

SE Term Project Report

OOAD 를 이용한 Bridge 게임 개발

20181594 김 형 기

목차

1 요구 분석

1.1 요구 정의 및 가정

1.2 요구 분석

1.2.1 Use case diagram

1.2.2 Use case specification

1.2.3 Domain model

1.2.4 System sequence diagram

1.2.5 Operation contracts

2 설계

2.1 Logical Architecture

2.2 Design Sequence Diagram

2.3 Design Class

2.3.1 Design Class

2.3.2 Design Class Diagram

3 구현

4 테스트

4.1 Unit test

4.2 프로그램 실행

4.2.1 GUI

4.2.2 CLI

1. 요구 분석

1.1. 요구 정의 및 가정

제시된 요구 조건은 다음과 같다.

- a. 게임은 2 ~ 4 명의 플레이어가 함께 수행할 수 있다.
- b. 게임 플레이가 시작되면 플레이어 수를 입력하여 원하는 참가자의 수만큼 게임을 진행할 수 있다.
- c. 지도 데이터 파일은 기본으로 default.map 을 사용하며, 임의의 파일을 입력 받아 사용할 수 있어야 한다.
- d. 게임 플레이에 사용할 지도 데이터를 표시할 수 있어야 한다.
- e. 모든 플레이어는 START 셀에서 시작한다.
- f. 각 플레이어의 현재 위치가 지도 상에 표시되어야 한다.
- g. 각 플레이어가 가진 카드의 개수가 표시되어야 한다.
- h. 각 플레이어의 턴의 되면, 주사위를 굴릴지 쉴지 결정할 수 있어야 한다.
- i. 플레이어가 자신의 턴을 쉬게 되면 다리 카드 1장을 반납한다.
- j. 플레이어는 주사위 눈금에서 다리 카드를 제외한 만큼 이동할 수 있다.
- k. 플레이어가 이동할 방향을 U, D, L, R 의 조합으로 입력 받고 올바른 방향인지 확인할 수 있어야 한다. 또한 올바르다면 이동한다.
- l. 플레이어는 다리를 건널 것인지 선택할 수 있으며, 다리를 이동하는 것도 하나의 셀처럼 한 칸의 이동으로 계산된다.
- m. 다리를 건너면 다리 카드 한 장을 받게 된다.
- n. 플레이어의 말이 도구 카드 위로 이동한다면 해당 카드에 해당하는 점수를 얻게 된다. (Saw : 3 점, Hammer : 2 점, Philips Driver : 1 점)
- o. 1명 이상의 플레이어가 도착지 (END 셀)에 도착한 이후에는 모든 플레이어가 뒤로 이동할 수 없다.
- p. 플레이어는 도착지에 도착한 순서에 따라 1등은 7 점, 2등은 3 점, 3등은 1 점을 받는다.
- q. 보드 위에 1명의 플레이어가 남으면 남은 턴을 더 진행하지 않고 게임이 종료된다.
- r. 게임이 종료될 때 가장 많은 점수를 획득한 플레이어가 승리한다.

다음은 요구사항에서 명확하게 제시되지 않은 것들을 가정한 것이다.

- a. 턴의 진행 순서는 플레이어 1 → 4 로 진행된다.
- b. 게임은 오프라인에서 하나의 기기를 이용해 플레이어들이 자신의 차례마다 번갈아가며 사용한다.
- c. 지도 데이터의 셀은 최소 2 개 이상 존재한다.
- d. 도구 아이템은 이동 후 완전히 도착한 지점에 대해서만 얻게 되며, 이동 경로를 지나칠 때는 얻을 수 없다.
- e. 플레이어가 다리를 건너기 위해서는 도구 아이템을 얻는 것처럼 다리 셀에 완전히 도착한 다음에 선택할 수 있다.

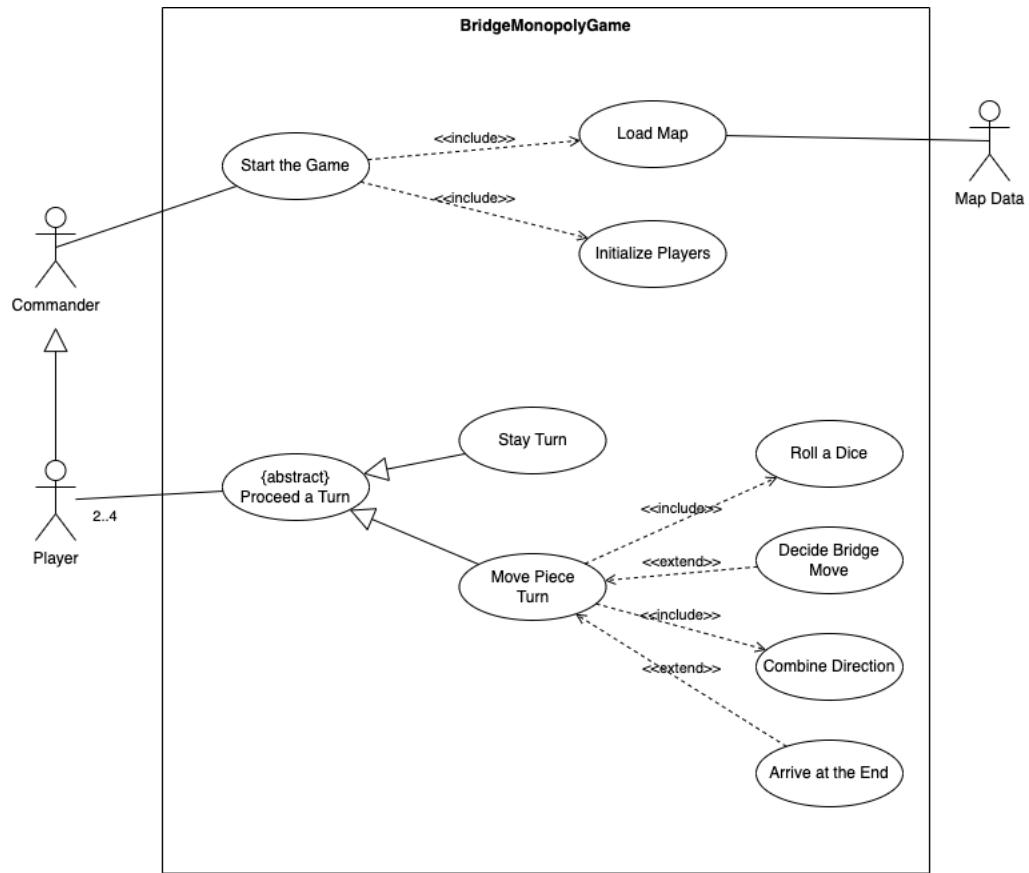
A. 이동 가능한 눈금이 다리를 건너기 위해 필요한 2 미만일 경우 다리를 건널 것인지 선택할 수 없다.

1.2. 요구 분석

1.2.1. Use case diagram

앞서 요구 정의에서 모든 플레이어는 하나의 기기에서 번갈아가며 차례를 수행한다고 가정했다.

따라서 게임을 시작하고 사용할 맵을 고르고, 참여할 인원의 수를 결정하는 것은 플레이어 중 한 명이 수행하게 된다. 따라서 게임의 전반적인 결정을 내리는 Actor 를 Commander 로 분석하고, 자신의 차례에 원하는 행동을 하는 Actor 를 Player 로 분석하였다.



각 유즈 케이스는 시스템과 Actor의 주요 상호작용 및 behavior들을 동사형으로 나타내어 분석하였다. 자세한 설명은 이후 유즈 케이스 명세서에서 다루도록 한다.

1.2.2. Use case specification

각 유즈 케이스를 fully-dressed 형태로 나타내었다.

a) UC101: Start the Game

게임을 시작하는 행동을 하는 유즈 케이스로 게임 전반에 걸쳐서 영향을 끼치는 중요한사항인 맵과 플레이어 수를 결정한다. Commander 는 사용할 맵을 선택하고 UC102 을 include 하여 사용할 맵을 불러온다. Commander 는 참여할 플레이어 수를 결정하고 UC103 을 include 하여 시스템은 플레이어들이 게임을 수행할 수 있도록 준비를 마친다.

Use Case Section	Comment
Use Case Name	Start the Game
Scope	System use case
Level	User-goal
Primary Actor	Commander
Stakeholders and Interests	Commander : 진행할 게임에 사용할 맵과 플레이어 수를 정하고자 한다. Player : 결정된 플레이어 수에 따라 번갈아 가며 턴을 수행하고자 한다. Map Data : 게임에 사용할 맵을 제공해야 한다.
Preconditions	Commander 는 사용할 맵 파일의 이름을 알고 있다.
Success Guarantee	두 개 이상의 셀을 가지는 맵을 불러온다. Player 의 수는 2~4 범위에서 결정된다. 결정한 수만큼 Player 를 생성한다.
Main Success Scenario	1. Commander 는 맵 파일의 이름을 입력한다. 2. include UC102 "Load Map". 3. Commander 는 플레이어 수를 입력한다 4. include UC103 "Initialize Players".
Extensions	1a. 해당하는 맵을 찾을 수 없는 경우 1. 파일이 존재하지 않는다는 메시지를 표시한다. 2. 스텝 1로 돌아간다.

Use Case Section	Comment
	2a. 입력된 셀의 개수가 2 개 미만인 경우 1. 올바르지 않은 맵이라고 표시한다. 2. 스텝 1로 돌아간다. 3a. Player 수가 2~4 범위가 아닌 경우 1. 스텝 3으로 돌아간다.
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	게임을 실행할 경우 처음 한 번 일어난다.
Miscellaneous	-

b) UC102: Load Map

사용할 맵이 결정된 이후에 맵 파일에서 셀의 정보를 불러온다. 맵 파일의 한 줄은 셀 한 개의 정보를 다음과 같이 가지고 있다.

(셀의 종류) (앞 방향으로 연결된 셀) (뒤 방향으로 연결된 셀)

게임을 진행하면서 한 명 이상의 플레이어가 End 셀에 도착하면 이후 모든 플레이어는 뒤 방향으로 연결된 셀로 이동할 수 없으므로, 연결된 방향도 구분해야 한다.

Use Case Section	Comment
Use Case Name	Load Map
Scope	System use case
Level	User-goal
Primary Actor	Commander
Stakeholders and Interests	Commander: 게임에 사용할 맵 데이터를 불러오고자 한다. Map Data: 게임에 사용할 맵을 제공해야 한다.

Use Case Section	Comment
Preconditions	올바른 맵 정보를 입력 받은 상태이다.
Success Guarantee	<p>모든 셀의 정보를 불러온다.</p> <p>각 셀끼리 연결된 방향의 정보를 가지게 된다.</p> <p>각 셀은 종류 또는 아이템 정보를 가지게 된다.</p>
Main Success Scenario	<ol style="list-style-type: none"> 1. Map Data 는 맵 정보 중 한 줄을 입력한다. 2. 첫 글자를 이용해 셀의 종류를 판단한다. 3. 현재 입력의 두 번째 글자가 나타내는 방향으로 현재 셀에 이전 셀을 연결한다. 4. 이전 입력의 마지막 글자가 나타내는 방향으로 이전 셀에 현재 셀을 연결한다. 5. 스텝 1~4 를 반복한다. 6. Bridge 셀과 인접한 셀을 연결한다.
Extensions	<ol style="list-style-type: none"> 2a. 첫 글자가 'S'이며, 입력된 글자가 2개인 경우 <ol style="list-style-type: none"> 1. Start 셀로 결정한다. 2b. 첫 글자가 'E'인 경우 <ol style="list-style-type: none"> 1. End 셀로 결정한다. 2c. 첫 글자가 'C'인 경우 <ol style="list-style-type: none"> 1. 일반 셀로 결정한다. 2d. 첫 글자가 'B'인 경우 <ol style="list-style-type: none"> 1. Bridge 셀로 결정한다. 2e. 첫 글자가 'H'인 경우 <ol style="list-style-type: none"> 1. Hammar 셀로 결정한다. 2f. 첫 글자가 'S'이며, 입력된 글자가 3개인 경우 <ol style="list-style-type: none"> 1. Saw 셀로 결정한다. 2g. 첫 글자가 'P'인 경우 <ol style="list-style-type: none"> 1. Philips Driver 셀로 결정한다. 3a. 현재 Start 또는 End 셀인 경우 <ol style="list-style-type: none"> 1. 해당 스텝을 건너뛴다.

Use Case Section	Comment
	4a. 현재 Start 셀인 경우 1. 해당 스텝을 건너뛴다.
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	게임마다 최대 한 번 일어난다.
Miscellaneous	-

c) UC103: Initialize Players

Commander 가 플레이어 수를 결정한 이후에 이에 맞게 Player 와 Piece 를 생성하고, Turn 을 초기화한다. 턴은 초기화될 때 첫번째로 입력된 플레이어를 가리키도록 한다.

Use Case Section	Comment
Use Case Name	Initialize Players
Scope	System use case
Level	User-goal
Primary Actor	Commander
Stakeholders and Interests	Commander : 선택한 플레이어 수를 이용해 게임을 진행하고자 한다. Player : 처음 선택된 플레이어 수만큼 턴을 분배 받고자 한다.
Preconditions	올바른 플레이어 수를 입력받은 상태이다.
Success Guarantee	플레이어를 생성한다.

Use Case Section	Comment
Main Success Scenario	1. 플레이어 수만큼 Player 를 생성한다. 2. 모든 Player 의 Piece 의 위치를 Start 셀에 위치한다. 3. 현재 턴을 Player 1 로 설정한다. 4. 모든 Player 및 Piece 를 화면에 표시한다.
Extensions	-
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	게임을 실행할 경우 처음 한 번 일어난다.
Miscellaneous	-

d) UC201: Proceed a Turn

시스템이 관리하는 턴에 따라 각 플레이어 Actor 는 차례를 지정 받는다. 자신의 차례에 해당하는 Actor Player 는 턴을 쉬거나, 주사위를 굴려서 말을 움직이는 결정을 할 수 있다. 따라서 abstract use case 로 표현하였으며 원하는 행동이 끝나면 시스템은 다음 플레이어에게 차례를 넘기게 된다.

Use Case Section	Comment
Use Case Name	{abstract} Proceed a Turn
Scope	System use case
Level	User-goal
Primary Actor	Player
Stakeholders and Interests	Player : 자신의 턴을 진행하고자 한다.

Use Case Section	Comment
Preconditions	게임은 초기화 완료된 상태이다. 현재 Player 의 턴이다.
Success Guarantee	Player 는 원하는 행동을 완료하고 턴을 넘겨준다.
Main Success Scenario	1. (abstract) Player 가 선택한 행동을 진행한다. 2. 턴을 다음 Player 에게 넘긴다.
Extensions	-
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	게임이 종료될 때까지 매 턴마다 실행된다.
Miscellaneous	-

e) UC202: Stay Turn

플레이어 Actor 가 이번 턴을 쉬어가기로 결정했다면 다리 카드를 감소시킨다. 이후 턴을 넘기는 것은 상위 유즈 케이스인 UC201에서 진행한다.

Use Case Section	Comment
Use Case Name	Stay Turn
Scope	System use case
Level	User-goal
Primary Actor	Player
Stakeholders and Interests	Player : 현재 턴에 말을 움직이지 않고 쉬고자 한다.

Use Case Section	Comment
Preconditions	Player 는 End 셀에 도달하지 않았다.
Success Guarantee	Player 의 다리 카드가 감소한다.
Main Success Scenario	1. Player 의 다리 카드를 감소한다. 2. generalization UC 201 "Proceed a Turn"
Extensions	1a. Player 의 다리 카드 수가 0 인 경우 1. 스텝을 건너뛴다.
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	턴마다 최대 한 번 일어난다.
Miscellaneous	-

f) UC203: Move Piece Turn

플레이어가 주사위를 굴려서 말을 이동하고자 한다.

UC204에서 Player Actor는 시스템에 주사위를 굴리도록 요청을 한다.

UC205에서 만약 현재 플레이어의 말이 다리 셀 위에 있고, 남은 눈금으로 다리 셀을 이동할 수 있다면 Actor에게 다리를 건널 것인지 선택하도록 한다.

이후 UC206에서 남은 눈금으로 이동할 방향을 입력 받는다.

모든 결정이 완료되었다면 해당하는 위치로 말을 이동한다.

Use Case Section	Comment
Use Case Name	Move Piece Turn
Scope	System use case
Level	User-goal

Use Case Section	Comment
Primary Actor	Player
Stakeholders and Interests	Player : 현재 턴에 주사위를 굴려서 말을 이동하고자 한다.
Preconditions	Player 는 End 셀에 도달하지 않았다.
Success Guarantee	Player 의 Piece 가 선택한 위치로 이동한다.
Main Success Scenario	1. include UC 204: "Roll a Dice". 2. include UC 205: "Decide Bridge Move". 3. include UC 206: "Combine Direction" 4. 해당하는 위치로 Piece 를 이동시킨다. generalization UC 201 "Proceed a Turn"
Extensions	-
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	턴마다 최대 한 번 일어난다.
Miscellaneous	-

g) UC204: Roll a Dice

Player actor에게 주사위를 굴리는 요청을 받으면 1 ~ 6의 무작위 수를 생성하고, 결과를 Player actor에게 표시한다.

Use Case Section	Comment
Use Case Name	Roll a Dice
Scope	System use case
Level	User-goal

Use Case Section	Comment
Primary Actor	Player
Stakeholders and Interests	Player : 주사위를 굴려 나온 수를 이용해 말을 이동하고자 한다.
Preconditions	-
Success Guarantee	주사위 결과로 1~6 의 수를 얻는다.
Main Success Scenario	<ol style="list-style-type: none"> 1. Player 는 주사위 굴리기를 시작하는 입력을 한다. 2. 1~6 범위의 무작위 수를 생성한다. 3. 결과를 표시한다.
Extensions	-
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	턴마다 최대 한 번 일어난다.
Miscellaneous	-

h) UC205: Decide Bridge Move

플레이어의 현재 말의 위치가 다리 셀 위에 있고 이동 가능한 눈금이 2칸 이상이라면 다리를 건널 것인지 선택할 수 있다. Player actor 가 다리를 건너도록 요청하면 이동 가능한 눈금 중 2칸을 소모하여 연결된 다리 셀로 말을 이동한다. 이후 남은 눈금으로 UC206 을 수행할 수 있다.

i) UC206: Combine Directions

이동 가능한 눈금(주사위 결과 - 다리 카드 개수)로 말을 이동시킬 방향을 입력 받는다.

Player actor 는 방향의 조합을 U, D, L, R 로 입력한다.

해당 방향으로 이동할 수 있는지 확인하고 올바른 방향이라면 그 방향으로 말을 이동하도록 턴을 진행한다.

Use Case Section	Comment
Use Case Name	Combine Direction
Scope	System use case
Level	User-goal
Primary Actor	Player
Stakeholders and Interests	Player : 주사위를 굴려 나온 수를 이용해 말을 이동하고자 한다.
Preconditions	-
Success Guarantee	주사위 결과로 이동할 수 있는 위치 중에서 Player 의 선택을 알아낸다.
Main Success Scenario	<p>1. Player 는 이동할 방향 조합을 입력한다.</p> <p>2. 현재 위치에서 입력된 방향을 이용해 다음 위치를 찾는다 .</p> <p>3. 모든 방향에 대해 스텝 2 를 반복한다.</p>
Extensions	<p>1a. 입력한 방향의 수가 주사위 수와 일치하지 않는 경우 1. 올바르지 않은 입력이라고 표시한다.</p> <p>2. 해당 스텝을 다시 실행한다. 2a. 해당 방향에 셀이 없는 경우 1. 불가능한 방향이라고 표시한다. 2. 스텝 1 로 돌아간다.</p> <p>2b. 뒤로 가는 방향이고, End 셀에 도착한 플레이어가 있는 경우 1. 뒤로 갈 수 없다고 표시한다. 2. 스텝 1 로 돌아간다.</p> <p>2c. End 셀일 경우 1. 모든 스텝을 건너뛴다.</p>
Special Requirements	-
Technology and Data Variations List	-

Use Case Section	Comment
Frequency of Occurrence	턴마다 최대 한 번 일어난다.
Miscellaneous	-

j) UC207: Arrive at the End

플레이어의 말이 End 셀에 도착했다면 스코어를 정산한다.

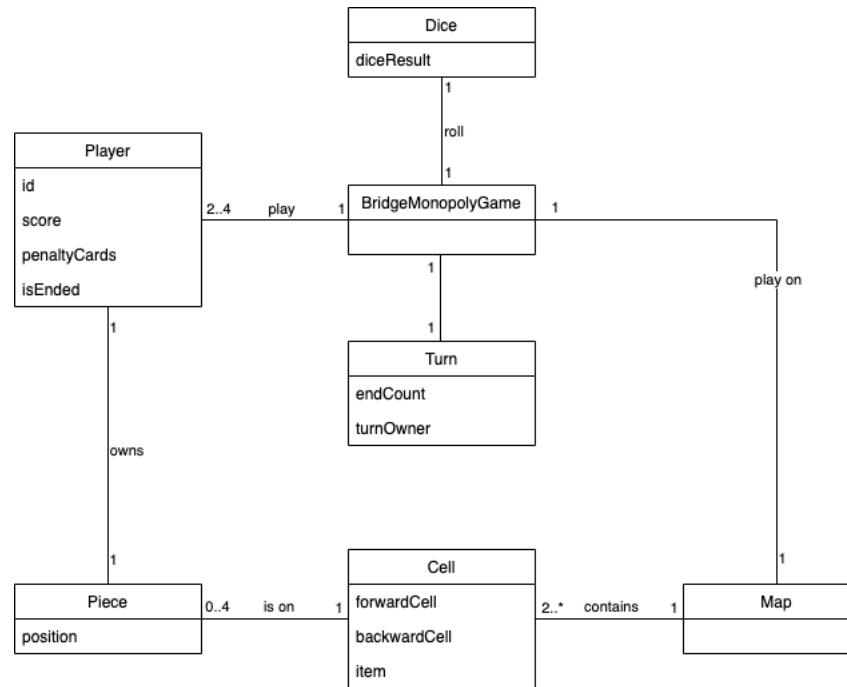
또한 이후 턴에서 모든 플레이어는 뒤로 이동할 수 없도록 이동 정책을 변경한다.

만약 1명을 제외한 모든 플레이어가 End 셀에 도착한 상태라면 게임을 종료한다.

Use Case Section	Comment
Use Case Name	Arrive at the End
Scope	System use case
Level	Subfunction
Primary Actor	Player
Stakeholders and Interests	Player : 도착지에 도달해서 스코어를 정산한다.
Preconditions	extend UC203 "Move Piece Turn" -> Player 의 Piece 가 End 셀로 이동한 경우
Success Guarantee	해당 Player 의 스코어를 정산하고 이후 다른 플레이어는 뒤로 갈 수 없다.
Main Success Scenario	<ol style="list-style-type: none"> 1. Player 가 도착한 순서에 따라 스코어를 추가한다. 2. Player 의 최종 점수를 표시한다. 3. Player 가 더 이상 턴을 가지지 않도록 변경한다. 4. 다음 턴부터 모든 Player 는 뒤로 이동할 수 없도록 변경한다. 5. 만약 1명을 제외한 모든 플레이어가 도착했다면 게임을 종료한다.

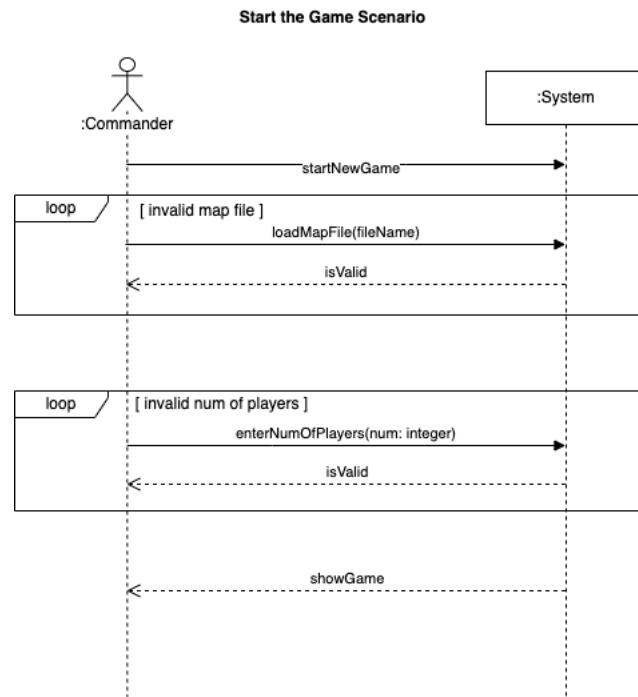
Use Case Section	Comment
Extensions	3a. 이미 End 셀에 도착한 Player 가 존재할 경우 1. 해당 스텝을 건너뛴다.
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	턴마다 최대 한 번 일어난다.
Miscellaneous	-

1.2.3. Domain model



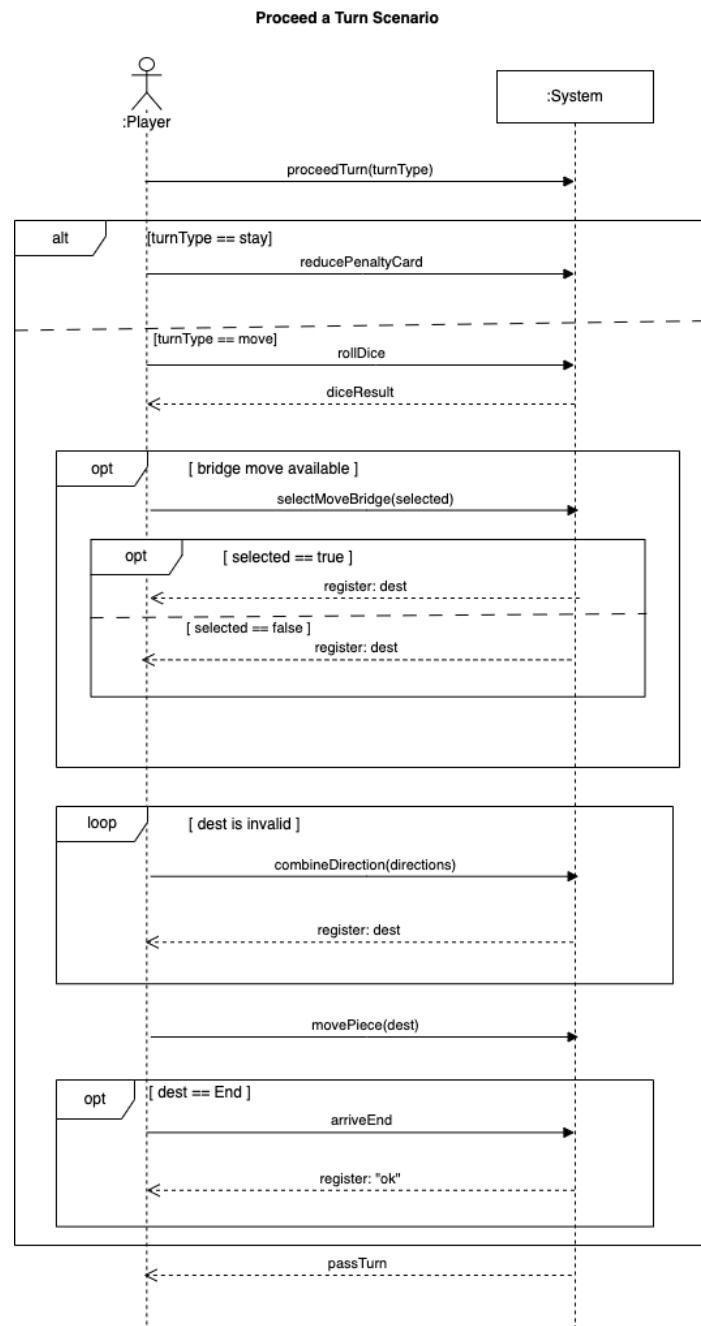
분석된 유즈 케이스를 바탕으로 명사형을 분석하고 시나리오에 필요한 attribute들을 추출한 결과이다. 다루고자 하는 비즈니스 모델에서 클래스로 책임을 가질 수 있는 것들이며, 세부 attribute는 이후 설계에서 다룬다.

1.2.4. System sequence diagram



System Sequence Diagram (SSD)는 Actor에 따라 두 가지로 구분할 수 있다.

첫 번째는 시스템 전체를 결정하는 Commander actor 와의 상호작용을 나타내는 UC101: Start the Game 의 전체적인 시나리오 흐름을 나타낸 것이다.



두 번째는 각 플레이어가 자신의 차례를 진행하는 UC201: Proceed a Turn의 전체적인 시나리오 흐름을 나타낸 것이다. UC201은 abstract use case로 UC202, UC203 중에 하나의 시나리오를 전반적으로 따르게 된다. 따라서 가장 먼저 proceedTurn 요청을 받을 때 이를 구분하고 해당하는 시나리오를 진행한다.

만약 Player actor 가 “stay” 를 결정했다면 다리 카드 한 장을 줄이는 reducePenaltyCard 를 실행하고 턴을 넘긴다.

actor 가 “move” 를 결정했다면 시스템은 rollDice 를 통해 주사위를 굴리기를 요청 받고 결과인 diceResult 를 보여준다. 만약 현재 말의 위치 및 이동 가능한 눈금을 이용해 다리 이동이 가능하다면 actor 는 selectMoveBridge 를 이용해 다리를 건널 것인지, 건너지 않을 것인지 결정할 수 있다. 다리를 건너는 상세 플로우는 설계에서 다루도록 한다.

이제 actor 는 남은 눈금으로 진행할 방향을 결정해서 combineDirection 을 통해 말의 이동을 요청할 수 있다. 그 결과가 이동 가능한 셀일 때까지 해당 플로우를 반복한다.

이동할 방향이 결정된 후에는 말을 이동한다. 만약 말이 도착한 위치가 End 셀이라면 UC207 을 수행하게 된다.

1.2.5. Operation Contracts

앞서 SSD에서 분석한 시나리오에서 주요한 operation 을 상세하게 분석하고자 한다.

a) startNewGame

Operation	startNewGame()
Cross References	Use cases: Start the Game
Preconditions	-
Postconditions	Turn 의 인스턴스인 turn 이 생성된다. (instance creation) Board 의 인스턴스인 board 가 생성된다. (instance creation) DirectionCombiner 의 인스턴스인 directionCombiner 가 생성된다. (instance creation) 맵 정보의 한 줄마다 Cell 의 인스턴스인 cell 이 생성된다. (instance creation) 각 cell 은 자신과 인접한 cell 을 가리킨다. (attribute modification) board 에 cell 을 추가한다. (attribute modification)

b) loadMapFile

Operation	loadMapFile(fileName: String)
Cross References	Use cases: Load Map
Preconditions	사용자는 맵 파일 목록이 없어도 파일 이름을 알고 있다.
Postconditions	Board 의 인스턴스인 board 가 생성되었다. (instance creation) 맵 정보의 한 줄마다 Cell 의 인스턴스인 cell 이 생성되었다. (instance creation) 각 cell 은 자신과 인접한 cell 을 가리키게 되었다. (attribute modification) board 에 cell 을 추가했다. (association formed)

c) enterNumberOfPlayers

Operation	enterNumberOfPlayers(num: integer)
Cross References	Use cases: Initialize Players
Preconditions	-
Postconditions	입력 개수만큼 Player 의 인스턴스 player 를 생성되었다. (instance creation) 각 player 는 생성된 순서별로 id 를 가지게 되었다. (attribute modification) turn 은 id 가 1 인 player 를 가리키도록 초기화 되었다. (attribute modification) player 수만큼 Piece 의 인스턴스인 piece 가 생성되었다. (instance creation) 각 player 는 piece 와 연결되었다. (association formed)

d) proceedTurn

Operation	proceedTurn(behavior: TurnType)
Cross References	Use cases: Proceed a Turn
Preconditions	player 는 End 셀에 도달하지 않았다.
Postconditions	turn 은 다음 player 를 가리키게 되었다. (attribute modification)

e) reducePenaltyCard

Operation	reducePenaltyCard()
Cross References	Use cases: Stay Turn
Preconditions	actor Player 는 turnType 으로 'stay'를 선택하였다.
Postconditions	player 의 penaltyCards 가 1 만큼 감소되었다. (attribute modification)

f) movePiece

Operation	movePiece(dest: Cell)
Cross References	Use cases: Move Piece Turn
Preconditions	actor Player 는 turnType 으로 'move'를 선택하였다.
Postconditions	player 와 연관된 piece 의 위치를 dest 로 변경한다. (attribute modification)

g) arriveEnd

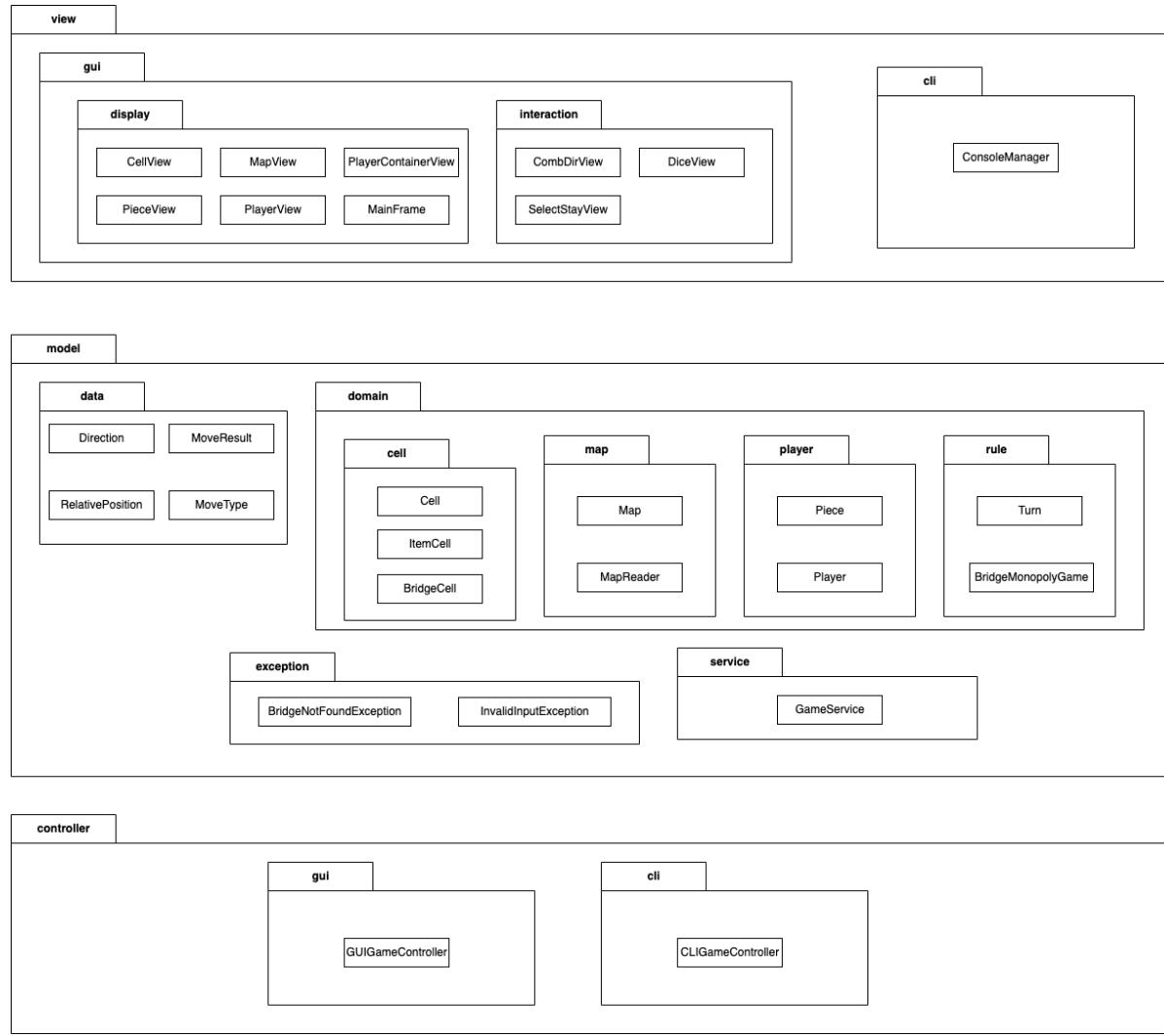
Operation	arriveEnd()
Cross References	Use cases: Move Piece Turn
Preconditions	dest 가 End 셀이다.

Operation	arriveEnd()
Postconditions	player 는 더 이상 턴을 가지지 않는다. (attribute modification) directionCombiner 는 앞으로 뒤로 가는 입력을 받을 수 없다. (attribute modification)

2. 설계

분석된 요구 사항을 바탕으로 시스템을 설계하고자 한다.

2.1. Logical Architecture



다음은 아키텍처 구조를 패키지 디아이어그램으로 나타낸 것이다. 아키텍처의 설계는 MVC (Model-View-Controller) 패턴을 적용하였다. model의 domain 영역은 앞서 요구사항 분석에서의 Domain model을 나타낸다. 여기서는 비즈니스 모델을 다룬다. Cell은 일반 셀인 ItemCell과 다리가 연결된 BridgeCell로 나뉜다. 이에 대한 상세 설명은 클래스 디아이어그램에서 다룬다.

전체적인 게임의 진행은 domain 영역의 BridgeMonopolyGame에서 다룬다. 여기서 핵심 설계는 domain 계층이 UI 계층의 변화에 영향을 받지 않도록 service를 구현하는 것이다. Actor 와의 상호작용이 필요한 부분은 모두 GameService 인터페이스를 통해서 진행되며, 각 UI에 해당하는 Controller는 GameService의 메소드를 구현하여 Actor 와의 상호작용을 지원해야 한다.

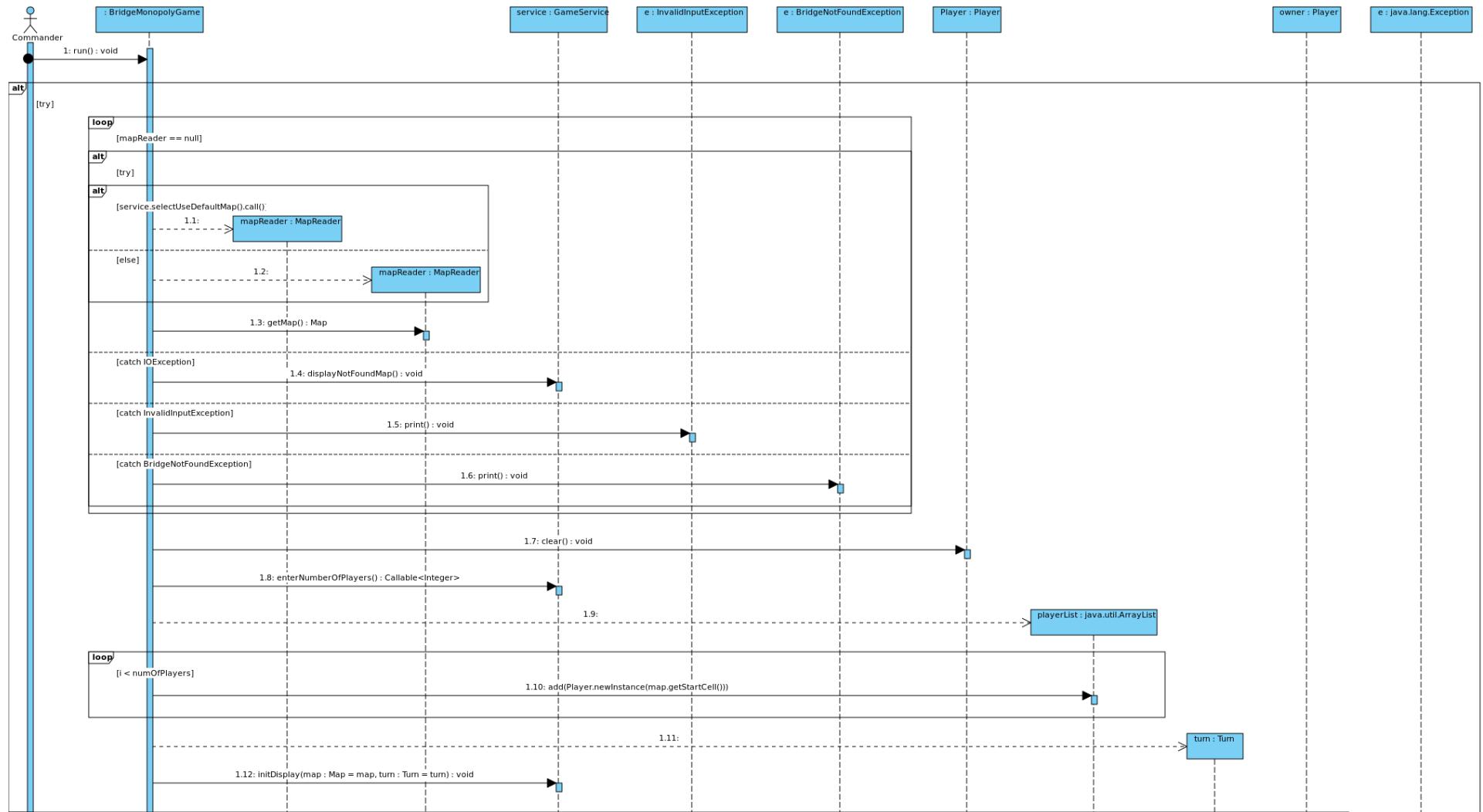
또한 각 계층에서도 보다 상세하게 관심사를 분리하였으며, 단일 콘솔에서 진행되는 CLI 환경과 달리 여러 뷰의 상호작용이 필요한 GUI에서는 view를 단순히 결과를 표시하는 display 영역과 Actor 의 입력을 다루는 interaction 영역으로 나눈다.

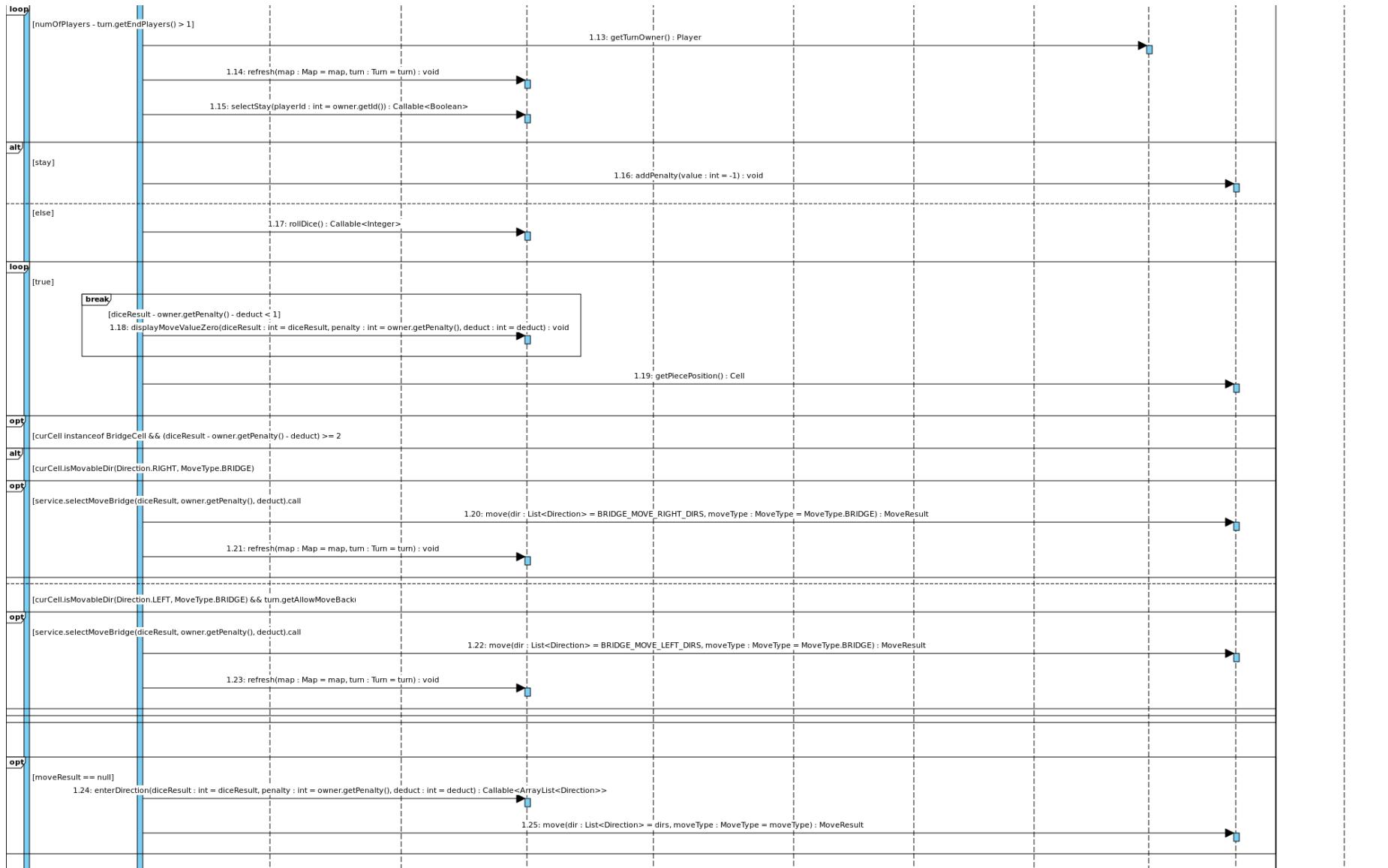
2.2. Design Sequence Diagram

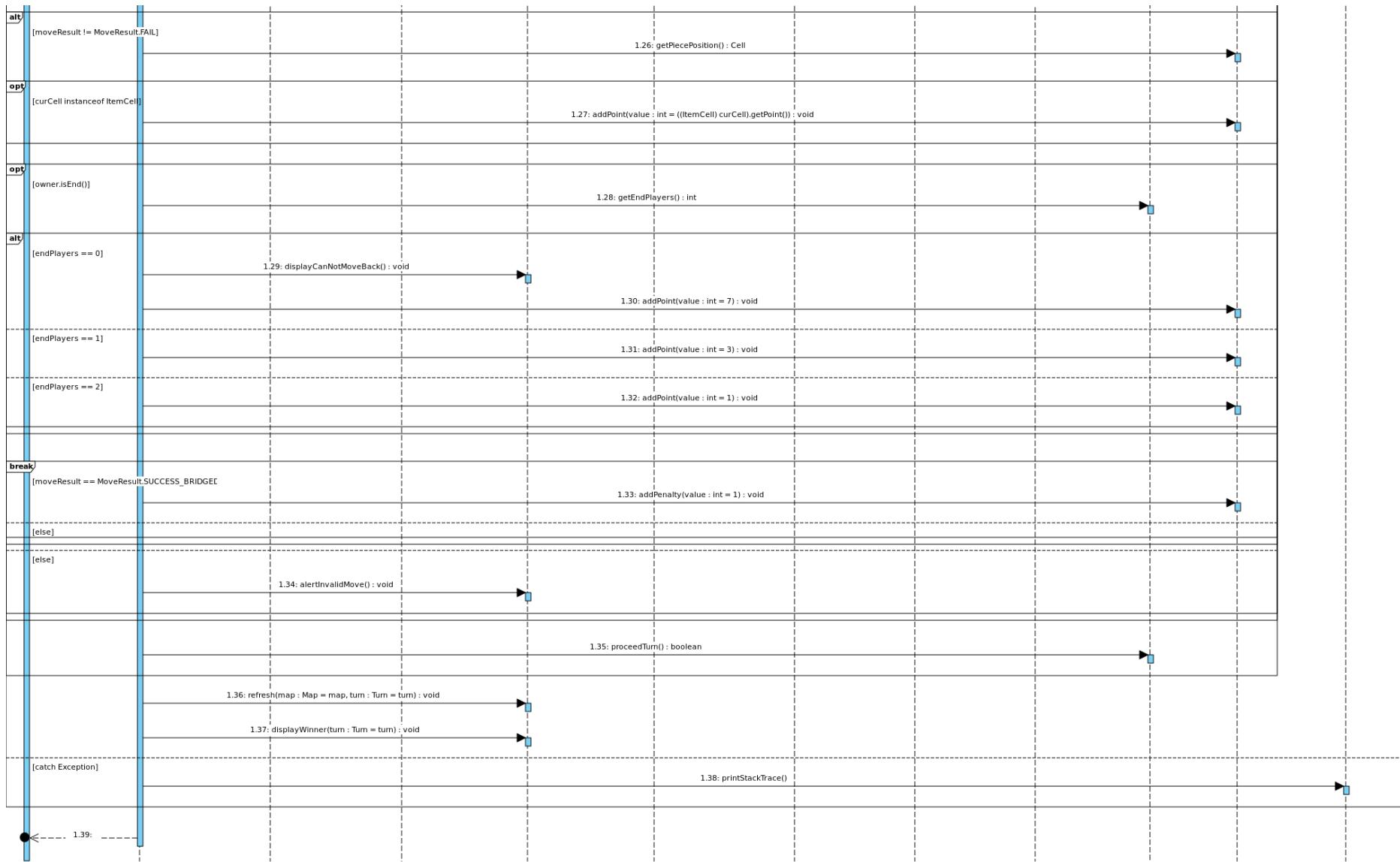
a) 게임 룰 [UC101: Start the Game]

게임의 룰과 전반적인 진행에 관한 책임은 BridgeMonopolyGame 클래스에 있다. 게임을 진행하는데 있어서 CLI, GUI 등 UI 계층에 종속되지 않고 게임 진행에 필요한 플레이어의 선택 및 게임 상황 표시를 GameService 인터페이스를 통해 구현하도록 설계한다. BridgeMonopolyGame 클래스에서 게임을 시작할 때 사용하는 메소드 run()에서는 GameService 및 model 계층의 클래스들의 상호작용을 통해 게임 룰이 진행된다.

Actor 에 의해서 게임이 실행될 때 호출되는 run() 메소드의 설계에 대한 시퀀스 다이어그램은 다음과 같다.



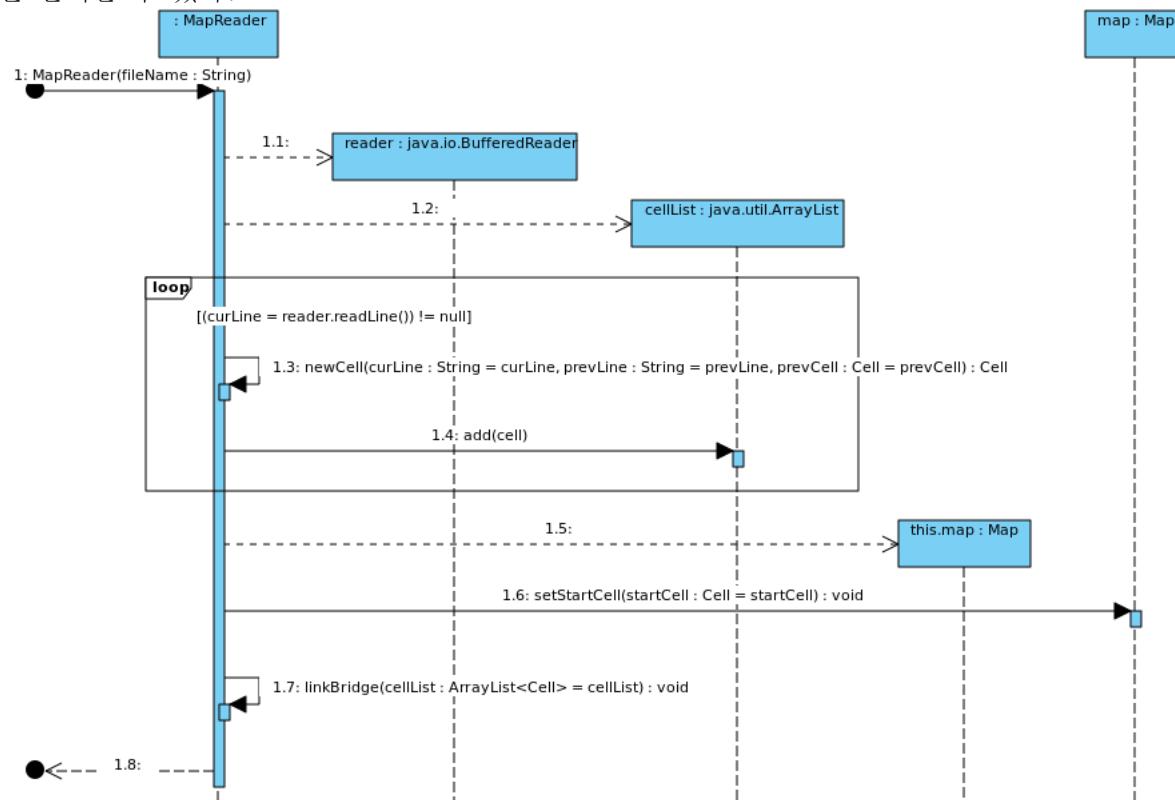




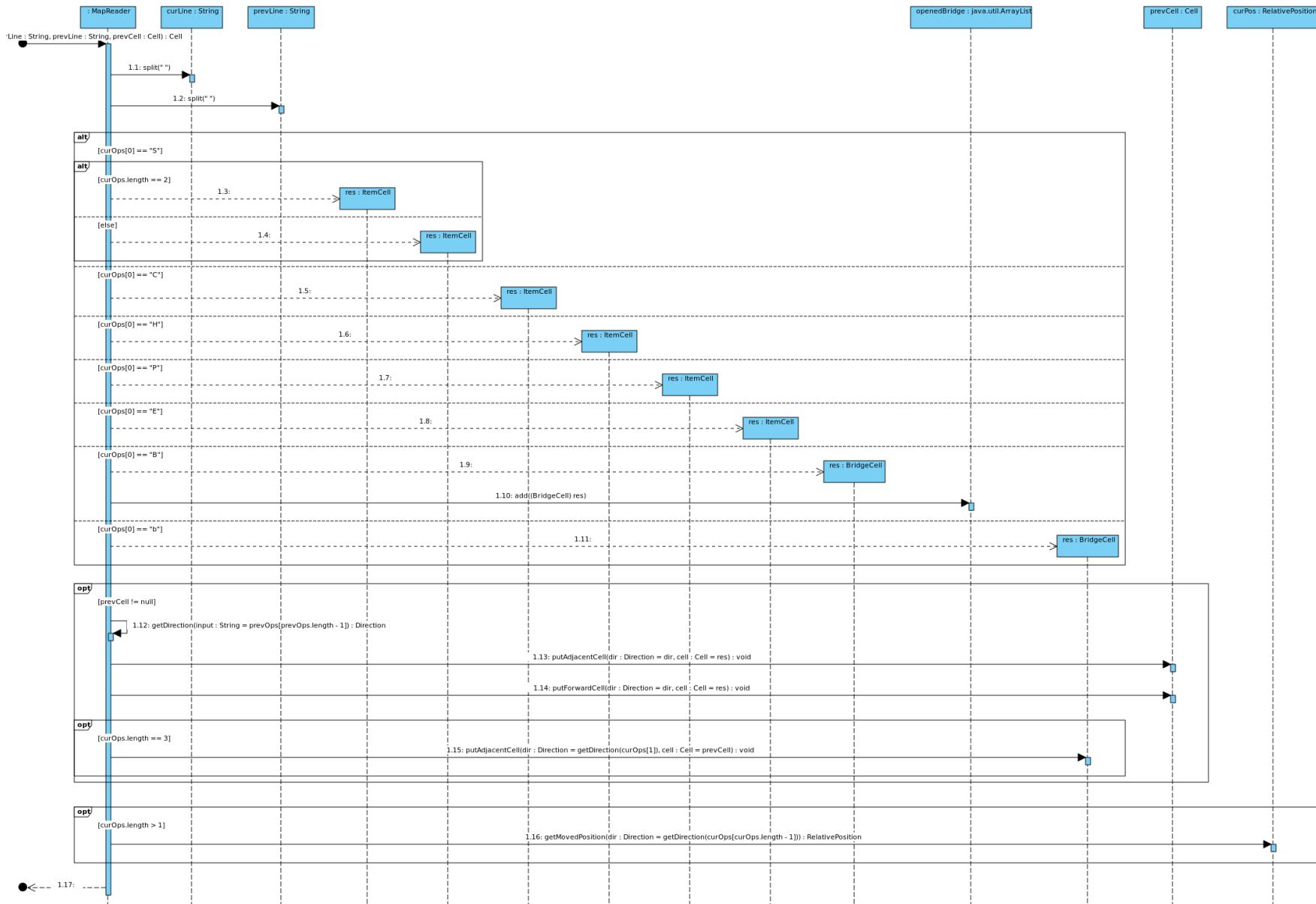
게임의 전반적인 흐름에 대한 설계는 끝났으므로 domain 영역의 주요 시퀀스에 대한 설계를 해본다.

b) 맵 불러오기 [UC102: Load Map]

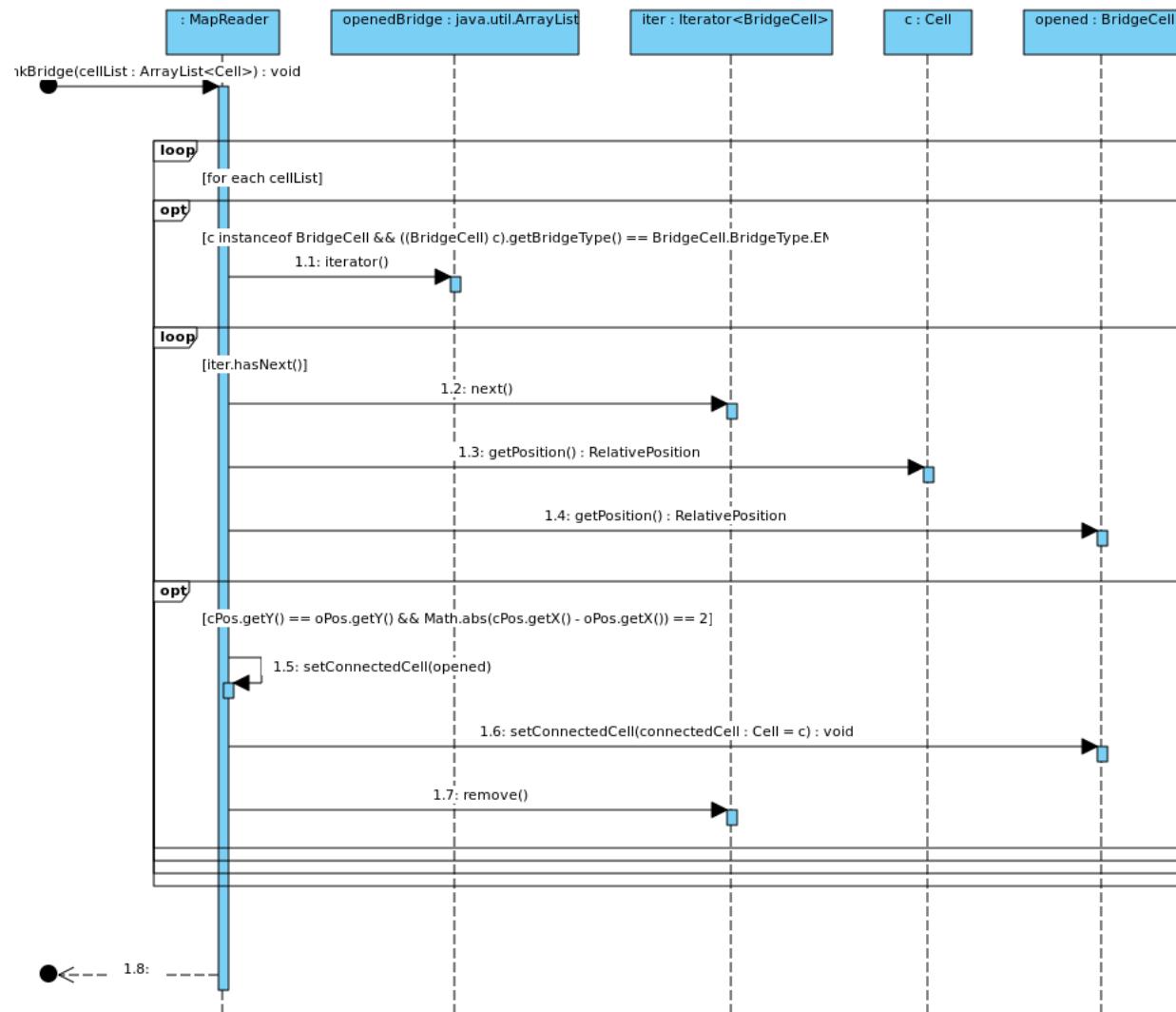
화면에 맵을 표시하고, 플레이어의 말을 움직이기 위해서는 맵 정보가 필요하다. 파일로 제시된 맵을 메모리에 Cell 클래스의 형태로 불러온다. 이 때 이동 가능한 셀에 대한 정보까지 처리가 완료되기 때문에 이후 플레이어 이동에서 별도의 계산을 할 필요 없이 Cell 클래스의 메소드를 통해 이동에 필요한 정보를 불러올 수 있다.



MapReader 클래스가 생성되면 전달 받은 파일의 이름을 이용해 맵 파일을 읽는다. 한 줄마다 newCell 메소드를 이용해서 셀을 생성한다.

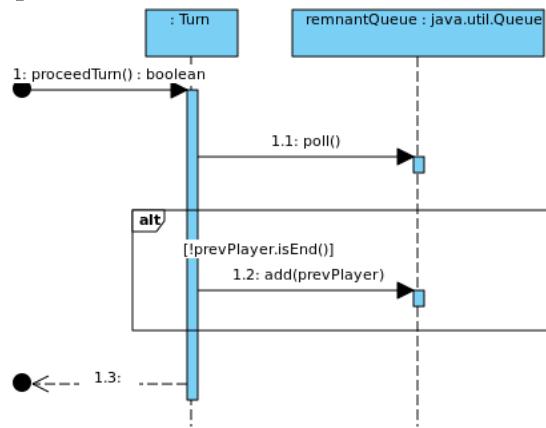


유즈 케이스 명세에서의 시나리오대로 셀의 종류를 구분하고, 이전 셀 정보와 현재 셀 정보를 이용해서 그래프 형태로 서로 연결한다.



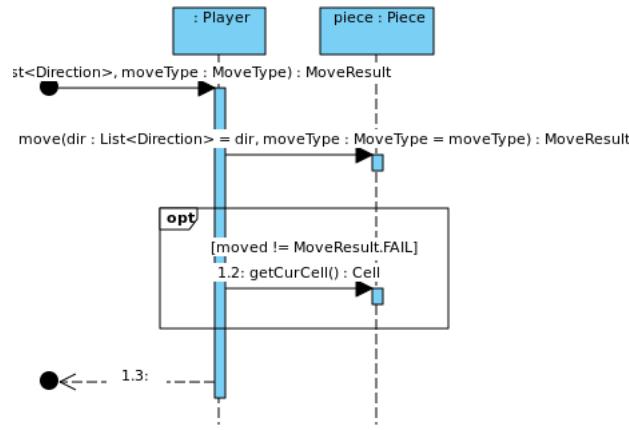
Bridge 셀은 맵 데이터에서 직접적인 연결 데이터를 표시하지 않기 때문에, 모든 셀을 생성한 후에 linkBridge 메소드를 이용해서 처리한다. 같은 y 축에 있으며 x 차이가 2인 ‘B’ 셀과 ‘b’ 셀을 서로 연결한다. 일반적인 셀은 adjacentSet으로 연결하지만, 브릿지 연결은 따로 분리해서 저장한다. Bridge 셀은 브릿지 시작 셀 또는 끝 셀의 정보가 있기 때문에 연결된 방향은 저장할 필요가 없다.

c) 다음 턴 진행 [UC201: Proceed a Turn]



턴을 다음 플레이어에게 넘기는 시퀀스는 간단하다. Turn 클래스가 관련 책임을 맡으며, Player 정보를 가지고 있기 때문에 아직 턴을 진행할 수 있는 플레이어들을 큐로 보관한다. 아직 End 셀에 도착하지 않은 플레이어는 큐에 다시 삽입하여 다음 순서를 기다리게 하고, 도착한 플레이어는 큐에서 제거되어 이후 턴을 받지 않는다.

d) 플레이어 이동 [UC203: Move Piece Turn]



플레이어를 원하는 위치로 이동시키기 위해서는 `Direction`의 `List`를 입력한다. 또한 `MoveType`을 이용해서 인접한 셀 이동/앞으로만 이동/브릿지 이동을 구분한다. 해당 위치로 이동한 결과는 `MoveResult`로 반환된다. 실패할 경우 `FAIL`이 반환될 것이다. 이동에 성공하면 `Player` 객체가 소유한 `Piece`가 가리키는 셀의 위치가 변경된다.

2.3. Design Class

2.3.1. Design Class

앞서 설계한 시퀀스 다이어그램을 기반으로 클래스를 설계하고자 한다.

각 영역의 클래스에서 설계의 핵심 고려사항은 다음과 같다.

- data
 - `RelativePosition`
START 셀로부터 상대적인 좌표를 나타낸다. y 좌표는 START 셀에서 위로 이동한 셀 수, x 좌표는 START 셀에서 오른쪽으로 이동한 셀 수이다. 방향 정보 `Direction`을 이용한 상대 좌표 계산 책임 및 오버라이딩을 통해 x, y를 이용해 `equal`을 반환하는 책임도 맡는다.
- domain
 - `Cell`
한 셀이 가지는 정보를 나타낸다. 인접한 셀과 특정 방향으로의 이동 가능 여부 판단의 책임을 갖는다.
 - `ItemCell`
일반적인 셀을 다룬다. 도구 아이템의 종류 및 해당하는 점수, START, END 셀 여부의 책임을 갖는다.
 - `BridgeCell`
브릿지 셀을 다룬다. 인접 셀 뿐만 아니라 다리로 연결된 셀의 정보를 가지며 다리를 통한 이동의 책임을 갖는다.

- Map

전체 셀의 정보를 나타낸다. 그래프로 표현된 셀의 정보와 이를 화면에 출력하기 위한 이차원 배열 형태의 재가공 책임을 갖는다.
- MapReader

파일로부터 맵 정보를 불러온다.

생성자 오버로딩을 통해 기본 맵인 default.map 과 지정 맵 불러오기를 모두 지원한다.
- Piece

플레이어가 소유한 말을 나타낸다. 말의 위치와 이동에 관한 책임을 맡는다.
- Player

플레이어 정보를 나타낸다. 플레이어의 말을 제어하는 책임을 맡는다.

중복되는 id를 가지는 플레이어가 생기는 것을 방지하기 위해서 **팩토리 패턴**을 사용한다. 유일한 private 생성자를 통해서만 생성되며, 정적 메소드 newInstance를 통해서 객체를 생성할 수 있다.
- Turn

전체 플레이어를 관리하는 책임을 갖는다. 플레이어의 행동이 끝나면 다음 플레이어의 턴으로 변경하도록 지원한다. END 셀에 도착한 플레이어는 더 이상 턴을 진행하지 않도록 한다. 게임이 종료된 후 승리한 플레이어를 계산하는 책임도 갖는다.
- BridgeMonopolyGame

게임 전반적인 로직을 담당한다. **인터페이스를 통해 게임을 진행**하기 때문에 컨트롤러는 게임 룰에 대해 신경 쓰지 않아도 된다. 또한 다양한 플랫폼으로 확장하더라도 룰을 재사용할 수 있다.
- service
 - GameService

게임 로직 진행 중 사용자에게 표시하거나, 입력 받는 모든 책임을 갖는다.

게임 룰과 사용자 인터페이스를 분리하는 역할을 한다.

단순히 결과를 표시하는 메소드는 void 타입이며, 사용자의 입력을 얻는 메소드는 UI 환경에 따라 **스레드 기반 처리**가 필요할 수 있으므로 Callable을 반환하도록 한다. BridgeMonopolyGame에서는 Callable.call() 메소드를 통해 새로운 메소드에서 사용자의 입력을 처리하고 그 결과를 반환할 수 있으며, 반환 지연이 필요한 경우 스레드를 블락시켜서 구현할 수 있다.
- controller
 - GUIGameController

자바 스윙 기반으로 GameService 인터페이스를 구현하여 사용자에게 GUI를 제공한다.

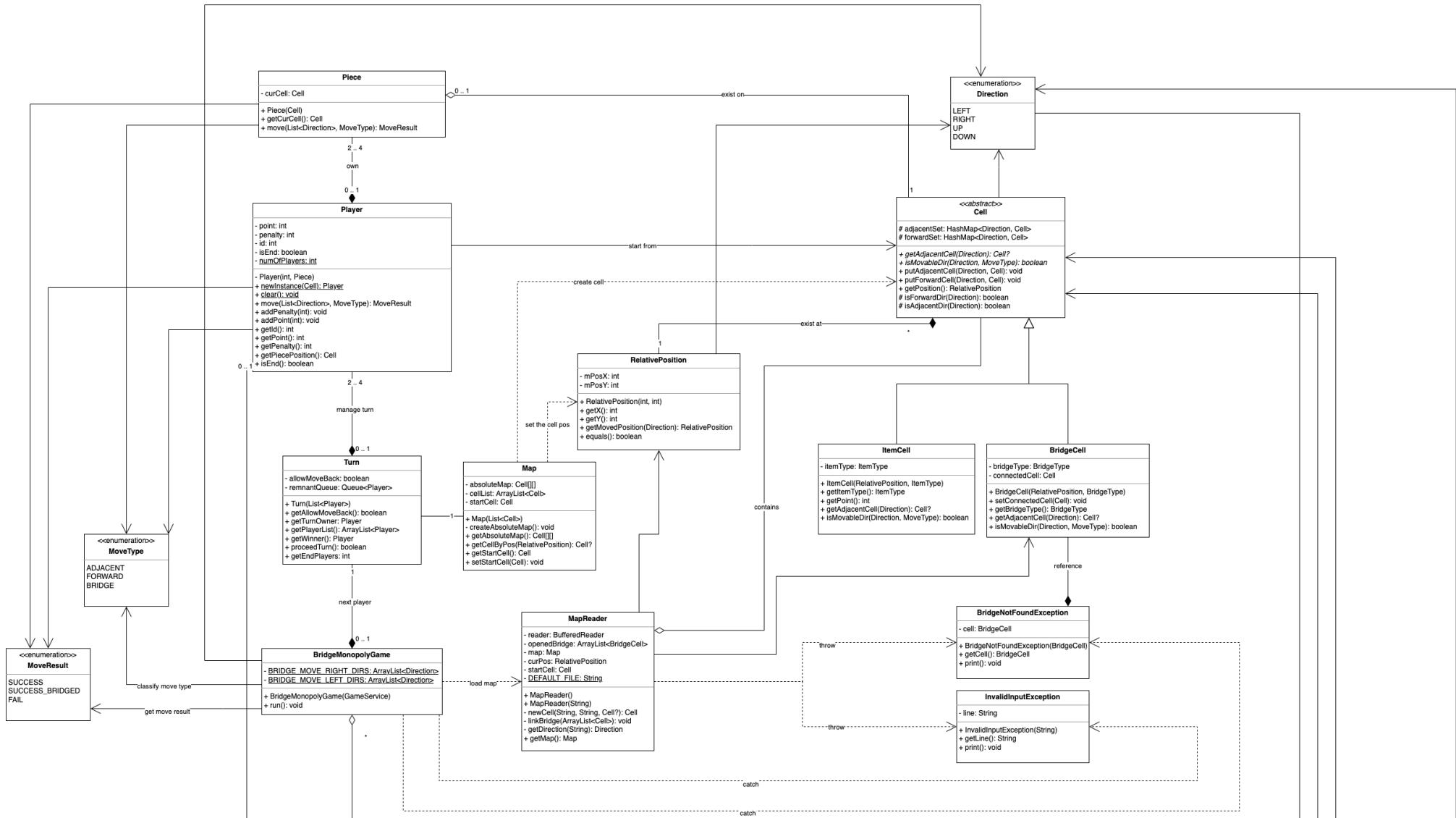
플레이어가 선택을 할 때까지 결과 반환을 지연시키기 위해서 Callable을 상속한 WaitCall을 inner class로 갖는다. 이를 이용해 이벤트 리스너에 의해 선택이 포착되기 전까지 해당 스레드를 블락시킨다. 이벤트가 발생하면 스레드를 깨우고 플레이어가 선택한 결과를 반환한다.
- CLIGameController

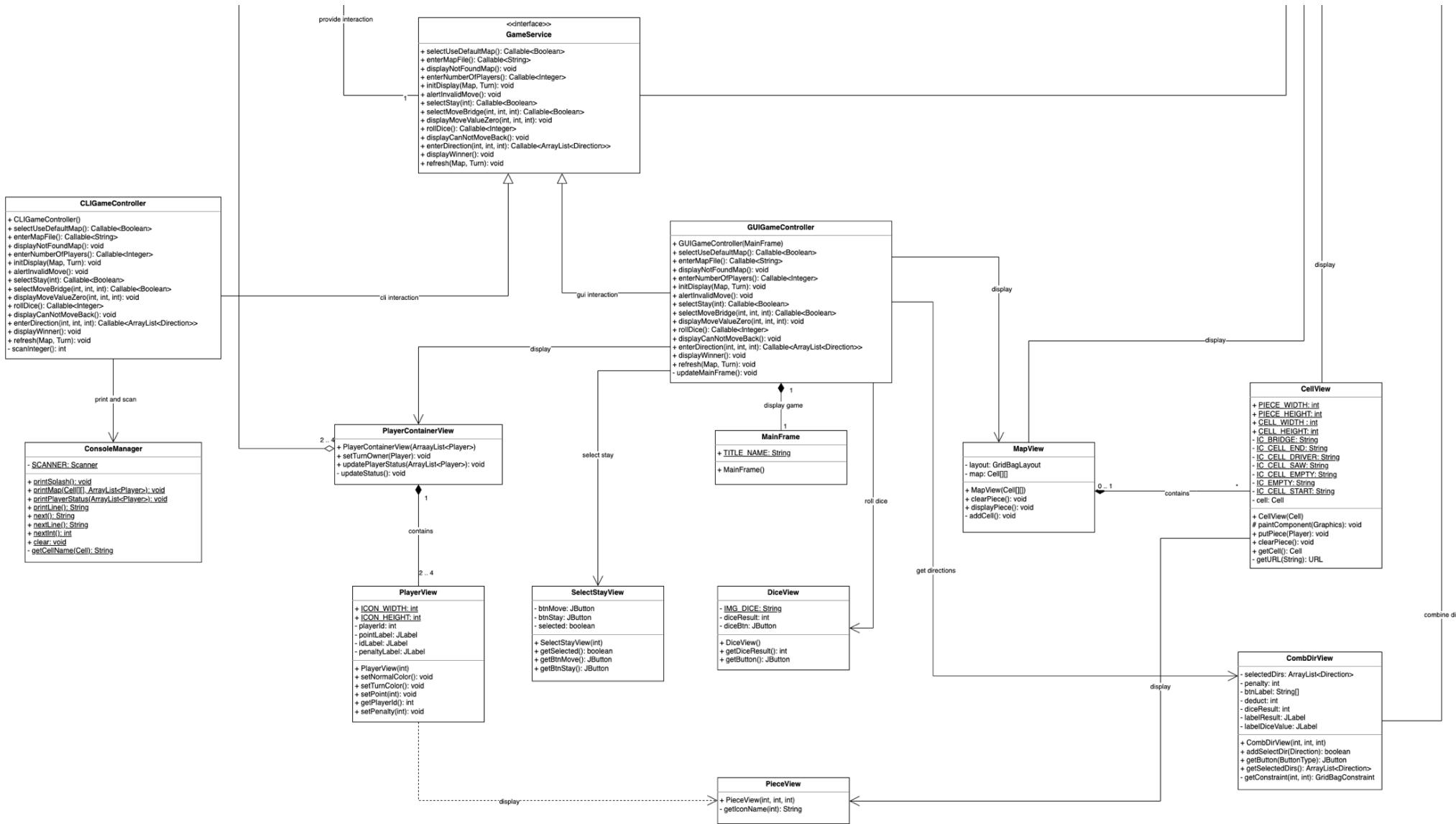
콘솔 기반으로 GameService 인터페이스를 구현하여 사용자에게 CLI를 제공한다.

콘솔 기반의 입력을 처리하는 java.util.Scanner는 입력이 있을 때까지 기다리기 때문에 별도의 스레드 블락은 사용하지 않는다.

2.3.2. Design Class Diagram

클래스의 attribute, operation을 정의하고 클래스 간의 연관관계를 나타낸 것이다.





3. 구현

```
public abstract class Cell {  
  
    protected final RelativePosition position;  
  
    // Cells connected in the forward direction  
    protected final HashMap<Direction, Cell> forwardSet = new HashMap<>();  
  
    // Cells connected in the backward direction  
    protected final HashMap<Direction, Cell> adjacentSet = new HashMap();  
  
    protected Cell(RelativePosition position) {  
        this.position = position;  
    }  
  
    protected boolean isAdjacentDir(Direction dir) {  
        return this.adjacentSet.get(dir) != null;  
    }  
  
    protected boolean isForwardDir(Direction dir) {  
        return this.forwardSet.get(dir) != null;  
    }  
  
    public abstract boolean isMovableDir(Direction dir, MoveType moveType);  
  
    public abstract Cell getAdjacentCell(Direction dir);  
  
    public void putAdjacentCell(Direction dir, Cell cell) {  
        this.adjacentSet.put(dir, cell);  
    }  
  
    public void putForwardCell(Direction dir, Cell cell) {  
        this.forwardSet.put(dir, cell);  
    }  
  
    public RelativePosition getPosition() {  
        return this.position;  
    }  
}
```

Cell

```
public class ItemCell extends Cell {  
  
    private ItemType itemType;  
  
    public ItemCell(RelativePosition position, ItemType itemType) {
```

```
        super(position);
        this.itemType = itemType;
    }

    public ItemType getItemType() {
        return this.itemType;
    }

    @Override
    public boolean isMovableDir(Direction dir, MoveType moveType) {
        if (itemType == ItemType.END)
            return true;
        if (moveType == MoveType.ADJACENT)
            return isAdjacentDir(dir);
        // forward move
        else
            return isForwardDir(dir);
    }

    @Override
    public @Nullable Cell getAdjacentCell(Direction dir) {
        if (itemType == ItemType.END)
            return this;
        return this.adjacentSet.get(dir);
    }

    @Override
    public String toString() {
        return itemType + " " + position.toString();
    }

    public int getPoint() {
        switch (itemType) {
            case HAMMER:
                return 2;
            case SAW:
                return 3;
            case PHILIPS_DRIVER:
                return 1;
            default:
                return 0;
        }
    }

    public enum ItemType {
        START, END,                      // base cell
        EMPTY,                           // empty cell
        HAMMER, SAW, PHILIPS_DRIVER // item cell
    }
}
```

```
        }
    }
}

```

ItemCell

```
public class BridgeCell extends Cell {

    private final BridgeType bridgeType;

    private Cell connectedCell;

    public BridgeCell(RelativePosition position, BridgeType bridgeType) {
        super(position);
        this.bridgeType = bridgeType;
    }

    @Override
    public boolean isMovableDir(Direction dir, MoveType moveType) {
        switch (moveType) {
            case ADJACENT:
                return isAdjacentDir(dir);
            case FORWARD:
                return isForwardDir(dir);
            case BRIDGE:
                if (bridgeType == START && dir == Direction.RIGHT || bridgeType == END && dir == Direction.LEFT)
                    return true;
        }
        return false;
    }

    @Override
    public @Nullable Cell getAdjacentCell(Direction dir) {
        if (this.bridgeType == START && dir == Direction.RIGHT || this.bridgeType == END && dir == Direction.LEFT) {
            return connectedCell;
        }
        return this.adjacentSet.get(dir);
    }

    @Override
    public String toString() {
        return bridgeType + " " + position.toString();
    }

    public void setConnectedCell(Cell cell) {
        this.connectedCell = cell;
    }

    public BridgeType getBridgeType() {

```

```
        return this.bridgeType;
    }

    public enum BridgeType {
        START, END, BRIDGE
    }
}
```

BridgeCell

```
public class Map {

    private final static int POS_INF = 23456789;
    private final static int NEG_INF = -23456789;

    private Cell startCell;

    private final ArrayList<Cell> cellList = new ArrayList<>();

    /*
        Used when displaying in gui and cli
        (0, 0) is the leftmost and bottommost position
     */
    private Cell[][] absoluteMap;

    public Cell getStartCell() {
        return this.startCell;
    }

    public void setStartCell(Cell cell) {
        this.startCell = cell;
    }

    public Map(final List<Cell> cellList) {
        this.cellList.addAll(cellList);

        createAbsoluteMap();
    }

    private void createAbsoluteMap() {
        if (cellList.isEmpty())
            return;

        int minX = POS_INF, minY = POS_INF;
        int maxX = NEG_INF, maxY = NEG_INF;

        // get arrange of relative position
        for (Cell cell : cellList) {
```

```

        int x = cell.getPosition().getX();
        int y = cell.getPosition().getY();

        if (x > maxX)
            maxX = x;
        if (x < minX)
            minX = x;
        if (y > maxY)
            maxY = y;
        if (y < minY)
            minY = y;
    }

    absoluteMap = new Cell[maxY - minY + 1][maxX - minX + 1];

    // load to absolute map
    for (Cell cell : cellList) {
        RelativePosition pos = cell.getPosition();
        absoluteMap[maxY - pos.getY()][pos.getX() - minX] = cell;
    }

    // generate bridge path
    for (int y = 0; y < absoluteMap.length; y++) {
        for (int x = 0; x < absoluteMap[y].length; x++) {
            // bridge path information
            if (absoluteMap[y][x] instanceof BridgeCell && ((BridgeCell) absoluteMap[y][x]).getBridgeType() ==
BridgeCell.BridgeType.START) {
                int lookX = x + 1;
                while (lookX < absoluteMap[y].length && absoluteMap[y][lookX] == null) {
                    absoluteMap[y][lookX] = new BridgeCell(null, BridgeCell.BridgeType.BRIDGE);
                    lookX++;
                }
            }
        }
    }
}

public Cell[][] getAbsoluteMap() {
    return this.absoluteMap;
}

public @Nullable Cell getCellByPos(RelativePosition pos) {
    for (Cell cell : cellList) {
        if (cell.getPosition().equals(pos))
            return cell;
    }
    return null;
}

```

```
}
```

Map

```
public class Piece {  
  
    private Cell curCell;  
  
    public Piece(Cell startCell) {  
        this.curCell = startCell;  
    }  
  
    public MoveResult move(@NotNull final List<Direction> dir, MoveType moveType) {  
        Cell cur = curCell;  
        MoveResult res = MoveResult.SUCCESS;  
  
        for (int i = 0; i < dir.size(); i++) {  
            if (cur.isMovableDir(dir.get(i), moveType)) {  
                Cell dest = cur.getAdjacentCell(dir.get(i));  
  
                if (moveType == MoveType.BRIDGE && cur instanceof BridgeCell && dest instanceof BridgeCell) {  
                    if (!(i+1 < dir.size()))  
                        return MoveResult.FAIL;  
                    if (!cur.isMovableDir(dir.get(i), MoveType.BRIDGE))  
                        return MoveResult.FAIL;  
                    i++;  
                    res = MoveResult.SUCCESS_BRIDGED;  
                }  
                cur = dest;  
            }  
            else  
                return MoveResult.FAIL;  
        }  
  
        // move successfully  
        this.curCell = cur;  
        return res;  
    }  
  
    public Cell getCurCell() {  
        return curCell;  
    }  
}
```

Piece

```
public class Player {

    private static int numPlayers = 0;

    private final int id;

    private int point = 0;

    private int penalty = 0;

    private final Piece piece;

    private boolean isEnd = false;

    private Player(int id, Piece piece) {
        this.id = id;
        this.piece = piece;
    }

    public static @NotNull Player newInstance(Cell startCell) {
        Player player = new Player(++numPlayers, new Piece(startCell));
        return player;
    }

    public static void clear() {
        numPlayers = 0;
    }

    public int getId() {
        return this.id;
    }

    public MoveResult move(@NotNull final List<Direction> dir, MoveType moveType) {
        MoveResult moved = piece.move(dir, moveType);

        // when moved
        if (moved != MoveResult.FAIL) {
            Cell curCell = piece.getCurCell();
            if (curCell instanceof ItemCell && ((ItemCell) curCell).getItemType() == ItemCell.ItemType.END)
                this.isEnd = true;
        }

        return moved;
    }

    public boolean isEnd() {
        return this.isEnd;
    }
}
```

```
    public void addPoint(int value) {
        this.point += value;
    }

    public int getPoint() {
        return this.point;
    }

    public void addPenalty(int value) {
        this.penalty += value;
        if (penalty < 0)
            penalty = 0;
    }

    public int getPenalty() {
        return this.penalty;
    }

    public Cell getPiecePosition() {
        return this.piece.getCurCell();
    }
}
```

Player

```
public class Turn {

    private boolean allowMoveBack = true;

    private final ArrayList<Player> players;

    private final Queue<Player> remnantQueue = new LinkedList<>();

    private int endPlayers = 0;

    public Turn(@NotNull List<Player> players) {
        this.players = new ArrayList(players);
        this.remnantQueue.addAll(players);
    }

    public boolean proceedTurn() {

        if (remnantQueue.isEmpty())
            return false;

        Player prevPlayer = remnantQueue.poll();
        if (!prevPlayer.isEnd())
            remnantQueue.add(prevPlayer);
    }
}
```

```

        else {
            allowMoveBack = false;
            endPlayers++;
        }

        return true;
    }

    public @Nullable Player getTurnOwner() {
        if (remnantQueue.isEmpty())
            return null;
        return remnantQueue.peek();
    }

    public boolean getAllowMoveBack() {
        return this.allowMoveBack;
    }

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

    public @NotNull Player getWinner() {
        int maxScore = -1;
        Player winner = null;

        for (Player p : players) {
            if (maxScore < p.getPoint()) {
                maxScore = p.getPoint();
                winner = p;
            }
        }

        return winner;
    }

    public int getEndPlayers() {
        return this.endPlayers;
    }
}

```

Turn

```

public class BridgeMonopolyGame {
    private final GameService service;

```

```

private @Nullable MapReader mapReader;

private Map map;

private Turn turn;

private static final ArrayList<Direction> BRIDGE_MOVE_RIGHT_DIRS = new ArrayList<>(Arrays.asList(Direction.RIGHT,
Direction.RIGHT));

private static final ArrayList<Direction> BRIDGE_MOVE_LEFT_DIRS = new ArrayList<>(Arrays.asList(Direction.LEFT, Direction.LEFT));

public BridgeMonopolyGame(GameService service) {
    this.service = service;
}

public void run() {
    try {

        while (mapReader == null) {
            try {
                if (service.selectUseDefaultMap().call())
                    mapReader = new MapReader(service.enterMapFile().call());
                else
                    mapReader = new MapReader();

                map = mapReader.getMap();
            } catch (IOException e) {
                service.displayNotFoundMap();
            } catch (InvalidInputException e) {
                e.print();
            } catch (BridgeNotFoundException e) {
                e.print();
            }
        }

        // create map by absolute position

        // initialize players
        Player.clear();
        int numOfPlayers = service.enterNumberOfPlayers().call();
        ArrayList<Player> playerList = new ArrayList<>();
        for (int i = 0; i < numOfPlayers; i++) {
            playerList.add(Player.newInstance(map.getStartCell()));
        }

        // initialize turn
        turn = new Turn(playerList);
    }
}

```

```

        service.initDisplay(map, turn);

        // run turn
        Player owner;
        while (numOfPlayers - turn.getEndPlayers() > 1) {
            owner = turn.getTurnOwner();
            service.refresh(map, turn);

            boolean stay = service.selectStay(owner.getId()).call();

            // stay turn
            if (stay) {
                owner.addPenalty(-1);
            }
            // move turn
            else {
                // roll dice
                int diceResult = service.rollDice().call();
                // combine direction
                ArrayList<Direction> dirs;

                int deduct = 0;

                while (true) {

                    if (diceResult - owner.getPenalty() - deduct < 1) {
                        service.displayMoveValueZero(diceResult, owner.getPenalty(), deduct);
                        break;
                    }

                    Cell curCell = owner.getPiecePosition();
                    MoveResult moveResult = null;

                    // select bridge move
                    if (curCell instanceof BridgeCell && (diceResult - owner.getPenalty() - deduct) >= 2) {
                        if (curCell.isMovableDir(Direction.RIGHT, MoveType.BRIDGE)) {
                            if (service.selectMoveBridge(diceResult, owner.getPenalty(), deduct).call()) {
                                moveResult = owner.move(BRIDGE_MOVE_RIGHT_DIRS, MoveType.BRIDGE);
                                deduct += 2;
                                service.refresh(map, turn);
                            }
                        } else if (curCell.isMovableDir(Direction.LEFT, MoveType.BRIDGE) && turn.getAllowMoveBack()) {
                            if (service.selectMoveBridge(diceResult, owner.getPenalty(), deduct).call()) {
                                moveResult = owner.move(BRIDGE_MOVE_LEFT_DIRS, MoveType.BRIDGE);
                                deduct += 2;
                                service.refresh(map, turn);
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        }

        // not bridge moved
        if (moveResult == null) {
            dirs = service.enterDirection(diceResult, owner.getPenalty(), deduct).call();
            MoveType moveType = MoveType.ADJACENT;

            // not allow move back
            if (!turn.getAllowMoveBack())
                moveType = MoveType.FORWARD;

            moveResult = owner.move(dirs, moveType);
        }

        // move successfully
        if (moveResult != MoveResult.FAIL) {
            curCell = owner.getPiecePosition();

            // get point
            if (curCell instanceof ItemCell) {
                owner.addPoint(((ItemCell) curCell).getPoint());
            }

            // player end
            if (owner.isEnd()) {
                int endPlayers = turn.getEndPlayers();

                if (endPlayers == 0) {
                    service.displayCanNotMoveBack();
                    owner.addPoint(7);
                }
                else if (endPlayers == 1)
                    owner.addPoint(3);
                else if (endPlayers == 2)
                    owner.addPoint(1);
            }

            // get penalty
            if (moveResult == MoveResult.SUCCESS_BRIDGED)
                owner.addPenalty(1);
            else
                break;
        }
        // can not move
        else
            service.alertInvalidMove();
    }
}

```

```
        }
        turn.proceedTurn();
    }
    service.refresh(map, turn);
    service.displayWinner(turn);
} catch (Exception e) {
    e.printStackTrace();
}
}
}
```

BridgeMonopolyGame

```
public interface GameService {

    /*
     * Decide whether to use the default map.
     */
    Callable<Boolean> selectUseDefaultMap();
    /*
     * Select map file name.
     */
    Callable<String> enterMapFile();

    /*
     * It outputs that the input file does not exist and proceeds to the default map.
     */
    void displayNotFoundMap();

    /*
     * Input the number of players to play the game.
     */
    Callable<Integer> enterNumberOfPlayers();

    /*
     * Initializes the elements of the screen.
     */
    void initDisplay(Map map, Turn turn);

    void refresh(Map map, Turn turn);

    /*
     * The player chooses whether to rest or roll the dice for that turn.
     */
}
```

```

    */
    Callable<Boolean> selectStay(int playerId);

    /*
        The dice are rolled based on the player's input.
    */
    Callable<Integer> rollDice();

    /*
        Display that there are no movable scales
    */
    void displayMoveValueZero(int diceResult, int penalty, int deduct);

    /*
        Displays the result of the dice and receives the input direction to move.
    */
    Callable<ArrayList<Direction>> enterDirection(int diceResult, int penalty, int deduct);

    /*
        If player are on a bridge cell, player can choose whether to move to the bridge or not.
    */
    Callable<Boolean> selectMoveBridge(int diceResult, int penalty, int deduct);

    /*
        Alert that the direction cannot be moved.
    */
    void alertInvalidMove();

    /*
        Display that you cannot move backwards
    */
    void displayCanNotMoveBack();

    /*
        Display the winner
    */
    void displayWinner(Turn turn);
}

```

GameService

```

public class GUIGameController implements GameService {
    private final MainFrame mainFrame;

```

```

private MapView mapView;
private PlayerContainerView playerContainerView;

public GUIGameController(final MainFrame view) {
    this.mainFrame = view;
}

@Override
public Callable<Boolean> selectUseDefaultMap() {
    Callable<Boolean> call = () -> {
        String option[] = new String[]{"맵 파일을 직접 선택", "기본 맵을 사용"};
        int answer = JOptionPane.showOptionDialog(null, "맵 파일을 불러올 방법을 선택해주세요.", "맵 선택",
JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE, null, option, null);
        if (answer == 0)
            return true;
        return false;
    };
    return call;
}

@Override
public Callable<String> enterMapFile() {
    Callable<String> call = () -> JOptionPane.showInputDialog("사용할 맵 파일 이름을 입력해주세요.");
    return call;
}

@Override
public void displayNotFoundMap() {
    JOptionPane.showMessageDialog(null, "파일이 존재하지 않습니다.", "File not found", JOptionPane.ERROR_MESSAGE);
}

@Override
public Callable<Integer> enterNumberOfPlayers() {

    Callable<Integer> call = () -> {
        while (true) {
            String input = JOptionPane.showInputDialog("플레이어 수를 입력해주세요. (2 ~ 4)");
            int res;
            try {
                res = Integer.parseInt(input);
                if (2 <= res && res <= 4)
                    return res;
                else
                    JOptionPane.showMessageDialog(null, "플레이어 수는 2 ~ 4 의 범위에서 선택할 수 있습니다.", "Invalid number",
JOptionPane.ERROR_MESSAGE);
            }
        }
    };
    return call;
}

```

```

        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "숫자를 입력해주세요.", "Invalid number", JOptionPane.ERROR_MESSAGE);
        }
    };
}

return call;
}

@Override
public void initDisplay(Map map, Turn turn) {
    mapView = new MapView(map.getAbsoluteMap());
    playerContainerView = new PlayerContainerView(turn.getPlayerList());
    mainFrame.add(mapView, BorderLayout.WEST);
    mainFrame.add(playerContainerView, BorderLayout.SOUTH);
    updateMainFrame();
}

@Override
public void refresh(Map map, Turn turn) {
    playerContainerView.setTurnOwner(turn.getTurnOwner());
    playerContainerView.updatePlayerStatus(turn.getPlayerList());

    mapView.clearPiece();

    for (Player player : turn.getPlayerList()) {
        mapView.displayPiece(player);
    }

    updateMainFrame();
}

@Override
public Callable<Boolean> selectStay(int playerId) {

    WaitCall<Boolean> call = new WaitCall() {
        @Override
        public Boolean call() {
            SelectStayView selectView = new SelectStayView(playerId);
            mainFrame.add(selectView, BorderLayout.EAST);
            updateMainFrame();

            selectView.getBtnStay().addActionListener(e -> {
                mainFrame.remove(selectView);
                updateMainFrame();
                signalResult(true);
            });
        }

        selectView.getBtnMove().addActionListener(e -> {
    
```

```

        mainFrame.remove(selectView);
        updateMainFrame();
        signalResult(false);
    });

    waitResult();
    return (Boolean) this.result;
};

return call;
}

@Override
public Callable<Integer> rollDice() {
    WaitCall<Integer> call = new WaitCall<Integer>() {
        @Override
        public Integer call() throws Exception {
            DiceView diceView = new DiceView();
            mainFrame.add(diceView, BorderLayout.EAST);
            updateMainFrame();

            diceView.getButton().addActionListener(e -> {
                int diceResult = diceView.getDiceResult();
                mainFrame.remove(diceView);
                updateMainFrame();
                signalResult(diceResult);
            });
        }

        waitResult();
        return result;
    };
    return call;
}

@Override
public void displayMoveValueZero(int diceResult, int penalty, int deduct) {
    JOptionPane.showMessageDialog(null, "주사위 결과 : " + diceResult + " 패널티 : " + penalty + " 사용한 눈금 : " + deduct + "\n남은
눈금이 0 이하입니다.", "Can not move", JOptionPane.INFORMATION_MESSAGE);
}

@Override
public @NotNull Callable<ArrayList<Direction>> enterDirection(int diceResult, int penalty, int deduct) {
    WaitCall<ArrayList<Direction>> call = new WaitCall<ArrayList<Direction>>() {

```

```

@Override
public ArrayList<Direction> call() throws Exception {
    CombDirView dirView = new CombDirView(diceResult, penalty, deduct);
    mainFrame.add(dirView);
    updateMainFrame();

    dirView.getButton(CombDirView.ButtonType.UP).addActionListener(e -> {
        if (dirView.addSelectDir(Direction.UP)) {
            mainFrame.remove(dirView);
            updateMainFrame();
            signalResult(dirView.getSelectedDirs());
        }
    });

    dirView.getButton(CombDirView.ButtonType.DOWN).addActionListener(e -> {
        if (dirView.addSelectDir(Direction.DOWN)) {
            mainFrame.remove(dirView);
            updateMainFrame();
            signalResult(dirView.getSelectedDirs());
        }
    });

    dirView.getButton(CombDirView.ButtonType.LEFT).addActionListener(e -> {
        if (dirView.addSelectDir(Direction.LEFT)) {
            mainFrame.remove(dirView);
            updateMainFrame();
            signalResult(dirView.getSelectedDirs());
        }
    });

    dirView.getButton(CombDirView.ButtonType.RIGHT).addActionListener(e -> {
        if (dirView.addSelectDir(Direction.RIGHT)) {
            mainFrame.remove(dirView);
            updateMainFrame();
            signalResult(dirView.getSelectedDirs());
        }
    });

    waitResult();
    return result;
}
};

return call;
}

@Override
public Callable<Boolean> selectMoveBridge(int diceResult, int penalty, int deduct) {
    Callable<Boolean> call = () -> {

```

```

        CombDirView dirView = new CombDirView(diceResult, penalty, deduct);
        mainFrame.add(dirView);
        updateMainFrame();

        String option[] = new String[]{"다리로 이동", "그냥 진행"};
        int answer = JOptionPane.showOptionDialog(null, "다리로 이동하시겠습니까? (penalty 가 증가합니다)", "다리 이동",
JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE, null, option, null);
        mainFrame.remove(dirView);
        if (answer == 0)
            return true;
        return false;
    };

    return call;
}

@Override
public void alertInvalidMove() {
    JOptionPane.showMessageDialog(null, "해당 방향으로 이동할 수 없습니다.", "Can not move", JOptionPane.ERROR_MESSAGE);
}

@Override
public void displayCanNotMoveBack() {
    JLabel textLabel = new JLabel("이제부터 앞으로만 이동할 수 있습니다.");
    textLabel.setFont(new Font("Arial", Font.BOLD, 25));
    mainFrame.add(textLabel, BorderLayout.NORTH);
}

@Override
public void displayWinner(Turn turn) {
    Player winner = turn.getWinner();
    JOptionPane.showMessageDialog(null, "게임이 종료되었습니다.\n승리 : Player" + winner.getId() + "\n획득한 점수 : " +
winner.getPoint(), "Game end", JOptionPane.ERROR_MESSAGE);
}

private void updateMainFrame() {
    mainFrame.revalidate();
    mainFrame.repaint();
}

public abstract static class WaitCall<T> implements Callable<T> {
    protected T result;

    public void signalResult(T result) {
        synchronized (this) {
            this.result = result;
            notify();
        }
    }
}

```

```
        }
    }

    public void waitResult() {
        try {
            synchronized (this) {
                wait();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

GUIGameController

```
public class CLIGameController implements GameService {

    @Override
    public Callable<Boolean> selectUseDefaultMap() {
        Callable<Boolean> call = () -> {
            System.out.println("맵 파일을 불러올 방법을 선택해주세요.\n맵 파일을 직접 선택 : 1 / 기본 맵을 사용 : 2");
            int answer;
            while (true) {
                answer = scanInteger();
                if (answer == 1)
                    return true;
                else if (answer == 2)
                    return false;
            }
        };
        return call;
    }

    @Override
    public Callable<String> enterMapFile() {
        Callable<String> call = () -> {
            System.out.println("사용할 맵 파일의 이름을 입력해주세요.");
            return ConsoleManager.next();
        };
        return call;
    }

    @Override
    public void displayNotFoundMap() {
```

```

        System.out.println("파일이 존재하지 않습니다.");
    }

    @Override
    public Callable<Integer> enterNumberOfPlayers() {
        Callable<Integer> call = () -> {
            System.out.println("플레이어 수를 입력해주세요. (2 ~ 4)");
            int res;
            while (true) {
                res = scanInteger();
                if (2 <= res && res <= 4)
                    return res;
                else
                    System.out.println("플레이어 수는 2 ~ 4 의 범위에서 선택할 수 있습니다.");
            }
        };
        return call;
    }

    @Override
    public void initDisplay(Map map, Turn turn) {
        // Displays a message for 2 seconds.
        ConsoleManager.printSplash();

        try {
            Thread.sleep(2000);
            ConsoleManager.clear();

        } catch (InterruptedException e) {

        }
    }

    @Override
    public void refresh(Map map, Turn turn) {
        if (!turn.getAllowMoveBack())
            System.out.println("이제부터 앞으로만 이동할 수 있습니다.");
        ConsoleManager.printMap(map.getAbsoluteMap(), turn.getPlayerList());
        ConsoleManager.printLine();
        System.out.println("Player " + turn.getTurnOwner().getId() + " 님의 차례입니다. ");
        ConsoleManager.printPlayerStatus(turn.getPlayerList());
        ConsoleManager.printLine();
    }

    @Override
    public Callable<Boolean> selectStay(int playerId) {
        Callable<Boolean> call = () -> {

```

```

        System.out.println("턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.\n쉬어가기 : 1, 주사위 굴리기 : 2");

        while (true) {
            int answer = scanInteger();
            if (answer == 1)
                return true;
            else if (answer == 2)
                return false;
            else
                System.out.println("잘못된 입력입니다. 다시 입력해주세요.");
        }
    };
    return call;
}

@Override
public Callable<Integer> rollDice() {
    Callable<Integer> call = () -> {
        Random random = new Random();
        random.setSeed(System.currentTimeMillis());
        int diceResult = random.nextInt(6) + 1;

        return diceResult;
    };
    return call;
}

@Override
public void displayMoveValueZero(int diceResult, int penalty, int deduct) {
    System.out.println("남은 눈금이 0이하입니다. (주사위 결과 : " + diceResult + ", 패널티 : " + penalty + ", 사용한 눈금 : " + deduct +
")");
    System.out.println("다음 턴을 진행하려면 아무 키나 입력해주세요.");
    ConsoleManager.nextLine();
}

@Override
public Callable<ArrayList<Direction>> enterDirection(int diceResult, int penalty, int deduct) {
    int dVal = diceResult - penalty - deduct;
    System.out.println("주사위 결과 : " + diceResult + ", 패널티 : " + penalty + ", 이동 가능 눈금 : " + dVal);
    System.out.println("이동할 방향을 공백없이 순서대로 입력하세요. (위 : U, 아래 : D, 왼쪽 : L, 오른쪽 : R)");
    Callable<ArrayList<Direction>> call = () -> {
        while (true) {

            ArrayList<Direction> dirs = new ArrayList<>();
            boolean success = true;
            String input = ConsoleManager.nextLine();

```

```

        if (input.length() != dVal)
            success = false;

        for (int i = 0; i < input.length(); i++) {
            switch (input.charAt(i)) {
                case 'U':
                    dirs.add(Direction.UP);
                    break;
                case 'D':
                    dirs.add(Direction.DOWN);
                    break;
                case 'L':
                    dirs.add(Direction.LEFT);
                    break;
                case 'R':
                    dirs.add(Direction.RIGHT);
                    break;
                default:
                    success = false;
            }
        }

        if (success)
            return dirs;
        else
            System.out.println("잘못된 입력입니다. 다시 입력해주세요.");
    };
}

return call;
}

@Override
public Callable<Boolean> selectMoveBridge(int diceResult, int penalty, int deduct) {
    Callable<Boolean> call = () -> {
        System.out.println("다리로 이동하시겠습니까? (penalty 가 증가합니다) \n 다리로 이동 : 1, 그냥 진행 : 2");
        while (true) {
            int answer = scanInteger();
            if (answer == 1)
                return true;
            else if (answer == 2)
                return false;
        }
    };
    return call;
}

@Override

```

```

        public void alertInvalidMove() {
            System.out.println("해당 방향으로 이동할 수 없습니다.");
        }

        @Override
        public void displayCanNotMoveBack() {
            // TO NOTHING
        }

        @Override
        public void displayWinner(Turn turn) {
            Player winner = turn.getWinner();
            System.out.println("게임이 종료되었습니다.");
            System.out.println("승리 : Player" + winner.getId());
            System.out.println("획득한 점수 : " + winner.getPoint());
        }

        private int scanInteger() {
            while (!ConsoleManager.hasNextInt()) {
                ConsoleManager.next();
                System.out.println("잘못된 입력입니다. 다시 입력해주세요.");
            }
            return ConsoleManager.nextInt();
        }
    }
}

```

CLIGameController

이외의 클래스들은 보고서에서 생략하도록 한다.

4. 테스트

4.1. Unit Test

구현 과정에서 Unit Test 를 기반으로 개발한다. model 계층이 분리되기 때문에 UI 가 구현되지 않은 상태에서도 정확하게 동작하는지 확인할 수 있다. 이번 프로젝트에서는 가장 중요한 기능인 맵 불러오기와 플레이어 움직이기를 테스트하였다.

a. 맵 불러오기

서로 다른 올바른 맵 데이터들이 준비된 상태에서 주어진 맵들을 오류 없이 정상적으로 불러오는지 확인한다.

만약 셀 정보를 불러오는 중 오류가 발생한다면 MapReader 는 BridgeNotFoundException 또는 InvalidInputException 을 반환하기 때문에 정상적으로 모든 셀이 연결된다면 테스트는 통과된다.

테스트 코드

```

public class MapReadTest {

    private static final String[] TEST_MAPS = { "default.map", "another.map", "maze.map", "snake.map" };

    public static void main(String args[]) {

        int cnt = 0;
        for (String name : TEST_MAPS) {
            try {
                MapReader reader = new MapReader(name);
                Map map = reader.getMap();

                System.out.println("Map test pass " + ++cnt + " / " + TEST_MAPS.length);
            } catch (Exception e) {
                System.out.println("Map test failed at " + name);
                e.printStackTrace();
            }
        }

        if (cnt == TEST_MAPS.length)
            System.out.println("All tests passed successfully.");
    }
}

```

MapReadTest

테스트 결과

```

Map test pass 1 / 4
Map test pass 2 / 4
Map test pass 3 / 4
Map test pass 4 / 4
All tests passed successfully.

Process finished with exit code 0

```

b. 플레이어 움직이기 테스트

기본 맵에서 미리 정해진 방향대로 플레이어를 움직인다. 미리 계산된 올바른 위치와 이동한 결과를 비교하여 테스트한다.

테스트 코드

```
public class PlayerMoveTest {

    private static final Direction[][] moveDirs = {
        {RIGHT, RIGHT, DOWN, DOWN},
        {UP, UP, DOWN},
        {DOWN, DOWN, DOWN, DOWN, DOWN},
        {UP, DOWN, RIGHT, RIGHT}
    };

    private static final RelativePosition[] correctPos = {
        createPos(2, -2),
        createPos(2, -1),
        createPos(2, -6),
        createPos(4, -6)
    };

    private static final Direction[] gotoBridge = {
        RIGHT, RIGHT, DOWN
    };

    public static void main(String args[]) {

        try {
            // use default map
            MapReader reader = new MapReader();
            Map map = reader.getMap();

            Player movePlayer = Player.newInstance(map.getStartCell());
            Player bridgePlayer = Player.newInstance(map.getStartCell());

            int cnt = 0;
            for (int i = 0; i < moveDirs.length; i++) {
                movePlayer.move(List.of(moveDirs[i]), MoveType.ADJACENT);
                if (movePlayer.getPiecePosition().getPosition().equals(correctPos[i]))
                    System.out.println("Player move test pass " + ++cnt + " / " + moveDirs.length);
                else {
                    System.out.println("Fail move test in case " + (i+1));
                }
            }

            bridgePlayer.move(List.of(gotoBridge), MoveType.ADJACENT);

            Direction[] crossBridge = {
                RIGHT, RIGHT
            };
            bridgePlayer.move(List.of(crossBridge), MoveType.BRIDGE);
        }
    }
}
```

```
        if (bridgePlayer.getPiecePosition().getPosition().equals(
            createPos(4, -1))
    )
        System.out.println("Bridge move test pass");
    else
        System.out.println("Fail bridge move test");

    } catch (Exception e) {
        e.printStackTrace();
    }
}

private static RelativePosition createPos(int x, int y) {
}
}
```

PlayerMoveTest

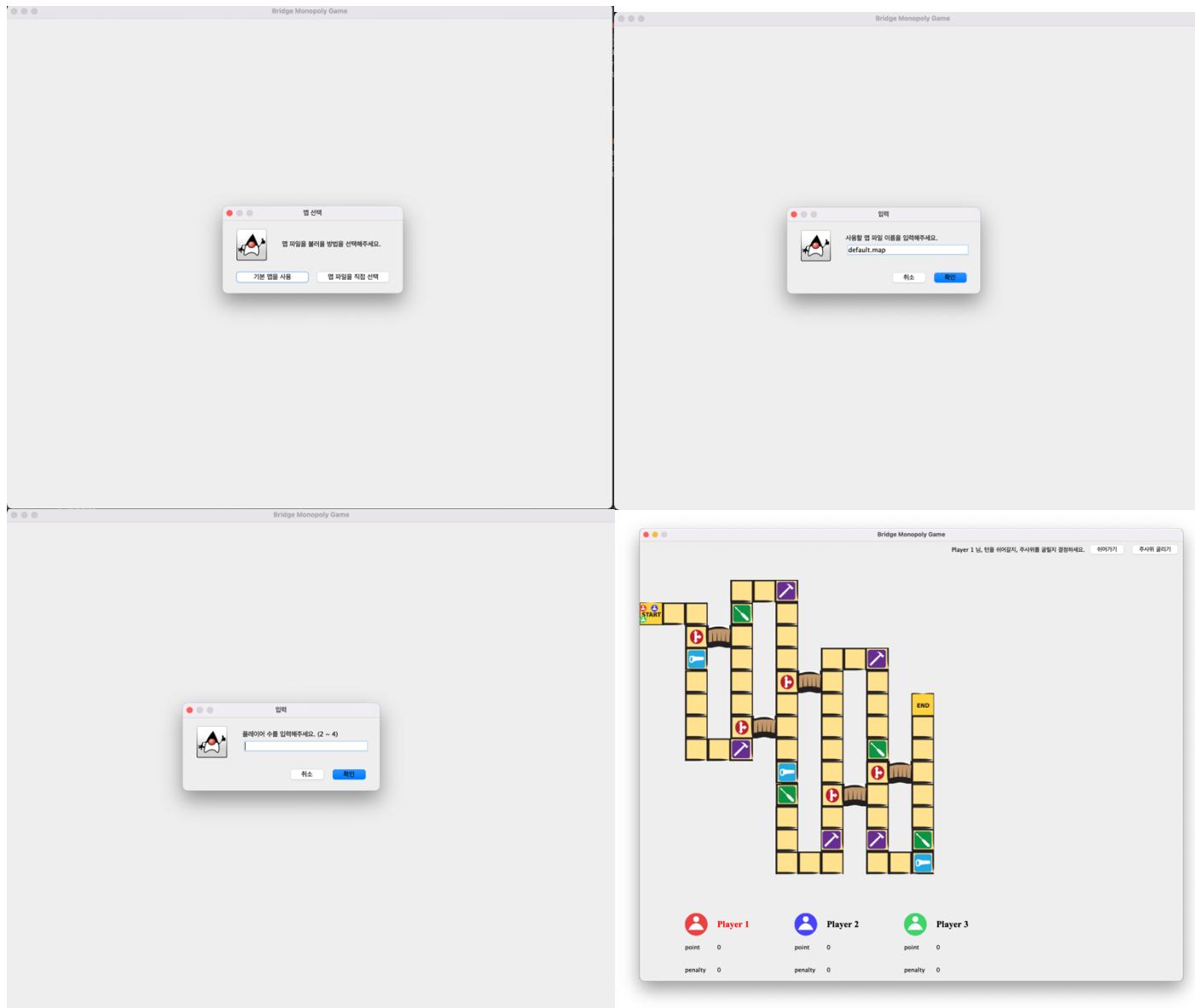
테스트 결과

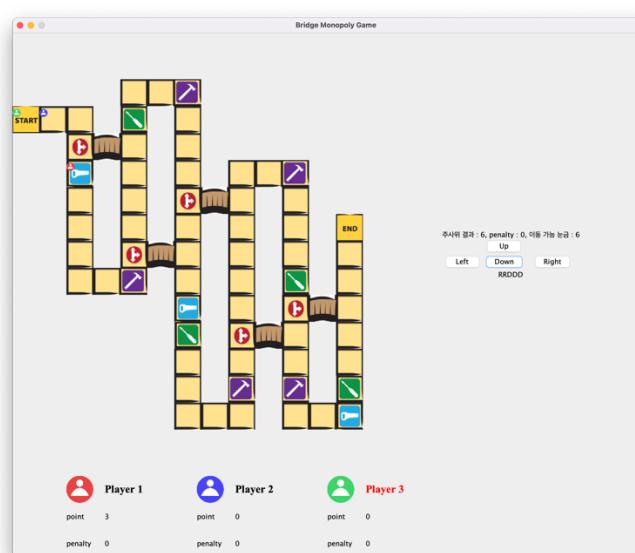
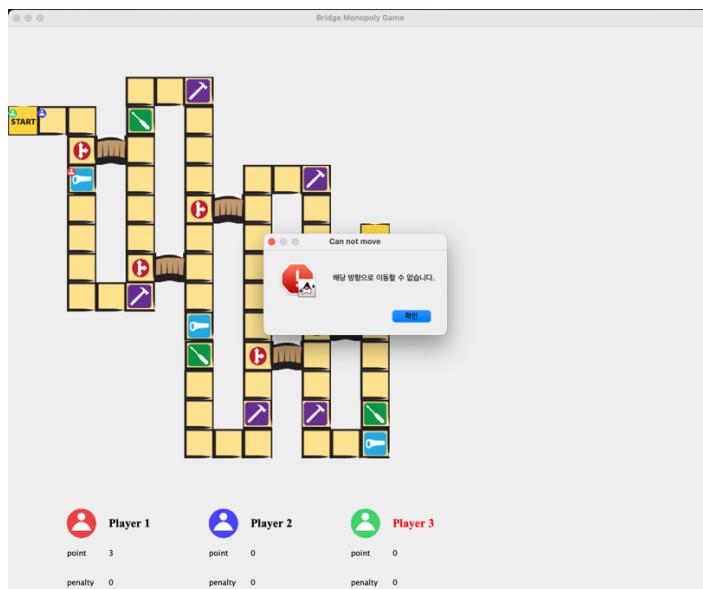
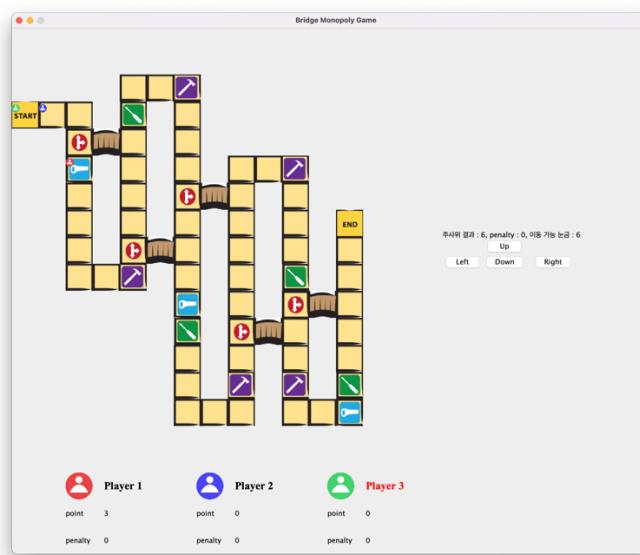
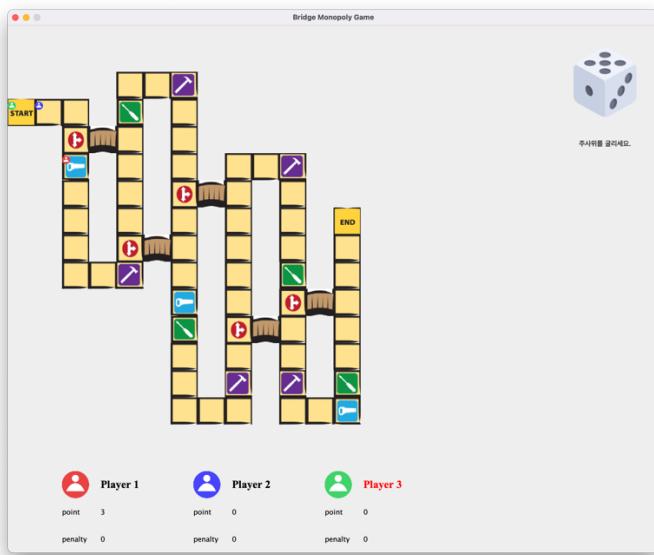
```
Player move test pass 1 / 4
Player move test pass 2 / 4
Player move test pass 3 / 4
Player move test pass 4 / 4
Bridge move test pass

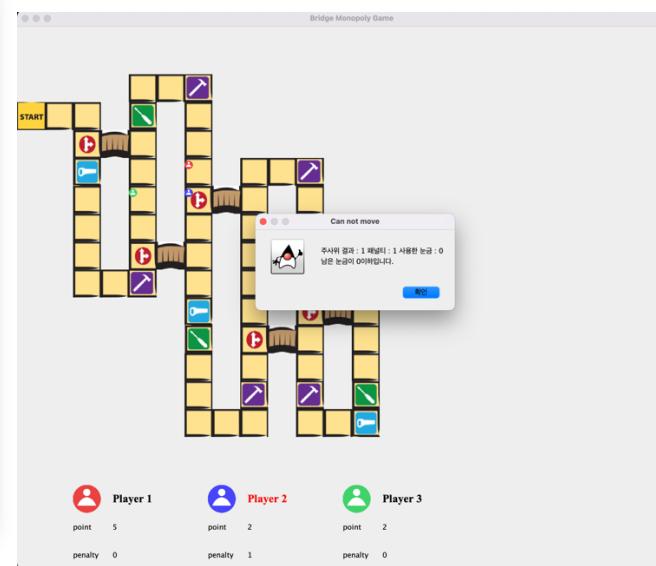
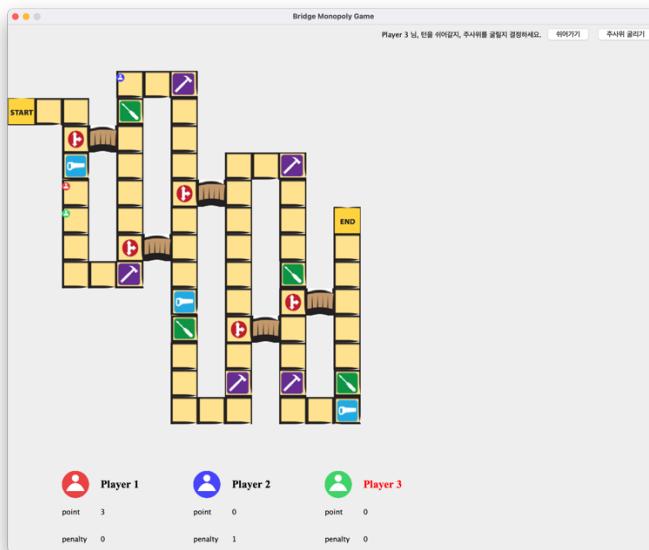
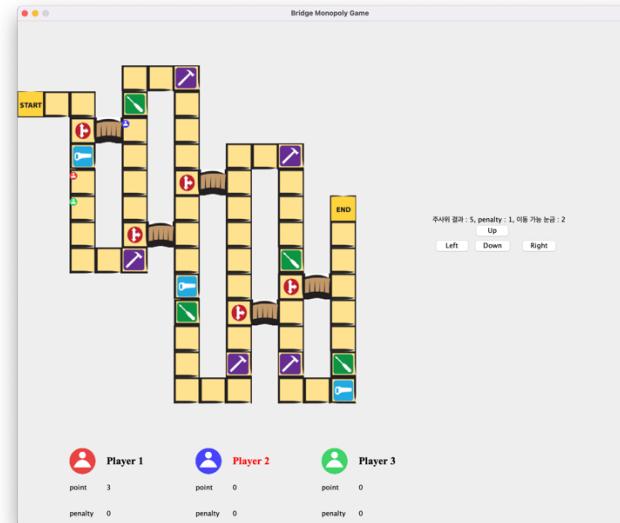
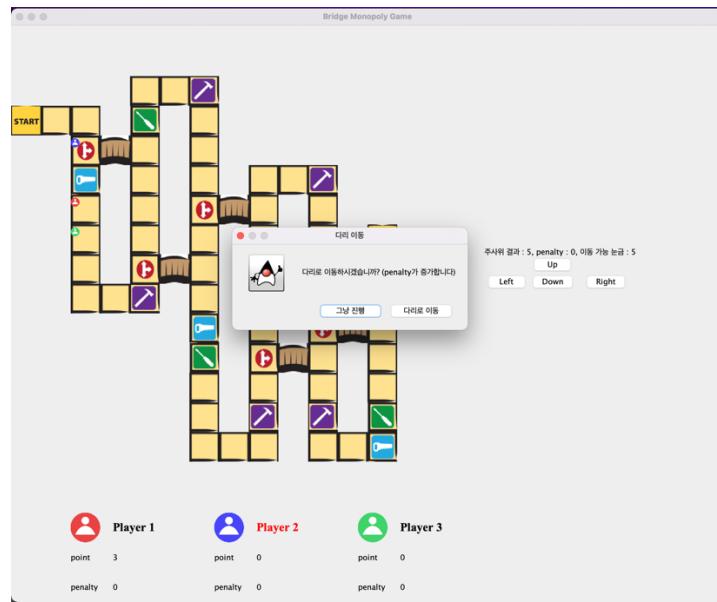
Process finished with exit code 0
```

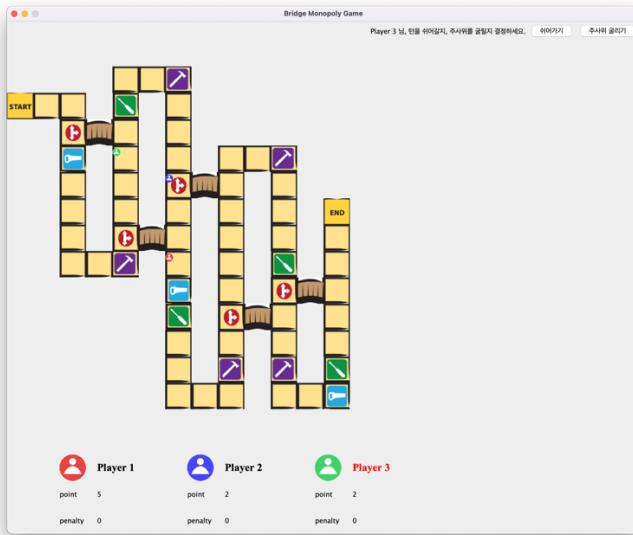
4.2. 프로그램 실행

4.2.1. GUI

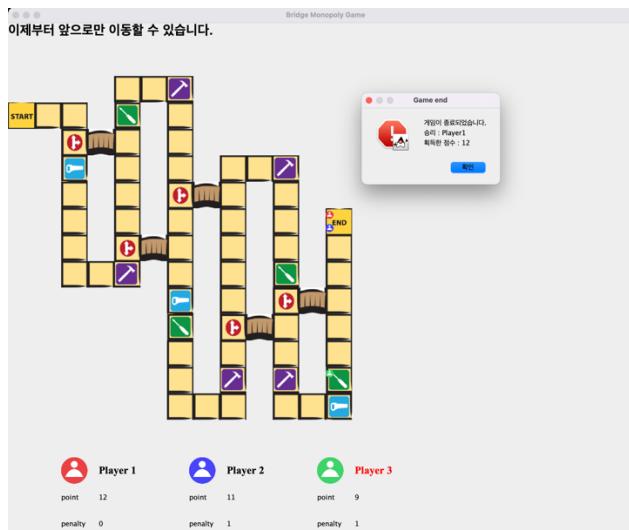








1



¹ Player 2가 쉬어가기를 선택하고 다리 카드를 하나 반납

4.2.2. CLI

```

java -jar BridgeMonopolyGameCLI.jar
주사위 결과 : 1, 배터리 : 0, 이동 가능 눈금 : 1
이동할 방향을 공백없이 순서대로 입력하세요. <위 : U, 아래 : D, 왼쪽 : L, 오른쪽 : R>
R
      h
      P
      h
      B
      S
      B
      E
      B
      h
      p
      s
      B
      p
      h
      h
      h
      s

Player 2 님의 차례입니다.
Player 1          Player 2          Player 3          Player 4
score : 0         score : 0         score : 0         score : 0
penalty : 0       penalty : 0       penalty : 0       penalty : 0

턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.
쉬어가기 : 1, 주사위 굴리기 : 2
^C

(base) x mangi ➜ ~/Github/Bridge-Monopoly/BridgeMonopolyGame ➜ main ↵
▶ ls
BridgeMonopolyGame.iml  BridgeMonopolyGameGUI.jar  out
BridgeMonopolyGameCLI.jar  maps  res
(base) mangi ➜ ~/Github/Bridge-Monopoly/BridgeMonopolyGame ➜ main ↵
▶ java -jar BridgeMonopolyGameCLI.jar
업 파일을 불러올 방법을 선택해주세요.
업 파일을 직접 선택 : 1 / 기본 맵을 사용 : 2
2

```

12% 21 GB 6/10 19:34

```

java -jar BridgeMonopolyGameCLI.jar
      h
      P
      h
      B
      S
      B
      E
      B
      h
      p
      s
      B
      p
      h
      h
      h
      s

Player 1 님의 차례입니다.
Player 1          Player 2          Player 3          Player 4
score : 0         score : 0         score : 0         score : 0
penalty : 0       penalty : 0       penalty : 0       penalty : 0

턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.
쉬어가기 : 1, 주사위 굴리기 : 2
^C

(base) x mangi ➜ ~/Github/Bridge-Monopoly/BridgeMonopolyGame ➜ main ↵
▶ ls
BridgeMonopolyGame.iml  BridgeMonopolyGameGUI.jar  out
BridgeMonopolyGameCLI.jar  maps  res
(base) mangi ➜ ~/Github/Bridge-Monopoly/BridgeMonopolyGame ➜ main ↵
▶ java -jar BridgeMonopolyGameCLI.jar
업 파일을 불러올 방법을 선택해주세요.
업 파일을 직접 선택 : 1 / 기본 맵을 사용 : 2
2

```

10% 20 GB 6/10 19:34

```

java -jar BridgeMonopolyGameCLI.jar

```

Player 1 님의 차례입니다.

Player 1	Player 2	Player 3	Player 4
score : 0	score : 0	score : 0	score : 0
penalty : 0	penalty : 0	penalty : 0	penalty : 0

턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.
쉬어가기 : 1, 주사위 굴리기 : 2
2
주사위 결과 : 6, 패널티 : 0, 이동 가능 눈금 : 6
이동할 방향을 공백없이 순서대로 입력하세요. (위 : U, 아래 : D, 왼쪽 : L, 오른쪽 : R)
RRDDDD

```

java -jar BridgeMonopolyGameCLI.jar

```

Player 1 님의 차례입니다.

Player 1	Player 2	Player 3	Player 4
score : 0	score : 0	score : 0	score : 0
penalty : 0	penalty : 0	penalty : 0	penalty : 0

턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.
쉬어가기 : 1, 주사위 굴리기 : 2
2
주사위 결과 : 6, 패널티 : 0, 이동 가능 눈금 : 6
이동할 방향을 공백없이 순서대로 입력하세요. (위 : U, 아래 : D, 왼쪽 : L, 오른쪽 : R)
RRDDDD

Player 2 님의 차례입니다.

Player 1	Player 2	Player 3	Player 4
score : 0	score : 0	score : 0	score : 0
penalty : 0	penalty : 0	penalty : 0	penalty : 0

턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.
쉬어가기 : 1, 주사위 굴리기 : 2
1
10%

java -jar BridgeMonopolyGameCLI.jar

Player 2 님의 차례입니다.

Player 1	Player 2	Player 3	Player 4
score : 0	score : 0	score : 0	score : 0
penalty : 0	penalty : 0	penalty : 0	penalty : 0

턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.
쉬어가기 : 1, 주사위 굴리기 : 2

2
주사위 결과 : 2, 패널티 : 0, 이동 가능 눈금 : 2
이동할 방향을 공백없이 순서대로 입력하세요. (위 : U, 아래 : D, 왼쪽 : L, 오른쪽 : R)
RR

45 2 2 p h

Player 3 님의 차례입니다.

Player 1	Player 2	Player 3	Player 4
score : 0	score : 0	score : 0	score : 0
penalty : 0	penalty : 0	penalty : 0	penalty : 0

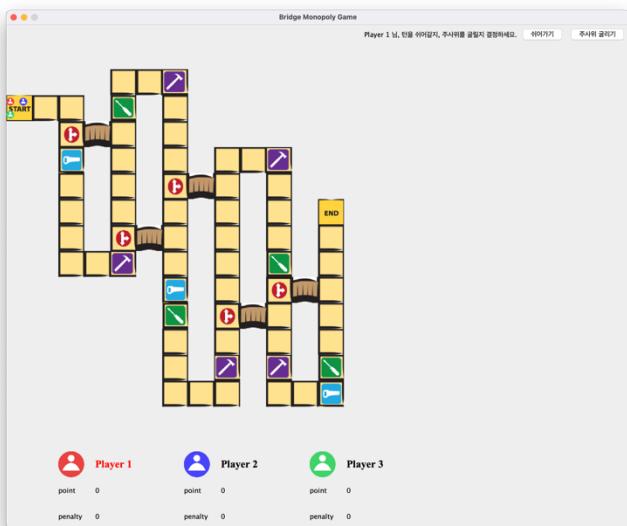
턴을 쉬어갈지, 주사위를 굴릴지 선택하세요.
쉬어가기 : 1, 주사위 굴리기 : 2

2
주사위 결과 : 6, 패널티 : 0, 이동 가능 눈금 : 6
이동할 방향을 공백없이 순서대로 입력하세요. (위 : U, 아래 : D, 왼쪽 : L, 오른쪽 : R)
RRDD

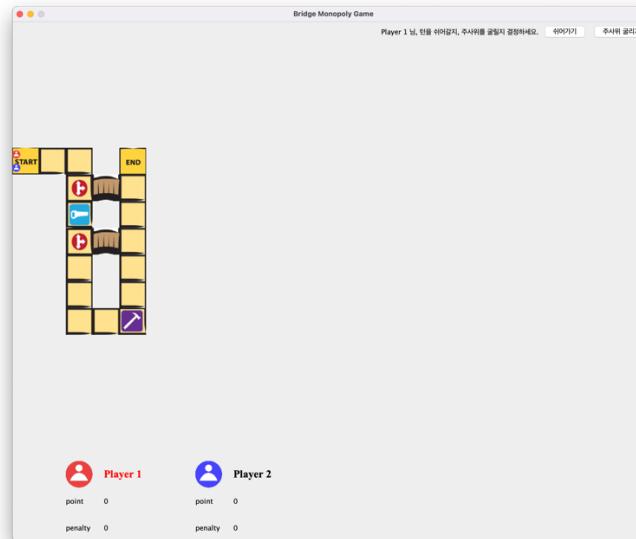
잘못된 입력입니다. 다시 입력해주세요.

기능들은 GUI 환경과 동일하게 작동하기 때문에 보고에서는 CLI 결과는 여기서 생략하도록 한다.

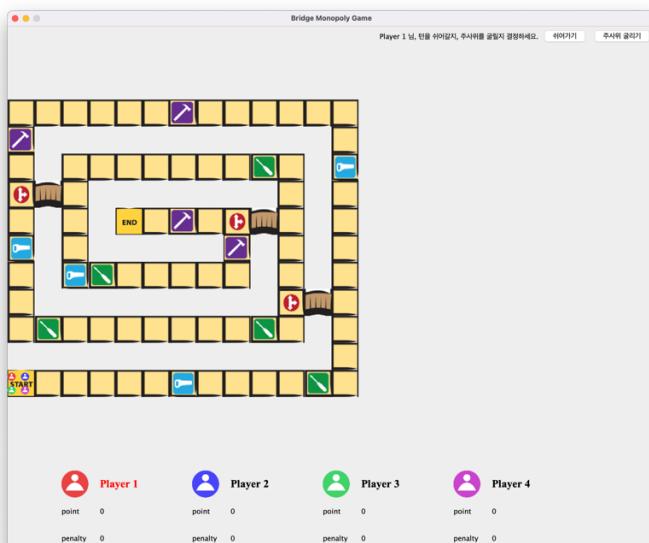
default.map 이외의 다른 맵을 불러왔을 때의 모습이다. another.map 은 제공된 것을 사용하였고, snake.map 과 maze.map 은 직접 정의하였다.



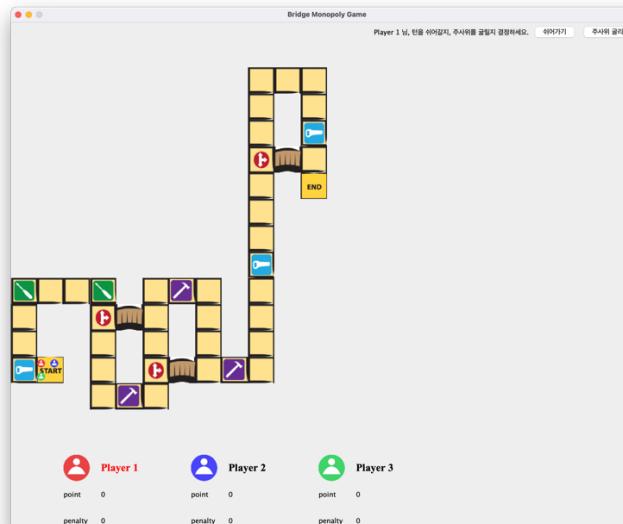
default.map



another.map



maze.map



snake.map