

第7讲：数组和函数实践：扫雷游戏

目录

1. 扫雷游戏分析和设计
2. 扫雷游戏的代码实现
3. 扫雷游戏的扩展

正文开始

1. 扫雷游戏分析和设计

1.1 扫雷游戏的功能说明

- 使用控制台实现经典的扫雷游戏
- 游戏可以通过菜单实现继续玩或者退出游戏
- 扫雷的棋盘是9*9的格子
- 默认随机布置10个雷
- 可以排查雷
 - 如果位置不是雷，就显示周围有几个雷
 - 如果位置是雷，就炸死游戏结束
 - 把除10个雷之外的所有雷都找出来，排雷成功，游戏结束

游戏的界面：

```

*****
*****  1. play      *****
*****  0. exit      *****
*****
请选择:>1
***** 扫雷 *****
0 1 2 3 4 5 6 7 8 9
1 * * * * *
2 * * * * *
3 * * * * *
4 * * * * *
5 * * * * *
6 * * * * *
7 * * * * *
8 * * * * *
9 * * * * *
请输入要排查的坐标:>

```

初始界面

比特就业课主页: <https://www.cctalk.com/inst/s9yewhfr>

```

***** 扫雷 *****
0 1 2 3 4 5 6 7 8 9
1 0 0 0 0 0 1 * * *
2 0 0 0 0 0 1 * * *
3 * * * * *
4 * * * * *
5 * * * * *
6 * * * * *
7 * * * * *
8 * * * * *
9 * * * * *
请输入要排查的坐标:>

```

排雷界面

```

请输入要排查的坐标:>1 7
很遗憾,你被炸死
***** 扫雷 *****
0 1 2 3 4 5 6 7 8 9
1 0 0 0 0 0 0 1 0 0
2 0 0 0 0 0 0 0 0 1
3 0 0 0 0 0 0 0 0 0
4 0 0 1 1 1 0 0 0 0
5 0 0 0 0 0 1 0 0 0
6 0 0 0 0 0 0 0 0 0
7 0 0 1 1 0 0 0 0 0
8 0 0 0 0 0 0 0 0 0
9 0 0 0 0 0 1 1 0 0

```

排雷失败界面

1.2 游戏的分析和设计

1.2.1 数据结构的分析

扫雷的过程中，布置的雷和排查出的雷的信息都需要存储，所以我们需要一定的数据结构来存储这些信息。

因为我们需要在9*9的棋盘上布置雷的信息和排查雷，我们首先想到的就是创建一个9*9的数组来存放信息。

		0	1	2	3	4	5	6	7	8
0										
1										
2										
3										
4										
5										
6										
7										
8										

空棋盘

那如果这个位置布置雷，我们就存放1，没有布置雷就存放0.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	1	0	0	0
3	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	1	0	1	0	1	0	0	0	0
7	0	1	0	0	1	0	1	0	0
8	0	0	0	0	0	0	0	0	0

布置雷的棋盘

假设我们排查(2,5)这个坐标时，我们访问周围的一圈8个黄色位置，统计周围雷的个数是1

假设我们排查(8,6)这个坐标时，我们访问周围的一圈8个黄色位置，统计周围雷的个数时，最下面的三个坐标就会越界，为了防止越界，我们在设计的时候，给数组扩大一圈，雷还是布置在中间的9*9的坐标上，周围一圈不去布置雷就行，这样就解决了越界的问题。所以我们将存放数据的数组创建成11*11是比较合适。

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	1
3	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	1	0	1	0	1	0	0	0	0
7	0	1	0	0	1	0	1	0	0
8	0	0	0	0	0	0	0	0	0

排雷的假设

	0	1	2	3	4	5	6	7	8	9	10
0											
1		0	0	0	0	0	0	0	0	0	
2		0	0	0	0	0	0	0	0	0	
3		0	0	1	0	0	0	0	0	1	
4		0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	0	0	0	
6		0	0	0	0	0	0	1	0	0	
7		1	0	1	0	1	0	0	0	0	
8		0	1	0	0	1	0	1	0	0	
9		0	0	0	0	0	0	0	0	0	
10											

周围加上一圈的棋盘

再继续分析，我们在棋盘上布置了雷，棋盘上雷的信息（1）和非雷的信息（0），假设我们排查了某一个位置后，这个坐标处不是雷，这个坐标的周围有1个雷，那我们需要将排查出的雷的数量信息记录存储，并打印出来，作为排查的重要参考信息的。那这个雷的个数信息存放在哪里呢？如果存放在布置雷的数组中，这样雷的信息和雷的个数信息就可能或产生混淆和打印上的困难。

这里我们肯定有办法解决，比如：雷和非雷的信息不要使用数字，使用某些字符就行，这样就避免冲突了，但是这样做棋盘上有雷和非雷的信息，还有排查出的雷的个数信息，就比较混杂，不够方便。

这里我们采用另外一种方案，我们专门给一个棋盘（对应一个数组mine）存放布置好的雷的信息，再给另外一个棋盘（对应另外一个数组show）存放排查出的雷的信息。这样就互不干扰了，把雷布置到mine数组，在mine数组中排查雷，排查出的数据存放在show数组，并且打印show数组的信息给后期排查参考。

同时为了保持神秘，show数组开始时初始化为字符 '*', 为了保持两个数组的类型一致，可以使用同一套函数处理，mine数组最开始也初始化为字符'0'，布置雷改成'1'。如下如：

	0	1	2	3	4	5	6	7	8	9	10
0											
1		'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	
2		'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	
3		'0'	'0'	'1'	'0'	'0'	'0'	'0'	'0'	'1'	
4		'0'	'0'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	
5		'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	
6		'0'	'0'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	
7		'1'	'0'	'1'	'0'	'1'	'0'	'0'	'0'	'0'	
8		'0'	'1'	'0'	'0'	'1'	'0'	'1'	'0'	'0'	
9		'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	
10											

mine数组布置雷后的状态

	0	1	2	3	4	5	6	7	8	9	10
0											
1		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
2		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
3		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
4		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
5		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
6		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
7		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
8		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
9		'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	'*'	
10											

show输出初始化的状态

对应的数组应该是：

```
1 char mine[11][11] = {0}; //用来存放布置好的雷的信息
2 char show[11][11] = {0}; //用来存放排查出的雷的个数信息
```

1.2.2 文件结构设计

之前学习了多文件的形式对函数的声明和定义，这里我们实践一下，我们设计三个文件：

```
1 test.c //文件中写游戏的测试逻辑
2 game.c //文件中写游戏中函数的实现等
```

2. 扫雷游戏的代码实现

game.h

```
1 #pragma once
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 #define EASY_COUNT 10
8
9 #define ROW 9
10 #define COL 9
11
12 #define ROWS ROW+2
13 #define COLS COL+2
14
15 //初始化棋盘
16 void InitBoard(char board[ROWS][COLS], int rows, int cols, char set);
17
18 //打印棋盘
19 void DisplayBoard(char board[ROWS][COLS], int row, int col);
20
21 //布置雷
22 void SetMine(char board[ROWS][COLS], int row, int col);
23
24 //排查雷
25 void FindMine(char mine[ROWS][COLS], char show[ROWS][COLS], int row, int col);
26
```

game.c

```
1 #include "game.h"
2
3 void InitBoard(char board[ROWS][COLS], int rows, int cols, char set)
4 {
5     int i = 0;
6     for (i = 0; i < rows; i++)
7     {
```

```
8         int j = 0;
9         for (j = 0; j < cols; j++)
10         {
11             board[i][j] = set;
12         }
13     }
14 }
15
16 void DisplayBoard(char board[ROWS][COLS], int row, int col)
17 {
18     int i = 0;
19     printf("-----扫雷游戏-----\n");
20     for (i = 0; i <= col; i++)
21     {
22         printf("%d ", i);
23     }
24     printf("\n");
25     for (i = 1; i <= row; i++)
26     {
27         printf("%d ", i);
28         int j = 0;
29         for (j = 1; j <= col; j++)
30         {
31             printf("%c ", board[i][j]);
32         }
33         printf("\n");
34     }
35 }
36
37 void SetMine(char board[ROWS][COLS], int row, int col)
38 {
39     //布置10个雷
40     //生成随机的坐标, 布置雷
41     int count = EASY_COUNT;
42     while (count)
43     {
44         int x = rand() % row + 1;
45         int y = rand() % col + 1;
46
47         if (board[x][y] == '0')
48         {
49             board[x][y] = '1';
50             count--;
51         }
52     }
53 }
54
```

```
55 int GetMineCount(char mine[ROWS][COLS], int x, int y)
56 {
57     return (mine[x-1][y]+mine[x-1][y-1]+mine[x][y-1]+mine[x+1][y-1]+mine[x
58         mine[x+1][y+1]+mine[x][y+1]+mine[x-1][y+1] - 8 * '0');
59 }
60
61
62 void FindMine(char mine[ROWS][COLS], char show[ROWS][COLS], int row, int col)
63 {
64     int x = 0;
65     int y = 0;
66     int win = 0;
67     while (win < row*col- EASY_COUNT)
68     {
69         printf("请输入要排查的坐标:>");
70         scanf("%d %d", &x, &y);
71         if (x >= 1 && x <= row && y >= 1 && y <= col)
72         {
73             if (mine[x][y] == '1')
74             {
75                 printf("很遗憾, 你被炸死了\n");
76                 DisplayBoard(mine, ROW, COL);
77                 break;
78             }
79             else
80             {
81                 //该位置不是雷, 就统计这个坐标周围有几个雷
82                 int count = GetMineCount(mine, x, y);
83                 show[x][y] = count + '0';
84                 DisplayBoard(show, ROW, COL);
85                 win++;
86             }
87         }
88         else
89         {
90             printf("坐标非法, 重新输入\n");
91         }
92     }
93     if (win == row * col - EASY_COUNT)
94     {
95         printf("恭喜你, 排雷成功\n");
96         DisplayBoard(mine, ROW, COL);
97     }
98 }
```

```
1 #include "game.h"
2
3 void menu()
4 {
5     printf("*****\n");
6     printf("***** 1. play *****\n");
7     printf("***** 0. exit *****\n");
8     printf("*****\n");
9 }
10
11 void game()
12 {
13     char mine[ROWS][COLS]; //存放布置好的雷
14     char show[ROWS][COLS]; //存放排查出的雷的信息
15     //初始化棋盘
16     //1. mine数组最开始是全'0'
17     //2. show数组最开始是全'*'
18     InitBoard(mine, ROWS, COLS, '0');
19     InitBoard(show, ROWS, COLS, '*');
20     //打印棋盘
21     //DisplayBoard(mine, ROW, COL);
22     DisplayBoard(show, ROW, COL);
23     //1. 布置雷
24     SetMine(mine, ROW, COL);
25     //DisplayBoard(mine, ROW, COL);
26     //2. 排查雷
27     FindMine(mine, show, ROW, COL);
28 }
29
30 int main()
31 {
32     int input = 0;
33     srand((unsigned int)time(NULL));
34     do
35     {
36         menu();
37         printf("请选择:>");
38         scanf("%d", &input);
39         switch (input)
40         {
41             case 1:
42                 game();
43                 break;
44             case 0:
45                 printf("退出游戏\n");
46                 break;
```



```

47         default:
48             printf("选择错误，重新选择\n");
49             break;
50     }
51     } while (input);
52
53     return 0;
54 }
```

3. 扫雷游戏的扩展

- 是否可以选择游戏难度
 - 简单 9*9 棋盘，10个雷
 - 中等 16*16棋盘，40个雷
 - 困难 30*16棋盘，99个雷
- 如果排查位置不是雷，周围也没有雷，可以展开周围的一片
- 是否可以标记雷
- 是否可以加上排雷的时间显示

在线扫雷游戏：<http://www.minesweeper.cn/>

完