# 推箱子攻略

> 由于要使用Windows提供的接口(Win32 API)，所以要包含头文件<Windows.h>，包含的其他头文件你一一确定即可

## 1.游戏逻辑

1. 设置控制台大小
2. 打印主菜单，接收键盘输入的选项
3. 判断输入内容分别进入不同的分支流（退出游戏，游戏说明，进入游戏等）
4. 进入游戏后，清屏
5. 设置地图
6. 渲染地图
7. 非阻塞式键盘监听，开始游戏移动逻辑
8. 每移动一次，即改变一次地图数组中的值，重新渲染地图
9. 判断是否完成本局目标，进入完成分支判断
10. 如果没有完成，则继续循环，否则判断是否通关，如果通关，进入通关逻辑
11. 如果没有通关，进入下一关

## 2.设置控制台窗口大小

```
system("mode con cols=50 lines=20");
```

**注:**

- system这个函数使用的是系统调用，详细使用涉及到Win32编程的知识，这里不深究
- 等号之间不能有空格
- 行和列的值可以自行设定，这里我只是提供一个参考值

## 3.设置游戏元素的颜色

```
void color(int m)
{
    HANDLE hconsole;
    hconsole = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hconsole, m);
}
```

**注:**

- 这是我封装好的一个函数，下次输出的颜色根据参数m的值而定

```
0 = 黑色     8 = 灰色     1 = 淡蓝       9 = 蓝色
2 = 淡绿     A = 绿色     3 = 湖蓝       B = 淡浅绿
C = 红色     4 = 淡红     5 = 紫色       D = 淡紫
6 = 黄色     E = 淡黄     7 = 白色       F = 亮白
使用低4位表示文字（前景）颜色，高4位表示文字背景颜色，所以它的取值为xx。x为一位16进制数，即0~F都可以使用，可以随意组合。

如果只希望设置文字颜色，背景保持黑色，那么也可以只给出一位16进制数，例如：
color(0xC);  //将文字颜色设置为红色
color(0xF);  //将文字颜色设置为白色
```

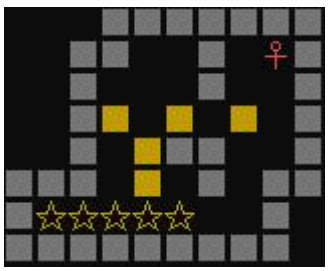
- 这个函数的实现涉及到Win32编程，同样不予深究，只要需要的时候查相关的接口即可

## 4.设置游戏地图

一般情况下，平面的存取都选用数据结构中的二维数组，由于这里的地图也是平面的，所以也不例外。

**精巧之处**：

虽说数组可以用来保存二维地图，但是实际中的地图是什么样子的，我们来看看：



那么难点就来了，数组是不能直接存放这些元素的（星星，小人...），那么怎么来用数组存放地图？你可以想一想再往下看。（元素是这样打印出来的，切换到中文输入法，输入“fangkuai”，选择出现的你需要的图案即可）eg：

解决方案就是约定一个协议，这个协议规定了数字和所需元素的映射关系，例如：用1表示砖块，2表示箱子等。所以这个整形的二维数组中存放的就是地图的原始值了，只要在显示地图的时候，专门调用一个负责渲染地图的函数，这个函数遍历二维数组，按照映射关系将对应位置的对应元素一一打印出来即可。（这个函数当然还是要自己实现）

```
//eg:
int map_1[10][10] = {
    { 0, 0, 1, 1, 1, 0, 0, 0 },
    { 0, 0, 1, 4, 1, 0, 0, 0 },
    { 0, 0, 1, 0, 1, 1, 1, 1 },
    { 1, 1, 1, 0, 0, 2, 4, 1 },
    { 1, 4, 2, 2, 0, 1, 1, 1 },
    { 1, 1, 1, 3, 2, 1, 0, 0 },
    { 0, 0, 0, 1, 4, 1, 0, 0 },
    { 0, 0, 0, 1, 1, 1, 0, 0 }
};
```

## 5.键盘监听

这个游戏需要实现的是 **阻塞式键盘监听**

eg: 按一下 'w' 键，角色就向上移动一步。

与之相对的是 **非阻塞式键盘监听**

eg: 即按一下 'w', 角色就一直向上移动。

下面我写一个伪代码框架，你慢慢理解：

```
while((pos = getch()) != EOF){
    system("cls");
    switch(pos){
        case 'w':
            //向上移动的代码
            break
        default:
            //其他事件
            break;
    }
```

```
}
/*
1.getch() 用来接收一个键盘输入的字符
2.EOF字符表示输入结束，键盘按键为：ctrl + d
3.system("cls") 用来清屏
4.注意，不要将第四个移动代码直接扔到default里面去，而是同样需要放到一个case分支里，详细原因参见我给你的那篇C语言的博客
*/
```

## 6.移动逻辑

这也是这个游戏的最核心部分了，一共要考虑十二种情况，我已经给你列出来了部分，另一部分你自己想想都有哪些情况，然后根据这十二种情况实现角色移动的代码。


> 为什么要考虑这些情况呢?

- 第 1 种情况处理方案：人物移动后，原位置恢复为空格（即数字0），人物移动后位于空格上，所以该位置对应的数字为 3，箱子也向前一步，此时箱子位于空格上，所有该位置对应的数字为2。
- 第 2 种情况处理方案：人物移动后，原位置恢复为空格，人物站在空格上，该位置对应的数字为 3，箱子向前一步，恰好对于目标位置处，对应的数字为 5。
- 第 3 种情况处理方案：人物移动后，原位置恢复为空格，人物位于箱子的目标位置上，此位置对应的数字为 6，箱子向前一步，位于空格上，该位置对应的数字为 2。
- 第 4 种情况处理方案：人物移动后，原位置恢复为空格，人物位于箱子的目标位置上，此位置对应的数字为 6，箱子向前一步，还是位于目标位置处，对应的数字还是 5。
- 第 5 种情况处理方案：人物移动后，原位置恢复为空格，移动后的人物位于空格处，该位置对应的数字为 3。
- 第 6 种情况处理方案：人物移动后，原位置恢复为空格，移动后的人物位于箱子的目标位置上，此位置对应的数字为 6。