Chapter II

Introduction

Create your player to fight other students on the world famous (or infamous) Filler board. The concept is simple: two players gain points by placing on a board, one after the other, the game piece obtained by the game master (in the form of an executable Ruby program). The game ends when the game piece cannot be placed anymore. Have fun!

Chapter III

Goals

The goal of this project is to introduce you to basic algorithm and to have you manipulate inputs/outputs.

Chapter IV

General Instructions

- You'll have to submit a file called `author` containing your username followed by a '\n' at the root of your repository.

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

- The executable file must be named `<login>.filler`.
- It must be at the root of the repository.
- You must submit a `Makefile`.
- Your `Makefile` must compile the project and must contain the usual rules. It must recompile and re-link the program only if necessary.
- If you are clever, you will use your library for your player. Submit also your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- Your project must be written in accordance with the Norm.
- You have to handle errors carefully. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Within your mandatory part, you are allowed to use the following functions:
  - `read`
  - `write`
  - `malloc`
  - `free`
  - `perror`
  - `strerror`
- 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 14x30 board

```
Plateau 14 30:
012345678901234567890123456789
000 ..............................
001 ..............................
002 ..............................
003 ..............................
004 ..............................
005 ..............................
006 ..............................
007 ..............................
008 ..............O...............
009 ..............................
010 ..............................
011 ..............................
012 ..............................
013 ..............................
```

V.3 The tokens

- An example of a 4x7 token

```
Piece 4 7:
...*...
...*...
...*...
..***..
```

- An example of a 4x5 token

```
Piece 4 5:
.**..
.***.
..*..
.....
```

- An example of a 3x6 token

```
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:
012345678901234567890123456789
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]
```

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

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.