중앙대학교
CHUNG ANG UNIVERSITY

소프트웨어공학

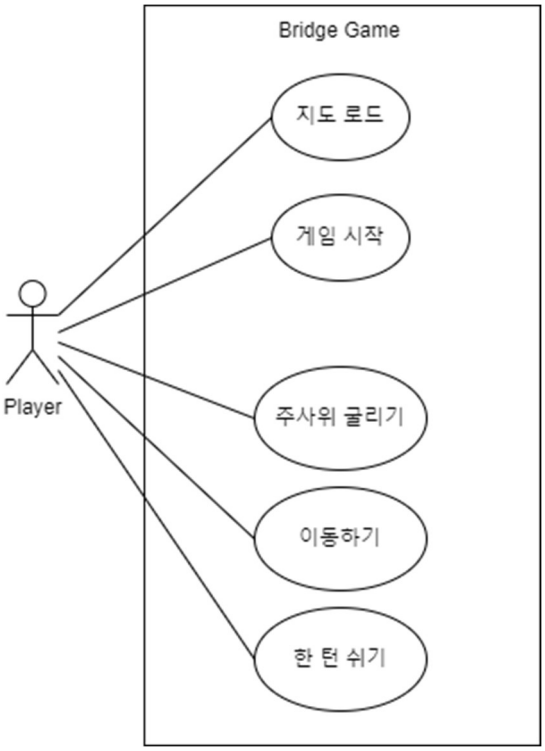
Term Project

과목 / 분반 :	소프트웨어공학 01분반
담당교수 :	이찬근
제출일자 :	2022. 06. 10
학번 :	20184757
이름 :	주영석

목차

[1] 요구 정의 및 분석 산출물

(1) Use Case Diagram



(2) Use Case Description

Use Case UC1: 지도 로드

Primary Actor: Player

Precondition: 지도 파일이 저장되어 있다.

Main Success Scenario

1. 지도 파일의 이름을 입력하라는 메세지가 출력된다.
2. 사용자가 지도 파일의 이름을 입력한다.
3. 지도 파일이 로드된다.

Extensions

2a. 입력된 이름의 파일이 존재하지 않는 경우

    1. 지도 파일의 이름을 입력하라는 메시지가 다시 출력된다.

2b. 사용자가 아무것도 입력하지 않은 경우

    1. 디폴트로 지정된 지도 파일이 로드된다.

Use Case UC2: 주사위 굴리기

Primary Actor: Player

Main Success Scenario

1. 사용자가 주사위 굴림 버튼을 누른다.
2. 주사위의 값이 출력된다.

Extensions

1a. 주사위가 이미 굴려진 경우

    1. 주사위가 이미 굴려졌다는 메시지를 출력한다.

Use Case UC3: 이동하기

Primary Actor: Player

Preconditions: 주사위가 이미 굴려져 있다.

Main Success Scenario

1. 사용자가 움직일 방향의 목록을 U, D, L, R 또는 u, d, l, r의 조합으로 입력한다.
2. 사용자가 움직임 버튼을 누른다.
3. 입력에 해당하는 만큼 사용자의 말을 이동한다.
4. 이동된 칸에 카드가 존재하면 사용자에게 해당 카드를 지급한다.
5. 다음 차례로 넘어간다.

Extensions

2a. 사용자의 입력 값이 사용자가 움직일 수 있는 칸 수와 다른 경우

    1. 사용자의 입력 값이 올바르지 않다는 메시지를 출력한다.

2b. 사용자의 입력 값이 사용자가 움직일 수 있는 방향과 일치하지 않는 경우

    1. 사용자의 입력 값이 올바르지 않다는 메시지를 출력한다.

3c. 사용자가 다리를 건너는 경우

    1. 사용자에게 다리카드를 지급한다.

Use Case UC4: 한 턴 쉬기

Primary Actor: Player

Main Success Scenario

1. 사용자가 한 턴 쉬기 버튼을 누른다.
2. 사용자가 다리 카드를 가지고 있는 경우 다리 카드 한 장을 제거한다.
3. 다음 차례로 넘어간다.

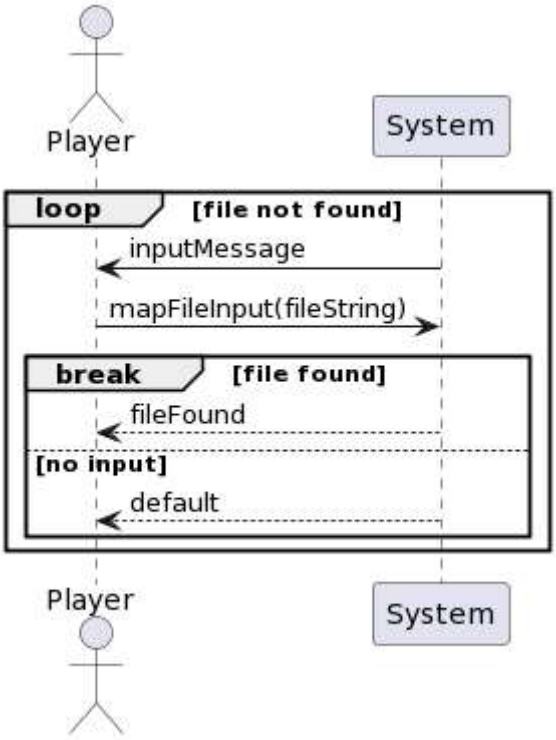
Extensions

1a. 주사위가 이미 굴려진 경우

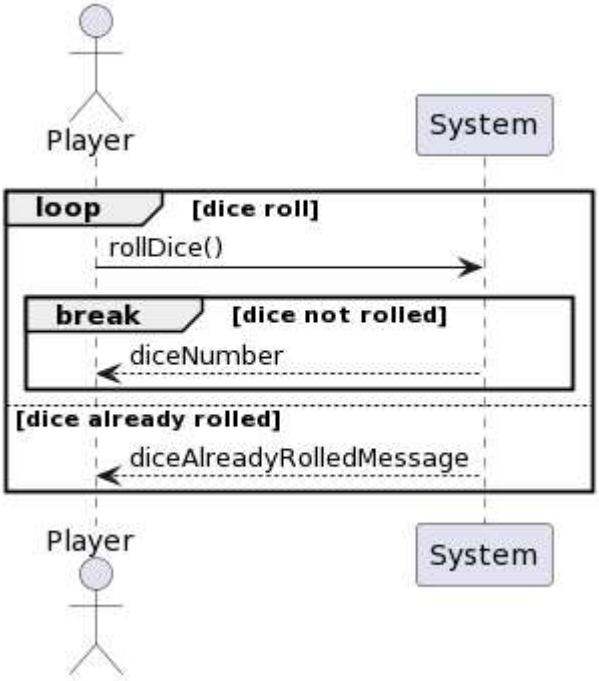
    1. 주사위가 이미 굴려져서 쉴 수 없다는 메시지를 출력한다.

(3) System Sequence Diagram

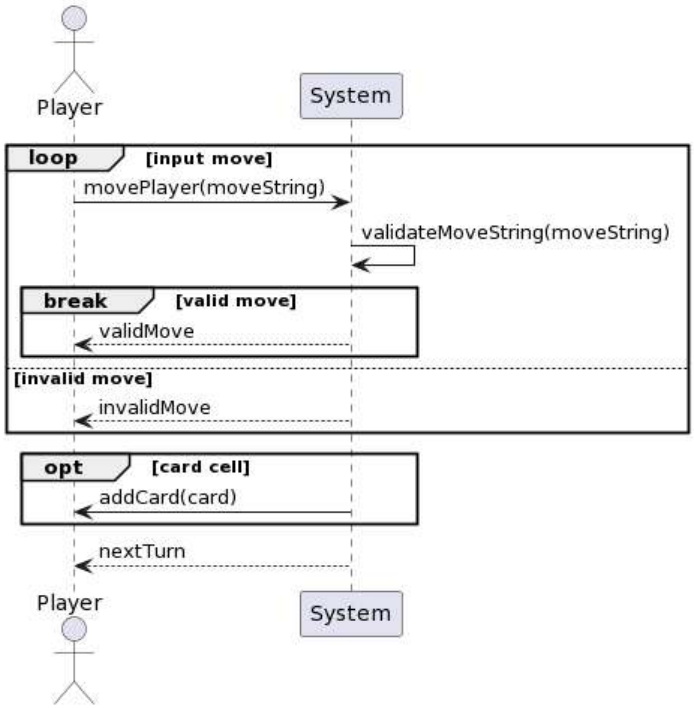
1. UC1



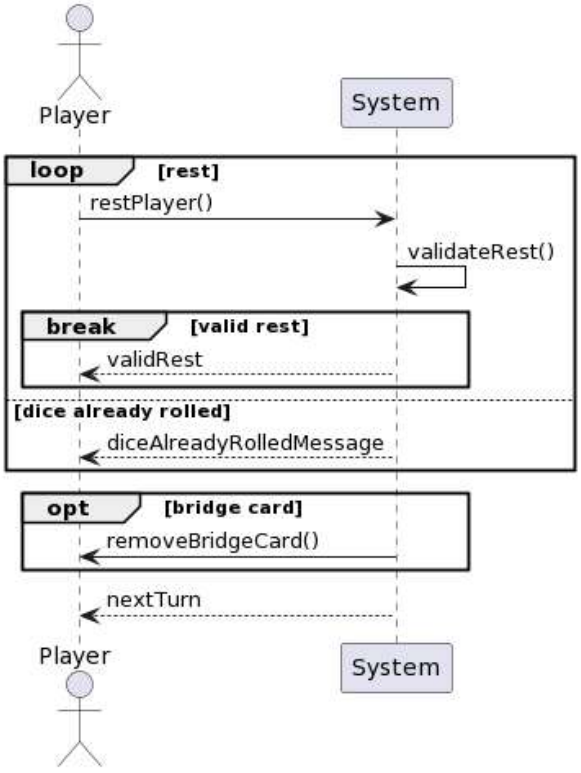
2. UC2



3. UC3


Player
System
loop
[input move]
movePlayer(moveString)
validateMoveString(moveString)
break
[valid move]
validMove
[invalid move]
invalidMove
opt
[card cell]
addCard(card)
nextTurn
Player
System

4. UC4


Player
System
loop
[rest]
restPlayer()
validateRest()
break
[valid rest]
validRest
[dice already rolled]
diceAlreadyRolledMessage
opt
[bridge card]
removeBridgeCard()
nextTurn
Player
System

(4) Operation Contract

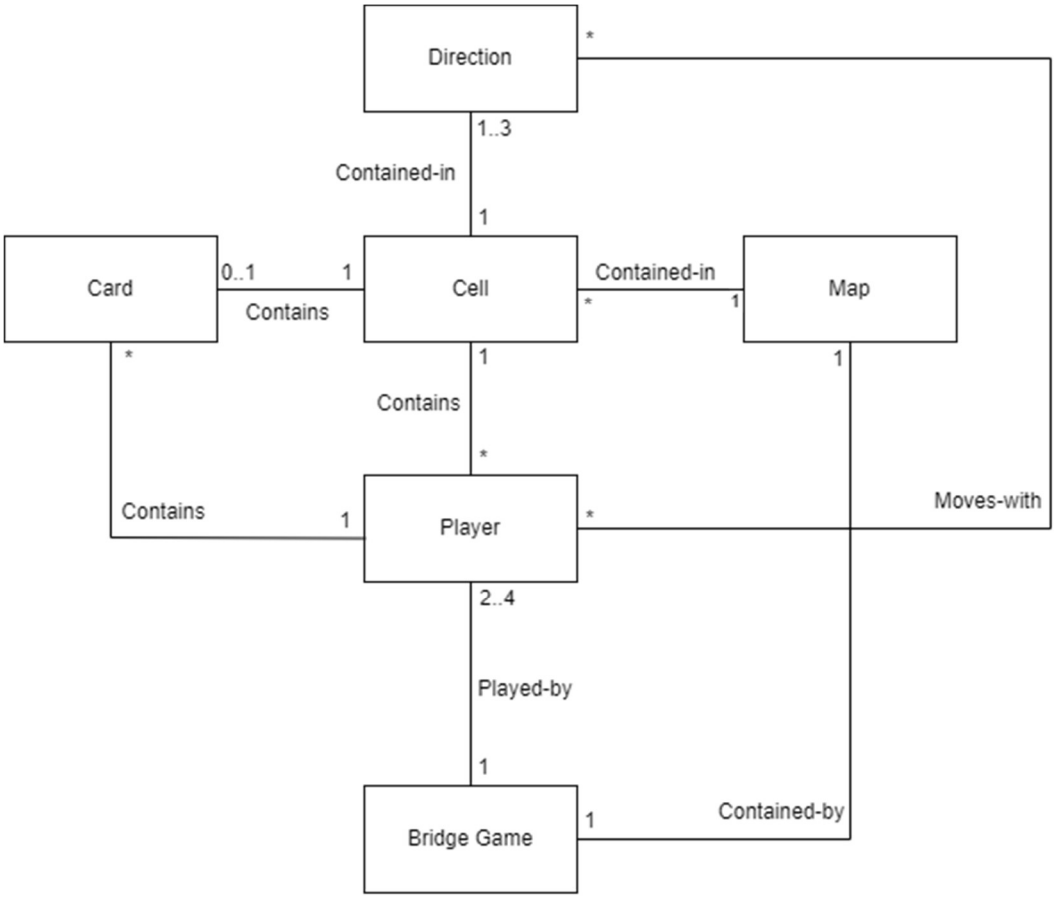
Operation	mapFileInput(fileString: String)
Cross References	Use Case UC1: 지도 로드
Preconditions	- 지도 파일이 저장되어 있음
Postconditions	- 해당 지도 파일을 로드했음 - 지도 클래스가 생성됨

Operation	rollDice()
Cross References	Use Case UC2: 주사위 굴리기
Preconditions	없음
Postconditions	- 주사위가 굴려짐 - 해당 주사위 숫자가 표시됨

Operation	movePlayer(moveString: String)
Cross References	Use Case UC3: 이동하기
Preconditions	주사위가 굴려져있음
Postconditions	- 플레이어의 위치가 이동됨 - 이동된 플레이어의 위치가 표시됨 - 플레이어 위치의 카드를 사용자가 가짐 - 다음 플레이어의 차례가 됨

Operation	restPlayer()
Cross References	Use Case UC4: 한 턴 쉬기
Preconditions	주사위가 굴려져 있지 않음
Postconditions	- 플레이어의 카드 중 다리 카드 하나가 제거됨 - 수정된 플레이어의 카드 목록이 표시됨 - 다음 플레이어의 차례가 됨

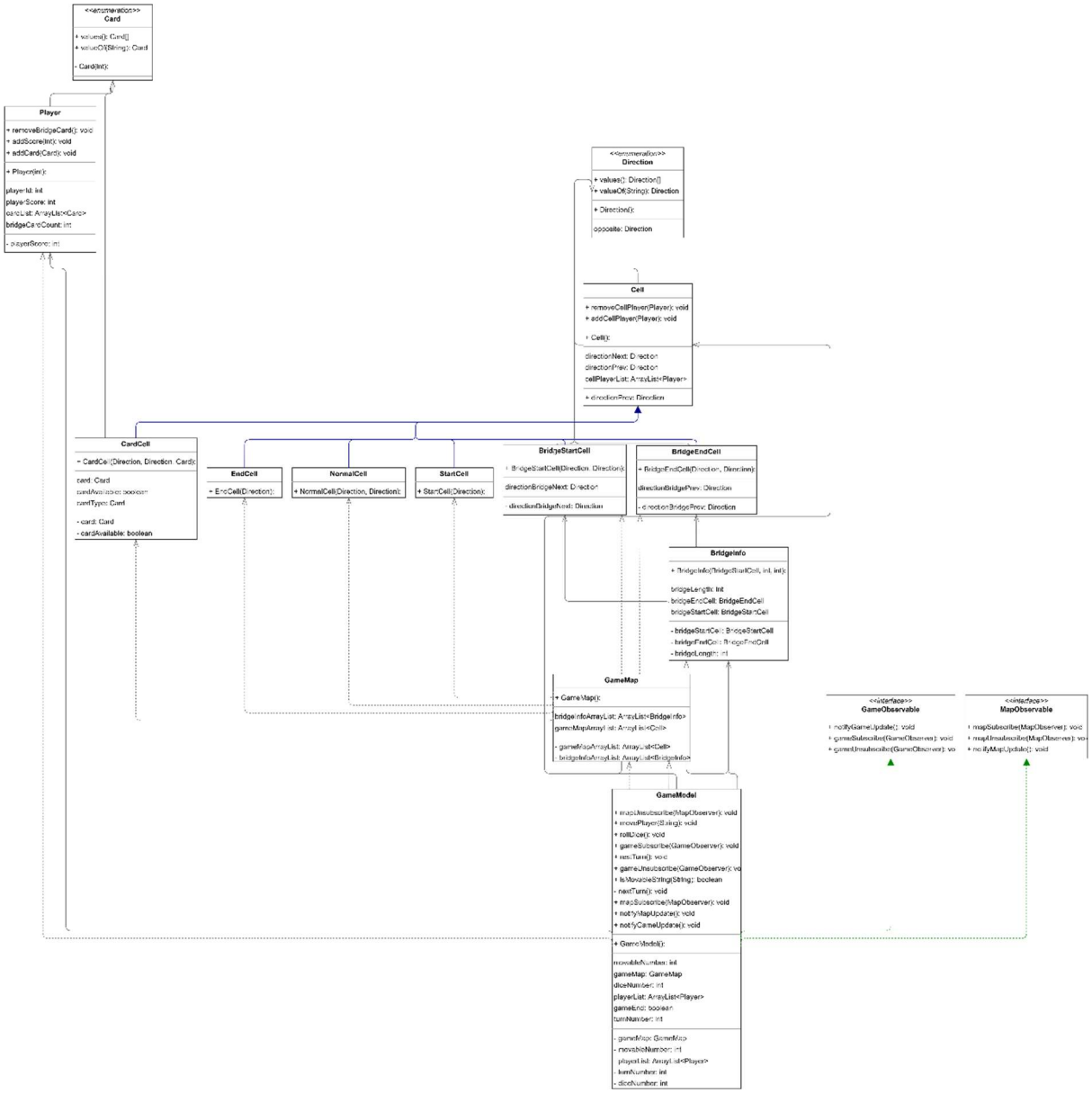
(5) Domain Model



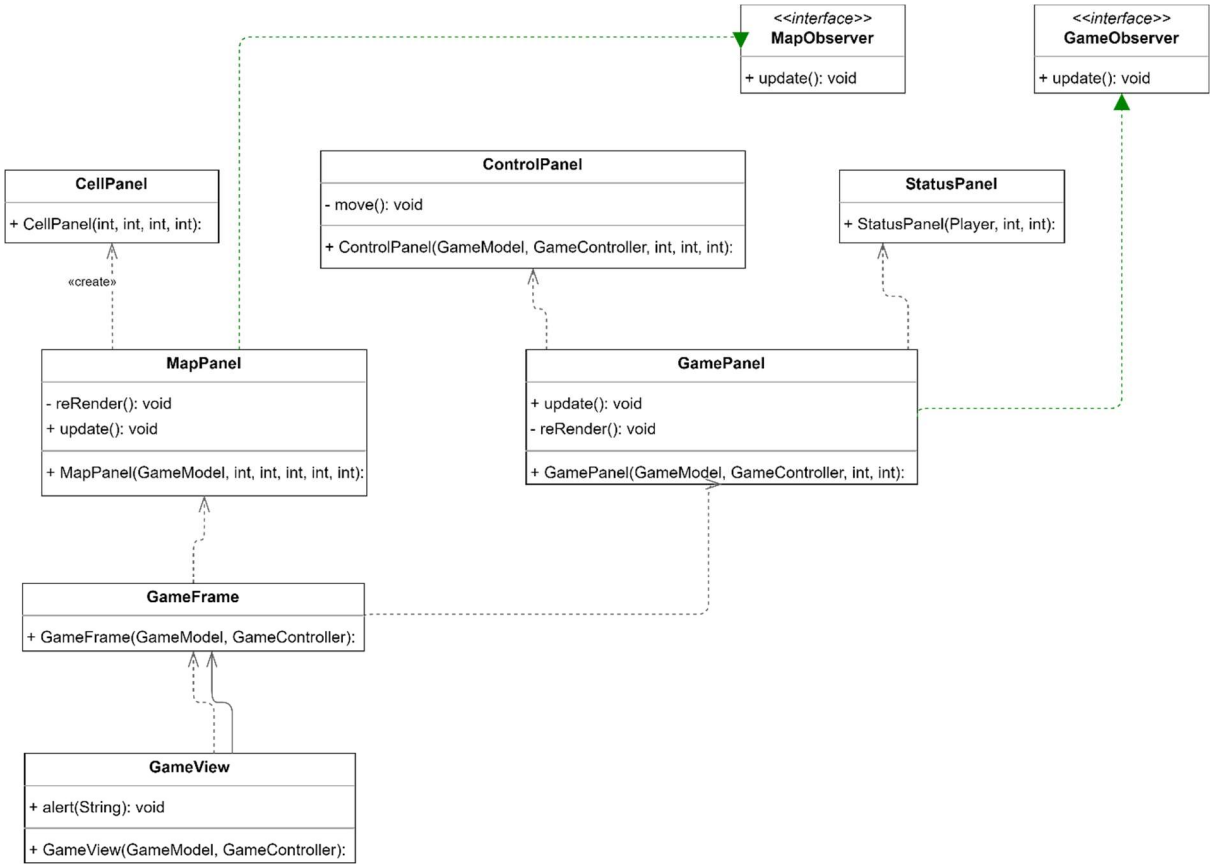
[2] 설계 산출물

(1) Design Class Diagram

1. Model



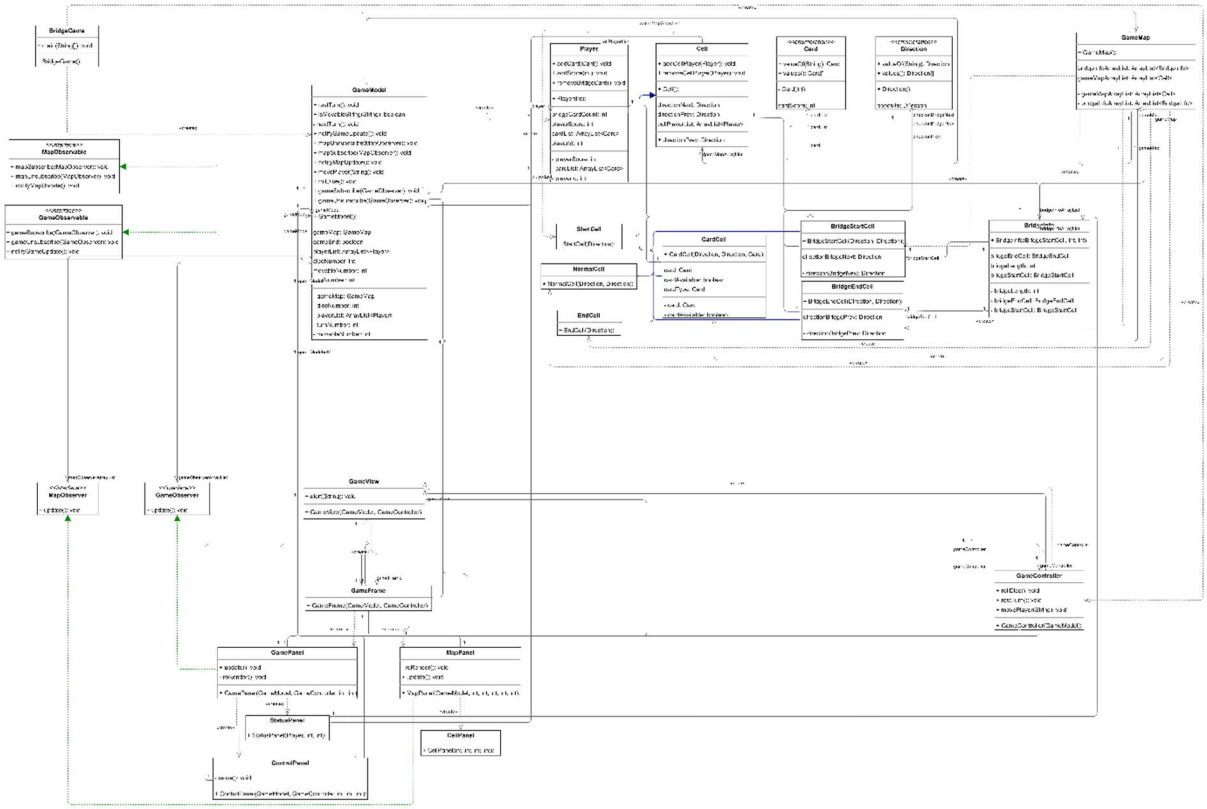
2. View


<<interface>>
MapObserver
+ update(): void
<<interface>>
GameObserver
+ update(): void
CellPanel
+ CellPanel(int, int, int, int):
ControlPanel
- move(): void
+ ControlPanel(GameModel, GameController, int, int, int):
StatusPanel
+ StatusPanel(Player, int, int):
«create»
MapPanel
- reRender(): void
+ update(): void
+ MapPanel(GameModel, int, int, int, int, int):
GamePanel
+ update(): void
- reRender(): void
+ GamePanel(GameModel, GameController, int, int):
GameFrame
+ GameFrame(GameModel, GameController):
GameView
+ alert(String): void
+ GameView(GameModel, GameController):

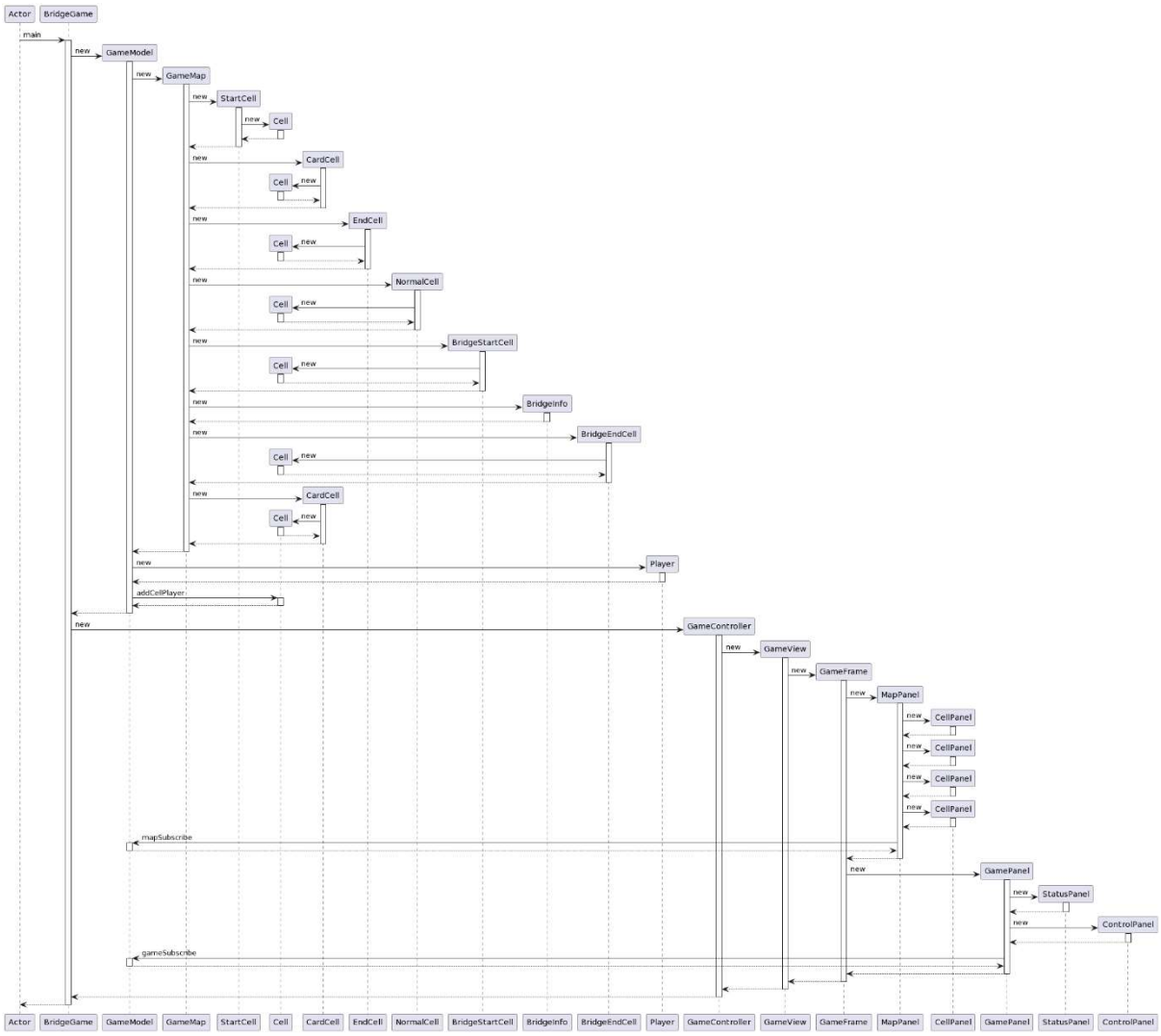
3. Controller

GameController
+ restTurn(): void + movePlayer(String): void + rollDice(): void
+ GameController(GameModel):
- gameView: GameView - gameModel: GameModel

4. Project Design Class Diagram



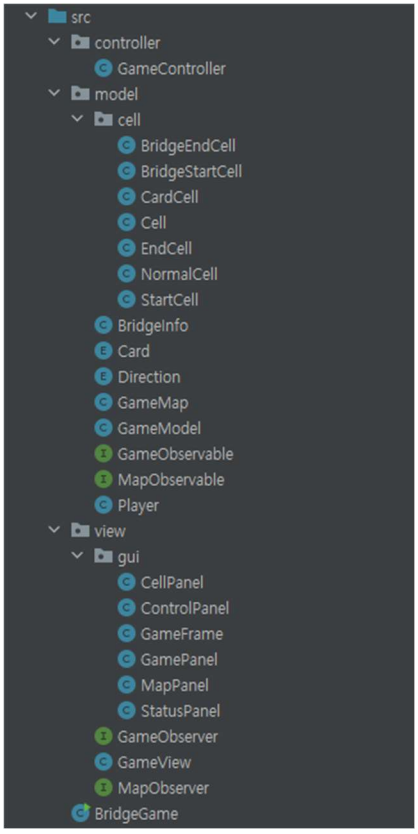
(2) Sequence Diagram



[3] 구현 산출물

(1) 소스 코드 및 실행 파일

/SE_TermProject/src 폴더에 java로 소스 코드가 저장되어있습니다. 전체 구조는 아래와 같습니다.



MVC 구조에 맞추어 개발하였으며, 모델은 model 폴더에, 뷰는 view 폴더에, 컨트롤러는 controller 폴더에 각각 소스 코드가 있습니다.

모델에는 우선 Card, Direction의 Enum 클래스가 존재합니다. 각각 카드와 방향에 대한 정보를 가지는 클래스입니다. 카드는 카드의 점수를 가집니다 (다리 카드의 경우 0점). 카드의 종류를 늘리길 원하는 경우 해당 Enum 클래스에 카드를 추가할 수 있습니다.

Card.java

```java
package model;

public enum Card {
    B( cardScore: 0),
    P( cardScore: 1),
    H( cardScore: 2),
    S( cardScore: 3);
    private final int cardScore;

    private Card(int cardScore) { this.cardScore = cardScore; }

    public int getCardScore() { return cardScore; }
}
```

Direction.java

```java
package model;

public enum Direction {
    U, D, L, R;

    public Direction getOpposite() {
        return switch (this) {
            case U -> D;
            case D -> U;
            case L -> R;
            case R -> L;
        };
    }
}
```

다음으로, Player.java는 플레이어 클래스를 정의합니다. 현재 플레이어가 가진 카드, 플레이어의 id(숫자), 플레이어의 점수를 저장합니다.

Player.java

```java
package model;

import java.util.ArrayList;

public class Player {
    private ArrayList<Card> cardList;
    private int playerId;
    private int playerScore;

    public Player(int id) {
        this.cardList = new ArrayList<Card>();
        this.playerId = id;
        this.playerScore = 0;
    }

    public int getPlayerId() { return playerId; }

    public ArrayList<Card> getCardList() { return this.cardList; }

    public void addCard(Card card) {...}

    public int getBridgeCardCount() {...}

    public void removeBridgeCard() {...}

    public void addScore(int score) { this.playerScore += score; }

    public int getPlayerScore() { return this.playerScore; }
}

```

다음으로, 모델 내에 cell이라는 별도의 패키지가 존재합니다. Cell 클래스는 방향 두개를 가지며, 셀에 존재하는 플레이어를 저장합니다. StartCell, EndCell, NormalCell은 기본적인 셀 클래스이고, CardCell은 카드를 저장하는 셀, BridgeStartCell, BridgeEndCell은 게임의 핵심이 되는 다리 셀입니다. 이 클래스들은 모두 Cell 클래스를 상속받아 구현됩니다. StartCell과 EndCell의 경우 Direction을 하나만 가지며, BridgeStartCell과 BridgeEndCell은 Direction을 세개 가집니다. Cell.java는 아래와 같습니다.

Cell.java

```java
package model.cell;

import model.Direction;
import model.Player;

import java.util.ArrayList;

public class Cell {
    public Direction directionPrev;
    public Direction directionNext;
    public ArrayList<Player> cellPlayerList;

    public Cell() { cellPlayerList = new ArrayList<Player>(); }

    public Direction getDirectionPrev() { return this.directionPrev; }

    public Direction getDirectionNext() { return this.directionNext; }

    public void addCellPlayer(Player player) { this.cellPlayerList.add(player); }

    public ArrayList<Player> getCellPlayerList() { return this.cellPlayerList; }

    public void removeCellPlayer(Player player) throws Exception {
        try {
            this.cellPlayerList.remove(player);
        } catch (Exception e) {
            throw e;
        }
    }
}
```

맵을 구성하기 전, 다리의 정보를 저장하기 위해 추가적인 클래스를 하나 더 구현했습니다. BridgeInfo 클래스가 이 역할을 수행하는데, BridgeStartCell과 BridgeEndCell, 그리고 다리의 길이를 저장합니다. BridgeInfo.java는 아래와 같습니다.

BridgeInfo.java

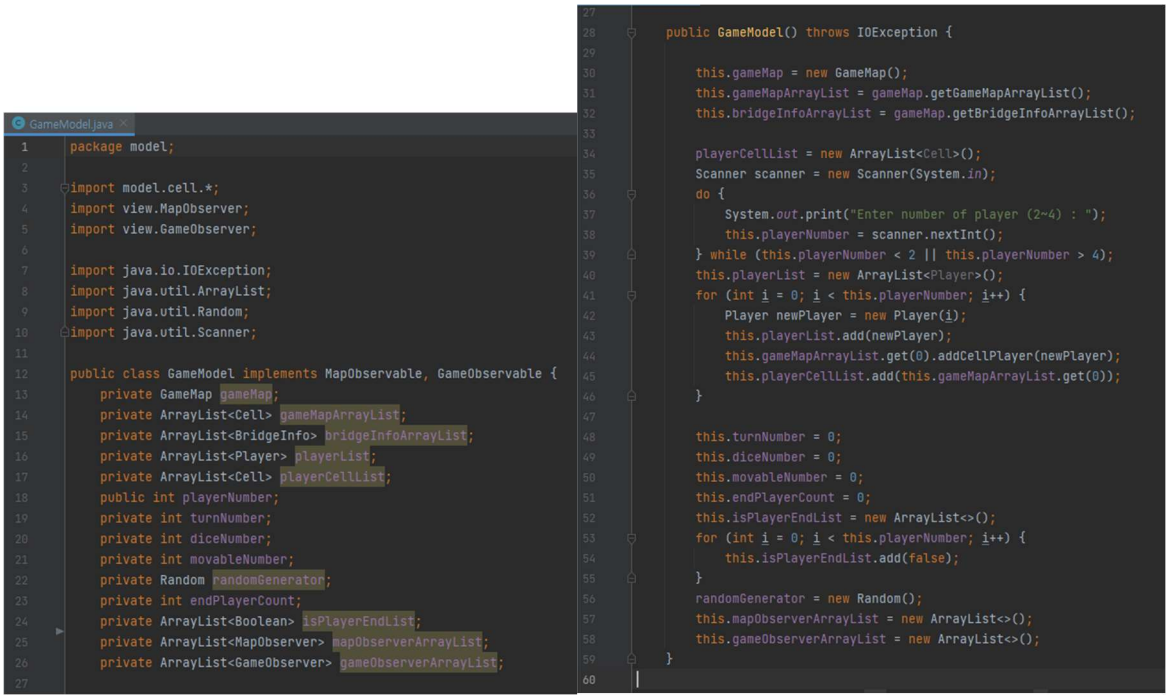
```java
package model;

import model.cell.BridgeEndCell;
import model.cell.BridgeStartCell;

public class BridgeInfo {
    private BridgeStartCell bridgeStartCell;
    private BridgeEndCell bridgeEndCell;
    public int x;
    public int y;
    private int bridgeLength;

    public BridgeInfo(BridgeStartCell bridgeStartCell, int x, int y) {
        this.bridgeStartCell = bridgeStartCell;
        this.bridgeEndCell = null;
        this.x = x;
        this.y = y;
    }

    public void setBridgeEndCell(BridgeEndCell bridgeEndCell) { this.bridgeEndCell = bridgeEndCell; }

    public BridgeEndCell getBridgeEndCell() { return this.bridgeEndCell; }

    public BridgeStartCell getBridgeStartCell() { return this.bridgeStartCell; }

    public void setBridgeLength(int bridgeLength) { this.bridgeLength = bridgeLength; }

    public int getBridgeLength() { return this.bridgeLength; }
}

```

이 클래스들을 바탕으로 게임의 지도를 생성하는데, GameMap 클래스가 게임의 지도를 구축합니다. 해당 클래스의 생성자에서 사용자로부터 지도 파일의 이름을 입력받습니다. 파일이 존재하면, 해당 지도 파일을 읽습니다. 한 줄씩 읽으며 각각 올바른 셀 객체를 만들어서 ArrayList에 저장합니다. 다리 시작 셀이 나올 경우 BridgeInfo의 ArrayList에 새로운 BridgeInfo를 만들어 저장합니다. 다리 끝 셀이 나올 경우 BridgeInfo의 ArrayList에서 해당하는 BridgeStartCell을 가지는 BridgeInfo를 찾아서 BridgeEndCell을 저장합니다. GameMap.java는 아래와 같습니다.

GameMap.java

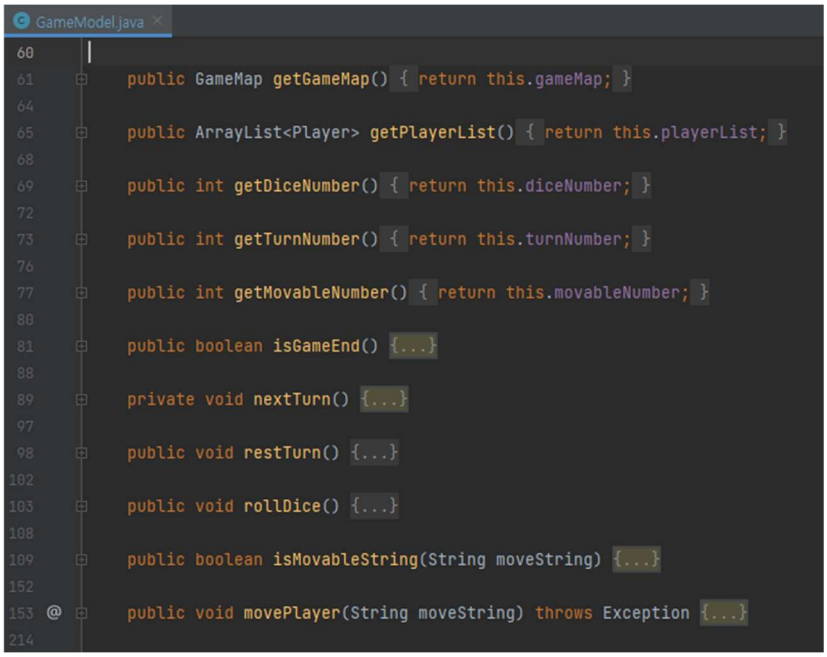
```java
package model;

import ...

public class GameMap {
    private ArrayList<Cell> gameMapArrayList;
    private ArrayList<BridgeInfo> bridgeInfoArrayList;

    public GameMap() throws IOException {

        Scanner scanner = new Scanner(System.in);

        BufferedReader mapFileReader;
        while (true) {...}

        gameMapArrayList = new ArrayList<>();
        bridgeInfoArrayList = new ArrayList<>();

        String mapReadBuffer;
        Direction directionPrev, directionNext = null;
        int x = 0, y = 0;
        while ((mapReadBuffer = mapFileReader.readLine()) != null) {...}
        mapFileReader.close();
    }

    public ArrayList<Cell> getGameMapArrayList() { return this.gameMapArrayList; }

    public ArrayList<BridgeInfo> getBridgeInfoArrayList() { return this.bridgeInfoArrayList; }
}

```

마지막으로 게임의 모든 로직을 담당하는 GameModel 클래스가 존재합니다. 해당 클래스의 생성자에서 GameMap 객체를 생성해서 지도 정보를 생성합니다. 또한 플레이어 수를 입력 받아 해당 수만큼 Player 객체를 생성한뒤 ArrayList에 저장합니다. 다음으로 게임 로직에 관련 된 모든 변수를 초기화시킵니다.



```java
package model;

import model.cell.*;
import view.MapObserver;
import view.GameObserver;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

public class GameModel implements MapObservable, GameObserver {
    private GameMap gameMap;
    private ArrayList<Cell> gameMapArrayList;
    private ArrayList<BridgeInfo> bridgeInfoArrayList;
    private ArrayList<Player> playerList;
    private ArrayList<Cell> playerCellList;
    public int playerNumber;
    private int turnNumber;
    private int diceNumber;
    private int movableNumber;
    private Random randomGenerator;
    private int endPlayerCount;
    private ArrayList<Boolean> isPlayerEndList;
    private ArrayList<MapObserver> mapObserverArrayList;
    private ArrayList<GameObserver> gameObserverArrayList;

    public GameModel() throws IOException {

        this.gameMap = new GameMap();
        this.gameMapArrayList = gameMap.getGameMapArrayList();
        this.bridgeInfoArrayList = gameMap.getBridgeInfoArrayList();

        playerCellList = new ArrayList<Cell>();
        Scanner scanner = new Scanner(System.in);
        do {
            System.out.print("Enter number of player (2~4) : ");
            this.playerNumber = scanner.nextInt();
        } while (this.playerNumber < 2 || this.playerNumber > 4);
        this.playerList = new ArrayList<Player>();
        for (int i = 0; i < this.playerNumber; i++) {
            Player newPlayer = new Player(i);
            this.playerList.add(newPlayer);
            this.gameMapArrayList.get(0).addCellPlayer(newPlayer);
            this.playerCellList.add(this.gameMapArrayList.get(0));
        }

        this.turnNumber = 0;
        this.diceNumber = 0;
        this.movableNumber = 0;
        this.endPlayerCount = 0;
        this.isPlayerEndList = new ArrayList<>();
        for (int i = 0; i < this.playerNumber; i++) {
            this.isPlayerEndList.add(false);
        }
        randomGenerator = new Random();
        this.mapObserverArrayList = new ArrayList<>();
        this.gameObserverArrayList = new ArrayList<>();
    }
```

아래의 스크린샷은 GameModel 클래스가 가지는 게임 로직과 관련된 함수들입니다.



```java
    public GameMap getGameMap() { return this.gameMap; }

    public ArrayList<Player> getPlayerList() { return this.playerList; }

    public int getDiceNumber() { return this.diceNumber; }

    public int getTurnNumber() { return this.turnNumber; }

    public int getMovableNumber() { return this.movableNumber; }

    public boolean isGameEnd() {...}

    private void nextTurn() {...}

    public void restTurn() {...}

    public void rollDice() {...}

    public boolean isMovableString(String moveString) {...}

    public void movePlayer(String moveString) throws Exception {...}
```

컨트롤러에는 단 하나의 클래스만 존재합니다. GameController 클래스는 생성자에서 GameModel 객체를 인자로 받으며, 새로운 뷰를 생성합니다. 또한 뷰에서 사용자가 어떠한 조작을 했을 때 컨트롤러의 함수를 호출하도록 만들었습니다. 해당 함수는 다시 모델의 함수를 호출함으로써 간단한 로직 판별을 하도록 설계되었습니다. 추후 설명할 옵저버 패턴을 이용하기 때문에 모델의 함수를 호출한 뒤 직접 뷰를 업데이트하지 않고, 뷰를 업데이트 하는 기능은 모델에 구현되어 있습니다. 간단한 로직 판별을 하고 사용자의 조작이 로직과 맞지 않다면 뷰에 구현된 alert 함수를 호출하여 사용자에게 해당하는 경고 메시지 팝업창을 띄우도록 설계했습니다.

GameController.java

```
1   package controller;
2
3   import model.GameModel;
4   import view.GameView;
5
6   import java.io.IOException;
7
8   public class GameController {
9       private GameModel gameModel;
10      private GameView gameView;
11
12      public GameController(GameModel gameModel) throws IOException {
13          this.gameModel = gameModel;
14          this.gameView = new GameView(this.gameModel,  gameController: this);
15      }
16
17      public void rollDice() {...}
24
25      public void restTurn() {...}
32
33      public void movePlayer(String moveString) throws Exception {...}
55  }
56
```

마지막으로 뷰는 GameView에서 모든 클래스를 생성합니다. GameView 클래스의 생성자에서 사용자에게 입력을 받아 어떠한 디스플레이 모드로 게임을 실행할 것인지 선택합니다. 현재 GUI 버전밖에 구현을 하지 않았지만, 추후 CUI 버전을 추가하기에 용이하게 설계했습니다. GameView.java는 아래와 같습니다.

GameView.java

```java
package view;

import ...

public class GameView {
    private GameModel gameModel;
    private GameController gameController;
    private int gameMode;
    private GameFrame gameFrame;

    public GameView(GameModel gameModel, GameController gameController) throws IOException {
        this.gameModel = gameModel;
        this.gameController = gameController;
        System.out.println("1) GUI");
        System.out.print("Select Mode (press enter for GUI): ");

        Scanner scanner = new Scanner(System.in);
        String modeInput = scanner.nextLine();

        if (modeInput.length() == 0 || modeInput.equals("1")) {
            this.gameMode = 1;
            GameFrame gameFrame = new GameFrame(this.gameModel, this.gameController);
            this.gameFrame = gameFrame;
            this.gameFrame.setVisible(true);
        }
    }

    public void alert(String alertMessage) {...}
}

```

GameView에서 GUI가 선택되면 gui 패키지 내의 GameFrame 클래스의 객체가 만들어집니다. GameFrame은 JFrame을 상속받으며, 지도, 플레이어 상태, 조작 등의 정보가 담긴 JPanel 객체를 생성하고 추가합니다.

GameFrame.java

```java
import ...

public class GameFrame extends JFrame {
    private final int cellSize;
    private int gameFrameWidth;
    private int gameFrameHeight;
    private final int gamePanelWidth;
    private GameModel gameModel;
    private GameMap gameMap;
    private GameController gameController;

    public GameFrame(GameModel gameModel, GameController gameController) throws IOException {
        this.setTitle("Bridge Game GUI");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(true);
        this.setLayout(null);
        this.setBackground(Color.white);
        this.gameModel = gameModel;
        this.gameMap = gameModel.getGameMap();
        this.gameController = gameController;
        int minx = 0, miny = 0, maxx = 0, maxy = 0, curx = 0, cury = 0;
        Cell currentCell = null;
        ArrayList<Cell> gameMapArrayList = this.gameMap.getGameMapArrayList();
        for (int i = 0; i < gameMapArrayList.size(); i++) {...}
        this.cellSize = 60;
        curx = this.cellSize * (1 - minx);
        cury = this.cellSize * (1 - miny);
        int mapPanelWidth = (maxx - minx + 6) * this.cellSize;
        int mapPanelHeight = (maxx - minx + 6) * this.cellSize;
        MapPanel mapPanel = new MapPanel(this.gameModel, curx, cury, this.cellSize, mapPanelWidth, mapPanelHeight);
        this.add(mapPanel);
        this.gamePanelWidth = 300;
        GamePanel gamePanel = new GamePanel(gameModel, gameController, gamePanelWidth, mapPanel.getWidth());
        this.gameFrameWidth = mapPanelWidth + this.gamePanelWidth;
        this.gameFrameHeight = Math.max(mapPanelHeight, gamePanel.getHeight());
        this.setSize(this.gameFrameWidth, this.gameFrameHeight);
        this.add(gamePanel);
        this.setLocationRelativeTo(null);
    }
}
```

다음으로, MVC 패턴과 동시에 옵저버 패턴을 구현하였습니다. 뷰 패키지 내에 Observer 와 모델 패키지 내에 Observable 인터페이스를 구현했습니다. 지도 정보 업데이트와 플레이어 상태 정보 업데이트를 각각 달리 하기 위해 MapObserver / MapObservable 과 GameObserver / GameObservable 을 따로 구현했습니다.

MapObservable.java

```java
package model;

import view.MapObserver;

public interface MapObservable {
    public void mapSubscribe(MapObserver o);

    public void mapUnsubscribe(MapObserver o);

    public void notifyMapUpdate();
}
```

GameObservable.java

```java
package model;

import view.GameObserver;

public interface GameObservable {
    public void gameSubscribe(GameObserver o);

    public void gameUnsubscribe(GameObserver o);

    public void notifyGameUpdate();
}
```

MapObserver.java

```java
package view;

public interface MapObserver {
    public void update();
}
```

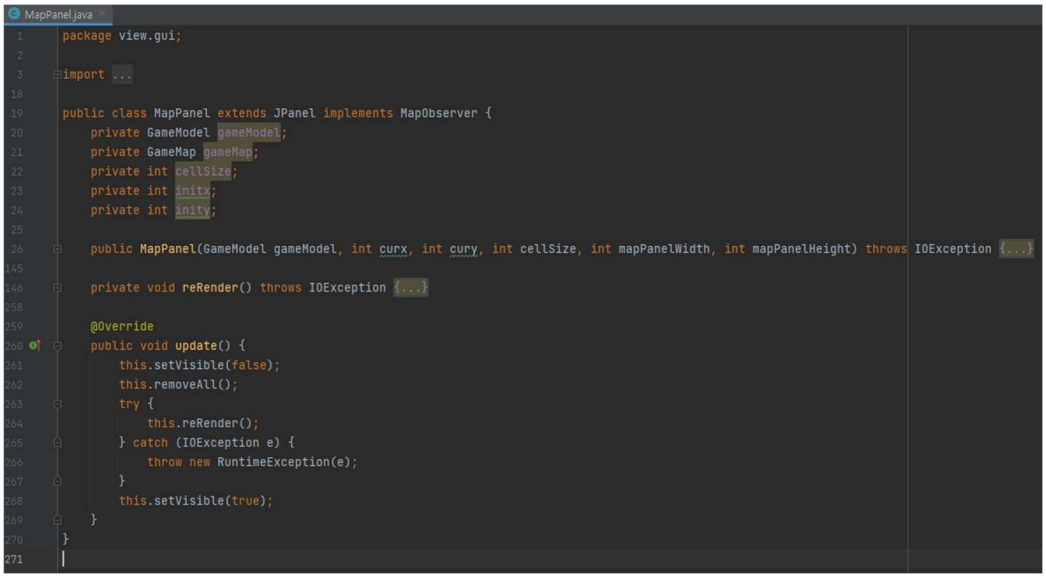
GameObserver.java

```java
package view;

public interface GameObserver {
    public void update();
}
```

모델에서는 GameModel 클래스가 MapObservable 과 GameObservable 을 모두 implement 합니다. 저장된 gameMap 과 Player 의 리스트가 변화하는 경우에 각각 Observable 인터페이스에 구현된 notify 메소드를 호출하여 구독중인 뷰를 업데이트하도록 합니다. 아래의 스크린샷은 GameModel.java 에 구현된 각 Observable 인터페이스의 메서드 구현입니다.

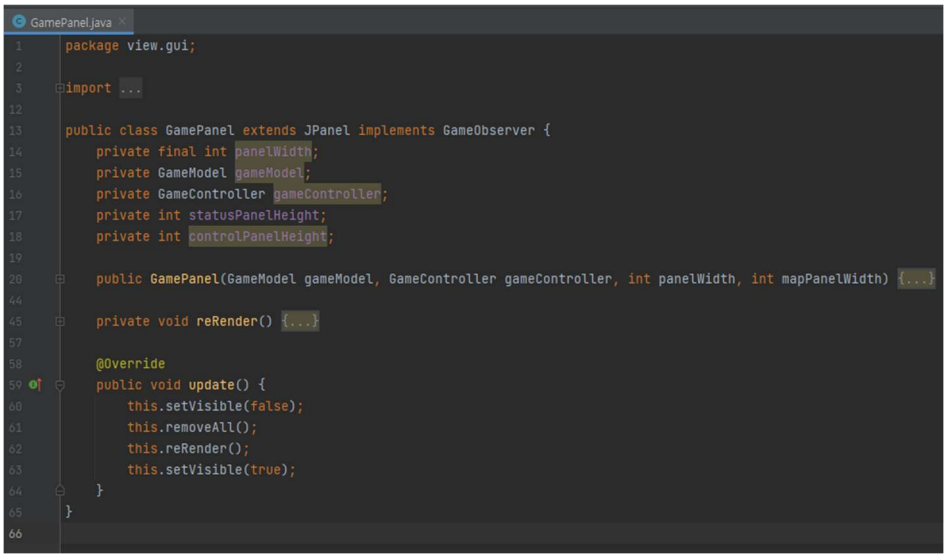
```java
    @Override
    public void mapSubscribe(MapObserver o) { this.mapObserverArrayList.add(o); }

    @Override
    public void mapUnsubscribe(MapObserver o) { this.mapObserverArrayList.remove(o); }

    @Override
    public void notifyMapUpdate() {
        for (int i = 0; i < this.mapObserverArrayList.size(); i++) {
            this.mapObserverArrayList.get(i).update();
        }
    }

    @Override
    public void gameSubscribe(GameObserver o) { this.gameObserverArrayList.add(o); }

    @Override
    public void gameUnsubscribe(GameObserver o) { this.gameObserverArrayList.remove(o); }

    @Override
    public void notifyGameUpdate() {
        for (int i = 0; i < this.gameObserverArrayList.size(); i++) {
            this.gameObserverArrayList.get(i).update();
        }
    }
}
```

뷰에서는 맵을 띄워주는 MapPanel 클래스가 MapObserver를 implement하며 update 메서드를 구현합니다. 해당 클래스의 생성자에서 모델의 subscribe 함수를 호출하여 구독하도록 설계되어있습니다. 또한 update 메서드가 호출되면 reRender 함수를 호출하도록 설계하였는데, 해당 함수의 구현부는 생성자의 구현부와 거의 유사합니다. 아래는 MapPanel.java입니다.



```java
package view.gui;

import ...

public class MapPanel extends JPanel implements MapObserver {
    private GameModel gameModel;
    private GameMap gameMap;
    private int cellSize;
    private int initx;
    private int inity;

    public MapPanel(GameModel gameModel, int curx, int cury, int cellSize, int mapPanelWidth, int mapPanelHeight) throws IOException {...}

    private void reRender() throws IOException {...}

    @Override
    public void update() {
        this.setVisible(false);
        this.removeAll();
        try {
            this.reRender();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        this.setVisible(true);
    }
}
```

GameObserver를 implement하는 GamePanel 클래스도 거의 동일한 구조를 지닙니다. 아래는 GamePanel.java입니다.



```java
package view.gui;

import ...

public class GamePanel extends JPanel implements GameObserver {
    private final int panelWidth;
    private GameModel gameModel;
    private GameController gameController;
    private int statusPanelHeight;
    private int controlPanelHeight;

    public GamePanel(GameModel gameModel, GameController gameController, int panelWidth, int mapPanelWidth) {...}

    private void reRender() {...}

    @Override
    public void update() {
        this.setVisible(false);
        this.removeAll();
        this.reRender();
        this.setVisible(true);
    }
}
```

마지막으로, 게임의 진입점은 src 폴더 내의 BridgeGame.java입니다. 해당 클래스의 main 함수에서 우선 게임의 모델인 GameModel 객체를 생성합니다. 그 다음 컨트롤러인 GameController 객체를 생성하고 직전에 생성한 모델 객체를 컨트롤러의 생성자에 인자로 줍니다. 뷰는 컨트롤러에서 생성되므로, 모델과 컨트롤러만 해당 클래스에서 생성하면 MVC 패턴에 맞추어 게임이 진행되도록 설계했습니다.

(2) 프로그램 사용 방법

README.txt에도 프로그램 사용 방법이 명시되어 있습니다. 프로젝트 개발은 Windows 10 운영체제에서 Java 18 버전을 사용하여 개발했습니다.

1. 게임 실행 방법

1. 프로젝트 디렉터리의 map 디렉터리에 원하는 지도 파일을 추가합니다.
2. 명령 프롬프트를 키고 프로젝트 디렉터리로 이동합니다.
3. java BridgeGame 을 입력합니다.
4. 원하는 맵 파일의 이름을 입력하고 엔터를 누릅니다. (ex. default.map, default)
   4-1. 아무것도 입력하지 않고 엔터를 누르면 default.map이 로드됩니다.
5. 플레이어의 수를 입력합니다. (2~4인)
6. 디스플레이 모드를 입력하고 엔터를 누릅니다. (현재 GUI만 제공)
   6-1. 아무것도 입력하지 않고 엔터를 누르면 GUI로 실행됩니다.

2. 게임 진행 방법

1. 우측 하단에 Roll 버튼을 누르거나 Rest 버튼을 누릅니다.
   1-1. Roll 버튼을 누른 경우 해당 플레이어의 주사위 숫자와 이동 가능한 칸 수가 표시됩니다.
      1-1-1. Roll 버튼 하단의 입력창에 원하는 움직임을 U, D, L, R 또는 u, d, l, r의 조합으로 입력합니다.
      1-1-2. 입력을 마치고 엔터를 누르거나 우측의 Move 버튼을 클릭합니다.
         1-1-2-1. 불가능한 움직임을 입력한 경우 경고창을 끄고 1-1-1부터 반복합니다.
   1-2. Rest 버튼을 누른 경우 해당 플레이어의 다리 카드(B) 한장이 제거됩니다.
2. 다음 플레이어의 차례로 넘어갑니다