第30讲-项目实践-贪吃蛇

目录：

正文开始

1. 游戏背景

贪吃蛇是久负盛名的游戏，它也和俄罗斯方块，扫雷等游戏位列经典游戏的行列。

在编程语言的教学中，我们以贪吃蛇为例，从设计到代码实现来提升学生的编程能力和逻辑能力。

2. 游戏效果演示

3. 课程目标

使用C语言在Windows环境的控制台中模拟实现经典小游戏贪吃蛇。

实现基本的功能：

- 贪吃蛇地图绘制
- 蛇吃食物的功能 （上、下、左、右方向键控制蛇的动作）
- 蛇撞墙死亡
- 蛇撞自身死亡
- 计算得分
- 蛇身加速、减速
- 暂停游戏

4. 课程定位

- 提高小比特对编程的兴趣
- 对C语言语法做一个基本的巩固。
- 对游戏开发有兴趣的同学做一个启发。
- 项目适合：C语言学完的同学，有一定的代码能力，初步接触数据结构中的链表。

5. 技术要点

C语言函数、枚举、结构体、动态内存管理、预处理指令、链表、Win32 API等。

6. Win32 API介绍

本次实现贪吃蛇会使用到的一些Win32 API知识，接下来我们就学习一下。

6.1 Win32 API

Windows 这个多作业系统除了协调应用程序的执行、分配内存、管理资源之外， 它同时也是一个很大的服务中心，调用这个服务中心的各种服务（每一种服务就是一个函数），可以帮应用程序达到开启视窗、描绘图形、使用周边设备等目的，由于这些函数服务的对象是应用程序(Application)， 所以便称之为 Application Programming Interface，简称 API 函数。**WIN32 API** 也就是Microsoft Windows 32位平台的应用程序编程接口。

6.2 控制台程序(Console)

平常我们运行起来的黑框程序其实就是控制台程序

我们可以使用cmd命令来设置控制台窗口的长宽：设置控制台窗口的大小，30行，100列

```
1 mode con cols=100 lines=30
```

参考：mode命令

也可以通过命令设置控制台窗口的名字：

```
1 title 贪吃蛇
```

参考：title命令

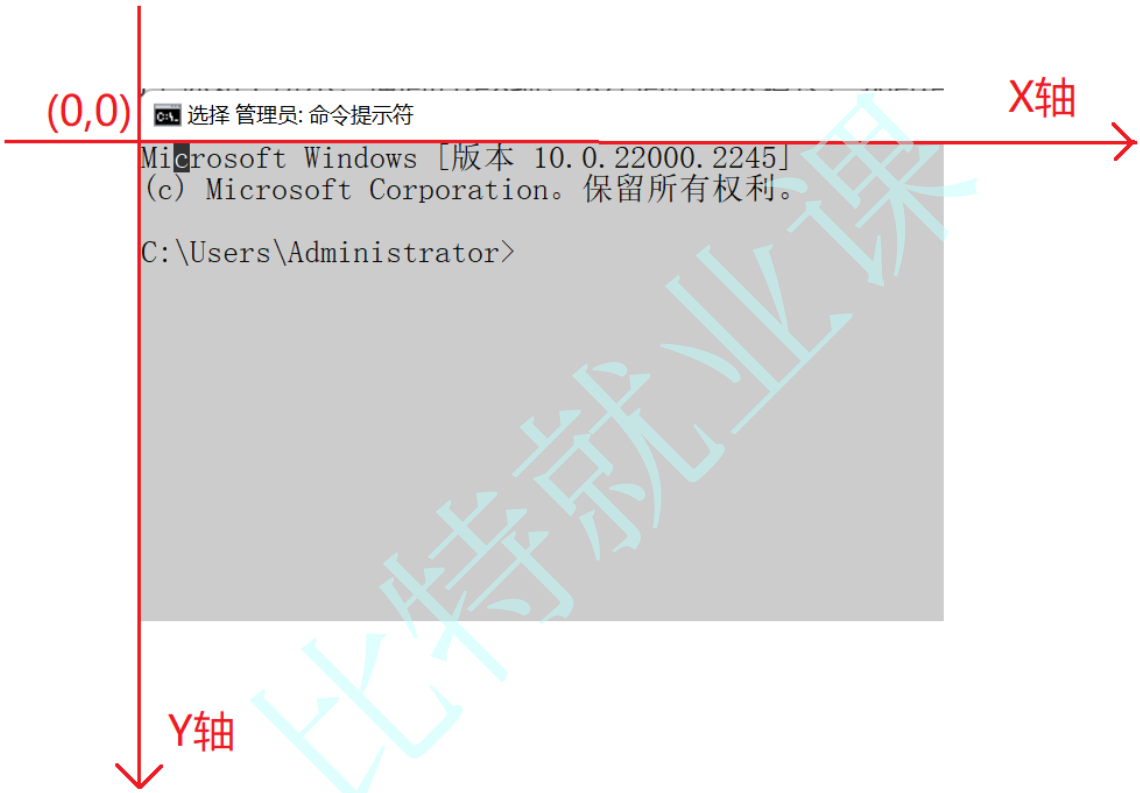
这些能在控制台窗口执行的命令，也可以调用C语言函数system来执行。例如：

```
1 #include <stdio.h>
2 int main()
3 {
```

```
    //设置控制台窗口的长宽：设置控制台窗口的大小，30行，100列
    system("mode con cols=100 lines=30");
    //设置cmd窗口名称
    system("title 贪吃蛇");
    return 0;
}
```

6.3 控制台屏幕上的坐标 COORD

COORD 是Windows API中定义的一个结构体，表示一个字符在控制台屏幕幕缓冲区上的坐标，坐标系(0，0) 的原点位于缓冲区的顶部左侧单元格。



COORD类型的声明：

```
typedef struct _COORD {
    SHORT X;
    SHORT Y;
} COORD, *PCOORD;
```

给坐标赋值：

```
COORD pos = { 10, 15 };
```

6.4 GetStdHandle

> GetStdHandle 是一个Windows API函数。它用于从一个特定的标准设备（标准输入、标准输出或标准错误）中取得一个句柄（用来标识不同设备的数值），使用这个句柄可以操作设备。

```
HANDLE GetStdHandle(DWORD nStdHandle);
```

实例：

```
HANDLE hOutput = NULL;

//获取标准输出的句柄(用来标识不同设备的数值)
hOutput = GetStdHandle(STD_OUTPUT_HANDLE);
```

6.5 GetConsoleCursorInfo

检索有关指定控制台屏幕缓冲区的光标大小和可见性的信息

```
BOOL WINAPI GetConsoleCursorInfo(
    HANDLE              hConsoleOutput,
    PCONSOLE_CURSOR_INFO lpConsoleCursorInfo
);

PCONSOLE_CURSOR_INFO  是指向 CONSOLE_CURSOR_INFO 结构的指针，该结构接收有关主机游标（光标）的信息
```

实例：

```
HANDLE hOutput = NULL;
//获取标准输出的句柄(用来标识不同设备的数值)
hOutput = GetStdHandle(STD_OUTPUT_HANDLE);

CONSOLE_CURSOR_INFO CursorInfo;
GetConsoleCursorInfo(hOutput, &CursorInfo);//获取控制台光标信息
```

6.5.1 CONSOLE_CURSOR_INFO

这个结构体，包含有关控制台光标的信息

```
typedef struct _CONSOLE_CURSOR_INFO {
  DWORD dwSize;
  BOOL  bVisible;
} CONSOLE_CURSOR_INFO, *PCONSOLE_CURSOR_INFO;
```

- dwSize，由光标填充的字符单元格的百分比。 此值介于1到100之间。 光标外观会变化，范围从完全填充单元格到单元底部的水平线条。
- bVisible，游标的可见性。 如果光标可见，则此成员为 TRUE。

```
CursorInfo.bVisible = false; //隐藏控制台光标
```

6.6 SetConsoleCursorInfo

设置指定控制台屏幕缓冲区的光标的大小和可见性。

```
BOOL WINAPI SetConsoleCursorInfo(
    HANDLE  hConsoleOutput,
    const CONSOLE_CURSOR_INFO *lpConsoleCursorInfo
);
```

实例：

```
HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE);

//影藏光标操作
CONSOLE_CURSOR_INFO CursorInfo;
GetConsoleCursorInfo(hOutput, &CursorInfo);//获取控制台光标信息
CursorInfo.bVisible = false; //隐藏控制台光标
SetConsoleCursorInfo(hOutput, &CursorInfo);//设置控制台光标状态
```

6.7 SetConsoleCursorPosition

设置指定控制台屏幕缓冲区中的光标位置，我们将想要设置的坐标信息放在COORD类型的pos中，调用SetConsoleCursorPosition函数将光标位置设置到指定的位置。

```
BOOL WINAPI SetConsoleCursorPosition(
    HANDLE hConsoleOutput,
    COORD  pos
);
```

实例：

```
    COORD pos = { 10, 5};
    HANDLE hOutput = NULL;
    //获取标准输出的句柄(用来标识不同设备的数值)
    hOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    //设置标准输出上光标的位置为pos
    SetConsoleCursorPosition(hOutput, pos);
```

SetPos：封装一个设置光标位置的函数

```
//设置光标的坐标
void SetPos(short x, short y)
{
    COORD pos = { x, y };
    HANDLE hOutput = NULL;
    //获取标准输出的句柄(用来标识不同设备的数值)
    hOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    //设置标准输出上光标的位置为pos
    SetConsoleCursorPosition(hOutput, pos);
}
```

6.8 GetAsyncKeyState

获取按键情况，`GetAsyncKeyState` 的函数原型如下：

```
SHORT GetAsyncKeyState(
    int vKey
);
```

将键盘上每个键的虚拟键值传递给函数，函数通过返回值来分辨按键的状态。

`GetAsyncKeyState` 的返回值是short类型，在上一次调用 `GetAsyncKeyState` 函数后，如果返回的16位的short数据中，最高位是1，说明按键的状态是按下，如果最高是0，说明按键的状态是抬起；如果最低位被置为1则说明，该按键被按过，否则为0。

如果我们要判断一个键是否被按过，可以检测GetAsyncKeyState返回值的最低值是否为1.

```
#define KEY_PRESS(VK)  ( (GetAsyncKeyState(VK) & 0x1) ? 1 : 0 )
```

参考：虚拟键码 (Winuser.h) - Win32 apps

实例：检测数字键

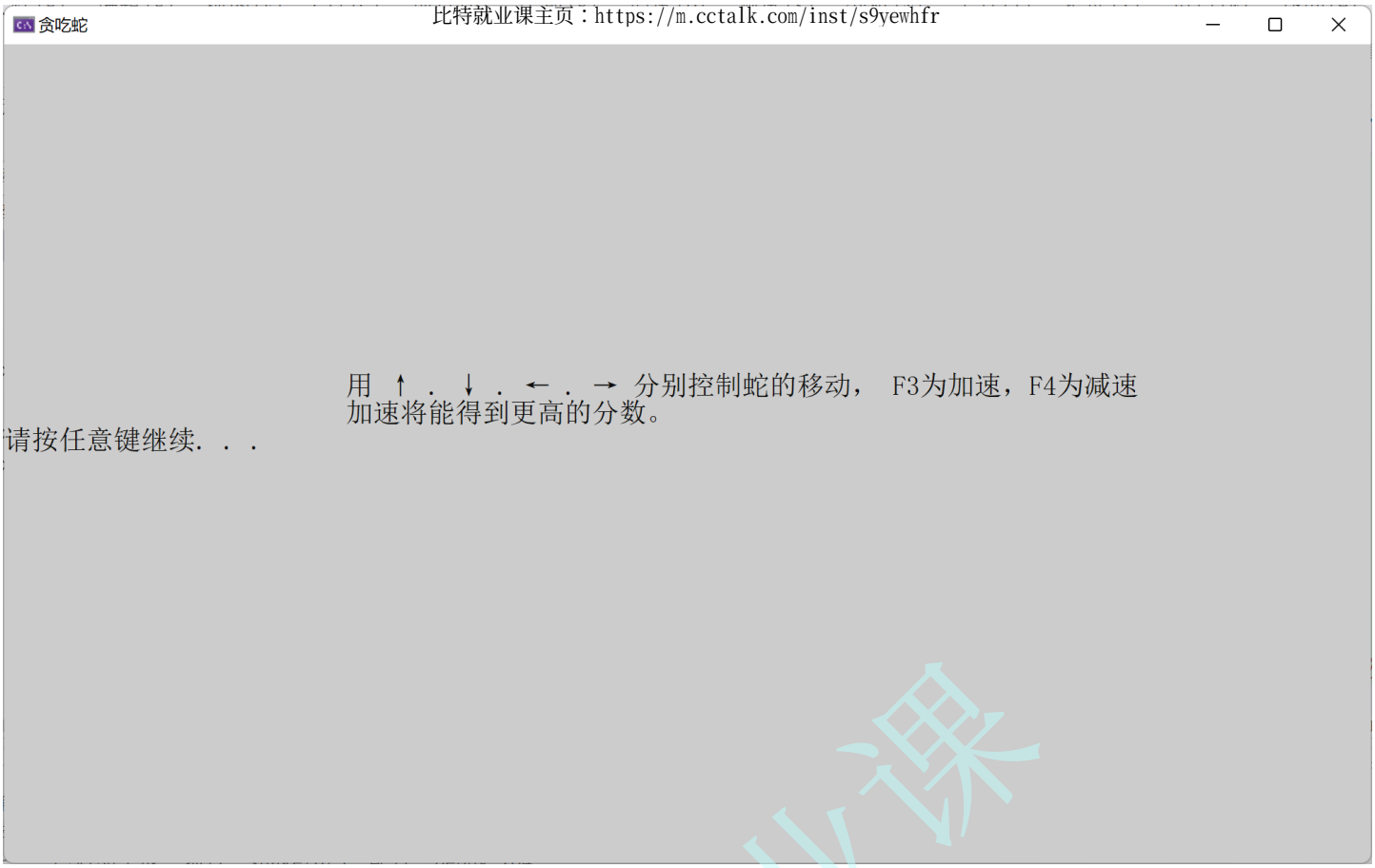
```
#include <stdio.h>
#include <windows.h>

int main()
{
    while (1)
    {
        if (KEY_PRESS(0x30))
        {
            printf("0\n");
        }
        else if (KEY_PRESS(0x31))
        {
            printf("1\n");
        }
        else if (KEY_PRESS(0x32))
        {
            printf("2\n");
        }
        else if (KEY_PRESS(0x33))
        {
            printf("3\n");
        }
        else if (KEY_PRESS(0x34))
        {
            printf("4\n");
        }
        else if (KEY_PRESS(0x35))
        {
            printf("5\n");
        }
        else if (KEY_PRESS(0x36))
```

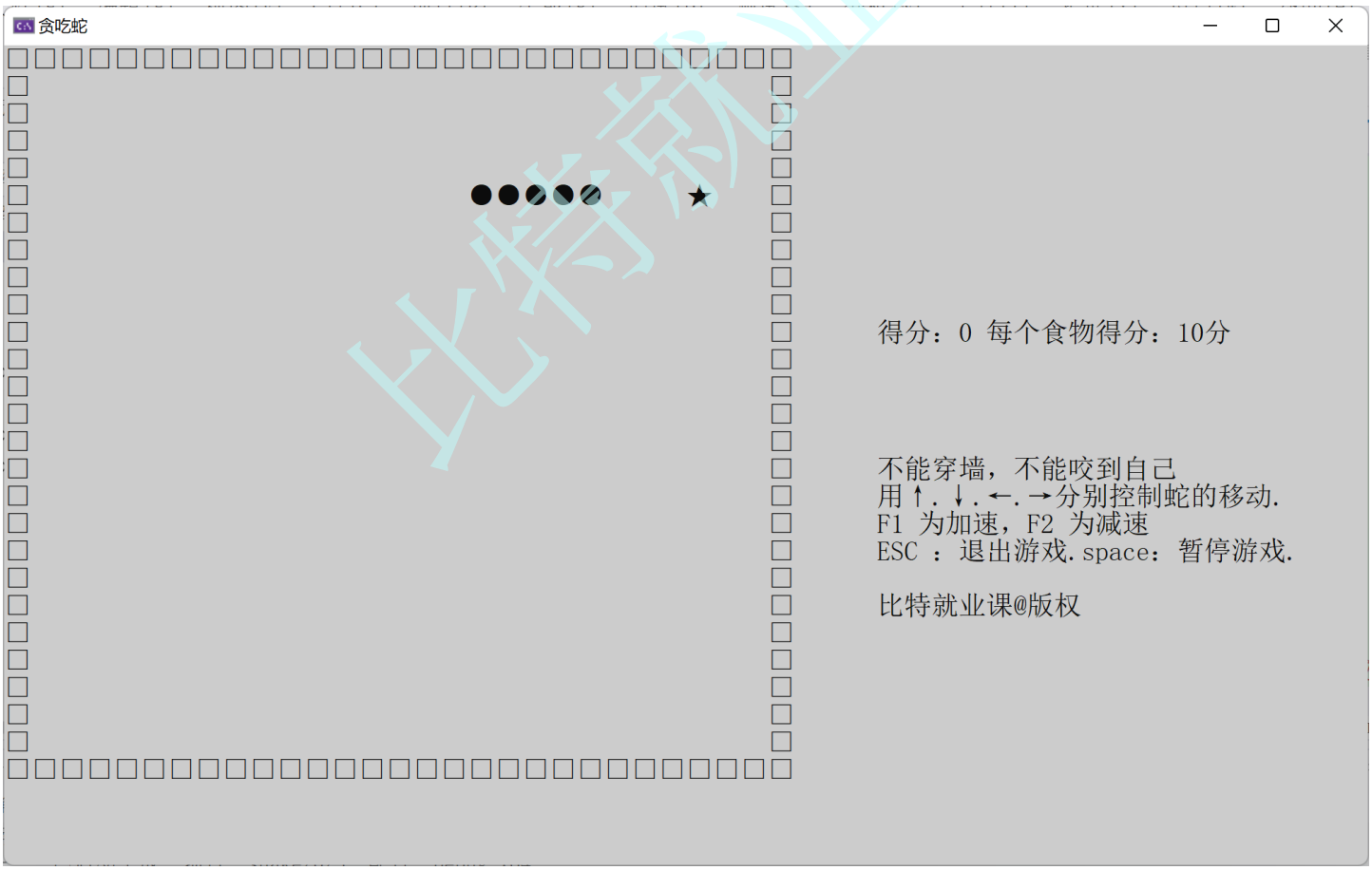
```
33          {
34              printf("6\n");
35          }
36          else if (KEY_PRESS(0x37))
37          {
38              printf("7\n");
39          }
40          else if (KEY_PRESS(0x38))
41          {
42              printf("8\n");
43          }
44          else if (KEY_PRESS(0x39))
45          {
46              printf("9\n");
47          }
48      }
49      return 0;
50  }
```

7. 贪吃蛇游戏设计与分析

7.1 地图

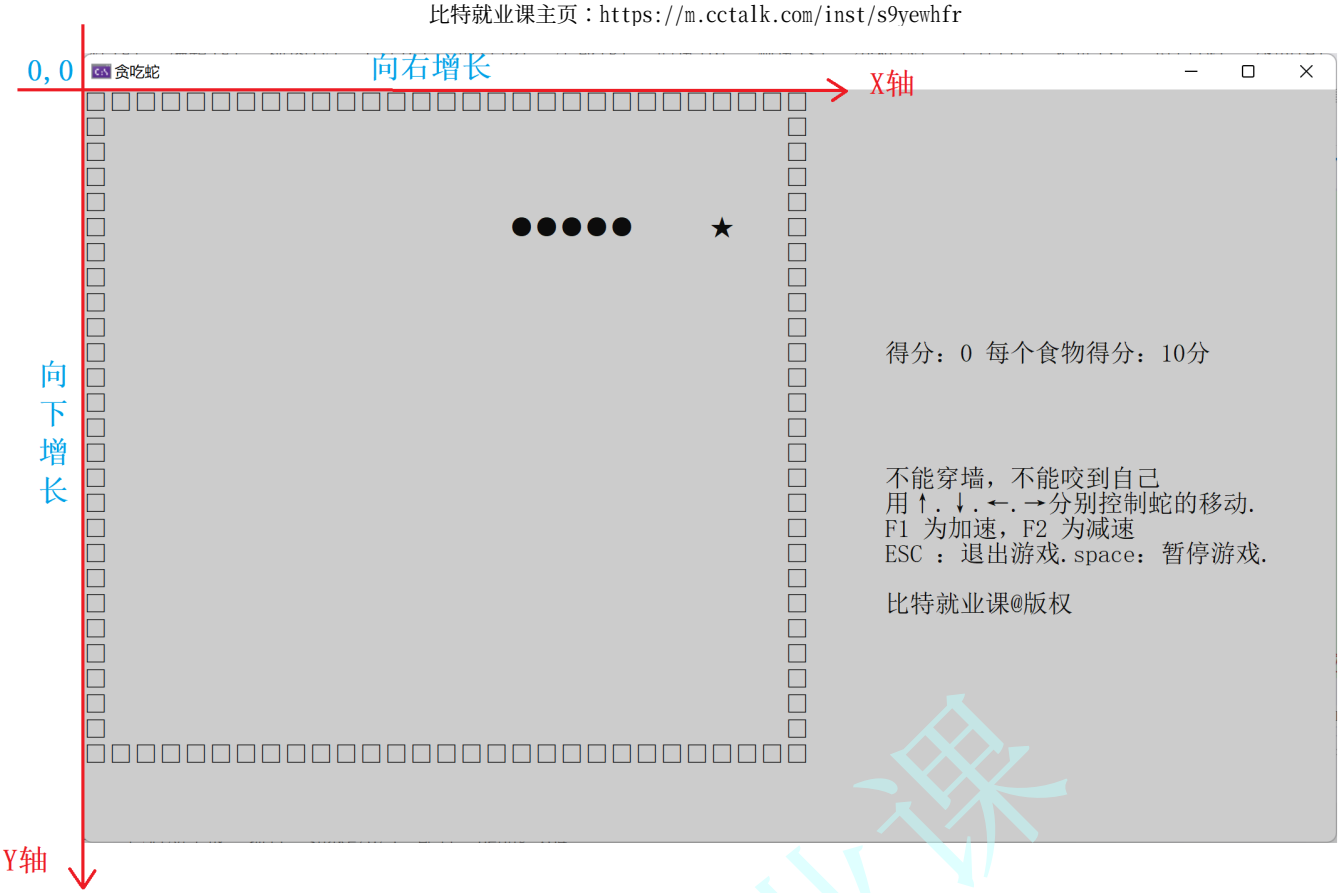
我们最终的贪吃蛇大纲要是这个样子，那我们的地图如何布置呢？





这里不得不讲一下控制台窗口的一些知识，如果想在控制台的窗口中指定位置输出信息，我们得知道该位置的坐标，所以首先介绍一下控制台窗口的坐标知识。

控制台窗口的坐标如下所示，横向的是X轴，从左向右依次增长，纵向是Y轴，从上到下依次增长。



在游戏地图上，我们打印墙体使用宽字符：□，打印蛇使用宽字符●，打印食物使用宽字符★

普通的字符是占一个字节的，这类宽字符是占用2个字节。

这里再简单的讲一下C语言的国际化特性相关的知识，过去C语言并不适合非英语国家（地区）使用。C语言最初假定字符都是单字节的。但是这些假定并不是在世界的任何地方都适用。

> C语言字符默认是采用ASCII编码的，ASCII字符集采用的是单字节编码，且只使用了单字节中的低7位，最高位是没有使用的，可表示为0xxxxxxx；可以看到，ASCII字符集共包含128个字符，在英语国家中，128个字符是基本够用的，但是，在其他国家语言中，比如，在法语中，字母上方有注音符号，它就无法用 ASCII 码表示。于是，一些欧洲国家就决定，利用字节中闲置的最高位编入新的符号。比如，法语中的é的编码为130（二进制10000010）。这样一来，这些欧洲国家使用的编码体系，可以表示最多256个符号。但是，这里又出现了新的问题。不同的国家有不同的字母，因此，哪怕它们都使用256个符号的编码方式，代表的字母却不一样。比如，130在法语编码中代表了é，在希伯来语编码中却代表了字母Gimel (ג)，在俄语编码中又会代表另一个符号。但是不管怎样，所有这些编码方式中，0--127表示的符号是一样的，不一样的只是128--255的这一段。
>
> 至于亚洲国家的文字，使用的符号就更多了，汉字就多达10万左右。一个字节只能表示256种符号，肯定是不够的，就必须使用多个字节表达一个符号。比如，简体中文常见的编码方式是 GB2312，使用两个字节表示一个汉字，所以理论上最多可以表示 256 x 256 = 65536 个符号。

后来为了使C语言适应国际化，C语言的标准中不断加入了国际化的支持。比如：加入了宽字符的类型`wchar_t`和宽字符的输入和输出函数，加入了<locale.h>头文件，其中提供了允许程序员针对特定地区（通常是国家或者说某种特定语言的地理区域）调整程序行为的函数。

7.1.1 <locale.h>本地化

<locale.h>提供的函数用于控制C标准库中对于不同的地区会产生不一样行为的部分。

在标准中，依赖地区的部分有以下几项：

- 数字量的格式
- 货币量的格式
- 字符集
- 日期和时间的表示形式

7.1.2 类项

通过修改地区，程序可以改变它的行为来适应世界的不同区域。但地区的改变可能会影响库的许多部分，其中一部分可能是我们不希望修改的。所以C语言支持针对不同的类项进行修改，下面的一个宏，指定一个类项：

- LC_COLLATE：影响字符串比较函数 `strcoll()` 和 `strxfrm()` 。
- LC_CTYPE：影响字符处理函数的行为。
- LC_MONETARY：影响货币格式。
- LC_NUMERIC：影响 `printf()` 的数字格式。
- LC_TIME：影响时间格式 `strftime()` 和 `wcsftime()` 。
- LC_ALL - 针对所有类项修改，将以上所有类别设置为给定的语言环境。

每个类项的详细说明，请参考

7.1.3 setlocale函数

```
1 char* setlocale (int category, const char* locale);
```

setlocale 函数用于修改当前地区，可以针对一个类项修改，也可以针对所有类项。

setlocale 的第一个参数可以是前面说明的类项中的一个，那么每次只会影响一个类项，如果第一个参数是LC_ALL，就会影响所有的类项。

C标准给第二个参数仅定义了2种可能取值："C"（正常模式）和 ""（本地模式）。

在任意程序执行开始，都会隐藏式执行调用:

```
1 setlocale(LC_ALL, "C");
```

当地区设置为"C"时，设置为C语言默认的模式，这时库函数按正常方式执行。

当程序运行起来后想改变地区，就只能显示调用setlocale函数。用" "作为第2个参数，调用setlocale函数就可以切换到本地模式，这种模式下程序会适应本地环境。

比如：切换到我们的本地模式后就支持宽字符（汉字）的输出等。

```
setlocale(LC_ALL, "");//切换到本地环境
```

`setlocale` 的返回值是一个字符串指针，表示已经设置好的格式。如果调用失败，则返回空指针 `NULL` 。

`setlocale()` 可以用来查询当前地区，这时第二个参数设为 `NULL` 就可以了。

```
#include <locale.h>
int main()
{
        char* loc;
        loc = setlocale(LC_ALL, NULL);
        printf("默认的本地信息: %s\n", loc);

        loc = setlocale(LC_ALL, "");
        printf("设置后的本地信息: %s\n", loc);
        return 0;
}
```

7.1.4 宽字符的打印

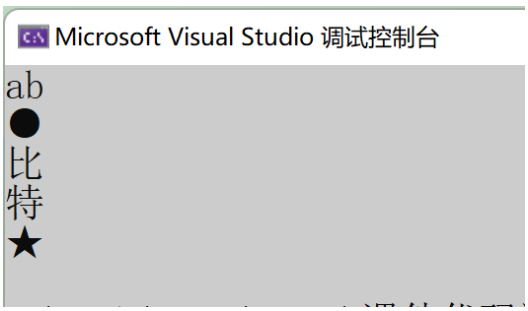
那如果想在屏幕上打印宽字符，怎么打印呢?

宽字符的字面量必须加上前缀 `L` ，否则 C 语言会把字面量当作窄字符类型处理。前缀 `L` 在单引号前面，表示宽字符，宽字符的打印使用 `wprintf` ，对应 `wprintf()` 的占位符为 `%lc` ；在双引号前面，表示宽字符串，对应 `wprintf()` 的占位符为 `%ls` 。

```
#include <stdio.h>
#include<locale.h>

int main() {
        setlocale(LC_ALL, "");
        wchar_t ch1 = L'●';
        wchar_t ch2 = L'比';
        wchar_t ch3 = L'特';
        wchar_t ch4 = L'★';
```

```
10
11          printf("%c%c\n", 'a', 'b');
12
13          wprintf(L"%lc\n", ch1);
14          wprintf(L"%lc\n", ch2);
15          wprintf(L"%lc\n", ch3);
16          wprintf(L"%lc\n", ch4);
17          return 0;
18  }
```

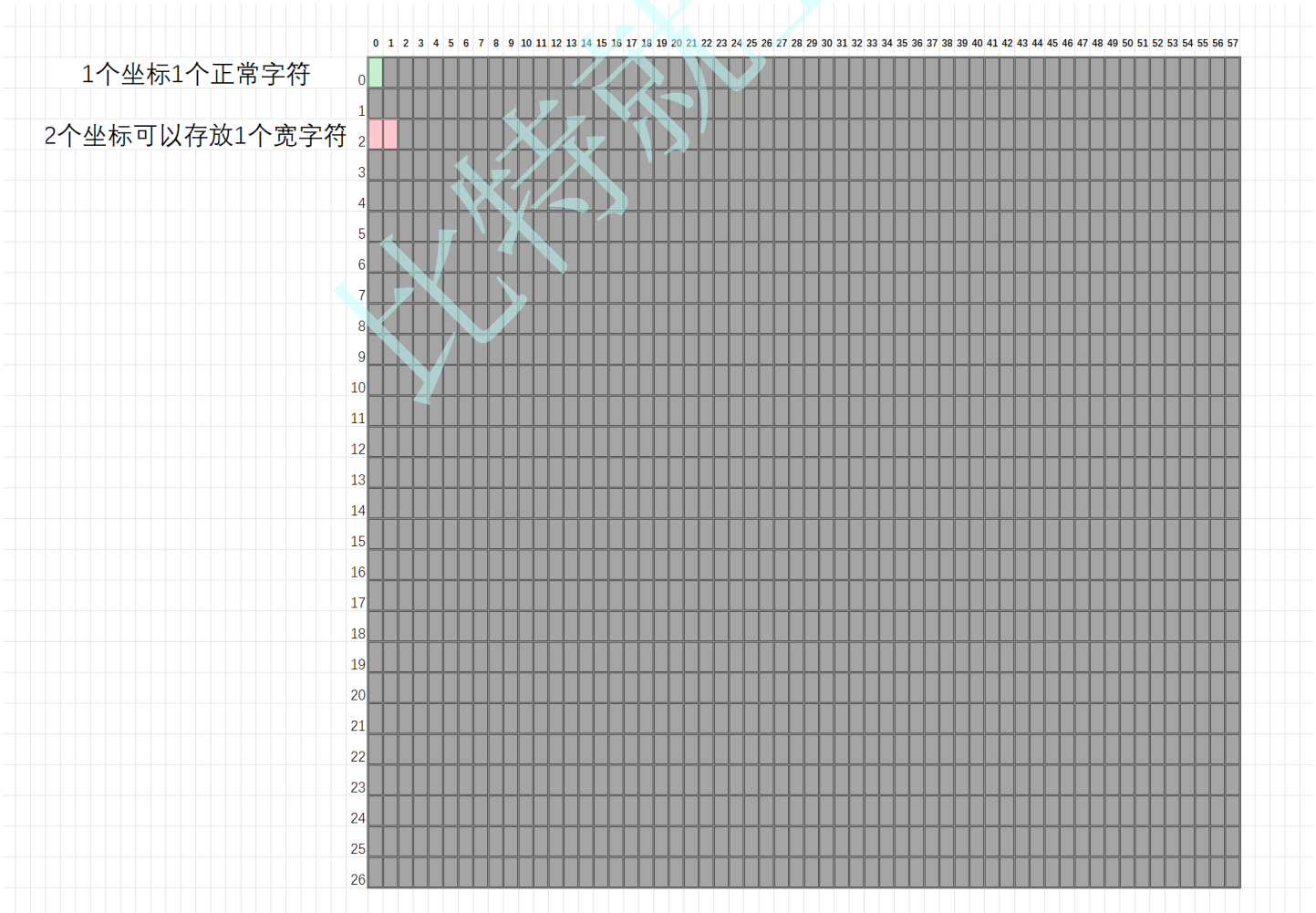
输出结果：



从输出的结果来看，我们发现一个普通字符占一个字符的位置

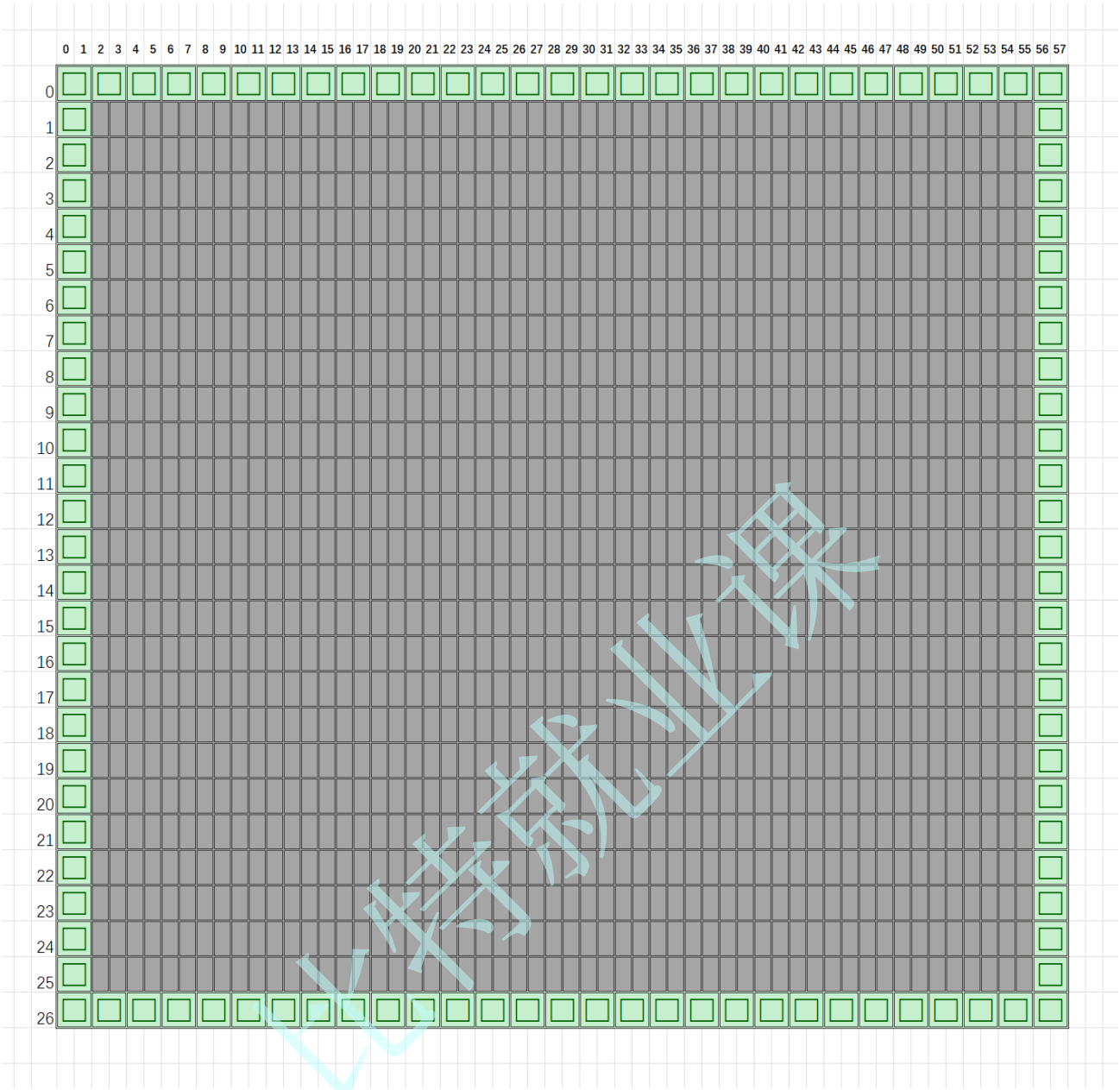
但是打印一个汉字字符，占用2个字符的位置，那么我们如果要在贪吃蛇中使用宽字符，就得处理好地图上坐标的计算。

普通字符和宽字符打印出宽度的展示如下：



7.1.5 地图坐标

我们假设实现一个棋盘27行，58列的棋盘（行和列可以根据自己的情况修改），再围绕地图画出墙，如下：

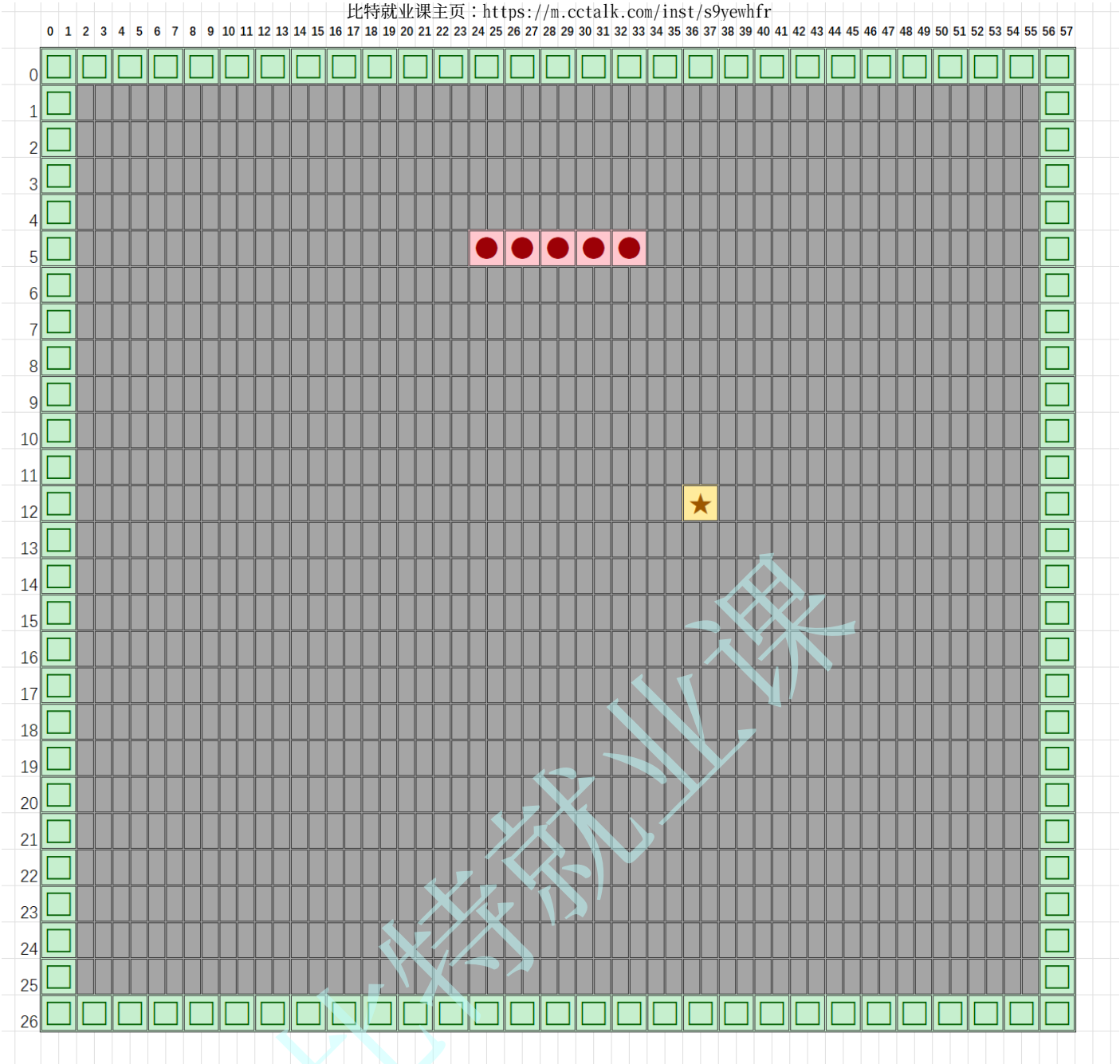


7.2 蛇身和食物

初始化状态，假设蛇的长度是5，蛇身的每个节点是●，在固定的一个坐标处，比如(24, 5)处开始出现蛇，连续5个节点。

注意：蛇的每个节点的x坐标必须是2个倍数，否则可能会出现蛇的一个节点有一半儿出现在墙体中，另外一般在墙外的现象，坐标不好对齐。

关于食物，就是在墙体内随机生成一个坐标（x坐标必须是2的倍数），坐标不能和蛇的身体重合，然后打印★。



7.3 数据结构设计

在游戏运行的过程中，蛇每次吃一个食物，蛇的身体就会变长一节，如果我们使用链表存储蛇的信息，那么蛇的每一节其实就是链表的每个节点。每个节点只要记录好蛇身节点在地图上的坐标就行，所以蛇节点结构如下：

```
typedef struct SnakeNode
{
    int x;
    int y;
    struct SnakeNode* next;
}SnakeNode, * pSnakeNode;
```

要管理整条贪吃蛇，我们再封装一个Snake的结构来维护整条贪吃蛇：

```
typedef struct Snake
{
    pSnakeNode _pSnake;//维护整条蛇的指针
    pSnakeNode _pFood;//维护食物的指针
    enum DIRECTION _Dir;//蛇头的方向,默认是向右
    enum GAME_STATUS _Status;//游戏状态
    int _Socre;//游戏当前获得分数
    int _foodWeight;//默认每个食物10分
    int _SleepTime;//每走一步休眠时间
}Snake, * pSnake;
```

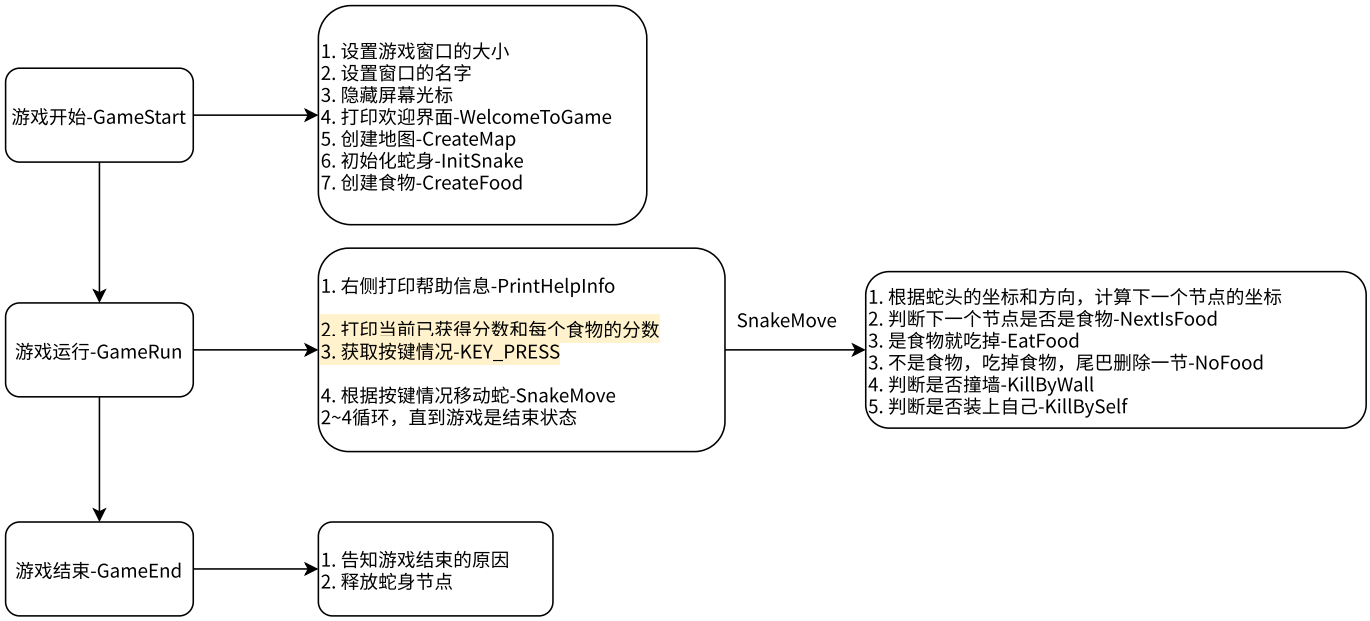
蛇的方向，可以一一列举，使用枚举

```
//方向
enum DIRECTION
{
    UP = 1,
    DOWN,
    LEFT,
    RIGHT
};
```

游戏状态，可以一一列举，使用枚举

```
//游戏状态
enum GAME_STATUS
{
    OK,//正常运行
    KILL_BY_WALL,//撞墙
    KILL_BY_SELF,//咬到自己
    END_NOMAL//正常结束
};
```

7.4 游戏流程设计



8. 核心逻辑实现分析

8.1 游戏主逻辑

程序开始就设置程序支持本地模式，然后进入游戏的主逻辑。

主逻辑分为3个过程：

- 游戏开始（GameStart）完成游戏的初始化
- 游戏运行（GameRun）完成游戏运行逻辑的实现
- 游戏结束（GameEnd）完成游戏结束的说明，实现资源释放

```c
#include <locale.h>

void test()
{
    int ch = 0;
    srand((unsigned int)time(NULL));

    do
    {
        Snake snake = { 0 };
        GameStart(&snake);
        GameRun(&snake);
        GameEnd(&snake);
        SetPos(20, 15);
        printf("再来一局吗？(Y/N):");
        ch = getchar();
```

```
        getchar();//清理\n

    } while (ch == 'Y');
    SetPos(0, 27);
}

int main()
{
    //修改当前地区为本地模式，为了支持中文宽字符的打印
    setlocale(LC_ALL, "");
    //测试逻辑
    test();
    return 0;
}
```

8.2 游戏开始（GameStart）

这个模块完成游戏的初始化任务：

- 控制台窗口大小的设置
- 控制台窗口名字的设置
- 鼠标光标的隐藏
- 打印欢迎界面
- 创建地图
- 初始化蛇
- 创建第一个食物

```
void GameStart(pSnake ps)
{
    //设置控制台窗口的大小，30行，100列
    //mode 为DOS命令
    system("mode con cols=100 lines=30");
    //设置cmd窗口名称
    system("title 贪吃蛇");

    //获取标准输出的句柄(用来标识不同设备的数值)
    HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE);

    //影藏光标操作
    CONSOLE_CURSOR_INFO CursorInfo;
    GetConsoleCursorInfo(hOutput, &CursorInfo);//获取控制台光标信息
    CursorInfo.bVisible = false; //隐藏控制台光标
    SetConsoleCursorInfo(hOutput, &CursorInfo);//设置控制台光标状态
```

```
17
18      //打印欢迎界面
19      WelcomeToGame();
20      //打印地图
21      CreateMap();
22      //初始化蛇
23      InitSnake(ps);
24      //创造第一个食物
25      CreateFood(ps);
26  }
```

8.2.1 打印欢迎界面

在游戏正式开始之前，做一些功能提醒

```
1  void WelcomeToGame()
2  {
3      SetPos(40, 15);
4      printf("欢迎来到贪吃蛇小游戏");
5      SetPos(40, 25);//让按任意键继续的出现的位置好看点
6      system("pause");
7      system("cls");
8      SetPos(25, 12);
9      printf("用 ↑ . ↓ . ← . → 分别控制蛇的移动， F3为加速，F4为减速\n");
10     SetPos(25, 13);
11     printf("加速将能得到更高的分数。\n");
12     SetPos(40, 25);//让按任意键继续的出现的位置好看点
13     system("pause");
14     system("cls");
15 }
```

8.2.2 创建地图

创建地图就是将墙打印出来，因为是宽字符打印，所有使用wprintf函数，打印格式串前使用L

打印地图的关键是要算好坐标，才能在想要的位置打印墙体。

墙体打印的宽字符：

```
#define WALL L'□'
```

易错点：就是坐标的计算

上：（0,0）到（56,0）

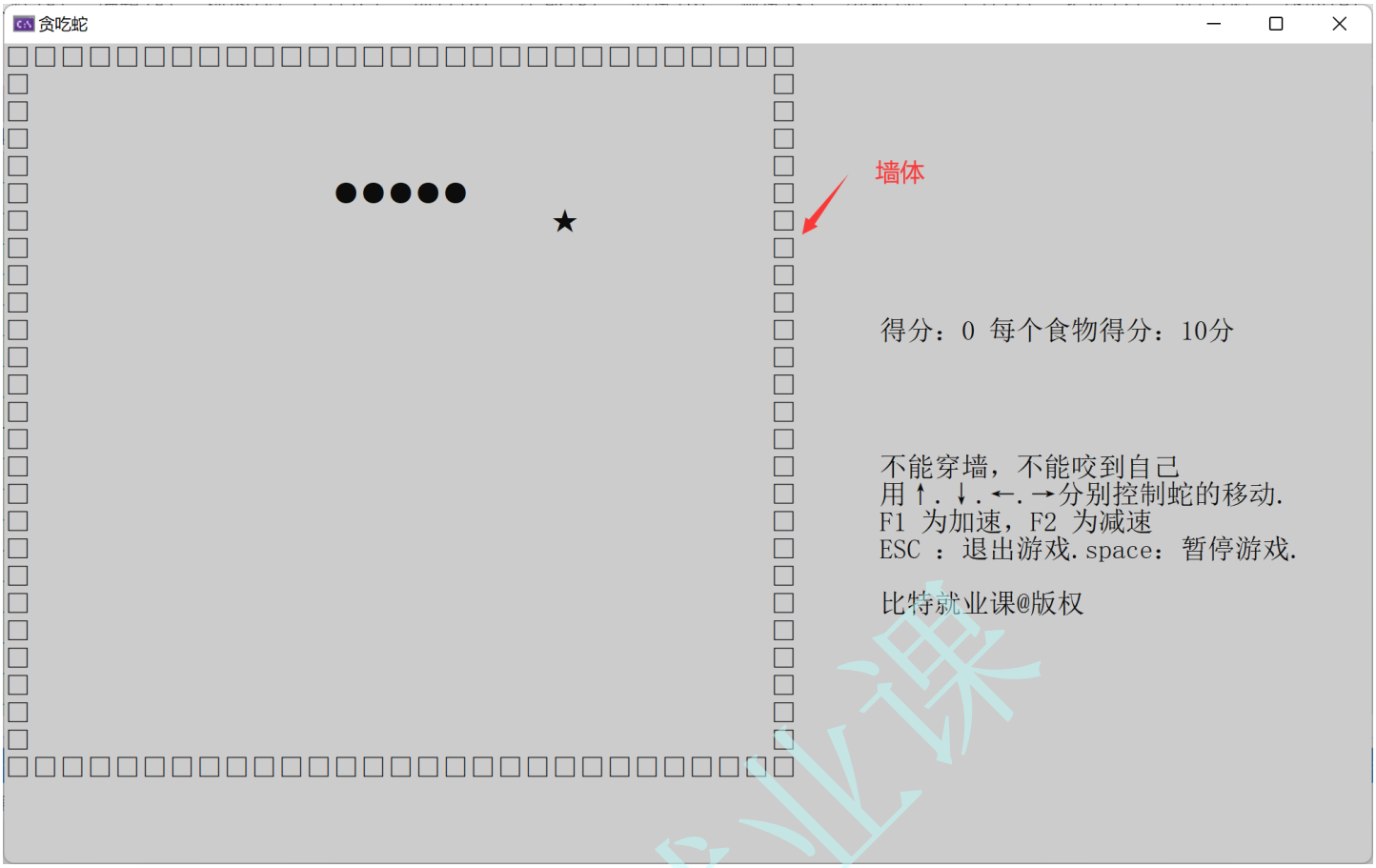
下：（0,26）到（56,26）

左：（0,1）到（0,25）

右：（56,1）到（56,25）

创建地图函数CreateMap

```
void CreateMap()
{
    int i = 0;
    //上(0,0)-(56, 0)
    SetPos(0, 0);
    for (i = 0; i < 58; i += 2)
    {
        wprintf(L"%c", WALL);
    }
    //下(0,26)-(56, 26)
    SetPos(0, 26);
    for (i = 0; i < 58; i += 2)
    {
        wprintf(L"%c", WALL);
    }
    //左
    //x是0，y从1开始增长
    for (i = 1; i < 26; i++)
    {
        SetPos(0, i);
        wprintf(L"%c", WALL);
    }
    //x是56，y从1开始增长
    for (i = 1; i < 26; i++)
    {
        SetPos(56, i);
        wprintf(L"%c", WALL);
    }
}
```



8.2.3 初始化蛇身

蛇最开始长度为5节，每节对应链表的一个节点，蛇身的每一个节点都有自己的坐标。

创建5个节点，然后将每个节点存放在链表中进行管理。创建完蛇身后，将蛇的每一节打印在屏幕上。

- 蛇的初始位置从 (24,5) 开始。

再设置当前游戏的状态，蛇移动的速度，默认的方向，初始成绩，每个食物的分数。

- 游戏状态是：OK
- 蛇的移动速度：200毫秒
- 蛇的默认方向：RIGHT
- 初始成绩：0
- 每个食物的分数：10

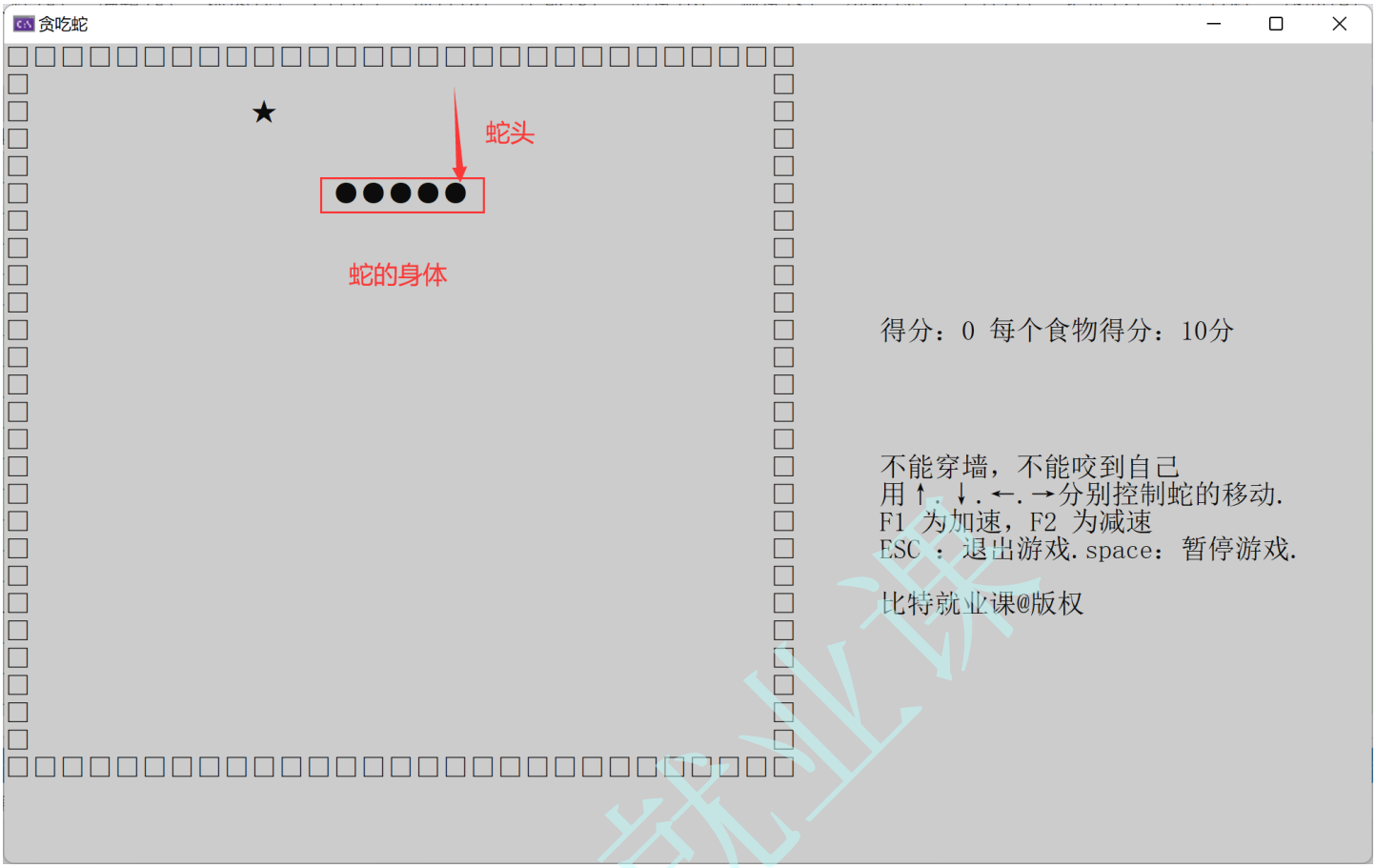
蛇身打印的宽字符：

```
1 #define BODY L'●'
```

初始化蛇身函数：InitSnake

```c
void InitSnake(pSnake ps)
{
    pSnakeNode cur = NULL;
    int i = 0;
    //创建蛇身节点，并初始化坐标
    //头插法
    for (i = 0; i < 5; i++)
    {
        //创建蛇身的节点
        cur = (pSnakeNode)malloc(sizeof(SnakeNode));
        if (cur == NULL)
        {
            perror("InitSnake()::malloc()");
            return;
        }
        //设置坐标
        cur->next = NULL;
        cur->x = POS_X + i * 2;
        cur->y = POS_Y;

        //头插法
        if (ps->_pSnake == NULL)
        {
            ps->_pSnake = cur;
        }
        else
        {
            cur->next = ps->_pSnake;
            ps->_pSnake = cur;
        }
    }

    //打印蛇的身体
    cur = ps->_pSnake;
    while (cur)
    {
        SetPos(cur->x, cur->y);
        wprintf(L"%lc", BODY);
        cur = cur->next;
    }

    //初始化贪吃蛇数据
    ps->_SleepTime = 200;
    ps->_Socre = 0;
    ps->_Status = OK;
    ps->_Dir = RIGHT;
```

```
47      ps->_foodWeight = 10;
48  }
```



8.2.4 创建第一个食物

- 先随机生成食物的坐标
  - x坐标必须是2的倍数
  - 食物的坐标得在墙体内部
  - 食物的坐标不能和蛇身每个节点的坐标重复
- 创建食物节点，打印食物

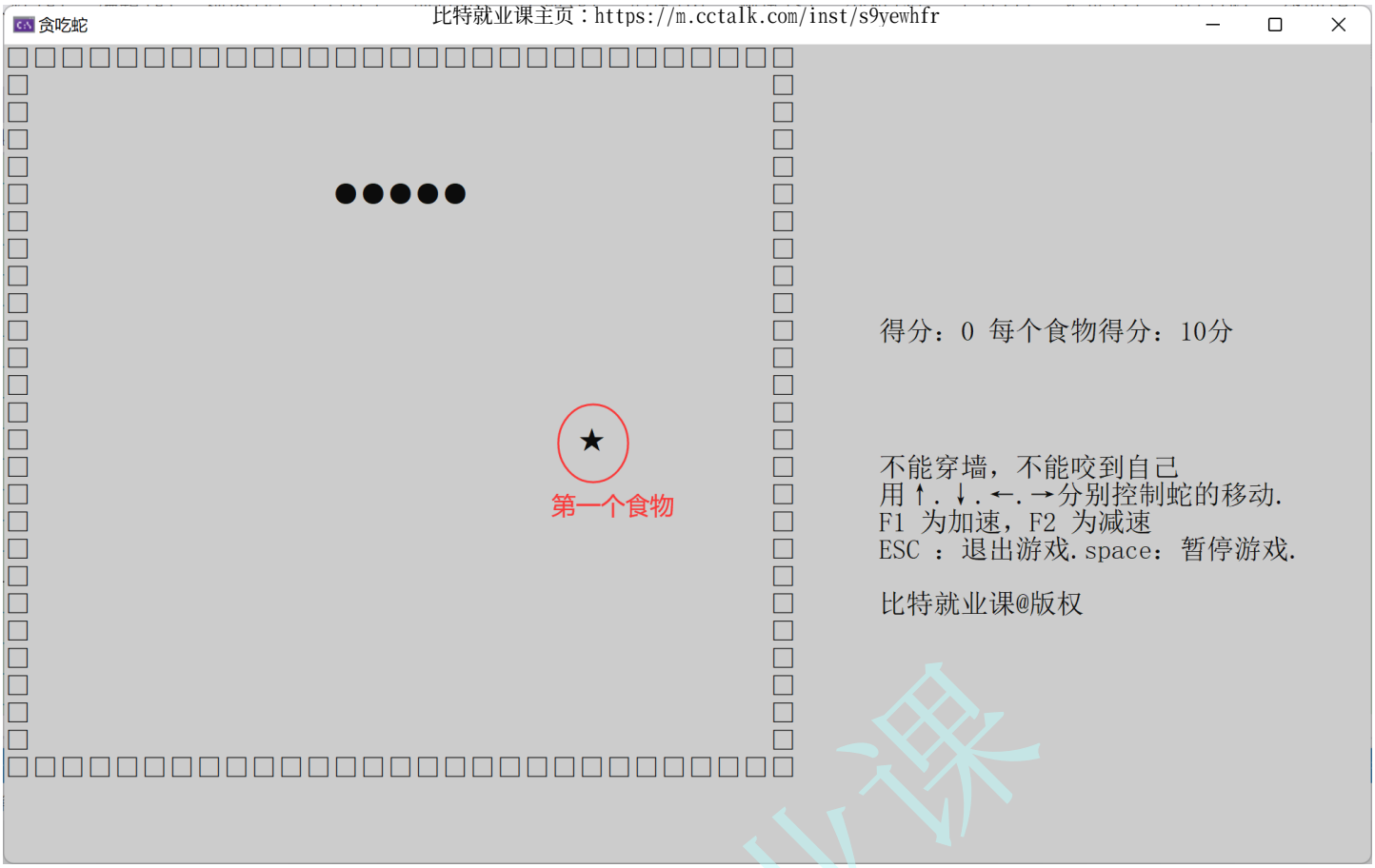
食物打印的宽字符:

```
1  #define FOOD L'★'
```

创建食物的函数：CreateFood

```
1  void CreateFood(pSnake ps)
2  {
```

```
    int x = 0;
    int y = 0;

again:
    //产生的x坐标应该是2的倍数，这样才可能和蛇头坐标对齐。
    do
    {
        x = rand() % 53 + 2;
        y = rand() % 25 + 1;
    } while (x % 2 != 0);

    pSnakeNode cur = ps->_pSnake;//获取指向蛇头的指针
    //食物不能和蛇身冲突
    while (cur)
    {
        if (cur->x == x && cur->y == y)
        {
            goto again;
        }
        cur = cur->next;
    }

    pSnakeNode pFood = (pSnakeNode)malloc(sizeof(SnakeNode)); //创建食物
    if (pFood == NULL)
    {
        perror("CreateFood::malloc()");
        return;
    }
    else
    {
        pFood->x = x;
        pFood->y = y;
        SetPos(pFood->x, pFood->y);
        wprintf(L"%c", FOOD);
        ps->_pFood = pFood;
    }
}
```



8.3 游戏运行（GameRun）

游戏运行期间，右侧打印帮助信息，提示玩家，坐标开始位置(64, 15)

根据游戏状态检查游戏是否继续，如果是状态是OK，游戏继续，否则游戏结束。

如果游戏继续，就是检测按键情况，确定蛇下一步的方向，或者是否加速减速，是否暂停或者退出游戏。

需要的虚拟按键的罗列：

- 上：`VK_UP`
- 下：`VK_DOWN`
- 左：`VK_LEFT`
- 右：`VK_RIGHT`
- 空格：`VK_SPACE`
- ESC：`VK_ESCAPE`
- F3：`VK_F3`
- F4：`VK_F4`

确定了蛇的方向和速度，蛇就可以移动了。

```c
void GameRun(pSnake ps)
{
    //打印右侧帮助信息
    PrintHelpInfo();
    do
    {
        SetPos(64, 10);
        printf("得分: %d ", ps->_Socre);
        printf("每个食物得分: %d分", ps->_foodWeight);
        if (KEY_PRESS(VK_UP) && ps->_Dir != DOWN)
        {
            ps->_Dir = UP;
        }
        else if (KEY_PRESS(VK_DOWN) && ps->_Dir != UP)
        {
            ps->_Dir = DOWN;
        }
        else if (KEY_PRESS(VK_LEFT) && ps->_Dir != RIGHT)
        {
            ps->_Dir = LEFT;
        }
        else if (KEY_PRESS(VK_RIGHT) && ps->_Dir != LEFT)
        {
            ps->_Dir = RIGHT;
        }
        else if (KEY_PRESS(VK_SPACE))
        {
            pause();
        }
        else if (KEY_PRESS(VK_ESCAPE))
        {
            ps->_Status = END_NOMAL;
            break;
        }
        else if (KEY_PRESS(VK_F3))
        {
            if (ps->_SleepTime >= 80)
            {
                ps->_SleepTime -= 30;
                ps->_foodWeight += 2;//一个食物分数最高是20分
            }
        }
        else if (KEY_PRESS(VK_F4))
        {
            if (ps->_SleepTime < 320)
            {
                ps->_SleepTime += 30;
```

```
                ps->_FoodWeight -= 2;//一个食物分数最低是2分
            }
        }
        //蛇每次一定之间要休眠的时间，时间短，蛇移动速度就快
        Sleep(ps->_SleepTime);
        SnakeMove(ps);

    } while (ps->_Status == OK);
}
```

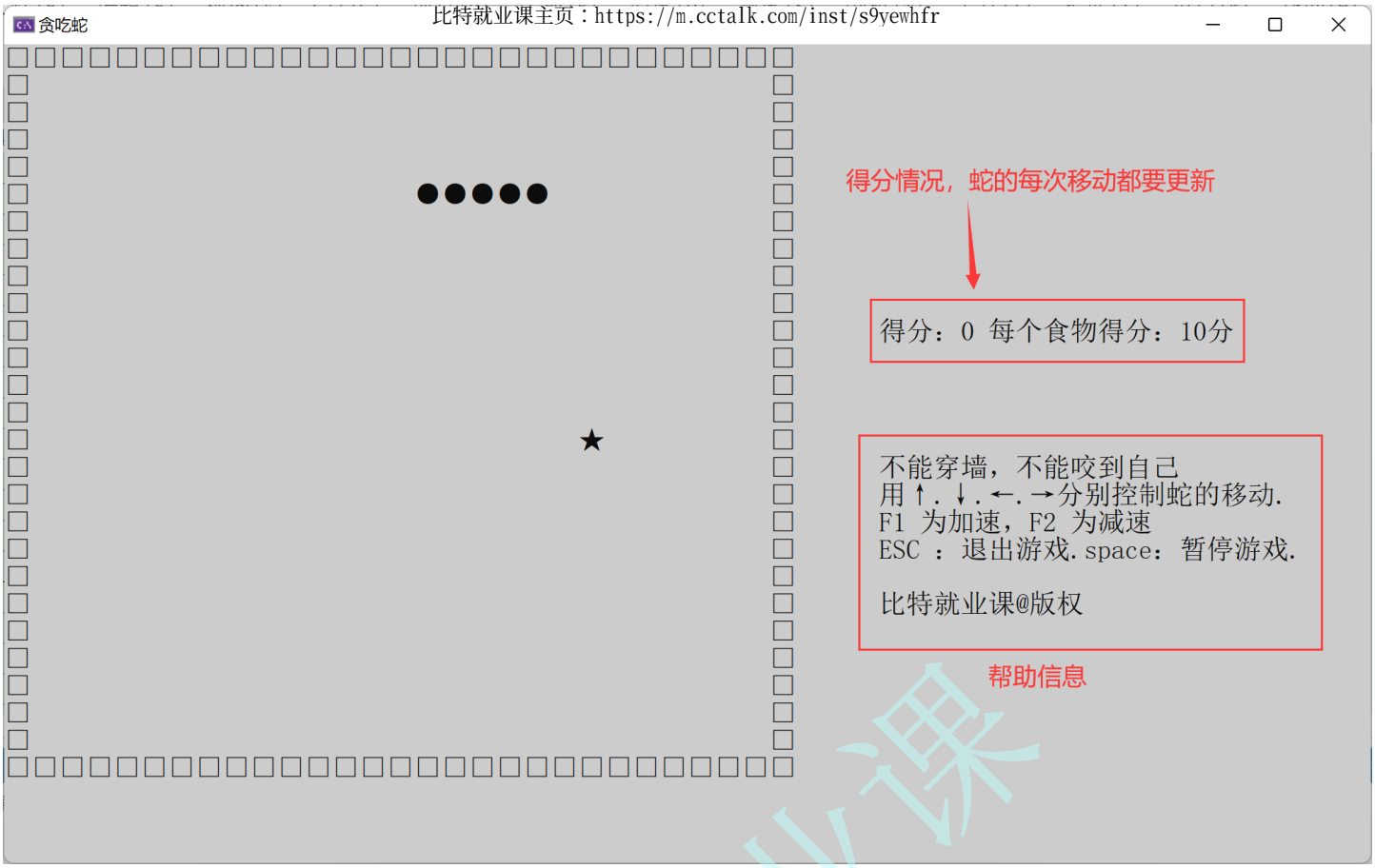
8.3.1 KEY_PRESS

检测按键状态，我们封装了一个宏

```
#define KEY_PRESS(VK)  ((GetAsyncKeyState(VK)&0x1) ? 1 : 0)
```

8.3.2 PrintHelpInfo

```
void PrintHelpInfo()
{
    //打印提示信息
    SetPos(64, 15);
    printf("不能穿墙，不能咬到自己\n");
    SetPos(64, 16);
    printf("用↑.↓.←.→分别控制蛇的移动.");
    SetPos(64, 17);
    printf("F3 为加速，F4 为减速\n");
    SetPos(64, 18);
    printf("ESC ： 退出游戏.space：暂停游戏.");
    SetPos(64, 20);
    printf("比特就业课@版权");
}
```



8.3.3 蛇身移动（SnakeMove）

先创建下一个节点，根据移动方向和蛇头的坐标，蛇移动到下一个位置的坐标。

确定了下一个位置后，看下一个位置是否是食物（NextIsFood），是食物就做吃食物处理（EatFood），如果不是食物则做前进一步的处理（NoFood）。

蛇身移动后，判断此次移动是否会造成撞墙（KillByWall）或者撞上自己蛇身（KillBySelf），从而影响游戏的状态。

```
1  void SnakeMove(pSnake ps)
2  {
3      //创建下一个节点
4      pSnakeNode pNextNode = (pSnakeNode)malloc(sizeof(SnakeNode));
5      if (pNextNode == NULL)
6      {
7          perror("SnakeMove()::malloc()");
8          return;
9      }
10     //确定下一个节点的坐标，下一个节点的坐标根据，蛇头的坐标和方向确定
11     switch (ps->_Dir)
12     {
13         case UP:
14         {
15             pNextNode->x = ps->_pSnake->x;
16             pNextNode->y = ps->_pSnake->y - 1;
```

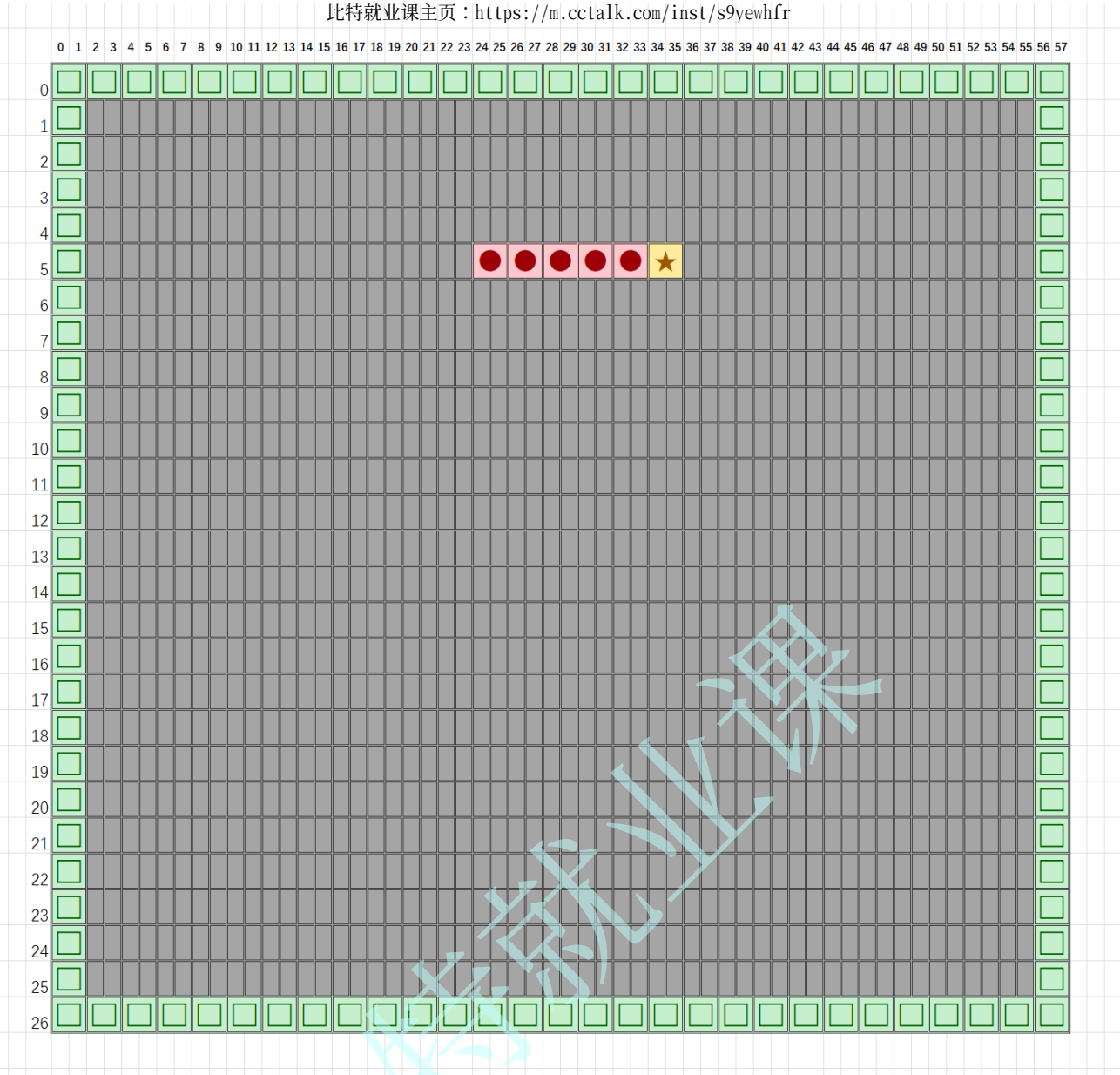
```
        }
        break;
        case DOWN:
        {
            pNextNode->x = ps->_pSnake->x;
            pNextNode->y = ps->_pSnake->y + 1;
        }
        break;
        case LEFT:
        {
            pNextNode->x = ps->_pSnake->x - 2;
            pNextNode->y = ps->_pSnake->y;
        }
        break;
        case RIGHT:
        {
            pNextNode->x = ps->_pSnake->x + 2;
            pNextNode->y = ps->_pSnake->y;
        }
        break;
    }

    //如果下一个位置就是食物
    if (NextIsFood(pNextNode, ps))
    {
        EatFood(pNextNode, ps);
    }
    else//如果没有食物
    {
        NoFood(pNextNode, ps);
    }

    KillByWall(ps);
    KillBySelf(ps);
}
```

8.3.3.1 NextIsFood

```
//pSnakeNode psn 是下一个节点的地址
//pSnake ps 维护蛇的指针
int NextIsFood(pSnakeNode psn, pSnake ps)
{
    return (psn->x == ps->_pFood->x) && (psn->y == ps->_pFood->y);
}
```

8.3.3.2 EatFood

```c
//pSnakeNode psn 是下一个节点的地址
//pSnake ps 维护蛇的指针
void EatFood(pSnakeNode psn, pSnake ps)
{
    //头插法
    psn->next = ps->_pSnake;
    ps->_pSnake = psn;

    //打印蛇
    pSnakeNode cur = ps->_pSnake;
    while (cur)
    {
        SetPos(cur->x, cur->y);
        wprintf(L"%c", BODY);
        cur = cur->next;
    }
    ps->_Socre += ps->_foodWeight;

    //释放食物节点
    free(ps->_pFood);
    //创建新的食物
    CreateFood(ps);
}
```



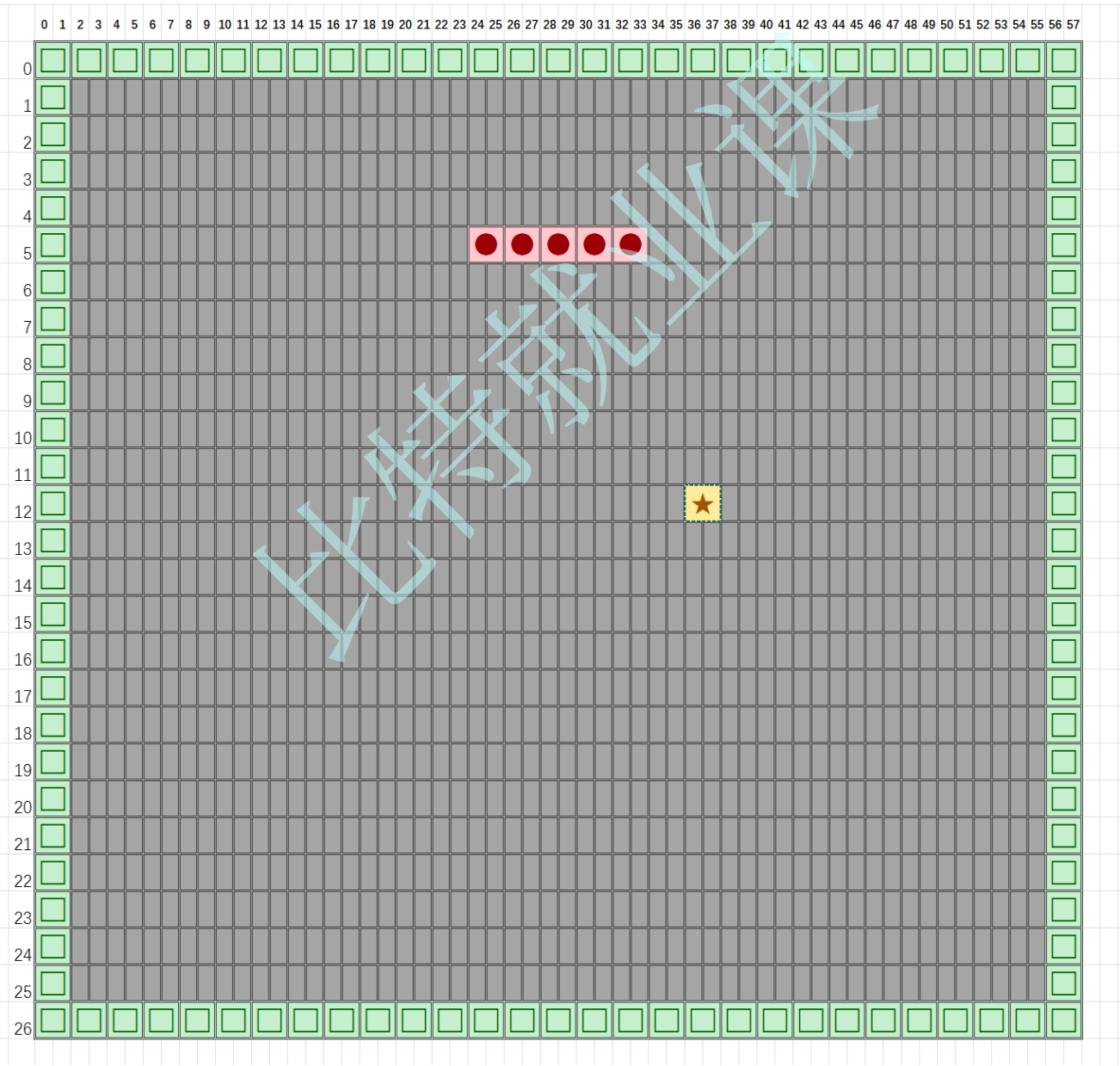
8.3.3.3 NoFood

将下一个节点头插入蛇的身体，并将之前蛇身最后一个节点打印为空格，释放掉蛇身的最后一个节点。

易错点：这里最容易错误的是，释放最后一个结点后，还得将指向在最后一个结点的指针改为NULL，保证蛇尾打印可以正常结束，不会越界访问。

```
1  //pSnakeNode psn 是下一个节点的地址
2  //pSnake ps 维护蛇的指针
3  void NoFood(pSnakeNode psn, pSnake ps)
4  {
5      //头插法
6      psn->next = ps->_pSnake;
7      ps->_pSnake = psn;
8
9      //打印蛇
10     pSnakeNode cur = ps->_pSnake;
```

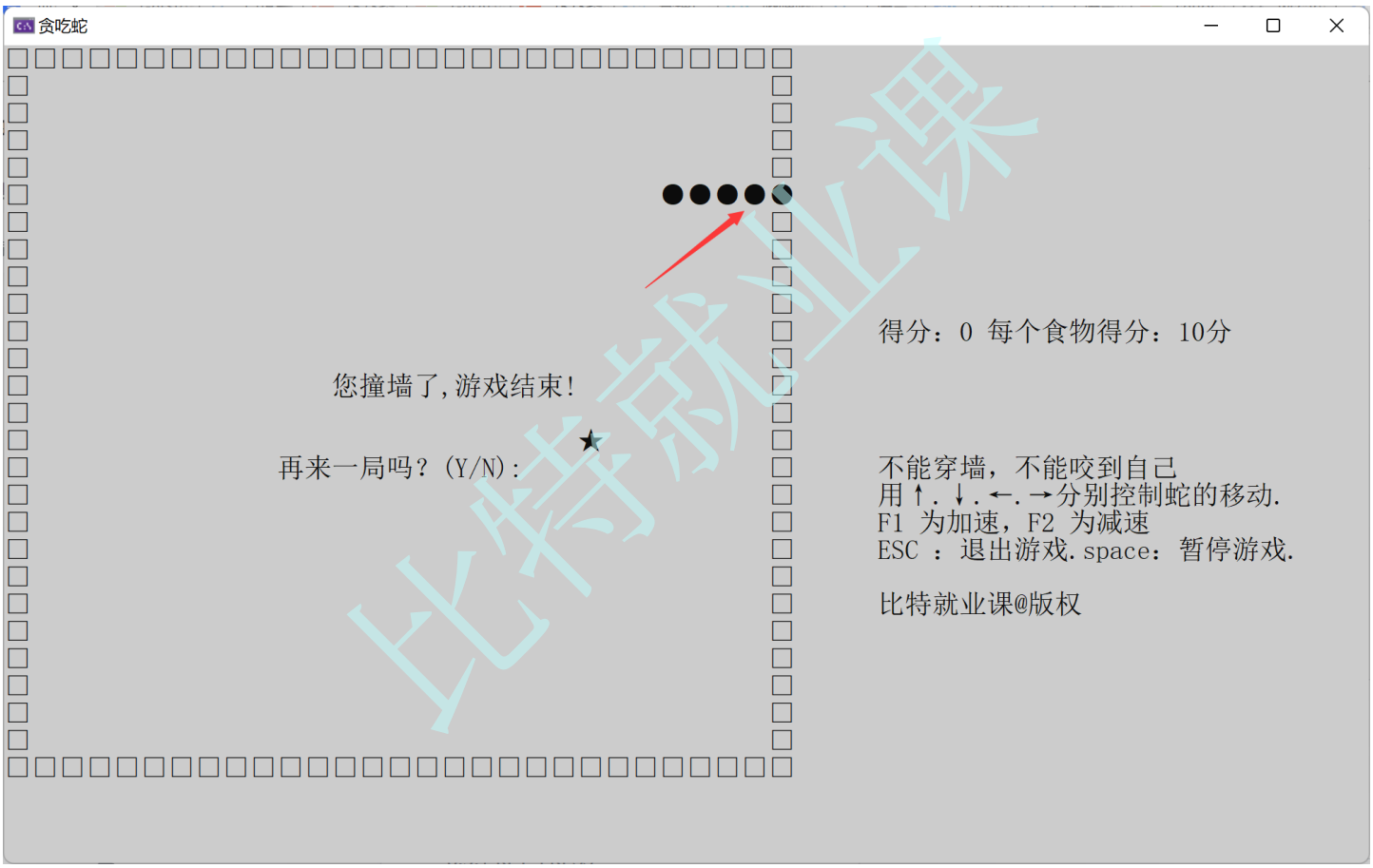
```
    while (cur->next->next)
    {
        SetPos(cur->x, cur->y);
        wprintf(L"%c", BODY);
        cur = cur->next;
    }

    //最后一个位置打印空格，然后释放节点
    SetPos(cur->next->x, cur->next->y);
    printf("  ");
    free(cur->next);
    cur->next = NULL;
}
```



8.3.3.4 KillByWall

判断蛇头的坐标是否和墙的坐标冲突

```c
//pSnake ps 维护蛇的指针
int KillByWall(pSnake ps)
{
    if ((ps->_pSnake->x == 0)
        || (ps->_pSnake->x == 56)
        || (ps->_pSnake->y == 0)
        || (ps->_pSnake->y == 26))
    {
        ps->_Status = KILL_BY_WALL;
        return 1;
    }
    return 0;
}
```



8.3.3.5 KillBySelf

判断蛇头的坐标是否和蛇身体的坐标冲突

```c
//pSnake ps 维护蛇的指针
int KillBySelf(pSnake ps)
{
    pSnakeNode cur = ps->_pSnake->next;
    while (cur)
    {
        if ((ps->_pSnake->x == cur->x)
```

```
8               && (ps->_pSnake->y == cur->y))
9           {
10              ps->_Status = KILL_BY_SELF;
11              return 1;
12          }
13          cur = cur->next;
14      }
15      return 0;
16  }
17
```

8.4 游戏结束

游戏状态不再是OK（游戏继续）的时候，要告知游戏结束的原因，并且释放蛇身节点。

```
1  void GameEnd(pSnake ps)
2  {
3      pSnakeNode cur = ps->_pSnake;
4      SetPos(24, 12);
5      switch (ps->_Status)
6      {
7      case END_NOMAL:
8          printf("您主动退出游戏\n");
9          break;
```

```
    case KILL_BY_SELF:
        printf("您撞上自己了 ,游戏结束!\n");
        break;
    case KILL_BY_WALL:
        printf("您撞墙了,游戏结束!\n");
        break;
    }

    //释放蛇身的节点
    while (cur)
    {
        pSnakeNode del = cur;
        cur = cur->next;
        free(del);
    }
}
```

9. 参考代码

完整代码实现，分3个文件实现

test.cpp

```
#include "Snake.h"
#include <locale.h>

void test()
{
    int ch = 0;
    srand((unsigned int)time(NULL));

    do
    {
        Snake snake = { 0 };
        GameStart(&snake);
        GameRun(&snake);
        GameEnd(&snake);
        SetPos(20, 15);
        printf("再来一局吗? (Y/N):");
        ch = getchar();
        getchar();//清理\n

    } while (ch == 'Y' || ch == 'y');
    SetPos(0, 27);
}
```

```
23
24 int main()
25 {
26     //修改当前地区为本地模式，为了支持中文宽字符的打印
27     setlocale(LC_ALL, "");
28     //测试逻辑
29     test();
30     return 0;
31 }
```

snake.h

```
1  #pragma once
2
3  #include <windows.h>
4  #include <time.h>
5  #include <stdio.h>
6
7  #define KEY_PRESS(VK)  ((GetAsyncKeyState(VK)&0x1) ? 1 : 0)
8
9  //方向
10 enum DIRECTION
11 {
12     UP = 1,
13     DOWN,
14     LEFT,
15     RIGHT
16 };
17
18 //游戏状态
19 enum GAME_STATUS
20 {
21     OK,//正常运行
22     KILL_BY_WALL,//撞墙
23     KILL_BY_SELF,//咬到自己
24     END_NOMAL//正常结束
25 };
26
27 #define WALL L'□'
28 #define BODY L'●'  //★○●◇◆□■
29 #define FOOD L'★'  //★○●◇◆□■
30
31 //蛇的初始位置
```

```c
#define POS_X 24
#define POS_Y 5

//蛇身节点
typedef struct SnakeNode
{
    int x;
    int y;
    struct SnakeNode* next;
}SnakeNode, * pSnakeNode;

typedef struct Snake
{
    pSnakeNode _pSnake;//维护整条蛇的指针
    pSnakeNode _pFood;//维护食物的指针
    enum DIRECTION _Dir;//蛇头的方向默认是向右
    enum GAME_STATUS _Status;//游戏状态
    int _Socre;//当前获得分数
    int _foodWeight;//默认每个食物10分
    int _SleepTime;//每走一步休眠时间
}Snake, * pSnake;



//游戏开始前的初始化
void GameStart(pSnake ps);

//游戏运行过程
void GameRun(pSnake ps);

//游戏结束
void GameEnd(pSnake ps);

//设置光标的坐标
void SetPos(short x, short y);

//欢迎界面
void WelcomeToGame();

//打印帮助信息
void PrintHelpInfo();

//创建地图
void CreateMap();

//初始化蛇
void InitSnake(pSnake ps);
```

```
79
80 //创建食物
81 void CreateFood(pSnake ps);
82
83 //暂停响应
84 void pause();
85
86 //下一个节点是食物
87 int NextIsFood(pSnakeNode psn, pSnake ps);
88
89 //吃食物
90 void EatFood(pSnakeNode psn, pSnake ps);
91
92 //不吃食物
93 void NoFood(pSnakeNode psn, pSnake ps);
94
95 //撞墙检测
96 int KillByWall(pSnake ps);
97
98 //撞自身检测
99 int KillBySelf(pSnake ps);
100
101 //蛇的移动
102 void SnakeMove(pSnake ps);
103
104 //游戏初始化
105 void GameStart(pSnake ps);
106
107 //游戏运行
108 void GameRun(pSnake ps);
109
110 //游戏结束
111 void GameEnd(pSnake ps);
```

snake.cpp

```
1 #include "Snake.h"
2
3 //设置光标的坐标
4 void SetPos(short x, short y)
5 {
6     COORD pos = { x, y };
7     HANDLE hOutput = NULL;
```

```
    //获取标准输出的句柄(用来标识不同设备的数值)
    hOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    //设置标准输出上光标的位置为pos
    SetConsoleCursorPosition(hOutput, pos);
}

void WelcomeToGame()
{
    SetPos(40, 15);
    printf("欢迎来到贪吃蛇小游戏");
    SetPos(40, 25);//让按任意键继续的出现的位置好看点
    system("pause");
    system("cls");
    SetPos(25, 12);
    printf("用 ↑ . ↓ . ← . → 分别控制蛇的移动， F3为加速，F4为减速\n");
    SetPos(25, 13);
    printf("加速将能得到更高的分数。\n");
    SetPos(40, 25);//让按任意键继续的出现的位置好看点
    system("pause");
    system("cls");
}

void CreateMap()
{
    int i = 0;
    //上(0,0)-(56, 0)
    SetPos(0, 0);
    for (i = 0; i < 58; i += 2)
    {
        wprintf(L"%c", WALL);
    }
    //下(0,26)-(56, 26)
    SetPos(0, 26);
    for (i = 0; i < 58; i += 2)
    {
        wprintf(L"%c", WALL);
    }
    //左
    //x是0，y从1开始增长
    for (i = 1; i < 26; i++)
    {
        SetPos(0, i);
        wprintf(L"%c", WALL);
    }
    //x是56，y从1开始增长
    for (i = 1; i < 26; i++)
    {
```

```
55          SetPos(56, i);
56          wprintf(L"%c", WALL);
57      }
58  }
59
60
61  void InitSnake(pSnake ps)
62  {
63      pSnakeNode cur = NULL;
64      int i = 0;
65      //创建蛇身节点，并初始化坐标
66      //头插法
67      for (i = 0; i < 5; i++)
68      {
69          //创建蛇身的节点
70          cur = (pSnakeNode)malloc(sizeof(SnakeNode));
71          if (cur == NULL)
72          {
73              perror("InitSnake()::malloc()");
74              return;
75          }
76          //设置坐标
77          cur->next = NULL;
78          cur->x = POS_X + i * 2;
79          cur->y = POS_Y;
80
81          //头插法
82          if (ps->_pSnake == NULL)
83          {
84              ps->_pSnake = cur;
85          }
86          else
87          {
88              cur->next = ps->_pSnake;
89              ps->_pSnake = cur;
90          }
91      }
92
93      //打印蛇的身体
94      cur = ps->_pSnake;
95      while (cur)
96      {
97          SetPos(cur->x, cur->y);
98          wprintf(L"%c", BODY);
99          cur = cur->next;
100     }
101
```

```
    //初始化贪吃蛇数据
    ps->_SleepTime = 200;
    ps->_Socre = 0;
    ps->_Status = OK;
    ps->_Dir = RIGHT;
    ps->_foodWeight = 10;
}


void CreateFood(pSnake ps)
{
    int x = 0;
    int y = 0;

again:
    //产生的x坐标应该是2的倍数，这样才可能和蛇头坐标对齐。
    do
    {
        x = rand() % 53 + 2;
        y = rand() % 25 + 1;
    } while (x % 2 != 0);

    pSnakeNode cur = ps->_pSnake;//获取指向蛇头的指针
    //食物不能和蛇身冲突
    while (cur)
    {
        if (cur->x == x && cur->y == y)
        {
            goto again;
        }
        cur = cur->next;
    }

    pSnakeNode pFood = (pSnakeNode)malloc(sizeof(SnakeNode)); //创建食物
    if (pFood == NULL)
    {
        perror("CreateFood::malloc()");
        return;
    }
    else
    {
        pFood->x = x;
        pFood->y = y;
        SetPos(pFood->x, pFood->y);
        wprintf(L"%c", FOOD);
        ps->_pFood = pFood;
```

```
149     }
150 }
151
152 void PrintHelpInfo()
153 {
154     //打印提示信息
155     SetPos(64, 15);
156     printf("不能穿墙，不能咬到自己\n");
157     SetPos(64, 16);
158     printf("用↑.↓.←.→分别控制蛇的移动.");
159     SetPos(64, 17);
160     printf("F3 为加速，F4 为减速\n");
161     SetPos(64, 18);
162     printf("ESC ： 退出游戏.space: 暂停游戏.");
163     SetPos(64, 20);
164     printf("比特就业课@版权");
165 }
166
167 void pause()//暂停
168 {
169     while (1)
170     {
171         Sleep(300);
172         if (KEY_PRESS(VK_SPACE))
173         {
174             break;
175         }
176     }
177 }
178
179 //pSnakeNode psn 是下一个节点的地址
180 //pSnake ps 维护蛇的指针
181 int NextIsFood(pSnakeNode psn, pSnake ps)
182 {
183     return (psn->x == ps->_pFood->x) && (psn->y == ps->_pFood->y);
184 }
185
186 //pSnakeNode psn 是下一个节点的地址
187 //pSnake ps 维护蛇的指针
188 void EatFood(pSnakeNode psn, pSnake ps)
189 {
190     //头插法
191     psn->next = ps->_pSnake;
192     ps->_pSnake = psn;
193     pSnakeNode cur = ps->_pSnake;
194     //打印蛇
195     while (cur)
```

```
196     {
197         SetPos(cur->x, cur->y);
198         wprintf(L"%c", BODY);
199         cur = cur->next;
200     }
201     ps->_Socre += ps->_foodWeight;
202 
203     free(ps->_pFood);
204     CreateFood(ps);
205 }
206 
207 //pSnakeNode psn 是下一个节点的地址
208 //pSnake ps 维护蛇的指针
209 void NoFood(pSnakeNode psn, pSnake ps)
210 {
211     //头插法
212     psn->next = ps->_pSnake;
213     ps->_pSnake = psn;
214     pSnakeNode cur = ps->_pSnake;
215     //打印蛇
216     while (cur->next->next)
217     {
218         SetPos(cur->x, cur->y);
219         wprintf(L"%c", BODY);
220         cur = cur->next;
221     }
222 
223     //最后一个位置打印空格，然后释放节点
224     SetPos(cur->next->x, cur->next->y);
225     printf("  ");
226     free(cur->next);
227     cur->next = NULL;
228 }
229 
230 //pSnake ps 维护蛇的指针
231 int KillByWall(pSnake ps)
232 {
233     if ((ps->_pSnake->x == 0)
234         || (ps->_pSnake->x == 56)
235         || (ps->_pSnake->y == 0)
236         || (ps->_pSnake->y == 26))
237     {
238         ps->_Status = KILL_BY_WALL;
239         return 1;
240     }
241     return 0;
242 }
```

```
243
244 //pSnake ps 维护蛇的指针
245 int KillBySelf(pSnake ps)
246 {
247     pSnakeNode cur = ps->_pSnake->next;
248     while (cur)
249     {
250         if ((ps->_pSnake->x == cur->x)
251             && (ps->_pSnake->y == cur->y))
252         {
253             ps->_Status = KILL_BY_SELF;
254             return 1;
255         }
256         cur = cur->next;
257     }
258     return 0;
259 }
260
261
262 void SnakeMove(pSnake ps)
263 {
264     //创建下一个节点
265     pSnakeNode pNextNode = (pSnakeNode)malloc(sizeof(SnakeNode));
266     if (pNextNode == NULL)
267     {
268         perror("SnakeMove()::malloc()");
269         return;
270     }
271     //确定下一个节点的坐标，下一个节点的坐标根据，蛇头的坐标和方向确定
272     switch (ps->_Dir)
273     {
274         case UP:
275         {
276             pNextNode->x = ps->_pSnake->x;
277             pNextNode->y = ps->_pSnake->y - 1;
278         }
279         break;
280         case DOWN:
281         {
282             pNextNode->x = ps->_pSnake->x;
283             pNextNode->y = ps->_pSnake->y + 1;
284         }
285         break;
286         case LEFT:
287         {
288             pNextNode->x = ps->_pSnake->x - 2;
289             pNextNode->y = ps->_pSnake->y;
```

```
290         }
291         break;
292         case RIGHT:
293         {
294             pNextNode->x = ps->_pSnake->x + 2;
295             pNextNode->y = ps->_pSnake->y;
296         }
297         break;
298     }
299 
300     //如果下一个位置就是食物
301     if (NextIsFood(pNextNode, ps))
302     {
303         EatFood(pNextNode, ps);
304     }
305     else//如果没有食物
306     {
307         NoFood(pNextNode, ps);
308     }
309 
310     KillByWall(ps);
311     KillBySelf(ps);
312 }
313 
314 
315 
316 void GameStart(pSnake ps)
317 {
318     //设置控制台窗口的大小，30行，100列
319     //mode 为DOS命令
320     system("mode con cols=100 lines=30");
321     //设置cmd窗口名称
322     system("title 贪吃蛇");
323 
324     //获取标准输出的句柄(用来标识不同设备的数值)
325     HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE);
326 
327     //影藏光标操作
328     CONSOLE_CURSOR_INFO CursorInfo;
329     GetConsoleCursorInfo(hOutput, &CursorInfo);//获取控制台光标信息
330     CursorInfo.bVisible = false; //隐藏控制台光标
331     SetConsoleCursorInfo(hOutput, &CursorInfo);//设置控制台光标状态
332 
333     //打印欢迎界面
334     WelcomeToGame();
335     //打印地图
336     CreateMap();
```

```
337     //初始化蛇
338     InitSnake(ps);
339     //创造第一个食物
340     CreateFood(ps);
341 }
342
343
344 void GameRun(pSnake ps)
345 {
346     //打印右侧帮助信息
347     PrintHelpInfo();
348     do
349     {
350         SetPos(64, 10);
351         printf("得分：%d ", ps->_Socre);
352         printf("每个食物得分：%d分", ps->_foodWeight);
353         if (KEY_PRESS(VK_UP) && ps->_Dir != DOWN)
354         {
355             ps->_Dir = UP;
356         }
357         else if (KEY_PRESS(VK_DOWN) && ps->_Dir != UP)
358         {
359             ps->_Dir = DOWN;
360         }
361         else if (KEY_PRESS(VK_LEFT) && ps->_Dir != RIGHT)
362         {
363             ps->_Dir = LEFT;
364         }
365         else if (KEY_PRESS(VK_RIGHT) && ps->_Dir != LEFT)
366         {
367             ps->_Dir = RIGHT;
368         }
369         else if (KEY_PRESS(VK_SPACE))
370         {
371             pause();
372         }
373         else if (KEY_PRESS(VK_ESCAPE))
374         {
375             ps->_Status = END_NOMAL;
376             break;
377         }
378         else if (KEY_PRESS(VK_F3))
379         {
380             if (ps->_SleepTime >= 50)
381             {
382                 ps->_SleepTime -= 30;
383                 ps->_foodWeight += 2;
```

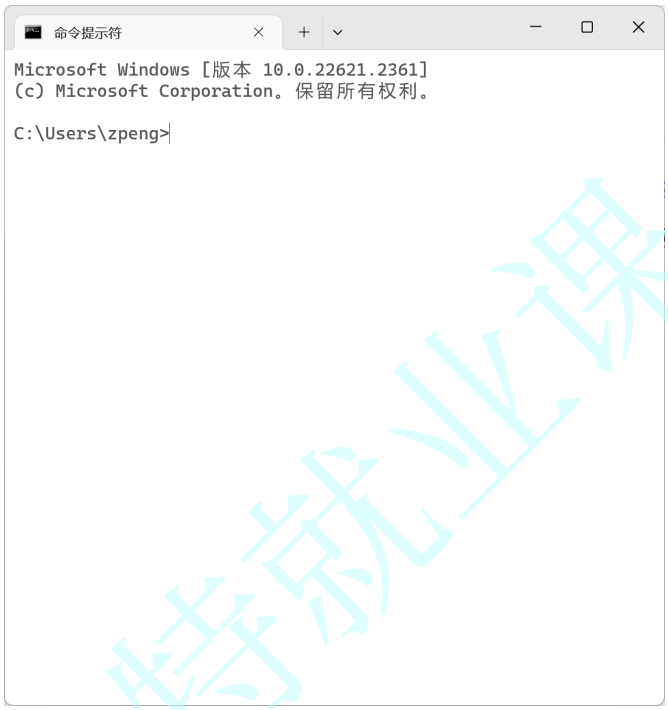
```
            }
        }
        else if (KEY_PRESS(VK_F4))
        {
            if (ps->_SleepTime < 350)
            {
                ps->_SleepTime += 30;
                ps->_foodWeight -= 2;
                if (ps->_SleepTime == 350)
                {
                    ps->_foodWeight = 1;
                }
            }
        }
        //蛇每次一定之间要休眠的时间，时间短，蛇移动速度就快
        Sleep(ps->_SleepTime);
        SnakeMove(ps);

    } while (ps->_Status == OK);
}

void GameEnd(pSnake ps)
{
    pSnakeNode cur = ps->_pSnake;
    SetPos(24, 12);
    switch (ps->_Status)
    {
    case END_NOMAL:
        printf("您主动退出游戏\n");
        break;
    case KILL_BY_SELF:
        printf("您撞上自己了 ,游戏结束!\n");
        break;
    case KILL_BY_WALL:
        printf("您撞墙了,游戏结束!\n");
        break;
    }

    //释放蛇身的节点
    while (cur)
    {
        pSnakeNode del = cur;
        cur = cur->next;
        free(del);
    }
}
```

代码仓库：https://gitee.com/bitpg/snake

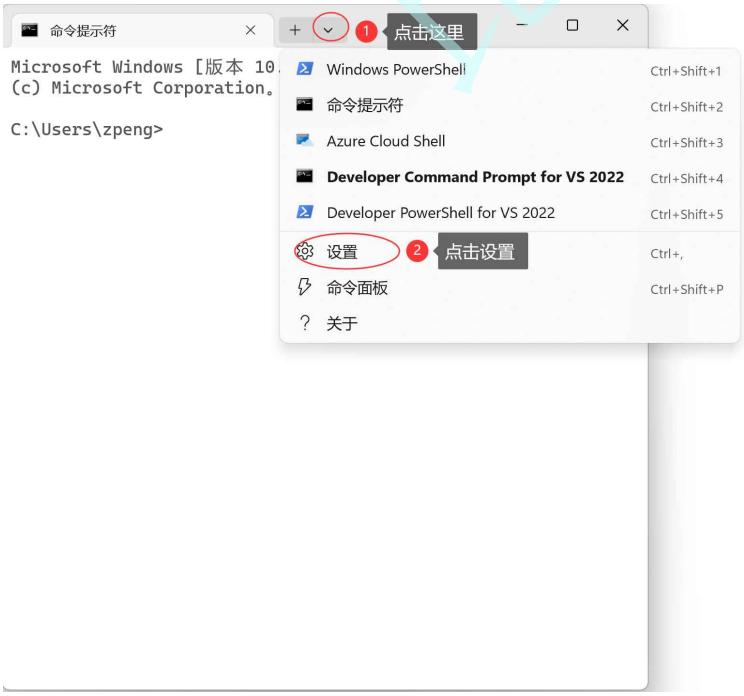
参考：汉字字符集编码查询;中文字符集编码:GB2312、BIG5、GBK、GB18030、Unicode

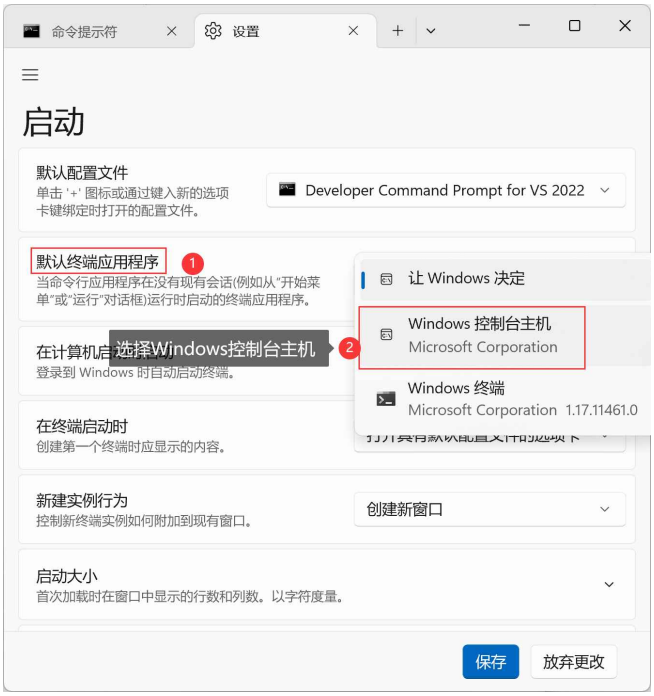
10. 控制台设置（补充）

如果同学Win11系统的控制台窗口是这样显示，可以调整一下



调整方式：





保存后，重新打开cmd就行