2021 计算概论 A——同化棋 AI 设计实验报告

王瑞环 2100013112

一、编写过程

（一）完善基础架构

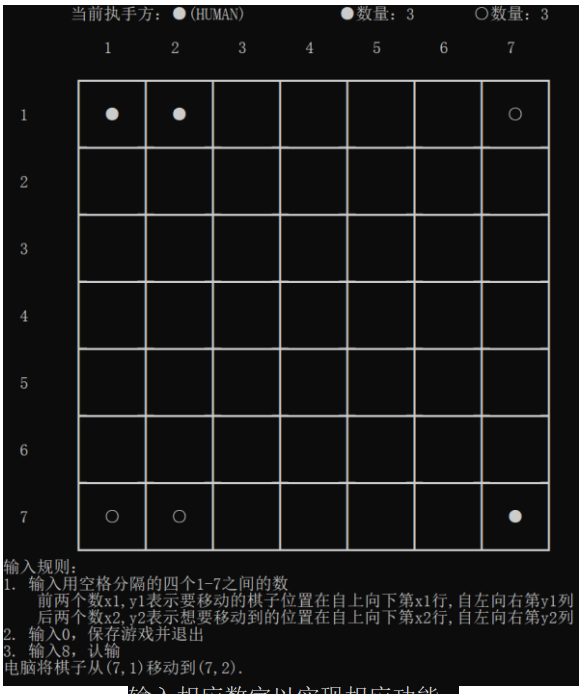
本阶段的主要目的是保证游戏可以正常运行，实现了如下功能：

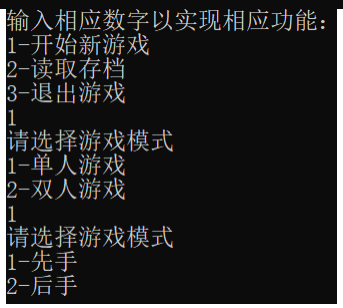
1. 棋盘显示功能

每走一步棋之后刷新界面并输出当前棋盘状态，主要通过 print_chessboard()函数实现。

通过 ┌、─、┐ 等表格符号和●、○等棋子符号显示整洁美观的棋盘，在棋盘边显示数字使玩家获取行列信息更为容易。

同时，在页面上显示输入规则、双方棋子的数目、当前执子方等信息、AI 走子后移动棋子的坐标、对局结束后的胜利方等，使玩家能迅速获得充分准确的信息。





2. 交互功能

对局中：玩家通过输入起点和落点的坐标操纵棋子移动。当玩家输入出错时（如坐标数字大于 7，或所选起点不是己方棋子，或落点位置已经有棋子等），显示“输入错误”并要求玩家重新输入。

对局外：玩家可通过输入相应数字以选择：

- 开始新游戏/读取存档/退出游戏；
- 单人游戏/双人游戏；
- （单人游戏时）执先手/后手；
- （对局结束后）查看复盘/重新游戏。

3. 存档、复盘、读档功能

自动记录：每一场对局开始时，程序自动开始记录对局情况到“save.txt”文件中，以供在当前对局结束后进行复盘。

手动记录：对局中途玩家可以输入 0 以保存退出，待下次打开游戏，可以从上次保存的局面继续游戏。

4. 用随机数实现的最基础自动走子

本功能用以进行试验，旨在确定单人游戏可以正常运行，为下阶段进行算法设计后程序的运行奠定基础。

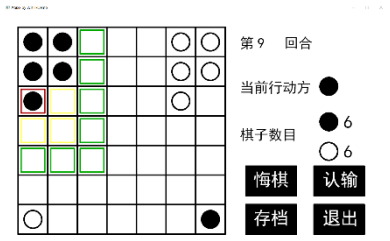
（二）算法设计与改进

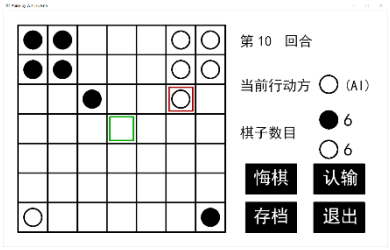
本阶段为 AI 配备了较强的算法，详见“核心算法”部分。

（三）UI 设计、交互改良与其他功能的设计

通过使用 EasyX[①]，对 UI 进行了设计，并且交互方式由键盘交互改为鼠标交互[②]。

1. 改进 UI 架构





如右图所示，在游戏过程中，新的 UI 界面相较于[第一阶段]有如下改良：

- 选择棋子后，显示可能落子的位置；
- 当 AI 走子时，用红色和绿色分别标记起始位置和落子位置，并调用<Windows.h>库中的 Sleep()函数使玩家能够充分获取信息；
- 增加回合数的展示。

2. 增加“查看帮助”功能

如左图所示，点击“查看规则”将跳转至 botzone 上介绍规则的网站 https://wiki.botzone.org.cn/index.php?title=Ataxx#，点击“观看演示”，则可跳转到 botzone 上演示同化棋规则的视频页面 https://www.botzone.org.cn/match/580ca98df1001dd47a928bdc 。

3. 增加“难度选择”功能

在单人游戏模式下，玩家在选择先后手后，能在“找自信”、“简单”、“普通”、“困难”四个难度中进行选择，分别对应于随机走子和搜索深度分别设置为 1~3 的 Minimax 算法。

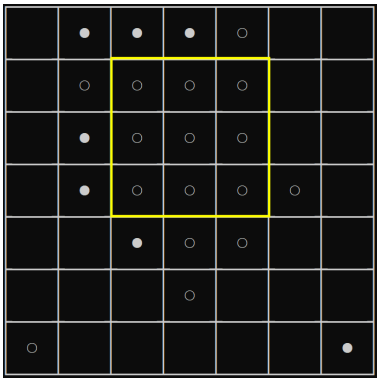
4. 增加悔棋功能

二、核心算法

第一版本：初始想法。

（一）算法

单步贪心算法



在当前局势下，仅考虑下一步的收益，考虑以下方面，将各个因素量化进行加权平均，找出最优走法：

①行动后己方与对方棋子相差数目。

②走子一格还是两格。

③走出一步之后，己方棋子图案中的“最大矩形”的面积。因为当棋子形成边长大于等于 2 的矩形时，中间的棋子总是安全的，这给决策增加了防御的因素。

（二）缺陷

虽然本想法的目的是想要攻防兼备，但是当全体因素加权后混合于同一个量内进行比较时，可能存在如攻击性较强的优解因防御力不足，使得整体估值低于某平凡解的情况。

---

① 官方网站：https://easyx.cn/，文档：https://docs.easyx.cn/。

② 鼠标交互代码参考：https://blog.csdn.net/weixin_45750972/article/details/108021021。

第二版本：参考谷雨助教关于亚马逊棋的博弈算法[①]，对第一版本进行了改进。

（一）算法

1. 贪心算法的改进

调整为仅考虑单一因素：走完一步之后本方的棋子占总棋子的比例。

一方面，第一版本的贪心算法，虽然尝试考虑多个因素，但是各方面混杂后效果并不理想；另一方面，棋子数占比相比较棋子相差数更能反映局势。

至于抛弃了所谓“最大矩形”的想法，是由于此版本并非分析单步局势，而是多步后的情况，将把对方的行子法考虑在内，故而可以将对单步防御力的衡量，转移到对对方下一步攻击力的衡量上。

根据改进后的贪心算法，定义估价函数 evaluation()，伪代码如下：

```
int evaluation(){ //对某个局面
    if(对方已获胜)return -1; //最不希望出现的情况
    else if(己方已获胜)return 1; //最希望出现的情况
    else(游戏能继续进行)return my_chess_num / total_chess_num;
}
```

2. MiniMax 算法（极大极小值算法）

本程序的策略为：我方所要走的每一步都要使得 evaluation()最大，并假定对方的策略为使我的 evaluation()最小。则我方可以通过分析按照这种策略多步之后，evaluation()的最大值，从而确定下一步。

定义函数 minimax(int depth, int minimax_depth)函数来实现此功能较优解，其中 minimax_depth 为设定的搜索深度，而 depth 为当前搜索深度，伪代码如下：

```
int minimax(int depth, int minimax_depth){
    depth++;
    if(depth >= minimax_depth || 游戏已结束)返回估值;
    if(我方走子){
        for(每一种可能的移动){
            行子;
            evaluation=minimax(depth, minimax);
            恢复原棋盘;
            result=max(result, evaluation); //我方走子，要使得估值尽可能大
        }
    }
    else if(我方走子){
        for(每一种可能的移动){
            行子;
            evaluation=minimax(depth, minimax);
            恢复原棋盘;
            result=min(result, evaluation); //对方走子，要使得估值尽可能小
        }
    }
    return result;
}
```

① GitHub 文档：https://github.com/Guyutongxue/Amazons。

实际程序中，出于对已完成部分整体结构的考虑，将 minimax()函数又分为三部分。考虑到时间因素，minimax_depth 设定为 3，但是这样一来，depth 和 minimax_depth 便处于一种较尴尬的地位，因为将三个函数分别调用一次后，depth 便达到 3，似乎并未达到用函数的递归调用简化程序的目的。虽然算法没有问题，但整体设计和架构方面是第二版本的一个不足之处。

3. 简单的剪枝

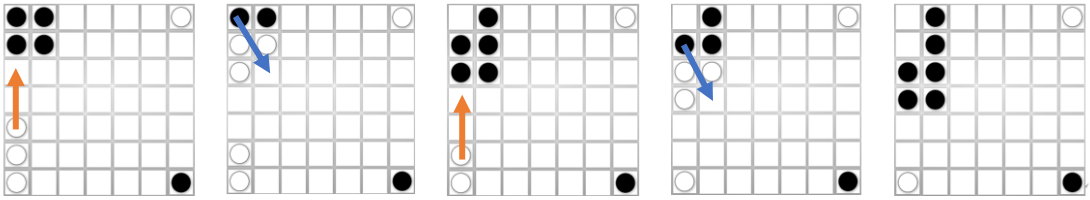
决策树节点随搜索深度指数级增加，即使最大深度设置为 4，检索用时也不可小觑。故而有必要在决策过程中进行剪枝，本程序采用最简单的剪枝方法，在 minimax()函数中增加一条 if 语句，实现下面的功能：

当我方走子时，若走过某一步后的 evaluation（记为 E0）相较于 result（已有的在 minimax_depth 步后最大可能的 evaluation）更小，则剪去。因为这一步之后轮到对方走子，在我们假定的策略下，它将走一步子使得我方的 evaluation 尽量小（记为 E1），则 E1 < E0。鉴于搜索深度仅为 3，几乎不可能在下一步挽回所有的损失，剪去最优解的概率并不大。

同样的道理，当模拟对方走子时，若走过某一步后的 evaluation 相较于 result 更大，则剪去。

（二）缺陷

在 botzone 上进行对局实验后，发现搜索深度为 3 时，在开局阶段，算法缺陷体现得十分明显。（我方执白）



1　　2　　3　　4　　5

当面对局势 1 时，AI 将会考虑到局势 4（3 步之后）是使得估值 evaluation 最大的情况之一，（在本程序对多种优解的取舍方式下）从而判断由局势 1 行子至局势 2 为自身的最优解；但是在对方行子至第 5 种情况时，己方反而将处于较大的劣势中。

第三版本：尝试解决第二版本的缺陷。

随机数

出于时间限制考虑，若不对剪枝算法进行较好的优化，便难以通过增加搜索深度的方式解决第二版本程序中的缺陷。

为了在一定程度上缓解这个问题，决定增加走子的随机性，即，将所有估值相同的走法记录下来，并随机选择其中的某一种，针对开局阶段，由于估值相同的情况较多，因而对局沿着“缺陷”方向进行的可能性会大大下降。

经 botzone 对局实验发现，引入随机数之后，便可能战胜第二版本无法战胜的对手。这说明，引入随机数虽然增加了对局的不确定性，但确实对于解决某些情况下“愈走愈劣”的问题有一定帮助。