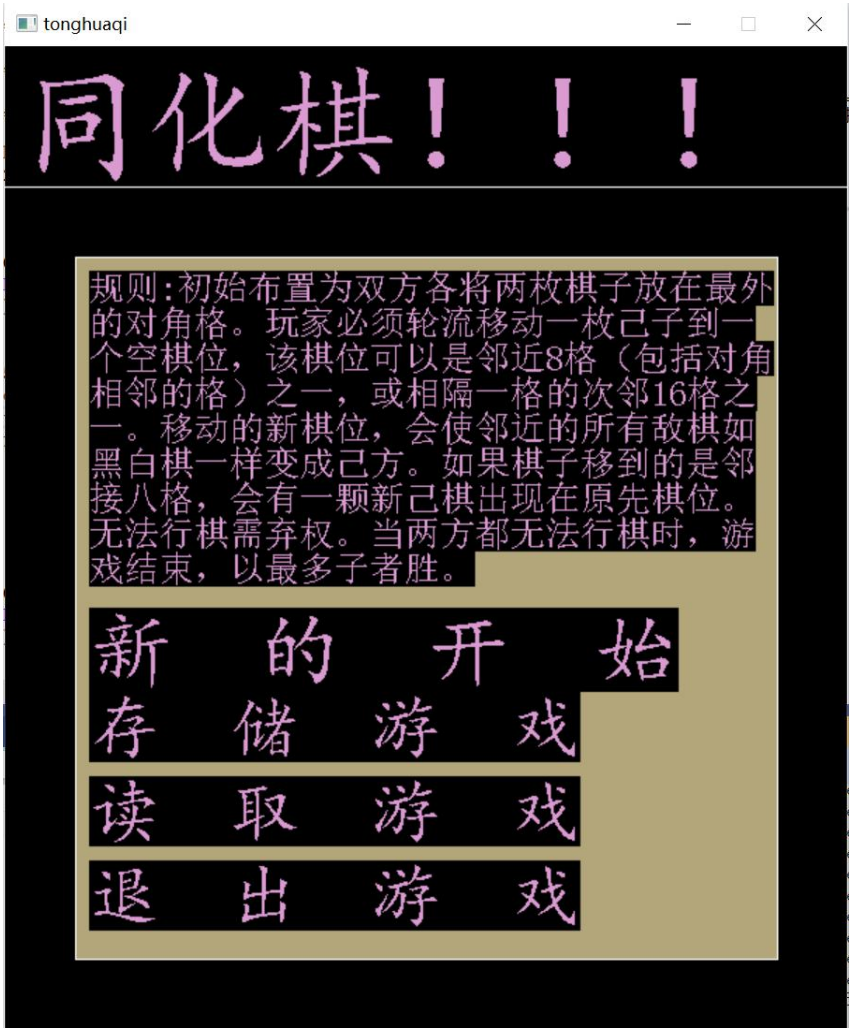
同化棋 ATAXX 实验报告

信息科学技术学院 张钧天

一、游戏规则

初始布置为双方各将两枚棋子放在最外的对角格。玩家必须轮流移动一枚己子到一个空棋位，该棋位可以是邻近 8 格（包括对角相邻的格）之一，或相隔一格的次邻 16 格之一。移动的新棋位，会使邻近的所有敌棋如黑白棋一样变成己方。如果棋子移到的是邻接八格，会有一颗新己棋出现在原先棋位。无法行棋需弃权。当两方都无法行棋时，游戏结束。以最多子者胜。（如图一界面上方所示）



图一（界面颜色完全随机）

二、实验说明

编写一个同化棋程序，能够实现人机对战。有菜单选择（选择，新开始，存盘，读盘，结束）用字符实现画棋盘和棋子。一方选手是用户，另一方选手是计算机。用户输入要落子的坐标（x，y），根据规则现实翻转后的布局。计算机作为另一个选手计算判断下一步棋子所放的位置，并根据同化棋规则，显示新的棋盘布局。允许中途停止游戏。有复盘的功能（玩到一半，存储棋盘的状态）。（如图一下方各大主要功能所示）

三、实验平台及工具

Windows 10 64 位系统， Visual Studio 2019 社区版，EasyX（Visual C++2019 配套版本）。编程语言：C++。

图二（Visual Studio 2019）

图三（EaxyX）

四、算法部分实现

（1）简单模式

1.遍历

对每一个棋子遍历其周围位置，若不合法则舍去，若合法则进入下一步。

2.估值

评估函数的标准为下子之后能够同化对方棋子的最大个数，将其记录。

3.优化

根据规则，若落子在原来棋子周围，则相当于多了一个棋子，更有利。因此在对每一次估值时，若遇到估值相同，则优先进行第一类落子。

4.落子

修改棋盘内容并判断是否游戏已经结束。

```
void ai() {
    int max = -2, x1,y1,xx, yy, flag = 0;
    for (int i = 0; i <= 6; i++) {
        for (int j = 0; j <= 6; j++) {
            if (board[i][j] == '#') {
                for (int ii = -2; ii <= 2; ii++) {
                    for (int jj = -2; jj <= 2; jj++) {
                        if (ii == 0 && jj == 0)continue;
                        else switch (check(i,j, i+ii, j+jj,'#')) {
                        case 1: {
                            if (max <= eval(i + ii, j + jj, '#', '@')) {
                                max = eval(i + ii, j + jj, '#', '@');
                                x1 = i;
                                y1 = j;
                                xx = i + ii;
                                yy = j + jj;
                                flag = 1;
                            }
                            break;
                        }
                        case 2: {
                            if (max < eval(i + ii, j + jj, '#', '@')) {
                                max = eval(i + ii, j + jj, '#', '@');
                                x1 = i;
                                y1 = j;
                                xx = i + ii;
                                yy = j + jj;
                                flag = 2;
                            }
                            break;
                        }
                        }
                    }
                }
            }
        }
    }
    if (flag == 0)return;
    if (flag == 1)aimove1(x1, y1, xx, yy);
    if (flag == 2)aimove2(x1, y1, xx, yy);
}
```

图四（AI-简单模式主体代码）

（2）普通模式

1.拷贝

由于需要进行深度为 3 层的搜索，因此首先拷贝一个当前状态下的棋盘。

2.遍历

同简单模式，对每一个棋子遍历其周围位置，若不合法则舍去，若合法则进入下一步。

3.估值

有两个估值函数。第一个函数同简单模式，标准为下子之后能够同化对方棋子的最大个数，将其记录。第二个函数统计场面下我方棋子与对方棋子个数的差值。

4.尝试

假设当前 ai 如此落子，进行下一步判断。

5.模拟

调用简单模式的 ai，模拟对方落子。

6.评估

首先，第一类落子的评估标准是我方棋子与对方棋子个数的差值，第二类落子的评估标准是下子之后能够同化对方棋子的最大个数。其次，根据规则，若落子在原来棋子周围，则相当于多了一个棋子，更有利。因此若第一类落子有合法情况，则不再考虑第二类落子。

7.落子

修改棋盘内容并判断是否游戏已经结束。

```
void newai() {
    int max1 = -100, max2 = -100, flag = 0;
    for (int i = 0; i <= 6; i++) {
        for (int j = 0; j <= 6; j++) {
            if (board[i][j] == '#') {
                copyboard();
                for (int ii = -2; ii <= 2; ii++) {
                    for (int jj = -2; jj <= 2; jj++) {
                        copyboard();
                        if (ii == 0 && jj == 0)continue;
                        else switch (newcheck(i, j, i + ii, j + jj, '#')) {
                        case 1: {
                            copyboard();
                            newaimove1(i, j, i + ii, j + jj);
                            moni();
                            if (max1 < eval2('#', '@') * 100 + 10000) {
                                max1 = eval2('#', '@') * 100 + 10000;
                                xx1 = i;
                                yy1 = j;
                                xx = i + ii;
                                yy = j + jj;
                                flag = 1;
                            }
                            break;
                        }
                        case 2: {
                            copyboard();
                            newaimove2(i, j, i + ii, j + jj);
                            moni();
                            if (max2 < eval(i + ii, j + jj, '#', '@') * 100 && max1 < 0) {
                                max2 = eval(i + ii, j + jj, '#', '@') * 100;
                                xx1 = i;
                                yy1 = j;
                                xx = i + ii;
                                yy = j + jj;
                                flag = 2;
                            }
                            break;
                        }
                        }
                    }
                }
            }
        }
    }
    if (flag == 0)return;
    if (flag == 1) {
        aimove1(xx1, yy1, xx, yy);
    }
    if (flag == 2) {
        aimove2(xx1, yy1, xx, yy);
    }
}
```

图五（AI-普通模式主体代码）

五、UI 及交互部分实现

（1）界面

1.使用 EasyX，graphics.h 图形库绘制游戏的界面（见上图一）（使用 Initgraph、setbkcolor、cleardevice、settextstyle、outtextxy、line 等函数）

开屏具有文字缓慢变大的动态特效（图六）

```
char str[] = { "同化棋！！！" };
srand((unsigned int)time(NULL));
int x = 200;
while (x <= 9000) {
    settextcolor(RGB(rand() % 256, rand() % 256, rand() % 256));
    settextstyle(x / 100, 0, "楷体");
    outtextxy(10, 10, str);
    x++;
}
```

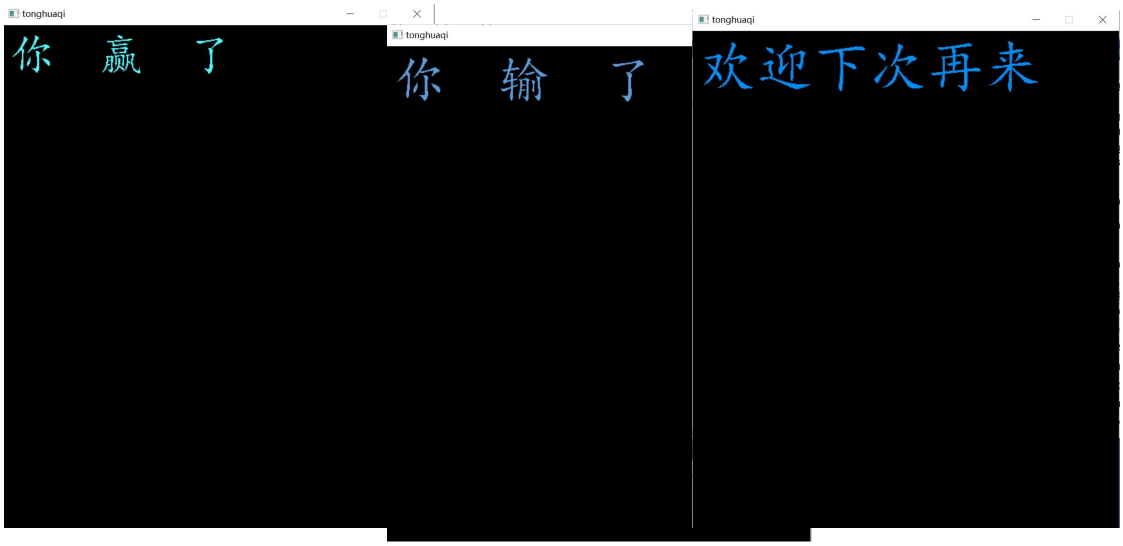
图六

以及前景背景色的完全随机呈现（图七）；

```
rectangle(50, 150, 550, 650);
setfillcolor(RGB(rand() % 256, rand() % 256, rand() % 256));
fillrectangle(50, 150, 550, 650);
setfillcolor(RGB(rand() % 256, rand() % 256, rand() % 256));
```

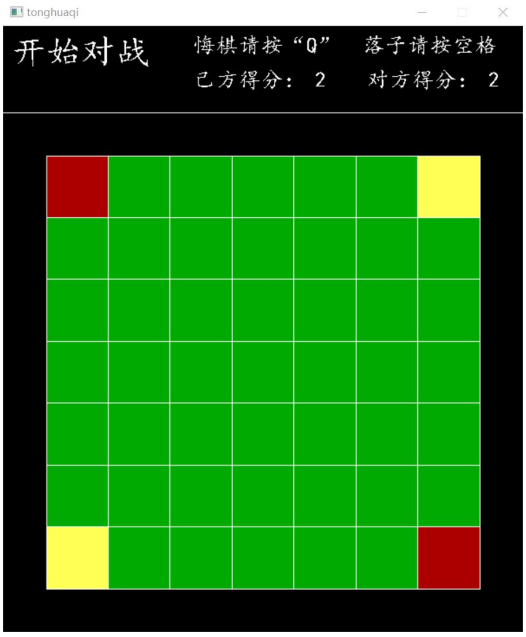
图七

退出、胜利、失败情况分别有相应的动画（图八、图九、图十）；



图八、图九、图十（胜利、失败、退出）

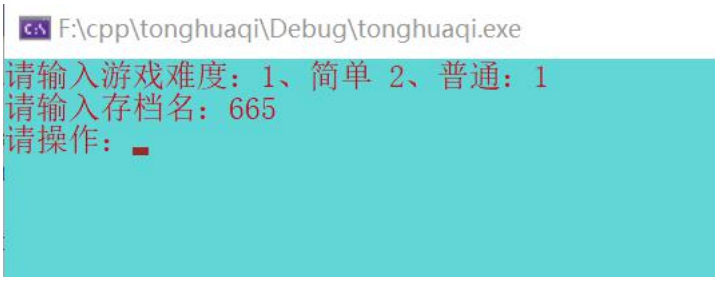
游戏进行界面，绿色表示棋盘，红黄两色表示我方与对方的棋子。（图十一）



图十一（游戏界面）

2.使用 color 函数修改控制台界面的前景色与背景色为红色和淡浅绿色，改善交互体验。（图十二）

```
system("color B4");
```

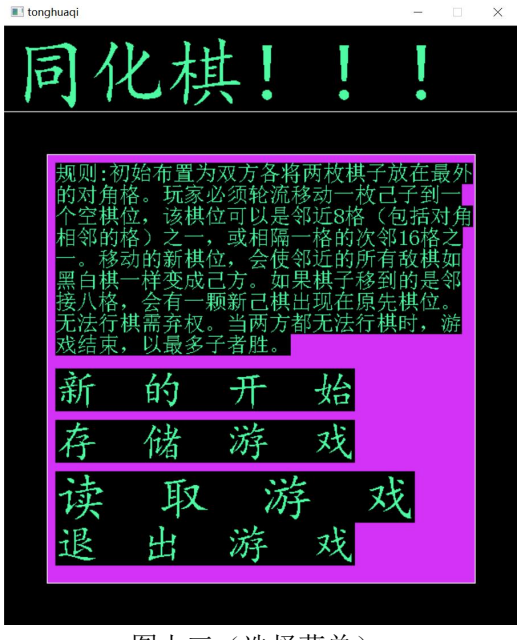


图十二（控制台界面）

3.使用 winmm.lib 库，在运行程序时输出一声音效，表示进入游戏。

```
（PlaySound(TEXT("D:\\laser.wav"), NULL, SND_FILENAME | SND_ASYNC);）
```

4.运用键盘的上下键控制想要进入的模式，按空格进入。（图十三）

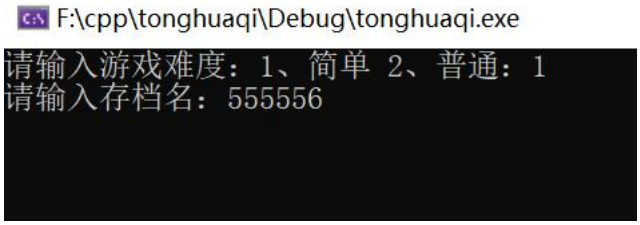


图十三（选择菜单）

（2）文件

1.在开始时创建新的存档，若已有则直接覆盖。（图十四）

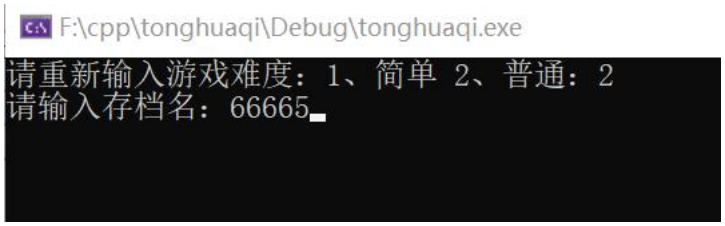
```
cout << "请输入存档名：";
cin >> sname;
ofstream Outfile;
Outfile.open(sname+".txt");
Outfile.close();
```



图十四（创建存档）

2.实现读取原有存档的功能。（图十五）

```
cout << "请输入存档名：";
cin >> readname;
ifstream fileread(readname + ".txt");
if (!fileread) {
    cout << "文件不存在，请重试。" << endl;
    continue;
}
else {
    sname = readname;
    for (int i = 0; i <= 6; i++) fileread >> board[i];
    break;
}
fileread.close();
```

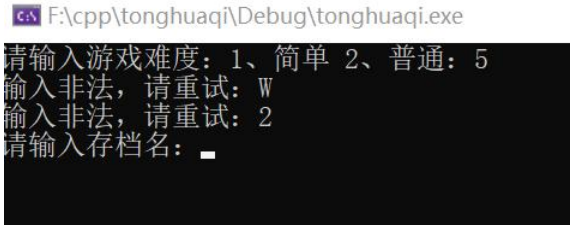


图十五（读取存档）

3.在游戏中实时保存上一步的棋盘。

（3）游戏

1.支持选择两种不同难度的模式。（图十六）



图十六（两种难度）

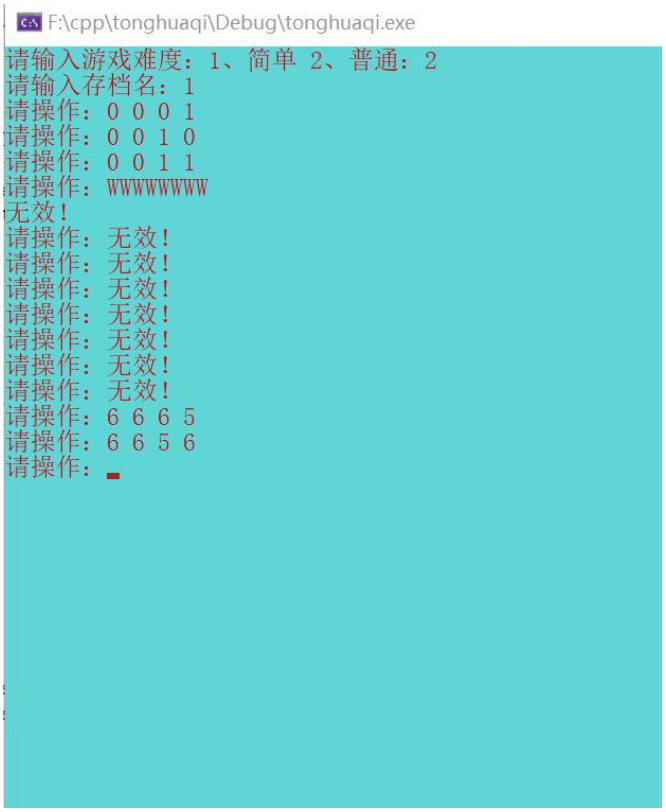
2.界面实时统计我方与对方的得分。（图十七）

图十七（双方得分）

3.支持按“Q”悔棋，按“Esc”退出，按“空格”落子。（图十八）

图十八（附加功能）

4.使用键盘分别输入棋子之前与之后的横纵坐标来进行操作。（图十九）



图十九（落子操作）

六、参考文献与博客

1.https://blog.csdn.net/sandalphon4869/article/details/80862023

2.https://blog.csdn.net/qq_46527915/article/details/104790010

3.https://github.com/Captain32/Ataxx