北京大学 2021 年秋计算概论 A 大作业：同化棋程序报告

信息科学技术学院　　杨铠

2022 年 1 月

目录

1 程序交互方式与主要功能

本同化棋程序的主要功能有同化棋规则介绍、同化棋对战（包括两人对战与人机对战两种模式）、暂停与继续游戏、保存与回放对局、称号与段位等功能，在游戏的主菜单和对战界面分别加入了不同的背景音乐。整个游戏程序的交互是通过键盘和鼠标来完成的。

♣♣ 以下所有图片均需要重新截图制作

1.1 交互方式：利用键盘与鼠标操作

本游戏程序在菜单内的功能选择中选择通过键盘进行交互，玩家只需按下所需功能对应的键盘即可转向对应功能；而在对战过程中，将输入起点与终点坐标的交互方式改进为通过鼠标点按起点与终点对应位置的格子实现交互，使局面的变化与玩家的操作更为直接（见图1、图2）。

图 1: 游戏主菜单：通过键盘交互

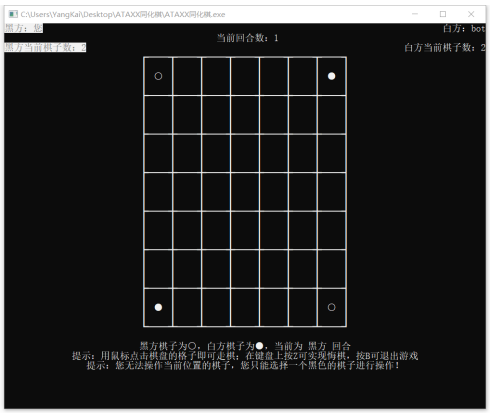

图 2: 对战过程：通过鼠标操作棋子

1.2 对战模式

本同化棋程序同时支持两人对战模式与人机对战模式，在新开始游戏界面内可进行选择（见图3）。在每一回合内，程序都会将控制台清屏并重新打印出当前棋盘的状态，并将此回合的执手方高亮（见图4）。

1.2.1 两人对战模式

在两人对战模式下，首先需要输入对战双方的用户名（不超过 20 个字符），然后方可开始对战。在对战过程中，双方可以轮流用鼠标操作本方棋子；同时，在游戏过程中可以按下 Z 键实现悔棋，

图 3: 对战模式选择

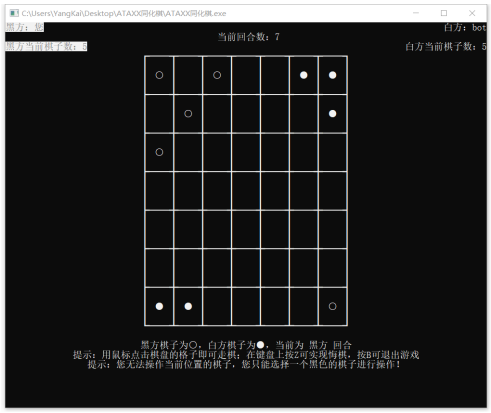

图 4: 此轮为黑方回合，左上角黑方信息高亮

按下 B 键回到主界面，按下 A 键可以实现由电脑代替玩家进行该回合的随机走棋。若在游戏中途回到主界面，此时本轮对局会被暂存，当下一次通过主界面的“继续”进入游戏时，再次输入对战双方的用户名之后，即可从上一次退出前的局面继续游戏（所谓的“下残局”）。

1.2.2 人机对战模式

在人机对战中，除了与两人对战模式中相同的悔棋、中途退出功能之外，本程序还提供了两项功能：先后手的选择与游戏难度的选择。其中游戏难度分为入门版、进阶版与困难版三个版本，其区别在于 AI 落子算法中搜索层数的不同。入门版搜索层数为 1 层，进阶版搜索层数为 2 层，困难版搜索层数为 4 层。关于 AI 落子算法的更多细节则是 §3的主要内容。

1.3 对局保存与回放

1.3.1 对局保存

在每次对局结束后，玩家都可以选择是否保存本次对局（见图5）。如果选择保存对局，则需要用一个不超过 20 个字符的名字命名本次对局（如果该名字已被使用过，则原先的对局将被覆盖；如果输入的名字超过 20 个字符则会提示重新输入）；保存后，在可执行文件.exe 的文件夹目录下会出现一个.txt 文档（见图6），其中所存储的即为此次对局的相关信息（包括对局的轮数、每一轮的棋盘状态以及每一轮执手方对棋子的操作等），此文档在“对局回放”功能中将会被使用到。

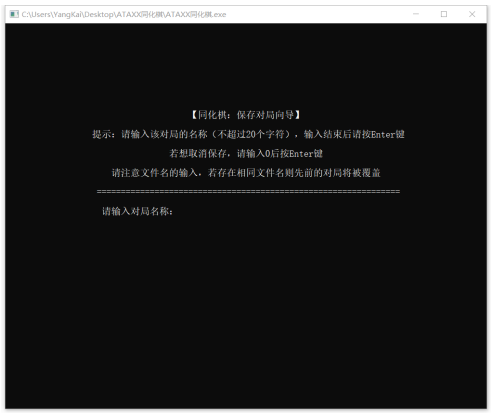

图 5: 游戏结束后，保存游戏的向导

图 6: 图中的 test 文档即为导出的对局相关信息

1.3.2 对局回放

对局回放的功能位于游戏主菜单下（见图1），进入后输入保存的对局名称即可进行对局回放（见图7）。在对局回放过程中，程序按照固定的时间间隔对局面进行刷新（见图8）。此外也可以通过 L、N 分别手动跳转至上一回合或下一回合；按 P 键可以暂停回放，暂停回放后按任意键可继续播放（见图9）；按 B 键可以直接终止回放并返回游戏的主界面。回放结束后，程序将输出对局结果并提示“按任意键返回游戏主菜单”（见图10）。

图 7: 用于回放历史对局的向导界面

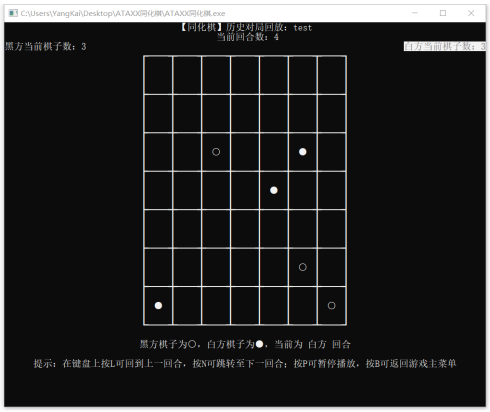

图 8: 回放中，本轮执手方（白方）被高亮

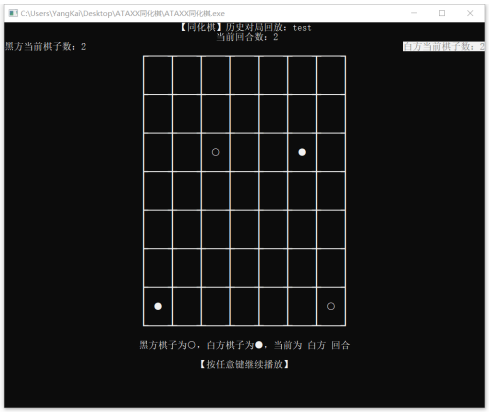

图 9: 回放暂停后的界面

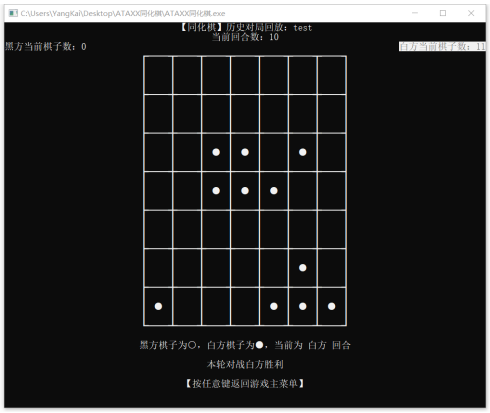

图 10: 回放结束后的界面

1.4 其它功能

- 背景音乐
  本程序中游戏菜单及对战过程中分别设置了不同的背景音乐，并在人机对战结束时根据玩家的胜负播放不同的音效。
- 规则介绍
  主界面中的“规则介绍”内有对规则的详细介绍（见图11）。
- 称号与段位主界面中的“称号与段位”内从人机对战的总局数、胜率两个角度进行评判。具体标准见图 ♣♣?。

♣♣ 待补图：版本号与作者介绍页面

♣♣ 待补：其它功能，如胜率计算、成就榜（打了多少场比赛、先后手和不同难度【先后手不做，不同难度已做】胜 AI 多少场、打了至少多少场且胜率不低于多少）【胜率计算和成就榜已经合起来制作完成，成就榜与玩的局数挂钩，段位与胜率及难度挂钩】、给出某种落子提示、自动辅助落子等等【已写，落子提示不再计划写】。
♣♣ 待改：键盘交互不要用数字，好像很不灵敏。（已经修改，前面的截图回头需要重做）
♣♣ 可修改部分：左上角的程序名称【已经修改】以及程序图标【已经修改】
♣♣ 待补图：版本号与作者介绍页面
♣♣ 报告【功能模块（第一章）】待补内容：中途退出，存盘与继续游戏
♣♣ 报告待修改内容：程序逻辑的说明，后续修改了功能，人人对战继续时重新输入用户名，人机对战取消重新选择难度的功能
♣♣ 报告待修改内容：诸如存盘读盘、中途退出游戏、难度选择等内容的具体实现方式（放在第二章）

图 11: 游戏规则介绍界面

2 程序架构和功能的设计与实现

♣♣subsection 的标题与内容待修改，2.2 2.4 考虑合并

2.1 游戏界面与功能间的切换

本游戏采取在控制台上运行的模式，将不同的界面写在不同的分文件内，通过 main 函数将所有的分文件连接起来。大部分的分文件均用于实现某一个功能（如两人对战模式、AI 先手的人机对战模式、历史对局回放等），也有少部分的分文件用于实现多个地方需要调取的操作（如隐藏和显示光标、控制字符串在某一行居中等）。而同一个功能内的分文件往往包含多个函数，main 函数在调用时仅调用每个分文件中主要的一个函数，分文件中函数在运行中再去调用分文件中的其它函数。这样写的优势在于可以较好地保持各个功能之间的相对独立性，也会更方便进行功能的调试与拓展。

2.2 游戏界面的设置与输出

本程序在 main 函数内设置了程序启动时的控制台宽度和高度（设置宽度为 100，高度为 41，并且禁止用户在使用过程中拖动控制台窗口大小以及最大化，具体代码见算法1）。在此基础上，通过获取要输出内容的长度，利用控制台位置坐标，将需要输出的内容按需要实现左对齐、居中或右对

齐的输出位置（其中在回放结束界面的效果可参见图10）。

```
1 system("mode con cols=100 lines=40");
```

算法 1: 控制输出台宽度为 100，高度为 40

2.3 键盘与鼠标交互的实现

2.3.1 键盘交互的实现

在本程序中，除了在两人对战模式中输入用户名、保存对局与回放对局是使用 cin 输入流进行数据的读取之外，其余场合均采用了 GetAsyncKeyState 函数进行键盘操作的获取。

GetAsyncKeyState 函数的返回值为 short 类型，规则为：自对 GetAsyncKeyState 函数的上一次调用以来，如键已被按过，则将最低位设为 1，否则设为 0；如键处于按下状态，则将最高位设为 1，如处于抬起状态，则设为 0。

在本程序内，目标是在每一个界面，通过按下对应的键实现功能的转向。因此在每一次需要监测键盘状态之前，采取了对所有本程序中用到过的键位运行一次 GetAsyncKeyState 函数的方法，并对当前需要监测的键位采用 GetAsyncKeyState 函数返回值与 1 进行按位与操作的办法，从而判断在当前界面被打开后，对应的按键是否被按下，并以此为基础进行界面与功能的切换。

2.3.2 鼠标交互的实现

本程序在对战过程中，将基本的输入起点和终点棋子坐标的交互方式改进为通过鼠标在界面上点按完成对棋子的操作。其主要的运行过程是通过监测鼠标左键在控制台上的点击事件，获取鼠标左键的点击坐标，再结合 §2.2中所述的棋盘输出的位置，计算出当前鼠标按下的位置是否在某个棋盘的格子内。如果当前鼠标按下的位置不在棋盘内或不符合落子的规则，则输出对应的错误提示；若符合规则，则进行对应的存储移动路径和棋盘新状态等操作，并进行清屏与重新输出新的棋面等操作。

2.4 Sleep() 函数的使用

由于程序运行的速度很快，若不在某些场合下暂停程序的运行，程序的运行结果便会一闪而过，使玩家无法动态地看到棋盘局面发生的变化。因此在人机对战、对局回放等场景，都采用了 Sleep() 函数，其中的参数设置为 500 到 1500 毫秒不等，目的是暂停程序的执行并让控制台上的棋局保持在当前状态，使用户在观看的时候对棋面变化有动态的感知。

此外，在三种对战模式下，均实现了每回合刷新轮数、执手方的功能。这里也使用了 Sleep() 函数，其中的参数设置为 500 毫秒，使得每轮在清屏并重新输出棋盘状态、将执手方信息高亮后，暂停 500 毫秒再输出轮数与执手方，实现较好的动画效果。

2.5 其它

- 禁止最大化与调整窗口大小
  由于程序中使用了大量的居中以及定位输出，因此为了使程序界面不出现混乱，本程序禁止了用户将界面最大化以及禁止了用户调整窗口大小。

- 待补
  ♣♣ 如：对输入异常的处理方式，

3 AI 落子算法的设计与实现

3.1 搜索算法：α-β 剪枝算法

α-β 剪枝算法是棋类对弈中常见的一种搜索算法，其主要方法是模拟双方在未来几步的所有可能落子方式，并在搜索的终点对每种棋面给出一个评估，在所有棋面中选择对己方最有利的一种进行落子。由于棋类游戏博弈树非常庞大，因此在每一步不可能计算出到终局的所有可能，因此只能做一定深度的搜索。而 α-β 剪枝算法在进行深度优先搜索的时候，引入了下界 α 和上界 β，在发现继续下一层的搜索已经无法找到更有利的解时发生剪枝，从而减少搜索量。α-β 剪枝算法的伪代码见算法2（此处给出的伪代码搜索层数为 4 层，事实上最后此 AI 采用的搜索层数也是 4 层；其中的 Evaluate 函数为局面评估函数）。

```
   Input: board board, depth depth, alpha α, beta β, side to move color
   Output: maxium of its child roots
 1 maxium ← -∞;
 2 if depth == 4 then
 3 |   return currBotColor · Evaluate(board);
 4 if the game is over then
 5 |   if the bot wins then
 6 |   |   return currBotColor · (1 << 30);
 7 |   return -currBotColor · (1 << 30);
 8 foreach move do
 9 |   MakeMove(board + 1, move, color);
10 |   result ← - AlphaBeta(board + 1, depth + 1, - β, - α, - color);
11 |   if result > α then
12 |   |   if result ≥ β then
13 |   |   |   return result;
14 |   |   α = max(result, α);
15 |   maxium = max(result, maxium);
16 end
17 return maxium;
```

算法 2: AlphaBeta

影响 α-β 剪枝算法效果的因素有二，一方面是搜索的层数，另一方面是评估函数的准确与否。在评估函数上，本人通过引入除双方棋子数之差外的两个参数：稳定子与行动力，并尝试调整评估函数中三者的系数之比，来尽量实现对局面评估的客观性与准确性，这是 §3.2的内容；而在搜索层数上，则采用了一些手段减少了重复运算，并对程序运行的时间进行检测，从而在保持搜索层数为 4 层的情况下降低了决策超时发生的频率，这是 §3.5的内容。

3.2 局面评估函数

正如在 §3.1最后所提到的，局面评估函数的设计会在很大程度上影响 AI 算法最后给出的落子结果。以下将先给出本人对同化棋棋面的一点个人的分析与思考，在此基础上提出同化棋局面的评估参数，最后简要说明确定参数间系数比例的思路。

3.2.1 关于同化棋棋面的分析与局面评估参数的构想

根据同化棋的规则，判断胜负的标准就是双方的棋子数，因此将当前双方棋子数之差作为局面评估参数之一并且占据较大比例是合理且较为客观的。

但仅用双方棋子数之差这一个参数显然不够客观。例如，倘若某一方的棋子在 3×3 的方格表内占据了除了中心以外的剩余 8 个位置，而对手在下一回合占据了该 3×3 的方格表的中心，则此时对手在此轮将增加 8 个棋子。可见，这 8 个棋子并不“稳定”，因为其周围仍然留有空位，且该空位周围均为同一方的棋子。此即我们引入 §3.2.3中所定义的稳定子的动机。

另外，当一方无法行棋的时候，游戏将结束；并且，倘若一方的合法行棋数量较少或者大都需要通过“剪切粘贴”实现，那么其走棋往往可能容易受制于对方，因此可以考虑用某种方式描述双方可以落子位置数目的多少，此即 §3.2.4中所定义的行动力。

3.2.2 局面评估参数 1：双方棋子数之差

双方棋子数之差是最朴素、但同时也是最重要的局面评估参数，其原因无非最终判断胜负的标准即为双方棋子数。在本 AI 中，为了 §3.3、§3.4中的试验参数更方便设置，此处将双方棋子数之差的权重值设置为 1000。

3.2.3 局面评估参数 2：稳定子

正如 §3.2.1中提到的，稳定子旨在刻画局面中双方“较不容易”被翻转的棋子个数。根据规则容易发现，倘若某一个棋子的周围 3×3 的范围内（不包括在棋盘外的部分）均已经有棋子，那么这个棋子在当前回合就不可能被翻转。而倘若一个棋子不是稳定子，且其周围有对方的棋子，那么它就有可能在本回合被翻转。而在实际的局面评估函数内，与 §3.2.2中相同地，采取的也是双方稳定子个数之差作为参数。

3.2.4 局面评估参数 3：行动力

行动力旨在刻画局面中双方可以到达的位置总数，具体到棋面上可以理解为两方之间的“包围”关系。从理论上来说，计算行动力需要对每个本方的棋子考虑其周围 5×5 范围内的所有位置，但这种方式的计算量较大，且可能将一些比较无意义的“剪切粘贴”也计入其中。因此，以牺牲一部分精度为代价，转为考虑某一方棋子通过“复制粘贴”可以到达的所有位置的总数（不计重数）。同样地，在评估函数内采用的参数也是双方行动力之差。

3.2.5 关于对局前期、中期和后期的思考

§3.2.2、§3.2.3、§3.2.4中介绍了本 AI 采用的三个局面评估参数，但是在对局前期、中期和后期三者的系数应当有所区别。

在对局的前期，棋盘上的棋子数较少，且双方之间每一回合的同化数也较少，因此计算稳定子和行动力的意义不大。因此在前几回合，估值函数采取了最简单的双方棋子数之差。

在对局的中期，棋盘上的棋子数较多，且每一回合的局面均有可能发生比较大的变化。此时在对棋局进行评估时，就应当加入 §3.2.3、§3.2.4中分别提到的稳定子与行动力。

而到了对局的后期，此时棋盘往往处于即将被填满的状态，此时较理想的评估函数应当与胜负的评判规则相同，因此在对局的后期评估函数采取的也是双方棋子数之差。

根据以上的讨论，本 AI 最后采用的局面评估函数形如：

$$f(x)=\begin{cases}1000\times f_1(x), & \text{回合数}\le c_1\\ 1000\times f_1(x)+c_2\times f_2(x)+c_3\times f_3(x), & \text{回合数}>c_1\ \text{且棋盘棋子数}<44\\ 1000\times f_1(x), & \text{棋盘棋子数}\ge 44\end{cases}\tag{1}$$

其中 $f(x)$ 代表评估函数，$f_1(x)$ 代表我方棋子个数 − 对方棋子个数，$f_2(x)$ 代表我方稳定子个数 − 对方稳定子个数，$f_3(x)$ 代表我方行动力 − 对方行动力，c_1,c_2,c_3 为待定系数。而这些系数的确定方法是 §3.3的主要内容。

3.3 局面评估函数中系数的确定方法

在评估函数式 (1) 内有三个待定参数。在 AI 的设计过程中，本人在上述 AI 的基础上，将 c_1,c_2,c_3 设置为可变量，并让 (c_1,c_2,c_3) 在取若干组不同的取值下，让对应的 AI 进行两两对弈并统计结果，从而选择出较好的估值系数。本试验中搜索算法、局面评估函数的伪代码分别如下：

♣♣ 表述及伪代码块在文中还是文末待调整

```
Input: board board, depth depth, alpha α, beta β, side to move color, parameter c1,
       parameter c2, parameter c3
Output: maxium of its child roots
maxium ← -∞;
if depth == 4 then
    return currBotColor · Evaluate(board, c1, c2, c3);
if the game is over then
    if the bot wins then
        return currBotColor · (1 << 30);
    return -currBotColor · (1 << 30);
foreach move do
    MakeMove(board + 1, move, color);
    result ← - AlphaBeta(board + 1, depth + 1, - β, - α, - color, c1, c2, c3);
    if result > α then
        if result ≥ β then
            return result;
        α = max(result, α);
    maxium = max(result, maxium);
end
return maxium;
```

算法 3: 加入试验参数的 AlphaBeta 算法

Input: board *board*, depth *depth*, parameter c_1, parameter c_2, parameter c_3
Output: result of evaluaion

```
num.blackchess ← 0;
num.whitechess ← 0;
num.black.stable ← 0;
num.white.stable ← 0;
movability.black ← 0;
movability.white ← 0;
foreach place on the board do
    if the color of the chess is black then
        num.blackchess++;
        foreach place on the borad near the chess do
            if all of them are not empty then
                num.black.stable++;
        end
    if the color of the chess is white then
        num.white.chess++;
        foreach place on the board near the chess do
            if all of them are not empty then
                num.white.stable++;
        end
    if the place is empty then
        foreach place on the board near it do
            if the color of the chess on some of the places is black then
                movability.black++;
            if the color of the chess on some of the places is white then
                movability.white++;
        end
end
if the turns of the board ≤ c1 then
    return 1000 × (num.blackchess − num.whitechess)
if the turns of the board > c1 and num.blackchess + num.whitechess < 44 then
    return 1000 × (num.blackchess − num.whitechess) + c2 ×
      (num.black.stable − num.white.stable) + c3 × (movability.black − movability.white);
if num.blackchess + num.whitechess ≥ 44 then
    return 1000 × (num.blackchess − num.whitechess);
```

算法 4: 加入试验参数的 Evaluate 局面评估函数

3.4 试验过程与试验结果

在试验过程中，本人给 c_1, c_2, c_3 均设置了 4 个不同的取值：♣♣ 先设置，选出最好的值，再进行 botzone 比赛试验微调

♣♣ 试验结果最后插入局面评估函数的最终伪代码

3.5 优化与改进

3.5.1 终局判断

♣♣ 手动修正为复制粘贴、对填满棋盘提出特判、并且防止出现双方反复移动导致局面复现

3.5.2 提升速度

♣♣ 减少重复计算，预存数组，只算一次

3.5.3 卡时间，到点退出

♣♣ 标题待改，内容待补

3.6 比赛结果

♣♣ 截图 botzone 比赛结果

Data: this text
Result: how to write algorithm with LaTeX2e

```
initialization;
while not at end of this document do
    read current;
    if understand then
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end
end
```

算法 5: ?

4 总结

4.1 收获

4.2 可改进的地方

4.2.1 字符界面的美化与图形界面的设计

4.2.2 AI 估值函数参数的进一步调整

♣♣ 主要问题：参数调整仅依靠互相对弈，一方面不完全客观，另一方面当参数取值相近时此方法往往难以奏效