

合肥工业大学

“数据结构”

课程设计报告

设计题目	一字棋游戏设计实现
姓名	周布伟
学号	2019217192
专业	计算机科学与技术
班级	19-2 班
完成日期	2021-07-09

（一）需求和规格说明

1. 需求描述

设计实现一字棋游戏程序，人为一方，计算机为一方，人下时字符 * 将放在所指定的位置，而计算机下时字符 @ 将放在某一空格位置。行、列、或两对角线有连续三个相同字符一方为胜方，也有平局情况。要求能动态演示。使用人工智能中的极大极小搜索算法，并可选定棋盘大小。

2. 需求分析

（1）已有条件分析

人机对决：该游戏程序的主体是玩家与计算机，玩家通过鼠标实现菜单选择、下棋等操作，计算机随机或根据玩家的操作下棋。

对决过程：玩家可以自主选择棋盘大小，并通过菜单选择开始游戏并选择首先下棋的一方，然后玩家可在生成的指定大小的棋盘下棋，在下棋过程中，已经下过的位置不允许重复，当行或列或两对角线之一中有连续三个相同字符的一方获胜，当棋盘都被填满时尚未有胜负则出现平局。当玩家想停止游戏时可以选择退出游戏。

（2）条件补充与创新

多难度模式：为了比较采用极大极小搜索算法的优越性，本程序采用多难度模式，玩家可以选择简单、困难等多个模式与计算机对决，其中，简单模式下，计算机随机选择落棋位置，玩家的落棋不会影响计算机的选择；困难模式下，计算机通过极大极小算法构建决策树进行决策。此外，本程序还能实现玩家与玩家的对决。

得分记录：玩家可重复与电脑对决，所得的成绩将会被记录，玩家可以选择清空记录。

（二）设计

1. 设计思想

在实现思想上，主要采用自顶向下、分而治之的模块化设计思想并主要面向过程，如此考虑一方面游戏的进行是过程的进行，结合游戏的过程更易于程序设计，另一方面分级的管理也更贴合菜单的制作。

在选择游戏难度的基础上，玩家首先要选择出棋顺序，其次开始游戏。游戏的主要过程为玩家与计算机的对弈，这里涉及到的过程是玩家选择落棋位置、保存落棋结果、展示落棋贴图、计算机选择落棋位置、判断比赛结果、计算比赛得分、选择是否继续游戏等。其中计算机选择落棋位置在不同难度下有所不同，简单模式下，计算机通过对空格进行编号，生成随机数字来落棋；困难模式下，采用人工智能中的极大极小搜索算法，对当前棋局进行预测

评估，计算机选择最有利的情况下棋。通过 Board 数据结构对玩家与计算机的出棋结果进行保存，并对棋盘进行编号，如此操作方便获得玩家与计算机的落棋信息。此外，不同模式下的下棋操作是一样的，唯一不同的是计算机落棋的决策，故在此使用函数指针来进行计算机的落棋决策。

在实现形式上，玩家主要通过鼠标的操作进行菜单的选择和游戏的进行，这里采用 easyX 来实现，其中游戏的背景与图标通过 win10 自带的画图软件绘制。

2. 设计表示

表 1 重要的数据结构

数据结构	数据类型	数据项	描述
Board	char**	cell	棋盘内容
	int	size	棋盘大小
	int	emptyNum	棋盘空余格子数量
Node	int	val	估测值
	int	row	棋盘上的行
	int	col	棋盘上的列
	int	alpha	最大下界
	int	beta	最小上界
	bool	level	层次类型 (MAX/MIN)
	struct tree*	firstChild	第一个孩子
	struct tree*	nextSibling	兄弟结点

表 2 重要的数据类型

数据名称	数据类型	描述
player	char	当前的玩家， ‘@’ 或 ‘*’
order	short	PVC 模式下的出棋顺序， 1：玩家优先；2：计算机优先
diff	short	PVC 模式下的难度， 0：未选择； 1：简单； 3：困难
ret	int	比赛结果， 0：未结束； 1：*胜；2：@胜；3：平局
cntRet[3]	short	保存得分
againGame	bool	是否继续游戏
againPVC	bool	是否继续 PVC 模式

表 3 函数

函数名称	参数	返回值	描述
initStart	start, bool	void	展示开始界面, 选择是否开始游戏
initBoard	bd, Board; size, int	bool	初始化棋盘
destroyBoard	bd, Board	void	销毁棋盘
chooseDiff	diff, int	void	选择难度是简单、中等还是困难
chooseSize	bd, Board	void	选择棋盘大小
chooseOrder	order, int	void	在 PVC 模式下, 选择玩家和电脑的出棋顺序
chooseAgainPVC	againMode, bool; againPVC, bool; cntRet[3], int	void	一局游戏结束后选择再试一次、返回主菜单、清空结果
getRet	bd, Board; cntStep, int	short	判断比赛时的胜负或还在进行中
getVal	bd, Board; player, char	void	获得估值函数的值
inputChess	bd, Board; player, char	void	检测鼠标选择的落棋位置
displayMes	player, char	void	展示轮到谁下棋
displayChess	loc, string; player, char	void	展示该轮落棋的棋子
displayRet	ret, string	void	展示该轮游戏结果
displayScorePVC	cntRet, char	void	展示 PVC 模式该难度下的得分结果
ModeDiff	bd, Board; order, int; inputModeDiff, (*)fun	void	PVC 模式, 玩家与电脑对弈
inputModeEasy	bd, Board	void	简单模式, 电脑的出棋结果
inputModeHard	bd, Board	void	困难模式, 电脑的出棋结果
MinMax	bd, Board; T, Node*; player, char; step, int	void	进行极大极小搜索

表 4 程序结构

一级菜单	二级菜单		三级菜单		四级菜单		五级菜单	
开始界面	模式选择	PVC	设置棋盘大小	3*3	难度选择	简单模式	玩家计算机轮流落棋	1. 再来一次(四级菜单)
				4*4		困难模式		2. 返回主菜单(二级菜单)
				5*5	/		基本同上	
		退出			/		/	

出棋决策决定出棋的位置并显示；如果是玩家，则根据玩家用鼠标的选择来得到出棋的位置并显示，之后轮流出棋，总棋子数目加一，判断对弈结果。分出胜负后对结果进行统计并展示，玩家选择是否继续游戏，若继续，则棋盘清空，总棋子数目归零，对弈结果归零；若退出，则 againDiff 归零，并退出由其控制游戏进行的外部循环，返回至二级菜单，此时依然在 againGame 控制的循环内。

对于对弈结果的判断，主要采用遍历。分别对行列对角线上的连线进行统计，当线段中出现三个连续的棋子时可判断胜负，若总棋子数目为棋盘格子总数而未分出胜负，则为平局，否则为游戏未结束。

(三) 用户手册

程序运行时显示开始界面，鼠标单击“开始游戏”即可进入。

进入游戏后可以选择多种模式，鼠标单击“人机对战”可以直接和计算机对弈，点击“玩家对战”可以进入玩家之间的对决，点击“退出”则可退出程序。

选择模式后可进一步选择棋盘的大小，棋盘有 3*3、4*4、5*5 三种大小供玩家选择。

PVC 模式(人机对战模式)下，首先玩家通过鼠标单击选择难度，之后选择出棋顺序，选择后游戏开始，由事先选择的出棋顺序开始落棋，玩家鼠标单击选择棋盘上的位置下棋，如果单击超出棋盘范围或已经落棋的位置则无效，需要重新单击直至做出有效选择，程序根据落棋来判断结果，最后展示结果语句。

PVP 模式(玩家对战模式)下，与 PVC 模式类似，由两位玩家分别下棋，该模式下无需选择出棋顺序，两名玩家鼠标单击选择棋盘上的位置下棋，如果单击超出棋盘范围或已经落棋的位置则无效，需要重新单击直至做出有效选择，程序根据落棋来判断结果，最后展示结果语句。

等待 3 秒后显示得分情况，玩家可以通过鼠标单击选择再来一次、返回主菜单、清空结果。如果选择“再来一次”，则会继续该难度下的对弈；如果选择“返回主菜单”，则会跳转到模式选择菜单，玩家可以选择其他模式或退出；如果选择“清空结果”，则比赛结果将会被清空，得分会被重新统计，选择后玩家可以继续选择再来一次或返回主菜单。

(四) 调试及测试

通过测试，所有测试的数据均符合游戏规则与结果，且比赛统计结果、游戏界面的跳转等都符合预期。

对于游戏模式多难度的实现，一开始考虑通过多个函数将每个难度进行的操作封装到一起，后来发现多难度下所有的操作都是一样的，唯一不同的就是计算机的出棋决策，故在此

考虑用函数指针实现多难度模式。对于出棋顺序更进一步的考虑，代码中最先出棋一方的选择是在游戏的循环之外的，也就是说当玩家选择再来一次的时候，此时的并没有提示选择出棋顺序，而此时出棋的实际上上一轮中失败的一方。至于原因，因为玩家和计算机每落一个棋子，轮次就会发生转换，也就是代码中的 player 会被赋值 '@' 或 '*'，当一方胜利时，胜利方肯定已经落棋，此时顺序就交给了失败方，由此第二轮开始时的顺序就是失败方，或者可以通过枚举各种情况也可以判断第二轮的优先出棋的一方。如此设计既符合常理，也省去用户重新选择顺序带来的麻烦。对于得分的设计，该程序采用比较简单的方案，即直接将胜负和平局显示在结果中，并且该结果只能是该难度下的结果。事实上，也可以考虑将多个难度混合在一起，每个难度的胜负对于不同的分值，难度越大，获胜或平局取得的分值越高，难度越低，失败失去的分值越高。最后根据相应的计算公式即可得到最后的分数。

(五) 运行实例



图 5-1 难度选择



图 5-2 顺序选择



图 5-3 对弈过程



图 5-4 胜负展示



图 5-5 比赛结果展示



图 5-6 清空结果

（六） 进一步改进

增加 PVP 模式和该模式下对应的结果展示。在 PVC 模式实现的基础上实现 PVP 较为简单，与 PVC 模式的实现类似，本来想像 PVC 模式下用函数指针的方式实现多难度整合封装的形式将 PVC 与 PVP 进行整合，但考虑到 PVP 模式与 PVC 模式有很多不同的变量，若整合在一起接口太多，而且 PVP 模式较 PVC 模式更加简洁，无需选择难度、顺序等，故直接将 PVP 模式封装成函数在 main 中调用。

在原有的程序基础上，可以更进一步地实现五子棋对决设计，不过需要设计五子棋的棋盘并重新设定对弈胜利的条件，即当行列对角线出现五子的情况，在核心算法上无需做过多修改。

尽管通过剪枝和预先设置的方式能够很好的提高计算机计算的速度，但若棋盘大小增大，或迭代次数变多，计算机有需要一定的时间才能计算结果，因此需要更进一步寻找优化的算法使计算机的速度有所提升。



图 6-1 模式选择



图 6-2 比赛结果展示

（七） 心得体会

这次课程设计需要学习与人工智能相关的算法，并在此基础上实现本课题。课程设计过程中除了运用课内的部分知识外，还需要其他的知识，可见学无止境，课内的知识远远不够实际的需要，以后还需要进一步地学习与锻炼。

在此次课程设计中主要涉及的数据结构是决策树，实际上和课内介绍的树并无太大差别，本次设计的树实际采用孩子-兄弟链表表示法，结点包含的内容更加丰富，在对树进行搜索时需要考虑剪枝等操作。

通过本次课程设计，从选题到程序设计到调试再到实现，这一步步走来，收获了成就感，收获了程序设计的经验。这次程序设计用到了并未在课堂中出现的 easyX 来帮助实现图形界面，并运用 C++ 中的相关特性，使用其中的布尔类型的数据类型，用变量引用的方式代替用指针传递函数参数等等。不断的调试和修改也为以后得程序设计提供了宝贵的经验。

(八) 对课程设计的建议

感谢课程设计提供锻炼的机会，尽管时间紧张，但在课程设计的过程中收获了很多。课程设计中涉及到了很多课内没有接触过的算法和思想，有时理解起来会有些困难，如果在课程设计期间能安排答疑，可能会对课程设计的进展有一些帮助。

(九) 附录——源程序

1. 头文件：TicTacToe.h

```
/******HEADER*START******/
#ifndef TicTacToe
#define TicTacToe
/******INCLUDE*START******/
#include <iostream>
#include <string>
#include <graphics.h>
#include <ctime>
using namespace std;
/******INCLUDE*END******/

/******DEFINE*START******/
#define N 3
#define MAX true
#define MIN false
#define PINF 50
#define NINF -50
/******DEFINE*END******/

/******DS*START******/
typedef struct {
    char** cell;
    int size;
    int emptyNum;
}Board;

typedef struct tree{
    int val;
    int row;
    int col;
    int alpha;
    int beta;
    bool level;
    struct tree* firstChild;
    struct tree* nextSibling;
}Node, *Tree;
/******DS*END******/

/******FUNCTION*START******/
void initStart (bool&); //初始化开始界面
bool initBoard (Board& bd, int size); //初始化棋盘
void destroyBoard (Board& bd); //销毁棋盘
void chooseMode (int&, bool&); //选择模式 PVP or PVC
void chooseSize (Board& bd); //选择棋盘大小
void chooseDiff (int&); //选择难度
void chooseOrder (int&); //选择出棋顺序
void chooseAgainPVC (bool&, bool&, int[3]); //选择是否继续
void chooseAgainPVP (bool&, int[3]); //选择是否继续
int getRet (Board&); //取得比赛结果
int getVal (Board&, char); //获得估测函数的值
void inputChess (Board&, char); //获得玩家落棋位置
void displayMes (char); //展示出棋信息
void displayChess (Board&, string, char); //展示棋子
void displayRet (int); //展示结果
void displayScorePVC (int[3]); //展示PVC下玩家的得分
void displayScorePVP (int[3]); //展示PVP下玩家双方的得分
void ModePVC (); //PVC模式
void ModePVP (); //PVP模式
void ModeDiff (Board&, int&, bool&, void (*inputModeDiff) (Board& bd)); //各种难度模式
void inputModeEasy (Board&); //简单模式下电脑出棋
void inputModeHard (Board&); //困难模式下电脑出棋
void MinMax (Board&, Node*, char, int); //极大极小搜索
/******FUNCTION*END******/
#endif // !TicTacToe
/******HEADER*END******/
```

2. 源文件：一字棋.cpp

```

//*****
//程序名称: 一字棋.cpp
//姓名: 周布伟
//修改时间: 2021/07/09
//学号: 2019217192
//*****

/*****FILE*START*****/

/*****INCLUDE*START*****/
#include "TicTacToe.h"
/*****INCLUDE*END*****/

/*****FUNCTION*START*****/

/*-----
/ 函数名: main
/ 作用: 初始化界面, 选择模式、是否继续游戏
/ 参数: 无
/ 返回值: 整型
/-----*/
int main(void)
{
    int mode = 0;
    bool againGame = 1;

    initStart(againGame);

    while (againGame)
    {
        mode = 0;
        chooseMode(mode, againGame);
        switch (mode)
        {
            case 1:
                ModePVP();
                break;
            case 2:
                ModePVC();
                break;
        }
    }

    return 0;
}

/*-----
/ 函数名: initStart
/ 作用: 展示开始界面, 选择是否开始游戏
/ 参数: start, 布尔型, 是否开始游戏
/ 返回值: 无
/-----*/
void initStart(bool &start)
{
    initgraph(1200, 800, SHOWCONSOLE);
    loadimage(0, T("./图片/start.jpg"), 1200, 800);
    bool isChoose = false; //玩家是否作出选择
    while (!isChoose)
    {
        if (MouseHit())
        {
            MOUSEMSG mouse = GetMouseMsg();
            if (mouse.uMsg == WM_LBUTTONDOWN)
            {
                if (mouse.y > 400 && mouse.y < 700 && mouse.x > 400 && mouse.x < 800)
                {
                    start = 1;
                    isChoose = true;
                }
                else if (mouse.y > 600 && mouse.y < 800 && mouse.x > 0 && mouse.x < 200)
                {
                    start = 0;
                    isChoose = true;
                }
            }
        }
    }
}

/*-----
/ 函数名: initBoard
/ 作用: 初始化棋盘
/ 参数: bd, 棋盘型, 保存棋盘结果
/         size, int型, 设定棋盘大小

```

```

/ 返回值: bool型, 初始化结果
/-----*/
bool initBoard(Board& bd, int size)
{
    if (size < 3 && size > 5)
    {
        return false;
    }

    bd.size = size;
    bd.cell = (char**)malloc(sizeof(char*) * size);
    if (bd.cell)
    {
        for (int i = 0; i < bd.size; i++)
        {
            bd.cell[i] = (char*)malloc(sizeof(char) * size);
        }
        for (int i = 0; i < bd.size; i++)
        {
            for (int j = 0; j < bd.size; j++)
            {
                bd.cell[i][j] = ' ';
            }
        }

        bd.emptyNum = bd.size * bd.size;
    }

    return true;
}

/*-----
/ 函数名: destroyBoard
/ 作 用: 销毁棋盘
/ 参 数: bd, 棋盘型, 保存棋盘结果
/ 返回值: 无
/-----*/
void destroyBoard(Board& bd)
{
    for (int i = 0; i < bd.size; i++)
    {
        free(bd.cell[i]);
    }

    free(bd.cell);
}

/*-----
/ 函数名: chooseSize
/ 作 用: 选择棋盘大小
/ 参 数: bd, 棋盘型, 保存棋盘结果
/ 返回值: 无
/-----*/
void chooseSize(Board& bd)
{
    loadImage(0, _T("./图片/SizeChoose.jpg"), 1200, 800);
    while (1)
    {
        if (MouseHit())
        {
            MOUSEMSG mouse = GetMessage();
            if (mouse.uMsg == WM_LBUTTONDOWN)
            {
                if (mouse.y > 0 && mouse.y < 350)
                {
                    bd.size = 3;
                    break;
                }
                else if (mouse.y > 350 && mouse.y < 550)
                {
                    bd.size = 4;
                    break;
                }
                else if (mouse.y > 550 && mouse.y < 690)
                {
                    bd.size = 5;
                    break;
                }
            }
        }
    }
}

/*-----
/ 函数名: chooseMode
/ 作 用: 选择游戏模式是PVP还是PVC
/ 参 数: mode, int型, 游戏模式

```

```

/          againGame, 布尔型, 是否继续游戏
/ 返回值: 无
/-----*/
void chooseMode(int &mode, bool& againGame)
{
    loadimage(0, _T("./图片/ModeChoose.jpg"), 1200, 800);
    while (mode == 0)
    {
        if (MouseHit())
        {
            MOUSEMSG mouse = GetMessage();
            if (mouse.uMsg == WM_LBUTTONDOWN)
            {
                if (mouse.x > 140 && mouse.x < 1060 && mouse.y > 160 && mouse.y < 350)
                {
                    mode = 1;    //PVP模式
                }
                else if (mouse.x > 140 && mouse.x < 1060 && mouse.y > 480 && mouse.y < 680)
                {
                    mode = 2;    //PVC模式
                }
                else if (mouse.x > 20 && mouse.x < 130 && mouse.y > 710 && mouse.y < 770)
                {
                    againGame = 0;    //退出游戏
                    break;
                }
            }
        }
    }
}

/-----*/
/ 函数名: chooseDiff
/ 作用: 选择难度是简单、中等还是困难
/ 参数: diff, 短整型, 游戏难度
/ 返回值: 无
/-----*/
void chooseDiff(int &diff)
{
    loadimage(0, _T("./图片/DiffChoose.jpg"), 1200, 800);
    while (!diff)
    {
        if (MouseHit())
        {
            MOUSEMSG mouse = GetMessage();
            if (mouse.uMsg == WM_LBUTTONDOWN)
            {
                if (mouse.y > 0 && mouse.y < 350)
                {
                    diff = 1;    //简单模式
                }
                else if (mouse.y > 350 && mouse.y < 550)
                {
                    diff = 2;    //中等模式
                }
                else if (mouse.y > 550 && mouse.y < 690)
                {
                    diff = 3;    //困难模式
                }
            }
        }
    }
}

/-----*/
/ 函数名: chooseOrder
/ 作用: 在PVC模式下, 选择玩家和电脑的出棋顺序
/ 参数: order, int型, 出棋顺序
/ 返回值: 无
/-----*/
void chooseOrder(int &order)
{
    loadimage(0, _T("./图片/OrderChoose.jpg"), 1200, 800);
    while (!order)
    {
        if (MouseHit())
        {
            MOUSEMSG mouse = GetMessage();
            if (mouse.uMsg == WM_LBUTTONDOWN)
            {
                if (mouse.y > 0 && mouse.y < 450)
                {
                    order = 1;    //玩家优先
                }
                else if (mouse.y > 450 && mouse.y < 800)
                {
                    order = 2;    //计算机优先
                }
            }
        }
    }
}

```

```

    }
}

/*-----
/ 函数名: chooseAgainPVC
/ 作用: 一局游戏结束后选择再试一次、返回主菜单、清空结果
/ 参数: againMode, 布尔型, 是否再试一次该模式下的游戏
        againPVC, 布尔型, 是否继续PVC
        cntRet[3], int型数组, 保存该模式下的游戏结果
/ 返回值: 无
/-----*/
void chooseAgainPVC(bool &againMode, bool &againPVC, int cntRet[3])
{
    bool isChoose = 0;
    IMAGE imageEmpty;
    loadimage(0, _T("./图片/PVCResDisplay.jpg"), 1200, 800);
    displayScorePVC(cntRet);
    while (!isChoose)
    {
        if (MouseHit())
        {
            MOUSEMSG mouse = GetMessage();
            if (mouse.uMsg == WM_LBUTTONDOWN)
            {
                if (mouse.x > 60 && mouse.x < 260 && mouse.y > 550 && mouse.y < 640)
                {
                    againMode = 1; //再来一次
                    isChoose = 1;
                } else if (mouse.x > 470 && mouse.x < 720 && mouse.y > 550 && mouse.y < 640)
                {
                    againPVC = 0; //返回主菜单
                    againMode = 0;
                    for (int i = 0; i < 3; i++) //清空得分
                    {
                        cntRet[i] = 0;
                    }
                    isChoose = 1;
                } else if (mouse.x > 920 && mouse.x < 1130 && mouse.y > 550 && mouse.y < 640)
                {
                    loadimage(&imageEmpty, _T("./图片/empty.jpg"), 140, 90);
                    putimage(315, 320, &imageEmpty);
                    putimage(530, 320, &imageEmpty);
                    putimage(720, 320, &imageEmpty);
                    for (int i = 0; i < 3; i++)
                    {
                        cntRet[i] = 0;
                    }
                }
            }
        }
    }
}

/*-----
/ 函数名: chooseAgainPVP
/ 作用: 一局游戏结束后选择再试一次、返回主菜单、清空结果
/ 参数: againPVC, 布尔型, 是否继续PVC
        cntRet[3], int型数组, 保存该模式下的游戏结果
/ 返回值: 无
/-----*/
void chooseAgainPVP(bool &againPVP, int cntRet[3])
{
    bool isChoose = 0;
    IMAGE imageEmpty;
    loadimage(0, _T("./图片/PVPResDisplay.jpg"), 1200, 800);
    displayScorePVP(cntRet);
    while (!isChoose)
    {
        if (MouseHit())
        {
            MOUSEMSG mouse = GetMessage();
            if (mouse.uMsg == WM_LBUTTONDOWN)
            {
                if (mouse.x > 85 && mouse.x < 290 && mouse.y > 715 && mouse.y < 790)
                {
                    againPVP = 1; //再来一次
                    isChoose = 1;
                } else if (mouse.x > 495 && mouse.x < 730 && mouse.y > 715 && mouse.y < 790)
                {
                    againPVP = 0; //返回主菜单
                    for (int i = 0; i < 3; i++) //清空得分
                    {
                        cntRet[i] = 0;
                    }
                    isChoose = 1;
                }
            }
        }
    }
}

```



```

    }
}
//副下三角对角线
for (int j = 1; j <= bd.size - 3; j++)
{
    for (int i = 0; i <= bd.size - 3 - j; i++)
    {
        if (bd.cell[i + j][bd.size - i - 1] != ' ' && bd.cell[i + j][bd.size - i - 1] == bd.cell[i + 1 + j][bd.size - i - 2] && bd.cell[i + 1 + j][bd.size - i - 2] == bd.cell[i + 2 + j][bd.size - i - 3])
        {
            return bd.cell[i][i + j] == '*' ? 1 : 2;
        }
    }
}

if (bd.emptyNum == 0)
{
    return 3; //平局
}
else
{
    return 0; //进行中
}
}

/*
/ 函数名: getVal
/ 作 用: 选择棋盘大小
/ 参 数: bd, 棋盘型, 保存棋盘结果
/         player, char型, 估值玩家
/ 返回值: int, 估值函数结果
*/
int getVal(Board& bd, char player)
{
    int val = 0;
    char oppo = player == '@' ? '*' : '@';

    //行
    for (int i = 0; i < bd.size; i++)
    {
        for (int j = 0; j <= bd.size - 3; j++)
        {
            if (bd.cell[i][j] != oppo && bd.cell[i][j + 1] != oppo && bd.cell[i][j + 2] != oppo)
            {
                val++;
            }
            if (bd.cell[i][j] == player && bd.cell[i][j + 1] == player && bd.cell[i][j + 2] == ' ')
            {
                val += 2;
            }
            if (bd.cell[i][j] == player && bd.cell[i][j + 1] == ' ' && bd.cell[i][j + 2] == player)
            {
                val += 2;
            }
            if (bd.cell[i][j] == ' ' && bd.cell[i][j + 1] == player && bd.cell[i][j + 2] == player)
            {
                val += 2;
            }
        }
    }

    //列
    for (int i = 0; i <= bd.size - 3; i++)
    {
        for (int j = 0; j < bd.size; j++)
        {
            if (bd.cell[i][j] != oppo && bd.cell[i + 1][j] != oppo && bd.cell[i + 2][j] != oppo)
            {
                val++;
            }
            if (bd.cell[i][j] == player && bd.cell[i + 1][j] == player && bd.cell[i + 2][j] == ' ')
            {
                val += 2;
            }
            if (bd.cell[i][j] == player && bd.cell[i + 1][j] == ' ' && bd.cell[i + 2][j] == player)
            {
                val += 2;
            }
            if (bd.cell[i][j] == ' ' && bd.cell[i + 1][j] == player && bd.cell[i + 2][j] == player)
            {
                val += 2;
            }
        }
    }

    //主上三角对角线
    for (int j = 0; j <= bd.size - 3; j++)
    {

```

```

for (int i = 0; i <= bd.size - 3 - j; i++)
{
    if (bd.cell[i][i + j] != oppo && bd.cell[i + 1][i + 1 + j] != oppo && bd.cell[i + 2][i + 2 + j] != oppo)
    {
        val++;
    }
    if (bd.cell[i][i + j] == player && bd.cell[i + 1][i + 1 + j] == player && bd.cell[i + 2][i + 2 + j] == ' ')
    {
        val += 2;
    }
    if (bd.cell[i][i + j] == player && bd.cell[i + 1][i + 1 + j] == ' ' && bd.cell[i + 2][i + 2 + j] == player)
    {
        val += 2;
    }
    if (bd.cell[i][i + j] == ' ' && bd.cell[i + 1][i + 1 + j] == player && bd.cell[i + 2][i + 2 + j] == player)
    {
        val += 2;
    }
}
}
//主下三角对角线
for (int j = 1; j <= bd.size - 3; j++)
{
    for (int i = 0; i <= bd.size - 3 - j; i++)
    {
        if (bd.cell[i + j][i] != oppo && bd.cell[i + 1 + j][i + 1] != oppo && bd.cell[i + 2 + j][i + 2] != oppo)
        {
            val++;
        }
        if (bd.cell[i + j][i] == player && bd.cell[i + 1 + j][i + 1] == player && bd.cell[i + 2 + j][i + 2] == ' ')
        {
            val += 2;
        }
        if (bd.cell[i + j][i] == player && bd.cell[i + 1 + j][i + 1] == ' ' && bd.cell[i + 2 + j][i + 2] == player)
        {
            val += 2;
        }
        if (bd.cell[i + j][i] == ' ' && bd.cell[i + 1 + j][i + 1] == player && bd.cell[i + 2 + j][i + 2] == player)
        {
            val += 2;
        }
    }
}
//副上三角对角线
for (int j = 0; j <= bd.size - 3; j++)
{
    for (int i = 0; i <= bd.size - 3 - j; i++)
    {
        if (bd.cell[i][bd.size - i - 1 - j] != oppo && bd.cell[i + 1][bd.size - i - 2 - j] != oppo && bd.cell[i + 2][bd.size - i - 3 - j] != oppo)
        {
            val++;
        }
        if (bd.cell[i][bd.size - i - 1 - j] == player && bd.cell[i + 1][bd.size - i - 2 - j] == player && bd.cell[i + 2][bd.size - i - 3 - j] == ' ')
        {
            val += 2;
        }
        if (bd.cell[i][bd.size - i - 1 - j] == player && bd.cell[i + 1][bd.size - i - 2 - j] == ' ' && bd.cell[i + 2][bd.size - i - 3 - j] == player)
        {
            val += 2;
        }
        if (bd.cell[i][bd.size - i - 1 - j] == ' ' && bd.cell[i + 1][bd.size - i - 2 - j] == player && bd.cell[i + 2][bd.size - i - 3 - j] == player)
        {
            val += 2;
        }
    }
}
//副下三角对角线
for (int j = 1; j <= bd.size - 3; j++)
{
    for (int i = 0; i <= bd.size - 3 - j; i++)
    {
        if (bd.cell[i + j][bd.size - i - 1] != oppo && bd.cell[i + 1 + j][bd.size - i - 2] != oppo && bd.cell[i + 2 + j][bd.size - i - 3] != oppo)
        {
            val++;
        }
        if (bd.cell[i + j][bd.size - i - 1] == player && bd.cell[i + 1 + j][bd.size - i - 2] == player && bd.cell[i + 2 + j][bd.size - i - 3] == ' ')
        {
            val += 2;
        }
        if (bd.cell[i + j][bd.size - i - 1] == player && bd.cell[i + 1 + j][bd.size - i - 2] == ' ' && bd.cell[i + 2 + j][bd.size - i - 3] == player)
        {
            val += 2;
        }
    }
}

```



```

j][bd.size - i - 3] == player)
{
    val -= 2;
}
if (bd.cell[i + j][bd.size - i - 1] == ' ' && bd.cell[i + 1 + j][bd.size - i - 2] == player && bd.cell[i + 2 +
j][bd.size - i - 3] == player)
{
    val -= 2;
}
}

//行
for (int i = 0; i < bd.size; i++)
{
    for (int j = 0; j <= bd.size - 3; j++)
    {
        if (bd.cell[i][j] != player && bd.cell[i][j + 1] != player && bd.cell[i][j + 2] != player)
        {
            val--;
        }
        if (bd.cell[i][j] == oppo && bd.cell[i][j + 1] == oppo && bd.cell[i][j + 2] == ' ')
        {
            val -= 2;
        }
        if (bd.cell[i][j] == oppo && bd.cell[i][j + 1] == ' ' && bd.cell[i][j + 2] == oppo)
        {
            val -= 2;
        }
        if (bd.cell[i][j] == ' ' && bd.cell[i][j + 1] == oppo && bd.cell[i][j + 2] == oppo)
        {
            val -= 2;
        }
    }
}

//列
for (int i = 0; i <= bd.size - 3; i++)
{
    for (int j = 0; j < bd.size; j++)
    {
        if (bd.cell[i][j] != player && bd.cell[i + 1][j] != player && bd.cell[i + 2][j] != player)
        {
            val--;
        }
        if (bd.cell[i][j] == oppo && bd.cell[i + 1][j] == oppo && bd.cell[i + 2][j] == ' ')
        {
            val -= 2;
        }
        if (bd.cell[i][j] == oppo && bd.cell[i + 1][j] == ' ' && bd.cell[i + 2][j] == oppo)
        {
            val -= 2;
        }
        if (bd.cell[i][j] == ' ' && bd.cell[i + 1][j] == oppo && bd.cell[i + 2][j] == oppo)
        {
            val -= 2;
        }
    }
}

//主上三角对角线
for (int j = 0; j <= bd.size - 3; j++)
{
    for (int i = 0; i <= bd.size - 3 - j; i++)
    {
        if (bd.cell[i][i + j] != player && bd.cell[i + 1][i + 1 + j] != player && bd.cell[i + 2][i + 2 + j] != player)
        {
            val--;
        }
        if (bd.cell[i][i + j] == oppo && bd.cell[i + 1][i + 1 + j] == oppo && bd.cell[i + 2][i + 2 + j] == ' ')
        {
            val -= 2;
        }
        if (bd.cell[i][i + j] == oppo && bd.cell[i + 1][i + 1 + j] == ' ' && bd.cell[i + 2][i + 2 + j] == oppo)
        {
            val -= 2;
        }
        if (bd.cell[i][i + j] == ' ' && bd.cell[i + 1][i + 1 + j] == oppo && bd.cell[i + 2][i + 2 + j] == oppo)
        {
            val -= 2;
        }
    }
}

//主下三角对角线
for (int j = 1; j <= bd.size - 3; j++)
{
    for (int i = 0; i <= bd.size - 3 - j; i++)

```

```

        {
            if (bd.cell[i + j][i] != player && bd.cell[i + 1 + j][i + 1] != player && bd.cell[i + 2 + j][i + 2] != player)
            {
                val--;
            }
            if (bd.cell[i + j][i] == oppo && bd.cell[i + 1 + j][i + 1] == oppo && bd.cell[i + 2 + j][i + 2] == ' ')
            {
                val -= 2;
            }
            if (bd.cell[i + j][i] == oppo && bd.cell[i + 1 + j][i + 1] == ' ' && bd.cell[i + 2 + j][i + 2] == oppo)
            {
                val -= 2;
            }
            if (bd.cell[i + j][i] == ' ' && bd.cell[i + 1 + j][i + 1] == oppo && bd.cell[i + 2 + j][i + 2] == oppo)
            {
                val -= 2;
            }
        }
    }

    //副上三角对角线
    for (int j = 0; j <= bd.size - 3; j++)
    {
        for (int i = 0; i <= bd.size - 3 - j; i++)
        {
            if (bd.cell[i][bd.size - i - 1 - j] != player && bd.cell[i + 1][bd.size - i - 2 - j] != player && bd.cell[i + 2][bd.size - i - 3 - j] != player)
            {
                val--;
            }
            if (bd.cell[i][bd.size - i - 1 - j] == oppo && bd.cell[i + 1][bd.size - i - 2 - j] == oppo && bd.cell[i + 2][bd.size - i - 3 - j] == ' ')
            {
                val -= 2;
            }
            if (bd.cell[i][bd.size - i - 1 - j] == oppo && bd.cell[i + 1][bd.size - i - 2 - j] == ' ' && bd.cell[i + 2][bd.size - i - 3 - j] == oppo)
            {
                val -= 2;
            }
            if (bd.cell[i][bd.size - i - 1 - j] == ' ' && bd.cell[i + 1][bd.size - i - 2 - j] == oppo && bd.cell[i + 2][bd.size - i - 3 - j] == oppo)
            {
                val -= 2;
            }
        }
    }

    //副下三角对角线
    for (int j = 1; j <= bd.size - 3; j++)
    {
        for (int i = 0; i <= bd.size - 3 - j; i++)
        {
            if (bd.cell[i + j][bd.size - i - 1] != player && bd.cell[i + 1 + j][bd.size - i - 2] != player && bd.cell[i + 2 + j][bd.size - i - 3] != player)
            {
                val--;
            }
            if (bd.cell[i + j][bd.size - i - 1] == oppo && bd.cell[i + 1 + j][bd.size - i - 2] == oppo && bd.cell[i + 2 + j][bd.size - i - 3] == ' ')
            {
                val -= 2;
            }
            if (bd.cell[i + j][bd.size - i - 1] == oppo && bd.cell[i + 1 + j][bd.size - i - 2] == ' ' && bd.cell[i + 2 + j][bd.size - i - 3] == oppo)
            {
                val -= 2;
            }
            if (bd.cell[i + j][bd.size - i - 1] == ' ' && bd.cell[i + 1 + j][bd.size - i - 2] == oppo && bd.cell[i + 2 + j][bd.size - i - 3] == oppo)
            {
                val -= 2;
            }
        }
    }

    return val;
}

/*
/ 函数名: inputChess
/ 作 用: 检测鼠标选择的落棋位置
/ 参 数: bd, 棋盘型, 保存棋盘结果
/         player, 字符型, 落棋的玩家
/ 返回值: 无
/-----*/
void inputChess(Board& bd, char player)
{
    //行分别编号为ABC, 列分别编号为123

```

```

bool isChoose = false;
string ret = "00";
while (!isChoose)
{
    if (MouseHit())
    {
        MOUSEMSG mouse = GetMessage();
        if (mouse.uMsg == WM_LBUTTONDOWN)
        {
            if (bd.size == 3)
            {
                // 检查行
                if (mouse.y >= 150 && mouse.y < 350)
                {
                    ret[0] = 'A';
                }
                else if (mouse.y >= 350 && mouse.y < 550)
                {
                    ret[0] = 'B';
                }
                else if (mouse.y >= 550 && mouse.y < 750)
                {
                    ret[0] = 'C';
                }
                // 检查列
                if (mouse.x >= 100 && mouse.x < 300)
                {
                    ret[1] = '1';
                }
                else if (mouse.x >= 300 && mouse.x < 500)
                {
                    ret[1] = '2';
                }
                else if (mouse.x >= 500 && mouse.x < 700)
                {
                    ret[1] = '3';
                }
            }

            if (bd.size == 4)
            {
                // 检查行
                if (mouse.y >= 150 && mouse.y < 300)
                {
                    ret[0] = 'A';
                }
                else if (mouse.y >= 300 && mouse.y < 450)
                {
                    ret[0] = 'B';
                }
                else if (mouse.y >= 450 && mouse.y < 600)
                {
                    ret[0] = 'C';
                }
                else if (mouse.y >= 600 && mouse.y < 750)
                {
                    ret[0] = 'D';
                }
                // 检查列
                if (mouse.x >= 100 && mouse.x < 250)
                {
                    ret[1] = '1';
                }
                else if (mouse.x >= 250 && mouse.x < 400)
                {
                    ret[1] = '2';
                }
                else if (mouse.x >= 400 && mouse.x < 550)
                {
                    ret[1] = '3';
                }
                else if (mouse.x >= 550 && mouse.x < 700)
                {
                    ret[1] = '4';
                }
            }

            if (bd.size == 5)
            {
                // 检查行
                if (mouse.y >= 150 && mouse.y < 270)
                {
                    ret[0] = 'A';
                }
                else if (mouse.y >= 270 && mouse.y < 390)
                {
                    ret[0] = 'B';
                }
            }
        }
    }
}

```

```

        else if (mouse.y >= 390 && mouse.y < 510)
        {
            ret[0] = 'C';
        }
        else if (mouse.y >= 510 && mouse.y < 630)
        {
            ret[0] = 'D';
        }
        else if (mouse.y >= 630 && mouse.y < 750)
        {
            ret[0] = 'E';
        }
        //检查列
        if (mouse.x >= 100 && mouse.x < 220)
        {
            ret[1] = '1';
        }
        else if (mouse.x >= 220 && mouse.x < 340)
        {
            ret[1] = '2';
        }
        else if (mouse.x >= 340 && mouse.x < 460)
        {
            ret[1] = '3';
        }
        else if (mouse.x >= 460 && mouse.x < 580)
        {
            ret[1] = '4';
        }
        else if (mouse.x >= 580 && mouse.x < 700)
        {
            ret[1] = '5';
        }
    }
}

//判断鼠标点击是否在棋盘上且为该位置无棋子
if (ret[0] != '0' && ret[1] != '0' && bd.cell[ret[0] - 'A'][ret[1] - '1'] == ' ')
{
    isChoose = true;
}

bd.cell[ret[0] - 'A'][ret[1] - '1'] = player;
displayChess(bd, ret, player);
}

/*
/ 函数名: displayMes
/ 作用: 展示轮到谁下棋
/ 参数: player, 字符型, 轮到落棋的玩家
/ 返回值: 无
*/
void displayMes(char player)
{
    IMAGE imageXmin, imageOmin;
    if (player == '*')
    {
        loadimage(&imageXmin, _T("./图片/Xmini.jpg"), 100, 100);
        putimage(1040, 190, &imageXmin);
    } else
    {
        loadimage(&imageOmin, _T("./图片/Omini.jpg"), 100, 100);
        putimage(1040, 190, &imageOmin);
    }
}

/*
/ 函数名: displayChess
/ 作用: 展示该轮落棋的棋子
/ 参数: bd, 棋盘型, 保存棋盘结果
/         loc, 字符串型, 落棋的位置
/         player, 字符型, 落棋的玩家
/ 返回值: 无
*/
void displayChess(Board& bd, string loc, char player)
{
    IMAGE image;
    if (bd.size == 3)
    {
        if (player == '*')
        {
            loadimage(&image, _T("./图片/X.jpg"), 180, 180);
        }
        else if (player == '@')
        {
            loadimage(&image, _T("./图片/O.jpg"), 180, 180);
        }
    }
}

```

```

        if (loc == "A1") putimage(110, 160, &image);
    else if (loc == "A2") putimage(314, 160, &image);
    else if (loc == "A3") putimage(518, 160, &image);
    else if (loc == "B1") putimage(110, 360, &image);
    else if (loc == "B2") putimage(314, 360, &image);
    else if (loc == "B3") putimage(518, 360, &image);
    else if (loc == "C1") putimage(110, 560, &image);
    else if (loc == "C2") putimage(314, 560, &image);
    else if (loc == "C3") putimage(518, 560, &image);
}

if (bd.size == 4)
{
    if (player == '*')
    {
        loadimage(&image, _T("./图片/X.jpg"), 130, 130);
    }
    else if (player == '@')
    {
        loadimage(&image, _T("./图片/@.jpg"), 130, 130);
    }

    if (loc == "A1") putimage(110, 160, &image);
    else if (loc == "A2") putimage(260, 160, &image);
    else if (loc == "A3") putimage(410, 160, &image);
    else if (loc == "A4") putimage(560, 160, &image);
    else if (loc == "B1") putimage(110, 310, &image);
    else if (loc == "B2") putimage(260, 310, &image);
    else if (loc == "B3") putimage(410, 310, &image);
    else if (loc == "B4") putimage(560, 310, &image);
    else if (loc == "C1") putimage(110, 460, &image);
    else if (loc == "C2") putimage(260, 460, &image);
    else if (loc == "C3") putimage(410, 460, &image);
    else if (loc == "C4") putimage(560, 460, &image);
    else if (loc == "D1") putimage(110, 610, &image);
    else if (loc == "D2") putimage(260, 610, &image);
    else if (loc == "D3") putimage(410, 610, &image);
    else if (loc == "D4") putimage(560, 610, &image);
}

if (bd.size == 5)
{
    if (player == '*')
    {
        loadimage(&image, _T("./图片/X.jpg"), 100, 100);
    }
    else if (player == '@')
    {
        loadimage(&image, _T("./图片/@.jpg"), 100, 100);
    }

    if (loc == "A1") putimage(110, 160, &image);
    else if (loc == "A2") putimage(230, 160, &image);
    else if (loc == "A3") putimage(350, 160, &image);
    else if (loc == "A4") putimage(470, 160, &image);
    else if (loc == "A5") putimage(590, 160, &image);
    else if (loc == "B1") putimage(110, 280, &image);
    else if (loc == "B2") putimage(230, 280, &image);
    else if (loc == "B3") putimage(350, 280, &image);
    else if (loc == "B4") putimage(470, 280, &image);
    else if (loc == "B5") putimage(590, 280, &image);
    else if (loc == "C1") putimage(110, 400, &image);
    else if (loc == "C2") putimage(230, 400, &image);
    else if (loc == "C3") putimage(350, 400, &image);
    else if (loc == "C4") putimage(470, 400, &image);
    else if (loc == "C5") putimage(590, 400, &image);
    else if (loc == "D1") putimage(110, 520, &image);
    else if (loc == "D2") putimage(230, 520, &image);
    else if (loc == "D3") putimage(350, 520, &image);
    else if (loc == "D4") putimage(470, 520, &image);
    else if (loc == "D5") putimage(590, 520, &image);
    else if (loc == "E1") putimage(110, 640, &image);
    else if (loc == "E2") putimage(230, 640, &image);
    else if (loc == "E3") putimage(350, 640, &image);
    else if (loc == "E4") putimage(470, 640, &image);
    else if (loc == "E5") putimage(590, 640, &image);
}
}

/*
/ 函数名: displayRet
/ 作 用: 展示该轮游戏结果
/ 参 数: ret, 字符串型, 游戏结果
/ 返回值: 无
*/
void displayRet(int ret)
{
    IMAGE imageXwin, imageOwin, imageDraw;
    switch (ret)
    {

```

```

        case 1:
            loadimage(&imageXwin, _T("./图片/Xwin.jpg"), 390, 500);
            putimage(780, 100, &imageXwin);
            break;
        case 2:
            loadimage(&image0win, _T("./图片/@win.jpg"), 390, 500);
            putimage(780, 100, &image0win);
            break;
        case 3:
            loadimage(&imageDraw, _T("./图片/draw.jpg"), 390, 500);
            putimage(780, 100, &imageDraw);
            break;
    }
    Sleep(3000);
}

/*-----
/ 函数名: displayScorePVC
/ 作 用: 展示PVC模式该难度下的得分结果
/ 参 数: cntRet, 字符型数组, 得分结果
/ 返回值: 无
*/-----*/
void displayScorePVC(int cntRet[3])
{
    TCHAR s1[5], s2[5], s3[5];

    setbkcolor(RGB(255, 127, 38));
    settextrstyle(80, 40, _T("Times New Roman"));
    settextrcolor(WHITE);
    swprintf_s(s1, _T("%d"), cntRet[0]);
    swprintf_s(s2, _T("%d"), cntRet[1]);
    swprintf_s(s3, _T("%d"), cntRet[2]);
    outtextxy(340, 310, s1);
    outtextxy(550, 310, s2);
    outtextxy(760, 310, s3);
}

/*-----
/ 函数名: displayScorePVP
/ 作 用: 展示PVP模式该难度下的得分结果
/ 参 数: cntRet, int数组, 得分结果
/ 返回值: 无
*/-----*/
void displayScorePVP(int cntRet[3])
{
    TCHAR s1[5], s2[5], s3[5], s4[5], s5[5], s6[5];

    setbkcolor(RGB(255, 127, 38));
    settextrstyle(80, 40, _T("Times New Roman"));
    settextrcolor(WHITE);
    swprintf_s(s4, _T("%d"), cntRet[0]);
    swprintf_s(s5, _T("%d"), cntRet[1]);
    swprintf_s(s6, _T("%d"), cntRet[2]);
    swprintf_s(s1, _T("%d"), cntRet[1]);
    swprintf_s(s2, _T("%d"), cntRet[0]);
    swprintf_s(s3, _T("%d"), cntRet[2]);
    outtextxy(470, 190, s1);
    outtextxy(665, 190, s2);
    outtextxy(885, 190, s3);
    outtextxy(470, 495, s4);
    outtextxy(665, 495, s5);
    outtextxy(885, 495, s6);
}

/*-----
/ 函数名: ModePVC
/ 作 用: 进入PVC模式, 选择PVC模式下的难度
/ 参 数: 无
/ 返回值: 无
*/-----*/
void ModePVC(void)
{
    int diff = 0, order = 0;
    bool againPVC = 1;
    Board bd;

    chooseSize(bd);
    initBoard(bd, bd.size);
    chooseDiff(diff);
    chooseOrder(order);

    while (againPVC)
    {
        switch (diff)
        {
            case 1:

```

```

        ModeDiff(bd, order, againPVC, inputModeEasy);
        break;
    case 3:
        ModeDiff(bd, order, againPVC, inputModeHard);
        break;
    }

}

destroyBoard(bd);
}

/*-----*/
/ 函数名: ModePVP
/ 作用: 进入PVP模式, 双方玩家对弈
/ 参数: 无
/ 返回值: 无
/-----*/
void ModePVP(void)
{
    bool againPVP = true;
    char player = '@'; // @玩家优先
    int ret = 0;
    int cntRet[3] = { 0 };
    Board bd;

    chooseSize(bd);
    initBoard(bd, bd.size);

    while (againPVP)
    {
        if (bd.size == 3)
        {
            loadimage(0, _T("./图片/Board_3-3.jpg"), 1200, 800);
        }
        else if (bd.size == 4)
        {
            loadimage(0, _T("./图片/Board_4-4.jpg"), 1200, 800);
        }
        else if (bd.size == 5)
        {
            loadimage(0, _T("./图片/Board_5-5.jpg"), 1200, 800);
        }
        while (!ret)
        {
            displayMes(player);
            if (player == '*') // 玩家1
            {
                inputChess(bd, player);
            }
            else if (player == '@') // 玩家2
            {
                inputChess(bd, player);
            }
            player = player == '* ? '@' : '*'; // 调换出棋顺序
            bd.emptyNum--;
            ret = getRet(bd);
        }
        if (ret == 1)
        {
            cntRet[0]++;
        }
        else if (ret == 2)
        {
            cntRet[1]++;
        }
        else if (ret == 3)
        {
            cntRet[2]++;
        }
        displayRet(ret);
        chooseAgainPVP(againPVP, cntRet);
        /*清空棋盘*/
        bd.emptyNum = bd.size * bd.size;
        ret = 0;
        for (int i = 0; i < bd.size; i++)
        {
            for (int j = 0; j < bd.size; j++)
            {
                bd.cell[i][j] = ' ';
            }
        }
    }
    destroyBoard(bd);
}

```

```

/*-----*/
/ 函数名: ModeDiff
/ 作用: PVC模式, 玩家与电脑对弈
/ 参数: bd, 棋盘型, 保存棋盘结果
/         order, 短整型, 玩家与电脑出棋的顺序
/         inputModeDiff, 函数指针, 不同难度下电脑的出棋结果
/ 返回值: 无
/*-----*/
void ModeDiff(Board& bd, int& order, bool &againPVC, void (* inputModeDiff)(Board& bd))
{
    bool againDiff = 1;
    //short cntStep = 0;
    char player = '\0';
    short ret = 0;
    int cntRet[3] = { 0 };

    if (order == 1)
    {
        player = '*';
    }else if (order == 2)
    {
        player = '@';
    }
    while (againDiff)
    {
        if (bd.size == 3)
        {
            loadimage(0, _T("./图片/Board_3-3.jpg"), 1200, 800);
        }
        else if (bd.size == 4)
        {
            loadimage(0, _T("./图片/Board_4-4.jpg"), 1200, 800);
        }
        else if (bd.size == 5)
        {
            loadimage(0, _T("./图片/Board_5-5.jpg"), 1200, 800);
        }

        while (ret == 0)
        {
            displayMes(player);
            if (player == '*') //玩家
            {
                inputChess(bd, player);
            }else if (player == '@') //计算机
            {
                (*inputModeDiff)(bd);
            }
            bd.emptyNum--;
            player = player == '* ? '@' : '*'; //调换出棋顺序
            //cntStep++;
            ret = getRet(bd);
        }
        if (ret == 1)
        {
            cntRet[0]++;
        }else if (ret == 2)
        {
            cntRet[1]++;
        }else if (ret == 3)
        {
            cntRet[2]++;
        }
        displayRet(ret);
        chooseAgainPVC(againDiff, againPVC, cntRet);
        bd.emptyNum = bd.size * bd.size;
        //cntStep = 0;
        ret = 0;
        for (int i = 0; i < bd.size; i++)
        {
            for (int j = 0; j < bd.size; j++)
            {
                bd.cell[i][j] = ' ';
            }
        }
    }
}

/*-----*/
/ 函数名: inputModeEasy
/ 作用: 简单模式, 得到电脑的出棋结果
/ 参数: bd, 棋盘型, 保存棋盘结果
/ 返回值: 无
/*-----*/
void inputModeEasy(Board& bd)
{

```



```

char n = '1';

Sleep(1000);
//空格编号
for (int i = 0; i < bd.size; i++)
{
    for (int j = 0; j < bd.size; j++)
    {
        if ((bd.cell[i][j] != '*' ) && (bd.cell[i][j] != '@'))
        {
            bd.cell[i][j] = n++;
        }
    }
}

srand((unsigned)time(NULL));
int ran = rand() % (n - '1') + '1';
for (int i = 0; i < bd.size; i++)
{
    for (int j = 0; j < bd.size; j++)
    {
        if (bd.cell[i][j] == ran)
        {
            bd.cell[i][j] = '@';
            char row = 'A' + i;
            char col = '1' + j;
            string ret = {row, col};
            displayChess(bd, ret, '@');
        }
    }
}
//编号复原
for (int i = 0; i < bd.size; i++)
{
    for (int j = 0; j < bd.size; j++)
    {
        if ((bd.cell[i][j] != '*' ) && (bd.cell[i][j] != '@'))
        {
            bd.cell[i][j] = ' ';
        }
    }
}
}

/*-----*/
/ 函数名: inputModeHard
/ 作 用: 困难模式, 得到电脑的出棋结果
/ 参 数: bd, 棋盘型, 保存棋盘结果
/ 返回值: 无
/*-----*/
void inputModeHard(Board& bd)
{
    //Sleep(1000);
    Tree T = new Node;
    T->alpha = NINF;
    T->beta = PINF;
    T->level = MAX;
    T->row = -1;
    T->col = -1;
    T->firstChild = NULL;
    T->nextSibling = NULL;

    if (bd.size == 3 && bd.emptyNum == bd.size * bd.size)
    {
        T->row = 2;
        T->col = 0;
    }
    else if (bd.size == 4 && bd.emptyNum == bd.size * bd.size)
    {
        T->row = 2;
        T->col = 2;
    }
    else if (bd.size == 5 && bd.emptyNum == bd.size * bd.size)
    {
        T->row = 4;
        T->col = 4;
    }
    else
    {
        MinMax(bd, T, '@', 0);
    }

    if (T->row != -1 && T->col != -1)
    {
        bd.cell[T->row][T->col] = '@';
        char row = 'A' + T->row;

```

```

        char col = '1' + T->col;
        string ret = { row, col };
        displayChess(bd, ret, 'e');
    }
    else
    {
        //inputModeEasy(bd);
    }
}

/*-----
/  函数名: MinMax
/  作  用: 实现极大极小搜索算法
/  参  数: bd, 棋盘型, 保存棋盘结果
           T, 结点指针, 指向子树的根结点
           player, 字符型, 上一层对应的玩家
           step, 整型, 迭代步长
/  返回值: 无
/*-----*/
void MinMax(Board& bd, Node* T, char player, int step)
{
    bool isFirst = true;
    char oppo = player == 'e' ? '*' : 'e';
    Tree q = NULL;
    int lim = 1;
    if (bd.size == 3)    lim = 7;
    if (bd.size == 4)    lim = 2;
    if (bd.size == 5)    lim = 1;

    if (step <= lim && getRet(bd) == 0)
    {
        for (int i = 0; i < bd.size; i++)
        {
            for (int j = 0; j < bd.size; j++)
            {
                if (bd.cell[i][j] == ' ')
                {
                    bd.cell[i][j] = oppo;
                    bd.emptyNum--;

                    Node* t = new Node;
                    t->row = i;
                    t->col = j;
                    t->val = 0;
                    t->alpha = T->alpha;
                    t->beta = T->beta;
                    t->level = !T->level;
                    t->firstChild = NULL;
                    t->nextSibling = NULL;
                    /*
                    for (int k = 0; k < bd.size; k++)
                    {
                        for (int l = 0; l < bd.size; l++)
                        {
                            cout << bd.cell[k][l] << "\t";
                        }
                        cout << endl;
                    }
                    cout << endl;
                    */
                    if (isFirst)
                    {
                        T->firstChild = t;
                        isFirst = false;
                    }
                    else
                    {
                        q->nextSibling = t;
                    }
                    q = t;

                    cout << "Tlevel:" << T->level << " Tval: " << T->val << " " << T->alpha << " " << T->beta << endl;
                    if (T->alpha < T->beta)
                    {
                        MinMax(bd, q, oppo, step + 1);
                    }

                    Node* p = T->firstChild;
                    while (p)
                    {
                        //cout << T->level << " : " << p->val << endl;
                        if (T->level == MIN)
                        {
                            if (p->val < T->beta)
                            {
                                T->beta = p->val;
                                T->val = p->val;

```

```

        T->row = p->row;
        T->col = p->col;
    }
}
else if (T->level == MAX)
{
    if (p->val > T->alpha)
    {
        T->alpha = p->val;
        T->val = p->val;
        T->row = p->row;
        T->col = p->col;
    }
    p = p->nextSibling;
}
if (T->firstChild && T->firstChild->nextSibling == NULL)
{
    T->val = T->firstChild->val;
    T->row = T->firstChild->row;
    T->col = T->firstChild->col;
}

//cout << T->level << " T: " << T->val << endl;

bd.cell[i][j] = ' ';
bd.emptyNum++;
    }
}
}
else
{
    T->val = getVal(bd, player);
    //if (T->level)
    //    cout <<"MAX " << T->val << endl;
    //if (!T->level)
    //    cout << "MIN " << T->val << endl;
}
}
/*****FUNCTION*END*****/
/*****FILE*END*****/

```