고급프로그래밍
2차 과제 보고서

학과	소프트웨어학부
학번	2020203090
이름	한용옥

23.05.27

퍼즐 클래스 구조

아래는 퍼즐 클래스의 멤버 변수와 함수를 접근 구분에 따라 정리한 표다 이름만 명시하였다.

public
멤버변수
없다
멤버함수
Puzzle 생성자 initialize, randomize 퍼즐 초기화를 위한 함수 update 퍼즐게임 진행을 위한 함수 valid_pos 위치가 올바른지 검사하는 함수 swapJewels 인접한 보석을 서로 바꾸는 함수 setJewel, getJewel 특정 위치의 보석을 지정/가져오는 함수 getNumRows, getNumColumns getcombo, getscore private로 선언된 멤버변수를 가져온다 getJewelType, getJewelLetter 문자와 보석을 변환하는 함수
멤버클래스
Chain 체인을 표현하기 위한 클래스

private
멤버변수
num_rows, num_columns 퍼즐의 사이즈 puzzle 보석을 담는 벡터 chain_arr 체인을 담는 벡터 update_status update함수 제어용 변수 combo, score 점수관련변수
멤버함수
horizon_check, vertical_check setnone update_A 내부 함수 bubbleup update_B 내부 함수 update_A, update_B update 내부 함수

퍼즐 내부 데이터와 퍼즐의 멤버함수를 구현하기 위해 필요한 함수는 모두 private로 선언해 외부에서 접근하지 못하도록 했다

데이터 저장 방식

퍼즐은 보석의 행렬로 볼 수있다
따라서 퍼즐을 vector<vector<Jewel>>형식으로 표현하기로 했다

```
vector<vector<Jewel>> puzzle;
```

마찬가지로 체인을 vector<Chain>형식으로 표현하기로 했다

```
vector<Chain> chain_arr;
```

퍼즐 생성

생성자는 다음과 같은 역할을 한다.

1. 인자 검사

```
Puzzle::Puzzle(int num_rows, int num_columns) {
    if (num_rows <= 0 || num_columns <= 0) error("invalid size");
```

행이나 열의 크기가 0 이하이면 오류가 나게 설정했다

2. num_rows, num_columns 초기화

```
this->num_rows = num_rows;
this->num_columns = num_columns;
```

멤버 변수에 값을 부여한다

3. 퍼즐을 Jewel::NONE으로 채우기

```
for (int i = 0; i < num_rows; i += 1) {
    vector<Jewel> temp;
    for (int j = 0; j < num_columns; j += 1) {
        temp.push_back(Jewel::NONE);
    }
    puzzle.push_back(temp);
}
```

vector<Jewel>을 만들고 열 개수만큼 Jewel::NONE을 채우고
그렇게 만들어진 vector<Jewel>을 행 개수만큼 puzzle에 넣는다

4. 난수생성을 위한 시드 설정

```
srand((int)time(NULL));
```

퍼즐 객체가 만들어질 때 시드를 설정하게 했다.

퍼즐 초기 설정

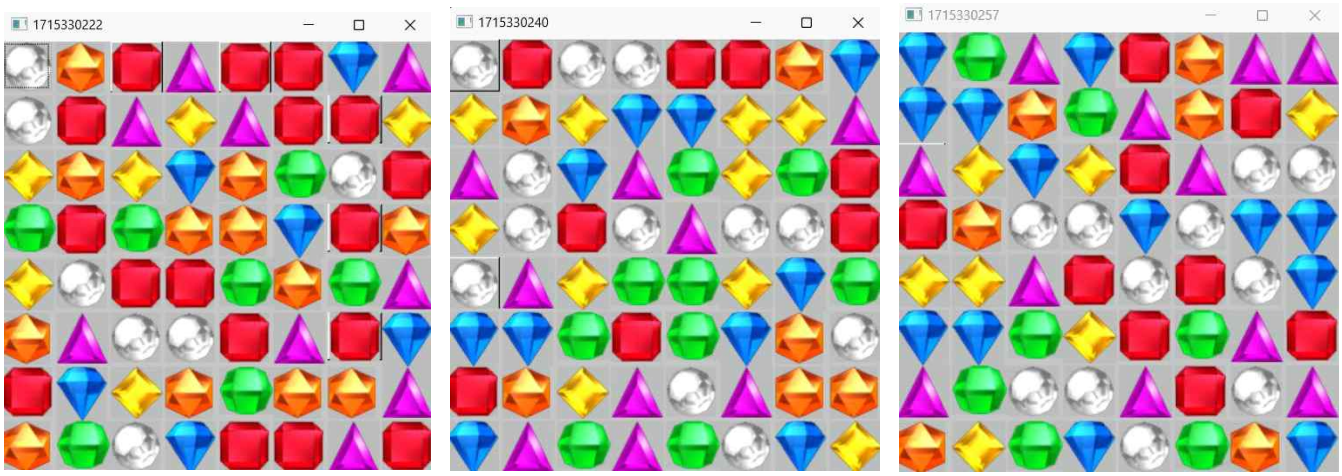
randomise()

배열의 모든 자리를 돌면서 그 때마다 난수를 생성해 그에 맞는 보석으로 채워넣었다.

```
void Puzzle::randomize() {
    for (int i = 0; i < num_rows; i += 1) {
        for (int j = 0; j < num_columns; j += 1) {
            int temp = rand();
            puzzle[i][j] = (Jewel)(temp % 7);
        }
    }
}
```

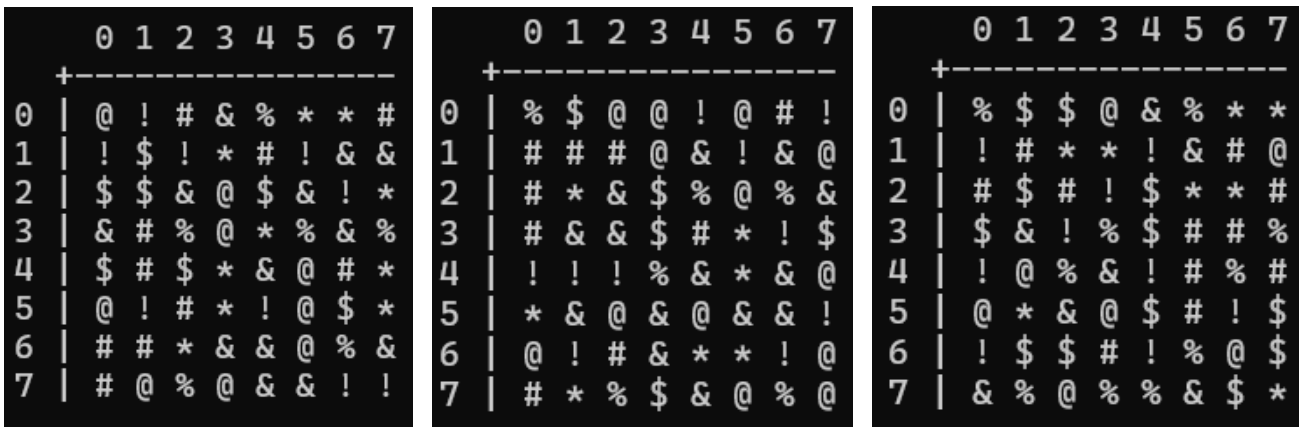
보석은 Jewel::NONE을 제외하면 7개이므로 난수를 7로 나눈 나머지에 보석이 대응되게 하여 랜덤하게 채워넣었다.

GUI상 실행사진



창 이름을 시간으로 하고 찍은 사진이다 랜덤하게 보석이 분포되어있고 할때마다 다르다는 것을 알 수 있다.

TUI 실행사진



보석들이 무작위로 분포되는 모습이다.

initialize()

랜덤과 비슷하게 배열의 모든 자리를 돌면서 그 때마다 텍스트를 평가해 그에 맞는 보석으로 채워넣었다.

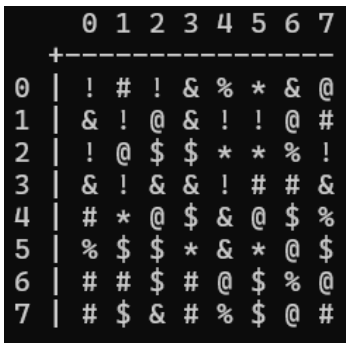
```
bool Puzzle::initialize(const std::string& jewel_list) {
    if (jewel_list.size() != num_rows * num_columns) return false;
```

문자열의 길이는 행*열이어야 하므로 맞지 않으면 false를 반환한다.

```
for (int i = 0; i < num_rows; i += 1) {
    for (int j = 0; j < num_columns; j += 1) {
        puzzle[i][j] = getJewelType(jewel_list[i * num_rows + j]);
    }
}
return true;
```

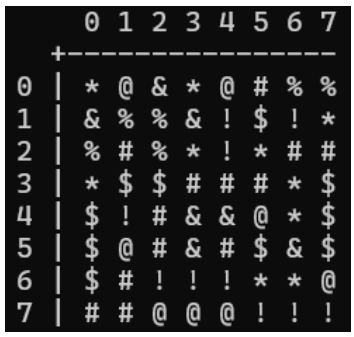
텍스트를 행 단위로 끊어가며 읽고, 돌면서 getJewelType를 이용해 텍스트에 맞는 보석을 채웠다.

실행 사진



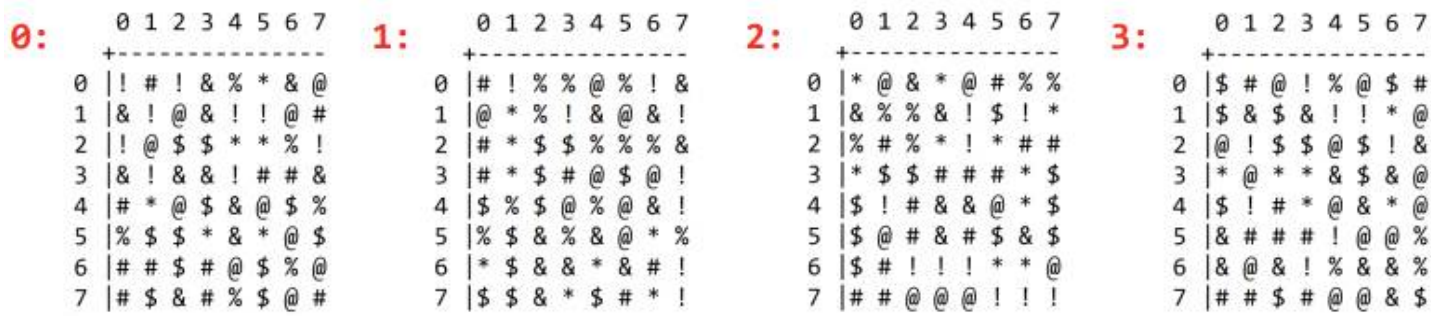
```
    0 1 2 3 4 5 6 7
  +----------------
0 | ! # ! & % * & @
1 | & ! @ & ! ! @ #
2 | ! @ $ $ * * % !
3 | & ! & & ! # # &
4 | # * @ $ & @ $ %
5 | % $ $ * & * @ $
6 | # # $ # @ $ % @
7 | # $ & # % $ @ #
```

```
    0 1 2 3 4 5 6 7
  +----------------
0 | # ! % % @ % ! &
1 | @ * % ! & @ & !
2 | # * $ $ % % % &
3 | # * $ # @ $ @ !
4 | $ % $ @ % @ & !
5 | % $ & % & @ * %
6 | * $ & & * & # !
7 | $ $ & * $ # * !
```



```
    0 1 2 3 4 5 6 7
  +----------------
0 | * @ & * @ # % %
1 | & % % & ! $ ! *
2 | % # % * ! * # #
3 | * $ $ # # # * $
4 | $ ! # & & @ * $
5 | $ @ # & # $ & $
6 | $ # ! ! ! * * @
7 | # # @ @ @ ! ! !
```

```
    0 1 2 3 4 5 6 7
  +----------------
0 | $ # @ ! % @ $ #
1 | $ & $ & ! ! * @
2 | @ ! $ $ @ $ ! &
3 | * @ * * & $ & @
4 | $ ! # * @ & * @
5 | & # # # ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $
```



```
0:
    0 1 2 3 4 5 6 7
  +----------------
0 |! # ! & % * & @
1 |& ! @ & ! ! @ #
2 |! @ $ $ * * % !
3 |& ! & & ! # # &
4 |# * @ $ & @ $ %
5 |% $ $ * & * @ $
6 |# # $ # @ $ % @
7 |# $ & # % $ @ #

1:
    0 1 2 3 4 5 6 7
  +----------------
0 |# ! % % @ % ! &
1 |@ * % ! & @ & !
2 |# * $ $ % % % &
3 |# * $ # @ $ @ !
4 |$ % $ @ % @ & !
5 |% $ & % & @ * %
6 |* $ & & * & # !
7 |$ $ & * $ # * !

2:
    0 1 2 3 4 5 6 7
  +----------------
0 |* @ & * @ # % %
1 |& % % & ! $ ! *
2 |% # % * ! * # #
3 |* $ $ # # # * $
4 |$ ! # & & @ * $
5 |$ @ # & # $ & $
6 |$ # ! ! ! * * @
7 |# # @ @ @ ! ! !

3:
    0 1 2 3 4 5 6 7
  +----------------
0 |$ # @ ! % @ $ #
1 |$ & $ & ! ! * @
2 |@ ! $ $ @ $ ! &
3 |* @ * * & $ & @
4 |$ ! # * @ & * @
5 |& # # # ! @ @ %
6 |& @ & ! % & & %
7 |# # $ # @ @ & $
```

위는 실행 결과, 아래는 문제에 나온 지정된 배열이다
둘이 똑같음을 알 수 있다.

위치 검사

인자로 넘겨준 위치가 0과 최대 행/열 크기 안의 값인지 확인한다.

```
bool Puzzle::valid_pos(std::pair<int, int> loc) const {
    if (loc.first < 0 || loc.second < 0) return false;
    if (loc.first >= getNumRows() || loc.second >= getNumColumns()) return false;
    return true;
}
```

맞으면 true, 틀리면 false반환

특정 위치의 보석 검출

```
Jewel Puzzle::getJewel(std::pair<int, int> loc) const{
    if (!valid_pos(loc)) return Jewel::NONE;
    return puzzle[loc.first][loc.second];
}
```

정당한 위치면 puzzle에서의 해당 위치 값 반환 아니면 Jewel::NONE 반환

특정 위치의 값을 특정 보석으로 설정

```
bool Puzzle::setJewel(std::pair<int, int> loc, Jewel jewel) {
    if (!valid_pos(loc)) return false;
    puzzle[loc.first][loc.second] = jewel;
    return true;
}
```

값이 정당하면 그 위치를 설정한다 아니면 false 반환

보석 위치 옮기기

서로 바꿀 보석의 위치가 a, b라 하자 보석을 바꾸기 위해선 a, b가 아래와 같은 조건을 만족해야한다

1. a가 정당한 위치
2. b가 정당한 위치
3. a와 b는 x, y 좌표 중 하나는 같으면서 다른 하나의 차이는 1이어야 함

```
bool Puzzle::swapJewels(std::pair<int, int> prev_loc, std::pair<int, int> next_loc) {
    if (!(valid_pos(prev_loc) && valid_pos(next_loc))) return false;
    if (!((prev_loc.first == next_loc.first && abs(prev_loc.second - next_loc.second) == 1)
        || (prev_loc.second == next_loc.second && abs(prev_loc.first - next_loc.first) == 1)))
        return false;
```

먼저 위 3개 조건을 검사한 뒤, 맞지 않으면 false를 반환한다.

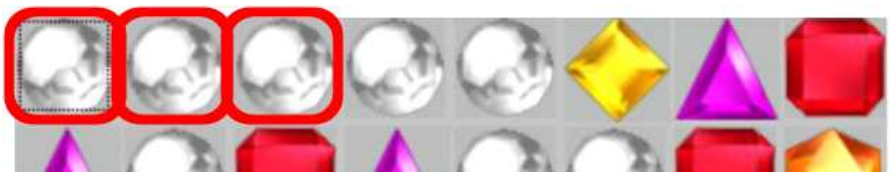
```
Jewel temp = puzzle[prev_loc.first][prev_loc.second];
puzzle[prev_loc.first][prev_loc.second] = puzzle[next_loc.first][next_loc.second];
puzzle[next_loc.first][next_loc.second] = temp;
return true;
```

조건이 맞는 경우에 puzzle에서 서로의 위치를 바꾸고 true를 반환한다.

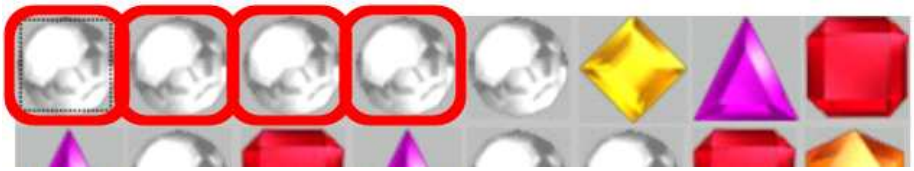
update()
update()는 크게 update_A와 update_B로 나뉘어있다

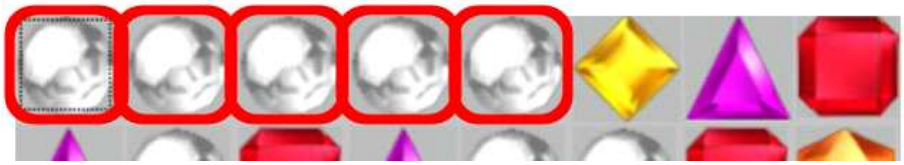
update_A
update_A에서는
1. 체인 검출
2. 체인에 해당하는 부분을 Jewel::NONE으로 변환
3. combo변수 변경
세 가지 역할을 수행한다

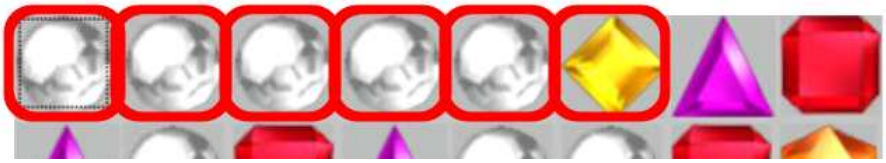
체인 검출
체인의 조건은 "같은 보석이 가로/세로로 3개 이상"이다.
따라서 체인을 검출하려면
1. 가로/세로 3개짜리 필터를 준비한다
2. 필터를 배열의 모든 위치에 대본다
그리고 필터 안의 보석이 같은지 확인한다



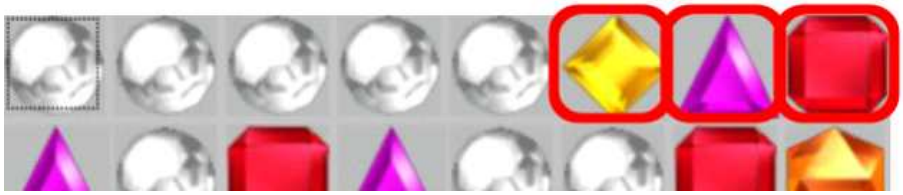
3. 2번 조건이 참이면 체인이 아닐 때까지 필터를 늘려본다







4. 필터를 바탕으로 체인을 저장한다
위 그림에서는 Chain{ 흰색, <필터의 처음>, <필터의 끝 - 1> }가 저장된다
5. 3.에서 나온 끝 지점부터 다시 반복한다



6. 가로 필터로 다 돌았으면 세로 필터로 한번 더 돌린다.
위와 같은 과정을 거치면 모든 체인을 chain_arr에 넣을 수 있다

코드 구현부

```
void Puzzle::horizon_check() {
    for (int i = 0; i < getNumRows(); i += 1) {
        for (int j = 0; j < getNumColumns() - 2; j += 1) {
```

2번을 위한 반복이다 puzzle의 모든 부분에 대해 접근한다
열에서 최대가 끝에서 두 번째인 이유는 필터가 3개이므로 그 너머로 가면 오류이기 때문이다.

```
if (puzzle[i][j] == puzzle[i][j + 1] && puzzle[i][j + 1] == puzzle[i][j + 2]) {
```

1, 2번 부분이다 조건식이 가로 3개짜리 필터이다

```
Jewel J = getJewel(make_pair(i, j));
int temp = 3;
```

체인을 저장하기 위한 정보 생성

```
while (j + temp < getNumColumns() && puzzle[i][j + temp] == J) temp += 1;
```

3번 과정이다 다른 보석이 나올 때까지 필터를 늘려 확인한다

```
chain_arr.push_back(Chain{ J, make_pair(i, j), make_pair(i, j + temp - 1) });
```

체인을 chain_arr에 저장한다 3번 과정에서 필터의 끝은
실제 체인보다 1 크므로 1을 빼고 저장한다.

```
j += temp - 1;
```

위와 같은 알고리즘은 같은 체인이 여러 번 저장될 수 있기 때문에 시작점을 체인의 끝으로 옮겨서 중복을 막는다.

같은 방식이지만 필터가 세로인 함수이다.

```
void Puzzle::vertical_check() {
    for (int j = 0; j < getNumColumns(); j += 1) {
        for (int i = 0; i < getNumRows() - 2; i += 1) {
            if (puzzle[i][j] == puzzle[i + 1][j] && puzzle[i + 1][j] == puzzle[i + 2][j]) {
                Jewel J = getJewel(make_pair(i, j));
                int temp = 3;
                while (i + temp < getNumRows() && puzzle[i + temp][j] == J) temp += 1;
                chain_arr.push_back(Chain{ J, make_pair(i, j), make_pair(i + temp - 1, j) });
                i += temp - 1;
            }
        }
    }
}
```

위 과정을 거치면 현재 체인이 chain_arr에 빠짐없이 저장된 상태이다.

Jewel::NONE으로 변환

chain_arr의 모든 원소에 대해
원소의 시작과 끝 위치를 통해 체인인 위치를 Jewel::NONE으로 변환한다.

```
void Puzzle::setnone() {
    for (const Chain& c : chain_arr) {
```

배열의 모든 원소에 대해 반복한다

```
for (int i = 0; i <= c.end.first - c.start.first; i += 1) {
    puzzle[c.start.first + i][c.start.second] = Jewel::NONE;
}
for (int j = 0; j <= c.end.second - c.start.second; j += 1) {
    puzzle[c.start.first][c.start.second + j] = Jewel::NONE;
}
```

체인은 모두 일자 형태이므로 원소들의 x나 y좌표 중 한 좌표는 모두 같다.
따라서 위의 두 for문 중 하나만 완전히 실행된다.
위는 세로형 체인, 아래는 가로형 체인에 대해 작동하며 체인의 시작과 끝까지 위치한 puzzle의 원소를 Jewel::NONE으로 바꾼다.

update_A의 구현과 콤보 변수의 변경

```
bool Puzzle::update_A() {
    horizon_check();
    vertical_check();
```

먼저 체인이 있는지 확인한다

```
if (chain_arr.size() == 0) {
    combo = 0;
    return false;
}
```

체인이 있다면 chain_arr에 저장되므로 체인이 없다면 chain_arr이 빈 배열이다 체인이 더 이상 없다면 콤보가 끊긴 것이므로 변수를 0으로 지정한다.

```
setnone();
chain_arr.clear();
combo += 1;
return true;
```

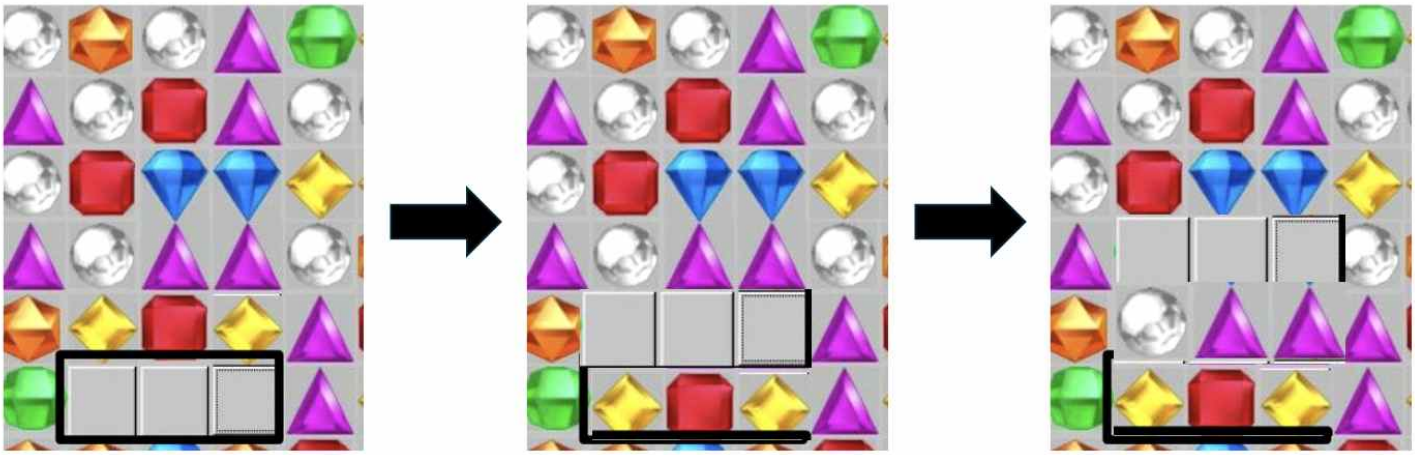
체인이 있다면 NONE으로 변환하고 콤보를 1증가하는 구문이다
체인이 있으므로 true를 반환한다.

update_B

update_A를 통해 체인을 검출하고, Jewel::NONE으로 바꿨다
update_B에서는 아래 세 가지 역할을 한다.

1. Jewel::NONE 자리를 위에서 내려오는 식으로 채운다.
2. 위에 생긴 빈 자리를 랜덤하게 채운다.
3. 없어진 보석을 세어 점수를 계산한다

1번의 구현을 보자
위에서 내려와 자리를 채운다는건



위 그림과 같이 NONE과 바로 위 원소를 바꾸는 것의 반복으로
생각할 수 있다.
따라서 puzzle의 모든 위치에 대해 NONE이면 바로 위와 계속 바꾸면
NONE은 위로 뜨고 나머지는 순서를 유지한 채 아래로 가라앉을 것이다.

1번 역할은 bubbleup()으로 구현하였다.

```
void Puzzle::bubbleup() {
    for (int i = 0; i < getNumRows(); i += 1) {
        for (int j = 0; j < getNumColumns(); j += 1) {
            if (puzzle[i][j] == Jewel::NONE) {
```

배열의 모든 위치에 대해 NONE인지 확인한다.

```
for (int k = i; k > 0; k -= 1) {
    swapJewels(make_pair(k, j), make_pair(k - 1, j));
}
```

NONE인 경우 swapJewels를 이용해 바로 위의 원소와 계속 교환한다

update_B 안에 2,3번의 구현이 담겨있다

```
bool Puzzle::update_B() {
    bubbleup();
    int temp_score = 0;
```

점수 계산을 위한 변수를 만든다.

bubbleup()으로 인해 NONE들은 전부 위로 뜬 상태이다.

```
for (int i = 0; i < getNumRows(); i += 1) {
    for (int j = 0; j < getNumColumns(); j += 1) {
        if (puzzle[i][j] == Jewel::NONE) {
```

배열의 모든 위치에 대해 NONE인지 확인한다.

```
if (puzzle[i][j] == Jewel::NONE) {
    puzzle[i][j] = (Jewel)(rand() % 7);
    temp_score += 1;
}
```

NONE인 경우 난수를 이용해 무작위로 보석을 정해 채워넣고 점수계산을 위한 변수는 1증가한다.

```
score += temp_score * combo;
return true;
```

클래스 멤버 변수인 score를 계산하여 저장한다.
점수 = (없앤 보석의 수) * (현재 콤보 수)이다.
동작이 완료되었으므로 true를 반환한다.

update의 구현

update는 실행될 때마다 update_A 또는 update_B만
정확히 한 번 실행되어야 한다

```
bool update_status{ true };
```

따라서 update가 실행되었는지 확인하는 멤버변수 update_status를 이용해,
update_status의 상태에 따라 update의 행동을 결정하면 된다.
맨 처음 update의 동작은 체인 감지이므로 true로 초기화한다.

아래는 update의 구현이다

```
bool Puzzle::update() {
    if (update_status) {
        update_status = false;
        if (!update_A()) return false;
        else return true;
    }
    else {
        update_status = true;
        return update_B();
    }
}
```

update_status의 상태에 따라 update_A 또는 update_B 하나만 실행된다.
update_A는 체인이 없을 때 거짓, 있을 때 참이다.
update_B는 에러가 나지 않으면 참을 반환한다.
따라서 update()는 둘 중 하나만 실행되며
체인이 없을 때 false를 반환, 나머지 경우에는 true를 반환한다.

TUI 게임 진행

게임 시작

```
int main()
try {
    promft_game_title();
    game_menu();
```

게임이 시작되면 안내 문구와 함께 메뉴창이 실행된다.

메뉴 선택

```
void game_menu() {
    while (true) {
        promft_game_menu();
        int temp = input();
        while (!(1 <= temp && temp <= 4)) {
            promft_menu_error();
            temp = input();
        }
```

메뉴는 1부터 4까지의 정수로 받는다. input() 함수로 정수를 받고 범위 안에 있는지 검사하는 구문이다.

```
switch (temp) {
case 1:
    game_random();
    break;
case 2:
    game_pre();
    break;
case 3:
    print_rank();
    break;
case 4:
    return;
}
```

번호에 맞게 실행할 함수가 정해져있다.

실행 사진

```
<< BEJEWELED >>

[1] 랜덤 퍼즐
[2] 정해진 퍼즐
[3] 등수 보기
[4] 종료
>> 메뉴를 선택하세요 (1~4) :
```

문자열로 초기화하는 경우

```
void game_pre() {
    promft_choose_pre();
    int temp = input();
    while (!(0 <= temp && temp <= 3)) {
        promft_choose_error();
        temp = input();
    }
```

몇 번 문자열로 생성할지 범위에 맞을때까지 물어본다.

```
Puzzle p{ presize,presize };
p.initialize(predefined_puzzles[temp]);
game_main(p);
```

퍼즐클래스의 initialize를 이용해 퍼즐을 생성한 뒤,
게임을 시작한다

실행 사진

```
[1] 랜덤 퍼즐
[2] 정해진 퍼즐
[3] 등수 보기
[4] 종료
>> 메뉴를 선택하세요 (1~4) : 2
몇 번 배열로 시작하겠습니까? (0~3) : 3

***
두 위치 모두 (0, 0)일 경우 게임이 종료됩니다
***

    0 1 2 3 4 5 6 7
  +----------------
0 | $ # @ ! % @ $ #
1 | $ & $ & ! ! * @
2 | @ ! $ $ @ $ ! &
3 | * @ * * & $ & @
4 | $ ! # * @ & * @
5 | & # # # ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $


    0 1 2 3 4 5 6 7
  +----------------
0 | $ # @ ! % @ $ #
1 | $ & $ & ! ! * @
2 | @ ! $ $ @ $ ! &
3 | * @ * * & $ & @
```

initialize() 로 생성되고 게임진행이 되는 모습이다.

랜덤 퍼즐인 경우

```
void game_random() {
    promft_psize();
    std::pair<int, int> psize = loc_input();
    while (!((rand_down <= psize.first && psize.first <= rand_up)
          &&(rand_down <= psize.second && psize.second <= rand_up))){
        promft_psize_error();
        psize = loc_input();
    }
```

먼저 플레이할 퍼즐의 사이즈를 물어본다. loc_input()으로 두 정수 쌍을 받고 유효한 퍼즐 사이즈인지 검사한다.

```
Puzzle p{ psize.first, psize.second };
p.randomize();
game_main(p);
```

유효하다면 퍼즐을 그 사이즈에 맞게 생성하고, randomize()를 이용해 초기상태로 만들고 게임을 시작한다.

실행 사진

```
[1] 랜덤 퍼즐
[2] 정해진 퍼즐
[3] 등수 보기
[4] 종료
>> 메뉴를 선택하세요 (1~4) : 1
퍼즐의 크기를 입력하세요 3<=크기<=20 인 (행, 열) : 4 10

***
두 위치 모두 (0, 0)일 경우 게임이 종료됩니다
***

    0 1 2 3 4 5 6 7 8 9
  +--------------------
0 | & & ! $ # # % @ @ *
1 | $ $ % $ % % @ * @ $
2 | @ @ % ! @ $ $ ! & $
3 | @ % % % ! ! % $ @ @


    0 1 2 3 4 5 6 7 8 9
  +--------------------
0 | & & ! $ # # % @ @ *
```

randomize()로 생성되고 게임진행이 되는 모습이다.

게임 진행

게임 진행은 game_main 함수에서 담당한다 아래는 구현이다

```
void game_main(Puzzle& p) {
    promft_game_info();
    while (true) {
```

먼저 게임을 어떻게 종료하는지 알린다.
게임은 계속되어야 한다. 따라서 무한루프로 들어간다

```
do{
    print_puzzle(p);
    promft_combo(p.getcombo());
} while (p.update() == true);
p.update();
```

일단 배열과 콤보 상태를 표시한다.
update는 체인이 없을때만 false이므로 체인이 없을 때까지
update()와 퍼즐 표시를 반복한다.
따라서 위 반복문은 (체인 감지->체인 none으로 변환->퍼즐/콤보 표시->
none 위로 올리고 빈자리 채우기->퍼즐/콤보 표시) 괄호 안을 반복하게 된다
반복문을 빠져나왔을 경우 update_A까지만 실행되었기 때문에
update를 한 번 더 호출해 update_B를 호출하고
다음 차례에 정상적으로 체인을 감지하게 했다.

```
promft_score(p.getscore());
if (!switchgem(p)) {
    game_score(p.getscore());
    return;
}
```

보석 교환에 대한 안내를 하고, 보석을 바꾼다.
바꾸는 위치가 종료 조건이면 getscore를 이용해 퍼즐의 점수를 넘기고
점수등록으로 넘어간다.

게임 진행 실행 사진

```
>> 메뉴를 선택하세요 (1~4) : 2
몇 번 배열로 시작하겠습니까? (0~3) : 3

***
두 위치 모두 (0, 0)일 경우 게임이 종료됩니다
***

    0 1 2 3 4 5 6 7
  +----------------
0 | $ # @ ! % @ $ #
1 | $ & $ & ! ! * @
2 | @ ! $ $ @ $ ! &
3 | * @ * * & $ & @
4 | $ ! # * @ & * @
5 | & # # # ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $
```

종료방법을 알려주고 정해진 배열로 시작한다

```
    0 1 2 3 4 5 6 7
  +----------------
0 | $ # @ ! % @ $ #
1 | $ & $ & ! ! * @
2 | @ ! $ $ @ $ ! &
3 | * @ * * & $ & @
4 | $ ! # * @ & * @
5 | &         ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $
```

맨 처음엔 update_A가 실행되어 체인을 찾아 없앤다

```
    0 1 2 3 4 5 6 7
  +----------------
0 | $ * * $ % @ $ #
1 | $ # @ ! ! ! * @
2 | @ & $ & @ $ ! &
3 | * ! $ $ & $ & @
4 | $ @ * * @ & * @
5 | & ! # * ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $
```

체인이 있었으므로 계속 반복한다 이번엔 update_B가 실행되어 보석을 밑으로 내리고 빈자리를 랜덤하게 채운다

```
    0 1 2 3 4 5 6 7
  +----------------
0 | $ * * $ % @ $ #
1 | $ # @         * @
2 | @ & $ & @ $ ! &
3 | * ! $ $ & $ & @
4 | $ @ * * @ & * @
5 | & ! # * ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $

2 COMBO!
```

update_B은 오류가 없는 한 true만을 반환하므로 계속 반복되어 update_A가 실행된다 더불어 두 번 연속이므로 콤보 표시가 나온다

```
    0 1 2 3 4 5 6 7
  +---------------
0 | $ * * ! $ $ $ #
1 | $ # @ $ % @ * @
2 | @ & $ & @ $ ! &
3 | * ! $ $ & $ & @
4 | $ @ * * @ & * @
5 | & ! # * ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $

2 COMBO!
```

update_B 실행

```
    0 1 2 3 4 5 6 7
  +---------------
0 | $ * * !       #
1 | $ # @ $ % @ * @
2 | @ & $ & @ $ ! &
3 | * ! $ $ & $ & @
4 | $ @ * * @ & * @
5 | & ! # * ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $

3 COMBO!!
```

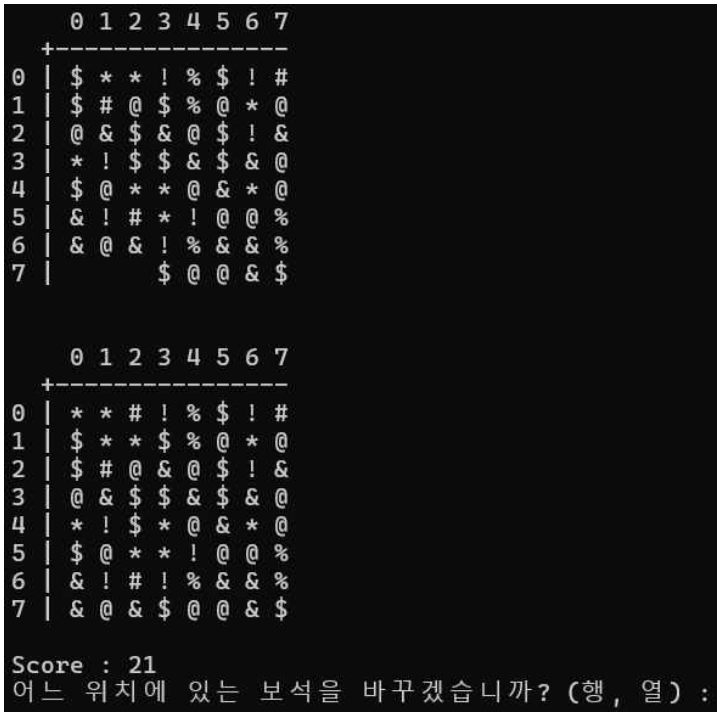
체인 발생

```
    0 1 2 3 4 5 6 7
  +---------------
0 | $ * * ! % $ ! #
1 | $ # @ $ % @ * @
2 | @ & $ & @ $ ! &
3 | * ! $ $ & $ & @
4 | $ @ * * @ & * @
5 | & ! # * ! @ @ %
6 | & @ & ! % & & %
7 | # # $ # @ @ & $

3 COMBO!!
Score : 18
어느 위치에 있는 보석을 바꾸겠습니까? (행, 열) :
```

체인이 끝나 반복을 멈추고 입력을 받는 모습이다

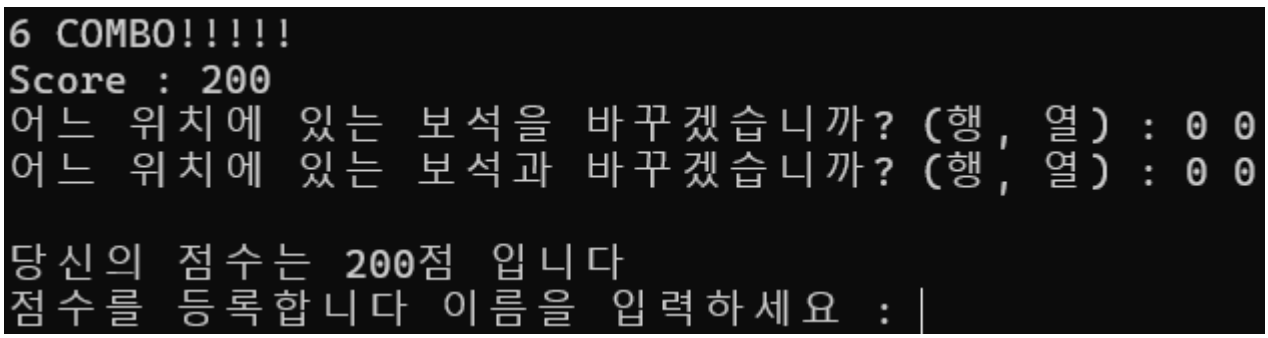
```
Score : 18
어느 위치에 있는 보석을 바꾸겠습니까? (행, 열) : 7 2
어느 위치에 있는 보석과 바꾸겠습니까? (행, 열) : 7 3

    0 1 2 3 4 5 6 7
  +---------------
0 | $ * * ! % $ ! #
1 | $ # @ $ % @ * @
2 | @ & $ & @ $ ! &
3 | * ! $ $ & $ & @
4 | $ @ * * @ & * @
5 | & ! # * ! @ @ %
6 | & @ & ! % & & %
7 | # # # $ @ @ & $
```

체인을 발생시키는 입력을 넣어보자



체인이 없을때까지 반복한 후 다시 입력을 받는다

game_main의 실행에서 알 수 있는 사실은 update가 의도했던 대로 한 번 실행 될 때 update_A 또는 update_B 둘 중 한 기능만 실행된다는 점이다. 그리고 점수 계산도 공식대로 잘됨을 알 수 있다.



종료 조건인 (0,0),(0,0)을 입력시 점수 등록으로 넘어가는 장면이다.

보석의 교환

게임에서 보석의 교환은 switchgem 함수가 담당한다 아래는 구현이다

```
bool switchgem(Puzzle& p) {
    while (true) {
        promft_choose_gem_1();
        std::pair<int, int> prev_loc = loc_input();
        promft_choose_gem_2();
        std::pair<int, int> next_loc = loc_input();
```

조건이 맞을 때까지 입력을 받아야하므로 무한 루프를 쓴다
서로 바꿀 두 개의 보석의 위치를 받는다

```
if (prev_loc == make_pair(0, 0) && next_loc == make_pair(0, 0)) return false;
if (p.swapJewels(prev_loc, next_loc)) return true;
else promft_choose_gem_error();
```

종료 조건은 두 위치가 (0,0), (0,0)인 경우이다. 종료 조건인지 확인한다.
퍼즐 클래스의 swapJewels 사용해 바꾼다 정상적으로 바꿔진다면 swapJewels는 true이므로 끝낸다 아니면 오류라고 알려준다 무한 루프 안에 있으므로 정상적으로 바꿔질 때까지 반복한다.

실행 사진

```
Score : 0
어느 위치에 있는 보석을 바꾸겠습니까? (행, 열) : 0 0
어느 위치에 있는 보석과 바꾸겠습니까? (행, 열) : 8 0
잘못된 위치 입력입니다 다시 입력하세요
어느 위치에 있는 보석을 바꾸겠습니까? (행, 열) : 9 1
어느 위치에 있는 보석과 바꾸겠습니까? (행, 열) : 6 6
잘못된 위치 입력입니다 다시 입력하세요
어느 위치에 있는 보석을 바꾸겠습니까? (행, 열) : 2 2
어느 위치에 있는 보석과 바꾸겠습니까? (행, 열) : 3 3
잘못된 위치 입력입니다 다시 입력하세요
```

1,2,3번의 조건에 맞지 않는 입력은 받지 않고 다시 입력하게 한다

```
    0 1 2 3 4 5 6 7
  +---------------
0 | ! # ! & % * & @
1 | & ! @ & ! ! @ #
2 | ! @ $ $ * * % !
3 | & ! & & ! # # &
4 | # * @ $ & @ $ %
5 | % $ $ * & * @ $
6 | # # $ # @ $ % @
7 | # $ & # % $ @ #
```

```
잘못된 위치 입력입니다 다시 입력하세요
어느 위치에 있는 보석을 바꾸겠습니까? (행, 열) : 0 0
어느 위치에 있는 보석과 바꾸겠습니까? (행, 열) : 0 1

    0 1 2 3 4 5 6 7
  +---------------
0 | # ! ! & % * & @
1 | & ! @ & ! ! @ #
2 | ! @ $ $ * * % !
3 | & ! & & ! # # &
4 | # * @ $ & @ $ %
5 | % $ $ * & * @ $
6 | # # $ # @ $ % @
7 | # $ & # % $ @ #
```

조건에 맞는 입력은 받아들여 서로 바꾼다. 왼쪽이 과거인데 (0,0), (0,1)입력을 받아 해당 위치에 있는 보석이 바뀌어 오른쪽에 있는 모습으로 변했다.
3번 조건과 실제 이동은 게임 내 함수가 아닌 클래스의 멤버함수가 담당하므로 swapJewels가 의도대로 잘 작동함을 알 수 있다.

점수 등록
점수 등록 및 순위 확인은 game_score에서 담당한다.

게임이 끝나면 플레이어는 점수를 등록해야 하고, 등록된 점수는 이름과 함께
```
vector<std::pair<int, string>> score_board;
```
score_board에 저장된다.

아래는 game_score의 구현이다
```
void game_score(int score) {
    promft_check_score(score);
```
먼저 받은 점수를 알려준다.

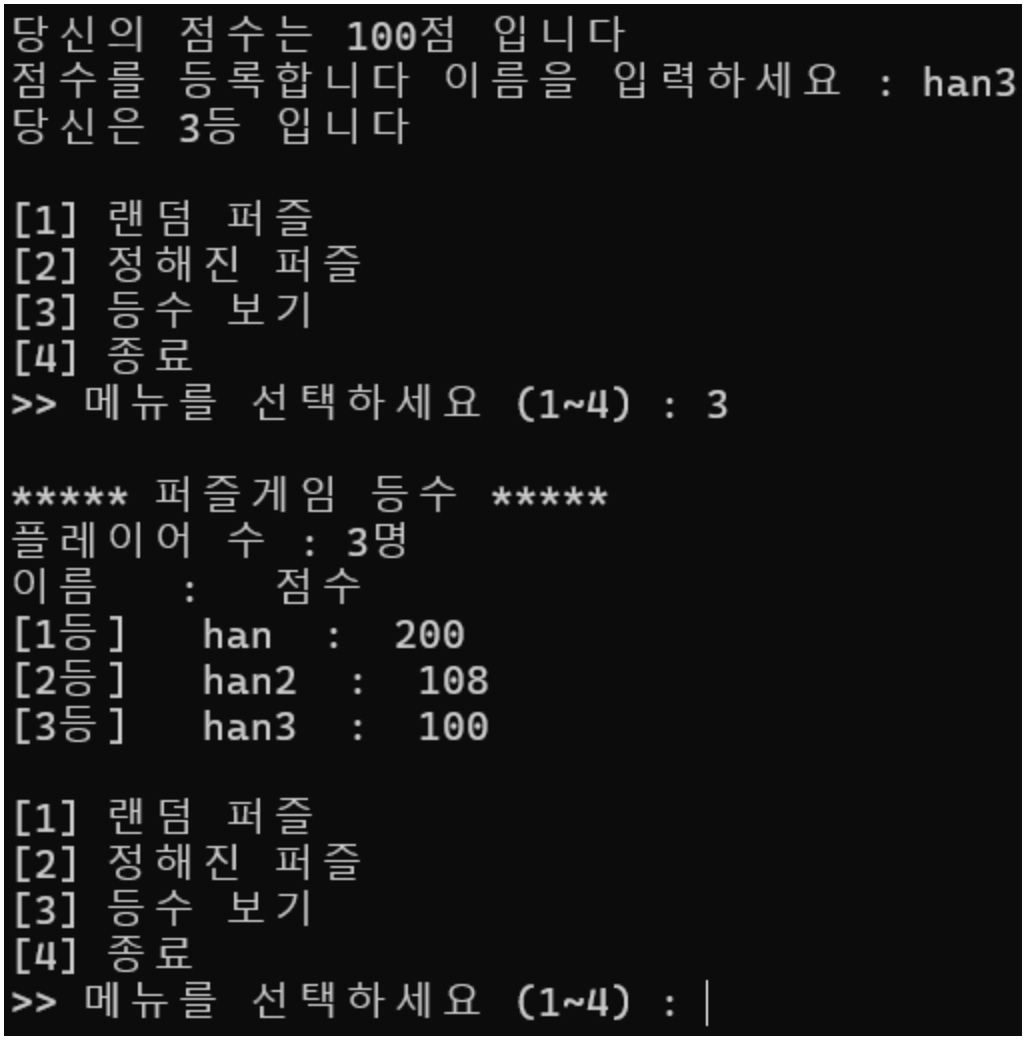
```
promft_ask_name();
string name = str_input();
```
이름을 물어본다

```
score_board.push_back(make_pair(score, name));
sort(score_board);
promft_check_rank(what_rank(make_pair(score, name), score_board));
```
score_board에 점수와 이름을 pair로 묶어 저장한다.
score_board를 정렬해 순위를 정한다
what_rank()함수를 이용해 플레이어가 몇등인지 알려준다

```
int what_rank(const std::pair<int, string>& x, const vector<std::pair<int, string>>& s)
    for (int i = 0; i < s.size(); i += 1) {
        if (x.first == s[i].first && x.second == s[i].second) {
            return s.size() - i;
```
what_rank는 배열을 순서대로 돌며 찾고자하는 플레이어가 몇등인지
알려준다 sort()는 오름차순이므로 배열 길이에서 순서를 뺀 값을 반환한다.

등수 보기
메뉴에서 3번을 누르면 지금까지의 점수기록을 볼 수 있다. 아래는 구현과
실행사진이다.
```
void print_rank() {
    cout << "\n***** 퍼즐게임 등수 *****\n";
    cout << "플레이어 수 : " << score_board.size() << "명" << '\n';
    cout << "이름    :    점수" << '\n';
    for (int a = score_board.size() - 1; a >= 0; a -= 1) {
        cout << "[" << score_board.size() - a << "등]    ";
        cout << score_board[a].second << "  :  " << score_board[a].first << '\n';
    }
}
```
score_board안의 원소를 역순으로 출력한다.

점수 등록 / 등수 보기 실행사진



점수를 알려주고, 이름을 등록하고, 등수를 알려준 뒤 메뉴로 돌아간다
등수 보기를 누를 때 등수가 출력된다.

입력 함수

앞에서 사용자에게 입력을 받을 때 input, loc_input, str_input을 사용했다
아래는 그 구현이다

```
int input() {
    int result;
```

반환할 지역 변수를 만든다

```
while (!(cin >> result)) {
    promft_input_error();
    cin.clear();
    cin.ignore(INT_MAX,'\n');
}
```

정수 입력이 잘 들어올때까지 받는다 cin.good()이 아니면 cin을 초기화하고 입력을 모두 무시한다.

```
cin.clear();
cin.ignore(INT_MAX, '\n');
return result;
```

입력이 잘 들어갔으면 다음 입력을 모조리 없애 뒷 입력에 방해가 되지 않도록 한 뒤, 반환한다.

loc_input 구현, str_input구현이다

```
std::pair<int, int> loc_input() {
    int a, b;
    while (!(cin >> a >>b)) {
        promft_input_error();
        cin.clear();
        cin.ignore(INT_MAX, '\n');
    }
    cin.clear();
    cin.ignore(INT_MAX, '\n');
    return make_pair(a, b);
}
```

```
string str_input() {
    string result;
    while (!(cin >> result)) {
        promft_input_error();
        cin.clear();
        cin.ignore(INT_MAX, '\n');
    }
    cin.clear();
    cin.ignore(INT_MAX, '\n');
    return result;
}
```

input()과 작동방식은 같다. 다만 반환형이 pair, string일 뿐이다

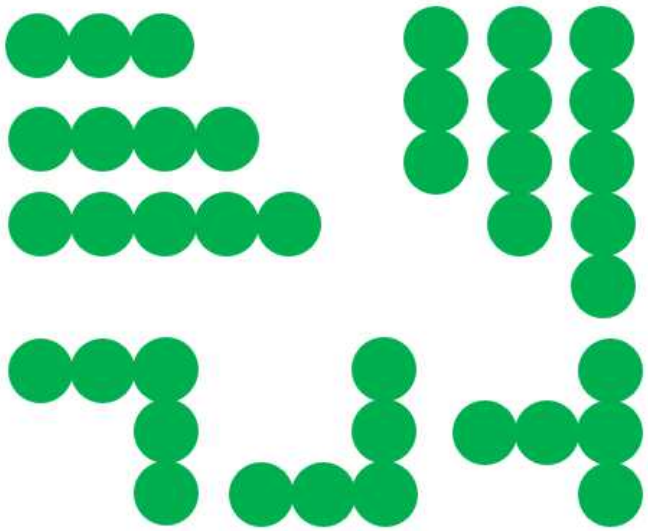
실행 사진

```
>> 메뉴를 선택하세요 (1~4) : sdfjkl
잘못된 입력입니다. 다시 입력하세요
입력 : r3
잘못된 입력입니다. 다시 입력하세요
```

정수 자리에 문자열이 들어왔을 때 잘 대처한다
올바른 범위인지는 각각의 함수에서 한다.

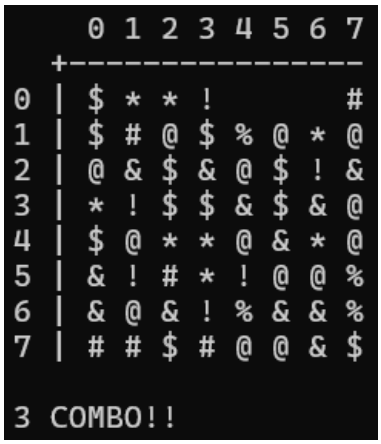
가장 어려웠던 것

체인을 감지하고 처리하는 방법을 생각하는 것이 가장 힘들었다.
게임이 잘 작동하려면 다양한 형태의 체인을 동시에 처리해야한다.
체인의 조건은 가로 혹은 세로로 3개 이상이므로



그림과 같이 다양한 형태의 체인이 있을 수 있다
처음에는 공식이 있지 않을까 했지만 되지 않았고
단순히 3개 이상이면 체인이니까 3개씩 한 번에 읽으면 되지 않을까 해서 update_A를 만들었다. 엄밀히 update_A는 ㄱ, ㄴ, ㅓ 형태의 체인을 한 체인이 아닌 두 개의 체인으로 인식하지만, 그것이 게임 진행에 특별한 영향을 미치지 못하므로 신경쓰지 않았다.
또한 동시에 처리하기 위해 체인을 만날 때마다 처리하는 것이 아닌 chain_arr에 담아두고 한 번에 처리하도록 했다.

개선할 점



처리 중에는 콤보가 뜨지 않았으면 좋을 것 같다.