Othello Lab A
AI - Fall 2023

Othello / Reversi is a game played on an $n \times n$ board (you may assume $n = 8$) between two players (white and black or light and dark) where the four center squares (implying n is even) have a starting configuration of

light	dark
dark	light

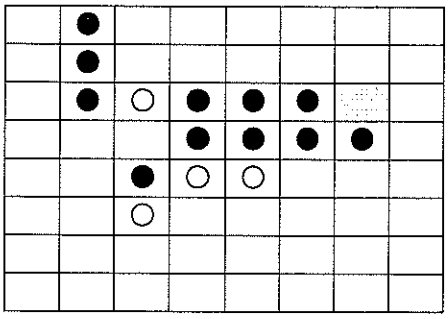
○	●
●	○

o	x
x	o

The dark color goes first. In the below, x, will correspond to a dark token and o will correspond to light tokens. Players alternate turns, and each player must move if possible. If a player in their turn has no legal move, then the other side will play again. Play continues until there are no legal moves for either side. The winner is the side with the most number of tokens on the board.

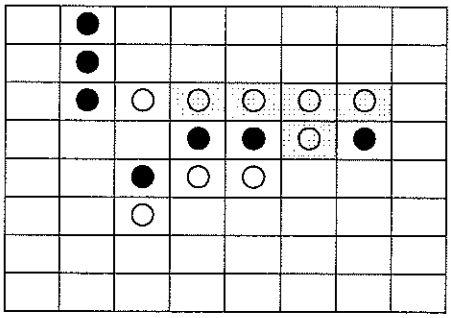
A move consists of a player placing one of hir tokens into an unoccupied square such that there is a straight horizontal, vertical, or diagonal line between it and a prior token of theirs between which there are only enemy tokens (at least one) and also no unoccupied squares.

If play is made into such a square, then all enemy tokens that are immediately bracketed between the new token and their already existing token (with only enemy tokens in between) are converted to that of the side playing the token.



If it is o's turn and the token is placed to the right of the rightmost token in the third row of the left image, then the tokens flip as indicated in the right image.

Note that tokens at 27 & 28 are not flipped.



Internally, the indexing should be the same way that it was for Sudoku: 0-63. However, it is also common to use a type of A1 notation (Excel terminology) meaning that columns are given labels of a-h, left to right, while rows are given labels of 1 to 8, top to bottom. Thus, the square that o plays into in the example above is g3.

Lab A – Development platform:

You should implement the basic architecture for Othello. This means that you should be able to
a) Determine what moves are possible, given a board and the token that is to move
b) Make a move, given a board, the token that is to move, and the new token's position
c) Print the board in 2D

You will write a script that takes a command line input in the form of:
`othello.py [board] [tokenToPlay] [move1 move2 …]`

All arguments are optional.
If provided, the board will be a length 64 string consisting of the symbols in `"xX.Oo"`. Case does not matter. If not provided, the board should default to `'.'*27+'OX......XO'+'.'*27`

The `tokenToPlay` is a symbol from `"xXOo"`. If only one side can move, it defaults to the side that can move. Otherwise, it defaults to the symbol that would play if there have been no passes in the play to this point. The dark symbol, `x`, moves first on a default board.

`[move1 move2 …]` is a sequence of moves, defaulting to the empty sequence. A move might be given in A1 notation (either capitalized or uncapitalized), but it will usually be given as an integer in [0,63]. If a move is a negative integer, it should be ignored. Excepting the occasional negative integer, only valid move sequences will be given. However, your code should not break if an invalid move is given. The reason for ignoring negative integers is that in some future labs, negative numbers will be used to convey certain status information (such as forced passes and timeouts).

Output:
Your code should output $n + 1$ snapshots where n is the length of the move sequence after any negative integers have been removed. A snapshot is the following information:

A) A move such as: `x plays to 25` or `o moves to 36`.
   This is omitted for the first snapshot as there has been no move yet.
B) A 2D representation of the board. Possible moves should be indicated with an asterisk
C) A blank line
D) A 1D representation of the board. Possible moves are not indicated and will have a period, just as any other unfilled position. To the right of the board will be the score as the number of x tokens on the board, a slash, followed by the number of o tokens.
E) Possible moves that can be made, such as
   `Possible moves for x: 10, 20, 26, 37, 42, 44`
   followed by a blank line.

If the game is over at this point, then E does not appear

The input sequence does not indicate that a pass happens. It is up to the script to detect that a side cannot move, so that the next move applies to the same side again.

An example follows:

```
>Othello.py 19 34 41
........
........
...*....
..*ox...
...xo*..
....*...
........
........

............................ox......xo.............................  2/2
Possible moves for x: 19, 26, 37, 44

x plays to 19
........
........
..*X*...
...xx...
..*xo...
........
........
........

.....................x.......xx......xo.............................  4/1
Possible moves for o: 18, 20, 34

o plays to 34
........
........
...x....
...xx...
..Ooo...
.*****..
........
........

.....................x.......xx.....ooo.............................  3/3
Possible moves for x: 41, 42, 43, 44, 45

x plays to 41
........
...*....
..*x**..
...xx...
.*xoo...
.X......
........
........

.....................x.......xx.....xoo....x.....................  5/2
Possible moves for o: 11, 18, 20, 21, 33
```

This lab (Lab A) is vital as almost →**all**← of the labs in January build off this one. You approach this lab in three specific steps, with labs Othello 1, Othello 2, and Othello 3. Othello 3 is Othello Lab A

Othello 1: `othello.py [board] [tokenToPlay]`

First, have a function that displays an Othello board in 2D.

Second, and most importantly, construct a **function** that takes a board and returns the set of all possible moves. This is a workhorse function, meaning that it must eventually become efficient. Expect that you will make revisions to it, so it must be modular (ie. be a function). For now, though, you should be concerned about correctness. As long as it's correct, you won't be able to make it so slow that it affects this lab.

Update your 2D display func so it shows asterisks instead of dots in the positions of possible moves.

Finish printing out B-E of the snapshot by printing out the possible moves or a "No moves possible" phrase as needed.

By the time you pass all 100 tests, your possible moves function should be in good shape!

Othello 2: `othello.py [board] [tokenToPlay] [singleMove]`

In addition to the two possible inputs, there is an (optional) 3rd input, that of a move.

You'll want to add a new function to your script to make a move, returning the updated board.

The default token is the token whose turn it would be if there had been no passes, unless only one side can move, in which case the default token is for that side.

You should now add a printout of an initial snapshot (B-E) followed by a second snapshot (A-E) assuming that a move has been given.

Othello 3: `othello.py [board] [tokenToPlay] [move1 move2 …]`

This is the generalization of Othello 2, where a sequence of moves may be given, and not just a single one. After an initial snapshot of B-E, have one snapshot per non-negative move. You should be able to slap what you did for Othello 2 into a loop and be all set.

You should now have a very solid development platform for the next Othello lab and January's AI efforts.