

[illegible]

-  _____
-  _____



https://www.youtube.com/embed/OoSlnw4h57U?si=FU4Sz5LdvoTB_84Q





```
#include "Enemy.h"
#include "./Stage.h"
#include "globals.h"
#include "Player.h"
#include <map>
#include <queue>
#include "ImGui/ImGui.h"
#include "BombFire.h"

namespace
{
    □const Point nDir[4] = { {0,-1},{0,1},{-1,0},{1,0} };
    □const float SPEED = 0.0f;
    □const int NEIGHBOURS = 9;
    □const Point nineNeibor[NEIGHBOURS] = { {0,0}, {1,0}, {0,1}, {1,1}, {-1,0}, {0,-1}, {-1,-1}, {1,-1}, {-1,1} };

    □const float ANIM_INTERVAL = 0.3f;
    □const float DEATH_ANIM_FRAME = 3.0f;
    □const int frameNum[4] = { 0, 1, 2, 1 };
    □const int yTerm[5] = { 3, 0, 1, 2, 0 };

    □const Pointf INIT_POS{ 5, 5 };
    □const DIR INIT_DIR = UP;
    □bool isGraphic = true;
    □enum ENEMY_TYPE
    □{
        □□OTAKU,
        □□NEKO,
        □□KABOCHA,
        □□OBAKE,
        □□ENEMY_TYPE_MAX
    □};
    □std::string enemyImagePlace[ENEMY_TYPE_MAX] = { "Assets/otaku.png",
        □□"Assets/neko.png",
        □□"Assets/kabocha.png",
```

```

    "Assets/obakeb.png" };

std::string GetEnemyImage(ENEMY_TYPE type)
{
    return enemyImagePlace[type];
}

Enemy::Enemy()
: pos_({ 0,0 }), isAlive_(true), nextPos_({ 0,0 }),
  animFrame_(0), animTimer_(0), enemyImage_(-1), isHitWall_(false), speed_(SPEED)
{
    //
    //int rx = 0;
    //int ry = 0;
    //while (rx % 2 == 0 || ry % 2 == 0)
    //{
    //    rx = GetRand(STAGE_WIDTH - 1);
    //    ry = GetRand(STAGE_HEIGHT - 1);
    //}
    enemyState_ = ENEMY_STATE::ENEMY_ALIVE;
    timeToDeath_ = DEATH_ANIM_FRAME;

    //
    pos_ = { (float)INIT_POS.x * CHA_WIDTH, (float)INIT_POS.y * CHA_HEIGHT };

    //
    forward_ = INIT_DIR;

    if (isGraphic) {
        std::string enemyImage = GetEnemyImage(KABOCHA);
        enemyImage_ = LoadGraph(enemyImage.c_str());
    }
}

Enemy::~~Enemy()
{
    DeleteGraph(enemyImage_);
    DestroyMe();
}

```

```

void Enemy::UpdateEnemyAlive()
{
    EnemyVSBombFire();
    static bool stop = false;

    if (!stop) {
        //
        Point move = { nDir[forward_].x, nDir[forward_].y };
        //
        Pointf npos = { pos_.x + speed_ * move.x * Time::DeltaTime() , pos_.y + speed_ * move.y * Time::DeltaTime() };
        //
        Point nposl = { (int)npos.x, (int)npos.y };
        Rect nRec = { nposl, CHA_WIDTH, CHA_HEIGHT };

        if (!isHitWall(nRec)) //
            {
                isHitWall_ = false;
                pos_ = npos;
            }
        else
            {
                isHitWall_ = true;
            }

        //

        int prgssx = (int)pos_.x % (CHA_WIDTH);
        int prgssy = (int)pos_.y % (CHA_HEIGHT);
        int cx = ((int)pos_.x / (CHA_WIDTH)) % 2;
        int cy = ((int)pos_.y / (CHA_HEIGHT)) % 2;
        static int count = 0;
        if (prgssx == 0 && prgssy == 0 && cx && cy)
            {
                if (isHitWall_) {

                }
                //
                //
            }
    }
}

```

```
    }
```

```
}
```

```
    // 动画
```

```
    animTimer_ += Time::DeltaTime();
```

```
    if (animTimer_ > 0.3f)
```

```
    {
```

```
        animFrame_ = (animFrame_ + 1) % 4;
```

```
        animTimer_ = animTimer_ - ANIM_INTERVAL;
```

```
    }
```

```
}
```

```
void Enemy::UpdateEnemyDeadReady()
```

```
{
```

```
    float ANIM_INTERVAL = 0.2f;
```

```
    float FLY_SPEED = 800.0f;
```

```
    timeToDeath_ -= Time::DeltaTime();
```

```
    if (timeToDeath_ < 0) {
```

```
        timeToDeath_ = DEATH_ANIM_FRAME;
```

```
        SceneManager::ChangeScene("CLEAR");
```

```
    }
```

```
}
```

```
void Enemy::UpdateEnemyDead()
```

```
{
```

```
}
```

```
void Enemy::Update()
```

```
{
```

```
    switch (enemyState_)
```

```
    {
```

```
        case ENEMY_STATE::ENEMY_ALIVE:
```

```
            UpdateEnemyAlive();
```

```
            break;
```

```
        case ENEMY_STATE::ENEMY_DEAD_READY:
```

```
    UpdateEnemyDeadReady();  
    break;  
case ENEMY_STATE::ENEMY_DEAD:  
    UpdateEnemyDead();  
    break;  
default:  
    break;  
}
```

```
}  
//turnRight(),turnLeft()  
//reverse(),forward()  
//Point GetXYDistance();
```

```
//
```

```
void Enemy::TurnRight()
```

```
{  
    if(forward_ == UP)  
        forward_ = NONE;  
    else if (forward_ == RIGHT)  
        forward_ = NONE;  
    else if (forward_ == DOWN)  
        forward_ = NONE;  
    else if (forward_ == LEFT)  
        forward_ = NONE;  
}
```

```
//
```

```
void Enemy::TurnLeft()
```

```
{  
    if(forward_ == UP)  
        forward_ = NONE;  
    else if (forward_ == LEFT)  
        forward_ = NONE;  
    else if (forward_ == DOWN)  
        forward_ = NONE;
```

```

else if (forward_ == RIGHT)
    forward_ = NONE;
}

```

```
void Enemy::Turn180()
{
    if (forward_ == UP)
        forward_ = NONE;
    else if (forward_ == LEFT)
        forward_ = NONE;
    else if (forward_ == DOWN)
        forward_ = NONE;
    else if (forward_ == RIGHT)
        forward_ = NONE;
}
```

```
Pointf Enemy::GetPlayerDist()
{
    □Player* p = FindGameObject<Player>();
    □Pointf dist;
    □dist.x = fabs(p->GetPos().x - pos_.x);
    □dist.y = fabs(p->GetPos().y - pos_.y);
    □return dist;
}
```

```
//Y[ ]
void Enemy::YCloserMove()
{
    [ ]Player* player = (Player*)FindObject<Player>();
    [ ]/*
    [ ]if (pos_.y > [ ] y[ ] )
    [ ]{
    [ ]    [ ]forward_ = NONE;
    [ ]}
    [ ]else if (pos_.y < [ ] y[ ] )
    [ ]{
    [ ]    [ ]forward_ = NONE;
    [ ]}
```



```

    {
        //XXXXXXXXXXXXXXXXXXXX
        if(pos_.y > player->GetPos().y)
        {
            forward_ = UP;
        }
        else if (pos_.y < player->GetPos().y)
        {
            forward_ = DOWN;
        }
    }
}

//
void Enemy::XYCloserMoveRandom()
{

    //1XXXXXXXXXXXXXXXXXXXX32XXXXXXXXXXXX
    Player* player = (Player*)FindGameObject<Player>();
    int xdis = abs(pos_.x - player->GetPos().x);
    int ydis = abs(pos_.y - player->GetPos().y);
    int rnum = GetRand(2);
    if (rnum == 0)
        XYCloserMove();
    else
    {
        forward_ = (DIR)GetRand(3);
    }
}

```

```

void Enemy::Draw()
{
    if (isGraphic)
    {

```

```

    DrawRectExtendGraph((int)pos_.x, (int)pos_.y, (int)pos_.x + CHA_WIDTH, (int)pos_.y + CHA_HEIGHT,
    frameNum[animFrame_] * 32, yTerm[forward_]*32, 32, 32, enemyImage_, TRUE);
}
else {

    Point p = { (int)pos_.x, (int)pos_.y };
    DrawBox(p.x, p.y, p.x + CHA_WIDTH, p.y + CHA_HEIGHT,
    GetColor(80, 89, 10), TRUE);
    Point tp[4][3] = {
        {{pos_.x + CHA_WIDTH / 2, pos_.y}, {pos_.x, pos_.y + CHA_HEIGHT / 2}, {pos_.x + CHA_WIDTH, pos_.y +
        CHA_HEIGHT / 2}},
        {{pos_.x + CHA_WIDTH / 2, pos_.y + CHA_HEIGHT}, {pos_.x, pos_.y + CHA_HEIGHT / 2}, {pos_.x + CHA_WIDTH,
        pos_.y + CHA_HEIGHT / 2}},
        {{pos_.x, pos_.y + CHA_HEIGHT / 2}, {pos_.x + CHA_WIDTH / 2, pos_.y}, {pos_.x + CHA_WIDTH / 2,
        pos_.y + CHA_HEIGHT}},
        {{pos_.x + CHA_WIDTH, pos_.y + CHA_HEIGHT / 2}, {pos_.x + CHA_WIDTH / 2, pos_.y}, {pos_.x + CHA_WIDTH /
        2, pos_.y + CHA_HEIGHT}}
    };

    DrawTriangle(tp[forward_][0].x, tp[forward_][0].y, tp[forward_][1].x, tp[forward_][1].y, tp[forward_][2].x,
    tp[forward_][2].y, GetColor(255, 255, 255), TRUE);
}
}

```

```

bool Enemy::CheckHit(const Rect& me, const Rect& other)
{
    if (me.x < other.x + other.w &&
        me.x + me.w > other.x &&
        me.y < other.y + other.h &&
        me.y + me.h > other.y)
    {
        return true;
    }
    return false;
}

```

```

bool Enemy::isHitWall(const Rect& me)
{
    Stage* stage = (Stage*)FindGameObject<Stage>();
}

```

```

for (int i = 0; i < NEIGHBOURS; i++) {
    int x = me.x / CHA_WIDTH + nineNeibor[i].x;
    int y = me.y / CHA_HEIGHT + nineNeibor[i].y;
    CheckBoundary(x, y);
    StageObj& tmp = stage->GetStageGrid()[y][x];
    if (tmp.type == STAGE_OBJ::EMPTY)
        continue;

    if (CheckHit(me, tmp.rect))
    {
        if (tmp.type == STAGE_OBJ::BRICK || tmp.type == WALL || tmp.type == BOMB)
        {
            return true;
        }
    }
}
return false;
}

void Enemy::EnemyVSBombFire()
{
    const float COLLISION_DIST = 0.6f;

    std::list<BombFire*> bfList = FindGameObjects<BombFire>();

    for (auto& itr : bfList)
    {
        std::vector<BomRect*> bRects = itr->GetBomRectList();
        for (auto& itrRec : bRects)
        {
            Point fc = itrRec.rect.GetCenter();
            Rect eRect = { (int)pos_.x, (int)pos_.y, CHA_WIDTH, CHA_HEIGHT };

            Point eCenter = eRect.GetCenter();
            //float dist = (float)((fc.x - eCenter.x) * (fc.x - eCenter.x) + (fc.y - eCenter.y) * (fc.y - eCenter.y));
            float dist = CalcDistance(fc, eCenter);
            if (dist < COLLISION_DIST * CHA_WIDTH)
            {
                //
            }

```

```

SetDrawBlendMode(DX_BLENDMODE_ALPHA, 128);
DrawBox((int)pos_.x, (int)pos_.y, (int)pos_.x + CHA_WIDTH, (int)pos_.y + CHA_HEIGHT, GetColor(0, 200, 200),
TRUE);
SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
enemyState_ = ENEMY_STATE::ENEMY_DEAD_READY;
//SceneManager::ChangeScene("CLEAR");
}
}
}
}
}

```

```

//
//void Enemy::RightHandMove()
//{
//DIR myRight[4] = { RIGHT, LEFT, UP, DOWN };
//DIR myLeft[4] = { LEFT, RIGHT, DOWN, UP };
//Point nposF = { pos_.x + nDir[forward_].x, pos_.y + nDir[forward_].y };
//Point nposR = { pos_.x + nDir[myRight[forward_]].x, pos_.y + nDir[myRight[forward_]].y };
//Rect myRectF{ nposF.x, nposF.y, CHA_WIDTH, CHA_HEIGHT };
//Rect myRectR{ nposR.x, nposR.y, CHA_WIDTH, CHA_HEIGHT };
//Stage* stage = (Stage*)FindObject<Stage>();
//bool isRightOpen = true;
//bool isForwardOpen = true;
//for (auto& obj : stage->GetStageRects()) {
//for (int y = 0; y < STAGE_HEIGHT; y++)
//{
//for (int x = 0; x < STAGE_WIDTH; x++)
//{
//Rect tmp = stage->GetStageGrid()[y][x].rect;
//if (CheckHit(myRectF, tmp)) {
//isForwardOpen = false;
//}
//if (CheckHit(myRectR, tmp)) {
//isRightOpen = false;
//}
//}
//}

```

```

//    if (isRightOpen)
//    {
//        forward_ = myRight[forward_];
//    }
//    else if (isRightOpen == false && isForwardOpen == false)
//    {
//        forward_ = myLeft[forward_];
//    }
// }
// }

//void Enemy::Dijkstra(Point sp, Point gp)
//{
//    using Mdat = std::pair<int, Point>;
//
//    dist[sp.y][sp.x] = 0;
//    std::priority_queue<Mdat, std::vector<Mdat>, std::greater<Mdat>> pq;
//    pq.push(Mdat(0, { sp.x, sp.y }));
//    vector<vector<StageObj>> stageData = ((Stage*)FindGameObject<Stage>())->GetStageGrid();
//
//    while (!pq.empty())
//    {
//        Mdat p = pq.top();
//        pq.pop();
//
//        //    Rect{ (int)p.second.x * STAGE_WIDTH, (int)p.second.y * BLOCK_SIZE.y, BLOCK_SIZE }.draw(Palette::Red);
//        //    getchar();
//        int c = p.first;
//        Point v = p.second;
//
//        for (int i = 0; i < 4; i++)
//        {
//            Point np = { v.x + (int)nDir[i].x, v.y + (int)nDir[i].y };
//            if (np.x < 0 || np.y < 0 || np.x >= STAGE_WIDTH || np.y >= STAGE_HEIGHT) continue;
//            if (stageData[np.y][np.x].obj == STAGE_OBJ::WALL) continue;
//            if (dist[np.y][np.x] <= stageData[np.y][np.x].weight + c) continue;
//            dist[np.y][np.x] = stageData[np.y][np.x].weight + c;
//            pre[np.y][np.x] = Point({ v.x, v.y });
//            pq.push(Mdat(dist[np.y][np.x], np));
//        }

```

//□}

//}

