YESTERNIGHTMARE

Presenters:
张越 黄梓纹 张学一

Date：2024.7.8

Columns

CCCCCCCCO

CCCCCCCCO

ONLINE GAME ONLINE GAME ONLINE GAME ONLINE GAME ONLINE GAME ONLINE GAME ONLINE GAME

01

The inspiration for the game

The inspiration for the game：

Author: Adolf Loos (1860-1933)
Nationality: Hungarian
Creation Time: 1898

The Spirits of the Danube

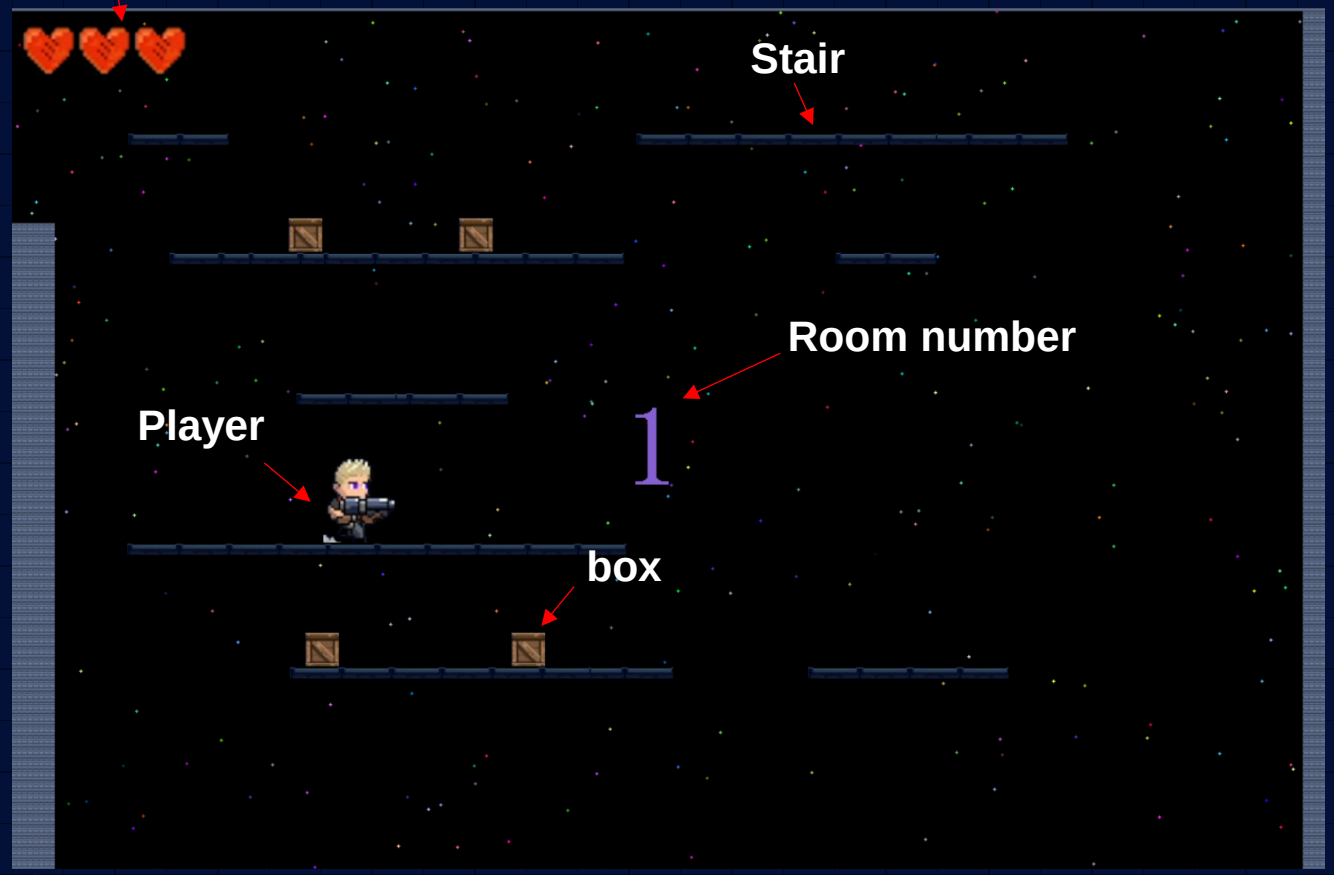
The Spirits of the Danube

Game background

The protagonist falls into an endless nightmare, encountering hidden fears manifested as horrors. A mysterious old man appears, challenging the protagonist and their companion to defeat monsters and break the curse. They face trials in the dream, striving to escape the nightmare cycle and seek the dawn.

Game content

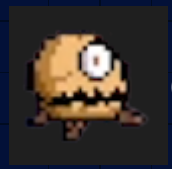
Health points (HP)



Monsters:

Will not track
Has collision damage
Only attacks while floating in the air

Will track
Has collision damage
Approaches the player to attack



Only has collision damage

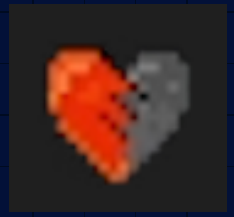
Common drops

Let bullet exist longer

Improve max health point

Add more bullet-number in one shot

Let bullet shoot faster



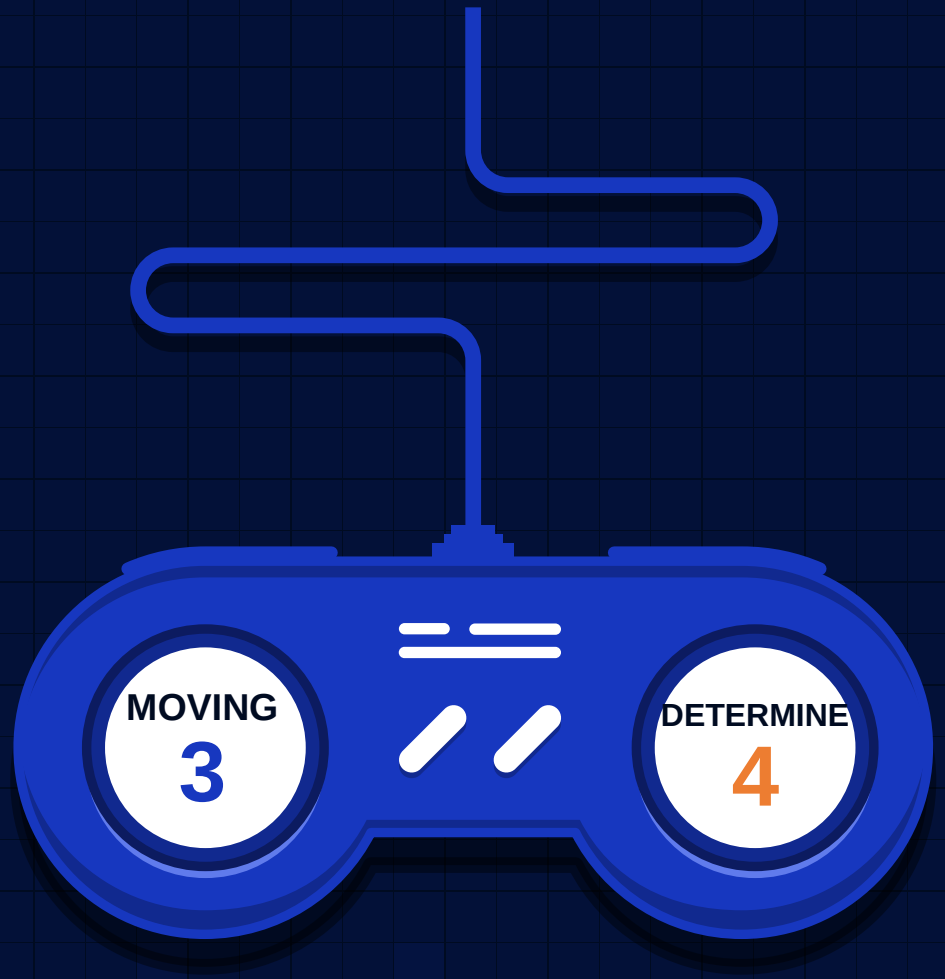
Add one half health point

Add one full point

Game control

In the game, we use the following keys to operate:

A - moves the player left
D - moves the player right
W - jumps
E – follow game orders
Left mouse button - single point attack
Mouse cursor - controls direction
Enter - confirm



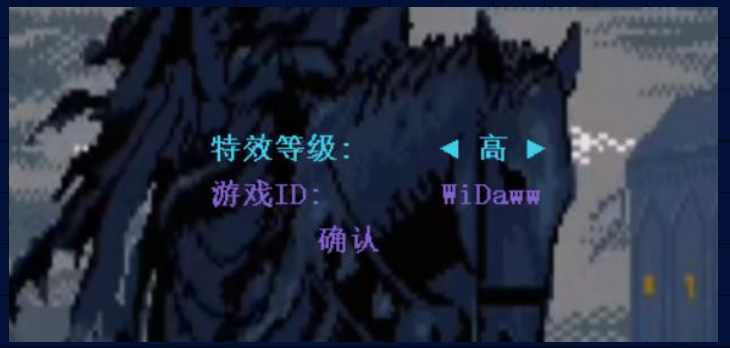
02

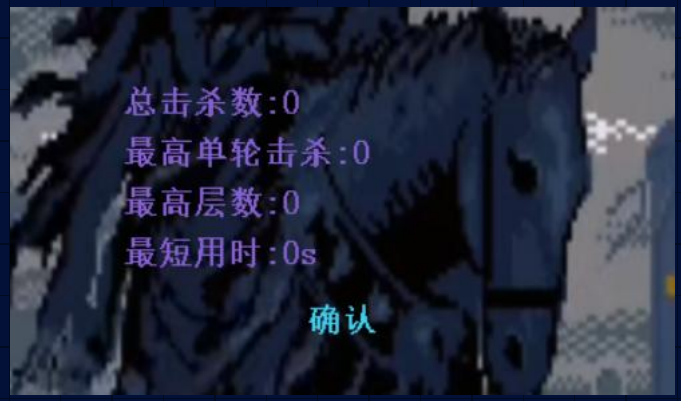
UI design

UI design：


LOADING


昨日梦魇
进入梦境
游戏设置
个人成就
游戏榜单
退出游戏


特效等级：
◀ 高 ▶
游戏ID：
WiDaww
确认


总击杀数:0
最高单轮击杀:0
最高层数:0
最短用时:0s
确认


游戏ID
总击杀数
最高单轮击杀
最高层数
最短用时
上次更新
WiDavv

03

Code analysis

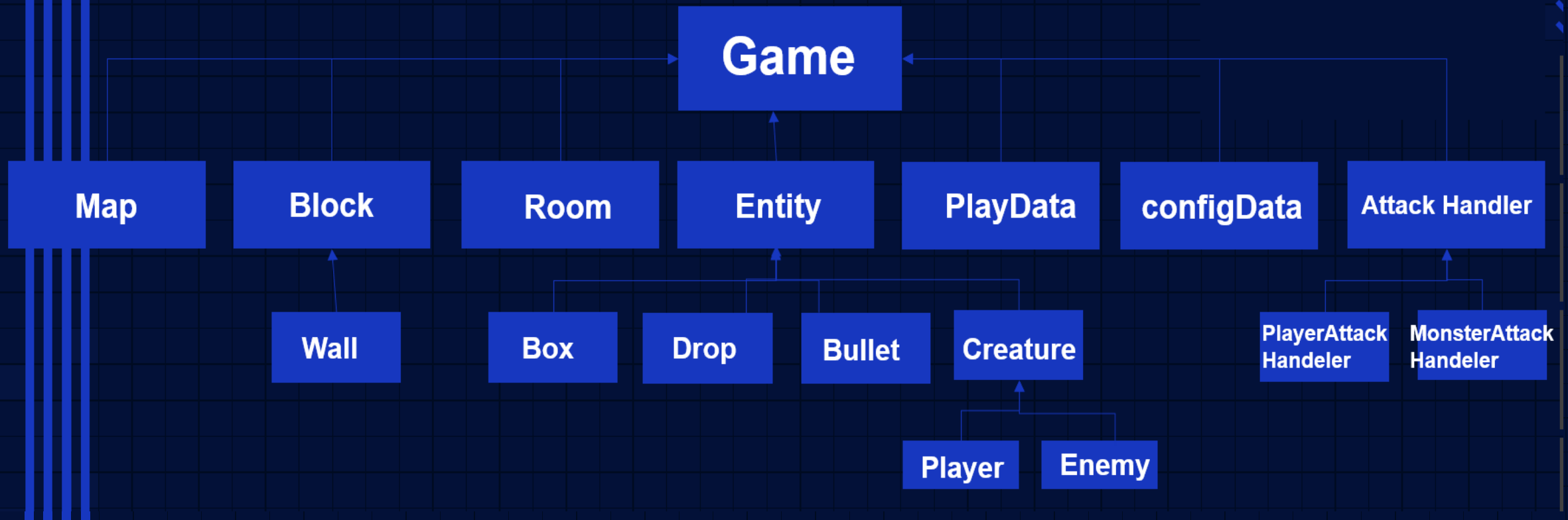
Code amount analysis

2211 lines Accounting for **100%**

Team work

Team member **3** **432** hours

Class association



Class association

```cpp
class Entity
{
public:
    float posx, posy, vx, vy; //实体坐标和速度
    float height, width;    //实体的高度和宽度
    int health; //耐久
    int inRoom; //所在的房间

    float changeX(float n);

    float changeY(float n);

    int isPassible();
};
```

```cpp
class Player : public Creature
{
private:
public:
    float vy;//角色坐标的速度a
    int status; //角色状态 站立 - 1，跑动 - 2 举枪30-60度 - 3 举枪 60-90度 - 4 压枪 - 5
    int level; //当前层数
    Player(int level);

    void init();

    void draw();

    void update();

    void drawHeart();

    int addHealth(int n);

    int minusShootCd(int n) {
        if (shootCd - n > 50) {
            shootCd -= n;
            return 0;
        }
        return 1;
    }
};
```

```cpp
class Creature : public Entity
{
public:
    int numBallistic;//弹道数
    int dirction, health, healthMax; //方向，血量,血量上限
    int lastShootTime; //上次发射子弹的时间
    int shootCd; //发射CD
    int jumpCd; //跳跃CD
    int jumpMax; //最大跳跃次数
    int jumpLeft; //当前剩余跳跃次数
    int onFloor; //是否落地
    int isRemoteAttack; //是否可以远程攻击(发射子弹)
    int bulletNums; // 单次发射的子弹数量
    int shootPower; //射出子弹的威力
    int bulletlifetime; //发射出子弹的存活时间(射程);
};
```

```cpp
class Enemy : public Creature//怪物的类
{
public:
    int kind, fly;
    Enemy(float posx, float posy, int kind, int inRoom);

    void draw();

    void update();

    void updateVelforTarge();

    void Shoot(int X1, int Y1, int X2, int Y2, int from, int power);
};
```

Class association

```
void Player::update()
{
    vy += g;
    if (changeY(vy) != 1)
    {
        if (vy > 0)
        {
            jumpLeft = jumpMax;
        }
        vy = 0;
    }

    if (posx + width - 30 < 0)//向左进入其他房间
    {
        int toRoom;
        if (posy <= 400)//左上角
        {
            toRoom = maps.getRoom(inRoom)->go[0];
        }
        else//左下角
        {
            toRoom = maps.getRoom(inRoom)->go[1];
        }
        for (int i = 0; i < 4; i++)
        {
            if (maps.getRoom(toRoom)->go[i] == inRoom) //找到传送过去是哪个门
            {
                posx = posTransformCoordinate(1, i, width);//位置转坐标
                posy = posTransformCoordinate(2, i, width);//位置转坐标
                inRoom = toRoom;
            }
        }
        bullets.clear();//进入其他房间就把子弹全清空
    }
    if (posx + width - 30 > WIDTH)//向右进入其他房间
    {
        int toRoom;
        if (posy <= 400)//右上角
        {
            toRoom = maps.getRoom(inRoom)->go[2];
        }
        else//右下角
        {
            toRoom = maps.getRoom(inRoom)->go[3];
        }
        for (int i = 0; i < 4; i++)
        {
            if (maps.getRoom(toRoom)->go[i] == inRoom)
            { //找到传送过去是哪个门
                posx = posTransformCoordinate(1, i, width);//位置转坐标
                posy = posTransformCoordinate(2, i, width);//位置转坐标
                inRoom = toRoom;
            }
        }
        bullets.clear();//进入其他房间就把子弹全清空
    }
    Room* k = maps.getRoom(inRoom);
    for (auto it = k->drops.begin(); it != k->drops.end(); it++) { //迭代所有的掉落物
        if (&it->posx == &posx) continue; //如果posx的地址相同 -> 同一个掉落物，跳过判断
        if (it->visible <= 0) continue; //被打爆的箱子跳过判断
        if (collisionBetweenEntity(posx, posy - height,
            posx + width, posy,
            it->posx, it->posy - it->height,
            it->posx + it->width, it->posy))
        {
            if (it->material == 1)
            {
                int k = addHealth(2);
                if (k == 0) {
                    it->visible = 0;
                }
                if (k == 1) {
                    it->material = 2;
                }
            }
            if (it->material == 2)
            {
                int k = addHealth(1);
                if (k == 0) {
                    it->visible = 0;
                }
            }
            if (it->material == 3) {
                healthMax += 2;
                it->visible = 0;
            }
            if (it->material == 4) {
                numBallistic += 1;
                it->visible = 0;
            }
            if (it->material == 5) {
                bulletlifetime += 50;
                it->visible = 0;
            }
            if (it->material == 6) {
                minusShootCd(50);
                shootCd -= 50;
                shootCd = max(shootCd, 100);
                it->visible = 0;
            }
        }
    }
}
```

Class association

```
}

void Enemy::update()
{
    //vx == 0 时不做修改
    if (vx > 0)
    {
        dirction = 0;
    }
    if (vx < 0)
    {
        dirction = 1;
    }
    if (fly)
    {
        if (kind == 2) {
            updateVelforTarge();
            posx += vx;
            posy += vy;
        }
        if (kind == 3) {
            ;
            posx += vx;
            if (posx < 40 || posx > WIDTH - width - 30) {
                vx = -vx;
            }
        }
    }
    else
    {
        vy = 5;
        if (!onFloor)//判断是否已经落地
        {
            if (changeY(vy) != 1)
            {
                onFloor = 1;
                vx = 0.5;
            }
```

```
            if (changeY(vy) != 1)
            {
                onFloor = 1;
                vx = 0.5;
            }
        }
        else
        {
            if (changeY(vy) == 1)
            {
                changeY(-vy);
                vx = -vx;
            }
        }
        if (changeX(vx) != 1 || posx < 0 || posx > WIDTH) {
            vx = -vx;
        }
    }
    if (isRemoteAttack)
    {
        Shoot(posx, posy, hero.posx, hero.posy, 2, 1);
    }
}

void Enemy::updateVelforTarge() // 让会飞的怪的速度瞄向角色
{
    if (hero.posx > posx) vx = 0.5;// 目标在怪左边，怪x方向速度向右
    else if (hero.posx < posx) vx = -0.5;// 目标在怪右边，怪x方向速度向左
    if (hero.posy > posy)vy = 0.5; // 目标在怪下方，怪y方向速度向下
    else if (hero.posy < posy)vy = -0.5; // 目标在怪上方，怪y方向速度向上
}
```

Class association

```cpp
// 抽象基类，用于处理攻击
class AttackHandler {
public:
    virtual void handleDamage(std::vector<Bullet>& bullets, std::vector<Enemy>& enemies, Entity& entity) = 0;
};
```

```cpp
// 玩家攻击的具体实现
class PlayerAttackHandler : public AttackHandler {
public:
    void handleDamage(std::vector<Bullet>& bullets, std::vector<Enemy>& enemies, Entity& entity) override {
        for (auto it1 = bullets.begin(); it1 != bullets.end(); ) {
            it1->update(); // 更新子弹位置

            for (auto it2 = enemies.begin(); it2 != enemies.end(); ++it2) {
                if (it1->from == 1 && it2->health > 0 && collisionBetweenEntity(it1->posx, it1->posy - 2 * it1->radius, it1->posx + 2 * it1->radius, it1->posy, it2->posx
                    --it2->health; // 减少敌人生命值
                    it1->status = 0; // 子弹失效
                    break;
                }
            }
            if (it1->getStatus() == 0) it1 = bullets.erase(it1);
            else ++it1;
        }
    }
};
```

Class association

```cpp
// 怪物攻击的具体实现
class MonsterAttackHandler : public AttackHandler {
private:
    float lastTime;
public:
    MonsterAttackHandler() : lastTime(0) {}

    void handleDamage(std::vector<Bullet>& bullets, std::vector<Enemy>& enemies, Entity& entity) override {
        float nowTime = clock();

        for (auto it1 = bullets.begin(); it1 != bullets.end(); ) {
            if (nowTime - lastTime > 1000 && it1->from == 2 && collisionBetweenEntity(it1->posx, it1->posy - 2 * it1->radius, it1->posx + 2 * it1->radius, it1->posy, entit
                lastTime = nowTime;
                entity.health--; // 减少实体（英雄）的生命值
                it1->status = 0; // 子弹失效
                break;
            }
            if (it1->getStatus() == 0) it1 = bullets.erase(it1);
            else ++it1;
        }

        for (auto it = enemies.begin(); it != enemies.end(); ++it) {
            if (nowTime - lastTime > 1000 && collisionBetweenEntity(entity.posx, entity.posy - entity.height, entity.posx + entity.width, entity.posy, it->posx, it->posy -
                lastTime = nowTime;
                entity.health--; // 减少实体（英雄）的生命值
                break;
            }
        }
    }
};
```

```cpp
// 更新函数，使用多态处理攻击
void updateWithoutInput1() {
    int tick = 1;
    tick++;
    if (tick > 0x3f3f3f3f) tick = 1;
    Player hero(1);
    Map maps;
    vector<Bullet> bullets;
    vector<Enemy>monsters[233];//每个房间都有一个存怪物的vector
    hero.update(); // 更新英雄位置

    int nowRoom = hero.inRoom;
    Room* now = maps.getRoom(nowRoom);

    // 使用多态处理攻击
    std::unique_ptr<AttackHandler> attackHandler;

    // 根据实体类型来决定使用哪种攻击处理器
    if (dynamic_cast<Player*>(&hero)) {
        attackHandler = std::make_unique<PlayerAttackHandler>();
    }
    else {
        attackHandler = std::make_unique<MonsterAttackHandler>();
    }

    attackHandler->handleDamage(bullets, monsters[nowRoom], hero);
}
```

File I/O Functionality

- **Sequential File Read/Write**
- **Configdata/Playerdata**
- **Game Effect**
- **Playerdata:Name、Totalkill、Maxoncekill、Level、Lasttime**

```cpp
//文件读写功能
configData::configData() {
    string folderPath = "./data";

    // 创建 data 文件夹
    struct stat info;
    if (stat(folderPath.c_str(), &info) != 0 || !(info.st_mode & S_
#ifdef _WIN32
        _mkdir(folderPath.c_str());
#else
        mkdir(folderPath.c_str(), 0733);
#endif
    }

    ifstream infile("./data/config.gm");
    if (infile.is_open()) {
        infile >> effects;
        infile.close();
    }
    else {
        ofstream outfile("./data/config.gm");
        effects = 1;
        if (outfile.is_open()) {
            outfile << effects << endl;
            outfile.close();
        }
        else {
            cerr << "Unable to open file for writing" << std::endl
        }
    }
}

void configData::save() {
    ofstream outfile("./data/config.gm");
    if (outfile.is_open()) {
        outfile << effects << std::endl;
        outfile.close();
    }
    else {
        std::cerr << "Unable to open file for writing" << std::end
    }
}
```

```cpp
playerData::playerData() {
    std::string folderPath = "./data";

    // 创建 data 文件夹
    struct stat info;
    if (stat(folderPath.c_str(), &info) != 0 || !(info.st_mode & S_IFDI
#ifdef _WIN32
        _mkdir(folderPath.c_str());
#else
        mkdir(folderPath.c_str(), 0733);
#endif
    }

    std::ifstream infile("./data/data.gm");
    if (infile.is_open()) {
        infile >> name >> totilKilled >> maxOnceKilled >> level >> last
        infile.close();
    }
    else {
        time_t t;
        srand(static_cast<unsigned>(time(&t)));
        std::string k = "Player_" + std::to_string(rand() % 10000);
        std::ofstream outfile("./data/data.gm");
        if (outfile.is_open()) {
            outfile << k << "\n"
                << 0 << "\n"
                << 0 << "\n"
                << 0 << "\n"
                << 0.0f << std::endl;
            outfile.close();
        }
        else {
            std::cerr << "Unable to open file for writing" << std::endl
        }
        std::strcpy(name, k.c_str());
        totilKilled = 0;
        maxOnceKilled = 0;
        level = 0;
        lastTime = 0.0f;
    }
}

void playerData::save() {
    std::ofstream outfile("./data/data.gm");
    if (outfile.is_open()) {
        outfile << name << "\n"
            << totilKilled << "\n"
            << maxOnceKilled << "\n"
            << level << "\n"
            << lastTime << std::endl;
        outfile.close();
    }
    else {
        std::cerr << "Unable to open file for writing" << std::endl;
    }
}
```

File I/O Functionality

Function Implementation：

- The playerData and configData classes use sequential file read/write operations to manage game data.

- Sequential file input/output is used to process the game configuration and player data files.

Game Logic

- **Initialization**
- **Drawmonsters（vector、map containers)**
- **Mouse Control**
- **Key Control**

```
void Init()  // 预处理
{
    maps.initMap(hero.level);
    cleardevice();
    if (hero.level == 1) {
        beginTime = clock();
        onceKilled = 0;
        hero.init();  // 角色初始化大小和坐标和血量
        maps.initMap(1);
        bullets.clear();
    }
    else {
        hero.posx = WIDTH / 2 - hero.width / 2;
        hero.posy = 770;
    }
    hero.inRoom = 0;  // 初始房间是0号
    for (int i = 0; i < maps.N; i++) {
        monsters[i].clear();
    }
    makeMonster();  // 造怪物
}

void drawMonster()  // 画怪物
{
    int nowRoom = hero.inRoom;
    for (vector<Enemy>::iterator it = monsters[nowRoom].begin();
        it != monsters[nowRoom].end(); ++it)
        it->draw();
}
```

```
void updateWithInput() {
    int flag = 1 /*判断是否有操作，以此来更新角色状态图片*/;
    MOUSEMSG m;
    static int shoot = 0;            // 判断是否正在开枪
    static float lastShootTime = 0;  // 上次开枪时间
    static float lastJumpTime = 0;   // 上次跳跃时间
    if (MouseHit())                  // 关于鼠标的操作
    {
        m = GetMouseMsg();
        posx_mouse = m.x;
        posy_mouse = m.y;  // 鼠标位置

        if (m.uMsg == WM_LBUTTONDOWN) {
            shoot = 1;  // 开枪了
        }
        if (m.uMsg == WM_LBUTTONUP) {
            /*mciSendString(_T("close shootMusic"), NULL, 0, NULL);//关闭开枪音效*/
            shoot = 0;  // 不开枪了
        }
        flag = 0;
    }

    if (shoot) {
        if (posx_mouse < hero.posx)
            hero.dirction = 1; /*枪口的朝向*/
        else
            hero.dirction = 0;
        if (abs(posx_mouse - hero.posx) < abs(posy_mouse - hero.posy))
            hero.status = 4;
        else
            hero.status = 3;
        if (posy_mouse - hero.posy > 0) hero.status = 5;
    }
```

```
if (GetKeyState(0x41) < 0)  // 按下了A
{
    int k = hero.changeX(-1);
    if (k >= 3) {
        if (maps.getRoom(hero.inRoom)->getBox(k)->changeX(-1) == 1) {
            maps.getRoom(hero.inRoom)->getBox(k)->changeX(-1);
        }
    }
    if (!shoot) hero.status = 2;
    if (flag) hero.dirction = 1;
    flag = 0;
}
if (GetKeyState(0x44) < 0)  // 按下了D
{
    int k = hero.changeX(1);
    if (k >= 3) {
        if (maps.getRoom(hero.inRoom)->getBox(k)->changeX(1) == 1) {
            maps.getRoom(hero.inRoom)->getBox(k)->changeX(1);
        }
    }
    if (!shoot) hero.status = 2;
    if (flag) hero.dirction = 0;
    flag = 0;
}
```

04

Team Performance Evaluation

Duty Assignments

- Zhang yue: User interface Design

- Zhang xue yi: Level design、Inheritance and polymorphism of design classes

- Huang zi wen: After the class design is completed, initialization creation, file reading、other operations are carried out

- Everyone does the same amount of work!!!

Everyone does equally to the game!!!

Person	Duty
Yue Zhang	Game and UI Design、Animation
Ziwen Huang	Initialization creation, file reading
Xueyi Zhang	Class structure design and Inheritance and polymorphism design

Statement of Originality & Conclusions

Developing "Yesterday's Nightmare" was a deeply engaging and enriching experience that demanded both creativity and technical expertise. Throughout the project, we took on the challenging task of integrating the storyline with game mechanics, striving to balance the game's difficulty while enhancing player experience. We worked extensively on translating complex game mechanics into intuitive and engaging user interfaces, ensuring that players could smoothly navigate the game and fully immerse themselves in its eerie and mysterious atmosphere. This required constant communication and collaboration with team members to ensure that our design expressed the core ideas of the game effectively. Additionally, applying my knowledge from C++ courses to real-world scenarios helped us solve complex technical problems, optimize game performance, and enhance backend logic. This project significantly improved our design skills, programming abilities, and understanding of user experience, teamwork, and technological innovation, laying a solid foundation for my future career in game design and software development.

The development of "Yesterday's Nightmare" showcased significant innovation and originality. Our goal was to create a game that provided not only entertainment but also deep emotional resonance through its unique background and profound themes. We achieved this by integrating strong color contrasts and lighting effects to evoke feelings of loneliness and fear, enhancing the narrative. Our game mechanics were designed to be both intuitive and engaging, balancing challenge with player comfort. Technically, we applied advanced concepts in object-oriented programming, data structures, and algorithms to optimize performance and solve complex problems. The collaborative effort of our team, leveraging each member’ s unique strengths, resulted in a truly original and innovative game that stands out in the industry.

Thank you!

THANK YOU