同济大学 实验报告

数独游戏

及数独的自动求解

1550431

王甯琪

计科三班

完成日期： 2017-4-9

1. 题目要求

本次作业要求制作一个程序，其中有三个菜单项。

第一个菜单项要求能够读取当前目录下所有符合 sudoku*.txt 格式的文件，并展示出来供用户选择。在用户选择完以后，需要能够读取用户选择的文件，并将其展示在cmd窗口中。

文件中给出的数需要用一种颜色标记，没有给出数的位置要用另一种颜色标记。如果文件有冲突需要提示用户修改文件，文件不合法也需要提示。

在进入游戏后，读取用户的输入并对数独进行操作。对于文件中已经固定的一些数不能修改，用户只能对于文件中没有给定数的位置进行操作。

在每一次操作后需要对数独是否产生冲突、游戏是否结束进行判断，冲突需要用特定颜色标出。还要支持用户选择步数回退时回退到之前任何一个状态。

第二个菜单项在第一个菜单项的基础上，需要进行伪图形界面输出。同时文件选择需要制作一个可以用键盘的上下键进行移动的菜单。

菜单项三需要实现读入文件后自动搜寻数独解决方案，并定时输出数独的解决状态。整个过程需要使用伪图形界面输出。

2. 整体设计思路

考虑整体的需求，可以将程序依据执行顺序以及功能要求划分成几个区块，按顺序执行区块即可完成题目。

具体来说，有以下区块。

2.1.读取文件列表，获取用户选择

在读取当前目录后，扫描符合要求的文件，并在屏幕上输出。待用户输入后返回，并尝试打开用户指定的文件。

如果文件打开成功即可运行接下来的操作。若不成功则直接结束本次游戏，并告知用户。

2.2.判断数独冲突以及完成情况

遍历每一行、每一列、每一宫。每次选取九个位置（来自同一行、列、宫）进行判断，如果九个位置对应的数分别对应1-9，则代表这九个位置不产生冲突。

如果任意一行、一列或一宫产生冲突，则需要对冲突位置进行标注，并告知用户。

如果所有行、列、宫都不产生冲突，则说明游戏结束。

2.3.用户指令读入以及步数操作

在游戏没有结束的前提下，需要得到用户的输入（对某一位置进行赋值或是步数回退）。

对于赋值操作，需要检测用户指定位置是否能够被赋值。如果该位置的值在是从文件中读入的，则赋值不成功。赋值完成后需要对记录步数的链表进行操作，即申请一个新的步数记录空间，将此次赋值操作记录下来，并将这个空间链接到原有链表上。

如果是选择步数回退操作，择需要读取链表中记录的最后一次的步数操作，将此次操作复原，并删除链表中的最后一项。

在执行完成后，需要再次判断数独冲突以及完成状况。

2.4.绘制图形

根据菜单项的不同，绘制简易图形/复杂伪图形。

在绘图时，需要检测该位置是由文件读入还是由用户操作填入，该位置是否是引起行、列、宫冲突的位置，冲突发生所在的行、列、宫等等。并用不同的颜色进行绘制，以提示用户。

3. 主要功能实现

根据程序实现的思路，便可以根据每一区块和步骤的要求写出代码。整体的框架是循环，每次一循环要求用户输入所选的菜单项或者选择退出循环、结束程序。

其中数独由一个二维数组map[9][9]来表示。每一个元素的类型都是自定义结构体point。其中step包括该位置的值value，是否由文件读入的changeable,冲突源标记conflict，是否需要高亮输出highlight等等。

步数由链表记录。链表中的每一个成员的类型是自定义结构体step，记录着每一次操作的对象position，修改前的数值change_from，储存上一步、下一步操作的成员地址previous、next。

3.1.读取文件列表，获取用户选择

可以使用DOS命令dir来获取当前目录下所有文件名，并指定其写入a.txt文件中。之后对于a.txt中的每一项进行判断，符合 sudoku*.txt 要求的文件则输出其名称到屏幕上。最后待所有符合要求文件输出后，则进入用户选择环节。最后需要用命令 del a.txt 删除产生的文件。

用户通过键盘键入文件名。确定好文件后，则尝试打开文件。如果文件打开成功则进入接下来的步骤，如果打开不成功则结束本次大循环。

3.2.判断数独冲突以及完成情况

生成一个记录9个位置的地址的数组。每一次填入一列、一行或者一宫的九个元素的地址。

循环调用判断这9个位置是否产生冲突的函数，以判断每一行、列、宫是否有冲突。具体的实现思路是使用一个int型数组，大小为9，初值全为0，读取九个位置的数值，并对int型数组进行操作。比如说读入第一个位置的数值5后，则对int型数组中的第五个元素的值进行自增操作。最后检测int型数组是否每一个元素都是1。如果不是的话则说明产生冲突。对于产生冲突的情况，需要查找九个位置中的冲突源。

如果遍历所有的行、列、宫后，没有任何冲突发生且所有位置都填上了数，则提示游戏结束，并结束此次大循环。

在判断完以后调用绘制图形的函数。

3.3.用户指令读入以及步数操作

在游戏没有结束的前提下，需要得到用户的输入（对某一位置进行赋值或是步数回退）。

对于赋值操作，需要检测用户指定位置是否能够被赋值。如果该位置的changeable为0，则代表该位置的数值由文件读入，不能修改，若为0则可以修改。赋值完成后需要对记录步数的链表进行操作，即申请一个新的步数记录空间，将此次赋值操作记录下来，并将这个空间链接到原有链表上。

如果是选择步数回退操作，择需要读取链表中记录的最后一次的步数操作。转移到position所指向的位置，并将change_from记录的修改前的数值赋值给当前位置。

在执行完成后，需要再次判断数独冲突以及完成状况。该区块和判断数独冲突以及完成情况的区块来回调用。形成一个小循环。

3.4.绘制图形

根据菜单项的不同，绘制简易图形/复杂伪图形。

在绘制每一个数独位置时，需要考虑conflict,changeable,highlight等成员的值，从而决

定最终输出的颜色。

3.5.自动求解

对于菜单项三，需要自动求解。

首先需要运用摒除法循环判断每一个位置是否有唯一可填入值。如果有的话则填入。如果没有则跳出摒除法填值循环。

生成一个数组，记录所有仍然为0的位置的地址。

从数组的第一个地址开始，每进入一个尚未填值的位置则通过摒除法生成一个该位置可以填入的所有数值，填入第一个可行值后进入数组中下一个地址。如果到达数组中最后一个地址，则尝试填入所有可行数，并判断游戏是否结束。如果到达末尾且遍历完所有可行值后游戏仍然没有结束，或者到达某一位置时没有任何值可以填入，则回溯到上一个可行值没有遍历完的位置，填入下一个可行值，再进入下一个地址。

使用摒除法以及深度优先算法进行尝试。

4. 碰到的问题

4.1.递归爆栈

最开始尝试的是使用函数递归的方式来完成深度优先算法，对于简单的数独可以解决。但是对于复杂的数独，如 sudoku-hard*.txt 系列则会爆栈。

最后选择在算法不变的情况下，将递归改成循环。最终解决问题。

5. 心得体会

5.1.使用递归时需要考虑是否爆栈的问题

由于深度优先等算法可能会导致递归过于庞大，所以需要考虑当前算法是否适合使用递归。如果不适合使用递归的话则可尝试采用循环。

5.2.找BUG时需要好好利用VS的自带功能

在自动求解的回溯过程中，总是会弹窗。一遍遍看代码也找不出问题，最后是利用了断点和变量监视的方法找到了步数链表删除时发生的错误。

使用VS的调试工具能加快效率，提高准确度。

6. 源程序

```
struct point
{
	int value = 0;
	int changable = 1;
	int conflict = 0;
	int highlight = 0;
	int all_fillable = 0;
	int now_fill = 0;
	int *possible = NULL;
};
```

```
struct step
{
	int position = 1;
	int change_from;
	step *previous = NULL;
	step *next = NULL;
};

//第三小题
{
	flag = first_operation(*map, pos, fillable);
	if (flag == -1)
	{
		end_until_end(0, 0);
		return;
	}
	else if (flag == 0)//游戏完成
	{
		system("cls");
		draw_file_choice(choice_X, choice_Y, from, highlight, file_name);
		draw_complex_sudoku(*map, complex_X, complex_Y, branch - 1);
		cout << "\n 游戏结束\n";
		end_until_end(0, 0);
		return;
	}

	//未完成游戏
	draw_file_choice(choice_X, choice_Y, from, highlight, file_name);
	draw_complex_sudoku(*map, complex_X, complex_Y, branch - 1);
	gotoxy(hout, complex_X, 0);

	flag = make_this_pos_possible_shuzu(*map, pos[0]);
	if (flag == -1)//内存申请失败
	{
		end_until_end(0, 0);
		return;
	}
	else if (flag == 0)//第一个空就没有可填入的数
	{
		system("cls");
		draw_file_choice(choice_X, choice_Y, from, highlight, file_name);
		draw_complex_sudoku(*map, complex_X, complex_Y, branch - 1);

		cout << "\n 该数独无解\n";
		delete pos;
		end_until_end(0, 0);
		return;
	}

	flag = auto_steps(*map, s_head, pos, fillable, complex_X, complex_Y);
	if (flag == -1)//内存申请失败
	{
		delete pos;
		end_until_end(0, 0);
		return;
	}
	else if (flag == 0)
	{
		cout << "\n 该数独无解\n";
		delete pos;
		end_until_end(0, 0);
		return;
	}

	//退出循环，释放所申请的空间
	system("cls");
	draw_file_choice(choice_X, choice_Y, from, highlight, file_name);
	draw_complex_sudoku(*map, complex_X, complex_Y, branch - 1);
```

```
	cout << "\n 游戏结束\n";
	free_possible(*map);
	free_space(s_head);
	delete pos;

	end_until_end(0, 0);
	return;
}
int first_operation(point *head, int *&pos, int &fillable)//找出能通过排除法确定的所有数独位置并填入 1-9。完成后生成一个记录没有填入数字的位置的数组。返回-1 代表申请空间错误
{
	int i, j;
	first_of_all_fill_in(head);// 找出所有可以初步填入的位置，并填入答案（摒除法）

	fillable = 0;
	for (i = 0; i<h_max*l_max; i++)
	{
		if (!(head + i)->value)
			fillable++;
	}

	if (!fillable)//全被填满
		return 0;

	pos = new(nothrow) int[fillable];
	if (pos == NULL)
	{
		system("cls");
		cout << "\n 内存申请失败";
		return -1;
	}

	//内存申请成功
	j = 0;
	for (i = 0; i<h_max*l_max; i++)
	{
		if (!(head + i)->value)
		{
			pos[j] = i;
			j++;
		}
	}

	return 1;
}
void draw_file_choice(const int X, const int Y, const int from, const int highlight, char file_name[100][50])//输出可选文件菜单
{
	int k;

	gotoxy(hout, X, Y);
	cout << "数独样本文件";
	gotoxy(hout, X, Y + 1);
	cout << "╔══════════════════════════╗";
	for (k = 1; k <= 8; k++)//一次显示 8 个文件
	{
		gotoxy(hout, X, Y + 1 + k);
		cout << "║                          ║";
	}
	gotoxy(hout, X, Y + 10);
	cout << "╚══════════════════════════╝";

	for (k = 1; k <= 8; k++)//一次显示 8 个文件
	{
		gotoxy(hout, X + 2, Y + 1 + k);
		if (k == highlight)
		{
```

```
                setcolor(hout, 7, 0);
                cout << file_name[from + k - 2];
                setcolor(hout, COLOR_BLACK, COLOR_WHITE);
            }
            else
                cout << file_name[from + k - 2];
        }
}
void draw_complex_sudoku(point *head, const int X, const int Y, int goxyornot)
{
    //在X,Y处输出图形化的数独分布情况
    int i, j, color;
    point *pos;

    if (goxyornot)
        gotoxy(hout, X, Y);

    //1 代表列的字母(a-i)输出在(X+6*j-2,Y)
    //2 代表行的数字(1-9)输出在(X,Y+3*i-1)
    //3 i 行 j 列的方框左上角坐标是(X+2+(j-1)*6,Y+3*i-2)

    //1
    for (j = 1; j <= 9; j++)
    {
        gotoxy(hout, X + 6 * j - 2, Y);
        cout << char('a' + j - 1);
    }

    //2
    for (i = 1; i <= 9; i++)
    {
        gotoxy(hout, X, Y + 3 * i - 1);
        cout << i;
    }

    //3
    for (i = 1; i <= 9; i++)
    {
        for (j = 1; j <= 9; j++)
        {
            pos = head + 9 * (i - 1) + j - 1;
            if (pos->changable)
                color = 14;
            else
                color = 9;

            if (pos->conflict)
                color = 10;

            setcolor(hout, pos->highlight ? 6 : 0, color);
            gotoxy(hout, X + 2 + (j - 1) * 6, Y + 3 * i - 2);
            cout << "╔═══╗";
            gotoxy(hout, X + 2 + (j - 1) * 6, Y + 3 * i - 1);
            cout << "║" << pos->value << " ║";
            gotoxy(hout, X + 2 + (j - 1) * 6, Y + 3 * i);
            cout << "╚═══╝";
        }
    }

    setcolor(hout, COLOR_BLACK, COLOR_WHITE);
    gotoxy(hout, X, Y + 29);
}
int make_this_pos_possible_shuzu(point *head, int pos)//自动求解过程中，找出某个位置能填入的数字,pos 0-80。返回0代表无可填入数字,-1代表空间申请失败
{
    int h, l, possible[10] = { 1,1,1,1,1,1,1,1,1 }, i, j, k, counts = 0;
    point *now;
    h = 1 + (pos / 9);
```

```
	l = pos + 1 - 9 * (h - 1);
	//该位是 h 行 l 个元素

	//行
	now = head + 9 * h - 9;//指向该行第一个元素
	for (i = 0; i<9; i++)
	{
		if ((now + i)->value)
			possible[(now + i)->value] = 0;
	}

	//列
	now = head + l - 1;//指向该列第一个元素
	for (i = 0; i<9; i++)
	{
		if ((now + 9 * i)->value)
			possible[(now + 9 * i)->value] = 0;
	}

	//宫
	now = head + ((h - 1) / 3) * 27 + ((l - 1) / 3) * 3;//指向该宫第一个元素
	for (i = 0; i<3; i++)
	{
		for (j = 0; j<3; j++)
		{
			if ((now + 9 * i + j)->value)
				possible[(now + 9 * i + j)->value] = 0;
		}
	}

	for (i = 1; i <= 9; i++)
	{
		if (possible[i])
			counts++;
	}

	if (counts)
	{
		(head + pos)->all_fillable = counts;

		(head + pos)->possible = new(nothrow) int[counts];
		if ((head + pos)->possible == NULL)
		{
			system("cls");
			cout << "\n 内存申请失败";
			return -1;
		}

		k = 0;
		for (i = 1; i <= 9; i++)
		{
			if (possible[i])
			{
				*((head + pos)->possible + k) = i;
				k++;
			}
		}

		return 1;
	}

	return 0;
}

int auto_steps(point *head, step * s_head, int* pos, int fillable, const int X, const int Y)//自动走步并更改，返回-1
代表申请空间失败,0 代表没有解,1 表示完成
{
	int now_pos = 1, flag, i, times = 0, counts = 0;
```

```
step *s_now = s_head;

while (1)
{
	times++;

	if (times == 1000)
	{
		times = 0;
		counts++;
		system("cls");
		draw_complex_sudoku(head, X, Y, 1);
		gotoxy(hout, X, Y + 29);
		cout << "搜索次数:" << counts;
	}

	if (now_pos == fillable)//到达最后一个
	{
		flag = make_this_pos_possible_shuzu(head, pos[now_pos - 1]);
		if (flag == -1)
			return -1;
		else if (flag == 1)//normally get
		{
			for (i = 0; i<(head + pos[now_pos - 1])->all_fillable; i++)
			{
				(head + pos[now_pos - 1])->value = *((head + pos[now_pos - 1])->possible + (head + pos[now_pos - 1])->now_fill);
				(head + pos[now_pos - 1])->now_fill++;

				if (game_over(head))
					return 1;//find it!
			}

			//未找到，置零、回溯
			while ((head + pos[now_pos - 1])->now_fill == (head + pos[now_pos - 1])->all_fillable)//如果这一位已经填完或者无数可填的话
			{
				if (now_pos == 1)
					return 0;//无解

				this_pos_set_0(head + pos[now_pos - 1]);//将这一位清零
				while ((s_now->previous)->position == now_pos)//步数记录回溯，清除
				{
					s_now = s_now->previous;
					delete s_now->next;
					s_now->next = NULL;
				}

				now_pos--;
			}

			//现在 now_pos 对应的位置可以取下一个可行值了
			//记录这一步的操作
			s_now->position = now_pos;
			s_now->change_from = (head + pos[now_pos - 1])->value;

			//对该位置进行数值更改
			(head + pos[now_pos - 1])->value = *((head + pos[now_pos - 1])->possible + (head + pos[now_pos - 1])->now_fill);
			(head + pos[now_pos - 1])->now_fill++;

			//对步数链表进行操作
			s_now->next = new(nothrow) step;
			if (s_now->next == NULL)
			{
				system("cls");
				cout << "\n 内存申请失败";
				return -1;
```

```
                    }
                    (s_now->next)->previous = s_now;
                    s_now = s_now->next;

                    //位置下移
                    now_pos++;
               }
               else //flag==0
               {
                    while ((head + pos[now_pos - 1])->now_fill == (head + pos[now_pos - 1])->all_fillable)//如果这一位已经填完或者无数可填的话
                    {
                         if (now_pos == 1)
                              return 0;//无解

                         this_pos_set_0(head + pos[now_pos - 1]);//将这一位清零
                         while ((s_now->previous)->position == now_pos)//步数记录回溯，清除
                         {
                              s_now = s_now->previous;
                              delete s_now->next;
                              s_now->next = NULL;
                         }

                         now_pos--;
                    }

                    //现在 now_pos 对应的位置可以取下一个可行值了
                    //记录这一步的操作
                    s_now->position = now_pos;
                    s_now->change_from = (head + pos[now_pos - 1])->value;

                    //对该位置进行数值更改
                    (head + pos[now_pos - 1])->value = *((head + pos[now_pos - 1])->possible + (head + pos[now_pos - 1])->now_fill);
                    (head + pos[now_pos - 1])->now_fill++;

                    //对步数链表进行操作
                    s_now->next = new(nothrow) step;
                    if (s_now->next == NULL)
                    {
                         system("cls");
                         cout << "\n 内存申请失败";
                         return -1;
                    }
                    (s_now->next)->previous = s_now;
                    s_now = s_now->next;

                    //位置下移
                    now_pos++;
               }
          }
          else//未达最后一个
          {
               flag = make_this_pos_possible_shuzu(head, pos[now_pos - 1]);
               if (flag == -1)
                    return -1;
               else if (flag == 0)//无数可填
               {
                    //int a, b;
                    //a = now_pos;
                    while ((head + pos[now_pos - 1])->now_fill == (head + pos[now_pos - 1])->all_fillable)//如果这一位已经填完或者无数可填的话
                    {
                         if (now_pos == 1)
                              return 0;//无解

                         this_pos_set_0(head + pos[now_pos - 1]);//将这一位清零
                         while ((s_now->previous)->position == now_pos)//步数记录回溯，清除
```

```
                    {
                        s_now = s_now->previous;
                        delete s_now->next;
                        s_now->next = NULL;
                    }

                    now_pos--;
                }

                /*b= now_pos;
                gotoxy(hout, 0, 0);
                cout << a << ' ' << b<<' ';
                cout << (s_now->previous)->position;
                getchar();*/

                //现在 now_pos 对应的位置可以取下一个可行值了
                //记录这一步的操作
                s_now->position = now_pos;
                s_now->change_from = (head + pos[now_pos - 1])->value;

                //对该位置进行数值更改
                (head + pos[now_pos - 1])->value = *((head + pos[now_pos - 1])->possible + (head + pos[now_pos -
1])->now_fill);
                (head + pos[now_pos - 1])->now_fill++;

                //对步数链表进行操作
                s_now->next = new(nothrow) step;
                if (s_now->next == NULL)
                {
                    system("cls");
                    cout << "\n 内存申请失败";
                    return -1;
                }
                (s_now->next)->previous = s_now;
                s_now = s_now->next;

                //位置下移
                now_pos++;
            }
            else//正常得到 possible 数组
            {
                //记录这一步的操作
                s_now->position = now_pos;
                s_now->change_from = (head + pos[now_pos - 1])->value;

                //对该位置进行数值更改
                (head + pos[now_pos - 1])->value = *((head + pos[now_pos - 1])->possible + (head + pos[now_pos -
1])->now_fill);
                (head + pos[now_pos - 1])->now_fill++;

                //对步数链表进行操作
                s_now->next = new(nothrow) step;
                if (s_now->next == NULL)
                {
                    system("cls");
                    cout << "\n 内存申请失败";
                    return -1;
                }
                (s_now->next)->previous = s_now;
                s_now = s_now->next;

                //位置下移
                now_pos++;
            }
        }
    }
}
```