数据结构与算法设计（C++描述）

专题报告

实验名称： **贪吃蛇游戏**

	姓名	学号	专业
小组成员	[redacted]	[redacted]	电子信息工程（徐特立英才班）
	[redacted]	[redacted]	电子信息工程（徐特立英才班）
	[redacted]	[redacted]	电子信息工程（徐特立英才班）
	[redacted]	[redacted]	电子信息工程（徐特立英才班）
	[redacted]	[redacted]	电子信息工程（徐特立英才班）
任课教师：	[redacted]	备注：无	

版本

版本号	描述/改动	日期	责任人
1.0	初版	[redacted]	[redacted]
2.0	控制台交互设计	[redacted]	[redacted]

一、基本情况

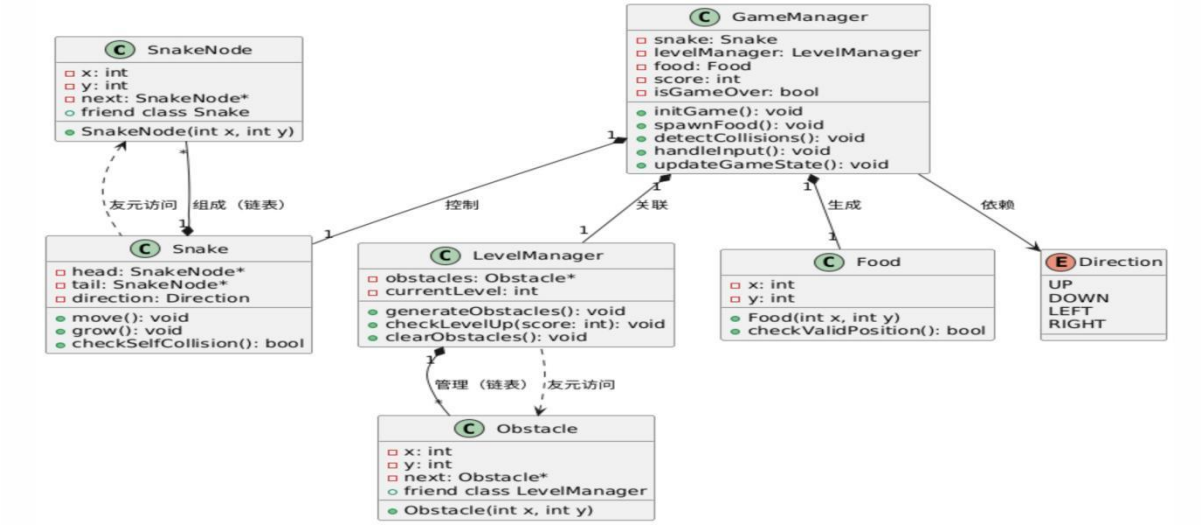
题目内容	设计并实现一款贪吃蛇游戏，在游戏中玩家可以控制贪吃蛇寻找食物，每吃一口食物蛇身就会变长并且获得游戏分数，当贪吃蛇碰到墙壁、障碍物或者咬到自己的身体则游戏结束。
题目要求及约束条件	在遵循上述游戏规则的前提下，玩家可以使用键盘控制贪吃蛇移动方向。在屏幕中可以随机产生食物，并且贪吃蛇吃到食物后蛇身变长同时获得一定分数。贪吃蛇的头不能碰到自己的身体，也不能碰到墙壁，否则游戏结束。游戏分为闯关模式和无限模式供玩家选择，并提供暂停\继续、加速\减慢蛇身移动等操作以增强玩家的游戏体验。
成员分工	[redacted]

二、设计与实现

1、设计思想

贪吃蛇作为经典实时交互游戏，其核心挑战在于动态数据管理与高效碰撞检测。本研究基于 C++语言，采用模块化架构与链式数据结构实现多模式贪吃蛇系统，支持无限模式与闯关模式供玩家选择。实现了贪吃蛇的蛇身生成、位置判定、移动控制和碰撞判定等功能。与此同时，程序基于 Windows API（SetConsoleCursorPosition SetConsoleTextAttribute）实现低延迟渲染（帧率 60 FPS），结合非阻塞输入（_kbhit）确保实时响应。通过独立类封装蛇身（SnakeNode）、障碍物（Obstacle）与游戏逻辑控制器（GameManager），提升代码可维护性

2、类结构



1. 蛇身节点(SnakeNode)类：存储每一节蛇身的坐标位置以及指向下一节点的指针。同时将相应的函数设为友元，实现蛇身创建、蛇身移动、碰撞检测等操作。
   优势：存储蛇身节点坐标及链表关系，通过友元类 Snake 实现坐标更新与碰撞检测的封装。友元机制在保证数据封装性的同时，允许逻辑类直接操作节点数据，避免冗余接口。

2. 障碍物(Obstacle)类：存储每一个障碍物的坐标位置以及指向下一个障碍物的指针。同时将创建障碍物以及碰撞检测等函数设为友元，保证游戏的正常进行。
   优势：存储障碍物坐标及链表关系，友元类 LevelManager 负责动态生成与碰撞检测。单向链表支持快速插入（0(1)）与按关卡动态扩容，内存占用仅与障碍物数量成正比。

3、主要数据结构

1. 链式队列：程序在创建蛇身运用到进队操作，蛇身移动和吃到食物延长蛇身的时候同时运用进队和出队操作，无需通过屏幕的刷新来进行移动，从而有效避免闪屏的问题。
2. 单向链表：程序在创建障碍物时运用单向链表，有利于后续关卡障碍物个数增加的操作。

使用这种数据结构主要有以下优点：

1. 蛇身伸缩的空间复杂度较低：链式队列的进队出队的时间复杂度为 O(1)，比数组的批量移动元素（O(n)）更高效。
2. 按需分配内存：系统根据蛇身的长度和障碍物的数量动态分配内存，避免了存储空间的浪费。
3. 高度契合游戏逻辑：链表的顺序遍历特性完美匹配蛇身移动时的坐标更新需求。蛇身和障碍物使用独立链表结构，避免数据混杂

4、算法设计

1. 游戏流程：通过 welcometogame、gamecircle、endgame 函数实现。首先进入开始界面，玩家可选择无限模式和闯关模式这两种游戏模式。进入游戏后，在控制台左侧显示游戏界面，右侧显示当前模式、得分和食物分值等提示信息，以及游戏操作说明。通过键盘可以控制蛇身移动以及游戏状态。当退出游戏时控制台会打印退出原因和最终得分等信息。
2. 部分键盘操作：在无尽模式下，F1 实现移动加速，两种状态最短间隔时间不超过 50ms，并且移速加快每次吃到食物可以获得一定加分。F2 实现移动减速，两种状态最长间隔时间不超过 350ms，并且移速加快每次吃到食物可以获得一定减分。F3 重置为游戏初始状态。
3. 关卡升级：通过 checkLevelUp 函数实现。在闯关模式下每获得 100 分进入下一关，每过一关障碍物的个数增加 3 个并且蛇身移动速度加快，从而增加闯关难度，一共有 5 关。
4. Windows 控制台 API 实现控制台交互：我们通过添加头文件” windows.h”实现控制台窗口的图形界面设计，编写 Pos 函数实现光标位置控制，SetColor 函数实现光标颜色设置，同时通过键盘可以控制贪吃蛇的移动以及游戏设置。
5. 游戏初始化：通过 creatMap 函数打印墙壁，并通过 initsnake 函数创建一个四个节点的链式队列初始化蛇身并打印在控制台上。

6. 食物创建：通过 createfood 函数实现。首先创建一个蛇身节点类的对象作为食物，坐标随机指定，并且保证食物节点与蛇身节点不重合，最后以绿色打印在控制台中。
7. 障碍物创建：障碍物只在闯关模式中出现，通过 createObstacles 函数进行创建。首先进行初始化，删除所有已存在的障碍物节点。然后根据关卡数设置障碍物数：障碍物数量=(关卡数-1) × 3，并且障碍物最多不超过 12 个。障碍物采用单向链表的形势创建，坐标随机指定，并且保证其不予已存在的蛇身节点、食物节点、墙壁和已存在的障碍物节点相重合，最后以蓝色打印在控制台上。
8. 蛇身移动：通过 snakemove 和 moveSnake 两个函数实现。通过之前键盘控制可以得到贪吃蛇当前的移动方向，并计算出下一时刻头部的坐标。根据这个坐标创建下一时刻头部的蛇身节点，如果吃到食物，直接将新的头部节点插入队列并打印在控制台上即可，并生成下一个食物节点。如果未吃到食物，将新的头部节点插入队列后并将尾部节点出队，最后再打印在控制台上，即可得到移动后的蛇身，通过出队入队的方式，可以避免闪屏移动问题。
9. 碰撞检测：通过 biteself、cantcrosswall、hitObstacle 三个函数实现。通过检测贪吃蛇的头部节点是否与其他蛇身节点、障碍物节点、墙壁的坐标重合，判断是否发生碰撞，以此决定是否结束游戏。

流程图

1. 主程序流程图

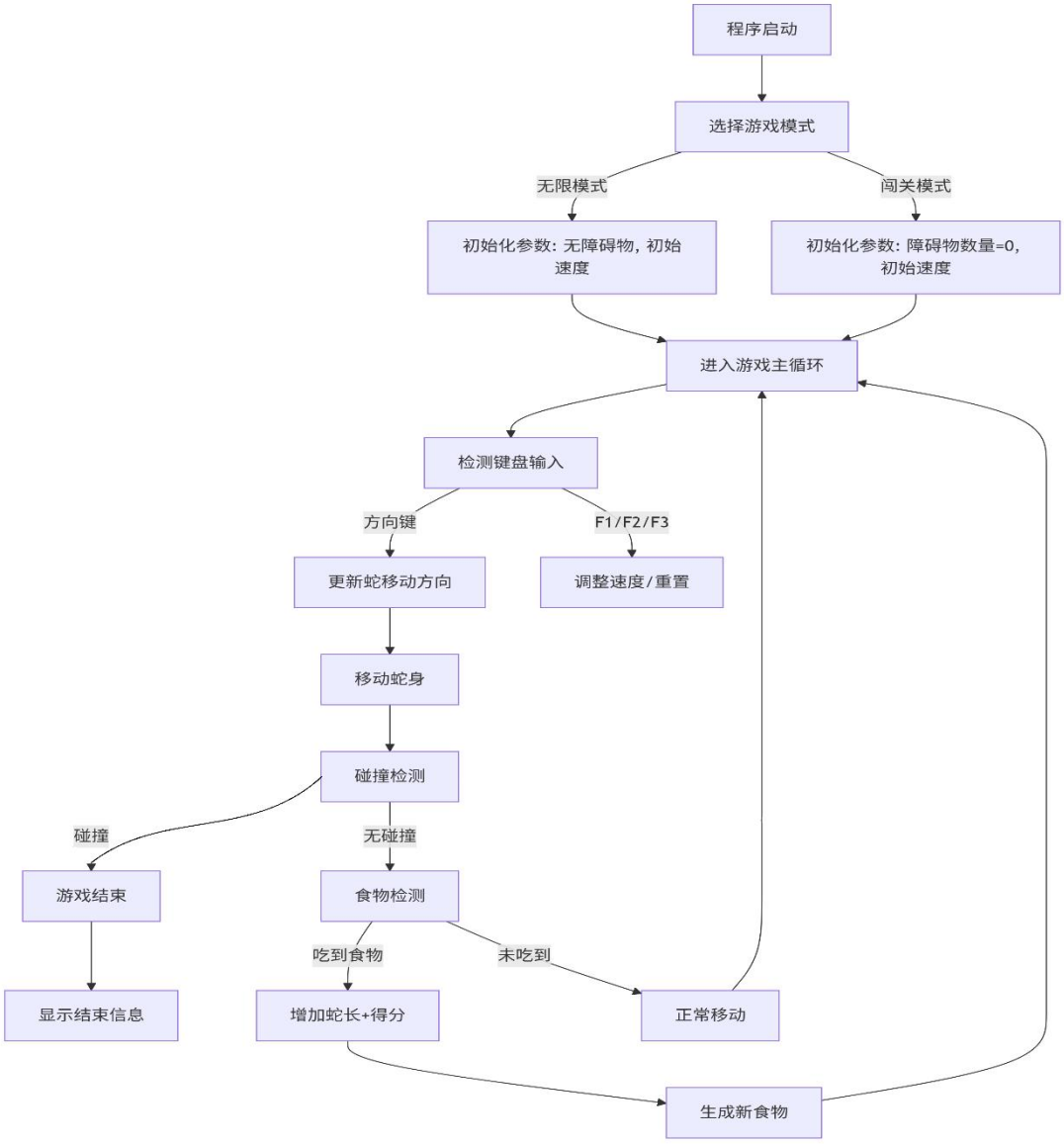


图 1：主函数逻辑流程图

- 程序启动后先选择游戏模式（无限模式/关卡模式）
- 根据模式初始化参数（障碍物数量、蛇速等）
- 主循环持续检测用户输入并更新游戏状态
- 碰撞检测和食物检测在每次移动后立即执行

2. 蛇移动逻辑

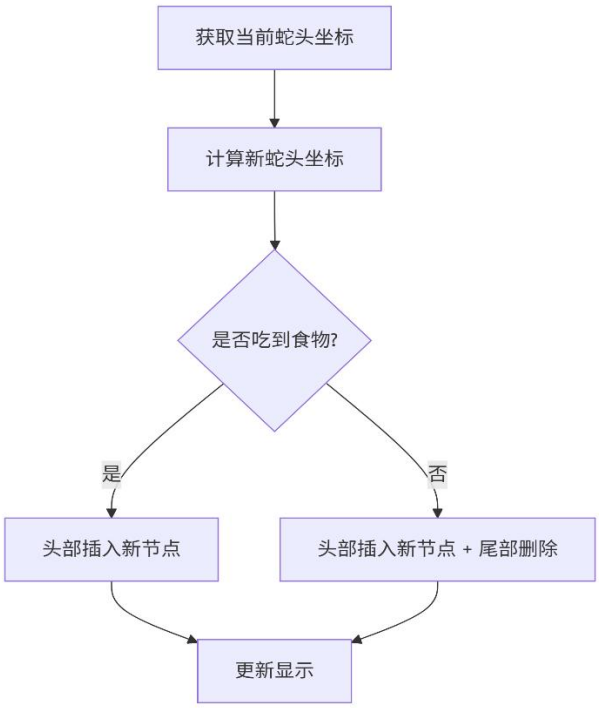


图 2：蛇移动逻辑流程图

- 若吃到食物，则使用链表头部插入实现快速增长
- 若没有吃到食物，则通过尾部删除维持蛇身长度恒定

3. 碰撞检测逻辑

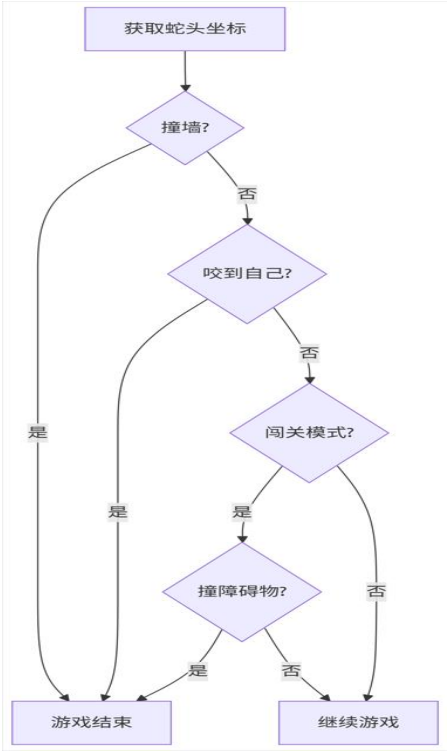


图 3：碰撞检测逻辑流程图

- 撞墙、自咬、撞障碍物依次检测，若发生则游戏结束
- 没有发生则游戏继续

4. 关卡升级逻辑

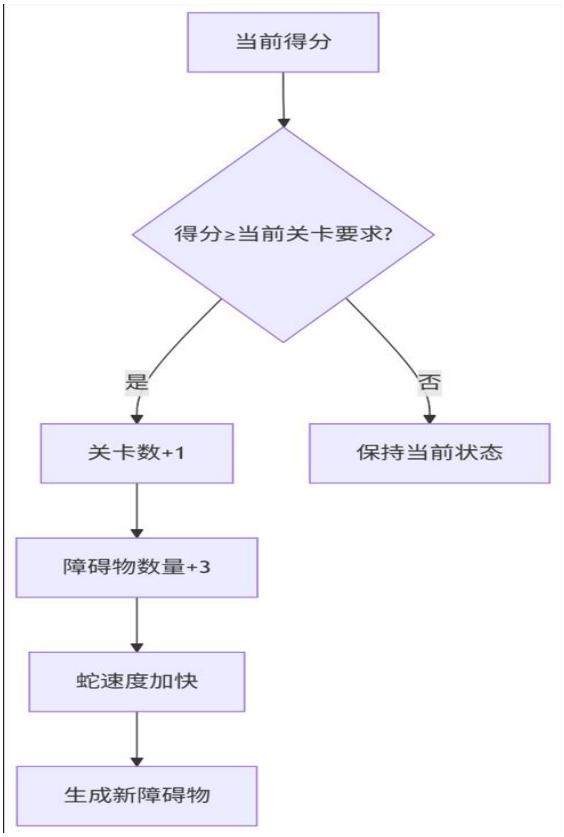


图 4：关卡升级逻辑流程图

- 检测得分是否超过当前关卡要求得分
- 若超过则关卡等级上升，出现更多障碍物且蛇移动速度加快
- 否则维持当前状态

5、核心代码展示

```
class SnakeNode {
private:
    int x;
    int y;
    SnakeNode* next;

    friend void moveSnake(int newX, int newY);
    friend int biteself();
    friend void createfood();
    friend void initsnake();
    friend void snakemove();
    friend void cantcrosswall();
    friend void createObstacles();
  friend int hitObstacle();
    friend void gamecircle();

public:
    SnakeNode(int px, int py, SnakeNode* pnext = nullptr)
        : x(px), y(py), next(pnext) {}
};
```

上述代码定义了蛇身节点类 SnakeNode，每个节点包含横纵坐标以及指向下一个节点的指针，为实现蛇的移动、碰撞检测等操作提供了基础。

```
        if (newX == food->x && newY == food->y) {
            head = nexthead;
            score += add;
            SetColor(COLOR_RED);
            Pos(head->x, head->y);
            printf("*");
            createfood();
        } else {
            head = nexthead;
            SetColor(COLOR_RED);
            Pos(head->x, head->y);
            printf("*");

            SnakeNode* current = head->next;
            while (current) {
                SetColor(COLOR_WHITE);
                Pos(current->x, current->y);
                printf("*");
                current = current->next;
            }

            SnakeNode* q = head;
            while (q->next->next) q = q->next;
            Pos(q->next->x, q->next->y);
            printf("  ");

            delete q->next;
            q->next = nullptr;
        }
```

这是控制贪吃蛇移动的代码，当按下方向键时，根据新的坐标创建新的蛇头节点，若蛇头移动到食物所在位置，则加分并重新生成食物；否则，正常移动蛇身，同时删除尾部节点，以实现蛇的匀速移动效果，并更新蛇在控制台的显示位置，控制台的蛇身颜色也做了相应区分。

```
void createfood() {
    while (1) {
        food = new SnakeNode((rand() % 27) * 2 + 2, rand() % 24 + 1);
        int valid = 1;
        SnakeNode* q = head;
        while (q) {
            if (q->x == food->x && q->y == food->y) {
                valid = 0;
                break;
            }
            q = q->next;
        }
        if (valid) break;
        delete food;
    }
    SetColor(COLOR_GREEN);
    Pos(food->x, food->y);
    printf("*");
}
```

该函数用于创建食物，通过随机函数生成食物的横纵坐标，并不断检查是否与蛇身重合，若重合则重新生成，确保食物不会出现在蛇身上。生成有效位置的食物后，在控制台显示为绿色的“□”号。

```
void createObstacles() {
    while (obs_head) {
        Obstacle* tmp = obs_head;
        obs_head = obs_head->next;
        delete tmp;
    }

    int obs_num = (level - 1) * 3;
    if (obs_num > 12) obs_num = 12;

    for (int i = 0; i < obs_num; i++) {
        Obstacle* new_obs = nullptr;
        do {
            int x = (rand() % 27) * 2 + 2;
            int y = rand() % 24 + 1;
            new_obs = new Obstacle(x, y);

            bool valid = true;
            SnakeNode* p = head;
            while (p) {
                if (p->x == x && p->y == y) valid = false;
                p = p->next;
            }
            if (food && food->x == x && food->y == y) valid = false;
            if (x <= 0 || x >= 56 || y <= 0 || y >= 26) valid = false;

            Obstacle* exist = obs head;

             while (exist) {
                 if (exist->x == x && exist->y == y) valid = false;
                 exist = exist->next;
             }

             if (!valid) {
                 delete new_obs;
                 new_obs = nullptr;
             }
         } while (!new_obs);

         new_obs->next = obs_head;
         obs_head = new_obs;
         SetColor(COLOR_BLUE);
         Pos(new_obs->x, new_obs->y);
         printf("□");
     }
 }
```

该函数用于创建障碍物，根据当前关卡计算障碍物数量，同样通过随机函数生成障碍物位置，并进行一系列有效性检查，确保障碍物不会与蛇、食物以及边界重合，创建成功后将障碍物节点添加到障碍物链表中，并在控制台显示为蓝色的“□”号，增加了游戏的挑战性。

```
void creatMap() {
    SetColor(COLOR_WHITE);
    for (int i = 0; i < 58; i += 2) {
        Pos(i, 0);
        printf("墙");
        Pos(i, 26);
        printf("墙");
    }
    for (int i = 1; i < 26; i++) {
        Pos(0, i);
        printf("墙");
        Pos(56, i);
        printf("墙");
    }
}
```

此函数用于绘制游戏地图的边界墙。通过循环在指定位置打印字符“墙”来构建游戏的边界。水平边界和垂直边界分别通过两个循环实现。使用 SetColor 和 Pos 函数设置字符颜色和位置，确保边界在控制台中清晰可见。

```
void initsnake() {
    SnakeNode* tail = new SnakeNode(28, 12);
    for (int i = 1; i <= 4; i++) {
        head = new SnakeNode(28 + 2 * i, 12, tail);
        tail = head;
    }

    SetColor(COLOR_RED);
    SnakeNode* current = tail;
    while (current) {
        Pos(current->x, current->y);
        printf("★");
        if (current != head) SetColor(COLOR_WHITE);
        current = current->next;
    }
}
```

初始化蛇身，创建一个由 4 个节点组成的蛇。蛇尾固定在初始位置 (28, 12)，蛇头向右延伸。使用循环创建蛇身节点，并通过链表连接。设置蛇头为红色，蛇身为白色，增强视觉区分。通过 Pos 和 printf 函数在控制台中绘制蛇的初始形态。

```
void gamecircle() {
    Pos(64, 15);
    SetColor(COLOR_YELLOW);
    printf("游戏提示信息：");
    Pos(64, 16);
    printf("游戏开始，按空格键暂停");
    Pos(64, 17);
    printf("ESC键退出");
    if (gameMode == 1) {
        Pos(64, 18);
        printf("F1加速 F2减速 F3恢复默认速度");
    }

    if (gameMode == 2) {
        createObstacles();
        sleeptime = 200;
    }

    status = R;
    while (1) {
        SetColor(COLOR_YELLOW);
        Pos(64, 9);
        printf("当前模式：");
        if (gameMode == 1) printf("无限模式");
        else printf("闯关模式");
        Pos(64, 10);
        printf("得分：%d ", score);

  if (gameMode == 2) {
      Pos(64, 11);
      printf("当前关卡：Lv.%d ", level);
      Pos(64, 12);
      printf("下一关所需得分：%d", level * 100);
  } else {
      Pos(64, 11);
      printf("当前食物分值：%d ", add);
  }

  if (GetAsyncKeyState(VK_UP) && status != D) status = U;
  else if (GetAsyncKeyState(VK_DOWN) && status != U) status = D;
  else if (GetAsyncKeyState(VK_LEFT) && status != R) status = L;
  else if (GetAsyncKeyState(VK_RIGHT) && status != L) status = R;
  else if (GetAsyncKeyState(VK_SPACE)) pause();
  else if (GetAsyncKeyState(VK_ESCAPE)) {
      endgamestatus = 3;
      endgame();
  }

  if (gameMode == 1) {
      if (GetAsyncKeyState(VK_F1)) {
          sleeptime = (sleeptime > 50) ? sleeptime - 30 : 50;
          add = (add < 20) ? add + 2 : 20;
      }
      else if (GetAsyncKeyState(VK_F2)) {
          sleeptime = (sleeptime < 350) ? sleeptime + 30 : 350;
          add = (add > 1) ? add - 2 : 1;
```

游戏主循环函数，负责处理游戏逻辑和用户输入。根据选择的游戏模式初始化游戏参数。实时检测用户输入以改变蛇的移动方向、暂停/继续游戏、调整游戏速度等。更新游戏状态，包括得分、关卡信息等。调用蛇移动函数更新游戏画面，并检测碰撞以决定游戏是否结束。

```
            }
        }

        Sleep(sleeptime);
        snakemove();

        if (gameMode == 2) {
            checkLevelUp();
            if (hitObstacle()) {
                endgamestatus = 4;
                endgame();
            }
        }
    }
}
```

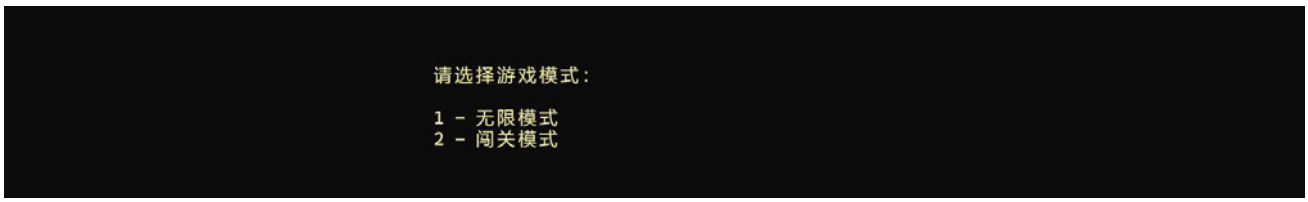
三、测试与结论

1、测试环境与数据

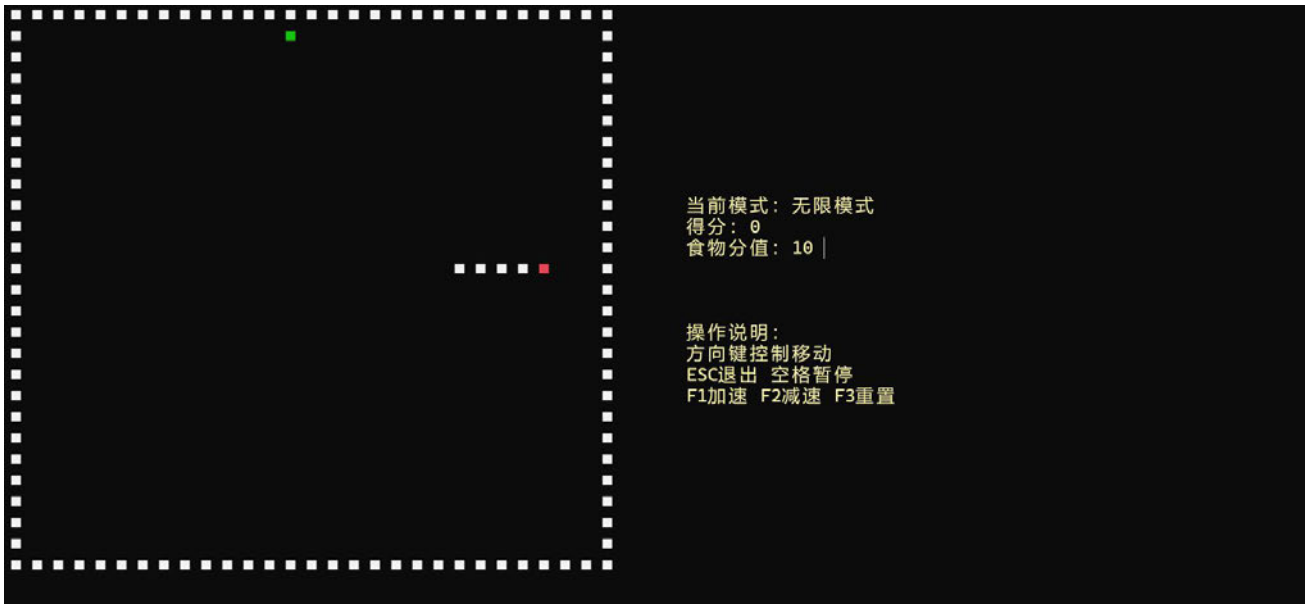
环境：Windows 11，Visual Studio 2022
数据：对游戏的各个环节进行模拟测试。

2、测试用例

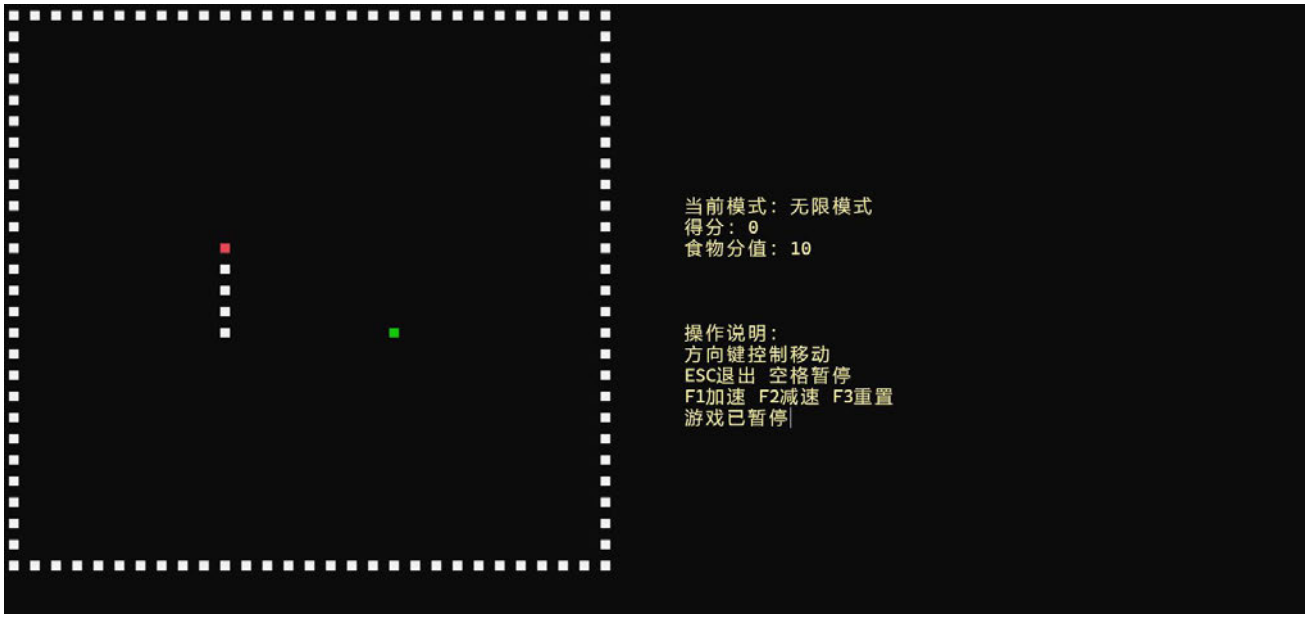
1. 开始游戏



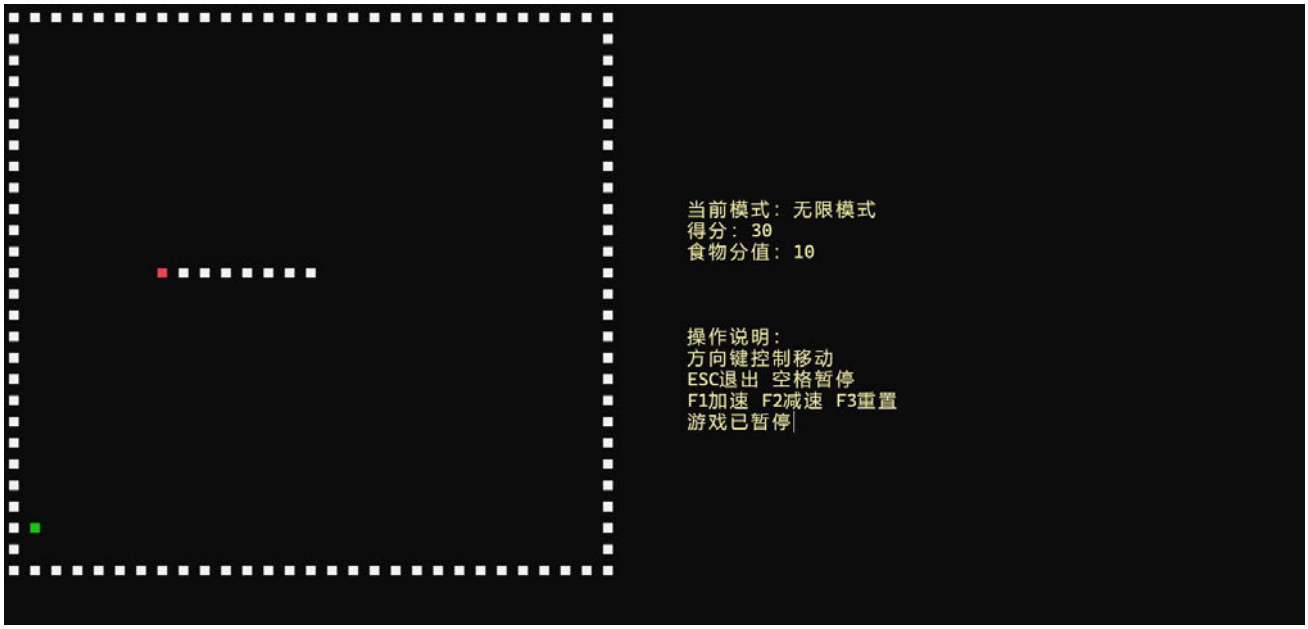
2. 选择无限模式的游戏界面



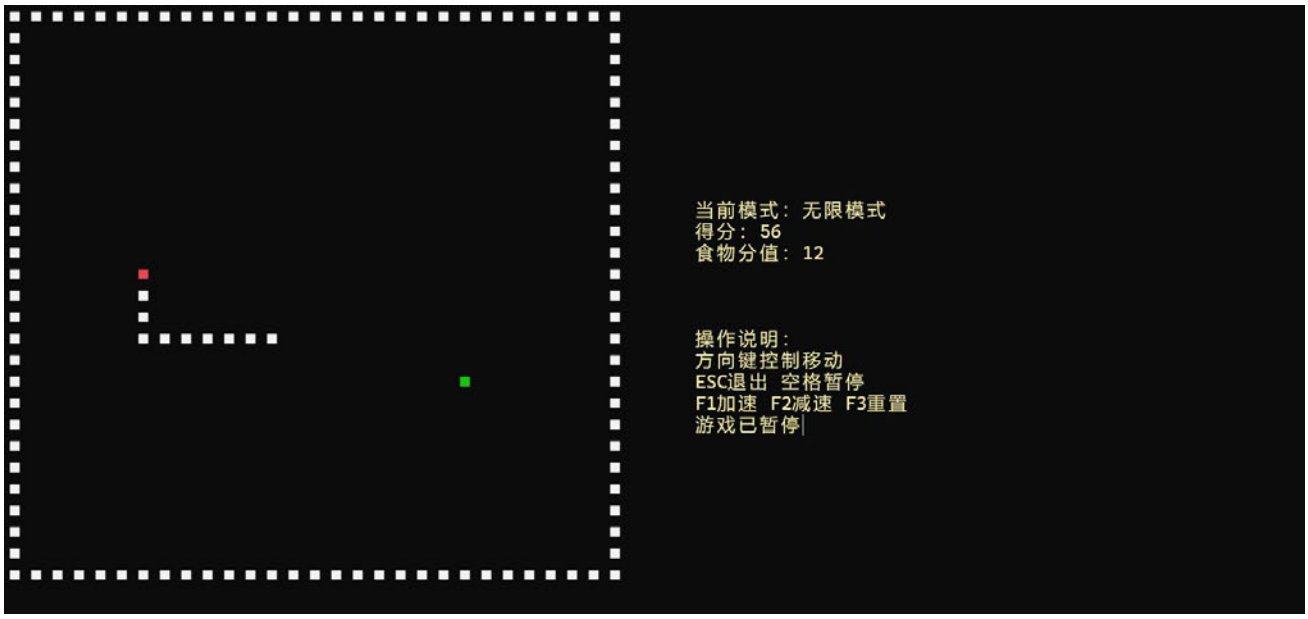
3. 测试中发现，贪吃蛇每隔固定的时间都会向前移动一个单位，通过键盘可以改变移动的方向以及游戏的退出、暂停与继续。



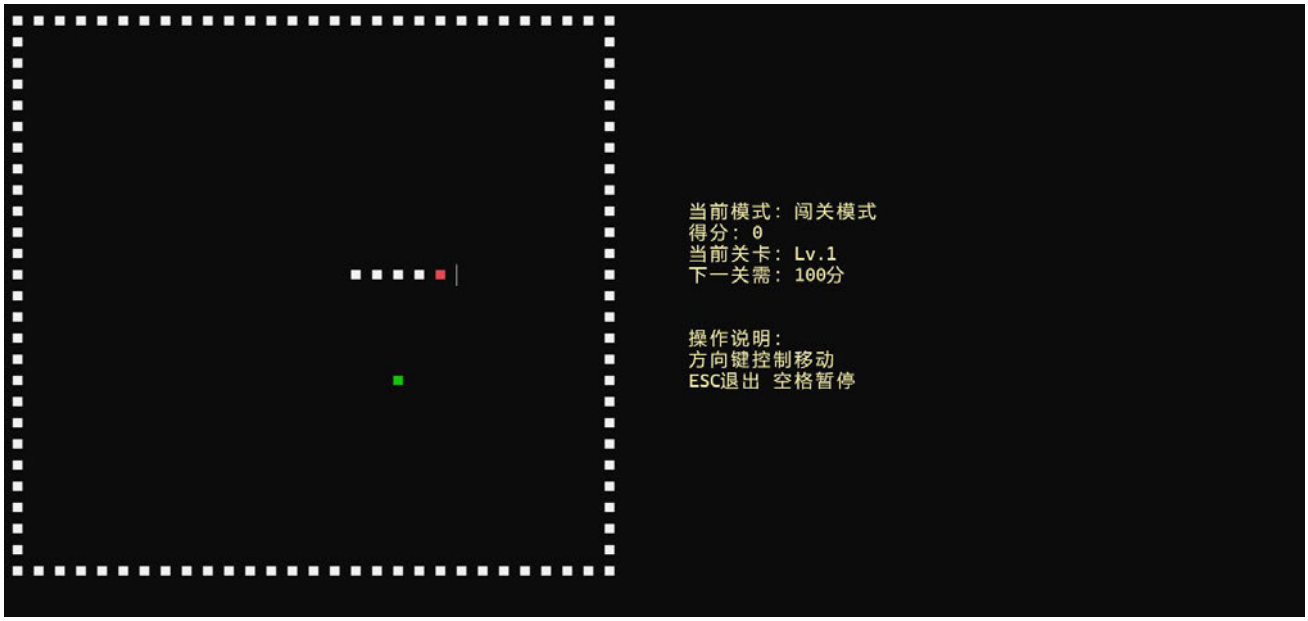
4. 贪吃蛇吃到食物后长度变长，并且游戏得分增加。



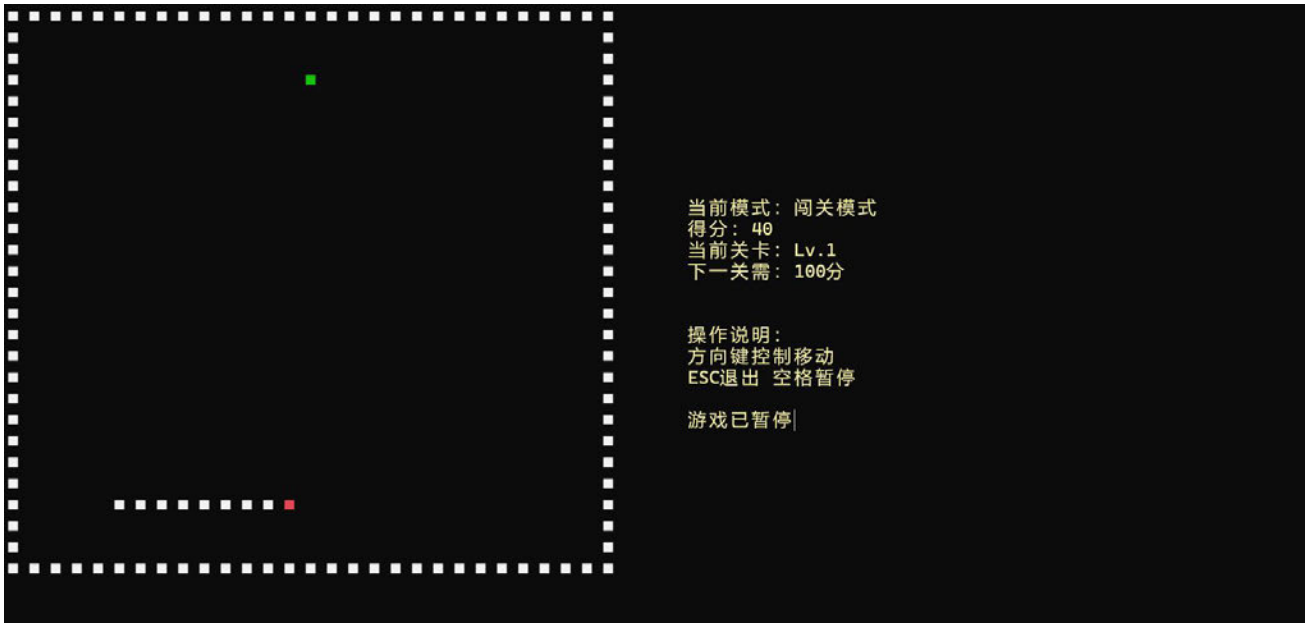
5. 在无限模式下快捷键 F1、F2、F3 可以控制蛇身移动速度并且改变食物分值，相应提示在控制台右侧。

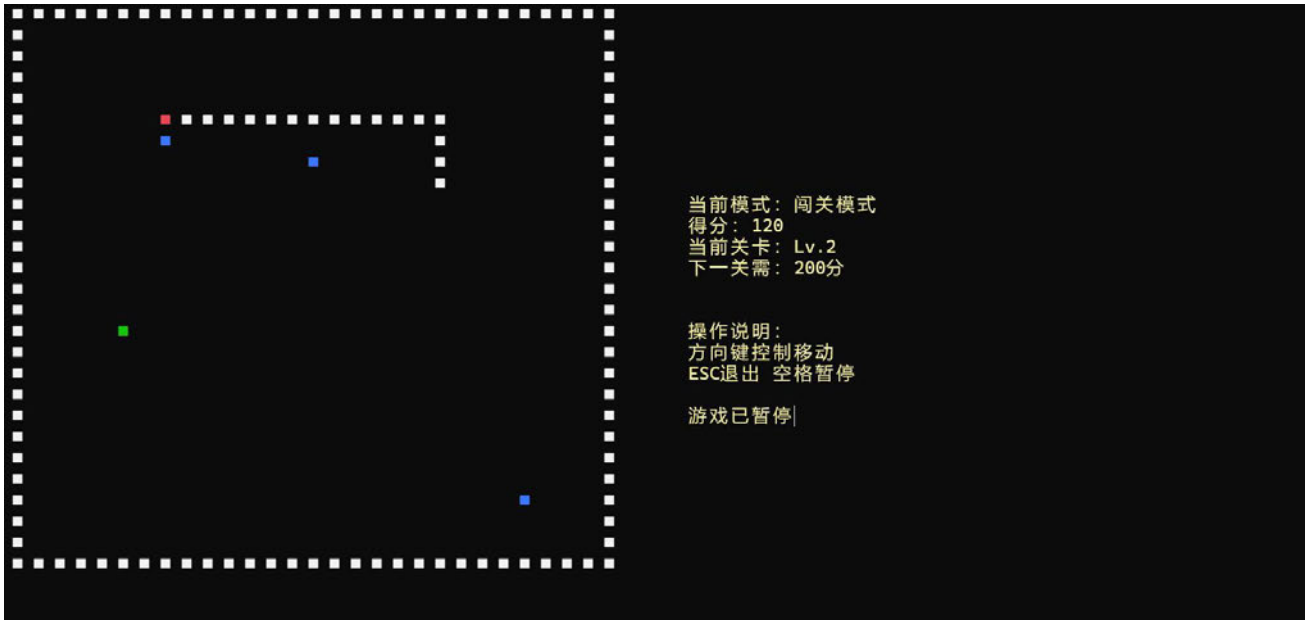


6. 选择闯关模式后的游戏界面，控制台右侧标明了关卡信息。



7. 闯关模式大部分操作与无线模式相同，但每过一关障碍物个数增加 3 个并且蛇身移动速度加快。





8. 游戏在贪吃蛇撞墙、撞上障碍物或者咬到自己时结束，控制台会打印相关提示信息。

3、测试结论

通过上述测试发现，程序基本满足贪吃蛇游戏的基本要求。在游戏中玩家可以选择闯关模式和无线模式，分别对应两种不同玩法。玩家通过键盘控制贪吃蛇的游动方向和游戏状态，能随机产生食物并且贪吃蛇吃到食物后蛇身长度和游戏分数增加。游戏具有完善的碰撞检测功能，以此判断退出游戏的时机。游戏还给玩家提供了充分的操作提示，进一步提高了玩家的体验感。

四、总结与思考

1、题目难点要点

链表的动态管理：在程序设计我们运用了链式队列和单向链表两种数据结构，这两种数据结构都涉及对指针的操作和管理，需要我们特别注意。
控制台交互界面的设计：通过添加” windows.h”头文件实现控制台的交互，从而对游戏界面进行设计。

2、本组工作特点

1. 游戏功能扩展：在游戏中我们设计了无限模式和闯关模式两种模式，给予玩家更多选择。
2. 玩家体验感增强：蛇头、蛇身、食物、障碍物标明不同颜色，区分各游戏元素。支持快捷键进行暂停、退出以及调整蛇身运动速度的操作，进一步提升玩家的体验感。
3. 严谨的内存管理：在贪吃蛇移动式动态增删节点，防止内存泄漏。在切换关卡时释放旧障碍物链表。
4. 蛇身移动优化：通过节点出队入队的方式进行移动，而非通过屏幕的不断刷新，可以避免闪屏问题。

3、本组改进方向

1. 添加文件的存取功能，有利于保存之前的游戏进度。
2. 引入特殊的食物和障碍物，增强游戏的趣味性。
3. 为游戏增加音效系统，提升玩家体验感。