南开大学 计算机大类

姓名：张容豪

学号：2410855

班级：0516

2025年5月13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](#_Toc198066246)

[**一.** **作业题目** 2](#_Toc198066247)

[**二.** **开发软件** 2](#_Toc198066248)

[**三.** **课题要求** 2](#_Toc198066249)

[**四.** **主要流程** 2](#_Toc198066250)

[**1． 整体流程** 2](#_Toc198066251)

[2. 关键算法与公式 4](#_Toc198066252)

[3. 单元测试设计 5](#_Toc198066253)

[**五.** **单元测试及结果** 5](#_Toc198066254)

[测试环境 6](#_Toc198066255)

[2. 典型测试案例 6](#_Toc198066256)

[3. 问题与修复 6](#_Toc198066257)

[**六.** **收获** 7](#_Toc198066258)

[1. 面向对象设计的实践 7](#_Toc198066259)

[2. 图形编程与事件处理 7](#_Toc198066260)

[3. 碰撞检测与逻辑控制 7](#_Toc198066261)

[4. 工程化思维 7](#_Toc198066262)

高级语言程序设计大作业实验报告

1. **作业题目**

贪吃蛇游戏的设计与实现

1. **开发软件**

Visual Studio 2022

1. **课题要求**
2. 面向对象。
3. 核心功能。
4. 用户交互。
5. 代码规范。
6. **主要流程**
   1. **整体流程**

#### （1）类设计与继承关系

* **基类Sprite**：定义游戏元素的通用属性（位置(x,y)、颜色COLORREF）和方法（绘制draw()、移动moveBy()、碰撞检测collision()）。
* **派生类**
  + Snake：继承Sprite，使用vector<Sprite>存储蛇身节点，实现移动逻辑（bodyMove()）、碰撞检测（自身、围墙、食物）、增长机制（increment()）。
  + Food：继承Sprite，重写draw()方法（加载食物图片），实现随机位置生成（changeFoodPos()）。
  + Wall：继承Sprite，实现围墙绘制（setWall()），定义游戏边界（640×480 像素，以 10×10 像素格子划分）。
  + GameSence：游戏场景管理类，整合所有元素，处理输入（onMsg()）、游戏循环（run()）、碰撞逻辑（吃食物、撞墙、撞自身）、音效播放（playEatSound()）。
* Sprite
* ├─ Snake
* │ ├─ nodes: vector<Sprite>
* │ ├─ dir: int（方向键常量）
* │ ├─ draw() override
* │ ├─ bodyMove(), increment(), collisionWall(), collisionSelf()
* ├─ Food
* │ ├─ draw() override
* │ ├─ changeFoodPos()
* ├─ Wall
* │ ├─ setWall()
* GameSence
* ├─ snake: Snake
* ├─ food: Food
* ├─ wall: Wall
* ├─ run(), onMsg(), snakeEatFood(), playEatSound()

#### （2）游戏循环逻辑

1. **初始化**：创建窗口（640×480）、加载资源（图片、音效）、显示开始界面。
2. **输入处理**：通过peekmessage监听键盘事件，控制蛇的转向（禁止反向移动）。
3. **状态更新**
   * 蛇移动：头部按方向移动，身体节点跟随。
   * 碰撞检测：检测蛇头与食物、围墙、自身的碰撞。
   * 食物生成：蛇吃到食物后，随机生成新食物（避开围墙区域）。
4. **渲染**：使用BeginBatchDraw/EndBatchDraw双缓冲绘图，减少画面闪烁。
5. **节奏控制**：通过Sleep(100)控制游戏速度，可调整参数改变难度。

### 2. 关键算法与公式

#### （1）碰撞检测算法

* **坐标精确匹配**：两物体碰撞条件为x坐标相同且y坐标相同（因所有元素以 10×10 像素格子为单位移动）。

bool Sprite::collision(const Sprite& other) {

return m\_x == other.m\_x && m\_y == other.m\_y;

}

* **围墙边界检测**：蛇头坐标等于边界格子坐标时判定碰撞（如左边界x=0，上边界y=0）。

#### （2）食物随机生成算法

1. 定义围墙区域网格wallData[64][48]，标记边界格子为1。
2. 生成随机坐标(tem\_x, tem\_y)，确保wallData[tem\_x/10][tem\_y/10] == 0（非围墙区域）。

void Sprite::changeFoodPos() {

int tem\_x, tem\_y;

do {

tem\_x = rand() % 64 \* 10;

tem\_y = rand() % 48 \* 10;

} while (wallData[tem\_x/10][tem\_y/10] == 1); // 避开围墙

m\_x = tem\_x; m\_y = tem\_y;

}

#### （3）方向控制逻辑

禁止蛇直接反向移动（如向上移动时不能直接向下），确保运动逻辑合理：

if (msg.vkcode == VK\_UP && snake.dir != VK\_DOWN) snake.dir = VK\_UP;

### 3. 单元测试设计

针对核心功能设计测试用例，验证逻辑正确性。

#### （1）蛇移动测试

| **输入** | **预期输出** | **目的** |
| --- | --- | --- |
| 初始方向右，调用bodyMove() | 头部x+10，身体节点跟随 | 验证移动逻辑 |
| 方向切换为上（当前方向右） | 头部向上移动，无反向错误 | 验证转向限制 |

#### （2）碰撞检测测试

| 输入 | 预期输出 | 目的 |  
| 蛇头坐标(0, 10)|collisionWall() == true| 验证围墙碰撞 |  
| 蛇头与身体节点坐标相同 |collisionSelf() == true| 验证自身碰撞 |  
| 蛇头与食物坐标相同 |snakeEatFood()触发，蛇长度 + 1 | 验证食物吞噬 |

#### （3）食物生成测试

| 输入 | 执行changeFoodPos()| 目的 |  
| 多次调用 | 食物坐标不在围墙区域（如x=0或x=630时不会出现）| 验证边界避开逻辑 |

1. **单元测试及结果**

### 测试环境

* 使用EasyX的图形窗口模拟游戏场景，通过手动输入方向键和观察界面变化验证功能。
* 代码中添加临时输出语句（如cout << "Collision Detected"）辅助判断逻辑是否触发。

### 2. 典型测试案例

#### （1）边界碰撞测试

* **操作**：控制蛇头移动至左边界（x=0）。
* **预期**：程序输出 “Game over” 并退出。
* **结果**：正确触发边界碰撞逻辑，游戏结束。

#### （2）自身碰撞测试

* **操作**：使蛇头移动至身体节点位置（如故意绕圈）。
* **预期**：程序输出 “Game over” 并退出。
* **结果**：正确检测到自身碰撞，游戏结束。

#### （3）食物吞噬测试

* **操作**：蛇头接触食物，观察长度变化和新食物生成。
* **预期**：蛇身增加一节，食物位置更新且不在围墙上。
* **结果**：功能正常，音效正确播放。

### 3. 问题与修复

* **问题**：初始版本中食物可能生成在围墙上（因wallData初始化不完整）。
* **修复**：补充围墙网格的四个边界初始化，确保wallData正确标记所有边界格子。

1. **收获**

### 1. 面向对象设计的实践

* **基类抽象**：通过Sprite基类统一游戏元素的基本行为（绘制、移动、碰撞），派生类专注于特有逻辑（如蛇的增长、食物的随机生成），提高代码复用性。
* **继承与多态**：虽然代码中未使用虚析构函数（因GameSence类直接管理对象生命周期），但通过派生类重写draw()方法，实现了不同元素的差异化绘制（如蛇头加载图片、食物显示图标）。

### 2. 图形编程与事件处理

* **双缓冲技术**：使用BeginBatchDraw/EndBatchDraw避免画面闪烁，理解图形渲染的基本优化策略。
* **键盘事件监听**：通过peekmessage实现非阻塞输入处理，确保游戏循环与用户交互同步。

### 3. 碰撞检测与逻辑控制

* **坐标网格设计**：将游戏区域划分为 10×10 像素的格子，简化碰撞检测逻辑（仅需坐标相等判断）。
* **方向限制逻辑**：通过禁止反向移动，避免蛇 “穿身” 等不合理行为，提升游戏体验。

### 4. 工程化思维

* **模块化设计**：将游戏逻辑拆分为Snake、Food、GameSence等类，降低代码耦合度，便于后续功能扩展（如添加障碍物、加速道具）。