**高级语言程序设计**

**课设报告**

**题 目 提瓦特幸存者**

**学 号 23025102**

**姓 名 何沛言**

**提交日期 2024.4.18**

**成绩评价表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 程序功能实现(15分) | 程序执行情况(15分) | 程序功能模块组织(10分) | 问题回答情况(10分) | 链表使用(10分) |
| □内容丰富、炫酷(13-15) | □流畅(13-15) | □模块和数据组织合理(9-10) | □清楚、正确(9-10) | □正确运用(10) |
| □扩展基本要求(9-12) | □界面有闪动 (9-12) | □合理、少量不妥(7-8) | □基本正确(7-8) | □基本正确(7-9) |
| □基本完成基本要求(5-8) | □操作不灵活(5-8) | □基本合理(5-6) | □有部分错误(5-6) | □存在问题(1-6) |
| □未完成基本要求(0-4) | □有错误(0-4) | □不合理(0-4) | □基本不能回答(0-4) | □未使用(0) |
|  |  |  |  |  |
| 报告需求分析、总体设计(10分) | 报告技术实现分析(10分) | 报告写作规范及总结(10分) | 总评成绩(90分) |  |
| □深入、全面(9-10) | □动静态、交互、核心功能、测试全面分析(9-10) | □格式规范、总结深刻(9-10) |  |  |
| □正确(7-8) | □完成4项功能分析(7-8) | □格式较规范，总结较深刻(7-8) |  |  |
| □基本符合要求(5-6) | □完成2-3项功能分析(4-6) | □格式基本规范、总结简单(5-6) |  |  |
| □内容不全(0-4) | □完成0-1功能分析(0-3) | □格式较混乱、没有总结(0-4) |  |  |
|  |  |  |  |  |

目录

[1 需求分析 3](#_Toc164343959)

[1.1 功能需求 3](#_Toc164343960)

[1.2数据需求 3](#_Toc164343961)

[1.3界面需求 3](#_Toc164343962)

[1.4 开发与运行环境需求 3](#_Toc164343963)

[2 游戏程序总体设计 4](#_Toc164343964)

[2.1 程序模块设计 4](#_Toc164343965)

[2.3主要数据结构 6](#_Toc164343966)

[3 游戏实现技术点分析 6](#_Toc164343967)

[3.1静态画面设计 6](#_Toc164343968)

[3.2游戏动画设计 6](#_Toc164343969)

[3.3游戏交互设计 7](#_Toc164343970)

[3.4实现技术点分析 7](#_Toc164343971)

[4 测试 7](#_Toc164343972)

[5 用户手册 7](#_Toc164343973)

[6 总结提高 7](#_Toc164343974)

[6.1 课程设计总结 7](#_Toc164343975)

[附件：程序源代码（根据代码的长度适当选取较小的字号和行距，Times New Roman字体） 7](#_Toc164343976)

# 1 需求分析

利用EASYX编写一个割草游戏提瓦特幸存者，游戏中玩家可以使用方向键操纵角色，利用角色四周旋转的武器击杀野猪敌人。

## 1.1 功能需求

1 程序动画实现和角色移动

2 敌人随机生成和索敌逻辑实现

3 2D碰撞检测和音乐音效播控

4 用户界面实现

## 1.2数据需求

输入数据：鼠标左键与主菜单界面按钮的交互，游戏内方向键

中间数据：角色与敌人的移动和状态

输出结果：角色死亡时的得分

## 1.3界面需求

主菜单界面：包括主菜单背景图，开始游戏、结束游戏两个按钮。

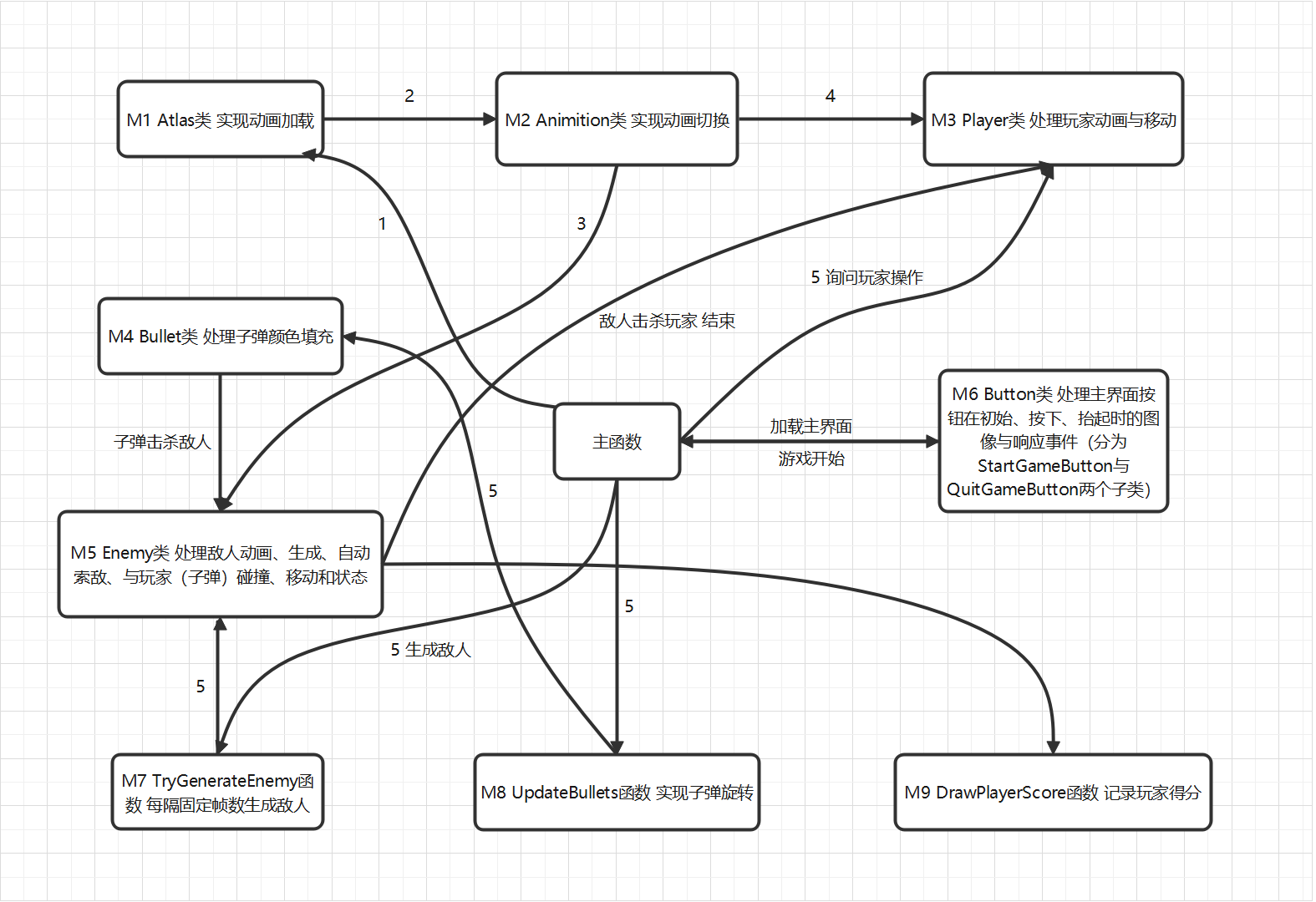
游戏内界面：包括游戏内背景图，角色，敌人若干和计分板。

## 1.4 开发与运行环境需求

VS2022与EASYX图形库

# 2 游戏程序总体设计

## 2.1 程序模块设计



**模块编号**：M1

**函数原型**：Atlas(LPCTSTR path, int num)

**功能**：加载动画

**参数**：png图片路径path，一组图片数量

**返回值**：Atlas\*

**模块编号**：M2

**函数原型**：Animition(Atlas\* atlas, int interval)

**功能**：切换动画

**参数**：Atlas\* atlas，帧间隔interval

**返回值**：Animition

**模块编号**：M3-1

**函数原型**：void ProcessEvent(const ExMessage& msg)

**功能**：询问键盘操作

**参数**：键盘方向键

**模块编号**：M3-2

**函数原型**：void Move()

**功能**：角色移动

**模块编号**：M3-3

**函数原型**：void Draw(int delta)

**功能**：绘制角色

**参数**：帧间隔delta

**模块编号**：M4-1

**函数原型**：void Draw()

**功能**：绘制子弹

**模块编号**：M5-1

**函数原型**：bool CheckBulletCollision(const Bullet& bullet)

**功能**：检测子弹击杀敌人

**参数**：Bullet位置

**返回值**：is\_over\_lap\_x && is\_over\_lap\_y

**模块编号**：M5-2

**函数原型**：bool CheckPlayerCollision(const Player& player)

**功能**：检测敌人碰撞角色

**参数**：Player位置

**返回值**：is\_over\_lap\_x && is\_over\_lap\_y

**模块编号**：M5-3

**函数原型**：void Move(const Player& player)

**功能**：敌人移动

**参数**：Player位置

**模块编号**：M5-4

**函数原型**：void Draw(int delta)

**功能**：绘制敌人

**参数**：帧间隔delta

**模块编号**：M5-5

**函数原型**：void Hurt()

**功能**：敌人受伤状况

**模块编号**：M5-6

**函数原型**：bool CheckAlive()

**功能**：敌人存活状况

**返回值**：alive

**模块编号**：M6-1

**函数原型**：void ProcessEvent(const ExMessage& msg)

**功能**：开始/退出游戏

**参数**：鼠标左键

**模块编号**：M6-2

**函数原型**：void Draw()

**功能**：绘制初始、按下、抬起时的按键

**模块编号**：M7

**函数原型**：void TryGenerateEnemy(std::vector<Enemy\*>& enemy\_list)

**功能**：每隔固定帧生成敌人

**参数**：enemy容器列表

**模块编号**：M8

**函数原型**：void UpdateBullets(std::vector<Bullet>& bullet\_list, const Player& player)

**功能**：实现子弹旋转

**参数**：bullet容器列表，角色位置

**模块编号**：M9

**函数原型**：void DrawPlayerScore(int score)

**功能**：记录玩家得分

**参数**：玩家得分score

## 2.3主要数据结构

1 采用面向对象编程 设计Player、Enemy、Button等类。

2 角色图片组（左）img/player\_left\_%d.png

角色图片组（右）img/player\_right\_%d.png

角色/敌人影子图片

敌人图片组（左）img/enemy\_left\_%d.png

敌人图片组（右）img/enemy\_right\_%d.png

开始/结束游戏按钮图片

主菜单栏图片、游戏背景图片

# 3 游戏实现技术点分析

## 3.1静态画面设计

使用loadimage函数加载主菜单界面和游戏内背景。

## 3.2游戏动画设计

使用Atlas类加载动画进入frame\_list容器，再在Animition类中加入计时器切换动画。

## 3.3游戏交互设计

使用switch case语法与WM\_KEY/MOUSE消息实现方向键和鼠标左键交互。

## 3.4实现技术点分析

使用loadimage，putimage，mciSendString和\_T等函数加载图片与音乐。

# 4 测试

错误用例：角色与敌人速度SPEED<根号2，会导致角色与敌人斜向移动时速度分量（整型）因除以根号2而等于0。

正确用例：令角色与敌人速度SPEED>根号2。

# 5 用户手册

(1) 应用程序功能的详细说明：在提瓦特幸存者游戏中实现用户界面，程序动画，角色移动，敌人随机生成和索敌，2D碰撞检测和音乐音效播控

(2) 应用程序运行环境的要求：VS2022和EASYX图形库

(3) 应用程序的安装与启动方法：打开VS2022启动本地调试器

(4) 程序的界面、交互方式和操作方法：在主界面点击开始游戏后利用方向键控制角色移动击杀敌人

# 6 总结提高

学习了面向对象编程，将变量与函数以类封装，学习了EASYX中多种函数的使用。解决了角色与敌人斜向移动时速度归零的问题。

# 附件：程序源代码

#include <graphics.h>

#include<string>

#include<vector>

const int WINDOW\_HEIGHT = 720;

const int WINDOW\_WIDTH = 1280;

const int BUTTON\_WIDTH = 192;

const int BUTTON\_HEIGHT = 75;

#pragma comment(lib,"MSIMG32.LIB")

#pragma comment(lib,"Winmm.lib")

bool running = true;

bool is\_game\_started = false;

inline void putimage\_alpha(int x, int y, IMAGE\* img) {

int w = img->getwidth();

int h = img->getheight();

AlphaBlend(GetImageHDC(NULL), x, y, w, h,

GetImageHDC(img), 0, 0, w, h, { AC\_SRC\_OVER,0,255,AC\_SRC\_ALPHA });

}

class Atlas {

public:

Atlas(LPCTSTR path, int num) {

TCHAR path\_flie[256];

for (size\_t i = 0; i < num; i++) {

\_stprintf\_s(path\_flie, path, i);

IMAGE\* frame = new IMAGE();

loadimage(frame, path\_flie);

frame\_list.push\_back(frame);

}

}

~Atlas() {

for (size\_t i = 0; i < frame\_list.size(); i++)

delete frame\_list[i];

}

public:

std::vector<IMAGE\*> frame\_list;

};

Atlas\* atlas\_player\_left;

Atlas\* atlas\_player\_right;

Atlas\* atlas\_enemy\_left;

Atlas\* atlas\_enemy\_right;

class Animition {

public:

Animition(Atlas\* atlas, int interval) {

anim\_atlas = atlas;

interval\_ms = interval;

}

~Animition() = default;

void Play(int x,int y,int delta) {

timer += delta;

if (timer >= interval\_ms) {

idx\_frame = (idx\_frame + 1) % anim\_atlas->frame\_list.size();

timer = 0;

}

putimage\_alpha(x, y, anim\_atlas->frame\_list[idx\_frame]);

}

private:

int timer = 0;

int idx\_frame = 0;

int interval\_ms = 0;

private:

Atlas\* anim\_atlas;

};

class Player {

public:

const int FRAME\_WIDTH = 80;

const int FRAME\_HEIGHT = 80;

public:

Player() {

loadimage(&img\_shadow, \_T("img/shadow\_player.png"));

anim\_left = new Animition(atlas\_player\_left, 90);

anim\_right = new Animition(atlas\_player\_right, 90);

}

~Player() {

delete anim\_left;

delete anim\_right;

}

void ProcessEvent(const ExMessage& msg) {

switch (msg.message) {

case WM\_KEYDOWN:

switch (msg.vkcode)

{

case VK\_UP:

is\_move\_up = true;

break;

case VK\_DOWN:

is\_move\_down = true;

break;

case VK\_LEFT:

is\_move\_left = true;

break;

case VK\_RIGHT:

is\_move\_right = true;

break;

}

break;

case WM\_KEYUP:

switch (msg.vkcode)

{

case VK\_UP:

is\_move\_up = false;

break;

case VK\_DOWN:

is\_move\_down = false;

break;

case VK\_LEFT:

is\_move\_left = false;

break;

case VK\_RIGHT:

is\_move\_right = false;

break;

}

break;

}

}

void Move() {

int dir\_x = is\_move\_right - is\_move\_left;

int dir\_y = is\_move\_down - is\_move\_up;

double len\_dir = sqrt(dir\_x \* dir\_x + dir\_y \* dir\_y);

if (len\_dir != 0) {

double normalized\_x = dir\_x / len\_dir;

double normalized\_y = dir\_y / len\_dir;

position.x += (int)(SPEED \* normalized\_x);

position.y += (int)(SPEED \* normalized\_y);

}

if (position.x < 0) position.x = 0;

if (position.y < 0) position.y = 0;

if (position.x + FRAME\_WIDTH > WINDOW\_WIDTH) position.x = WINDOW\_WIDTH - FRAME\_WIDTH;

if (position.y + FRAME\_HEIGHT > WINDOW\_HEIGHT) position.y = WINDOW\_HEIGHT - FRAME\_HEIGHT;

}

void Draw(int delta) {

int pos\_shadow\_x = position.x + (FRAME\_WIDTH / 2 - SHADOW\_WIDTH / 2);

int pos\_shadow\_y = position.y + FRAME\_HEIGHT - 8;

putimage\_alpha(pos\_shadow\_x, pos\_shadow\_y, &img\_shadow);

static bool facing\_left = false;

int dir\_x = is\_move\_right - is\_move\_left;

if (dir\_x < 0) {

facing\_left = true;

}

else if (dir\_x > 0) {

facing\_left = false;

}

if (facing\_left)

anim\_left->Play(position.x, position.y, delta);

else

anim\_right->Play(position.x, position.y, delta);

}

const POINT& GetPosition() const {

return position;

}

private:

const int SPEED = 2;

const int SHADOW\_WIDTH = 32;

private:

IMAGE img\_shadow;

Animition\* anim\_left;

Animition\* anim\_right;

POINT position = { 640,360 };

bool is\_move\_up = false;

bool is\_move\_down = false;

bool is\_move\_left = false;

bool is\_move\_right = false;

};

class Bullet {

public:

POINT position = { 0,0 };

public:

Bullet() = default;

~Bullet() = default;

void Draw() const {

setlinecolor(RGB(255, 155, 50));

setfillcolor(RGB(200, 75, 10));

fillcircle(position.x, position.y, RADIUS);

}

private:

const int RADIUS = 10;

};

class Enemy {

public:

Enemy() {

loadimage(&img\_shadow, \_T("img/shadow\_enemy.png"));

anim\_left = new Animition(atlas\_enemy\_left, 90);

anim\_right = new Animition(atlas\_enemy\_right, 90);

enum class SpawnEdge {

Up = 0,

Down,

Left,

Right

};

SpawnEdge edge = (SpawnEdge)(rand() % 4);

switch (edge) {

case SpawnEdge::Up:

position.x = rand() % WINDOW\_WIDTH;

position.y = -FRAME\_HEIGHT;

break;

case SpawnEdge::Down:

position.x = rand() % WINDOW\_WIDTH;

position.y = WINDOW\_HEIGHT;

break;

case SpawnEdge::Left:

position.x = -FRAME\_WIDTH;

position.y = rand() % WINDOW\_HEIGHT;

break;

case SpawnEdge::Right:

position.x = WINDOW\_WIDTH;

position.y = rand() % WINDOW\_HEIGHT;

break;

default:

break;

}

}

bool CheckBulletCollision(const Bullet& bullet) {

bool is\_over\_lap\_x = bullet.position.x >= position.x && bullet.position.x <= position.x + FRAME\_WIDTH;

bool is\_over\_lap\_y = bullet.position.y >= position.y && bullet.position.y <= position.y + FRAME\_HEIGHT;

return is\_over\_lap\_x && is\_over\_lap\_y;

}

bool CheckPlayerCollision(const Player& player) {

POINT check\_position = { position.x + FRAME\_WIDTH / 2,position.y + FRAME\_HEIGHT / 2 };

POINT player\_position = player.GetPosition();

bool is\_over\_lap\_x = check\_position.x >= player\_position.x && check\_position.x <= player\_position.x + player.FRAME\_WIDTH;

bool is\_over\_lap\_y = check\_position.y >= player\_position.y && check\_position.y <= player\_position.y + player.FRAME\_HEIGHT;

return is\_over\_lap\_x && is\_over\_lap\_y;

}

void Move(const Player& player) {

const POINT& player\_position = player.GetPosition();

int dir\_x = player\_position.x - position.x;

int dir\_y = player\_position.y - position.y;

double len\_dir = sqrt(dir\_x \* dir\_x + dir\_y \* dir\_y);

if (len\_dir != 0) {

double normalized\_x = dir\_x / len\_dir;

double normalized\_y = dir\_y / len\_dir;

position.x += (int)(SPEED \* normalized\_x);

position.y += (int)(SPEED \* normalized\_y);

}

if (dir\_x < 0) {

facing\_left = true;

}

else if (dir\_x > 0) {

facing\_left = false;

}

}

void Draw(int delta) {

int pos\_shadow\_x = position.x + (FRAME\_WIDTH / 2 - SHADOW\_WIDTH / 2);

int pos\_shadow\_y = position.y + FRAME\_HEIGHT - 35;

putimage\_alpha(pos\_shadow\_x, pos\_shadow\_y, &img\_shadow);

if (facing\_left)

anim\_left->Play(position.x, position.y, delta);

else

anim\_right->Play(position.x, position.y, delta);

}

~Enemy() {

delete anim\_left;

delete anim\_right;

}

void Hurt() {

alive = false;

}

bool CheckAlive() {

return alive;

}

private:

const int SPEED = 2;

const int FRAME\_WIDTH = 80;

const int FRAME\_HEIGHT = 80;

const int SHADOW\_WIDTH = 48;

private:

IMAGE img\_shadow;

Animition\* anim\_left;

Animition\* anim\_right;

POINT position = { 0,0 };

bool facing\_left = false;

bool alive = true;

};

class Button {

public:

Button(RECT rect, LPCTSTR path\_img\_idle, LPCTSTR path\_img\_hovered, LPCTSTR path\_img\_pushed) {

region = rect;

loadimage(&img\_idle, path\_img\_idle);

loadimage(&img\_hovered, path\_img\_hovered);

loadimage(&img\_pushed, path\_img\_pushed);

}

~Button() = default;

void ProcessEvent(const ExMessage& msg) {

switch (msg.message)

{

case WM\_MOUSEMOVE:

if (status == Status::Idle && CheckCursorHit(msg.x, msg.y))

status = Status::Hovered;

else if (status == Status::Hovered && !CheckCursorHit(msg.x, msg.y))

status = Status::Idle;

break;

case WM\_LBUTTONDOWN:

if (CheckCursorHit(msg.x, msg.y))

status = Status::Pushed;

break;

case WM\_LBUTTONUP:

if (status == Status::Pushed)

OnClick();

break;

default:

break;

}

}

void Draw() {

switch (status)

{

case Status::Idle:

putimage(region.left, region.top, &img\_idle);

break;

case Status::Hovered:

putimage(region.left, region.top, &img\_hovered);

break;

case Status::Pushed:

putimage(region.left, region.top, &img\_pushed);

break;

}

}

protected:

virtual void OnClick() = 0;

private:

enum class Status {

Idle = 0,

Hovered,

Pushed

};

private:

RECT region;

IMAGE img\_idle;

IMAGE img\_hovered;

IMAGE img\_pushed;

Status status = Status::Idle;

private:

bool CheckCursorHit(int x, int y) {

return x >= region.left && x <= region.right && y >= region.top && y <= region.bottom;

}

};

class StartGameButton : public Button {

public:

StartGameButton(RECT rect, LPCTSTR path\_img\_idle, LPCTSTR path\_img\_hovered, LPCTSTR path\_img\_pushed)

: Button(rect, path\_img\_idle, path\_img\_hovered, path\_img\_pushed) {}

~StartGameButton() = default;

protected:

void OnClick() {

is\_game\_started = true;

mciSendString(\_T("play bgm repeat from 0"), NULL, 0, NULL);

}

};

class QuitGameButton : public Button {

public:

QuitGameButton(RECT rect, LPCTSTR path\_img\_idle, LPCTSTR path\_img\_hovered, LPCTSTR path\_img\_pushed)

: Button(rect, path\_img\_idle, path\_img\_hovered, path\_img\_pushed) {}

~QuitGameButton() = default;

protected:

void OnClick() {

running = false;

}

};

void TryGenerateEnemy(std::vector<Enemy\*>& enemy\_list) {

const int INTERVAL = 200;

static int counter = 0;

if ((++counter) % INTERVAL == 0) {

enemy\_list.push\_back(new Enemy());

}

}

void UpdateBullets(std::vector<Bullet>& bullet\_list, const Player& player) {

const double RADIAL\_SPEED = 0.0045;

const double TANGENT\_SPEED = 0.0055;

double radian\_interval = 2 \* 3.14159 / bullet\_list.size();

POINT player\_position = player.GetPosition();

double radius = 100 + 25 \* sin(GetTickCount() \* RADIAL\_SPEED);

for (size\_t i = 0; i < bullet\_list.size(); i++) {

double radian = GetTickCount() \* TANGENT\_SPEED + radian\_interval \* i;

bullet\_list[i].position.x = player\_position.x + player.FRAME\_WIDTH / 2 + (int)(radius \* sin(radian));

bullet\_list[i].position.y = player\_position.y + player.FRAME\_HEIGHT / 2 + (int)(radius \* cos(radian));

}

}

void DrawPlayerScore(int score) {

static TCHAR text[64];

\_stprintf\_s(text, \_T("当前玩家得分：%d"),score);

setbkmode(TRANSPARENT);

settextcolor(RGB(255, 85, 185));

outtextxy(10, 10, text);

}

int main() {

initgraph(1280, 720);

atlas\_player\_left = new Atlas(\_T("img/player\_left\_%d.png"), 6);

atlas\_player\_right = new Atlas(\_T("img/player\_right\_%d.png"), 6);

atlas\_enemy\_left = new Atlas(\_T("img/enemy\_left\_%d.png"), 6);

atlas\_enemy\_right = new Atlas(\_T("img/enemy\_right\_%d.png"), 6);

mciSendString(\_T("open mus/bgm.mp3 alias bgm"), NULL, 0, NULL);

mciSendString(\_T("open mus/hit.wav alias hit"), NULL, 0, NULL);

int score = 0;

Player player;

ExMessage msg;

IMAGE img\_menu;

IMAGE img\_background;

std::vector<Enemy\*> enemy\_list;

std::vector<Bullet> bullet\_list(3);

RECT region\_btn\_start\_game, region\_btn\_quit\_game;

region\_btn\_start\_game.left = (WINDOW\_WIDTH - BUTTON\_WIDTH) / 2;

region\_btn\_start\_game.right = region\_btn\_start\_game.left + BUTTON\_WIDTH;

region\_btn\_start\_game.top = 430;

region\_btn\_start\_game.bottom = region\_btn\_start\_game.top + BUTTON\_HEIGHT;

region\_btn\_quit\_game.left = (WINDOW\_WIDTH - BUTTON\_WIDTH) / 2;

region\_btn\_quit\_game.right = region\_btn\_quit\_game.left + BUTTON\_WIDTH;

region\_btn\_quit\_game.top = 550;

region\_btn\_quit\_game.bottom = region\_btn\_quit\_game.top + BUTTON\_HEIGHT;

StartGameButton btn\_start\_game = StartGameButton(region\_btn\_start\_game,

\_T("img/ui\_start\_idle.png"), \_T("img/ui\_start\_hovered.png"), \_T("img/ui\_start\_pushed.png"));

QuitGameButton btn\_quit\_game = QuitGameButton(region\_btn\_quit\_game,

\_T("img/ui\_quit\_idle.png"), \_T("img/ui\_quit\_hovered.png"), \_T("img/ui\_quit\_pushed.png"));

loadimage(&img\_menu, \_T("img/menu.png"));

loadimage(&img\_background, \_T("img/background.png"));

BeginBatchDraw();

while (running) {

DWORD start\_time = GetTickCount();

while (peekmessage(&msg)) {

if(is\_game\_started)

player.ProcessEvent(msg);

else {

btn\_start\_game.ProcessEvent(msg);

btn\_quit\_game.ProcessEvent(msg);

}

}

if (is\_game\_started) {

TryGenerateEnemy(enemy\_list);

player.Move();

UpdateBullets(bullet\_list, player);

for (Enemy\* enemy : enemy\_list)

enemy->Move(player);

for (Enemy\* enemy : enemy\_list) {

for (const Bullet& bullet : bullet\_list) {

if (enemy->CheckBulletCollision(bullet)) {

mciSendString(\_T("play hit from 0"), NULL, 0, NULL);

enemy->Hurt();

score++;

}

}

}

for (size\_t i = 0; i < enemy\_list.size(); i++) {

Enemy\* enemy = enemy\_list[i];

if (!enemy->CheckAlive()) {

std::swap(enemy\_list[i], enemy\_list.back());

enemy\_list.pop\_back();

delete enemy;

}

}

for (Enemy\* enemy : enemy\_list) {

if (enemy->CheckPlayerCollision(player)) {

static TCHAR text[128];

\_stprintf\_s(text, \_T("最终得分：%d！"), score);

MessageBox(GetHWnd(), \_T("扣“1”观看战败CG"), \_T("游戏结束"), MB\_OK);

running = false;

break;

}

}

}

cleardevice();

if (is\_game\_started) {

putimage(0, 0, &img\_background);

player.Draw(1000 / 144);

for (Enemy\* enemy : enemy\_list)

enemy->Draw(1000 / 144);

for (const Bullet& bullet : bullet\_list)

bullet.Draw();

DrawPlayerScore(score);

}

else {

putimage(0, 0, &img\_menu);

btn\_start\_game.Draw();

btn\_quit\_game.Draw();

}

FlushBatchDraw();

DWORD end\_time = GetTickCount();

DWORD delta\_time = end\_time = start\_time;

if (delta\_time < 1000 / 144) {

Sleep(1000 / 144 - delta\_time);

}

}

delete atlas\_player\_left;

delete atlas\_player\_right;

delete atlas\_enemy\_left;

delete atlas\_enemy\_right;

EndBatchDraw();

return 0;

}