보고서

주제 : Bejeweled

과 목 명	:	고급프로그래밍
제출일자	:	2021년 5월 21일
학 과	:	소프트웨어학부
학 번	:	2020203002
이 름	:	박 상 천

KwangWoon University

목 차

Ⅰ. 코드의 흐름

먼저 8X8의 퍼즐판을 만들고, 게임 옵션을 선택할 때 입력을 받는 값들을 저장할 option1,2를 선언하였다. 그리고 또 Jewel을 바꿀 때 입력받을 좌표를 p1, p2로 선언하였다.

```
Puzzle puzzle(8, 8);
int option1 = 0;
int option2 = 0;
pair<int, int> p1;
pair<int, int> p2;
int col = 0;
```

먼저 if 문을 사용하여 각 option에 값에 따라 기능을 하도록 하였다. [1] random puzzle을 선택하였을 때 random 함수를 이용하여 퍼즐에 무작위 Jewel을 넣고, chain이 있으면 update 함수를 이용하여 chain이 없어질 때까지 진행시켰다. 그 후에 swapJewels 함수를 좌표를 받아 그 좌표들의 Jewel을 서로 바꾸어 주었다. 그리고 swapJewels 함수 안에는 update 함수가 들어가 있어 chain이 없어질 때까지 진행시켰다.

```
cout << "<<< BEJEWELED >>>"<<endl<<endl;
cout << "[1] Start a new random puzzle" << endl;
cout << "[2] Start a pre-defined puzzle" << endl;
cout << "[3] Exit" << endl << endl;
cout << "> Choose a menu option (1~3): "; cin >> option1;
if (option1 == 1)
{
    cout << endl;
    puzzle.randomize();
    cout << endl;
    do
    {

    } while (puzzle.update());
    do
    {
        cout << "Input the first swap position (row, col): "; cin >> p1.second >> p1.first;
        cout << "Input the second swap position (row, col): "; cin >> p2.second >> p2.first;
    } while (puzzle.swapJewels(p1, p2));
}
```

다음으로 [2] pre-defined puzzle을 선택하였을 때 다시 if 문을 사용하여 0~3까지 퍼즐을 선택할 수 있도록 하였다. 여기서 0번째 퍼즐을 골랐을 때 이미 정의된 기호로 퍼즐을 채운다. 그리고 그 이후로는 update 함수 사용, 좌표를 받고 swapJewels 함수 사용은 바로 위의 내용과 일치한다. 그리고 [2] pre-defined puzzle에서는 퍼즐에 넣는 Jewel만 다르고 나머지는 다 같게 설계를 하였습니다.

```
else if(option1 == 2)
{
    cout << "> Choose a puzzle number (0~3): "; cin >> option2;
    if (option2 == 0)
    {
        cout << endl;
        puzzle.initialize("!#!&%*&@&!@&!!@#!@$$**%!&!&&!##&#*@$&@$%%$$*&*@$##$#@$%@#$&#%$@#");
        do
        {

        } while (puzzle.update());
        do
        {
            cout << "Input the first swap position (row, col): "; cin >> p1.second >> p1.first;
            cout << "Input the second swap position (row, col): "; cin >> p2.second >> p2.first;
        } while (puzzle.swapJewels(p1, p2));
    }
```

그리고 [2] pre-defined puzzle을 선택한 후 이상한 좌표를 넣었을 때 게임을 종료하도록 설계하였다.

```
else
{
    exit;
    cout << endl;
    Game();
}
```

마지막으로는 [3] Exit를 선택하였을 때 게임이 종료될 수 있도록 설계하였다.

```
else if (option1 == 3)
{
    exit;
}
```

그리고 나머지 오류들이 발생하였을 때는 메인화면으로 돌아갈 수 있도록 설계하였다.

```
else
{
    exit;
    cout << endl;
    Game();
}
```

Ⅱ. 설계 이론

```
Puzzle::Puzzle(int num_rows, int num_columns) // 맨처음 가로, 세로의 크기를 입력받아 그 길이만큼 벡터에 NONE을 넣는다
{                                             // 그리고 그것을 2차원 배열에 각각에 좌표랑 연결시킨다.
    this->num_rows = num_rows;
    this->num_columns = num_columns;

    int num_jewels = num_rows * num_columns;

    for (int i = 0; i < num_jewels; i++)
    {
        Jewel jewel = Jewel::NONE;
        jewel_arr.push_back(jewel);
    }
    int a = 0;
    for (int i = 0; i < num_rows; i++)
    {
        for (int j = 0; j < num_columns; j++)
        {
            grid[i][j].jewel = jewel_arr[a];
            a++;
        }
    }
}
```

Puzzle class이다. 입력받은 가로, 세로의 길이의 곱만큼 벡터에 NONE을 넣어주었다. 그리고 chain struct를 2차원 배열인 grid로 선언한 후, 거기에 방금 NONE이 들어간 벡터를 연결해서 2차원 배열의 jewel이 모두 NONE이 되게 하였다.

```
bool  Puzzle::check() // 체인이 남아 있으면 true, 없으면 false로 반환한다.
{
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
        {
            if (grid[i][j].jewel == grid[i][j - 1].jewel && grid[i][j].jewel == grid[i][j + 1].jewel
                || grid[i][j].jewel == grid[i - 1][j].jewel && grid[i][j].jewel == grid[i + 1][j].jewel)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
}
```

check 함수이다. 이것은 내가 따로 헤더 파일에 추가한 것이다. 여기에서는 grid.jewel을 모두 검사하여 만약 chain이 있으면 true로, 없으면 false로 반환하게 하였다.

```
bool Puzzle::full() // 빈칸이 있으면 true, 없으면 false로 반환한다.
{
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
        {
            if (grid[i][j].jewel == Jewel::NONE)
            {
                return true;
            }
            else
            {
                return true;
            }
        }
}
```

full 함수이다. 이것도 check 함수와 마찬가지로 헤더 파일에 따로 추가하여 만든 함수이다. 이 함수는 grid.jewel에서 하나라도 NONE이 들어가 있으면 true로, 없으면 false로 반환하게 하였다.

```
bool Puzzle::initialize(const std::string& jewel_list)
{
    int a = 0;
    int num_jewels = num_rows * num_columns;
    if ((int)jewel_list.length() != num_jewels) //길이가 맞지 않으면 false로 반환
    {
        return false;
    }
    else
    {
        for (int i = 0; i < num_rows; i++)
        {
            for (int j = 0; j < num_columns; j++)
            {
                grid[i][j].jewel = getJewelType(jewel_list[a]);
                a++;
            }
        }
        return true;
    }
}
```

initialize 함수이다. 먼저 미리 정의된 jewel list를 받는데, 그 길이가 퍼즐판과 다르면 false를 반환한다. 그리고 길이가 맞는다면 grid.jewel에 입력받은 jewel list를 좌표에 맞게 넣어주는 함수이다.

```
void Puzzle::randomize() // 2차원 배열에 각각 무작위의 Jewel을 넣는다.
{
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            grid[i][j].jewel = Jewel(rand() % 7);
        }
    }
}
```

randomize 함수이다. 이 함수가 호출이 되면 모든 grid.jewel에 무작위의 jewel을 넣어주는 함수이다.

```
void Puzzle::show() // 2차원 배열을 이용하여 각 좌표에 있는 Jewel을 퍼즐판과 함께 출력한다.
{
    cout << "   0 1 2 3 4 5 6 7" << endl;
    cout << "  +---------------" << endl;
    cout << "0 |";
    for (int j = 0; j < 8; j++)
    {
        cout << getJewelLetter(grid[0][j].jewel) << " ";
    }cout << endl;

    cout << "1 |";
    for (int j = 0; j < 8; j++)
    {
        cout << getJewelLetter(grid[1][j].jewel) << " ";
    }cout << endl;

    cout << "2 |";
    for (int j = 0; j < 8; j++)
    {
        cout << getJewelLetter(grid[2][j].jewel) << " ";
    }cout << endl;
```

show 함수이다. 이 함수는 헤더 파일에 새로 추가한 함수이다. 여기서는 인터페이스를 출력해 주는데 gird.jewel을 맨 좌상우하 순으로 출력을 하게 해준다. 여기서 gird.jewel에는 jewel 값이 들어가 있기 때문에 getJewelLetter을 이용하여 각 jewel에 맞는 char형으로 변환을 시켜서 출력을 해주는 함수이다. (아랫부분이 더 있지만 grid[?][j]에서 ? 부분만 숫자를 7까지 늘려서 추가한 것이고, 길이가 길어 중간에서 사진을 잘랐다)

```
bool Puzzle::swapJewels(std::pair<int, int> prev_loc, std::pair<int, int> next_loc) // 2개의 좌표를 입력받아 그 좌표에 들어있는 쥬얼을 서로 바꾼다
                                                                                    // 이때 update함수를 사용하여 chain을 체크한다.
{
    if(prev_loc.first >= 0 && prev_loc.first < num_rows && prev_loc.second >= 0 && prev_loc.second < num_columns&& next_loc.first >= 0
        && next_loc.first < num_rows && next_loc.second >= 0 && next_loc.second < num_columns)
        if (prev_loc.first + 1 == next_loc.first && prev_loc.second == next_loc.second
            || prev_loc.first - 1 == next_loc.first && prev_loc.second == next_loc.second
            || prev_loc.first == next_loc.first && prev_loc.second + 1 == next_loc.second
            || prev_loc.first == next_loc.first && prev_loc.second - 1 == next_loc.second)
        {
            cout << endl;
            Jewel jA, jB, temp;
            jA = getJewel(prev_loc);
            jB = getJewel(next_loc);
            temp = jA;
            jA = jB;
            jB = temp;
            setJewel(prev_loc, (Jewel)jA);
            setJewel(next_loc, (Jewel)jB);
            do
            {

            } while (update());
            cout << endl;
            return true;
        }
        else
        {
            exit;
            return false;
        }
    else
    {
        return false;
    }
}
```

swapJewels 함수이다. 입력을 받은 좌표가 서로 옆에 있거나, 위나 아래에 있을 때 그 입력받은 좌표들의 jewel을 서로 바꾸어 준다. 그리고 중간에 do while 문에 update 함수를 넣어 update 함수가 false가 될 때까지 계속해서 chain을 지워주게 하고, 그때 true로 반환을 하게 하였다. 이 이외의 상황에 대해서는 모두 false로 반환하게 하였다.

```
bool Puzzle::update() // chain이 있는지 검사하고, 있으면 그 chain위치의 jewel을 NONE으로 바꾼다.
{                     // 그후 NONE이 있는지 검사하고, 있으면 위의 Jewel들을 한칸씩 내려 빈칸을 채워주고, 맨 위칸은 무작위 Jewel로 채운다
    int a = 0;
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++)
        {
            if (grid[i][j].jewel == grid[i + 1][j].jewel)
                if (grid[i][j].jewel == grid[i - 1][j].jewel)
                {
                    grid[i - 1][j].jewel = Jewel::NONE;
                    grid[i][j].jewel = Jewel::NONE;
                    grid[i + 1][j].jewel = Jewel::NONE;
                    a = 1;
                }

            if (grid[i][j].jewel == grid[i][j + 1].jewel)
                if (grid[i][j].jewel == grid[i][j - 1].jewel)
                {
                    grid[i][j - 1].jewel = Jewel::NONE;
                    grid[i][j].jewel = Jewel::NONE;
                    grid[i][j + 1].jewel = Jewel::NONE;
                    a = 1;
                }
        }
    if (a == 1)
    {
        show();
    }
        cout << endl;
```

마지막으로 update 함수이다. update 함수는 길이가 길어 3개로 쪼개어 설명을 하겠다. 먼저 위 사진은 맨 처음 부분이다. 여기서는 chain이 있는지를 검사하고, 체인이 있다면 a=1을 넣어 인터페이스를 출력할 수 있게 하였다. 만약 다시 update에 진입하면 a가 다시 0으로

초기화가 되어 a에 대한 오류가 발생하지 않도록 하였다.

```
do
{
    for (int i = 0; i < 8; i++)
    {
        if (grid[7][i].jewel == Jewel::NONE)
        {
            while (grid[7][i].jewel == Jewel::NONE)
            {
                    grid[7][i].jewel = grid[6][i].jewel;
                    grid[6][i].jewel = grid[5][i].jewel;
                    grid[5][i].jewel = grid[4][i].jewel;
                    grid[4][i].jewel = grid[3][i].jewel;
                    grid[3][i].jewel = grid[2][i].jewel;
                    grid[2][i].jewel = grid[1][i].jewel;
                    grid[1][i].jewel = grid[0][i].jewel;
                    grid[0][i].jewel = Jewel(rand() % 7);
            }
        }
            if (grid[6][i].jewel == Jewel::NONE)
            {
                while (grid[6][i].jewel == Jewel::NONE)
                {
                    grid[6][i].jewel = grid[5][i].jewel;
                    grid[5][i].jewel = grid[4][i].jewel;
                    grid[4][i].jewel = grid[3][i].jewel;
                    grid[3][i].jewel = grid[2][i].jewel;
                    grid[2][i].jewel = grid[1][i].jewel;
                    grid[1][i].jewel = grid[0][i].jewel;
                    grid[0][i].jewel = Jewel(rand() % 7);
                }
            }
            if (grid[5][i].jewel == Jewel::NONE)
            {
                while (grid[5][i].jewel == Jewel::NONE)
                {
                    grid[5][i].jewel = grid[4][i].jewel;
                    grid[4][i].jewel = grid[3][i].jewel;
                    grid[3][i].jewel = grid[2][i].jewel;
                    grid[2][i].jewel = grid[1][i].jewel;
                    grid[1][i].jewel = grid[0][i].jewel;
                    grid[0][i].jewel = Jewel(rand() % 7);
                }
            }
            if (grid[4][i].jewel == Jewel::NONE)
            {
                while (grid[4][i].jewel == Jewel::NONE)
                {
                    grid[4][i].jewel = grid[3][i].jewel;
                    grid[3][i].jewel = grid[2][i].jewel;
                    grid[2][i].jewel = grid[1][i].jewel;
                    grid[1][i].jewel = grid[0][i].jewel;
                    grid[0][i].jewel = Jewel(rand() % 7);
                }
            }
```

```
                if (grid[3][i].jewel == Jewel::NONE)
                {
                    while (grid[3][i].jewel == Jewel::NONE)
                    {
                        grid[3][i].jewel = grid[2][i].jewel;
                        grid[2][i].jewel = grid[1][i].jewel;
                        grid[1][i].jewel = grid[0][i].jewel;
                        grid[0][i].jewel = Jewel(rand() % 7);
                    }
                }
                if (grid[2][i].jewel == Jewel::NONE)
                {
                    while (grid[2][i].jewel == Jewel::NONE)
                    {
                        grid[2][i].jewel = grid[1][i].jewel;
                        grid[1][i].jewel = grid[0][i].jewel;
                        grid[0][i].jewel = Jewel(rand() % 7);
                    }
                }
                if (grid[1][i].jewel == Jewel::NONE)
                {
                    while (grid[1][i].jewel == Jewel::NONE)
                    {
                        grid[1][i].jewel = grid[0][i].jewel;
                        grid[0][i].jewel = Jewel(rand() % 7);
                    }
                }
                if (grid[0][i].jewel == Jewel::NONE)
                {
                    while (grid[0][i].jewel == Jewel::NONE)
                    {
                        grid[0][i].jewel = Jewel(rand() % 7);
                    }
                }
            }
        } while (!full());
        show();
        cout << endl;

        if (check())
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

위 2개의 캡처가 update 함수의 아랫부분이다. update의 맨 윗부분을 실행을 한 뒤에 실행이 된다. 만약 위에 chain을 NONE으로 만들어 grid.jewel에 NONE이 있다면 그 위치를 기준으로 그 NONE 위로부터의 y축 라인에 모든 jewel들을 한 칸씩 내린다. 그 후에 맨 위에는 random 함수를 이용하여 쥬얼을 채워주었다. 이 과정을 모든 jewel들이 NONE이 아닌 어떤 jewel들로 채워질 때까지 반복하였는데, 이 반복을 do while 문에 full 함수를 넣어 false로 반환될 때까지 반복하였다. 그 후 인터페이스를 출력하였다. 그리고 모든 과정이 마치고 또 grid에 chain이 있을 수 있으니 check 함수를 이용하여 chain이 있으면 ture, 없으면 false로 반환하게 하였다.

Ⅲ. 느낀점

처음 이 과제가 주어졌을 때 참 막막했다. 지금까지 코딩들은 한 파일 안에서 모든 코딩을 끝냈기 때문이다. 이번 Bejeweled는 여러 헤더 파일들이 존재하였고, 그 헤더 파일에 대해서 각각 코딩을 해야 했다. 마지막으로는 그것들을 하나로 연결해서 사용해야 했기 때문에 어떤 식으로 함수를 불러오는지 몰라서 초반에는 다소 어려움을 느꼈다. 하지만 고생을 한 만큼 배운 것도 많은 것 같았다. Class를 다른 함수에서 사용하는 방법, pair 함수의 사용법 등과 같이 Bejeweled를 코딩하며 지금까지 이론으로 배웠지만, 실제 적용법을 모르는 것이나, 처음 접해보는 이론 등을 하나씩 찾아보며 코딩을 해나갔습니다. 그 과정 속에서 저는 실전 코딩 실력을 기를 수 있었다.

다음은 오류에 대한 내용이다. 모든 기능들은 구현을 하였다. 하지만 그것을 합치는 과정에서 오류를 2개 가지 발견을 하였고, 아무리 고민을 하여도 방법을 몰라서 고칠 수 없었다.
먼저 첫 번째, 퍼즐을 고르고 좌표를 각각 2개 받을 때 알맞지 않은 값을 받았을 때 메인화면으로 돌아가는 것이다. 처음 이 부분을 좌표값에서 벗어났을 때 다시 Game 함수를 불러 진행을 시켰지만, 나중에 게임이 종료가 되어야 할 때 다시 게임이 진행이 되는 버그가 있어 아예 게임을 종료시키게 하였다. 두 번째, 대부분의 퍼즐은 그렇지 않지만 가끔씩(정확히 언제인지 모름) chain이 있지만 지우지 않고 좌표값을 받으러 넘어가는 때가 있다. 여기서 좌표값을 넣으면 새로 생성되는 chain이 있으면 새로 생성된 chain과 함께 지워진다. 또 새로 생성된 chain이 없더라도 기존에 있던 chain이 지워진다.

이렇게 Bejeweled를 설계하며 여러 오류들을 접하였고, 그 오류들에 대해 고민하고, 찾아보며 프로그래머로서 한걸음 내디딜 수 있는 계기가 되었다.