안승모

DirectX 포트폴리오

목차

소개 3

기본 목표 4

구현 내용 5

결론 19

3 소개

매직크래프트

목적: DirectX 프로그래밍 경험
장르 : 로그라이크, 매직크래프트
사용도구: VisualStudio, DirectX

기본 목표

- **DirectX 환경에서 개발**

상용 게임 엔진을 사용하지 않는 DiretX 를 사용하여 개발

- **자동 지형 생성 개발**

로그라이크 게임들에서 사용되는 지형 생성 알고리즘을 직접 적용하고 구현

- **오토 타일링 적용**

개임 개발에서 흔히 사용되는 오토 타일링을 직접 구현하고 개발

- **스펠 커스텀 시스템 개발**

RPG에서 사용되는 인벤토리 시스템을 응용하여 스펠 커스텀 시스템을 개발

자동 지형 생성

1 방 배치 알고리즘

- 초기 방 배치

```
int centerX = width / 2;
int centerY = height / 2;
// 첫 방 중앙에 배치
Room first = { centerX - ROOM_MIN_SIZE / 2, centerY - ROOM_MIN_SIZE / 2, ROOM_MIN_SIZE, ROOM_MIN_SIZE };
rooms.push_back(first);
CarveRoom(first, 0);
```

- 겹침 방지

```
// 겹침 체크
bool overlap = false;
for (size_t j = 0; j < rooms.size(); ++j) {
    if (IsOverlap(newRoom, rooms[j])) {
        overlap = true;
        break;
    }
}
// 실제 겹치는 구간이 3칸 이상인지 확인
if (!overlap) {
    int overlapW = OverlapLen(base.x, base.x + base.w - 1, newRoom.x, newRoom.x + newRoom.w - 1);
    int overlapH = OverlapLen(base.y, base.y + base.h - 1, newRoom.y, newRoom.y + newRoom.h - 1);
    bool valid = false;
    if (dir == 0 || dir == 1) valid = (overlapW >= MIN_OVERLAP);
    else if (dir == 2 || dir == 3) valid = (overlapH >= MIN_OVERLAP);
    else valid = (overlapW >= MIN_OVERLAP && overlapH >= MIN_OVERLAP);
    if (!valid) continue;
    rooms.push_back(newRoom);
    CarveRoom(newRoom, rooms.size() - 1);
    MergeBoundary(base, newRoom, dir);
    placed = true;
}
```

- 방 간 연결 생성

```
void MapGenerator::MergeBoundary(const Room& a, const Room& b, int dir) {
    // a와 b가 붙은 경계선을 바닥으로 강제
    int ax1 = a.x, ay1 = a.y, ax2 = a.x + a.w - 1, ay2 = a.y + a.h - 1;
    int bx1 = b.x, by1 = b.y, bx2 = b.x + b.w - 1, by2 = b.y + b.h - 1;
    vector<pair<int, int>> boundary_coords;
    if (dir == 0) { // 위
        int sx = max(ax1, bx1);
        int ex = min(ax2, bx2);
        int y = ay1;
        for (int x = sx; x <= ex; ++x) {
            map[y][x] = FLOOR;
            boundary_coords.push_back({ x, y });
        }
    }
    else if (dir == 1) { // 아래
        int sx = max(ax1, bx1);
        int ex = min(ax2, bx2);
        int y = ay2;
        for (int x = sx; x <= ex; ++x) {
            map[y][x] = FLOOR;
            boundary_coords.push_back({ x, y });
        }
    }
    else if (dir == 2) { // 왼쪽
        int sy = max(ay1, by1);
        int ey = min(ay2, by2);
        int x = ax1;
        for (int y = sy; y <= ey; ++y) {
            map[y][x] = FLOOR;
            boundary_coords.push_back({ x, y });
        }
    }
    else if (dir == 3) { // 오른쪽
        int sy = max(ay1, by1);
        int ey = min(ay2, by2);
        int x = ax2;
        for (int y = sy; y <= ey; ++y) {
            map[y][x] = FLOOR;
            boundary_coords.push_back({ x, y });
        }
    }
    else { // 대각선
        int sx = max(ax1, bx1);
        int ex = min(ax2, bx2);
        int sy = max(ay1, by1);
        int ey = min(ay2, by2);
        for (int y = sy; y <= ey; ++y) {
            for (int x = sx; x <= ex; ++x) {
                map[y][x] = FLOOR;
                boundary_coords.push_back({ x, y });
            }
        }
    }
    // b(새 방)의 연결 경계 좌표 저장
    int b_idx = rooms.size() - 1;
    room_connections[b_idx].insert(room_connections[b_idx].end(), boundary_coords.begin(), boundary_coords.end());
}
```

1 방 내부 생성 알고리즘(시작방)

- 모든 타일을 바닥으로 설정

```
// 시작 방은 모든 타일이 바닥
if (room_id == 0) {
    for (int y = 0; y < rh; ++y)
        for (int x = 0; x < rw; ++x)
            local[y][x] = FLOOR;
}
```

- 중앙에 플레이어 스폰 배치

```
// 플레이어 스포너 배치(시작방)
if (room_id == 0) {
    local[rh / 2][rw / 2] = PLAYER_SPAWN;
}
```

1 방 내부 생성 알고리즘(일반방)

- 연결 경계 주변 5x5 영역을 바닥으로 고정

```
// 일반 방은 기존 로직
else {
    // 1. 연결 경계(문) 위치에 6x6 바닥 컴포넌트 생성 (최우선)
    for (const auto& p : room_connections[room_id]) {
        int sx = p.first - room.x;
        int sy = p.second - room.y;
        for (int dy = -2; dy <= 2; ++dy)
            for (int dx = -2; dx <= 2; ++dx) {
                int ny = sy + dy, nx = sx + dx;
                if (0 <= ny && ny < rh && 0 <= nx && nx < rw)
                    local[ny][nx] = FLOOR;
            }
    }
```

- 셀룰러 오토마타로 바닥/물 타일 랜덤 배치

```
// 2. 셀룰러 오토마타 적용 (연결 경계는 보존)
uniform_int_distribution<int> cellDist(0, 99);
for (int y = 0; y < rh; ++y) {
    for (int x = 0; x < rw; ++x) {
        if (local[y][x] != FLOOR) { // 이미 바닥인 곳은 유지
            // 연결 경계 영역인지 확인
            bool isConnectionArea = false;
            for (const auto& p : room_connections[room_id]) {
                int sx = p.first - room.x;
                int sy = p.second - room.y;
                if (abs(y - sy) <= 2 && abs(x - sx) <= 2) {
                    isConnectionArea = true;
                    break;
                }
            }
            if (!isConnectionArea) {
                local[y][x] = (cellDist(rng) < 45) ? FLOOR : WATER;
            }
        }
    }
}

vector<vector<int>> next = local;
for (int y = 1; y < rh - 1; ++y) {
    for (int x = 1; x < rw - 1; ++x) {
        // 연결 경계 영역은 셀룰러 오토마타에서 제외
        bool isConnectionArea = false;
        for (const auto& p : room_connections[room_id]) {
            int sx = p.first - room.x;
            int sy = p.second - room.y;
            if (abs(y - sy) <= 2 && abs(x - sx) <= 2) {
                isConnectionArea = true;
                break;
            }
        }

        if (!isConnectionArea) {
            int cnt = 0;
            for (int dy = -1; dy <= 1; ++dy)
                for (int dx = -1; dx <= 1; ++dx)
                    if (local[y + dy][x + dx] == FLOOR) ++cnt;
            if (cnt >= 6) next[y][x] = FLOOR;
            else next[y][x] = WATER;
        }
    }
}
local = next;
```

1 방 내부 생성 알고리즘(보스방)

- 미리 정의된
보스 레이아웃을 사용

```
// 보스 방 미리 준비된 배치들
vector<vector<vector<int>>> bossLayouts = {
    // 패턴 1: 중앙 원형 공간 (연결 경계 주변에 바닥 보장)
    {
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,2,2,2,2,2,0,0,0,0,2,2,2,2,2,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,2,2,2,2,2,0,0,0,0,2,2,2,2,2,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
    },
    // 패턴 2: 십자형 공간 (연결 경계 주변에 바닥 보장)
    {
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,2,2,2,2,0,0,0,0,0,0,0,2,2,2,2,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,2,2,2,2,0,0,0,0,0,0,0,2,2,2,2,0,0},
        {0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
    },
```

1 방 내부 생성 알고리즘(결과물)

오토타일링

12 오토 타일링

- 방 내부 생성 알고리즘에서 얻은 타일 데이터기준 근처 타일의 상태를 확인

```
int GameMap::NeighTileData(int index)
{
    int data = 0;
    int checkData = 0b00000001;

    Vector2 pos = CalTilePos(index);
    Tile::State state = tileDatas.at(index)->state;

    vector<Vector2> checkPos = {
        {0,-1},
        {0,1},
        {-1,0},
        {1,0},
        {-1,-1},
        {1,-1},
        {-1,1},
        {1,1}
    };

    for (int i = 0;i < checkPos.size();i++) {
        Vector2 target = pos + checkPos.at(i);
        bool check = target.x < 0 || target.x >= tileCount.x || target.y < 0 || target.y >= tileCount.y;
        if (check || tileDatas.at(CalTilePos(target))->state == state)
            data |= checkData;
        checkData <<= 1;

    }
    return data;
}
```

13 오토 타일링

- 해당 타일 1개를 4분할 그리고 인근 분할된 타일을 기록

```
void Tile::CalTileData(int data) {
    int type = 0;
    int quaterData[4][4] = {0,};
    quaterData[1][1] = 1;
    quaterData[1][2] = 1;
    quaterData[2][1] = 1;
    quaterData[2][2] = 1;

    if (data & 0b00000001) {
        quaterData[1][0] = 1;
        quaterData[2][0] = 1;
    }
    if (data & 0b00000010) {
        quaterData[1][3] = 1;
        quaterData[2][3] = 1;
    }
    if (data & 0b00000100) {
        quaterData[0][1] = 1;
        quaterData[0][2] = 1;
    }
    if (data & 0b00001000) {
        quaterData[3][1] = 1;
        quaterData[3][2] = 1;
    }
    if (data & 0b00010000) {
        quaterData[0][0] = 1;
    }
    if (data & 0b00100000) {
        quaterData[3][0] = 1;
    }
    if (data & 0b01000000) {
        quaterData[0][3] = 1;
    }
    if (data & 0b10000000) {
        quaterData[3][3] = 1;
    }
```

```
for (int i = 0;i < 4;i++) {
    int checkData = 0;
    int check = 0b00000001;
    for (int j = 0;j < checkPos.size();j++) {
        Vector2 target = pos.at(i) + checkPos.at(j);
        if (quaterData[(int)target.x][(int)target.y]) {
            checkData |= check;
        }
        check <<= 1;
    }
    quaterTileShapeData[i] = checkData;
}
```

14 오토 타일링

- 분할된 타일의 근처 타일을 기준으로 타일의 형태를 결정

```
void Tile::CalTilesetPos()
{
    if (state == WALL) {
        for (int i = 0; i < 4; i++) {

            try {
                int data = quaterTileShapeData.at(i);
                auto it = wallShapePos.find(data);
                if (it == wallShapePos.end()) {
                    data &= 0b1111;
                }
                quaterTileShape.at(i) = wallShapePos.at(data);

            }
            catch (const std::out_of_range& oor){
                quaterTileShape.at(i) = wallShapePos.at(0);
            }
        }
    }
    else if (state == WATER) {
        for (int i = 0; i < 4; i++) {

            try {
                int data = quaterTileShapeData.at(i);
                auto it = waterShapePos.find(data);
                if (it == waterShapePos.end()) {
                    data &= 0b1111;
                }
                quaterTileShape.at(i) = waterShapePos.at(data);

            }
            catch (const std::out_of_range& oor) {
                quaterTileShape.at(i) = waterShapePos.at(0);
            }
        }
    }

    else {
        for (int i = 0;i < 4;i++)
            quaterTileShape.at(i) = floorShapePos.at(0);
    }
}
```

15 오토 타일링

- 분할된 타일의 근처 타일을 기준으로 타일의 형태를 결정

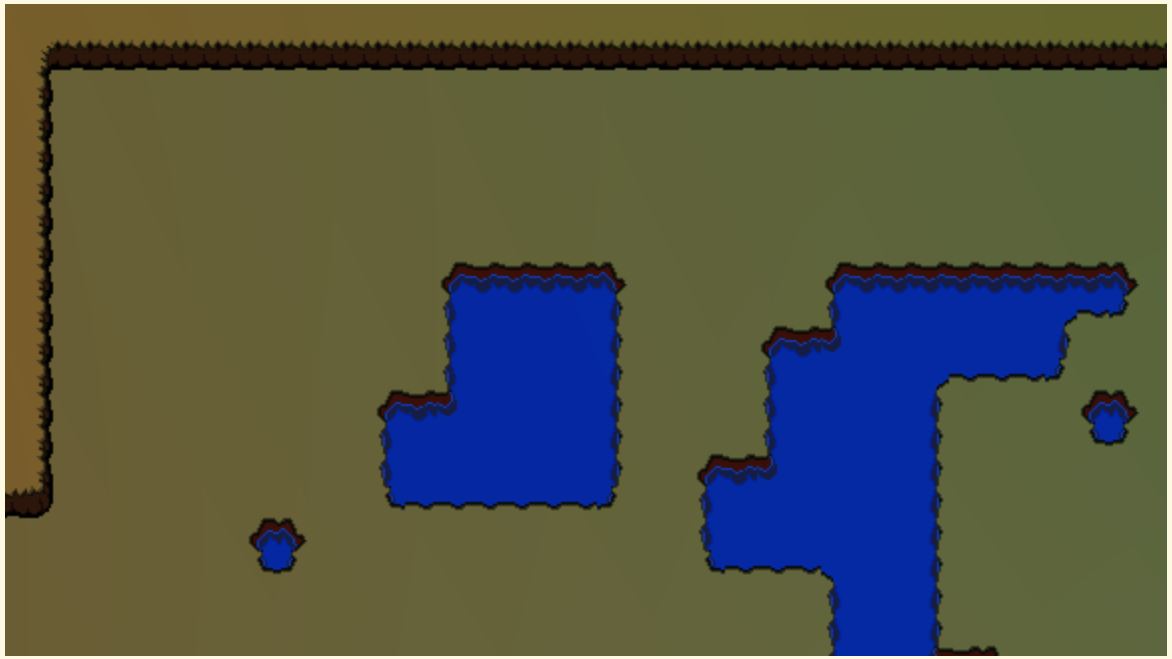
```
void Tile::CalTilesetPos()
{
    if (state == WALL) {
        for (int i = 0; i < 4; i++) {

            try {
                int data = quaterTileShapeData.at(i);
                auto it = wallShapePos.find(data);
                if (it == wallShapePos.end()) {
                    data &= 0b1111;
                }
                quaterTileShape.at(i) = wallShapePos.at(data);

            }
            catch (const std::out_of_range& oor){
                quaterTileShape.at(i) = wallShapePos.at(0);
            }
        }
    }
    else if (state == WATER) {
        for (int i = 0; i < 4; i++) {

            try {
                int data = quaterTileShapeData.at(i);
                auto it = waterShapePos.find(data);
                if (it == waterShapePos.end()) {
                    data &= 0b1111;
                }
                quaterTileShape.at(i) = waterShapePos.at(data);

            }
            catch (const std::out_of_range& oor) {
                quaterTileShape.at(i) = waterShapePos.at(0);
            }
        }
    }

    else {
        for (int i = 0;i < 4;i++)
            quaterTileShape.at(i) = floorShapePos.at(0);
    }
}
```

16 오토 타일링

- 타일맵 인스턴싱 적용

```
void GameMap::SetInstanceBuffer(vector<Tile*> tiles, vector<InstanceData>& instances ,VertexBuffer*& buffer)
{

    int i = 0;
    int j = 0;
    for (InstanceData& instance : instances)
    {
        if (j >= 4) {
            i++;
            j = 0;
        }
        float x = tiles.at(i)->GetQuaterTilePos(j).x;
        float y = tiles.at(i)->GetQuaterTilePos(j).y;
        float z = tiles.at(i)->GetZPos();
        Matrix world = XMMatrixTranslation(x, y, z);

        instance.world = XMMatrixTranspose(world);
        instance.maxFrame = Float2(30, 16);

        TileData* data = tileDatas.at(CalPosToIndex(tiles.at(i)->GetGlobalPosition()));
        Vector2 biomePos = data->biomePos.at(data->biome);

        float frameX = tiles.at(i)->GetQuaterTileShape(j).x;
        float frameY = tiles.at(i)->GetQuaterTileShape(j).y;
        instance.curFrame = Float2(frameX, frameY);
        j++;

    }
    buffer = new VertexBuffer(instances.data(), sizeof(InstanceData), instances.size());

}

void GameMap::SetInstanceBuffers()
{

    floorInstances.resize(floors.size() * 4);
    SetInstanceBuffer(floors, floorInstances, floorInstanceBuffer);

    objectInstances.resize(objects.size() * 4);
    SetInstanceBuffer(objects, objectInstances, objectInstanceBuffer);

}
```

17 오토 타일링



18 스펠 커스텀 시스템


스펠 옵션의 정보
원형 UI
피어싱
몬스터를 관통하며 지난간다.
호밍
가까운 적을 자동으로 추적한다.
복제
투사체를 복제한다.
사이즈업
투사체의 크기가 증가한다.
반사
적에게 충돌하고 반사된다.
인벤토리 시스템

19 소감

구현 목표사항을 전부 달성

기능 구형에만 신경쓴 나머지 게임의 완성도가 낮다.
개발을위한 리소스 확보에 너무 많은 시간이 할당되었다.

감사합니다.

안승