

Assignment 3

Connect Four (Lists of Lists)



Due: Friday June 4th at 2:00pm

Submit a single zip file called **A3.zip**.

The assignment has 10 marks.

Notes: It is essential that you use the built-in, default, archiving program to create this zip file. If we cannot easily open your zip file and extract the python files from it then we cannot grade your assignment. Other file formats, such as rar, 7zip, etc, will not be accepted.

Windows: Highlight (select with ctrl-click) all of your files for submission. Right-click and select "Send to" and then "Compressed (zipped) folder". Change the name of the new folder "A3.zip".

MacOS: Highlight (select with shift-click) all of your files for submission in Finder. Right-click on one of the files and select "compress N items..." where N is the number of files you have selected. Rename the "Archive.zip" file "A3.zip".

Linux: use the zip program.



After submitting your A3.zip file to brightspace, be sure that you download it and then unzip it to be certain that what you have submitted is what you wanted to submit. This also checks that your zip file is not corrupted and can be unzipped.

Please note that reasons similar in nature to "*I submitted the wrong files*" or "*I didn't know the zip file was corrupt*" will not be accepted as an excuse after the due date.

Submit early and often. Brightspace will save your latest submission. I would highly suggest that you submit as soon as you have one question done and keep re-submitting each time you add another problem (or partial problem).

In this assignment, you will implement the game connect four. If you are not familiar with the game, please see https://en.wikipedia.org/wiki/Connect_Four

Briefly, connect four is a two-player game where players (red and black) alternate taking turns (making moves). The game is played with a 2-dimensional grid (typically 6 rows and 7 columns) and a player takes a turn by “dropping” a checker in a column that has a free spot in it. The checker drops down as far as it can (stopping when it reaches lowest free space available in that column; being stopped by either another checker or the bottom of the game). A player wins the game if they can position four of their checkers consecutively in either a row, a column or diagonally. If the entire grid is filled up and there is no winner then the game ends in a tie.

You are NOT allowed to import any modules in your connect4.py file. In your c4game.py program, you will import your connect4 module but are NOT allowed to import anything else.

P1: The Game Board (Grid) [7 marks]

Save all your functions from this part in a file called **connect4.py**.

The game will take place in a 2-dimensional grid. A typical game has 6 rows and 7 columns giving 42 possible places, which we will call **locations**, for the pieces, called **checkers**, to be and a maximum of 42 moves (21 for each player). You will use a 2-dimensional list (list of lists) to represent the game grid. The inner lists will store strings. Each string must be one of 'red', 'black' or 'empty'. If `grid[2][4] == 'red'`, then a red checker is row 2 and column 4. Columns and rows are labeled starting with ZERO, so `grid[2][4]` corresponds to the 3rd row and the 5th column.

column 0 is the left-most column of the game and **row 0 is the top-most row** of the game.

Make a function called `makeGrid(nRows, nCols)` that takes two integers as input and outputs (returns) a 2-dimensional list that is an empty game consisting of `nRows` rows and `nCols` columns. All `nCols` strings in each row must be the string 'empty'. For example,

```
>>> makeGrid(5,4)
[ ['empty', 'empty', 'empty', 'empty'], ['empty', 'empty', 'empty', 'empty'],
  ['empty', 'empty', 'empty', 'empty'], ['empty', 'empty', 'empty', 'empty'],
  ['empty', 'empty', 'empty', 'empty'] ]
```

The function will always be called with inputs satisfying $4 \leq \text{nRows} \leq 10$ and $4 \leq \text{nCols} \leq 10$.

Next, make a boolean function called **play(grid, column, checker)**. The function tries to play the checker (either 'red' or 'black') in the specified column of the grid. If column is valid (i.e., it is in the right range) and there is room to play another checker in that column of the grid, then the function should modify the grid to add the checker in the given column and return **True**. Otherwise, it returns **False**. For example,

```
>>> grid = makeGrid(4,4)
>>> play(grid, 1, 'red')
True
>>> play(grid, 5, 'red')
False
>>> print(grid)
[ ['empty', 'empty', 'empty', 'empty'], ['empty', 'empty', 'empty', 'empty'],
  ['empty', 'empty', 'empty', 'empty'], ['empty', 'red', 'empty', 'empty'] ]
```

Next, make a function called **win(grid, column)** that returns a string. The function checks if a player has won the game (four checkers of the same colour in a row, column or diagonal) or not. The specified column is the last column in which the piece was played in the game (this should make it easier to check if that last play was a winning play). If a player has won the game then the function returns the checker name ('red' or 'black') that won. Otherwise, it returns 'empty'. To make the function more robust, it should also return 'empty' if the input column is out of the valid range of columns or if there is no piece played in the specified column.

Next, make a function called **toString(grid)** that returns a string representation of the game.

The checker 'red' will be represented by an 'X', the checker 'black' will be represented by an 'O' and empty locations will be represented by a single space (" "). The string must contain newline characters, border characters ('|' pipes, '-' dashes and '+' pluses) and labels (numbering) of the rows and columns. The format of the output string should follow this example:

```
>>> grid = makeGrid(4,5)
>>> play(grid, 1, 'red')
True
>>> play(grid, 1, 'black')
True
>>> play(grid, 3, 'red')
True
```

```
>>> print( toString(grid) )
|      |0
|      |1
| 0    |2
| X X  |3
+-----+
01234
```

Put all FOUR functions in a file called **connect4.py**.

P2: Connect Four Game [2 marks]

Write a program (in a file called **c4game.py**) that lets two players play a game of connect four.

Program

Your program will be driven by a **main()** function. Be sure to include a main guard (if statement) in your file. The program will proceed as follows:

1. The users are asked for the size of the game to play in a single question. The expected input should something like "4,6" to play a game with 4 rows and 6 columns. There can be any amount of whitespace around the numbers and the comma when the user enters this. So "4, 6", "4 , 6" and "4,6" are all valid. If the users enter 'quit' then the program ends with a parting message like "Thanks for playing". The user will NEVER enter anything other a valid row,column combination or quit.
2. The program checks if there are empty locations in the game grid. If the grid is full, it outputs a message "the game is a tie" and then repeated step 1. If there is at least one empty location, then proceed to step 3.
3. The program asks 'red' which column to play (remember column labels start with 0) and then tries to play a red checker in the given column.
 - a. If this is successful, the grid is shown and then the program checks if the last move was a winning move. If it was a winning move the game ends with an appropriate message ('Red wins the game after M moves') and the program goes back to step 1. If was not a winning move, the program proceeds to step 4.
 - b. If this is unsuccessful, a message is displayed (that it was an invalid move) and we retry step 3.
 - c. If the user entered 'quit' then the program terminates with a parting message like "Thanks for playing".

4. The program checks if there are empty locations in the game grid. If the grid is full, it outputs a message "The game is a tie" and then repeated step 1. If there is at least one empty location, then proceed to step 5.
5. The program asks 'black' which column to play (remember column labels start with 0) and then tries to play a black checker in the given column.
 - a. If this is successful, the grid is shown and then the program checks if the last move was a winning move. If it was a winning move the game ends with an appropriate message ('Black wins the game after M moves') and the program goes back to step 1. If it was not a winning move, we proceed to step 6.
 - b. If this is unsuccessful, a message is displayed (that it was an invalid move) and we retry step 5.
 - c. If the users enter 'quit' then the program terminates with a parting message like "Thanks for playing".
6. Go back and repeat step 2.

In the above description, M is the total number of valid moves that have been played in the given game. The minimum number that this can be for a valid game is 7 (when red wins after its first 4 moves).

A sample run of the game is as follows: (User input is shown highlighted light yellow; your game will NOT show this highlighting; it is just there for illustrative purposes).

```
Please enter the size of the game you want to play: 4, 5

Where does red (X) want to play? 4
|      |0
|      |1
|      |2
|      X|3
+-----+
01234

Where does black (O) want to play? 4
|      |0
|      |1
|      O|2
|      X|3
+-----+
01234

Where does red (X) want to play? 7
That is not a valid move.
```

Where does red (X) want to play? 1

```
|      |0
|      |1
|      O|2
| X  X|3
+-----+
01234
```

Where does black (O) want to play? 4

```
|      |0
|      O|1
|      O|2
| X  X|3
+-----+
01234
```

Where does red (X) want to play? 2

```
|      |0
|      O|1
|      O|2
| XX X|3
+-----+
01234
```

Where does black (O) want to play? 0

```
|      |0
|      O|1
|      O|2
|OXX X|3
+-----+
01234
```

Where does red (X) want to play? 3

```
|      |0
|      O|1
|      O|2
|OXXXX|3
+-----+
01234
```

Red wins after 7 moves

Please enter the size of the game you want to play: quit

Thanks for playing

Recap [A3.zip]

Submit a single zip file called **A3.zip**. Your zip file should have TWO files in it.

- **connect4.py**
- **c4game.py**