

Reversi AI implementation

Developer: Yiding Chang

Instructor: Jigang Wu

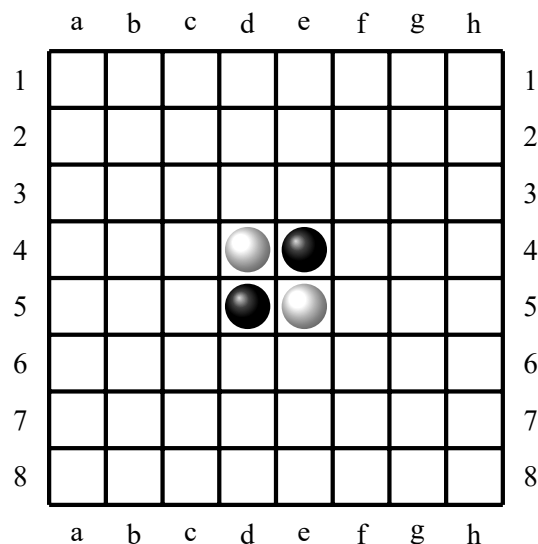
Goals of the project

- Implement a strategy board game **Reversi**
- Implement an AI (Artificial Intelligence) playing **Reversi**
- Compete with others

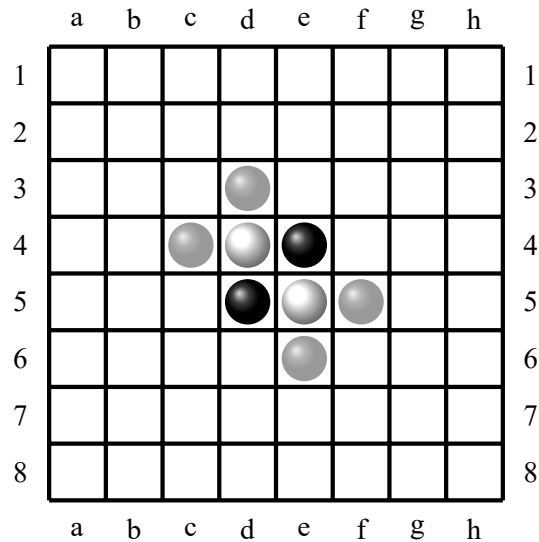
1 Introduction

1.1 Reversi Rules

The game begins with four disks placed in a square in the middle of the grid, two facing white side up, two pieces with the dark side up, with same-colored disks on a diagonal with each other. Convention has initial board position such that the disks with dark side up are to the north-east and south-west (from both players' perspectives), though this is only marginally meaningful to play (where opening memorization is an issue, some players may benefit from consistency on this). The dark player moves first.

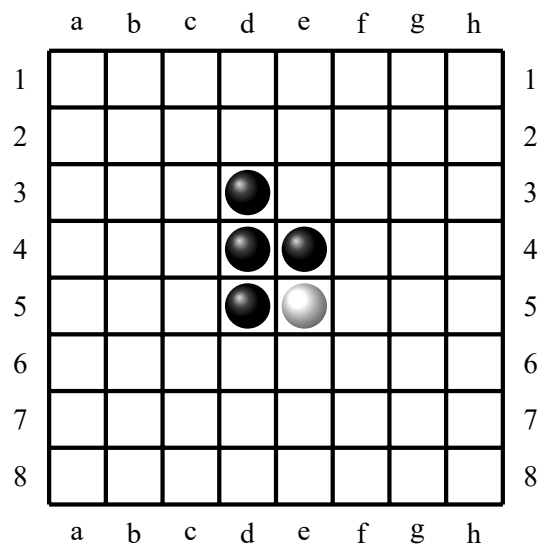


Dark must place a piece with the dark side up on the board, in such a position that there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another dark piece, with one or more contiguous light pieces between them. In the below situation, dark has the following options indicated by translucent pieces:

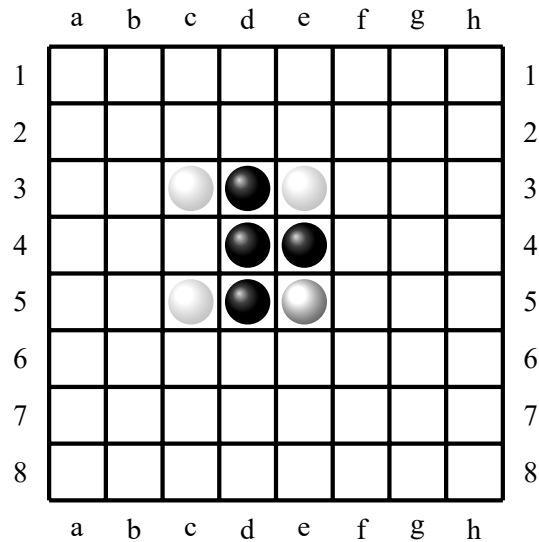


After placing the piece, dark turns over (flips, captures) all light pieces lying on a straight line between the new piece and any anchoring dark pieces. All reversed pieces now show the dark side, and dark can use them in later moves—unless light has reversed them back in the meantime. In other words, a valid move is one where at least one piece is reversed.

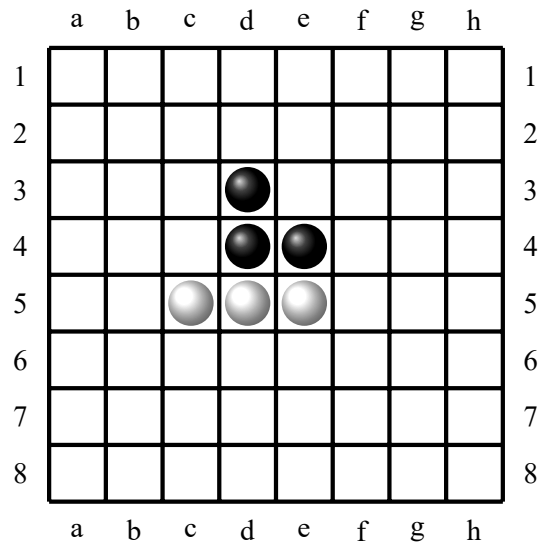
If dark decided to put a piece in the topmost location (all choices are strategically equivalent at this time), one piece gets turned over, so that the board appears thus:



Now light plays. This player operates under the same rules, with the roles reversed: light lays down a light piece, causing a dark piece to flip. Possibilities at this time appear thus (indicated by transparent pieces):






























































Light takes the bottom left option and reverses one piece:



Players take alternate turns. If one player cannot make a valid move, play passes back to the other player. When neither player can move, the game ends. This occurs when the grid has filled up or when neither player can legally place a piece in any of the remaining squares. This means the game may end before the grid is completely filled. This possibility may occur because one player has no pieces remaining on the board in that player's color. In over-the-board play this is generally scored as if the board were full (64–0).

Examples where the game ends before the grid is completely filled:

	a	b	c	d	e	f	g	h	
1									1
2									2
3									3
4									4
5									5
6									6
7									7
8									8
	a	b	c	d	e	f	g	h	

The player with the most pieces on the board at the end of the game wins.

2 Tasks

Design a **Reversi** AI according the rules stated above, and submit it to the online judge to compete with peers.

2.1 Compilation

The **Reversi** program is written in *C/C++*, the supported language levels are *C99*, *C11*, *C++98*, *C++11*, *C++14*, *C++17*, the supported compilers are *gcc* (*g++*) and *clang* (*clang++*). The program is able to compile and run on a *Linux* hosted server,

2.2 Application Programming Interface (API)

2.2.1 Standard Input

The standard input will always be tuples of commands and arguments, the format is

COMMAND <*ARG1*> <*ARG2*> <*ARG3*>

The first several commands are ensured to be:

```

1  START
2  PLACE d 5 (1|2)
3  PLACE c 5 (2|1)
```

```
4  PLACE e 4 (1|2) 5
PLACE d 4 (2|1) 6
DONE
```

In *START*, the board and AI are initialized; in *PLACE*, the AI will get the position and owner of the four pieces, the third argument is 1 or 2, where 1 means this is the AI's chess and 2 means this is the opponent's chess; After *DONE*, *OK* is outputted, and the initialization is completed and the game begins.

If the AI is the first player (black) to play, it will receive a command *BEGIN*, then it starts to play a chess (see the Standard Output section).

After it has played a chess, the opponent will play another one and the AI will receive a command *TURN X Y*, where *X* is an letter in a-h and *Y* is an integer between 1-8, which means the position of the opponent's chess. Then it should play a chess again (see the Standard Output section).

Note that the opponent may not be able to play a valid move, then the AI will receive a command *PASS*, and then it should also play a chess. If it is not able to play a valid move as well, the game will end in the next command.

When *END* is received, the program should exit with code 0.

2.2.2 Standard Output

Except receiving *DONE* and printing *OK*, the AI should not output anything other than the position of your next chess to standard output. When it cannot make a valid move, print *PASS* to the standard output.

All commands which need an output (*BEGIN* and *TURN*) must be responded in 1 second. The output position must be valid. Or the AI will directly lose the game.

The output format is *X Y*, where *X* is an letter in a-h and *Y* is an integer between 1-8, which represents the position the next chess is played.

2.2.3 Standard Error

The *Reversi* program also supports printing on standard error for debugging. Once a *BEGIN* or *TURN* is received, or a chess is played, a board of the played chess (arbitrary) and the available positions (optional) to play are printed on *stderr*.

2.2.4 Example

Black Input:

```
1  START
2  PLACE d 5 1
3  PLACE e 5 2
```

```

4  PLACE e 4 1
5  PLACE d 4 2
6  DONE
7  BEGIN
8  TURN e 5
9  ...
10 END

```

White Input:

```

1  START
2  PLACE d 5 2
3  PLACE e 5 1
4  PLACE e 4 2
5  PLACE d 4 1
6  DONE
7  TURN d 3
8  ...
9  END

```

Sample Board (X means black, O means white, ' ' means available positions):

```

1      a  b  c  d  e  f  g  h
2  +---+---+---+---+---+---+---+
3  1|  |  |  |  |  |  |  |
4  +---+---+---+---+---+---+---+
5  2|  |  |  |  |  |  |  |
6  +---+---+---+---+---+---+---+
7  3|  |  |  |  '  |  |  |  |
8  +---+---+---+---+---+---+---+
9  4|  |  |  '  |  0  |  X  |  |  |
10 +---+---+---+---+---+---+---+
11 5|  |  |  |  X  |  0  |  '  |  |  |
12 +---+---+---+---+---+---+---+
13 6|  |  |  |  |  '  |  |  |  |
14 +---+---+---+---+---+---+---+
15 7|  |  |  |  |  |  |  |  |
16 +---+---+---+---+---+---+---+
17 8|  |  |  |  |  |  |  |  |
18 +---+---+---+---+---+---+---+

```

3 Matching and Grading

3.1 Game

Each game is consisted of two rounds. Both players take the black side who serves first for one rounds for the sake of equality. Besides, four pieces are placed in the center in a standard diagonal pattern, rather than being placed by players. (See the introduction)

3.2 ELO Ranking

ELO Ranking is used as the basic ranking method which is commonly applied in many online competitive games. Every participant has an initial rank as 1500 which will be increased, decreased or kept unchanged when there's a win, lose or draw game, respectively. The amount of changed rank is dependent on the rank difference between the two participants of one game. The higher the rank difference, the more amount of rank will be changed.

3.3 Matching

The matching mechanism can viewed as such a model: the server keeps looking for two feasible participants as opponents to hold a game. There are several rules defining what kind of participants are feasible to hold a game:

- These two participants' rank difference is no larger than 100 except one of them has just updated the AI and is of lower rank.
- Both participants have submitted one effective AI program (effectiveness means this AI is able to play at least one feasible step).
- These two participants' win game difference is no larger than 1 unless the one with such advantage is of lower rank. For example, if A has a 3-1 winning relation with B, A and B will not be matched together to hold a game any more unless A's rank is lower than B's rank.

Since there may exist multiple feasible games to be hold, in order to make sure those who needs to know the their AI's combat results can get their results quickly, the following priority is designed for finding two players:

- The player who has just updated a new AI will be set with top priority.
- The player whose AI is under a win streak will be set with relatively high priority, ranked by length of streak.

