어 숫자가 2와 4 사이가 아닌 경우, 빈칸으로 입력한 경우, 숫자가 아닌 경우

  ➔ 사용자에게 다시 입력하도록 메시지를 띄워 다시 입력하게 한다.

2) Move 유스케이스 명세서

: 시스템이 주사위를 굴린 후 주사위 값에서 다리 카드 수를 뺀 값 만큼의 인풋값을 받는다.

사용자가 인풋을 입력 한 것을 전제로 입력 값이 현재 상태에서 유효하면 움직인다.

- 성공 시나리오

  1) 사용자가 입력 한 인풋값을 시스템에서 유효성을 검사한다.

  2) 인풋 값에 따라 한 칸 씩 플레이어 말이 움직인다.

- 예외 흐름

  : 사용자의 입력이 유효하지 않는 경우

  ➔ 사용자에게 움직임이 유효하지 않다는 메시지를 띄우고 다시 입력하도록 한다. 플레이어의 말은 움직이지 않는다.

  : 주사위를 굴린 값에서 다리 카드 수를 뺀 값이 0인 경우

  ➔ 사용자에게 움직일 수 없다는 메시지를 띄우고 플레이어의 말은 움직이지 않고 다음 플레이어 턴으로 넘어간다.

3) Rest 유스케이스 명세서

: 사용자의 다리 카드 수가 1개 이상일 때 사용자가 한 턴 쉬려고 할 때 다리 카드를 하나 사용하여 한 턴 쉰다.

- 성공 시나리오

  1) 사용자가 Rest 버튼을 클릭한다.

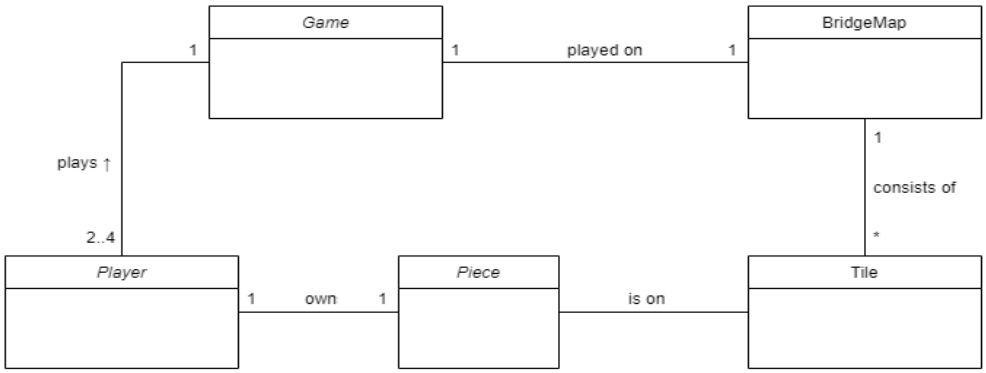
  2) 다리 카드를 하나 사용 후 다음 플레이어 턴으로 넘어간다.

- 예외 흐름

  : 다리 카드 수가 없는 경우

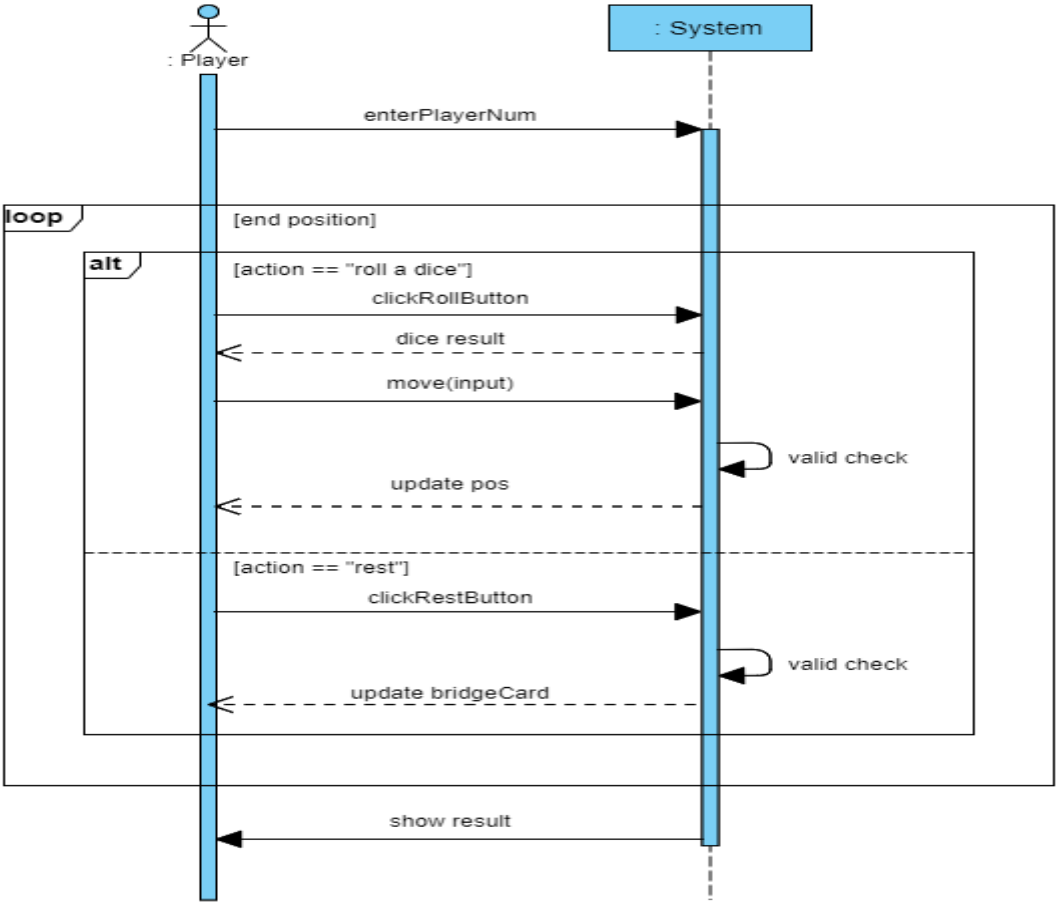
  ➔ Rest 버튼이 클릭 되지 않게 설정한다.

(2) 도메인 모델



게임은 BridgeMap에서 플레이 되며 이 맵은 여러 개의 타일로 구성되어 있다. 게임은 또한 2~4명의 플레이어가 플레이하며 각각의 플레이어는 말을 가진다. 이 말은 타일 위에 있다.

(3) SSD



플레이어 Actor가 플레이어 숫자를 입력하면서 게임이 시작되며 플레이어는 자신의 턴에 두 가지의 행동을 선택할 수 있다.

첫째, 플레이어가 주사위를 굴리기로, 즉 이동하기로 선택한 경우, 시스템에서 주사위를 굴린다. 주사위 값이 나오는 과정과 결과를 사용자에게 보여주며 이 결과값에서 다리 카드 수를 뺀 만큼 길이의 인풋을 입력한다. 시스템은 이 입력 값이 유효한지 체크를 하며 유효하면 플레이어의 말을 움직인다. 말의 움직임이 끝나면, 다음 플레이어로 턴을 넘긴다.

또한 플레이어는 자신의 턴을 쉬면서 넘길 수 있다. 쉬는 경우 다리 카드가 하나 소모되며 플레이어에게 이를 알리고 다음 플레이어로 턴을 넘긴다.

플레이어가 이동을 하다 도착점에 도달한 경우, 해당 플레이어 턴은 끝나며 나머지 플레이어들만 게임을 하게 된다. 만약 플레이 하는 플레이어가 한 명 남게 되는 경우, 게임은 끝나며 게임의 결과를 보여준다.

(4) Operation Contracts

Operation 1 : enterPlayerNum(input: int)

Cross Reference: 유스케이스: Enter num of players

Preconditions: None

Postconditions: - 입력한 input 숫자 크기의 플레이어 클래스의 배열이 생성된다. (객체 생성)

- 플레이어 객체가 가지는 말의 위치를 시작 위치로 조정한다. (속성 변경)

Operation 2 : move(input: String)

Cross Reference: 유스케이스: Move

Preconditions: 시스템에서 주사위를 굴린다.

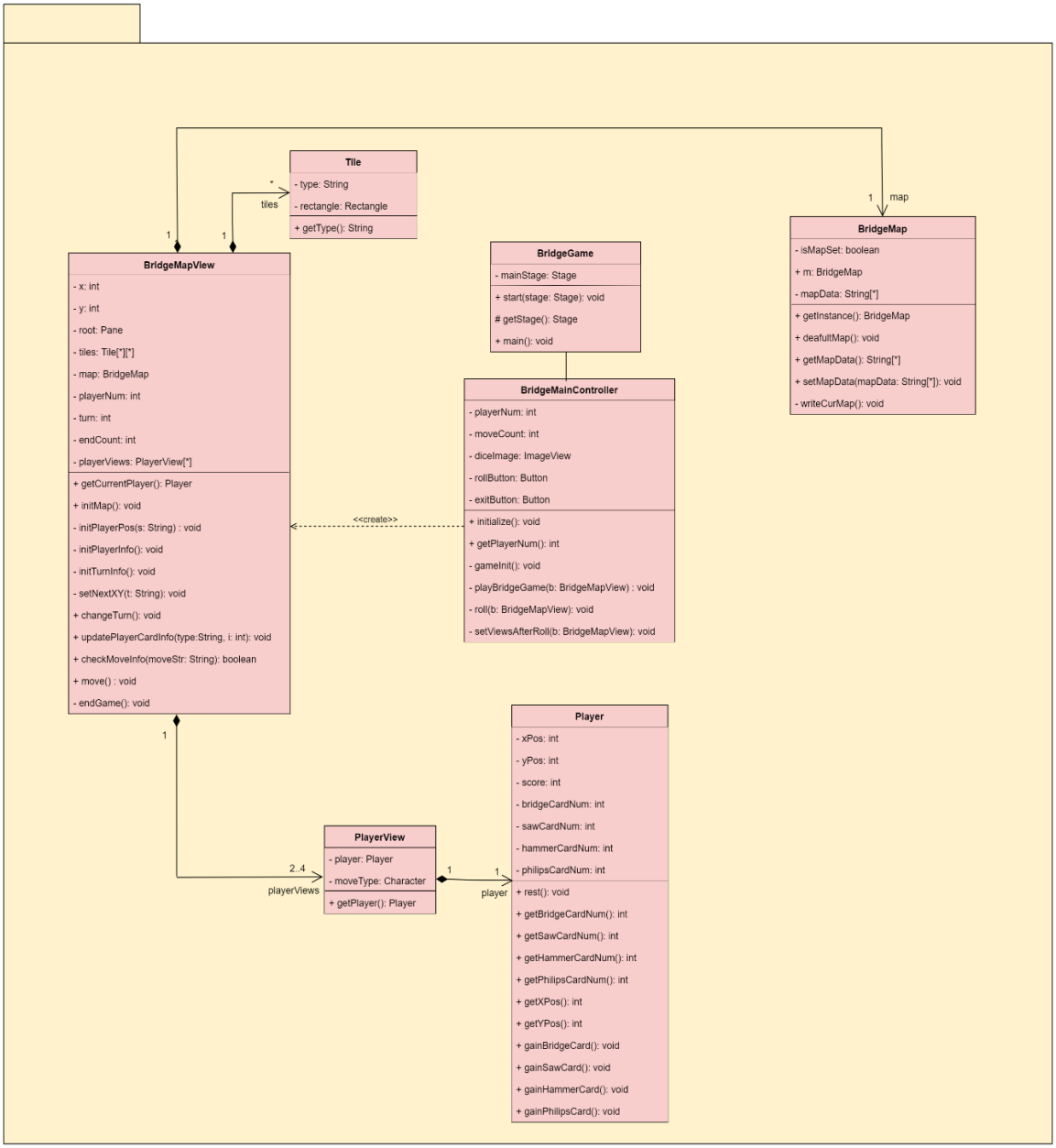
Postconditions: - 한 칸씩 움직이는데 이용하는 스레드 객체를 생성한다. (객체 생성)

- 스레드가 정해진 시간마다 sleep하면서 말의 위치를 변경한다. (속성 변경)

- 말의 위치를 맵에서도 보여준다. 맵에서의 위치랑 플레이어의 위치를 연동한다.(관계 형성)

2. 설계

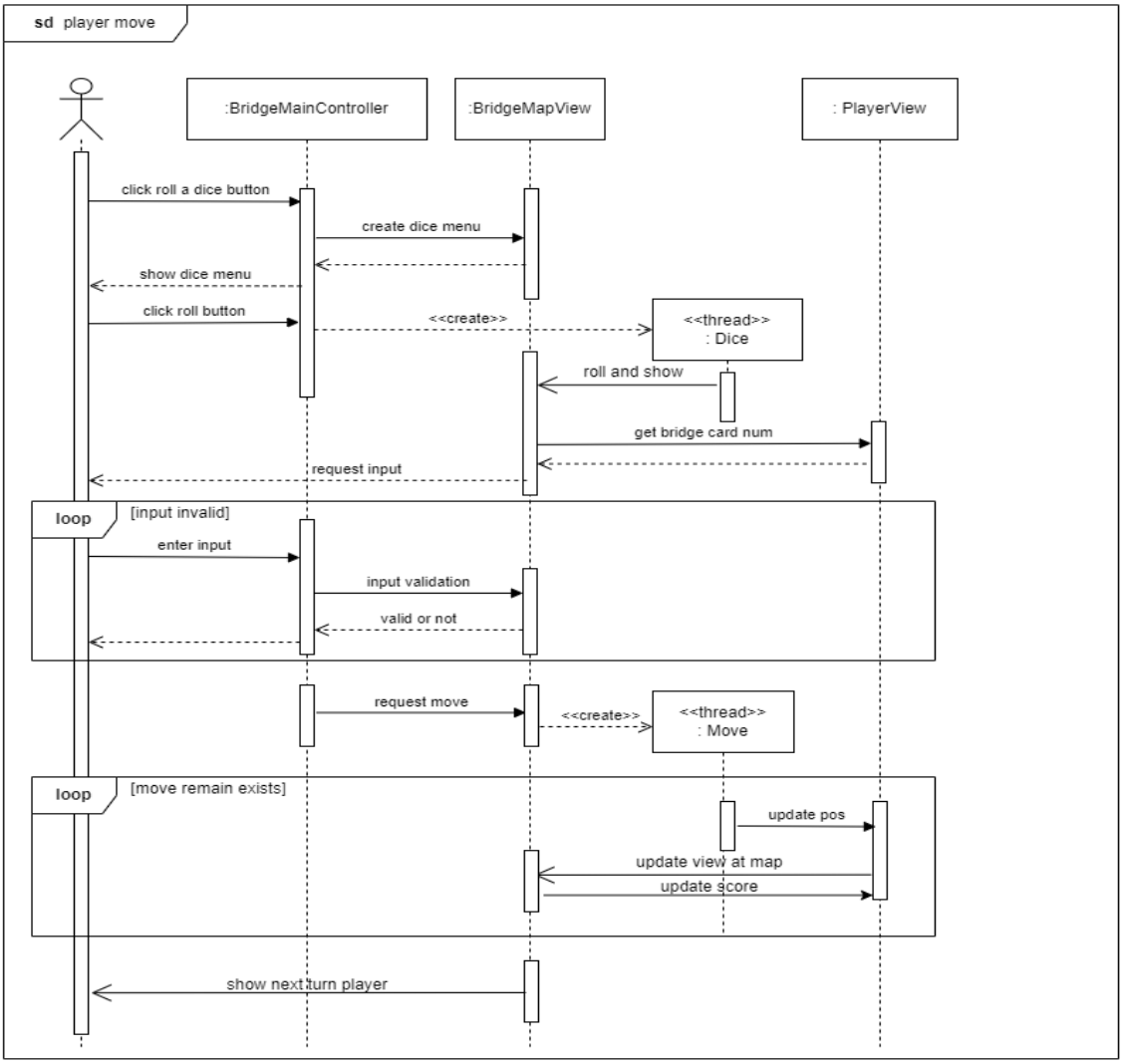
(1) Design Class Diagram



Domain Model을 기반으로 작성하였으며 유지 보수를 용이하게 하기 위해 MVC 패턴에 맞게 클래스를 설계하였다. BridgeMainController 클래스는 사용자가 버튼을 클릭하거나 인풋 값을 입력한 경우 이와의 상호작용을 정의하였다. BridgeMapView 클래스는 맵 화면 및 전체 게임화면을 보여주는 기능을 한다. 이 클래스의 객체는 Tile 클래스의 객체와 PlayerView 클래스의 객체를 가지고 있다. 또한 BridgeMap 클래스의 데이터를 기반으로 타일 클래스의 객체들이 생성되어 맵을 구성한다. Player 클래스는 PlayerView 클래스의 모델 역할을 하며 Player의 데이터를 담고 있다.

(2) Sequence Diagram

1) Player Move

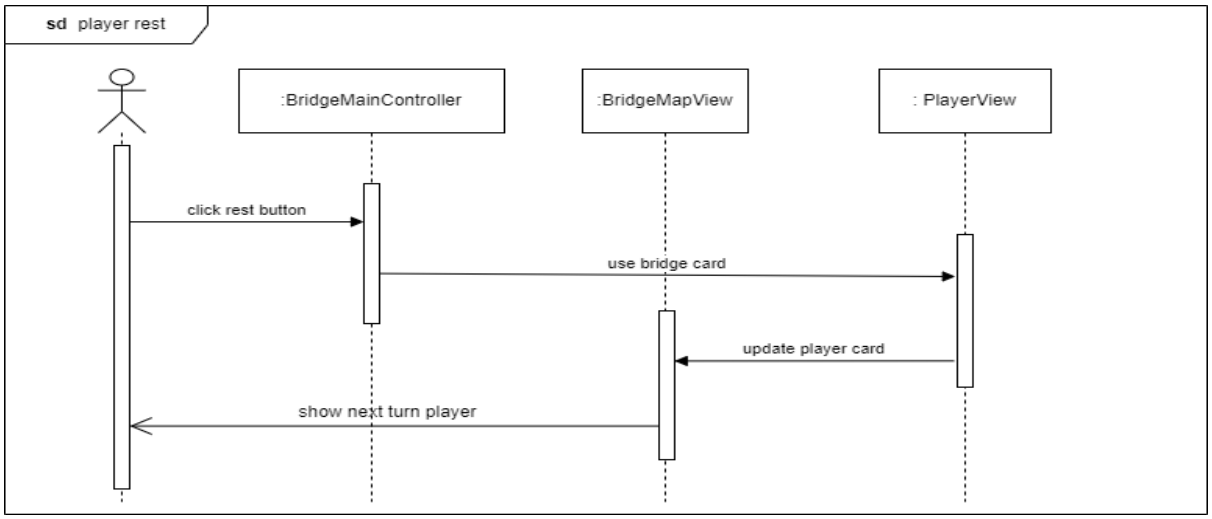


플레이어가 자신의 턴에 roll a dice 버튼을 클릭 한 것이며, 이는 자신의 턴에 말을 움직이기로 결정한 것이다. 버튼 클릭 시, 컨트롤러는 게임 화면에 주사위 화면을 동적으로 생성한다. 이 화면에는 주사위의 사진과 roll 버튼, 플레이어가 주사위를 굴리고 난 후 입력할 텍스트 필드가 포함된다. 플레이어가 roll 버튼을 누르면, 컨트롤러는 주사위 스레드를 생성하여 주사위를 굴린다. 주사위를 다 굴리면, 결과값을 보여준다. 이 결과값에서 플레이어의 다리 카드 수를 뺀 만큼 플레이어는 입력할 수 있다. 플레이어는 유효한 입력(맵에서 움직일 수 있는 입력, 게임을 끝난 플레이어가 있으면 뒤로 가는 이동이 없는 입력)을 입력 할 때 까지 입력하게 되며 시스템은 입력이 들어올 때마다 유효성을 체크한다.

유효한 입력 값이 들어왔으면 컨트롤러는 플레이어의 말이 움직이도록 요청하며 맵 뷰는 Move

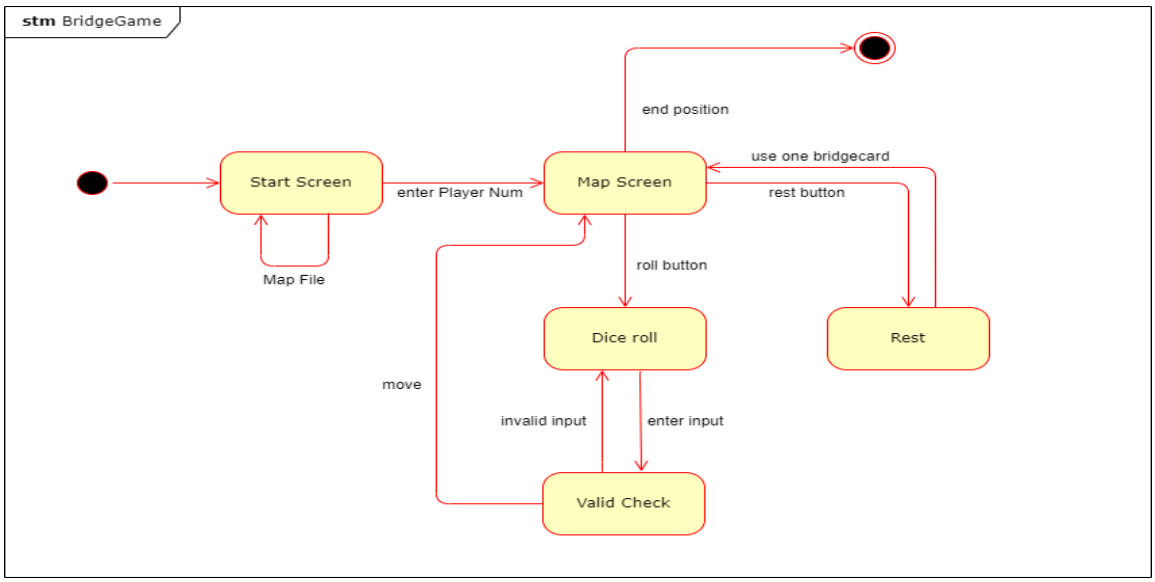
스레드를 생성한다. 입력 값의 한 글자 씩 분석하여 맞게 이동하며 남은 글자가 없을 때 까지 이동한다. 이때 플레이어 말의 위치를 한 칸 씩 이동하며 이후에 맵 뷰에 보여진다. 만약 도구 카드를 먹게 된 경우 점수를 업데이트 하며 다리를 지나간 경우 다리 카드를 하나 늘린다. 더 이상 이동할 게 없다면 맵 뷰는 다음 플레이어의 턴을 보여주며 넘긴다.

2) Player Rest



플레이어가 자신의 턴에 rest button을 클릭한 것이며, 이는 자신의 턴에 한 턴 쉬기로 결정한 것이다. 이 버튼은 플레이어의 카드가 없는 경우는 활성화되지 않는다. 버튼을 클릭하면, 컨트롤러는 플레이어의 다리 카드를 사용하게 하며 사용한 후 플레이어 카드 수를 게임 화면에 업데이트 한다. 이 후 맵 뷰는 다음 턴의 플레이어를 보여준다.

3) StateChart



플레이어는 플레이어 숫자를 입력하며 시작화면에서 게임 맵화면으로 넘어가며 매 턴 주사위를 굴리거나 쉬며 턴을 진행한다. 플레이어가 도착점에 도달한 경우, 해당 플레이어의 게임은 끝나게 된다.

3. 구현

(1) 가정

- 게임의 맵 데이터는 게임을 처음 실행 했을 때 아무 파일을 선택하지 않은 경우 default.map 파일의 데이터로 설정되며 다음 실행부터는 파일을 선택하지 않으면 직전 실행했던 맵 데이터로 설정된다.
- 맵에 있는 각각의 타일 위의 카드들은 플레이어 중 최초 한 명의 플레이어가 먼저 얻은 경우 카드는 타일에서 사라지게 된다.
- 게임이 종료될 때 점수가 같다면 먼저 종료지점에 들어온 사람의 순위가 더 높다.

(2) 소스코드

1) UI 패키지 및 리소스 파일

- BridgeGame.java

  : 게임을 시작하는 메인 함수가 있는 클래스

```
public class BridgeGame extends Application {
    // 화면이 보이는 Stage
    public static Stage mainStage;

    // 시작 화면 보여주는 함수
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(BridgeGame.class.getResource("main.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        mainStage = stage;
        stage.setTitle("Bridge Game");
        stage.setResizable(true);
        stage.setScene(scene);
        stage.show();
    }

    // Stage getter
    protected static Stage getStage(){
        return mainStage;
    }

    public static void main(String[] args) {
        launch();
    }
}
```

- main.fxml

: 시작 화면을 나타내는 fxml 파일

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<StackPane prefHeight="240.0" prefWidth="320.0" xmlns="http://javafx.com/javafx/11.0.2"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.lee.ui.BridgeMainController">
   <children>
      <VBox alignment="CENTER" prefHeight="200.0" prefWidth="100.0">
         <children>
            <Label text="Bridge Game">
               <font>
                  <Font name="Berlin Sans FB Bold" size="30.0" />
               </font>

            </Label>
            <Label fx:id="mainActionLabel" prefHeight="50.0"  text="플레이 할 플레이어 수를
입력하세요. (2~4)" alignment="BOTTOM_CENTER">
               <font>
                  <Font name="Hancom Gothic Regular" size="12.0" />
               </font>
               <VBox.margin>
                  <Insets bottom="5.0" />
               </VBox.margin>
            </Label>
            <HBox alignment="CENTER" maxHeight="-Infinity" maxWidth="-Infinity"
prefHeight="50.0" prefWidth="200.0">
               <TextField fx:id="playerNumTF" alignment="CENTER" layoutX="84.0" layoutY="32.0"
maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="30.0" prefWidth="40.0">
                  <font>
                     <Font name="Hancom Gothic Regular" size="12.0" />
                  </font></TextField>
               <Button fx:id="playerNumBtn" mnemonicParsing="false" text="입력 확인"
prefWidth="70" prefHeight="30">
                  <HBox.margin>
                     <Insets left="10.0" />
                  </HBox.margin></Button>
            </HBox>
            <Button fx:id="fileBtn" layoutX="100.0" layoutY="100.0" mnemonicParsing="false" text
= "맵 설정" prefWidth="60" prefHeight="30">
               <VBox.margin>
                  <Insets top="5.0"/>
               </VBox.margin>
            </Button>
         </children>
         <StackPane.margin>
            <Insets />
         </StackPane.margin></VBox>
   </children>
</StackPane>
```

- BridgeMainController.java

: 사용자가 입력한 input 및 버튼 클릭을 관리하는 클래스

```
public class BridgeMainController implements Initializable {
    // 게임 플레이어 수
    private static int playerNum;
    // 현재 플레이어
    private Player curPlayer = null;
    // 움직일 수 있는 칸 수
    private int moveCount;

    // 주사위 사진 및 버튼, 라벨
    private ImageView diceImage;
    private Button rollButton;
    private Label rollLabel;
    private TextField moveTF;
    private Button exitButton;

    // 시작 화면 뷰들
    @FXML private Label mainActionLabel;
    @FXML private Button playerNumBtn;
    @FXML private TextField playerNumTF;
    @FXML private Button fileBtn;


    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        playerNumBtn.setOnAction((ActionEvent -> {
            gameInit();
        }));

        playerNumTF.setOnKeyPressed(keyEvent -> {
            if (keyEvent.getCode().equals(KeyCode.ENTER)){
                gameInit();
            }
        });
        fileBtn.setOnAction((ActionEvent -> {
            loadFile();
        }));
    }

    // 시작 화면을 설정하는 함수
    private void gameInit() {
        String strNum = playerNumTF.getText();
        if (strNum.isEmpty()) {
            mainActionLabel.setText("숫자를 입력해 주세요. (2 ~ 4)");
            return;
        }
        else{
            for (int i=0; i<strNum.length(); i++){
                if (Character.isDigit(strNum.charAt(i)) == false){
                    mainActionLabel.setText("숫자를 입력해 주세요. (2 ~ 4)");
                    return;
                }
            }
            if (Integer.parseInt(strNum) < 2 || Integer.parseInt(strNum) > 4){
                mainActionLabel.setText("잘못된 숫자 범위입니다. 숫자를 입력해주세요. (2 ~ 4)");
                return;
            }

            // 플레이어 수 설정
            playerNum = Integer.parseInt(strNum);

            try{
                // 새로운 화면 만들기
                StackPane stackPane = (StackPane) playerNumBtn.getScene().getRoot();
```

```
                Stage primaryStage = BridgeGame.getStage();
                primaryStage.setX(500);
                primaryStage.setY(50);
                primaryStage.setWidth(1250);
                primaryStage.setHeight(900);
                BridgeMapView bridgeMapView = new BridgeMapView();
                stackPane.getChildren().remove(0);
                stackPane.getChildren().add(bridgeMapView.getScrollPane());

                playBridgeGame(bridgeMapView);
            } catch (Exception e){
                e.printStackTrace();
            }
        }
    }
    // 플레이어 숫자 getter
    public static int getPlayerNum() {return playerNum;}

    // 파일 설정 버튼을 클릭한 경우
    private void loadFile(){
        FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Map 파일(*.map)",
"*.map"));
        File file = fileChooser.showOpenDialog(BridgeGame.getStage());

        if (file != null) {
            ArrayList<String> mapData = new ArrayList<>();
            BufferedReader br;
            try {
                br = new BufferedReader(new InputStreamReader(new FileInputStream(file)));
                String line;
                while ((line = br.readLine()) != null) {
                    mapData.add(line);
                }
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            BridgeMap.getInstance().setMapData(mapData.toArray(new String[mapData.size()]));
            mainActionLabel.setText("\t    맵 데이터가 변경되었습니다. \n플레이 할 플레이어 수를
입력하세요. (2~4)");
        }
        else mainActionLabel.setText("플레이 할 플레이어 수를 입력하세요. (2~4)");

    }

    // BridgeGame 게임 화면 내부의 버튼들의 클릭과 인풋 입력을 관리하는 함수
    private void playBridgeGame(BridgeMapView bridgeMapView){
        bridgeMapView.setMyClickListener(new BridgeMapView.onItemClick() {
            @Override
            public void onRollDiceClick(Button diceButton, Button restButton) {
                // 다른 버튼들 비활성화
                diceButton.setDisable(true);
                restButton.setDisable(true);
                // 현재 플레이어
                curPlayer = bridgeMapView.getCurrentPlayer();

                // 주사위 화면을 만들고 보여준다
                Pane root = (Pane) bridgeMapView.getContentPane();
                StackPane dicePane = new StackPane();
                dicePane.setPrefSize(350, 230);
                dicePane.relocate(820, 200);

                Rectangle rectangle = new Rectangle(350, 230);
                rectangle.setFill(Color.WHITE);
                rectangle.setStroke(Color.BLACK);
                rectangle.setStrokeWidth(2);

                File file = new File("src/main/resources/com/lee/image/dice1.png");
```

```
            diceImage= new ImageView(new Image(file.toURI().toString()));
            diceImage.setFitHeight(100);
            diceImage.setFitWidth(100);
            diceImage.setPreserveRatio(true);
            StackPane.setAlignment(diceImage, Pos.TOP_LEFT);
            StackPane.setMargin(diceImage, new Insets(20, 0, 0, 10));

            rollButton = new Button("Roll");
            rollButton.setPrefSize(50, 30);
            StackPane.setAlignment(rollButton, Pos.BOTTOM_LEFT);
            StackPane.setMargin(rollButton, new Insets(0, 0 , 50, 35));

            rollLabel = new Label("Click Roll button to \nroll a dice.");
            rollLabel.setPrefSize(220, 100);
            rollLabel.setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR, 17));
            StackPane.setAlignment(rollLabel, Pos.TOP_CENTER);
            StackPane.setMargin(rollLabel, new Insets(0, 0, 0 , 120));

            moveTF = new TextField();
            moveTF.setPrefSize(180, 30);
            moveTF.setMaxWidth(180);
            moveTF.setFont(Font.font("Arial", 13));
            moveTF.setPromptText("Combinations of U,D,L,R");
            moveTF.setFocusTraversable(false);
            moveTF.setDisable(true);
            StackPane.setAlignment(moveTF, Pos.CENTER_RIGHT);
            StackPane.setMargin(moveTF, new Insets(0, 45, 20, 0));

            exitButton = new Button("Exit");
            exitButton.setPrefSize(50, 30);
            exitButton.setDisable(true);
            StackPane.setAlignment(exitButton, Pos.BOTTOM_RIGHT);
            StackPane.setMargin(exitButton, new Insets(0, 35, 50, 0));

            dicePane.getChildren().addAll(rectangle, diceImage, rollButton, rollLabel,
moveTF, exitButton);
            root.getChildren().add(dicePane);

            // roll 버튼
            rollButton.setOnAction(actionEvent -> {
                roll(bridgeMapView);
            });

            // 주사위 화면 종료 버튼
            exitButton.setOnAction(actionEvent -> {
                root.getChildren().remove(dicePane);
                diceButton.setDisable(false);
                exitButton.setDisable(false);
                bridgeMapView.changeTurn(restButton);
            });

            // 주사위 굴린 후 움직일 이동을 입력한 경우
            moveTF.setOnKeyPressed(keyEvent -> {
                if (keyEvent.getCode().equals(KeyCode.ENTER)){
                    String moveStr = moveTF.getText().toUpperCase(Locale.ROOT);
                    if (moveStr.isEmpty()){
                        rollLabel.setText("Length should be "+moveCount +".\nInput
combinations of \nU, D, L, R or u, d, l, r.");
                    }
                    else {
                        // 인풋값의 유효성 검사
                        moveStr = moveStr.replaceAll(" ", "");
                        if (moveStr.length() != moveCount) {
                            rollLabel.setText("Length should be "+moveCount +".\nInput
combinations of \nU, D, L, R or u, d, l, r.");
                        }
                        else if (bridgeMapView.checkMoveInfo(moveStr) == false){
                            rollLabel.setText("Invalid move!! Input again.\nLength should be
"+moveCount +".");
                        } else{
                            // 유효한 경우
                            rollLabel.setText("Player is moving..");
```

```
                            bridgeMapView.move(moveStr, exitButton, rollLabel);
                            moveTF.setDisable(true);
                        }
                    }
                }
            });
        }

        @Override
        public void onRestClick() {
            // rest 버튼 클릭시
            Player curPlayer = bridgeMapView.getCurrentPlayer();
            curPlayer.rest();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            bridgeMapView.updatePlayerCardInfo("Bridge", 1);
        }

        @Override
        public void onExitClick() {
            // 게임 종료 버튼 클릭시
            System.exit(1);
        }
    });
}

// 주사위를 굴리는 스레드를 생성 후 주사위 결과를 보여주는 함수
private void roll(BridgeMapView bridgeMapView){
    Random random = new Random();
    final int[] randomNum = {0};

    Thread thread = new Thread(){
        public void run() {
            try {
                rollButton.setDisable(true);
                for (int i = 0; i < 15; i++) {
                    randomNum[0] = (random.nextInt(6) + 1);
                    File file = new File("src/main/resources/com/lee/image/dice" +
randomNum[0] + ".png");
                    diceImage.setImage(new Image(file.toURI().toString()));
                    diceImage.setFitHeight(100);
                    diceImage.setFitWidth(100);
                    diceImage.setPreserveRatio(true);
                    Thread.sleep(50);
                }
                moveCount = randomNum[0] - curPlayer.getBridgeCardNum();
                setViewsAfterRoll(bridgeMapView);
                this.interrupt();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    };
    thread.start();
}

// 주사위를 굴린 후 텍스트 필드 및 라벨(얼마나 움직일 수 있는지 보여주는 라벨) 설정
private void setViewsAfterRoll(BridgeMapView bridgeMapView){
    moveTF.setDisable(false);
    // 이동 가능한 move 가 0 번일 때
    if (moveCount <= 0 ) {
        moveCount = 0;
        moveTF.setDisable(true);
        exitButton.setDisable(false);
        bridgeMapView.getCurrentPlayer().rest();
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                rollLabel.setText("Cannot move. Rest\nusing one bridge card.");
```

```
                bridgeMapView.updatePlayerCardInfo("Bridge", 1);
            }
        });
    }
    else {
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                rollLabel.setText("You can move "+moveCount + " moves.\n" + "Input
combinations of \nU, D, L, R or u, d, l, r.");
                moveTF.setPromptText("   " + moveCount + " moves possible");
            }
        });
    }

  }
}
```

- BridgeMapView.java

: BridgeGame의 게임화면 및 맵 화면을 보여주는 클래스, 게임화면에서 보여지는 모든 것들을 담고 있음. 플레이어의 말(PlayerView)가 움직이는 move함수는 스레드를 새로 생성하여 한 칸씩 이동하도록 하였다. onItemClick이라는 인터페이스를 클래스 내부에 생성하여 컨트롤러 클래스에서 구현하도록 하여 클릭리스너를 외부에서 구현하도록 정의함.

```
public class BridgeMapView {
    // 타일 사이즈 및 플레이어 뷰의 컬러, 시작 위치 등을 정의
    public static final int TILE_SIZE = 36;
    public static final Color[] colors = {Color.BLUE, Color.RED, Color.GREEN, Color.PURPLE};
    public static int WIDTH = 25;
    public static int HEIGHT = 18;
    public static int START_X = 3;
    public static int START_Y = 5;

    private int x, y;
    private Pane root;
    private ScrollPane scrollPane;
    private StackPane infoPane, turnPane;
    private Group tileGroup = new Group();

    // 맵을 구성하는 타일 객체의 배열과 Model 클래스의 객체
    private Tile[][] tiles;
    private BridgeMap map;

    // 게임에 사용되는 초기 설정값들
    private int playerNum = 0;
    private int turn = 1;
    private int endCount = 0;
    private int[] endOrders;
    private PlayerView[] playerViews;
    private Label[][] playerCards;
    private Label turnLabel;
    private onItemClick myItemClick;

    // 버튼 클릭시 수행 행동을 인터페이스에 정의하고 컨트롤러에서 이를 구현하게 하여 클릭 리스너를 컨트롤러에
선언
    public interface onItemClick {
        void onRollDiceClick(Button diceButton, Button restButton);

        void onRestClick();

        void onExitClick();
    }

    public void setMyClickListener(onItemClick myItemClick) {
```

```
        this.myItemClick = myItemClick;
    }

    public BridgeMapView() {
        initMap();
    }

    // 각종 getter
    public Parent getContentPane() {
        return root;
    }

    public Parent getScrollPane() {
        return scrollPane;
    }

    public Player getCurrentPlayer() {
        return playerViews[turn - 1].getPlayer();
    }

    // 초기 맵 설정 함수
    public void initMap() {
        map = BridgeMap.getInstance();
        // 입력 받은 플레이어 수 만큼 플레이어 뷰 객체 생성
        playerNum = BridgeMainController.getPlayerNum();
        playerViews = new PlayerView[playerNum];

        initEndOrders();
        String[] mapData = map.getMapData();
        initMapSize();
        if (mapData == null) {
            throw new IllegalArgumentException("map 데이터가 null 일 수 없습니다.");
        }

        String prevMove = null;
        // mapData 의 각 줄의 첫 번째 값에 따라 타일 객체 생성자 호출
        for (int i = 0; i < mapData.length; i++) {
            String[] temp = mapData[i].split(" ");
            Tile tile;
            if (i == 0 && temp[0].equals("S")) {
                tile = new Tile("START", x, y, null, temp[1]);
                switch (temp[1]) {
                    case "R":
                        tiles[x + 1][y] = tile;
                        x += 2;
                        break;
                    case "D":
                        tiles[x][y + 1] = tile;
                        y += 2;
                        break;
                }
                initPlayerPos(temp[1]);
            } else if (temp[0].equals("E")) {
                if (prevMove.equals("U") || prevMove.equals("R"))
                    tile = new Tile("END", x, y - 1, null, null);
                else tile = new Tile("END", x, y, null, null);
                tiles[x][y] = tile;

            } else if (temp[0].equals("S")) {
                tile = new Tile("SAW", x, y, temp[1], temp[2]);
                tiles[x][y] = tile;
                setNextXY(temp[2]);
            } else if (temp[0].equals("P")) {
                tile = new Tile("PHILIPS", x, y, temp[1], temp[2]);
                tiles[x][y] = tile;
                setNextXY(temp[2]);
            } else if (temp[0].equals("H")) {
                tile = new Tile("HAMMER", x, y, temp[1], temp[2]);
                tiles[x][y] = tile;
                setNextXY(temp[2]);
            } else if (temp[0].equals("B")) {
                tile = new Tile("INTERSECT", x, y, temp[1], temp[2]);
```

```
            tiles[x][y] = tile;
            Tile bridge = new Tile("BRIDGE", x + 1, y, "L", "R");
            tiles[x + 1][y] = bridge;
            tileGroup.getChildren().add(bridge);
            setNextXY(temp[2]);
        } else {
            tile = new Tile("CELL", x, y, temp[1], temp[2]);
            tiles[x][y] = tile;
            setNextXY(temp[2]);

        }
        tileGroup.getChildren().add(tile);
        if (1 <= i && i < mapData.length - 1) prevMove = temp[2];

    }

    initPlayerInfo();
    initTurnInfo();

}

// 끝난 순서를 담는 array 초기화
private void initEndOrders() {
    endOrders = new int[playerNum];
    for (int i = 0; i < playerNum; i++) {
        endOrders[i] = -1;
    }
}

// 맵 사이즈 초기화
private void initMapSize() {
    int width = map.getMapWidth();
    int height = map.getMapHeight();
    int widthGap = map.getStartWidthGap();
    int heightGap = map.getStartHeightGap();

    START_X += widthGap;
    START_Y += heightGap;
    WIDTH = START_X + width;
    HEIGHT = START_Y + height;

    x = START_X;
    y = START_Y;
    tiles = new Tile[WIDTH + 1][HEIGHT + 1];

    scrollPane = new ScrollPane();
    scrollPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.AS_NEEDED);
    root = new Pane();
    root.setPrefSize(WIDTH * TILE_SIZE + 150, HEIGHT * TILE_SIZE + 150);
    root.getChildren().add(tileGroup);
    scrollPane.setContent(root);
}

// player 들 시작 위치로 배치
private void initPlayerPos(String s) {
    int temp_x = START_X;
    int temp_y = START_Y;

    int temp_XMargin = 0;
    int temp_YMargin = 0;

    if (s.equals("R")) {
        temp_XMargin = 12;
        temp_YMargin = 4;
        temp_x++;
    } else {
        temp_YMargin = 12;
        temp_XMargin = 4;
        temp_y++;
    }

    for (int i = 0; i < playerNum; i++) {
        playerViews[i] = new PlayerView(i + 1, temp_x, temp_y, colors[i]);
```

```
            StackPane.setAlignment(playerViews[i], Pos.BASELINE_CENTER);
            playerViews[i].relocate(temp_x * TILE_SIZE + temp_XMargin, temp_y * TILE_SIZE +
temp_YMargin);
            playerViews[i].getPlayer().setPos(temp_x, temp_y);
            if (s.equals("R")) temp_YMargin += 12;
            else temp_XMargin += 12;
            root.getChildren().add(playerViews[i]);
        }
    }

    // 각 플레이어의 카드 각각의 수가 담긴 보드 판 생성
    private void initPlayerInfo() {
        int width = 720;
        infoPane = new StackPane();
        infoPane.setPrefSize(width, 120);
        infoPane.relocate(2 * TILE_SIZE, 10);
        infoPane.setAlignment(Pos.CENTER_LEFT);


        Rectangle rectangle = new Rectangle(width, 120);
        rectangle.setFill(Color.WHITE);
        rectangle.setStroke(Color.BLACK);
        rectangle.setStrokeWidth(2);

        Label label = new Label("Boards");
        label.setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR, 16));
        StackPane.setAlignment(label, Pos.TOP_CENTER);
        StackPane.setMargin(label, new Insets(8, 0, 0, 0));
        infoPane.getChildren().addAll(rectangle, label);

        StackPane[] playerStackPane = new StackPane[playerNum];
        Rectangle[] playerRect = new Rectangle[playerNum];
        playerCards = new Label[playerNum][5];
        int xMargin = 0;
        for (int i = 0; i < playerNum; i++) {
            playerStackPane[i] = new StackPane();
            playerStackPane[i].setPrefSize(width / playerNum, 100);
            StackPane.setMargin(playerStackPane[i], new Insets(30, 0, 0, xMargin));
            playerStackPane[i].setAlignment(Pos.TOP_CENTER);

            playerRect[i] = new Rectangle();
            playerRect[i].setWidth(width / playerNum);
            playerRect[i].setHeight(100);
            playerRect[i].setStroke(Color.BLACK);
            playerRect[i].setStrokeWidth(2);
            playerRect[i].setFill(Color.WHITE);
            StackPane.setAlignment(playerRect[i], Pos.CENTER_LEFT);
            playerStackPane[i].getChildren().add(playerRect[i]);

            playerCards[i][0] = new Label("Player " + (i + 1));
            playerCards[i][0].setFont(Font.font("Arial", FontWeight.EXTRA_BOLD,
FontPosture.REGULAR, 14));
            playerCards[i][0].setTextFill(colors[i]);
            StackPane.setAlignment(playerCards[i][0], Pos.TOP_LEFT);
            StackPane.setMargin(playerCards[i][0], new Insets(10, 0, 0, 20));

            playerCards[i][1] = new Label("Bridge Card : " +
playerViews[i].getPlayer().getBridgeCardNum() + " cards");
            playerCards[i][1].setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR,
12));
            StackPane.setAlignment(playerCards[i][1], Pos.TOP_LEFT);
            StackPane.setMargin(playerCards[i][1], new Insets(30, 0, 0, 20));

            playerCards[i][2] = new Label("Saw Card : " +
playerViews[i].getPlayer().getSawCardNum() + " cards");
            playerCards[i][2].setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR,
12));
            StackPane.setAlignment(playerCards[i][2], Pos.TOP_LEFT);
            StackPane.setMargin(playerCards[i][2], new Insets(45, 0, 0, 20));

            playerCards[i][3] = new Label("Hammer Card : " +
playerViews[i].getPlayer().getHammerCardNum() + " cards");
            playerCards[i][3].setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR,
12));
            StackPane.setAlignment(playerCards[i][3], Pos.TOP_LEFT);
```

```
            StackPane.setMargin(playerCards[i][3], new Insets(60, 0, 0, 20));

            playerCards[i][4] = new Label("Philips Card : " +
playerViews[i].getPlayer().getPhilipsCardNum() + " cards");
            playerCards[i][4].setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR,
12));
            StackPane.setAlignment(playerCards[i][4], Pos.TOP_LEFT);
            StackPane.setMargin(playerCards[i][4], new Insets(75, 0, 0, 20));

            playerStackPane[i].getChildren().addAll(playerCards[i][0], playerCards[i][1],
playerCards[i][2], playerCards[i][3], playerCards[i][4]);
            infoPane.getChildren().add(playerStackPane[i]);

            xMargin += width / playerNum;
        }

        root.getChildren().add(infoPane);
    }

    // 현재 턴 정보를 보여주는 판 생성
    private void initTurnInfo() {
        turnPane = new StackPane();
        turnPane.setPrefSize(350, 125);
        turnPane.relocate(820, 15);

        Rectangle rectangle = new Rectangle(350, 125);
        rectangle.setFill(Color.WHITE);
        rectangle.setStroke(Color.BLACK);
        rectangle.setStrokeWidth(2);

        turnLabel = new Label("Player " + turn + " Turn");
        turnLabel.setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR, 18));
        turnLabel.setTextFill(colors[turn - 1]);
        turnPane.setAlignment(Pos.TOP_CENTER);
        StackPane.setMargin(turnLabel, new Insets(10, 0, 0, 0));

        Button diceButton = new Button("Roll a dice");
        diceButton.setPrefSize(120, 30);
        diceButton.setFont(Font.font("Arial", 14));
        StackPane.setMargin(diceButton, new Insets(45, 0, 0, 0));

        Button restButton = new Button("Rest");
        restButton.setPrefSize(120, 30);
        restButton.setFont(Font.font("Arial", 14));

        if (playerViews[0].getPlayer().getBridgeCardNum() == 0) restButton.setDisable(true);
        StackPane.setMargin(restButton, new Insets(80, 0, 0, 0));

        diceButton.setOnAction(actionEvent -> {
            myItemClick.onRollDiceClick(diceButton, restButton);
        });

        restButton.setOnAction(actionEvent -> {
            myItemClick.onRestClick();
            changeTurn(restButton);
        });

        turnPane.getChildren().addAll(rectangle, turnLabel, diceButton, restButton);

        root.getChildren().add(turnPane);
    }

    // mapData의 각 줄의 정보에 따라 다음 타일의 위치를 결정하는 함수
    private void setNextXY(String t) {
        switch (t) {
            case "R":
                x++;
                break;
            case "D":
                y++;
                break;
            case "U":
                y--;
                break;
```

```
            case "L":
                x--;
                break;
        }
    }

    // 게임 턴을 다음 플레이어로 넘기는 함수
    public void changeTurn(Button restButton) {
        turn = (playerNum == turn) ? 1 : turn + 1;
        while (endOrders[turn - 1] != -1) {
            turn = (playerNum == turn) ? 1 : turn + 1;
        }
        turnLabel.setText("Player " + turn + " Turn");
        turnLabel.setTextFill(colors[turn - 1]);

        restButton.setDisable(playerViews[turn - 1].getPlayer().getBridgeCardNum() == 0);
    }

    // 카드 정보판을 카드별로 업데이트 하는 함수
    public void updatePlayerCardInfo(String type, int i) {
        int temp = playerViews[turn - 1].getPlayer().getTypeCardNum(type);
        playerCards[turn - 1][i].setText(type + " Card : " + temp + " cards");
    }

    // 플레이어가 입력한 문장이 유효한 이동인지 검사하는 함수
    public boolean checkMoveInfo(String moveStr) {
        int curX = getCurrentPlayer().getXPos();
        int curY = getCurrentPlayer().getYPos();
        for (int i = 0; i < moveStr.length(); i++) {
            Character temp = moveStr.charAt(i);
            switch (temp) {
                case 'U':
                    curY--;
                    break;
                case 'D':
                    curY++;
                    break;
                case 'R':
                    curX++;
                    break;
                case 'L':
                    curX--;
                    break;
                default:
                    return false;
            }
            if (curX > WIDTH || curY > HEIGHT)
                return false;

            if (tiles[curX][curY] == null)
                return false;

            if (tiles[curX][curY].getType() == "END")
                return true;

            if (endCount >= 1) {
                if (tiles[curX][curY].isBackMove(temp.toString())) {
                    return false;
                }
            }

        }
        return true;
    }

    // 스레드를 만들어 단위 초
    public void move(String moveStr, Button exitButton, Label rollLabel) {
        Thread thread = new Thread() {
            private int curX = getCurrentPlayer().getXPos();
            private int curY = getCurrentPlayer().getYPos();

            public void run() {
                try {
```

```
                for (int i = 0; i < moveStr.length(); i++) {
                    Character temp = moveStr.charAt(i);
                    switch (temp) {
                        case 'U':
                            curY--;
                            break;
                        case 'D':
                            curY++;
                            break;
                        case 'R':
                            curX++;
                            break;
                        case 'L':
                            curX--;
                            break;
                    }
                    Thread.sleep(500);
                    switch (tiles[curX][curY].getType()) {
                        // 올라간 타일의 종류에 따라 점수 획득 및 카드 획득
                        case "BRIDGE":
                            playerViews[turn - 1].setIsOnBridge(true, temp);
                            break;
                        case "SAW":
                            getCurrentPlayer().gainSawCard();
                            tiles[curX][curY].removeCard();
                            Platform.runLater(new Runnable() {
                                @Override
                                public void run() {
                                    updatePlayerCardInfo("Saw", 2);
                                }
                            });
                            break;
                        case "HAMMER":
                            getCurrentPlayer().gainHammerCard();
                            tiles[curX][curY].removeCard();
                            Platform.runLater(new Runnable() {
                                @Override
                                public void run() {
                                    updatePlayerCardInfo("Hammer", 3);
                                }
                            });
                            break;
                        case "PHILIPS":
                            getCurrentPlayer().gainPhilipsCard();
                            tiles[curX][curY].removeCard();
                            Platform.runLater(new Runnable() {
                                @Override
                                public void run() {
                                    updatePlayerCardInfo("Philips", 4);
                                }
                            });
                            break;
                        case "END":
                            endCount++;
                            endOrders[turn - 1] = endCount;
                            getCurrentPlayer().gainRankScore(endCount);

                            playerViews[turn - 1].relocate(curX * TILE_SIZE + 8, curY *
TILE_SIZE + 8);
                            Platform.runLater(new Runnable() {
                                @Override
                                public void run() {
                                    exitButton.setDisable(false);
                                    if (endCount == 1) {
                                        rollLabel.setText("Player " + turn + " finish!!\nCannot
move backwards\nfrom now on.");
                                    } else rollLabel.setText("Player " + turn + " finish!!\n");
                                }
                            });

                            // 게임 종료조건
                            if (endCount == playerNum - 1) {
                                Platform.runLater(new Runnable() {
                                    @Override
```

```
                                public void run() {
                                    endGame();
                                }
                            });
                            this.interrupt();
                        }
                        return;

                    default:
                        if (playerViews[turn - 1].getIsOnBridge()) {
                            if (playerViews[turn - 1].checkBridgeCrossed(temp)) {
                                getCurrentPlayer().gainBridgeCard();
                                Platform.runLater(new Runnable() {
                                    @Override
                                    public void run() {
                                        updatePlayerCardInfo("Bridge", 1);
                                    }
                                });
                            }
                            playerViews[turn - 1].setIsOnBridge(false, null);
                        }
                    }
                    playerViews[turn - 1].relocate(curX * TILE_SIZE + 8, curY * TILE_SIZE +
8);
                    getCurrentPlayer().setPos(curX, curY);
                }
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        exitButton.setDisable(false);
                        rollLabel.setText("Turn Finished!! \nClick Exit Button.");
                    }
                });
                this.interrupt();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    };
    thread.start();
}

// 게임을 종료하는 함수(남은 플레이어가 한명일 때)
private void endGame() {
    turnPane.getChildren().remove(2);
    turnPane.getChildren().remove(2);

    turnLabel.setText("End Game");
    turnLabel.setTextFill(Color.BLACK);
    turnPane.setAlignment(Pos.TOP_CENTER);
    StackPane.setMargin(turnLabel, new Insets(10, 0, 0, 0));

    Button exitButton = new Button("Exit");
    exitButton.setPrefSize(120, 30);
    exitButton.setFont(Font.font("Arial", 14));
    StackPane.setMargin(exitButton, new Insets(45, 0, 0, 0));
    turnPane.getChildren().add(exitButton);
    exitButton.setOnAction(actionEvent -> {
        myItemClick.onExitClick();
    });

    StackPane resPane = new StackPane();
    resPane.setPrefSize(350, 230);
    resPane.relocate(820, 200);
    resPane.setAlignment(Pos.TOP_CENTER);
    root.getChildren().add(resPane);

    Rectangle rectangle = new Rectangle(350, 230);
    rectangle.setFill(Color.WHITE);
    rectangle.setStroke(Color.BLACK);
    rectangle.setStrokeWidth(2);
    resPane.getChildren().add(rectangle);

    // 점수 결과 판 만드는 함수
```

```
        Label scoreLabel = new Label("Score");
        scoreLabel.setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR, 18));
        StackPane.setAlignment(scoreLabel, Pos.TOP_CENTER);
        StackPane.setMargin(scoreLabel, new Insets(10, 0, 0, 0));
        resPane.getChildren().add(scoreLabel);

        // 각 플레이어의 점수를 알려준다
        Label[] playerScores = new Label[playerNum];
        int yMargin = 0;
        for (int i = 0; i < playerNum; i++) {
            if (playerNum == 3 && endOrders[i] == -1)
playerViews[i].getPlayer().gainRankScore(3);
            else if (playerNum == 2 && endOrders[i] == -1)
playerViews[i].getPlayer().gainRankScore(2);
            playerScores[i] = new Label("Player " + (i + 1) + " : " +
playerViews[i].getPlayer().getScore() + " points");
            playerScores[i].setFont(Font.font("Arial", FontWeight.BOLD, FontPosture.REGULAR,
17));
            playerScores[i].setTextFill(colors[i]);
            StackPane.setAlignment(playerScores[i], Pos.CENTER_LEFT);
            StackPane.setMargin(playerScores[i], new Insets(yMargin, 0, 20, 30));
            resPane.getChildren().add(playerScores[i]);
            yMargin += 60;
        }

        // 승자 알려줌
        int winnerId = getFirstRankPlayerId();
        Label resLabel = new Label("Winner is Player " + winnerId + " !!");
        resLabel.setTextFill(Color.FIREBRICK);
        resLabel.setFont(Font.font("Verdana", FontWeight.BOLD, FontPosture.REGULAR, 24));
        StackPane.setAlignment(resLabel, Pos.TOP_CENTER);
        StackPane.setMargin(resLabel, new Insets(45, 0, 0, 0));

        resPane.getChildren().add(resLabel);
    }

    // 게임 승자의 ID를 알려주는 함수
    private int getFirstRankPlayerId() {
        int max = -1, idx = 0;
        int temp;
        for (int i = 0; i < playerNum; i++) {
            temp = playerViews[i].getPlayer().getScore();
            if (max < temp) {
                max = temp;
                idx = i;
            } else if (max == temp) {
                if (endOrders[idx] > i)
                    idx = i;
            }
        }
        return (idx + 1);
    }


}
```

- Tile.java

: 맵을 구성하는 타일 클래스

```
public class Tile extends StackPane {
    // 타일의 타입
    private String type;
    private Rectangle rectangle = new Rectangle();
    private String backMove;
    private String frontMove;

    public Tile(String type, int x, int y, String backMove, String frontMove){
```

```
        this.type = type;
        this.backMove = backMove;
        this.frontMove = frontMove;

        // 타일의 종류 별로 다르게 보여지도록 생성
        if (type.equals("START") || type.equals("END")){
            rectangle.setWidth(BridgeMapView.TILE_SIZE * 2);
            rectangle.setHeight(BridgeMapView.TILE_SIZE * 2);
            Text text = new Text(type);
            Font font = Font.font("Verdana", FontWeight.EXTRA_BOLD, 16);
            text.setFont(font);
            relocate(x * BridgeMapView.TILE_SIZE, y * BridgeMapView.TILE_SIZE);
            rectangle.setFill(Color.GOLD);
            rectangle.setStroke(Color.BLACK);
            rectangle.setStrokeWidth(3);
            getChildren().addAll(rectangle, text);
        }
        else if (type.equals("BRIDGE")){
            relocate(x * BridgeMapView.TILE_SIZE, y * BridgeMapView.TILE_SIZE);
            File file = new File("src/main/resources/com/lee/image/bridge.png");
            ImageView image = new ImageView(new Image(file.toURI().toString()));
            image.setFitWidth(BridgeMapView.TILE_SIZE);
            image.setFitHeight(BridgeMapView.TILE_SIZE);
            getChildren().addAll(image);
        }
        else {
            rectangle.setWidth(BridgeMapView.TILE_SIZE);
            rectangle.setHeight(BridgeMapView.TILE_SIZE);
            relocate(x * BridgeMapView.TILE_SIZE, y * BridgeMapView.TILE_SIZE);
            rectangle.setFill(Color.valueOf("feb"));
            rectangle.setStroke(Color.BLACK);
            rectangle.setStrokeWidth(3);
            if (type.equals("SAW")){
                File file = new File("src/main/resources/com/lee/image/saw.png");
                ImageView image = new ImageView(new Image(file.toURI().toString()));
                image.setFitWidth(30);
                image.setFitHeight(30);
                getChildren().addAll(rectangle, image);
            } else if (type.equals("PHILIPS")){
                File file = new File("src/main/resources/com/lee/image/philipsDriver.png");
                ImageView image = new ImageView(new Image(file.toURI().toString()));
                image.setFitWidth(30);
                image.setFitHeight(30);
                getChildren().addAll(rectangle, image);
            } else if (type.equals("HAMMER")){
                File file = new File("src/main/resources/com/lee/image/hammer.png");
                ImageView image = new ImageView(new Image(file.toURI().toString()));
                image.setFitWidth(30);
                image.setFitHeight(30);
                getChildren().addAll(rectangle, image);
            } else if (type.equals("INTERSECT")){
                File file = new File("src/main/resources/com/lee/image/intersect.png");
                ImageView image = new ImageView(new Image(file.toURI().toString()));
                image.setFitWidth(25);
                image.setFitHeight(25);
                getChildren().addAll(rectangle, image);
            }

            else getChildren().addAll(rectangle);
        }
    }

    public String getType() {return type;}

    // 카드를 최초로 획득한 이후 타일에서 카드가 사라지게 함(가정에서 설정함)
    public void removeCard(){
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                getChildren().remove(1);
            }
        });
        type = "CELL";
    }
```

```
    // 다음 타일의 frontMove가 현재 움직일려는 move의 반대면 뒤로 가는 것이다.
    public boolean isBackMove(String input){
        if(backMove == null) return true;
        switch(frontMove){
            case "R":
                if (input.equals("L")) return true;
                break;
            case "L":
                if (input.equals("R")) return true;
                break;
            case "U":
                if (input.equals("D")) return true;
                break;
            case "D":
                if (input.equals("U")) return true;
                break;
        }
        return false;
    }
}
```

- PlayerView.java

: 각 플레이어의 말에 대한 클래스

```
public class PlayerView extends StackPane {
    // 플레이어
    private Player player;
    private Circle circle;
    private Text text;

    // 다리를 지나가는지 판단용
    private Boolean isOnBridge = false;
    private Character moveType;

    PlayerView(int num, int xPos, int yPos, Color color){
        this.player = new Player(xPos, yPos);
        circle = new Circle();
        circle.setRadius(12);
        circle.setFill(color);
        text = new Text(Integer.toString(num));
        text.setFont(Font.font("Arial", FontWeight.EXTRA_BOLD, 18));
        text.setFill(Color.WHITE);
        getChildren().addAll(circle, text);
    }

    // getter
    public Player getPlayer() {
        return player;
    }
    public Boolean getIsOnBridge() { return isOnBridge; }

    public void setIsOnBridge(Boolean isOnBridge, Character moveType){
        this.isOnBridge = isOnBridge;
        this.moveType = moveType;
    }

    // 다리를 지나갔는지 체크하는 함수
    public boolean checkBridgeCrossed(Character nowMove){
        if (isOnBridge && this.moveType == nowMove) return true;
        return false;
    }
}
```

2) Model 패키지

- BridgeMap.java

: BridgeMapView에서 보여질 맵 화면에 대한 데이터를 담고 있는 클래스. getMapWidth(), getMapHeight(), getStartHeightGap(), getStartWidthGap() 메서드들을 통해 맵의 사이즈를 동적으로 설정할 수 있게 함. Singleton 기법을 적용하여 BridgeMap 객체가 단 하나만 생성되도록 하여 여러 맵 객체가 생성되지 않도록 함.

```
public class BridgeMap {
    // Singleton 기법
    public static BridgeMap m = null;
    private boolean isMapSet = false;
    private String[] mapData;

    // BridgeMap 객체가 하나만 생성되게 함
    public static BridgeMap getInstance() {
        if (m == null)
            m = new BridgeMap();
        return m;
    }

    // 초기에 파일을 설정 안한 경우 default.map 으로 맵 데이터를 설정함
    public void defaultMap() {
        File file = new File("src/main/resources/com/lee/mapdata/default.map");
        if (file.exists()) {
            fileToData(file);
        } else {
            mapData = new String[]{
                    "S R", "C L R", "C L D", "B U D", "S U D", "C U D", "C U D", "C U R", "C L
R", "H L U", "B D U", "C D U", "C D U",
                    "C D U", "b D U", "P D U", "C D R", "C L R", "H L D", "C U D", "C U D", "C U
D", "B U D", "C U D", "b U D", "C U D",
                    "S U D", "P U D", "C U D", "C U D", "C U R", "C L R", "C L U", "H D U", "C D
U", "B D U", "C D U", "C D U", "C D U",
                    "C D U", "b D U", "C D R", "C L R", "H L D", "C U D", "C U D", "C U D", "P U
D", "B U D", "b U D", "C U D", "H U D",
                    "C U R", "C L R", "S L U", "P D U", "C D U", "C D U", "b D U", "C D U", "C D
U", "E"
            };
        }
        writeCurMap();
    }

    // getter
    public String[] getMapData() {
        if (isMapSet == false) {
            if (isPrevMapExists()){
                File file = new File("src/main/resources/com/lee/mapdata/prev.map");
                fileToData(file);
            }
            else defaultMap();
            isMapSet = true;
        }
        return mapData;
    }

    // setter
    public void setMapData(String[] mapData) {
        this.mapData = mapData;
        writeCurMap();
        isMapSet = true;
    }

    // 파일에서 데이터 읽어오기
    private void fileToData(File file){
```

```
        ArrayList<String> mapData = new ArrayList<>();
        BufferedReader br;
        try {
            br = new BufferedReader(new InputStreamReader(new FileInputStream(file)));
            String line;
            while ((line = br.readLine()) != null) {
                mapData.add(line);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        this.mapData = mapData.toArray(new String[mapData.size()]);
    }

    // 현재 플레이 중인 맵을 작성하는 함수, 다음에 게임을 실행할 때 prev.map 을 이용해서 사용함
    private void writeCurMap(){
        try {
            File file = new File("src/main/resources/com/lee/mapdata/prev.map");
            if (!file.exists())
                file.createNewFile();
            else {
                file.delete();
                file.createNewFile();
            }
            BufferedWriter bw = new BufferedWriter(new FileWriter(file));
            for (int i=0; i<mapData.length; i++)
                bw.append(mapData[i]+"\n");
            bw.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    private boolean isPrevMapExists(){
        return new File("src/main/resources/com/lee/mapdata/prev.map").exists();
    }

    // 맵의 가로 길이
    public int getMapWidth() {
        int leftCount = 0;
        int rightCount = 0;
        int heightMax = 0;
        for (int i = 0; i < mapData.length; i++) {
            if (mapData[i].endsWith("L")) {
                leftCount++;
                if (rightCount > 0) rightCount--;
                if (leftCount > heightMax)
                    heightMax = leftCount;
            } else if (mapData[i].endsWith("R")) {
                rightCount++;
                if (leftCount > 0) leftCount--;
                if (rightCount > heightMax)
                    heightMax = rightCount;
            }
        }
        return heightMax + 1;
    }

    // 맵의 세로 길이
    public int getMapHeight() {
        int upCount = 0;
        int downCount = 0;
        int heightMax = 0;
        for (int i = 0; i < mapData.length; i++) {
            if (mapData[i].endsWith("U")) {
                downCount++;
                if (upCount > 0) upCount--;
                if (downCount > heightMax)
                    heightMax = downCount;
            } else if (mapData[i].endsWith("D")) {
                upCount++;
```

```
            if (downCount > 0) downCount--;
            if (upCount > heightMax)
                heightMax = upCount;
        }
    }
    return heightMax + 1;
}

// 시작 포인트와 게임에서 가장 높은 타일의 높이 차이
public int getStartHeightGap() {
    int upCount = 0;
    int downCount = 0;
    int gapMax = 0;
    for (int i = 0; i < mapData.length; i++) {
        if (mapData[i].endsWith("U"))
            upCount++;
        else if (mapData[i].endsWith("D"))
            downCount++;
        else {
            if (downCount > 0 && upCount > 0) {
                if (upCount - downCount > gapMax) {
                    gapMax = upCount - downCount;
                    upCount = 0;
                    downCount = 0;
                }
            }
        }
    }
    return gapMax > 0 ? gapMax : 0;
}

// 시작 포인트와 게임에서 가장 멀리있는 타일의 폭 차이
public int getStartWidthGap() {
    int leftCount = 0;
    int rightCount = 0;
    int gapMax = 0;
    for (int i = 0; i < mapData.length; i++) {
        if (mapData[i].endsWith("L"))
            leftCount++;
        else if (mapData[i].endsWith("R"))
            rightCount++;
        else {
            if (leftCount > 0 && rightCount > 0) {
                if (leftCount - rightCount > gapMax) {
                    gapMax = leftCount - rightCount;
                    leftCount = 0;
                    rightCount = 0;
                }
            }
        }
    }
    return gapMax > 0 ? gapMax : 0;
}

}
```

- Player.class

: 플레이어의 데이터를 담고 있는 클래스. 플레이어가 얻은 각종 카드 수, 점수, 위치에 대한 정보를 담고 있음.

```
public class Player{
    // 플레이어 정보들
    private int bridgeCardNum = 0;
    private int sawCardNum = 0;
    private int hammerCardNum = 0;
    private int philipsCardNum = 0;
    private int score = 0;
```

```
    private int xPos = 0;
    private int yPos = 0;


    public Player(int xPos, int yPos){
        this.xPos = xPos;
        this.yPos = yPos;
    }

    public void setPos(int xPos, int yPos){
        this.xPos = xPos;
        this.yPos = yPos;
    }

    // rest
    public void rest(){
        bridgeCardNum--;
    }

    // getter 들
    public int getBridgeCardNum() {return bridgeCardNum;}

    public int getSawCardNum() {return sawCardNum;}

    public int getHammerCardNum() {return hammerCardNum;}

    public int getPhilipsCardNum() {return philipsCardNum;}

    public int getTypeCardNum(String cardType){
        if (cardType.equals("Bridge"))
            return getBridgeCardNum();
        else if (cardType.equals("Saw"))
            return getSawCardNum();
        else if (cardType.equals("Hammer"))
            return getHammerCardNum();
        else return getPhilipsCardNum();
    }

    public int getXPos() {return xPos;}

    public int getYPos() {return yPos;}

    public int getScore() {return score;}

    // 카드를 얻은 경우
    public void gainBridgeCard() {
        bridgeCardNum++;
    }

    public void gainSawCard() {
        sawCardNum++;
        score += 3;
    }

    public void gainHammerCard() {
        hammerCardNum++;
        score += 2;
    }

    public void gainPhilipsCard() {
        philipsCardNum++;
        score += 1;
    }

    // 종료 지점에 들어온 순위에 대한 점수를 얻는 함수
    public void gainRankScore(int rank){
        switch (rank){
            case 1:
                score += 7;
                break;
            case 2:
                score += 3;
                break;
```

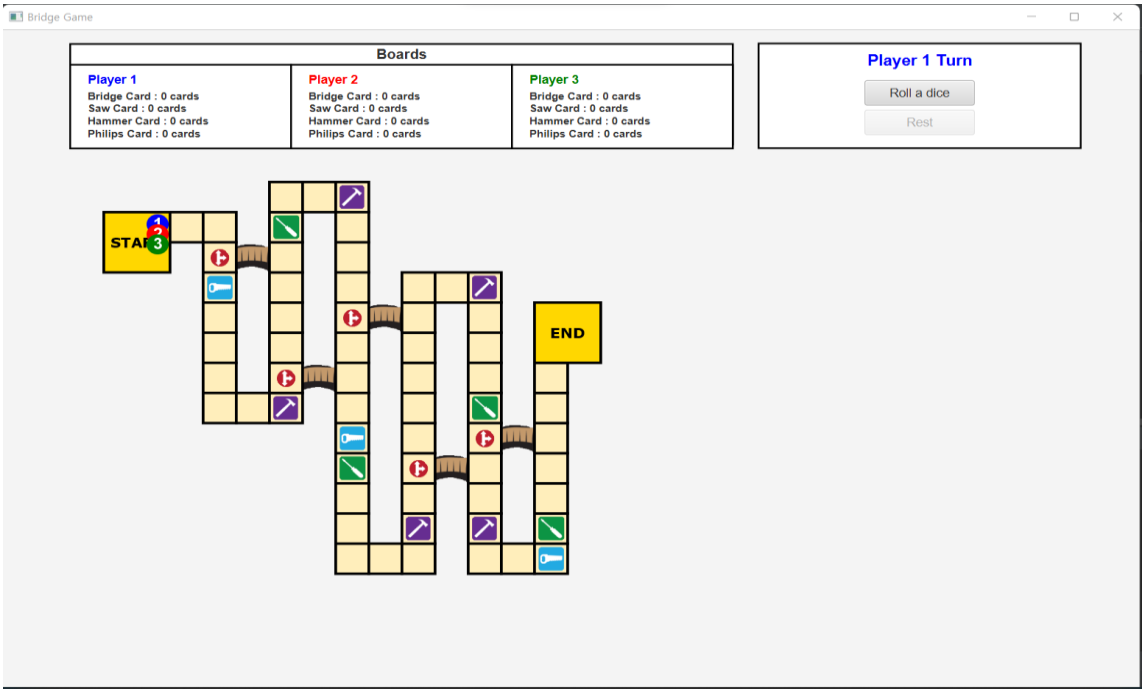
```
            case 3:
                score += 1;
                break;
        }
    }
}
```

(3) 프로그램 사용 방법 및 결과

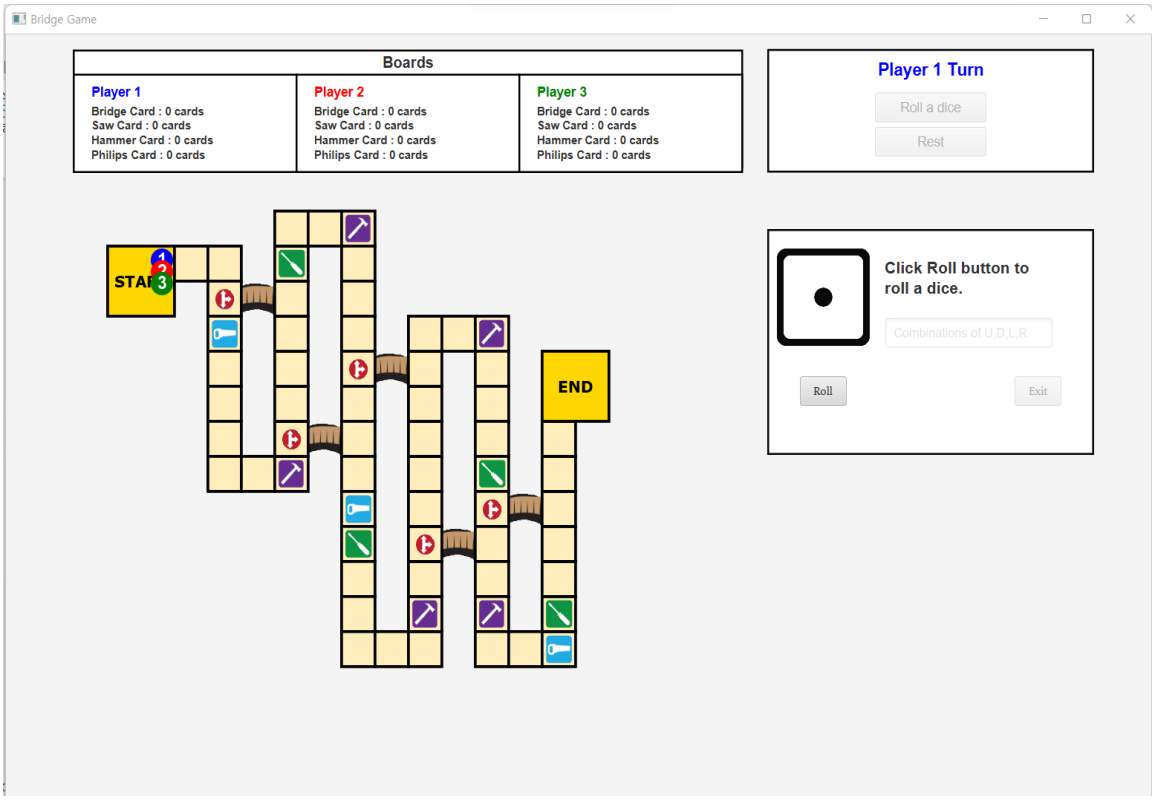
- 시작 화면

플레이어 수를 누르고 엔터를 하거나 입력 확인 버튼을 누르면 게임이 시작된다. 맵 설정 버튼을 눌러 맵을 임의로 지정할 수 있다.

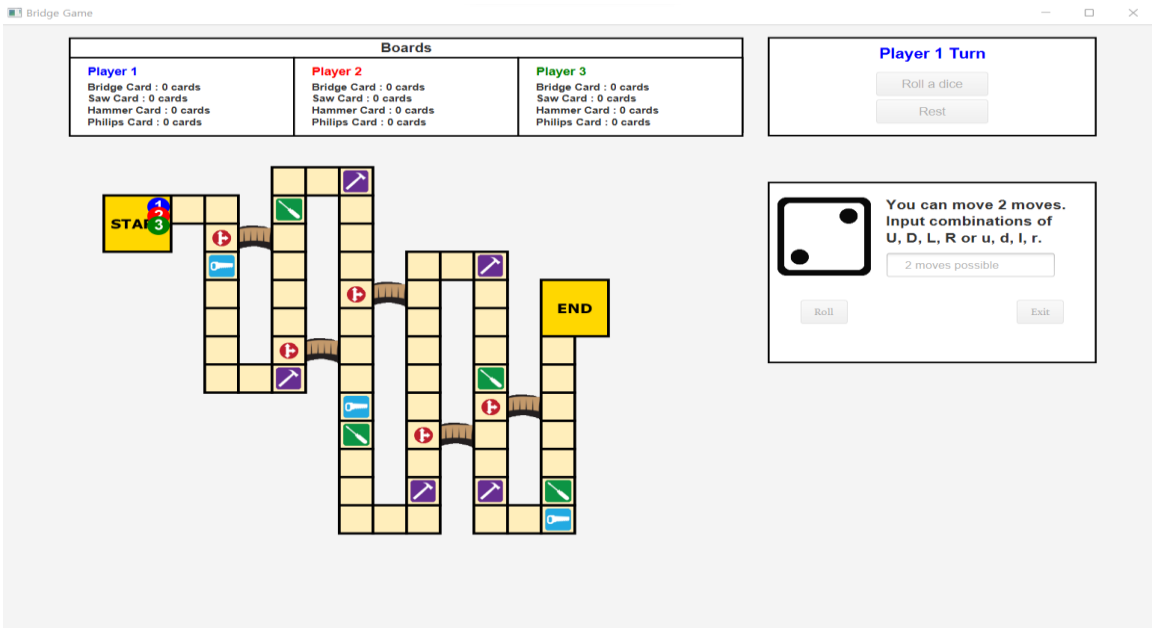
- 게임 화면



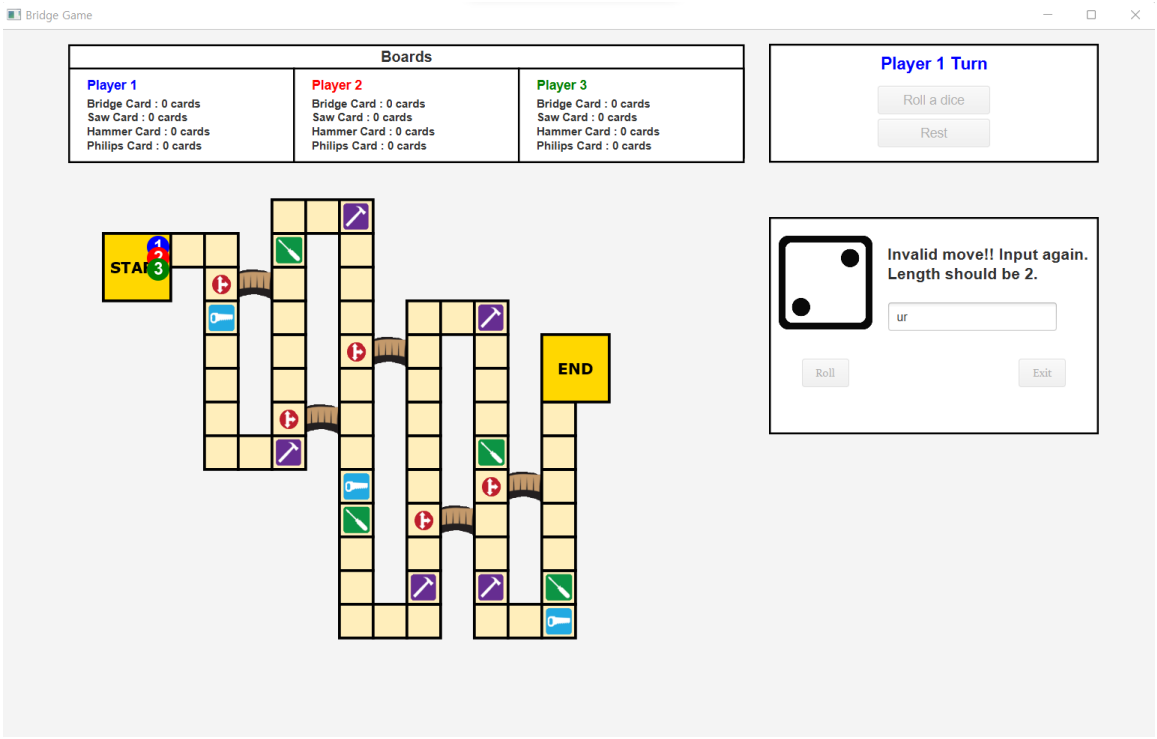
플레이어는 자신의 턴에 Roll a dice 버튼 혹은 Rest 버튼을 눌러 움직일지 쉴지 정할 수 있다. 게임을 시작한 바로 다음엔 획득한 다리 카드가 없기 때문에 쉴 수 없으므로 Rest 버튼이 비활성화 되어있으며 다리 카드수가 한 장 이상이면 버튼이 활성화된다.



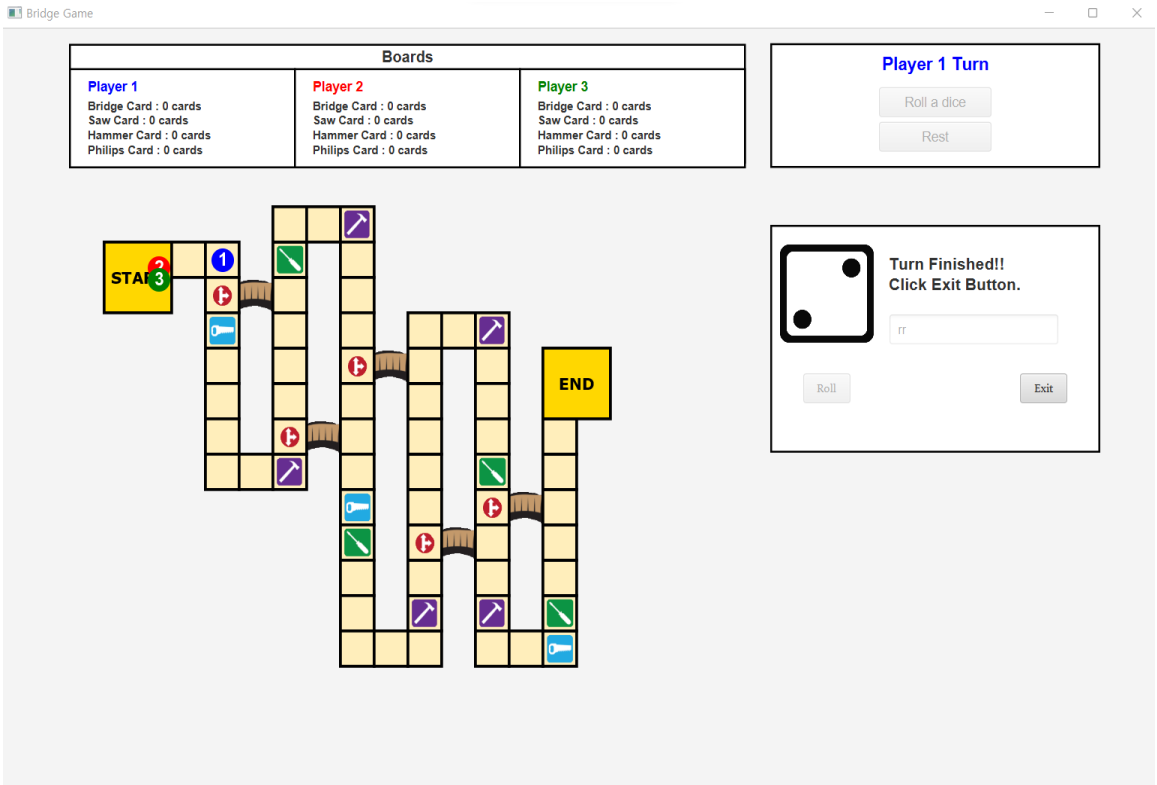
Roll a dice 버튼을 누르면 다음과 같이 주사위 화면이 생기며 Roll 버튼을 누르면 시스템에서 주사위를 굴리며 결과값을 보여주며 얼만큼 이동이 가능한지 메시지를 띄운다.



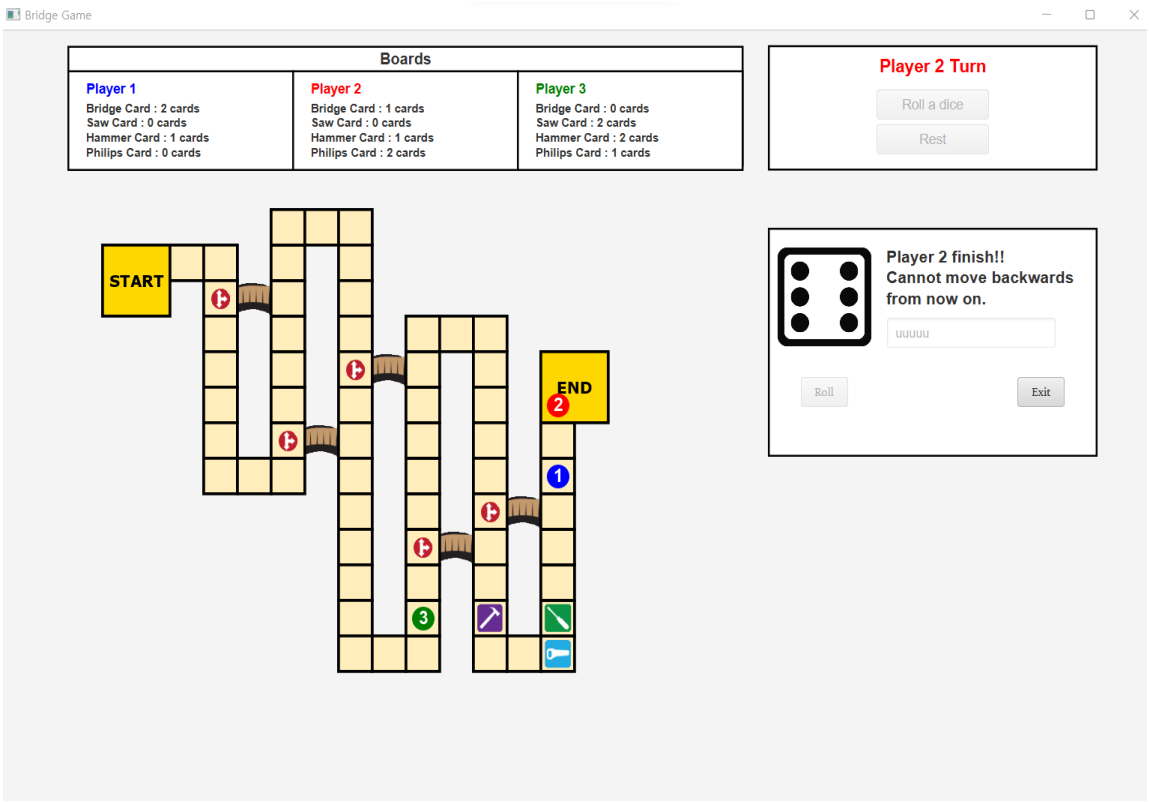
위의 화면에서 텍스트 필드에 2 moves possible이라며 표시되었다. 플레이어는 U, D, L, R 혹은 u, d, l, r의 조합을 한 줄로 입력한다.



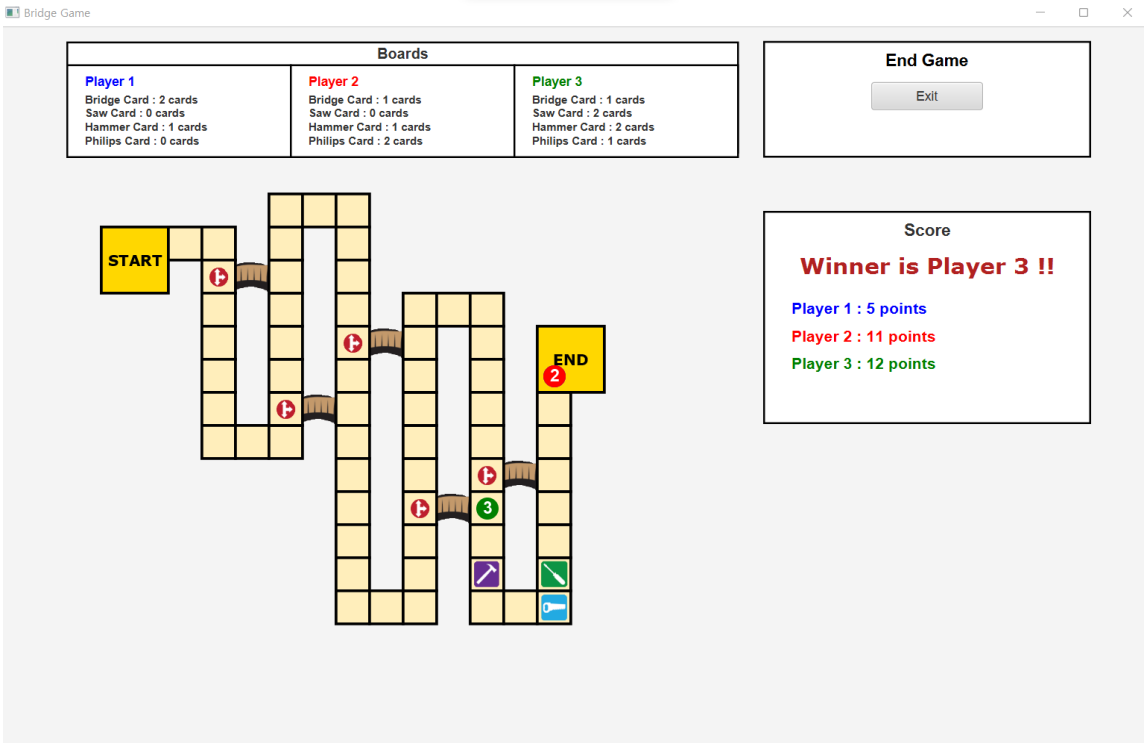
유효하지 않은 인풋값을 입력한 경우 다시 입력하도록 메시지를 띄운다.



플레이어가 움직이고 난 후에는 Exit 버튼을 눌러 자신의 턴을 종료한다.



2번 플레이어가 게임이 끝나 뒤로 이동하지 못한다는 메시지를 띄운다.



남은 플레이어가 한 명인 경우 게임을 종료하며 승자와 각 플레이어의 점수를 보여준다. 플레이어 1,2에 비해 등수는 낮지만 카드를 많이 획득해 플레이어 3이 우승한 상황이다. Exit 버튼을 누르면 게임 창이 닫아진다.