

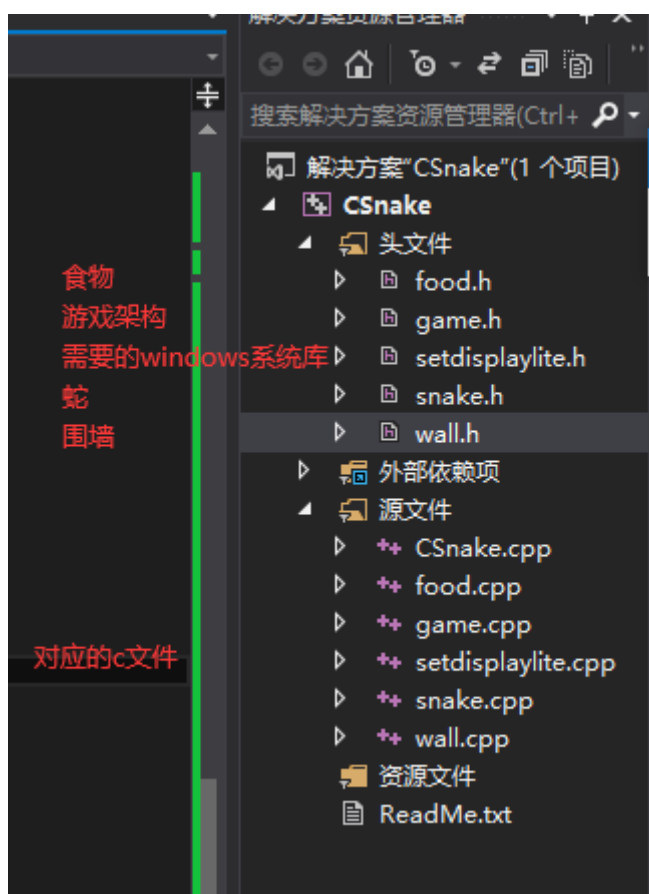
C语言贪吃蛇

一、贪吃蛇的元素

- 围墙（游戏区域）
- 蛇
- 食物
- 玩法（游戏架构）

为了方便调试，以及功能添加与查错，最好将工程按元素相应封装，这样显得项目更有逻辑性和层次感

因此先创建一下c文件及其对应的头文件



二、贪吃蛇的实现

1) 围墙（游戏）

为了方便修改区域，我们可以将游戏区域定义为宏

```
#define GAME_ROWS 20  
#define GAME_COLS 20
```

墙的初始化代码如下

```
5
6 void WallInit(void)//围墙初始化
7 {
8     //在边缘画■建立围墙
9     for (size_t nRow = 0; nRow < GAME_ROWS; nRow++)
10     {
11         for (size_t nCol = 0; nCol < GAME_COLS; nCol++)
12         {
13             if (nRow == 0 ||
14                 nRow == GAME_ROWS - 1 ||
15                 nCol == 0 ||
16                 nCol == GAME_COLS - 1)
17             {
18                 MoveTo(nRow, nCol);
19                 printf("■");
20             }
21         }
22     }
23 }
```

在这里要讲一下

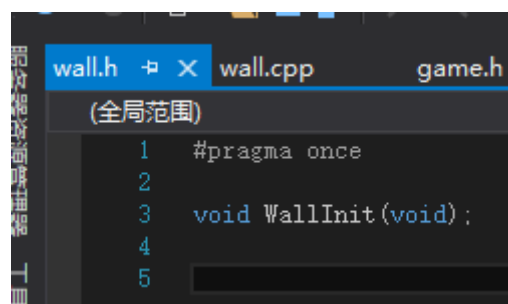
- 1 MoveTo(nRow, nCol); //这是封装好的windows系统库下的一个函数
- 2 //作用是将鼠标移动到指定坐标点(nRow, nCol)

关于其内容不需要理解（其中用的都是windows系统库下的函数） 只要知道怎么用就行了

```
10
11 /*将光标移动到特定的行和列*/
12 void MoveTo(int nRow, int nCol)
13 {
14     CONSOLE_CURSOR_INFO cii;
15     cii.dwSize = 1;
16     cii.bVisible = FALSE;
17     SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &cii);
18     COORD loc;
19     loc.X = nCol * (WIDTH_UNIT);
20     loc.Y = nRow;
21     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), loc);
22 }
```

将对应函数包含至头文件

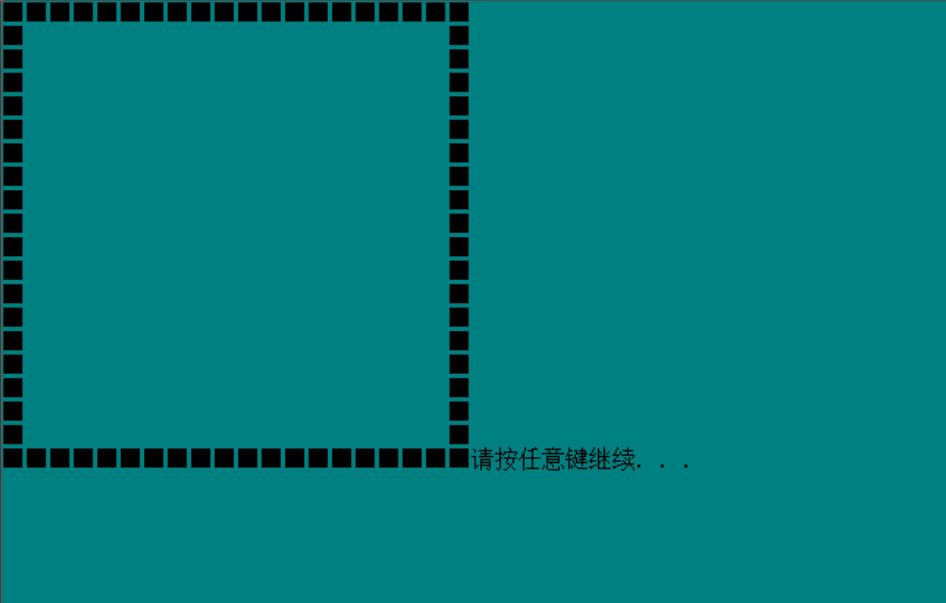
（头文件的封装，与函数调用过程后面不再赘述啦，只展示各阶段效果图与最终的c文件内容）



```
wall.h x wall.cpp game.h
(全局范围)
1 #pragma once
2
3 void WallInit(void);
4
5
```

经主函数调用后效果为

```
1 // Snake.cpp : 定义控制台应用程序的入口点。
2 //
3 #include <stdio.h>
4 #include "wall.h"
5
6
7 int main(int argc, char* argv[])
8 {
9     WallInit();
10    return 0;
11 }
```



到这里墙的初始化就完成了

2) 蛇

蛇的孕育（准备工作）

蛇的几个元素

- 控制蛇的走向的**蛇头**及储存食物（即吃到食物就会增加一节）的 **蛇身**位置
- 蛇头与蛇身及蛇身每个节点之间的链接关系【因此为了使蛇身体健康（各节点链接紧密），在蛇的构造方面，将会用到链表】

为了使编程方便、可读性更强（你好我好大家好）

在这里用一个结构体来封装蛇

```
4 typedef struct tagSnake{//蛇身结构体链表
5     int S_nRow = 0;//蛇横坐标
6     int S_nCol = 0;//蛇纵坐标
7     struct tagSnake* next;//保存下一个节蛇身
8 } Snake;
```

定义一个枚举用以保存蛇的头的朝向（影响着蛇下一步的走向）

```

11  enum nDirection//朝向枚举
12  {
13      UP,
14      DOWN,
15      LEFT,
16      RIGHT
17  };
18

```

然后定义好重要的几个元素（由于会活跃于蛇的一生，所以都用全局变量）

- 蛇头（蛇的每一节都是由头一节一节链接下去的）
- 蛇的长度（遍历蛇的时候会有大用，同时也可以做计分用）
- 蛇头的朝向（用于之后蛇的运动，以及蛇身体的生长方向）

```

5
6  Snake* g_pSHead = NULL;//蛇头
7  int g_CountNode = 1;//记录当前节点数
8
9  nDirection g_SDir;//存储蛇头朝向
10

```

蛇的诞生（初始化）

完成了上面的孕育工作后，就可以让蛇出生了

- 先随机储蛇头的朝向，用于之后蛇的运动，以及蛇身体的生长方向（自然是不能出现蛇身长在蛇脸上的情况啦！）
- 接着就是随机蛇头出现的位置

那么蛇的初始化就如下了

```

11 void SnakeInit(void)//初始化蛇
12 {
13     //随机蛇头的朝向
14     switch (rand() % 4)
15     {
16     case UP:
17         g_SDir = UP;
18         break;
19     case DOWN:
20         g_SDir = DOWN;
21         break;
22     case LEFT:
23         g_SDir = LEFT;
24         break;
25     case RIGHT:
26         g_SDir = RIGHT;
27         break;
28     default:
29         break;
30     }
31
32     //创建蛇头
33     g_pSHead = (Snake*)malloc(sizeof(Snake)); //申请一片放置为蛇头相关信息的内存
34     //check
35     if (g_pSHead == NULL)
36     {
37         printf("蛇头初始化失败");
38         return;
39     }
40     memset(g_pSHead, 0, sizeof(Snake)); //清理杂乱的内存赋予0
41
42     //随机生成蛇头的位置
43     g_pSHead->S_nRow = rand() % (GAME_ROWS - 8) + 4; //蛇头的横坐标
44     g_pSHead->S_nCol = rand() % (GAME_COLS - 8) + 4; //蛇头的纵坐标
45
46     g_pSHead->next = NULL; //约定蛇的末端是空
47
48
49     AddNode();
50     AddNode();
51 }

```

你可能有点迷惑最后两个

```
1 AddNode();
```

是什么呢，这是因为出生的小蛇肯定不会只有头呀！（想想就恐怖）

所以在出生的时候还伴随着两节（多少节自定）身体，而这个函数就是蛇长大的奥秘（为蛇增加一个节点）

- 增长的方向：
 - 只有蛇头时：于蛇头方向相反
 - 有两节及以上节点时：最后两节节点（两点确定一条直线）的直线上，最后一个节点的后面
- 增长位置：在蛇的尾部

```

73 void AddNode(void)//增加一个节点
74 {
75     if (g_CountNode == 1)//若只有一个节点（即只有蛇头）
76     {
77         Snake* pNewSNode = NULL;//创建新节点
78         pNewSNode = (Snake*)malloc(sizeof(Snake)); //为新节点申请一片空间存放相关信息
79         if (pNewSNode == NULL)
80         {
81             printf("节点创建失败");
82             return;
83         }
84         memset(pNewSNode, 0, sizeof(Snake));
85
86         int nNewRow = g_pSHead->S_nRow;
87         int nNewCol = g_pSHead->S_nCol;
88         switch (g_SDir)
89         {
90             case UP:
91                 nNewRow += 1;
92                 break;
93             case DOWN:
94                 nNewRow -= 1;
95                 break;
96             case LEFT:
97                 nNewCol += 1;
98                 break;
99             case RIGHT:
100                 nNewCol -= 1;
101                 break;
102             default:
103                 break;
104         }
105         pNewSNode->S_nRow = nNewRow;
106         pNewSNode->S_nCol = nNewCol;
107         g_pSHead->next = pNewSNode;
108         pNewSNode->next = NULL;//约定链表最后以NULL结束
109         g_CountNode++; //节点数加一
110     }

```

```

111 else//若大于一个节点
112 {
113     Snake* pTail = BodyFind(g_CountNode);//存储倒数第一个节点
114     Snake* pTail2 = BodyFind(g_CountNode - 1);//存储倒数第二个节点 下面讲
115
116     Snake* pNewSNode = NULL;//创建新节点
117     pNewSNode = (Snake*)malloc(sizeof(Snake));
118     if (pNewSNode == NULL)
119     {
120         printf("节点创建失败");
121         return;
122     }
123     memset(pNewSNode, 0, sizeof(Snake));
124
125     int nNewRow = pTail->S_nRow;
126     int nNewCol = pTail->S_nCol;
127     //根据最后两个节点的相对关系，进行判断新节点的位置
128     if (pTail->S_nRow == pTail2->S_nRow
129         && pTail->S_nCol > pTail2->S_nCol)
130     {
131         nNewCol += 1;
132     }
133     else if (pTail->S_nRow == pTail2->S_nRow
134         && pTail->S_nCol < pTail2->S_nCol)
135     {
136         nNewCol -= 1;
137     }
138     else if (pTail->S_nRow > pTail2->S_nRow
139         && pTail->S_nCol == pTail2->S_nCol)
140     {
141         nNewRow += 1;
142     }
143     else if (pTail->S_nRow < pTail2->S_nRow
144         && pTail->S_nCol == pTail2->S_nCol)
145     {
146         nNewRow -= 1;
147     }
148
149     pNewSNode->S_nRow = nNewRow;
150     pNewSNode->S_nCol = nNewCol;

```

```

151     pTail->next = pNewSNode;
152     pNewSNode->next = NULL;
153
154     g_CountNode++; //节点数加一
155 }
156 }
157

```

其中出现了个没见过的函数

```
1 BodyFind(g_CountNode);
```

这是为了以后查找蛇的第n个节点的时候方便，所以将其封装成了一个函数

如下

```

157
158 //寻找第几个节点，并返回那个节点的地址
159 //其中参数 NodeNum 就是指定的第几个节点（这里正常取值从1开始到g_CountNode结束）
160 Snake* BodyFind(int NodeNum)
161 {
162     int nCount = 0;
163     Snake* pNode = g_pSHead;
164     if (NodeNum >= 1 && NodeNum <= g_CountNode) //为防止传参错误限定一下区间
165     {
166         while (NULL != pNode)
167         {
168             nCount++;
169             if (NodeNum == nCount)
170                 break;
171             pNode = pNode->next;
172         }
173         return pNode; //找到该节点后返回该节点
174     }
175     else
176         printf("没有找到该段身体呢");
177     return NULL;
178 }
179

```

好了到了这步，蛇就可以出生了

把隐身的蛇显示出来

```

52
53 void ShowSnake() //显示蛇
54 {
55     Snake *pNode = g_pSHead;
56     for (int i = 0; i < g_CountNode; i++)
57     {
58         if (i == 0) //如果是蛇头则在蛇头位置打印■
59         {
60             MoveTo(pNode->S_nRow, pNode->S_nCol);
61             printf("■");
62         }
63         else //如果是蛇身则挨个在蛇身位置打印◆
64         {
65             pNode = pNode->next;
66             MoveTo(pNode->S_nRow, pNode->S_nCol);
67             printf("◆");
68         }
69     }
70 }
71

```

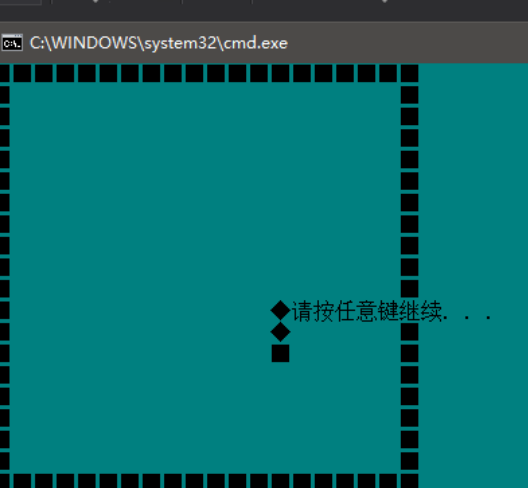
```
snake.cpp : 定义控制台应用程序的入口点。
//
#include <stdio.h>
#include "time.h"
#include "windows.h"
#include "wall.h"
#include "snake.h"

int main(int argc, char* argv[])
{
    srand((unsigned)time(NULL)); //产生随机数种子
    WallInit();
    SnakeInit();
    ShowSnake();
    return 0;
}
```



```
snake.cpp : 定义控制台应用程序的入口点。
//
#include <stdio.h>
#include "time.h"
#include "windows.h"
#include "wall.h"
#include "snake.h"

int main(int argc, char* argv[])
{
    srand((unsigned)time(NULL)); //产生随机数种子
    WallInit();
    SnakeInit();
    ShowSnake();
    return 0;
}
```



蛇的蹒跚学步（移动）

蛇的移动

- 蛇头会根据方向：进行横纵坐标的改变
 - 向上：横坐标 -1
 - 向下：横坐标+1
 - 向左：纵坐标 -1
 - 向右：纵坐标+1
- 蛇身会一个覆盖一个：这里需要讲一下思想
 - 运动剖析：由于蛇身实质上是在重复蛇头的运行轨迹，所以每一节蛇身移动后都是跑到其前一节蛇身原来的位置上去
 - 覆盖的方式：如果从前向后覆盖（即 第二节 = 第一节，第三节=第二节....）最终会出现问题，身头重合为一点。因此应该采用从后往前覆盖（倒数第一节 = 倒数第二节，倒数第二节 = 倒数第三节）

理解了以上思想就可以实现蛇的移动了

如下


```

217 void MoveSnake(void)//移动蛇
218 {
219     for (int i = g_CountNode; i >= 2; i--)//身体从后往前一次覆盖
220     {
221         BodyFind(i)->S_nRow = BodyFind(i - 1)->S_nRow;
222         BodyFind(i)->S_nCol = BodyFind(i - 1)->S_nCol;
223     }
224
225     switch (g_SDir)//头部坐标由朝向决定
226     {
227     case UP:
228         g_pSHead->S_nRow -= 1;
229         break;
230     case DOWN:
231         g_pSHead->S_nRow += 1;
232         break;
233     case LEFT:
234         g_pSHead->S_nCol -= 1;
235         break;
236     case RIGHT:
237         g_pSHead->S_nCol += 1;
238         break;
239     default:
240         break;
241     }
242 }
243

```

蛇的理想（移动方向）

众所周知，蛇不是一个耿直僵硬的生物，在行走上有着“蛇皮走位”美誉的它，学会走路后自然是希望可以不老实的自由乱拐啦

因此需要能够改变移动方向的能力：

```

180 //改蛇头朝向
181 //参数 chDirection 为从玩家哪里获取的方向键值
182 // w
183 // a s d
184 void ChangeDir(char chDirection)
185 {
186     if ((chDirection == 'a' && g_SDir == RIGHT) || //方向相反无法改变方向
187         (chDirection == 'w' && g_SDir == DOWN) ||
188         (chDirection == 's' && g_SDir == UP) ||
189         (chDirection == 'd' && g_SDir == LEFT))
190     {
191         return;
192     }
193
194     switch (chDirection)
195     {
196     case 'w':
197         g_SDir = UP;
198         break;
199     case 's':
200         g_SDir = DOWN;
201         break;
202     case 'a':
203         g_SDir = LEFT;
204         break;
205     case 'd':
206         g_SDir = RIGHT;
207         break;
208     default:
209         break;
210     }
211 }
212

```

3) 食物

同样的，为了可读性先把食物信息封装为结构体，

```
3  typedef struct tagFood{
4      int F_nRow = 0;
5      int F_nCol = 0;
6  }Food;
```

并且由于食物要在蛇的一生中都要持续诱惑蛇，所以定义一个全局变量存放食物的信息

```
5
6  Food* g_pFoodPlace = NULL;
```

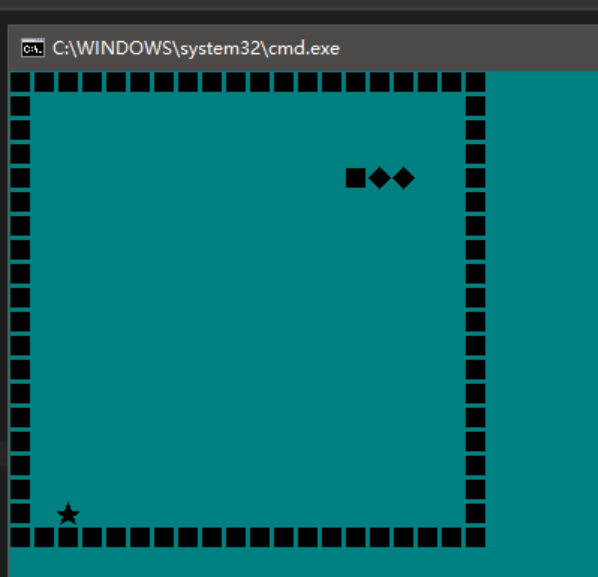
关于食物就没有什么复杂的东西了哦

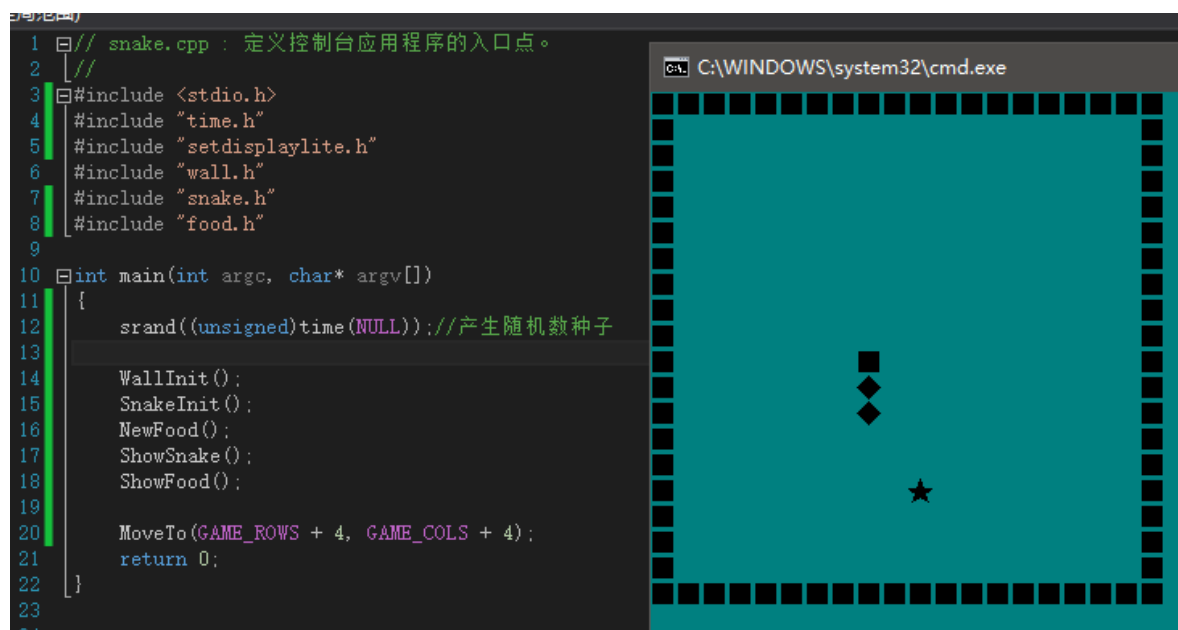
只有简单的两个函数，就可以去诱惑蛇、祸害蛇了

```
8 void NewFood(void)//新食物的产出
9 {
10     g_pFoodPlace = (Food*)malloc(sizeof(Food)); //为存储食物相关信息申请一段空间
11     if (NULL == g_pFoodPlace)
12     {
13         printf("食物申请失败");
14         return;
15     }
16     memset(g_pFoodPlace, 0, sizeof(Food));
17
18     RSTFLAGE:
19     Snake* pNode = g_pSHead;
20     //随机产出食物的位置
21     int nRow = rand() % (GAME_ROWS - 2) + 1;
22     int nCol = rand() % (GAME_COLS - 2) + 1;
23
24     while (NULL != pNode)
25     {
26         if (pNode->S_nCol == nCol && pNode->S_nRow == nRow) //判断新食物是否于蛇体重合，
27         {
28             goto RSTFLAGE; //若重合则重新产出
29         }
30         pNode = pNode->next;
31     }
32     g_pFoodPlace->F_nRow = nRow;
33     g_pFoodPlace->F_nCol = nCol;
34 }
35
36 void ShowFood(void)
37 {
38     MoveTo(g_pFoodPlace->F_nRow, g_pFoodPlace->F_nCol);
39     printf("★");
40 }
```

三要素都有后就是这个样子

```
1 // snake.cpp : 定义控制台应用程序的入口点。
2 //
3 #include <stdio.h>
4 #include "time.h"
5 #include "setdisplaylite.h"
6 #include "wall.h"
7 #include "snake.h"
8 #include "food.h"
9
10 int main(int argc, char* argv[])
11 {
12     srand((unsigned)time(NULL)); //产生随机数种子
13
14     WallInit();
15     SnakeInit();
16     NewFood();
17     ShowSnake();
18     ShowFood();
19
20     MoveTo(GAME_ROWS + 4, GAME_COLS + 4);
21     return 0;
22 }
23
24
```





4) 玩法 (游戏架构)

游戏初始化

三要素全齐了，剩下的就是将这些要素拼装起来了

首先将游戏初始化

```
9
10 void GameInit(void) //游戏初始化
11 {
12     g_CountNode = 1;
13     WallInit(); //初始化墙
14     SnakeInit(); //初始化蛇
15     NewFood(); //初始化食物
16     ShowGame(); //将初始化完成的结果可视化
17 }
18
19 void ShowGame(void) //显示游戏
20 {
21     system("cls"); //清屏
22     WallInit();
23     ShowSnake();
24     ShowFood();
25     ShowScore(); //分数显示
26     MoveTo(GAME_ROWS + 4, GAME_COLS + 4);
27 }
28
```

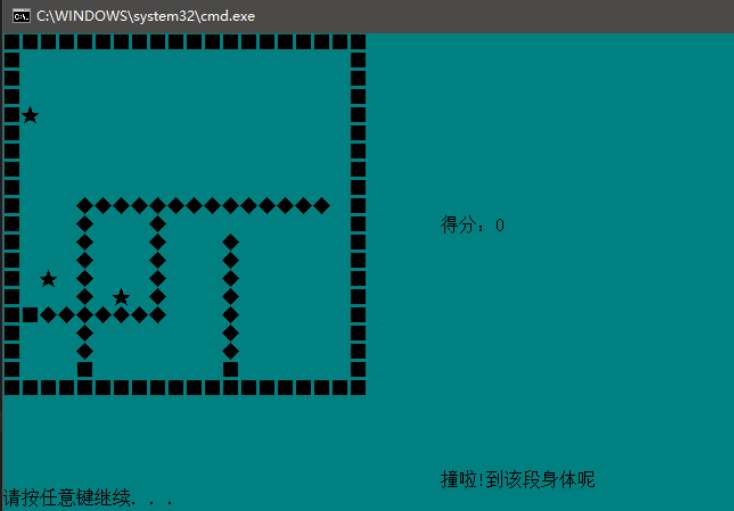
这里的分数显示就不赘述了，很简单的一个函数哦，可以自己思考一下

这里讲一下：

```
1 system("cls");
2 //清屏是“windows.h”下的一个函数，由于在每次食物刷新及移动后，需要清除之前显示的东西
3 //以保证不会出现打印残留的现象
```

如果不清屏就会变成鬼画图：

```
10 void GameInit(void)//游戏初始化
11 {
12     g_CountNode = 1;
13     WallInit();//初始化墙
14     SnakeInit();//初始化蛇
15     NewFood();//初始化食物
16     ShowGame();//将初始化完成的结果可视化
17 }
18
19 void ShowGame(void)//显示游戏
20 {
21     //system("cls");//清屏
22     WallInit();
23     ShowSnake();
24     ShowFood();
25     ShowScore();//分数显示
26     MoveTo(GAME_ROWS + 4, GAME_COLS + 4);
27 }
28
29
30
31
32
33
34
35
36
37
38
39
40
```



嗨起来

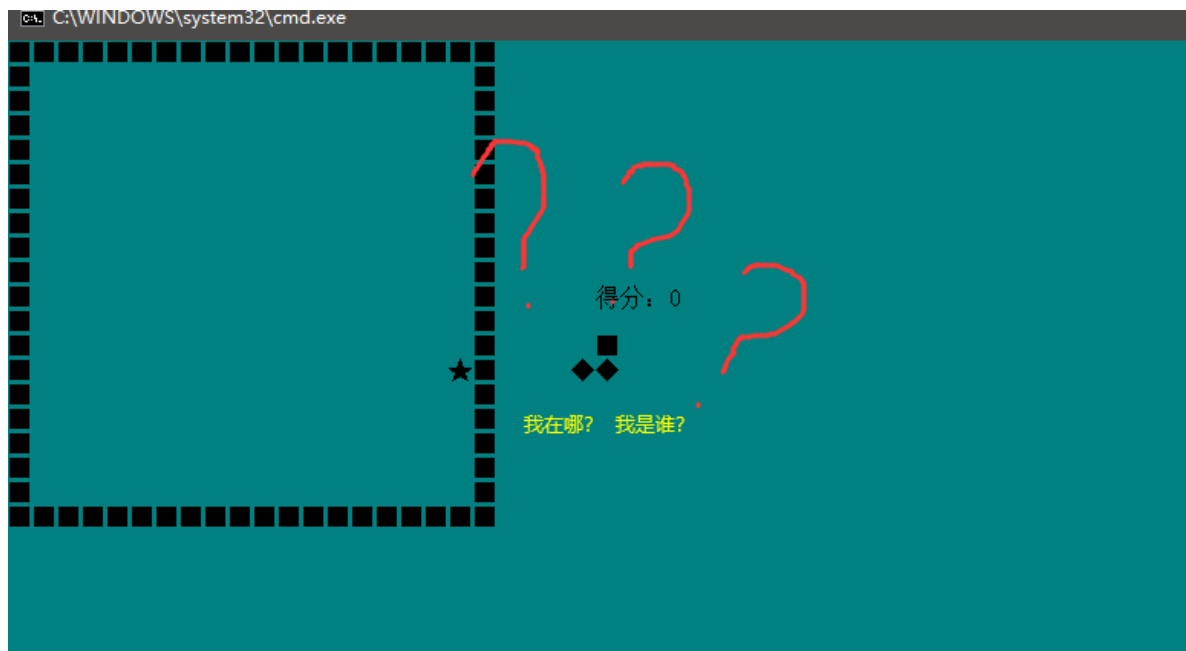
元素都有了那么，咱玩起来！

- 要有按键捕获控制蛇行
- 要有时间捕获，让蛇自动前行

```
80 //游戏运行
81 void Run(void)
82 {
83
84     clock_t clkStart = clock(); //初始化记录系统当前时间为开始时间
85     clock_t clkEnd = clock(); //初始化记录系统当前时间为结束时间
86     char chInput = 0;
87     while (1)
88     {
89         clkEnd = clock(); //记录当前系统时间为结束时间
90         if (clkEnd - clkStart > 500) //结束时间与开始时间差值若大于200ms
91         {
92             if (_kbhit() != 0) //windows系统库下检测是否有按键按下的函数
93             {
94                 chInput = _getch(); //windows系统库下获取按键值的函数
95                 ChangeDir(chInput); //改变蛇头朝向
96                 ShowGame();
97             }
98             MoveSnake(); //移动蛇
99
100
101             ShowGame();
102             clkStart = clkEnd; // 让时间再次相等
103         }
104     }
105 }
106
```

好像很简单 可爱的贪吃蛇就结束了

然而



这样自由而不贪吃的小蛇不是我们认识的贪吃蛇!

这是因为三个元素之间的联系还没正真的构建起来 (约定与规则)

填补规则

- 赋予 墙 以威严 (碰撞检测): 我是游戏范围的代言人, 你 (蛇) 撞了我, 就要被打回娘胎, 不允许你到外面玩!
- 赋予 蛇 以成长欲望 (吃食物成长): 我是主角, 我碰了你 (食物), 你就要归我所有, 成为我的一部分
- 限定 蛇 不能穿过自己的身体 (自食检测): 本事同根生相煎何太急 (蛇头与蛇身), 再饿也不能吃自己, 否则也是要回娘胎的

```

30 int IsCanMove(void)//碰撞检测
31 {
32     if (g_pSHead->S_nRow == 0 ||
33         g_pSHead->S_nRow == GAME_ROWS - 1 ||
34         g_pSHead->S_mCol == 0 ||
35         g_pSHead->S_mCol == GAME_COLS - 1 ||
36         IsEatSelf()) //判断蛇头是否和墙重合或是与身体重合
37     {
38         return 0;//撞了
39     }
40     return 1;//没撞
41 }
42
43
44 void IsEatFood(void)//检测是否吃到食物
45 {
46     if (g_pSHead->S_nRow == g_pFoodPlace->F_nRow &&
47         g_pSHead->S_mCol == g_pFoodPlace->F_mCol) //判断蛇头是否与食物
48     {
49         AddNode(); //重合则增加一个节点
50         free(g_pFoodPlace); //释放存储当前食物的内存空间
51         NewFood(); //产生新的食物
52     }
53 }
54
55 int IsEatSelf(void)//判断是否自食
56 {
57     if (g_CountNode >= 5)
58     {
59         Snake* pNode = BodyFind(5); //总长5个以内不存在自食，所以从第五个开始
60         while (pNode != NULL) //遍历整条蛇，判断是不是自己吃了自己 即头部坐标与身体坐标是否有重合
61         {
62             if (g_pSHead->S_nRow == pNode->S_nRow &&
63                 g_pSHead->S_mCol == pNode->S_mCol)
64             {
65                 return 1;
66             }
67             pNode = pNode->next;
68         }
69     }
70     return 0;
71 }

```

真·嗨起来

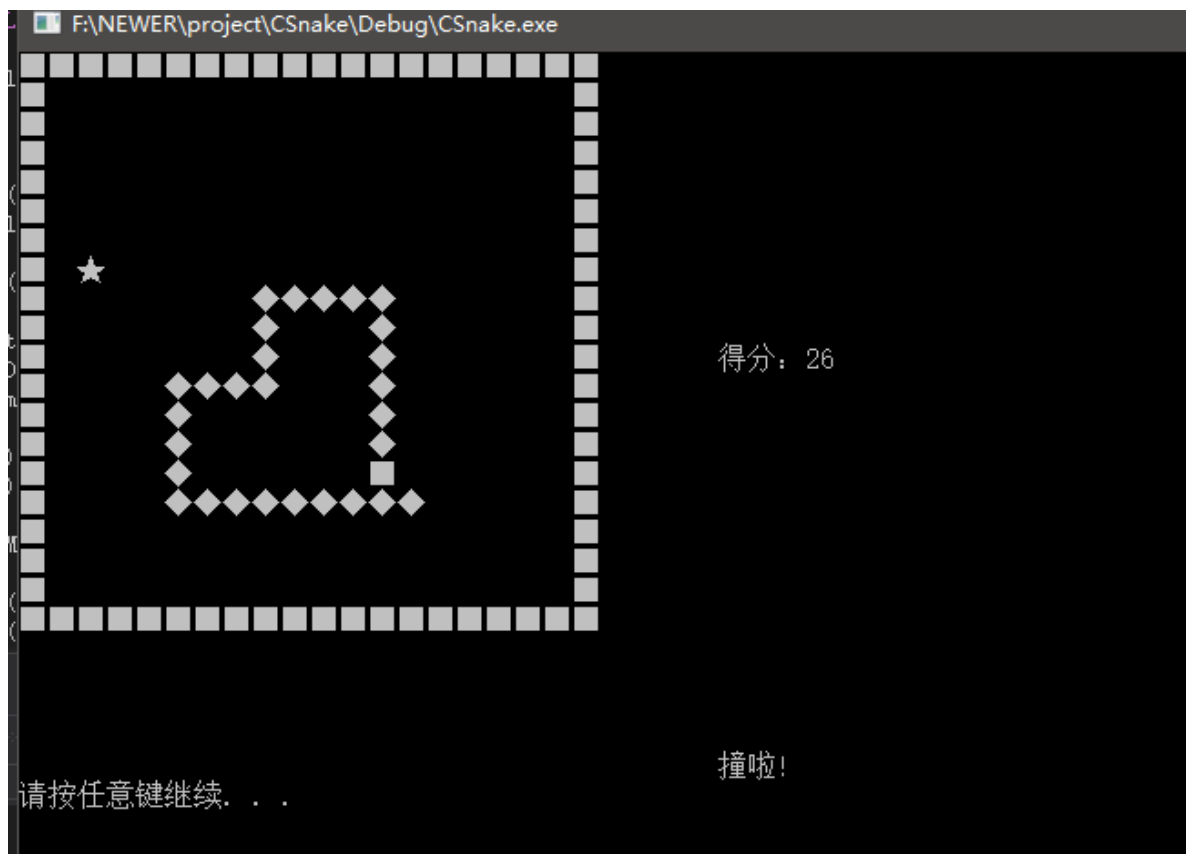
现在规则都以及填补了，剩下的就是整合到游戏运行中去了

```

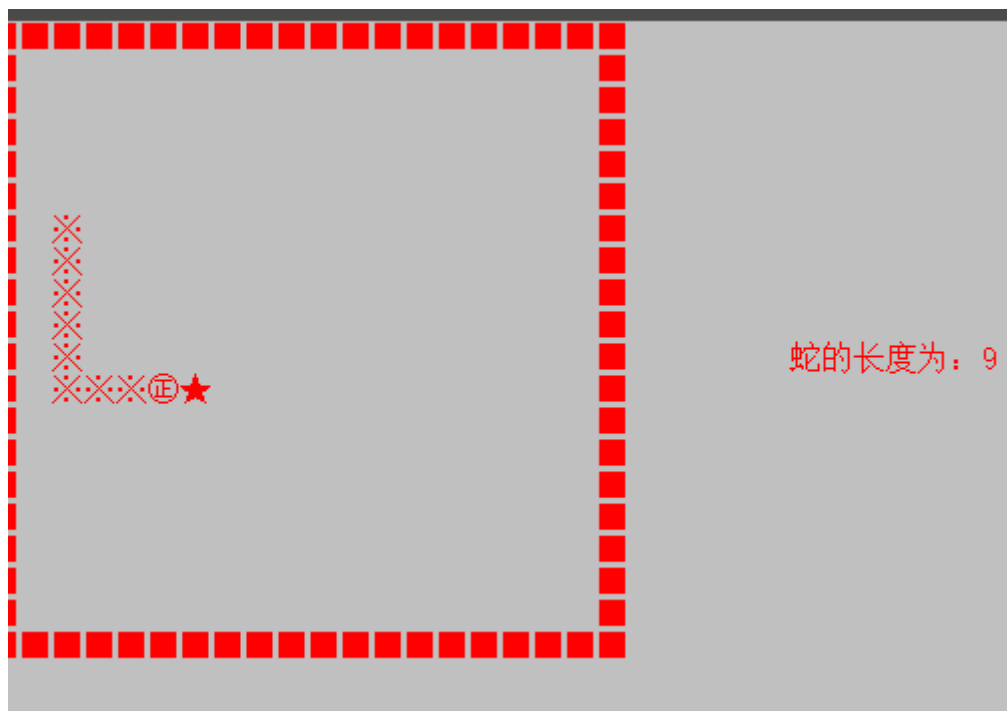
82 //游戏运行
83
84 void Run(void)
85 {
86     Snake *pTail = NULL;
87     clock_t clkStart = clock(); //初始化记录系统当前时间为开始时间
88     clock_t clkEnd = clock(); //初始化记录系统当前时间为结束时间
89     char chInput = 0;
90     while (1)
91     {
92         clkEnd = clock(); //记录当前系统时间为结束时间
93         if (clkEnd - clkStart > 500) //结束时间与开始时间差值若大于200ms
94         {
95             if (_kbhit() != 0) //windows系统库下检测是否有按键按下的函数
96             {
97                 chInput = _getch(); //windows系统库下获取按键值的函数
98                 ChangeDir(chInput); //改变蛇头朝向
99                 ShowGame();
100             }
101             MoveSnake(); //移动蛇
102             IsEatFood();
103
104             if (!IsCanMove())
105             {
106                 printf("撞啦!\n");
107                 system("pause");
108                 break;
109             }
110
111             ShowGame();
112             clkStart = clkEnd; // 让时间再次相等
113         }
114     }
115     for (int i = g_CountNode; i >= 1; i--)//从后往前遍历整条蛇，并释放存储节点的每个空间
116     {
117         pTail = BodyFind(i);
118         free(pTail);
119     }
120     free(g_pFoodPlace); //释放存储空间
121 }

```

好了大功告成



游戏可能有我没有发现的bug，有兴趣还可以在改进改进如下



好了C语言版贪吃的小蛇到这里就全都就结束了

小蛇的蛇生就交给你来守护了!