**《球球大作战游戏项目报告》**

**一、项目概述**

本项目是一个基于 EasyX 图形库开发的球球大作战游戏，玩家操控一个或多个小球在游戏区域内移动，通过吃掉食物、其他 AI 小球或分裂出的小球来不断增大自身半径。游戏中包含多种操作，如分裂、吐球等，同时有 AI 对手与玩家竞争。游戏有开始界面、暂停界面、结束界面和胜利界面，通过按钮实现交互。

**二、项目目标**

开发一个具有一定趣味性和挑战性的球球大作战游戏，实现以下功能：

1. 游戏元素的初始化，包括食物、玩家小球、AI 小球和吐球。
2. 玩家小球的移动、分裂和吐球操作。
3. AI 小球的智能移动和进食行为。
4. 游戏界面的绘制和交互，包括开始、暂停、结束和胜利界面。
5. 碰撞检测和进食逻辑的实现。

**三、项目需求分析**

**功能需求**

1. **游戏初始化**：随机生成食物、玩家小球和 AI 小球的初始位置和属性。
2. **玩家操作**：支持鼠标移动控制玩家小球的移动方向，空格键实现分裂，Ctrl 键实现吐球。
3. **AI 行为**：AI 小球能够寻找最近的食物、猎物和威胁，并做出相应的移动和进食行为。
4. **碰撞检测**：检测玩家小球与食物、AI 小球、吐球之间的碰撞，以及 AI 小球之间的碰撞。
5. **界面交互**：提供开始、暂停、结束和胜利界面，通过按钮实现游戏的开始、继续、重新开始和退出操作。

**非功能需求**

1. **性能需求**：游戏运行流畅，响应及时，避免出现卡顿现象。
2. **兼容性需求**：支持 Windows 操作系统，使用 EasyX 图形库进行开发。
3. **可维护性需求**：代码结构清晰，模块化设计，便于后续的功能扩展和维护。

**四、项目设计**

**总体架构**

项目采用模块化设计，主要分为以下几个模块：

1. **游戏初始化模块**：负责游戏元素的初始化，包括食物、玩家小球、AI 小球和吐球。
2. **游戏绘制模块**：负责游戏界面的绘制，包括食物、玩家小球、AI 小球和吐球。
3. **玩家控制模块**：负责处理玩家的操作，包括鼠标移动、分裂和吐球。
4. **AI 模块**：负责 AI 小球的智能移动和进食行为。
5. **碰撞检测模块**：负责检测游戏元素之间的碰撞，并处理相应的逻辑。
6. **界面交互模块**：负责提供开始、暂停、结束和胜利界面，以及按钮的交互逻辑。

**类设计**

1. **Ball 类**：表示游戏中的小球，包含小球的位置、半径、颜色和状态等属性。
2. **Offset 类**：表示游戏视图的偏移量，用于实现视图的滚动。
3. **mouse 类**：表示鼠标的位置。
4. **next\_position 类**：表示小球的下一个位置。
5. **button 类**：表示游戏中的按钮，包含按钮的位置、大小、颜色和点击事件等属性。
6. **scene 类**：表示游戏中的场景，包含场景的位置、大小和绘制方法等属性。

**流程图**

plaintext

开始

|

V

初始化游戏

|

V

显示开始界面

|

V

等待玩家点击开始按钮

|

V

进入游戏循环

| |

| V

| 处理玩家输入

| |

| V

| 更新游戏状态

| |

| V

| 绘制游戏界面

| |

| V

| 检测碰撞

| |

| V

| 判断游戏是否结束

| |

| |--- 是 --- V

| | 显示结束或胜利界面

| | |

| | V

| | 等待玩家操作

| | |

| | |--- 重新开始 --- V

| | | 初始化游戏

| | | |

| | | V

| | | 进入游戏循环

| | |

| | |--- 退出 --- V

| | 结束程序

| |

| |--- 否 --- V

| 继续游戏循环

**五、项目实现**

**代码结构**

项目主要由以下几个文件组成：

1. **demo.cpp**：主程序文件，包含游戏的主要逻辑和控制流程。
2. **button.h** 和 **button.cpp**：按钮类的头文件和实现文件，负责按钮的绘制和交互逻辑。
3. **scene.h** 和 **scene.cpp**：场景类的头文件和实现文件，负责场景的绘制和显示。

**关键代码实现**

**游戏初始化**

cpp

void GameInit() {

srand((unsigned)time(NULL));

// 初始化食物

for (int i = 0; i < food\_num; i++) {

food[i].x = rand() % Width;

food[i].y = rand() % Height;

food[i].flag = true;

food[i].r = 4;

food[i].color = RGB(rand() % 256, rand() % 256, rand() % 256);

}

// 初始化AI小球

for (int i = 0; i < Ai\_num; i++) {

ai[i].x = rand() % Width;

ai[i].y = rand() % Height;

ai[i].flag = true;

ai[i].r = 20;

ai[i].color = RGB(rand() % 256, rand() % 256, rand() % 256);

}

// 初始化玩家小球

player[0].x = rand() % Width;

player[0].y = rand() % Height;

player[0].flag = true;

player[0].r = 20;

player[0].color = RGB(rand() % 256, rand() % 256, rand() % 256);

for (int i = 1; i < max\_split; i++) {

player[i].flag = false;

}

// 初始化偏移量

offset.x = player[0].x - VIEW\_WIDTH / 2;

offset.y = player[0].y - VIEW\_HEIGHT / 2;

// 初始化吐球

for (int j = 0; j < max\_split; j++) {

for (int i = 0; i < SPIT\_BALL\_NUM; i++) {

spitBalls[j][i].flag = false;

}

}

}

**玩家控制**

cpp

void Control() {

ExMessage msg;

while (peekmessage(&msg)) {

if (msg.message == WM\_MOUSEMOVE) {

M.x = msg.x;

M.y = msg.y;

}

else if (msg.message == WM\_KEYDOWN) {

if (msg.vkcode == VK\_SPACE) {

for (int i = 0; i < max\_split; i++) {

if (player[i].flag) {

split(player[i].x, player[i].y, i);

}

}

}

else if (msg.vkcode == VK\_CONTROL) {

for (int i = 0; i < max\_split; i++) {

if (player[i].flag) {

spit(i);

}

}

}

}

}

}

**AI 行为**

cpp

void aiplay() {

for (int i = 0; i < Ai\_num; i++) {

if (ai[i].flag == 1) {

double target\_x, target\_y;

double threat\_x, threat\_y;

double threat\_x1, threat\_y1;

double threat\_x2, threat\_y2;

// 寻找最近的威胁

findNearestThreat(i, threat\_x1, threat\_y1);

findNearestThreat\_player(i, threat\_x2, threat\_y2);

if (getDistance(ai[i].x, threat\_x1, ai[i].y, threat\_y1) < getDistance(ai[i].x, threat\_x2, ai[i].y, threat\_y2)) {

threat\_x = threat\_x1;

threat\_y = threat\_y1;

}

else {

threat\_x = threat\_x2;

threat\_y = threat\_y2;

}

if (getDistance(ai[i].x, threat\_x, ai[i].y, threat\_y) < 50) {

double dx = ai[i].x - threat\_x;

double dy = ai[i].y - threat\_y;

target\_x = ai[i].x + dx;

target\_y = ai[i].y + dy;

}

else {

bool foundPrey = false;

// 寻找最近的猎物

double prey\_x1, prey\_y1;

findNearestPrey(i, prey\_x1, prey\_y1);

double prey\_x2, prey\_y2;

findNearestPrey\_player(i, prey\_x2, prey\_y2);

if (getDistance(ai[i].x, prey\_x1, ai[i].y, prey\_y1) < getDistance(ai[i].x, prey\_x2, ai[i].y, prey\_y2)) {

target\_x = prey\_x1;

target\_y = prey\_y1;

foundPrey = true;

}

else {

target\_x = prey\_x2;

target\_y = prey\_y2;

foundPrey = true;

}

if (!foundPrey) {

// 没有猎物，寻找最近的食物

findNearestFood(i, target\_x, target\_y);

}

}

aimove(target\_x, target\_y, i);

}

ai\_eat(i);

}

}

**六、项目测试**

**测试环境**

* 操作系统：Windows 10
* 开发工具：Visual Studio 2019
* 图形库：EasyX

**测试用例**

| **测试用例编号** | **测试场景** | **测试步骤** | **预期结果** | **实际结果** | **是否通过** |
| --- | --- | --- | --- | --- | --- |
| 1 | 游戏开始 | 启动程序，点击开始按钮 | 进入游戏界面，显示食物、玩家小球和 AI 小球 | 与预期结果一致 | 是 |
| 2 | 玩家移动 | 在游戏界面中移动鼠标 | 玩家小球向鼠标指针方向移动 | 与预期结果一致 | 是 |
| 3 | 玩家分裂 | 在游戏界面中按下空格键 | 玩家小球分裂成多个小球 | 与预期结果一致 | 是 |
| 4 | 玩家吐球 | 在游戏界面中按下 Ctrl 键 | 玩家小球吐出小球 | 与预期结果一致 | 是 |
| 5 | AI 移动 | 观察 AI 小球的移动 | AI 小球能够寻找最近的食物、猎物和威胁，并做出相应的移动 | 与预期结果一致 | 是 |
| 6 | 碰撞检测 | 让玩家小球与食物、AI 小球、吐球发生碰撞 | 玩家小球半径增大，相应的食物、AI 小球或吐球消失 | 与预期结果一致 | 是 |
| 7 | 游戏暂停 | 在游戏界面中按下 ESC 键 | 显示暂停界面，游戏暂停 | 与预期结果一致 | 是 |
| 8 | 游戏结束 | 让玩家小球被 AI 小球吃掉 | 显示结束界面，提供重新开始和退出选项 | 与预期结果一致 | 是 |
| 9 | 游戏胜利 | 让玩家小球吃掉所有 AI 小球 | 显示胜利界面，提供重新开始和退出选项 | 与预期结果一致 | 是 |

**七、项目总结**

**项目成果**

本项目成功开发了一个贪吃球游戏，实现了游戏的基本功能，包括游戏初始化、玩家操作、AI 行为、碰撞检测和界面交互等。游戏运行流畅，响应及时，具有一定的趣味性和挑战性。

**存在问题**

1. **AI 智能度有待提高**：AI 小球的行为逻辑相对简单，只能寻找最近的食物、猎物和威胁，缺乏更复杂的策略和决策能力。
2. **游戏性能优化**：在游戏中，当小球数量较多时，可能会出现卡顿现象，需要进一步优化游戏性能。
3. **界面设计**：游戏界面的设计较为简单，缺乏美观性和交互性，需要进一步改进。

**改进方向**

1. **提升 AI 智能度**：引入更复杂的 AI 算法，如遗传算法、神经网络等，提高 AI 小球的决策能力和策略水平。
2. **优化游戏性能**：采用更高效的算法和数据结构，减少不必要的计算和渲染，提高游戏的运行效率。
3. **改进界面设计**：使用更丰富的图形资源和动画效果，提升游戏界面的美观性和交互性。

**八、附录**

**代码清单**

1. **demo.cpp**
2. **button.h**
3. **button.cpp**
4. **scene.h**
5. **scene.cpp**

**参考文献**

1. EasyX 图形库官方文档
2. 《C++ Primer》
3. 《数据结构与算法分析》