Software Engineering Project 1

(Comp 10050)

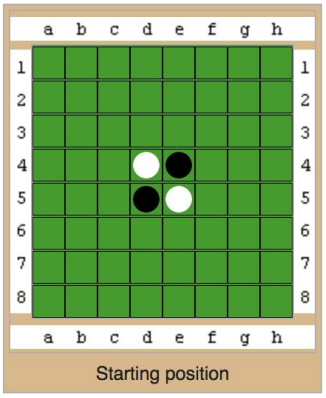
Assignment 3 – the Othello/Reversi Game

Aim: to create a program that implements the Othello game (https://en.wikipedia.org/wiki/Reversi)

You are expected to perform this assignment collaboratively in a group

A. Game Logic

A.1 Start condition: The board looks like that shown in the figure below. This reflects exactly the outcome of the code provided on moodle.



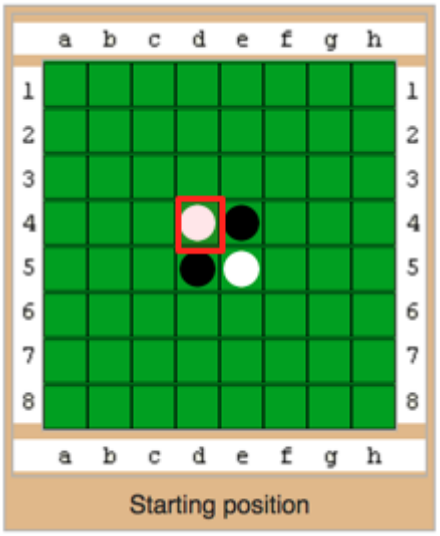
A.2 Managing turns: The player associated with the black disk starts the game. For each player turn it is necessary to do the following steps.

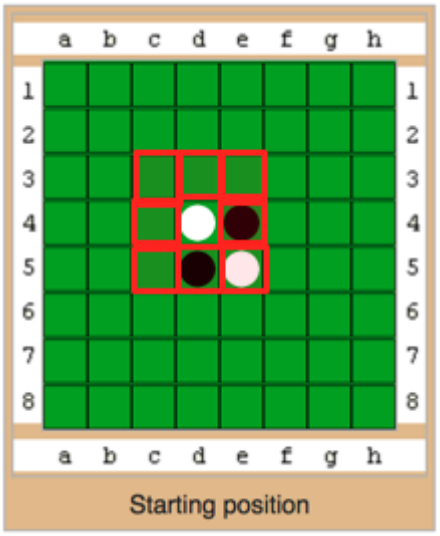
- Compute the positions where the player can move
- Ask the player from the standard input output where s/he wants to place his/her disk allowing him/her to select among the positions computed in the previous step.
- After the player decides the position, place a disk of the same colour assigned to the player on the board and change the colour of disks of the adversary player if necessary.

A.2.1 Computing positions where the player can move

- Identify all the positions in which there is an adversary disk. For example, in turn 1, the current player is the BLACK and the adversary player is the WHITE. WHITE disks are placed at positions (4,d) and (5,e).

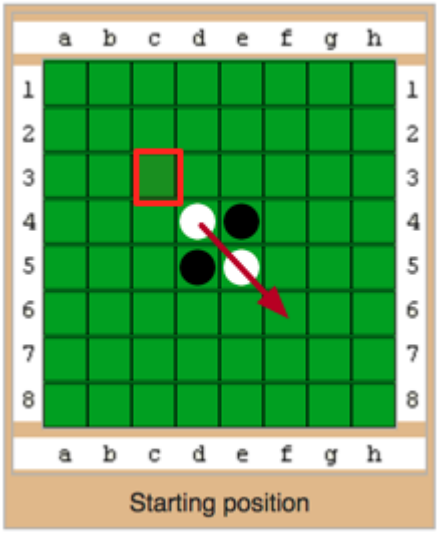
- For each position on the board where there is an adversary disk, consider all the adjacent cells.
  For example, if we consider position (4,d), it is possible to find adjacent cells in 6 different directions: up (3,d), up-right (3,e), right (4,e), down-right (5,e), down (5,d), down-left (5,c), left (4,c) and up-left (3,c). See figures below, indicating the adversary's disk position considered (on the left) and all the adjacent cells (on the right)

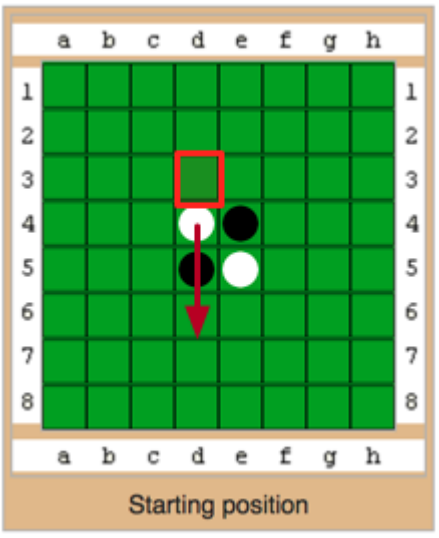




- Considering <u>each empty adjacent cell</u> it is necessary to understand whether it is possible to form a sequence of the adversary's disks terminated by a disk of the current player in the opposite direction.

For example, if we consider adjacent cell up-left (3,c), it is not possible to form a sequence of WHITE disks terminated by a BLACK disks in direction down-right (See Figure on the left).
However, if we consider adjacent cell up (3,d), it is possible to form a sequence of WHITE disk(s) terminated by a BLACK disk in direction down (See Figure on the right).





Therefore Position (3,d) can be added to the set of possible positions.
Applying this mechanism for all the adjacent cells of the adversary cells, it is possible to obtain the following set of possible positions (3,d) and (4,c).

If we apply the same process, for the adversary WHITE disk placed in (5,e), the possible positions we can identify are (5,f) and (6,e)

Note that the same possible positions can be discovered when considering different adversary's disk. So please make sure you don't have duplicates!.

A.2.2 Asking the player where s/he wants to move

Once the positions are computed the current player will be asked to select among one of them. For our previous example the program should show the following on the standard output.

Player <player_name> choose you next move:
1. (3,d) 2. (4,c) 3. (5,f) 4. (6,e)

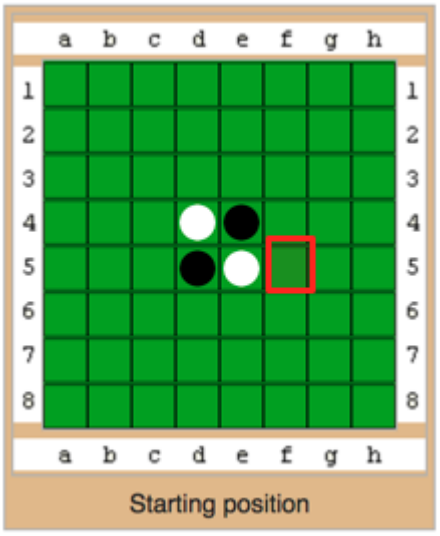
The user can subsequently input the desired move providing as input the number corresponding to that position.

A.2.3 Disks placement

Once a specific position was selected, for example it is necessary to identify all adjacent cells associated with the selected one.

For example, if we consider position (5,f) (see Figure below), it is possible to find adjacent cells in 6 different directions: up (4,f), up-right (4,g), right (5,g), down-right

(6,g), down (6,f), down-left (6,e), left (5,e), and up-left (4,e). See figures below, indicating the adversary's disk position considered (on the left) and all the adjacent cells (on the right)



- Considering <u>each empty adjacent cell</u> it is necessary to understand whether it is possible to form a sequence of the adversary's disks terminated by a disk of the current player in the opposite direction. If this condition is satisfied, all the adversaries' disks will have to change colour.

  In our example the WHITE disk at position (5,e) will change colour

- After disk colours are changed the board should be reprinted.

Termination condition: The game can end due to 2 situations:

1. 64 turns are played
2. None of the 2 players can place his/her disk anywhere.

A.3 Printing the final result:

Player1 <player_name>, points: <points_player1>
Player2 <player_name>, points: <points_player2>

The winner is <name_of_winning_player>

The same content should be saved in a text file.

B. Requirements

1. Create a Git project for your third assignment (1 project for each team).
   Please, create a new Git repository for this assignment

2. Implement the full logic of the game. Your main function should invoke 3 main methods
   - A method that initializes the players and the board (provided on moodle);
   - A method that supports the game logic.
   - A method that prints the final result of the game:
     - name and points of each player
     - who is the winner
   - The final result of the game should also be saved in a text file.

Code Design Requirements:

- Comment your code.
- Separate your code into independent modules and avoid creating big methods

Submission:

- Add me as a collaborator on your GitLab project (use my username liliana.pasquale)
- In your repository include a text file describing
  - How you decided to implement the different methods used to implement the game logic
  - How did you divide the work (e.g., who did what)?
- Make sure you fill the questionnaire to evaluate your teammate after the deadline https://goo.gl/forms/tcBwKAsjt74EV95q1
  - Note that the results of the questionnaire will be kept confidential

Evaluation Criteria

A mark [0-100] will be give according to the following criteria

- The code is well commented and appropriately divided into modules (10 points)
- Work is appropriately distributed and placed in the Git repository (10 points)
- Turns management (10 points)
- Compute where players can move (25)
- Ask players where to move (5)
- Disk placement (25)
- Compute, print and save the final game result (15)