코드설명

PACMAN

고성준

목차

GAMEMAP

GAMEMAP

- Char 2차원 배열을 사용하여 맵을 그리고 switch를 이용하여 그려지는 구성을 나눴다.

```
#include "GameMap.h"

GameMap::GameMap()
{
    char copyMap[Height][Width + 1] = {
        "############################" ,
        "#............#............#" ,
        "#.####.#####.#.#####.####.#" ,
        "#+####.#####.#.#####.####+#" ,
        "#.####.#####.#.#####.####.#" ,
        "#........................#" ,
        "#.####.##.######.##.####.#" ,
        "#.####.##.######.##.####.#" ,
        "#......##....#....##......#" ,
        "######.#### # ####.######" ,
        "     #.#### # ####.#     " ,
        "     #.##       ##.#     " ,
        "     #.## ##   ## ##.#     " ,
        "######.## #     # ##.######" ,
        "      .   #     #   .      " ,
        "######.## #     # ##.######" ,
        "     #.## ###### ##.#     " ,
        "     #.##       ##.#     " ,
        "     #.## ###### ##.#     " ,
        "######.## ###### ##.######" ,
        "#............#............#" ,
        "#.####.#####.#.#####.####.#" ,
        "#.####.#####.#.#####.####.#" ,
        "#+..##.......  .......##..+#" ,
        "###.##.##.######.##.##.###" ,
        "###.##.##....#....##.##.###" ,
        "#......#####.#.#####......#" ,
        "#.########.#.########.#" ,
        "#........................#" ,
        "############################" ,
    };
```

```
void GameMap::Draw()
{
    for (int i = 0; i < Height; i++)
    {
        for (int j = 0; j < Width; j++)
        {
            if (back_buffer[i][j] != front_buffer[i][j])
            {
                gotoxy(j * 2, i);
                switch (back_buffer[i][j])
                {
                    case p_POINT:
                        SetColor(DARKYELLOW, BLACK);
                        cout << ". ";
                        break;
                    case ITEM:
                        SetColor(DARKYELLOW, BLACK);
                        cout << "☆";
                        break;
                    case SPACE:
                        SetColor(WHITE, BLACK);
                        cout << "  ";
                        break;
                    case WALL:
                        SetColor(BLUE, BLACK);
                        cout << "■";
                        break;
                    case PLAYER:
                        SetColor(YELLOW, BLACK);
                        cout << "●";
                        break;
                    case ENEMY:
                        SetColor(RED, BLACK);
                        cout << "▲";
                        break;
                    case DIED_ENEMY:
                        SetColor(BLUE, BLACK);
                        cout << "▲";
                }
            }
```

PLAYER

PLAYER

- 키 입력은 GetAsyncKeyState 함수를 사용했으며 방향키에 따라 입력을 받는다.
- 받은 데이터에 따라서 플레이어가 움직일 위치가 정해진다.

```
void Player::InputKey()
{
    if (GetAsyncKeyState(VK_UP) & 0x8000)
        state = UP;
    else if (GetAsyncKeyState(VK_DOWN) & 0x8000)
        state = DOWN;
    else if (GetAsyncKeyState(VK_LEFT) & 0x8000)
        state = LEFT;
    else if (GetAsyncKeyState(VK_RIGHT) & 0x8000)
        state = RIGHT;
}
```

```
void Player::Update(GameMap* map)
{
    switch (state)
    {
    case UP:
        if (map->GetMap(x, y - 1) == WALL)
            break;
        y -= 1;
        break;
    case DOWN:
        if (map->GetMap(x, y + 1) == WALL)
            break;
        y += 1;
        break;
    case LEFT:
        if (x - 1 < 0)
            x = Width;
        if (map->GetMap(x - 1, y) == WALL)
            break;
        x -= 1;
        break;
    case RIGHT:
        if (x + 1 > Width - 1)
            x = 0;
        if (map->GetMap(x + 1, y) == WALL)
            break;
        x += 1;
        break;
    default:
        break;
    }
```

PLAYER

- Point를 먹었을 때나 아이템을 먹었을 때의 처리를 담당한다.
- Gotoxy 함수와 SetColor 함수를 정의하고 사용하여 좌표를 화면에 고정시킨 뒤 색상을 입혀 텍스트를 출력한다.

```
switch (map->GetItem(x, y))
{
case p_POINT:
    score += 10;
    break;
case ITEM:
    GetBuff();
    break;
}

buffEndTime = clock();
if (buffEndTime - buffStartTime > 5000)
    buff = false;

map->SetBuffer(x, y, PLAYER);
```

```
void Player::Draw()
{
    gotoxy(4, 31);
    SetColor(WHITE, BLACK);
    cout << "SCORE 『 " << score << " 』";

    gotoxy(30, 31);
    SetColor(WHITE, BLACK);
    if (buff == true)
        cout << "BUFF 『 ON 』";
    else
        cout << "BUFF 『 OFF 』";
}
```

```
static void gotoxy(int x, int y)
{
    COORD pos = { x,y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}
```

```
static void SetColor(int txt, int bg)
{
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), txt + bg * 16);
}
```

GHOST

GHOST

- FSM 머신을 사용하여 상황에 따른 행동을 통제한다.
- ChangeState 함수를 통해서 자신의 행동을 바꾼다.

```
#pragma once
#include "Player.h"
#include "AStar.h"
#include "Interfaces.h"

class Ghost
{
private:
    int x;
    int y;
    char state;
    bool isDie;
    IState* curState;

public:
    Ghost(int _x, int _y);
    ~Ghost();

    void Update(Player* player, GameMap* map);
    void SetPoint(int _x, int _y) { x = _x, y = _y; }
    Point GetPoint() { return { x,y }; }
    void ChangeState(IState* state);
    void SetDie(bool isdie) { isDie = isdie; }
    bool GetDie() { return isDie; }
};
```

GHOST

- FSM 머신을 사용하여 상황에 따른 행동을 통제한다.
- ChangeState 함수를 통해서 자신의 행동을 바꾼다.

```cpp
#pragma once
#include "Player.h"
#include "AStar.h"
#include "Interfaces.h"

class Ghost
{
private:
    int x;
    int y;
    char state;
    bool isDie;
    IState* curState;

public:
    Ghost(int _x, int _y);
    ~Ghost();

    void Update(Player* player, GameMap* map);
    void SetPoint(int _x, int _y) { x = _x, y = _y; }
    Point GetPoint() { return { x,y }; }
    void ChangeState(IState* state);
    void SetDie(bool isdie) { isDie = isdie; }
    bool GetDie() { return isDie; }
};
```

```cpp
void Ghost::Update(Player* player, GameMap* map)
{
    curState->Execute(this, player, map);

    if (player->GetX() == x && player->GetY() == y && !isDie)
    {
        if (player->IsBuff() == true)
            isDie = true;
        else
            player->SetIsDie(true);
    }

    map->SetBuffer((int)x, (int)y, !isDie ? ENEMY : DIED_ENEMY);
}

void Ghost::ChangeState(IState* state)
{
    if (curState != nullptr)
        curState->Exit(this);
    curState = state;
    curState->Enter(this);
}
```

FSM

FSM

- Istate 를 부모로 받아 가상함수 Enter, Execute, Exit 함수를 사용하여 행동을 통제하도록 한다.

```
class InBox : public IState
{
private:
    clock_t startTime;
    clock_t endTime;
public:
    virtual void Enter(Ghost* instance);
    virtual void Execute(Ghost* instance, Player* player, GameMap* map);
    virtual void Exit(Ghost* instance);
};

class Hunter : public IState
{
private:
    list<Point> playerPoint;

    int randNum;
    char state;
    bool isFind;
public:
    virtual void Enter(Ghost* instance);
    virtual void Execute(Ghost* instance, Player* player, GameMap* map);
    virtual void Exit(Ghost* instance);

    Hunter() { isFind = false; randNum = 0; state = IDLE; };
    ~Hunter() {};
};

class Hunted : public IState
{
private:
    Ghost* Instance;
    int randNum;
    char state;
public:
    virtual void Enter(Ghost* instance);
    virtual void Execute(Ghost* instance, Player* player, GameMap* map);
    virtual void Exit(Ghost* instance);
```

FSM

- InBox 상태에서는 스폰 지점 안에 있는 역할을 한다. 초기 상태로 설정되며 죽었을 때도 이 상태로 돌아간다.

- 적은 5초가 지나면 헌터 상태에 돌입하게 된다.

```
void InBox::Enter(Ghost* instance)
{
    startTime = clock();
}

void InBox::Execute(Ghost* instance, Player* player, GameMap* map)
{
    endTime = clock();
    if (endTime - startTime >= 5)
    {
        instance->ChangeState(hunter);
    }
}

void InBox::Exit(Ghost* instance)
{
    startTime = 0;
    endTime = 0;
}
```

FSM

- Hunter 상태에서는 4방향을 매 프레임 마다 무작위로 움직이게 된다.
- Astar 알고리즘을 사용해 플레이어와 거리가 10 걸음 이내 차이가 난다면 플레이어를 쫓아가게 된다. 이때 플레이어가 아이템을 먹은 상태라면 상태가 바뀐다.

```
void Hunter::Enter(Ghost* instance)
{
    srand((unsigned int)time(NULL));
    isFind = true;
}

void Hunter::Execute(Ghost* instance, Player* player, GameMap* map)
{
    AStar astar(map);
    playerPoint = astar.FindPath(instance->GetPoint().x, instance->GetPoint().y, player->GetX(), player->GetY());
    if (playerPoint.size() <= 10)
    {
        playerPoint.pop_front();
        instance->SetPoint(playerPoint.front().x, playerPoint.front().y);
    }
    else
    {
        randNum = rand() % 4 + 1;

        switch (randNum)
        {
        case 1: // UP
            if (map->GetMap(instance->GetPoint().x, instance->GetPoint().y + 1) == WALL)
                break;
            instance->SetPoint(instance->GetPoint().x, instance->GetPoint().y + 1);
            break;
        case 2: // DOWN
            if (map->GetMap(instance->GetPoint().x, instance->GetPoint().y - 1) == WALL)
                break;
            instance->SetPoint(instance->GetPoint().x, instance->GetPoint().y - 1);
            break;
        case 3: // RIGHT
            if (map->GetMap(instance->GetPoint().x + 1, instance->GetPoint().y) == WALL)
                break;
            instance->SetPoint(instance->GetPoint().x + 1, instance->GetPoint().y);
            break;
        case 4: // LEFT
            if (map->GetMap(instance->GetPoint().x - 1, instance->GetPoint().y) == WALL)
                break;
            instance->SetPoint(instance->GetPoint().x - 1, instance->GetPoint().y);
            break;
        }
    }

    if (player->IsBuff() == true)
        instance->ChangeState(hunted);

    if (instance->GetDie())
        instance->ChangeState(eaten);
}
```

FSM

- Hunted 상태에서는 Hunter 상태와 똑같이 움직이되 플레이어를 더 이상 쫓지 않는다.
- 플레이어의 아이템 버프가 꺼졌다면 다시 Hunter 상태로 돌입한다.
- Hunter 상태와 동일하게 플레이어에게 먹혔다면 Eaten 상태로 돌입한다.

```
void Hunted::Enter(Ghost* instance)
{
    srand((unsigned int)time(NULL));
    randNum = 0;
}

void Hunted::Execute(Ghost* instance, Player* player, GameMap* map)
{
    randNum = rand( ) % 4 + 1;

    int x = instance->GetPoint( ).x;
    int y = instance->GetPoint( ).y;
    switch (randNum)
    {
    case 1: // UP
        if (map->GetMap(x, y + 1) == WALL || map->GetMap(x, y + 1) == ENEMY)
            break;
        instance->SetPoint(x, y + 1);
        break;
    case 2: // DOWN
        if (map->GetMap(x, y - 1) == WALL || map->GetMap(x, y - 1) == ENEMY)
            break;
        instance->SetPoint(x, y - 1);
        break;
    case 3: // RIGHT
        if (map->GetMap(x + 1, y) == WALL || map->GetMap(x + 1, y) == ENEMY)
            break;
        instance->SetPoint(x + 1, y);
        break;
    case 4: // LEFT
        if (map->GetMap(x - 1, y) == WALL || map->GetMap(x - 1, y) == ENEMY)
            break;
        instance->SetPoint(x - 1, y);
        break;
    }

    if (player->IsBuff( ) == false)
        instance->ChangeState(hunter);

    if (instance->GetDie( ))
        instance->ChangeState(eaten);
}
```

FSM

- Eaten 상태에선 5초의 대기시간을 가지고 스폰지점으로 이동하며 다시 InBox 상태로 돌입한다.

```
void Eaten::Enter(Ghost* instance)
{
    startTime = clock();
    isDie = false;
}

void Eaten::Execute(Ghost* instance, Player* player, GameMap* map)
{
    endTime = clock();
    if (endTime - startTime >= 5)
    {
        if (map->GetMap(12, 14) != ENEMY)
            instance->SetPoint(12, 14);
        else if (map->GetMap(13, 14) != ENEMY)
            instance->SetPoint(13, 14);
        else if (map->GetMap(12, 15) != ENEMY)
            instance->SetPoint(12, 15);
        else if (map->GetMap(13, 15) != ENEMY)
            instance->SetPoint(13, 15);

        instance->SetDie(isDie);
        instance->ChangeState(pinbox);
    }
}

void Eaten::Exit(Ghost* instance)
{
    startTime = 0;
    endTime = 0;
}
```

ASTAR

ASTAR

- FindPath 함수이다.
- 자신의 위치 기준 8방향을 탐색하여 목표 위치까지의 거리를 알기 위해 가중치를 계산한다.
- 탐색 도중 벽인 곳은 제외하며 향후 목표까지의 노드를 반환한다.

```
    int width = m_gameMap->GetWidth();
    int height = m_gameMap->GetHeight();

    Point choicePos;
    Point targetPos = { dx, dy };
    m_map[sy][sx] = 0;
    choicePos = { sx, sy };

    for (int i = 0; i < width * height; ++i)
    {
        m_visitInfo[choicePos.y][choicePos.x] = true;
        m_visitNode.push_back(choicePos);

        // 목표 노드를 만나면 종료
        if (choicePos.x == dx && choicePos.y == dy)
        {
            m_bFound = true;
            break;
        }

        // 엣지 완화 알고리즘 적용
        for (int ty = -1; ty <= 1; ++ty)
        {
            for (int tx = -1; tx <= 1; ++tx)
            {
                int nx = choicePos.x + tx;
                int ny = choicePos.y + ty;
                int dist;
                if (nx < 0 || nx > width - 1 || ny < 0 || ny > height - 1)
                    continue;

                if (m_gameMap->GetMap(nx, ny) == WALL)
                {
                    m_visitInfo[ny][nx] = true;
                    continue;
                }

                if (m_visitInfo[ny][nx] == false)
                {
                    dist = (tx == 0 || ty == 0) ? 10 : 14;

                    // 엣지 완화 처리
                    if (m_map[ny][nx] > m_map[choicePos.y][choicePos.x] + dist)
                    {
                        int tCost = m_map[choicePos.x][choicePos.y] + dist;
                        m_map[ny][nx] = tCost;
                    }
                }
            }
        }

        ExtractMin(choicePos, targetPos);

    }

    if (m_bFound)
    {
        return m_visitNode;
    }
}
```

ASTAR

- ExtractMin 함수이다.
- 자신의 위치 기준 8방향을 탐색하여 목표 위치까지의 거리를 가중치를 사용하여 계산한다.

```
void AStar::ExtractMin(Point& choicePos, Point& targetPos)
{
    int min = INT_MAX;

    int curX, curY;
    list<Point>::reverse_iterator curPos;

    for (curPos = m_visitNode.rbegin(); curPos != m_visitNode.rend(); curPos++)
    {
        // 현재 노드를 기준으로 8방향을 스캐닝
        for (int i = -1; i <= 1; ++i)
        {
            for (int j = -1; j <= 1; ++j)
            {
                curX = curPos->x + j;
                curY = curPos->y + i;
                if (curX < 0 || curX > Width - 1 || curY < 0 || curY > Height - 1 || (i == 0 && j == 0))
                    continue;

                if (m_gameMap->GetMap(curX, curY) == WALL)
                    continue;

                int hx = abs(targetPos.x - curX) * 10;
                int hy = abs(targetPos.y - curY) * 10;
                int hDist = hx + hy;

                if (m_map[curY][curX] + hDist < min && m_visitInfo[curY][curX] == false)
                {
                    min = m_map[curY][curX] + hDist;
                    choicePos = { curX, curY };
                }
            }
        }
    }
}
```

감사합니다!