소프트웨어학부

20185144 이나경

보고서

2022 SE TERM PROJECT

내용

[여는 말 1](#_Toc105721248)

[수행과제 2](#_Toc105721249)

[요구 기능 2](#_Toc105721250)

[게임 특징 2](#_Toc105721251)

[INCEPTION 3](#_Toc105721252)

[핵심개념 3](#_Toc105721253)

[Use-Case Diagram 4](#_Toc105721254)

[Brief Use case 6](#_Toc105721255)

[Supplementary Specification 9](#_Toc105721256)

[Glossary 10](#_Toc105721257)

[ELABORATION 1 11](#_Toc105721258)

[핵심개념 11](#_Toc105721259)

[Full-Dressed Use case 12](#_Toc105721260)

[Domain model 15](#_Toc105721261)

[System Sequence diagram 17](#_Toc105721262)

[Sequence diagram 18](#_Toc105721263)

[operation contracts 19](#_Toc105721264)

[Implementation 20](#_Toc105721265)

[ELABORATION 2 32](#_Toc105721266)

[핵심개념 32](#_Toc105721267)

[Full-Dressed Use case 33](#_Toc105721268)

[Domain model 35](#_Toc105721269)

[System Sequence diagram 37](#_Toc105721270)

[operation contracts 38](#_Toc105721271)

[Implementation 39](#_Toc105721272)

[ELABORATION 3 46](#_Toc105721273)

[핵심개념. 46](#_Toc105721274)

[Full-Dressed Use case 47](#_Toc105721275)

[System Sequence diagram 49](#_Toc105721276)

[operation contracts 52](#_Toc105721277)

# 여는 말

소프트웨어 공학 강의의 중간고사 대체과제로 게임을 만들게 되었습니다.

평소 게임을 좋아하여, 많을 때는 하루 16시간씩 게임하며 ‘게임은 어떻게 만드는걸까?’ 라고 막연히 생각만 해오던 제가, 직접 만들게 되니 감회가 새롭습니다.

소프트웨어 공학에 대해 처음 배우면서 어떤 프로그램을 체게적으로 짠다는게 얼마나 중요한지 조금씩 알아가고 있지만, 아직 많이 미숙하여 어떻게 해야 알맞고 적절하게 디자인 되는지 알기는 어려운 것 같습니다.

그래도 이러한 과제를 통해 한번이라도 더 경험해보고 생각할 수 있다는 것이 좋았던 경험이었습니다.

해당 보고서는, 크게 Inception과 Elaboration 파트로 나뉘어져있으며, 각 단계에서 요구되는(할 것이라 예상되는) 행동들을 수행하며 작성되었습니다.

모든 산출물의 최종본들은 마지막 Artifact 파트에서 확인하실 수 있습니다.

# 수행과제

OOAD기법을 이용해 2~4명이 규칙에 따라 Bridge 보드 게임을 개발하는 것

## 요구 기능

* 지도 로드 기능
* 플레이 기능
* 유저 인터페이스 (GUI/ 콘솔)

## 게임 특징

* 게임 시작 전

- 해당 게임은 처음에 게임에 대한 정보를 입력한 뒤, 게임이 시작하는 방식입니다.

- 따라서 게임이 시작되기 전, 메인화면에서 Host가 게임을 엽니다.

- 해당 호스트는 게임 시작 후 Player 중 한 명이 됩니다.

* 게임 중
* 게임시작 전, Host가 게임을 열 때 정한 Player의 수만큼의 Player가 생성됩니다.
* 단 Player 수는 2명 이상 4명 이하라고 가정합니다. 이는 게임이 길어져 지루해지는 것을 막기위함입니다.
* 각 Player는 이름을 가지며 시작 전 각자 **주사위를 굴려 본인의 순서를 정합니다**.
* 플레이 규칙에 따라 게임을 진행 합니다.
* 게임 중에는 게임을 중단하거나 종료할 수 없습니다.
* 한명의 Player만 Map에 남을 경우 게임이 종료됩니다.
* 게임 종료 후
* 한명의 Player만 Map에 남을 경우 게임이 종료됩니다.
* 승자가 정해지며 화면에 표시됩니다.
* 메인 화면
* 게임을 시작하거나, Host가 열었던 게임 결과 기록들을 확인할 수 있습니다.

# INCEPTION

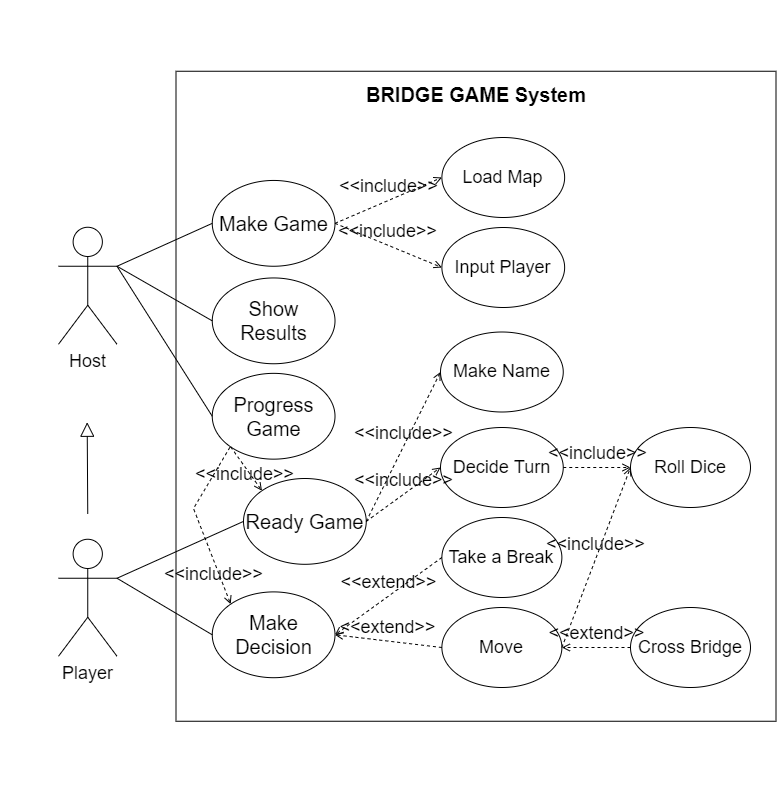
## 핵심개념

* Waterfall 방식처럼 하나의 설계를 전부 끝낸 후 다음 단계를 진행하는 것이 아닌, 프로젝트의 플랜을 수립하며 점진적으로 완성해 나가야 합니다.
* Inception은 본격적인 반복(Elaboration)이 시작되기 전의 단계입니다.
* 그렇다 하더라도 대부분의 요구사항 분석은 해당 단계에서 끝나야합니다.

INCEPTION 단계에서 시작되는 ARTIFACT 목록

|  |  |
| --- | --- |
| aRTIFACT | 작성 여부 |
| * Vision and business Case | **X** |
| * Use-case Model | V |
| * Supplementary Specification | V |
| * Glossary | V |
| * Risk List & Risk Management Plan | X |
| * Prototypes and proof-of-concepts | X |
| * Iteration Plan | X |
| * Phase Plan and Software Development Plan | X |
| * Development case | X |
|  |  |

## Use-Case Diagram



Host 는 게임을 시작하는 Actor이며, 해당 Actor은 추후 게임을 Play하는 Player의 일원으로써 행동합니다.

* Host를 분리한 이유?

Host를 분리한 이유는, Player는 하지 못하지만 Host로써 할 수 있는 특정 기능들이 있기 때문입니다.

그들은 메인화면(Player이 존재하지 않음)에서 게임을 만들거나, 진행했던 게임들의 결과에 대해 확인할 수 있습니다.

단순히 Player들이 게임을 하고자 하는 Actor라면, Host는 게임을 하는 것을 넘어 처음에 게임을 게획(Make Game)하고, 후에 게임에 대해 돌아보고(Show Results) 싶어할 가능성이 있는 Actor입니다.

* Player?

이들은 게임의 일원으로 참여하여 직접 게임을 플레이하고싶어하는 Actor입니다.

이들의 주 목표는 게임을 플레이하는 것이며, 당연히 게임 내에 어떠한 오류도 생성되지 않기를 원할 것입니다.

이들은 자신만의 전략을 세워 게임에서 이기기 위해 최선을 다할 것입니다.

물론 통신 기능은 지원하지 않으므로, 하나의 디스플레이에서 플레이하여 플레이어간의 뚜렷한 차이 및 경계는 옅을 것입니다.

## Brief Use case

* Make Game

Host는 게임을 열어야합니다. 게임을 열 때는, 플레이할 Map을 선택해 불러오고(Load Map 참고), 몇 명의 Player가 게임할 것인지 정해야합니다(Input Player). 위 과정을 모두 완료할 시 입력한 인원수에 맞게 플레이어가 생성되며 게임이 열립니다.

* Show Results

Host가 열었던 게임의 결과는 모두 시스템 특정 폴더 내에 개별 파일로써 기록됩니다. 따라서 별도의 GUI를 통해 파일의 목록을 확인할 수 있으며 목록에서 지난 게임 중 하나를 선택할 수 있습니다. 호스트는 목록을 선택하여 확인한 결과에서 그 게임의 Winner 및 각 플레이어의 최종 점수를 확인할 수 있습니다.

* Progress Game

Host가 직접적으로 화면의 특정 버튼을 누르면서 게임이 진행되는 것은 아니지만, 기본적으로 Host가 게임을 실행하므로Host의 주관 아래 게임이 진행된다고 가정합니다. Host는 게임을 진행시키며 각 Player들에게 정해진 순서에 따라 본인의 행동을 결정(Make Decision)하도록 할 것입니다. 이를 통해 게임이 진행되며, 한 명의 Player만 빼고 모두 Map의 End에 도착했을 경우 게임이 종료되며 본인이 가진 카드 및 도착한 순서에 따른 점수를 고려하여 최종 점수가 계산됩니다. 그 후 각자의 점수에 따라 제일 점수가 높은 Player가 Winner로 결정될 것입니다.

* Ready Game

Player는 Host가 생성한 게임에 참여합니다. 게임이 본격적으로 시작되기 전 각 플레이어의 이름을 정하고(Make Name), 주사위를 굴려(Roll Dice) 의사 결정 순서를 부여합니다(Decide Turn). 이후 게임이 시작됩니다.

* Make Decision

Player는 자기 순서가 되면 해당 순서는 쉴지(Take Break) 이동할지 (Move) 선택합니다. 해당 선택에 따라 본인의 턴을 소모하여 선택된 행동을 합니다.

* Load Map

Host는 파일 선택 GUI를 통해 해당 게임에서 플레이할 Map을 선택하여 Map 데이터 파일을 불러옵니다. 불러온 데이터 파일은 알고리즘에 따라 파싱되어 플레이어가 플레이 가능한 Map으로 변환됩니다. 변환된 Map은 게임이 진행되는 곳으로써 동작합니다.

* Input Player

Host는 몇 명의 Player와 플레이할 것인지 결정합니다. 2~4명 사이의 Player가 한 게임 내에서 플레이할 수 있으며, Host가 입력한 수대로 Player가 생성됩니다.

* Make Name

각 Player에 별도의 이름을 정합니다. 한 기기 내에서 모든 Player가 플레이해야하므로 생기는 모호성 문제(본인이 몇 번째 Player인지 모름)를 해결할 수 있습니다. 각 Player는 게임이 진행되면서 본인의 턴에 본인이 정한 플레이어의 이름이 표시되므로 원활히 게임을 진행할 수 있습니다.

* Decide Turn

게임 시작 전, Player는 주사위를 굴려(Roll Dice) 본인의 주사위 값으로 게임의 순서가 정해집니다. 가장 큰 수를 뽑은 Player부터 제일 작은 수를 뽑은 Player로 의사 결정 순서가 정해지며, 만약 같은 숫자를 뽑은 Player들이 존재할 경우 해당 Player들끼리는 랜덤으로 순서가 부여됩니다. 이를 통해 게임이 무질서하게 진행되는 것을 막습니다.

* Take Break

게임 진행 중, 본인의 차례가 되었을 때 Player는 휴식을 선택하였습니다. 이에 따라 Player는 본인이 가지고있던 Bridge 카드가 한장 삭제되며 해당 턴에 아무런 행동도 하지 않고 다음 순서의 Player에게 턴이 넘어갑니다.

* Move

게임 진행 중, 본인의 차례가 되었을 때 Player는 이동을 선택하였습니다. 이에 따라 Player는 주사위를 굴려 나온 값(Roll Dice)에서 본인이 보유한 Bridge카드 개수를 뺀 만큼 이동할 수 있습니다. Player는 지도를 통해 유효한 경로를 확인하고, 최종적으로 이동하려는 경로를 입력하여 이동합니다. 도착한 최종 위치에 Card가 있을 경우 해당 카드를 얻습니다.

* Roll Dice

Player는 주사위를 굴려 1부터 6 사이의 자연수를 무작위로 얻습니다. 이를 통해 Player의 순서를 결정하거나 게임 중 이동 가능한 횟수를 결정할 수 있습니다.

* CrossBridge

Player가 게임 진행 중 본인 순서에 이동을 선택하여 이동하는 중, 다리를 마주칠 수 있습니다. 해당 Player가 입력한 경로 상에 다리를 건너는 행위가 포함되어 있습니다. Player는 다리를 건너며 Bridge카드를 하나 얻습니다. 해당 카드는 다음 턴부터 주사위를 던져 얻은 본인의 이동 가능 횟수에 제약을 받습니다.

## Supplementary Specification

|  |  |
| --- | --- |
| **Revision History** | |
| Version | Description |
| 1 | 첫 문서. |

|  |
| --- |
| **소개** |
| Use Case에는 포함되지 않는 공통사항 등에 대해 서술된 문서이다. |

|  |
| --- |
| **콘솔, GUI 모두 지원** |
| 해당 게임은 GUI기반으로 플레이되나, GUI 사용이 불가능한 상황에 대비해 콘솔로도 플레이할 수 있도록 설계된다. |

## Glossary

|  |  |
| --- | --- |
| **Revision History** | |
| Version | Description |
| 1 | 첫 문서. |

|  |  |  |  |
| --- | --- | --- | --- |
| **definitions** | | | |
| Term | **Definition and Info** | **Format** | **Validation Rules** |
| Host | 시스템을 실행하는 사용자 |  |  |
| Player | 시스템에서 생성된 게임을 플레이하는 사용자 |  |  |
| Cell | Map의 각 칸 |  |  |

# ELABORATION 1

## 핵심개념

* 반복의 시작입니다.
* 핵심적이고, 리스크가 높은 아키텍처가 먼저 프로그래밍되고 테스트됩니다.

ELABORATION단계에서 시작되는 ARTIFACT 목록

|  |  |
| --- | --- |
| aRTIFACT | 작성 여부 |
| * Domain Model | **V** |
| * Design Model | V |
| * Software Architecture Document | X |
| * Data Model | X |
| * Use-Case Story Boards, UI Prototypes | V |

## Full-Dressed Use case

UseCase Diagram을 통해 보여준 UseCase들 중, 가장 핵심적인 UseCase에 대해서 먼저 파악하였습니다.

게임이 진행되는 것도 중요하지만, 제일 먼저 게임이 준비되지 않으면 게임은 시작될 수 없습니다.

따라서 이번 Elaboration에서는 Make Game UseCase에 대해 먼저 명세하고, 게임의 핵심인 Map Data파일을 불러와 게임에서 사용 가능한 Map의 형태로 변환하는 Load Map까지 함께 명세할 예정입니다.

|  |  |  |
| --- | --- | --- |
| Usecase #Make Game | | |
|  | |  |
| SCOPE | Bridge Game Program | |
| LEVEL | User Goal | |
| Primary Actor | Host | |
|  | |  |
| STAKEHOLDERS AND INTERESTS | | |
| Host | * Game을 열어 플레이하고 싶어한다 * 플레이할 Map을 선택한다 * 함께 게임할 Player의 수를 정한다 | |
|  | |  |
| PRECONDITIONS | | |
| * Map 데이터는 Local에 저장되어있다 | |  |
|  | | |
| SUCCESS GUARANTEE | | |
| * 게임 사전 정보가 정해진다 * 게임이 시작된다 * Map 데이터를 읽고 Map을 구성한다 * 정해진 Player수만큼 Player가 생성된다 | | |
|  | | |
| MAIN SUCCESS SCENARIO | | |
| 1. Host가 게임을 시작하기 위해 System에서 새로운 게임 생성을 요청한다 2. Local에 저장되어있는 Map들 중 하나를 선택하고, System에서 Map 데이터 파일을 읽어 Map을 구성한다. 이는 LoadMap UseCase를 따른다.   Host는 Map 선택이 완료될 때까지 스텝 2를 반복한다   1. Host 본인을 포함해 함께 플레이할 Player수를 2명 이상, 4명 이하로 정한다.   Host는 Player수가 정해질 때까지 스텝 3을 반복한다   1. System에서 정해진 정보를 다시 한번 알리며, 정상적으로 설정했는지 Host에게 확인을 요청한다 2. 게임 구성이 완료되고 게임을 시작한다 | | |
|  | | |

|  |
| --- |
| **extensions** |
| 3a. 범위 밖(2명 미만, 4명 초과)의 Player 수를 선택한다   1. Error 메시지와 함께 Player 수 재선택을 요청한다 2. Host는 올바른 Player 수를 입력할 것이다 |

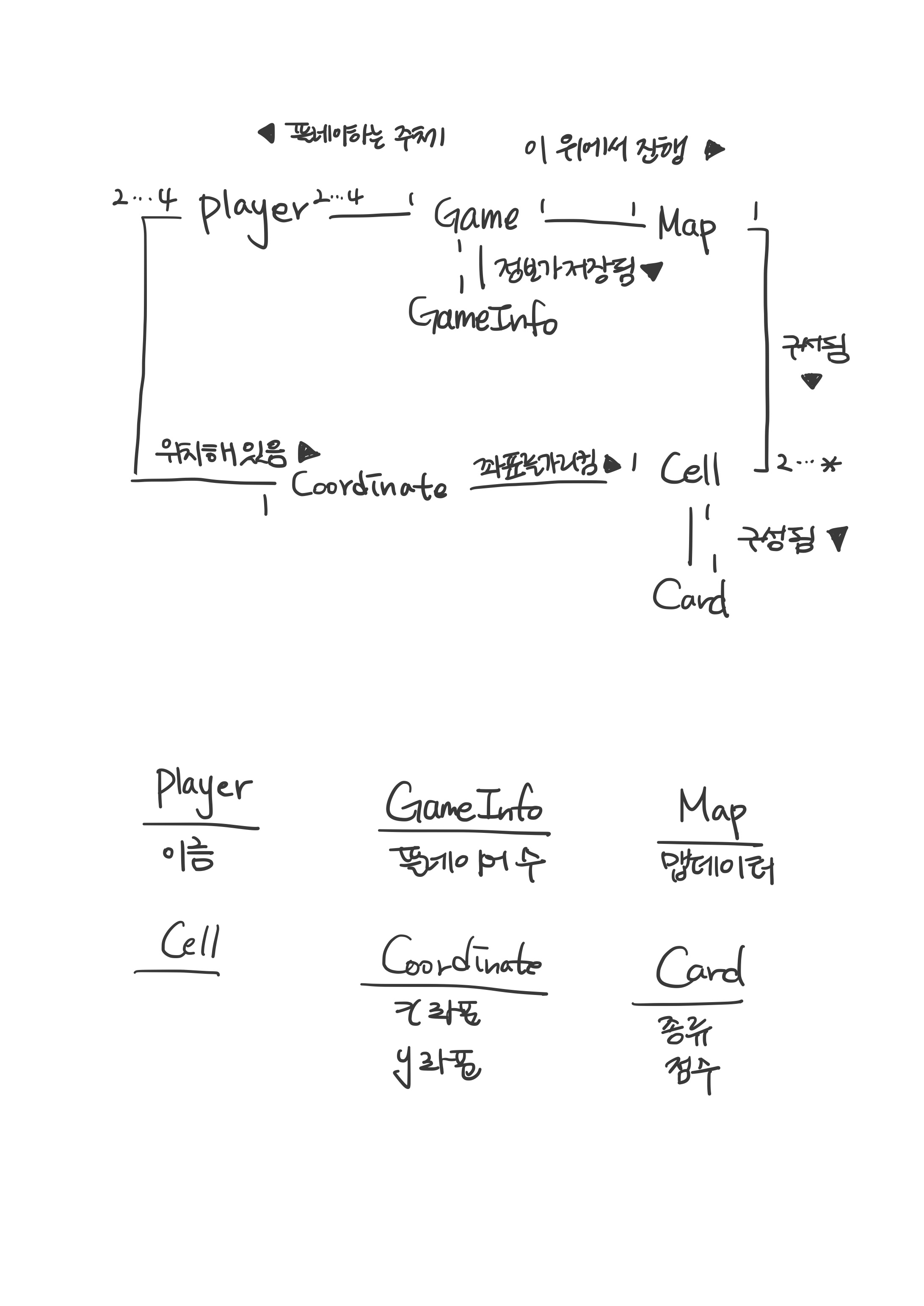
|  |  |  |
| --- | --- | --- |
| Usecase #Load map | | |
|  | |  |
| SCOPE | Bridge Game Program | |
| LEVEL | Subfunction | |
|  | |  |
| PRECONDITIONS | | |
| * Map이 있어야지만 게임을 진행할 수 있다 * System 내부에는 Default Map이라는 Map 데이터 파일을 내장하고있다 | | |
|  | | |
| SUCCESS GUARANTEE | | |
| * 게임의 핵심요소인 Map을 System을 통해 표시할 수 있다 | | |
|  | | |
| MAIN SUCCESS SCENARIO | | |
| 1. System에서 Host에게 Map 선택을 요청한다 2. Host가 Local에 저장된 Map 데이터 파일을 선택한다 3. System에서 선택한 Map으로 진행할 것인지 Host에게 확인을 요청한다 4. System에서 해당 Map 파일을 불러온다 5. Map데이터 파일의 각 줄에서 Cell의 종류 (C,S,B,S,H,P,E //첫 줄의 S는 Start의 S이며 나머지 줄의 S는 Saw의 S)와 해당 Cell과 이어진 Cell의 방향(U,D,L,R)을 읽어 Map의 가로 세로 길이를 구해 해당 크기의 Cell 이중 배열을 만든다. 6. 다시 한 번 데이터 파일을 읽어 이번에는 데이터대로 Map을 구성한다. Cell에 Card가 존재할 경우 Cell에 저장한다 7. Map 구성을 완성한다 | | |
|  | | |

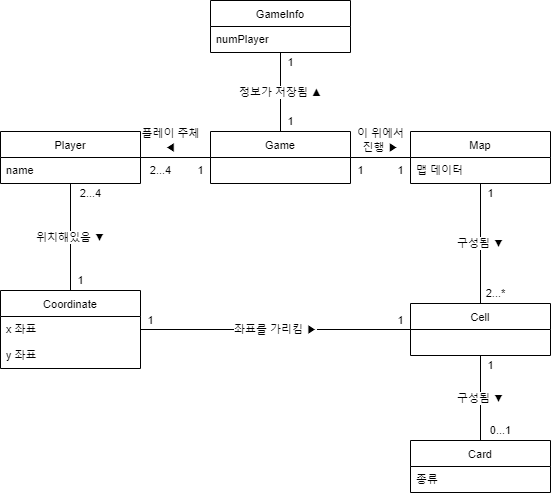
|  |
| --- |
| **extensions** |
| 1a. Local에 저장된 Map이 없다   1. System은 저장된 Map데이터 파일이 없다고 알린다 2. System이 내장하고있는 Default Map을 선택한다   1b. Host가 Default Map으로 플레이하기를 원한다   1. System이 내장하고있는 Default Map을 선택한다   2a. 선택한 파일이 Map 파일이 아니다   1. Error 메시지와 함께 파일 재선택을 요청한다   올바른 Map 데이터 파일을 선택할 때까지 스텝 1을 반복한다   1. Host는 올바른 Map 데이터 파일을 선택한다   4a. Map 데이터 파일을 불러왔으나 올바른 구성이 아니다   1. Error 메시지와 함께, 해당 Map 데이터 파일로는 Map을 구성할 수 없음을 알린다 2. Host는 다시 다른 Map 데이터 파일을 선택할 것이다 |

## Domain model

해당Elaboration에서 명세된 UseCase를 토대로 생성하였습니다.

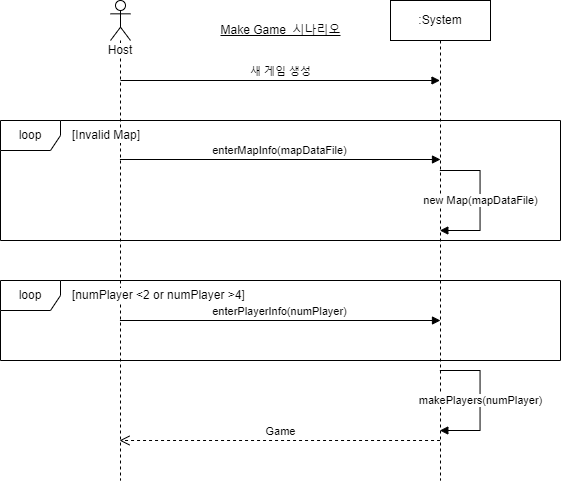
이는 추후 반복 회차를 거듭하며 세분화되거나 추가될 수 있습니다.





## System Sequence diagram

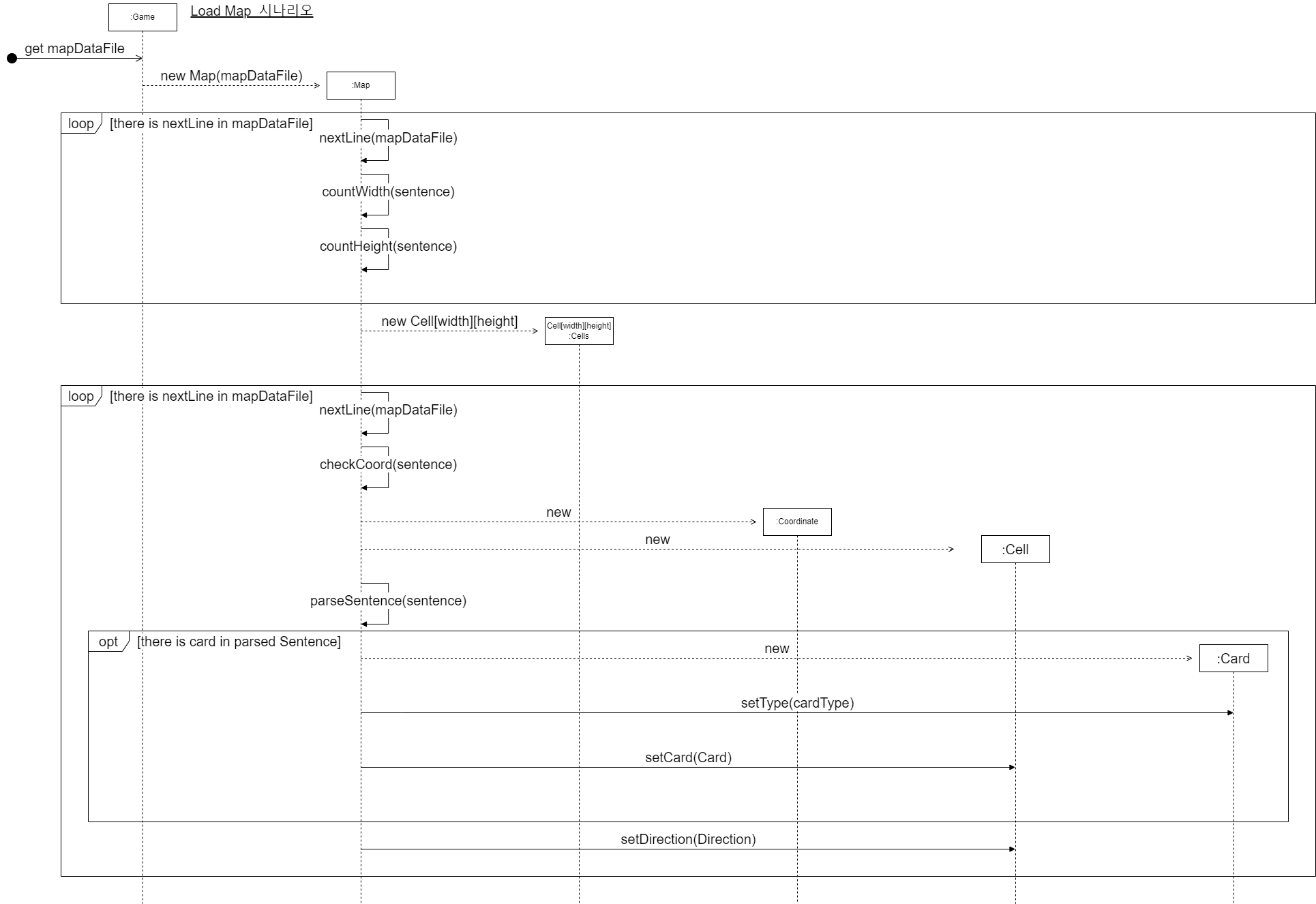
* Make Game 시나리오



## Sequence diagram

* Load Map 시나리오

Load Map 시나리오의 경우에는 Actor와의 Interaction이 없어 sequence diagram으로 작성하였습니다.



## operation contracts

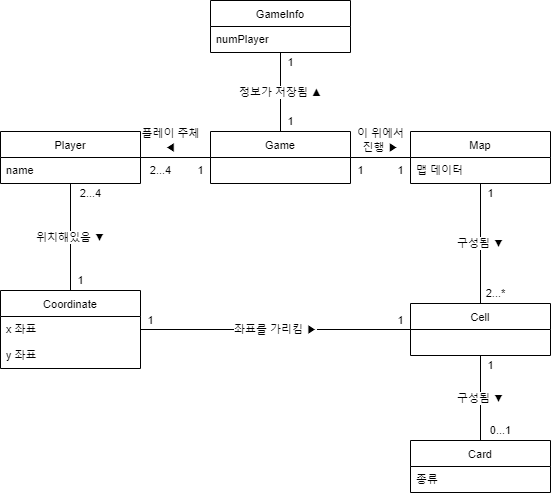
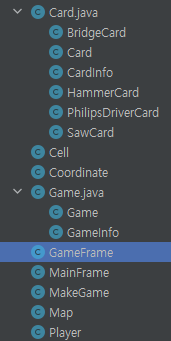
|  |  |
| --- | --- |
| Contract c01: enter map info | |
| Operation | EnterMapInfo(mapDataFile: .map File) |
| Cross References | Use Cases: Make Game |
| PreConditions | 없음 |
| PostConditions | * Game 객체 game이 생성되었다 * Map 객체 map이 생성되었다 * map.mapDataFile에 mapDataFile이 할당되었다 * game.map에 map이 할당되었다 * map은 game과 관련이 생겼다 |

|  |  |
| --- | --- |
| Contract c02: enter player info | |
| Operation | EnterPlayerInfo(numPlayer: Integer) |
| Cross References | Use Cases: Make Game |
| PreConditions | game객체와 game.map에 값이 할당된 상태이다 |
| PostConditions | * numPlayer만큼 Player객체 player가 생성되었다 * game.Players 배열에 생성된 player들이 요소로써 추가되었다 * player은 game과 관련이 생겼다 |

## Implementation

우선 콘솔 기반 게임 제작에 한계를 느끼고, GUI기반으로 제작을 하는 것을 먼저 목표로 삼았습니다. 만약 과제 완료까지 시간이 여유로울 경우 콘솔플레이 또한 부가로 제공할 예정입니다.

먼저 Domain Model에서 작성한 내용을 기반으로 클래스를 만들었습니다.

다만, 작성중에 보완된 부분이 있어, 해당 부분을 설명드리겠습니다.

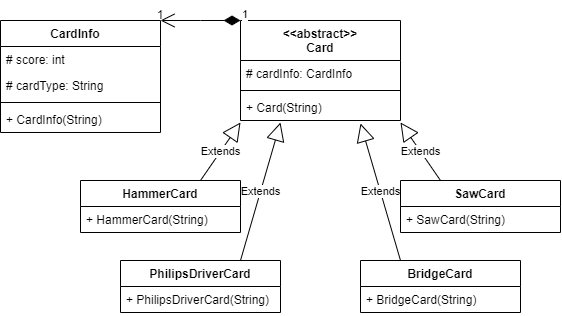
* Card

Card로 묶고 그 안에 종류로써만 속성을 가졌던 Domain Model과는 다르게, 각 카드의 종류에 따라 다른 점수 혹은 다른 행동들이 발생할 수 있는 점을 염두에 두고, Card클래스를 상속한 4개의 클래스 BridgeCard, HammerCard, PhilipsDriverCard, SawCard를 두었습니다.

또한, 각 카드에 필요한 공통적인 속성은 CardInfo클래스를 통해 따로 정의하였고, 해당 클래스는 모든 Card 종류의 슈퍼클래스인 Card의 속성으로써 위치합니다.

CardInfo에는 map 데이터 파일을 읽으면서 각 칸의 첫 글자인CardType(B, H, P, S), 그리고 각 카드에 대한 점수를 저장합니다.

getter와 setter을 제외하고 클래스 다이어그램을 그리면 다음과 같습니다.



* Coordinate’

일반적으로 알고있는 이차원 좌표평면에서의 위치를 표시하는 좌표 값을 속성으로 가지는 클래스입니다.

따라서, x와y로 이루어져있습니다.

또한, 좌표 평면 내에서 x와 y의 값은 언제나 바뀔 수 있기 때문에, up,down,left,right 메소드를 구현하여 현재 x,y좌표에서 up down left right 할 경우에, 경우에 맞게 x와 y값이 변하게됩니다.

다만, 기본적인 좌표평면에서는 오른쪽으로 갈수록 x축의 값이 증가하고, 위로 갈수록 y축의 값이 증가하는 것과는 다르게, 배열에서는 x축은 같으나 y축으로는 아래로 내려갈수록 y축의 값이 증가한다는 차이점을 인지하였습니다.

따라서 위로 갈 때는 y값에 -1을, 아래로 갈 때는 y값에 +1을 하였습니다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | x++ |  |  |
| Y++ |  | up  y-- |  |
|  | left  x-- | x,y | right  x++ |
|  |  | down  y++ |  |

* Map

흐름은 기존에 Sequence Diagram으로 전달한 내용과 같으나, 좀 더 상세히 Cells 배열을 구현한 방법에 대해 서술하겠습니다.

해당 내용은 loadmap 메소드에서 확인하실 수 있습니다.

private void loadMap() throws FileNotFoundException {

Scanner reader = new Scanner(mapDataFile);

ArrayList<String[]> parsedSentenceList = new ArrayList<>();

String sentence = "";

int width = 0;

int height = 0;

int min = 0;

int max = 0;

//기본 맵을 구축하기 위해 cells의 width와 height를 구하여 할당

while(reader.hasNext()){

sentence = reader.nextLine();

// System.out.println(sentence);

String[] parsedSentence = parseSentence(sentence);

parsedSentenceList.add(parsedSentence);

width += countWidth(parsedSentence);

height += countHeight(parsedSentence);

if (height<min){

min = height; //맵 높이의 최소상대값(START기준)를 구함

// System.out.println("new min: "+min);

}

if (height>max){

max = height; //맵 높이의 최대상대값(START기준)를 구함

// System.out.println("new max: "+max);

}

}

width += 1; //맵의 너비

height = max - min + 1; //맵의 높이

System.out.println("width:"+width+" height:"+height);

cells = new Cell[height][width];

Coordinate coord = new Coordinate(0-min,0); //map의 start 시작지점 정함

startCoordinate = new Coordinate(0-min, 0); //시작지점 정보 저장

//기본 맵의 정보를 채우는 작업

for(int i = 0; i<parsedSentenceList.size(); i++){

Cell tempCell = new Cell();

String[] tempStringArray = parsedSentenceList.get(i);

if(i != 0){tempCell.setCard(MakeGame.findCardType(tempStringArray[0]));} //시작줄은 S cell이지만 Saw의 S와 겹치므로 제외

else{}

for(int j=1; j<tempStringArray.length; j++){tempCell.addDirection(tempStringArray[j]);} //cell에 방향 저장

cells[coord.getY()][coord.getX()] = tempCell;

if(i != parsedSentenceList.size()-1){

coord.move(tempCell.getDirectionList().get(tempCell.getDirectionList().size()-1)); //다음 계산을 위해 이동}

}else{

endCoordinate = new Coordinate(coord.getY(), coord.getX());

}

}

}

if (height<min){

min = height; //맵 높이의 최소상대값(START기준)를 구함

// System.out.println("new min: "+min);

}

if (height>max){

max = height; //맵 높이의 최대상대값(START기준)를 구함

// System.out.println("new max: "+max);

}

}

width += 1; //맵의 너비

height = max - min + 1; //맵의 높이

System.out.println("width:"+width+" height:"+height);

cells = new Cell[height][width];

Coordinate coord = new Coordinate(0-min,0); //map의 start 시작지점 정함

startCoordinate = new Coordinate(0-min, 0); //시작지점 정보 저장

//기본 맵의 정보를 채우는 작업

for(int i = 0; i<parsedSentenceList.size(); i++){

Cell tempCell = new Cell();

String[] tempStringArray = parsedSentenceList.get(i);

if(i != 0){tempCell.setCard(MakeGame.findCardType(tempStringArray[0]));} //시작줄은 S cell이지만 Saw의 S와 겹치므로 제외

else{}

for(int j=1; j<tempStringArray.length; j++){tempCell.addDirection(tempStringArray[j]);} //cell에 방향 저장

cells[coord.getY()][coord.getX()] = tempCell;

if(i != parsedSentenceList.size()-1){

coord.move(tempCell.getDirectionList().get(tempCell.getDirectionList().size()-1)); //다음 계산을 위해 이동}

}else{

endCoordinate = new Coordinate(coord.getY(), coord.getX());

}

}

}

//기본 맵의 정보를 채우는 작업

for(int i = 0; i<parsedSentenceList.size(); i++){

Cell tempCell = new Cell();

String[] tempStringArray = parsedSentenceList.get(i);

if(i != 0){tempCell.setCard(MakeGame.findCardType(tempStringArray[0]));} //시작줄은 S cell이지만 Saw의 S와 겹치므로 제외

else{}

for(int j=1; j<tempStringArray.length; j++){tempCell.addDirection(tempStringArray[j]);} //cell에 방향 저장

cells[coord.getY()][coord.getX()] = tempCell;

if(i != parsedSentenceList.size()-1){

coord.move(tempCell.getDirectionList().get(tempCell.getDirectionList().size()-1)); //다음 계산을 위해 이동}

}else{

endCoordinate = new Coordinate(coord.getY(), coord.getX());

}

}

}

1. Scanner 객체를 통해 mapDataFile을 불러옵니다. 확장자는 map입니다.
2. 해당 파일을 한 줄씩 읽습니다.
3. 한번 읽을 때 마다, 문장을 “ “ 공백을 기준으로 나눠 String 배열에 담습니다.
4. 생성된 배열은 String 배열을 담는 ArrayList<String[]> 객체 parsedSentenceList에 add됩니다.
5. countWidth()를 통해 가로 길이를 구합니다. 가로길이는 해당 파일의 R 개수 +1입니다.
6. countHeight()를 통해 세로 길이를 구합니다. 세로길이는 문장이 “\* L D”,”\* U D”,”S D”일 때 셀이 위로 올라가고, “\* L U”, “\* D U”, “S U”일 때 셀이 아래로 내려간다는 점에 주목하였습니다. 단어를 읽고, 셀이 위로 올라갈때는 height가 하나씩 줄고, 내려갈때는 height가 하나씩 늘어납니다. 이는 coordinate와 동일한 사유입니다. (y가 아래로 내려갈수록 커지는 특성) 이와 동시에 height값의 min과 max 값을 찾아줍니다. 최종적으로 max에서 min을 뺀 값에 +1을 하면 height가 됩니다.
7. 이를 토대로 cells 속성의 cell배열 크기를 구해 생성해줍니다. (new Cell[height][width])
8. 셀의 시작점을 찾기 위해, Coordinate 객체를 y값에는 0-min, x값에는 0을 넣어 생성해줍니다. 이는 게임이 왼쪽에서 오른쪽으로 진행되기 때문에, Start 셀의 x좌표는 무조건 0이며, min값은 start보다 얼마나 위에가 배열이 끝인지 알려주는 값이므로 0에서 해당 값을 빼면, start의 y좌표값을 알 수 있습니다.
9. 해당 정보는 startCoordinate라는 Map의 속성에 따로 저장해줍니다.
10. 본격적으로 Cells 배열의 내부를 채워줍니다.
11. 이전에 저장해두었던 parsedSentencedList를 활용합니다.
12. temp라는 이름의 Cell 객체를 만들어줍니다.
13. MakdGame 클래스 (게임을 만들기 위해 필요한 public 메소드들이 static으로 정의된 클래스입니다)의 findCardType을 통해 해당 parsedSentece가 어떤 카드를 가지고 있는 지 확인하고, 만약 카드가 존재하는 Cell이라면 해당 카드타입의 객체를 생성하여 반환합니다.
14. 해당 카드는 Cell의 Card setter를 통해 tempCell의 card 속성에 저장됩니다.
15. 그 후, 해당 Cell을 올바른Cells 배열의 좌표의 Cell 객체로 할당합니다.
16. ParsedSentence의 마지막 글자를 확인하여 다음 방향이 어딘지 체크하고 (ex. “L D”인 경우 다음 방향이 D 아래방향) Coordinate의 move 메소드를 이용해 coord객체의 값을 이동시킵니다.
17. 텍스트이(가) 표시된 사진

    자동 생성된 설명for문이 끝날때까지 반복합니다.

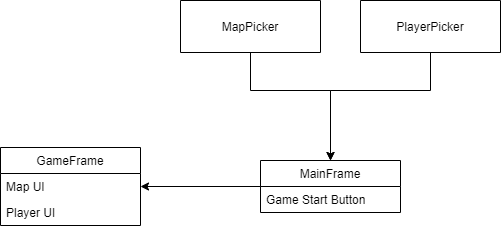
이를 활용해 default.map을 콘솔에 표시하면 다음과 같습니다.

* GUI

새로운 문제에 봉착했습니다.

해당 게임은 GUI기반 게임이므로 GUI를 위한 클래스가 따로 존재해야합니다.

따라서 해당 ELABORATION에서 구현되어야 할 GUI를 구상하였습니다.



먼저 메인 화면인 MainFrame, 그리고 맵을 고르기 위한 MapPicker, Player 수를 입력하기 위한 PlayerPicker, Game을 표시하기 위한 GameFrame이 필요합니다.

* MapPicker

MapPicker은 map파일만을 가져와야 합니다. 다른 파일은 읽을 수 없기 때문입니다.

파일을 고르는 GUI 구현은 쉽습니다.JFileChooser을 이용하면 쉽게 파일을 가져올 수 있습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

다만 이후에 map파일만 가져오게 하는 것은 다른 문제였습니다.

LoadMap 시나리오의 예외 두가지에 대해 생각했습니다.

1. 올바르지 않은 파일(확장자), 2. Map을 선택하지 않는 경우

우선 파일을 선택하긴 했으나 올바르지 않은 확장자를 가져온 유저의 경우에는 올바른 확장자를 가져올 때까지 JFileChooser를 열었습니다.

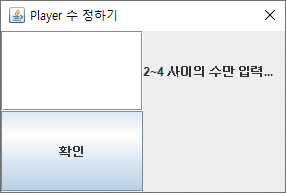
하지만 Map파일을 선택하지 않고, JFileChooser의 닫기 버튼을 눌러버리는 경우에는 해당 프로그램의 resource/map 폴더에 저장되어있는 default.map 파일을 불러와 게임을 시작합니다.

* PlayerPicker

그 후에는 PlayerPicker, 즉 플레이어의 수를 정해야했습니다.

Player의 수는 modal 다이얼로그(InputNumPlayer클래스)를 통해 값을 입력받았습니다. modal 다이얼로그는 해당 창이 끝날때까지 다른 창의 업무가 중단되기 때문에, 다른 스레드에서 동작해버려 값을 받아오지 못하는 문제를 해결할 수 있었습니다.

자연스러운 흐름을 위해 map을 고르는 행위가 끝나면 곧바로 player의 수를 입력하게 합니다.



JLabel을 통해 가능한 Player수를 안내합니다.

하지만 그럼에도 불구하고 해당 범위의 밖의 수를 입력하거나 문자를 입력하는 상황이 발생할 수 있습니다.

따라서 우선 값을 입력받는 TextField를 JFormattedTextField 클래스를 사용하여 NumberFormat만 허용하도록 하였습니다.

|  |
| --- |
| NumberFormat numberFormat = NumberFormat.getNumberInstance();  numPlayerTextField = new JFormattedTextField(numberFormat); |

그 후, 원하는 값이 들어올 때까지 창을 지속적으로 실행시켜 다른 활동이 일어나지 않도록 하였습니다.

|  |
| --- |
| while(inputNumPlayer.getNumPlayer()<2 || inputNumPlayer.getNumPlayer()>4){  inputNumPlayer.setVisible(true);  } |

올바르게 Player 인원 수까지 입력을 완료하였다면 이제 Map GUI를 구현해야합니다.

* GameFrame – Map GUI

Map GUI는 각 셀의 특성을 담고있어야 했습니다.

Start, Cell, Hammer, PhilipsDriver 등 다양한 셀의 종류를 유저에게 알리기 위해서는 해당 부분을 표시할 고려를 하며 UI를 제작해야했습니다.

우선 해당 게임의 Map을 배열로 구현하였기 때문에, Map GUI또한, 이미지를 배열로 나열하여 표시하였습니다.

JComponent에서는 그림을 그리기 위한 메소드 paintComponent가 존재합니다.

이와 for문을 활용해 각 좌표에 맞게 이미지를 뿌렸습니다.

먼저, 이미지는 포토샵을 이용하여 직접 그려 제작하였습니다. 포토샵을 평소에 자주 사용하였기 때문에 이를 활용하여 이미지를 만들 수 있었습니다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bridge | Cell | End | Hammer | NoTILE | Philips | saw | start |
|  |  |  |  |  |  |  |  |

먼저 해당 Cell의 카드의 종류, 및 시작위치 끝위치인지를 확인하고 종류에 따라 g.drawImage();를 활용하여 알맞은 그림을 알맞은 좌표에 뿌리도록 하였습니다.

제작한 이미지의 크기는 가로 세로 50px이었으므로 len을 50으로 설정한 후, 해당 배열의 좌표에 세로, 가로 각 len만큼 곱해주면 타일의 위치가 됩니다.

Bridge는 CardType이 b인 Cell를 그릴 때 왼쪽에 함께 그리도록 하였습니다 (맵이 왼쪽부터 오른쪽으로 그려짐) 만약 B일 때 오른쪽에 함께 그리면, 이후에 실제 Bridge이미지가 그려진 위치는 NoTile이 그려져야하는(Cell이 없는) 위치였으므로 NoTile이미지가 그 위를 대체하기 때문입니다.

이를 토대로 default.map을 불러오면 다음과 같은 맵을 생성합니다.

텍스트, 다채로운이(가) 표시된 사진

자동 생성된 설명

* Supplementary Specification

|  |  |
| --- | --- |
| **Revision History** | |
| Version | Description |
| 1 | 첫 문서. |
| 2 | 콘솔 기능 제공 삭제 |

|  |
| --- |
| **소개** |
| Use Case에는 포함되지 않는 공통사항 등에 대해 서술된 문서이다. |

|  |
| --- |
| **GUI 지원** |
| 해당 게임은 GUI기반으로 플레이된다. |

# ELABORATION 2

## 핵심개념

* 첫번째 Iteration이 끝난 후 두번째 반복입니다.
* 그 다음으로 핵심적이고, 리스크가 높은 아키텍처가 프로그래밍되고 테스트됩니다.

ELABORATION단계에서 시작되는 ARTIFACT 목록

|  |  |
| --- | --- |
| aRTIFACT | 작성 여부 |
| * Domain Model | **V** |
| * Design Model | V |
| * Software Architecture Document | X |
| * Data Model | X |
| * Use-Case Story Boards, UI Prototypes | V |

## Full-Dressed Use case

지난번에는 UseCase Diagram을 통해 보여준 UseCase들 중, 가장 핵심적인 UseCase에 대해서 먼저 파악하였습니다.

게임은 이제 거의 다 준비되었습니다. 이제 Player의 이름을 정하고, 순서를 정해주면 준비가 완료됩니다.

따라서 이번 Elaboration에서는 Ready UseCase에 대해 명세하겠습니다.

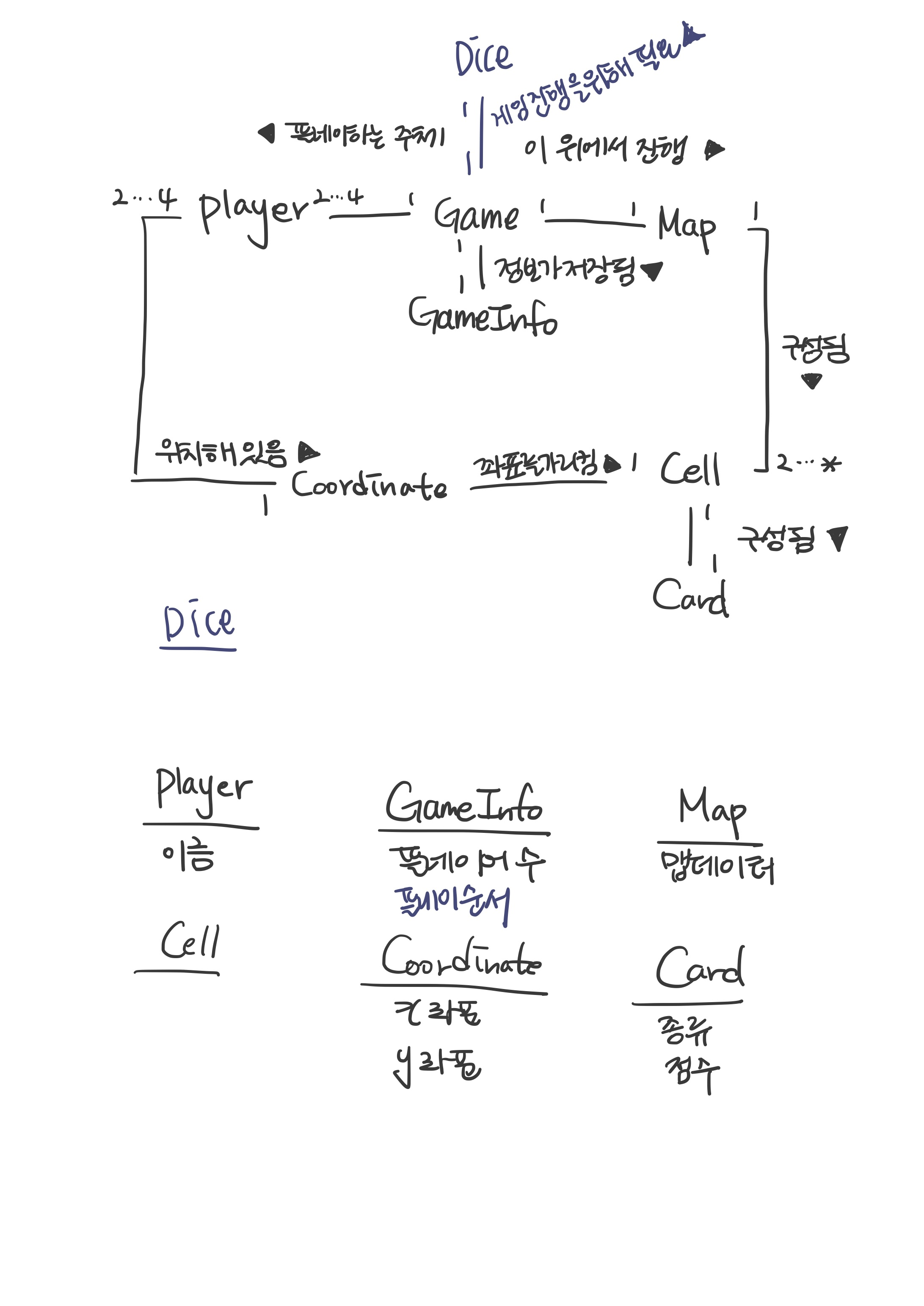
|  |  |  |
| --- | --- | --- |
| Usecase #Ready Game | | |
|  | |  |
| SCOPE | Bridge Game Program | |
| LEVEL | User Goal | |
| Primary Actor | Player | |
|  | |  |
| STAKEHOLDERS AND INTERESTS | | |
| Player | * Game을 플레이하고 싶어한다 * 본인의 이름을 정하고 싶어한다. * 주사위를 굴려 순서를 명확히 하고자한다. | |
|  | |  |
| PRECONDITIONS | | |
| * Player의 수는 이미 정해진 상황이다. | |  |
|  | | |
| SUCCESS GUARANTEE | | |
| * Player 정보를 얻는다 * Play 순서가 정해진다. * 게임이 본격적으로 시작된다 | | |
|  | | |
| MAIN SUCCESS SCENARIO | | |
| 1. Player수에 맞게 순차적으로 이름을 입력한다. 2. 주사위를 굴린다(Roll Dice UseCase) 3. 나온 값에 따라 Play 큰수부터 작은수까지로 순서를 지정한다. (Decide Turn UseCase) 4. 플레이 순서대로 재배치된다. 5. Start 좌표에 Player를 위치시킨다. 6. 게임을 시작한다. | | |
|  | | |

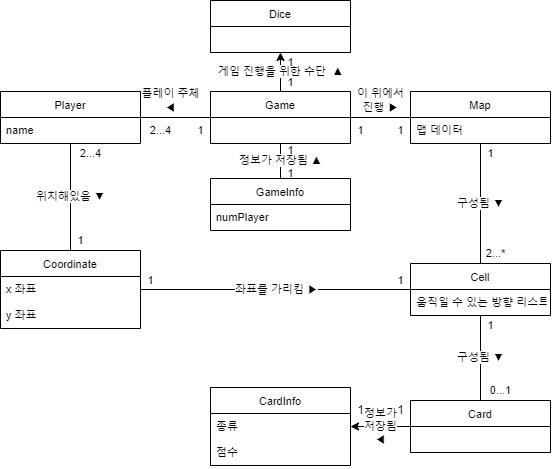
|  |
| --- |
| **extensions** |
| 1a. 이름을 입력하지 않고 확인을 누른다.   1. “Player#번호” 로 이름을 자동 생성한다.   3a. 두명 이상의 유저에게서 같은 숫자가 나온다.   1. 같은 숫자가 나온 유저끼리는 랜덤으로 순서가 정해진다. |
|  |

## Domain model

지난 Domain Model에 추가하여 해당Elaboration에서 추가로 명세된 UseCase를 토대로 수정하였습니다.

이는 추후 반복 회차를 거듭하며 세분화되거나 추가될 수 있습니다.



.

## System Sequence diagram

* Ready Game 시나리오

테이블이(가) 표시된 사진

자동 생성된 설명

## operation contracts

|  |  |
| --- | --- |
| Contract c01: setname | |
| Operation | setName(name: String) |
| Cross References | Use Cases: SetName |
| PreConditions | Player 객체 내 Name 속성이 존재한다. |
| PostConditions | * Player의 이름이 결정되었다. |

|  |  |
| --- | --- |
| Contract c02: rolldice | |
| Operation | rollDice() |
| Cross References | Use Cases: RollDice |
| PreConditions | 없음 |
| PostConditions | * 랜덤의 주사위 값을 얻었다. |

|  |  |
| --- | --- |
| Contract c03: randomorder | |
| Operation | randomOrder(overlapPlayers:Player[]) |
| Cross References | Use Cases: Decide Turn |
| PreConditions | 동일한 값이 나온 Player들이 있다. |
| PostConditions | * 해당 Player들에게 순서를 임의로 부여하여 플레이시 순서가 꼬이지 않도록 하였다. |

|  |  |
| --- | --- |
| Contract c04: SetStart | |
| Operation | setStart(player:Player) |
| Cross References | Use Cases: Ready Game |
| PreConditions | Player의 순서가 모두 다 정해졌다. |
| PostConditions | * Player들을 시작점에 위치시켜 게임을 시작할 준비를 완료하였다. |

## Implementation

구현 과정에서, Decide Turn에 수정사항이 생겼습니다.

순서는 랜덤으로 정해지기 때문에 주사위를 돌려 번거롭게 재정렬하는 행위는 효율적이지 못하다고 판단되었습니다. 따라서 ReadyGame의 내용을 대폭 수정하였습니다.

|  |  |  |
| --- | --- | --- |
| Usecase #Ready Game | | |
|  | |  |
| SCOPE | Bridge Game Program | |
| LEVEL | User Goal | |
| Primary Actor | Player | |
|  | |  |
| STAKEHOLDERS AND INTERESTS | | |
| Player | * Game을 플레이하고 싶어한다 * 본인의 이름을 정하고 싶어한다. * 주사위를 굴려 순서를 명확히 하고자한다. | |
|  | |  |
| PRECONDITIONS | | |
| * Player의 수는 이미 정해진 상황이다. | |  |
|  | | |
| SUCCESS GUARANTEE | | |
| * Player 정보를 얻는다 * Play 순서가 정해진다. * 게임이 본격적으로 시작된다 | | |
|  | | |
| MAIN SUCCESS SCENARIO | | |
| 1. Player수에 맞게 순차적으로 이름을 입력한다. 2. 플레이 순서가 랜덤으로 정해진다. 3. Start 좌표에 Player를 위치시킨다. 4. 게임을 시작한다. | | |
|  | | |

Ready Game은 Game 클래스의 public void ready() 메소드를 통해 구현되었다.

|  |
| --- |
| public void ready(){  MakeGame.setName(this.gameInfo.getPlayers()); //player 이름 정하기  makeTurn();  setPlayerStartLocation();  } |

먼저 MakeGame의 static 메소드 setName을 통해 플레이어의 이름을 정합니다.

해당 함수가 호출되면 modal JDialog 객체가 생성되어 Visible하게 처리되며,

해당 Dialog에는 이름을 입력하는 칸과, 확인 버튼이 존재합니다.

텍스트이(가) 표시된 사진

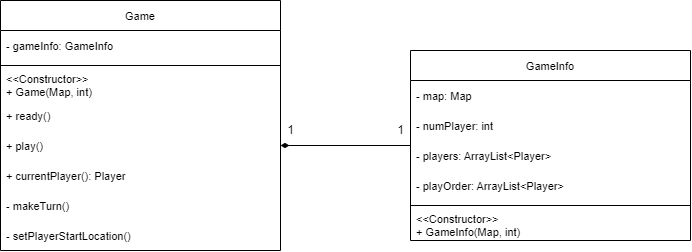
자동 생성된 설명

더 이상 이름을 정할 Player가 없을 때까지, 확인버튼을 누르면 다음 유저로 넘어갑니다.

만약 이름을 정하지 않고 빈칸으로 확인을 누를 경우, Player#번호 로 이름이 자동 지정됩니다.

만약 더 이상 남은 Player가 없으면 makeTurn()메소드를 통해 순서를 랜덤으로 지정합니다.

순서를 지정하기 위해, 기존의 GameInfo에는 새로운 속성 playOrder가 생성되었습니다.



|  |
| --- |
| private void makeTurn(){  for(int i = 0; i<gameInfo.getNumPlayer(); i++){  gameInfo.getPlayOrder().add(gameInfo.getPlayers().get(i));  }  int maxShuffle = new Random().nextInt(10);  for(int i = 0;i<maxShuffle;i++) {  Collections.shuffle(gameInfo.getPlayOrder());  }  } |

makeTurn()메소드에서 기존 gameInfo.players 변수의 player 객체들은 playOrder에 add됩니다. 이때 playOrder에 바로 players를 할당하지 않은 것은, playOrder은 순서가 정해져있는 리스트로 makeTurn()메소드에서 shuffle될 것인데, 이때 players리스트까지 함께 순서가 바뀌지 않도록 하기 위해서입니다.

이후, 항상 똑 같은 순서로 shuffle되는 것을 막기 위해, 0부터 9번까지의 shuffle할 횟수를 정하여 maxShuffle 변수에 할당합니다.

마지막으로 Collections 클래스의 shuffle 메소드를 활용하여 정해진 maxShuffle횟수만큼 shuffle을 진행합니다.

결과는 다음과 같습니다. 아래의 사진들은 게임을 3회 실행하였을 때 매 게임마다 정해진 순서를 콘솔로 출력한 것입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

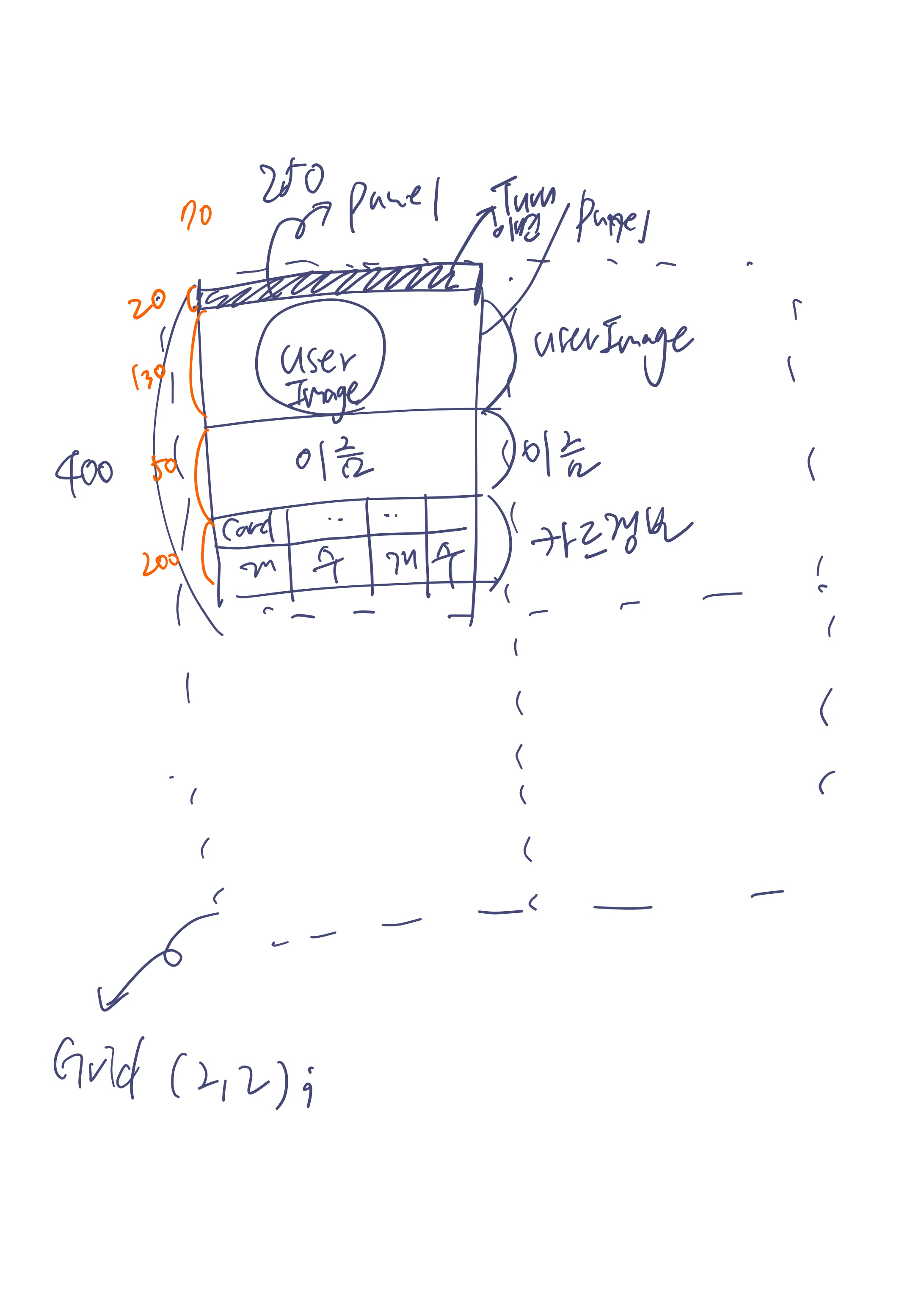
마지막으로 setPlayerStartLocation() 메소드를 통해 플레이어들을 모두 시작 위치에 위치시킵니다.

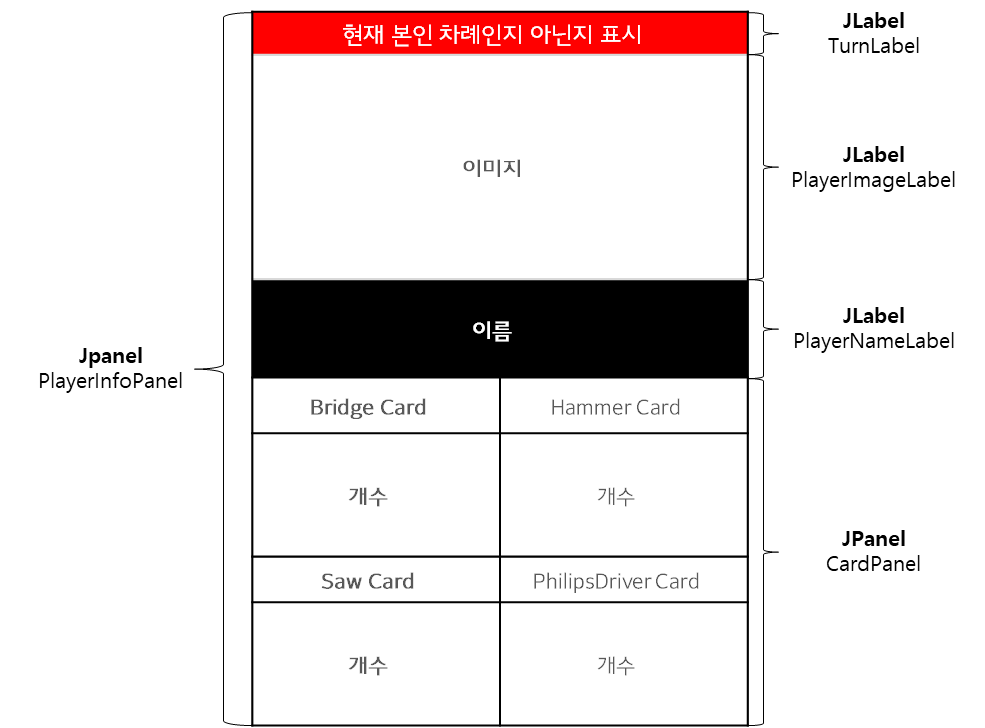
Map의 시작 Coordinate에 대한 정보는 gameInfo.getMap().getStartCoordinate()을 통해 얻을 수 있습니다.

* GUI

플레이어의 정보 입력 또한 완료하였으므로 이제 Player의 정보를 화면에 띄울 시간입니다.

먼저 Player UI에 필요할 내용들을 구상하였습니다.





Player의 정보는 각각 PlayerInfoPanel에 저장됩니다.

표시되는 정보는 다음과 같습니다.

1. 현재 차례

2. Player 이미지

3. Player 이름

4. Player 보유 카드

해당 PlayerInfoPanel은 JDialog의 하위클래스인 PlayDialog 클래스의 객체에 add()되어 표시됩니다.

PlayDialog는 GridLayout으로 레이아웃 구성되어있으며, 행,열 각각 2\*2 혹은 1\*2로 구성된다.

만약 Player수가 2명이라면 1\*2, 3명 이상이라면 2\*2로 구성된다.

텍스트, 슬롯머신, 창문, 클립아트이(가) 표시된 사진

자동 생성된 설명 텍스트, 창문이(가) 표시된 사진

자동 생성된 설명

[2\*2] [1\*2]

PlayerImageLabel의 이미지는 MapUI에서 유저를 표시할때도 사용됩니다.

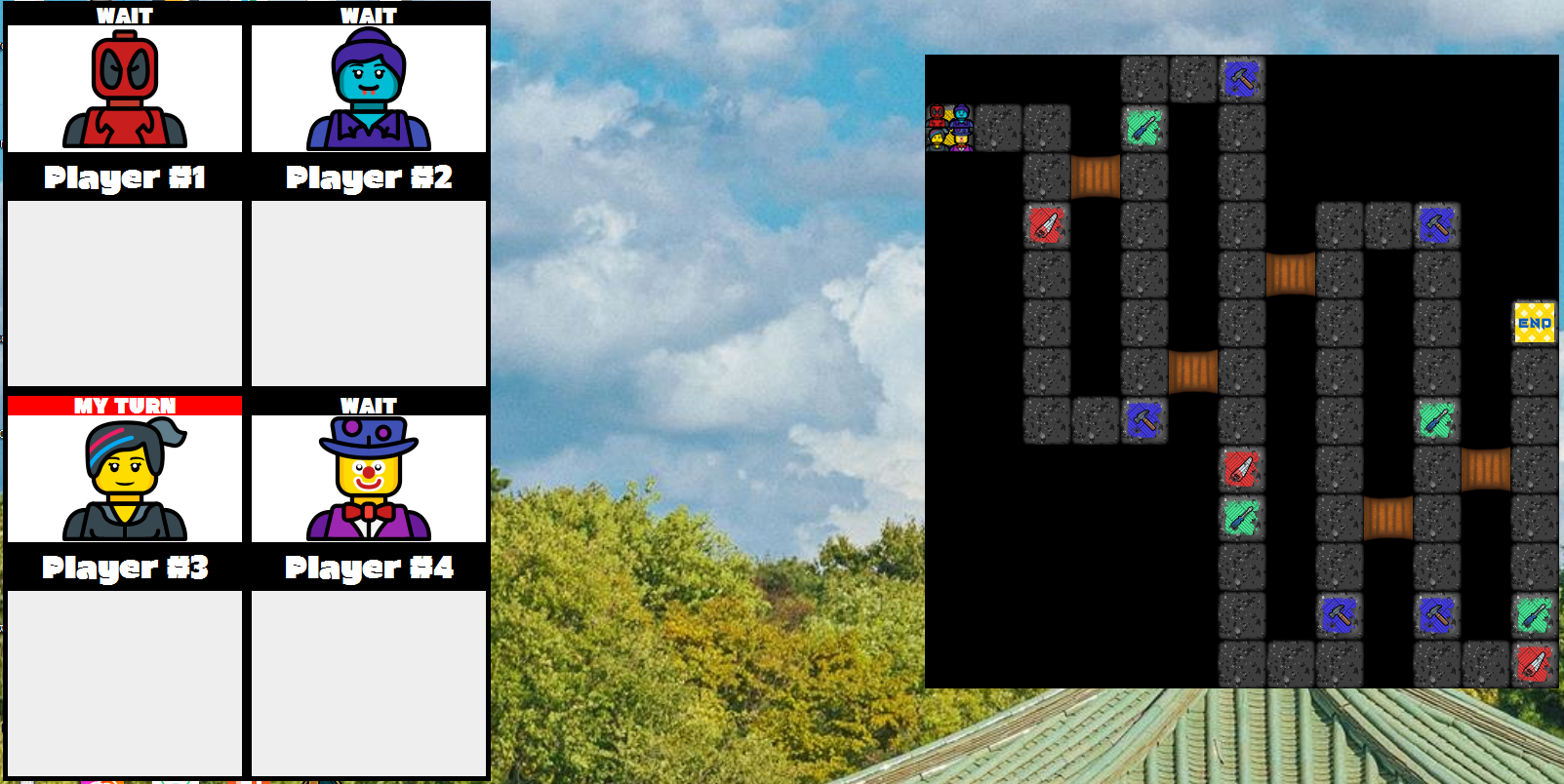
각 Cell에서 Player 1은 좌측 상단, Player 2는 우측 상단, Player 3은 좌측 하단, Player 4는 좌측 하단에 그려집니다. 이는 당연히 현재 플레이 중인 Player 수에 따라 그려집니다.

텍스트, 다채로운이(가) 표시된 사진

자동 생성된 설명 텍스트, 창문이(가) 표시된 사진

자동 생성된 설명

Ready Game까지의 화면 구성은 다음과 같습니다.



# ELABORATION 3

## 핵심개념.

* 그 다음으로 핵심적이고, 리스크가 높은 아키텍처가 프로그래밍되고 테스트됩니다.

ELABORATION단계에서 시작되는 ARTIFACT 목록

|  |  |
| --- | --- |
| aRTIFACT | 작성 여부 |
| * Domain Model | **X** |
| * Design Model | V |
| * Software Architecture Document | X |
| * Data Model | X |
| * Use-Case Story Boards, UI Prototypes | V |

## Full-Dressed Use case

게임은 이제 다 준비되었습니다. 이제 순서에 맞춰서 게임을 플레이하면 됩니다.

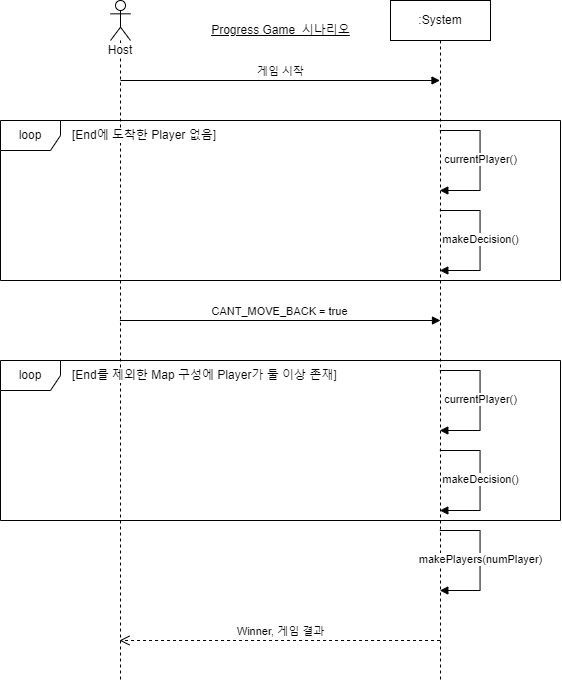
|  |  |  |
| --- | --- | --- |
| Usecase #Progress game | | |
|  | |  |
| SCOPE | Bridge Game Program | |
| LEVEL | User Goal | |
| Primary Actor | Host | |
|  | |  |
| PRECONDITIONS | | |
| * 모든 게임 준비가 완료된 상황이다. * Player의 게임 순서 또한 확정되었다. | | |
|  | | |
| SUCCESS GUARANTEE | | |
| * 게임을 끝낸다 * Winner를 정할 수 있다. | | |
|  | | |
| MAIN SUCCESS SCENARIO | | |
| 1. 첫번째 순서 Player에게 기회를 제공합니다 (Make Decision)   어떤 Player가 End에 도착할 때까지 순서에 따라 1을 반복합니다.   1. 한 Player가 End Cell에 도착합니다. 2. 이후부터 남은 Player는 뒤로 돌아갈 수 없게 합니다.   한 Player만 Board에 남을 때까지 남은 Player가 본인의 순서에 따라 1을 반복합니다.   1. 기회 제공을 멈추고, 보유한 카드 개수 및 도착한 순서에 따라 점수를 계산합니다. 2. 점수가 제일 높은 Player를 Winner로 설정하고, 알려줍니다. 3. 게임이 종료됩니다. | | |
|  | | |

따라서 이번 Elaboration에서는 Progress Game과 Game 플레이의 핵심인 Make Decision UseCase에 대해 명세하겠습니다.

|  |  |  |
| --- | --- | --- |
| Usecase #make decision | | |
|  | |  |
| SCOPE | Bridge Game Program | |
| LEVEL | User Goal | |
| Primary Actor | Player | |
|  | |  |
| PRECONDITIONS | | |
| * 해당 Player가 기회를 제공받았다. | | |
|  | | |
| SUCCESS GUARANTEE | | |
| * Player가 본인의 행동을 선택할 수 있습니다. | | |
|  | | |
| MAIN SUCCESS SCENARIO | | |
| 1. 해당 Player가 주사위를 굴려 이동 가능한 값을 얻습니다 (Roll Dice) 2. 이동가능한 값만큼 유효한 이동 방향을 입력합니다. 3. 이동경로 유효성 체크를 통과합니다. 4. Player은 이동경로를 따라 이동합니다. 5. 도착한 Cell에 Card가 있을 경우, 해당 Card를 획득합니다. 6. 턴을 종료합니다. | | |
| |  | | --- | | **extensions** | | 1a. 휴식을 선택합니다. (Take a Break)   1. 해당 Player가 보유중인 Bridge Card가 있다면 하나 삭제합니다. 2. 다음 순서 Player에게 기회가 넘어갑니다.   1b. Bridge Card를 가지고 있습니다.   1. 얻은 주사위 값에서 보유한 Bridge Card 개수만큼 뺀 값을 이동 가능한 값으로 설정합니다.   2a. 유효하지 않은 이동방향을 입력했습니다.   1. 해당 Player에게 이동방향 재입력을 요청합니다.   유효한 이동방향을 입력할 때까지 1을 반복합니다.   1. 해당 방향으로 이동합니다.   2b. End에 도착한 플레이어가 있는데 뒤로 가는 방향을 입력했습니다.   1. 유효성 체크에서 실패해 재 입력을 요청합니다.   유효한 이동방향을 입력할 때까지 1을 반복합니다.   1. 해당 방향으로 이동합니다. | | | |

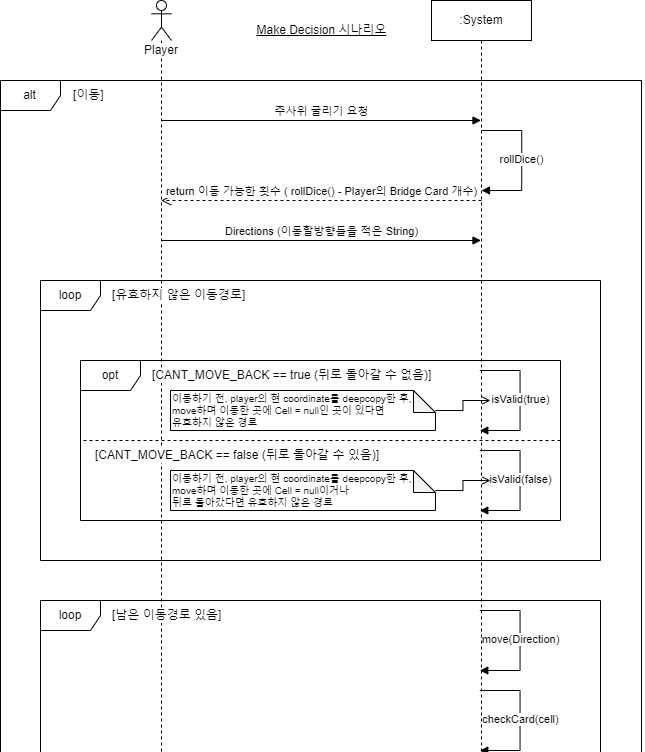
## System Sequence diagram

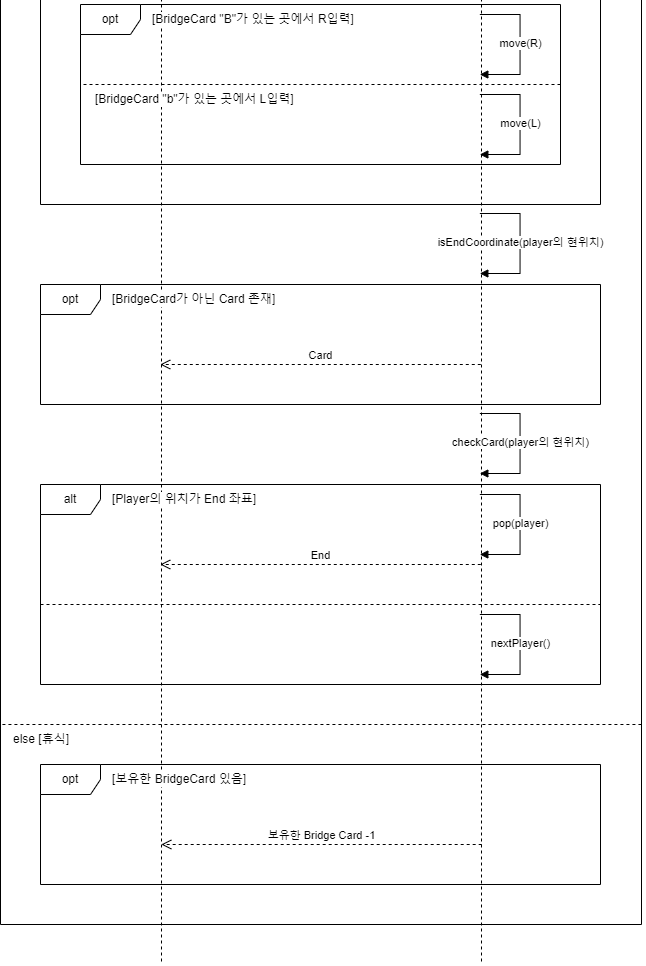
* Progress Game시나리오



calculateScore()

* Make Decision 시나리오





## operation contracts

|  |  |
| --- | --- |
| Contract c01: CurrentPlayer | |
| Operation | currentPlayer() |
| Cross References | Use Cases: Progress Game, Make Decision |
| PreConditions | Player의 순서가 정해졌다. |
| PostConditions | * 현재 차례인 Player를 알게되었다. |

|  |  |
| --- | --- |
| Contract c02: calculateScore | |
| Operation | calculateScore() |
| Cross References | Use Cases: Progress Game |
| PreConditions | 한명의 Player를 제외하고 모두 End에 들어왔다. |
| PostConditions | * Winner가 누군지 알 수 있다. * 게임이 완전히 종료된다. |

|  |  |
| --- | --- |
| Contract c03: isvalid | |
| Operation | isValid(canNotMoveBack: Boolean, directions: String) |
| Cross References | Use Cases: Make Decision |
| PreConditions | Player가 이동 경로를 설정했다. |
| PostConditions | * 해당 방향이 유효한 이동인지 확인할 수 있다 * 뒤로 돌아갈 수 있는 경우와 없는 경우 모두 체크할 수 있다. |

|  |  |
| --- | --- |
| Contract c04: nextPlayer | |
| Operation | nextPlayer() |
| Cross References | Use Cases: Make Decision |
| PreConditions | Player의 순서가 끝이 났다. |
| PostConditions | * 다음 순서를 지정했다. * currentPlayer에서 이제 다음 순서의 player객체를 얻을 수 있다. |

## Implementation