四川大學
SICHUAN UNIVERSITY

研究与开发实践结题报告

计算机学院

2024 年 12 月 17 日

项目名：**Ark-knights 方舟骑士**

学号：**2022141460176**

姓名：杨一舟

指导老师：张轶

目录

一、 游戏概述

1.1 游戏简介

本游戏的玩法参考自凉屋游戏出品的 2D 冒险游戏《元气骑士》，素材与故事背景设定源自鹰角网络出品的《明日方舟》，核心内容与剧情流程则为原创。游戏讲述了画之大者夕进入自己的画卷，闭关苦修，沉心静气，清除心中杂念化为的敌人，以达至臻画境的故事。游戏分为三关，夕需要击败各种机制的敌人，在通过每个关卡后会学会新的技能，最终在 BOSS 关击败“自己”，突破自我的桎梏，达到新的画境，获得最终的胜利。

本作的风格以水墨画卷为主，素材是从明日方舟百科站 PRTS 中下载的角色动画，再经自行处理后获得合适的贴图，配以国风音乐，与故事主题相契合。玩家需要通过键盘灵活操控角色躲避攻击，消灭敌人，合理规划法力值释放技能，才能更好地通关。

1.2 游戏开发环境

Visual studio 2022

EasyX 图形库

1.3 游戏创新点

多种机制与技能：不同关卡的敌人具有不同的机制，如晦明、化生等，同时角色也会使用不同的方式对敌人造成伤害。角色在不同的关卡也会学习不同的技能，丰富了游戏体验。

多关卡保留悬念：玩家在通过当前关卡后才能进入下一关，保留了一定的悬念。

音乐与受击效果：背景音乐选用的是角色 ep 贯穿始终，符合故事背景。敌人在被攻击时会有受击动画，优化了攻击手感。

关卡间故事串联：不同关卡之间有一定背景故事的串联，带来更沉浸的体验。

1.4 游戏代码结构

本游戏代码共计 2420 行，动画素材共计 274 张，整体逻辑在 demo.cpp 的主函数中实现，但各个关卡内部的具体逻辑由各自关卡具体的对象完成。

整体的类、衍生类及实例化对象如下：

敌人类（基类）——第一关敌人、第二关敌人、第三关 BOSS 敌人（衍生类）——敌人阿咬、敌人小躁、敌人匕现、敌人沉沙、敌人磨磐、BOSS 自在（实例化对象）

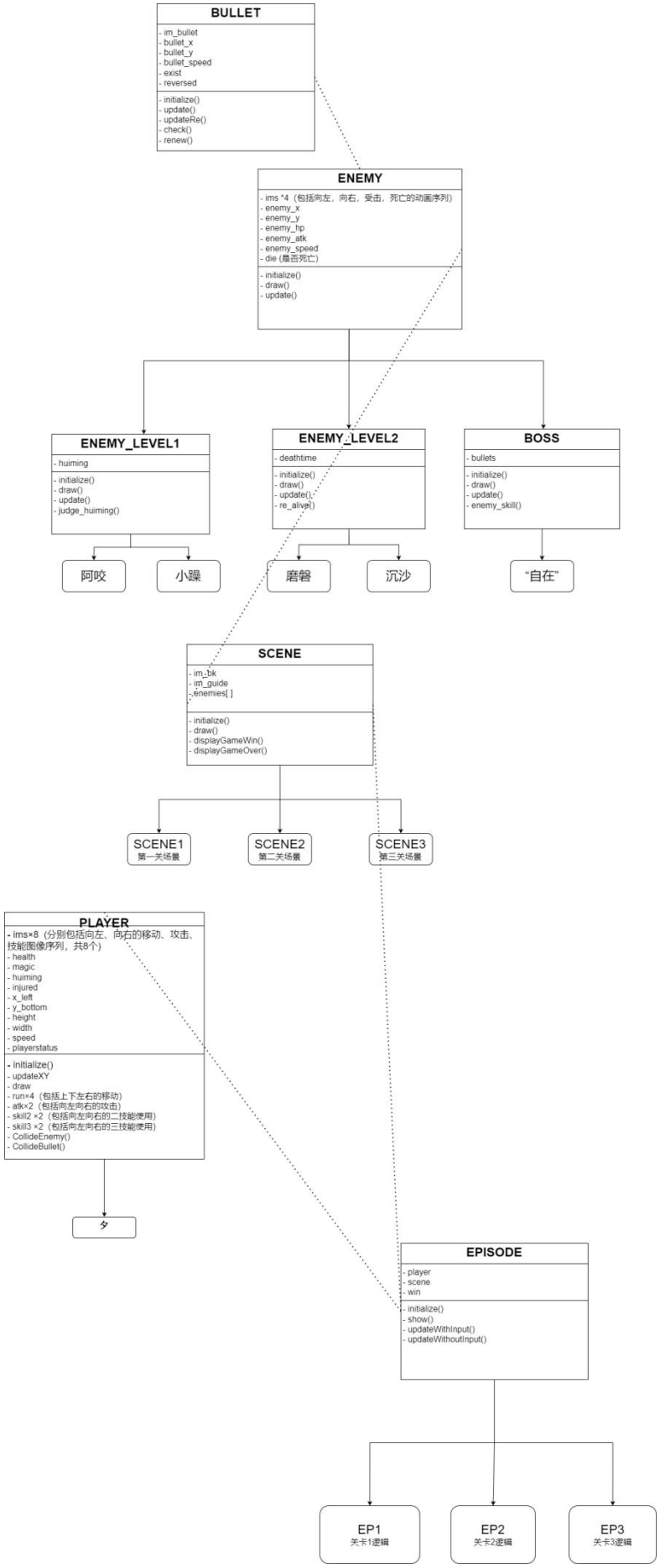
人物类——角色夕

场景类（基类）——关卡 1 场景、关卡 2 场景、关卡 3 场景（衍生类）

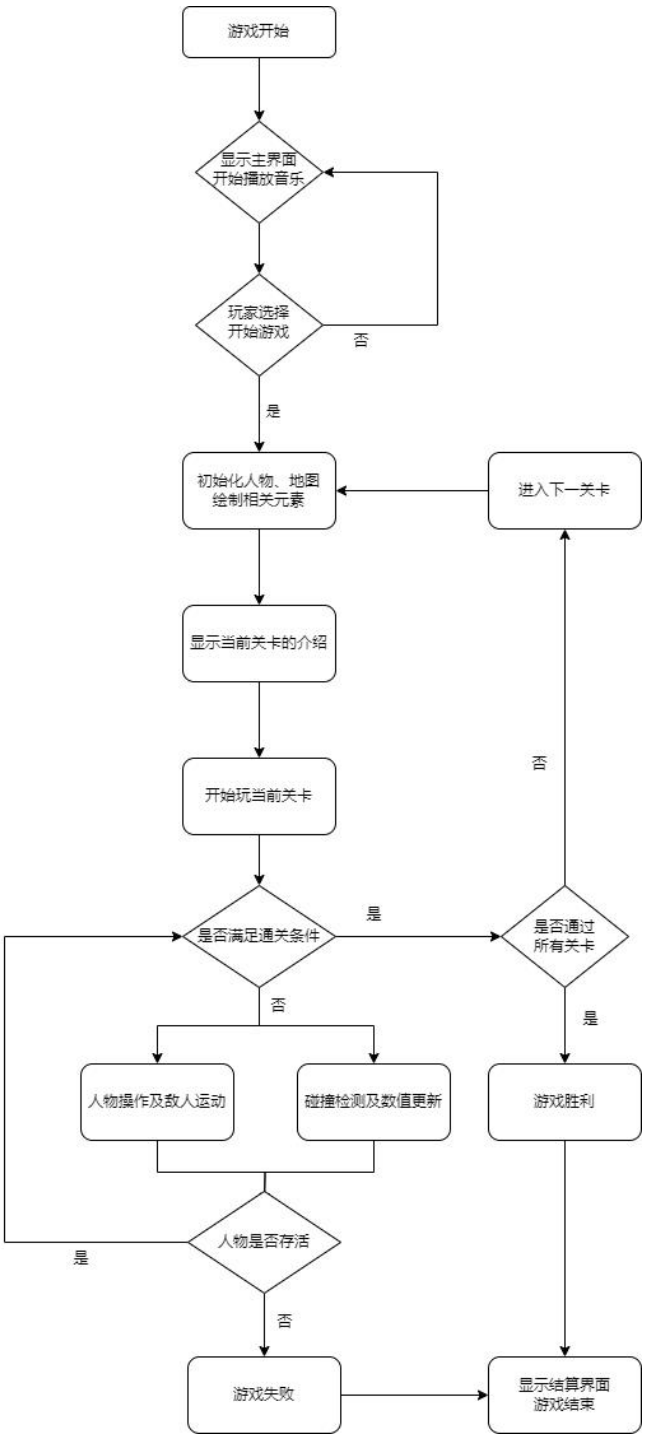
关卡类（基类）——关卡 1 逻辑、关卡 2 逻辑、关卡 3 逻辑

子弹类——敌人的子弹

类图架构具体如下所示：


BULLET
- im_bullet
- bullet_x
- bullet_y
- bullet_speed
- exist
- reversed
- initialize()
- update()
- updateRe()
- check()
- renew()
ENEMY
- ims *4（包括向左，向右，受击，死亡的动画序列）
- enemy_x
- enemy_y
- enemy_hp
- enemy_atk
- enemy_speed
- die (是否死亡)
- initialize()
- draw()
- update()
ENEMY_LEVEL1
- huiming
- initialize()
- draw()
- update()
- judge_huiming()
ENEMY_LEVEL2
- deathtime
- initialize()
- draw()
- update()
- re_alive()
BOSS
- bullets
- initialize()
- draw()
- update()
- enemy_skill()
阿咬
小躁
磨磐
沉沙
“自在”
SCENE
- im_bk
- im_guide
- enemies[ ]
- initialize()
- draw()
- displayGameWin()
- displayGameOver()
SCENE1
第一关场景
SCENE2
第二关场景
SCENE3
第三关场景
PLAYER
- ims×8（分别包括向左、向右的移动、攻击、技能图像序列，共8个）
- health
- magic
- huiming
- injured
- x_left
- y_bottom
- height
- width
- speed
- playerstatus
- initialize()
- updateXY
- draw
- run×4（包括上下左右的移动）
- atk×2（包括向左向右的攻击）
- skill2 ×2（包括向左向右的二技能使用）
- skill3 ×2（包括向左向右的三技能使用）
- CollideEnemy()
- CollideBullet()
夕
EPISODE
- player
- scene
- win
- initialize()
- show()
- updateWithInput()
- updateWithoutInput()
EP1
关卡1逻辑
EP2
关卡2逻辑
EP3
关卡3逻辑

1.5 游戏流程


游戏开始
显示主界面
开始播放音乐
玩家选择
开始游戏
否
是
初始化人物、地图
绘制相关元素
进入下一关卡
显示当前关卡的介绍
开始玩当前关卡
否
是否满足通关条件
是
是否通过
所有关卡
否
是
人物操作及敌人运动
碰撞检测及数值更新
游戏胜利
人物是否存活
是
否
游戏失败
显示结算界面
游戏结束

二、 游戏内容

2.1 游戏基本规则

玩家通过键盘控制夕进行上下左右移动、进行攻击和释放技能，在躲避敌人攻击的同时寻找机会攻击敌人。在每个关卡中消灭所有敌人即可打开进入下一关的门，走入门中即可进入下一关，通过 BOSS 关即为通关。

被敌人撞到会失去生命，生命为 0 时则失败。攻击敌人需要消耗法力，成功击杀敌人时会回复生命与法力，法力也会随时间慢慢恢复，每当进入新关卡时会重置生命与法力。这需要玩家合理规划法力，不能盲目进行攻击。

2.2 游戏操作方式

玩家依照游戏指引中的操作进入关卡，通过键盘操作：

‘WSAD’或者‘上下左右’方向键操作人物的上下左右移动

‘J’键使用普通攻击

‘K’键使用技能

2.3 游戏开始界面

游戏开始运行后显示游戏开始界面，按照提示点击空格开始游戏。

2.4 游戏主界面

玩家开始游戏后便显示游戏主界面，左上角显示了人物生命与法力。游戏共分为三个关卡，在中间上方显示关卡名字。

2.4.1 第一关：灰齐山

玩家进入第一关后会看到第一关的关卡介绍：

第一关特别机制：晦明

在第一关中，人物与每个敌人都具有“晦”或“明”的属性，人物对相同属性敌人攻击时会造成一半的伤害，而对不同属性的敌人攻击时造成两倍的伤害。每当角色击杀一个敌人时，会改变自身的晦明属性。

第一关敌人：

阿咬：平平无奇的墨魑，会朝着人物方向攻击

小躁：移动迅速的墨魑，会快速地朝着人物方向攻击

第一关比较简单，同时人物的技能也不复杂，只需要进行普通攻击即可击败所有敌人。但是玩家需要注意合理利用晦明的机制来提高通关效率

2.4.2 第二关：纂江峰

玩家进入第二关后会看到第二关的关卡介绍：

第二关特别机制：化生

在第二关中，敌人被击杀后会化为一个物件，在死亡后再经过一段时间会复活，只有当所有敌人都被击杀后才能通关。

第二关新技能：工笔入化

消耗一定法力值，对全屏敌人造成一次较高的伤害。

第二关敌人：

磨磐：移动迅速的化物，并且会在一段时间后隐身，可能会对人物发起偷袭，死亡之后化为木刻印章，在少许时间后复生。

沉沙：移动缓慢的化物，但是攻击与生命较高，死亡之后化为铜矛头，在略长时间后复生。

在第二关中，增加了略复杂的机制，同时要将全部敌人都在复活前击杀才能通关，但人物也获得了可以进行全屏打击的技能，对于玩家有一定的挑战，需要合理规划法力值与操作才能顺利通过。

2.4.3 第三关：玉门关

玩家进入第三关后会看到第三关的关卡介绍：

第三关特别机制：弹反与能量

在第三关中，人物无法直接对 BOSS 造成伤害，而是通过攻击反弹 BOSS 的攻击对其造成伤害。成功反弹一次后会积攒一点能量，消耗五点能量可以使用技能。

第三关新技能：写意胜形

消耗五点反弹能量，召唤一只小自在对直线路径上的敌人造成伤害，此技能可蓄力，蓄力时间越长伤害越高，但是人物在蓄力过程中会被攻击时会受到更多伤害。

第三关 BOSS 敌人：自在

威严冷傲的大型化物，无法直接对其造成伤害。每隔一段时间朝着人物会发射五个速度不同的雷球，被人物反弹后的雷球会对 BOSS 造成伤害。

在第三关中，玩家需要面对体型较大的 BOSS 敌人，BOSS 的雷球有的速度极快，有的速度较慢，玩家需要控制好攻击弹反的时机，从而达到不浪费法力，又成功反弹攻击。蓄力技能有利有弊，玩家进行蓄力能够造成较多伤害，但是蓄力过程中被攻击到则会受到更多伤害。

2.5 结算界面

2.5.1 游戏失败

在人物的生命归零后，将会出现失败界面，意味着夕在画卷中的求索之旅以迷失结尾，玩家可以按空格结束游戏，然后重新开始。

2.5.2 游戏胜利

在人物通过所有关卡后，将会出现胜利界面，意味着夕在画卷中的闭关求索突破了自我的桎梏，达到了逍遥自在的境界，玩家可以按空格结束游戏，然后重新开始。

三、 游戏元素

3.1 素材获取

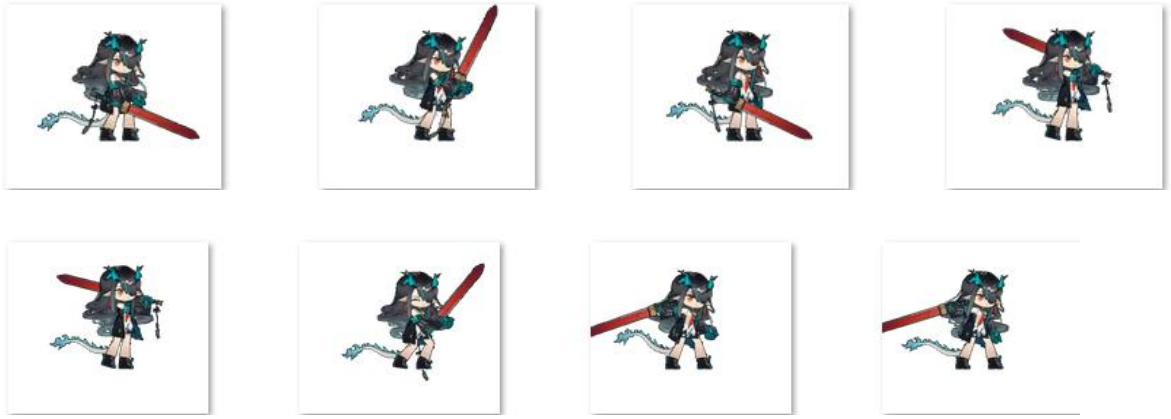
本作的素材是从 PRTS 上获取的《明日方舟》原作中使用的角色模型动画，以视频格式下载，再通过自己编写的 Python 程序进行截帧、去除背景、缩放与翻转，从而获取大小与方向合适的贴图。

```
# 遍历输入文件夹中的每个视频文件
for video_file in os.listdir(input_folder):
    if video_file.endswith(('.mp4', '.avi', '.mov', '.mkv','webm')):  # 支持的视频格式
        video_path = os.path.join(input_folder, video_file)

        # 创建以视频文件名为名的输出文件夹
        video_name = os.path.splitext(video_file)[0]
        output_folder = os.path.join(output_base_folder, video_name)
        os.makedirs(output_folder, exist_ok=True)

        # 打开视频文件
        video = cv2.VideoCapture(video_path)

        # 检查是否成功打开
        if not video.isOpened():
            print(f"Error: Could not open video {video_file}.")
            continue

        # 设置保存帧的间隔
        save_frame_interval = 12  # 每12帧保存一次
        frame_count = 0
        current_frame = 0

        # 设置目标尺寸
        target_size = (500, 500)

        while True:
            ret, frame = video.read()
            if not ret:
                break  # 结束循环，当无法读取到下一帧时

            current_frame += 1
            if current_frame % save_frame_interval == 0:
                # 将帧从 BGR 转换为 RGBA
                frame_rgba = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)

                # 将黑色部分的 alpha 通道设置为 0
                black_threshold = 15  # 黑色阈值，可以根据实际情况调整
                frame_rgba[(frame_rgba[:, :, 0] < black_threshold) &
                           (frame_rgba[:, :, 1] < black_threshold) &
                           (frame_rgba[:, :, 2] < black_threshold), 3] = 0
```

本作的音乐取自《明日方舟》的角色 ep，从官方音乐网站塞壬唱片下载，与人物背景故事契合。

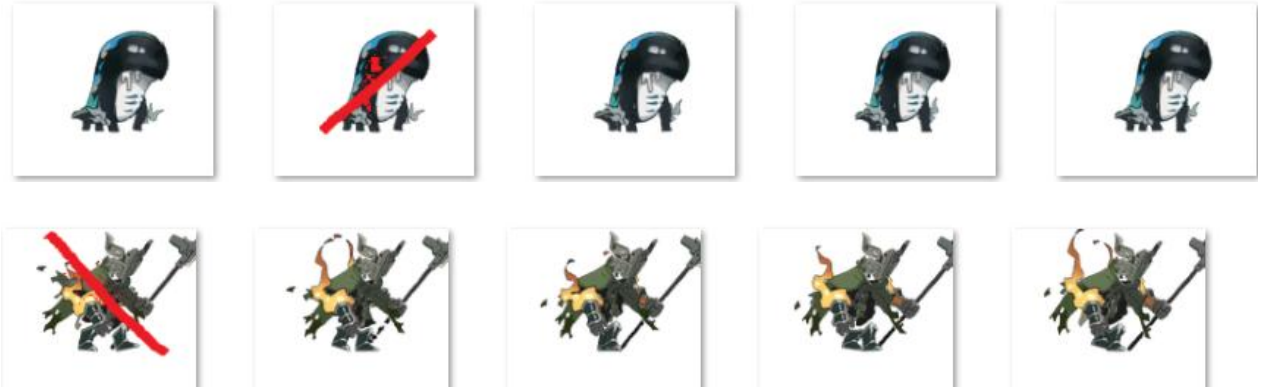
3.2 人物

人物，夕，即玩家操作的对象。玩家通过 WSAD 控制人物上下左右移动，J 使用普通攻击，K 使用技能，在不同的移动状态及攻击状态下会有不同的动作序列。人物拥有“生命”与“法力”属性，只有法力值足够进行攻击或者技能时才能成功进行攻击与技能。



3.3 敌人

不同的敌人有着不同的机制，碰撞到人物时会对其造成伤害，会始终朝着人物的方向冲去，在不同的方向上会有不同的动作序列，受到人物的攻击时会有受击动画。



3.4 传送门

在击败当前关卡中所有敌人后会出现传送门，走入传送门即可进入下一关，通过所有关卡后可获得游戏胜利。

四、 核心逻辑

4.1 动画加载

整体通过 player status 这一变量来管理人物的各种状态，方便后续操作，人物状态共有 standleft, standright, runleft, runright, runup, rundown, atkleft, atkright 这几种，分别表示上下左右的移动以及向左向右的攻击。

4.1.1 站立与行走

在人物类中分别加载站立与行走所需要的贴图，向左走与向右走有着不同的动画，然后设置状态为移动，并且移动人物的坐标，并通过计数器 animID 来控制所播放贴图的顺序。而贴图切换的速度则是在关卡中由定时器 timer 设置绘制间隔来实现的。敌人的动画加载与人物同理

以人物向右奔跑为例展示代码：

```
void runRight() // 游戏角色向右奔跑
{
    x_left += v; // 横坐标增加，向右移动


    if (playerStatus != runright) // 如果之前角色状态不是向右奔跑
    {
        playerStatus = runright; // 切换为向右奔跑状态
        animId = 0; // 动画播放id初始化为0
    }
    else // 表示之前就是向右奔跑状态了
    {
        animId++; // 动画图片开始切换
        if (animId >= ims_runright.size()) // 循环切换
            animId = 0;
    }
    im_show = ims_runright[animId];  // 设置要显示的对应图片

}
```

4.1.2 人物攻击与技能

在人物类中分别加载攻击与技能所需要的贴图，向左与向右有着不同的动画，然后设置状态为攻击，并通过计数器 animID 来控制所播放贴图的顺序。而贴图切换的速度同样是在关卡中由定时器 timer 设置绘制间隔来实现的。释放技能的动画加载与其同理。

以向左攻击为例展示代码：

```
void atkLeft() // 游戏角色向左攻击
{

        if (playerStatus != atkleft) // 如果之前角色状态不是向左攻击
                {
                    playerStatus = atkleft; // 切换为向左攻击状态
                    animId = 0; // 动画播放id初始化为0

                }
                else // 之前就是向左攻击状态了
                {
                    animspeed++;
                    if(animspeed%2==0)animId++; // 动画图片开始切换

                    if (animId >= ims_atkleft.size()) {
                        animId = 0;

                    } // 循环切换

                }

                im_show = ims_atkleft[animId];   // 设置要显示的对应图片

                if (animspeed % 9 == 0) magic-=5;
                if (magic <= 0)magic = 0;
}
```

4.1.3 文字的淡入与淡出动画

把文字的颜色逐渐变为背景色达到文字淡出的效果，相反就是文字淡入。这里主要是把 RGB 颜色的三个值按照同一个变化频率逐渐向目标色靠拢。

```
void fade_in() {
    TCHAR text[50] = _T("一更天，对镜烧烛展玉宣");
    LOGFONT font;
    //gettextstyle(&font);
    //settextstyle(20, 0, _T("华文隶书"));
    COLORREF bk_color = getbkcolor();
    COLORREF text_color = WHITE;
    int bk_color_arr[3] = { GetRValue(bk_color), GetGValue(bk_color), GetBValue(bk_color) };
    int text_color_arr[3] = { GetRValue(text_color), GetGValue(text_color), GetBValue(text_color) };

    int times = 80,
        r_incr = (text_color_arr[0] - bk_color_arr[0]) / times,
        g_incr = (text_color_arr[1] - bk_color_arr[1]) / times,
        b_incr = (text_color_arr[2] - bk_color_arr[2]) / times;

    BeginBatchDraw();
    for (int i = 0; i < times; i++) {
        //font.lfHeight = 80;
        //settextstyle(&font);
        settextstyle(80, 0, _T("华文行楷"));
        int text_width = textwidth(text),
            text_height = textheight(text);
        cleardevice();
        //putimage(0, 0, &win_image);
        settextcolor(RGB(bk_color_arr[0] + r_incr * i, bk_color_arr[1] + g_incr * i, bk_color_arr[2] + b_incr * i));
        outtextxy((WIDTH - text_width) / 2, (HEIGHT - text_height) / 2, text);
        FlushBatchDraw();
        Sleep(15);
    }
    EndBatchDraw();
}
```

4.2 敌人追逐人物移动

在敌人类中分别加载移动所需要的贴图，在敌人的移动（即位置更新函数 update）中加入一个目标位置，该位置由一个类型为 player 的指针形参传入，将目标位置设置为玩家的坐标，然后通过勾股定理计算水平方向速度与竖直方向速度，再体现在水平坐标与数值坐标的更新上。

同时，如果玩家的位置在敌人的左侧，则将状态更新为向左移动，播放向左移动的贴图；如果玩家的位置在敌人的右侧，则将状态更新为向右移动，播放向右移动的贴图。

```
void update(float targetX, float targetY) {
    float dx = targetX - x;
    float dy = targetY - y;
    float distance = sqrt(dx * dx + dy * dy);

    // 归一化方向并设置速度
    float vx = (dx / distance) * 2.0f;
    float vy = (dy / distance) * 2.0f;
    if (dx < 0)enemystatus = enemyleft;
    else enemystatus = enemyright;
    // 更新位置
    x += vx;
    y += vy;
}
```

4.3 物体碰撞检测

4.3.1 人物与敌人的伤害碰撞检测：

在人物与敌人的碰撞检测中，将二者都视为了矩形，有相交则认为有碰撞，以人物伤害敌人为例展示代码，敌人伤害人物同理：

```
int isCollideEnemy(Enemy& enemy)
{
    x_right = x_left + width;
    y_top = y_bottom - height;
    float x_right_enemy = enemy.x + enemy.enemy_width;
    float y_top_enemy = enemy.y - enemy.enemy_height; // 同样注意y轴方向
    float x_left_enemy = enemy.x;
    float y_bottom_enemy = enemy.y;

    if (x_left< x_right_enemy && x_right > x_left_enemy &&
        y_bottom > y_top_enemy && y_top < y_bottom_enemy)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

4.3.2 人物与子弹的碰撞检测：

在人物与子弹的碰撞检测中，将二者都视为了圆形，有相交则认为有碰撞：

```
void CollideBullet(Bullet& bullet) {

    // 获取子弹和玩家的半径)
    float bulletRadius = bullet.im_bullet.getwidth() / 2.0f - 5; // 如果图像宽度等于直径
    float playerRadius = this->width / 2.0f - 50; // 假设 width 是直径

    // 计算圆心之间的距离
    float dx = bullet.x + 0 - (this->x_left + playerRadius + 20); // 玩家圆心的 x 坐标是 x_left + 半径
    float dy = bullet.y + 10 - (this->y_bottom + playerRadius - 120); // 玩家圆心的 y 坐标是 y_bottom + 半径
    float distanceSquared = dx * dx + dy * dy;
    //setlinecolor(BLACK);
    //circle(this->x_left + playerRadius + 20, this->y_bottom + playerRadius - 120, playerRadius);
    //circle(bullet.x + 50, bullet.y + 10, bulletRadius);
    // 检查两个圆是否相交
    float radiiSum = bulletRadius + playerRadius;
    if ((distanceSquared <= radiiSum * radiiSum) && playerStatus != atkleft && playerStatus != atkright) {
        // 如果发生碰撞，减少玩家的生命值并标记子弹为不存在
        this->health -= 10;
        bullet.exisit = false;
    }
    if ((distanceSquared <= radiiSum * radiiSum) && (playerStatus == atkleft || playerStatus == atkright) && magic > 0) {
        // 成功反弹则标记为反弹
        this->energy++;
        if (energy >= 8)energy = 8;
        bullet.reverse = true;

    }
}
```

（展示代码中含有部分调试时的显示）

4.4 键盘控制

在游戏中将使用 kbhit 函数以及 GetAsyncKeyState()函数来获取键盘状态，查看当前哪个键被按下了，从而实现对人物的各种操作。

```
if (_kbhit()) // 当按键时，切换角色显示图片，更改位置
{
    if (player.x_left >= 500)player.x_left = 500;
    if (player.x_left <= 0)player.x_left = 0;
    if (player.y_bottom >= 600)player.y_bottom = 600;
//  if (player.y_bottom <= 30)player.y_bottom = 30;
    if (GetAsyncKeyState(VK_RIGHT) || GetAsyncKeyState('D'))  // 按下D键或右方向键
    {
        player.runRight();
        //player.standStill();
    }
    else if (GetAsyncKeyState(VK_LEFT) || GetAsyncKeyState('A'))// 按下A键或左方向键
    {
        player.runLeft();
        //player.standStill();
    }
    if (GetAsyncKeyState(VK_UP) || GetAsyncKeyState('W'))     // 按下W键或上方向键
    {
        player.runUp();
        //player.standStill();
    }
    if (GetAsyncKeyState(VK_DOWN) || GetAsyncKeyState('S'))     // 按下W键或上方向键
    {
        player.runDown();
        //player.standStill();
    }
    if (GetAsyncKeyState('J') && (player.playerStatus == runleft || player.playerStatus == atkleft || player.playerStatus == rundown) && player.magic > 0)// 向左攻击
    {
        player.atkLeft();
        player.standStill();
        //player.magic--;
    }
    if (GetAsyncKeyState('J') && (player.playerStatus == runright || player.playerStatus == atkright || player.playerStatus == runup) && player.magic > 0)   // 向左攻击
    {
        player.atkRight();
        player.standStill();
        //player.magic--;
    }
    if (GetAsyncKeyState('K')&&(player.playerStatus == runleft || player.playerStatus == atkleft || player.playerStatus == rundown))     // 按下K键使用技能
    {
        player.skill3Left();
        player.isskill3 = true;
        zzx = player.x_left;
    }
    if (GetAsyncKeyState('K') && (player.playerStatus == runright || player.playerStatus == atkright || player.playerStatus == runup)&&player.energy>=5)     // 按下K键使用技能
```

4.5 音乐播放

在游戏中使用#pragma comment(lib,"Winmm.lib")库的mciSendString函数进行音乐加载，在每个关卡开始前会切换不同的音乐。

```
mciSendString(_T("close bkmusic"), NULL, 0, NULL);//停止播放
mciSendString(_T("open ZFC.mp3 alias bkmusic"), NULL, 0, NULL);//播放音乐
mciSendString(_T("play bkmusic repeat"), NULL, 0, NULL);//循环播放
```

五、 游戏难点及解决方案

5.1 生命与法力值的消耗

由于游戏绘制与检测是以帧为单位进行，故若单纯以检测到碰撞则减少生命、检测到攻击就减少法力为逻辑，则会导致血量与法力快速地持续减少。

对于生命，所提出的解决方案是增加一个受伤标志位 hurt，在碰撞时将其置为 true 并减少一次生命，当不产生碰撞时将其重制为 false。只有当 hurt 为 false 且碰撞时才会减少生命，这样就实现了每次碰撞只会减少一次生命而不会持续减少生命。这样的方案同时适用于敌人受到我方攻击与我方受到敌人攻击。

```
if (player.injured == false) {

    player.health -= 1;
    player.injured = true;
}
```

对于法力，虽然从结果上来说，以帧为单位减少法力值与攻击完之后一次性减少法力值是等效的，但是明显后者的观感要好得多。所提出的解决方案是增加一个计时器，设置好播放完一遍攻击动作的时间后减少一定的法力值，这样能够使得在攻击动作的过程中不会减少法力，而是在攻击动作完成之后一次性减少总量的法力值。

```
void atkLeft() // 游戏角色向左攻击
{

        if (playerStatus != atkleft) // 如果之前角色状态不是向左攻击
                {
                    playerStatus = atkleft; // 切换为向左攻击状态
                    animId = 0; // 动画播放id初始化为0

                }
                else // 之前就是向左攻击状态了
                {
                    animspeed++;
                    if(animspeed%2==0)animId++; // 动画图片开始切换

                    if (animId >= ims_atkleft.size()) {
                        animId = 0;

                    } // 循环切换

                }

                im_show = ims_atkleft[animId];   // 设置要显示的对应图片

                if (animspeed % 9 == 0) magic-=5;
                if (magic <= 0)magic = 0;
}
```

5.2 敌人受击动画的播放

为了正确播放敌人的受击动画，所提出的解决方案是为敌人设置一个受伤标志位 injured，只有当有碰撞且我方的状态是正在攻击时，才会将 injured 置为 true，否则重置为 false。只有当 injured 为 true 时才会将贴图更换为受击时的贴图。

```
if (!player.isCollideEnemy(scene2.en5))scene2.en5.hurt = false;
player.injured = false;
```

5.3 蓄力技能的释放

为了使得技能的弹道从人物身上开始，在按下 K 键时会记录人物当前的横坐标，并更新到子弹的坐标上。

为了在正确的时候释放技能，技能的释放被设置为了只有能量足够时按下对应按键才能有效释放技能。

为了实现三技能的蓄力，所提出的解决方案是为人物设置一个释放三技能的标志位 isskill3，当按下 K 键时此标志位记录为 true，且会一直将子弹的坐标更新为人物的坐标，记录蓄力时间的变量 xuli 也会增加。而当松开 K 键时子弹坐标不再更新，而是在技能函数中逐渐向右移动，最终伤害会由基础伤害值加上 xuli 值乘以系数决定。

```
if (player.isskill3 == true) {

    zzx+=20;
    player.skill3(zzx);
    if (zzx > 700) {
        zzx = player.x_left;
        ZZ.boss_hp -= 10 + xuli * 10;
        player.isskill3 = false;
        xuli = 0;
        player.energy -= 5;
    }
}
```

```
if (GetAsyncKeyState('K') && (player.playerStatus == runright || player.playerStatus == atkright || player.playerStatus == runup)&&player.energy>=5)
{
    player.skill3Right();
    player.isskill3 = true;
    xuli++;
    zzx = player.x_left;

}
```

5.4 全屏技能的释放

为了二技能对全屏敌人造成打击，所提出的解决方案是设置一个释放二技能的标志位 isskill2，当按下 K 时将其置为 true。而当其为 true 时遍历当前场景中的所有敌人，对其造成一次伤害。（此时需要用到上述中提到的敌人受到我方伤害的逻辑）同时为了使得使用一次技能不会持续造成伤害，还需要在造成伤害完成之后将标志位重置为 false。

5.5 子弹弹反的处理

为了能够使子弹能够正常伤害人物，也能在人物使用攻击时被正常反弹，所提出的解决方案是在碰撞时再对 player status 进行判定，如果发生碰撞时是正在进行攻击的状态则设置为反弹，否则受到伤害。

```
void CollideBullet(Bullet& bullet) {

    // 获取子弹和玩家的半径）
    float bulletRadius = bullet.im_bullet.getwidth() / 2.0f - 5; // 如果图像宽度等于直径
    float playerRadius = this->width / 2.0f - 50; // 假设 width 是直径

    // 计算圆心之间的距离
    float dx = bullet.x + 0 - (this->x_left + playerRadius + 20); // 玩家圆心的 x 坐标是 x_left + 半径
    float dy = bullet.y + 10 - (this->y_bottom + playerRadius - 120); // 玩家圆心的 y 坐标是 y_bottom + 半径
    float distanceSquared = dx * dx + dy * dy;
    //setlinecolor(BLACK);
    //circle(this->x_left + playerRadius + 20, this->y_bottom + playerRadius - 120, playerRadius);
    //circle(bullet.x + 50, bullet.y + 10, bulletRadius);
    // 检查两个圆是否相交
    float radiiSum = bulletRadius + playerRadius;
    if ((distanceSquared <= radiiSum * radiiSum) && playerStatus != atkleft && playerStatus != atkright) {
        // 如果发生碰撞，减少玩家的生命值并标记子弹为不存在
        this->health -= 10;
        bullet.exisit = false;
    }
    if ((distanceSquared <= radiiSum * radiiSum) && (playerStatus == atkleft || playerStatus == atkright) && magic > 0) {
        // 成功反弹则标记为反弹
        this->energy++;
        if (energy >= 8)energy = 8;
        bullet.reverse = true;

    }
}
```

5.6 关卡机制的实现

晦明：在我方对敌人造成伤害的碰撞检测中增加对晦明属性的判断，如果相同则造成一半伤害，不同则造成两倍伤害。

```
if ((player.playerStatus == atkleft || player.playerStatus == atkright) && player.isCollideEnemy(scene.en1)) {
    if (scene.en1.hurt == false && player.magic > 0) {
        if (player.huiming == scene.en1.huiming)scene.en1.enemy_hp -= 5;//角色对敌人造成伤害，同属性减半，不同属性加倍
        else scene.en1.enemy_hp -= 10;
    }
    scene.en1.hurt = true;
}
```

化生：为敌人增加一个死亡计时器 death time，在敌人死亡后这个变量开始计时，到达一定值时重置敌人的生命，同时重置 death time 为 0。

```
void drawdie() {
    putimagePng(x, y, &im_die);
    deathtime++;
    if (deathtime == 100) {
        enemy_hp = 80;
        deathtime = 0;
    }
}
```

六、 心得体会

终于是完成了啊。

无论是《元气骑士》的简洁明了，还是《明日方舟》的精美角色、精彩故事，又或是《Ori and the Will of the Wisps》、《Gorogoa》、《Neva》这样小而精致的作品，乃至于《黑神话悟空》《霍格沃兹之遗》这样的 3A 大作，游戏作为“第九艺术”，从小到大一直带给我们许多快乐与感动。

我体验过不少优秀作品，也尝试过写一些简单的小游戏，但今年张轶老师的《研究与开发实践》这门课程，则是我第一次完整地、独立地、真正意义上地进行游戏开发。虽然没有用到 unity、UE 等先进的游戏引擎，但 C++与 EasyX 的一步一脚印，亦是值得铭记的。虽然经验不足，时间与技术有限，最后的成品仍然有许多待完善可拓展的部分，但是能够让自己与朋友玩到亲手一行行代码敲出来的游戏，仍然十分有成就感。

“手作之所以珍贵，是因为作者将生命中的一段时间物化为了实体。”

程序开发又何尝不是如此呢？此次开发也花费了不少的时间，夕是我十分喜欢的一位角色，本作以夕为主角，虽然开发过程十分艰苦，也遇到了不少困难，但是热爱可抵岁月漫长，对角色的感情也支持着我一步步地将它做的更好，最终也算得上是献给她的一份心意。

在进行这样一个规模较大的项目开发时，我深刻体会到了先架构后编码的重要性。即使我有一定的预先架构规划，但是在编码过程中仍然遇到了不可避免的耦合，只能在实现功能的同时尽量保持代码的规范。

对于本作仍然有许多可拓展的内容由于种种限制未能完成。比如，在《明日方舟》原作中，夕有数位兄弟姐妹，也都是我很喜欢的角色，可以将他们也加入可操作的角色中，为每个人设计不同的攻击方式与技能；每位角色其实都有各自的 side story，遭遇着他们各自故事中的最终敌人，我希望在未来能够有空将我喜爱的这个世界编织的更加丰富。

本课程是我对游戏开发的一次初步但正式的尝试，希望在未来我能够尝试 unity、UE 等更强大的游戏开发工具，甚至于说，未来的工作就入职一家游戏公司也说不定呢。