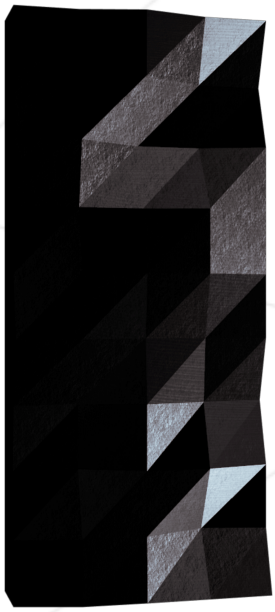

Piscine 101

Python rush

Summary: This document is the subject for the PYTHON rush of the Piscine 101 @ 42Tokyo.

Contents

Chapter I

Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- On top of that, your exercises will be checked and graded by a program called Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.
- Exercises in Shell must be executable with /bin/sh.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- If no other explicit information is displayed, you must assume the following versions of languages : `Python - python3.8.2`.
- No code in the global scope(only import). We want functions!
- you're free to define as many function as you like and name them as you like also.

- Each turned-in file must end with a function call in a condition identical to:

```
if __name__ == '__main__':
    your_function( whatever, parameter, is, required )
```

- You can set an error management in this condition.
- No import will be authorized, except the ones explicitly mentioned in the 'Autorized functions' in each exercise's description.
- You won't have to manage the exceptions raised by the `open` function.

Chapter II

Foreword

https://xkcd.com/353/

Chapter III

Introduction

Filler is an algorithmic game which consists in filling a grid of a known size in advance with pieces of random size and shapes, without the pieces being stacked more than one square above each other and without them exceeding the grid. If one of these conditions is not met, the game stops.

Each successfully placed piece yields a number of points, and has only one player, the goal of the game could be to get the best score possible. However, it is with two players that the filler takes all his interest. Each player has for the purpose of placing as many pieces as possible while attempting to prevent his opponent from doing the same. At the end of the game, the one with the most points wins the match...

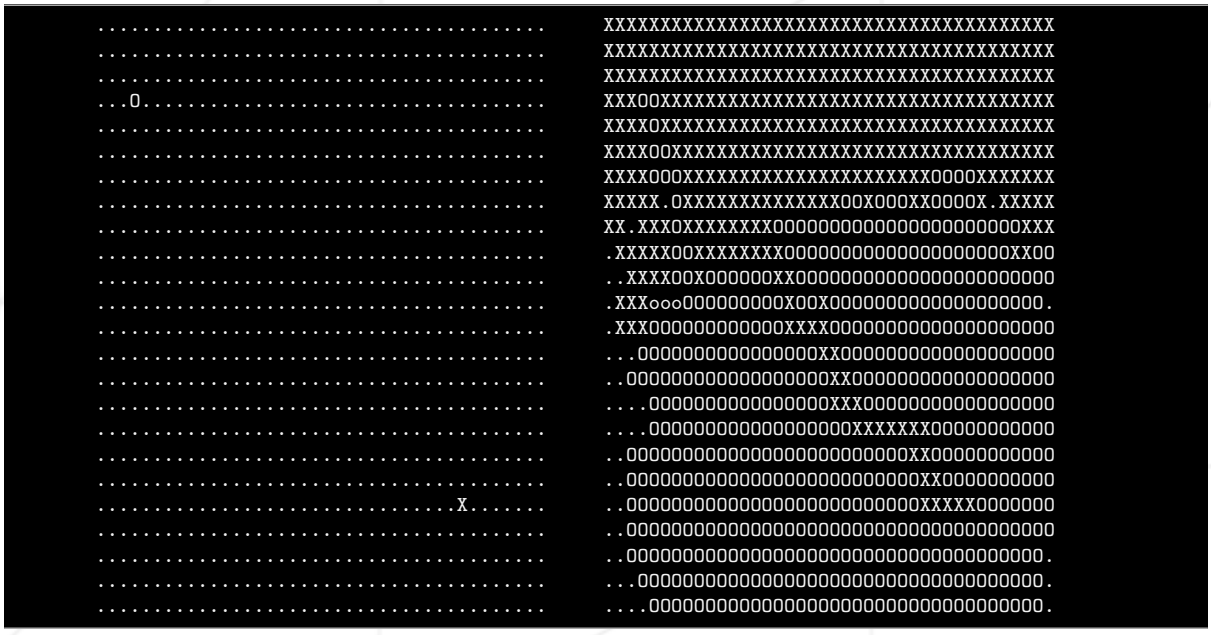

Figure III.1: On the left: the initial state of the grid, at the right: the result after the fight between two players.

Chapter IV

General Instructions

- This project will only be corrected by humans. You are therefore free to organize and name your files as you wish, while respecting the constraints listed here.
- You'll have to submit a file called `author` containing your username followed by a '\n' at the root of your repository.

```
$>cat -e author
xlogin$
ylogin$
zlogin$
```

- the file includes main must be named `<name of choice>_filler.py`.
- It must be at the root of the repository.
- You have to handle errors carefully.
- Good luck and GOOD FIGHT to all!

Chapter V

Mandatory part

V.1 The Filler

- In this game, two players fight each other. They play one after the other.
- The goal is to collect as many points as possible by placing the highest number of pieces on the the game board.
- The board is defined by X columns and N lines, it will then become X*N cells.
- At the beginning of each turn, you will receive your game piece.
- A game piece is defined by X columns and N lines, so it will be X*N cells. Inside each game piece, will be included a shape of one or many cells.
- To be able to place a piece on the board, it is mandatory that one, and only one cell of the shape (in your piece) covers the cell of a shape placed previously (your territory).
- The shape must not exceed the dimensions of the board
- When the game starts, the board already contains one shape.
- The game stops at the first error: either when a game piece cannot be placed anymore or it has been wrongly placed.

V.2 The Board

A board is a two-dimensional grid with an arbitrary number of rows and columns. To launch the game an initial board must be passed as an argument to the VM. This initial board must have a starting form for each player.

Here is an example of an initial board of 14 by 30 :

```
Plateau 14 30:
   01234567890123456789012345678 9
000 ..............................
001 ..............................
002 ..X...........................
003 ..............................
004 ..............................
005 ..............................
006 ..............................
007 ..............................
008 ..............................
009 ..............................
010 ..............................
011 ............................O..
012 ..............................
013 ..............................
```

V.3 The tokens

The tokens are managed randomly by the VM. You can't predict their size or shape until the VM transmits them to your program. Here are some arbitrary examples of possible tokens to give you an idea:

```
Piece 4 7  Piece 4 5: Piece 3 6:
 ...*...     .**..      .****.
 ...*...     .***.      **....
 ...*...     ..*..      *.....
 ..***..     .....
```

V.4 The Topic

V.4.1 The Player

- The executable that will enable you to play the filler is attached to this subject.
- For this project, you will have to create a filler player. Your goal is to win:
  - It will read the board and the game pieces placed on the standard output.
  - Each turn the filler rewrites the board map and includes a new piece to be placed.
  - In order to place the game piece on the board, the player will have to write it's coordinates on the standard ouput.
  - The following format must be used "X Y\n".
  - You will collect points each time you place a piece.

```
Plateau 14 30:
01234567890123456789012345678 9
000 ..............................
001 ..............................
002 ..............................
003 ..............................
004 ..............................
005 ..............................
006 ..............................
007 ..............................
008 ..............O...............
009 ..............................
010 ..............................
011 ..............................
012 ..............................
013 ..............................
Piece 4 7:
...*...
...*...
...*...
..***..
```

Watch out! You must write the coordinates of the token and not those of the shape.

V.4.2 Multi Players

- Player number:
  - The first two lines of the filler must be in the following format:

    ```
    $$$ exec pPLAYER_NUMBER : [PLAYER_NAME]
    ```

  - The filler will only send the line that concerns your program. You'll have to get your player number.
  - If you are Player 1 your program will be represented by "o" and "O". If you are Player 2, your program will be represented by "x" and "X". The first step will be to get your player number.
  - The lowercases ("x" or "o") will highlight the piece last placed on the board. At the following turns, that same piece will be represented by the uppercase letters ("X" or "O"), as it won't be the piece last placed anymore.
  - You will collect points each time you place a piece.
- How the game works
  - At each turn, the filler will send the updated map and a new token to the player concerned.
  - The player concerned will write on the standard output his or her piece's coordinates to place it on the board.
  - The filler will send the map and a new piece to the other player.

V.4.3 How the game rolls

- Here is an example on how a game will roll out.

```
$>./filler_vm -p1 user1 -p2 user2 -v -f samples/w1.flr
$$$ exec p1 : [user1]
$$$ exec p2 : [user2]
Plateau 14 30:
01234567890123456789012345678 9
000 ..............................
001 ..............................
002 ..............................
003 ..............................
004 ......X.......................
005 ..............................
006 ..............................
007 ........................O...
008 ..............................
009 ..............................
010 ..............................
011 ..............................
012 ..............................
013 ..............................
Piece 3 6:
.****.
**....
*.....
<got (O) : [7 24] (7,24)
Plateau 14 30:
01234567890123456789012345678 9
000 ..............................
001 ..............................
002 ..............................
003 ..............................
004 ......X.......................
005 ..............................
006 ..............................
007 ........................oooo.
008 ......................oo....
009 ......................o.....
010 ..............................
011 ..............................
012 ..............................
013 ..............................
Piece 3 8:
......*.
......**
.......*
<got (X) : [4 0] (4,0)
Plateau 14 30:
01234567890123456789012345678 9
000 ..............................
001 ..............................
002 ..............................
003 ..............................
004 ......x.......................
005 ......xx......................
006 .......x......................
007 ........................OOOO.
008 ......................OO....
009 ......................O.....
010 ..............................
011 ..............................
012 ..............................
013 ..............................
[...]
== X fin : 175 [1018918090]
== O fin : 168 [1018918090]
```

To specify a player written in Python, excute as follows

```
$>./filler_vm -p1 "python3 user1_filler.py" -p2 "python3 user2_filler.py" -v -f samples/w1.flr
```

V.4.4 VM

If you experience problems with the VM, please contact us on slack.
Really make sure the problem is indeed coming from the VM and not
from your program.

Chapter VI

Bonus part

Bonuses will only be evaluated if your mandatory game is PERFECT. By PERFECT, we mean of course that it is fully realized, and that it is not possible to put its behavior in default, even in case of error as vicious as it is, misuse, etc. Concretely this means that if your mandatory game does not get ALL points the rating, your bonuses will be fully IGNORED.

As bonus points, we will take into account:

- A graphic visualizer.
- Any additional bonuses that you will consider useful and that your peers will approve and enjoy.