프로그래밍 역량 강화 전문기관, 민코딩

Trivia KATA

Trivia KATA 소개

Trivia Game 이란?

✔ Trivia Game : 미국과 영국 등 영어권 나라의 Pub 에서 즐겨하는 “상식퀴즈“ 게임

✔ (모두의 마블 같은) 보드판이 있고,
각 칸에는 퀴즈의 주제가 써 있음.

✔ 퀴즈의 주제 : 팝, 락, 과학, 스포츠

✔ 주사위를 굴려 말을 이동시키고,
말이 도착한 지점에 대한 주제의
퀴즈를 맞추어야 한다.

모두의 마블 이미지

이미지출처 : 나무위키, KAKAO 모두의 마블 게임 소개 페이지
(https://namu.wiki/w/%EB%AA%A8%EB%91%90%EC%9D%98%EB%A7%88%EB%B8%94%20for%20kakao/%EB%A7%B5)

Legacy 코드 다운로드

사외링크

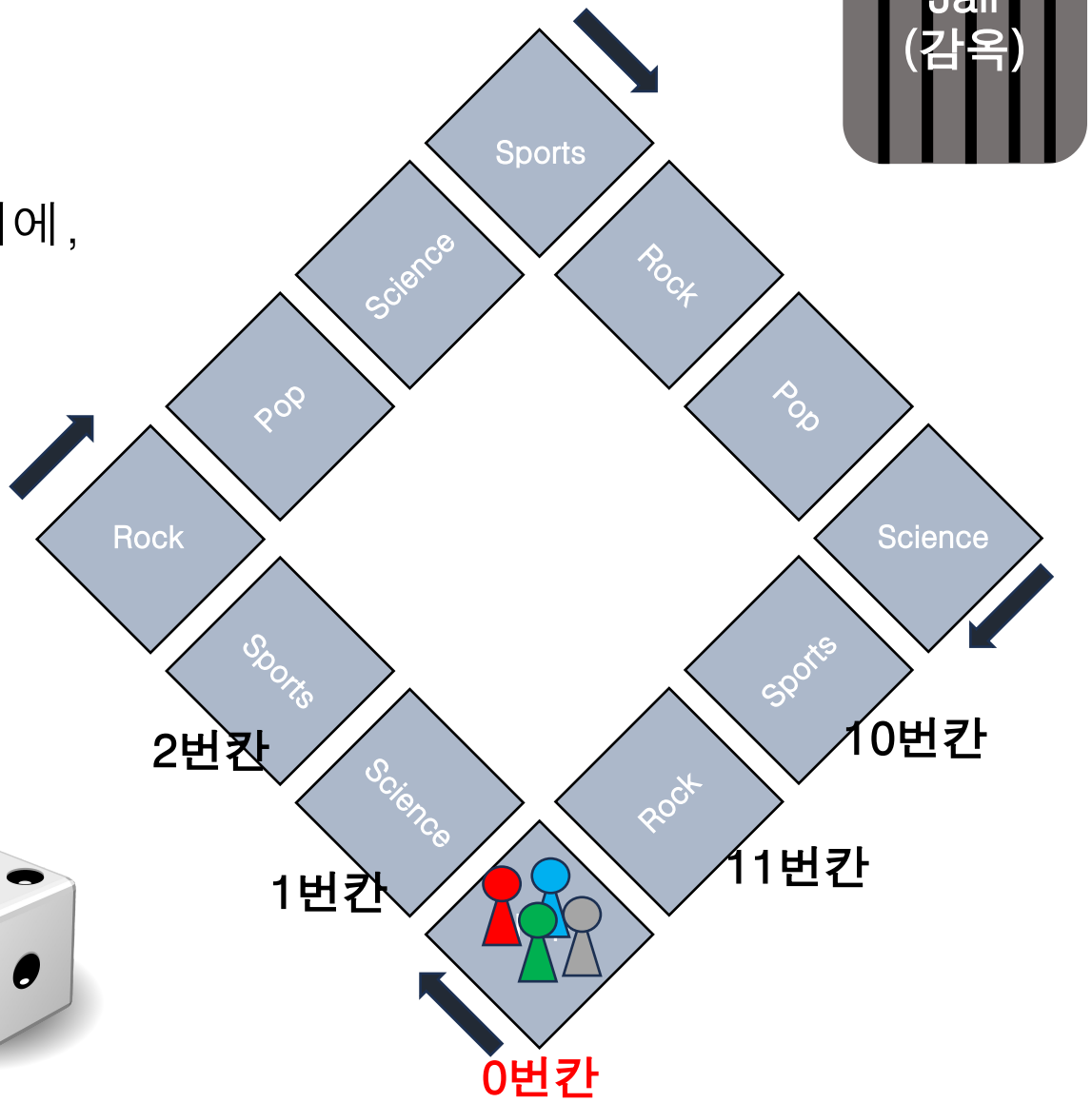
- https://github.com/mincoding1/Trivia

Trivia Game 예시 1

0 ~ 11번 칸 까지, 총 12개 칸으로 이뤄진 맵 위에,
총 4명의 플레이어가 게임을 한다고
가정한다.

모두 0번에서 출발한다.

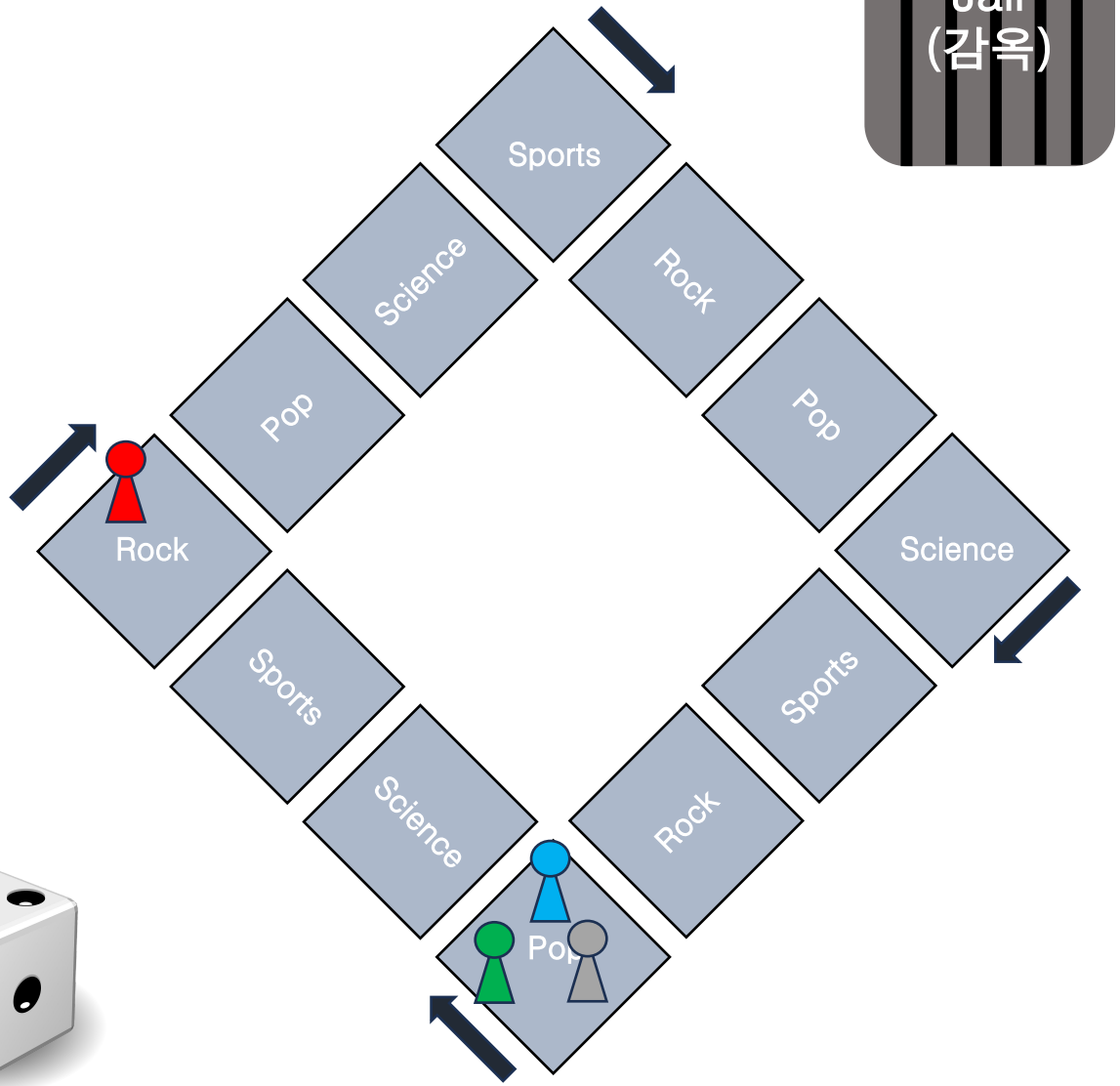
플레이어는 한 명씩 주사위를 굴려,
주사위 눈금만큼 말을 이동시킨다.



Trivia Game 예시 2

첫 번째 Player이
눈금 3이 나왔다면 세 칸 이동한다.

3번 칸의 주제는 Rock이다.
Rock 음악에 대한 상식 퀴즈를 맞춰야 한다.



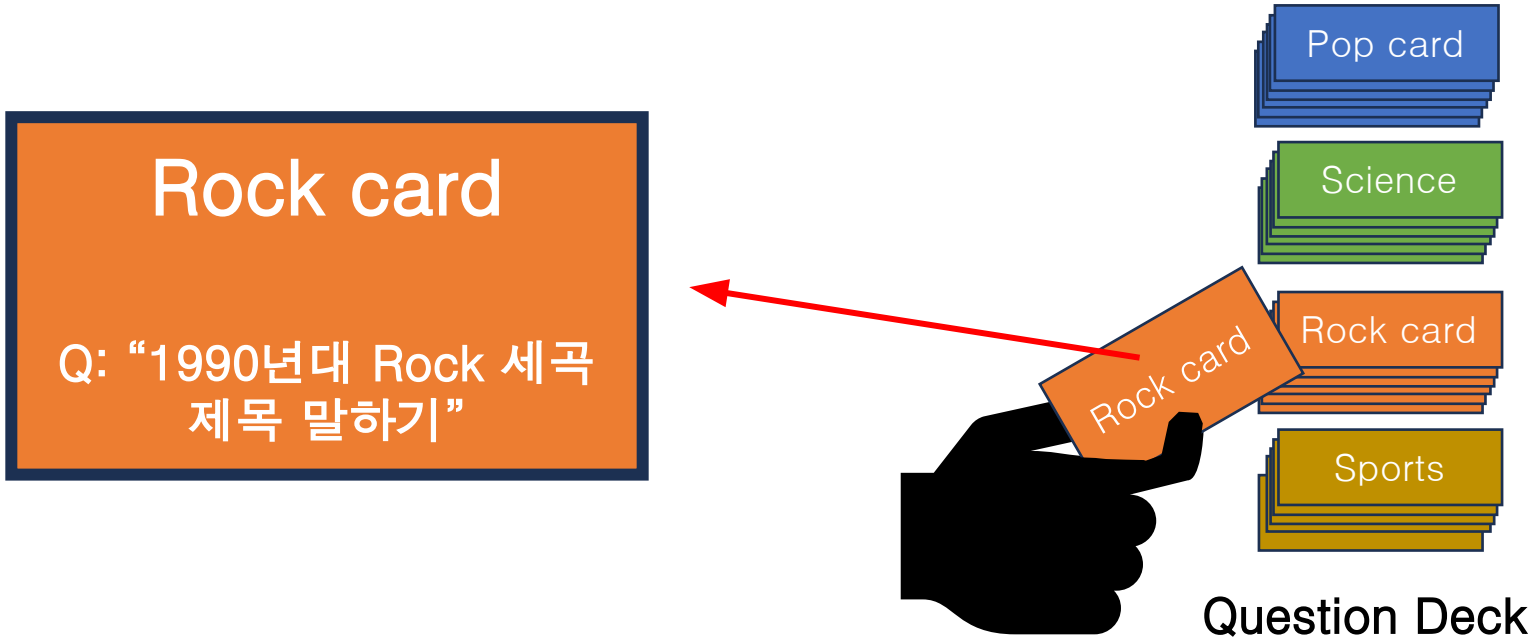
Trivia Game 예시 3

Rock 에 도착한 플레이어는

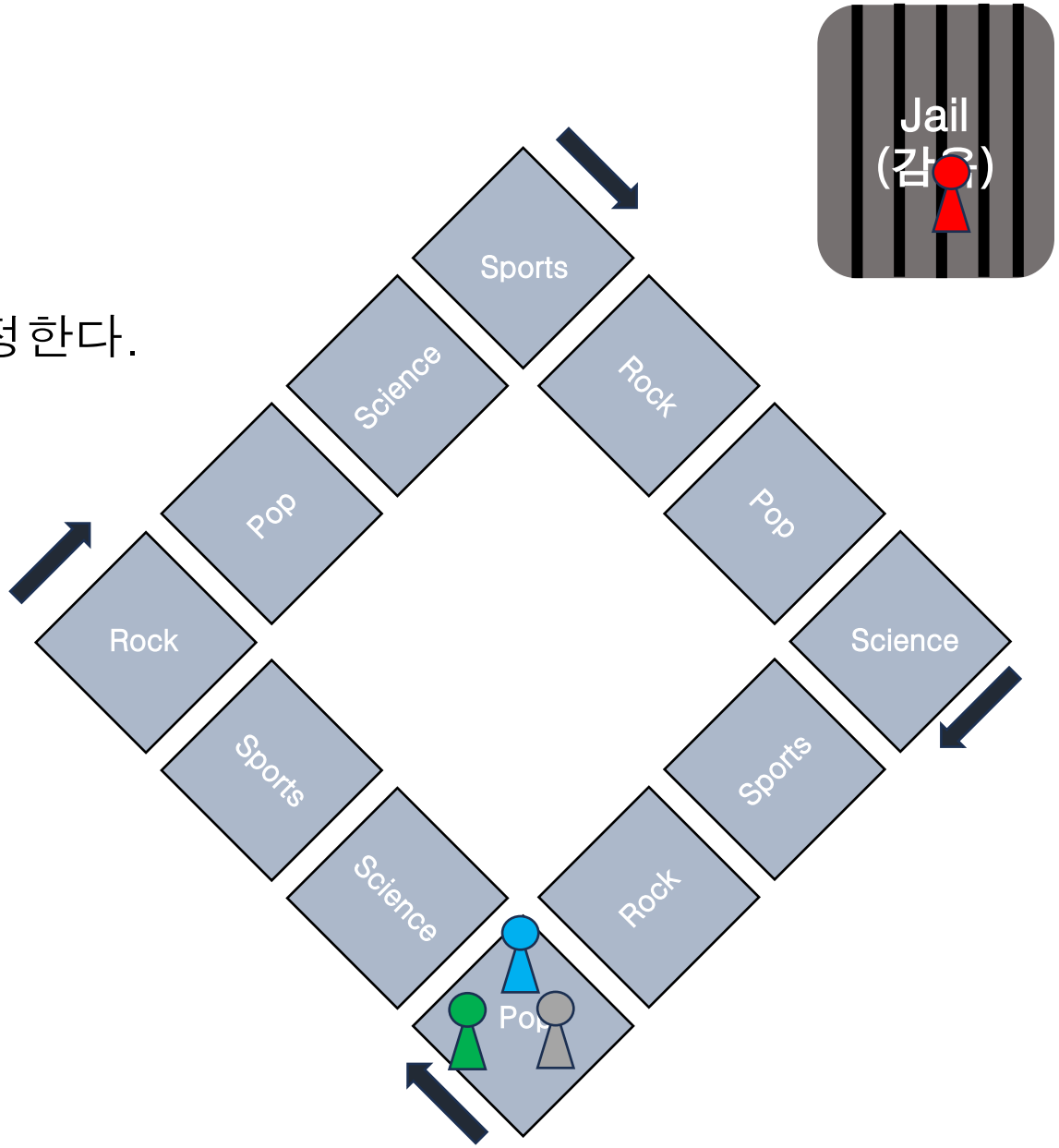
Question Deck 에 쌓여 있는 Rock 카드를 한 장 뽑는다.

이 카드에는 퀴즈 내용이 적혀 있다.

퀴즈를 맞추면 Coin 1개를 얻고, 틀리면 감옥으로 가야한다.



Trivia Game 예시 4



플레이어가 퀴즈에 정답을 맞추지 못했다고 가정한다.
따라서 감옥으로 말을 옮긴다.

돌아오는 턴에서
주사위의 눈금이 홀수일때만
감옥 탈출과 동시에 말을 이동시킬 수 있다.

위와 같은 방법으로 게임이 반복된다.
총 6개의 코인을 먼저 얻는 사람이,
이 게임의 승자가 된다.

Trivia Game 세부규칙1

게임을 시작할 때, 플레이어를 추가한다.

플레이어의 초기상태는 위치는 0, 코인은 0 이다.

```
bool Game::add(string playerName)
{
    players.push_back(playerName);
    places[howManyPlayers()] = 0;
    purses[howManyPlayers()] = 0;
    inPenaltyBox[howManyPlayers()] = false;

    cout << playerName << " was added" << endl;
    cout << "They are player number " << players.size() << endl;
    return true;
}
```

기존 코드의 Game.add 메서드, place는 0 (시작지점) Purses 는 0 (코인) 으로 확인할 수 있다.

Trivia Game 세부규칙2

감옥은 돌아오는 턴에서 홀수의 주사위 눈금이 나와야 탈출할 수 있다.
탈출 직후, 해당 주사위 눈금만큼 이동한다.

```
void Game::rolling()
{
    int roll = rand() % 6 + 1;

    cout << players[currentPlayer] << " is the current player" << endl;
    cout << "They have rolled a " << roll << endl;

    if (inPenaltyBox[currentPlayer]) {
        if (roll % 2 != 0) {
            isGettingOutOfPenaltyBox = true;

            cout << players[currentPlayer] << " is getting out of the penalty box" << endl;
            places[currentPlayer] = places[currentPlayer] + roll;
            if (places[currentPlayer] > 11) places[currentPlayer] = places[currentPlayer] - 12;

            cout << players[currentPlayer] << "'s new location is " << places[currentPlayer] << endl;
            cout << "The category is " << currentCategory() << endl;
            askQuestion();
        }
        else {
```

Trivia Game 세부규칙3

기존 코드에서 어느 한 플레이어가 코인 6개를 모으면 게임이 끝난다.

```
do {
    aGame.rolling();

    if (rand() % 9 == 7) {
        notAWinner = aGame.wrongAnswer();
    }
    else {
        notAWinner = aGame.wasCorrectlyAnswered();
    }
} while (notAWinner);
```

```
bool Game::wasCorrectlyAnswered()
{
    if (inPenaltyBox[currentPlayer]) {
        if (isGettingOutOfPenaltyBox) {
            inPenaltyBox[currentPlayer] = false;
            cout << "Answer was correct!!!!" << endl;

            purses[currentPlayer]++;
            cout << players[currentPlayer] << " now has "
                 << purses[currentPlayer] << " Gold Coins." << endl;

            bool winner = didPlayerWin();
            currentPlayer++;
            if (currentPlayer == players.size()) currentPlayer = 0;

            return winner;
        }
        else {
            currentPlayer++;
            if (currentPlayer == players.size()) currentPlayer = 0;
            return true;
        }
    }
    else {
        cout << "Answer was correct!!!!" << endl;
        purses[currentPlayer]++;
        cout << players[currentPlayer] << " now has "
             << purses[currentPlayer] << " Gold Coins." << endl;

        bool winner = didPlayerWin();
        currentPlayer++;
        if (currentPlayer == players.size()) currentPlayer = 0;

        return winner;
    }
}
```

```
bool Game::didPlayerWin()
{
    return !(purses[currentPlayer] == 6);
}
```

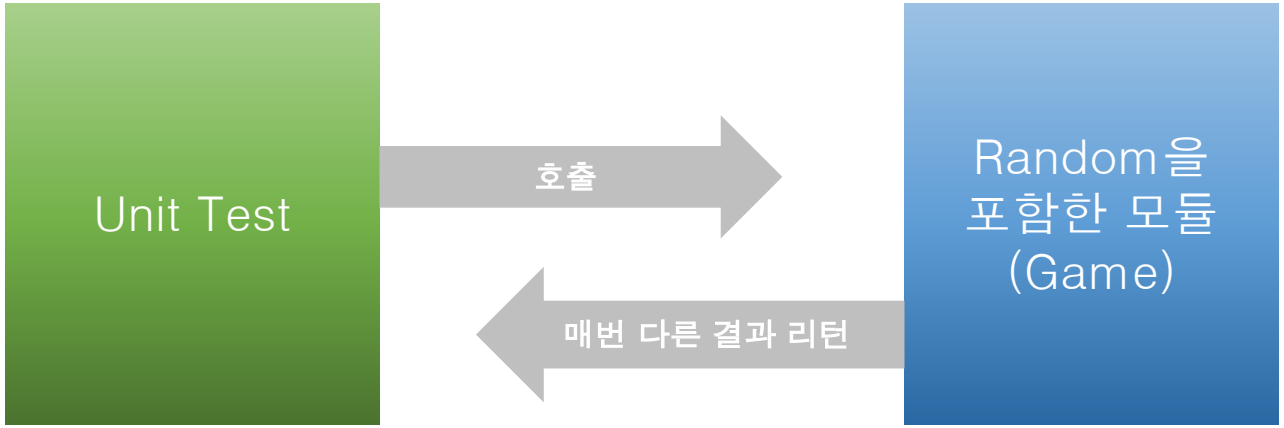
리팩토링을 위한 Unit Test 준비

Unit Test가 가능한 구조로 변경 1

✔ **Unit Test 원칙 – Repeatable**

- 테스트는 반복 가능 해야 한다.

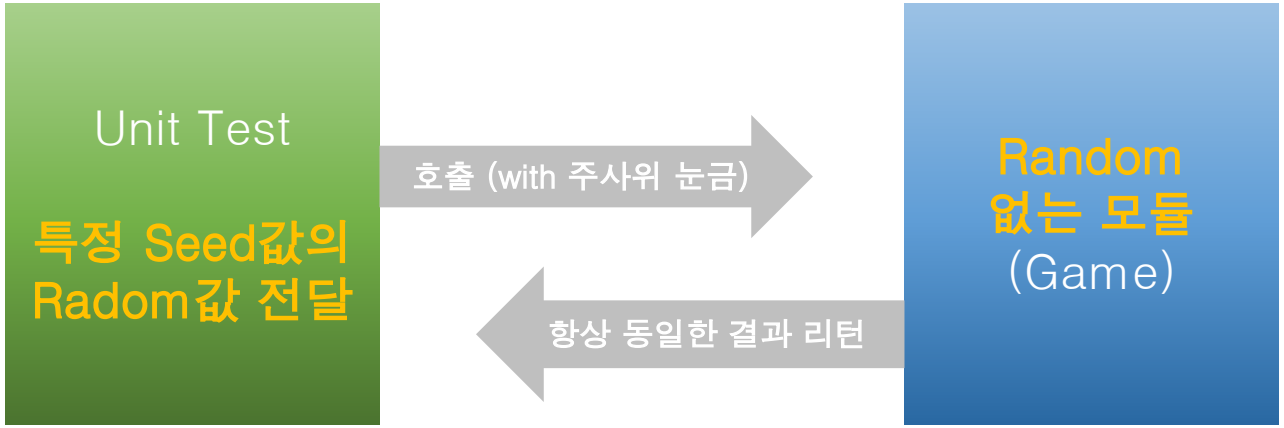
☐ 테스트 할 때 마다 동일한 입력 값에 대해 출력 값은 일정해야 한다.



현재 Trivia 모듈은 Random을 포함하고 있기에
Repeatable한 Unit Test를 만들 수 없다.

Unit Test가 가능한 구조로 변경 2

✔ Random 값을 밖에서 만들어 넣어준다.



인자 값으로 주사위 눈금을 넘겨주면
매번 동일한 결과를 얻을 수 있다.

Unit Test가 가능한 구조로 변경 3

✔ Game 모듈에서 랜덤 값을 구하지 않는다.

```
void Game::rolling()
{
    int roll = rand() % 6 + 1;

    cout << players[currentPlayer] << " is the currer
    cout << "They have rolled a " << roll << endl;
```

랜덤 생성 코드 제거

변경

game.h | game.cpp | game_runner.cpp

Project1

```
20      int howManyPlayers();
21      void rolling(int);
```

```
void Game::rolling(int roll)
{
    cout << players[currentPlayer] << " is the curr
    cout << "They have rolled a " << roll << endl;
```

roll 파라미터 추가

```
int main()
{
    bool notAWinner;

    Game aGame;

    aGame.add("Chet");
    aGame.add("Pat");
    aGame.add("Sue");

    do {
        int roll = rand() % 6 + 1;
        aGame.rolling(roll);
```

랜덤 생성 코드 추가

Golden Master 테스팅

Golden Master 테스트를 다음과 같은 방식으로 제작한다.

1. game.cpp 를 복사하여 game_better.cpp를 만든다.
   game.cpp : 원본
   game_better.cpp : 타겟 = 리팩토링 할 대상

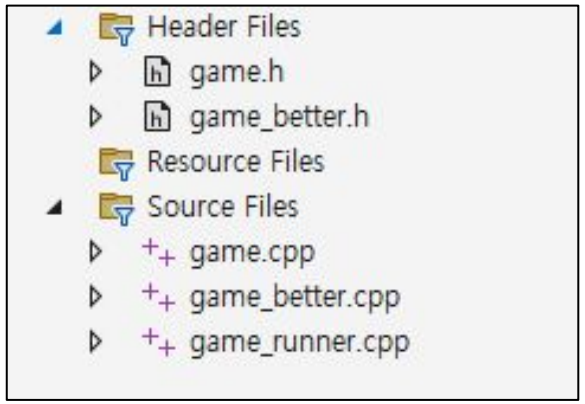
2. 원본(game.cpp)과 타겟(game_better.cpp)에 똑같은 입력을 넣었을 때,
   동일한 출력이 나오는지 비교하는 Unit Test 작성

3. 고정된 Random Seed값 마다 하나의 테스트 케이스를 만든다.
   - 또는 Parameterized testing 수행

game_better.cpp 파일 생성하기

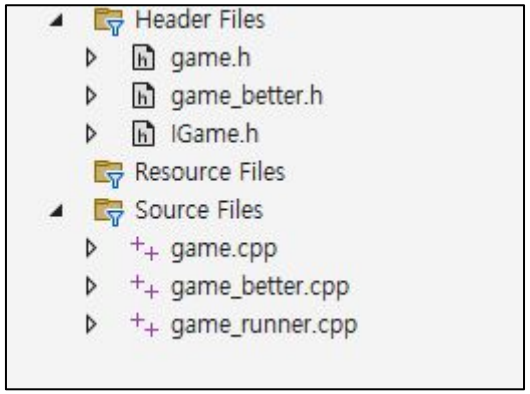
✔ game.cpp, game.h 파일을 복사하여
game_better.cpp, game_better.h 파일을 생성한다.

✔ game_better.cpp 파일, game_better.h 은 리팩토링 대상이 된다.



Interface 추가하기

✔ game 과 game_better 를 하나의 인터페이스로 다루기 위해,
인터페이스 클래스를 하나 추가한다.




```
class IGame {
public:
    virtual bool add(string playerName) = 0;
    virtual void rolling(int roll) = 0;
    virtual bool wasCorrectlyAnswered() = 0;
    virtual bool wrongAnswer() = 0;
};
```

Test 파일 만들기

✔ game_test 파일을 생성하여 우측과 같이 runner 코드를 이용해 완성한다.

✔ playGame 함수를 호출하여 출력 결과를 비교할 수 있다.

```
class ParameterizedTestFixture : public testing::TestWithParam<int> {
public:
    string playGame(IGame& aGame, int seed) {
        std::ostringstream oss;
        auto oldCoutStreamBuf = std::cout.rdbuf();
        std::cout.rdbuf(oss.rdbuf()); // 새로운 버퍼로 redirection

        bool notAWinner = true;

        aGame.add("Chet");
        aGame.add("Pat");
        aGame.add("Sue");

        srand(seed);
        do {
            aGame.rolling(rand() % 6 + 1);

            if (rand() % 9 == 7) {
                notAWinner = aGame.wrongAnswer();
            }
            else {
                notAWinner = aGame.wasCorrectlyAnswered();
            }
        } while (notAWinner);

        std::cout.rdbuf(oldCoutStreamBuf); // 복원
        return oss.str();
    }
};
```

Test Case 추가하기

✔ playGame 을 두 번 후출하고, 두 결과값을 비교하는 방식으로 검증한다

```
TEST_F(ParameterizedTestFixture, goldenMaster_sample) {
    int seed = 1;
    Game game_origin;
    GameBetter game_refactor;

    string expected = playGame(game_origin, seed);
    string actual   = playGame(game_refactor, seed);

    EXPECT_EQ(expected, actual);
}
```

✔ 여기까지 완성된 소스코드

- https://gist.github.com/mincoding1/402d03b655a7aa0946f2f8716daf56cf

개선하기 : Parameterized Test

✔ gTest의 TEST_P 를 사용하여 파라미터 값을 변경하며 테스트하는 코드로 Unit Test 를 개선한다.

```
INSTANTIATE_TEST_CASE_P(
    goldenMaster,
    ParameterizedTestFixture,
    testing::Values(
        1, 50, 100, 777
    ));
```

```
TEST_P(ParameterizedTestFixture, goldenMaster) {
    int seed = GetParam();
    Game game_origin;
    GameBetter game_refactor;

    string expected = playGame(game_origin, seed);
    string actual = playGame(game_refactor, seed);

    EXPECT_EQ(expected, actual);
}
```

리팩토링 시작

리팩토링 연습 포인트

연습 포인트

- 중복 코드 식별 후 Extract Method
- Pure function 으로 추출하기 (no state, no side effect 로 만들어서 커플링 최소화)
- Game 클래스로부터 다른 클래스 추출하기

객체지향의 원칙 적용하기

- 책임 식별하기 (SRP, Single Responsibility Principle)
- 중복 제거 (DRY, Do not Repeat Yourself)

리팩토링 이후, 기능추가 요구사항

기능추가 요구사항

1. 최대 플레이어 수를 6명으로
2. 새로운 질문 카테고리 추가 ex) “ART”
3. 게임을 시작하려면 최소 2명 이상
4. 게임이 시작된 이후 새로운 플레이어 추가 금지
5. 플레이어 동일한 이름 금지
6. 3번 연속 정답을 맞추는 경우, 그 다음 정답에 대해 2점을 얻기