[ 게임 클라이언트 응용프로그래밍 ]

**지형 프로그래밍**

인하대학교 미래인재개발원 문화콘텐츠프로그래밍

김예슬

**지형 관리 클래스 소스 코드**

#include "stdafx.h"

#include "MapClass.h"

#include "PlayerClass.h"

#include "GameManager.h"

#include "UIClass.h"

#include "SoundSystem.h"

#include <fstream>

#include <string>

#define dGameManager GameManager::GetInstance()

#define dPlayer Player::GetInstance()

#define dUI UI::GetInstance()

#define dSoundSys SoundSystem::GetInstance()

#define dKeyCode 'k'

using namespace std;

Map::Map()

{

hMapBitmap = (HBITMAP)LoadImage(NULL, TEXT("Image/tile.bmp"), IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE | LR\_CREATEDIBSECTION);

GetObject(hMapBitmap, sizeof(BITMAP), &mapBitmap);

}

Map::~Map()

{

DeleteObject(hMapBitmap);

}

Map\* Map::GetInstance()

{

static Map map;

return &map;

}

void Map::Update()

{

if (!dPlayer->GetIsPlayerDead())

{

RECT area;

RECT playerPos = dPlayer->GetPlayerPos();

for (int i = 0; i < mapPos.size(); i++)

{

if (PlayerGetItem())

break;

if (mapPos[i].type == eMapSpike && IntersectRect(&area, &mapPos[i].pos, &playerPos))

{

dPlayer->SetIsPlayerDead(true);

break;

}

if (mapPos[i].type == eMapBtn\_0 && IntersectRect(&area, &mapPos[i].pos, &playerPos))

{

mapPos[i].type = eMapBtn\_1;

for (int j = 0; j < mapPos.size(); j++)

{

if (mapPos[j].pos.top == mapPos[i].pos.bottom && mapPos[j].pos.left == mapPos[i].pos.left && mapPos[j].pos.right == mapPos[i].pos.right

&& mapPos[j].type == eMapBtn\_2)

mapPos[j].type = eMapBtn\_3;

}

dSoundSys->PlayBtnOffSound();

}

if (mapPos[i].type == eMapGateOpen && IntersectRect(&area, &mapPos[i].pos, &playerPos))

isNextStage = true;

}

if (CheckOffBtn())

{

for (int i = 0; i < mapPos.size(); i++)

{

if (mapPos[i].type == eMapGate\_0 || mapPos[i].type == eMapGate\_1 || mapPos[i].type == eMapGate\_2 || mapPos[i].type == eMapGate\_3)

{

if(mapPos[i].type==eMapGate\_0) dSoundSys->PlayGateBreakSound();

mapPos[i].type = eMapGateOpen;

}

}

}

}

}

bool Map::CheckOffBtn()

{

for (int i = 0; i < mapPos.size(); i++)

{

if (mapPos[i].type == eMapBtn\_2)

return false;

}

return true;

}

void Map::CheckShotOffBtn(const RECT &hitPos)

{

RECT area;

bool isOff;

for (int i = 0; i < mapPos.size(); i++)

{

isOff = true;

if (IntersectRect(&area, &mapPos[i].pos, &hitPos) && mapPos[i].type == eMapBtn\_2)

{

for (int j = 0; j < mapPos.size(); j++)

{

if (mapPos[i].pos.top == mapPos[j].pos.bottom && mapPos[i].pos.left == mapPos[j].pos.left && mapPos[i].pos.right == mapPos[j].pos.right)

isOff = false;

}

if (isOff)

mapPos[i].type = eMapBtn\_3;

}

}

}

void Map::DrawObject(HDC hdc)

{

if (dGameManager->GetDrawRect())

{

for (int i = 0; i<mapPos.size(); i++)

Rectangle(hdc, mapPos[i].pos.left, mapPos[i].pos.top, mapPos[i].pos.right, mapPos[i].pos.bottom);

}

}

void Map::RenderObject(HWND hWnd, HDC hdc)

{

HDC hMapDc;

HBITMAP hMapBit;

POINT blength = { 16,16 };

POINT pos = { 0,0 };

hMapDc = CreateCompatibleDC(hdc);

hMapBit = (HBITMAP)SelectObject(hMapDc, hMapBitmap);

for (int i = 0; i < mapPos.size(); i++)

{

if (mapPos[i].type == eMapBlock)

pos = { 0,0 };

else if (mapPos[i].type == eMapHalfBlock)

pos = { 16,0 };

else if (mapPos[i].type == eMapSpike)

pos = { 0,16 };

else if (mapPos[i].type == eMapBtn\_0)

pos = { 0,32 };

else if (mapPos[i].type == eMapBtn\_1)

pos = { 16,32 };

else if (mapPos[i].type == eMapBtn\_2)

pos = { 0,48 };

else if (mapPos[i].type == eMapBtn\_3)

pos = { 16,48 };

else if (mapPos[i].type == eMapGate\_0)

pos = { 0,64 };

else if (mapPos[i].type == eMapGate\_1)

pos = { 16,64 };

else if (mapPos[i].type == eMapGate\_2)

pos = { 32,64 };

else if (mapPos[i].type == eMapGate\_3)

pos = { 48,64 };

else if (mapPos[i].type == eMapGateOpen)

pos = { 64,64 };

else if (mapPos[i].type == eMapCannon\_0)

pos = { 0,80 };

else if (mapPos[i].type == eMapCannon\_1)

pos = { 16,80 };

else if (mapPos[i].type == eMapCannon\_2)

pos = { 32,80 };

else if (mapPos[i].type == eMapCannon\_3)

pos = { 48,80 };

else if (mapPos[i].type == eMapCannon\_4)

pos = { 0,96 };

else if (mapPos[i].type == eMapCannon\_5)

pos = { 16,96 };

else if (mapPos[i].type == eMapCannon\_6)

pos = { 32,96 };

else if (mapPos[i].type == eMapCannon\_7)

pos = { 48,96 };

else if (mapPos[i].type == eMapGateCloseVertical)

pos = { 0,112 };

else if (mapPos[i].type == eMapGateCloseHorizen)

pos = { 16,112 };

else if (mapPos[i].type == eMapItem)

pos = { 0,128 };

TransparentBlt(hdc, mapPos[i].pos.left, mapPos[i].pos.top, blength.x, blength.y, hMapDc, pos.x, pos.y, blength.x, blength.y, RGB(255, 0, 255));

}

SelectObject(hMapDc, hMapBit);

DeleteObject(hMapBit);

DeleteDC(hMapDc);

}

void Map::SetNextStage()

{

if (dGameManager->GetNowStage() == -1)

dUI->SetIsGoMain(false);

if (dUI->GetIsGoMain() == false)

dGameManager->SetNowStage(dGameManager->GetNowStage() + 1);

else

{

dGameManager->SetNowStage(dGameManager->GetNowStage());

dUI->SetIsGoMain(false);

}

mapPos.clear();

resetPos.clear();

ReadMapData();

}

vector<TileMap> Map::GetMapPos()

{

return mapPos;

}

vector<ParceCannonStruct> Map::CheckInCannon()

{

ParceCannonStruct tempVal;

vector<ParceCannonStruct> result;

vector<TileMap>::iterator it;

for (it = mapPos.begin(); it < mapPos.end(); it++)

{

if (it->type == eMapCannon\_0)

{

tempVal.pos = { it->pos.right, it->pos.bottom };

tempVal.type = dNormal;

result.push\_back(tempVal);

}

if (it->type == eMapCannon\_4)

{

tempVal.pos = { it->pos.right, it->pos.bottom };

tempVal.type = dHoming;

result.push\_back(tempVal);

}

}

return result;

}

void Map::Reset()

{

mapPos = resetPos;

}

void Map::ReadMapData()

{

string fileName;

ifstream mapFile;

if (dGameManager->GetNowStage() < 10)

fileName = "map\_0";

else

fileName = "map\_";

fileName += to\_string(dGameManager->GetNowStage()) + ".dat";

fileName = "./Map/" + fileName;

mapFile.open(fileName, ios::in | ios::binary);

if (mapFile.is\_open())

{

TileMap tileMap;

tileMap.pos = { 0,0,0,0 };

tileMap.type = 0;

mapFile.read((char\*)&resenSpot.x, sizeof(int));

mapFile.read((char\*)&resenSpot.y, sizeof(int));

resenSpot.x = (resenSpot.x / dKeyCode) - dKeyCode;

resenSpot.y = (resenSpot.y / dKeyCode) - dKeyCode;

while (!mapFile.eof())

{

mapFile.read((char\*)&tileMap.type, sizeof(int));

mapFile.read((char\*)&tileMap.pos.left, sizeof(int));

mapFile.read((char\*)&tileMap.pos.top, sizeof(int));

mapFile.read((char\*)&tileMap.pos.right, sizeof(int));

mapFile.read((char\*)&tileMap.pos.bottom, sizeof(int));

tileMap.type = (tileMap.type / dKeyCode) - dKeyCode;

tileMap.pos.left = (tileMap.pos.left / dKeyCode) - dKeyCode;

tileMap.pos.top = (tileMap.pos.top / dKeyCode) - dKeyCode;

tileMap.pos.right = (tileMap.pos.right / dKeyCode) - dKeyCode;

tileMap.pos.bottom = (tileMap.pos.bottom / dKeyCode) - dKeyCode;

if (tileMap.type <= 0)

break;

mapPos.push\_back(tileMap);

}

dSoundSys->PlayChangeStageSound();

}

else

{

dGameManager->SetNowStage(-1);

dGameManager->SetFocusLv(0);

dGameManager->SetNowScene(eResultScene);

}

mapFile.close();

resetPos = mapPos;

// >> resetValue

isNextStage = false;

}

POINT Map::GetResenSpot()

{

return resenSpot;

}

void Map::SetIsNextStage(bool set)

{

isNextStage = set;

}

bool Map::GetIsNextStage()

{

return isNextStage;

}

bool Map::PlayerGetItem()

{

RECT area;

for (int i = 0; i < mapPos.size(); i++)

{

if (mapPos[i].type == eMapItem && IntersectRect(&area, &mapPos[i].pos, &dPlayer->GetPlayerPos()))

{

dSoundSys->PlayGetItemSound();

dPlayer->SetFocusLv();

mapPos.erase(mapPos.begin() + i);

return true;

}

}

return false;

}