**浙江工商大学计算机与信息工程学院**

**数据结构实验大作业报告**

**专 业： 计算机科学与技术**

**班 级： 计科1704**

**学 号： 1712190426**

**姓 名： 钱非凡**

**指导教师： 谢满德**

**2017年5月**

一、问题描述

1、编写一个五子棋的游戏程序：

实现人与机对下的功能。

2、要求：

（1）要有棋盘。

（2）设计输、赢判断规则函数。

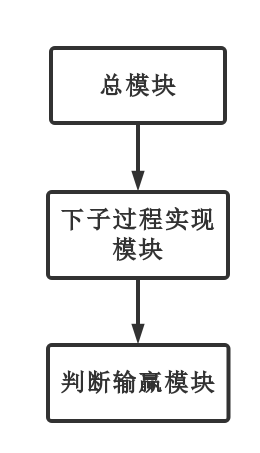
（3）给出下棋过程。

二、设计

1、数据结构设计和核心算法设计描述

根据五子棋游戏程序设计的要求，将该程序分成4个模块：棋盘输出模块、下棋过程实现模块、判断输赢模块、电脑下子实现模块。在此基础上美化了欢迎界面，兼容人机和人人对战，增加了悔棋功能、并且在Unix下实现了getch()和kbhit()函数。

游戏功能结构图如下：



2、主控及功能模块层次结构

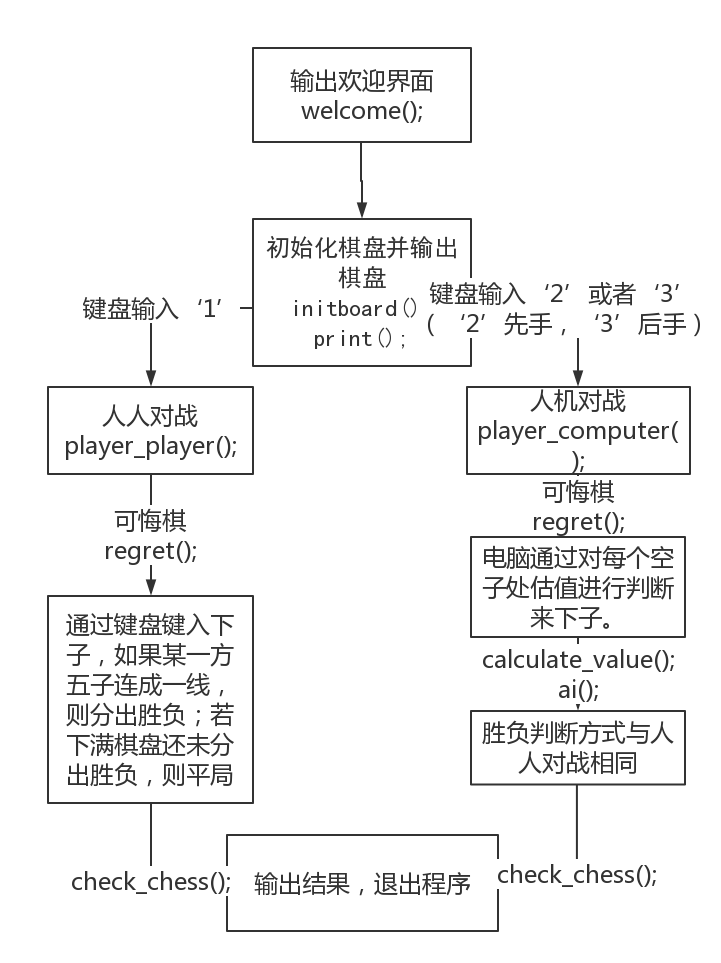
模块函数构造

根据系统功能结构图的描述，可以构造出该系统的抽象数据类型和相对应的函数，其方法名和功能如下表。

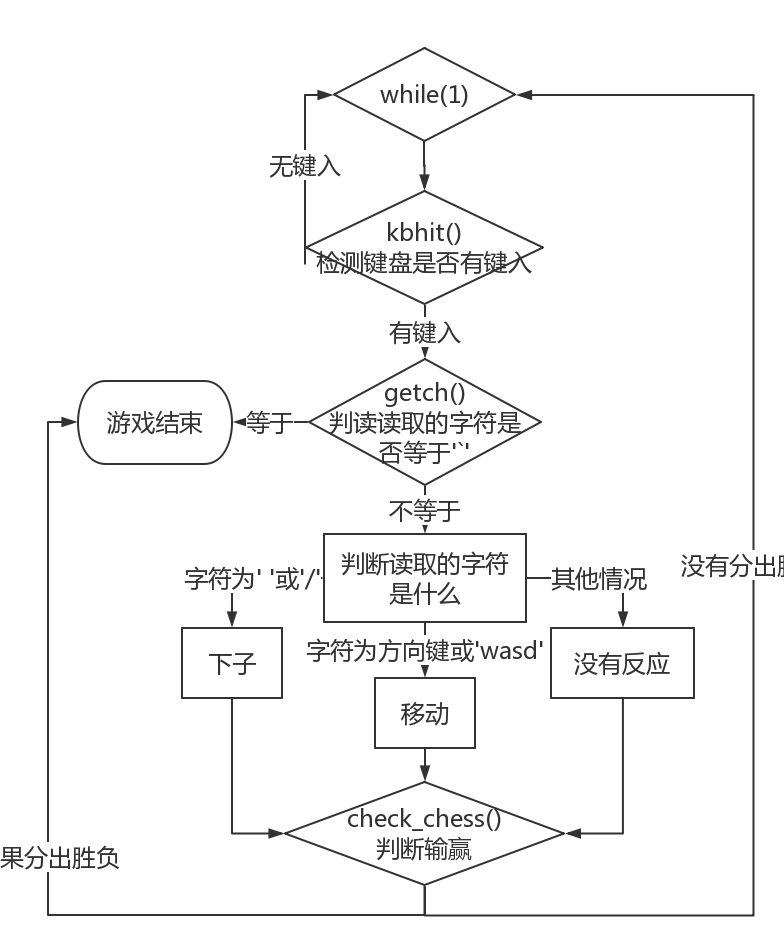
函数功能表

|  |  |  |
| --- | --- | --- |
| 功能模块 | 函数名 | 函数功能 |
| 主模块 | int main(void); | 主函数 |
|  | oid welcome(void); | 欢迎界面 |
|  | void initboard(void); | 棋盘初始化 |
|  | void print(void); | 打印棋盘 |
| 下子过程实现模块 | void ai(void); | 电脑智能下子实现 |
|  | void calculate\_value(void); | 对棋盘估值 |
|  | int player\_player(void); | 人人对战实现 |
|  | void go\_to\_xy(unsigned, unsigned); | 终端下的光标位置移动 |
| 判断输赢模块 | void game\_over(int); | 游戏结束后的判定输出 |
|  | void regret(void); | 悔棋函数 |
|  | check\_chess(void); | 判断输赢函数 |

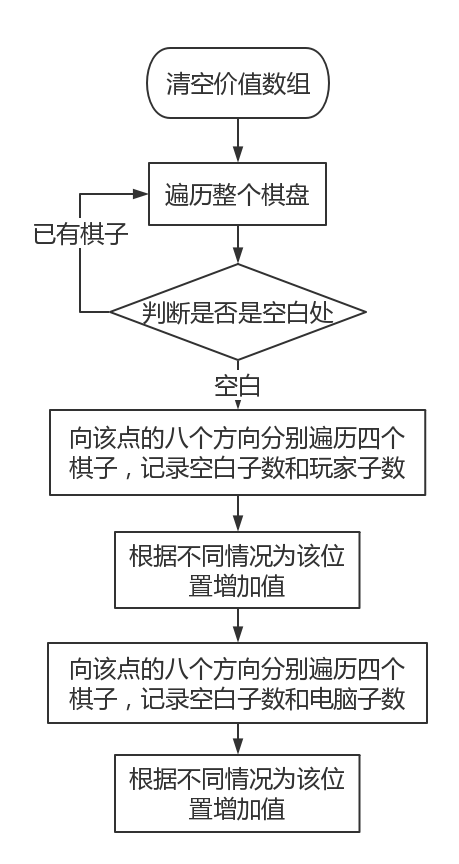
1. 主要功能模块的输入、处理(算法框架描述)和输出



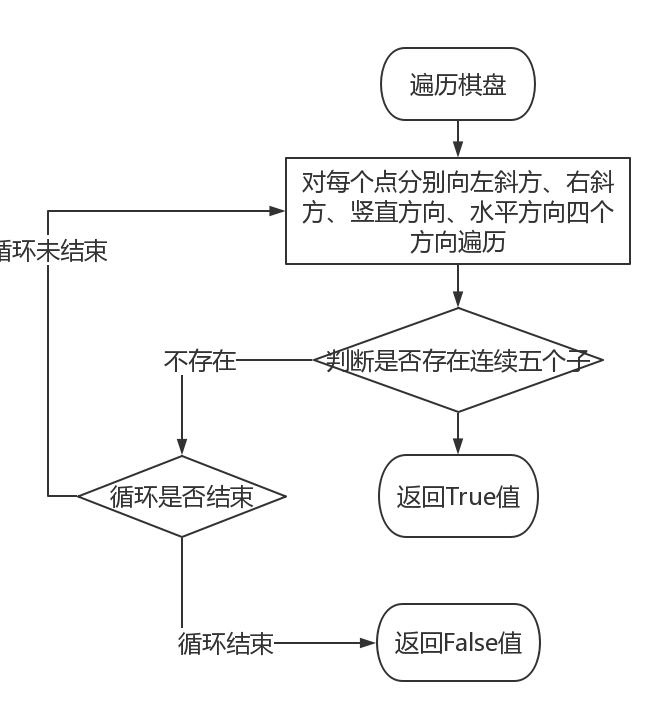
1. 人人/人机下棋函数



1. 估值函数



1. 判断输赢函数

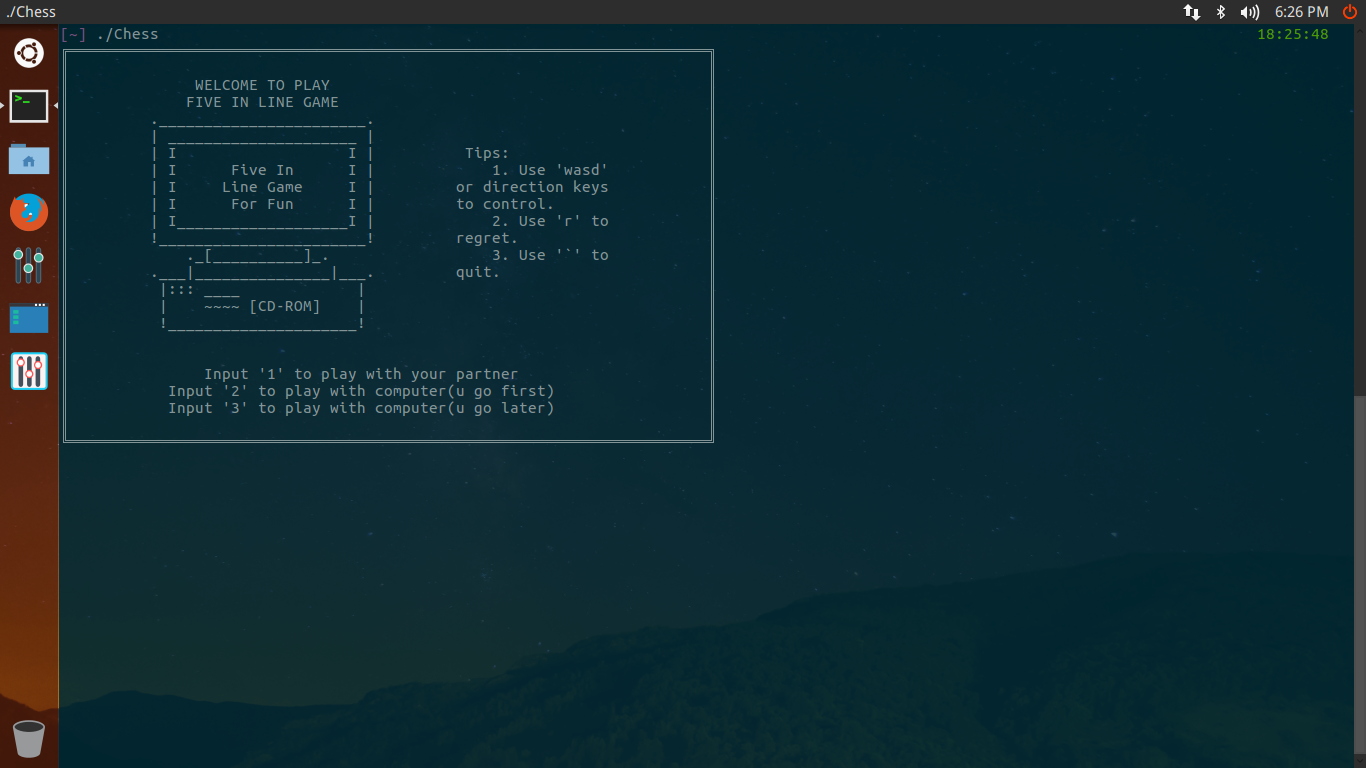


三、测试

1、测试范例与测试结果

运行环境：终端

终端输入“gcc main.cpp -o Chess -std=c++11”，生成Chess可执行文件后，执行“./Chess”。运行功能测试范例如下所示，运行首页界面如下图所示：



欢迎界面

、

下棋效果图

2、测试结果的分析与讨论

测试是使用人工或者自动手段来运行或测试某个系统的过程，其目的在于检验是否满足规定的需求或弄清预期结果与实际结果之间的差别。

四、使用说明和作业小结

1、使用说明

运行环境：需要支持c11标准，即要求gcc4.7及以上。

使用方法：在终端下用gcc编译源文件，按照界面提示选择功能操作。

2、作业小结

通过这次大作业，对调试程序更加有经验，了解了一些不熟悉的头文件。代码编写能力和阅读能力有了很大的提升。

五、程序清单

程序主要包括两部分：conio.h头文件、main.cpp源文件。

//

// main.cpp

// Debug

//

// Created by 欠费 on 2018/5/21.

// Copyright © 2018年 AssassinQ. All rights reserved.

//

#include "conio.h"

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <ctime>

#include <random>

#define LEN 10 //棋盘大小

#define BOARD (LEN \* 2 + 1) //整体大小

#define NOTCHESS check[row][col] == 0 //格子对应值的判断

#define ISWHITE check[row][col] == -1 //白子对应值的判断

#define ISBLACK check[row][col] == 1 //黑子对应值的判断

#define COVERSPACE check[row][col] = 3 //光标覆盖空格

#define COVERWHITE check[row][col] = -2 //光标覆盖白子

#define COVERBLACK check[row][col] = 2 //光标覆盖黑子

#define ISCOVERSPACE check[row][col] == 3 //光标覆盖空格的判断

#define ISCOVERWHITE check[row][col] == -2 //光标覆盖白子的判断

#define ISCOVERBLACK check[row][col] == 2 //光标覆盖黑子的判断

int getch(void); //输入不需要回车键便返回

int kbhit(void); //检测键盘是否有输入

void welcome(void); //欢迎界面

void init\_board(void); //环境构建

void print(void); //棋盘输出

int player\_computer(void); //人机

int player\_player(void); //人人

int check\_chess(void); //棋子布局检测

void regret(void); //悔棋

void game\_over(int); //游戏结束后的判定输出

void go\_to\_xy(unsigned, unsigned); //将输入光标移动到指定位置

void ai(void); //电脑下子

void calculate\_value(void); //计算每格的价值

const char \*board[BOARD][BOARD] = {0}; //棋盘格子样式

int check[LEN][LEN] = {0}; //格子的状态

int value[LEN][LEN] = {0}; //格子的价值

int reg[LEN \* LEN][3] = {0}; //悔棋：回合下子记录

int turn = -1, order = 0; //white == -1 black == 1 | 先后手

int row = 0, col = 0, sign; //行列 | 输入记录

int count = 0; //下子数

const char \*BLACK = "○", \*WHITE = "●", \*POS = "¤"; // UTF-8 棋子、棋盘字符

const char \*LT = "┌", \*TOP = "┬", \*RT = "┐";

const char \*LEFT = "├", \*CENTER = "┼", \*RIGHT = "┤";

const char \*LF = "└", \*FOOT = "┴", \*RF = "┘";

const char \*SPACE = " ";

void welcome() //欢迎界面

{

printf("╔═══════════════════════════════════════════════════════════════════════╗\n");

printf("║ ║\n");

printf("║ WELCOME TO PLAY ║\n");

printf("║ FIVE IN LINE GAME ║\n");

printf("║ .\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. ║\n");

printf("║ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | ║\n");

printf("║ | I I | Tips: ║\n");

printf("║ | I Five In I | 1. Use \'wasd\' ║\n");

printf("║ | I Line Game I | or direction keys ║\n");

printf("║ | I For Fun I | to control. ║\n");

printf("║ | I\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_I | 2. Use \'r\' to ║\n");

printf("║ !\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_! regret. ║\n");

printf("║ .\_[\_\_\_\_\_\_\_\_\_\_]\_. 3. Use \'`\' to ║\n");

printf("║ .\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_. quit. ║\n");

printf("║ |::: \_\_\_\_ | ║\n");

printf("║ | ~~~~ [CD-ROM] | ║\n");

printf("║ !\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_! ║\n");

printf("║ ║\n");

printf("║ ║\n");

printf("║ Input \'1\' to play with your partner ║\n");

printf("║ Input \'2\' to play with computer(u go first) ║\n");

printf("║ Input \'3\' to play with computer(u go later) ║\n");

printf("║ ║\n");

printf("╚═══════════════════════════════════════════════════════════════════════╝\n");

printf("\033[?25l"); //隐藏光标 '\033'是'ESC'的八进制表示形式

}

void init\_board() //棋盘构建

{

memset(check, 0, sizeof(check));

for(int i = 0; i < BOARD; i++)

for(int j = 0; j < BOARD; j++)

board[i][j] = SPACE; // 先全部赋为空格

//棋盘四个角的搭建

board[0][0] = LT;

board[0][BOARD - 1] = RT;

board[BOARD - 1][0] = LF;

board[BOARD - 1][BOARD - 1] = RF;

for(int i = 2; i < BOARD - 2; i += 2) //四边搭建

{

board[0][i] = TOP;

board[BOARD - 1][i] = FOOT;

board[i][0] = LEFT;

board[i][BOARD - 1] = RIGHT;

}

for(int i = 2; i < BOARD - 2; i += 2)

for(int j = 2; j < BOARD - 2;j += 2)

board[i][j] = CENTER; //非边角搭建

}

void print() //打印棋盘

{

system("clear"); //清除欢迎界面

for(int i = 0; i < BOARD; i++)

{

for(int j = 0; j < BOARD; j++)

{

if((i - 1) % 2 == 0 && (j - 1) % 2 == 0) //根据格子对应check[][]的值输出

{

int temp = check[(i - 1) / 2][(j - 1) / 2];

if(temp == 0) //打印空格

printf(" ");

else if(temp == -1) //打印白子

printf("%s ", WHITE);

else if(temp == 1) //打印黑子

printf("%s ", BLACK);

else //打印光标

printf("%s ", POS);

}

else //打印棋盘

printf("%s ", board[i][j]);

}

putchar('\n');

}

if(turn == 1) //轮到黑子

printf(" %s Black Turn", BLACK);

else //轮到白子

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count); //光标的位置和反悔的次数

}

int player\_player() //人人

{

COVERSPACE; //初始光标位置

print(); //棋盘整体输出

system("stty -echo"); //Linux命令 命令不被显示

while(1)

{

if(kbhit()) //检测键盘是否有输入 有输入情况下

{

while((sign = getch()) != '`') //退出按键

{

if(sign == 'w' || sign == 's' || sign == 'a' || sign == 'd' || sign == ' ' || sign == 65 || sign == 66 || sign == 67 || sign == 68 || sign == '/' || sign == 'r')

{

if(sign != ' ' && sign != '/') //移动光标前，先把原格子复位

{

if(ISCOVERSPACE)

{

check[row][col] = 0;

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", SPACE);

}

else if(ISCOVERWHITE)

{

check[row][col] = -1;

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", WHITE);

}

else if(ISCOVERBLACK)

{

check[row][col] = 1;

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", BLACK);

}

}

switch(sign) //光标移动、下子

{

case 'w':

if(row > 0)

row--;

else

row = LEN - 1;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 's':

if(row < LEN - 1)

row++;

else

row = 0;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

break;

case 'a':

if(col > 0)

col--;

else

col = LEN - 1;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 'd':

if(col < LEN - 1)

col++;

else

col = 0;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 65:

if(row > 0)

row--;

else

row = LEN - 1;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 66:

if(row < LEN - 1)

row++;

else

row = 0;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 68:

if(col > 0)

col--;

else

col = LEN - 1;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 67:

if(col < LEN - 1)

col++;

else

col = 0;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case ' ':

if(ISCOVERSPACE)

{

if(turn == -1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", WHITE);

check[row][col] = turn;

}

else if(turn == 1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", BLACK);

check[row][col] = turn;

}

reg[count][0] = turn;

reg[count][1] = row;

reg[count][2] = col;

turn = -turn;

count++;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

}

break;

case '/':

if(ISCOVERSPACE)

{

if(turn == -1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", WHITE);

check[row][col] = turn;

}

else if(turn == 1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", BLACK);

check[row][col] = turn;

}

reg[count][0] = turn;

reg[count][1] = row;

reg[count][2] = col;

turn = -turn;

count++;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

}

break;

case 'r': //悔棋按钮

regret();

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

default:

break;

}

}

break;

}

if(sign == '`') //强制退出

return 2;

}

if(count == LEN \* LEN) //如果棋盘满 返回平局

return 0;

if(check\_chess()) //如果一方赢了 返回分出胜负

return 1;

}

}

int player\_computer() //人机

{

if(order == -1)

COVERSPACE; //初始光标位置

print();

system("stty -echo"); //使命令不被显示

while(1)

{

if(kbhit()) //检测键盘是否有输入

{

while((sign = getch()) != '`') //退出按键

{

if(sign == 'w' || sign == 's' || sign == 'a' || sign == 'd' || sign == ' ' || sign == 65 || sign == 66 || sign == 67 || sign == 68 || sign == '/' || sign == 'r')

{

if(sign != ' ' && sign != '/') //移动光标前把原格子复位

{

if(ISCOVERSPACE)

{

check[row][col] = 0;

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", SPACE);

}

else if(ISCOVERWHITE)

{

check[row][col] = -1;

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", WHITE);

}

else if(ISCOVERBLACK)

{

check[row][col] = 1;

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", BLACK);

}

}

switch(sign) //光标移动和下子

{

case 65:

if(row > 0)

row--;

else

row = LEN - 1;

if(NOTCHESS) //光标覆盖时空白处的值

COVERSPACE;

else if(ISWHITE) //光标覆盖时白子的值

COVERWHITE;

else if(ISBLACK) //光标覆盖时黑子的值

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 66:

if(row < LEN - 1)

row++;

else

row = 0;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 68:

if(col > 0)

col--;

else

col = LEN - 1;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case 67:

if(col < LEN - 1)

col++;

else

col = 0;

if(NOTCHESS)

COVERSPACE;

else if(ISWHITE)

COVERWHITE;

else if(ISBLACK)

COVERBLACK;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

break;

case ' ':

if(ISCOVERSPACE)

{

if(turn == -1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", WHITE);

check[row][col] = turn;

}

else if(turn == 1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", BLACK);

check[row][col] = turn;

}

reg[count][0] = turn;

reg[count][1] = row;

reg[count][2] = col;

turn = -turn;

count++;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

}

break;

case 'r': //悔棋按钮

if(count == 1)

regret();

else

{

regret();

regret();

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", POS);

}

break;

default:

break;

}

}

break;

}

if(sign == '`') //强制退出

return 2;

}

else if(turn == -order) //电脑下棋

{

ai();

if(turn == -1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", WHITE);

check[row][col] = turn;

}

else if(turn == 1)

{

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", BLACK);

check[row][col] = turn;

}

reg[count][0] = turn;

reg[count][1] = row;

reg[count][2] = col;

turn = -turn;

count++;

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

}

if(count == LEN \* LEN) //如果棋盘满 返回平局

return 0;

if(check\_chess()) //如果一方赢了 返回分出胜负

return 1;

}

}

void ai() //电脑下子

{

calculate\_value();

int sum\_i[LEN \* LEN], sum\_j[LEN \* LEN];

int max = 0, sum = 0;

for(int i = 0; i < LEN; i++)

for(int j = 0; j < LEN; j++)

if(check[i][j] == 0 && value[i][j] > max)

max = value[i][j];

for(int i = 0; i < LEN; i++)

{

for(int j = 0; j < LEN; j++)

{

if(check[i][j] == 0 && value[i][j] == max)

{

sum\_i[sum] = i;

sum\_j[sum] = j;

sum++;

}

}

}

srand((unsigned)time(0));

row = sum\_i[rand() % sum];

col = sum\_j[rand() % sum];

}

void calculate\_value()

{

//统计玩家或者电脑连成的子

int white\_num = 0; //玩家连成子的个数

int black\_num = 0; //AI连成子的个数

int empty\_num = 0; //各方向空白位的个数

int player = order, computer = -order;

memset(value, 0, sizeof(value)); //清空评分数组

//计分

for(int i = 0; i < LEN; i++)

{

for(int j = 0; j < LEN; j++)

{

if(check[i][j] == 0) //空白点

{

//遍历周围八个方向

for(int y = -1; y <= 1; y++)

{

for(int x = -1; x <= 1; x++)

{

white\_num = 0;

black\_num = 0;

empty\_num = 0;

if(!(y == 0 && x == 0)) //原坐标不算

{

//每个方向延伸4个子 对玩家评分

for(int k = 1; k <= 4; k++)

{

if(i + k \* y > 0 && i + k \* y < LEN && j + k \* x > 0 && j + k \* x < LEN && check[i + k \* y][j + k \* x] == player) //玩家的子

white\_num++;

else if(i + k \* y > 0 && i + k \* y < LEN && j + k \* x > 0 && j + k \* x < LEN && check[i + k \* y][j + k \* x] == 0) //空白位

{

empty\_num++;

break;

}

else //出边界

break;

}

for(int k = 1; k <= 4; k++)

{

if(i - k \* y > 0 && i - k \* y < LEN && j - k \* x > 0 && j - k \* x < LEN && check[i - k \* y][j - k \* x] == player) //玩家的子

white\_num++;

else if(i - k \* y > 0 && i - k \* y < LEN && j - k \* x > 0 && j - k \* x < LEN && check[i - k \* y][j - k \* x] == 0) //空白位

{

empty\_num++;

break;

}

else //出边界

break;

}

if(white\_num == 1) //杀二

value[i][j] += 10;

else if(white\_num == 2) //杀三

{

if(empty\_num == 1)

value[i][j] += 30;

else if(empty\_num == 2)

value[i][j] += 40;

}

else if(white\_num == 3) //杀四

{

//量变空位不一样，优先级不一样

if(empty\_num == 1)

value[i][j] += 60;

else if(empty\_num == 2)

value[i][j] += 110;

}

else if(white\_num == 4) //杀五

value[i][j] += 10100;

empty\_num = 0;

//对电脑的子评分

for(int k = 1; k <= 4; k++)

{

if(i + k \* y > 0 && i + k \* y < LEN && j + k \* x > 0 && j + k \* x < LEN && check[i + k \* y][j + k \* x] == computer) //电脑的子

black\_num++;

else if(i + k \* y > 0 && i + k \* y < LEN && j + k \* x > 0 && j + k \* x < LEN && check[i + k \* y][j + k \* x] == 0) //空白位

{

empty\_num++;

break;

}

else //出边界

break;

}

for(int k = 1; k <= 4; k++)

{

if(i - k \* y > 0 && i - k \* y < LEN && j - k \* x > 0 && j - k \* x < LEN && check[i - k \* y][j - k \* x] == computer) //电脑的子

black\_num++;

else if(i - k \* y > 0 && i - k \* y < LEN && j - k \* x > 0 && j - k \* x < LEN && check[i - k \* y][j - k \* x] == 0) //空白位

{

empty\_num++;

break;

}

else //出边界

break;

}

if(black\_num == 0) //普通下子

value[i][j] += 5;

else if(black\_num == 1) //活二

value[i][j] += 10;

else if(black\_num == 2)

{

if(empty\_num == 1) //死三

value[i][j] += 25;

else if(empty\_num == 2) //活三

value[i][j] += 50;

}

else if(black\_num == 3)

{

if(empty\_num == 1) //死四

value[i][j] += 55;

else if (empty\_num == 2) //活四

value[i][j] += 100;

}

else if(black\_num >= 4) //活五

value[i][j] += 10000;

}

}

}

}

}

}

}

int check\_chess() //检测是否分出胜负

{

bool judge; //胜利判断

int temp; //当前格子对应值

for(int i = 0; i < LEN; i++) //行

{

for(int j = 0; j < LEN; j++) //列

{

if(check[i][j] != 0)

{

temp = check[i][j];

judge = true;

if(i <= LEN - 5 && j <= LEN - 5) //右斜

{

for(int l = 1; l < 5; l++)

if(check[i + l][j + l] != temp)

judge = false;

if(judge == true)

return 1;

}

judge = true;

if(i <= LEN - 5 && j >= 5 - 1) //左斜

{

for(int l = 1; l < 5; l++)

if(check[i + l][j - l] != temp)

judge = false;

if(judge == true)

return 1;

}

judge = true;

if(j <= LEN - 5) //水平

{

for(int l = 1; l < 5; l++)

if(check[i][j + l] != temp)

judge = false;

if(judge == true)

return 1;

}

judge = true;

if(i <= LEN - 5) //竖直

{

for(int l = 1; l < 5; l++)

if(check[i + l][j] != temp)

judge = false;

if(judge == true)

return 1;

}

}

}

}

return 0;

}

void game\_over(int res) //游戏结束后的判定输出

{

go\_to\_xy(BOARD + 2, 0); //把输入光标移去底部

if(res)

{

if(res == 2) //s=2则平局

printf("Force Exit\n");

else if(turn == 1) //turn=1即最后一个落子的为白棋，故白棋赢

printf("\t-----%s White Won-----\n", WHITE);

else if(turn == -1) //同理

printf("\t-----%s Black Won-----\n", BLACK);

else

printf("Error!\n");

}

else

printf("\t-----Tied-----\n");

system("stty echo");

printf("\033[?25h"); //显示光标

}

void go\_to\_xy(unsigned x, unsigned y) //将输入光标移动到指定位置

{

printf("\033[%d;%dH", x, y); //设置光标位置

}

void regret() //悔棋

{

if(count)

{

count--;

turn = -turn;

row = reg[count][1]; //根据数组记录内容悔棋

col = reg[count][2];

for(int j = 0; j < 3; j++)

reg[count][j] = 0;

}

go\_to\_xy(BOARD + 1, 0);

if(turn > 0)

printf(" %s Black Turn", BLACK);

else

printf(" %s White Turn", WHITE);

printf(" Pos(%2d,%2d) Done:%2d\n", row + 1, col + 1, count);

go\_to\_xy(2 + 2 \* row, 1 + 2 \* (1 + 2 \* col));

printf("%s ", SPACE);

COVERSPACE;

}

int main()

{

int res = 0; //游戏结束的判定状态输出

welcome(); //欢迎界面

while(1) //选择游戏模式

{

sign = getch(); //读取键盘输入

if(sign == '1') //选择人人模式

{

init\_board();

res = player\_player();

break;

}

else if(sign == '2') //选择人机模式（先手

{

init\_board();

order = -1;

res = player\_computer();

break;

}

else if(sign == '3') //选择人机模式（后手

{

init\_board();

order = 1;

res = player\_computer();

break;

}

else if(sign == '`') //退出游戏

break;

else //键入其他字符不会有任何显示

continue;

}

game\_over(res); //游戏结束

return 0;

}

//

// conio.h

// Debug

//

// Created by 欠费 on 2018/5/21.

// Copyright © 2018年 AssassinQ. All rights reserved.

//

#include <stdio.h>

#include <termio.h> //Linux专用头文件，用于实现getch()、kbhit()

#include <unistd.h> //定义了大量针对系统调用的封装的接口 提供对 POSIX 操作系统 API 的访问功能

#include <fcntl.h> //定义了很多宏和函数原型 对文件的打开、数据写入、数据读取、关闭文件的操作

int getch() //输入不需要回车键便返回

{

struct termios tm, tm\_old;

if(tcgetattr(STDIN\_FILENO, &tm) < 0) //STDIN\_FILENO是终端文件描述符，是一个打开文件句柄

return -1;

tm\_old = tm;

cfmakeraw(&tm); //设置串口属性 将终端设置成原始模式，此时终端是不可回显的

if(tcsetattr(STDIN\_FILENO, TCSANOW, &tm) < 0) //更改终端设置

return -1;

int ch = getchar();

if(tcsetattr(STDIN\_FILENO, TCSANOW, &tm\_old) < 0) //恢复终端原始设置

return -1;

return ch;

}

int kbhit() //检测键盘是否有输入

{

struct termios oldt, newt; //控制终端状态的数据结构

int ch;

int oldf;

tcgetattr(STDIN\_FILENO, &oldt); //得到当前终端的状态

newt = oldt;

//设置新的终端属性：ICANON和ECHO取反，表示关闭输入行编辑模式，即能够直接读入字符

newt.c\_lflag &= ~(ICANON | ECHO); //ICANON：如果在输入中看到'\n'或者'EOF'，会返回缓冲区内容 //ECHO：控制回显

tcsetattr(STDIN\_FILENO, TCSANOW, &newt); //TSCSANOW：告诉函数立即改变终端的STDIN\_FILENO的属性值

oldf = fcntl(STDIN\_FILENO, F\_GETFL, 0); //获取flags，即open函数的第二个参数

//fcntl()的返回值与命令有关。如果出错，所有命令都返回－1，如果成功则返回某个其他值

fcntl(STDIN\_FILENO, F\_SETFL, oldf | O\_NONBLOCK); //增加一个flags，把文件设置成非阻塞 //F\_SETFL设置文件状态标记

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt); //恢复终端设置

fcntl(STDIN\_FILENO, F\_SETFL, oldf);

if(ch != EOF)

{

ungetc(ch, stdin); //把字符退回到输入流

return 1;

}

return 0;

}