#include <windows.h>

#include <graphics.h>

#include <conio.h>

#include <string>

#include <vector>

#include <ctime>

#include <mmsystem.h>

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

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

using namespace std;

const int GAME\_OVER\_DELAY = 30; // 额外延迟帧数

int gameOverDelayCounter = 0;

IMAGE imgBg;

// 透明背景色

const COLORREF TRANSPARENT\_COLOR = RGB(186, 254, 202);

const int COLOR\_TOLERANCE = 20;

// 定义动画帧结构

struct Animation {

IMAGE\* frames;

int frameCount;

int currentFrame;

int frameDelay;

int frameCounter;

};

// 游戏对象结构体

struct GameObject {

int x, y;

int width, height;

Animation anim;

bool active;

};

struct Bullet : public GameObject {

int speed;

Bullet(int x, int y, int w, int h, Animation anim, bool active, int spd) {

this->x = x;

this->y = y;

this->width = w;

this->height = h;

this->anim = anim;

this->active = active;

this->speed = spd;

}

};

struct Enemy : public GameObject {

int speed;

Enemy(int x, int y, int w, int h, Animation anim, bool active, int spd) {

this->x = x;

this->y = y;

this->width = w;

this->height = h;

this->anim = anim;

this->active = active;

this->speed = spd;

}

};

struct Explosion : public GameObject {

int life;

Explosion(int x, int y, int w, int h, Animation anim, bool active, int life) {

this->x = x;

this->y = y;

this->width = w;

this->height = h;

this->anim = anim;

this->active = active;

this->life = life;

}

};

const int WIN\_WIDTH = 480;

const int WIN\_HEIGHT = 573;

// 全局资源

Animation playerAnim;

Animation enemyAnim;

Animation bulletAnim;

Animation explosionAnim;

// 音效

const wchar\_t\* FIRE\_SOUND = L"fire.wav";

const wchar\_t\* EXPLODE\_SOUND = L"explode.wav";

const wchar\_t\* GAME\_OVER\_SOUND = L"gameover.wav";

// 检查颜色是否匹配背景色（带容差）

bool IsBackgroundColor(COLORREF color) {

int r = GetRValue(color);

int g = GetGValue(color);

int b = GetBValue(color);

int bgR = GetRValue(TRANSPARENT\_COLOR);

int bgG = GetGValue(TRANSPARENT\_COLOR);

int bgB = GetBValue(TRANSPARENT\_COLOR);

int diff = abs(r - bgR) + abs(g - bgG) + abs(b - bgB);

return diff <= COLOR\_TOLERANCE;

}

// 使用AlphaBlend绘制带透明效果的图像

void DrawTransparentImage(int x, int y, IMAGE\* srcImg) {

// 创建临时图像

IMAGE tempImg;

tempImg.Resize(srcImg->getwidth(), srcImg->getheight());

// 获取图像数据

DWORD\* pSrc = GetImageBuffer(srcImg);

DWORD\* pTemp = GetImageBuffer(&tempImg);

int width = srcImg->getwidth();

int height = srcImg->getheight();

// 处理透明色

for (int i = 0; i < width \* height; i++) {

if (IsBackgroundColor(BGR(pSrc[i]))) {

pTemp[i] = 0; // 完全透明

}

else {

pTemp[i] = pSrc[i] | 0xFF000000; // 不透明

}

}

// 设置混合参数

BLENDFUNCTION bf;

bf.BlendOp = AC\_SRC\_OVER;

bf.BlendFlags = 0;

bf.SourceConstantAlpha = 255;

bf.AlphaFormat = AC\_SRC\_ALPHA;

// 执行Alpha混合绘制

HDC dstDC = GetImageHDC();

HDC srcDC = GetImageHDC(&tempImg);

AlphaBlend(dstDC, x, y, width, height, srcDC, 0, 0, width, height, bf);

}

// 加载动画资源

void LoadAnimation(Animation& anim, const wstring& basePath, int count, int delay) {

anim.frames = new IMAGE[count];

anim.frameCount = count;

anim.currentFrame = 0;

anim.frameDelay = delay;

anim.frameCounter = 0;

for (int i = 0; i < count; i++) {

wstring path = basePath + to\_wstring(i + 1) + L".png";

// 检查文件是否存在

if (GetFileAttributes(path.c\_str()) == INVALID\_FILE\_ATTRIBUTES) {

wstring msg = L"图像文件不存在: " + path;

MessageBox(NULL, msg.c\_str(), \_T("错误"), MB\_ICONERROR);

exit(1);

}

// 尝试加载图像

try {

loadimage(&anim.frames[i], path.c\_str());

}

catch (...) {

wstring msg = L"加载图像失败: " + path;

MessageBox(NULL, msg.c\_str(), \_T("错误"), MB\_ICONERROR);

exit(1);

}

}

}

// 更新动画

void UpdateAnimation(Animation& anim) {

anim.frameCounter++;

if (anim.frameCounter >= anim.frameDelay) {

anim.frameCounter = 0;

anim.currentFrame = (anim.currentFrame + 1) % anim.frameCount;

}

}

// 绘制游戏对象

void DrawGameObject(const GameObject& obj) {

if (obj.active) {

DrawTransparentImage(obj.x, obj.y, &obj.anim.frames[obj.anim.currentFrame]);

}

}

// 初始化游戏

void InitGame() {

// 加载动画

LoadAnimation(playerAnim, L"plane\_anim", 2, 5);

LoadAnimation(enemyAnim, L"enemy\_anim", 2, 8);

LoadAnimation(bulletAnim, L"bullet\_anim", 2, 1);

LoadAnimation(explosionAnim, L"explosion\_anim", 5, 5);

// 预加载音效

PlaySound(NULL, NULL, SND\_ASYNC); // 停止任何正在播放的音效

}

// 碰撞检测（基于像素精确检测）

bool CheckCollision(const GameObject& obj1, const GameObject& obj2) {

if (!obj1.active || !obj2.active) return false;

// 粗略检测边界框

if (obj1.x + obj1.width < obj2.x || obj1.x > obj2.x + obj2.width ||

obj1.y + obj1.height < obj2.y || obj1.y > obj2.y + obj2.height) {

return false;

}

// 精确像素检测

int x1 = max(obj1.x, obj2.x);

int x2 = min(obj1.x + obj1.width, obj2.x + obj2.width);

int y1 = max(obj1.y, obj2.y);

int y2 = min(obj1.y + obj1.height, obj2.y + obj2.height);

IMAGE& img1 = obj1.anim.frames[obj1.anim.currentFrame];

IMAGE& img2 = obj2.anim.frames[obj2.anim.currentFrame];

DWORD\* pBuf1 = GetImageBuffer(&img1);

DWORD\* pBuf2 = GetImageBuffer(&img2);

for (int y = y1; y < y2; y++) {

for (int x = x1; x < x2; x++) {

int ox1 = x - obj1.x;

int oy1 = y - obj1.y;

int ox2 = x - obj2.x;

int oy2 = y - obj2.y;

COLORREF c1 = BGR(pBuf1[oy1 \* img1.getwidth() + ox1]);

COLORREF c2 = BGR(pBuf2[oy2 \* img2.getwidth() + ox2]);

if (!IsBackgroundColor(c1) && !IsBackgroundColor(c2)) {

return true;

}

}

}

return false;

}

int main() {

initgraph(WIN\_WIDTH, WIN\_HEIGHT);

setbkcolor(WHITE);

cleardevice();

loadimage(&imgBg, \_T("background.png"));

// 初始化游戏

InitGame();

BeginBatchDraw();

// 创建玩家

GameObject player = {

WIN\_WIDTH / 2 - playerAnim.frames[0].getwidth() / 2,

WIN\_HEIGHT - 80,

playerAnim.frames[0].getwidth(),

playerAnim.frames[0].getheight(),

playerAnim,

true

};

vector<Bullet> bullets;

vector<Enemy> enemies;

vector<Explosion> explosions;

int score = 0;

int enemyTimer = 0;

bool gameOver = false;

bool gamePaused = false;

int bulletCooldown = 0;

srand((unsigned)time(0));

while (!gameOver) {

// 处理输入

if (GetAsyncKeyState(VK\_LEFT) & 0x8000 && player.x > 0) player.x -= 8;

if (GetAsyncKeyState(VK\_RIGHT) & 0x8000 && player.x < WIN\_WIDTH - player.width) player.x += 8;

if (GetAsyncKeyState(VK\_UP) & 0x8000 && player.y > 0) player.y -= 8;

if (GetAsyncKeyState(VK\_DOWN) & 0x8000 && player.y < WIN\_HEIGHT - player.height) player.y += 8;

// 暂停游戏

if (GetAsyncKeyState('P') & 0x8000) {

gamePaused = !gamePaused;

Sleep(200); // 防抖

}

if (gamePaused) {

settextcolor(BLACK);

outtextxy(WIN\_WIDTH / 2 - 50, WIN\_HEIGHT / 2, \_T("游戏暂停"));

FlushBatchDraw();

Sleep(50);

continue;

}

// 发射子弹

if (bulletCooldown > 0) bulletCooldown--;

if ((GetAsyncKeyState(VK\_SPACE) & 0x8000) && bulletCooldown == 0) {

Bullet bullet = {

player.x + player.width / 2 - bulletAnim.frames[0].getwidth() / 2,

player.y,

bulletAnim.frames[0].getwidth(),

bulletAnim.frames[0].getheight(),

bulletAnim,

true,

10

};

bullets.push\_back(bullet);

bulletCooldown = 20;

PlaySound(FIRE\_SOUND, NULL, SND\_ASYNC | SND\_FILENAME);

}

// ESC退出

if (GetAsyncKeyState(VK\_ESCAPE) & 0x8000) break;

// 敌机生成

enemyTimer++;

if (enemyTimer > 30) {

Enemy enemy = {

rand() % (WIN\_WIDTH - enemyAnim.frames[0].getwidth()),

0,

enemyAnim.frames[0].getwidth(),

enemyAnim.frames[0].getheight(),

enemyAnim,

true,

3 + rand() % 3

};

enemies.push\_back(enemy);

enemyTimer = 0;

}

// 更新动画

UpdateAnimation(player.anim);

for (auto& b : bullets) UpdateAnimation(b.anim);

for (auto& e : enemies) UpdateAnimation(e.anim);

for (auto& exp : explosions) {

UpdateAnimation(exp.anim);

exp.life--;

if (exp.life <= 0) exp.active = false;

}

// 子弹移动

for (auto& b : bullets) b.y -= b.speed;

bullets.erase(remove\_if(bullets.begin(), bullets.end(),

[](const Bullet& b) { return b.y < 0 || !b.active; }), bullets.end());

// 敌机移动

for (auto& e : enemies) e.y += e.speed;

enemies.erase(remove\_if(enemies.begin(), enemies.end(),

[](const Enemy& e) { return e.y > WIN\_HEIGHT || !e.active; }), enemies.end());

// 碰撞检测

for (auto it = enemies.begin(); it != enemies.end();) {

bool hit = false;

for (auto bit = bullets.begin(); bit != bullets.end();) {

if (CheckCollision(\*it, \*bit)) {

// 创建爆炸效果

Explosion exp = {

it->x + it->width / 2 - explosionAnim.frames[0].getwidth() / 2,

it->y + it->height / 2 - explosionAnim.frames[0].getheight() / 2,

explosionAnim.frames[0].getwidth(),

explosionAnim.frames[0].getheight(),

explosionAnim,

true,

explosionAnim.frameCount \* explosionAnim.frameDelay

};

explosions.push\_back(exp);

bit->active = false;

it->active = false;

hit = true;

score += 10;

PlaySound(EXPLODE\_SOUND, NULL, SND\_ASYNC | SND\_FILENAME);

break;

}

else {

++bit;

}

}

if (hit) {

it = enemies.erase(it);

}

else if (CheckCollision(\*it, player) && player.active) {

// 玩家飞机爆炸

Explosion exp = {

player.x + player.width / 2 - explosionAnim.frames[0].getwidth() / 2,

player.y + player.height / 2 - explosionAnim.frames[0].getheight() / 2,

explosionAnim.frames[0].getwidth(),

explosionAnim.frames[0].getheight(),

explosionAnim,

true,

explosionAnim.frameCount \* explosionAnim.frameDelay

};

explosions.push\_back(exp);

player.active = false; // 禁用玩家控制

PlaySound(GAME\_OVER\_SOUND, NULL, SND\_ASYNC | SND\_FILENAME);

// 不立即设置 gameOver = true

}

else {

++it;

}

}

// 更新爆炸动画的代码保持不变

for (auto& exp : explosions) {

UpdateAnimation(exp.anim);

exp.life--;

if (exp.life <= 0) exp.active = false;

}

// 清理已完成的爆炸

explosions.erase(std::remove\_if(explosions.begin(), explosions.end(),

[](const Explosion& exp) { return !exp.active; }), explosions.end());

// 清理无效的爆炸效果

/\*explosions.erase(remove\_if(explosions.begin(), explosions.end(),

[](const Explosion& exp) { return !exp.active; }), explosions.end());\*/

// 绘制

cleardevice();

putimage(0, 0, &imgBg);

// 绘制游戏对象

DrawGameObject(player);

for (const auto& b : bullets) DrawGameObject(b);

for (const auto& e : enemies) DrawGameObject(e);

for (const auto& exp : explosions) DrawGameObject(exp);

LOGFONT font = { 0 };

font.lfHeight = 36; // 字体高度

font.lfWidth = 0; // 自动计算宽度

font.lfWeight = FW\_BOLD; // 加粗 (FW\_NORMAL 为正常)

wcscpy\_s(font.lfFaceName, \_T("黑体")); // 字体名称

settextstyle(&font); // 应用字体样式

settextcolor(BLUE); // 蓝色文字

// 绘制UI

settextcolor(BLACK);

setbkmode(TRANSPARENT);

TCHAR s[32];

\_stprintf\_s(s, \_T("分数: %d"), score);

outtextxy(10, 540, s);

if (gamePaused) {

settextcolor(BLUE);

outtextxy(WIN\_WIDTH / 2 - 50, WIN\_HEIGHT / 2, \_T("游戏暂停"));

}

// 在主循环中

if (!player.active && explosions.empty()) {

if (++gameOverDelayCounter >= GAME\_OVER\_DELAY) {

gameOver = true;

}

}

FlushBatchDraw();

Sleep(20);

if (!player.active && explosions.empty()) {

gameOver = true; // 所有爆炸动画播放完毕后才结束游戏

}

}

// 游戏结束画面

settextcolor(BLACK);

outtextxy(WIN\_WIDTH / 2 - 80, WIN\_HEIGHT / 2 + 120, \_T("游戏结束!"));

TCHAR finalScore[32];

\_stprintf\_s(finalScore, \_T("最终分数: %d"), score);

outtextxy(WIN\_WIDTH / 2 - 80, WIN\_HEIGHT / 2 + 155, finalScore);

/\*if (!player.active) {

settextcolor(RED);

outtextxy(WIN\_WIDTH / 2 - 100, WIN\_HEIGHT / 2 + 60, \_T("按任意键退出"));

}\*/

FlushBatchDraw();

\_getch();

// 释放资源

EndBatchDraw();

delete[] playerAnim.frames;

delete[] enemyAnim.frames;

delete[] bulletAnim.frames;

delete[] explosionAnim.frames;

closegraph();

return 0;

}