南开大学 工科试验班

姓名 王斯毅

学号 2312999

班级 人工智能3-3班

2024年5月10日

高级语言程序设计

实验报告

目录

[高级语言程序设计大作业实验报告 2](#_Toc17650)

[一. 作业题目 2](#_Toc32368)

[二. 开发软件 2](#_Toc2015)

[三. 课题要求 2](#_Toc1032)

[四. 主要流程 2](#_Toc22430)

[1． 整体流程 2](#_Toc4191)

[2． 算法或公式 27](#_Toc20422)

[（1） 两点间距离计算公式 27](#_Toc7200)

[（2） 勾股定理 27](#_Toc7364)

[3． 单元测试 27](#_Toc9519)

[五. 单元测试 27](#_Toc654)

[六. 收获 27](#_Toc14445)

高级语言程序设计大作业实验报告

1. **作业题目**

制作一款考验身法的割草小游戏。

名字：肘术回战之肘术高专的幸存者

1. **开发软件**

Visual Studio 2022

1. **课题要求**
2. 面向对象。
3. 单元测试。
4. 模型部分
5. 验证
6. **主要流程**
   1. **整体流程**

实现思路：（代码基于easyx平台）  
首先考虑两个主要部分，就是玩家类（Player类）和敌人类（Enemy类）的实现

Player类的实现如下：

//定义Player类

class Player

{

public:

Player()

{

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

anim\_left = new Animation(atlas\_player\_left,45);

anim\_right = new Animation(atlas\_player\_right,45);

init\_bullet(bullet\_list,circle\_bullet\_list);

};//首先是Player类的构造函数，它需要生成Player的向左和向右的动画，并且对玩家的子弹进行初始化

~Player()

{

delete anim\_left;

delete anim\_right;

};//这是Player类的析构函数，因为在构造函数时用new关键字来创建anim\_left和anim\_right，所以在析构函数中要删除anim\_left和anim\_right

void processevent(const ExMessage& msg)

{

switch (msg.message)

{

case WM\_KEYDOWN:

switch (msg.vkcode)

{

case 'W':

is\_move\_up = true;

break;

case 'A':

is\_move\_left = true;

break;

case 'S':

is\_move\_down = true;

break;

case 'D':

is\_move\_right = true;

break;

case VK\_SPACE:

skill = true;

break;

default:

break;

}

break;

case WM\_KEYUP:

switch (msg.vkcode)

{

case 'W':

is\_move\_up = false;

break;

case 'A':

is\_move\_left = false;

break;

case 'S':

is\_move\_down = false;

break;

case 'D':

is\_move\_right = false;

break;

case VK\_SPACE:

skill = false;

break;

default:

break;

}

break;

}

};//这一部分是根据键盘信息改变玩家相应状态的函数，当键盘上对应按键被按下时，Player类中对应的成员就会改变，比如按下A键时，Player类中is\_move\_left就会被改为right，于是玩家就会向左移动

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);//勾股定理

if (len\_dir != 0)

{

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

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

player\_position.x += (int)(speed \* normalized\_x);

player\_position.y += (int)(speed \* normalized\_y);

}//这前半部分是用来使玩家移动的，如果dir\_x>0，那么玩家就是向右移动，如果dir\_x<0，那么就是向左移动，dir\_y同理。引入的dir\_len是为了标准化x和y方向上的位移，避免分类讨论

if (player\_position.x < 0)

{

player\_position.x = 0;

}

if (player\_position.y < 0)

{

player\_position.y = 0;

}

if (player\_position.x + frame\_width > window\_width)

{

player\_position.x = window\_width - frame\_width;

}

if (player\_position.y + frame\_height > window\_height)

{

player\_position.y = window\_height - frame\_height;

}

};//下半部分是为了检测玩家位置将玩家限制在窗口中

void draw(int delta)

{

int pos\_shadow\_x = player\_position.x + (frame\_width / 2 - shadow\_width / 2);

int pos\_shadow\_y = player\_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(player\_position.x, player\_position.y, delta);

}

else

{

anim\_right->play(player\_position.x, player\_position.y, delta);

}

};//这个函数是依据玩家的朝向来播放对应的动画，如果dir\_x<0，那么玩家就是向左移动，朝向就是左，于是就要播放向左的动画，反之播放向右的动画

const POINT& get\_position() const

{

return player\_position;

}//这个函数是用来返回玩家位置的，因为玩家的位置在其它函数或类中会有用到（例如检测子弹和玩家是否碰撞）

void init\_bullet(std::vector<Bullet>& bullet\_list,std::vector<circle\_bullet>&circle\_bullet\_list)

{

for (int i = 0; i < 2; i++)

{

Bullet new\_bullet;

circle\_bullet new\_circle\_bullet;

bullet\_list.push\_back(new\_bullet);

circle\_bullet\_list.push\_back(new\_circle\_bullet);

}

}//这是初始化玩家子弹的函数，它将玩家的子弹初始化为两个螺旋子弹和两个圆周子弹

void add\_bullet(std::vector<Bullet>&bullet\_list,std::vector<circle\_bullet>&circle\_bullet\_list)

{

int i = rand() % 2;

if (i == 0)

{

Bullet new\_bullet;

bullet\_list.push\_back(new\_bullet);

}

else

{

circle\_bullet new\_bullet;

circle\_bullet\_list.push\_back(new\_bullet);

}

}//这是增加玩家子弹的函数，后面在判断玩家等级是否增长时会用到，若玩家等级增长，则随机增加一种子弹的数量

bool release\_skill()

{

return skill && sp == 100;

}//判断玩家是否满足释放技能的函数，当且仅当玩家按下空格键且玩家能量值为100时才返回true

public:

const int speed = 3;//玩家移动速度

const int frame\_width = 55;//玩家图形宽度

const int frame\_height = 92;//玩家图形宽度

const int shadow\_width = 32;//阴影宽度

const int window\_width = 1280;//窗口宽度

const int window\_height = 720;//窗口高度

std::vector<Bullet> bullet\_list;

std::vector<circle\_bullet> circle\_bullet\_list;

int grade = 1;//定义等级

int hp = 100;//定义血量

int sp = 0;//定义能量

private:

IMAGE img\_shadow;

Animation\* anim\_left;

Animation\* anim\_right;

POINT player\_position = { 500,500 };//玩家初始位置

bool is\_move\_up = false;

bool is\_move\_down = false;

bool is\_move\_left = false;

bool is\_move\_right = false;

bool skill = false;

friend Bullet;

friend circle\_bullet;

};

然后是Enemy类的实现

//定义Enemy类

class Enemy

{

public:

Enemy()

{

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

anim\_left = new Animation(atlas\_enemy\_left, 45);

anim\_right = new Animation(atlas\_enemy\_right, 45);

//前半部分是和Player类相同的创建anim\_left和anim\_right

//敌人生成边界

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;

}//依据edge的值确定敌人生成的边界

birth\_time = GetTickCount();//在敌人生成时调用GetTickCount函数，在后面检测敌人是否需要发射子弹时提供时间差

};

bool check\_bullet\_collision(const Bullet& bullet)

{

bool is\_overlap\_x = bullet.position.x >= position.x && bullet.position.x <= position.x + frame\_width;

bool is\_overlap\_y = bullet.position.y >= position.y && bullet.position.y <= position.y + frame\_height;

return is\_overlap\_x && is\_overlap\_y;

}//检测敌人和玩家子弹是否碰撞

bool check\_player\_collision(const Player& player)

{

POINT check\_position = { position.x + frame\_width / 2,position.y + frame\_height / 2 };

POINT player\_position = player.get\_position();

bool is\_overlap\_x = check\_position.x >= player\_position.x && check\_position.x <= player\_position.x + player.frame\_width;

bool is\_overlap\_y = check\_position.y >= player\_position.y && check\_position.y <= player\_position.y + player.frame\_height;

return is\_overlap\_x && is\_overlap\_y;

}//检测敌人和玩家是否碰撞，将敌人抽象为一个点，当这个点的位置落在玩家的区域内时，才会判定为玩家和敌人相碰（不这么做碰撞箱会出问题）

void move(const Player& player)

{

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

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;

}

}//和Player类move函数差不多的move函数，只不过敌人会朝着玩家的位置移动，所以要传入玩家位置参数

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);

}

}//和Player类中draw函数差不多的draw函数

~Enemy()

{

delete anim\_left;

delete anim\_right;

};//和Player类一样的析构函数

void hurt()

{

alive = false;

};//如果敌人和子弹碰撞，便会将alive变为false

bool checkalive()

{

return alive;

};//检测敌人是否存活，方便后面删去敌人

bool check\_time()

{

DWORD time = GetTickCount();

DWORD delta\_time = time - birth\_time;

if (delta\_time != 0 && delta\_time % 3000 == 0)

{

return true;

}

return false;

}//检测敌人是否达到发射子弹的条件，设定敌人每3秒发射一颗子弹（这里似乎还有点bug，找不出来）

const POINT& get\_position() const

{

return position;

}//提供敌人位置的函数（和Player类一样）

private:

const int speed = 2;//敌人速度

const int frame\_width = 80;//敌人宽度

const int frame\_height = 80;//敌人高度

const int shadow\_width = 48;//敌人阴影宽度

public:

IMAGE img\_shadow;

Animation\* anim\_left;

Animation\* anim\_right;

POINT position = { 0,0 };

bool facing\_left = false;

bool alive = true;

DWORD birth\_time;

};

这两部分都有一个共同点，它们都需要循环播放动画，于是我们在此基础上定义一个图集类（Atlas类）。Atlas类是用来存储玩家和敌人动画的类。以下是实现方法：

//定义Atlas类

class Atlas

{

public:

Atlas(LPCTSTR path,int num)

{

TCHAR path\_file[256];

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

{

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

IMAGE\* frame = new IMAGE();

loadimage(frame, path\_file);

frame\_list.push\_back(frame);

}

};//构造函数，输入文件路径和数量构建图集

~Atlas()

{

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

{

delete frame\_list[i];

}

};//析构函数

public:

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

};

而动画类（Animation类）负责接受对应图集类（Atlas类）并实现动画的循环播放。以下是实现方法：  
//定义Animation类

class Animation

{

public:

Animation(Atlas\* atlas, int interval)

{

interval\_ms = interval;

anim\_atlas = atlas;

};//构造函数，传入图集并设定播放间隔

~Animation() = 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]);

};//如果计时器大于播放间隔就切换到下一张图片并重置计时器，借此完成动画的循环播放

private:

int interval\_ms = 0;

int timer = 0;//计时器

int idx\_frame = 0;//动画帧数

private:

Atlas\* anim\_atlas;

};

这样，有关玩家和敌人大部分代码就都完成了，接下来就是子弹类（Bullet类）的实现以及Bullet类所派生出的其他类的实现：

//定义Bullet类

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);

};//设置边界颜色，填充颜色，并且画一个填充的圆

private:

const int RADIUS = 10;//圆的半径

};

//定义圆周子弹

class circle\_bullet :public Bullet

{

public:

void draw()const

{

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

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

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

};

private:

const int RADIUS = 10;

};//我也不知道为什么要这么写

//定义敌人子弹类

class enemy\_bullet :public Bullet

{

public:

void init\_nor\_x\_and\_nor\_y(POINT& enemy\_position,POINT &player\_position)

{

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

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

double dir\_len = sqrt(pow(dir\_x, 2) + pow(dir\_y, 2));

normalized\_x = dir\_x / dir\_len;

normalized\_y = dir\_y / dir\_len;

};//传入玩家和敌人的位置初始化子弹的方向

void set\_position(POINT& enemy\_position)

{

position.x = enemy\_position.x + 40;

position.y = enemy\_position.y + 40;

};//将子弹生成在敌人的中心

void draw()const

{

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

setfillcolor(RGB(255, 0, 0));

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

};//设置敌人子弹颜色并画一个填充的圆

//检测子弹撞墙

bool check\_collision()

{

bool is\_collision\_x = position.x<0 || position.x>window\_width;

bool is\_collision\_y = position.y<0 || position.y>window\_height;

return is\_collision\_x || is\_collision\_y;

};

//检测与玩家相撞

bool check\_player\_collision(Player& player)

{

POINT player\_position = player.get\_position();

bool is\_collision\_x = position.x > player\_position.x + 5 && position.x < player\_position.x + 35;

bool is\_colloision\_y= position.y > player\_position.y && position.y < player\_position.y + 80;

return is\_collision\_x && is\_colloision\_y;

}

public:

double normalized\_x;

double normalized\_y;

private:

const int RADIUS = 10;

};

然后是游戏按钮类（Button类）以及所派生出的开始游戏按钮和推出游戏按钮类的实现：  
//定义Button类

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);

};//构造函数，需要传入一个矩形，和按钮三个状态（默认，悬停，按下）的路径

~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;

}

};//依据鼠标的位置和点击状态来切换对应的状态，不在矩形区域就是默认状态，在矩形区域内但未点击就是悬停状态，在矩形区域并且点击鼠标左键就进入点击状态，当处于点击状态时，就要执行自己的OnClick函数

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;

}

};//依据状态切换对应的图片

protected:

//所有继承Button类的类都必须实现自己的OnClick函数

virtual void OnClick() = 0;

private:

//枚举Button的状态

enum class Status

{

Idle=0,

Hovered,

Pushed

};

private:

RECT region;

IMAGE img\_idle;

IMAGE img\_hovered;

IMAGE img\_pushed;

Status status=Status::Idle;

private:

//检测鼠标点击

bool CheckCursorHit(int x, int y)

{

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

};

};

//开始游戏按钮

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) {}

//构造函数，和基类一样

~StartGameButton() = default;

protected:

void OnClick()

{

is\_game\_started = true;

};//如果开始按钮被点击，那么游戏就开始了

};

//退出游戏按钮

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) {}

//构造函数，和基类一样

~QuitGameButton() = default;

protected:

void OnClick()

{

running = false;

};//如果结束按钮被点击，那么就结束游戏

};

至此，我们就完成了游戏中所有的类，但是还缺少一些函数，比如更新子弹位置的函数，生成敌人的函数等等，以下是这些函数的实现：

//将素材中透明色背景删去的函数

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 });

}

//生成敌人的函数，生成速度随着玩家的等级上升而增加

void generate\_enemy(std::vector<Enemy\*>& enemy\_list,Player &player)

{

const int interval = max(100 - 3 \* player.grade, 10);

static int counter = 0;

if ((++counter) % interval == 0)

{

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

}

}

//更新螺旋子弹位置

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

{

const double radial\_speed = 0.0045;//径向速度

const double tangent\_speed = 0.0055;//切向速度

double radian\_interval = 2 \* pi / bullet\_list.size();//子弹之间的弧度间隔

POINT player\_position = player.get\_position();

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;//当前子弹所在弧度

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));

}

}

//更新圆周子弹位置

void update\_bullets(std::vector<circle\_bullet>& circle\_bullet\_list, const Player& player)

{

const double tangent\_speed = 0.002;//切向速度

double radian\_interval = 2 \* pi / circle\_bullet\_list.size();//子弹间隔弧度

POINT player\_position = player.get\_position();

double radius = 100;

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

{

double radian = GetTickCount() \* tangent\_speed + radian\_interval \* i;//子弹所在弧度

circle\_bullet\_list[i].position.x = player\_position.x + player.frame\_width / 2 + (int)(radius \* sin(radian));

circle\_bullet\_list[i].position.y = player\_position.y + player.frame\_height / 2 + (int)(radius \* cos(radian));

}

}

//更新敌人子弹位置

void update\_bullets(std::vector<enemy\_bullet\*>& enemy\_bullet\_list)

{

const double speed = 5;//子弹速度

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

{

enemy\_bullet\_list[i]->position.x += enemy\_bullet\_list[i]->normalized\_x \* speed;

enemy\_bullet\_list[i]->position.y += enemy\_bullet\_list[i]->normalized\_y \* speed;

}

}

//绘制玩家得分

void draw\_player\_score(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);

}

//绘制玩家等级

void draw\_player\_grade(int grade)

{

static TCHAR text[64];

\_stprintf\_s(text, \_T("当前玩家等级:%d"), grade);

setbkmode(TRANSPARENT);

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

outtextxy(10, 30, text);

}

//绘制玩家血量

void draw\_player\_hp(Player& player)

{

setlinecolor(RGB(255, 255, 255));

setfillcolor(RGB(255, 0, 0));

rectangle(10, 50, 210, 60);

fillrectangle(10, 50, 10 + 2\*player.hp, 60);

}

//绘制玩家能量条

void draw\_player\_sp(Player& player)

{

setlinecolor(RGB(255, 255, 255));

setfillcolor(RGB(0, 0, 255));

rectangle(10, 80, 210, 90);

fillrectangle(10, 80, min(10 + 2 \* player.sp,210), 90);

}

//绘制玩家释放技能动画（清除全屏子弹并且回复生命值）

void draw\_skill(Player& player,std::vector<enemy\_bullet\*>&enemy\_bullet\_list)

{

player.hp = min(100, player.hp + 10);

for (auto& bullet : enemy\_bullet\_list)

{

delete bullet;

}

enemy\_bullet\_list.clear();

}

于是，我们就完成了所有函数的设计，接下来就是main函数中游戏框架的搭建了，实现如下：

int main()

{

//创建窗口

initgraph(1080, 720);

//加载音乐并取名

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

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

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

//创建玩家向左向右和敌人向左向右的图集

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"), 3);

atlas\_enemy\_right = new Atlas(\_T("img/enemy\_right\_%d.png"), 3);

//定义变量

int score = 0;

ExMessage msg;

IMAGE background;

Player player;//创建玩家类的对象

IMAGE img\_menu;

std::vector<Enemy\*> enemy\_list;//敌人的容器

std::vector<enemy\_bullet\*>enemy\_bullet\_list;//敌人子弹的容器

//生成开始按钮和结束按钮的矩形

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

region\_btn\_start\_game.left = (window\_width - button\_width) / 2 + 240;

region\_btn\_start\_game.right = region\_btn\_start\_game.left + button\_width;

region\_btn\_start\_game.top = 330;

region\_btn\_start\_game.bottom = region\_btn\_start\_game.top + button\_height;

region\_btn\_quit\_game.left = (window\_width - button\_width) / 2 + 240;

region\_btn\_quit\_game.right = region\_btn\_quit\_game.left + button\_width;

region\_btn\_quit\_game.top = 450;

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

BeginBatchDraw();//开启批量绘图

while (running)

{

DWORD begin\_time = GetTickCount();

while (peekmessage(&msg))

{

if (is\_game\_started)

{

player.processevent(msg);

}//游戏开始就处理玩家的各类信息

else

{

btn\_start\_game.ProcessEvent(msg);

btn\_quit\_game.ProcessEvent(msg);

}//游戏未开始就处理开始按钮和结束按钮的信息

}

if (is\_game\_started)

{

player.move();//更新玩家位置

//更新子弹位置

update\_bullets(player.bullet\_list, player);

update\_bullets(player.circle\_bullet\_list, player);

update\_bullets(enemy\_bullet\_list);

//调用生成敌人的函数

generate\_enemy(enemy\_list,player);

for (Enemy\* enemy : enemy\_list)

{

//调用敌人向玩家移动的函数

enemy->move(player);

//敌人生成子弹

if (enemy->check\_time())

{

enemy\_bullet\* new\_bullet = new enemy\_bullet();

POINT enemy\_position = enemy->get\_position();

POINT player\_position = player.get\_position();

new\_bullet->set\_position(enemy\_position);

new\_bullet->init\_nor\_x\_and\_nor\_y(enemy\_position, player\_position);

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

}

}

//检测敌人和玩家碰撞，如果碰撞就清除敌人且扣除玩家生命值

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

{

if (enemy\_list[i]->check\_player\_collision(player))

{

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

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

enemy\_list.pop\_back();

delete this\_enemy;

player.hp = player.hp - 20;

//如果玩家生命值低于0，游戏结束

if (player.hp <= 0)

{

static TCHAR text[128];

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

MessageBox(GetHWnd(), text, \_T("对不起，你没能打赢复活赛！"), MB\_OK);

running = false;

break;

}

}

}

//检测玩家和敌人子弹碰撞，如果碰撞就清除子弹且扣除玩家生命值

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

{

if (enemy\_bullet\_list[i]->check\_player\_collision(player))

{

enemy\_bullet\* bullet = enemy\_bullet\_list[i];

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

enemy\_bullet\_list.pop\_back();

delete bullet;

player.hp = player.hp - 10;

//如果玩家生命值低于0，游戏结束

if (player.hp<=0)

{

static TCHAR text[128];

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

MessageBox(GetHWnd(), text, \_T("对不起，你没能打赢复活赛！"), MB\_OK);

running = false;

break;

}

}

}

//检测敌人和螺旋子弹碰撞，如果碰撞，清除敌人，玩家加分且回复能量

for (Enemy\* enemy : enemy\_list)

{

for (const Bullet& bullet : player.bullet\_list)

{

if (enemy->check\_bullet\_collision(bullet))

{

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

enemy->hurt();

score++;

player.sp = min(player.sp + 10, 100);

//更新玩家等级，如果等级上升，随机增加一种类型的子弹

if (score % 10 == 0)

{

player.grade++;

player.add\_bullet(player.bullet\_list, player.circle\_bullet\_list);

}

}

}

//检测敌人和圆周子弹的碰撞，如果碰撞，清除敌人，玩家加分且回复能量

for (const Bullet& bullet : player.circle\_bullet\_list)

{

if (enemy->check\_bullet\_collision(bullet))

{

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

enemy->hurt();

score++;

player.sp = min(player.sp + 10, 100);

//更新玩家等级，如果等级上升，随机增加一种类型的子弹

if (score % 10 == 0)

{

player.grade++;

player.add\_bullet(player.bullet\_list, player.circle\_bullet\_list);

}

}

}

}

//清除碰到墙壁的子弹

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

{

enemy\_bullet\* bullet = enemy\_bullet\_list[i];

if (bullet->check\_collision())

{

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

enemy\_bullet\_list.pop\_back();

delete bullet;

}

}

//清除生命值为0的敌人

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;

}

}

//检测玩家是否释放技能

if (player.release\_skill())

{

draw\_skill(player,enemy\_bullet\_list);

player.sp = 0;

}

}

//如果间隔时间小于帧数，则休眠剩下的时间

DWORD end\_time = GetTickCount();

DWORD delta\_time = end\_time-begin\_time;

if (delta\_time < 1000 / 144)

{

Sleep(1000 / 144 - delta\_time);

}

cleardevice();//清除屏幕内容

if (is\_game\_started)

{

putimage\_alpha(0, 0, &background);//加载背景图片

player.draw(1000 / 144);//绘制玩家

for (Enemy\* enemy : enemy\_list)

{

enemy->draw(1000 / 144);

}//绘制敌人

for (const Bullet& bullet : player.bullet\_list)

{

bullet.draw();

}//绘制玩家的螺旋子弹

for (const circle\_bullet& circle\_bullet : player.circle\_bullet\_list)

{

circle\_bullet.draw();

}//绘制玩家的圆周子弹

for (enemy\_bullet\*& enemy\_bullet : enemy\_bullet\_list)

{

enemy\_bullet->draw();

}//绘制敌人的子弹

//绘制玩家分数、等级、血量和能量

draw\_player\_score(score);

draw\_player\_grade(player.grade);

draw\_player\_hp(player);

draw\_player\_sp(player);

}

else

{

putimage(0, 0,&img\_menu);//绘制目录图片

btn\_start\_game.Draw();//绘制开始按钮

btn\_quit\_game.Draw();//绘制结束按钮

}

FlushBatchDraw();//执行未完成的绘制任务

}

delete atlas\_player\_left;

delete atlas\_player\_right;

//删除atlas\_player\_left和atlas\_player\_right

EndBatchDraw();//结束批量绘图

return 0;

}

全局变量有：

const int window\_width = 1280;//窗口宽度

const int window\_height = 720;//窗口高度

const int button\_width = 192;//按钮宽度

const int button\_height = 75;//按钮高度

const double pi = 3.14159;//π

bool is\_game\_started = false;//游戏是否开始

bool running = true;//游戏是否进行

Atlas\* atlas\_player\_left;//玩家向左的图集

Atlas\* atlas\_player\_right;//玩家向右的图集

Atlas\* atlas\_enemy\_left;//敌人向左的图集

Atlas\* atlas\_enemy\_right;//敌人向右的图集

这样我们就做出来了一款躲避弹幕的割草小游戏。

* 1. **算法或公式**

1. 两点间距离计算公式：



1. 勾股定理：

直角三角形ABC，设c为斜边，a、b为两直角边则有：



* 1. **单元测试**

针对不同的两个点都可以正确地计算它们之间的距离。

1. **单元测试**

经过测试，每一个类都可以很好地执行自己的功能。

1. **收获**

深入学习了有关类的知识，还初步学习了游戏的一般框架，也初步学习了有关容器的一些知识。