miniAlphaGo for Reversi

1. Project Introduction

（1） 开发环境：Windows 10

开发工具：Visual Studio 2017

开源库：EasyX 图形库

（2） 工作分配

邓墨琳：MCTS 算法分析，游戏流程分析，UCT 算法实现，游戏及图形界面实现；

张超波：报告撰写。

2. Technical Details

（1） UCT（信心上限树算法）

UCT 算法是蒙特卡洛方法的一种改进，比蒙特卡洛方法更容易得到最优解，其基本结构和蒙特卡洛方法相同，主要分为四个步骤：选择 (Selection)，拓展 (Expansion)，模拟 (Simulation) 和反向传播 (Backpropagation)。

$$\left(\frac{Q(v')}{N(v')}+C\sqrt{\frac{2\ln(N(v))}{N(v')}}\right) \quad ①$$

与蒙特卡洛方法不同的地方在于，在选择这个环节中，UCT 评价节点的优劣利用的是信心上限的估计值，估计信心上限值的公式如式①所示，c 是比例系数，控制后一项在整体估计中的重要程度；Q 表示收益（value），N 表示被访问次数，两者的比值表示胜率。在最终选择最佳落子位置时，仅通过①式中的前一项即胜率进行判断。

信心上限树中的每个节点都包含当前的棋局与玩家的情况、该节点被访问次数、节点的总 value 以及基本的子节点父节点。

（2） UCT（信心上限树算法）的实现

从根节点开始选择一个子节点进行下一步的迭代，而这样的子节点共有三种情况：

1、 该节点已经拓展完全，比如当前节点各个方向上都已经进行探索

了，那么就是拓展完全；

2、 该节点未拓展完全，比如当前节点的某一方向上还未进行探索，那么就是未拓展完全；

3、 该节点游戏已经结束，比如在翻转棋中，双方都没有可以位置可以落子了，就是游戏结束。

针对以上三种情况，我们在程序中判断了不同的情况并进行不同的操作：

1、 如果当前节点已经拓展完全并且游戏未结束，则反复选择（通过信心上限的估计值）最佳的节点进行迭代，直到找到未拓展完全的节点或游戏已结束的节点；

2、 如果当前节点未拓展完全并且游戏未结束，则反复拓展直到拓展完全或游戏结束；

3、 如果当前节点游戏已经结束，则直接进行模拟

经过以上步骤之后，我们将选出来的节点先进行筛选，如果节点游戏已经结束，则直接跳过模拟步骤进行反向传播，如果节点游戏未结束则模拟下棋，也就是创建一个模拟的对局，让这个对局进行到结束，模拟过程中首先利用当前棋局状态创建一个临时棋局，之后双方交替下棋，过程中双方的落子位置都是从他们的可落子集合中随机选取的，得到最后的反馈结果（delta），若 AI 赢则为 1，反之为 0；

之后反向传播，也就是将模拟得到的结果倒着加回到这次模拟涉及到的节点，并且更新其访问次数。

之后就一直重复以上四个步骤，直到时间达到上限或迭代次数达到上限停止循环，通过胜率（节点胜率/节点被访问次数）选取出最佳落子位置的节点。

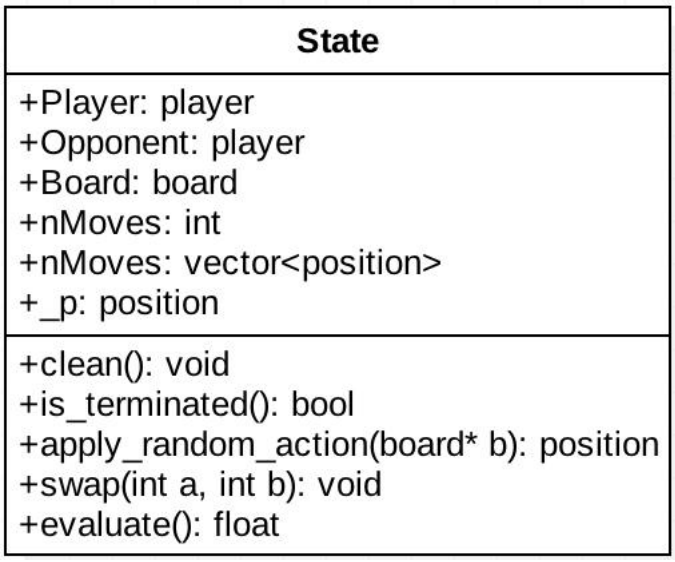
伪代码入下：

```
position run() {
创建根节点;
初始化各种参数;
while (true) {
    //选择Selection
    while (已经扩展完全且游戏未结束)
        选择最佳子节点;

    //扩展Expansion
    if (未扩展完全且游戏未结束)
```

```
            拓展子节点(之后对当前扩展的子节点进行模拟);

        //模拟Simulation
        if (当前节点游戏未结束) {
            创建模拟游戏;
            while (棋盘上空位置数量不为0) {
                if (黑色棋没有位置可以落子&&白色棋没有位置可以落子)
                    break;
                确定当前下棋方;
                得到当前可落子位置;
                if (可落子集合不为空) {
                    随机选择一个可落子围住落子;
                }
                else 将当前角色标记为无位置可落子;
                交换落子权;
            }
        }

        //反向传播Back propagation
        返回输赢，若AI赢则为1，反之为0;
        while (节点不为空) {
            更新节点的value和访问数;
            节点更新为其父节点;
        }

        选择当前棋局下的最佳子结点;
        if (搜索时间到达上限或者迭代次数到达上限) break;
        迭代次数++;
    }

    return 最佳位置;
}
```

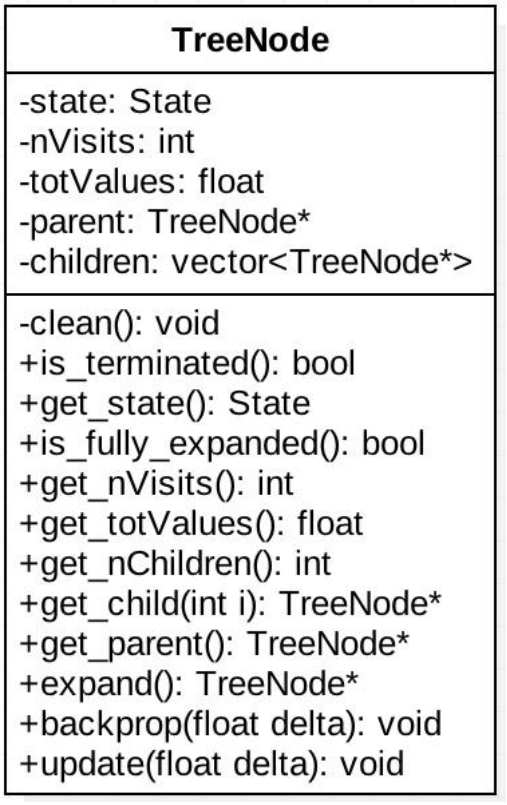
（3） 开发中涉及的重要类

1、State 类



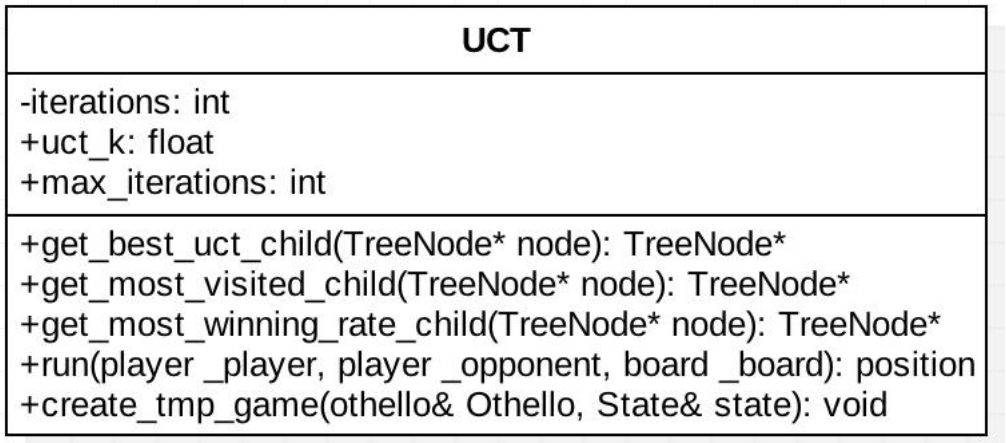
State 类是表示当前棋局状态的类，其中包含玩家属性、棋盘当前情况、当前情况上一步的落子位置，下一步的落子位置集合以及落子位置的个数；操作包含判断棋局是否已结束、模拟结束后返回 value 值等。

2、TreeNode 类



TreeNode 类是 UCT 中用到一个重要的节点类，其中包含了当前棋局的状态、节点被访问的次数、节点的总获利、节点的子节点集合和父节点；类的方法主要用到 expend()和 update(float delta)，分别用于扩展子节点和模拟游戏后更新节点属性。

3、UCT 类



UCT 类，信心上限树算法包含在其中，包括已迭代次数，最大迭代次数属性，主要用到的方法有通过信心上限值估计值选取最佳子节点方法 get_best_uct_child()、选取最高胜率子节点方法 get_most_winning_rate_child()以及 UCT 算法的主控方法 run()。

开发中用到的重要函数：

```
void SetWindows();
```

设置窗口的函数，主要功能为加载图片、初始化棋盘、设置窗口大小位置等，用到了 EasyX 图形库中的 loadimage()，putimage()等函数加载图片并组建成棋盘的图形界面，还调用了 windows 自己的窗口函数；

```
TreeNode* get_best_uct_child(TreeNode* node);
```

在选择环节用于选取最佳的子节点，利用每个节点的访问次数和 value 值通过估计信心上限值的公式计算通过比较最终得到最佳子节点；

```
TreeNode* get_most_winning_rate_child(TreeNode* node);
```

选取最佳落子位置，在模拟结束之后，此函数负责根据所有子节点的胜率选择出最佳的落子位置对应的子节点；

```
bool UCT_GamePlay();
```

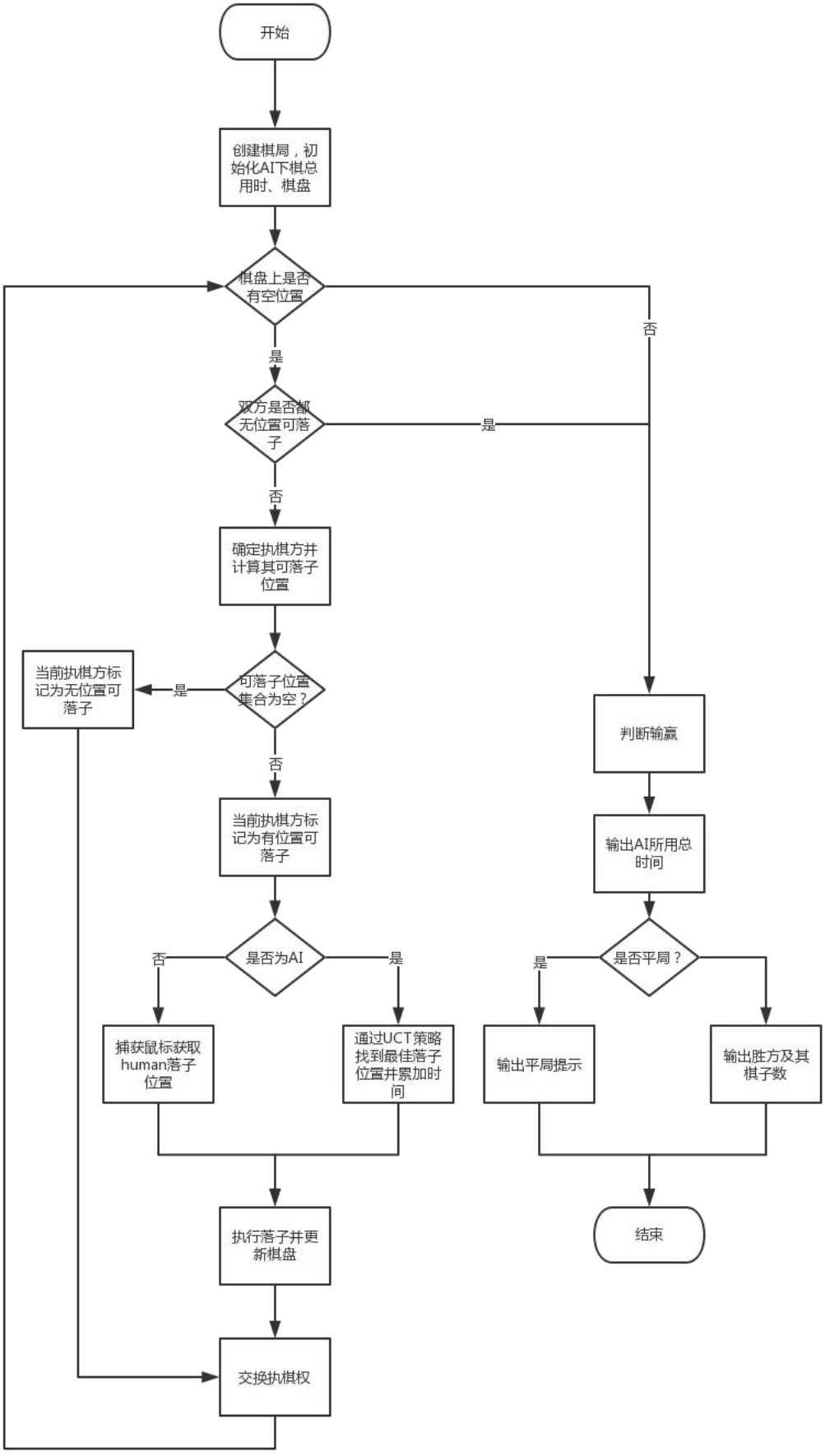
程序的主控函数，负责创建棋局初始化各种变量，之后执行循环，让 AI 和 human 依次落子，直到棋盘上没有空位置了或者双方都无法落子时停止循

环，之后计算双方棋子数并且判断输赢；

```
position run(player _player, player _opponent, board _board);
```

程序中的信心上限树搜索函数，其中包含了选择、扩展、模拟和反向传播四个步骤，最终返回最佳落子位置；

（4） 游戏流程图


开始
创建棋局，初始化AI下棋总用时、棋盘
棋盘上是否有空位置
否
是
双方是否都无位置可落子
是
否
确定执棋方并计算其可落子位置
当前执棋方标记为无位置可落子
是
可落子位置集合为空？
否
当前执棋方标记为有位置可落子
是否为AI
否
是
捕获鼠标获取human落子位置
通过UCT策略找到最佳落子位置并累加时间
执行落子并更新棋盘
交换执棋权
判断输赢
输出AI所用总时间
是否平局？
是
输出平局提示
输出胜方及其棋子数
结束

3. Experiment Results

运行程序，程序创建出棋盘并初始化为双方都有两枚棋子，AI 和 human 的先后手可以通过创建棋局时调用不同的函数来决定，相关函数如下：

```
CreateNewGame_AIfist()//创建 AI 先手棋局
CreateNewGame_Humanfist()//创建 human 先手棋局
```

我们测试时选择的是 AI 先手，AI 执黑棋，human 执白棋，创建出的棋局如图 1 所示：

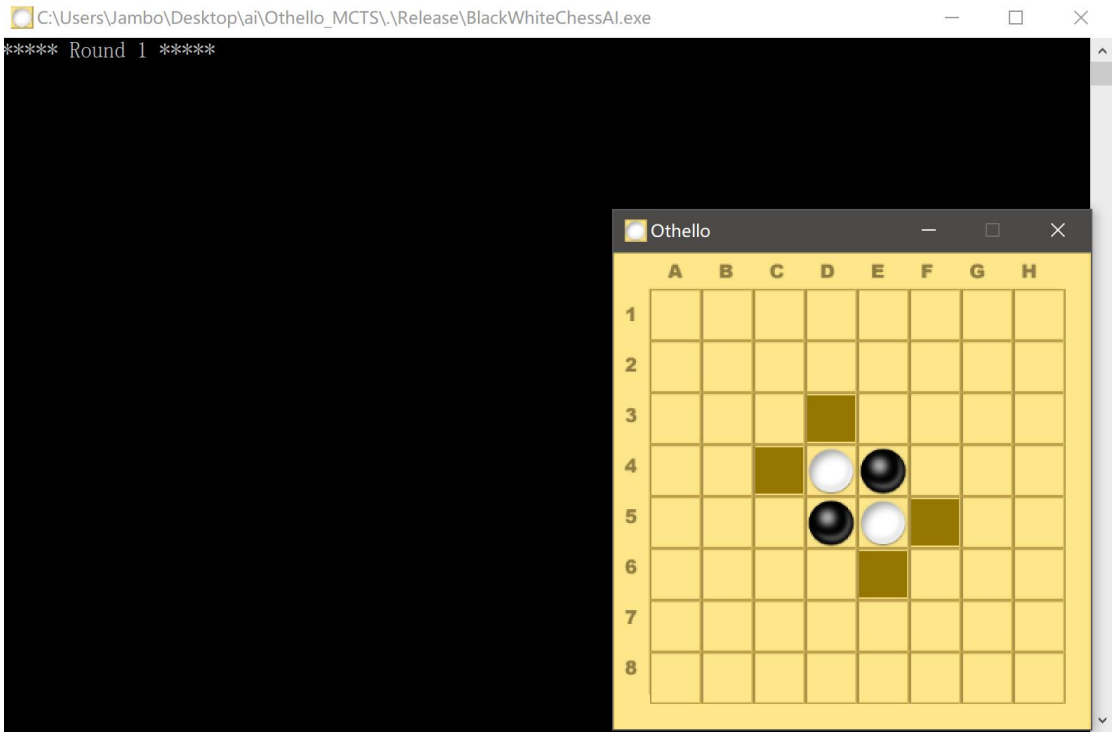


图 1

从图 1 中可以看到，深色的位置是 AI 可以落子的位置，因为我们选择的是 AI 先手的棋局，所以此时 AI 正在计算；

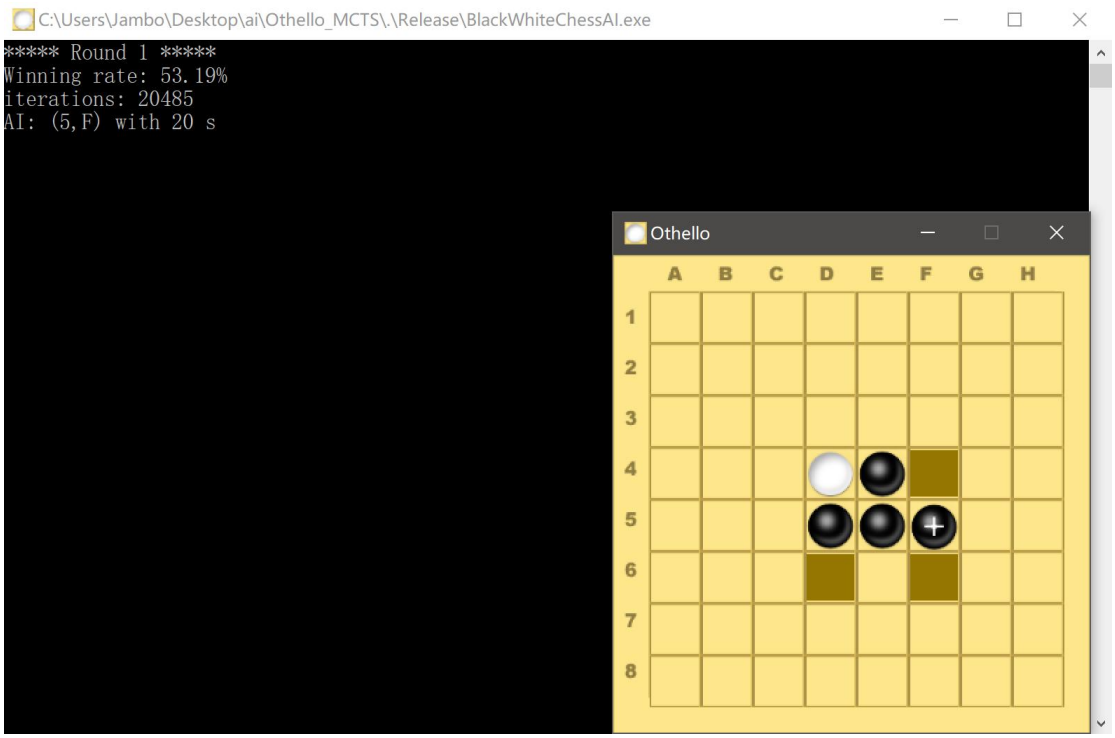


图 2

图 2 显示了 AI 计算完成并且落子之后的棋局情况，此时棋盘上深色位置代表的是 human 可落子位置，控制台此时依次输出了 AI 此次落子的预测胜率、总共的迭代次数、AI 的落子位置坐标以及所用时间，此次 AI 落子的胜率为 53.19%，迭代次数为 20485，落子位置为（5，F），用时 20s；

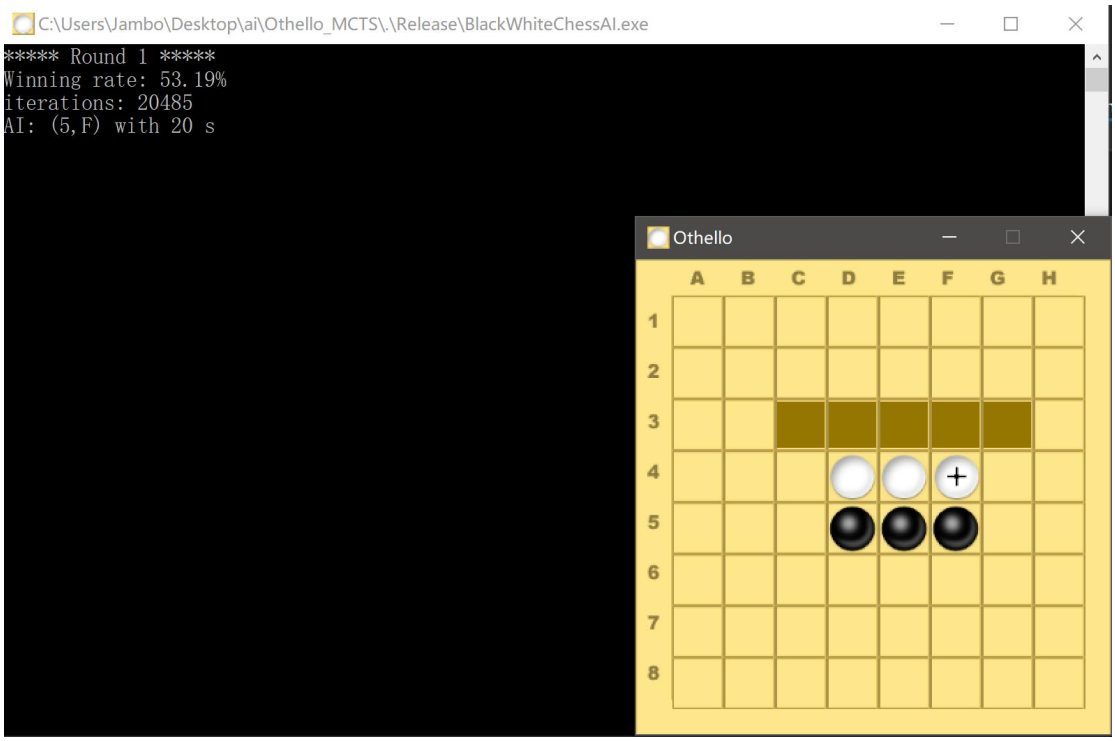


图 3

Human 在（4，F）落子之后，棋盘上深色位置又变为代表 AI 可落子位置，AI 开始计算最佳落子位置；

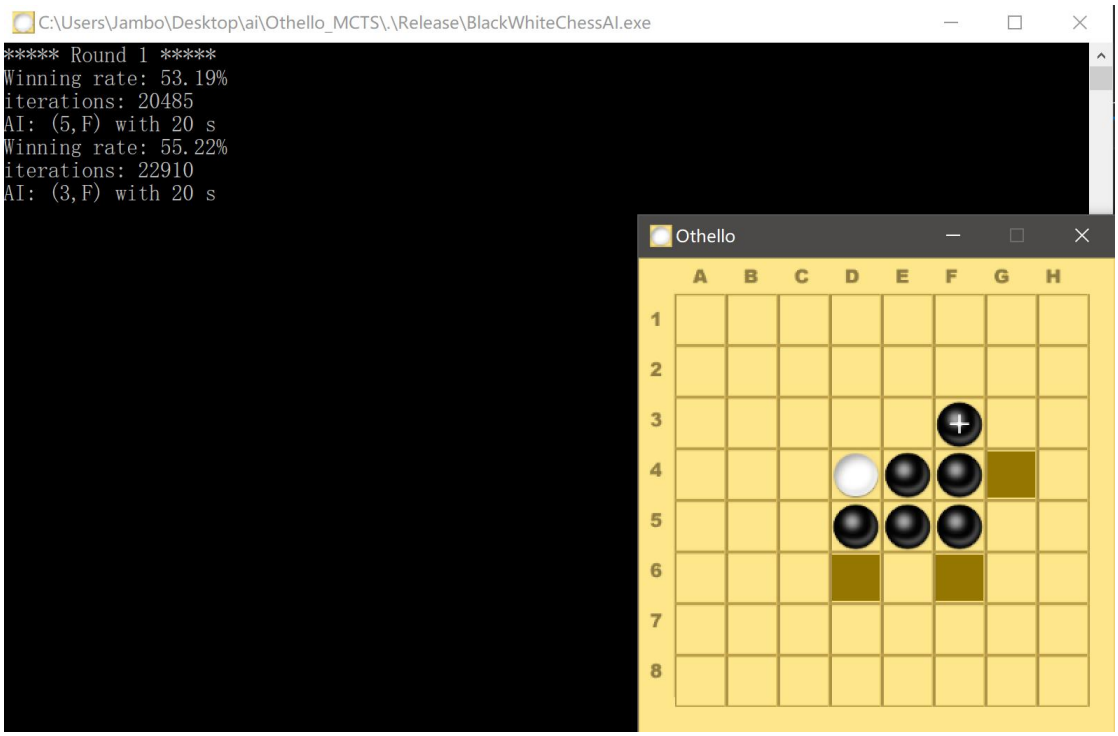


图 4

AI 计算完成，在（3，F）位置落子，胜率为 55.22%，迭代次数为 22910，用时 20s；

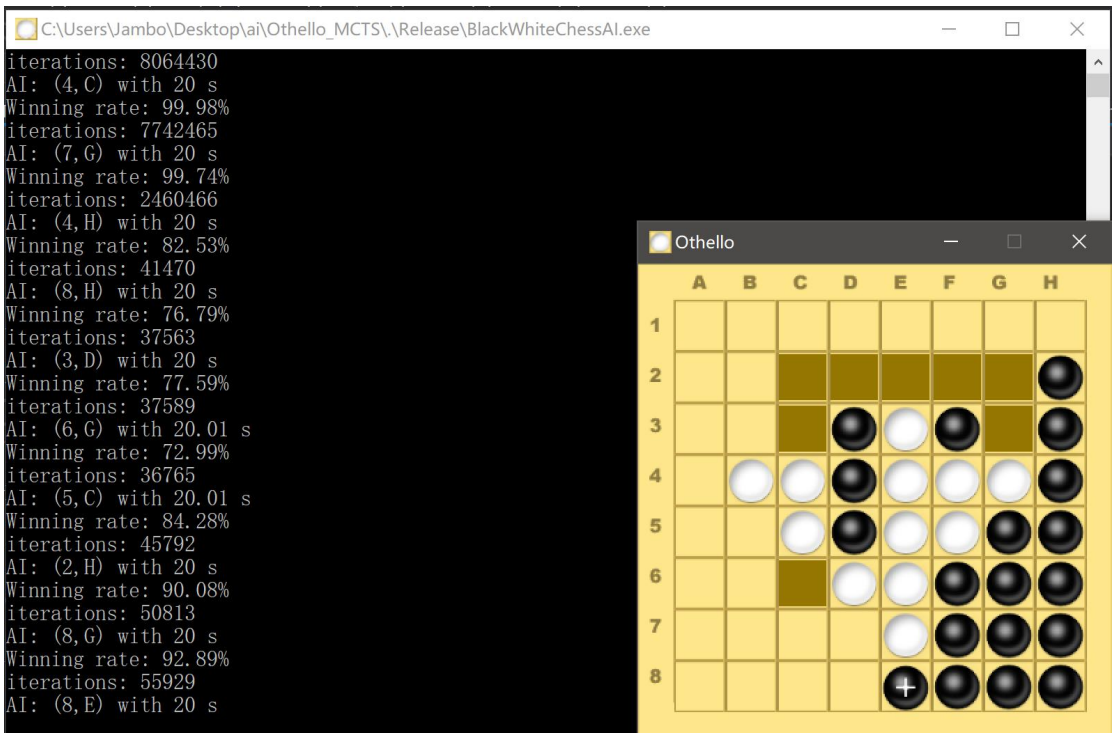


图 5

AI 和 human 依次交替落子，棋局进行到中间时如图 5 所示，此时轮到 human 落子；

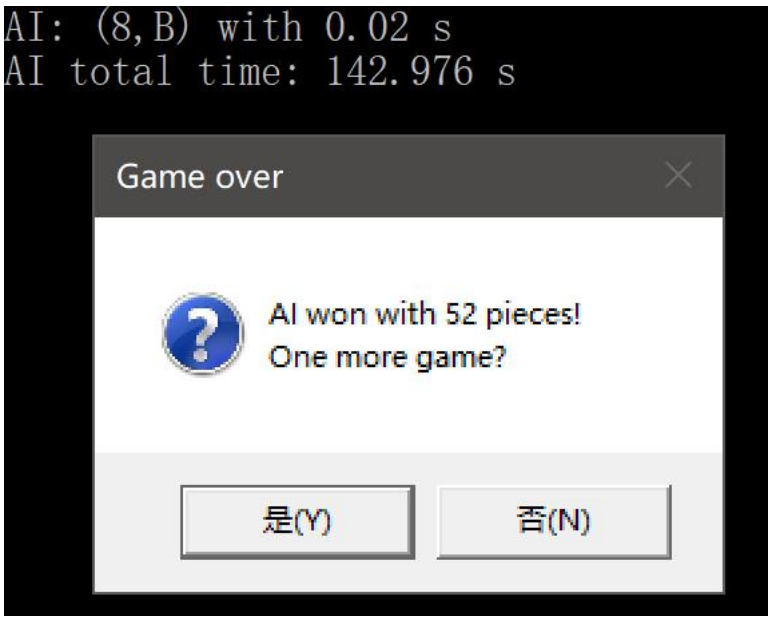


图 6

最终结果如图 6 所示，AI 赢得胜利，棋盘上共 52 枚 AI 的棋子，总用时 172.796s。

References:

http://mcts.ai/code/java.html

https://blog.csdn.net/u014397729/article/details/27366363