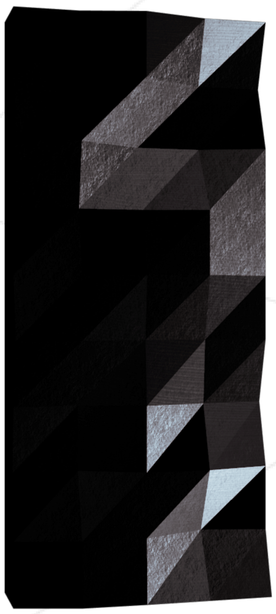


# OCaml Pool - Rush 00

## Tic-tac-tic-tac-toe

42 pedago pedago@42.fr
kashim vbazenne@student.42.fr
nate alafouas@student.42.fr

*Abstract: This is the subject for rush00 of the OCaml pool.*

# Contents

# Chapter I

# Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, . . . , `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the `OCaml` syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercices must be done in order. The graduation will stop at the first failed exercice. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerfull ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the `OCaml` piscine. You can use any style you like, no restrictions. But remember that a code your peer-

evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.

- You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.

- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.

- By Odin, by Thor! Use your brain!!!

# Chapter II

# Day-specific rules

- You **MUST** provide an author file named `auteur` at the root folder of your repository, as always.

```
$> cat -e auteur
alafouas$
vbazenne$
$>
```

- This project will be entirely graded by humans. As such, you are free to have some variation between this subject's examples and your program's actual output. However, you do have to obey the spirit of the subject and turn in a program with consistent and relevant output formatting.

- For the same reasons, you are entirely free to choose your file names and general hierarchy. But keep in mind that a consistent and relevant file organization is a good code practice and will earn you bonus points.

- Any unexpected termination, stack overflow, segmentation fault, uncaught exception or unsafe behaviour means a grade of 0 — with the exception of `End_of_file`.

- You can use the modules `Pervasives`, `String` and `List` and every syntaxes and semantics covered in this first week's videos. On the other hand, you cannot use any of the following: `ref`, exceptions, arrays, mutable records, objects. If you do, you will get a grade of -42 for cheating.

- You are free to organize your files and modules as you see fit, but you must turn in a `Makefile` to build your work. Your `Makefile` **MUST** be able to build your work as a bytecode executable and as a native executable. Obviously, both binaries must behave exactly the same way. You can use `OCamlMakeFile` if you want.

# Chapter III

# Foreword

Food for thought:

> This is the only time! The players fight with all their strength: the fans cheer for their favorite team. They forget pain, suffering... Only the game matters! That's why blitz has been around for so long. Least that's what I think.

By Wakka, from Final Fantasy X.

# Chapter IV

# Mandatory Part

This week was a lot of fun; or at least, I hope it was a lot of fun for you. But now that you've polished your skills in training, fighting your way through mud, dirt, and horrendously boring exercises only limited by the cruel imagination of yours truly, here's your time to rise and shine, and show the world what you can do with what you've learnt of OCaml so far !

This first rush will be your very first full and "complex" program. But don't worry, let's take a few minutes so everything's clear in your minds, and I'm sure you'll do great.

For the next two days, you'll be working on a game of tic-tac-toe. This game is played by two players on a 3x3 cells board.

```
- - -
- - -
- - -
```

Let's number the cells like this:

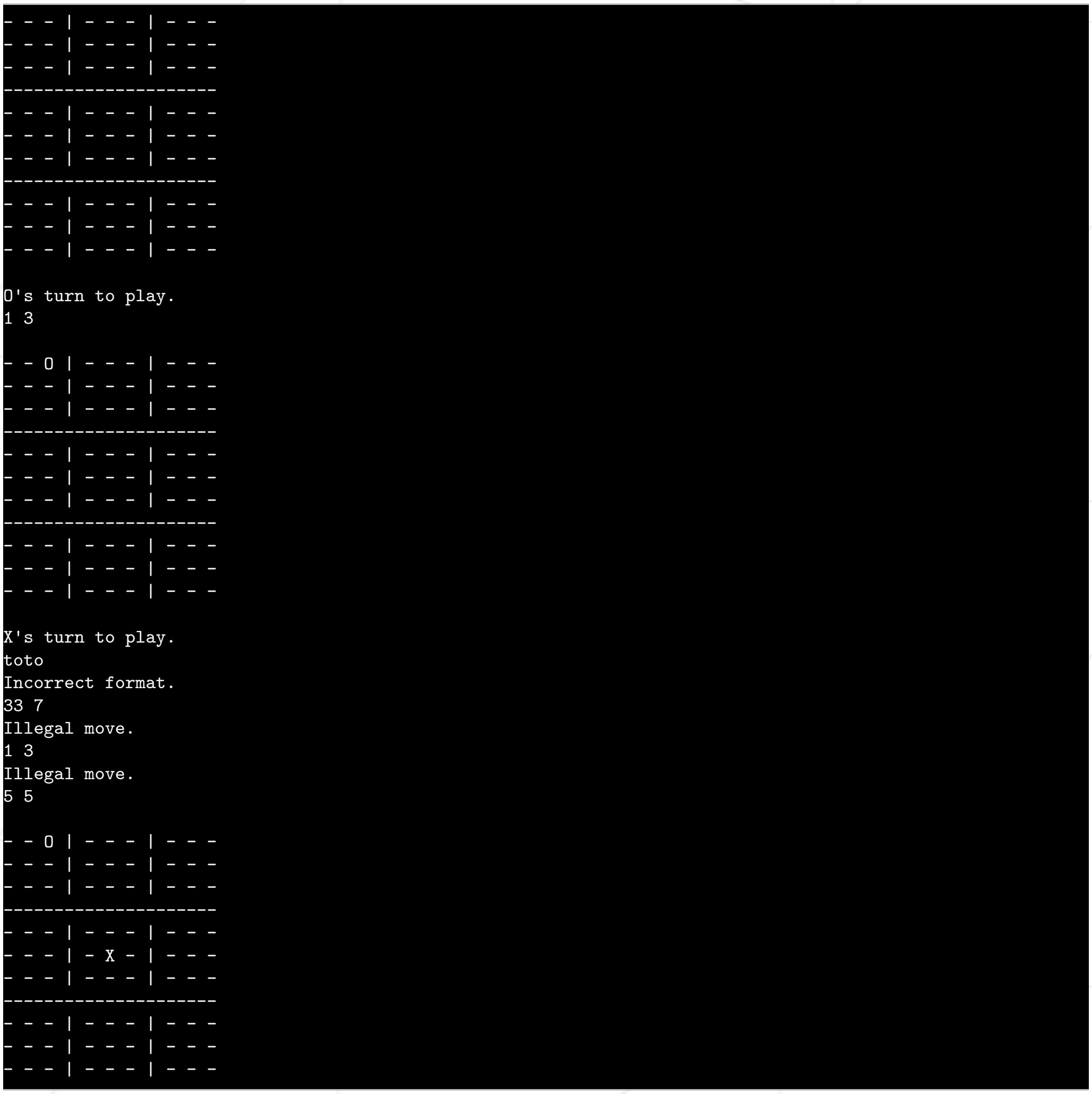
```
1 2 3
4 5 6
7 8 9
```

Each player fills a cell, player one with `O` and player two with `X`. By convention, the player with `O`s always starts. The player's goal is to make a straight line with three of his symbols.

```
X - -
O O O
- - X

O wins!

X - -
X O O
X O O

X wins!
```

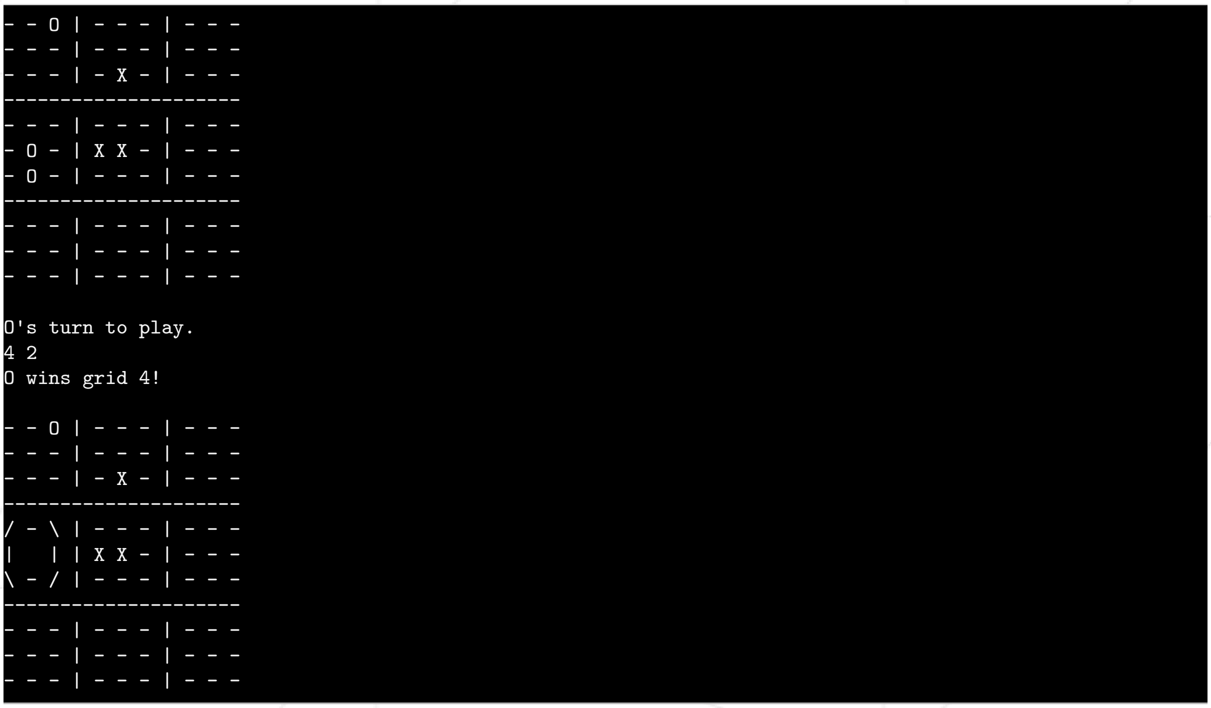
This is the simple tic-tac-toe; but our tic-tac-tic-tac-toe is a bit more complicated. Each cell is a board itself, meaning that a game of tic-tac-tic-tac-toe is actually a set of nine tic-tac-toe games.

Players pick a cell one after another, using a simple format like `(row) (column)`. No line discipline at all is expected. Use a simple `read_line`, and note that `ctrl-d` won't be tested in defence.



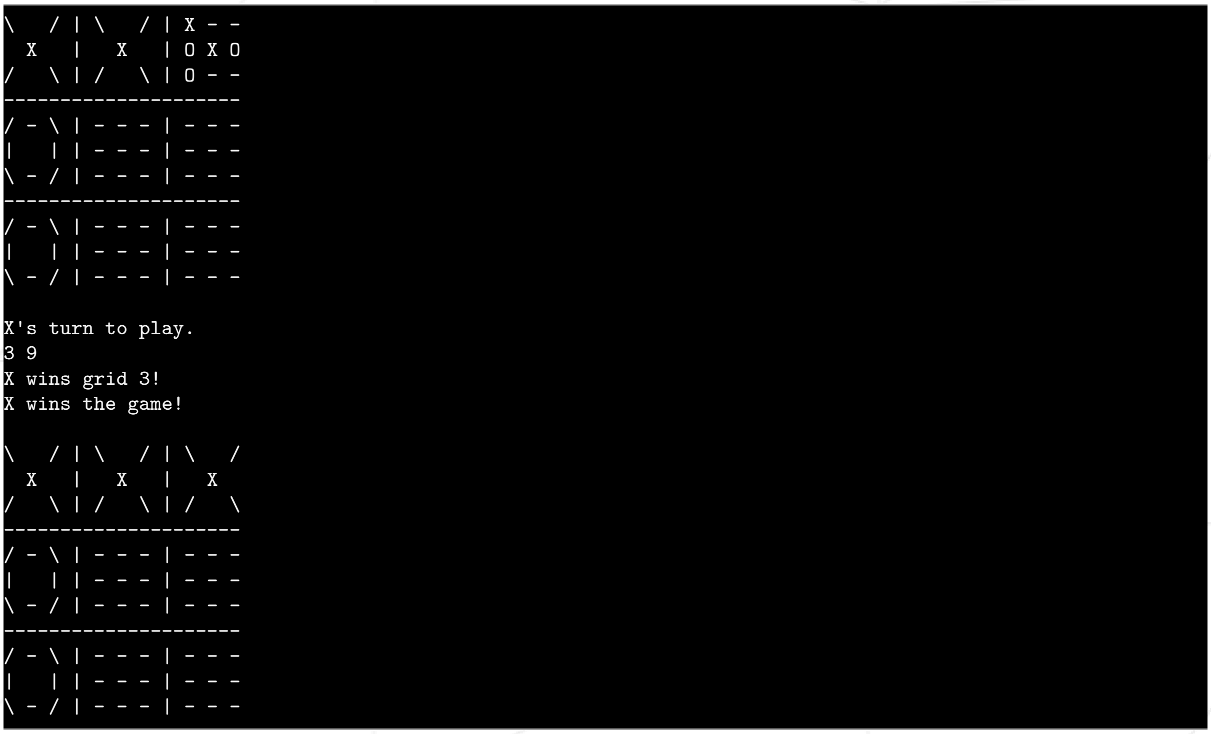
```
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -

O's turn to play.
1 3

- - O | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -

X's turn to play.
toto
Incorrect format.
33 7
Illegal move.
1 3
Illegal move.
5 5

- - O | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - X - | - - -
- - - | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -
```

Both players try to win the game by winning the main board by winning the nested grids. If a nested grid ends up in a draw, the winner is the one who puts the last (and ninth) symbol into the grid.



```
- - O | - - - | - - -
- - - | - - - | - - -
- - - | - X - | - - -
---------------------
- - - | - - - | - - -
- O - | X X - | - - -
- O - | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -

O's turn to play.
4 2
O wins grid 4!

- - O | - - - | - - -
- - - | - - - | - - -
- - - | - X - | - - -
---------------------
/ - \ | - - - | - - -
|   | | X X - | - - -
\ - / | - - - | - - -
---------------------
- - - | - - - | - - -
- - - | - - - | - - -
- - - | - - - | - - -
```

The game ends when one player has a straight line of his symbols on the main board.



```
\   / | \   / | X - -
  X   |   X   | O X O
/   \ | /   \ | O - -
---------------------
/ - \ | - - - | - - -
|   | | - - - | - - -
\ - / | - - - | - - -
---------------------
/ - \ | - - - | - - -
|   | | - - - | - - -
\ - / | - - - | - - -

X's turn to play.
3 9
X wins grid 3!
X wins the game!

\   / | \   / | \   /
  X   |   X   |   X
/   \ | /   \ | /   \
---------------------
/ - \ | - - - | - - -
|   | | - - - | - - -
\ - / | - - - | - - -
---------------------
/ - \ | - - - | - - -
|   | | - - - | - - -
\ - / | - - - | - - -
```

# Chapter V

# Bonus Part

## V.1 Mandatory bonuses

These bonuses are considered “mandatory” because they are fundamental and common improvements for your work. Other bonuses will not be counted if these are not implemented fully and perfectly.

- A one-player mode against an IA. It **must** try to win and to prevent you from winning.
- Letting the players enter their name, and display it instead of `O`/`X`. Both players can’t use the same names, or use empty names.
- Letting the players begin a new game after the previous one has ended.

## V.2 Optional bonuses

You’re free to provide any feature you want as bonus. But keep in mind that everything you present as a bonus must not keep the program from behaving like specified in the subject’s mandatory part. If your feature does alter the program’s behaviour as such, the end-user must be able to disable and enable it at runtime (i.e. without having to recompile the application).

Here are some suggestions for you:

- A badass graphical interface, with mouse control and colors!
- A badass ncurses interface, with an arrow-controlled cursor and colors!
- A customizable grid size! Why not 4x4, 5x5...?
- A customizable grid nesting! Because we need to go deeper.
- A game timer like in chess, Go or other zero-sum games!

Really the only limit for this section is your imagination, so go ahead and show the world how you can create rainbows and unicorns with OCaml!