《C 语言案例训练》课程设计报告

题目：表情大作战

完成日期：

2023 年 6 月 1 日

一、代码情况

代码总行数：733 行，其中，自写代码 733 行。

二、游戏功能描述及其完成情况

1.总体介绍

本项目基于 VS2022 + EasyX 开发，游戏机制模拟知名手游《球球大作战》，设计了 AI 人机系统，可以进行功能完整的本地游戏。
项目运用了面向对象和设计模式的思想，高内聚，低耦合，提高了代码可读性。
按下键盘任意键游戏随即开始，玩家通过键盘的方向键（"↑"，"↓"，"←"，"→"）操控自己的球球移动。
游戏分为玩家球（以下简称玩家）和食物球（以下简称食物），玩家可以食用食物，增加自身体积。
同时玩家也可以吞食其他比自身体积小的玩家，将根据被吞食玩家的体积大小获得相应体积加成收益。
游戏每局时长 1:30，游戏倒计时结束或人类玩家失败视作本局游戏结束，游戏结束将进行对局结算，显示得分。

1.1 新手保护期

任何玩家体重小于 20 都为新手保护期，此时为无敌状态，可以从其他玩家体内穿过。

1.2 吞食规则

当两玩家距离小于大球半径 且 大球体积比小球体积大 110%时，大球可以吞食小球。
吞食后大球体积将为

$$大球体积 = 大球体积 + \sqrt{小球体积}$$

这样可以保证球球体积越大吞食所增加的幅度越小，防止球球体积指数级增大。

1.3 人机规则

人机运动逻辑是本项目设计重难点，纵观电子游戏有史以来的人机设计思路，简单来说无非两条规则：战 or 逃，打得过则战，打不过则逃，猥琐发育后再战。
基于这个原则，本项目设计如下规则：

1、若人机玩家体积大于人类玩家，人机玩家会优先追击人类玩家；
2、若人机玩家体积小于人类玩家，首先判断是否被人类玩家追击；
3、若判断为正在被人类玩家追击，则寻找最短路径逃离出人类玩家体积 300%的范围，直到安全地带；
4、此时人机玩家会遵循最短路径原则优先吃最近食物，直到体积大于人类玩家；
5、重复过程 1。
人机算法代码实现将于 2.6 详细演示。

2. 技术点

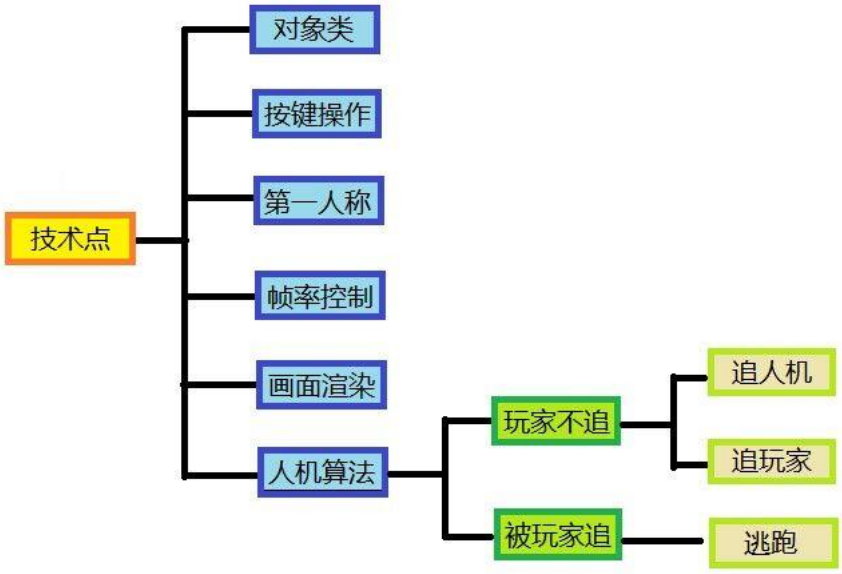

图 2-1 技术路线图

2.1 对象类

Ball 结构体的参数有 x, y, r, speed, life, color, index, cnt，分别控制 x/y 坐标、半径、速度、生命、颜色、追击参数、计时器。

```
struct Ball
{
    double x;
    double y;
    double r;
    double speed;
    bool life;
    COLORREF color;
    int index;
    int cnt;
};
```

玩家、食物皆使用 Ball 类创建对象。
game 数组表示所有玩家(包括人机玩家和人类玩家)，game[0] 是人类玩家；food 数组表示所有食物。
追击参数 index 系人机玩家使用，用于关联追击目标的下标，缺省值为-1，该参数具体使用见 2.6。
计时器 cnt 系复活刷新倒计时使用，该参数具体使用见 2.4。

2.2 按键操作

使用 *GetAsyncKeyState* 函数获取异步按键状态，检测某键是否按下。

相比于使用 *getchar* 函数获取按键状态，上述函数能使操作更加丝滑，无冲突响应多个按键，得以实现八个方向上的运动。

为了防止斜向运动比直线运动速度快，当同时按下多个按键时，球球运动需要基于斜向投影的 x, y 真实改变量。

引入极坐标变换公式

$$x = \rho * sin\alpha$$
$$y = \rho * cos\alpha$$

代入

$$\alpha = 45^\circ$$
$$r = speed$$

计算得

$$x = y = speed * \frac{\sqrt{2}}{2}$$

得到 speed 投影在 x 轴 y 轴上的坐标，即得出斜向运动的 x, y 真实改变量
代码如下

```
/*上面略*/
double realMove = game->speed;
if (GetAsyncKeyState(VK_UP) && GetAsyncKeyState(VK_LEFT) ||
GetAsyncKeyState(VK_UP) && GetAsyncKeyState(VK_RIGHT) ||
GetAsyncKeyState(VK_DOWN) && GetAsyncKeyState(VK_LEFT) ||
GetAsyncKeyState(VK_DOWN) && GetAsyncKeyState(VK_RIGHT))
{
    realMove *= sqrt(2) / 2;
}
/*其余略*/
```

2.3 第一人称视角

为模仿原版《球球大作战》游戏画面，将人类玩家球球锁定在屏幕中央，人类玩家球球运动的 x/y 坐标变化量将映射到地图上的其它元素上，作出相应反方向的 x/y 坐标变化，从而实现游戏的“第一人称视角”。
利用 EasyX 超出画布的元素依旧存在的原理，同时实现了无限地图，只要玩家设备内存够用，玩家一直往一个方向运动，地图将可以无限渲染。
代码如下

```
if (GetAsyncKeyState(VK_UP))
{
	for (int i = 0; i < FOOD_NUM; i++)
	{
		food[i].y += realMove;
	}
	for (int i = 1; i < GAME_NUM; i++)
	{
		game[i].y += realMove;
	}
}
/*其余略*/
```

2.4 帧率控制

为了保证球球匀速运动和游戏流畅度，保持游戏帧率为 60 FPS，本项目设计了帧率控制
原理是首先获取循环初的时间，再获取循环末的时间，得出循环一轮所需的时间，利用 Sleep 函数睡眠剩余时间，得以保证每一帧都均匀保持 60 FPS。
得益于实现了稳定的帧率锁定，游戏有关时间的操作（如：游戏对局倒计时，复活倒计时）都基于此，即 60 帧为一秒。
代码如下

```
#define FPS 1000/60.0
while (true) {
		int startTime = clock();
		/*中间略*/
		int frameTime = clock() - startTime;
		if (frameTime < FPS) {
			Sleep(FPS - frameTime);
		}
}
```

2.5 画面渲染

为实现较大体积球球渲染画面可以覆盖在较小体积球球渲染画面上，使用 STL:*sort* 函数自定义排序对玩家结构体数组进行排序再渲染。
代码如下

```
bool cmp(const Ball& a, const Ball& b){return a.r < b.r;}
void draw()
{
    //绘制所有食物
    for (int i = 0; i < FOOD_NUM; i++)
    {
        drawBall(food + i);
    }
    //绘制所有玩家  保证大的覆盖小的
    Ball temp[GAME_NUM];
    for (int i = 0; i < GAME_NUM; i++)
    temp[i] = game[i];
    sort(temp, temp + GAME_NUM, cmp);
    for (int i = 0; i < GAME_NUM; i++)
    {
        drawBall(temp + i);
    }
}
```

2.6 人机算法

为了实现人机玩家的有意义运动，编写 *moveGame* 函数实现人机对运动目标的锁定，使用 2.1 提及的 Ball 类中的 index 追击下标，index 缺省值为-1；若 index 为正数，则为锁定追击目标为关联食物的下标，若为-2，则为锁定追击目标为人类玩家。
之后，人机会判断是否正在被人类玩家追击，若是则寻找最短路径逃离出人类玩家体积 300%的范围，直到安全地带，调用逃跑算法 runGame 函数；若不是，则追击 index 追击下标中

的目标，调用追击算法 chaseGame 函数。
锁定追击目标，代码如下

```
void moveGame(Ball* game, Ball* food){
    //
    for (int i = 1; i < GAME_NUM; i++){
        if (!game[i].life) continue;
        if (game[i].r > game->r && game->r > PROTECT_R && game->life){
            game[i].index = -2;
            continue;
        }
        double minDistance = getwidth();
        for (int j = 0; j < FOOD_NUM; j++){
            if (!food[j].life) continue;
            double distance = DISTANCE((game + i), (food + j));
            if (distance < minDistance){
                minDistance = distance;
                game[i].index = j;
            }
        }
    }
```

战 or 逃，代码如下

```
for (int i = 1; i < GAME_NUM; i++){
        if (!game[i].life)continue;
        if (game[i].index < 0){
            chaseGame(game + i, game);
            continue;
        }
        if (game[i].r < game->r && DISTANCE((game + i), game) <= game->r * 3){
            runGame(game + i, game);
            continue;
        }
        chaseGame(game + i, food + game[i].index);
    }
}
```

三、 程序运行截图

图 3-1 开始界面

图 3-2 加载界面

图 3-3 新手保护期

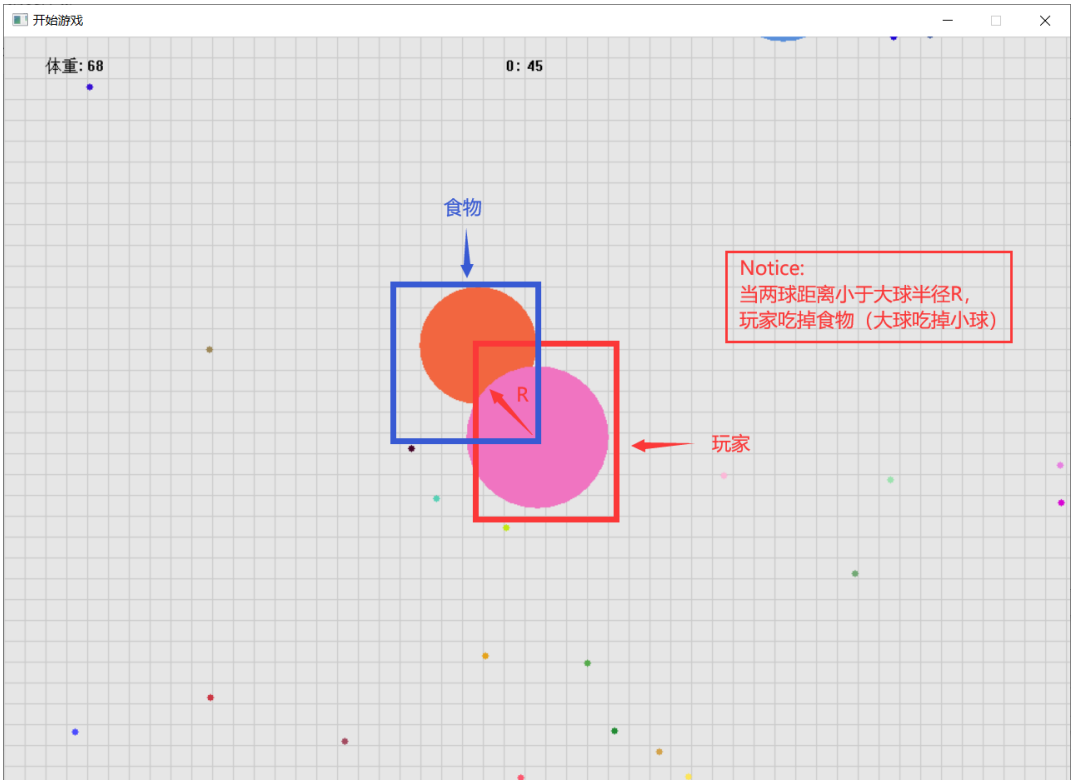

图 3-4 吞食规则

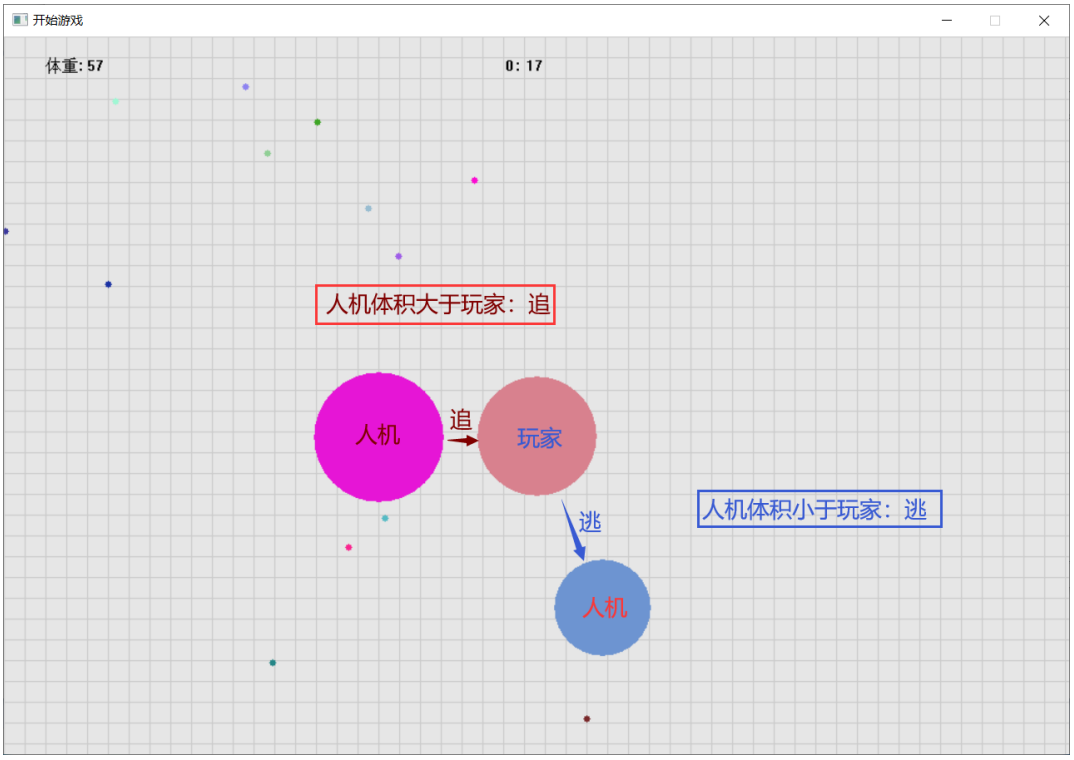

图 3-5 人机规则

图 3-6 结束界面_没存活到最后

图 3-7 结束界面_存活到最后

四、小组开发经验总结与心得体会

五、程序演示视频见-附件 1

六、程序源代码见-附件 2