第 2 章

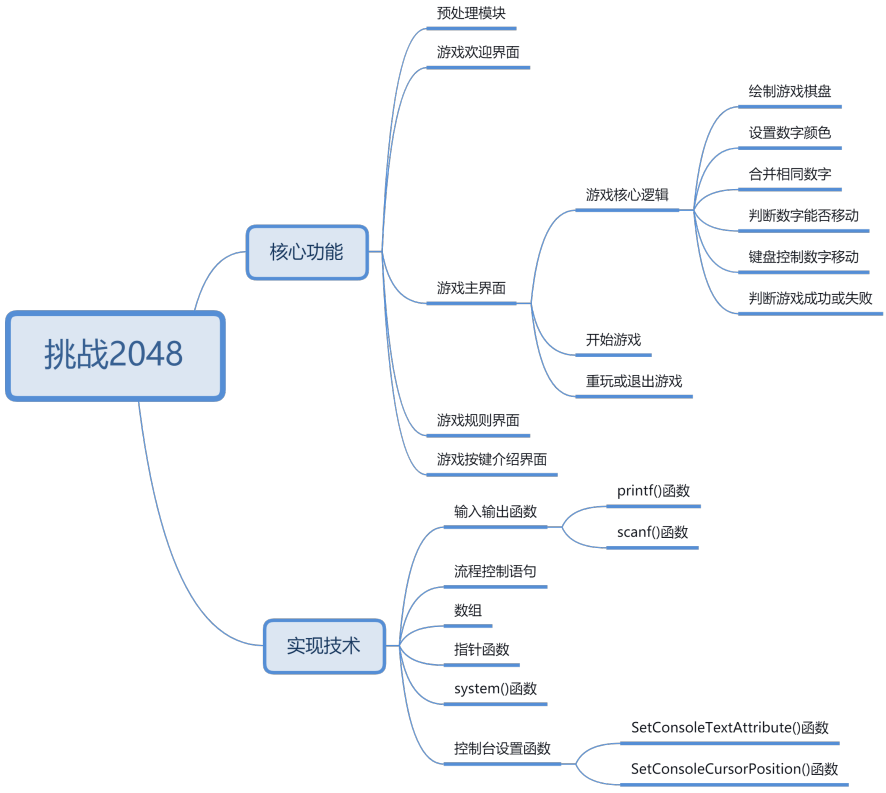
挑战 2048

——输入输出函数 + 流程控制语句 + 数组 + 指针函数 + system()函数 + 控制台设置函数

2048 游戏是一款简单而又具有挑战性的数字益智游戏。它的规则简单易懂，但随着游戏的进行，难度逐渐增加，需要玩家具备良好的逻辑判断能力和规划能力。近年来，2048 游戏因其界面简洁、趣味性强、能给玩家带来成就感，在全球范围内受到了广泛的欢迎。C 语言在游戏引擎和图形库的底层开发中具有独特的性能优势，所以本章使用 C 语言开发一个挑战 2048 游戏。这个项目不仅可以帮助读者巩固 C 语言基础知识，锻炼编程能力，还可以帮助读者了解游戏开发的过程，积累 C 语言开发实战经验。


项目微视频

本项目的核心功能及实现技术如下：



2.1 开 发 背 景

挑战 2048 游戏最早于 2014 年 3 月 20 日发行，是一款十分流行的数字游戏，其基于“数字组合 1024”

和“小 3 传奇”的玩法而开发。该游戏规则十分简单，其目标是通过键盘上的方向键移动数字，合并相同的数字，最终合成 2048 这个数字，表示游戏成功，其具体规则如下：

- ☑ 游戏初始运行时，默认显示一个 4×4 的方格棋盘，在少数方格内会随机生成一个数字（2 或 4）。
- ☑ 玩家可以通过方向键控制数字向指定方向移动。
- ☑ 当两个相同数字在移动过程中相遇时，它们会合并成为一个新的数字，其值为两者之和。例如，两个 2 合并成 4，两个 4 合并成 8，以此类推。
- ☑ 每次玩家做出有效移动后，被移动的空白方格位置会自动生成一个新的数字（2 或 4）。
- ☑ 每次移动时，由数字合并所产生的新数字即为本次移动所得的分数，游戏总得分为每次得分的累加。

本项目的实现目标如下：

- ☑ 游戏界面直观、简洁，能够清晰地显示游戏棋盘和得分。
- ☑ 实现基本的 2048 游戏机制，包括棋盘初始化、数字移动、新数字生成、游戏结束判断等。
- ☑ 提供方便的重玩或结束游戏功能。当玩家成功地在棋盘上生成了数字 2048 时，视为游戏胜利；所有方格都被数字填满，且无法再通过任何移动操作使数字合并时，游戏失败。无论游戏胜利或失败，程序都可以提示是否重玩或结束游戏，玩家可根据需要进行选择。
- ☑ 良好的用户交互性，游戏需要支持用户的键盘输入，以便玩家能够控制数字的移动。

2.2 系 统 设 计

2.2.1 开发环境

本项目的开发及运行环境要求如下：

- ☑ 操作系统：推荐 Windows 10、Windows 11 或更高版本，兼容 Windows 7（SP1）。
- ☑ 开发工具：Dev C++ 5.11 或更高版本。
- ☑ 开发语言：C 语言。

2.2.2 业务流程

在游戏启动后，首先呈现的是游戏欢迎界面，该界面中包含“开始游戏”“游戏规则”“按键说明”和“退出”4 个菜单，用户可以通过输入菜单编号来执行相应操作。选择“开始游戏”菜单，即输入编号 1，进入游戏主界面，这时即可按照游戏规则进行 2048 游戏的挑战，每次挑战成功或者失败后，程序会询问用户是否重玩或结束游戏。如果重玩，输入编号 1，程序会重新回到游戏主界面；如果结束游戏，则输入编号 2。

本项目的业务流程如图 2.1 所示。

2.2.3 功能结构

本项目的功能结构已经在章首页中给出。作为一个经典的 2048 游戏项目，本项目实现的具体功能如下：

- ☑ 游戏欢迎界面：游戏运行后的首屏界面，主要包含“开始游戏”“游戏规则”“按键说明”和“退出”4 个游戏菜单项。

☑ 游戏主界面：在该界面中，玩家按照规则挑战 2048 游戏，其中主要包括绘制游戏棋盘、设置数字显示不同颜色、合并相同数字、判断数字能否移动、通过键盘控制数字移动、判断游戏成功或失败、开始游戏、重新开始游戏等功能。

☑ 游戏规则界面：显示游戏规则及玩法。

☑ 按键说明界面：显示游戏中按键的具体使用方法。

☑ 退出：退出当前游戏。

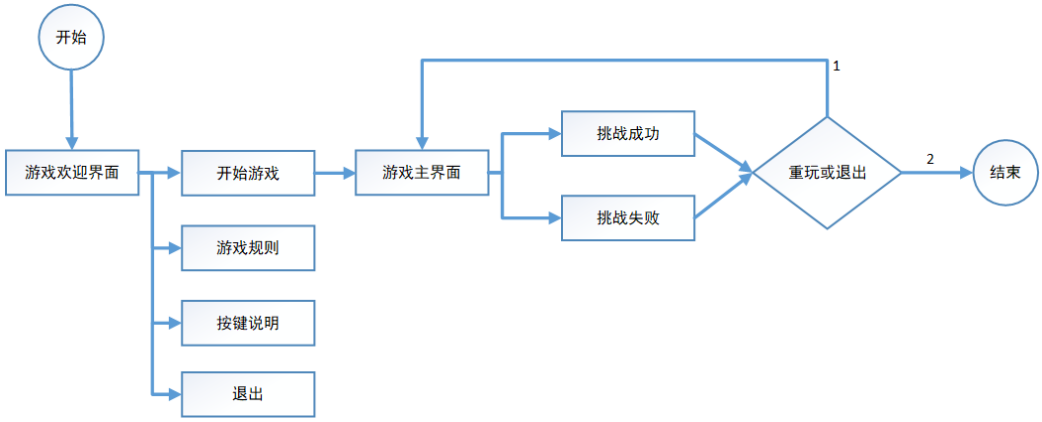


图 2.1　挑战 2048 游戏业务流程图

2.3　技 术 准 备

2.3.1　技术概览

☑ printf()函数：该函数是 C 语言标准输出函数，用于将格式化的数据输出到标准输出设备。该函数能够打印各种类型的数据，并允许按照指定的格式输出数据。printf()函数的声明定义在<stdio.h>头文件中。例如，下面代码使用 printf()函数格式化输出多种类型的数据：

```
#include <stdio.h>

int main()
{
    int a = 10;
    float b = 3.14159;
    char c = 'A';
    char str[] = "Hello, World!";
    printf("整数：%d\n", a);
    printf("浮点数：%f\n", b);
    printf("字符：%c\n", c);
    printf("字符串：%s\n", str);
    //格式化输出：宽度、精度控制
    printf("浮点数（保留两位小数）：%.2f\n", b);
    //左对齐、宽度为 10 的整数，右对齐、宽度自适应的浮点数
    printf("宽度控制：%-10d   %.2f\n", a, b);
    return 0;
}
```

☑ scanf()函数：该函数是 C 语言标准输入函数，用于从标准输入设备（通常是键盘）读取输入数据，

并根据提供的格式字符串解析这些数据，然后将其存储到相应的变量中。scanf()函数的声明定义在<stdio.h>头文件中。例如，下面代码使用 scanf()函数接收用户输入的不同类型数据，并使用 printf()函数进行输出：

```
#include <stdio.h>

int main()
{
    int num;
    float decimal;
    char ch;
    char str[20];
    //读取整数
    printf("请输入一个整数：");
    scanf("%d", &num);
    //读取浮点数
    printf("请输入一个小数：");
    scanf("%f", &decimal);
    //读取单个字符
    printf("请输入一个字符：");
    scanf(" %c", &ch);
    //读取字符串
    printf("请输入一个字符串：");
    scanf("%s", str);
    printf("您输入的信息是：\n 整数：%d\n 小数：%f\n 字符：%c\n 字符串：%s\n", num, decimal, ch, str);
    return 0;
}
```

☑ 流程控制语句：流程控制是指在编程中管理和指导程序执行顺序的一系列技术和策略，它能够确保程序按照预定的逻辑和需求，有效地执行任务，处理数据，并最终达到预期的结果。本项目主要使用 switch 多分支语句和 for 循环语句。例如，下面代码使用 for 循环打印棋盘的边框：

```
for(j = 2;j <= 22;j += 5)                    //打印棋盘边框
{
    gotoxy(15,j);
    for(k = 1;k<42;k++)
    {
        printf("-");
    }
    printf("\n");
}
```

☑ 数组：数组是一种基本的数据结构，用于存储相同类型的元素。C 语言中最常用的数组类型有一维数组和二维数组。一维数组是最基本的数组形式，用于存储单一序列的数据元素；而二维数组则可以视为由多个一维数组组成的数组，用于表示表格状的数据结构。例如，下面代码用来定义并初始化一维数组和二维数组：

```
/*一维数组定义与初始化*/

//声明一个整型一维数组，未初始化
int arr1[5];

//声明并初始化一个整型一维数组
int arr2[5] = {1, 2, 3, 4, 5};

//不指定数组长度，由初始化元素数量决定
int arr3[] = {10, 20, 30};

/*二维数据定义与初始化*/
```

```
//声明一个 3x4 的整型二维数组，未初始化
int matrix[3][4];

//声明并初始化一个二维数组
int matrix2[3][2] = {
      {1, 2},
      {3, 4},
      {5, 6}
};

//不完全初始化，未给出的元素自动初始化为 0
int matrix3[][3] = {
      {1, 2, 3},
      {4, 5}
};
```

☑ 指针函数：即返回值是指针的函数，本质上仍然是一个函数。指针函数很容易与函数指针混淆，函数指针本质是一个指针，它指向一个函数。例如，下面代码展示了指针函数与函数指针的不同写法：

```
//指针函数
int *myfun(int x,int y);

//函数指针
int (*p)(int, int);
```

☑ Dev C++工具： Dev C++是一款适用于 Windows 环境的轻量级 C/C++集成开发环境（IDE），同时兼容多种操作系统。它主要面向初学者和教育用途，因其简单易用且免费而广受欢迎。本章使用该工具来开发挑战 2048 游戏。

有关 printf()函数、scanf()函数、流程控制语句、数组、指针函数、DevC++工具等知识，在《C 语言从入门到精通（第 6 版）》中有详细的讲解。对这些知识不太熟悉的读者，可以参考该书对应的内容。下面将对本项目中使用的其他 C 语言知识进行必要的介绍，包括 system()函数、控制台设置函数（如 SetConsoleTextAttribut ()函数、SetConsoleCursorPosition()函数），以确保读者可以顺利完成本项目的开发。

2.3.2 system()函数

system()函数是一个标准库函数，位于<stdlib.h>头文件中。该函数允许开发人员执行操作系统命令，为 C 语言程序提供了一种能够与操作系统交互的方式，使其能够执行外部命令，如打开文件、运行其他程序或改变终端设置等。例如，下面代码通过在 system()函数中指定 TITLE 关键字来修改命令行窗口的标题：

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

int main(void)
{
      system("TITLE 自定义标题");
      return 0;
}
```

上面代码的运行效果如图 2.2 所示。

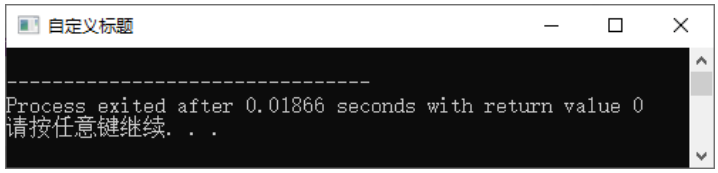


图 2.2　命令行自定义标题

除了可以修改命令行窗口的标题，system()函数还可以通过指定其他关键字参数来执行系统的相关命令，可以指定的关键字参数如表 2.1 所示。

表 2.1　system()函数可以指定的关键字参数

参　数	功　能
ASSOC	显示或修改文件扩展名关联
ATTRIB	显示或更改文件属性
BREAK	中断或终止程序的执行，类似在 CMD 控制台窗口中按 Ctrl+C 快捷键
CALL	通过另一个批处理程序来调用指定的命令
CD	切换当前目录
CHDIR	更改当前目录到指定的目录
CHKDSK	检查磁盘并显示状态报告
CHKNTFS	启动磁盘检查
CLS	清除屏幕
CMD	打开另一个 CMD 控制台窗口
COLOR	设置默认控制台前景和背景颜色
COMPACT	显示或更改 NTFS 分区上文件的压缩
CONVERT	将 FAT 格式转换为 NTFS 格式，但不能转换当前正在使用的磁盘
COPY	将至少一个文件复制到另一个位置
DATE	显示或设置日期
DEL	删除至少一个文件
DIR	显示一个目录中的文件和子目录
DISKPART	显示或配置磁盘分区属性
DRIVERQUERY	显示当前设备驱动程序状态和属性
ECHO	显示消息，或将命令回显打开或关闭
ERASE	删除一个或多个文件
EXIT	退出 CMD 控制台窗口
FC	比较两个文件或两个文件集并显示它们之间的不同
FIND	在一个或多个文件中搜索一个文本字符串
FINDSTR	在多个文件中搜索字符串
FOR	为一组文件中的每个文件运行一个指定的命令
FORMAT	格式化磁盘
FTYPE	显示或修改在文件扩展名关联中使用的文件类型
HELP	提供 Windows 命令的帮助信息

2.3.3 控制台设置函数

1. SetConsoleTextAttribute()函数

SetConsoleTextAttribute()函数是一个 Windows 系统的 API 函数，用于设置 CMD 控制台窗口中的文本属性，如字体颜色、背景等。在 C 语言中，可以使用#include <windows.h>来引入该函数。例如，下面代码使用 SetConsoleTextAttribute()函数来改变 CMD 控制台窗口中文本的颜色：

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

int main()
{
    //获取控制台句柄
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    //设置文本颜色为红色
    SetConsoleTextAttribute(hConsole, FOREGROUND_RED);
    printf("这是红色文本\n");
    //重置文本颜色为默认颜色
    SetConsoleTextAttribute(hConsole, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE);
    printf("这是默认颜色文本\n");
    return 0;
}
```

2. SetConsoleCursorPosition()函数

SetConsoleCursorPosition()函数用来移动 CMD 控制台窗口中光标的位置，这里要注意的是，每次调用该函数时，光标都是默认从左上角开始偏移，而与当前光标停留的位置无关。使用该函数时，需要传入两个参数：HANDLE 和 COORD。其中，HANDLE 参数表示控制台输出句柄，COORD 参数表示一个坐标位置。例如，下面代码在 CMD 控制台窗口的指定位置输出文本内容：

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

void SetCCPos(int x, int y);
void SetCCPos(int x, int y) {
    HANDLE hOut;
    hOut = GetStdHandle(STD_OUTPUT_HANDLE);                    //获取标注输出句柄
    COORD pos;
    pos.X = x;pos.Y = y;
    SetConsoleCursorPosition(hOut, pos);                       //偏移光标位置
}

int main(){
    SetCCPos(1, 0);
    printf("R");
    SetCCPos(0, 1);
    printf("I");
    return 0;
}
```

上面代码的运行效果如图 2.3 所示。其中，字母 R 是在坐标 x 等于 1、y 等于 0 的位置输出；字母 T 是在坐标 x 等于 0、y 等于 1 的位置输出。

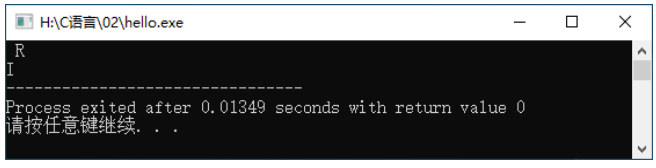


图 2.3 SetConsoleCursorPosition()函数的使用

2.4 预处理模块设计

2.4.1 文件引入

开发挑战 2048 游戏项目时，首先需要引入项目中需要的库文件，以便调用其中的函数。在引用库文件时，需要使用#include 命令，代码如下：

```
/* 引入头文件 */
#include <stdio.h>                          //标准输入输出函数库（printf、scanf）
#include <conio.h>                          //为了读取方向键
#include <windows.h>                        //设置 CMD 控制台窗口(获取控制台上的坐标位置、设置字体颜色)
#include <math.h>                           //引入数学库文件
```

2.4.2 定义全局变量

在挑战 2048 游戏中，需要实时显示游戏的移动步数、分数和运行时间，这里将这些变量定义为全局变量。定义一个二维数组 BOX，表示 4×4 棋盘中每个方格的位置。定义一个 HANDLE 类型的变量，表示控制台句柄，主要通过它来获取控制台上的坐标位置，并且设置字体颜色。代码如下：

```
/* 定义全局变量 */
int step=0;                                 //已执行的游戏步数
int score=0;                                //存储当前的游戏分数
long int gameTime;                          //游戏运行时间
int BOX[4][4]={{0,0,0,0},                   //游戏中的 16 个格子
               {0,0,0,0},
               {0,0,0,0},
               {0,0,0,0}};
HANDLE hOut;                                //控制台句柄
```

2.4.3 函数声明

在代码文件中声明程序中将要使用的函数，代码如下：

```
/* 函数声明 */
void gotoxy(int x, int y);                  //将屏幕光标移动到指定的(x,y)位置
int color(int c);                           //设置文字颜色
int TextColors(int i);                      //根据数字修改颜色
void drawTheGameBox();                      //绘制游戏界面
int *add(int item[]);                       //合并数字
int ifMove(int item[]);                     //判断数组中的数字是否可以进行移动操作
                                            //若可以移动，则返回 1；否则返回 0
void Gameplay();                            //开始游戏
void Replay();                              //重新游戏
int if2n(int x);                            //判断 x 是否是 2 的 n 次方
//判断是否能够上移，若可以上移（方格中的两个数相加是 2 的 n 次方），则返回 1；若不能上移，则返回 0
int ifup();
//判断是否能够下移，若可以下移，则返回 1；若不能下移，则返回 0
int ifdown();
//判断是否能够左移，若可以左移，则返回 1；若不能左移，则返回 0
int ifleft();
//判断是否能够右移，若可以右移，则返回 1；若不能右移，则返回 0
```

```
int ifright();
int BOXmax();                          //返回棋盘最大数
int Gamefaile();                       //判断是否失败
int Gamewin();                         //判断是否胜利
int keyboardControl(int key);          //根据按键输入控制数字的移动
void close();                          //关闭游戏
void title();                          //绘制标题
void choice();                         //选择框
void regulation();                     //游戏规则介绍
void explanation();                    //按键说明
```

2.5 游戏欢迎界面设计

2.5.1 游戏欢迎界面概述

挑战 2048 游戏的游戏欢迎界面主要由两部分组成：第一部分是标题，它以 2048 字符画形式显示；第二部分是菜单选项，包括开始游戏、游戏规则、按键说明、退出 4 个菜单。另外，在游戏欢迎界面中，系统还会提示用户输入要操作的菜单编号。游戏欢迎界面运行效果如图 2.4 所示。

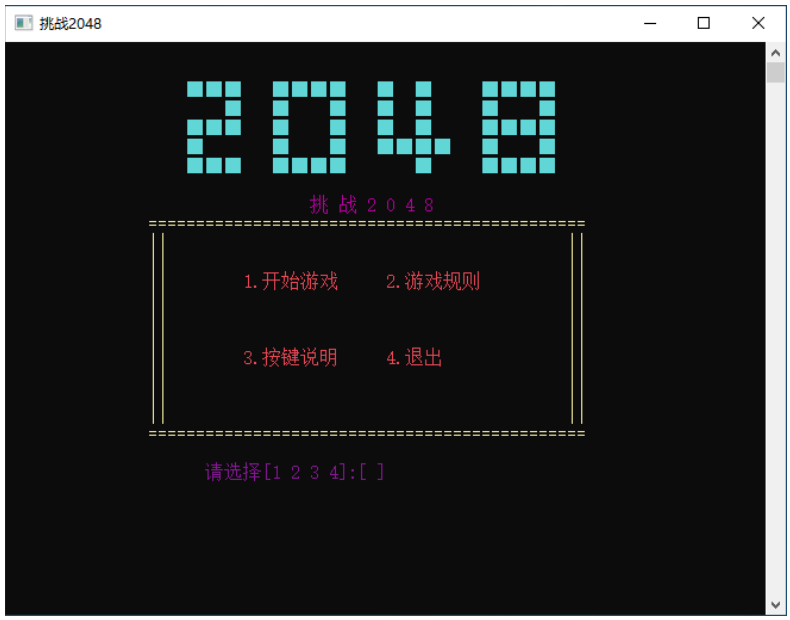


图 2.4 游戏欢迎界面

2.5.2 设置游戏欢迎界面标题

游戏欢迎界面的标题是以“2048”字符画的形式体现的，该功能主要通过自定义的 title()函数实现，代码如下：

```
/**
 * 设置标题
 */
void title()
```

```
{
    color(11);                                                    //浅淡绿色
    gotoxy(19,2);
    printf("■■■   ■■■■   ■  ■    ■■■■");                          //输出 2048 字符画
    gotoxy(19,3);
    printf("   ■   ■    ■  ■    ■    ■");
    gotoxy(19,4);
    printf("■■■   ■    ■  ■    ■■■■");
    gotoxy(19,5);
    printf("■     ■    ■  ■■■■  ■    ■");
    gotoxy(19,6);
    printf("■■■   ■■■■     ■    ■■■■");
}
```

上面代码中用到了 color()函数和 gotoxy()函数。其中，color()函数用于设置控制台中的文字颜色，其实现代码如下：

```
/**
 * 设置文字颜色的函数
 */
int color(int c)
{
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), c);          //设置文字颜色
    return 0;
}
```

gotoxy()函数用于设置控制台中的光标坐标位置，其实现代码如下：

```
/**
 * 设置屏幕光标位置
 */
void gotoxy(int x, int y)
{
    COORD c;
    c.X = x;
    c.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), c);         //定位光标位置
}
```

2.5.3 实现欢迎界面菜单选项

游戏欢迎界面提供了挑战 2048 游戏的主菜单，主要包括开始游戏、游戏规则、按键说明、退出 4 个菜单，它们显示在游戏标题的下方，如图 2.5 所示。

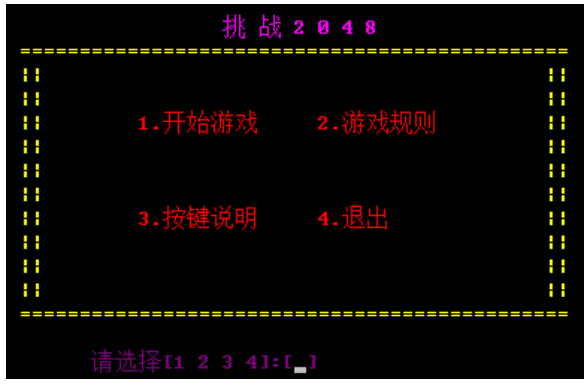


图 2.5 菜单选项

菜单选项的实现分为两部分：绘制边框和显示文字。其中：边框的绘制可以通过嵌套 for 循环实现；而文字的显示只需要确定相应的坐标位置，然后进行打印输出即可。另外，用户在游戏欢迎界面中输入菜单项对应的编号并按 Enter 键后，可以进入相应的功能界面，这一过程主要通过使用 switch 分支语句来实现。代码如下：

```
/**
 * 菜单选项
 */
void choice()
{
    system("title 挑战 2048");
    int n;
    int i,j = 1;
    gotoxy(32,8);
    color(13);
    printf("挑 战 2 0 4 8");
    color(14);                                          //黄色边框
    for (i = 9; i <= 20; i++)                           //输出上下边框===
    {
        for (j = 15; j <= 60; j++)                      //输出左右边框||
        {
            gotoxy(j, i);
            if (i == 9 || i == 20)
                printf("=");
            else if (j == 15 || j == 59)
                printf("||");
        }
    }
    color(12);
    gotoxy(25, 12);
    printf("1.开始游戏");
    gotoxy(40, 12);
    printf("2.游戏规则");
    gotoxy(25, 16);
    printf("3.按键说明");
    gotoxy(40, 16);
    printf("4.退出");
    gotoxy(21,22);
    color(5);
    printf("请选择[1 2 3 4]:[ ]\b\b");
    scanf("%d", &n);                                    //输入选项
    switch (n)
    {
        case 1:
            Gameplay();                                 //游戏开始函数
            break;
        case 2:
            regulation();                               //游戏规则函数
            break;
        case 3:
            explanation();                              //按键说明函数
            break;
        case 4:
            close();                                    //关闭游戏函数
            break;
    }
}
```

说明

上面代码使用了 Gameplay()、regulation()、explanation()和 close() 4 个函数，这些函数分别用于开始游戏、显示游戏规则、显示游戏按键说明、退出游戏。关于这些函数的详细介绍和用法，我们将在后续的相应模块中进行详细讲解。

2.6 游戏主界面设计

2.6.1 游戏主界面概述

在游戏欢迎界面输入数字 1，并按 Enter 键后，即可进入游戏主界面，即挑战 2048 游戏界面，如图 2.6 所示。在该界面中，玩家可以进行游戏的挑战。

图 2.6 游戏主界面

2.6.2 实现游戏核心逻辑功能函数

挑战 2048 游戏的核心逻辑功能主要包括以下 6 个部分：

- ☑ 绘制游戏棋盘：本游戏使用 4×4 的方格作为棋盘，棋盘主要由黄色的细横条“-”和竖条“|”组成。
- ☑ 设置数字显示不同颜色：游戏中出现的数字均为 2 的 n 次方，为了美观和便于操作，需要将 2 的不同次方设置为不同的颜色。
- ☑ 合并相同数字：相同的数字发生碰撞时，它们会合并；如果一个格子为空（没有数字），而另一个

格子有数字，碰撞时也会发生合并。

- ☑ 判断数字能否移动：并不是所有条件下都可以移动数字，只有在数字发生合并时，才能够对数字进行移动。
- ☑ 通过键盘控制数字移动：本游戏使用上、下、左、右方向键来控制数字的移动，因此需要分别设置不同的按键所要进行的操作。
- ☑ 判断游戏是否成功或失败：当棋盘中出现的最大数字为 2048 时，表示游戏成功，弹出游戏成功界面；当棋盘中所有数字都不能进行移动时，表示游戏失败，弹出游戏失败界面。

下面分别介绍实现以上功能的函数实现过程。

1. 绘制游戏棋盘

挑战 2048 游戏使用 4×4 的方格来做棋盘，因此首先需要绘制该棋盘，这里主要通过自定义的 drawTheGameBox()函数实现。该函数主要采用黄色的细横条“-”和竖条“|”组成棋盘。其中，在打印横边框时，使用双重循环控制，外层循环控制开始打印横边框的起始位置，内层循环控制横边框的长度并打印输出；打印竖边框时，使用 4 个 for 循环语句单独控制，其中每个循环用于控制打印出每行格子的 5 条竖边框。另外，在游戏主界面中还需要显示“游戏分数”“执行步数”和“已用时”等信息，这主要使用 gotoxy()函数、color()函数和 printf()函数。其中，gotoxy()函数用于确定输出信息的位置，color()函数用于设置输出文字的颜色，printf()函数用于输出信息。drawTheGameBox()函数的实现代码如下：

```
/**
 * 绘制游戏界面  4×4 的网格
 */
void drawTheGameBox()
{
	int i,j,k;
	gotoxy(16,1);                                   //屏幕坐标位置
	color(11);                                      //淡浅绿色
	printf("游戏分数: %ld",score);
	color(13);                                      //粉色
	gotoxy(42,1);                                   //屏幕坐标位置
	printf("执行步数: %d\n",step);
	color(14);                                      //黄色
	for(j = 2;j <= 22;j += 5)                       //打印棋盘边框
	{
		gotoxy(15,j);
		for(k = 1;k<42;k++)
		{
			printf("-");
		}
		printf("\n");
	}
	for (i = 3;i < 7;i ++)
	{
		gotoxy(15,i);
		printf("|           |           |           |           |");
	}
	for (i = 8;i<12;i++)
	{
		gotoxy(15,i);
		printf("|           |           |           |           |");
	}
	for (i = 13;i<17;i++)
	{
		gotoxy(15,i);
		printf("|           |           |           |           |");
	}
```

```
    for (i = 18;i<22;i++)
    {
        gotoxy(15,i);
        printf("|          |          |          |          | ");
    }
    gotoxy(44,23);
    color(10);                                          //绿色
    printf("已用时：%d s",time(NULL) - gameTime);       //输出游戏运行时间
}
```

说明

在计算游戏运行时间时，上面代码通过将当前时间减去游戏开始时间来得出结果。在这个过程中，time(NULL)函数用于获取当前的时间。

2. 设置数字显示不同颜色

为数字设置不同的颜色，需要使用自定义函数 color()，该函数负责为每个数字（2^n，其中 0<n<12）都分配一种颜色。例如，数字 2 的颜色被设置为 12 号红色，数字 4 的颜色被设置为 11 号亮蓝色等，如图 2.7 所示。

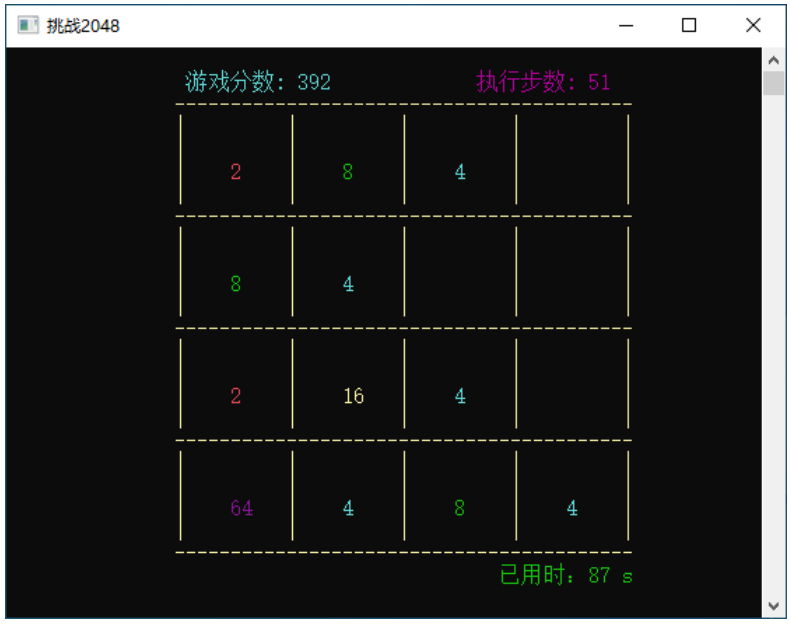


图 2.7　设置不同数字显示不同颜色的结果

设置数字显示不同颜色是通过一个自定义的 TextColors()函数实现的。该函数主要使用 switch 多分支语句，按照 2 的 n 次方进行分类，并使用 color()函数分别设置不同的颜色。TextColors()函数实现代码如下：

```
/**
 * 根据数字修改颜色
 */
int TextColors(int number)
{
    switch (number)                         //格子中出现的数字（2^n）（0<n<12）显示为不同颜色
    {                                       //数字 1~15 代表不同的文字颜色，超过 15 表示文字背景色
        case 2:                             //数字 2
            return color(12);               //显示色号为 12 的颜色，红色
```

```
                break;
            case 4:                                    //数字 4
                return color(11);                      //显示色号为 11 的颜色，亮蓝色
                break;
            case 8:
                return color(10);
                break;
            case 16:
                return color(14);
                break;
            case 32:
                return color(6);
                break;
            case 64:
                return color(5);
                break;
            case 128:
                return color(4);
                break;
            case 256:
                return color(3);
                break;
            case 512:
                return color(2);
                break;
            case 1024:
                return color(9);
                break;
            case 2048:
                return color(7);
                break;
            default:
                break;
        }
        return 0;
}
```

3. 合并相同数字

当使用上、下、左、右方向键移动数字时，在同一方向上，两个相邻位置上的数字如果相同，就可以进行合并操作，合并后的数字会翻倍并积累得分；两个相邻位置上的数字如果不同，则不会进行合并操作。数字合并前后的效果分别如图 2.8 和图 2.9 所示。

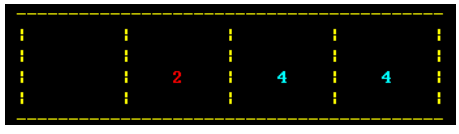


图 2.8　合并之前

图 2.9　合并之后

合并相同数字功能是通过自定义的 add()指针函数实现的。该函数首先定义了 3 个数组，并遍历 item[]数组中的元素，将值不为 0 的元素放入 tep[]数组中；然后使用 for 循环遍历 tep[]数组，在遍历时，比较 tep[i]和 tep[i+1]的值是否相同，如果相同，则 tep[i]的值翻倍，并将 tep[i+1]的值设置为 0，从而实现合并相同数字的效果。add()指针函数的代码如下：

```
int* add(int item[])
{
    int i = 0, j = 0;
    int tep[4] = {0, 0, 0, 0}, tmp[4] = {0, 0, 0, 0};
    for(i = 0; i < 4; i ++)
```

```
    {
        if(item[i] != 0)                    //如果这个格子里有数字
        {
            tep[j ++] = item[i];
        }
    }
    //把两个相邻的相同的数加起来
    for(i = 0; i < 4; i ++)
    {
        if(tep[i] == tep[i + 1])            //两个数字如果相同，则进行合并
        {
            tep[i] *= 2;                    //一个格子中的数字翻倍，另一个为空
            tep[i + 1] = 0;
            score=score+tep[i];             //加分，加的分数为消除的数字*2
        }
    }
    j = 0;
    for(i = 0; i < 4; i ++)
    {
        if(tep[i] != 0)
        {
            tmp[j ++] = tep[i];
        }
    }
    return (int *)(&tmp);                   //tmp 为指针的引用，*&指针本身可变
}
```

4. 判断数字能否移动

在挑战 2048 游戏中，移动数字之前，首先需要判断该数字是否可以移动。游戏中，数字能够移动的情况有两种：一是当两个相邻位置上的数字相同时；二是当相邻的两个位置中，一个位置是空的，而另一个位置上有数字时。例如，在图 2.10 中，第一行有两个相同的数字 4，玩家按向右的方向键时，这两个数字 4 可以移动；在图 2.11 中，第一行的数字 2 左侧位置为空，玩家按向左的方向键时，数字 2 也可以移动。

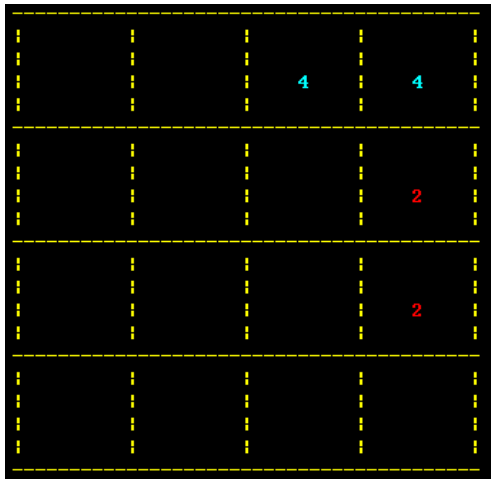


图 2.10　数字可移动 1

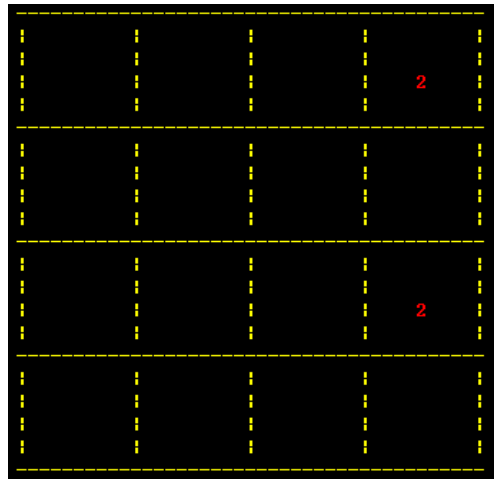


图 2.11　数字可移动 2

判断数字能否移动的功能是通过自定义的 ifMove()函数实现的。该函数使用 for 循环遍历棋盘，然后判断两个相邻位置上的数字。如果相邻位置上的数字满足上面描述的两种情况，则返回 1，表示数字可移动；否则返回 0，表示数字不可移动。ifMove()函数实现代码如下：

```
/**
```

```
 * 判断数字是否可移动。返回 1 表示数字可移动；返回 0 表示数字不可移动
 */
int ifMove(int item[])
{
    int i = 0;
    for(i = 0; i < 3; i ++)
    {
        //如果两个相邻位置上的数字相同，表示该数字可移动，返回 1
        if(item[i] != 0 && item[i] == item[i + 1])
        {
           return 1;
        }
        //如果两个相邻位置上，一个是空格子，一个上有数字，也能移动，返回 1
        if(item[i] == 0 && item[i + 1] != 0)
        {
            return 1;
        }
    }
    return 0;                                          //不能合并，返回 0
}
```

5. 通过键盘控制数字移动

挑战 2048 游戏中，玩家可以通过上、下、左、右键来移动棋盘中的数字，以实现数字的移动与合并。该功能是通过自定义的 keyboardControl()函数实现的，该函数接收一个 key 参数，表示玩家按下的键值。该函数中，主要通过 switch 多分支语句判断玩家按下的方向键，并根据不同方向执行相应的数字移动操作。keyboardControl()函数的实现代码如下：

```
/**
 * 键盘控制移动
 */
int keyboardControl(int key)
{
    int i = 0, j = 0;
    int change = 0;
    int *p;
    int tp[4] = {0, 0, 0, 0};
    switch(key)                                        //LEFT = 75, UP = 72, RIGHT = 77, DOWN = 80
    {
        case 72:                                       //UP，向上键
            j = 0;
            for(i = 0; i < 4; i++)
            {
                tp[0] = BOX[0][i];                     //把一列数移到中间变量
                tp[1] = BOX[1][i];
                tp[2] = BOX[2][i];
                tp[3] = BOX[3][i];
                p = add(tp);                           //获得合并之后的数值
                //判断是否可以移动，可以移动，则新出现一个数字
                if(!ifMove(tp))
                {
                    j++;                               //向上移动
                }
                BOX[0][i] = p[0];                      //把处理好的中间变量移回来
                BOX[1][i] = p[1];
                BOX[2][i] = p[2];
                BOX[3][i] = p[3];
            }
            return j != 4;                             //当 j 不超过 4 时，可以执行 UP 操作
        case 80:                                       //DOWN，向下键
            j  = 0;
```

```
        for(i = 0; i < 4; i++)
        {
            tp[0] = BOX[3][i];
            tp[1] = BOX[2][i];
            tp[2] = BOX[1][i];
            tp[3] = BOX[0][i];
            p = add(tp);
            if(!ifMove(tp))
            {
                j++;
            }
            BOX[3][i] = p[0];
            BOX[2][i] = p[1];
            BOX[1][i] = p[2];
            BOX[0][i] = p[3];
        }
        return j != 4;
    case 75:                                    //LEFT，向左键
        j = 0;
        for(i = 0; i < 4; i++)
        {
            tp[0] = BOX[i][0];
            tp[1] = BOX[i][1];
            tp[2] = BOX[i][2];
            tp[3] = BOX[i][3];
            p = add(tp);
            if(!ifMove(tp))
            {
                j++;
            };
            BOX[i][0] = p[0];
            BOX[i][1] = p[1];
            BOX[i][2] = p[2];
            BOX[i][3] = p[3];
        }
        return j != 4;
    case 77:                                    //RIGHT，向右键
        j = 0;
        for(i = 0; i < 4; i++)
        {
            tp[0] = BOX[i][3];
            tp[1] = BOX[i][2];
            tp[2] = BOX[i][1];
            tp[3] = BOX[i][0];
            p = add(tp);
            if(!ifMove(tp))
            {
                j++;
            }
            BOX[i][3] = p[0];
            BOX[i][2] = p[1];
            BOX[i][1] = p[2];
            BOX[i][0] = p[3];
        }
        return j != 4;
    case 27:                                    //按 Esc 键
        gotoxy(20,23);
        color(12);
        printf("确定退出游戏么? (y/n)");
        char c = getch();                       //获得键盘输入
        if(c == 'y' ||c == 'Y')                 //如果输入的是大写或者小写的 Y
        {
            exit(0);                            //退出游戏
        }
```

```
            if(c == 'n'||c == 'N')                    //如果输入的是大写或者小写的 N
            {
                gotoxy(20,23);
                printf("                          ");      //继续游戏
            }
            break;
        default: return 0;
    }
}
```

6. 判断游戏是否成功

在玩本项目的游戏时，如果棋盘中出现数字 2048，则表示游戏成功，这时程序会自动弹出游戏成功界面，如图 2.12 所示。

图 2.12　游戏成功界面

定义一个 Gamewin()函数，用于判断棋盘中的最大数是否为 2048。如果是，则游戏成功，这时程序会跳转到一个新的界面，在该界面中绘制一个由“■”组成的“WIN”型的字符画，并且将返回值设置为 1，表示游戏成功。Gamewin()函数的实现代码如下：

```
/**
 * 判断是否胜利
 */
int Gamewin()
{
    int flag = 0;
    if(BOXmax() == 2048)                      //如果棋盘中的最大值为 2048，达到目标，则游戏胜利
    {
        system("cls");
        gotoxy(1,6);
        color(2);                             //暗绿色
        //输出胜利 WIN 的字符画
        printf("    ■          ■          ■      ■■■■■      ■          ■  \n");
        gotoxy(1,7);
        printf("    ■          ■■         ■          ■      ■ ■        ■  \n");
        gotoxy(1,8);
        printf("    ■        ■  ■         ■          ■      ■  ■       ■  \n");
        gotoxy(1,9);
        printf("    ■        ■  ■         ■          ■      ■   ■      ■   \n");
        gotoxy(1,10);
        printf("    ■        ■  ■         ■          ■      ■    ■     ■  \n");
        gotoxy(1,11);
```

```
            printf("  ■   ■     ■   ■          ■        ■    ■   ■ \n");
            gotoxy(1,12);
            printf("  ■ ■     ■ ■          ■        ■    ■ ■    \n");
            gotoxy(1,13);
            printf("  ■■      ■■           ■        ■     ■ ■  \n");
            gotoxy(1,14);
            printf("   ■       ■           ■■■■■    ■       ■    \n");
            gotoxy(35,17);
            color(13);
            printf("胜利啦，你真棒！！！");
            flag = 1;
        }
        return flag;                                    //flag 的值默认是 0，返回 1，则表示游戏成功
}
```

上面代码使用了一个 BOXmax()函数，该函数为自定义函数，主要用于获取棋盘中的最大数，代码如下：

```
/**
 * 返回棋盘中的最大数
 */
int BOXmax()
{
    int max = BOX[0][0];                                //初始化 BOX 数组
    int i,j;
    for(i = 0;i < 4;i ++)                               //遍历整个数组
    {
        for(j = 0;j < 4;j ++)
        {
            if(BOX[i][j] > max)                         //如果数组中有数值大于 max 的值
            {
                max = BOX[i][j];                        //将数组中的值赋值给 max，这样找出数组中的最大数
            }
        }
    }
    return max;                                         //返回 max 的值，也就是当前棋盘中的最大值
}
```

7. 判断游戏是否失败

在玩本游戏时，如果游戏界面被数字填满，并且不能再进行移动或合并，则表示游戏失败。游戏失败后，玩家将看到如图 2.13 所示的界面。

图 2.13　游戏失败界面

定义一个 Gamefaile()函数，用于绘制游戏失败时的界面，并将返回值设置为 0，以表示游戏失败。Gamefaile()函数的实现代码如下：

```
/**
 * 判断是否失败，并输出棋盘中的最大数
 */
int Gamefaile()
{
	int flag = 0;
	int max;
	//当上下左右都不能移动时，游戏失败
	if(ifup() + ifdown() + ifleft() + ifright() == 0)
	{
		system("cls");
		gotoxy(34,3);
		color(14);
		printf("合并出的最大数是：");
		gotoxy(52,3);
		color(11);
		max = BOXmax();
		printf("%d",max);
		gotoxy(19,6);
		color(4);                                                    //暗红色
		printf("  ■■■■■   ■     ■   ■■    \n");    //输出 END 字符画
		gotoxy(19,7);
		printf("  ■       ■■    ■   ■ ■   \n");
		gotoxy(19,8);
		printf("  ■       ■ ■   ■   ■  ■  \n");
		gotoxy(19,9);
		printf("  ■       ■  ■  ■   ■   ■ \n");
		gotoxy(19,10);
		printf("  ■■■■    ■   ■ ■   ■    ■ \n");
		gotoxy(19,11);
		printf("  ■       ■    ■■   ■    ■ \n");
		gotoxy(19,12);
		printf("  ■       ■     ■■  ■   ■  \n");
		gotoxy(19,13);
		printf("  ■       ■      ■■  ■  ■  \n");
		gotoxy(19,14);
		printf("  ■■■■■   ■       ■■  ■■   \n");
		gotoxy(34,17);
		color(13);
		printf("无法移动，游戏失败！");                                  //提示文字
		flag = 1;
	}
	return flag;                                                     //flag 的值默认是 0，返回 1，则表示游戏失败
}
```

上面代码使用了 ifup()、ifdown()、ifleft()和 ifright() 4 个函数，这些函数分别用于判断指定的数字是否能够在向上、向下、向左、向右 4 个方向上进行移动。它们的实现代码如下：

```
/**
 * 判断是否能够上移。如果可以上移，返回 1；如果不能上移，则返回 0
 */
int ifup()
{
	int i,j;
	int flag = 0;                                                    //定义标志变量，只有 0 或 1
	for(j = 0;j < 4;j ++)
		for(i = 0;i < 3;i ++)
		{
```

```
                //如果上下两个格子相加是 2 的 n 次方，并且下面的格子中有数
                if((if2n(BOX[i][j] + BOX[i+1][j]) == 1) && BOX[i+1][j])
                {
                        flag = 1;                                       //可以上移
                }
        }
        return flag;                                                    //返回 1，表示可以上移；返回 0，表示不能上移
}

/**
 * 判断是否能够下移。如果可以下移，则返回 1；不能下移，则返回 0
 */
int ifdown()
{
        int i,j;
        int flag = 0;
        for(j = 0;j < 4;j ++)
                for(i = 3;i > 0;i --)
                {
                        //如果上下两个格子相加是 2 的 n 次方，并且上面的格子中有数
                        if((if2n(BOX[i][j] + BOX[i-1][j]) == 1) && BOX[i-1][j])
                        {
                                flag = 1;                               //可以下移
                        }
                }
        return flag;                                                    //返回 1，表示可以下移；返回 0，表示不能下移
}

/**
 * 判断是否能够左移。如果可以左移，则返回 1；如果不能左移，则返回 0
 */
int ifleft()
{
        int i,j;
        int flag = 0;
        for(i = 0;i < 4;i ++)
                for(j = 0;j < 3;j ++)
                {
                        //如果左右两个格子相加是 2 的 n 次方，并且右面的格子中有数
                        if((if2n(BOX[i][j] + BOX[i][j+1]) == 1) && BOX[i][j+1])
                        {
                                flag = 1;                               //可以左移
                        }
                }
        return flag;                                                    //返回 1，表示可以左移；返回 0，表示不能左移
}

/**
 * 判断是否能够右移。如果可以右移，则返回 1；如果不能右移，则返回 0
 */
int ifright()
{
        int i,j;
        int flag = 0;
        for(i = 0;i < 4;i ++)
                for(j = 3;j > 0;j --)
                {
                        //如果左右两个格子相加是 2 的 n 次方，并且左面的格子中有数
                        if((if2n(BOX[i][j] + BOX[i][j-1]) == 1) && BOX[i][j-1])
                        {
                                flag = 1;                               //可以右移
                        }
```

```
            }
            return flag;                                    //返回 1，表示可以右移，返回 0，表示不能右移
}
```

上面 4 个函数中都使用了一个 if2n()函数，该函数为自定义函数，主要用于判断传入的参数 x 是否为 2 的 n 次方。如果是，那么该函数返回 1；否则，返回 0。if2n()函数的实现代码如下：

```
/**
 * 判断 x 是否是 2 的 n 次方
 */
int if2n(int x)
{
        int flag = 0;
        int n;
        int N = 1;
        for(n = 1;n <= 11;n++)                              //2 的 11 次方是 2048，游戏目标是达到 2048
        {
                if(x == pow(2,n))                           //计算 2 的 n 次方
                {
                        flag = 1;
                        if(n>N)
                                N = n;
                        return flag;
                }
        }
        return flag;
}
```

2.6.3 开始游戏功能的实现

在实现了游戏逻辑功能相关的函数后，我们定义一个 Gameplay()函数，该函数主要用于实现开始游戏功能。该函数首先调用自定义的 drawTheGameBox()函数绘制游戏界面；然后使用随机函数 rand()在游戏棋盘的随机空格位置上显示初始数字 2 或者 4；接下来使用 conio.h 库文件中的 kbhit()函数来检测玩家是否按下键盘上的方向键，并通过调用自定义的 keyboardControl()函数控制数字的移动和合并；最后调用自定义的 Gamewin()函数和 Gamefaile()函数判断游戏的成功或者失败，并在判断完成后，询问玩家是否重玩或是退出游戏。Gameplay()函数实现代码如下：

```
/**
 * 开始游戏
 */
void Gameplay()
{
    system("cls");                          //清屏
    int i = 0, j = 0;
    gameTime = time(NULL);                  //获取当前时间为开始时间
    drawTheGameBox();                       //绘制游戏界面
    int a,b;                                //BOX[][]数组的横纵坐标
    srand(time(NULL));                      //设置随机数种子，初始化随机数
    do
    {
        a = rand()%4;                       //获得 4×4 棋盘中的随机位置
        b = rand()%4;
    }while(BOX[a][b]!=0);                   //一直到棋盘中没有空格
    if(rand() % 4 == 0)                     //2 或 4 随机出现在空格处（最开始出现在棋盘上的 2 或 4）
    {
        BOX[a][b] = 4;
    }
```

```
else
{
    BOX[a][b] = 2;
}
for(i = 0; i < 4; i ++)                          //遍历整个网格
{
    for(j = 0; j < 4; j ++)
    {
        if(BOX[i][j] == 0)                       //如果网格中有空位，就继续下去
        {
            continue;
        }
        gotoxy(15 + j * 10 + 5, 2 + i * 5 + 3);  //设置棋子显示位置
        int c = BOX[i][j];                       //获得棋盘上 BOX[i][j]上的数字
        TextColors(c);                           //设置棋子的颜色，不同数字显示不同颜色
        printf("%d", c);                         //打印棋子
    }
}
while(1)
{
    //kbhit()检查当前是否有键盘输入。如果有则返回 1；否则返回 0
    while (kbhit())
    {
        //如果按下的按键不是在 keyboardControl()函数中定义的，会没有反应，直到按下定义的按键
        if(!keyboardControl(getch()))
        {
            continue;
        }
        drawTheGameBox();                        //绘制棋盘
        for(i = 0; i < 4; i ++)                  //循环整个 4×4 的棋盘
        {
            for(j = 0; j < 4; j ++)
            {
                if(BOX[i][j] == 0)               //如果棋盘中有空位，则可一直进行按键
                {
                    continue;
                }
                gotoxy(15 + j * 10 + 5, 2 + i * 5 + 3);//合并后的数出现的位置
                int c = BOX[i][j];
                TextColors(c);
                printf("%d", c);
            }
        }
        do{
            a = rand()%4;
            b = rand()%4;                        //获得随机位置
        }while(BOX[a][b]!=0);
        if(rand() % 4 == 0)                      //2 或 4 随机出现在空格处（进行方向操作合并之后，在空白处出现）
        {
            BOX[a][b] = 4;                       //随机位置上设置为 4
        } else {
            BOX[a][b] = 2;                       //随机位置上设置为 2
        }
        step++;                                  //进行计步
        gotoxy(15 +  b * 10 + 5, 2 + a * 5 + 3); //随机出现的 2 或 4
        int c = BOX[a][b];
        TextColors(c);
        printf("%d", c);
    }
    //只要 Gamefaile()或者 Gamewin()任意一个函数返回 1，也就是成功或是失败都会出现下面的内容
    if(Gamefaile()+Gamewin() != 0)
    {
```

```
            int n;
            gotoxy(20,20);
            color(12);
            printf("我要重新玩一局-------1");
            gotoxy(45,20);
            printf("不玩了，退出吧-------2\n");
            gotoxy(43,21);
            color(11);
            scanf("%d", &n);
            switch (n)
            {
                case 1:
                    Replay();                          //重玩游戏
                    break;
                case 2:
                    close();                           //关闭游戏
                    break;
            }
        }
    }
}
```

2.6.4 重玩或退出游戏

在 2.6.3 节的 Gameplay()函数中，当游戏成功或者失败时，会提醒用户选择重玩或退出游戏，这主要是通过自定义的 Replay()函数和 close()函数来实现的。其中，Replay()函数用于实现重玩游戏的功能，该函数首先通过 system()函数清除屏幕，并初始化分数、步数、BOX 棋盘数组，然后调用 Gameplay()函数重新开始游戏。Replay()函数的实现代码如下：

```
/**
 * 重新游戏
 */
void Replay()
{
    system("cls");                                     //清屏
    score = 0,step = 0;                                //分数、步数归零
    memset(BOX,0,16*sizeof(int));                      //初始化 BOX 数组
    Gameplay();                                        //开始游戏
}
```

close()函数用于实现退出游戏功能，其代码如下：

```
/**
 * 退出
 */
void close()
{
    exit(0);
}
```

2.7 游戏规则介绍界面设计

2.7.1 游戏规则介绍界面概述

在游戏欢迎界面输入数字 2，并按 Enter 键，即可进入游戏规则介绍界面，该界面主要以不同的文字颜

色显示游戏规则，如图 2.14 所示。

图 2.14　游戏规则介绍界面

2.7.2　游戏规则介绍的实现

定义一个 regulation()函数，用于绘制游戏规则介绍界面。该函数首先使用两个嵌套的 for 循环来绘制边框，然后通过调用自定义的 color()函数和 gotoxy()函数来确定规则介绍文字的颜色和位置，并使用 printf()函数输出相应的文字。regulation()函数的实现代码如下：

```
/**
 * 游戏规则介绍
 */
void regulation()
{
    int i,j = 1;
    system("cls");
    color(13);
    gotoxy(34,3);
    printf("游戏规则");
    color(2);
    for (i = 6; i <= 18; i++)                                    //输出上下边框===
    {
        for (j = 15; j <= 70; j++)                               //输出左右边框||
        {
            gotoxy(j, i);
            if (i == 6 || i == 18)
                printf("=");
            else if (j == 15 || j == 69)
                printf("||");
        }
    }
    color(3);
    gotoxy(18,7);
    printf("tip1: 玩家可以通过↑、↓、←、→方向键来移动方块");
    color(10);
    gotoxy(18,9);
    printf("tip2: 按 ESC 退出游戏");
    color(14);
    gotoxy(18,11);
    printf("tip3: 玩家选择的方向上,若有相同的数字则合并");
```

```
    color(11);
    gotoxy(18,13);
    printf("tip4: 每移动一步，空位随机出现一个 2 或 4");
    color(4);
    gotoxy(18,15);
    printf("tip5: 棋盘被数字填满，无法进行有效移动，游戏失败");
    color(5);
    gotoxy(18,17);
    printf("tip6: 棋盘上出现 2048，游戏胜利");
    getch();                                                    //按任意键返回游戏欢迎界面
    system("cls");
    main();
}
```

2.8 游戏按键说明功能设计

2.8.1 游戏按键说明功能概述

在游戏欢迎界面输入数字 3，并按 Enter 键，游戏欢迎界面的下方将会显示游戏的按键说明，如图 2.15 所示。

图 2.15 游戏按键说明效果

2.8.2 游戏按键说明的实现

定义一个 explanation()函数，该函数用于以不同的颜色显示游戏按键的说明文字，其实现代码如下：

```
/**
 * 按键说明
 */
void explanation()
{
      gotoxy(20,22);
```

```
        color(13);
        printf("①、↑、↓、←、→方向键进行游戏操作！");
        gotoxy(20, 24);
        printf("②、Esc 键退出游戏");
        getch();                                                    //按任意键返回游戏欢迎界面
        system("cls");
        main();
}
```

2.9 项 目 运 行

通过前述步骤，我们成功设计并完成了“挑战 2048”游戏项目的开发。接下来，我们运行该游戏，以检验我们的开发成果。如图 2.16 所示，在 Dev-C++开发工具中，选择菜单栏中的“运行”→“编译运行”菜单项，即可成功运行程序。

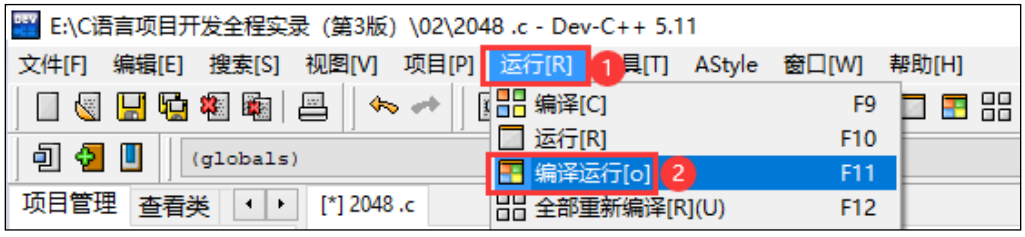


图 2.16 编译运行“挑战 2048”游戏项目

挑战 2048 游戏运行效果如图 2.17 所示，用户通过在图 2.17 中输入相应的菜单编号即可执行操作。

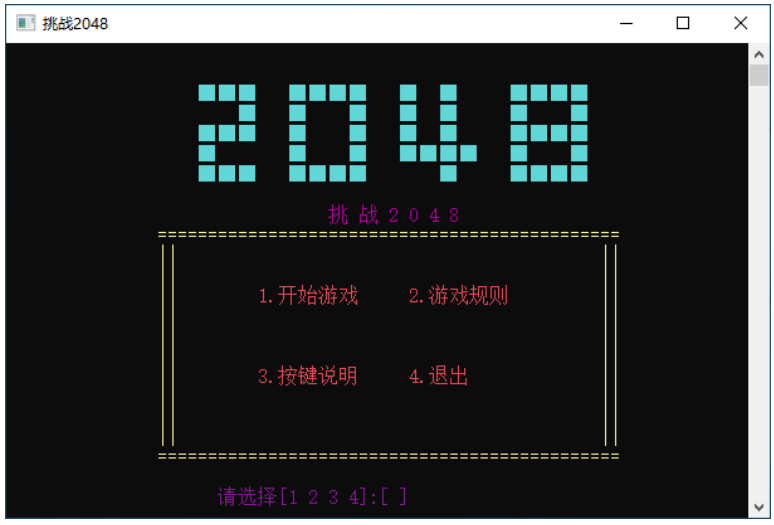


图 2.17 挑战 2048 游戏

说明

通过在 Dev C++开发工具中选择“编译运行”菜单项，我们可以在项目的根目录中生成与项目同名的.exe 可执行文件，用户直接双击该文件即可运行程序。

本章主要使用 C 语言的输入输出函数、数组、指针函数、system()函数、控制台设置函数等技术，开发了一款名为“挑战 2048”的游戏。这款游戏具有规则简单、容易上手、趣味性强等特点，其商业版广受大众喜爱，因此具有很高的学习价值。同时，该游戏中的合并相同数字算法非常值得读者深入学习和反复实践。

2.10 源 码 下 载

本章虽然详细地讲解了如何编码实现“挑战 2048”游戏的各个功能，但给出的代码都是代码片段，而非完整源码。为了方便读者学习，本书提供了完整的项目源码，读者只需扫描右侧的二维码，即可下载这些源码。

源码下载