사보타지

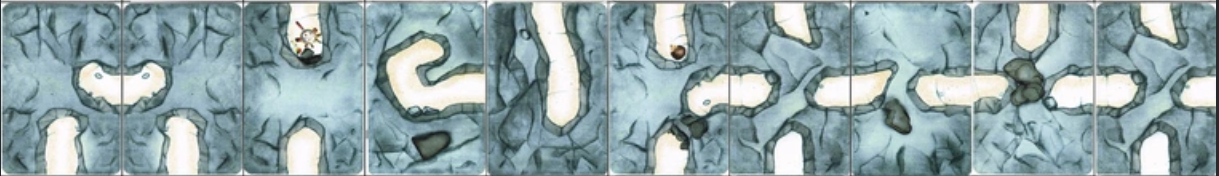
**카드 플레이어 카드** [**사보타지**x4, **광부**x7],

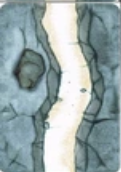
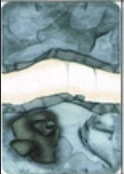
**금덩이 카드** **3개**x4, **2개**x8, **1개**x16,

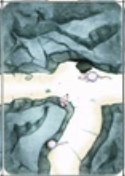
**행동 카드**x27

[**지도**x6, **디버프 3종류**x3 x3, **해제 3종류 단일**x2 x3, **복수** x3, **낙석**x3],

**굴 카드**x40

****

****

****

**x4 x5 x5 x5 x4 x5 x3**

class Card {

static int restCardNum;

private void useCard() { restCardNum--; }

void discard(); // 카드 버리기

};

class ActionCard : Card {};

class MapCard : ActionCard {

// static int restMapCard = 5;

};

class RockDown : ActionCard {

// static int restRockDownCard = 3;

};

class 장비Card : ActionCard {};

class 장비수리Card : 장비Card {

static int lanternRepair;

static int cartRepair;

static int pickaxRepair;

// 2개 합쳐진 카드도 각각 따로 count, 사용하면 minus도 둘 다 적용

};

class 장비파괴Card : 장비Card {};

[Flags] enum Dir {

RIGHT = 1,

LEFT = 2,

DOWN = 4,

UP = 8

}

class CaveCard : Card {

bool isConnected // 중간에 막혀있는 지 검사

Dir dir // 카드의 방향

void rotate() // 카드 180도 돌리기 => 카드 dir 뒤집어서 저장

}

class DestCard : CaveCard {

bool isOpen;

bool nearByCardExist;

}

**Case1 – 선택한 위치에 이미 카드가 존재하는지 확인**

**Case2 -**

1. **플레이어가 굴 카드를 놓을 때 마다**

* **놓으려는 지점 주변에 카드가 있는지 확인.**

**놓으려는 굴 카드의 비트 확인, 주변 굴 카드와 비트 연산하여**

**주변 카드와 이어지는지 조건 확인.**

* **BFS로 놓으려는 지점 ~ 시작 지점까지 굴 카드가 놓여져있는지 확인**

1. **목적지 카드 주변에 카드가 놓여지면 DestCard 객체의 nearByCardExist를 true로 바꿈. nearByCardExist가 true이면, 플레이어가 카드를 놓을 때마다 BFS로 목적지 ~ 시작 지점까지 길이 모두 유효하게 이어져있는지 확인**

**Class Map {**

private CaveCard[][];

isEmpty();

getter; // Dest 금 위치 반환

**}**

1. **게임 시작 시 출발점, 목적지 3군데 Map 클래스의 배열 초기화**

* **목적지: 금 한 군데를 랜덤으로 정하고 나머지 돌 카드**

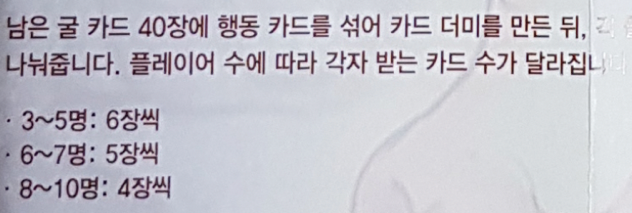
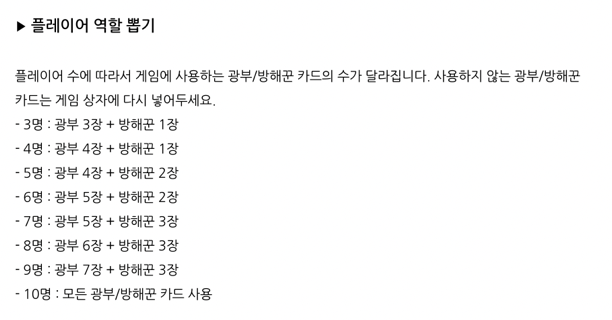
**규칙**

광부는 금을 찾고, 사보타지는 광부가 금을 찾지 못하게 방해하는 것

**광부 승리조건** 길을 이어서 금을 찾게 되면 승리

**사보타지 승리조건** 덱에 있는 카드와, 광부에게 있는 카드를 모두 소진 시 승리

**총 3라운드**

****

* 랜덤으로 시작 플레이어 정하기
* 방에 들어오는 순서대로 차례 정하기
* **본인 차례에는 카드 1장 반드시 사용**

1. 굴 카드 1장을 동굴 미로에 배치
2. 행동 카드 1장을 플레이어 1명에게 사용
3. 아무 카드 1장을 뒷면이 보이게 엎어서 버림
4. 본인에게 카드가 없다면 차례 건너뛰고 계속 진행

카드 사용 후 카드 더미에서 카드 1장을 가져 감

카드 더미가 바닥났다면, 카드를 가져오지 않음

카드 사용 후에는 ‘버린 카드 더미’로 앞면이 보이게 카드를 버림

‘버린 카드 더미’에는 앞면과 뒷면 두 개가 공존

* **행동 카드**
  + 장비 파괴 카드

1. 등불x3, 곡괭이x3, 수레x3
2. 장비가 파괴된 플레이어는, 본인 턴에 굴 카드를 배치하지 못함.
   * 장비 수리 카드
3. 등불x2, 곡괭이x2, 수레x2, [등불, 곡괭이]x1, [등불, 수레]x1, [곡괭이, 수레]x1
4. 장비가 파괴된 플레이어의 장비를 수리해 줄 수 있음. 단, 2종류가 한 카드에 들어있는 경우, 둘 중 하나만 수리해줄 수 있음.
5. 장비가 파괴되지 않은 플레이어에겐 사용 불가능
   * 낙석
6. 출발지 / 목적지를 제외한, 미로에 있는 굴 카드 중 아무거나 1장을 골라서 파괴. 파괴한 굴 카드와 사용한 낙석 카드는 모두 ‘버린 카드 더미’로 버림
7. 파괴되어 빈 공간은 나중에 다시 채울 수 있음.
   * 지도
8. 목적지 카드 3장 중 하나를 선택해서, 사용한 플레이어에게만 선택한 카드를 보여줌.

* **라운드 종료 조건 (3라운드 모두 끝나면 게임 종료)**
  + **사보타지 승리 조건**

1. 플레이어가 금을 찾기 전에 모든 광부들이 카드를 소진할 때 승리
2. 광부가 항복 선언하면 승리
   1. **광부 항복**
      1. 광부 중 한 명이 항복 선언을 하고, 서버에서 광부들에게만 항복 투표를 시킴 🡪 광부의 2/3 이상 항복 찬성 시 사보타지 승리
   * **광부 승리 조건**
3. 금을 찾으면 승리

* **금덩이 분배**
  + 이긴 팀에서, 마지막으로 길을 놓은 사람에게 금덩이 우선 선택권 주어짐
  + 차례대로 금덩이 선택
  + 금덩이 모음을 랜덤으로 서버에서 골라 줌

**User**

**플레이어 방장 관리 => Queue 자료구조**

**플레이어 입장 시 순서대로 Queue에 추가,**

**Queue Head에 있는 플레이어가 방장**

**class Room {**

**private int[] roomCode;**

**int roomCodeGenerator(); // room code 생성**

**};**

**// Player 개인의 data**

**class Player {**

**private name;**

**private role; // 광부 or 사보타지**

**private int score; // 현재 점수(금 소지 개수)**

**private Card[]; // Player가 가지고 있는 카드**

**};**

**class Server {**

**Player queue;**

**Player currentTurnPlayer, nextTurnPlayer;**

**private int 사보티지 인원 수;**

**private int 광부 인원 수;**

**private int currentRound = 1; // 1 ~ 3 라운드**

**};**

**네트워크 통신**

1. **Turn 관리**
2. **Card 관리**
3. **기권**
4. **채팅**

**라운드 종료하면**

* **모든 Player의 name, role 정보 공개**
* **Card 초기 값으로 다시 설정**
* **Player 직업(role), Card 재분배**
* **Map 초기화**

**3라운드 모두 종료하면, 모든 Player의 score(소유한 금) 공개**

**다시 room으로 복귀**

**Or room 폭파**