PARIS PANTHÉON-ASSAS UNIVERSITÉ

**Team n°1**
**2022**

# Tetris Puzzle

**By Adèle Chamoux & Iriantsoa Rasoloarivalona**

# Table of content

I. Introduction (p. 2)

II. The finished project : necessary & practical functions (p. 3 - 5)

III. How do these functions all cooperate ? (p. 6 - 7)

IV. Results presentation (p.7 - 9)

V. Conclusion (p. 10)

# I. Introduction

For this project, we were asked to code a game similar to the famous Tetris game that dates back to the 1980's.

Only here, the objective was using Python, and playing it only by executing the codes in the console !

With the use of text files (.txt), many functions distributed over 3 different python files (.py), and especially the use of 1D and 2D lists, it was possible to create such a game.

# II. The finished project : functions

## a. Mandatory functions

**Key Map :** **parameters** / **output**

**IN GRID.PY:**

- *functions filling text files with 0s & 1s :* **grid_triangle, grid_diamond & grid circle (size)**
  **:size: size of the board given by the user → string / no return (data modified)**

- *a function reading the file of the grid chosen by the user :* **read_grid(path)**
  **:path : name of the grid chosen by the user → string**
  **:return : the data of the file (the 0s and 1s) in a matrix M of strings → 2D list**

- *a function that saves the grid in a file specified by path:* **save_grid(path, grid)**
  **:path : name of the grid chosen by the user → string**
  **:grid : the matrix M of the grid → a 2D list**
  **:return : None (data modified)**

- *a function which display the status of the grid in ASCII symbols:* **print_grid(grid)**
  **:grid : the matrix M of the grid → 2D list**
  **:return : None (only prints)**

- *a function that checks if the column j (resp. row i) in grid is full:* **col_state(grid, j) [resp. row_state(grid, i)]**
  **:grid : the matrix M of the grid → 2D list**
  **:j : | :i : the coordinate x (resp. y) of the column (resp. row) to be checked → integer**
  **:return : True if the column j (resp. row i) is full | | else False → boolean**

- *a function that cancels the row i and drops the blocks above by a row :* **row_clear(grid, i)**
  **:grid : the matrix M of the grid → 2D list**
  **:i : the coordinate y of the row to be cleared → integer**
  **:return: the new matrix with i cleared → 2D list**

- *a function which cancels the column j in a grid if it's full:* **col_clear(grid, j)**
  **:grid : the matrix M of the grid → 2D list**
  **:j : the coordinate x of the column to be cleared → integer**
  **:return : the new matrix with j cleared → 2D list**

## IN BLOCKS.PY:

⇨ *a function which displays the list of all the blocks associated with it:* **print_blocks(list_blocks)**
**: list_blocks: list of matrices → list of 2D lists**

/// **modification in the parameters: list_block instead of grid**
- list of all the blocks associated to the grid is already defined in the main (because we need it in another function) => no need to specify the grid in the parameters
- we can print any other smaller/bigger list of blocks (ex : the 3 random blocks according to policy 2)

**:return: None (only prints)**

⇨ *a function which checks if a block can be placed (the lower left square is the reference):* **valid_position(grid, block, i, j)**
**:grid : the matrix M of the grid → 2D list**
**:block : matrix of the chosen block → 2D list**
**:i : (resp. :j :) index of the row (resp. column) → integer**
**:return: True if the block can be placed | else False → boolean**

⇨ *a function that positions the block chosen by the user on the grid if its position is valid :* **emplace_block(grid, block, i, j)**
**:grid : the matrix M of the grid → 2D list**
**:block : matrix of the chosen block → 2D list**
**:i : (resp. :j :) index of the row (resp. column) → integer**
**:return: temp_grid (modified matrix M) → 2D list**

⇨ *a function which displays the blocks proposed to the user according to the policy and type of tray chosen :*
**select_blocks(list_blocks, policy)**
**:list_blocks : list of matrices representing the blocks → list of 2D lists**
**:policy : either '1' or '2' → string**

/// **added parameters : list_blocks & policy**
- list_blocks to know all the available blocks according to the selected grid (defined in the main)
- policy to know if the choice is among the entire list_blocks or just a part of it (defined in the main)

**:return : the matrix of the selected block → 2D list**

⇨ *a function which updates the score each time a row or a column is canceled :* **update score(grid, mode, index)**
**:grid : the matrix M of the grid → 2D list**
**:mode : either 'row' or 'column' → string**
**:index : index of the row or column → integer**

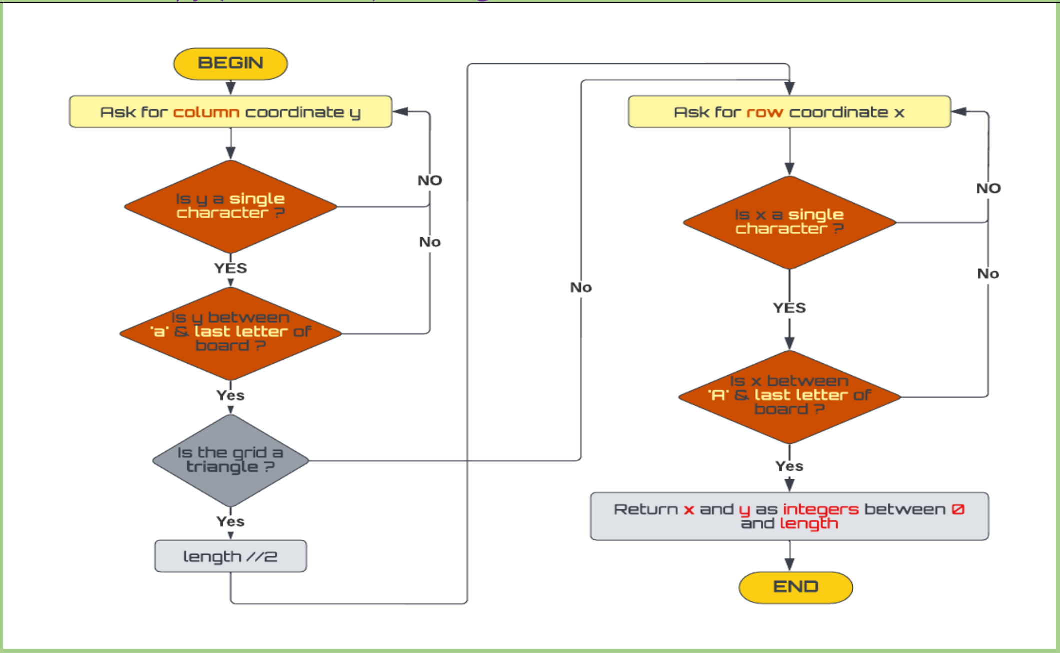
/// **added parameters : grid, mode & index**
- grid to have the matrix of the board with the emplaced blocks
- mode to know whether a row or a column was cleared
- index to know at which position was the row/column cleared

**:return: s (the score of the line) → integer**

## *b. Added functions*

⇨ *a function which asks for **correct** coordinates to put a block :* **coordinates(grid, size)**

- **:grid : the matrix M of the grid → 2D list**
- **:size : the length of the grid → integer**
- **:return: x, y (coordinates) → integers**



BEGIN
Ask for column coordinate y
Is y a single character ?
NO
YES
Is y between 'a' & last letter of board ?
No
Yes
Is the grid a triangle ?
No
Yes
length //2
Ask for row coordinate x
Is x a single character ?
NO
YES
Is x between 'A' & last letter of board ?
No
Yes
Return x and y as integers between 0 and length
END

⇨ *a function that can rotate a block:* **rotate(block)**
**: block: matrix of the chosen block → 2D list**
**: return: block (modified) → 2D list**

⇨ *function that saves all data from the finished game in another file for later use:* **backup_save(path, grid, size, shape, score)**

**: path: name (given by the user) of the new file to save the data of his game performance → string**
**: grid: matrix of the game to be saved into the path specified above → 2D list**
**: size: size of the game grid to remember for later re-use → string**
**: shape: shape of the grid to be played on for later re-use → string**
**: score: final score of the game for later re-use → string**
**:return: None (only executes)**

⇨ *a function that recovers all the data (size & shape of grid & score) directly from the file:* **reloading_all_data(path)**
**: path: the name of the file the user chose to reload → a string**
**: return: the grid size, the score of the game, the new name of the current file he will play in → integers and a string**

# III. How do these functions all cooperate?

## *a. Explanation of the main implemented algorithms*

⇨ **print_blocks :**
- Consider the blocks **8 by 8** (or all of the blocks if the remainder is less than 8)
- Print the rows of the considered blocks (1st row of all the blocks then the 2nd etc.)
- If the blocks aren't of the same size, the small blocks will be printed in **delay** (it will print empty lines before starting printing the blocks such that they are all aligned with respect to the bottom line)

⇨ **select_blocks :**
- If the player has chosen policy 1 => we print **all the blocks** with associated number (starting from 0) and the player has to give the number of the wanted block
- If the player has chosen policy 2 => we **randomly select 3 blocks**, print them with associated number and the player has to give the number of the wanted block

⇨ **print_grid :**
- Goes through the matrix created by read_grid
- If the element of the matrix is:
  - 0 => displays the ASCII symbol of a **blank**;
  - 1 => displays the ASCII symbol of a little **hollow cube**;
  - 2 => displays the ASCII symbol of a bigger **filled cube**
- For the letters around the grid => we run through the ASCII codes of **upper and lowercase alphabets** & we display them letter by letter
- For the frame around the grid, the simple use of ASCII code characters at the end of each line made the trick

⇨ **row_state & col_state** *(both functions have the same reasoning)*
- A boolean is set to True
- We go through the matrix's lines (or columns) and if it **contains a 1,** then the row (or column) is not full and the boolean is set to **False**
- If all the **elements are 2s**, then the boolean remains **True**, meaning that the grid (or column) is full

## *b. Choice of data structure*

The data structures that were used for this project were mainly **1D and 2D lists**. We used these more than another kind because:

⇨ they're the ones we're the more **familiar** with
⇨ physically, the matrix of a grid displayed line by line looks a lot like the files filled of 0s and 1s used for the grids, so it's much **easier to visualize**
⇨ memory wise, **dictionaries occupy much more space than lists**. Even an empty dictionary occupies more space than a list!
⇨ **N.B** : We have used **characters instead of integers** in the matrices of the **grids** because it is easier to manipulate from the files to the matrices and vice-versa

## *c. Difficulties encountered & solutions*

- Most difficulties encountered were related to the **complexification** of our code :

=> We wanted to make the game **simple** and **the most aesthetic possible** for the user to understand and appreciate.

ex : Printing the blocks 8 by 8 such that the user can see all of them without having to scroll down the screen

=> We had to use **built-in functions** to make the elements of the game as visually pleasing as possible

ex : ***.strip()*** to reduce the space when skipping lines, ***.format()*** to center the strings, ***.split()*** etc.

- Another difficulty was to make sure the game could never be interrupted no matter what the user wrote in the console

=> To allow the game to continue, all the required inputs are in strings, we had to add several conditions to accept the inputs and if necessary we cast them into integers

- One of the biggest constraints of this project was that **it had to be done uniquely in the python console.**
**Transforming a dark & serious command line interface into a pretty & fancy game was indeed challenge !**

N.B : Some of our codes can be extravagant in order to achieve that

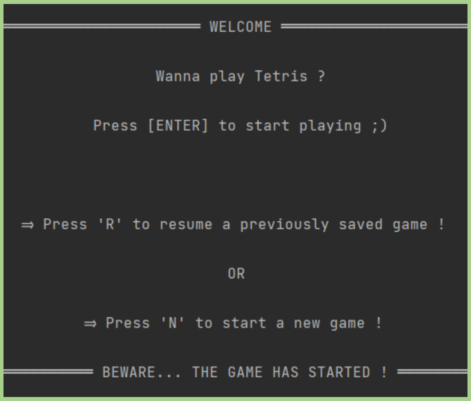
- We wondered for a long time why some functions did **not recognize the variables** that were declared in different files.

=> The solution we found was to use these very variables **as parameters** to these functions. And that's why in some cases, functions have **added parameters** to what was asked.

**Overall there are no difficulties that we did not, with time, find a solution to.**
**If we didn't think of the solutions ourselves, we asked for help t**
**our teachers & several internet pages such as StackOverflow.**

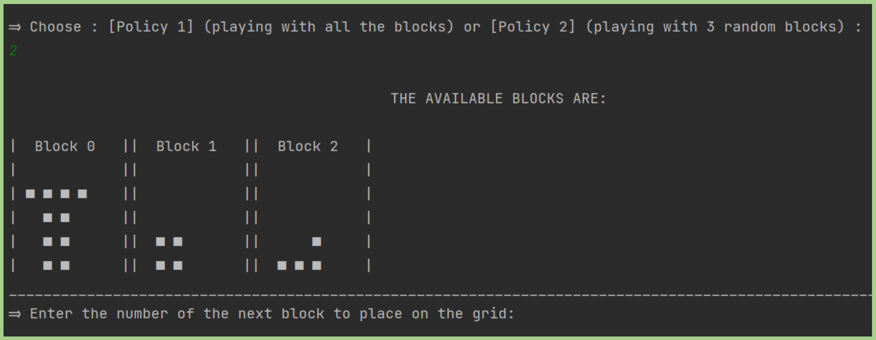
# IV. Results presentation

The user has the choice between resuming a game or starting a brand new one



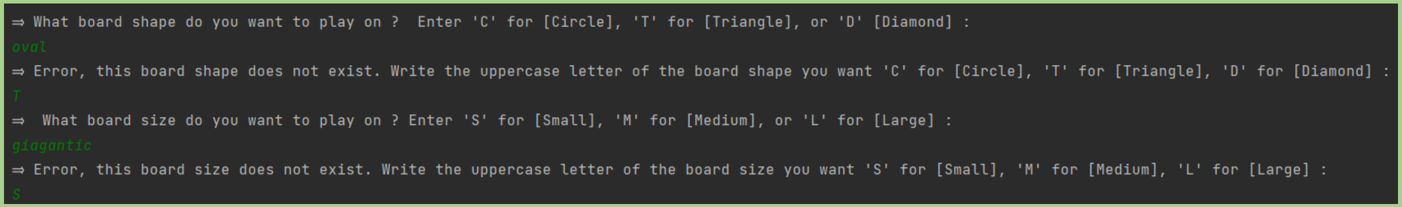
```
═════════════════════ WELCOME ═════════════════════

               Wanna play Tetris ?

         Press [ENTER] to start playing ;)



 ⇒ Press 'R' to resume a previously saved game !

                       OR

            ⇒ Press 'N' to start a new game !

═══════════ BEWARE... THE GAME HAS STARTED ! ═══════════
```

The user has chosen policy 2 and only wants to play with 3 random blocks



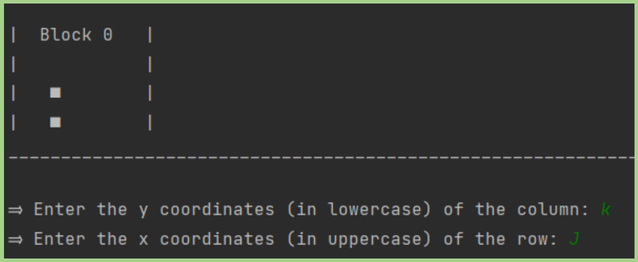
```
⇒ Choose : [Policy 1] (playing with all the blocks) or [Policy 2] (playing with 3 random blocks) :
2

                                            THE AVAILABLE BLOCKS ARE:

|  Block 0   ||  Block 1   ||  Block 2   |
|            ||            ||            |
| ■ ■ ■ ■    ||            ||            |
|   ■ ■      ||            ||            |
|   ■ ■      ||  ■ ■       ||      ■     |
|   ■ ■      ||  ■ ■       ||  ■ ■ ■     |
------------------------------------------------------------------------------------------------------
⇒ Enter the number of the next block to place on the grid:
```

For absolutely every wrong input or mistake given by the user, the program has an equivalent response and will not crash

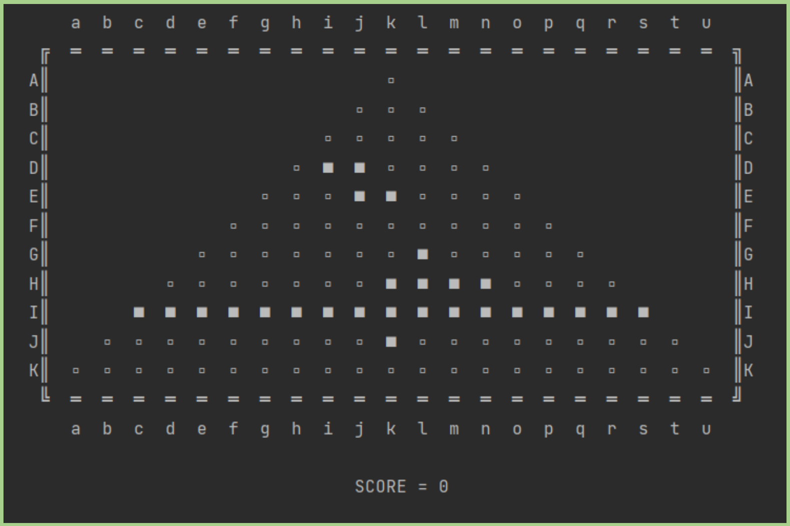


```
⇒ What board shape do you want to play on ?  Enter 'C' for [Circle], 'T' for [Triangle], or 'D' [Diamond] :
oval
⇒ Error, this board shape does not exist. Write the uppercase letter of the board shape you want 'C' for [Circle], 'T' for [Triangle], 'D' for [Diamond] :
T
⇒  What board size do you want to play on ? Enter 'S' for [Small], 'M' for [Medium], or 'L' for [Large] :
giagantic
⇒ Error, this board size does not exist. Write the uppercase letter of the board size you want 'S' for [Small], 'M' for [Medium], 'L' for [Large] :
S
```

Here, the user entered the coordinates k and J for a block he chose.

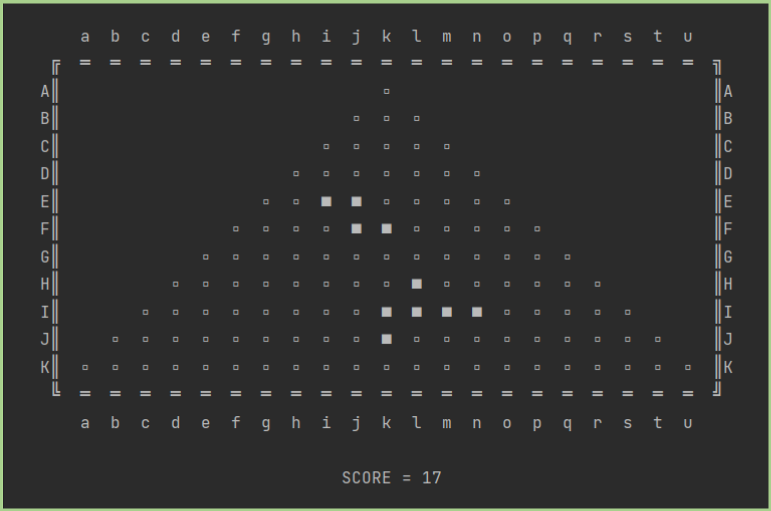


```
|  Block 0   |
|            |
|  ■         |
|  ■         |
---------------------------------------------------------------

⇒ Enter the y coordinates (in lowercase) of the column: k
⇒ Enter the x coordinates (in uppercase) of the row: J
```



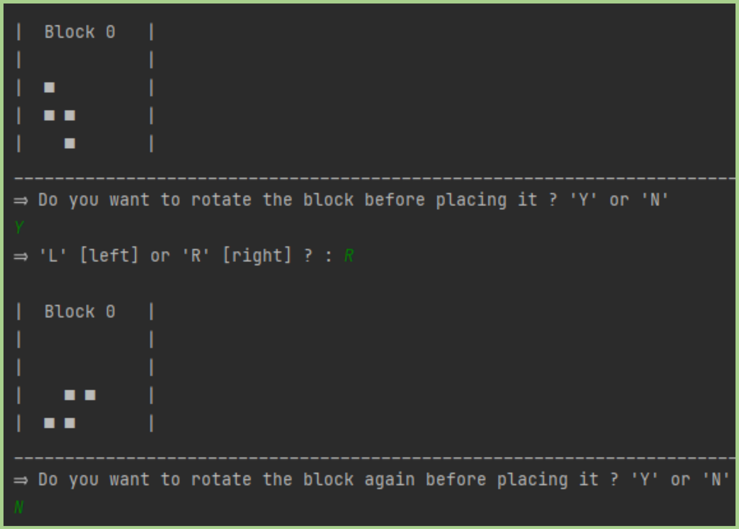
```
   a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u
 ╓ ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═ ╖
A║                               ▫                               ║A
B║                            ▫  ▫  ▫                            ║B
C║                         ▫  ▫  ▫  ▫  ▫                         ║C
D║                      ▫  ■  ■  ▫  ▫  ▫  ▫                      ║D
E║                   ▫  ▫  ▫  ■  ■  ▫  ▫  ▫  ▫                   ║E
F║                ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫                ║F
G║             ▫  ▫  ▫  ▫  ▫  ▫  ▫  ■  ▫  ▫  ▫  ▫  ▫             ║G
H║          ▫  ▫  ▫  ▫  ▫  ▫  ▫  ■  ■  ■  ■  ▫  ▫  ▫  ▫          ║H
I║       ■  ■  ■  ■  ■  ■  ■  ■  ■  ■  ■  ■  ■  ■  ■  ■  ■       ║I
J║    ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ■  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫    ║J
K║ ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫ ║K
 ╙ ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═ ╜
   a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u

                            SCORE = 0
```

The user filled up a line, therefore it is cleared. The blocks that were above it go down, and the score goes up by the number of little squares eliminated



```
   a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u
 ╓ ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═ ╖
A║                               ▫                               ║A
B║                            ▫  ▫  ▫                            ║B
C║                         ▫  ▫  ▫  ▫  ▫                         ║C
D║                      ▫  ▫  ▫  ▫  ▫  ▫  ▫                      ║D
E║                   ▫  ▫  ■  ■  ▫  ▫  ▫  ▫  ▫                   ║E
F║                ▫  ▫  ▫  ▫  ■  ■  ▫  ▫  ▫  ▫  ▫                ║F
G║             ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫             ║G
H║          ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ■  ▫  ▫  ▫  ▫  ▫  ▫          ║H
I║       ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ■  ■  ■  ■  ▫  ▫  ▫  ▫  ▫       ║I
J║    ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ■  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫    ║J
K║ ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫  ▫ ║K
 ╙ ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═  ═ ╜
   a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u

                            SCORE = 17
```

The user rotating the block he chose to the right



```
|  Block 0   |
|            |
| ■          |
| ■ ■        |
|   ■        |
------------------------------------------------------------------------
⇒ Do you want to rotate the block before placing it ? 'Y' or 'N'
Y
⇒ 'L' [left] or 'R' [right] ? : R

|  Block 0   |
|            |
|            |
|   ■ ■      |
| ■ ■        |
------------------------------------------------------------------------
⇒ Do you want to rotate the block again before placing it ? 'Y' or 'N'
N
```

## *Data to test …*

| | |
|---|---|
| *print_grid* | *col_state* |
| **An example of matrix :**<br>p = [['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1']]<br><br>**Instructions :**<br>print_grid(p) | **An example of matrix :**<br>p = [['0','1','0','1','0','2','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','2','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','2','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','2','0','1','0','1','0','1','0','1','0','1','0','1']]<br><br>**Instructions :**<br>print_grid(p) #to visually verify that the col is full<br>col_state(p, 4) #must return True |
| *row_clear* | *update_score* |
| **An example of matrix :**<br>p = [['0','0','1','1','2','2','2','1','1','0','2','1','0','1','0','1','0','1'],<br>['0','2','2','2','2','2','2','2','2','2','2','2','0','0','0','0','0','0'],<br>['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1']]<br><br>**Instructions :**<br>print_grid(p) #to visually verify that the row is full<br>p = row_clear(p, 1) #will clear row & drop the blocks above<br>print_grid(p) #to visually verify that it was done | **An example of matrix :**<br>p = [['0','0','1','1','2','2','2','1','1','0','2','1','0','1','0','1','0','1'],<br>['0','2','2','2','2','2','2','2','2','2','2','2','0','0','0','0','0','0'],<br>['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1'],<br>['0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1','0','1']]<br><br>**Instructions :**<br>score = 3<br>print_grid(p) #to visually verify that the row is full<br>score = update_score(M, ‘row’, score, 1)<br>print(score) #to verify the new score (14) |
| *print_blocks (case 1 – with more than 8)* | *print_blocks (case 2 – with less than 8)* |
| **An example of list of blocks :**<br>triangle_list = [[] for i in range(11)]<br>triangle_list[0] = [[1, 0, 0], [1, 1, 1], [0, 0, 1]]<br>triangle_list[1] = [[1, 1, 0], [0, 1, 0], [0, 1, 1]]<br>triangle_list[2] = [[0, 0, 1], [1, 1, 1], [1, 0, 0]]<br>triangle_list[3] = [[0, 1, 1], [0, 1, 0], [1, 1, 0]]<br>triangle_list[4] = [[0, 0, 1], [0, 1, 0], [1, 0, 0]]<br>triangle_list[5] = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]<br>triangle_list[6] = [[1, 0, 0], [1, 0, 0], [1, 0, 0]]<br>triangle_list[7] = [[0, 0, 0], [1, 1, 1], [1, 1, 1]]<br>triangle_list[8] = [[0, 0, 0], [1, 0, 0], [1, 0, 0]]<br>triangle_list[9] = [[0, 1, 0], [1, 1, 1], [0, 1, 0]]<br>triangle_list[10] = [[0, 0, 0], [0, 0, 0], [1, 1, 0]]<br><br>**Instructions :**<br>print_blocks(triangle_list) | **An example of list of blocks :**<br>blocks = [[[1,1,1],[1,1,1],[1,1,1]], [[1,0,0],[1,0,0],[1,0,0]],<br>[[1,0,1],[0,1,0],[1,0,1]]]<br><br>**Instructions :**<br>print_blocks(blocks) |
| *emplace_block* | *rotate* |
| **An example of grid and block :**<br>M =[['0','0','1','1','1','1','1','1','0','0'],['0','0','1','1','1','1','1','1','0','0'],<br>['0','0','1','1','1','1','1','1','0','0'],['0','0','1','1','1','1','1','1','0','0']]<br><br>b = [[1,1,1],[1,1,1],[1,1,1]]<br><br>**Instructions :**<br>print_grid(M) & print_blocks([b])<br>emplace_block(M, b, 3, 3)<br>print_grid(M) #to visualize the block on the grid | **An example of matrix :**<br>b = [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 1, 0], [1, 1, 1, 1, 0],<br>[0, 0, 0, 1, 0]]<br><br>**Instructions :**<br>print_blocks([b]) #to visualize the initial block<br>b = rotate(b, True) #will rotate to the right<br>print_blocks([b]) #to visualize the block rotated<br>b = rotate(b, False) #will rotate to the left (initial state)<br>print_blocks([b]) #to verify the state |

# V. Conclusion

## *a. Lessons learned*

- ⇨ **Writing** the algorithm on paper, **drawing schemes** with examples on smaller scales & trying our codes with those smaller examples is actually very useful and helps to visualize the situation a lot better

- ⇨ We learned a few **new built-in functions** like .strip() for example, or some other .format() functions that were not in the course, and other built-in conditions like try/except,...

- ⇨ We learnt that we have to add parameters in our functions if we want to use variables declared in other files

## *b. Working as a team, communication & time management*

The work was asked to be done in pairs and it went very well. We both would meet each other every Wednesday at 7h30 at EFREI to work on the project together until our other courses at 13h50.

But we also worked a lot at home, continuously sharing our findings via GITHUB.

## *c. What it taught us overall*

This project definitely made us acquire discipline and a sense of priorities. There was a deadline date, we had many other lessons and we absolutely needed to be organized.

Finally, now that the project is done, we see it not just as a painful & stressful chore but more as a satisfying & interesting one, that's making us even more curious to learn and pursue Python on our own.