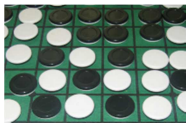
FINAL ASSIGNMENT (#7) –DESCRIPTION, INCLUDING SUBMISSION INSTRUCTIONS



READ ALL THIS DOCUMENT

This is an up to 3 people team work (you can also work in a team of 2 people or individually if you prefer so). While you may discuss generalities with colleagues from other teams about this exercise, you cannot develop the same code, nor share code among teams, nor obtain code from other sources.

Being a team exercise it places a big responsibility on each individual. You want to respect and be honest with your partners and with yourself: DO YOUR SHARE, and BE KNOWLEADGEABLE OF THE WHOLE ASSIGNMENT. Working on this assignment is also a way for you to prepare for the final exam.

START WORKING ON THIS AS SOON AS POSSIBLE

Two Submission deadlines:

I) *First part:* ***Wed Aug 3,11:59 pm*** CodeWrite exercises. Join Codewrite **course #25**. *The solutions that will be posted may be used in the final code, identical or variations.*

II) *Final Deadline:* ***Mon Aug 8, 8:00 AM*** . "Flipping dance" complete game Python code; top level flowchart; and admin file, via Canvas.

Note: You certainly can submit before the deadlines. Solutions will be posted soon after the corresponding deadlines. No extensions will be possible. Last day of classes: Aug 4. Review optional class time and room to be announced: Aug 8.

I) GENERAL DESCRIPTION OF GAME "Flipping Dance"

You are asked to implement a board game called "Flipping Dance", which the user plays against the computer. Your program should allow the user to play many games as long as the user wants to keep playing.

BASIC VERSION OF THE GAME, level 1, general idea:

In the basic version of the game (level 1) , the program first generates an initial board which is a square matrix with 0's and 1's given a dimension as requested by the user. You can imagine the board filled with pieces that are black on one side and white on the other, to fix ideas we will assume that 1 stands for black and 0 for white. The user will have to choose a color (black or white). (At this game level the color choice only affects the final results of the game). See Fig 0.

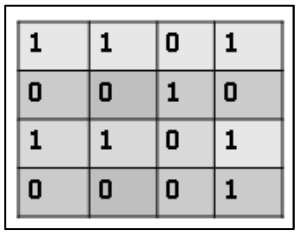


Fig 0. Possible initial board, 4x4

Then the game gives the user the opportunity to "play" by selecting **any** piece (i.e. any position in the board) and flip that piece (i.e. change the value: from 0 to 1 or from 1 to 0). This is the only change that is performed at this game level. Then the user would be invited to do more flips, and after the user did all the flips that he /she wants to do (up to a maximum allowed), the game is over and the program determines whether the user or "the computer" wins, as explained below.

The user is then asked if he/she wants to play again and a new game starts, at the same level or possibly another level, with a new randomly filled board and another possible choice of color for the user. When the user does not want to play any more games, the program shows totals for all the games (as described below).

ADVANCED VERSION OF THE GAME, level 2, general idea.
The advanced version (level 2) adds to the basic level 1 version two aspects:

a) The flip of one position that the user does affects also other positions in the board (as described below, for this level)
b) The computer will also have the opportunity to "play" by flipping **any** piece (chosen randomly) and correspondingly affecting the board (in the same way as it is affected when the player plays). The computer will play each time and immediately after the user plays.

EXPERT VERSION OF THE GAME, level 3, general idea (for bonus points)
The most advanced version (level 3) changes the rules of the game as follows:

a) The user will only be able to choose a piece of his/her own color (i.e. the color chosen at the beginning of the game) and the computer will still choose randomly a piece, but analogously, only a piece of it's own color (i.e. the color not chosen by the user).
b) The board will be affected differently than in the previous level.

You are recommended to first implement level 1, and then incorporate level 2 (and possibly level 3 if you have a program working for the two levels 1 and 2).No graphical user interface is required ; all is expected to be done with a text interface.

A the start of the whole program inform what game levels are available. If more than one level may be played, for each game, provide the user the possibility to choose a level to play.

II) MORE DETAILED DESCRIPTION OF ONE GAME:

Recall that the basic version (level 1) would not include the part when the computer chooses a position (piece) in the board. As additional description of the game also see the sample runs.

Starting a game

One game starts with the user being asked the dimension (dim) of the board (between 3 or 6, extremes included). Then the program should create a square matrix of dimension dim x dim, with randomly generated 0's and 1's. The board is then shown to the user and the user is asked his/her color. This color will remain the same during the game (i.e. during the various flips). This color only affects the final points for game levels 1 and 2.

Development of one game

The game unfolds by asking the user if he/she wants to "choose" some position in the board (up to a maximum number of times). This maximum times that the user may choose a piece is the value dim asked to the user, i.e. if the dimension of the board is 4x4 then the user may make up to 4 choices.

The changes and flips caused by choosing a piece (i.e. a position) varies according to the game level. See details below by level.

After the changes are made in the board, the board is shown again.
If the game is played at one of the advanced levels, after the user chose one position and the board is changed and shown, then the computer would make its move, and the user will be shown which position the computer chose and the board with the changes caused by the computer's move.

End of one game – winning one game

After all the flips and changes in one game took place (the user does not want to choose a piece again or the maximum changes was reached), the program should indicate whether the user or the computer wins, according to the following rules:

The user wins if the board has at least one row AND at least one column with an even number of his/her own color pieces. The user wins as many points as the number of rows and columns have an even number of his color. Correspondingly, if there are no rows and no columns with an even number of the user color pieces, the computer wins the game.

End of all games and board value

After no more games are played (i.e. the user does not want to play anymore) the program should show the total games played, the total games that the user won and the total points that the user won. As well a board value should be shown. This board value will be calculated based on the last state of the board in the last game played. The board value is calculated considering the first row of the board as a binary number (leftmost digit is the most significant digit) and converting it to a decimal number.

How a piece choice affects the board

At level 1

Only the chosen piece by the user is flipped (changed for 0 to 1 or from 1 to 0) (the computer does not play at this level. See Figure 1.

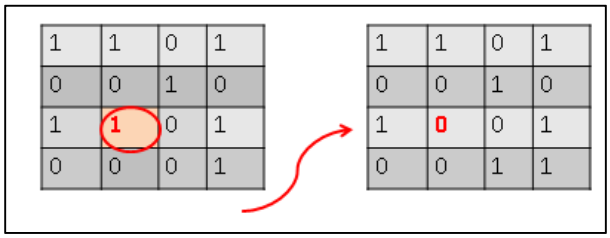


Fig 1. Flipping at level 1 of the game.

At level 2

The chosen piece by the user (or the computer) is flipped , and then, all the pieces in the same column and in the same row, from the selected position and until the border, increasing the column (i.e. along the row towards the right) and increasing the row number, (i.e. along the column and going down), should be flipped. See Fig 2.

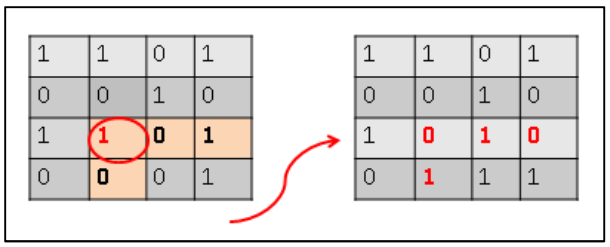


Fig 2. Flipping at level 2 of the game.

At level 3

The chosen piece by the user is flipped, and additionally up to four more positions will be flipped: the four positions vertically and horizontally surrounding the chosen position. Less than five positions may be flipped in total if the chosen position was in the board's border. At least one position will be flipped (the chosen one). See Figure 3.

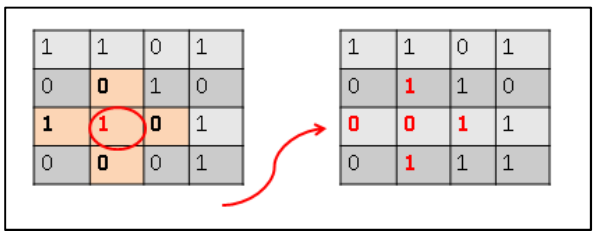


Fig 3. Flipping at level 3 of the game

III) Examples – Sample runs.

See the sample runs. The sample runs show various games, played at different levels.

IV) Requirement – Validating the input that the user provides

a. Validate that the user types digits when asked to do so , and that a row number or column number are within the matrix dimension.

V) Requirements – coding style

a. Your program should have at least 6 "fruitful functions", i.e. a function returning a value , and which is correctly used when the function is called
b. Your program should have at least 3 "void functions" (or subroutines, i.e. a function not returning any value but for example just printing)
c. Your program should have at least 6 functions receiving parameters (and so that they are correctly used inside the function)
d. Your program should have a reasonable top level (or main function) which shows the general structure of the program
e. You may use the some variables as global (i.e. defined at the top level and not passed as parameters). The reason to have these functions as global would be that they are really used by many functions. Yet, given the requirements above you will have to be variables in functions that are not global variables.
f. All the global variables used in the program should be initialized at the top of the code (even before the functions are defined) including a brief comment of what the variable is about. No comments are needed if the names of the variables are self-explanatory.
g. Name your variables and functions appropriately
h. Include as comment the authors names and dates of the versions at the top of the file
i. Include comments, with general descriptions of functions, special situations being true at a certain place in the program, etc. On the other hand, do not include redundant comments. For example the statement "i = i + 1" does not need the comment "i is increased by 1". Keep in mind that good naming of variables and functions reduce the need of comments.

VI) Requirement – extra clarification file (txt)

a. When there is a group of 2 or 3 working in the problem you need to clarify how you distributed tasks among the team members. People working individually should just create an additional file with the name workedAlone.txt. In this latter case the file may be empty or it may include any comments you may want to pass to the marker.
b. It would be useful for you if you keep track of the time you spend on this exercise. You are asked but not required to share this information
c. Include in this file any special clarification you may want to add

VII) Requirement – Flowchart for top level.

a. Submit a flowchart describing the only the general/top level and the main global variables. You may use flowgorithm or draw it by hand and take a picture/capture the screen and submit a jpg, png file.

VIII) Recommendations

a. Understand the game before implementing it! Check the sample runs, ask the teaching staff if you are not sure about the game rules.

b. **Codewrite:** The functions that you wrote in Codewrite for the first part of this assignment should be useful for the game implementation, perhaps not directly but with some variations, but keep them in mind. You may use your own solutions to these functions or the posted solutions and/or variations as needed.

c. *Hint*: You can start developing the whole code and top level and leave some functions without being completely defined, but just returning a realistic value and possibly some trace print to remind you, so that it allows you to advance and test the whole code, and you can later or in stages completely define the function/s. (This is regular practice when developing code).

d. You are recommended to start by developing your code to implement the level 1 of the game. After this is working for you, enhance your code to include level 2. Once your level 2 is working you may want to aim to implement level 3. Implementing the level 3 is not required and will give you bonus points.

e. Include "Trace printing" as you debug your code. When you submit your solution you may keep these additional printing messages or leave in the code but commented out.

f. *Hint*: You may want to create auxiliary variables to aid in your calculations. For example, you may have several accumulators which are increased as each game is played, such as how many games the user won, etc, and at the end of all games you can print these contents.

g. *Hint*: You may want to define some intermediate lists to aid your calculations. For example you may create and calculate a single list for rows where you include a value 1 in position p if the row number p has an even number of a certain digit (1 or 0 as it suits your needs) and analogous for a column. These lists would then be useful to calculate your points for a game.

If you have any questions consult with the Teaching Team.

End of description of the final assignment.