第11章 综合项目-贪吃蛇

学习目标

- **熟悉项目开发流程**
- **理解项目需求，划分项目模块，学会设计数据结构及接口**
- **掌握项目实现流程，完成贪吃蛇游戏项目**

本书前 1-10 章对数据结构的基本知识及使用规则都进行了详细阐述，学习数据结构的目的是要将其应用到项目开发中来解决实际问题，并且在应用的过程中增强各种数据结构的认知理解和使用能力，锻炼编程思维能力。本章将带领大家以数据结构为基础，用 C 语言来实现一个有趣的游戏项目——贪吃蛇。

11.1 项目分析

相信读者都知道贪吃蛇这款游戏，它是一款经典的手机游戏。通过控制蛇头方向吃食物，使得蛇变长，从而获得积分，既简单又耐玩。通过上下左右键控制蛇的方向，寻找吃的东西，每吃一口就能得到一定的积分，而且蛇的身子会越吃越长，身子越长玩的难度就越大，不能碰墙，不能咬到自己的身体，更不能咬自己的尾巴，等到了一定的分数，就能过关，然后继续玩下一关。

将这些游戏功能划分成一个个模块，然后对各个模块再进行深入分析，划分成更小的模块，最后划分到具体的功能函数，将这些功能函数实现，再把模块整合，那么项目就初步实现了。

11.1.1 模块设计

本游戏使用 MVC（模型(Model)－视图(View)－控制器(Controller)）设计模式，按照这种设计模式将应用程序的输入、处理和输出分开，使应用程序被分成三个核心部分：模型、视图、控制器。每个部分各自处理自己的任务。这种设计模式可以映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面结构中。

在本游戏中，从键盘输入来获取移动方向和蛇的移动是异步的关系，即一个线程执行蛇的移动，相当于是数据显示层（View），一个线程执行玩家的输入，相当于是控制器（Controller）（关于线程，将在后文中讲解），两个过程是独立的，但又需要全局变量进行通讯，所需要的地图、食物等数据存储于 Model 层。

在本项目中，根据游戏需求分析， M 部分包括蛇、食物、随机数生成三个模块；V 部分为界面处理模块；C 部分为游戏控制模块。具体如图 11-1 所示。

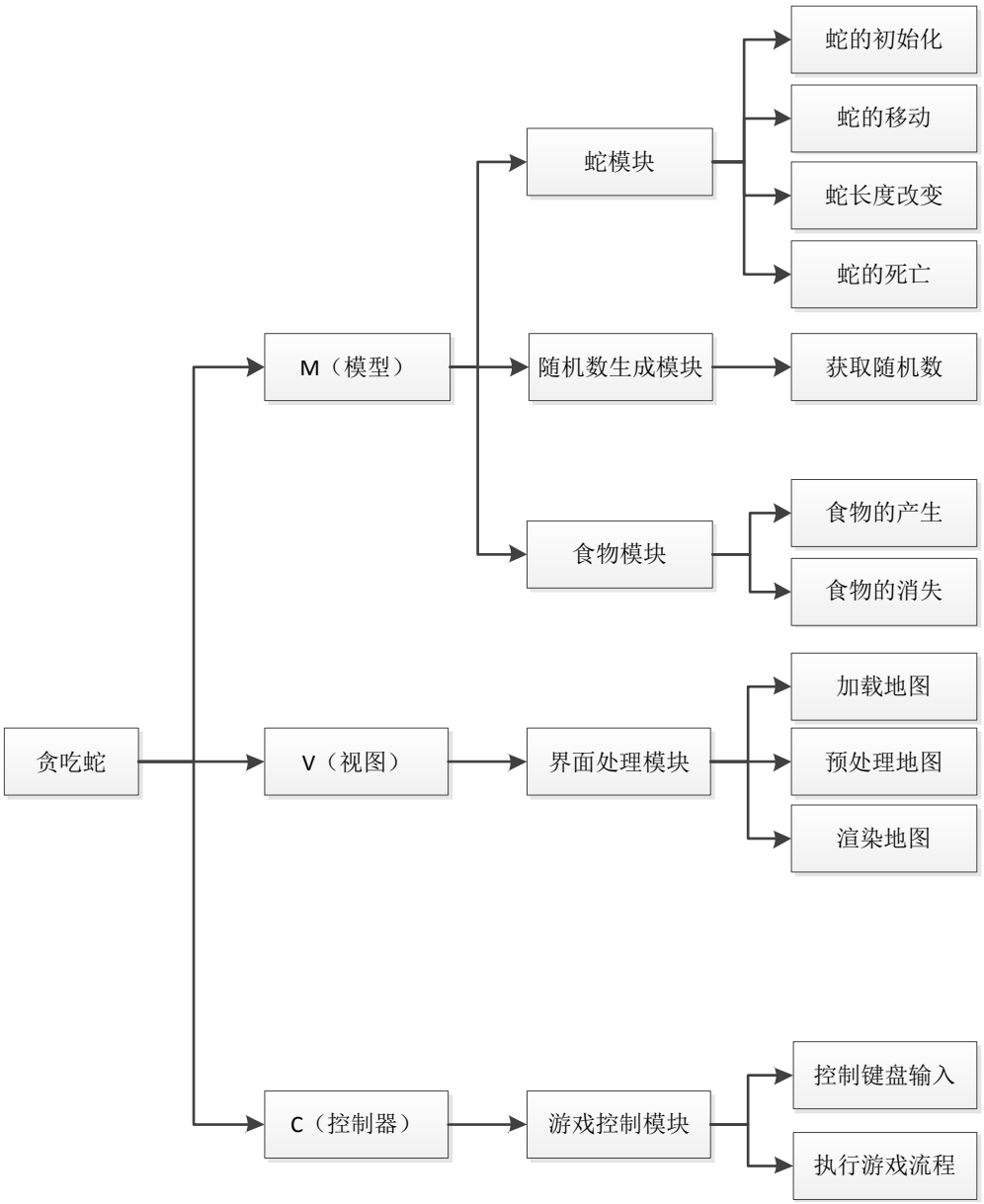

图11-1 贪吃蛇功能模块划分

根据图 11-1 的划分描述，下面介绍系统中各模块的主要功能：

1、蛇模块：蛇吃了食物后会增长，不是一个固定长度，选用链表更为合适，这里选用单链表来模拟蛇。因为蛇在移动时是将蛇尾结点移动，插入到蛇头结点前成为新的蛇头结点，在这个过程中需要用到链表的相关知识，比如查找、增加、删除等操作。

蛇这一模块主要完成关于蛇的一系列动作：蛇的创建，当蛇吃了不同的食物时，有对应的长度增加、长度减小和蛇的移动速度增加等反应，当蛇撞到障碍物、自身或者通关时就死亡。

2、食物模块：食物模块主要是控制食物随机出现的种类和位置，食物在出现时，不能出现在蛇的身体上、障碍物上；当蛇把食物吃掉后，食物要消失。

3、随机数生成模块：由于项目中需要用到一些随机数，比如随机选择食物的种类、随机摆放食物的位置等，所以设计了随机数生成模块。

4、界面处理模块：游戏运行的整个界面称之为地图，而界面处理模块主要就是实现地图的加载、预处理地图、和地图的显示等功能，直观形象地显示蛇、食物、障碍物以及实时得分等界面信息。

5、控制游戏流程模块：此模块就是程序入口模块，即 main()函数，在这个函数中创建了两个线程，一个线程获取键盘的输入；另一个线程实现游戏流程的不停循环，即当游戏运行状态正常时，不停调用蛇的移动、预处理地图和显示地图等函数；当游戏状态为进入下一关时，重新加载地图；当游戏状态为闯关失败时，则进入游戏的结束流程。该模块可以控制游戏数据模型和游戏界面的同步。

多学一招：多线程

在设计项目模块时，控制游戏流程模块用到了多线程，这里就讲解一下什么是多线程。学习多线程之前，要先学习两个概念：进程与线程。

所谓进程就是“正在运行的程序”，在操作系统中，每个独立执行的程序都可称为一个进程。例如我们电脑上正在运行的 QQ、微信、美图等等都是一个进程。在多操作任务系统中，进程是不可并发执行的，虽然我们可以一边听音乐一边聊天，但实际上这些进程并不是同时运行的。在计算机中，所有的应用程序都是由 CPU 执行的，对于一个 CPU 来说，在某个时间点只能运行一个程序，也就是说只能执行一个进程，只是 CPU 速度很快，在不同进程间的切换也很快，从表层看，用户会以为是同时执行多个程序。

在一个进程中可以有多个执行单元同时运行，这些执行单元可以看作程序执行的组成部分，被称为线程。当产生一个进程时就会默认创建一个线程，这个线程运行是 main()方法中的代码。

我们都知道，代码是从上向下执行的，不会出现两段程序代码交替运行的效果，这样的程序称为单线程程序。如果程序中开启多个线程交替运行程序代码，则需要创建多个线程，即多线程程序。进程与线程的关系如图 11-2 所示。

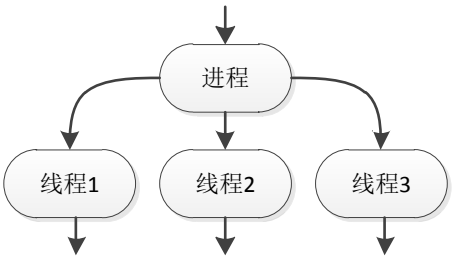

图11-2 多线程

多线程也是由 CPU 轮流执行的。CPU 好比餐馆中的厨师，线程好比服务员点菜上菜时的订单，而这整个流程可以看作是一个进程。若餐馆中只有一位服务员，当一位顾客点单时，好比一个进程只开启一个线程；若来了另一位顾客，就只能等第一位顾客点菜完成后才能点菜。这时若启用另一个服务员来为另一位顾客点菜，那么两个服务员就相当于两个线程。如果一位顾客点了两个菜，当厨师为第一位顾客做第菜时就是在执行线程 1，当第一位顾客的第一道菜做完，第一位顾客开始享用这道菜，而厨师并不急于完成第一位顾客的订单，而是开始为第二位顾客做第一道菜，就是在执行线程 2。

创建多线程后，原来从上而下执行代码的线程称为主线程，而去执行别的程序代码的线程称为子线程。在我们的项目中，main()函数中就开启了两个线程，主线程用来加载地图、随机生成食物、控制蛇的移动等，子线程用来获取键盘的输入来判断蛇的移动方向，整个流程中两个线程交替执行。

多线程在实际开发中应用非常广泛，因为这并非数据结构中的课程内容，所以这里只进行简单讲解，以帮助读者理解项目中所用到的多线程。

11.1.2 模块描述

确定系统功能后，将依据功能进行数据结构的设计。在该游戏中，用二维数组来构建地图坐标，用一个单向链表来构建一条蛇，通过插入结点来使蛇增长、删除节点使蛇变短。下面将介绍系统设计思路及功能的具体实现。

1、加载地图

游戏中的地图是通过二维数组标记实现的，二维数组的行列索引标记着地图的位置，元素值通过一个枚举来赋值，其值从 1 到 7 分别标识路径、蛇身、蛇头、正常食物、障碍物、致幻食物、有毒食物。在初始时，加载地图实际就是读取一个存放于文本的二维数组，该二维数组元素值只有 1 和 5，即路径和障碍物。

2、预处理地图

由蛇模块获取蛇头蛇身的坐标位置，由食物模块获取食物的坐标，然后根据坐标来改变对应二维数组位置的元素值（元素取值为 1-7，参考 1：加载地图）。

3、渲染地图

渲染就是打印二维数组，根据元素值的不同打印对应的符号。蛇身用“□”表示，蛇头用“○”表示，可以走的路用“ ”（空格）表示，障碍物用“╬”表示，正常食物用“▲”表示，有毒食物（减速变短）用“★”表示， 致幻食物（加速变长）用“◆”表示。这样就可以实现游戏画面的显示了。假如每隔 0.5 秒显示一次的话，就能看到运行时游戏场景的变化了。

4、蛇的初始化

蛇的初始化实际就是链表的初始化，该链表有四个结点，每个节点都是个结构体，里面包含该节点的坐标信息，它出现的初始位置是首行的前四个元素，即二维数组中的前四个元素。

5、蛇的移动

蛇的移动是通改变链表头尾结点的坐标位置来实现的，例如当蛇向右前进一个单位，则将尾结点插入到头结点前，同时改变蛇头、蛇身以及蛇尾的枚举值。这样整体来看来蛇就前进了一个单位，如图 11-3 所示。

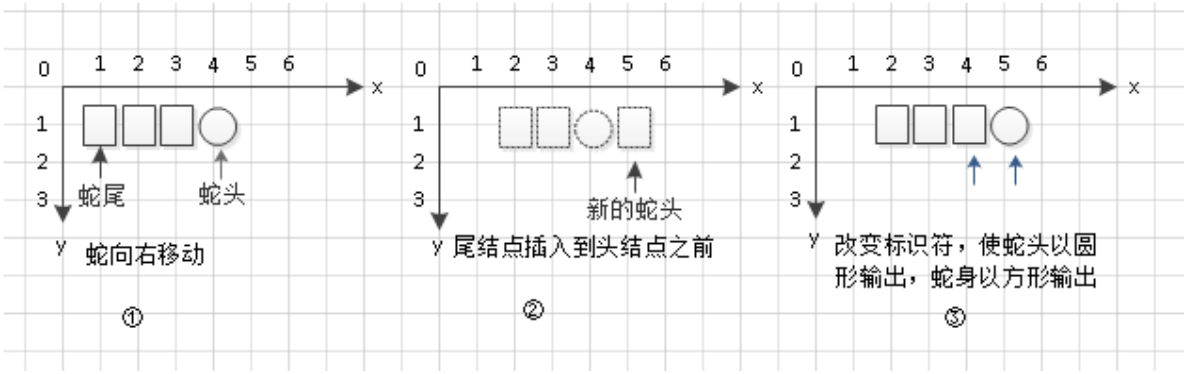

图11-3 蛇的前进

如图 11-2 所示，在坐标系（x,y）中，蛇向右移动一个单位，则将蛇尾结点移动插入到蛇头结点前，如图中②所示，原先的蛇尾结点坐标由（1,1）变为（5,1），然后将蛇头指针指向该节点，将蛇尾指针指向新的蛇尾，即（2,1）点。并且将原蛇头（4,1）位置对应的二维数组中的元素值修改为蛇身的枚举值（即 3），这样就可以保证打印时以“□”形式显示；将新蛇头（5,1）位置上对应的二维数组中的元素值修改为蛇头的枚举值（即 2），保证打印时能以“○”形式显示。

同理，当蛇向左、向上、向下移动时，也是将移动蛇尾结点到相应坐标位置。

注意：地图中的坐标系（x,y）对应二维数组 map[a][b]中的 a、b 时是相反的，如原来蛇头的位置为（4，1），而这个位置在二维数组中的位置为 map[1][4]。地图坐标系与二维数组下标的对应关系如下所示：

坐标（x,y）<=>二维数组[y][x]

之所以这样对应是由于二维数组的二重循环打印，有兴趣的同学可以自己分析下。

6、蛇的增长

当蛇吃了正常食物后，蛇的长度会增加，增加蛇的长度就是在食物的位置增加一个链表节点并且将这个链表结点变为蛇头，在这个过程中需要注意：把蛇头指针指向新的蛇头，蛇尾指针不用改变）。

7、蛇的减短

当蛇吃了有毒食物后，蛇的长度会变短，其实现过程为在食物位置产生一个链表结点并且将这个结点变为蛇头，同时把链表的尾部两个结点删除，这样增加一个结点删除两个结点就可以实现蛇的前行和长度减短的功能。在这个过程中同时也要注意蛇头指针和蛇尾指针的指向改变。

8、蛇移动速度的改变

在讨论蛇的移动速度改变之前我们先思考，如何实现蛇的移动？蛇的移动是通过改变蛇在地图中的坐标位置并且刷新界面完成的。在这个过程中刷新速率就是蛇的移动速度，这里通过睡眠函数（sleep()）来控制刷新速率的，通过设置在睡眠函数中设置睡眠时间就能影响蛇的移动速度，读者可以去代码中查看该部分代码，理解把握。

9、蛇的死亡

当蛇撞上障碍物、自身或者通关时，蛇会死亡，蛇死亡就是链表的销毁。

10、食物的产生

食物的产生都是随机的，包括食物出现的位置和食物的种类，这些因素由通过随机函数获取的随机数决定。食物的位置不能出现在障碍物和边界上。食物有三种：正常食物（蛇吃了增长）、有毒食物（蛇吃了长度减小）、致幻食物（吃了会兴奋加速）。

11、食物的消失

蛇吃了食物之后，地图中就不能再显示食物了，本质就是该食物位置对应的二维数组元素值变为路径的枚举值（值为 1）。

12、获取随机数

随机数由随机数种子产生，用于控制食物出现的位置和种类。

13、控制键盘输入（子线程）

子线程通过获取键盘输入的 W/w(上)、 S/s(下)、 A/a(左)、 D/d(右)来改变蛇模块中移动方向的枚举值，从而影响蛇的移动方向。

14、执行游戏流程（主线程）

根据蛇的存活状态来判断接下来的操作（进行进入下一关、失败、继续游戏），当继续本关游戏时，按照获取的键盘输入方向让蛇移动，进行界面的预加载和渲染。关于这一部分将在下一节详细讲解。

11.1.3 项目分析

前两个小节对项目进行了模块划分与分析，当完成各模块之后需要把模块整合起来，使其协同工作，形成一个完整的、可以运行的项目。下面用一张流程图来表示各个模块，演示项目的运行流程，如图 11-4 所示

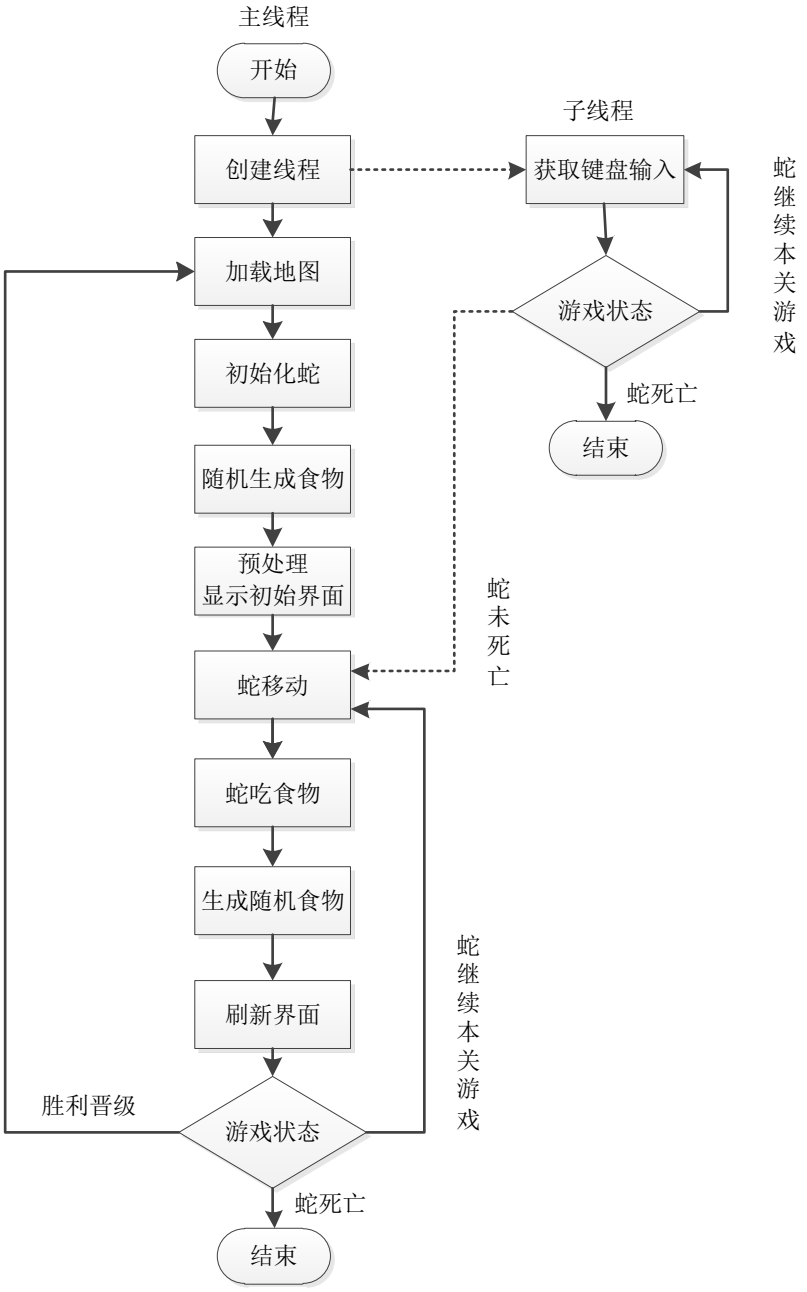

图11-4 贪吃蛇运行流程图

在这张流程图中，程序运行之初创建了一个子线程，这个子线程负责获取键盘的输入，当游戏继续时，就一直读取键盘输入控制蛇的移动方向，当游戏结束时，子线程退出。

主线程在创建了子线程后就加载地图并初始化蛇，然后产生随机数随机生成食物，并预处理地图，渲染地图显示初始游戏界面。

游戏界面显示之后，会根据子线程读取的移动方向来控制蛇的移动，蛇移动之后会吃食物，食物被吃掉之后又会随机生成，然后刷新界面（预处理、渲染地图），将界面显示出来。

刷新界面时会判断蛇的状态，如果蛇死亡，则结束游戏，即主线程和子线程都结束；如果蛇胜利晋级则进入下一关，重新加载地图，显示新的一关的游戏初始界面；如果蛇没有死亡也没有晋级过关，继续本关游戏，则按照子线程读取的键盘输入控制蛇的移动。

11.2 项目实现

11.2.1 创建项目

根据 11.1 节的项目分析，需要创建界面处理模块、蛇模块、随机数生成模块、食物模块及全局变量对应的头文件及其实现文件。各个文件及功能如下所示：

- map.h 头文件、map.c 源文件：map.h 头文件中定义了枚举 Emap 用来标识地图中的路径、蛇、食物、障碍物等；定义了枚举 EgameStatus 来标识游戏状态。并且定义了 PreviewMap()、LoadMap()和 DisplaMap()三个函数，分别用来预处理地图、加载地图、显示地图。map.c 源文件用于实现 map.h 中的函数。
- snack.h 头文件、snack.c 源文件：snack.h 头文件中定义了枚举 Edirection 来标识蛇的移动方向；定义了枚举 EsnackeStutas 标识蛇吃了食物后的状态；蛇用双向链表来表示，则定义 struct Snack 表示链表结点。此外，还定义了 SnackInit()、SnackMove()、SnackInsert()、SnackDestroy()、SnackNormalFn()、SnackShorten()、SnackAccelerate()函数，分别用于蛇的生成、移动、增长等一系列动作。snack.c 源文件用于实现 snack.h 头文件中的函数。
- random.h 头文件、random.c 源文件：random.h 头文件定义了 InitRandomSystem()、GetRandomNumber()两个函数，分别用于初始化随机数系统、产生随机数。random.c 源文件用于实现这两个函数。
- food.h 头文件、food.c 源文件：food.h 头文件定义了 struct Food 用于标识食物的种类与位置。并且定义了 FoodCreate()、FoodRelease()函数，分别用于生成、释放食物。food.c 源文件用于实现 food.h 头文件中的函数。
- global.h 头文件：定义了整个项目中所需要的全局变量。

在整个项目中，随机数的生成决定着食物的生成种类与位置；食物的生成消失与蛇的一系列活动都要在地图中完成；随机数、蛇、食物、地图这几个模块都要用到 global.h 头文件中定义的全局变量。

11.2.2 项目设计

由 11.2.1 节的分析得出，本项目一共需定义 5 个头文件，下面我们就详细分析每个头文件的定义。

1、map.h 头文件

map.h 头文件定义地图加载时的相关信息，代码如下所示：

```
#pragma once


enum EMap //标识地图中的路径、蛇头、蛇身、食物、障碍物
{
    MAP_ROAD = 1,
```

```
    MAP_BODY,
    MAP_HEAD,
    MAP_FOOD_NORMAL,
    MAP_OBSTACLE,
    MAP_FOOD_ACCELERATE,
    MAP_FOOD_SHORTEN
};

enum EGameStatus //游戏状态
{
    GAME_LOOP,
    GAME_VICTORY,
    GAME_FAILURE
};

int width, height;                    // 地图的长和宽
volatile enum EGameStatus status;     // 游戏状态
int map[MAX_LENGTH][MAX_LENGTH];      // 地图最大容量为 32 * 32
int mapTemp[MAX_LENGTH][MAX_LENGTH]; // 地图最大容量为 32 * 32
int selectNum;                        // 所选关卡

// 地图预处理
void PreviewMap();
// 载入地图
void LoadMap(int scene);
// 地图显示（多线程）
void DisplayMap();
```

枚举 EMap 用来标识地图中的路径、蛇、食物、障碍物等，用于渲染地图。

枚举 EGameStatus 用于标识游戏状态：GAME_LOOP 表示继续本关游戏；GAME_VICTORY 表示胜利过关，进入下一关游戏；GAME_FAILURE 表示失败。

后面接着定义了地图处理的相关函数。

2、snack.h 头文件

snack.h 头文件定义了蛇的状态标识及一系列动作，代码如下所示：

```
//蛇
#pragma once
enum EDirection
{
    SNACK_UP,       // W
    SNACK_LEFT,     // A
    SNACK_DOWN,    // S
    SNACK_RIGHT    // D
};
// type declaration
typedef struct Snack
```

```
{
    int x;
    int y;
    struct Snack *pNext;
}Snack;

enum ESnakeStutas//蛇吃了食物后的状态
{
    SnakeNormal = 1,
    SnakeShorten,
    SnakeAccelerate
};
int snackLength;                            // 蛇的长度
Snack *pHeader, *pTail;                     // 蛇头、蛇尾
enum EDirection direction;                  // 蛇的移动方向
enum ESnakeStutas SnakeStutas;              //蛇吃了食物后的状态

// 初始化蛇
Snack* SnackInit();
// 链表蛇移动（多线程）
void SnackMove();
// 头插法链表增长
Snack* SnackInsert();
// 某个具体关卡中的游戏结束
void SnackDestroy();

void  SnackNormalFn();                      //吃了正常食物
Snack*  SnackShorten();                     //吃了有毒食物，蛇缩短
void SnackAccelerate();                     //吃了致幻食物，蛇加速移动
```

枚举 EDirection 标识蛇上下左右的移动方向；枚举 EsnakeStutas 标识蛇吃了食物后的状态：SnakeNormal 表示蛇吃了食物状态正常增长；SnakeShorten 表示蛇吃了食物长度减短；SnakeAccelerate 表示蛇吃了食物加速。struct Snack 定义了蛇结点：x,y 标识蛇出现的坐标位置。这些枚举、结构之后定义了一系列蛇动作的相关函数。

3、random.h 头文件

random.h 头文件的定义如下所示：

```
#pragma once

// 初始化随机数系统
void InitRandomSystem();

// 产生 [leftVal, rightVal] 之间的随机整数
int GetRandomNumber(int leftVal, int rightVal);
```

4、food.h 头文件

food.h 头文件用于定义关于食物的信息，代码如下所示：

```
#pragma once

// type declaration
typedef struct Food
{
    int x;
    int y;
    enum EMap foodKind;
}Food;

Food food;      // 食物

// 创建食物
Food FoodCreate();
// 食物被吃掉
void FoodRelease();
```

struct Food 中的 x,y 标识食物出现的位置，enum EMap 则是在 map.h 中定义的枚举。

5、global.h 头文件

global.h 头文件中定义了整个项目中的全局变量，代码如下所示：

```
#pragma once

// Boolean declaration
#define TRUE 1
#define FALSE 0
#define OTHER -1

// Macro value declaration
#define INIT_SNACK_LEN      4       // 初始蛇的长度
#define VICTORE_SNACK_LEN   10      // 过关时蛇的长度
#define MAX_LENGTH          32      // 路径字符串数组最大长度
#define CONSOLE_MAX_WIDTH   80     // 控制台最大宽度
#define DELAY_TIME          500     // 延时时间
#define DELAY_TIME_ACCELERATE   100 // 吃了致幻食物后的延时时间
#define DELAY_TIME_SHORTEN 1000    // 吃了有毒食物后延时时间

// Basic type declaration
typedef int BOOL;
```

11.2.3 项目实现

头文件中定义了项目需要的功能，这些功能在各自对应的源文件中实现，下面讲解各源文件中功能函数的实现。

1、map.c 源文件

map.c 源文件中需要实现三个功能函数：PreviewMap()、LoadMap()、DisplayMap()函数。文件代码如下所示：

```
#define _CRT_SECURE_NO_WARNINGS
#include "global.h"
#include "snack.h"
#include "food.h"
#include "map.h"
#include <stdio.h>
#include <stdlib.h>

static FILE *fp = NULL;  //定义文件指针
volatile enum EGameStatus status = GAME_LOOP;
//加载地图
void LoadMap(int scene)
{
    int i, j;
    char str[MAX_LENGTH] = "";

    sprintf(str, "Map\\%d.map", scene);
    fp = fopen(str, "r");
    fscanf(fp, "%d%d", &width, &height);

    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++)
        {
            fscanf(fp, "%d", &map[j][i]);
            mapTemp[j][i] = map[j][i];
        }
    }
    fclose(fp);
    fp = NULL;
}

//预处理地图
void PreviewMap()
{
    int i, j;
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++)
        {
```

```
41              map[i][j] = mapTemp[i][j];
42          }
43      }
44
45      Snack *pSnack = pHeader;
46      map[pSnack->x][pSnack->y] = MAP_HEAD;
47      pSnack = pSnack->pNext;
48      while (pSnack)
49      {
50          map[pSnack->x][pSnack->y] = MAP_BODY;
51          pSnack = pSnack->pNext;
52      }
53      map[food.x][food.y] = food.foodKind;
54  }
55
56      //显示地图
57  void DisplayMap()
58  {
59      int i, j;
60
61      system("cls");
62      printf("               传智案例\n\n");
63      for (i = 0; i < height; i++)
64      {
65          for (j = 0; j < width; j++)
66          {
67              if (i==0||j==0)
68              {
69                  printf("回");
70                  continue;
71              }
72              switch (map[j][i])
73              {
74              case MAP_ROAD:
75                  printf("  ");
76                  break;
77              case MAP_BODY:
78                  printf("□");
79                  break;
80              case MAP_HEAD:
81                  printf("o");
82                  break;
83              case MAP_FOOD_NORMAL:
84                  printf("▲");
```

```
85                  break;
86
87              case MAP_FOOD_ACCELERATE:
88                  printf("◆");
89                  break;
90              case MAP_FOOD_SHORTEN:
91                  printf("★");
92                  break;
93              case MAP_OBSTACLE:
94                  printf("╬");
95                  break;
96              }
97          }
98          printf("回");
99
100             switch (i)
101             {
102             case 3:
103                 printf("\t 您的得分是：%d", snackLength);
104                 break;
105
106
107             case 7:
108                 printf("\t 本关过关蛇长需要 10");
109                 break;
110             case 8:
111                 printf("\t 目前蛇长度是：%d", snackLength);
112                 break;
113             case 9:
114                 printf("\t 距过关还需的长度：%d", 10 - snackLength);
115                 break;
116
117             case 13:
118                 switch (food.foodKind)
119                 {
120                 case MAP_FOOD_NORMAL:
121                     printf("\t 出现的食物为：健康食物！");
122                     break;
123                 case MAP_FOOD_ACCELERATE:
124                     printf("\t 出现的食物为：有毒食物！");
125                     break;
126                 case MAP_FOOD_SHORTEN:
127                     printf("\t 出现的食物为：致幻食物！");
128                     break;
```

```
129                 default:
130                     break;
131                 }
132                 break;
133 
134             case 14:
135                 switch (food.foodKind)
136                 {
137                 case MAP_FOOD_NORMAL:
138                     printf("  效果：吃后增长 1，健康快乐成长！");
139                     break;
140                 case MAP_FOOD_ACCELERATE:
141                     printf("  效果：吃后增长 1，但是蛇速变快！");
142                     break;
143                 case MAP_FOOD_SHORTEN:
144                     printf("  效果：吃后减短 1，并且蛇速变慢！");
145                     break;
146                 default:
147                     break;
148                 }
149                 break;
150 
151             default:
152                 break;
153             }
154 
155             printf("\n");
156         }
157         for (j = 0; j <= width; j++)
158             printf("回");
159         printf("\n");
160     }
```

下面对三个函数的实现稍作讲解：

（1）LoadMap()函数

此函数用于加载地图，将存储于文本文件中的二维数组加载到内存，代码 15 行创建一个字符数组，16-17 行代码将存储于外存的某一个文本文件读入到此字符数组中；代码 18-19 行读取地图的宽度和长度，即二维数组的大小。代码 21-28 行，用双重 for 循环将文本中的二维数组读入到内存中的二维数组。

（2）PreviewMap()函数

此函数用于对二维数组进行赋值处理，代码 37-43 行用双层 for 循环对二维数组中元素进行赋值。代码 45-46 行创建了一个链表结点，并用 MAP_HEAD 赋值，在渲染时显示为蛇头；代码 48-52 行用 while 循环创建蛇身结点；代码 53 行设置食物的位置和种类。

（3）DisplayMap()函数

此函数用于渲染地图，即将预处理过的地图打印。代码 63-156 行控制地图的高度范围；在这个高度范围内，代码 65-97 行控制地图的宽度范围；在这个宽度范围内，代码 67-71 行打印地图边界；代码 72-96 行判断某个位置是路径、蛇头、蛇身、食物或障碍物，然后将其打印。代码 100-156 行打印游戏状态信息：游戏得分、蛇的长度、食物种类、蛇吃掉食物后的反应等。

2、snack.c 源文件

snack.c 源文件主要实现蛇的初始化、蛇的移动以及蛇吃了食物后的反应等一系列函数，文件代码如下所示：

```
#include <malloc.h>
#include <windows.h>
#include "global.h"
#include "snack.h"
#include "food.h"
#include "map.h"
//蛇的初始化
Snack* SnackInit()
{
    direction = SNACK_RIGHT;
    snackLength = INIT_SNACK_LEN;
    int cnt = INIT_SNACK_LEN ;//计算蛇身长度 for 循环的边界
    SnakeStutas = SnakeNormal;
    //蛇头创建
    pHeader = pTail = (Snack *)malloc(sizeof(Snack));
    pTail->x = cnt;
    pTail->y = 1;

    //蛇身创建
    while (cnt-- > 1)
    {
        pTail->pNext = (Snack *)malloc(sizeof(Snack));
        pTail = pTail->pNext;
        pTail->x = cnt;
        pTail->y = 1;
    }
    pTail->pNext = NULL;
    return pHeader;
}


void SnackMove()
{
    BOOL result;
    Snack *pSnack = NULL;
    int x = pHeader->x, y = pHeader->y, newX = x, newY = y;
```

```
37      switch (direction)
38      {
39      case SNACK_UP:
40          result = SnackJudge(x, y - 1);
41          if (result == TRUE)
42              newY = y - 1;
43          break;
44      case SNACK_LEFT:
45          result = SnackJudge(x - 1, y);
46          if (result == TRUE)
47              newX = x - 1;
48          break;
49      case SNACK_DOWN:
50          result = SnackJudge(x, y + 1);
51          if (result == TRUE)
52              newY = y + 1;
53          break;
54      case SNACK_RIGHT:
55          result = SnackJudge(x + 1, y);
56          if (result == TRUE)
57              newX = x + 1;
58          break;
59      }
60      if (result == TRUE)
61      {
62          Snack *pTemp = pHeader;
63          pTail->pNext = pHeader;
64          map[pTail->x][pTail->y] = MAP_ROAD;
65          while (pTemp->pNext != pTail)
66              pTemp = pTemp->pNext;
67          pTemp->pNext = NULL;
68          pHeader = pTail;
69          pHeader->x = newX;
70          pHeader->y = newY;
71          map[pTail->x][pTail->y] = MAP_ROAD;
72          pTail = pTemp;
73      }
74      else if (result == FALSE)
75          status = GAME_FAILURE;
76  }
77
78  // 链表蛇事件判定
79  static BOOL SnackJudge(int x, int y)
80  {
```

```
81      Snack *pSnack = pHeader;
82      if (x >= width || x <= 0 || y >= height || y <= 0)
83          return FALSE;
84      switch (map[x][y])
85      {
86          case MAP_ROAD:
87              return TRUE;
88          case MAP_BODY:
89              while (pSnack && (pSnack->x != x || pSnack->y != y))
90                  pSnack = pSnack->pNext;
91              if (!pSnack || pSnack == pTail)
92                  return TRUE;
93              return FALSE;
94          case MAP_FOOD_NORMAL:
95              SnackNormalFn();
96              FoodRelease();
97              return OTHER;
98              break;
99          case MAP_FOOD_SHORTEN:
100                 SnackShorten();
101                 FoodRelease();
102                 return OTHER;
103                 break;
104             case MAP_FOOD_ACCELERATE:
105                 SnackAccelerate();
106                 FoodRelease();
107                 return OTHER;
108                 break;
109             case MAP_OBSTACLE:
110             default:
111                 return FALSE;
112         }
113     }
114
115     // 头插法链表增长
116     Snack* SnackInsert()
117     {
118         Snack *pSnack = (Snack *)malloc(sizeof(Snack));
119         pSnack->x = food.x;
120         pSnack->y = food.y;
121         pSnack->pNext = pHeader;
122         pHeader = pSnack;
123         if (++snackLength == VICTORE_SNACK_LEN)
124             status = GAME_VICTORY;
```

```
125
126         return pHeader;
127     }
128
129     void  SnackNormalFn()
130     {
131         SnackInsert();
132         SnakeStutas = SnakeNormal;
133     }
134
135     Snack * SnackShorten()
136     {
137         if (snackLength <= 2)
138         {
139             status = GAME_FAILURE;
140             return pHeader;
141         }
142         Snack *pTempHeader = pHeader;
143         Snack *pTempTail = pTail;
144         Snack * pTempBehTail = pTail;
145
146         while ((pTempHeader->pNext)->pNext != pTempTail)
147         {
148             pTempHeader = pTempHeader->pNext;
149         }
150         pTail = pTempHeader;
151         pTempBehTail = pTempHeader->pNext;
152         pTail->pNext = NULL;
153
154         pTempBehTail->x = food.x;
155         pTempBehTail->y = food.y;
156         pTempBehTail->pNext = pHeader;
157         pHeader = pTempBehTail;
158
159         free(pTempTail);
160         snackLength--;
161
162         SnakeStutas = SnakeShorten;
163         return pHeader;
164     }
165
166     void SnackAccelerate()
167     {
168         SnackInsert();
```

```
169         SnakeStutas = SnakeAccelerate;
170     }
171
172     // 蛇的死亡销毁
173     void SnackDestroy()
174     {
175         Snack *pSnack = pHeader;
176         while (pHeader)
177         {
178             pHeader = pHeader->pNext;
179             free(pSnack);
180             pSnack = NULL;
181         }
182         pTail = NULL;
183     }
```

下面对 snack.c 源文件中实现的几个函数进行讲解：

（1）SnackInit()函数

代码 8-29 行实现了 SnackInit()函数，此函数用于蛇的初始化，即创建一条蛇；

代码 11 行定义蛇的初始长度为 4（INIT_SNACK_LEN 是在 global.h 中定义的宏）；

代码 13 行定义蛇的初始状态为正常（SnakeNormal 为枚举 EsnakeStutas 中定义的变量）；

代码 15-17 行创建蛇头，在地图坐标系中的位置为（4，1）；

代码 20-26 行用 while()循环创建蛇身，蛇身横坐标逐减，纵坐标不变；

代码 27-28 行使蛇尾指针指向空，并返回蛇头指针。

（2）SnackMove()函数

代码 32-76 行实现了 SnackMove()函数，此函数用于实现蛇的移动。蛇的移动是通过改变蛇头或蛇身的坐标来实现的，这一点在 11.1.2 节中也有讲解，此处不针对原理进行赘述，只进行代码分析。

代码 37-59 行是确实蛇的移动方向，获取新的坐标；

代码 60-73 行判断新的坐标处是否是可以通行的路径，如果是，则将蛇尾结点移动到新坐标处；

代码 74-75 行判断如果新的坐标不是可通行的路径，则蛇死亡，游戏失败。

（3）SnackJudge()函数

此函数用于判断蛇头运行到下一位置时游戏的状态。若是食物蛇就进行吃食物的操作：如果是正常的食物，则调用SnackNormalFn()和FoodRelease()函数，使蛇将食物吃掉并释放食物；如果是减短蛇身的食物，则调用SnackShorten()和FoodRelease()函数，吃掉食物并将食物释放；如果是加速的致幻食物，则调用SnackAccelerate()和FoodRelease()函数，吃掉食物并将食物释放。若是障碍物或者蛇身，游戏就失败了，若是路径则游戏就继续进行。

需要注意的如果是蛇身，需要判断蛇尾移动后还能否相撞。

（4）SnackInsert()函数

此函数实现蛇的增长，如果吃了食物后蛇的长度需要增长，则分配一个结点，采用头插法使链表长度增加，即蛇的长度增加。代码 123-124 行，如果蛇的长度达到过关长度，则本关游戏胜利过关。

（5）SnackNormalFn()函数

此函数用于描述蛇吃了正常食物后的状态，它调用了 SnackInsert()函数使蛇的长度增加，又将蛇的状态改为 SnakeNormal 状态。

（6）SnackShorten()函数

此函数用于实现蛇长度的减短。代码 137-141 行，如果蛇的长度减短到小于 2，则游戏失败；如果还没有减短到小于 2，则吃掉食物后，将蛇头结点删除，使蛇头指针指向下一个结点，成为新的蛇头。然后将蛇头结点释放，蛇长度减 1，蛇的状态改为 SnakeShorten。

（7）SnackAccelerate()函数

此函数用于描述蛇吃了加速的致幻食物的状态，它调用了它调用了 SnackInsert()函数使蛇的长度增加，又将蛇的状态改为 SnakeAcceleratel 状态。

（8）SnackDestroy()函数

此函数用于蛇的死亡销毁，蛇死亡或者过关开始下一关游戏，就将链表结点释放。

3、random.c 源文件

random.c 源文件用来初始化随机数系统，获取一个随机数，文件代码如下所示：

```
1  #include "global.h"
2  #include "random.h"
3  #include <stdlib.h>
4  #include <time.h>
5
6  static BOOL bIsInit = FALSE;    // 限定本文件内使用
7  //初始化随机数系统，以时间为种子
8  void InitRandomSystem()
9  {
10     if (!bIsInit)
11     {
12         time_t t;
13         bIsInit = TRUE;
14         srand((unsigned)time(&t));
15     }
16 }
17 //获取一个随机数
18 int GetRandomNumber(int leftVal, int rightVal)
19 {
20     return rand() % (rightVal - leftVal + 1) + leftVal;
21 }
```

此源文件有两个函数：InitRandomSystem()和 GetRandomNumber()，这两个函数也可以合并为一个，但为了使代码更清晰，在本项目中分为两个函数。第一个函数 InitRandomSystem()以时间为种子初始化随机数系统；第二个函数 GetRandomNumber()获取产生的随机数。这两个函数都比较简单，不过多讲解。

4、food.c 源文件

food.c 源文件需要实现三个函数：FoodKindFn()、FoodCreate()和 FoodRelease()。这三个函数分别用于确定食物种类、创建食物、释放食物。代码实现如下所示：

```
1  #include "global.h"
2  #include "random.h"
3  #include "food.h"
4  #include "map.h"
```

```
5
6  static BOOL bIsExisted = FALSE;
7  //确定食物种类
8  enum EMap FoodKindFn()
9  {
10     int xx = GetRandomNumber(1, 10);
11     if (xx > 0 && xx <=8)
12     {
13         return MAP_FOOD_NORMAL;
14     }
15     else if (xx == 9)
16     {
17         return MAP_FOOD_ACCELERATE;
18     }
19     else
20     {
21         return MAP_FOOD_SHORTEN;
22     }
23 }
24
25 //创建食物
26 Food FoodCreate()
27 {
28     int x, y;
29     BOOL result;
30     if (bIsExisted)
31         return food;
32     do
33     {
34         result = TRUE;
35         x = GetRandomNumber(0, width - 1);
36         y = GetRandomNumber(0, height - 1);
37         if (map[x][y] != MAP_ROAD||x==0||y==0)
38             result = FALSE;
39     }while (!result);
40
41     bIsExisted = TRUE;
42     food.x = x;
43     food.y = y;
44     food.foodKind = FoodKindFn();
45
46     return food;
47 }
```

```
48 //释放食物
49 void FoodRelease()
50 {
51     if (bIsExisted)
52     {
53         bIsExisted = FALSE;
54         FoodCreate();
55     }
56 }
```

上述代码中出现了三个函数，下面分别对这三个函数进行讲解：

（1）FoodKindFn()函数

此函数用于确定食物的种类，代码第 10 行获取一个随机数；代码 11-22 行由产生的随机数来确定食物的种类：如果随机在 0-8 范围内，则产生正常食物；如果随机数值为 9，则产生加速的致幻食物；其他情况则产生使蛇长度减短的食物。

（2）FoodCreate()函数

此函数用于创建食物，代码 32-39 行采用 do...while 循环来不断确定产生食物的坐标位置；当位置确定后，代码 44 行调用 FoodKindFn()函数来确定食物。这样两个函数结合便确定了食物产生的位置与种类。

（3）FoodRelease()函数

当蛇将食物吃掉，食物要消失时，就用到该函数。其实食物本身只是栈上的一个结构体，该结构体有两个整形数据存储坐标，当该位置食物消失时，就刷新食物结构体的坐标，同时在地图模块的预处理函数中把以前位置的食物对应的二维数组中的元素值改为路径(1)。

11.2.4 主函数实现

前面已经完成了贪吃蛇项目中所有功能模块的编写，但是功能模块是无法独立运行的，需要一个程序去将这些功能模块按照项目的逻辑思路整合起来，这样才能完成一个完整的项目。此时就需要创建一个 main.c 文件来整合这些代码，main.c 文件中包含 main()函数，是程序的入口。

除此之外，在 main.c 文件中还需要定义 InitGame()、MainLoop()、Failure()、VictoryFn()四个函数，这四个函数的作用分别如下：

（1）InitGame()函数

此函数用于游戏开始的初始化准备，调用已经完成的函数来加载、预处理地图，初始化蛇等。

（2）MainLoop()函数

此函数用于获取键盘的输入，在项目之初讲解过，游戏的运行需要两个线程来执行，主线程控制游戏界面的显示，子线程获取键盘的输入。子线程获取键盘输入就是通过调用 MainLoop()函数来实现的。

（3）Failure ()函数

此函数用于显示游戏失败时的界面。

（4）VictoryFn()函数

此函数用于显示游戏胜利时的界面。

实现了这四个函数后，在 main()函数中创建子线程，两个线程执行不同的代码块共同控制游戏的运行状态。main.c 文件的代码实现如下所示：

```
1  #include "global.h"
2  #include "map.h"
```

```
3  #include "food.h"
4  #include "snack.h"
5  #include "random.h"
6  #include <stdio.h>
7  #include <conio.h>
8  #include <process.h>//windows 线程头文件
9  #include <windows.h>//获取键盘输入的头文件
10
11 void InitGame(int n)
12 {
13     LoadMap(n);                              //加载地图
14     SnackInit();                             //蛇初始化
15     PreviewMap();                            //预处理地图
16    FoodCreate();                             //创建食物
17     PreviewMap();                            //预处理地图
18     DisplayMap();                            //显示地图
19 }
20 void MainLoop(void *param)//游戏的子循环，获取键盘输入
21 {
22     char ch;
23     while (status == GAME_LOOP)              //该线程执行读取用户输入功能
24     {
25         ch = _getch();                       //获取键盘输入
26         switch (ch)
27         {
28         case 'w':
29         case 'W':
30             direction = SNACK_UP;            //向上移动
31             break;
32         case 'a':
33         case 'A':
34             direction = SNACK_LEFT;          //向左移动
35             break;
36         case 's':
37         case 'S':
38             direction = SNACK_DOWN;          //向下移动
39             break;
40         case 'd':
41         case 'D':
42             direction = SNACK_RIGHT;         //向右移动
43             break;
44         default:
45             break;
46         }
```

```
47      }
48  }
49
50  void Failure() //闯关失败，打印游戏结束界面
51  {
52      int i, tmp;
53      system("cls");
54      printf("\n\n\n\n");
55      for (i = 0; i < CONSOLE_MAX_WIDTH; i++) //打印第一排"#"符号
56          printf("#");
57
58      tmp = CONSOLE_MAX_WIDTH - 30;
59      for (i = 0; i < tmp / 2; i++)
60          printf(" ");
61      printf("很抱歉，你失败了！请再次开启游戏\n"); //打印结束提示
62
63      for (i = 0; i < CONSOLE_MAX_WIDTH; i++) //打印第二排"#"符号
64          printf("#");
65      system("pause");
66  }
67
68  void VictoryFn() //打印胜利过关界面
69  {
70      int i, tmp;
71      system("cls");
72      printf("\n\n\n\n");
73      for (i = 0; i < CONSOLE_MAX_WIDTH; i++)
74          printf("#");
75
76      tmp = CONSOLE_MAX_WIDTH - 30;
77      for (i = 0; i < tmp / 2; i++)
78          printf(" ");
79      printf("恭喜您，顺利进入第%d 关,继续愉快地玩耍\n", snackLength);
80
81      for (i = 0; i < CONSOLE_MAX_WIDTH; i++)
82          printf("#");
83      Sleep(500);
84  }
85
86
87  int main()
88  {
89      HANDLE hThread;
90      hThread = (HANDLE)_beginthread(MainLoop, 0, NULL);//创建一个子线程
```

```
91      int selectNum = 1;
92      InitRandomSystem();                 //初始化随机数系统
93  AA:
94      InitGame(selectNum);                //根据随机选的关卡数初始化游戏
95
96      while (status == GAME_LOOP)         //状态一直是 GAME_LOOP 时，一直执行
97      {
98          switch (SnakeStutas)
99          {
100             case SnakeShorten:
101                 Sleep(DELAY_TIME_SHORTEN);      //显示完地图后，停顿 DELAY_TIME 时间
102                 SnackMove();                    //蛇正常移动（为了游戏的健壮性）
103                 break;
104             case SnakeNormal:
105                 Sleep(DELAY_TIME);              //显示完地图后，停顿 DELAY_TIME 时间

106                 SnackMove();                    //蛇正常移动（为了游戏的健壮性）
107                 break;
108             case SnakeAccelerate:
109                 Sleep(DELAY_TIME_ACCELERATE);   //显示完地图后，停顿 DELAY_TIME 时间
110                 SnackMove();                    //蛇正常移动（为了游戏的健壮性）
111                 break;
112             default:
113                 SnackMove();                    //蛇正常移动（为了游戏的健壮性）
114                 break;
115             }
116
117             PreviewMap();//预处理地图
118             DisplayMap();//显示地图
119
120         }
121
122         if (status == GAME_FAILURE)//游戏失败
123         {
124             Failure();//退出
125             SnackDestroy();
126
127         }
128         else if (status == GAME_VICTORY)
129         {
130             VictoryFn();
131             status = GAME_LOOP;
132             selectNum = selectNum % 3+1;
133             goto AA;
```

```
134         }
135     }
```

下面对 main.c 文件代码进行分析，该文件主要包含四个函数：InitGame()、MainLoop()、Failure()、VictoryFn()。

代码 11-19 行实现 InitGame()函数，调用其他模块中已经完成的函数实现游戏状态的初始化；

代码 20-48 行实现 MainLoop()函数，根据键盘的输入控制蛇的移动方向：如果输入'w'/'W'则向上移动；如果输入'a'/'A'则向左移动；如果输入's'/'S'则向下移动；如果输入'd'/'D'则向下移动。

代码 50-66 行实现 Failure()函数，打印游戏失败的界面显示，其显示如图 11-5 所示。

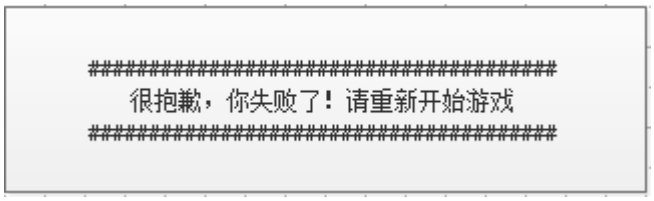

图11-5 游戏失败

代码 68-84 行实现了 VictoryFn()函数，打印游戏胜利过关的界面显示，其显示如图 11-6 所示：

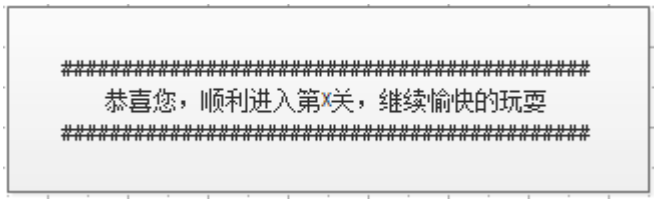

图11-6 胜利过关

代码 87-136 行是 main()函数的实现，在程序运行之初，代码第 90 行就创建了一个子线程 hThread，调用 MainLoop()函数读取键盘的输入。

接下来由主线程执行下面的代码，代码 92 行初始化随机数系统，代码 94 行根据随机选关初始化游戏。

代码 96-120 行用 while()循环控制本关游戏的循环，用 switch...case 语句判断蛇的状态，然后调用 Sleep()函数设置休眠时长 DELAY_TIME_ACCELERATE，这个休眠是为了让游戏界面有一定的动态缓冲，否则界面会刷新的特别快而使游戏无法进行；之后调用 SnackMove()函数使蛇移动。

代码117-118行调用PreviewMap()和DisplayMap()函数，预处理和显示状态改变之后的地图。

代码 122-134 行判断游戏是否失败或胜利过关，如果失败则调用 Failure()函数退出游戏，并调用 SnackDestroy()函数将蛇销毁。如果胜利则调用 VictoryFn()函数显示胜利界面，将 status 状态改为 GAME_LOOP，进行游戏选关，最后调用 goto 语句回到 AA 处重新开始游戏。

11.2.5 效果展示

本节将演示游戏运行效果，方便了解项目功能。

1、 游戏开始

游戏开始运行后，显示运行界面，如图 11-7 所示。

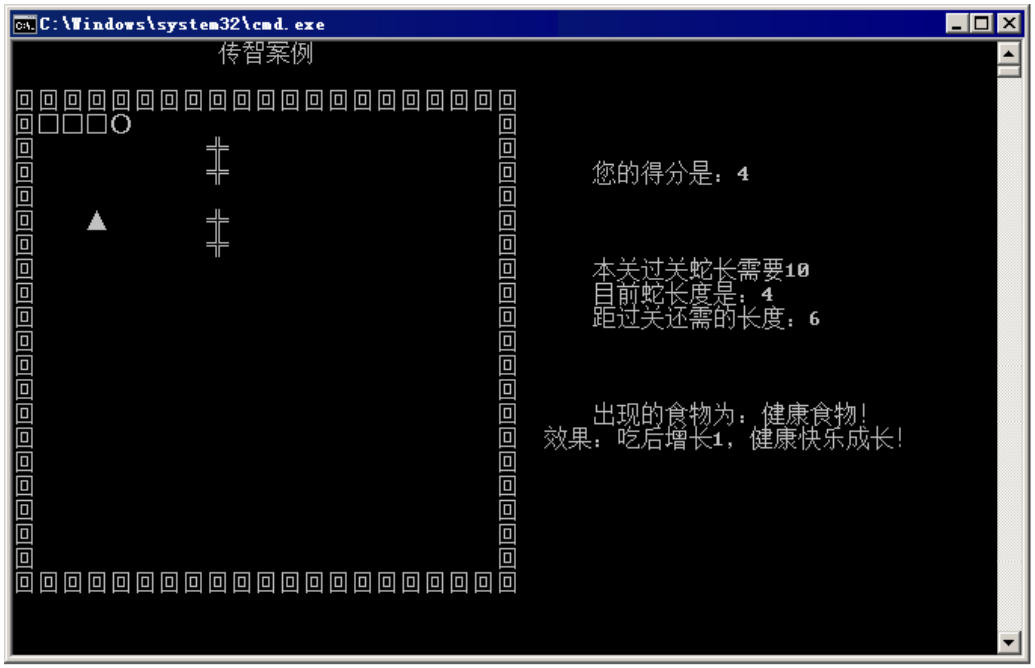

图11-7 游戏运行界面

在图 11-7 的运行界面中，地图边界由符号“回”组成；蛇的起始位置在左上角，长度为 4，蛇头由符号“○”表示，蛇身由符号“□”表示；障碍物由“+”表示；此运行界面中出现的“△”表示正常食物。在地图右边打印了游戏运行状态信息，它显示，目前蛇的长度是 4，过关需要蛇增长到 10，距离过关长度还差 6。地图中出现的食物是健康食物，吃掉后，蛇的长度为增加 1。

2、 游戏失败

当蛇撞到障碍物、或撞到自身、或吃掉过多有毒食物使自身长度减短到长度小于 2，蛇会死亡，则游戏结束，游戏结束界面如图 11-8 所示。

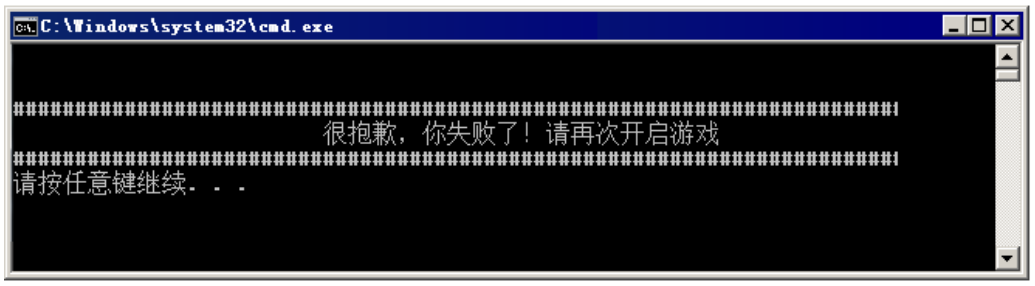

图11-8 游戏结束界面

3、 胜利过关

当蛇吃掉足够多的健康食物，增长到一定长度，则本关游戏胜利，会进入下一关，游戏胜利界面如图 11-9 所示。

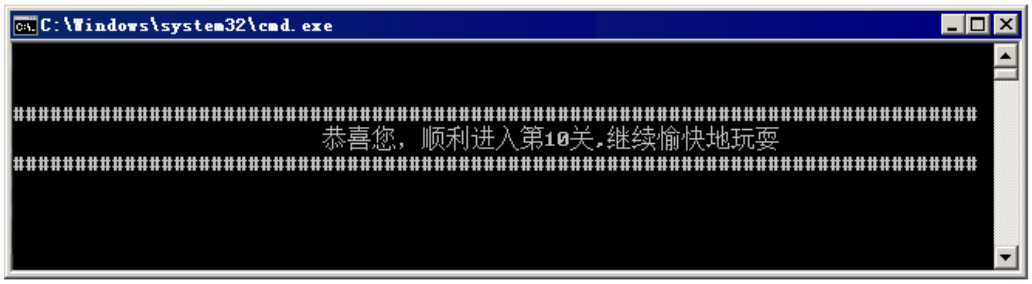

图11-9 游戏胜利过关界面

从图 11-9 中可以看出，本关游戏胜利后将会进入第 10 关游戏。

11.3 项目心得

项目内容介绍完毕，本节将对项目进行简单总结。初学者也应养成这样的习惯，在项目完成后，及时回顾遇到的问题及解决方法，总结得失，为今后的开发工作积累经验。

1、 项目整体规划

每一个项目，在实现之前都要进行分析设计，项目整体要实现哪些功能。将这些功能划分成不同的模块，如果模块较大还可以在内部划分成更小的功能模块。这样逐个实现每个模块，条理清晰。在实现各个模块后，需要将模块整合，使各个功能协调有序的进行。在进行模块划分和模块整合时，可以使用流程图来表示模块之间的联系与运行流程。

2、 多线程

在项目中往往会有一些功能需要同时进行，那么就需要多个线程来执行这些功能，多个线程协调工作。例如本项目中蛇的移动和从键盘读取输入方向，当从键盘读取到输入时，蛇就立刻要作出反应，这需要两个线程同时工作。当子线程读取键盘输入时，通过改变全局变量 Edirection 使蛇改变移动方向。在几个线程协调工作时要设计如何让线程间进行通信。

3、 代码复用

代码复用一直是软件设计追求的目标，本系统实现时也想尽量做到这点，因此将很多操作功能封装成函数，方便进行多次使用。例如本项目中地图的加载、预处理和显示等相应封装成 LoadMap()、PreviewMap()和 DisplayMap()函数，在每一次流界面的刷新中，都要重复调用这些函数，避免了相同代码多次重复编写。

4、 资源回收

资源清理回收往往是初学者容易忽视的问题，而这个问题又容易造成不可预知的严重后果，在资源使用完毕后主动清理完成回收是很重要的。本项目中对于使用完毕的资源进行了有效回收，例如，当蛇吃长度变短时，则删除一个结点并将结点释放完成资源的回收；当蛇死亡时，也是将蛇的结点逐一释放完成资源回收。

【思考题】

在项目中使用单链表模拟蛇，增加结点使蛇长度增加，删除结点使蛇长度减短。请编写函数实现这两个操作。

扫描右方二维码，查看思考题答案！