

## Python 大作业：黑白棋游戏

黑白棋 (Reversi or Othello) 在西方和日本很流行。游戏通过相互翻转对方的棋子，最后以棋盘上谁的棋子多来判断胜负。

黑白棋的每颗棋子由黑白两色组成，一面白，一面黑。每次落子，把本方颜色的棋子放在棋盘的空格上，若在横、竖、斜八个方向的任一方向上有本方棋子，则被夹在中间的对手棋子全部翻转为本方棋子颜色；并且，仅在可以翻转棋子的地方才能落子。如果一方至少有一步合法棋步可下，他就必须落子，不得弃权。棋盘已满或双方都没有棋子可下时棋局结束，以棋子数目来计算胜负，棋子多的一方获胜。在棋盘还没有下满时，如果一方的棋子已经被对方吃光，则棋局也结束，将对手棋子吃光的一方获胜。

该游戏非常复杂，是一种得分会戏剧性变化并且需要长时间思考的策略性游戏。我们仅尝试实现该游戏的一个简化策略版本，即，人和计算机下黑白棋，计算机根据事先设定的策略下棋。游戏具体描述如下：

在  $n \times n$  的棋盘上 ( $n$  是偶数，且  $4 \leq n \leq 26$ )，两个玩家分别是人和计算机：一个玩家持白棋，另一个玩家持黑棋，棋子的颜色可以通过翻转发生改变。 $4 \times 4$  棋盘的初始状态如图 1 所示：在棋盘中心分别摆放了 2 颗黑棋和 2 颗白棋。棋盘的行和列用字母 a、b、c、d、.....标明。

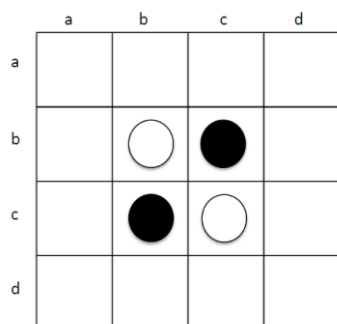


图 1. 棋盘的起始状态

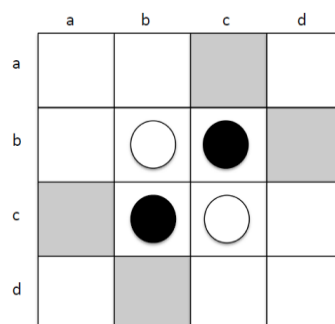


图 2. 白棋落子的候选位置

每次落子，玩家把本方棋子放在一个空棋盘格内，落子需遵守以下 2 条规则：

- 1、以空棋盘格为中心的 8 个方向（东南西北及对角线方向）中，至少在一个方向上，对手的棋子与该空棋盘格构成连续直线；
- 2、在该直线的末端必须已经放置有一颗本方棋子。

玩家落子后，满足上述规则的对手棋子被翻转为本方棋子颜色。

图 2 阴影位置显示了持白棋玩家下一步可以落子的所有候选棋盘格。假如白棋玩家决定走行 c 列 a，在行 c 列 b 的黑棋将被翻转成白色。如图 3 所示：

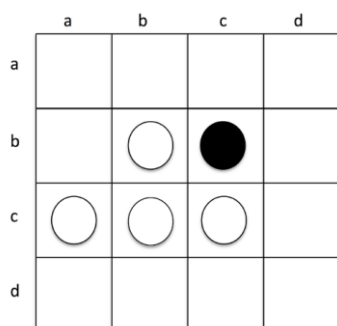


图 3. 白棋落子行 c 列 a 后的棋盘状况

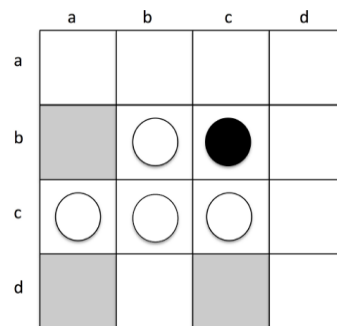


图 4. 黑棋落子的候选位置

图 4 显示了黑棋玩家可能的落子位置。假如黑棋玩家在行 b 列 a 放置一颗棋子，棋盘将如图 5 所示：

	a	b	c	d
a				
b	●	●	●	
c	○	○	○	
d				

图 5. 黑棋落子行 b 列 a 后的棋盘状况

	a	b	c	d
a			○	
b	●	○	○	
c	○	○	○	
d				

图 6. 白棋落子行 a 列 c 后的棋盘状况

接下来，如果白棋玩家在行 a 列 c 落子，则棋盘如图 6 所示。注意：此次白棋的落子使得西南和南两个方向上的黑棋变白。

两位玩家轮流下棋，直到一方没有符合规则的落子位置，在这种情况下，剩下的一方继续下棋，直到对手有了可以落子的位置。此时，恢复两者轮流下棋的顺序。如果一方落子在非法位置，则视为放弃本次对弈，对方获胜。**游戏结束的条件：1) 整个棋盘满了；2) 一方的棋子已经被对方吃光；3) 两名玩家都没有可以落子的棋盘格；4) 一方落子在非法位置。**前 3 种情况以棋子数目来计算胜负，棋子多的一方获胜；第 4 种情况判定对方获胜。

## 人机对弈流程

首先，程序询问用户棋盘的大小。接着，程序询问用户“计算机持黑棋还是白棋”。在本程序中，我们用字母‘x’代表黑棋，用字母‘o’代表白棋，**并且假设总是黑棋玩家先走**。所以，如果计算机持黑棋，计算机就先走；否则，程序提示人类玩家先走。每走一步，程序输出棋盘。黑白棋玩家轮流下棋，直到一个玩家无符合规则的落子位置。此时，程序输出信息“O player has no valid move.”（假设白棋玩家无棋可走），并且提示黑棋玩家继续下棋。每走一步，程序除输出棋盘外，还要检测游戏是否结束。如果程序检查出游戏结束，输出输赢信息并中止程序。输赢信息可以是：“O player wins.”，“X player wins.” 或者“Draw! ”。如果用户落子非法，程序应检测到并且输出“Invalid move.”，结束程序，宣布赢家。**游戏结束时，将本次人机对弈的相关信息写入日志文件。**

## 功能函数：

根据人机对弈流程，可将程序划分为不同函数。在把函数连接成一个大程序之前，请仔细测试每个函数。

1、Init\_board()：读入棋盘大小 n (n 为偶数，且  $4 \leq n \leq 26$ )，按照要求初始化棋盘。程序使用如下字符表示每个棋盘格的状态：

- . - 未被占用
- x - 被黑棋占用
- o - 被白棋占用

2、printBoard()：输出棋盘。例如：4×4 棋盘的初始状态如下：

```
a b c d
a . . . .
b . o x .
c . x o .
d . . . .
```

3、computer\_move(color)：计算机下棋。落子位置表示为“行列字母”的格式，如：“ba”代表棋子落在行 b 列 a。

4、human\_move(color)：用户下棋

- 5、`check_board()`: 检测游戏是否结束
- 6、`check_legal_move(row, col, color)`: 检测颜色为 `color` 的棋子落在棋盘格 `(row, col)` 上是否合法
- 7、`flip(row, col, color)`: 翻转 8 个方向上的对手棋子
- 8、`position_score(row, col, color)`: 计算“分值”（详情参见“计算机选择落子位置的策略”）
- 9、`gameover()`: 游戏结束，统计得分，输出结果
- 10、`saveinfo()`: 把每次人机对弈的信息作为一行写入文件 `reversi.csv` 中，这些信息包括：游戏开始的时间、单次游戏使用的时间、棋盘大小、黑棋玩家、白棋玩家、游戏比分，信息之间使用逗号字符分隔。

以上仅列出部分功能函数，请同学们根据需要，自行添加更多功能函数。

### 检测 8 个方向的策略

定义一个保存 8 个方向纵横坐标位移的元组：

```
direction = ((-1, -1), (-1, 0), (-1, 1),    # NW, N, NE
             (0, -1), (0, 1),                # W, E
             (1, -1), (1, 0), (1, 1))        # SW, S, SE
```

对某个棋盘格 `(row, col)` 循环访问该元组的每个元素 `(x, y)`，即可获得 8 个方向的对应坐标 `(row+x, row+y)`，如： `x=-1, y=1` 时， `(row-1, col+1)` 代表东北方向。

### 计算机选择落子位置的策略

对每个可能的落子位置，都进行尝试，计算该位置的“分值”（可以翻转的对手棋子数量），分值越高则在该位置落子越有利。计算每个可能位置的分值，选择最大值位置落子。图7是计算机持白棋时的分值情况。注意：行a列a的分值是2，因为该位置可以使2个黑棋翻转。无效的落子位置没有分值。







	a	b	c	d
a	2	1	1	
b				
c				
d				

图 7. 白棋的可能落子位置的分值情况

需要注意的是：可能有2个或多个棋盘格有相同的分值。这种情况下，选择行字母最小的棋盘格。如果两个棋盘格分值相同且在同一行，则选择列字母较小的棋盘格。按照 `direction` 元组的方向设定，循环计算分值、并测试是否为最高分值，就能确保满足该规则。

### 程序执行样例

以下是程序执行样例（你的程序必须遵循相同的输出样式）。为了获得对齐的棋盘输出，请使用等宽字体（如：Courier New）。注意：在本轮人机对弈即将结束时，计算机（持黑棋）无合法落子位置，用户（持白棋）连续落子；紧接着，程序检测到双方均无合法落子位置，游戏结束。

```

Enter the board dimension: 4
Computer plays (X/O) : X
  a b c d
a . . . .
b . O X .
c . X O .
d . . . .
Computer places X at ab.
  a b c d
a . X . .
b . X X .
c . X O .
d . . . .
Enter move for O (RowCol): aa
  a b c d
a O X . .
b . O X .
c . X O .
d . . . .
Computer places X at ba.
  a b c d
a O X . .
b X X X .
c . X O .
d . . . .
Enter move for O (RowCol): ac
  a b c d
a O O O .
b X X O .
c . X O .
d . . . .
Computer places X at ad.
  a b c d
a O O O X
b X X X .
c . X O .
d . . . .
Enter move for O (RowCol): ca
  a b c d
a O O O X
b O O X .
c O O O .
d . . . .
Computer places X at da.
  a b c d
a O O O X
b O O X .
c O X O .
d X . . .
Enter move for O (RowCol): dc
  a b c d
a O O O X
b O O X .
c O O O .
d X . O .
X player has no valid move.
Enter move for O (RowCol): cd
  a b c d

```

```

a O O O X
b O O O .
c O O O O
d X . O .
Both players have no valid move.
Game over.
X : O = 2 : 11
O player wins.

```

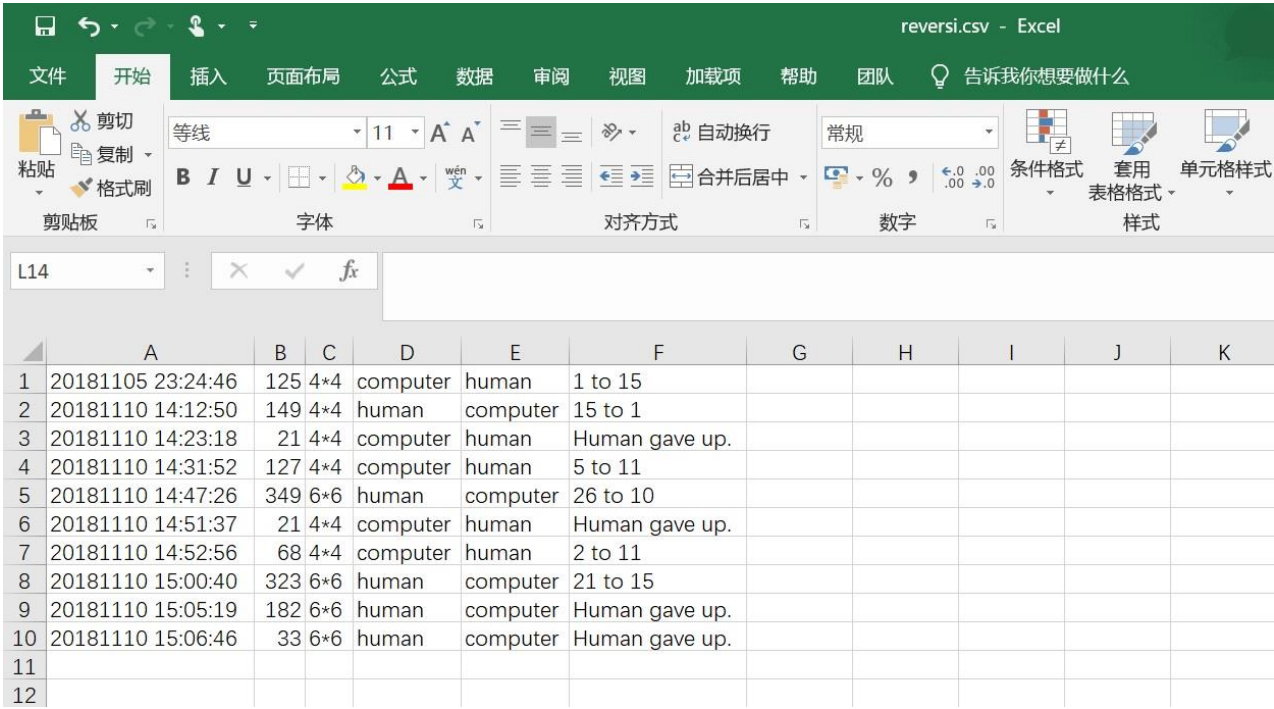
以下是另一次程序执行示例。在这次对弈中，用户落子在非法位置，计算机获胜。

```

Enter the board dimension: 6
Computer plays (X/O) : O
  a b c d e f
a . . . . .
b . . . . .
c . . O X .
d . . X O .
e . . . . .
f . . . . .
Enter move for X (RowCol): cb
  a b c d e f
a . . . . .
b . . . . .
c . X X X .
d . . X O .
e . . . . .
f . . . . .
Computer places O at bb.
  a b c d e f
a . . . . .
b . O . . .
c . X O X .
d . . X O .
e . . . . .
f . . . . .
Enter move for X (RowCol): aa
Invalid move.
Game over.
O player wins.

```

日志文件 Reversi.csv 的示例如下图所示（每行信息对应一次人机对弈）：



	A	B	C	D	E	F	G	H	I	J	K
1	20181105 23:24:46	125	4*4	computer	human	1 to 15					
2	20181110 14:12:50	149	4*4	human	computer	15 to 1					
3	20181110 14:23:18	21	4*4	computer	human	Human gave up.					
4	20181110 14:31:52	127	4*4	computer	human	5 to 11					
5	20181110 14:47:26	349	6*6	human	computer	26 to 10					
6	20181110 14:51:37	21	4*4	computer	human	Human gave up.					
7	20181110 14:52:56	68	4*4	computer	human	2 to 11					
8	20181110 15:00:40	323	6*6	human	computer	21 to 15					
9	20181110 15:05:19	182	6*6	human	computer	Human gave up.					
10	20181110 15:06:46	33	6*6	human	computer	Human gave up.					
11											
12											