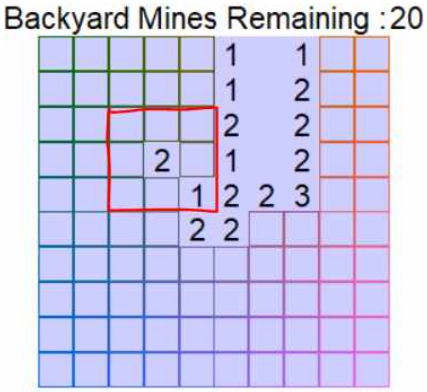
기말 프로젝트 - MineSweeper

2019102212 이정호

기말 프로젝트로 MineSweeper, 지뢰찾기 게임을 만들기로 했습니다. 기본적으로 이 게임은 실습1로 만든 벽돌 깨기 게임의 확장이 아닌, 처음부터 직접 짠 게임입니다. 본격적으로 시작하기에 앞서 지뢰 찾기 게임에 대한 기본적인 소개를 하겠습니다.

지뢰 찾기 '게임'이라고 했으니 알아야 할 사항은 간단합니다. 바로 '룰'과 '승리 조건'입니다. 기본적인 지뢰 찾기 게임의 화면을 보게 되면 다음과 같은 바둑판 모양의 그림을 볼 수 있습니다.



다음 그림을 보면 빨간색 네모가 쳐진 9개의 칸의 정가운데에 숫자 '2'가 위치하고 있는 것을 알 수 있습니다. 이는 저 숫자 '2'를 둘러싸고 있는 8개의 칸에 지뢰는 총 2개가 있다는 의미입니다. 8개의 칸 중 숫자 '1'이 써져 있는 칸이 있는데, 저기에는 당연히 지뢰가 없습니다.

다음으로는 승리 조건입니다. 보통 지뢰를 모두 찾아내는 것을 승리 조건으로 알고 있는데, 지뢰 찾기 게임의 승리 조건은 '지뢰가 없는 칸을 모두 여는 것'입니다.

게임 화면의 바둑판 모양의 칸은 좌클릭을 하면 열리게 됩니다. 칸이 열리는 데에도 규칙이 있습니다. 만일 클릭한 칸이 지뢰가 있는 칸이라면, 게임은 그대로 패배로 끝나게 됩니다. 숫자가 적힌 칸이라면, 그 칸만 열리고 끝납니다. 운이 좋게도 주위 8개의 칸에 지뢰가 하나도 없는, '빈칸'이라면 8방향으로 숫자가 적힌 칸과 만날 때까지 모든 칸을 열게 됩니다.

이제 기본적인 게임의 '룰'과 '승리 조건에 대해서 알게 되었습니다. 이제 구현만 하면 될거 같습니다.

본격적으로 지뢰 찾기 게임을 구현해보도록 하겠습니다.

```
int input = 5;

std::cout << "Input Your Name : ";
std::cin >> stt;

fin.open("LeaderBoard.txt");
if (!fin) {
    fout.open("LeaderBoard.txt");
    fout << std::string("AAA₩n999₩nAAA₩n999₩nAAA₩n999₩nAAA₩n999₩nAAA₩n999₩nAAA₩n999₩nAAA₩n999₩nAAA₩n999₩nAAA₩n99
    fout.close();
}
fin.close();

while (input) {

    time_ = true;
    std::cout << "|-------------------------|₩n";
    std::cout << "|* M I N E S W E E P E R *|₩n";
    std::cout << "|-------------------------|₩n";
    std::cout << "|    Select Difficulty    |₩n";
    std::cout << "|                         |₩n";
    std::cout << "|    1. BackYard          |₩n";
    std::cout << "|    2. WarZone           |₩n";
    std::cout << "|    3. DMZ               |₩n";
    std::cout << "|    4. LeaderBoard       |₩n";
    std::cout << "|-------------------------|₩n";
    std::cout << "|    0. Quit              |₩n₩n>> ";

    std::cin >> input;

    if (input == 1) {
        difficulty = "BackYard";

        CGameLayout* pGameLayout = new CGameLayout(300, 300);

        KhuGleWinInit(pGameLayout);
    }
    else if (input == 2) {
        difficulty = "WarZone";

        CGameLayout* pGameLayout = new CGameLayout(425, 425);

        KhuGleWinInit(pGameLayout);
    }
    else if (input == 3) {
        difficulty = "DMZ";

        CGameLayout* pGameLayout = new CGameLayout(800, 800);

        KhuGleWinInit(pGameLayout);
    }
    else if (input == 4) {
```

```
            std::cout << "₩nLeaderBoard₩n₩nBackYard₩n₩n";

            fin.open("LeaderBoard.txt");
            for (int i = 0; i < 3; i++) {
                std::cout << i + 1 << ". ";
                getline(fin, difficulty);
                std::cout << difficulty << ' ';
                getline(fin, difficulty);
                std::cout << difficulty << '₩n';
            }

            std::cout << "₩nWarZone₩n₩n";

            for (int i = 0; i < 3; i++) {
                std::cout << i + 1 << ". ";
                getline(fin, difficulty);
                std::cout << difficulty << ' ';
                getline(fin, difficulty);
                std::cout << difficulty << '₩n';
            }

            std::cout << "₩nDMZ₩n₩n";

            for (int i = 0; i < 3; i++) {
                std::cout << i + 1 << ". ";
                getline(fin, difficulty);
                std::cout << difficulty << ' ';
                getline(fin, difficulty);
                std::cout << difficulty << '₩n';
            }

            std::cout << '₩n';
            fin.close();
        }
        else if (input == 0) {
            std::cout << "";
        }
        else {
            std::cout << "Wrong Command!₩n";
        }
    }
```

가장 먼저 게임 시작 준비 화면입니다. 이는 콘솔창에 띄워지게 되며, 처음에 플레이어가 최고 기록을 갱신했을 때 리더보드를 업데이트하기 위해 플레이어의 이름을 입력받습니다. 다음으로 플레이어로 하여금 난이도를 선택할 수 있도록 합니다. 1단계부터 3단계까지의 난이도를 구현해놓았으며 플레이어가 0을 입력하여 프로그램을 종료시키기 전까지 계속해서 리플레이할 수 있도록 구현했습니다. 4를 입력하면 난이도별로 리더보드를 보여줍니다. 이 프로그램을 처음 실행한 상황의 경우 리더보드의 정보를 가지고 있지 않으므로 리더보드의 정보를 저장할 txt 파일을 새로 생성하여 가상의 최고 기록 정보를 넣어둡니다.

```
if (difficulty == "Backyard") size = 10;
else if (difficulty == "Warzone") size = 15;
else size = 30;

mine = 0;

trap = false;
win = false;

endrow = endcol = 0;

for (int row = 0; row < size; row++) {
    std::vector<CKhuGleSprite*> m_pRect;
    for (int col = 0; col < size; col++) {
        m_pRect.push_back(new CKhuGleSprite(GP_STYPE_RECT, GP_CTYPE_KINEMATIC, CKgLine(CKgPoint(25 * row, 25 * col),
            CKgPoint(25 * (row + 1) - 1, 25 * (col + 1) - 1)), KG_COLOR_24_RGB(255 * row / size, 100, 255 * col / size), false, 30));

        m_pGameLayer->AddChild(m_pRect[col]);
    }
    m_pRects.push_back(m_pRect);
}
```

이제 입력 받은 난이도에 따라서 게임 화면의 크기를 정합니다. 쉬운 난이도에서는 10 * 10 의 크기의 게임 화면을 설정하게 되고, 그 크기 만큼의 칸을 2중 반복문을 이용하여 vector 로 할당해줍니다. 색상 설정을 저런 식으로 하게 되면 단조로운 색상 대신 그라데이션을 구현할 수 있습니다.

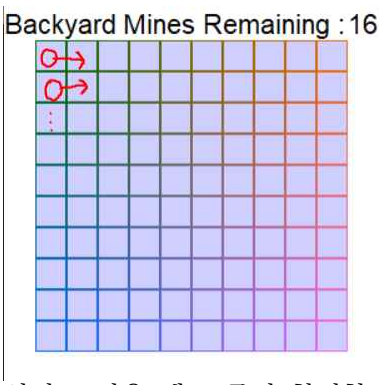
```
int mine_limit = size * size / 5;

for (int row = 0; row < size; row++) {
    std::vector<bool> m_Mine;
    for (int col = 0; col < size; col++) {
        if (mine_limit > mine) {
            if (rand() % 10 < 2) {
                m_Mine.push_back(true);
                mine++;
            }
            else m_Mine.push_back(false);
        }
        else m_Mine.push_back(false);
    }
    m_Mines.push_back(m_Mine);
}
```

다음으로 전체 칸 개수의 20%가 넘지 않도록 랜덤하게 지뢰가 있는 칸을 지정해줍니다. 게임을 할 때마다 지뢰가 있는 칸은 랜덤하게 지정되며, 지뢰의 개수도 랜덤하게 지정됩니다.

```
for (int row = 0; row < size; row++) {
    std::vector<int> m_Num;
    for (int col = 0; col < size; col++) {
        if (m_Mines[row][col]) m_Num.push_back(-1);
        else {
            int num = 0;

            //row == 0
            if (!row) {
                if (m_Mines[row + 1][col]) num++;
                if (!col) {
                    if (m_Mines[row][col + 1]) num++;
                    if (m_Mines[row + 1][col + 1]) num++;
                }
                else if (col == size - 1) {
                    if (m_Mines[row][col - 1]) num++;
                    if (m_Mines[row + 1][col - 1]) num++;
                }
                else {
                    if (m_Mines[row][col + 1]) num++;
                    if (m_Mines[row][col - 1]) num++;
                    if (m_Mines[row + 1][col + 1]) num++;
                    if (m_Mines[row + 1][col - 1]) num++;
                }
            }
            else if (row == size - 1) {
                if (m_Mines[row - 1][col]) num++;
                if (!col) {
                    if (m_Mines[row][col + 1]) num++;
                    if (m_Mines[row - 1][col + 1]) num++;
                }
                else if (col == size - 1) {
                    if (m_Mines[row][col - 1]) num++;
                    if (m_Mines[row - 1][col - 1]) num++;
                }
                else {
                    if (m_Mines[row][col + 1]) num++;
                    if (m_Mines[row][col - 1]) num++;
                    if (m_Mines[row - 1][col + 1]) num++;
                    if (m_Mines[row - 1][col - 1]) num++;
                }
            }
            else {
                if (m_Mines[row + 1][col]) num++;
                if (m_Mines[row - 1][col]) num++;
                if (!col) {
                    if (m_Mines[row - 1][col + 1]) num++;
                    if (m_Mines[row][col + 1]) num++;
                    if (m_Mines[row + 1][col + 1]) num++;
                }
                else if (col == size - 1) {
                    if (m_Mines[row - 1][col - 1]) num++;
                    if (m_Mines[row][col - 1]) num++;
                    if (m_Mines[row + 1][col - 1]) num++;
                }
                else {
                    if (m_Mines[row - 1][col + 1]) num++;
                    if (m_Mines[row - 1][col - 1]) num++;
```

```
                if (m_Mines[row][col + 1]) num++;
                if (m_Mines[row][col - 1]) num++;
                if (m_Mines[row + 1][col + 1]) num++;
                if (m_Mines[row + 1][col - 1]) num++;
            }
        }
        m_Num.push_back(num);
    }
    m_Nums.push_back(m_Num);
}
```

다음으로 숫자가 있는 칸을 지정하는 반복문입니다. 위에서 설명했듯이 칸에는 3가지 종류가 있습니다. 지뢰가 있는 칸, 숫자가 있는 칸, 빈칸입니다. 가장 먼저 지뢰가 있는 칸은 어떠한 연산도 필요하지 않으므로 -1을 할당해줍니다. 다음으로 확인할 것은 숫자가 있는 칸입니다. 숫자가 있는 칸은 기본적으로 0에서 출발합니다. 당연하게도 주위에 칸들 중 지뢰가 있는 칸이 있는지 확인하며 만약 있다면 숫자를 1씩 늘리면 됩니다. 여기서 중요한 점은 모서리에 있는 칸들입니다. 모서리에 있는 칸들도 그냥 연산을 한다면, 예를 들어 [0][0] 칸의 연산을 하기 위해 그 위 칸인 [0][-1]을 액세스하려고 한다면, 당연히 vector out of range error가 발생하게 될 것입니다. 따라서, 칸들을 위치에 따라서 나눌 필요가 있었습니다.



위의 그림을 예로 들어 확인할 수 있듯이 왼쪽이 없는 칸은 그 칸이 위나 아래도 없는 칸인지와 관계 없이, 오른쪽에는 반대쪽에는 반드시 칸이 있습니다. 오른쪽이 없는 칸도 마찬가지로 왼쪽 칸이 반드시 있습니다. 그렇다면, row == 0일 때는 무조건 row + 1인 칸의 지뢰 여부를 확인해야 합니다. 마찬가지로 row == size - 1일 때는 무조건 row - 1인 칸의 지뢰 여부를 확인해야 합니다. row에 대한 상황을 나누고 나서는 col에 대한 상황도 동일하게 나눠 수행하면 됩니다.

```
int open_num = 0;
int flag = 0;
for (int row = 0; row < size; row++)
    for (int col = 0; col < size; col++) {
        if (m_Opens[row][col]) open_num++;
        if (m_Flags[row][col]) flag++;
    }

win = size * size - mine == open_num;
```

다음은 본격적인 구현부입니다. open_num은 승리 조건을 판별하기 위한 현재 열려있는 칸의 수, flag는 플레이어가 지뢰라고 판단하여 깃발을 꽂아놓은 칸의 수입니다. flag는 현재 남은 지뢰의 수를 표시하기 위해 사용됩니다.

```
if (m_bMousePressed[0]) {
    int row = m_MousePosX / 25 - 1;
    int col = m_MousePosY / 25 - 1;

    if (row > -1 && row < size && col > -1 && col < size) {
        Open(row, col);
        if (m_Mines[row][col]) {
            trap = true;
            endrow = row;
            endcol = col;
        }
    }

    m_bMousePressed[0] = false;
}

if (m_bMousePressed[2]) {
    int row = m_MousePosX / 25 - 1;
    int col = m_MousePosY / 25 - 1;

    if (row > -1 && row < size && col > -1 && col < size && !m_Opens[row][col]) m_Flags[row][col] = !m_Flags[row][col];

    m_bMousePressed[2] = false;
}
```

플레이어와의 상호작용 부분입니다. 플레이어가 마우스 왼쪽 버튼을 누르게 되면 먼저 플레이어의 클릭 위치를 칸 vector의 index로 변환하게 됩니다. vector out of range error가 발생하는 것을 방지하기 위해 조건문을 걸고, 칸의 모든 변화는 단순히 Open(row, col)함수 하나로 해결하게 됩니다. 만일 클릭한 위치가 지뢰였다면, 게임 패배를 뜻하는 trap 변수를 true로 바꾸고, 게임 종료 화면에서 클릭한 지뢰의 위치를 표시하기 위해 index 값을 저장합니다.

플레이어가 마우스 오른쪽 버튼을 누르게 되면 플레이어가 클릭한 위치를 깃발 칸으로 변경합니다.

```
void CGameLayout::Open(int row, int col) {
    if (m_Opens[row][col]) return;
    else if (m_Mines[row][col]) return;
    else {
        if (m_Nums[row][col] > 0) {
            m_Opens[row][col] = true;
            return;
        }
        else {
            m_Opens[row][col] = true;
            if (!row) {
                Open(row + 1, col);
                if (!col) {
                    Open(row, col + 1);
                    Open(row + 1, col + 1);
                }
                else if (col == size - 1) {
                    Open(row, col - 1);
                    Open(row + 1, col - 1);
                }
                else {
                    Open(row, col + 1);
                    Open(row, col - 1);
                    Open(row + 1, col + 1);
                    Open(row + 1, col - 1);
                }
            }
            else if (row == size - 1) {
                Open(row - 1, col);
```

```
            if (!col) {
                Open(row, col + 1);
                Open(row - 1, col + 1);
            }
            else if (col == size - 1) {
                Open(row, col - 1);
                Open(row - 1, col - 1);
            }
            else {
                Open(row, col + 1);
                Open(row, col - 1);
                Open(row - 1, col + 1);
                Open(row - 1, col - 1);
            }
        }
        else {
            Open(row + 1, col);
            Open(row - 1, col);
            if (!col) {
                Open(row - 1, col + 1);
                Open(row, col + 1);
                Open(row + 1, col + 1);
            }
            else if (col == size - 1) {
                Open(row - 1, col - 1);
                Open(row, col - 1);
                Open(row + 1, col - 1);
            }
            else {
                Open(row - 1, col + 1);
                Open(row - 1, col - 1);
                Open(row, col + 1);
                Open(row, col - 1);
                Open(row + 1, col + 1);
                Open(row + 1, col - 1);
            }
        }
    }
}
```

다음은 Open(row, col)함수입니다. 가장 처음에 설명했던 방법처럼 칸이 열리는 과정은 연쇄적입니다. 이 점을 착안하여 재귀적인 방법으로 함수를 구현하게 되었습니다. 가장 먼저 플레이어가 클릭한 칸이 열린 칸인지 판단합니다. 클릭한 칸이 만약 이미 열려있는 칸이거나 지뢰가 있는 칸이라면, 그 어떠한 추가 행동도 필요 없기 때문에 return하게 됩니다. 만약 숫자가 있는 칸이라면, 단순히 클릭한 칸만 열고 return합니다. 만약 빈칸이라면, 주위 8칸을 모두 Open하게 됩니다. 위에서 칸의 숫자를 구했던 방식과 동일하게 row와 col이 0이거나 size - 1 인 모든 상황을 고려하여 재귀적으로 수행하게 됩니다. 그렇게 칸들은 연쇄적으로 열리다가 숫자가 있는 칸을 만나게 되면 멈추게 됩니다.

```
m_pScene->Render();
DrawSceneTextPos(difficulty.c_str(), CKgPoint(0, 0), KG_COLOR_24_RGB(0, 0, 0));
DrawSceneTextPos("Mines Remaining :", CKgPoint(95, 0), KG_COLOR_24_RGB(0, 0, 0));
DrawSceneTextPos(std::to_string(mine - flag).c_str(), CKgPoint(275, 0), KG_COLOR_24_RGB(0, 0, 0));

for (int row = 0; row < size; row++)
    for (int col = 0; col < size; col++) {
        if (m_Opens[row][col]) {
            if (m_Nums[row][col]) DrawSceneTextPos(std::to_string(m_Nums[row][col]).c_str(), CKgPoint(25 * (row + 1) + 7, 25 * (col + 1) + 1),
                KG_COLOR_24_RGB(0, 0, 0));
            m_pRects[row][col]->ChangeColor(KG_COLOR_24_RGB(200, 200, 255), true);
        }
        if (m_Flags[row][col]) DrawSceneTextPos("*", CKgPoint(25 * (row + 1) + 8, 25 * (col + 1) + 5), KG_COLOR_24_RGB(255, 0, 0));
    }
```

다음으로 화면 표시입니다. 가장 먼저 DrawSceneTextPos 함수에 color 파라미터를 추가하여 작성되는 텍스트의 색상을 정할 수 있도록 했습니다. 따라서 일반적인 텍스트와 숫자는 검정색으로, 깃발 표시는 빨간색으로 될 수 있도록 설정했습니다. 또한 모든 칸에 대해서 만일 그 칸이 열려있는 칸이라면 칸의 색상이 배경색과 똑같게 바꾸고 fill을 true로 하여 열린 듯한 느낌이 나도록 ChangeColor 함수를 이용했습니다. 또한 숫자가 있는 칸이라면 숫자를 표시하고, 빈칸이라면 조잡하게 0이 도배되는 것이 아닌, 그냥 비어있도록 구현했습니다.

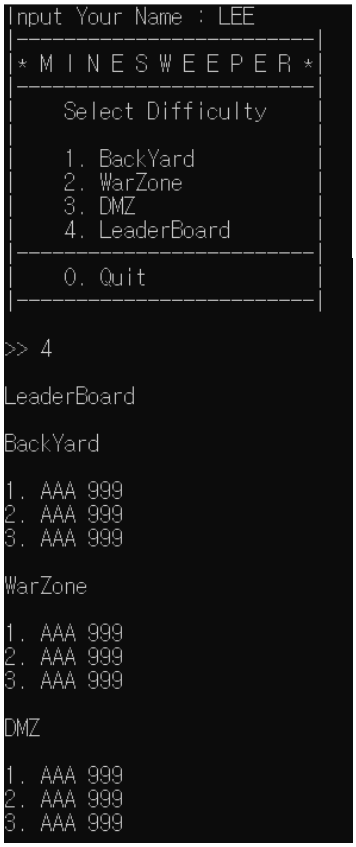
```
if (trap) {
    m_pScene->Render();
    DrawSceneTextPos("X", CKgPoint(25 * (endrow + 1) + 5, 25 * (endcol + 1) + 1), KG_COLOR_24_RGB(255, 0, 0));
    DrawSceneTextPos("You Lose! Time :", CKgPoint(0, 0), KG_COLOR_24_RGB(0, 0, 0));
    DrawSceneTextPos(std::to_string((int)(finish - start)).c_str(), CKgPoint(165, 0), KG_COLOR_24_RGB(0, 0, 0));

    for (int row = 0; row < size; row++)
        for (int col = 0; col < size; col++) {
            if (m_Opens[row][col]) {
                if (m_Nums[row][col]) DrawSceneTextPos(std::to_string(m_Nums[row][col]).c_str(), CKgPoint(25 * (row + 1) + 7, 25 * (col + 1) + 1),
                    KG_COLOR_24_RGB(0, 0, 0));
                m_pRects[row][col]->ChangeColor(KG_COLOR_24_RGB(200, 200, 255), true);
            }
            if (m_Flags[row][col]) DrawSceneTextPos("*", CKgPoint(25 * (row + 1) + 8, 25 * (col + 1) + 5), KG_COLOR_24_RGB(255, 0, 0));
        }
}
```

패배 게임 종료 화면입니다. 가장 먼저 클릭한 지뢰의 위치를 빨간색 X표로 표시하여 플레이어가 어떤 지뢰가 있는 칸을 클릭하여 패배했는지 확인할 수 있도록 합니다. 또한 게임이 시작된 뒤 흐른 시간을 표시해줍니다.

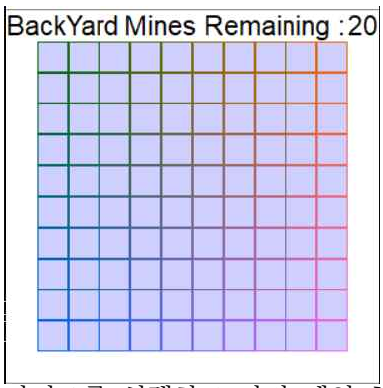
```
else if (win) {
    if (time_) {
        finish = time(NULL);
        time_ = false;
        fin.open("LeaderBoard.txt");
        int t = 0;
        if (size == 10) t = 0;
        else if (size == 15) t = 3;
        else if (size == 30) t = 6;

        std::vector<int> num;
        std::vector<std::string> str;

        for (int k = 0; k < 9; k++) {
            getline(fin, difficulty);
            str.push_back(difficulty);
            getline(fin, difficulty);
            num.push_back(stoi(difficulty));
        }
        fin.close();

        if (num[t] > (int)(finish - start)) {
            num[t + 2] = num[t + 1];
            num[t + 1] = num[t];
            num[t] = (int)(finish - start);

            str[t + 2] = str[t + 1];
            str[t + 1] = str[t];
            str[t] = stt;
```

```
        else if (num[t + 1] > (int)(finish - start)) {
            num[t + 2] = num[t + 1];
            num[t + 1] = (int)(finish - start);

            str[t + 2] = str[t + 1];
            str[t + 1] = stt;
        }
        else if (num[t + 2] > (int)(finish - start)) {
            num[t + 2] = (int)(finish - start);

            str[t + 2] = stt;
        }

        fout.open("LeaderBoard.txt");

        for (int k = 0; k < 9; k++) {
            fout << str[k] << '₩n';
            fout << std::to_string(num[k]);
            if (k != 8) fout << '₩n';
        }

        fout.close();
    }
    m_pScene->Render();
    DrawSceneTextPos("You Win! Congrats! Time :", CKgPoint(0, 0), KG_COLOR_24_RGB(0, 0, 0));
    DrawSceneTextPos(std::to_string((int)(finish - start)).c_str(), CKgPoint(257, 0), KG_COLOR_24_RGB(0

    for (int row = 0; row < size; row++)
        for (int col = 0; col < size; col++) {
            if (m_Opens[row][col]) {
                if (m_Nums[row][col]) DrawSceneTextPos(std::to_string(m_Nums[row][col]).c_str(), CKgPoi
                    KG_COLOR_24_RGB(0, 0, 0));
                m_pRects[row][col]->ChangeColor(KG_COLOR_24_RGB(200, 200, 255), true);
            }
            if (m_Flags[row][col]) DrawSceneTextPos("*", CKgPoint(25 * (row + 1) + 8, 25 * (col + 1) +
        }
}
```

마지막으로 승리했을 때입니다. 승리 조건은 가장 처음에 말했듯이 지뢰가 없는 칸을 모두 열었을 때 승리합니다. 승리 조건은 win = size * size - mine == open_num으로 전체 칸에서 지뢰가 있는 칸의 개수와 열려 있는 칸의 개수가 같은지로 판단합니다. 만약 승리했다면, 리더보드를 가져와서 리더보드 상의 이름과 점수를 vector에 각각 저장합니다. 이후, 최종 결과가 리더보드를 갱신하는지를 판단합니다. 만약 1등을 차지한다면, 1등의 점수를 2등으로, 2등의 점수를 3등으로 내린 뒤 1등의 점수를 갱신하고, 2등을 차지한다면 2등의 점수를 3등으로 내린 뒤 2등의 점수를 갱신하고, 3등을 차지한다면 3등의 점수만 갱신합니다. 이 연산은 이름 vector와 점수 vector 모두에서 진행됩니다. 이름 vector와 점수 vector는 난이도가 낮은 순서대로 값이 저장되어 있기 때문에, 인덱스 상으로 0, 1, 2는 낮은 난이도, 3, 4, 5는 중간 난이도, 6, 7, 8은 높은 난이도의 리더보드 정보를 가지고 있습니다. 이름과 점수 vector의 내용을 갱신하고 난 뒤에 이를 다시 리더보드 txt 파일에 저장합니다. 화면 상에는 승리하는데까지 걸린 시간이 출력됩니다.

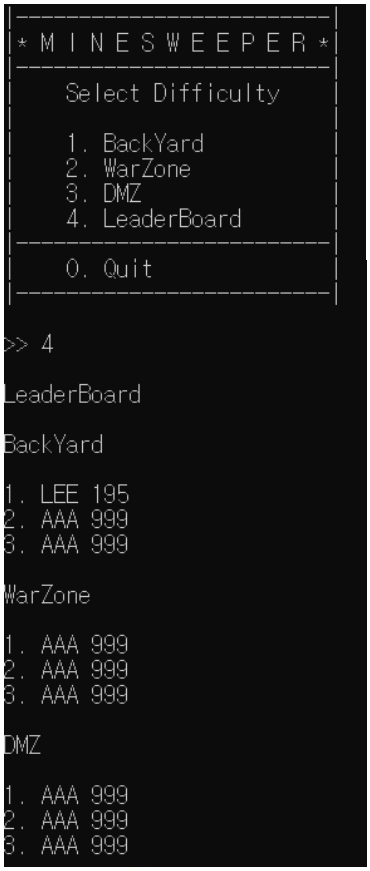


엪의 화면은 프로그램을 실행했을 때 나오게 되는 화면입니다. 먼저 플레이어는 이름을 입력하게 됩니다. 그 다음에는 게임 시작 화면이 등장합니다. 4가지의 옵션 중 하나를 고르게 되며, 그림의 예시는 LeaderBoard 옵션을 선택했을 때 출력되는 예시입니다. 위에서 설명한대로 가상의 최고 기록이 작성되어있는 모습입니다.



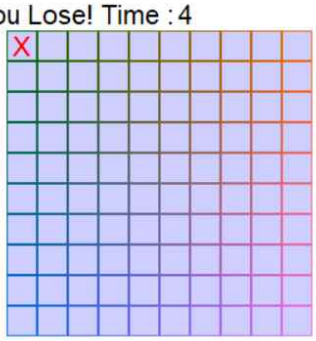
난이도를 선택하고 나면 게임 화면이 등장합니다. 게임 화면 상에는 현재 난이도와 남은 지뢰의 개수를 표현해줍니다. 남은 지뢰의 개수는 실제 지뢰의 개수에서 플레이어가 깃발로 지운 칸의 개수의 차입니다.

승리 화면입니다. 승리하면 다음과 같이 축하 메시지가 나오고, 해결하는데 걸린 시간이 표시됩니다.



최고 기록을 갱신한 뒤에 LeaderBoard를 확인해보면 갱신된 최고 기록을 확인할 수 있습니다.

You Lose! Time : 4



다음은 패배 화면입니다. 다음과 같이 플레이어가 클릭한 지뢰를 빨간색 X로 표시해주고, 소요된 시간을 표시해줍니다.