计算概论 A 大作业（亚马逊棋）报告

老师：李戈教授

姓名：尤俊浩（1900094810）　　姓名：汪蔚涝（1900094813）

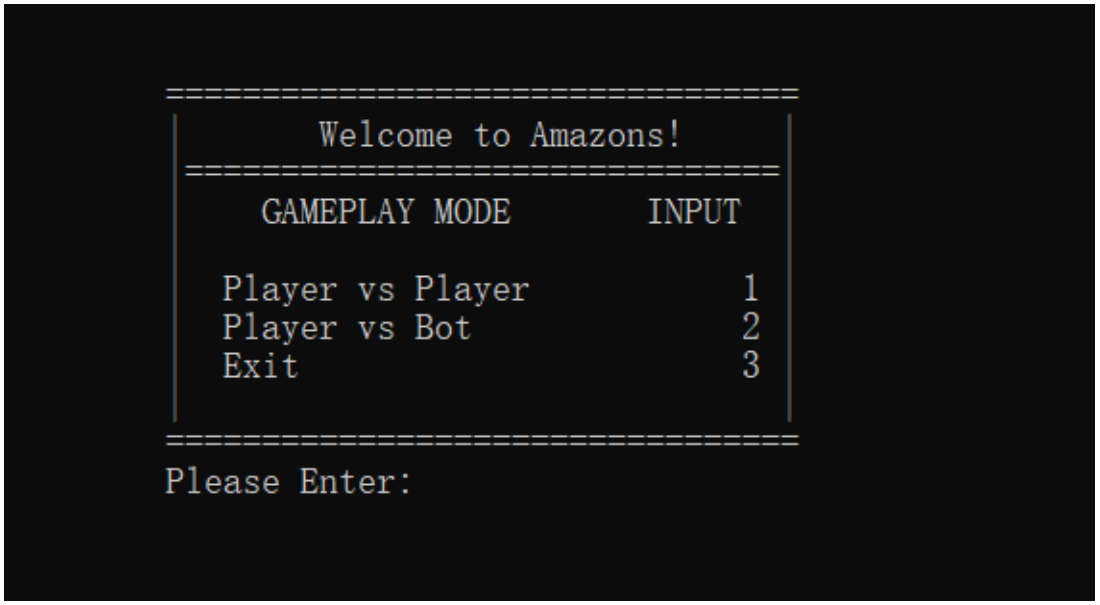
概要：

这是一个 8 x 8 的棋盘。本篇报告将介绍所有的界面和解释在代码中所有 26 个函数的作用和使用。其中，黑棋将被赋值 1，而白棋则被赋值-1，障碍被赋值 2。界面分为三类，有第一界面、人人对弈界面、人机对弈界面和游戏界面。函数分为六大类，其中有 7 个界面类函数、5 个判断类函数、4 个移动类、3 个算法类、6 个存储类和 1 个 main 函数。

界面：

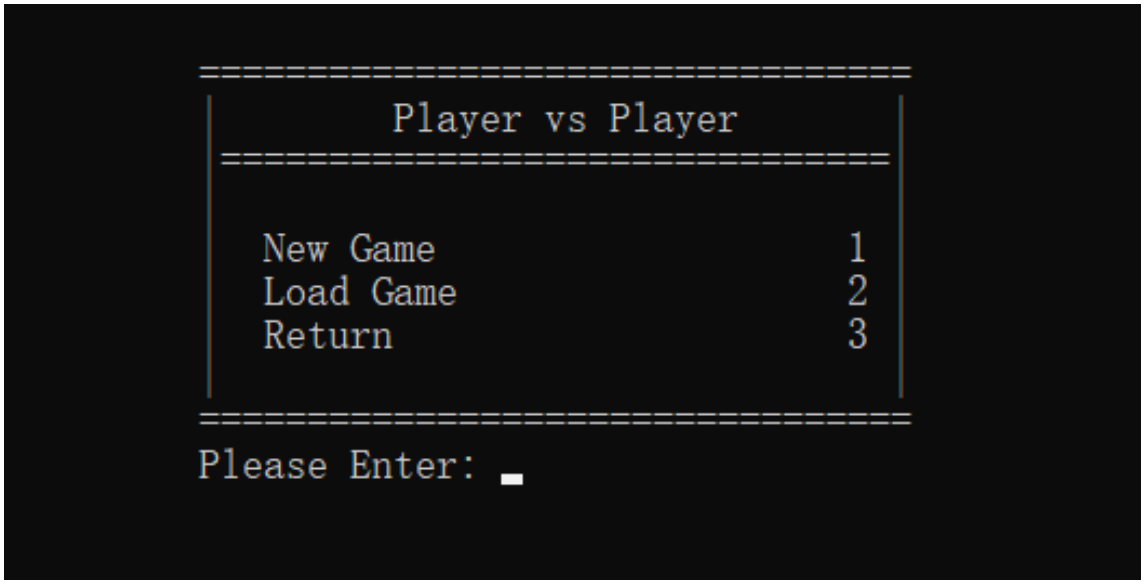
(a) 第一界面

1. 可以选择要人人对弈、人机对弈或退出游戏。

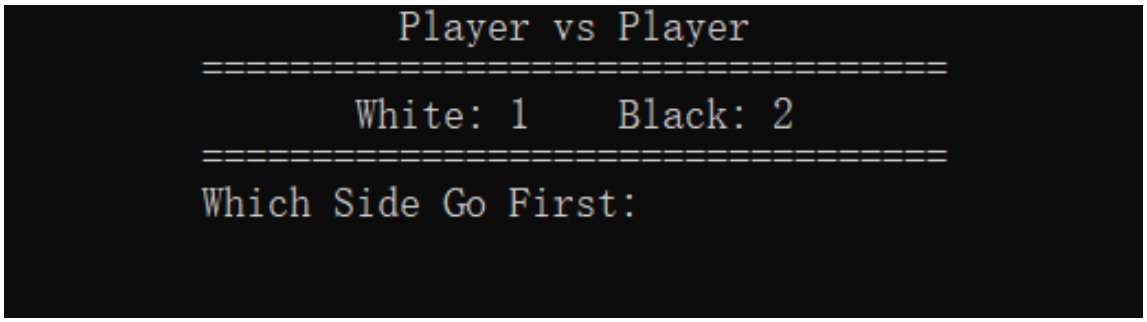


(b) 人人对弈

1. 可以选择要重新开始游戏、复盘之前已存储的游戏，或返回第一界面。

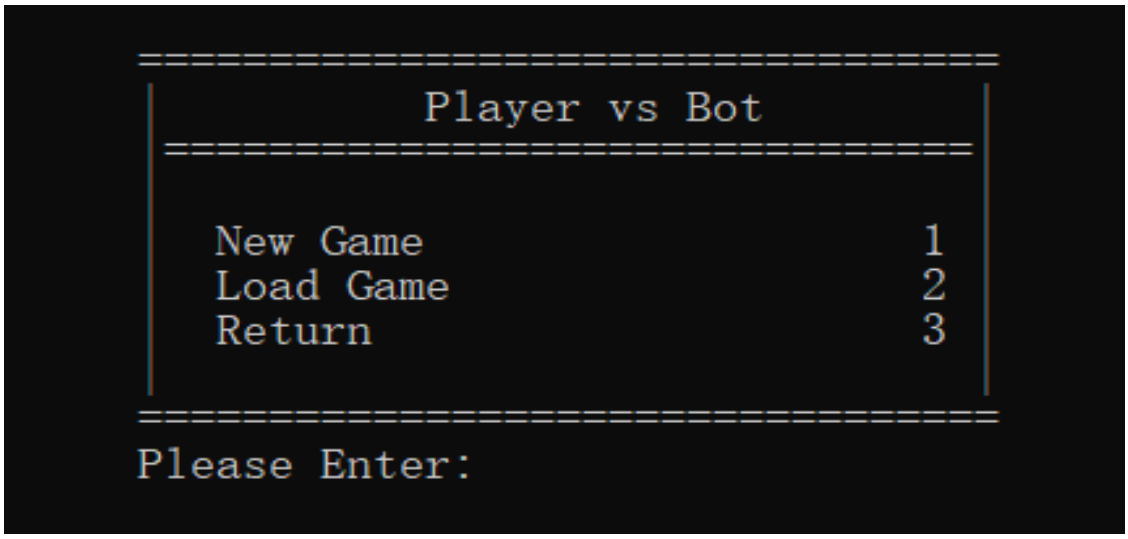


2. 选择哪一方先行棋

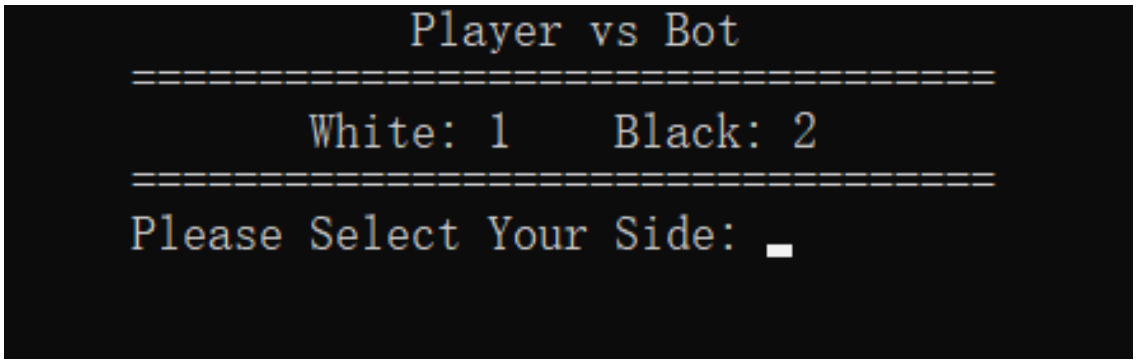


(c) 人机对弈

1. 可以选择要重新开始游戏、复盘之前已存储的游戏，或返回第一界面。

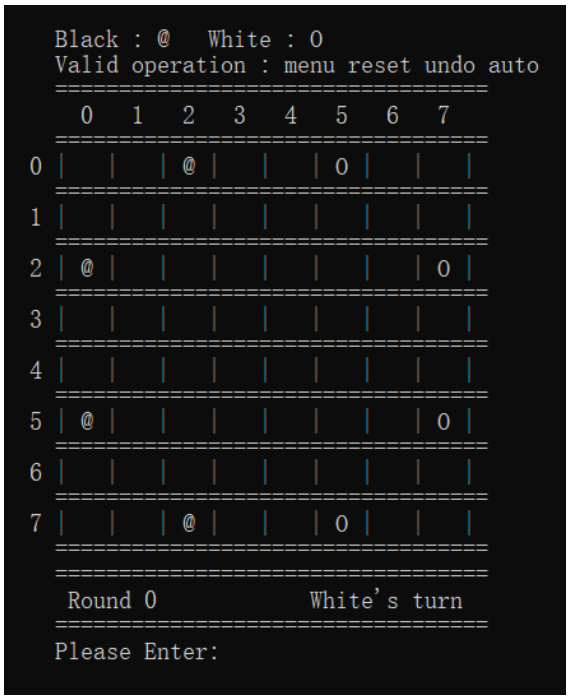


2. 玩家选择棋子的颜色

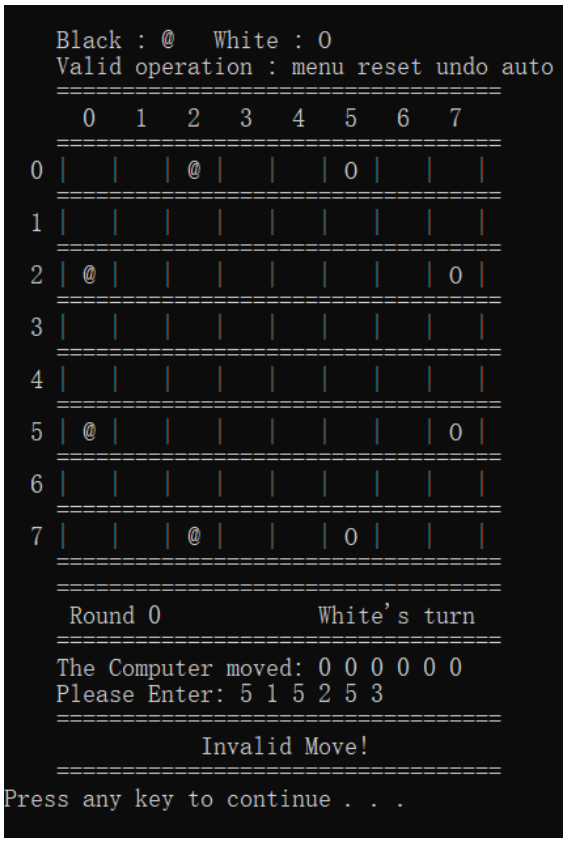


(d) 游戏界面

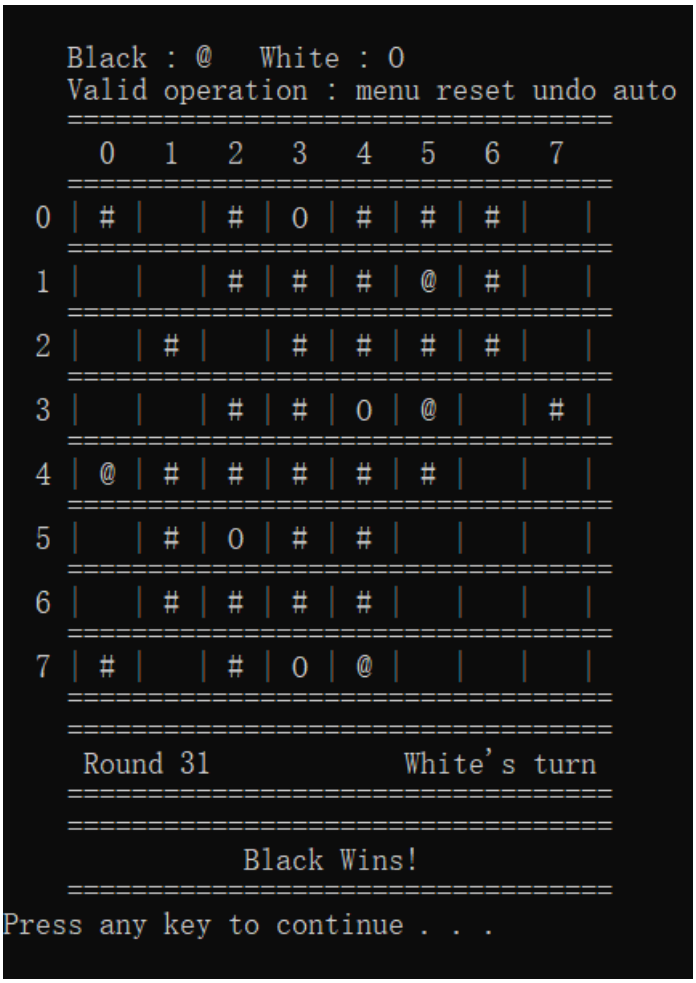
1. 黑方在棋盘里为@，白方在棋盘里为 O。玩家需输入三个坐标（要移动棋子的坐标，移动至哪的坐标和布置障碍的坐标)。移动规则和西洋棋皇后走法一样。



2. 非法走步



3. 一方赢棋



函数：

(a) 界面类

1. 初始化棋盘

```
void resetGame()
{
	currRound = 0;
	ansx1 = ansx2 = ansx3 = ansy1 = ansy2 = ansy3 = 0;
	memset(gridInfo, 0, sizeof(gridInfo));
	gridInfo[0][2] = gridInfo[2][0] = gridInfo[5][0] = gridInfo[7][2] = grid_black;
	gridInfo[0][5] = gridInfo[2][7] = gridInfo[5][7] = gridInfo[7][5] = grid_white;
}
```

2. 绘制棋盘

```
void printTable()
{
	system("cls");
	int x1 = 0, x2 = 0, y1 = 0, y2 = 0;
	cout << endl << endl;
	cout << "    Black : @   White : O             " << endl;
	cout << "    Valid operation : menu reset undo auto " << endl;
	cout << "    ================================" << endl;
	cout << "      0   1   2   3   4   5   6   7   " << endl;
	cout << "    ================================" << endl;
	for (int i = 0; i < GRIDSIZE; i++)
	{
		cout << "  " << i << " |";
		for (int j = 0; j < GRIDSIZE; j++)
		{
			if (gridInfo[i][j] == grid_black)cout << " @ |";
			else if (gridInfo[i][j] == grid_white)cout << " O |";
			else if (gridInfo[i][j] == OBSTACLE)cout << " # |";
			else cout << "   |";
		}
		cout << endl << "    ================================" << endl;
	}
	cout << "    ================================" << endl;
	cout << "     Round " << currRound << "           ";
	cout << (currBotColor == grid_white ? " White's turn" : " Black's turn") << endl;
	cout << "    ================================" << endl;
}
```

3. 主要界面

```
void mainMenu()
{
	system("CLS");
	cout << endl;
	cout << endl;
	cout << "         ================================" << endl;
	cout << "         |       Welcome to Amazons!     |" << endl;
	cout << "         |===============================|" << endl;
	cout << "         |    GAMEPLAY MODE       INPUT  |" << endl;
	cout << "         |                               |" << endl;
	cout << "         |  Player vs Player           1 |" << endl;
	cout << "         |  Player vs Bot              2 |" << endl;
	cout << "         |  Exit                       3 |" << endl;
	cout << "         |                               |" << endl;
	cout << "         ================================" << endl;
```

```
        cout << "         Please Enter:";
        string n;
        cin >> n;
        if (n == "1")return pvpMenu();
        else if (n == "2")return pvcMenu();
        else if (n == "3")return;
        else return mainMenu();
}
```

4. 人人对弈的第一界面

```
void pvpMenu()
{
        system("CLS");
        cout << endl;
        cout << endl;
        cout << "         ================================" << endl;
        cout << "         |        Player vs Player       |" << endl;
        cout << "         |===============================|" << endl;
        cout << "         |                               |" << endl;
        cout << "         |  New Game                  1 |" << endl;
        cout << "         |  Load Game                 2 |" << endl;
        cout << "         |  Return                    3 |" << endl;
        cout << "         |                               |" << endl;
        cout << "         ================================" << endl;
        cout << "         Please Enter: ";
        string n;
        cin >> n;
        if (n == "1")
        {
                resetGame();
                return pvpUI(1);
        }
        else if (n == "2")
        {
                loadPvp();
                return pvpUI(2);
        }
        else if (n == "3")return mainMenu();
        return pvpMenu();
}
```

5. 人机对弈的第一界面

```
void pvcMenu()
{
	system("CLS");
	cout << endl;
	cout << endl;
	cout << "         ================================" << endl;
	cout << "         |          Player vs Bot        |" << endl;
	cout << "         |===============================|" << endl;
	cout << "         |                               |" << endl;
	cout << "         |  New Game                  1 |" << endl;
	cout << "         |  Load Game                 2 |" << endl;
	cout << "         |  Return                    3 |" << endl;
	cout << "         |                               |" << endl;
	cout << "         ================================" << endl;
	cout << "         Please Enter: ";
	string n;
	cin >> n;
	if (n == "1")
	{
		resetGame();
		return pvcUI(1);
	}
	else if (n == "2")
	{
		loadPvc();
		return pvcUI(2);
	}
	else if (n == "3")return mainMenu();
	return pvcMenu();
}
```

6. 人人对弈的第二界面

```
void pvpUI(int a)
{
	if (a == 1)
	{
		system("cls");
		cout << "                  Player vs Player         " << endl;
		cout << "         ================================" << endl;
		cout << "                White: 1    Black: 2       " << endl;
		cout << "         ================================" << endl;
		cout << "         Which Side Go First: ";
		string n;
```

```
		cin >> n;
		if (n == "1")currBotColor = grid_white;
		else if (n == "2")currBotColor = grid_black;
		else
		{
			cout << "    Invalid Input!" << endl;
			system("pause");
			system("cls");
			return pvpUI(1);
		}
	}
	while (true)
	{
		string temp1, temp2;
		bool flag = true;
		int coor[2][3] = { 0 };
		system("cls");
		printTable();
		cout << "    Please Enter: ";
		for (int i = 0; i < 3; i++)
		{
			cin >> temp1;
			if (temp1 == "reset")
			{
				system("cls");
				cout << "    ==================================" << endl;
				cout << "           The game is resetted.      " << endl;
				cout << "    ==================================" << endl;
				system("pause");
				resetGame();
				flag = false;
				break;
			}
			else if (temp1 == "menu")return mainMenu();
			else if (temp1 == "save")
			{
				savedPvp();
				system("cls");
				cout << "    ==================================" << endl;
				cout << "             Saved Successfully.      " << endl;
				cout << "    ==================================" << endl;
				system("pause");
				return mainMenu();
			}
			else if (temp1 == "undo")
```

```
{
            undoLoad();
            system("cls");
            cout << "    ================================" << endl;
            cout << "             Undo Successfully.       " << endl;
            cout << "    ================================" << endl;
            system("pause");
            return pvpUI(2);
}
else if (temp1 == "auto")
{
            currBotColor *= -1;
            computer_move();
            currRound++;
            det++;
            scanPosition();
            if (check_win(black))
            {
                        system("cls");
                        printTable();
                        cout << "    ================================" << endl;
                        cout << "               White Wins!            " << endl;
                        cout << "    ================================" << endl;
                        system("pause");
                        break;
            }
            else if (check_win(white))
            {
                        system("cls");
                        printTable();
                        cout << "    ================================" << endl;
                        cout << "               Black Wins!            " << endl;
                        cout << "    ================================" << endl;
                        system("pause");
                        break;
            }
            return pvpUI(2);
}
cin >> temp2;
if (!isDigit(temp1) || !isDigit(temp2))
{
            cout << "    ================================" << endl;
            cout << "               Invalid Input!         " << endl;
            cout << "    ================================" << endl;
            flag = false;
```

```
                                        break;
                                }
                                coor[0][i] = temp1[0] - '0';
                                coor[1][i] = temp2[0] - '0';
                        }
                        if (!flag)continue;
                        if (isValid(coor[0][0], coor[1][0], coor[0][1], coor[1][1], coor[0][2], coor[1][2], currBotColor))
                        {
                                undoSave();
                                procStep(coor[0][0], coor[1][0], coor[0][1], coor[1][1], coor[0][2], coor[1][2],
currBotColor);
                                currBotColor = -1 * currBotColor;
                                currRound++;
                                scanPosition();
                                if (check_win(black))
                                {
                                        system("cls");
                                        printTable();
                                        cout << "    ==================================" << endl;
                                        cout << "               White Wins!            " << endl;
                                        cout << "    ==================================" << endl;
                                        break;
                                }
                                else if (check_win(white))
                                {
                                        system("cls");
                                        printTable();
                                        cout << "    ==================================" << endl;
                                        cout << "               Black Wins!            " << endl;
                                        cout << "    ==================================" << endl;
                                        break;
                                }
                        }
                        else
                        {
                                cout << "    ==================================" << endl;
                                cout << "               Invalid Move!          " << endl;
                                cout << "    ==================================" << endl;
                                system("pause");
                        }
                }
                system("pause");
                return mainMenu();
}
```

7. 人机对弈的第二界面

```
void pvcUI(int a)
{
	if (a == 1)
	{
		system("cls");
		cout << "                    Player vs Bot          " << endl;
		cout << "         ================================" << endl;
		cout << "                White: 1    Black: 2       " << endl;
		cout << "         ================================" << endl;
		cout << "         Please Select Your Side: ";
		string n;
		cin >> n;
		if (n == "1")currBotColor = grid_white;
		else if (n == "2")currBotColor = grid_black;
		else
		{
			cout << "    Invalid Input!" << endl;
			system("pause");
			return pvcUI(1);
		}
	}
	if (currBotColor == grid_black)det = 1;
	while (true)
	{
		if (det % 2 == 0)
		{
			system("cls");
			printTable();
			cout << "    The Computer moved: " << ansx1 << " " << ansy1 << " " << ansx2 << " " << ansy2
<< " " << ansx3 << " " << ansy3 << endl;
			string temp1, temp2;
			bool flag = true;
			int coor[2][3] = { 0 };
			cout << "    Please Enter: ";
			for (int i = 0; i < 3; i++)
			{
				cin >> temp1;
				if (temp1 == "reset")
				{
					system("cls");
					cout << "    ================================" << endl;
					cout << "           The game is resetted.      " << endl;
					cout << "    ================================" << endl;
```

```
            system("pause");
            resetGame();
            flag = false;
            break;
        }
        else if (temp1 == "menu")return mainMenu();
        else if (temp1 == "save")
        {
            savedPvc();
            system("cls");
            cout << "    ================================" << endl;
            cout << "             Saved Successfully.      " << endl;
            cout << "    ================================" << endl;
            system("pause");
            return mainMenu();
        }
        else if (temp1 == "auto")
        {
            currBotColor *= -1;
            computer_move();
            currRound++;
            det++;
            scanPosition();
            if (check_win(black))
            {
                system("cls");
                printTable();
                cout << "    ================================" << endl;
                cout << "               White Wins!            " << endl;
                cout << "    ================================" << endl;
                system("pause");
                break;
            }
            else if (check_win(white))
            {
                system("cls");
                printTable();
                cout << "    ================================" << endl;
                cout << "               Black Wins!            " << endl;
                cout << "    ================================" << endl;
                system("pause");
                break;
            }
            currBotColor *= -1;
            return pvcUI(2);
```

```
                }
                else if (temp1 == "undo")
                {
                        undoLoad();
                        system("cls");
                        cout << "    ================================" << endl;
                        cout << "             Undo Successfully.       " << endl;
                        cout << "    ================================" << endl;
                        system("pause");
                        return (pvcUI(2));
                }
                cin >> temp2;
                if (!isDigit(temp1) || !isDigit(temp2))
                {
                        cout << "    ================================" << endl;
                        cout << "               Invalid Input!         " << endl;
                        cout << "    ================================" << endl;
                        flag = false;
                        break;
                }
                coor[0][i] = temp1[0] - '0';
                coor[1][i] = temp2[0] - '0';
        }
        if (!flag)continue;
        if (isValid(coor[0][0], coor[1][0], coor[0][1], coor[1][1], coor[0][2], coor[1][2],
currBotColor))
        {
                undoSave();
                procStep(coor[0][0], coor[1][0], coor[0][1], coor[1][1], coor[0][2], coor[1][2],
currBotColor);
                det++;
                currRound++;
                scanPosition();
                if (check_win(black))
                {
                        system("cls");
                        printTable();
                        cout << "    ================================" << endl;
                        cout << "               White Wins!            " << endl;
                        cout << "    ================================" << endl;
                        system("pause");
                        break;
                }
                else if (check_win(white))
                {
```

```
                                        system("cls");
                                        printTable();
                                        cout << "    ================================" << endl;
                                        cout << "               Black Wins!            " << endl;
                                        cout << "    ================================" << endl;
                                        system("pause");
                                        break;
                                }
                        }
                        else
                        {
                                cout << "    ================================" << endl;
                                cout << "               Invalid Move!          " << endl;
                                cout << "    ================================" << endl;
                                system("pause");
                        }
                }
                else
                {
                        computer_move();
                        currRound++;
                        det++;
                        scanPosition();
                        if (check_win(black))
                        {
                                system("cls");
                                printTable();
                                cout << "    ================================" << endl;
                                cout << "               White Wins!            " << endl;
                                cout << "    ================================" << endl;
                                system("pause");
                                break;
                        }
                        else if (check_win(white))
                        {
                                system("cls");
                                printTable();
                                cout << "    ================================" << endl;
                                cout << "               Black Wins!            " << endl;
                                cout << "    ================================" << endl;
                                system("pause");
                                break;
                        }
                }
        }
```

```
        return mainMenu();
}
```

(b) 判断类

1. 判断是否在地图内

```
bool inMap(int x, int y)
{
        if (x < 0 || x >= GRIDSIZE || y < 0 || y >= GRIDSIZE)
                return false;
        return true;
}
```

2. 判断棋子与落点是否处于斜线

```
bool isLined(int x0, int y0, int x1, int y1)
{
        if (abs(x1 - x0) == abs(y1 - y0))return true;
        return false;
}
```

3. 判断棋子的落点是否合法

```
bool isWay(int x0, int y0, int x1, int y1, int x2, int y2, int color)
{
        if (color == OBSTACLE && x1 == x2 && y1 == y2)return true;
        if (x0 == x1 && y0 == y1)return false; //如果重叠
        if (gridInfo[y1][x1] != 0)return false; //如果已经被占领

        if (x0 == x1) //直线
        {
                if (y1 - y0 > 0)
                {
                        for (int i = y0 + 1; i <= y1; i++)
                        {
                                if (gridInfo[i][x0] != 0)return false;
                        }
                }
                else
                {
                        for (int i = y1; i < y0; i++)
                        {
                                if (gridInfo[i][x0] != 0)return false;
                        }
                }
        }
```

```
if (y0 == y1) //横线
{
	if (x1 - x0 > 0)
	{
		for (int i = x0 + 1; i <= x1; i++)
		{
			if (gridInfo[y0][i] != 0)return false;
		}
	}
	else
	{
		for (int i = x1; i < x0; i++)
		{
			if (gridInfo[y0][i] != 0)return false;
		}
	}
}
if (x1 != x0 && y1 != y0 && !isLined(x0, y0, x1, y1))return false;
int gap = abs(y1 - y0);
if (x1 - x0 > 0 && y1 - y0 > 0)//右下角
{
	for (int i = 1; i <= gap; i++)
	{
		if (gridInfo[y0 + i][x0 + i] != 0)return false;
	}
}
if (x1 - x0 > 0 && y1 - y0 < 0)//右上角
{
	for (int i = 1; i <= gap; i++)
	{
		if (gridInfo[y0 - i][x0 + i] != 0)return false;
	}
}
if (x1 - x0 < 0 && y1 - y0 > 0)//左下角
{
	for (int i = 1; i <= gap; i++)
	{
		if (gridInfo[y0 + i][x0 - i] != 0)return false;
	}
}
if (x1 - x0 < 0 && y1 - y0 < 0)//左上角
{
	for (int i = 1; i <= gap; i++)
	{
		if (gridInfo[y0 - i][x0 - i] != 0)return false;
```

```
                }
        }
        return true;
}
```

4. 检查步法是否合法

```
bool isValid(int x0, int y0, int x1, int y1, int x2, int y2, int color)
{
        if ((!inMap(x0, y0)) || (!inMap(x1, y1)) || (!inMap(x2, y2)))//任何一点不在期盼内的话，就返回false
                return false;
        if (gridInfo[y0][x0] != color || gridInfo[y1][x1] != 0)//自己的棋子如果不在选择的起点上或者要去的地方已经被占领，就返回
false
                return false;
        if ((gridInfo[y2][x2] != 0) && !(x2 == x0 && y2 == y0))//选择的障碍位置已经被占领，除非障碍位置是自己的起始位置
                return false;
        if (!(isWay(x0, y0, x1, y1, x2, y2, color) && isWay(x1, y1, x2, y2, x0, y0, OBSTACLE)))//如果是非法步法，返回false
                return false;
        return true;
}
```

5. 判断输入是否合法

```
bool isDigit(string a)
{
        if (a.length() == 1 && a[0] >= '0' && a[0] <= '9')return true;
        return false;
}
```

(c) 移动类

1. 在坐标处落子

```
void procStep(int x0, int y0, int x1, int y1, int x2, int y2, int color)
{
        gridInfo[y0][x0] = 0;
        gridInfo[y1][x1] = color;
        gridInfo[y2][x2] = OBSTACLE;
}
```

2. 获取黑白棋的位置

```
void scanPosition()
{
        int b1 = 0;
        int w1 = 0;
        for (int i = 0; i < GRIDSIZE; i++)
```

```
        {
                for (int j = 0; j < GRIDSIZE; j++)
                {
                        if (gridInfo[i][j] == grid_black)
                        {
                                black[b1][0] = i;
                                black[b1++][1] = j;
                        }
                        else if (gridInfo[i][j] == grid_white)
                        {
                                white[w1][0] = i;
                                white[w1++][1] = j;
                        }
                }
        }
}
```

3. 判断游戏结束

```
bool check_win(int coor[][2])
{
        for (int i = 0; i < 4; i++)
        {

                for (int j = 0; j < 8; j++)
                {
                        int y1 = coor[i][0] + dir[0][j];
                        int x1 = coor[i][1] + dir[1][j];
                        if (!(!inMap(x1, y1) || gridInfo[y1][x1] != 0))
                        {
                                return false;
                        }
                }
        }
        return true;
}
```

4. 电脑移动

```
void computer_move()
{
        if (currRound < 12)enddeep = 1;
        else if (currRound < 18)enddeep = 2;
        else enddeep = 3;
        mcPosition(-1 * currBotColor);
        dfs(0, MAX_VAL);
        procStep(ansx1, ansy1, ansx2, ansy2, ansx3, ansy3, -1 * currBotColor);
```

```
}
```

(d) 算法类

1. 重建现在的棋盘到 a 保持敌我坐标到 mx my cx cy

```
void mcPosition(int color)
{
	memcpy(a, gridInfo, sizeof(a));
	int wlink = 0, dlink = 0;
	if (color == grid_white)
	{
		for (int i = 0; i < 8; i++)
		{
			for (int j = 0; j < 8; j++)
			{
				if (a[i][j] == grid_white)
				{
					mx[wlink] = j;
					my[wlink] = i;
					wlink++;
				}
				if (a[i][j] == grid_black)
				{
					cx[dlink] = j;
					cy[dlink] = i;
					dlink++;
				}
			}
		}
	}
	else if (color == grid_black)
	{
		for (int i = 0; i < 8; i++)
		{
			for (int j = 0; j < 8; j++)
			{
				if (a[i][j] == grid_black)
				{
					mx[wlink] = j;
					my[wlink] = i;
					wlink++;
				}
				if (a[i][j] == grid_white)
```

```
                                        {
                                                cx[dlink] = j;
                                                cy[dlink] = i;
                                                dlink++;
                                        }
                                }
                        }
                }
        }
```

2. 计算

```
double val()
{
        for (int i = 0; i < GRIDSIZE; i++)
        {
                for (int j = 0; j < GRIDSIZE; j++)
                {
                        meKing[i][j] = comKing[i][j] = meQuen[i][j] = comQueen[i][j] = MAX_VAL;
                }
        }
        double t1 = 0, t2 = 0, p1 = 0, p2 = 0, m = 0;
        for (int z = 0; z < 4; z++)
        {
                memset(bj, 0, sizeof(bj));
                top = 0;
                end1 = 1;
                queuex[0] = mx[z];
                queuey[0] = my[z];
                qdeep[0] = 0;
                bj[my[z]][mx[z]] = 1;
                while (top != end1)
                {
                        int x = queuex[top];
                        int y = queuey[top];
                        int deep = qdeep[top];
                        meKing[y][x] = min(meKing[y][x], deep);
                        for (int i = 0; i < 8; i++)
                        {
                                int x1 = x + dir[1][i];
                                int y1 = y + dir[0][i];
                                if (!inMap(x1, y1)) continue;
                                if (bj[y1][x1]) continue;
                                if (a[y1][x1]) continue;
                                queuex[end1] = x1;
                                queuey[end1] = y1;
```

```
                                qdeep[end1] = deep + 1;
                                bj[y1][x1] = 1;
                                end1++;
                        }
                        top++;
                }
                meKing[my[z]][mx[z]] = MAX_VAL;
        }
        for (int z = 0; z < 4; z++)
        {
                memset(bj, 0, sizeof(bj));
                top = 0;
                end1 = 1;
                queuex[0] = cx[z];
                queuey[0] = cy[z];
                qdeep[0] = 0;
                bj[cy[z]][cx[z]] = 1;
                while (top != end1)
                {
                        int x = queuex[top];
                        int y = queuey[top];
                        int deep = qdeep[top];
                        comKing[y][x] = min(comKing[y][x], deep);
                        for (int i = 0; i < 8; i++)
                        {
                                int x1 = x + dir[1][i];
                                int y1 = y + dir[0][i];
                                if (!inMap(x1, y1)) continue;
                                if (bj[y1][x1]) continue;
                                if (a[y1][x1]) continue;
                                queuex[end1] = x1;
                                queuey[end1] = y1;
                                qdeep[end1] = deep + 1;
                                bj[y1][x1] = 1;
                                end1++;
                        }
                        top++;
                }
                comKing[cy[z]][cx[z]] = MAX_VAL;
        }
        for (int z = 0; z < 4; z++)
        {
                memset(bj, 0, sizeof(bj));
                top = 0;
                end1 = 1;
```

```
            queuex[0] = mx[z];
            queuey[0] = my[z];
            qdeep[0] = 0;
            bj[my[z]][mx[z]] = 1;
            while (top != end1)
            {
                        int x = queuex[top];
                        int y = queuey[top];
                        int deep = qdeep[top];
                        meQuen[y][x] = min(meQuen[y][x], deep);
                        for (int i = 0; i < 8; i++)
                        {
                                    for (int p = 1;; p++)
                                    {
                                                int x1 = x + p * dir[1][i];
                                                int y1 = y + p * dir[0][i];
                                                if (!inMap(y1, x1)) break;
                                                if (bj[y1][x1]) continue;
                                                if (a[y1][x1]) break;
                                                queuex[end1] = x1;
                                                queuey[end1] = y1;
                                                qdeep[end1] = deep + 1;
                                                bj[y1][x1] = 1;
                                                end1++;
                                    }
                        }
                        top++;
            }
            meQuen[my[z]][mx[z]] = MAX_VAL;
}
for (int z = 0; z < 4; z++)
{
            memset(bj, 0, sizeof(bj));
            top = 0;
            end1 = 1;
            queuex[0] = cx[z];
            queuey[0] = cy[z];
            qdeep[0] = 0;
            bj[cy[z]][cx[z]] = 1;
            while (top != end1)
            {
                        int x = queuex[top];
                        int y = queuey[top];
                        int deep = qdeep[top];
                        comQueen[y][x] = min(comQueen[y][x], deep);
```

```
                        for (int i = 0; i < 8; i++)
                        {
                                for (int p = 1;; p++)
                                {
                                        int x1 = x + p * dir[1][i];
                                        int y1 = y + p * dir[0][i];
                                        if (!inMap(x1, y1)) break;
                                        if (bj[y1][x1]) continue;
                                        if (a[y1][x1]) break;
                                        queuex[end1] = x1;
                                        queuey[end1] = y1;
                                        qdeep[end1] = deep + 1;
                                        bj[y1][x1] = 1;
                                        end1++;
                                }
                        }
                        top++;
                }
                comQueen[cy[z]][cx[z]] = MAX_VAL;
        }
        for (int i = 0; i < 8; i++)
        {
                for (int j = 0; j < 8; j++)
                {
                        if (meQuen[i][j] > 1)continue;
                        mm = 0;
                        for (int p = 0; p < 8; p++)
                        {
                                if (!inMap(i + dir[0][p], j + dir[1][p])) continue;
                                if (a[i + dir[0][p]][j + dir[1][p]]) continue;
                                mm++;
                        }
                        m += mm / meKing[i][j];
                }
        }
        for (int i = 0; i < 8; i++)
        {
                for (int j = 0; j < 8; j++)
                {
                        if (meKing[i][j] > comKing[i][j]) t1 += -1;
                        else if (meKing[i][j] < comKing[i][j]) t1 += 1;
                        else if (meKing[i][j] == MAX_VAL) t1 += 0;
                        else t1 += 0.17;

                        if (meQuen[i][j] > comQueen[i][j]) t2 += -1;
```

```
                                else if (meQuen[i][j] < comQueen[i][j]) t2 += 1;
                                else if (meQuen[i][j] == MAX_VAL) t2 += 0;
                                else t2 += 0.17;

                                p1 += 2.0 * (pow(2.0, -1 * meQuen[i][j]) - pow(2.0, -1 * comQueen[i][j]));

                                p2 += min(1.0, max(-1.0, (comKing[i][j] - meKing[i][j]) / 6.0));
                        }
                }
                double vv;
                if (n <= 7) vv = 0.24 * t1 + 0.47 * t2 + 0.13 * p1 + 0.13 * p2 + 0.20 * m;
                else if (n <= 16) vv = 0.30 * t1 + 0.25 * t2 + 0.30 * p1 + 0.30 * p2 + 0.05 * m;
                else vv = 0.8 * t1 + 0.1 * t2 + 0.1 * p1 + 0.1 * p2;
                return vv;
}
```

3. 深度优先搜索+αβ剪枝

```
double dfs(int deep, double lastmaxmin)
{
        if (deep == enddeep) return val();
        double maxmin;
        ////////////////////////////////////////////////////如果我方先行棋
        if (deep % 2 == 0)
        {
                maxmin = MIN_VAL;
                int x1, y1, x2, y2, x3, y3;
                for (int z = 0; z <= 3; z++)
                {
                        x1 = mx[z];
                        y1 = my[z];
                        for (int p = 0; p < 8; p++)
                        {
                                for (int i = 1;; i++)
                                {
                                        x2 = x1 + i * dir[1][p];
                                        y2 = y1 + i * dir[0][p];
                                        if (!inMap(x2, y2)) break;
                                        if (a[y2][x2]) break;

                                        mx[z] = x2;
                                        my[z] = y2;
                                        a[y1][x1] = 0;
                                        a[y2][x2] = 1;
                                        ////////////////////////////////////释放障碍
                                        for (int p = 0; p < 8; p++)
```

```
                                        {
                                                for (int i = 1;; i++)
                                                {
                                                        x3 = x2 + i * dir[1][p];
                                                        y3 = y2 + i * dir[0][p];
                                                        if (!inMap(x3, y3)) break;
                                                        if (a[y3][x3]) break;

                                                        a[y3][x3] = OBSTACLE;
                                                        double newval = dfs(deep + 1, maxmin);
                                                        a[y3][x3] = 0;
                                                        if (deep != 0) maxmin = max(newval, maxmin);
                                                        else if (newval > maxmin)
                                                        {
                                                                maxmin = newval;
                                                                ansx1 = x1;
                                                                ansx2 = x2;
                                                                ansx3 = x3;
                                                                ansy1 = y1;
                                                                ansy2 = y2;
                                                                ansy3 = y3;
                                                        }
                                                        if (maxmin >= lastmaxmin)
                                                        {
                                                                a[y1][x1] = 1;
                                                                a[y2][x2] = 0;
                                                                mx[z] = x1;
                                                                my[z] = y1;
                                                                return maxmin;
                                                        }
                                                }
                                        }
                                        ////////////////////////////////释放障碍结束
                                        a[y1][x1] = 1;
                                        a[y2][x2] = 0;
                                        mx[z] = x1;
                                        my[z] = y1;
                                }
                        }
                }
                if (maxmin == MIN_VAL) return val();
        }
        ////////////////////////////////////////////////如果敌方行棋
        else
        {
```

```
maxmin = MAX_VAL;
int x1, y1, x2, y2, x3, y3;
for (int z = 0; z <= 3; z++)
{
		x1 = cx[z];
		y1 = cy[z];
		for (int p = 0; p < 8; p++)
		{
			for (int i = 1;; i++)
			{
				x2 = x1 + i * dir[0][p];
				y2 = y1 + i * dir[1][p];
				if (!inMap(x2, y2)) break;
				if (a[y2][x2]) break;

				cx[z] = x2;
				cy[z] = y2;
				a[y1][x1] = 0;
				a[y2][x2] = grid_black;
				//////////////////////////////////释放障碍
				for (int p = 0; p < 8; p++)
				{
					for (int i = 1;; i++)
					{
						x3 = x2 + i * dir[1][p];
						y3 = y2 + i * dir[0][p];
						if (!inMap(x3, y3)) break;
						if (a[y3][x3]) break;

						a[y3][x3] = OBSTACLE;
						double newval = dfs(deep + 1, maxmin);
						a[y3][x3] = 0;
						maxmin = min(newval, maxmin);
						if (maxmin <= lastmaxmin)
						{
							a[y1][x1] = grid_black;
							a[y2][x2] = 0;
							cx[z] = x1;
							cy[z] = y1;
							return maxmin;
						}
					}
				}
				//////////////////////////////////释放障碍结束
				a[y1][x1] = grid_black;
```

```
                                        a[y2][x2] = 0;
                                        cx[z] = x1;
                                        cy[z] = y1;
                                }
                        }
                }
                if (maxmin == MAX_VAL) return val();
        }
        return maxmin;
}
```

(e) 存储

1. 人机对弈的存盘

```
void savedPvc()
{
        fstream outfile("pvcmap.txt");
        for (int i = 0; i < GRIDSIZE; i++)
        {
                for (int j = 0; j < GRIDSIZE; j++)
                {
                        outfile << gridInfo[i][j] << " ";
                }
                outfile << endl;
        }
        outfile << currBotColor << endl;
        outfile << currRound << endl;
        outfile << det << endl;
        outfile << ansx1 << " " << ansy1 << " " << ansx2 << " " << ansy2 << " " << ansx3 << " " << ansy3 << endl;
        outfile.close();
        }
```

2. 人人对弈的存盘

```
void savedPvp()
{
        fstream outfile("pvpmap.txt");
        for (int i = 0; i < GRIDSIZE; i++)
        {
                for (int j = 0; j < GRIDSIZE; j++)
                {
                        outfile << gridInfo[i][j] << " ";
                }
                outfile << endl;
```

```
        }
        outfile << currBotColor << endl;
        outfile << currRound << endl;
        outfile.close();
}
```

3. 人机对弈的读盘

```
void loadPvc()
{
        fstream infile("pvcmap.txt");
        for (int i = 0; i < GRIDSIZE; i++)
        {
                for (int j = 0; j < GRIDSIZE; j++)
                {
                        infile >> gridInfo[i][j];
                }
        }
        infile >> currBotColor;
        infile >> currRound;
        infile >> det;
        infile >> ansx1 >> ansy1 >> ansx2 >> ansy2 >> ansx3 >> ansy3;
        infile.close();
}
```

4. 人人对弈的读盘

```
void loadPvp()
{
        fstream infile("pvpmap.txt", ios::in);
        for (int i = 0; i < GRIDSIZE; i++)
        {
                for (int j = 0; j < GRIDSIZE; j++)
                {
                        infile >> gridInfo[i][j];
                }
        }
        infile >> currBotColor;
        infile >> currRound;
        infile.close();
}
```

5. 取消存盘

```
void undoSave()
{
        fstream outfile("undo.txt");
        for (int i = 0; i < GRIDSIZE; i++)
```

```
	{
		for (int j = 0; j < GRIDSIZE; j++)
		{
			outfile << gridInfo[i][j] << " ";
		}
		outfile << endl;
	}
	outfile << currBotColor << endl;
	outfile << currRound << endl;
	outfile << det << endl;
	outfile << ansx1 << " " << ansy1 << " " << ansx2 << " " << ansy2 << " " << ansx3 << " " << ansy3 << endl;
	outfile.close();
}
```

6. 取消读盘

```
void undoLoad()
{
	fstream infile("undo.txt", ios::in);
	for (int i = 0; i < GRIDSIZE; i++)
	{
		for (int j = 0; j < GRIDSIZE; j++)
		{
			infile >> gridInfo[i][j];
		}
	}
	infile >> currBotColor;
	infile >> currRound;
	infile >> det;
	infile >> ansx1 >> ansy1 >> ansx2 >> ansy2 >> ansx3 >> ansy3;
	infile.close();
}
```

（f）Main 函数

```
int main()
{
	mainMenu();
}
```

在完成作业时遇到的困难和收获

在界面方面，我们有分为人机对弈和人人对弈。起初我们最先设计出的棋盘内容有点过于单调，只有简单的棋盘和把坐标轴标记出来。而后考虑到美观的问题，我们又重新设计了棋盘的界面，同时也加了菜单页。

在判断类，刚开始要写能检查走法是否正确的函数时，刚开始写得过于长和啰嗦，随后在思考和不断地改进后，终于写出了我们认为还算满意的判断函数，简洁有力。

存储类的函数对我们来说是一个很陌生的方面，我们之前并没有学习过有关这一方面的知识。但是现在我们能写出三大类有关这一方面的函数（存盘、读盘和复盘），我们都感到很高兴，因为又学习到了一个新的东西。

算法部分对我们两个刚学计概不久的人来说一切都很新颖。我们不断地上网找有关的资料和论文，以求更加了解亚马逊棋的思路。在参考了几篇有关的文献和寻求朋友的帮助后，我们简略的完成了算法的部分。过后发现随着游戏持续在进行、电脑搜索的深度越来越深，电脑回复的时间会慢慢变慢。因此这点是我们仍需要改进的地方。

在通过写大作业的这段期间，我们真的学到了很多，这和一般我们完成的作业完全是截然不同的感受。在完成大作业的当儿，我们深感到自己现在的能力有限，但同时也很期待未来在信科会学习到的东西。

参考文献

1. 郭琴琴，李淑琴，包华，亚马逊棋机器博弈系统中评估函数的研究，北京大学，2012.