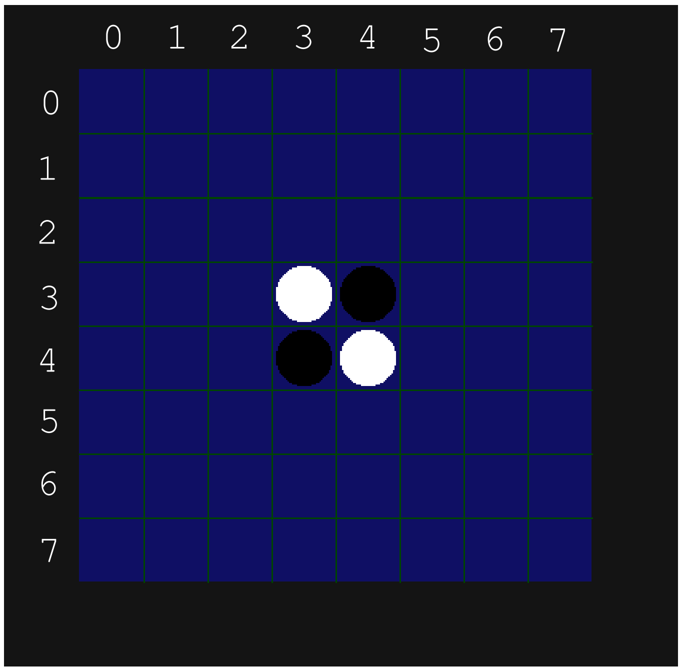
CPSC 231 – Fall 2012, L01/L03
Assignment 5: The Reversi Game

Due: *Friday December 07 2012, 4:00PM*

This is the final assignment, and you are free to use all Python skills that you have accumulated, including in particular conditionals, iteration, functions, lists and objects.

The Game Reversi



Reversi is a 2-player board game with the initial board configuration shown above. The white and black players take turns to make a move, by placing a new piece of stone on the board. If the new stone, together with another existing stone of the same color, encloses one or more stones of the opposite color, then the move is valid and the enclosed stones are 'captured' and change in color. A player can make only valid moves. If one player runs out of valid moves, the turn is passed to the other player. The game ends when either the board is full or both players run out of valid moves. The player with a larger number of stones wins the game. A tie is possible.

For example, in the beginning of the game, white has 4 valid moves at [2,4], [3,5], [4,2] and [5,3]. If you want to read a more detailed introduction on the rules of Reversi, it is available at:

`http://en.wikipedia.org/wiki/Reversi`

Implementing the Reversi Game

In this assignment, you will write (part of) a Python program that implements the Reversi game, so that a human player can play with the computer. The program draws the board and stones using QuickDraw, and takes human input in the form of mouse clicks in the QuickDraw window.

You are provided with two files: *reversi.py* and *game.py*. The file *reversi.py* is complete and you do not need to modify it. It implements (a) the high level control of the game play (e.g., getting moves from the computer player and the human player, determining whether a player runs out of moves, determining the end of the game and declare the result of the game), (b) the rendering of the game through QuickDraw, as well as (c) taking the input from the human player.

reversi.py imports definitions from *game.py*. *game.py* is incomplete and your task is to fill in the appropriate details in the definition of a *Game* class. More details on exactly what you need to do are given in the next section. A *Game* object is created in *reversi.py*, and is central to the implementation of the game.

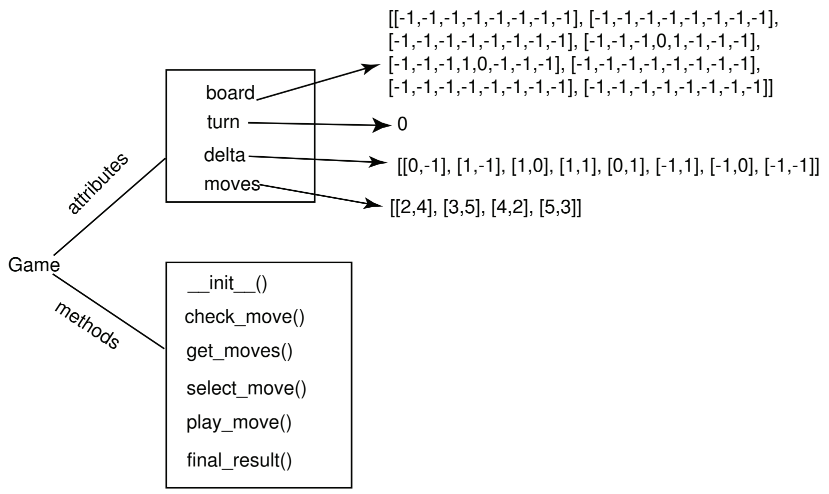
To run the program, type the following on the command line:

```
$ python reversi.py
```

You need to have *reversi.py*, *game.py* and *quickdraw.jar* all in the current directory. The communication between the python program and Quickdraw is now handled using a bidrectional pipe created within *reversi.py*.

Completing *game.py*

The file *game.py* contains mainly the definition of the *Game* class, which has a number of attributes and methods, as shown in the figure.

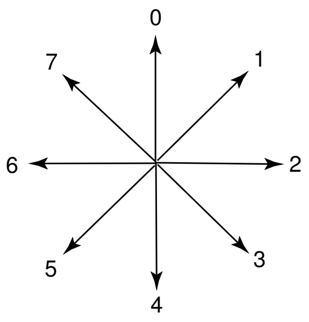


The attribute *board* is a 2-D list that stores the status of each grid on the 8×8 board: −1 means empty, 0 means there is a white stone and 1 means there is a black stone.

The attribute *turn* denotes the current turn of the game, *i.e.*, which player is to make a move next. Here again 0 means white and 1 means black.

The attribute *delta* is a constant list that stores the change in coordinates when moving along one of the eight possible directions on the board. The eight directions and their numbering are shown in the figure below. This attribute may be convenient to use when you are implementing the *check_move()* and *play_move()* methods.

The attribute *moves* is a list of moves that are valid, given the current board configuration and the current turn of play.



The *__init__(self, board, turn)* method creates a *Game* instance, with a given board configuration and a specified turn. It initializes the *moves* attribute appropriately during the creation.

The *check_move(self, move, d)* method checks whether a given *move* is validated along a certain direction *d*, assuming the given *turn* of play. It returns *True* or *False* accordingly. In other words, if starting at the grid specified by *move*, a number of opponent stones along the direction *d* can be captured, then a *True* should be returned. This method may be convenient to use in other methods such as *get_moves()* and *play_move()*.

The *get_moves(self)* method finds all the valid moves for the current *turn*, and returns them in a list. For example, if there are two moves at [2,4] and [3,6], then the return value should be [[2,4], [3,6]].

The *select_move(self)* method picks a move from *self.move* and returns it. The easiest way is to return a random move. If you wish, you can try to identify a better than random move and return that instead.

The *play_move(self, move)* method makes a move specified in the *move* argument in the form of [x, y]. You need to update *self.board*, *self.turn*, and *self.moves*.

The *final_result(self)* method examines *self.board* and returns a tuple of two numbers: the number of white stones and the number of black stones (in that order).

Your task in this assignment is to fill in the details in the definition of *check_move()*, *get_moves()*, and *selelct_move()*. In realworld software development, you are often programming among a team. You are responsible to write parts/modules of the entire program, with clearly defined input format and desired output — similar to what you are doing in this assignment.

Bonus

What can you do in the `select_move()` method? While randomly selecting a move among all current valid moves is an easy approach, much more sophisticated strategies are possible, for implementing various heuristics that improve the strength of your program. If you receive an `A` (4.0) in this assignment during the first stage of marking, your TA will run your AI against one that chooses moves at random. A W:L ratio above 1.75 will warrant an `A+` (4.3) for this assignment.

Submission

Submit your solution by sending an email to your TA, with your solution program in the attachment. Name your program in the form of `game-sid.py`. For example, if your student ID is 123456, then name your program `game-123456.py`.

Submissions received after the deadline will not be accepted.

Collaboration

Although it can be helpful to you to discuss your program with other people, and that is a reasonable thing to do and a good way to learn, the program you submit must ultimately be *your own work* that has been written by you independently.

All submissions from the class will be checked by `MOSS`, a computer program that identifies similar programs. Remember, if you are having trouble with an assignment, it is always possible to go to your TA and instructor to get help.