객체기반SW설계 기말 프로젝트

임베디드시스템공학과 202201658 박소윤

- Sfml 2.6.1에서도 실행되고 , 비쥬얼 스튜디오는 2022년, 64bit 입니다.

0-1. 공통 단계 (참고용)

1) 신규 클래스 정의

1 단계: Mouse, Animal, Penguin

2 단계: Mouse, Animal, Penguin, Obstacle, K, N, D, L, Row, Column

3 단계: Mouse, Animal, Penguin, Obstacle, M, Column_2

2) 클래스 상속

1 단계: (Animal – Penguin)

2 단계: (Animal – Penguin), (Obstacle - K, N, D, L, Row, Column)

3 단계: (Animal – Penguin), (Obstacle - M, Column_2)

3) Operator overloading (main 에서 사용)

1 단계: Mouse class 안에 있는 Mouse& operator++

2 단계: Mouse class 안에 있는 Mouse& operator++

3 단계: Mouse class 안에 있는 Mouse& operator++

4) Pure virtual function

1 단계: Animal - `virtual void animal()`

2 단계: Animal - `virtual void animal()=0`, Obstacle - `virtual bool make_Obstacle()=0`

3 단계: Animal - `virtual void animal()=0`, Obstacle - `virtual bool make_Obstacle()=0`

5) Function overloading (main 에서 사용)

1 단계: `void file_overloading(ofstream& _os)` - `void file_overloading(ifstream& _is)`

2 단계: `void file_overloading(ofstream& _os)` - `void file_overloading(ifstream& _is)`

3 단계: `void file_overloading(ofstream& _os)` - `void file_overloading(ifstream& _is)`

6) Function overriding

1 단계: (Animal: `animal()` – Penguin: `animal()`)

2 단계: (Animal: animal() – Penguin: animal()),

(Obstacle: make_Obstacle – K: make_Obstacle,

N: make_Obstacle, D: make_Obstacle, L: make_Obstacle,

Row: make_Obstacle, Column: make_Obstacle)

3 단계: (Animal: animal() – Penguin: animal()),

(Obstacle: make_Obstacle – M: make_Obstacle, Column: make_Obstacle)

7) Lambda expression (main 에서 사용)

1 단계: `auto version = [](string version_name){};`

2 단계: `auto version = [](string version_name){};`

3 단계: `auto version = [](string version_name){};`

8) Try-catch 예외처리 (main 에서 사용)

1 단계: 자기 자신의 위치를 목적지를 설정했을 때의 예외처리

2 단계: 자기 자신의 위치를 목적지를 설정했을 때의 예외처리

3 단계: 자기 자신의 위치를 목적지를 설정했을 때의 예외처리

9) File input/output (main 에서 사용)

1단계:
```
ofstream os{ "obstacle.txt" }; , file_overloading(os);
ifstream is{ "obstacle.txt" }; , file_overloading(is);
```
2단계:
```
ofstream os{ "obstacle.txt" }; , file_overloading(os);
ifstream is{ "obstacle.txt" }; , file_overloading(is);
```
3단계:
```
ofstream os{ "obstacle.txt" }; , file_overloading(os);
ifstream is{ "obstacle.txt" }; , file_overloading(is);
```

10) STL map (main 에서 사용)

1 단계: `gbl::Map<> map = {};`

2 단계: `gbl::Map<> map = {};`

3 단계: `gbl::Map<> map = {};`

- 코드 첨부한 사진 순서대로 sw.cpp 파일의 내용 줄을 적었습니다. Ex) (sw.cpp : 1~5)

0-2. 공통 단계 구현 사례

1) Lambda expression 사례 (sw.cpp : 800~802)

```
auto version = [](string version_name) {
    std::cout << "SFML Version:" << version_name << "₩n";
};

version("2.6.0");
```

: 버전 출력을 위한 함수입니다. 일회용 함수로 사용하기 위해 람다 연산자인 []을 이용해서 람다식을 구현했습니다. Main 문 안에 넣어 1, 2, 3 단계 공통 구현 충족했습니다.

2) try – catch 예외처리 (sw.cpp : 990~1016)

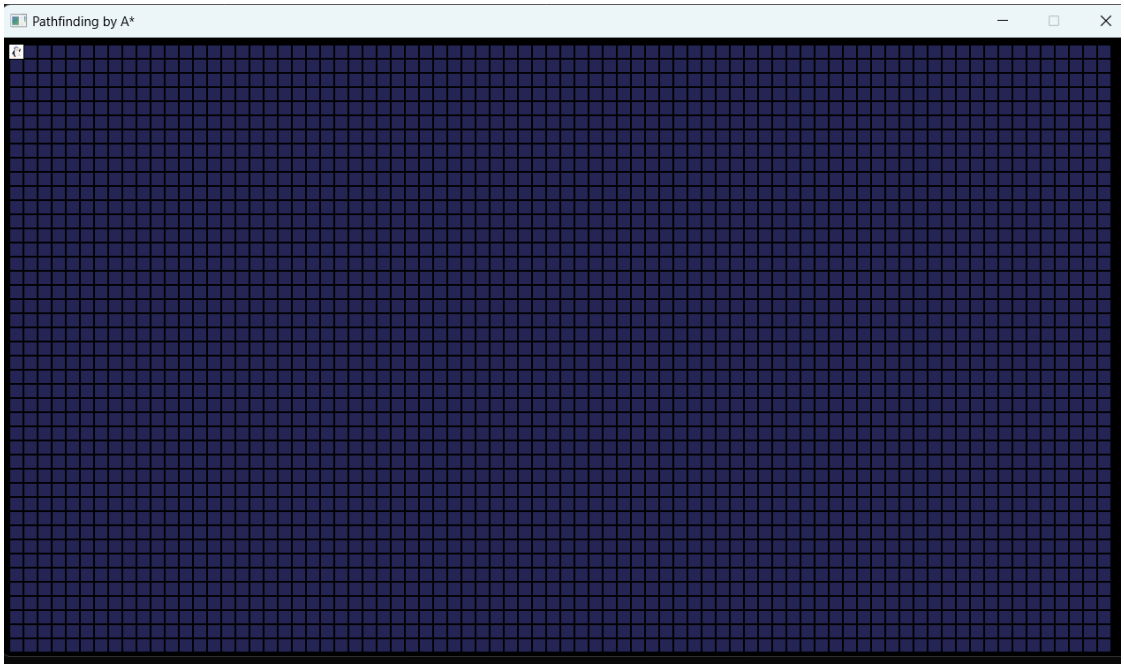
```
bool me_ok = true;
try {
    if (finish_position == static_cast<gbl::Position<>>(mouse_cell) || start_position ==
        static_cast<gbl::Position<>>(mouse_cell))
    {
        throw me_ok;
    }

    if (gbl::MAP::Cell::Wall1 != map[mouse_cell.first][mouse_cell.second]
        && gbl::MAP::Cell::Wall2 != map[mouse_cell.first][mouse_cell.second]
        && gbl::MAP::Cell::Wall3 != map[mouse_cell.first][mouse_cell.second]
        && gbl::MAP::Cell::Wall4 != map[mouse_cell.first][mouse_cell.second]
        && gbl::MAP::Cell::Wall5 != map[mouse_cell.first][mouse_cell.second]
        && gbl::MAP::Cell::Wall6 != map[mouse_cell.first][mouse_cell.second]
        && gbl::MAP::Cell::Wall7 != map[mouse_cell.first][mouse_cell.second]
        && gbl::MAP::Cell::Wall8 != map[mouse_cell.first][mouse_cell.second]
        )
    {
        map_updated = 1;
        path_cell_save.clear();
        finish_position = mouse_cell;
    }

}
catch (bool _me_ok) {
    cout << "이미 위치에 있습니다, " << "₩n";
}
```

: 자기 자신을 목적지로 설정했는지 판단하는 변수 me_ok 만들었습니다. 만약 try 바로 밑 if 문에 걸려서 자기 자신을 목적지로 설정한 예외에 걸렸다면 throw 하고 try 바깥에서 catch 로 예외처리를 해줍니다. 위 코드는 이미 위치에 있다는 것을 출력하도록 예외처리 했습니다.

1. 1 단계

1) 캐릭터 (sw.cpp : 73~91, 415~438, 844)

<결과> 캐릭터 생성



- 동물 모양은 펭귄으로 했습니다.

```
class Animal {
public:
    Animal() {}
    sf::Sprite s;
    sf::Texture penguin_texture;
    virtual void animal(sf::Sprite& map_sprite) = 0;
};
```

```
class Penguin : public Animal {
public:
    Penguin() {
        penguin_texture.loadFromFile("penguin copy.png");
    }
    void animal(sf::Sprite& _sprite) {
        _sprite.setTexture(penguin_texture);
    }
};
```

: Penguin class 안에서 생성자는 이미지를 로드하고 animal 함수는 sprite 와 texture 를 set 해서 창에 sprite 를 그리게 합니다.

```
/*
화면에 셀을 그리는 함수
*/
void draw_map(const unsigned short _x,
    const unsigned short _y,
    const gbl::Position<>& _finish_position,
    const gbl::Position<>& _start_position,
    sf::RenderWindow& _window,
    sf::Sprite& _sprite,
    sf::Texture map_texture,
    Penguin penguin,
    const gbl::Map<>& _map)
{
    for (unsigned short a = 0; a < gbl::MAP::COLUMNS; a++)
    {
        for (unsigned short b = 0; b < gbl::MAP::ROWS; b++)
        {
            _sprite.setPosition((float)gbl::MAP::ORIGIN_X + _x + a * gbl::MAP::CELL_SIZE,
                (float)gbl::MAP::ORIGIN_Y + _y + b * gbl::MAP::CELL_SIZE);

            _sprite.setTexture(map_texture);
            if (a == _start_position.first && b == _start_position.second)
            {
                _sprite.setColor(sf::Color(255, 255, 255));
                _window.draw(_sprite);
                penguin.animal(_sprite);
            }
```

: Draw 을 시작하면 map 을 먼저 창에 띄우고 펭귄을 그리는 코드입니다.

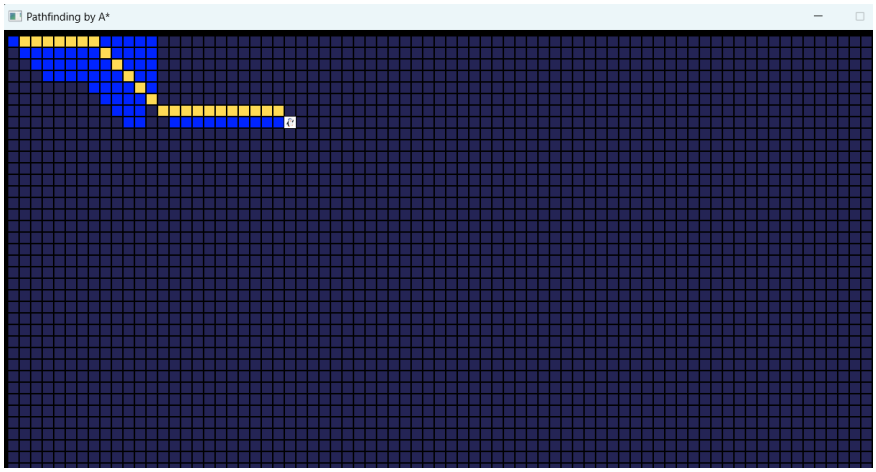
sprite.setColor 을 한 이유는 penguin copy.png 가 잘 안 보여서 추가했습니다. 셀에 흰색을 설정하고 먼저 그리고 그 위에 펭귄 객체에 애니몰 함수를 써서 스프라이트 펭귄을 위에 덮었습니다.

- 셀 크기 (16X16)을 맞추기 위해 사진의 크기를 16X16 로 맞췄습니다.
- 최초위치는 좌상단으로 하기 위해 x, y 위치를 0,0 으로 설정했습니다.

`gbl::Position<> start_position(0, 0);` (main 문)

2) 캐릭터 이동 (sw.cpp : 856~857, 925~976, 1036~1067)

<결과> 노란색 PATH 을 찾고 캐릭터가 이동




```
int leftbutton = 0;  //좌클릭 후 연속으로 목적지가 바뀌는 걸 막기 위한 button
int leftflag = 0;    //캐릭터가 이동을 종료했는지 확인하는 flag
```

: leftbutton 는 좌클릭 후 다음 좌클릭 할 때까지 목적지를 고정하기 위한 변수입니다.

leftflag=0 이면 캐릭터가 이동 종료한 것이고 1 이면 아직 이동 중인 상태를 알려주는 변수입니다.

```
// 키보드 입력에 대한 검사
while (1 == window.pollEvent(event))
{
    switch (event.type)
    {
    case sf::Event::Closed:
    {
        window.close();
        break;
    }
    case sf::Event::MouseButtonPressed:
    {
        gbl::Position<short> mousePosition = gbl::get_mouse_cell(window);
        ++mouse;
        cout << mouse.mouse_cnt << "\n";
        if (leftflag == 0) {
            if (mousePosition.first % 8 != 3) {
                if (event.mouseButton.button == sf::Mouse::Left) {
                    leftbutton = 1;
                    leftflag = 1;
                    around_wall_flag = 0;
                    break;
                }
            }
        }
    }
    }
}
```

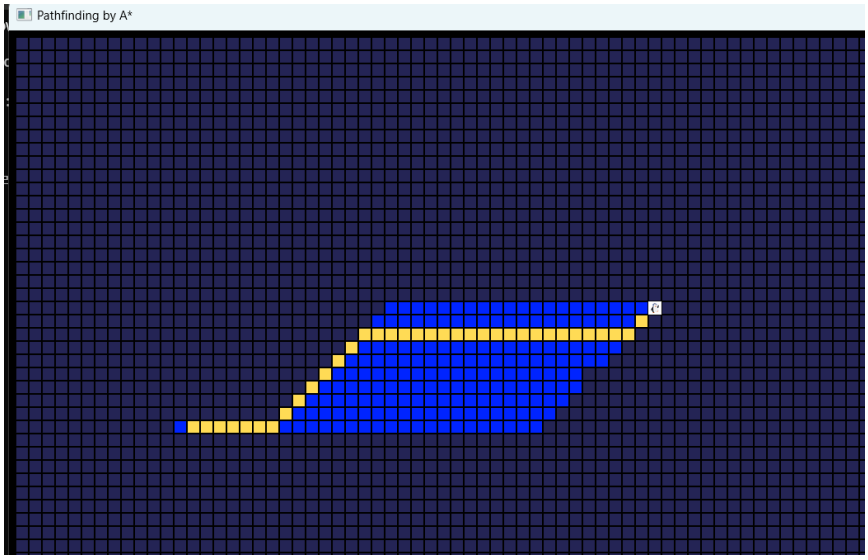
: 만약 leftflag 가 0 이면 이동 종료을 했으니 좌클릭 할 수 있고 1 이면 추가 버튼입력을 무시합니다.

```
if (false == astar_finished)
{
    astar_finished = astar_search(
        astar_path_length,
        astar_total_checks,
        astar_duration,
        astar_previous_cell,
        astar_path_vector,
        astar_f_scores,
        astar_g_scores,
        astar_h_scores,
        finish_position,
        start_position,
        map,
        path_cell_save,
        around_wall_flag);
}
else {
    if (!path_cell_save.empty()) {
        path_position = path_cell_save.back();
        if (map[path_position.first][path_position.second] == gbl::MAP::Cell::Path
            || map[path_position.first][path_position.second] == gbl::MAP::Cell::Empty
            || map[path_position.first][path_position.second] == gbl::MAP::Cell::Visited)
        {
            start_position = path_position;
            path_cell_save.pop_back();
        }
        if (path_cell_save.empty()) {
            leftflag = 0;
        }
    }
}
```

: path_cell_save 는 PATH 를 저장하는 vector 입니다. Astar search 가 끝나면 start_position 을 새롭게 저장하면서 path_cell_save 를 pop_back() 합니다. Vector 가 empty 가 되면 이동 종료를 한 것이기 때문에 leftflag=0 으로 해줍니다.

3) 계속 이동 (sw.cpp : 982~1019)

<결과> 목적지 도달 후 신규 목적지 설정되면 그곳으로 이동



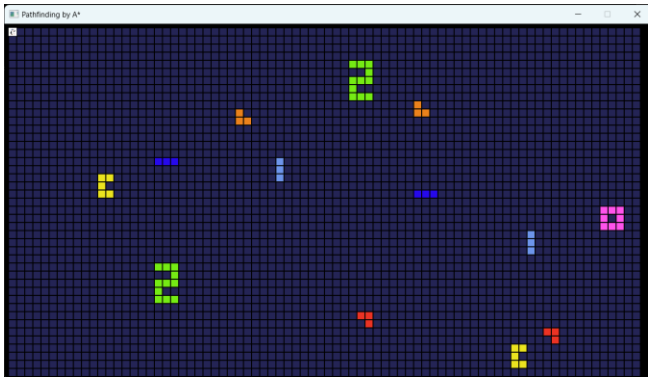

```
if (leftbutton == 1) {
    leftbutton = 0;
    gbl::Position<short> mouse_cell = gbl::get_mouse_cell(window);
    if (mouse_cell.first < gbl::MAP::COLUMNS && mouse_cell.second < gbl::MAP::ROWS)
    {
        bool idx_ok = true;
        try {
            if (finish_position == static_cast<gbl::Position<>>(mouse_cell) || start_position ==
                static_cast<gbl::Position<>>(mouse_cell))
            {
                throw idx_ok;
            }

            if (gbl::MAP::Cell::Wall1 != map[mouse_cell.first][mouse_cell.second])
            {
                map_updated = 1;
                path_cell_save.clear();
                finish_position = mouse_cell;
            }

        }
        catch (bool _idx_ok) {
            cout << "이미 위치에 있습니다, " << "\n";
        }
    }
}
```

: 좌클릭 좌표가 벽이 아니면 맵을 업데이트하고 그 전에 추가했던 vector 인 path_cell_save 를 clear 해서 새로운 PATH 를 저장할 수 있게 합니다. 위에서 설명한 캐릭터 이동 부분 중에 계속 start_position 이 PATH 에 따라 변경돼서 목적지에 도달하면 그 위치가 출발점이 됩니다. 그리고 finish_position=mouse_cell 을 이용해서 좌클릭 후 신규 목적지를 설정합니다.

2. 2 단계

1) 장애물 (sw.cpp : 864~908, 403~410)

<결과> 랜덤 위치에 장애물 10 개 ~ 30 개



- 모형 구분을 위해 여러 색을 사용했고 ㅁ만 pink 색으로 표시했습니다.

```
pair<int, int>random_p;
random_p = random_position();

while ((m.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
    random_p = random_position();
}

for (int i = 0; i < 2; i++)
{
    while ((k.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
        random_p = random_position();
    }
    while ((n.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
        random_p = random_position();
    }
    while ((d.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
        random_p = random_position();
    }
    while ((l.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
        random_p = random_position();
    }
    while ((row.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
        random_p = random_position();
    }
    while ((column.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
        random_p = random_position();
    }
}
```

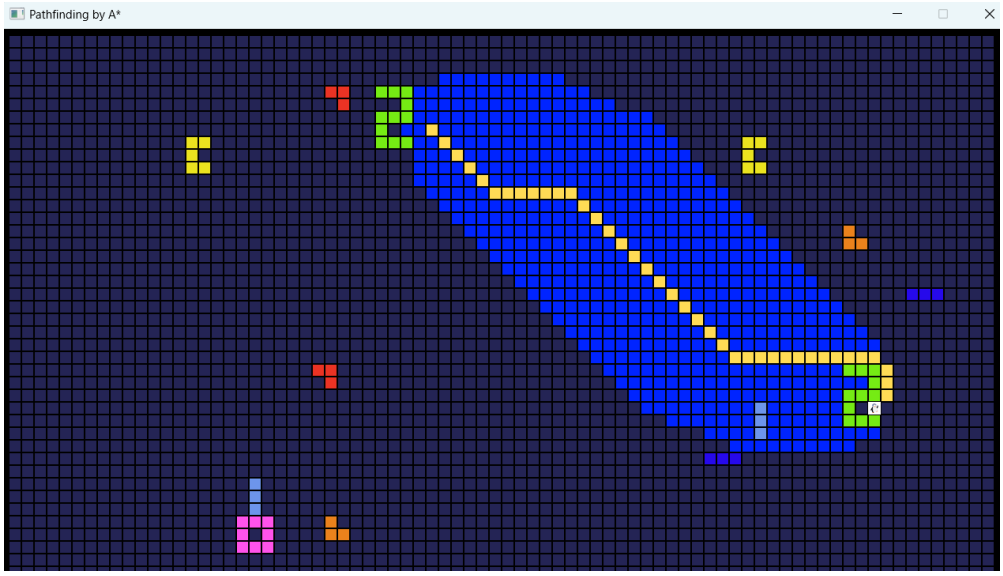
: 랜덤으로 x, y 구하고 그 위치에 맞게 ㄱ,ㄴ,ㄷ,ㄹ 모양과 가로 3 칸 모양, 세로 3 칸 모양을 2 개씩, ㅁ은 1 개만 나타냈습니다. 장애물을 구현하는 자세한 내용은 밑에 장애물 모양 복잡도에서 코드 과정을 설명하겠습니다.

```
pair<int, int> random_position() {
    int x = rand() % 78;
    int y = rand() % 43;
    return make_pair(x, y);
}
```

: x, y 를 랜덤으로 구하는 함수입니다.

2) 장애물 회피 (sw.cpp : 560~581)

<결과> 장애물을 회피한 경로 선택 후 이동

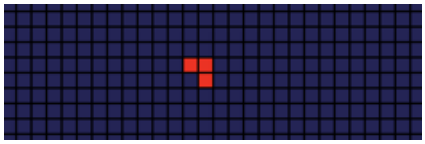



```
for (auto a = -1; a < 2; a++)
{
    for (auto b = -1; b < 2; b++)
    {
        // 자기자신 제외

        if (0 == a && 0 == b)
        {
            continue;
        }

        gbl::MAP::Cell _type = get_cell(gbl::Position<short>(_cell.first + a, _cell.second + b), _map);
        // 벽이거나, 범위 밖이 아닌 경우 포함
        if (gbl::MAP::Cell::Invalid != _type && gbl::MAP::Cell::Wall1 != _type
            && gbl::MAP::Cell::Wall2 != _type && gbl::MAP::Cell::Wall7 != _type
            && gbl::MAP::Cell::Wall3 != _type
            && gbl::MAP::Cell::Wall8 != _type
            && gbl::MAP::Cell::Wall4 != _type && gbl::MAP::Cell::Wall9 != _type
            && gbl::MAP::Cell::Wall5 != _type && gbl::MAP::Cell::Wall6 != _type)
        {
            adjacent_cells.push_back(gbl::Position<>(a + _cell.first, b + _cell.second));
        }
    }
}
```

:Wall1~9 는 장애물입니다. 이 코드에서 타입이 Wall 이거나 Invlid 이면 adjacent_cells 에 push_back 을 안 하기 때문에 캐릭터가 이동 중 장애물을 회피하는 경로로 이동합니다.

3) 장애물 모양 복잡도(ㄱ) (sw.cpp :96~132, 880~882)

<결과> ㄱ



```
class Obstacle {
public:
    Obstacle() {

    }
    virtual bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& data) = 0;
};

class K :public Obstacle {
public:
    K() {}
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
        if (x + 2 < 78 && y + 2 < 43
            && _data[x][y] != 1
            && _data[x + 1][y] != 1
            && _data[x + 1][y + 1] != 1)
        {
            obstacle_cnt += 3;
            _map[x][y] = gbl::MAP::Cell::Wall1;
            _map[x + 1][y] = gbl::MAP::Cell::Wall1;
            _map[x + 1][y + 1] = gbl::MAP::Cell::Wall1;
            _data[x][y] = 1;
            _data[x + 1][y] = 1;
            _data[x + 1][y + 1] = 1;
            return 1;
        }
        else {
            return 0;
        }
    }
};
```

: class Obstacle 을 만들어서 생성자와 pure virtual 함수인 make_Obstacle 를 만들었습니다. 이 함수는 모양에 맞게 맵에 나타내주는 함수입니다.

: Class K 의 make_Obstacle 에서는 랜덤으로 받은 x, y 가 맵 범위 안에 들어가면 셀 수를 세기 위한 obstacle_cnt 와 ㄱ 의 모양대로 map 에서 벽으로 타입을 바꿔서 장애물을 만들어줍니다.

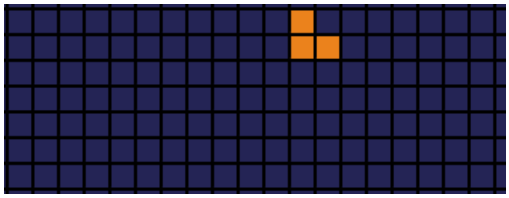
_data 는 main 에서 받아온 data 를 의미합니다. data 는 vector<vector<int>> 타입이고 0 으로 초기화한 상태 입니다. data 의 사용은 장애물이 겹치는 걸 방지하기 위한 것입니다. 모양대로 벽을 만들면 1 로 업데이트를 해서 벽이 있다는 상태를 알려줍니다. 그래서 위 코드에서 if 문을 보면 data 의 x, y 위치 값이 1 이 아닐 때만 if 문 안으로 들어가는 걸 볼 수 있습니다. if 문에 들어가지 못한다면 0 을 return 해줍니다. 밑에 코드랑 이어서 봐 보겠습니다.

```
while ((k.make_Obstacle(random_p.first, random_p.second, map, data)) == 0) {
    random_p = random_position();
}
```

: 이 코드는 main 문에 있는 코드입니다. 만약 개체 k 가 make_Obstacle 을 이용해서 return 0 을 받아오면 다시 하고 1 을 받으면 while 문을 나오게 코드를 구현하였습니다.

4) 장애물 모양 복잡도(ㄴ) (sw.cpp : 135~158)

<결과> ㄴ



```
class N :public Obstacle {
public:
    N() {}
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
        if (x + 2 < 78 && y + 2 < 43
            && _data[x][y] != 1
            && _data[x][y + 1] != 1
            && _data[x + 1][y + 1] != 1)
        {
            obstacle_cnt += 3;
            _map[x][y] = gbl::MAP::Cell::Wall2;
            _map[x][y + 1] = gbl::MAP::Cell::Wall2;
            _map[x + 1][y + 1] = gbl::MAP::Cell::Wall2;
            _data[x][y] = 1;
            _data[x][y + 1] = 1;
            _data[x + 1][y + 1] = 1;
            return 1;
        }
        else {
            return 0;
        }
    }
};
```

: 모양 바꾸는 것을 제외하고는 ㄱ 자형 모형 만드는 class 내용과 동일합니다.

5) 장애물 모양 복잡도(ㄷ) (sw.cpp :161~190)

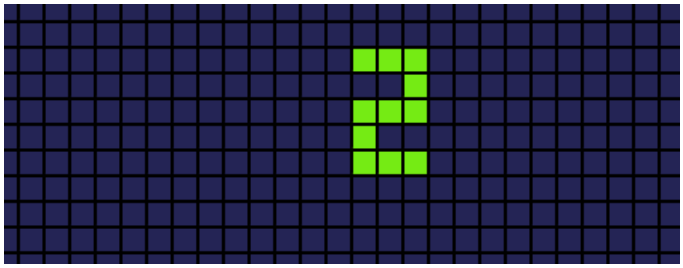
<결과> ㄷ

```
class D :public Obstacle {
public:
    D() {}
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
        if (x + 2 < 78 && y + 3 < 43
            && _data[x][y] != 1
            && _data[x + 1][y] != 1
            && _data[x][y + 1] != 1
            && _data[x][y + 2] != 1
            && _data[x + 1][y + 2] != 1)
        {
            obstacle_cnt += 5;
            _map[x][y] = gbl::MAP::Cell::Wall3;
            _map[x + 1][y] = gbl::MAP::Cell::Wall3;
            _map[x][y + 1] = gbl::MAP::Cell::Wall3;
            _map[x][y + 2] = gbl::MAP::Cell::Wall3;
            _map[x + 1][y + 2] = gbl::MAP::Cell::Wall3;
            _data[x][y] = 1;
            _data[x + 1][y] = 1;
            _data[x][y + 1] = 1;
            _data[x][y + 2] = 1;
            _data[x + 1][y + 2] = 1;
            return 1;
        }
        else {
            return 0;
        }
    }
};
```

: 모양 바꾸는 것을 제외하고는 ㄱ 자형 모형 만드는 class 내용과 동일합니다.

6) 장애물 모양 복잡도(ㄹ) (sw.cpp :193~240)

<결과> ㄹ



```
class L :public Obstacle {
public:
    L() {}
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
        if (x + 3 < 78 && y + 5 < 43
            && _data[x][y] != 1
            && _data[x + 1][y] != 1
            && _data[x + 2][y] != 1
            && _data[x + 2][y + 1] != 1
            && _data[x + 2][y + 2] != 1
            && _data[x + 1][y + 2] != 1
            && _data[x][y + 2] != 1
            && _data[x][y + 3] != 1
            && _data[x][y + 4] != 1
            && _data[x + 1][y + 4] != 1
            && _data[x + 2][y + 4] != 1)
        {
            obstacle_cnt += 11;
            _map[x][y] = gbl::MAP::Cell::Wall4;
            _map[x + 1][y] = gbl::MAP::Cell::Wall4;
            _map[x + 2][y] = gbl::MAP::Cell::Wall4;
            _map[x + 2][y + 1] = gbl::MAP::Cell::Wall4;
            _map[x + 2][y + 2] = gbl::MAP::Cell::Wall4;
            _map[x + 1][y + 2] = gbl::MAP::Cell::Wall4;
            _map[x][y + 2] = gbl::MAP::Cell::Wall4;
            _map[x][y + 3] = gbl::MAP::Cell::Wall4;
            _map[x][y + 4] = gbl::MAP::Cell::Wall4;
            _map[x + 1][y + 4] = gbl::MAP::Cell::Wall4;
            _map[x + 2][y + 4] = gbl::MAP::Cell::Wall4;
            _data[x][y] = 1;
            _data[x + 1][y] = 1;
            _data[x + 2][y] = 1;
            _data[x + 2][y + 1] = 1;
            _data[x + 2][y + 2] = 1;
            _data[x + 1][y + 2] = 1;
            _data[x][y + 2] = 1;
            _data[x][y + 3] = 1;
            _data[x][y + 4] = 1;
            _data[x + 1][y + 4] = 1;
            _data[x + 2][y + 4] = 1;
            return 1;
        }
        else {
            return 0;
        }
    }
};
```

: 모양 바꾸는 것을 제외하고는 ㄱ 자형 모형 만드는 class 내용과 동일합니다.

7) 장애물 겹침 방지(sw.cpp :283~333, 864~898, 916~922)

```cpp
// 가로 3칸 모양
class Row :public Obstacle {
public:
    Row() {}
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
        if (x + 2 < 78 && y + 2 < 43
            && _data[x][y] != 1
            && _data[x + 1][y] != 1
            && _data[x + 2][y] != 1)
        {
            obstacle_cnt += 3;
            _map[x][y] = gbl::MAP::Cell::Wall6;
            _map[x + 1][y] = gbl::MAP::Cell::Wall6;
            _map[x + 2][y] = gbl::MAP::Cell::Wall6;

            _data[x][y] = 1;
            _data[x + 1][y] = 1;
            _data[x + 2][y] = 1;
            return 1;
        }
        else {
            return 0;
        }
    }
};

// 세로 3칸 모양
class Column :public Obstacle {
public:
    Column() {}
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
        if (x + 2 < 78 && y + 2 < 43
            && _data[x][y] != 1
            && _data[x][y + 1] != 1
            && _data[x][y + 2] != 1)
        {
            obstacle_cnt += 3;
            _map[x][y] = gbl::MAP::Cell::Wall7;
            _map[x][y + 1] = gbl::MAP::Cell::Wall7;
            _map[x][y + 2] = gbl::MAP::Cell::Wall7;

            _data[x][y] = 1;
            _data[x][y + 1] = 1;
            _data[x][y + 2] = 1;
            return 1;
        }
        else {
            return 0;
        }
    }
};
```

Class Obstacle 만들어서 이 class를 상속받은 장애물 ㄱ,ㄴ,ㄷ,ㄹ,ㅁ,세로 3칸, 가로 3칸 class을 만들었습니다. 함수 make_Obstacle에 랜덤좌표 x, y와 셀을 벽으로 만들기 위한 _map, 겹침 확인을 위한 2차원 배열 _data(78*43)을 인자로 받았습니다. 우선 data배열은 다 0으로 초기화하고 문자 모양이면 1로 변환할 것입니다. 만약 랜덤인 x, y 좌표가 화면 범위 안에 있고 _data의 각자 모양이 1이 아니면 문자 모양으로 벽을 만들겠다와 셀 수를 count하라는 if문을 만듭니다. 모양이 1이면 벽이 있으니까 겹침을 방지할 수 있는 것입니다. 만약 if문에 들어가지 않고 else에 들어갈 때는 셀 수를 count 막는 경우입니다.

```
A a;
N n;
D d;
L l;
M m; //-------------------------------------------------
Row row;
Column column;
Mini mini;
Row_2 row_2;
pair<int, int>random_p;
vector<vector<int>>data(78, vector <int>(43, 0));

for (int i = 0; i < 15; i++)
{
    random_p = random_position( );
    a.make_Obstacle(random_p.first, random_p.second, map, data);
    random_p = random_position( );
    n.make_Obstacle(random_p.first, random_p.second, map, data);
    random_p = random_position( );
    d.make_Obstacle(random_p.first, random_p.second, map, data);
    random_p = random_position( );
    l.make_Obstacle(random_p.first, random_p.second, map, data);
    //random_p = random_position( );
   // mini.make_Obstacle(random_p.first, random_p.second, map);
}

while (obstacle_cnt <= 2350) {

    random_p = random_position( );
    row.make_Obstacle(random_p.first, random_p.second, map, data);
    random_p = random_position( );
    column.make_Obstacle(random_p.first, random_p.second, map, data);
}
```

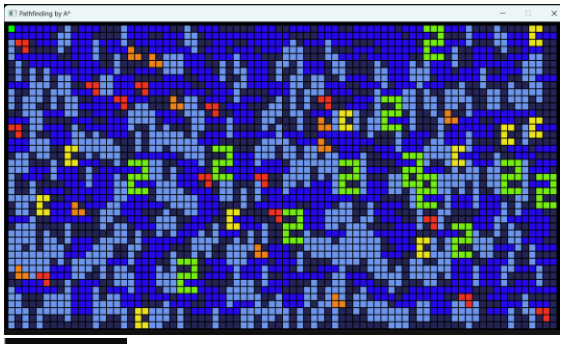
ㄱ,ㄴ,ㄷ,ㄹ 문자 15개 반복하고 전체 셀 수 70%인 2347개를 넘어서 약 2350개의 장애물을 카운트합니다. 이때는 작은 장애물인 세로 3칸, 가로 3칸을 이용해서 70% 이상의 셀을 벽으로 만듭니다.

<결과>

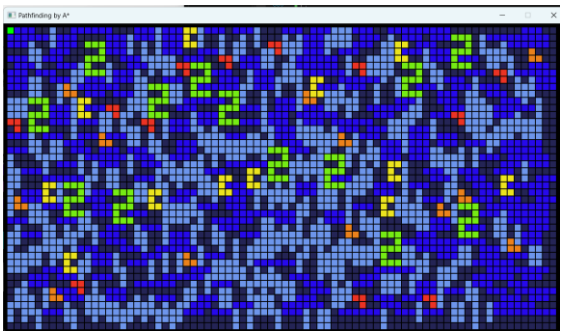
ㄱ: 빨간색, ㄴ: 주황색, ㄷ: 노란색, ㄹ: 초록색, 세로 3칸: 어두운 하늘색, 가로 3칸: 남색

(좌측이 시작점, 핑크색으로 벽을 만들면 구분이 어려워서 다양한 색깔로 구현했습니다.)

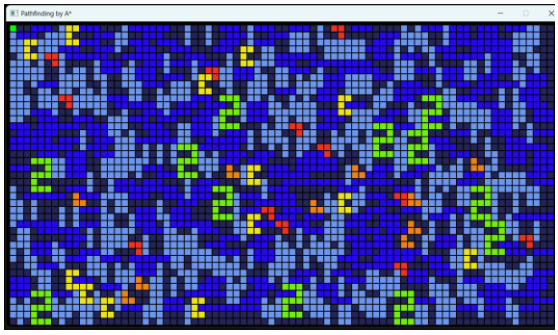
Cout 출력: 78*43, ₩n , 벽이 된 셀 수, ₩n, 78*43의 70% 셀



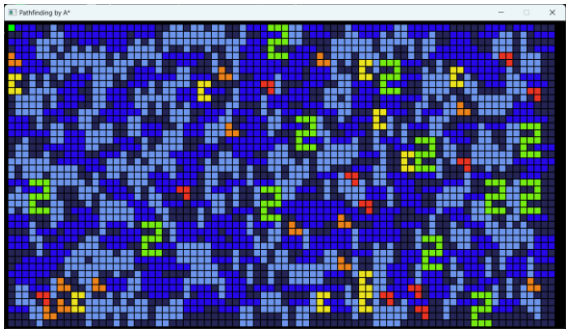
3354
2351
2347.8



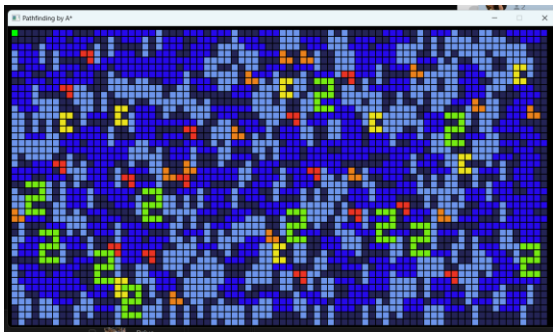
3354
2351
2347.8



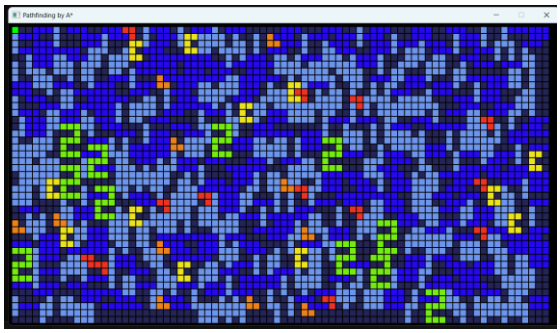
3354
2351
2347.8



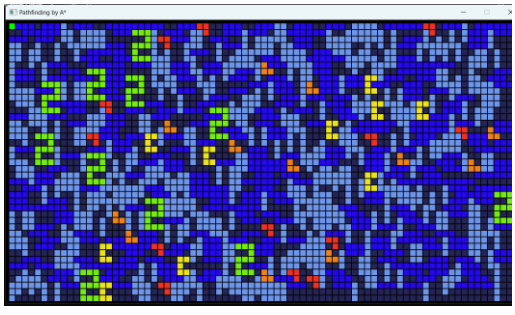
3354
2351
2347.8

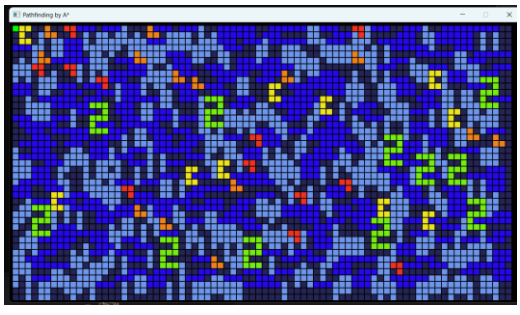


3354
2351
2347.8

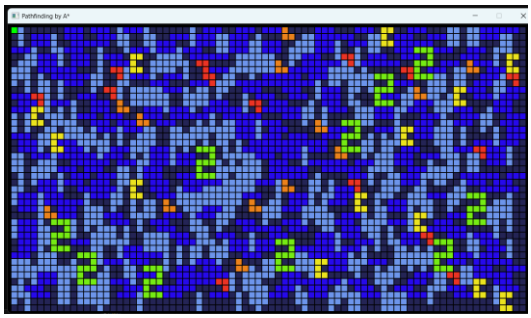


3354
2353
2347.8

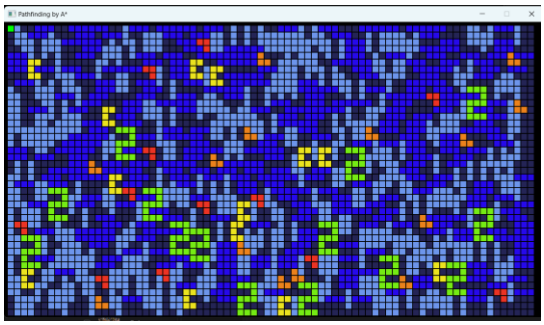




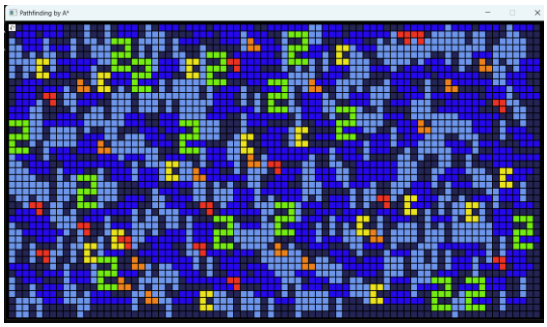
3354
2352
2347.8



3354
2353
2347.8

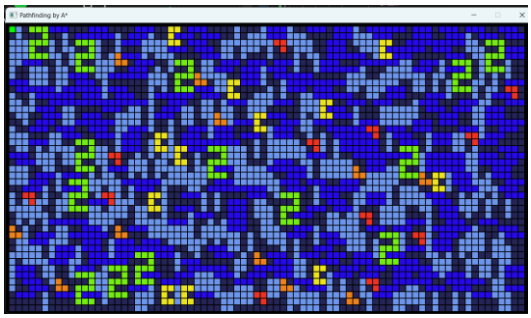


3354
2352
2347.8

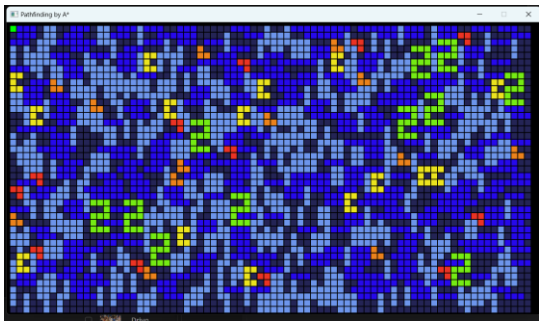


3354
2352
2347.8

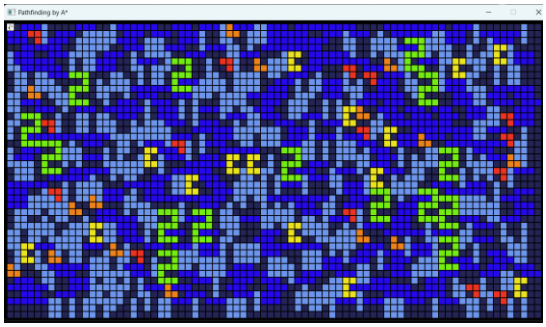
3354
2352
2347.8



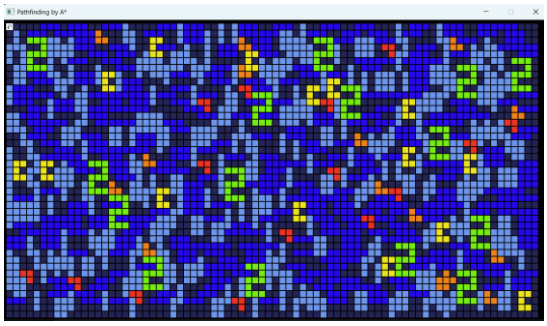
3354
2353
2347.8



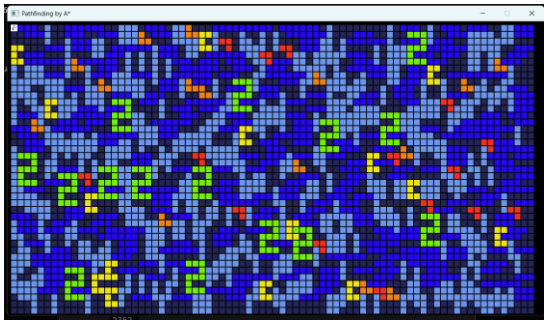
3354
2353
2347.8



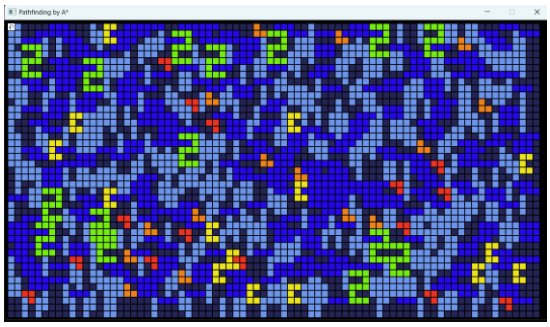
3354
2351
2347.8



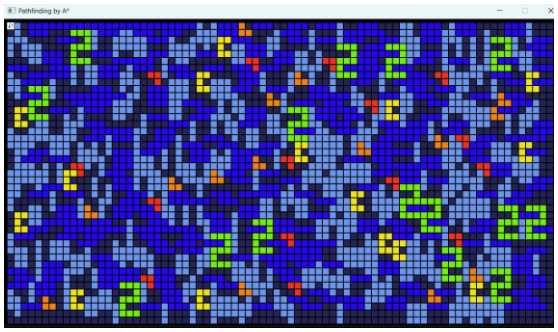
3354
2353
2347.8



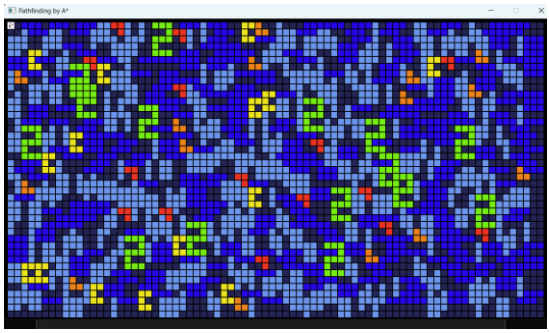
3354
2352
2347.8



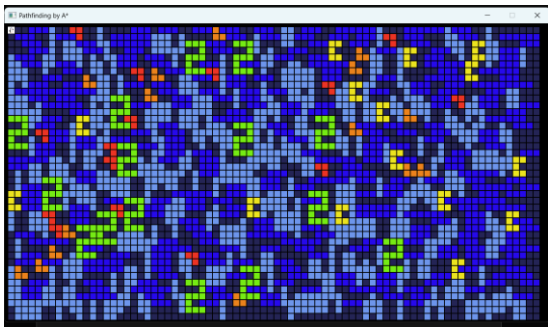
3354
2352
2347.8



3354
2351
2347.8



3354
2351
2347.8

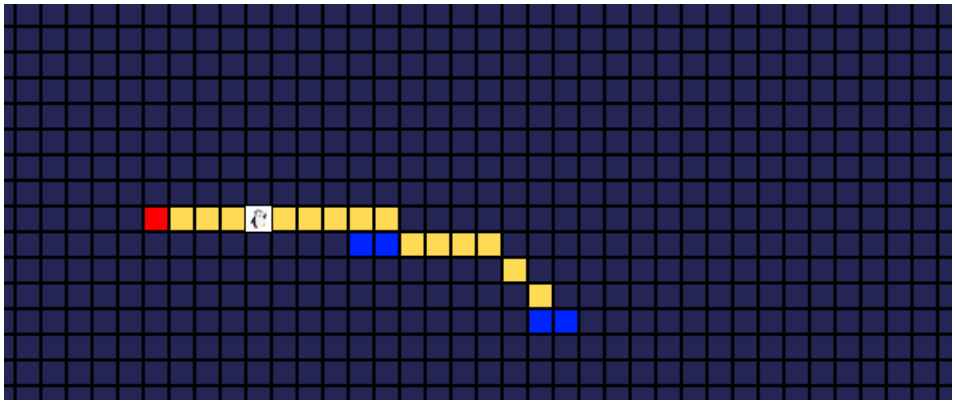


3354
2353
2347.8

3. 3 단계

1) 이동 중 목적지 변경 (sw.cpp :940~981)

<결과> 캡쳐는 이동 중 다른 셀을 누르면 경로가 바뀌는 걸 보여 수 없어서 코드 참고

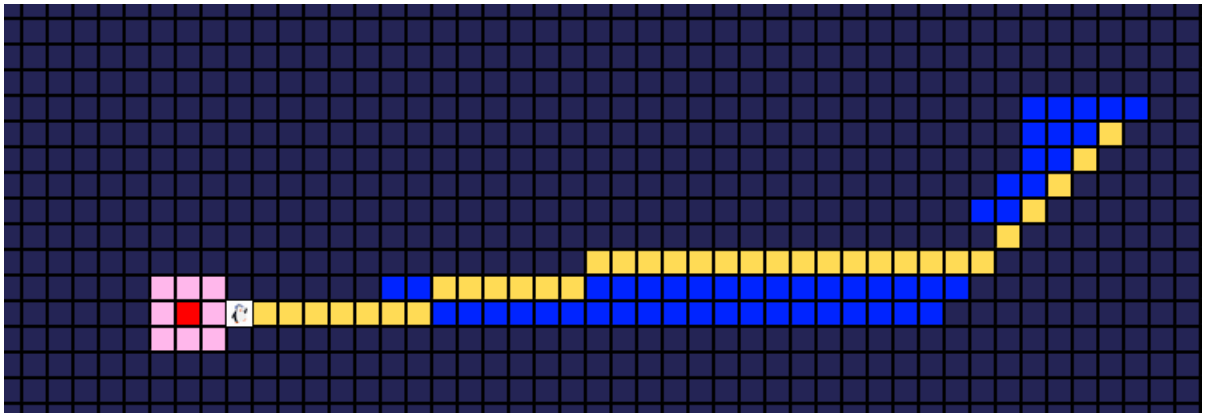


```
// 키보드 입력에 대한 검사
while (1 == window.pollEvent(event))
{
    switch (event.type)
    {
    case sf::Event::Closed:
    {
        window.close();
        break;
    }
    case sf::Event::MouseButtonPressed:
    {
        gbl::Position<short> mousePosition = gbl::get_mouse_cell(window);
        ++mouse;
        cout << mouse.mouse_cnt << "₩n";
        //if (leftflag == 0) {
            if (mousePosition.first % 8 != 3) {
                if (event.mouseButton.button == sf::Mouse::Left) {
                    leftbutton = 1;
                    leftflag = 1;
                break;
                }
            }
    //  }
    }
    }
}
```

: 앞에서 설명한 1 단계의 (2)번에서 추가 버튼입력을 무시하기 위해 leftflag 을 만들었습니다. 하지만 이번 단계에서는 추가 버튼 입력을 무시하지 않고 목적지를 바꿔야하니까 if 문 leftflag 를 주석처리 하면 캐릭터가 목적지로 이동하는 중에 목적지와 다른 셀을 누르면 신규 목적지로 경로 이동합니다.

2) 이동불가지역 (sw.cpp :243~281, 956~976, 529~558)

<결과> 목적지가 도달불가하면, 그 지점 인접하느 접근 가능한 좌표로 이동



```
class M :public Obstacle {
public:
    M() {}
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
        if (x + 2 < 78 && y + 2 < 43
            && _data[x][y] != 1
            && _data[x + 1][y] != 1
            && _data[x + 2][y] != 1
            && _data[x][y + 1] != 1
            && _data[x][y + 2] != 1
            && _data[x + 1][y + 2] != 1
            && _data[x + 2][y + 2] != 1
            && _data[x + 2][y + 1] != 1)
        {
            obstacle_cnt += 8;
            _map[x][y] = gbl::MAP::Cell::Wall5;
            _map[x + 1][y] = gbl::MAP::Cell::Wall5;
            _map[x + 2][y] = gbl::MAP::Cell::Wall5;
            _map[x][y + 1] = gbl::MAP::Cell::Wall5;
            _map[x][y + 2] = gbl::MAP::Cell::Wall5;
            _map[x + 1][y + 2] = gbl::MAP::Cell::Wall5;
            _map[x + 2][y + 1] = gbl::MAP::Cell::Wall5;
            _map[x + 2][y + 2] = gbl::MAP::Cell::Wall5;
            _data[x][y] = 1;
            _data[x + 1][y] = 1;
            _data[x + 2][y] = 1;
            _data[x][y + 1] = 1;
            _data[x][y + 2] = 1;
            _data[x + 1][y + 2] = 1;
            _data[x + 2][y + 2] = 1;
            _data[x + 2][y + 1] = 1;
            return 1;
        }
        else {
            return 0;
        }
    }
};
```

: 위의 2 단계의 장애물 모양을 만드는 부분과 동일하게 ㅁ 모양 장애물도 만들었습니다.

```
if (event.mouseButton.button == sf::Mouse::Left) {
    leftbutton = 1;
    leftflag = 1;
    around_wall_flag = 0;

    if (data[mousePosition.first - 1][mousePosition.second - 1] == 1
        && data[mousePosition.first][mousePosition.second - 1] == 1
        && data[mousePosition.first - 1][mousePosition.second - 1] == 1
        && data[mousePosition.first - 1][mousePosition.second] == 1
        && data[mousePosition.first + 1][mousePosition.second] == 1
        && data[mousePosition.first - 1][mousePosition.second + 1] == 1
        && data[mousePosition.first][mousePosition.second + 1] == 1
        && data[mousePosition.first + 1][mousePosition.second + 1] == 1
        )
    {
        around_wall_flag = 1;
    }

    break;
}
```

: main 문 안에 around_wall_flag 을 선언했습니다. 값이 0 이면 목적지로 좌표 잡은 곳이 벽으로 막혀있지 않다는 의미이고 1 이면 사방이 벽으로 막혀있다는 것을 의미합니다. 위에서 설명한 것처럼 vector 인 data 의 값이 1 인지 0 인지 확인해서 벽을 확인합니다.

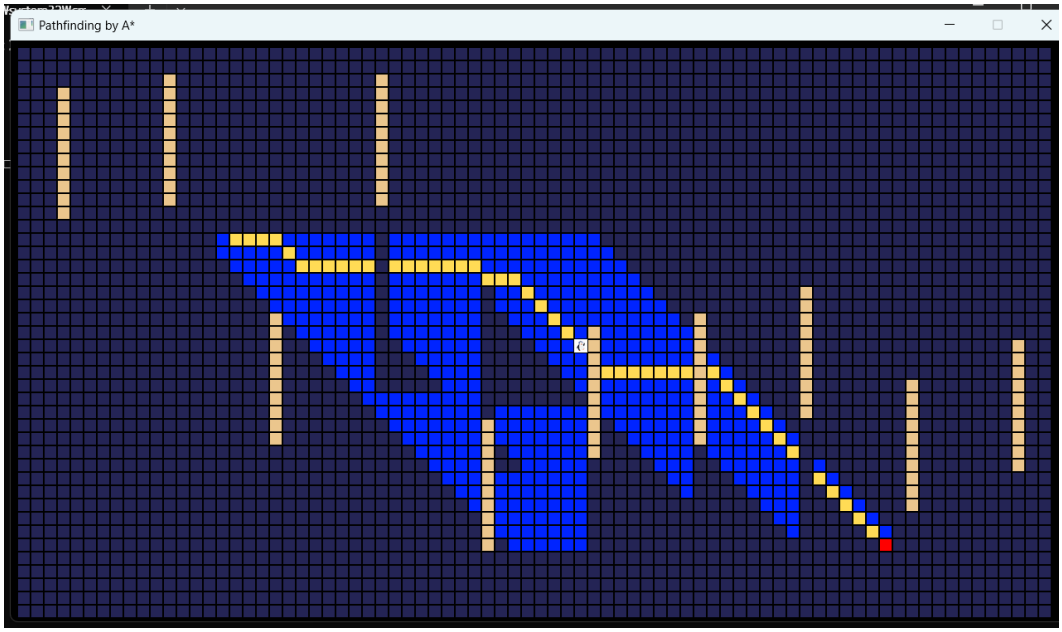
`if (_around_wall_flag == 1)` (목적지 사방이 벽이면 밑에 for 문 실행)

```
for (auto a = -2; a < 3; a++)
{
    for (auto b = -2; b < 3; b++)
    {
        if (0 == a && 0 == b
            || -1 == a && -1 == b
            || -1 == a && 0 == b
            || -1 == a && 1 == b
            || 0 == a && -1 == b
            || 0 == a && 1 == b
            || 1 == a && -1 == b
            || 1 == a && 0 == b
            || 1 == a && 1 == b)
        {
            continue;
        }
        gbl::MAP::Cell _type = get_cell(gbl::Position<short>(_cell.first + a, _cell.second + b), _map);
        // 벽이거나, 범위 밖이 아닌 경우 포함
        if (gbl::MAP::Cell::Invalid != _type && gbl::MAP::Cell::Wall1 != _type
            && gbl::MAP::Cell::Wall2 != _type && gbl::MAP::Cell::Wall7 != _type
            && gbl::MAP::Cell::Wall3 != _type && gbl::MAP::Cell::Wall8 != _type
            && gbl::MAP::Cell::Wall4 != _type && gbl::MAP::Cell::Wall9 != _type
            && gbl::MAP::Cell::Wall5 != _type && gbl::MAP::Cell::Wall6 != _type)
        {
            adjacent_cells.push_back(gbl::Position<>(a + _cell.first, b + _cell.second));
        }
    }
}
```

: 원래의 목적지에서는 자기 자신을 제외한 사방을 둘러싼 3x3 셀을 확인했는데 이번에는 ㅁ 모양을 갖고 있는 목적지에서는 모양까지 포함한 5x5 셀을 확인해야합니다. 그래서 범위를 -2 부터 2 까지 설정했고 ㅁ 모양에서의 경로 셀은 추가하지 않기 위해 ㅁ 모양 위에 있다면 셀을 adjacent_cells 에 push_back() 하지 않도록 건너뛰게 했습니다. 그게 아니라면 한 번 더 조건을 봐서 벽이거나 범위 밖이 아니면 adjacent_cells 에 push_back 해줍니다. 결과적으로 ㅁ 모양의 장애물에서 가운데를 목적지로 설정하면, 대체 목적지는 ㅁ 모양과 연결된 출발점에서 가장 가까운 빈 셀이 됩니다.

(3) 이동하는 장애물 회피 (sw.cpp :336~400, 1069~1072,955~958, 1055~1067)

<결과> 캐릭터 이동 중 장애물과 충돌하지 않고 장애물이 지나가기를 기다리는 장면 캡쳐



- 출발점은 좌상단, 도착점은 마우스 좌측버튼으로 설정, 이동 중 경로변경 가능은 위에서 했던 경우들과 동일하게 했습니다.

```
class Column_2 :public Obstacle {
public:
    Column_2() {}
    int k;
    vector<int> flag;
    vector<int>y_position;
    bool make_Obstacle(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {

        obstacle_cnt += 10;
        for (int i = 3; i < 77; i = i + 8)
        {
            y = rand() % 33;
            flag.push_back(0);
            y_position.push_back(y);
            _map[i][y] = gbl::MAP::Cell::Wall8;
            _map[i][y + 1] = gbl::MAP::Cell::Wall8;
            _map[i][y + 2] = gbl::MAP::Cell::Wall8;
            _map[i][y + 3] = gbl::MAP::Cell::Wall8;
            _map[i][y + 4] = gbl::MAP::Cell::Wall8;
            _map[i][y + 5] = gbl::MAP::Cell::Wall8;
            _map[i][y + 6] = gbl::MAP::Cell::Wall8;
            _map[i][y + 7] = gbl::MAP::Cell::Wall8;
            _map[i][y + 8] = gbl::MAP::Cell::Wall8;
            _map[i][y + 9] = gbl::MAP::Cell::Wall8;

            _data[i][y] = 1;
            _data[i][y + 1] = 1;
            _data[i][y + 2] = 1;
            _data[i][y + 3] = 1;
            _data[i][y + 4] = 1;
            _data[i][y + 5] = 1;
            _data[i][y + 6] = 1;
            _data[i][y + 7] = 1;
            _data[i][y + 8] = 1;
            _data[i][y + 9] = 1;
        }
        return 1;
    }
```

: Column_2 class는 수직 1자형 장애물을 만들기 위한 class입니다. Vector인 flag는 장애물이 맵의 상단이 닿았을 때 방향을 바꾸기 위한 vector입니다. 또 vector인 y_position은 계속 업데이트 되는 y의 좌표 값을 저장해주는 vector입니다. 업데이트되지 않는 y값을 유지한다면 움직이는 것을 볼 수 없기 때문에 선언했습니다. make_Obstacle 함수에서 3부터 76까지, 8씩 증가한다는 for문을 볼 수 있습니다. 이것은 x좌표를 셀의 x좌표 3부터 8 간격으로 폭 1셀, 길이 10셀인 장애물을 10개 만들기 위한 것입니다. 또한 장애물이 최소 가로 1셀 간격이상으로, 10개가 있어야 하기 때문에 8씩 증가하게 구현했습니다. For문 안에서 flag을 다 0으로 초기화하고 y_position에 y를 push_back() 합니다. 10번 반복하기 때문에 사이즈는 전부 10이 됩니다.

```
void move(int x, int y, gbl::Map<>& _map, vector<vector<int>>& _data) {
    k = 0;
    for (int i = 3; i < 77; i = i + 8, k++) {
        if (flag[k] == 0) {
            if (y_position[k] + 10 == 43) {
                flag[k] = 1;
            }
            else {
                _map[i][y_position[k]] = gbl::MAP::Cell::Empty;
                _map[i][y_position[k] + 10] = gbl::MAP::Cell::Wall8;
                y_position[k] = y_position[k] + 1;
            }
        }
        else if (flag[k] == 1) {
            if (y_position[k] - 1 == -1) {
                flag[k] = 0;
            }
            else {
                _map[i][y_position[k] + 9] = gbl::MAP::Cell::Empty;
                _map[i][y_position[k] - 1] = gbl::MAP::Cell::Wall8;
                y_position[k] = y_position[k] - 1;
            }
        }
    }
```

: Column_2 class 안에 move 함수는 수직 1자형 장애물이 상하로 왕복 운동하는 함수입니다. 변수 K는 10개의 장애물 flag를 관리하기 위한 변수입니다. 10개의 장애물을 움직이기 위해 위에서 설명한 것과 같이 8씩 증가하는 for문을 사용합니다. for문 안에서 장애물을 한 번에 관리하기 때문에 상하이동 속도는 고정되어 있습니다. Flag의 각 인덱스의 값은 0 아니면 1인데, 0이면 장애물이 내려가고 1이면 올라갑니다. flag가 0이면 현재 x, y의 위치가 Empty가 되고 장 x, y_position+10이 Wall8이 됩니다. flag가 1이면 반대로 적용되면 장애물이 위로 올라가게 됩니다. 그리고 각 상단의 위치에 도착하면 flag을 0이면 1로, 1이면 0으로 바꿔주기 때문에 왕복 운동이 가능합니다.

```
if (count % 2 == 0) {
    column_2.move(random_p.first, random_p.second, map, data);
    count = 0;
}
count++;
```

: 이 코드는 main문 안에 있습니다. Count를 0으로 초기화 시키고 count를 1씩 증가시킵니다. 만약 짝수면 장애물을 움직이고 다시 count를 0으로 하는 코드입니다. 의미하는 것은 원래 한 프레임에 장애물과 캐릭터 동시에 움직였는데 장애물을 두 번 카운트해서 두 프레임에 한 번 움직이게 했습니다. 이걸 통해서 캐릭터의 이동속도는 장애물 상하이동 속도보다 2배 이하로 빨라지게 됩니다.

```
if (mousePosition.first % 8 != 3) {
    if (event.mouseButton.button == sf::Mouse::Left) {
        leftbutton = 1;
        leftflag = 1;
```

: 이 코드는 main 문 안에 있습니다. 좌클릭할 때 x 좌표를 8로 나눴을 때 3이면 좌클릭이 안 된다는 것을 의미합니다. 지금 세로 10칸의 장애물은 x 좌표의 3부터 8간격으로 존재하고, 그 존재하는 장애물이 움직이는 곳을 클릭하지 않기 위해 구현했습니다. 그래서 목적지는 수직 1자형 장애물 위나, 그 경로 위에 설정되지 않도록 구현했습니다.

```
if (!path_cell_save.empty()) {
    path_position = path_cell_save.back();
    if (map[path_position.first][path_position.second] == gbl::MAP::Cell::Path
        || map[path_position.first][path_position.second] == gbl::MAP::Cell::Empty
        || map[path_position.first][path_position.second] == gbl::MAP::Cell::Visited)
    {
        start_position = path_position;
        path_cell_save.pop_back();
    }
    if (path_cell_save.empty()) {
        leftflag = 0;
    }
}
```

: 이 코드는 main문 안에 있고 astar을 통해 finished를 찾았을 때의 경우입니다. 경로를 다 찾고 난 후 path_position_save를 back() 하는 부분입니다. 만약 가야하는 그 경로가 Path거나 Empty거나 Visited이면 갈 수 있습니다. 그래서 start_position을 바꾸고 path_cell_sace를 pop_back() 할 수 있습니다. 또 장애물이 움직이는 그 수직 1자 셀들은 타입이 벽이기 때문에 이동할 수 없어서 기다리고 장애물이 지나가면 이동할 수 있습니다.

감사합니다.