高级语言程序设计

实验报告

南开大学 计算机大类

王刘浏

2411272

计算机学院模拟1-3班

2025年4月26日

目录

[高级语言程序设计大作业实验报告 1](#_Toc21286)

[一. 作业题目 1](#_Toc29969)

[二. 开发软件 1](#_Toc18364)

[三. 课题要求 1](#_Toc742)

[四. 主要流程 1](#_Toc25240)

[1． 整体流程 1](#_Toc20452)

[2． 算法或公式 2](#_Toc25354)

（1） 开始界面设计 [2](#_Toc24674)

[（2） 蛇的移动 2](#_Toc22508)

[（3） 食物及障碍物的随机生成 3](#_Toc28665)

[（4） 碰撞检测 3](#_Toc32428)

[（5） 鼠标输入及控制处理 4](#_Toc23860)

[（6） 星星的闪烁及移动 5](#_Toc13349)

（7） 旋转五角星绘制 [6](#_Toc24674)

（8） 分数系统及结束设计 [6](#_Toc24674)

[五. 代码及功能展示 7](#_Toc28265)

[六. 总结与收获 7](#_Toc23116)

[1．对文件属性进行修改](#_Toc10799) 7

[2．类的相关知识的学习与运用](#_Toc16275) 7

高级语言程序设计大作业实验报告

一、作业题目

实现动态星空背景下的贪吃蛇小游戏。

二、开发软件

Visual Studio 2022

EasyX图形库

三、课题要求

1. 面向对象。
2. 学习使用图形库提供的函数实现图形编程。
3. 在gitee/github的提交体现大作业完成过程，反映阶段性完成的情况。

大作业GitHub网址：https://github.com/Wll-nku/C-dazuoye

四、主要流程

1. 整体流程：首先创建了两个文件login .cpp以及main.cpp。

对于login文件，其主要实现创建窗口以创建开始界面；对于main文件，其主要通过星星类（用于设计背景）、食物类（用于设计得分系统）、障碍物类（用于提高游戏难度）、蛇类（用于设计与贪吃蛇）等构成。

下面让我们通过相应的有关的主要算法以及公式来了解下该小游戏大致的实现方式。

1. 算法或公式：

（1）、开始界面设计： case WM\_COMMAND: {

if (LOWORD(wParam) == 1) {

// 处理开始游戏按钮点击事件

MessageBox(hwnd, \_T("游戏即将开始！"), \_T("提示"), MB\_OK);

// 关闭登录窗口

DestroyWindow(hwnd);

// 启动贪吃蛇游戏

playSnakeGame();

}

break;

}

当 LOWORD(wParam) == 1 时，就表明用户点击了 “开始游戏” 按钮。

游戏启动流程：

弹出提示框：利用 MessageBox 函数弹出一个提示框，告知用户游戏即将开始。

关闭登录窗口：使用 DestroyWindow 函数关闭当前的登录窗口。

启动游戏：调用 playSnakeGame 函数来启动贪吃蛇游戏。

（2）、蛇的移动：

void move() {

for (int i = length - 1; i > 0; i--) {

x[i] = x[i - 1];

y[i] = y[i - 1];

}

switch (direction) {

case 1:

x[0] += BLOCK\_SIZE;

break;

case 2:

y[0] += BLOCK\_SIZE;

break;

case 3:

x[0] -= BLOCK\_SIZE;

break;

case 4:

y[0] -= BLOCK\_SIZE;

break;

}

#### }**算法思路：**

蛇的每个身体部分都用一个坐标 (x, y) 来表示，这些坐标存储在数组里。蛇移动时，每个身体部分会移动到前一个身体部分的位置。

（3）、食物及障碍物的随机生成：

void Food::init(Snake& snake, Obstacle& obstacle) {

count = rand() % MAX\_FOODS + 1; // 随机生成 1 到 MAX\_FOODS 个食物

int maxAttempts = 1000; // 最大尝试次数，避免死循环

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

int attempts = 0;

do {

x[i] = rand() % (WIDTH / BLOCK\_SIZE) \* BLOCK\_SIZE;

y[i] = rand() % (HEIGHT / BLOCK\_SIZE) \* BLOCK\_SIZE;

attempts++;

if (attempts > maxAttempts) {

// 超出最大尝试次数，输出错误信息

MessageBox(NULL, \_T("食物初始化失败，可能地图空间不足！"), \_T("错误"), MB\_OK | MB\_ICONERROR);

return;

}

} while (isSnakeBody(snake, x[i], y[i]) ||

isObstacle(obstacle, x[i], y[i]) ||

isFoodDuplicate(i));

angle[i] = 0; // 初始角度为 0

}

}

食物和障碍物的位置是随机生成的，不过要保证它们不会出现在蛇的身体上。

（4）、碰撞检测：

// 判断是否吃到食物，检查蛇头是否和任意一个食物位置重合

bool checkEat(Snake& snake, Food& food) {

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

if (snake.x[0] == food.x[i] && snake.y[0] == food.y[i]) {

return true;

}

}

return false;

}

// 判断是否撞到障碍物

bool checkHitObstacle(Snake& snake, Obstacle& obstacle) {

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

if (snake.x[0] == obstacle.x[i] && snake.y[0] == obstacle.y[i]) {

return true;

}

}

return false;

}

主函数中：

// 检查是否撞到边界、自己或障碍物

if (snake.x[0] < 0 || snake.x[0] >= WIDTH || snake.y[0] < 0 || snake.y[0] >= HEIGHT ||

checkHitObstacle(snake, obstacle)) {

break;

}

for (int i = 1; i < snake.length; i++) {

if (snake.x[0] == snake.x[i] && snake.y[0] == snake.y[i]) {

break;

}

}

碰撞检测用于判断蛇是否吃到食物、撞到障碍物或者撞到自己。

#### **算法思路**

通过比较蛇头的坐标和食物、障碍物、蛇身其他部分的坐标，来判断是否发生碰撞。

（5）、鼠标输入及控制处理：

// 处理鼠标输入

void handleMouseInput(Snake& snake) {

MOUSEMSG msg;

while (MouseHit()) {

msg = GetMouseMsg();

if (msg.uMsg == WM\_LBUTTONDOWN) {

int clickX = msg.x;

int clickY = msg.y;

int headX = snake.x[0];

int headY = snake.y[0];

if (abs(clickX - headX) > abs(clickY - headY)) {

if (clickX > headX && snake.direction != 3) {

snake.direction = 1;

}

else if (clickX < headX && snake.direction != 1) {

snake.direction = 3;

}

}

else {

if (clickY > headY && snake.direction != 4) {

snake.direction = 2;

}

else if (clickY < headY && snake.direction != 2) {

snake.direction = 4;

}

}

}

}

}

依据鼠标点击的位置，改变蛇的移动方向。

算法思路

获取鼠标点击的坐标，计算点击位置与蛇头位置的差值，然后依据差值的大小来确定蛇的移动方向。

（6）、星星的闪烁及移动：

x = rand() % WIDTH;

y = rand() % HEIGHT;

brightness = rand() % 256;

changeRate = (rand() % 3) - 1; // -1, 0, 1

if (changeRate == 0) {

changeRate = 1;

}

moveX = (rand() % 3) - 1; // -1, 0, 1

moveY = (rand() % 3) - 1; // -1, 0, 1

}

void update() {

brightness += changeRate;

if (brightness > 255) {

brightness = 255;

changeRate = -changeRate;

}

else if (brightness < 0) {

brightness = 0;

changeRate = -changeRate;

}

x += moveX;

y += moveY;

// 边界处理

if (x < 0) x = WIDTH;

if (x > WIDTH) x = 0;

if (y < 0) y = HEIGHT;

if (y > HEIGHT) y = 0;

}

};

星星的亮度会随机变化，并且会在屏幕上随机移动。

#### **算法思路**

给每个星星设定一个初始亮度和变化速率，每帧更新亮度，当亮度超出范围时，改变变化速率的方向。同时，星星会在 x 和 y 方向上随机移动，到达边界时会从另一侧出现。

1. 、旋转五角星绘制：

// 绘制旋转的五角星

for (int i = 0; i < 10; i++) {

double a = i \* 2 \* M\_PI / 10 + angle;

int r = (i % 2 == 0) ? r1 : r2;

pts[i].x = static\_cast<int>(x + r \* cos(a));

pts[i].y = static\_cast<int>(y + r \* sin(a));

vertexArray[2 \* i] = pts[i].x;

vertexArray[2 \* i + 1] = pts[i].y;

}

COLORREF yellowColor = RGB(255, 255, 0);

setfillcolor(yellowColor);

fillpoly(10, vertexArray);

在绘制旋转的五角星时，运用了三角函数公式来计算五角星的顶点坐标。

#### **公式思路**

借助三角函数 cos 和 sin 来计算五角星顶点在旋转后的坐标。

1. 、分数系统及结束设计：

/ 绘制得分信息

TCHAR scoreText[20];

\_stprintf\_s(scoreText, \_T("Score: %d"), score);

settextcolor(WHITE);

settextstyle(30, 0, \_T("宋体"));

outtextxy(WIDTH - 200, 10, scoreText);

// 显示游戏结束信息和最终得分

settextcolor(RED);

settextstyle(60, 0, \_T("宋体"));

outtextxy(WIDTH / 2 - 150, HEIGHT / 2 - 30, \_T("Game Over!"));

TCHAR finalScoreText[20];

\_stprintf\_s(finalScoreText, \_T("Final Score: %d"), score);

settextcolor(WHITE);

settextstyle(30, 0, \_T("宋体"));

outtextxy(WIDTH / 2 - 100, HEIGHT / 2 + 30, finalScoreText);

Sleep(2000);

五、代码及功能展示



1. 总结与收获

在完成这次C++大作业的过程中，我遇到了不少困难，也通过各种渠道了解并尝试进行处理。

1. 对文件属性进行修改来实现特定内容

比如我在进行界面设计的时候就出现了“无法解析的外部符号 \_\_imp\_InitCommonControls”这种我从来没有见过的问题，经过一番努力，我才了解到这是链接时没有包含comctl32.lib这个库导致链接器无法找到InitCommonControls 函数的实现，于是，我对文件的属性进行修改，在“配置属性”->“链接器”->“输入”->“附加依赖项”中添加“comctl32.lib”才得以解决这种问题。

1. 类的相关知识的学习与使用

本次大作业也加深了我对于类与对象的理解与使用，比如在main文件中，snake类、food类以及obstacle类，这三个类中每一个类定义时都离不开另外两个类，而另外两个类要想定义却又离不开我需要进行定义的这个类，一开始我一直苦恼于这个死循环，随后我进行一番研究后才学会前向声明这种方法，最终得以实现类的定义。

同时，还有一个令我印象深刻的在完成大作业时学习到的知识点是类定义与方法实现的分离，比如通过将 Food 类和 Obstacle 类的方法实现移到类定义的外面，当编译器处理类定义时，只需要知道类的基本结构（有哪些成员变量和成员函数的声明），而不需要知道具体的方法实现细节。

当编译器处理完所有类的定义后，再去处理方法实现部分。此时，所有类的完整定义都已经被编译器知晓，所以在 Food 类的方法实现中使用 Obstacle 类的成员，或者在 Obstacle 类的方法实现中使用 Food 类的成员，编译器都能够正确识别，从而避免了因为类定义不完整而导致的报错问题。