# OCaml Exercise: Minesweeper

**Ziqi Yang**
ziqi.yang@uzh.ch

## 1 Introduction & Environment Setup

### 1.1 Project Introduction

In the exercise, you will implement a simplified version of the game `Minesweeper` with OCaml, a popular functional programming language. Your task is to model the game grid, generate random mines, compute numbers indicating adjacent mines, and allow the user to interactively uncover cells until either all safe cells are revealed or a mine is triggered. We will focus on logical reliability but not on a graphical interface.

### 1.2 OCaml Setup

We need the OCaml compiler, the package manager `opam`, and the build system `dune`.

- **For macOS (with Homebrew):**
  Open the terminal and execute the following commands:

  ```
  brew install opam
  opam init
  eval $(opam env)
  opam install dune
  ```

- **For Windows (WSL):**
  I will recommend using Windows Subsystem for Linux (WSL) which can be installed from the Microsoft Store. Open the WSL terminal and execute the following commands:

  ```
  sudo apt-get install opam
  opam init -y
  eval $(opam env)
  opam install dune
  ```

Please follow the link below for your operating system if you meet any problems or try installing platform tools:

`https://ocaml.org/docs/up-and-running`

I strongly recommend using **Visual Studio Code** with the **OCaml Platform** extension. It provides helpful features like type-checking, code completion, and error highlighting.

## 2 Key Concepts in OCaml

The Minesweeper game board and its updates are pure, immutable structures—each player move produces a new version of the board rather than modifying it in place. Essentially, you will design a small state transition system that reacts to user actions and returns an updated state. Here are some OCaml features you will use during the development.

### 2.1 Algebraic Data Types (ADTs)

Instead of using `int` or `string` to represent concepts (e.g., `cell_state = 0` for hidden, `1` for revealed), OCaml lets you define a new type that can only be one of a few specific things.

```
(* A cell can only hold the following four states *)
type cell =
  | Hidden
  | Revealed of int
  | Flagged
  | Mine
```

This is a powerful modeling tool because it's impossible to create a cell you do not expect.

### 2.2 Pattern Matching

You can use `match ... with` to safely handle all possibilities. The OCaml compiler will warn you if you forget a case.

```
type board = cell array array
let reveal (b: board) (x: int) (y: int) : board =
    match (x, y) with
    | (* Hidden *)
    | (* Revealed *)
    | (* Flagged *)
    | (* Mine *)
    ...
```

### 2.3 Immutability & Recursion

In OCaml, data is **immutable** by default so that it cannot be changed after creation. You should avoid writing a standard `for` loop that changes a counter variable. Instead, we use **recursion**. A recursive function calls itself with a new, slightly different version of the data.

```
(* assume that func is the function you handle if users continue the game or lose *)
let rec reveal (b: board) (x: int) (y: int) : board =
    match (x, y) with
    | (* Hidden *) reveal b (func x) (func y)
    | (* Revealed: set up an action up to you *)
    | (* Flagged: cancel the flag *)
    | (* Mine: fail *)
    ...
```

## 3 Task Description

The Minesweeper game is played on a 2D grid of cells as usual. Here are some details for reference. You can feel free to implement with your own idea!

### 3.1 Data Model

- For the 2D grid, using OCaml's `array array` is the most straightforward approach.
- A cell can either be a `Mine` or a `Number` representing the number of its mine neighbors.
- Every cell also has a `State`: `Hidden`, `Flagged`, or `Revealed`.

### 3.2 Program Flow

The program's main responsibility is to handle the revealing action:

1. A player inputs a coordinate (row, col) to reveal.
2. The program reveal the cell and update the board state.
3. (Challenge!) If the revealed cell is a `Number 0`, you must recursively reveal all 8 of its neighbors and the process continues if any of those neighbors are also `Number 0`. Otherwise, just show the number.
4. The player wins when there is no mine. Otherwise, end the game and output the result: show all the mines including those which are not revealed yet on the board.

### 3.3 Implement Tips

- You need to generate a grid and randomly place the mines. Use `Random.init seed` and `Random.init` to select random coordinates and make sure no two mines occupy the same cell. You can use a fixed integer seed during testing to generate the same board each run.
- Count the number of neighbor mine cells for each non-mine cell and set the states of all cells as `Hidden`. Maximum 8 neighbors.
- Do not forget to check if users' input are valid two numbers in bounds.
- Build **a new board** when a move is made instead of changing cells in place. Recall the immutability.
- Try to use recursion for revealing multiple cells.
- After each move, check whether all non-mine cells are revealed. You can use `Array.for_all` to do so.
- Use `Printf.sprintf` to align numbers and add spaces for better readability when outputting the result. You can also use it to add row/column indices which will help debugging.

For more useful features in OCaml, I will recommend the official document again:

`https://ocaml.org/docs/values-and-functions`

# 4 Passing Criteria and Submission

## 4.1 Acceptance Criteria

Your solution will be considered as **pass** if it meets these criteria:

1. **Formatted Execution:** The program should be executed with formatted parameters such as grid size and mine number to configure the game :

```
dune exec minesweeper -- --rows 8 --cols 8 --mines 10 --seed 42
```

2. **Successful Initialization:** The initialization function correctly creates a grid in a specified size, places the specified number of mines and prints the initial board. The number of mines and the grid size can either be a fixed value or set by users.
3. **Single Reveal:** Inputting the coordinate of a `Hidden` cell with a positive number reveals *only* the cell.
4. **Chain Reveal:** Inputting the coordinate of a `Hidden` cell with `0` correctly and recursively reveals all adjacent cells, stopping when it hits non-zero numbers.
5. **Game Over:** Inputting the coordinate of a `Hidden` cell with `Mine` content correctly identifies the game as `Lost`.
6. **Safety:** The program does not crash from out-of-bounds errors, e.g. trying to check neighbors of a cell on the edge of the board.

## 4.2 Submission Format

Please submit a single `.zip` archive of your project directory. The directory should be a standard `dune` project created with the command:

```
dune init proj minesweeper
```

You can feel free to implement the features in a single file or between files.