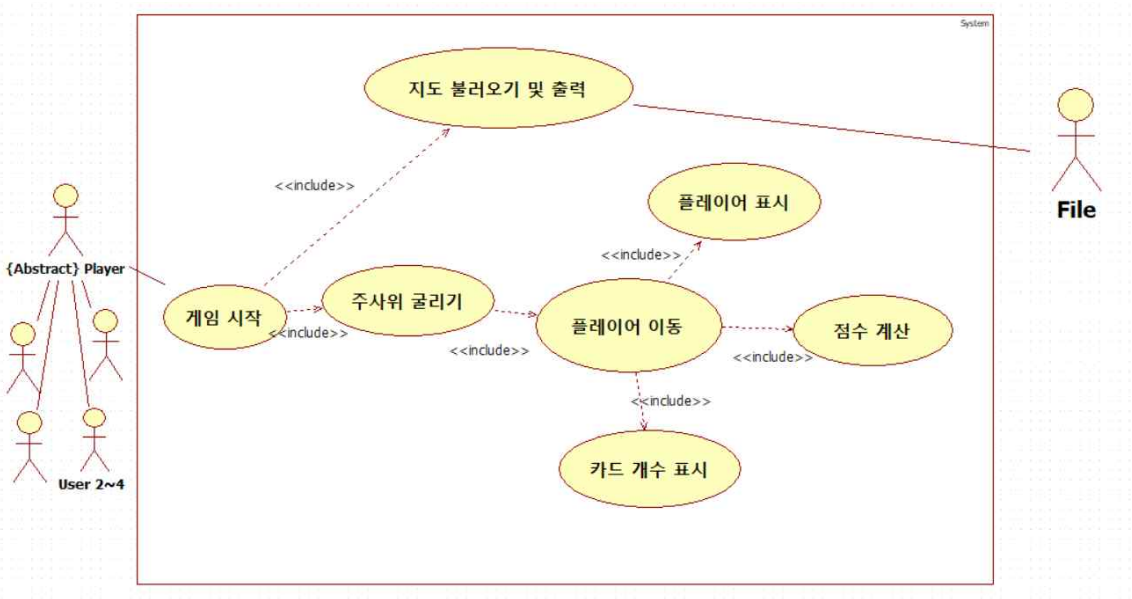
소프트웨어공학
Term Project

소프트웨어공학 02분반
20161283 최영찬

목차

1. 요구 정의 및 분석 산출물

(1) Usecase Diagram



2~4명의 Player가 게임을 시작합니다. 게임을 시작함과 동시에 지도를 불러오고 출력합니다. 그 후 주사위를 굴리고 그 결과에 따라 플레이어가 이동합니다. 플레이어가 이동한 후에 최종 위치를 지도위에 표시하고, 플레이어가 얻은 카드의 개수를 표시합니다. 게임이 종료되면 점수를 계산해 우승자를 정합니다.

(2) Usecase 명세

Usecase Brief Format을 표로 간단하게 나타냈습니다.

Use Case UC 1	게임 시작
Primary Actor	게임 플레이어
Preconditions	-
Main Success Scenario	1. 게임을 실행합니다. 2. 플레이어 수를 입력합니다.

Use Case UC 2	지도 불러오기 및 출력
Primary Actor	게임 플레이어
Preconditions	게임을 시작했습니다.
Main Success Scenario	1. 지도를 불러옵니다. 1.a 기본 맵을 불러온 뒤 특정 맵을 불러 올 수 있습니다. 2. 지도를 화면에 출력합니다.

Use Case UC 3	주사위 굴리기
Primary Actor	게임 플레이어
Preconditions	지도를 불러왔습니다.
Main Success Scenario	1. 주사위를 굴릴지 쉴지 결정을 합니다. 1.a 쉰다면 다리카드 1개를 반납합니다. 1.b 굴렸다면 주사위 눈의 수 - 다리카드 개수를 보여줍니다.

Use Case UC 4	플레이어 이동
Primary Actor	게임 플레이어
Preconditions	주사위를 굴렸습니다.
Main Success Scenario	1. 플레이어가 이동할 경로를 입력합니다. 1.a 이동할 수 없거나 주사위 값과 다르면 재입력합니다. 2. 이동하면서 다리를 건너면 다리카드를 얻고, 다른 도구 카드에 도착 시 해당 카드를 얻을 수 있습니다.

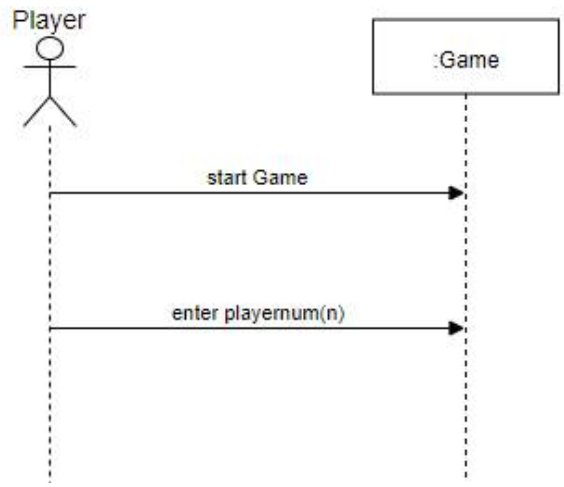
Use Case UC 5	플레이어 표시
Primary Actor	-
Preconditions	플레이어가 이동했습니다.
Main Success Scenario	1. 플레이어가 이동 후 해당 플레이어의 위치를 표시합니다.

Use Case UC 6	카드 개수 표시
Primary Actor	-
Preconditions	플레이어가 이동했습니다.
Main Success Scenario	1. 플레이어가 이동한 후 얻은 카드 개수를 표시합니다. 2. 한 턴이 종료되면 모든 플레이어의 카드 개수를 보여줍니다.

Use Case UC 7	점수 계산
Primary Actor	-
Preconditions	게임이 종료되었습니다.
Main Success Scenario	1. 각 카드마다의 점수의 합과 END를 넘는 순으로 얻은 점수의 합을 계산합니다. 2. 높은 점수 인 우승자를 출력합니다. 2.a 동점이라면 먼저 들어온 사람을 우승자로 합니다.

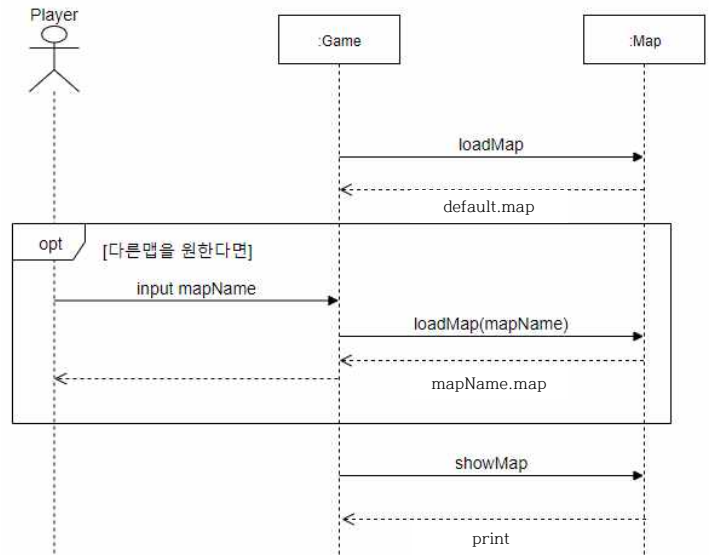
(3) System Sequence Diagram (SSD)

UC-1 게임 시작



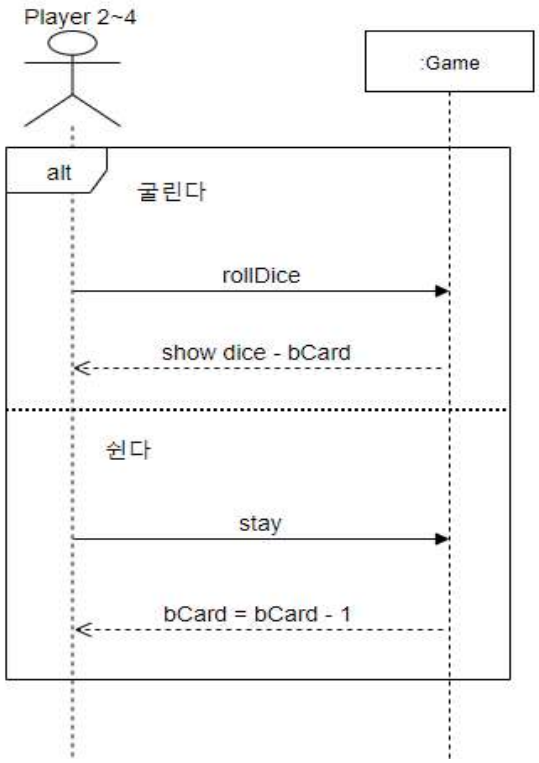
Player가 게임을 시작하면 Game object를 생성한 뒤 게임을 시작합니다. 그 후 플레이할 인원 수(n)을 입력받습니다.

UC-2 지도 불러오기 및 출력



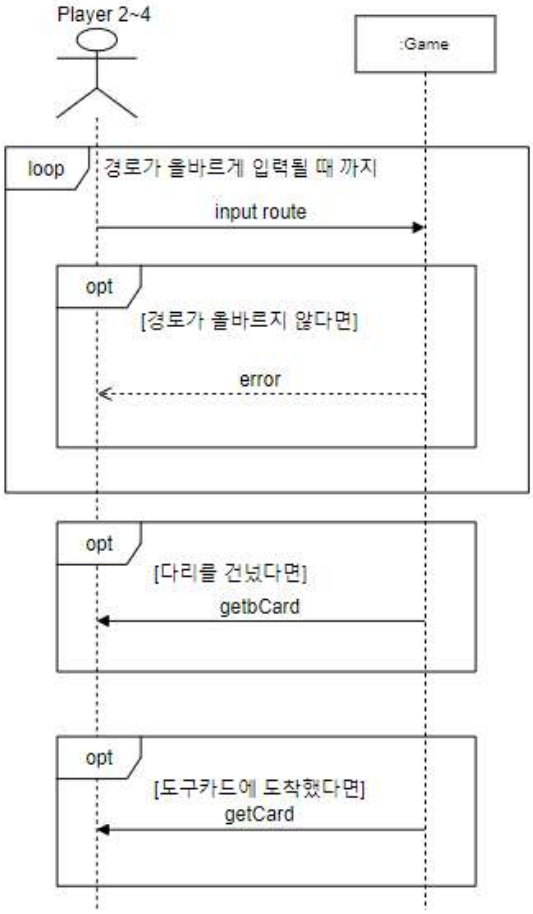
게임을 시작한 뒤 loadMap를 통해 default.map 파일을 불러옵니다. 만약 다른 지도파일을 불러오고 싶으면 지도파일 이름을 입력 한 뒤 loadMap를 통해 불러옵니다. 그 후 showMap를 통해 지도를 출력합니다.

UC-3 주사위 굴리기



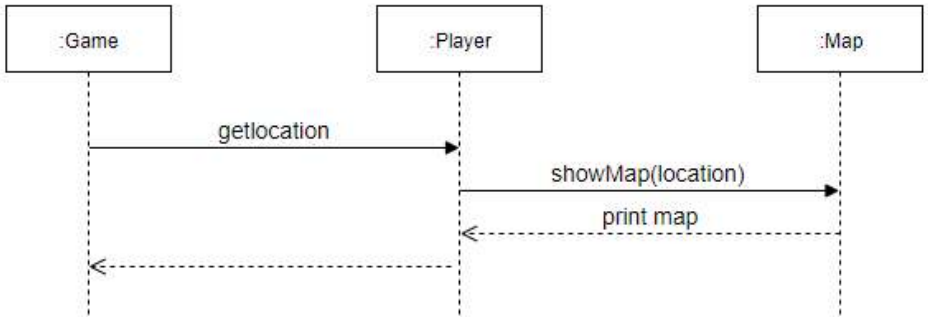
지도를 불러 온 뒤 매 턴마다 플레이어는 주사위를 굴릴지 쉴지 선택합니다. 주사위를 굴린다면 rollDice를 통해 주사위를 굴립니다. 주사위에서 나온 결과에서 본인이 가진 다리카드의 개수만큼 뺀 수를 보여줍니다. 만약 쉰다면 다리카드를 한 개 반납합니다.

UC-4 플레이어 이동



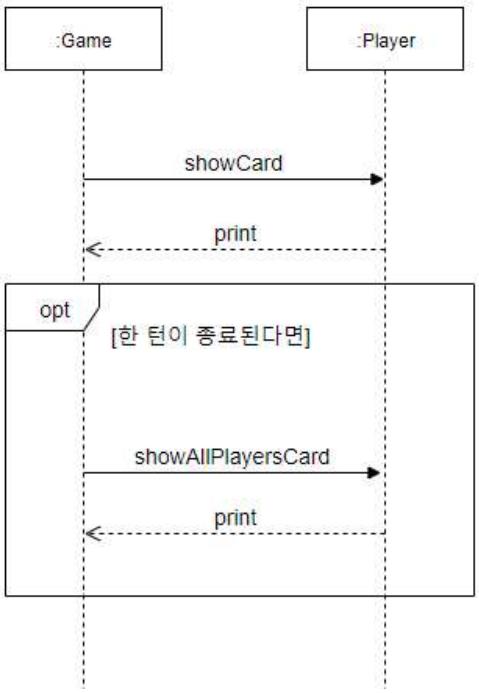
플레이어가 주사위를 굴린 후 경로가 올바르게 입력될 때 까지 경로를 입력 받습니다. 경로가 올바르면 이동을 합니다. 이동 중 다리를 건넜다면 getbCard를 통해 다리카드를 얻고, 도구카드에 도착했다면 getCard를 통해 해당 카드를 얻습니다.

UC-5 플레이어 표시



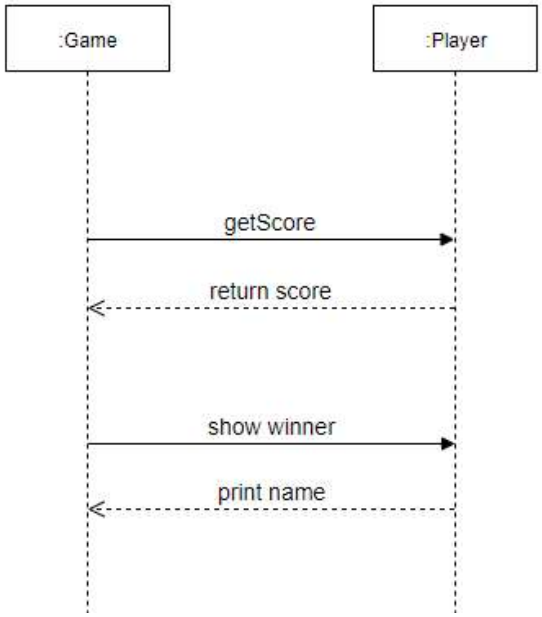
플레이어가 이동한 후 getLocation를 통해 player의 위치를 받고, showMap를 통해 플레이어의 위치를 지도에 나타냅니다.

UC-6 카드 개수 표시



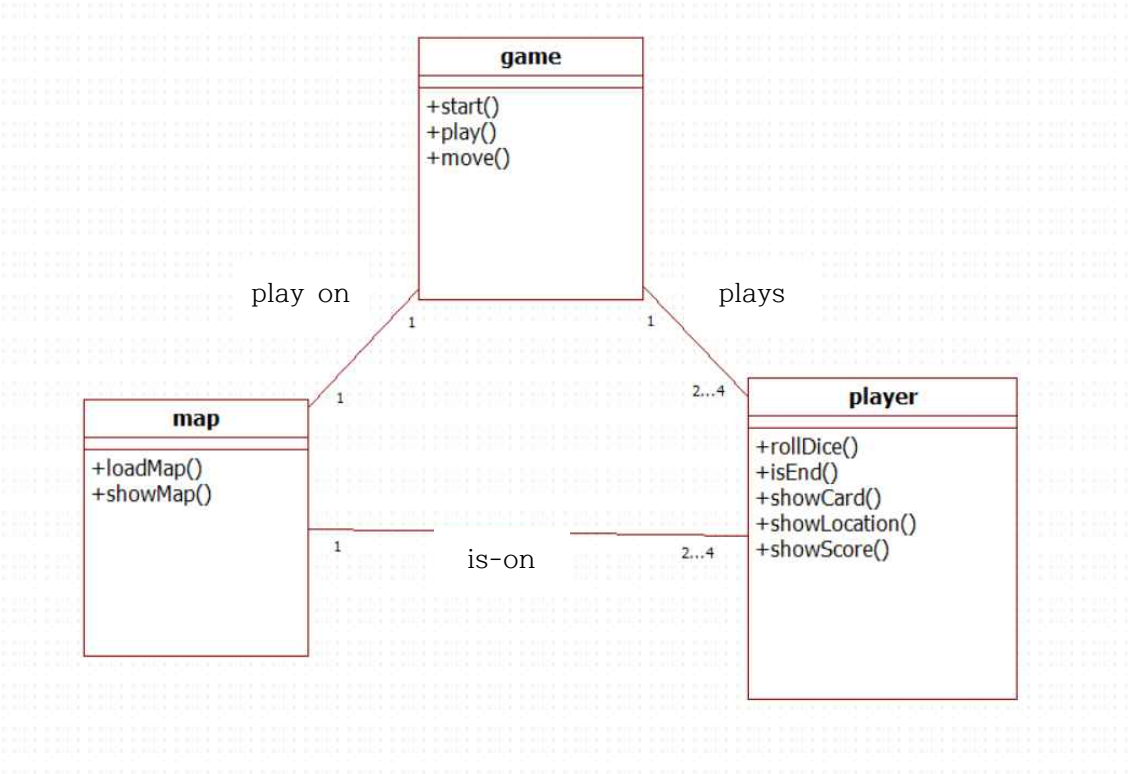
플레이어의 턴이 끝날 때 마다 showCard를 통해 각 플레이어가 가지고 있는 카드들을 보여줍니다. 한 턴이 종료되면 전체 플레이어의 카드개수를 보여줍니다.

UC-7 점수 계산



게임이 종료되면 getScore를 통해 점수를 계산하고 보여줍니다. 그 후 승리자의 이름을 출력합니다.

(4) Domain model

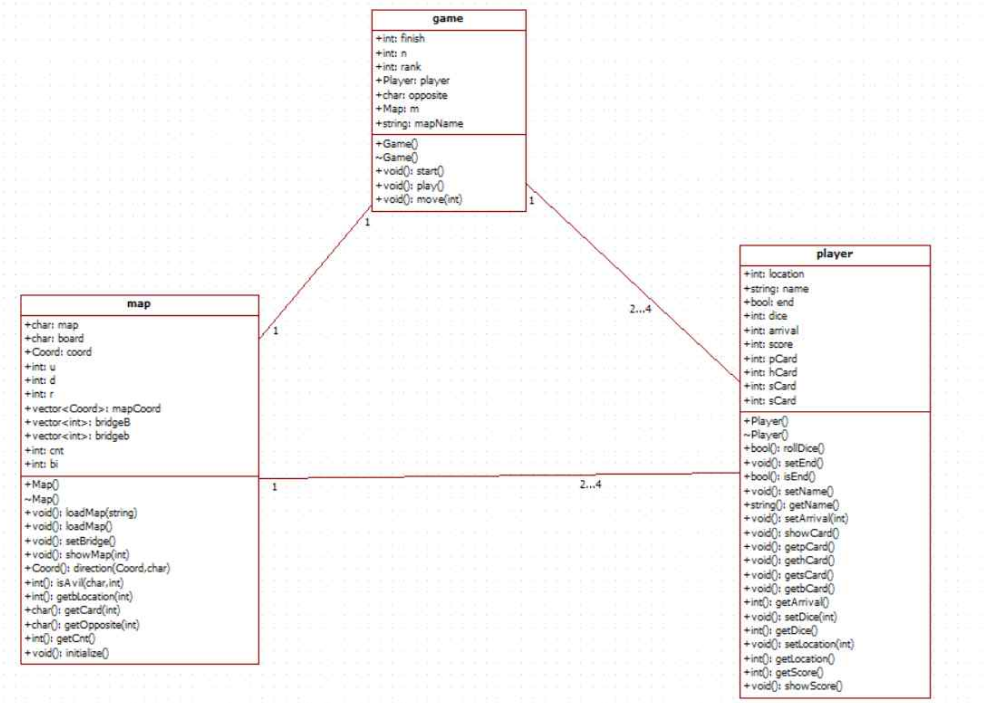


game class에는 게임을 시작하는 start(), 게임을 진행하는 play(), player를 움직이는 move()가 있습니다. map class에는 지도파일을 불러오는 loadMap(), 지도를 보여주는 showMap()가 있습니다. player class에는 주사위를 굴리는 rollDice(), 플레이어가 End를 넘었는지 판단하는 isEnd(), 플레이어가 가지고 있는 카드들을 보여주는 showCard(), 플레이어의 현재 위치를 보여주는 showLocation(), 점수를 보여주는 showScore()가 있습니다.

지도 파일 위에서 게임을 진행하기 때문에 game class와 map class는 play on 관계입니다. 한 개의 지도 파일을 불러와 게임을 진행하기 때문에 1대1 관계입니다. 지도 파일 위에 player의 위치를 표시하기 때문에 player class와 map class는 is-on 관계입니다. 2~4명의 플레이어가 게임을 진행하기 때문에 1대다 관계입니다. player가 게임을 진행하기 때문에 player class와 game class는 plays 관계 입니다. 한 개의 지도 파일 위에서 2~4명의 플레이어가 게임을 진행하기 때문에 1대다 관계입니다.

2. 설계 산출물

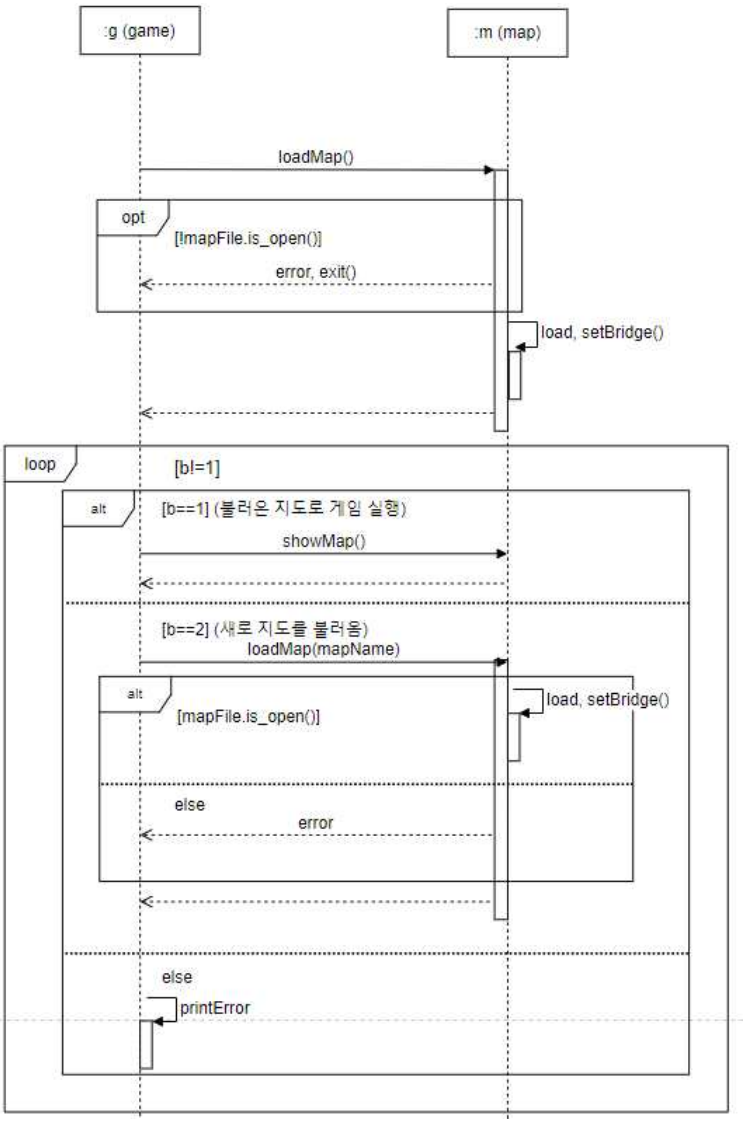
(1) Class Diagram



game	map	player
finish : end를 넘어 끝낸 사람의 수 n : 플레이 하는 플레이어의 수 rank : 들어간 순서 player[5] : 각 플레이어 객체, 편의상 1번부터 4번까지 사용하기 위해 5개 생성 opposite : end 로 들어오는 반대방향 m : 지도 객체 mapName : 입력받을 지도 이름	map : 지도데이터를 입력받아 저장하는 배열 board : 지도 좌표를 통해 지도 출력하는 배열 coord : 지도 데이터 좌표를 저장하는 구조체 u : 지도의 위쪽 끝 값 d : 지도의 아래쪽 끝 값 r : 지도의 오른쪽 끝 값 mapCoord : 지도 데이터의 좌표를 저장하는 벡터 bridgeB : B칸 순서대로 좌표를 저장하는 벡터 bridgeb : b칸 순서대로 좌표를 저장하는 벡터 cnt : 지도의 칸 수 bi : 다리의 index	location : player가 몇번째 칸에 있는지 저장 name : 플레이어의 이름 end : 골인 했는지 판단 arrival : 도착 순서 score : 합산 점수 pCard, hCard, sCard, bCard : 각각 얻은 카드들의 개수
start() : 게임을 시작 play() : 게임 세팅 후 진행 move(int) : 플레이어의 인덱스를 전달받아 해당 플레이어를 움직임	loadMap(string) : 입력받은 지도 이름의 파일을 불러옴 loadMap() : default.map 파일을 불러옴 setBridge() : bridgeB 벡터와 좌표들을 이용해 bridgeb에 도착 좌표를 저장하고 다리를 그림 showMap(int) : 지도를 그려줌 direction(Coord, char) : 지도 데이터를 이용해 다음 칸의 방향을 계산 isAvail(char, int) : 입력받은 방향이 이동 가능한 방향인지 계산 gebtLocation(int) : 다리를 건넌 후 위치 반환 getCard(int) : 도구카드를 반환 getOpposite(int) : End에서 반대 진행하는 방향 계산 getCnt() : 몇번째 칸인지 반환 initialize() : 새로운 지도를 저장하기 위해 기존의 변수들을 초기화	rollDice() : 주사위 굴리기 isEnd() : 플레이어가 골인했는지 판단 setName() : 이름 설정 getName() : 이름 반환 setArrival(int) : 도착 후 등수 저장 showCard() : 가지고 있는 카드 출력 getpCard() : P카드 획득 후 +1 gethCard() : H카드 획득 후 +1 getsCard() : S카드 획득 후 +1 getbCard() : B카드 획득 후 +1 getArrival() : 등수 반환 setDice(int) : 주사위 결과 저장 getDice() : 주사위 결과 반환 setLocation(int) : 현재 위치 저장 getLocation() : 현재 위치 반환 getScore() : 점수 계산 후 반환 showScore() : 점수 출력

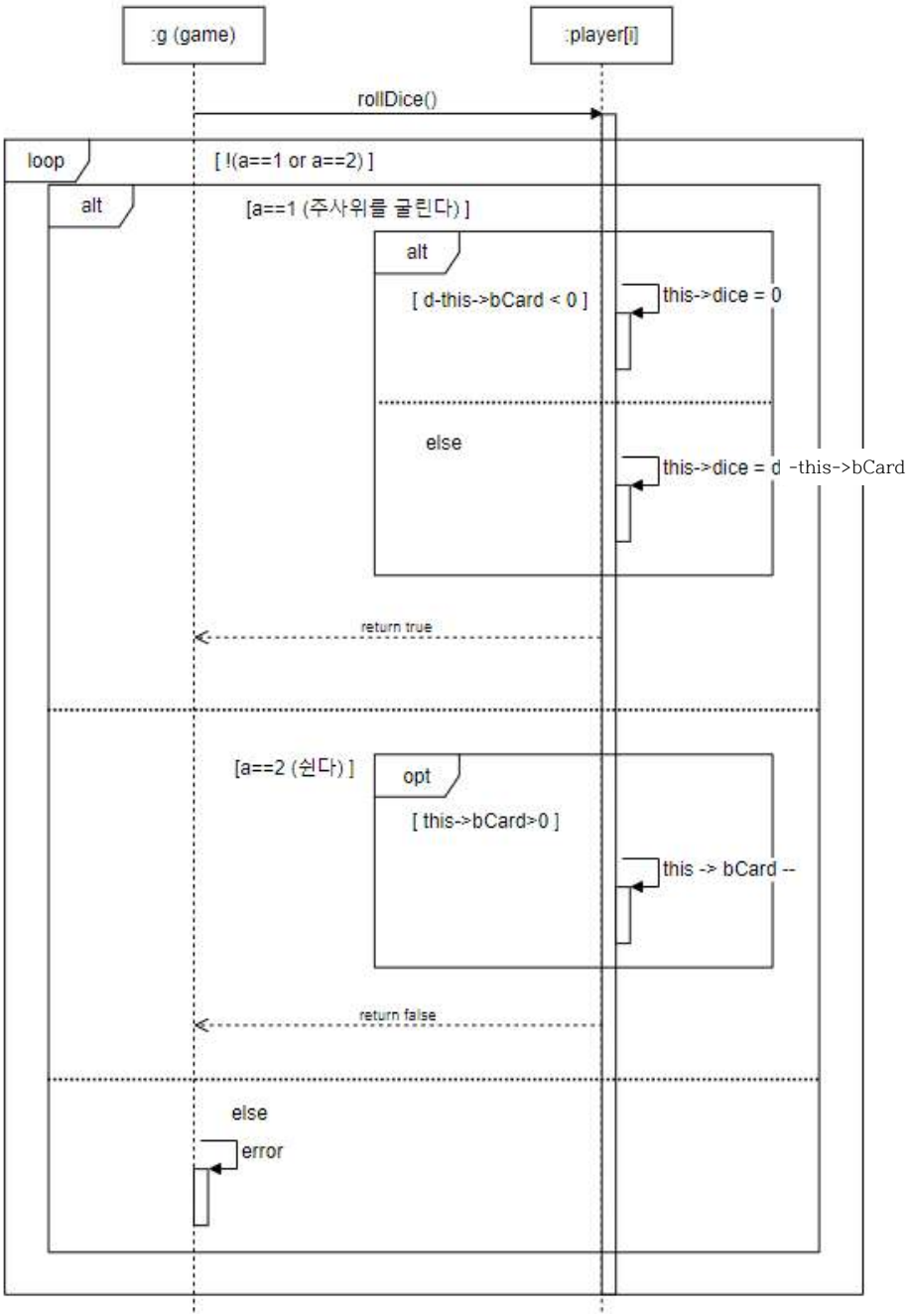
(2) Sequence Diagram

UC-2 지도 불러오기 및 출력



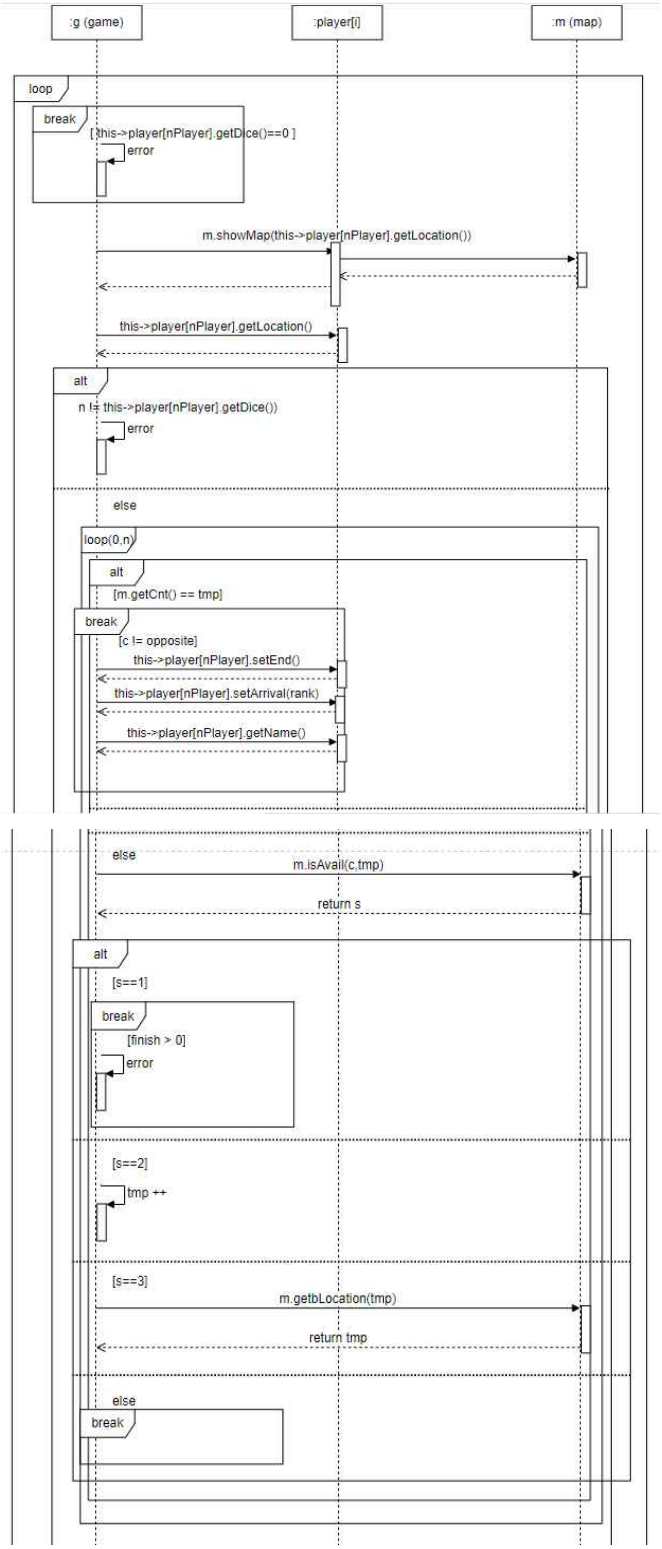
loadMap()을 이용해 default.map을 불러옵니다. 만약 해당 파일이 없다면 에러메시지를 출력하고 종료합니다. 그 후 1을 입력하면 showMap()을 통해 지도를 출력하고 게임을 진행합니다. 2를 입력하면 지도 이름을 입력받아 새로운 지도를 불러옵니다. 해당 파일이 없다면 에러메시지를 출력하고 다시 선택화면으로 갑니다. 이외의 값을 입력하면 에러메시지를 출력하고 다시 입력받습니다.

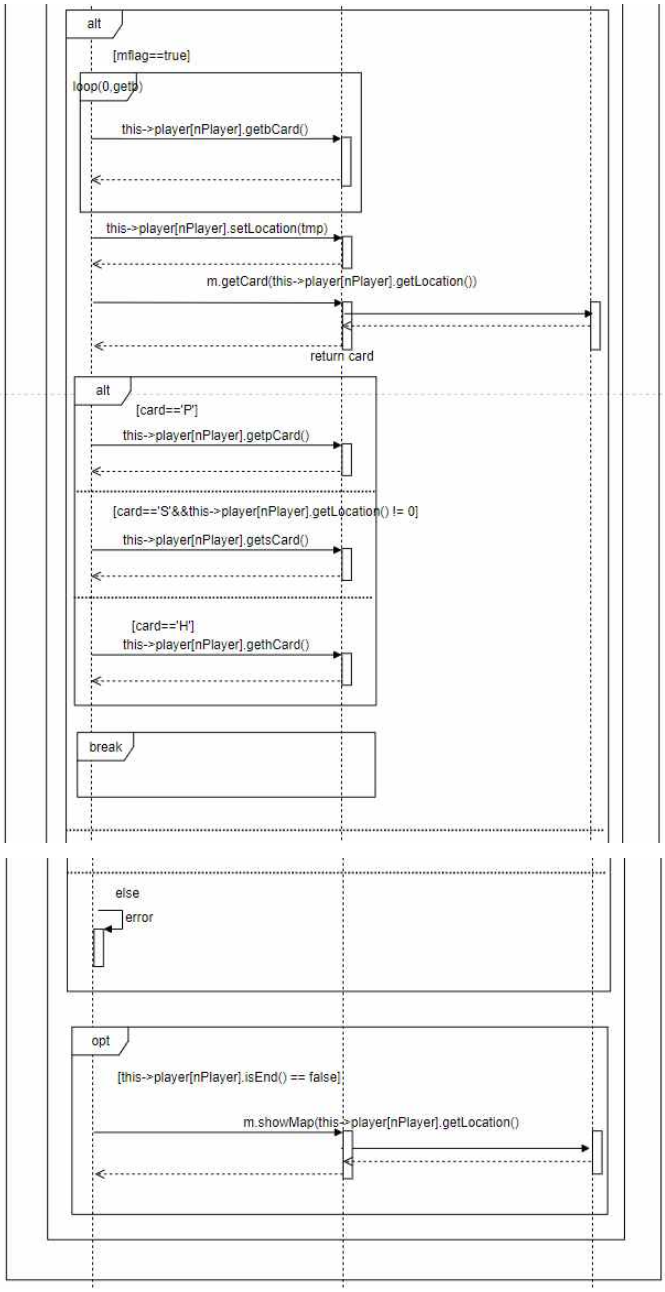
UC-3 주사위 굴리기



rollDice()를 통해 주사위를 굴립니다. 만약 1을 입력하면 주사위를 굴리고 true를 반환하고 2를 입력하면 쉽고 false를 반환합니다. 이외의 값을 입력하면 에러메시지를 출력 후 다시 입력받습니다. 주사위를 굴리면 주사위값에서 플레이어가 가지고있는 다리카드 개수를 뺍니다. 해당 값이 0보다 작으면 결과 값을 0 으로 합니다. 쉰다면 플레이어가 가지고 있는 다리카드의 개수가 0보다 크면 다리카드를 한개 반납합니다.

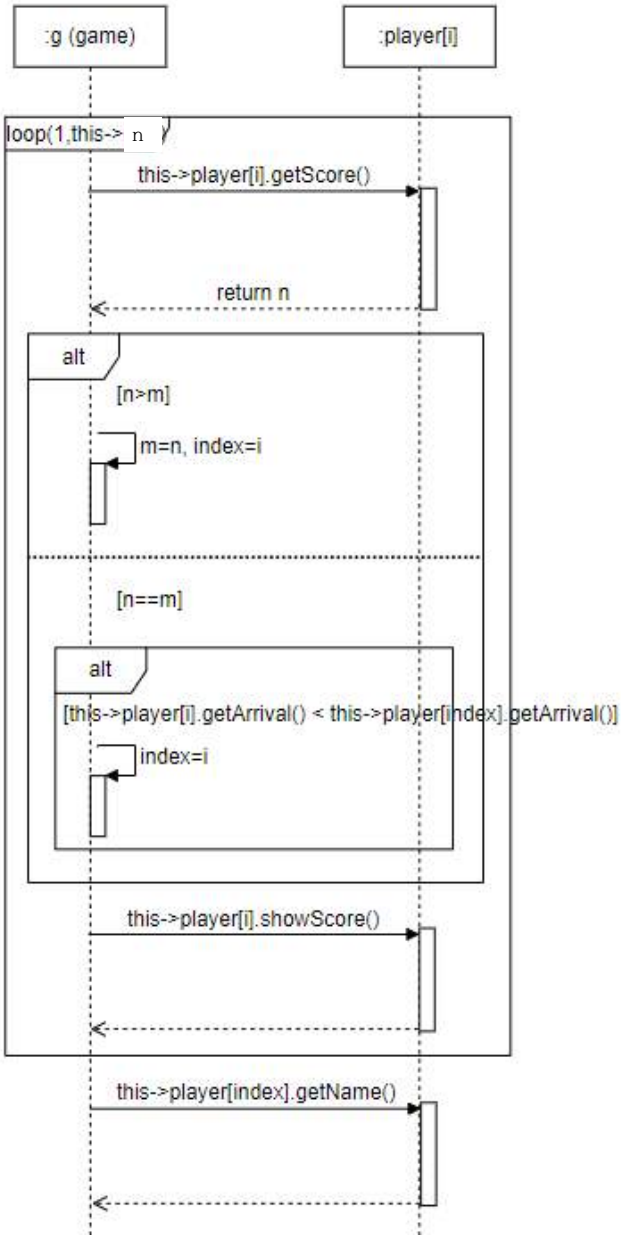
UC-4 플레이어 이동


:g (game)
:player[i]
:m (map)
loop
break
[this->player[nPlayer].getDice()==0]
error
m.showMap(this->player[nPlayer].getLocation())
this->player[nPlayer].getLocation()
alt
n != this->player[nPlayer].getDice())
error
else
loop(0,n)
alt
[m.getCnt() == tmp]
break
[c != opposite]
this->player[nPlayer].setEnd()
this->player[nPlayer].setArrival(rank)
this->player[nPlayer].getName()
else
m.isAvail(c,tmp)
return s
alt
[s==1]
break
[finish > 0]
error
[s==2]
tmp ++
[s==3]
m.getbLocation(tmp)
return tmp
else
break



만약 주사위 결과 값이 0이면 이동할 수 없고 break 합니다. 지도와 현재 위치를 보여주고 주사위 결과 값만큼 이동할 경로를 입력받은 뒤 플레이어의 현재 위치를 불러옵니다. 만약 입력한 경로의 수가 주사위 결과 값과 일치하지 않는다면 다시 입력받습니다. 입력받은 경로를 한 글자씩 판단합니다. 만약 현재 위치가 End라면 반대방향으로 나가지 않는다면 플레이어를 setEnd()로 골인시키고 setArrival()로 등수를 매깁니다. 그 후 getName()으로 이름을 받아와서 출력합니다. 뒤로 간다면 현재 위치를 뒤로 이동합니다. 만약 현재 위치가 End가 아니라면 isAvail()을 통해 가능한 경로인지 판단해 다시 입력을 받거나 해당 위치로 이동시킵니다. 이동을 완료하면 상황에 따라 카드들을 얻고 플레이어의 위치를 저장합니다. 골인을 안했다면 플레이어의 최종 위치를 지도에 출력합니다.

UC-7 점수 계산



1번 플레이어부터 n번 플레이어까지 getScore()를 통해 점수를 반환 받습니다. 만약 점수가 그 전 최대 점수보다 크면 점수와 플레이어의 index를 저장합니다. 만약 점수가 동점이라면 getArrival()을 통해 먼저 들어온 사람의 index를 저장합니다. 그 후 showScore()를 통해 플레이어의 점수를 출력하고 getName()을 통해 우승자의 이름을 출력합니다.

3. 구현 산출물

(1) 프로그램 사용방법

1. Bridge Game.exe와 맵 파일들을 한 폴더 안에 저장합니다.
2. Bridge Game.exe를 실행합니다.
3. 실행 후 콘솔 화면에 나오는 텍스트에 따라 플레이 합니다.

(2) 세부명세 가정 설명

- 지도의 최대 칸수는 100칸으로 제한합니다.
- 지도의 최대 크기는 20X20으로 제한합니다.
- start와 end가 각각 좌우 양 끝이라 가정합니다.
- 지도에는 결점이 없습니다.
- 다리는 한 칸으로 치지 않습니다. 즉, 다리를 건너면 B에서 b로 바로 넘어갑니다.
- 경기 종료 시 점수가 같다면 먼저 들어온 순으로 등수를 매깁니다.
- 플레이어의 위치를 지도에 표시할 때 해당 칸의 정보는 표현하지 않습니다.

(3) 소스코드 설명

main.cpp

```
#include "game.h"
using namespace std;

int main()
{
    Game g;
    g.start();
    cout << endl;
    system("PAUSE");
    return 0;
}
```

Game object g 생성 후 게임 시작
게임이 한 판 끝나면 게임 종료

game.h

```
#pragma once
#ifndef __GAME_H__
#define __GAME_H__

#include <iostream>
#include <string>
#include "map.h"
#include "player.h"
using namespace std;

class Game {
private:
    int finish = 0; // 끝낸 사람의 수
    int n; // 플레이어 수
    int rank = 1; // 들어간 순서
    Player player[5]; // 1~4 배열만 사용 편의상 5까지 설정
    char opposite; // end에서 판단을 위한 방향, end 전칸에서 들어오는 방향과 반대방향이 아니면 끝
    Map m;
    string mapName;
public:
    Game() {}
    ~Game() {}
    void start();
    void play();
    void move(int);
};
#endif
```

finish : 끝낸 사람의 수
n : 입력 받은 플레이어의 수
rank : 골인 한 순서
player[5] : player객체, 5개를 먼저 생성한 뒤 1~4번만 사용
opposite : END로 들어가는 방향을 이용해 END에서 다시 지도로 들어가는 방향을 저장
만약 지도파일이
C U D
E
이렇게 끝난다면 D를 이용해 U를 저장한 뒤 END에 들어갔을 때 U가 아니면 골인으로 판단
m : Map 객체
mapName : 입력받을 지로 이름

game.cpp

start()

```
void Game::start() {
    while (1) {
        cout << "몇명이서 플레이 할건가요? (2 ~ 4)₩n₩n입력 : ";
        char a;
        cin >> a;
        n = int(a - 48);
        if (n >= 2 && n <= 4)
            break;
        else {
            cout << "₩n다시 입력하세요.₩n" << endl;
        }
    }
    char a;
    m.loadMap();
    cout << "₩n기본 지도를 불러왔습니다." << endl;
    while (1) {
        cout << "₩n1. 불러온 지도로 게임을 시작한다.₩n2. 다른 지도를 불러온다.₩n₩n입력 : ";
        cin >> a;
        int b = int(a - 48);
        if (b == 1) {
            cout << "₩n불러온 지도₩n" << endl;
            break;
        }
        else if (b == 2) {
            cout << "₩n불러올 지도 이름을 입력하세요. (ex. default.map.txt)₩n₩n입력 : ";
            cin >> mapName;
            m.loadMap(mapName);
        }
        else {
            cout << "₩n다시 입력하세요." << endl;
        }
    }
    m.showMap(-1); // 처음에는 -1
    this->opposite = m.getOpposite(m.getCnt() - 1);
    for (int i = 1; i < n + 1; i++)
    {
        player[i].setName(i);
    }
    this->play();
}
```

플레이 인원수를 입력받습니다. 만약 입력받은 수가 2~4가 아니라면 다시 입력받습니다. 그 후 m.loadMap()으로 default.map을 불러온 후 어떤 지도로 플레이 할 지 물어봅니다. 1을 입력받으면 그대로 게임을 시작하고 2를 입력받으면 지도 이름을 입력받아 불러옵니다. 둘 다 아니라면 다시 입력받습니다. 그 후 1을 선택하면 showMap()으로 불러온 지도를 출력하고 End로 들어가는 반대방향을 opposite에 저장합니다. 그 후 player의 이름을 저장한 후 게임을 진행합니다.

play()

```
void Game::play( ) {
    int i = 1;
    int turn = 1;
    while (1) {
        if (this->player[i].isEnd( ) == false) {
            bool d = this->player[i].rollDice( );
            if (d == true)
                this->move(i);
            if (this->n - 1 == this->finish) // 한명 남겨놓고 끝나면 끝
                break;
            cout << "₩n";
            this->player[i].showCard( );
            i += 1;
            if (i > this->n) {
                cout << "₩n₩n" << turn << "턴 종료" << endl;
                turn++;
                for (int i = 1; i < n + 1; i++)
                {
                    this->player[i].showCard( );
                }
                i = 1;
            }
        }
        else // 이미 끝냈으면 다음사람으로 넘어감
            i++;
    }
    // 점수계산
    int m = -1;//max score
    int index = 0;
    for (i = 1; i < this->n + 1; i++) {
        int n = this->player[i].getScore( );
        if (n > m) {
            m = n;
            index = i;
        }
        else if (n == m) { // 같은 점수면 먼저 들어온 사람이 승리
            if (this->player[i].getArrival( ) < this->player[index].getArrival( ))
                index = i;
        }
        this->player[i].showScore( );
    }
    cout << "₩n₩n승리자 : " << this->player[index].getName( ) << endl;
```

rollDice()를 이용해 주사위 결과를 얻습니다. 만약 주사위 결과가 0보다 크면 move()를 통해 움직입니다. 이동 후 게임을 끝낸 플레이어 수가 (플레이어 수 - 1) 이라면 게임을 종료합니다. 그렇지 않다면 이동 후 플레이어의 카드 현황을 보여줍니다. 게임이 끝났다면 getScore()를 통해 각 플레이어의 점수를 얻어 각자의 점수를 출력한 후 승리자의 이름을 보여줍니다.

move() - 1

```
void Game::move(int nPlayer) {
    string input;
    int n = 0;
    while (1)
    {
        int i = 0;
        if (this->player[nPlayer].getDice() == 0) {
            cout << "\n이동 할 수 없습니다." << endl;
            break;
        }
        cout << "\n현재 위치\n" << endl;
        m.showMap(this->player[nPlayer].getLocation());
        cout << "\n주사위의 결과 ( " << this->player[nPlayer].getDice() << " ) 만큼 이동 할 경로를 입력하세요(ex.UDRL or ududdu)\n\n입력 : ";
        cin >> input;
        int tmp = this->player[nPlayer].getLocation();
        n = input.length();
        if (n != this->player[nPlayer].getDice()) {
            cout << "\n다시 입력하세요.\n";
        }
        else {
            cout << "\n";
            const char* arr = input.c_str();
            int getb = 0;
            bool mflag = true; // 이동 완료 했는지 판단하는 moveflag
            for (i = 0; i < n; i++) {
                char c = toupper(arr[i]);
                if (m.getCnt() == tmp) { // END로 들어옴
                    if (c != opposite) { // END로 들어온 방향과 반대로 나가지 않으면 끝
                        this->player[nPlayer].setEnd();
                        this->player[nPlayer].setArrival(rank);
                        rank++;
                        this->finish++;
                        cout << this->player[nPlayer].getName() << " 골인\n" << endl;
                        break;
                    }
                    else
                        tmp--;
                }
```

주사위의 결과가 0이라면 이동을 할 수 없습니다. 그렇지 않다면 showMap()을 통해 지도에서의 자신의 현재 위치를 보여주고 어떻게 이동할지 입력받습니다. 만약 입력받은 길이가 주사위결과와 같지 않다면 다시 입력받습니다. 그 후 입력받은 값을 한 글자씩 대문자로 변환해 판단합니다. 만약 현재 위치가 End라면 들어왔던 방향(뒤)으로 나가는지 판단합니다. 만약 End에서 밖으로 나간다면 해당 플레이어가 끝났다는 것을 setEnd()를 통해 설정하고 등수를 매깁니다. 그 후 골인했다고 출력을 합니다. 만약 뒤로 갔다면 위치 값(몇번째 칸)을 한칸 뒤로 바꿉니다.

move() - 2

```
            else {
                int s = m.isAvail(c, tmp);
                if (s == 1) {
                    if (finish > 0) {
                        cout << "뒤로 갈 수 없습니다.₩n";
                        mflag = false;
                        break;
                    }
                    tmp--;
                }
                else if (s == 2)
                    tmp++;
                else if (s == 3) { // 다리 건너기
                    tmp = m.getbLocation(tmp);
                    getb++;
                }
                else {
                    mflag = false;
                    break;
                }
            }
        }
        if (mflag == true) { // 이동 완료
            for (i = 0; i < getb; i++) { // 다리 건넌 수 만큼 b카드를 얻는다
                this->player[nPlayer].getbCard();
            }
            this->player[nPlayer].setLocation(tmp);
            char card = m.getCard(this->player[nPlayer].getLocation()); // 카드 줍기
            if (card == 'P')
                this->player[nPlayer].getpCard();
            else if (card == 'S' && this->player[nPlayer].getLocation() != 0)
                this->player[nPlayer].getsCard();
            else if (card == 'H')
                this->player[nPlayer].gethCard();
            break;
        }
        else {
            cout << "다시 입력하세요.₩n";
        }
    }
}
if (this->player[nPlayer].isEnd() == false) { // 골인을 안했다면 이동 후 위치 출력
    cout << "₩n이동 후 위치₩n" << endl;
    m.showMap(this->player[nPlayer].getLocation());
}
```

만약 End가 아닌 다른 칸이면 isAvail()을 통해 해당 방향으로 움직일 수 있는지 판단 후 위치값을 변경합니다. 만약 한명이라도 골인 한 사람이 있다면 뒤로 갈 수 없습니다. 이동할 수 없는 위치라면 mflag(이동완료를 판단하는 move flag)를 false로 바꿔 다시 입력받습니다. 이동을 완료했다면 다리를 건넌 수만큼 getbCard()를 통해 다리 카드를 얻습니다. 도착한 위치에 도구카드가 있다면 도구 카드의 개수를 증가시킵니다. 이동 완료 후 위치를 지도에 표시합니다.

map.h

```
#pragma once
#ifndef __MAP_H__
#define __MAP_H__

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
using namespace std;
struct Coord { // 지도 좌표
    int x = 0;
    int y = 0;
    int n = 0; // 몇번째 칸인지 저장
};
class Map {
private:
    char map[101][4]; // 지도 최대 칸은 100칸으로 제한
    char board[50][21]; // 지도의 최대 크기는 20X20으로 제한, start와 end는 좌우 양 끝이라고 가정
    Coord coord = { 25,0,0 }; // 시작지점부터 위 아래로 20칸 있을 수 있기 때문에 지도 좌표 [25][0]에서 시작
    int u = 25;//지도의 위쪽 끝 초기값
    int d = 25;//지도의 아래쪽 끝 초기값
    int r = 0;//지도의 오른쪽 끝 초기값
    vector<Coord> mapCoord; // 각 칸의 좌표 저장
    vector<int> bridgeB; // B좌표
    vector<int> bridgeb; // b좌표
    int cnt = 0;
    int bi = 0; // bridge index 몇번째 bridge인지
public:
    Map() {}
    ~Map() {}
    void loadMap(string);
    void loadMap();

    void setBridge();
    void showMap(int);
    Coord direction(Coord, char);
    int isAvail(char, int);
    int getbLocation(int);
    char getCard(int l) { return this->map[l][0]; }
    char getOpposite(int);
    int getCnt() { return this->cnt; }
    void initialize();
};
#endif
```

map[][] : 지도의 칸수를 100칸으로 제한해 [101][4]로 생성. 지도 데이터가 C U D 가 있다면 [][0]=C, [][1]=U, [][2]=D 저장, [][3]에는 B가 나올 때 마다 몇 번째로 나온 다리인지 저장
board[][] : 지도의 좌표를 이용해 지도를 그리는 배열. 지도의 최대 크기를 20X20으로 제한했고, 시작지점은 [25][0], 지도의 양끝을 start와 end로 가정했기 때문에 [50][21]로 생성. 각 칸의 지도 데이터에서 첫 번째로 나온 지도의 정보를 저장
coord : 각 칸의 지도 좌표와 몇 번째 칸인지 저장하는 구조체
u : 지도의 위쪽 끝, d : 아래쪽 끝, r : 오른쪽 끝
mapCoord : coord 구조체를 저장하는 벡터
bridgeB : 다리의 순서에 따라 B의 좌표를 저장하는 벡터
bridgeb : 다리의 순서에 따라 b의 좌표를 저장하는 벡터
cnt : 지도가 몇 번째 칸인지 저장
bi : 다리의 index

map.cpp

loadMap() - 1

```
void Map::loadMap() {
    ifstream mapFile;
    mapFile.open("default.map.txt");
    if (!mapFile.is_open()) {
        cout << "\n" << "기본 지도 파일이 없습니다.\n\n게임을 종료합니다.\n";
        exit(1);
    }
    char line[10];

    while (mapFile.getline(line, sizeof(line)))
    {
        map[cnt][0] = line[0];
        if (cnt == 0) {
            map[cnt][1] = line[2];
            board[coord.x][coord.y] = map[cnt][0];
            coord.n = cnt;
            mapCoord.push_back(coord);
            coord = direction(coord, map[cnt][1]);
        }
        if (map[cnt][0] == 'E') {
            board[coord.x][coord.y] = map[cnt][0];
            mapCoord.push_back(coord);
            break;
        }
        else if (map[cnt][0] == 'B')
        {
            map[cnt][1] = line[2];
            map[cnt][2] = line[4];
            map[cnt][3] = (char)(bi + 48); // 몇번째 bridge인지 저장
            bi++;
            bridgeB.push_back(cnt); // B 좌표 저장
            board[coord.x][coord.y] = map[cnt][0];
            coord.n = cnt;
            mapCoord.push_back(coord);
            coord = direction(coord, map[cnt][2]);
        }
```

기본 default.map을 불러옵니다. 만약 파일이 없다면 종료합니다. 지도 데이터를 한줄 씩 읽어서 첫번째 칸인지, 혹은 각 칸의 종류에 따라 map과 board에 저장합니다. 현재 지도에서의 좌표를 이용해 board에 셀 종류를 저장합니다. 그 후 mapCoord에 해당 칸의 좌표정보를 저장한 후 direction()을 통해 좌표를 다음칸의 좌표로 변경합니다. 만약 B 칸이라면 map에 몇번째 B 칸인지 저장합니다. 그 후 bridgeB에 B 칸의 좌표를 저장합니다.

loadMap() - 2

```
        else if (map[cnt][0] == 'b')
        {
            map[cnt][1] = line[2];
            map[cnt][2] = line[4];
            board[coord.x][coord.y] = map[cnt][0];
            coord.n = cnt;
            mapCoord.push_back(coord);
            coord = direction(coord, map[cnt][2]);
        }
        else
        {
            if (cnt != 0) {
                map[cnt][1] = line[2];
                map[cnt][2] = line[4];
                board[coord.x][coord.y] = map[cnt][0];
                coord.n = cnt;
                mapCoord.push_back(coord);
                coord = direction(coord, map[cnt][2]);
            }
        }
        cnt++;
    }
    mapFile.close();
    this->setBridge();
}
```

모든 칸을 다 저장한 후 setBridge()를 통해 다리를 만들어 줍니다.

loadMap(string)

```
void Map::loadMap(string mapName) {
    ifstream mapFile;
    mapFile.open(mapName);
    if (mapFile.is_open()) {
        cout << "₩n" << mapName << " 지도 파일을 불러옵니다.₩n₩n";
        this->initialize(); // 변수 초기화
```

입력받은 지도 파일 이름을 이용해 파일을 불러옵니다. 파일을 저장하기 전 initialize를 통해 이전에 저장했던 변수들을 초기화 해줍니다. 이후는 loadMap()과 동일하게 진행합니다.

```
else {
    cout << "₩n" << mapName << " 파일이 없습니다." << endl;
}
```

만약 입력한 지도 파일이 없다면 오류메시지를 출력합니다.

setBridge()

```
void Map::setBridge() {
    for (int i = 0; i < bi; i++) {
        int a = mapCoord[bridgeB.at(i)].x;
        int b = mapCoord[bridgeB.at(i)].y;
        int n = mapCoord.size();
        if (board[a][b + 2] == 'b') { // B -> b
            for (int j = 0; j < n; j++) {
                if (mapCoord[j].x == a && mapCoord[j].y == b + 2) {
                    bridgeb.push_back(mapCoord[j].n); // b의 좌표 bridgeb에 순서대로 저장
                    break;
                }
            }
            board[a][b + 1] = '-';
        }
        else if (board[a][b - 2] == 'b') {
            for (int j = 0; j < n; j++) {
                if (mapCoord[j].x == a && mapCoord[j].y == b - 2) {
                    bridgeb.push_back(mapCoord[j].n); // b의 좌표 bridgeb에 순서대로 저장
                    break;
                }
            }
            board[a][b - 1] = '-';
        }
    }
}
```

B 칸 좌표를 받아서 B칸의 오른쪽 두 칸 옆에b가 있는지 왼쪽 두 칸 옆에 b가 있는지 판단해 bridgeb에 b 칸 좌표를 저장합니다. 그 후 지도에 다리를 놓습니다.

showMap(int)

```
void Map::showMap(int location) {
    int a = 0, b = 0;
    if (location != -1) {
        a = mapCoord.at(location).x;
        b = mapCoord.at(location).y;
    }
    for (int i = this->u; i < this->d + 1; i++) {
        cout << " ";
        for (int j = 0; j < this->r + 1; j++) {
            if (i == a && j == b) {
                cout << "* ";
            }
            else if (board[i][j] == '₩0')
                cout << "  ";
            else
                cout << board[i][j] << " ";
        }
        cout << "₩n";
    }
}
```

지도의 UI 부분 입니다. 플레이어의 위치 값(몇번째 칸)을 이용해 지도와 플레이어의 위치를 출력합니다. 만약 위치 값이 -1이면 그냥 지도만 출력합니다. 그렇지 않다면 board에 저장된 지도 칸의 종류와 플레이어의 위치(*)를 순서대로 출력합니다.

direction(Coord, char)

```
Coord Map::direction(Coord c, char d) {
    if (d == 'U') {
        c.x--;
        if (this->u > c.x)
            this->u = c.x;
    }
    else if (d == 'D') {
        c.x++;
        if (this->d < c.x)
            this->d = c.x;
    }
    else if (d == 'R') {
        c.y++;
        if (this->r < c.y)
            this->r = c.y;
    }
    else
        c.y--;
    return c;
}
```

지도를 불러들일 때 한줄의 데이터에서 세번째 값이 다음 칸으로 가는 방향임을 이용해 다음 지도의 좌표를 계산합니다. 만약 데이터가 C U D 이렇게 있고, 지도에서의 좌표가 (27,2)라면 다음 칸은 그 아래 칸인 (28,3)입니다.

isAvail(char, int)

```
int Map::isAvail(char c, int location) { // 이동 가능한지 계산
    if (location == 0 && c == this->map[location][1])
        return 2;
    else if (c == this->map[location][1])
        return 1;
    else if (c == this->map[location][2])
        return 2;
    else if (this->map[location][0] == 'B') {
        int a = int(this->map[location][3] - 48);
        if ((this->bridgeB.at(a) < this->bridgeb.at(a) && c == 'R') || (this->bridgeB.at(a) > this->bridgeb.at(a) && c == 'L')) // B->b 이고 R 이거나 b->B 이고 L이면 이동 가능
            return 3;
        else
            return 4;
    }
    else
        return 4;
}
```

입력받은 이동경로의 한 글자와 플레이어의 현재 위치 값을 이용해 이동가능한지 판별합니다. 현재의 위치 값을 이용해 map에 저장해뒀던 데이터를 이용해 각 상황에 따라 1,2,3,4를 반환합니다.

getbLocation(int)

```
int Map::getbLocation(int i) { // 다리 건넌 후 위치 반환
    int a = int(this->map[i][3] - 48);
    return this->bridgeb.at(a);
}
```

map에 저장해 뒀던 몇번째 다리인지를 이용해 다리를 건넌 후의 위치를 반환합니다.

getOpposite(int)

```
char Map::getOpposite(int a) { // end로 들어가는 방향의 반대방향 계산
    if (map[a][2] == 'U')
        return 'D';
    else if (map[a][2] == 'D')
        return 'U';
    else if (map[a][2] == 'R')
        return 'L';
    else
        return 'R';
}
```

end 바로 앞 칸의 위치 값을 이용해 end로 들어가는 방향의 반대방향을 반환합니다.

initialize()

```
void Map::initialize() { // map을 새로 입력받을 때 초기화
    for (int i = 0; i < 101; i++) {
        this->map[i][0] = '#0';
        this->map[i][1] = '#0';
        this->map[i][2] = '#0';
        this->map[i][3] = '#0';
    }
    for (int i = 0; i < 50; i++) {
        for (int j = 0; j < 21; j++) {
            this->board[i][j] = '#0';
        }
    }
    this->coord = { 25,0,0 };
    this->u = 25;
    this->d = 25;
    this->r = 0;
    this->mapCoord.clear();
    this->bridgeB.clear();
    this->bridgeb.clear();
    this->cnt = 0;
    this->bi = 0;
}
```

원하는 지도를 새로 불러들일 때 기존 데이터들을 초기화합니다.

player.h

```
#pragma once
#ifndef __PLAYER_H__
#define __PLAYER_H__

#include <iostream>
#include <ctime>
#include <string>
#include "map.h"
using namespace std;

class Player {
private:
    int location = 0; // player가 몇번째 칸인지
    string name;
    bool end = false;  //if end, F->T
    int dice = 0; // 주사위 결과, 움직일 수 있는 칸 수
    int arrival = 0; // 도착순서
    int score = 0;
    int pCard = 0; // Philips Driver
    int hCard = 0; // Hammer
    int sCard = 0; // Saw
    int bCard = 0; // Bridge
public:
    Player() {}
    ~Player() {}
    bool rollDice();
    void setEnd() { this->end = true; }
    bool isEnd();
    void setName(int);
    string getName() { return this->name; }
    void setArrival(int i) { this->arrival = i; }
    void showCard();
    void getpCard() { this->pCard += 1; cout << "P 카드를 얻습니다.₩n"; }
    void gethCard() { this->hCard += 1; cout << "H 카드를 얻습니다.₩n"; }
    void getsCard() { this->sCard += 1; cout << "S 카드를 얻습니다.₩n"; }
    void getbCard() { this->bCard += 1; cout << "B 카드를 얻습니다.₩n"; }
    int getArrival() { return this->arrival; }
    void setDice(int d) { this->dice = d; }
    int getDice() { return this->dice; }
    void setLocation(int i) { this->location = i; }
    int getLocation() { return this->location; }
    int getScore();
    void showScore();
};
#endif
```

location : 플레이어가 현재 몇 번째 칸에 있는지 저장
name : 플레이어의 이름
dice : 주사위 결과이자 플레이어가 움직일 수 있는 칸의 수
arrival : 도착 순서
score : 최종 점수
pCard, hCard, sCard, bCard : 각각 얻은 카드의 수

player.cpp

rollDice()

```
bool Player::rollDice() {
    int a;
    cout << "₩n=================================================================" << endl;
    while (1) {
        cout << "₩n" << this->name << " 차례입니다.₩n₩n";
        cout << "주사위를 굴릴지 쉴지 선택하세요.₩n₩n1. 굴린다₩n2. 쉰다₩n₩n입력 : ";
        cin >> a;
        if (a == 1)
        {
            srand((unsigned int)time(NULL));
            int d = (int)(1 + rand() % 6);
            cout << "₩n" << "주사위에서 나온 숫자 : " << d << endl;
            cout << "가지고 있는 다리카드 개수 : " << this->bCard << endl;
            if (d - this->bCard < 0)
                this->dice = 0;
            else
                this->dice = d - this->bCard;
            cout << "주사위 결과 : " << dice << endl;
            return true;
        }
        else if (a == 2) {
            cout << "₩n" << "이번 차례는 쉽니다." << endl;
            if (this->bCard > 0) {
                this->bCard--;
                cout << "다리카드 한개를 제거합니다." << endl;
            }
            return false;
        }
        else {
            cout << "₩n다시 입력하세요." << endl;
        }
    }
}
```

1을 입력하면 srand()를 이용해 주사위를 굴립니다. 주사위의 최종 결과 값은 주사위에서 나온 숫자에서 다리카드 개수를 뺀 값입니다. 2를 입력하면 쉬고 다리카드 한 개를 제거합니다. 이외의 값을 입력하면 다시 입력 받습니다.

isEnd()

```
bool Player::isEnd() { // 끝났는지 판단
    if (this->end == true)
        return true;
    else
        return false;
}
```

플레이어가 골인을 했는지 판단합니다. 골인을 했다면 true를 반환합니다.

setName(int)

```
void Player::setName(int n) {
    string temp1 = "Player ";
    string temp2 = to_string(n);
    this->name = temp1 + temp2;
}
```

몇번째 플레이어인지에 따라 이름을 Player n 으로 저장합니다.

showCard()

```
void Player::showCard() {
    cout << this->name << " " << " P : " << this->pCard << "  H : " << this->hCard << "  S : " << this->sCard << "  B : " << this->bCard << endl;
}
```

플레이어가 가지고 있는 카드들을 보여줍니다.

getScore()

```
int Player::getScore() {
    int bonus[4] = { 0,7,3,1 }; // 도착 순서에 따른 점수 도착 못하면 0점
    this->score = this->pCard + this->hCard * 2 + this->sCard * 3 + bonus[this->arrival];
    return this->score;
}
```

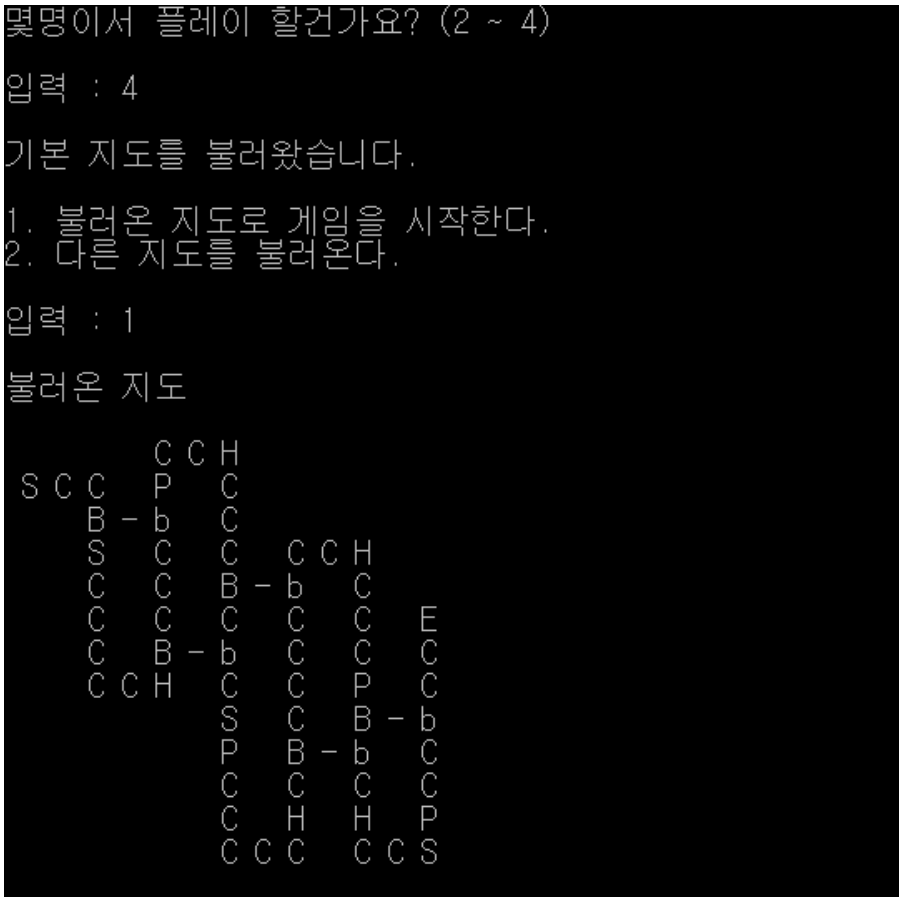
플레이어의 도착순서와 가지고 있는 카드에 따라 점수를 계산해서 반환합니다.

showScore()

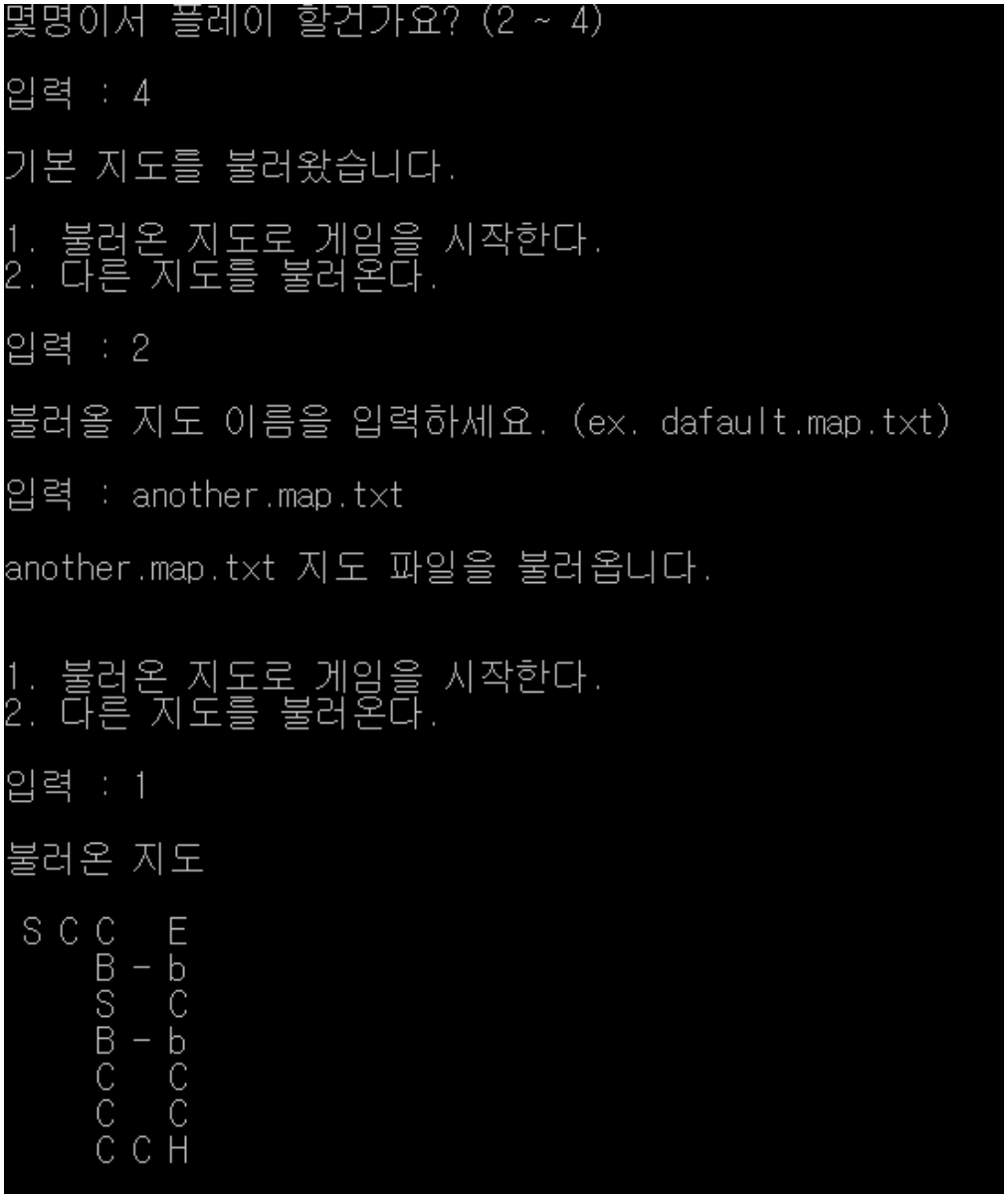
```
void Player::showScore() {
    cout << this->name << " 의 점수 : " << this->score << endl;
}
```

플레이어의 점수를 출력합니다.

(4) 실행 결과



플레이어 수를 입력하고 기본지도로 게임을 시작했을 때의 화면



플레이어 수를 입력 후 another.map을 불러와서 게임을 시작했을 때의 화면

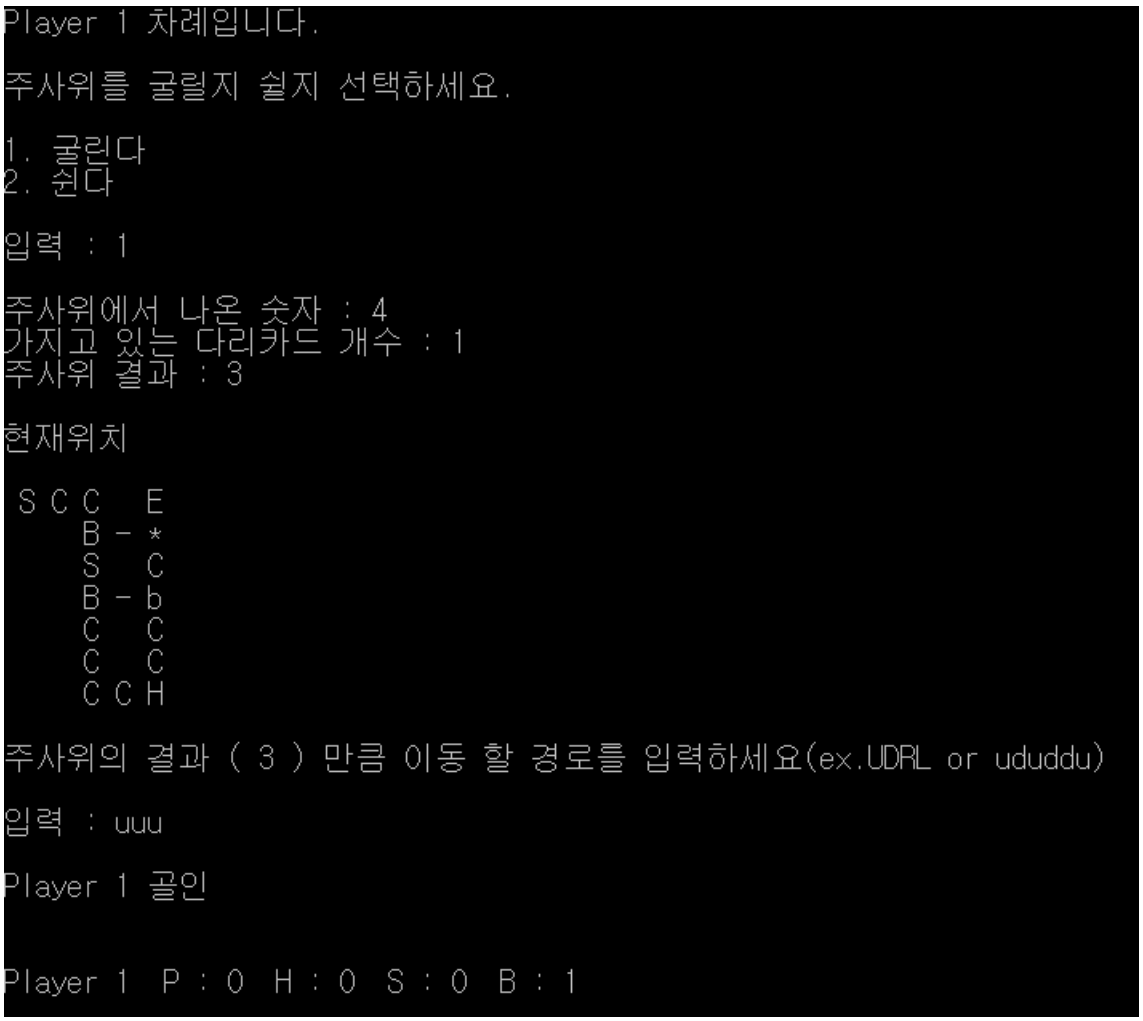
```
Player 1 차례입니다.

주사위를 굴릴지 쉴지 선택하세요.

1. 굴린다
2. 쉰다

입력 : 1

주사위에서 나온 숫자 : 4
가지고 있는 다리카드 개수 : 0
주사위 결과 : 4

현재위치

 * C C   E
     B - b
     S   C
     B - b
     C   C
     C   C
     C C H

주사위의 결과 ( 4 ) 만큼 이동 할 경로를 입력하세요(ex.UDRL or ududdu)

입력 : rrrr

다시 입력하세요.
```

주사위를 굴린 후 이상한 경로를 입력했을 때 다시 입력을 받는 화면

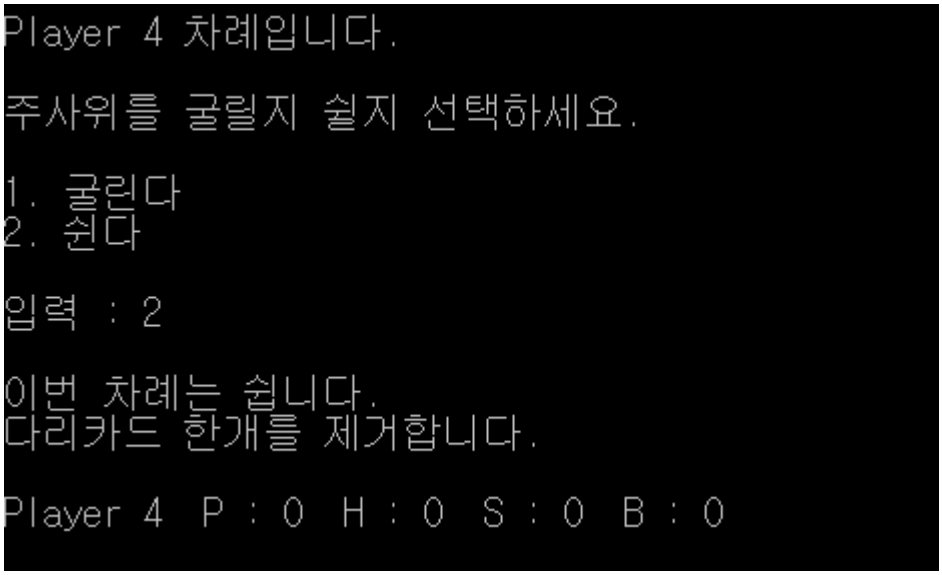
```
현재위치

 * C C   E
     B - b
     S   C
     B - b
     C   C
     C   C
     C C H

주사위의 결과 ( 4 ) 만큼 이동 할 경로를 입력하세요(ex.UDRL or ududdu)

입력 : rrdr

B 카드를 얻습니다.

이동 후 위치

 S C C   E
     B - *
     S   C
     B - b
     C   C
     C   C
     C C H

Player 1  P : 0  H : 0  S : 0  B : 1
```

이동 가능한 경로를 입력 한 후 이동한 뒤 지도에 위치를 표현하고 카드 현황을 출력한 화면

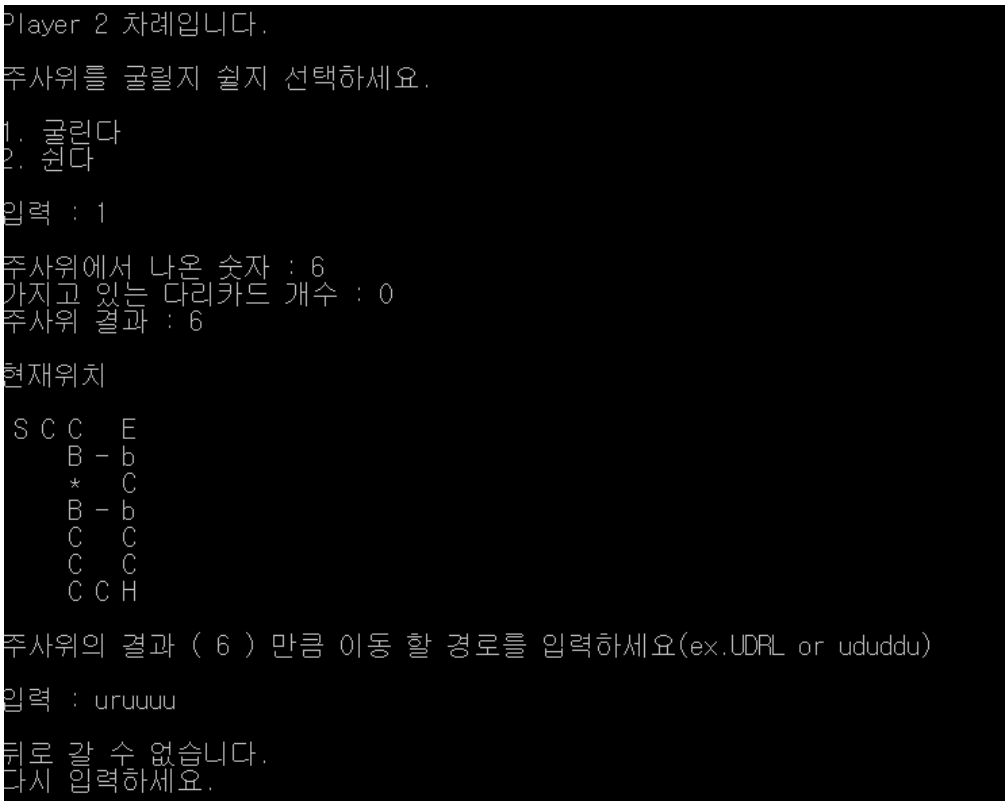
한턴 진행 후 플레이어의 카드 현황을 보여준 화면



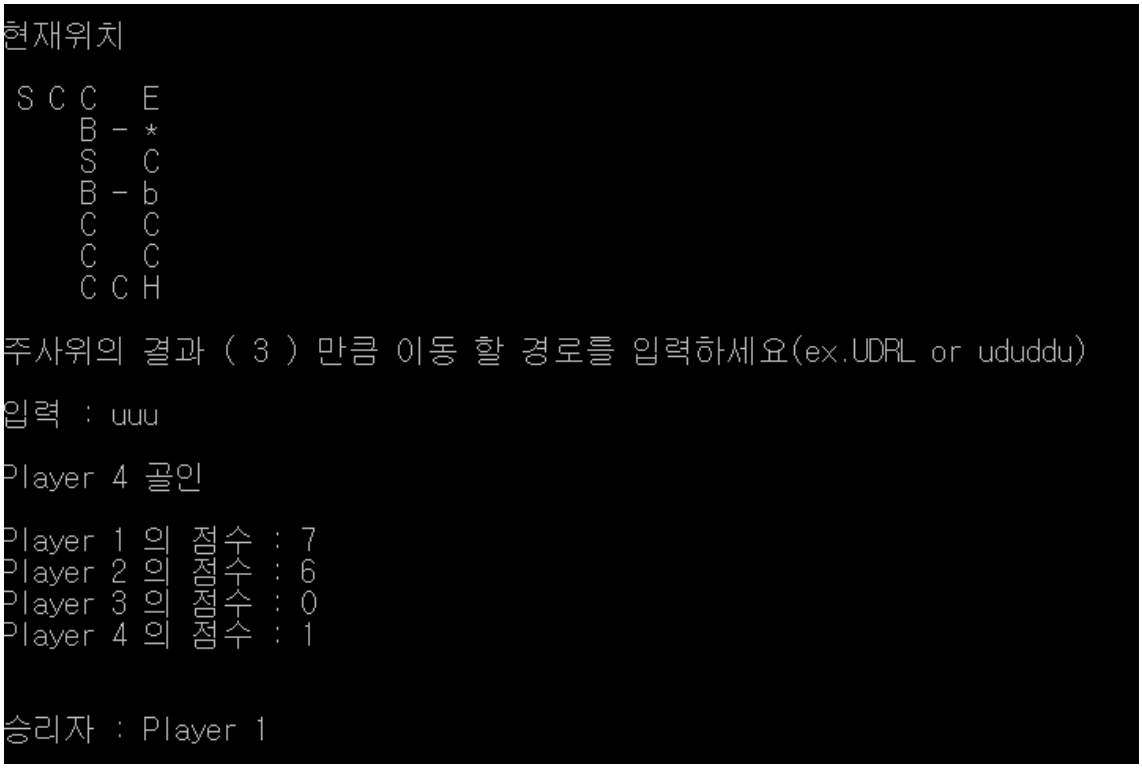
플레이어가 골인을 했을때의 화면



주사위를 굴리지 않고 쉬었을 때 다리카드 한개를 제거한 화면



플레이어 한명이 골인 한 후 뒤로 가려 했을 때 갈 수 없는 화면



4명의 플레이어 중 3명이 골인했을 때 게임을 종료하고 플레이어들의 점수를 출력하고 승리자를 출력한 화면