

```

(*)
                                CS51 Lab 19
                                2-dimensional cellular automata
*)

module G = Graphics ;;

  (*****
  Do not change either of the two module type signatures in this
  file. Doing so will likely cause your code not to compile
  against our unit tests.
  *****)

(*.....
Specifying automata

A cellular automaton (CA) is a square grid of cells each in a
specific state. The grid is destructively updated according to an
update rule. The evolving state of the CA can be visualized by
displaying the grid in a graphics window.

In this implementation, a particular kind of automaton, with its
state space and update function, is specified by a module satisfying
the 'AUT_SPEC' signature. *)

module type AUT_SPEC =
  sig
    (* Automaton parameters *)
    type state                (* the space of possible cell states *)
    val initial : state       (* the initial (default) cell state *)
    val grid_size : int       (* width and height of grid in cells *)
    val update : (state array array) -> int -> int -> state
                                (* update grid i j -- returns the new
                                state for the cell at position 'i, j'
                                in 'grid' *)

    (* Rendering parameters *)
    val name : string          (* a display name for the automaton *)
    val side_size : int        (* width and height of cells in pixels *)
    val cell_color : state -> G.color
                                (* color for each state *)
    val legend_color : G.color (* color to render legend *)
    val font : string option   (* optional font for legend as per
                                Graphics module *)
    val render_frequency : int (* how frequently grid is rendered
                                (in ticks) *)
  end ;;

(*.....
Automata functionality

Implementations of cellular automata provides for the following
functionality:

* A current (mutable) grid that can be modified ('replace_grid').

* Creating additional fresh grids ('fresh_grid').

* Initializing the graphics window in preparation for rendering
  ('graphics_init').

* Mapping the automaton's update function simultaneously over all
  cells of the current grid ('update_grid').

* Running the automaton using a particular update function, and

```

rendering it as it evolves ('run\_grid').

as codified in the 'AUTOMATON' signature.

Note the difference between the argument structure of functions in 'life.ml', where the grid is passed in as argument to several functions, as compared to the approach here, where there is a single mutable current grid that is updated by the functions, so that no grid argument is required.

\*)

```

module type AUTOMATON =
  sig
    (* state -- Possible states that a grid cell can be in *)
    type state

    (* grid -- 2D grid of cell states *)
    type grid = state array array

    (* current_grid -- The current grid of the appropriate size as per
       the spec, which is initialized with all cells being in the
       initial state. It can be modified directly or by 'replace_grid'
       or updated with the automaton's update rule. *)
    val current_grid : grid

    (* fresh_grid () -- Returns a fresh grid of the appropriate size
       as per the spec, initialized with all cells being in the
       initial state. *)
    val fresh_grid : unit -> grid

    (* replace_grid new_grid -- Destructively changes the current grid
       ('current_grid', above) to 'new_grid'. *)
    val replace_grid : grid -> unit

    (* graphics_init () -- Initialize the graphics window to the
       correct size and other parameters. Auto-synchronizing is off,
       so our code is responsible for flushing to the screen upon
       rendering. *)
    val graphics_init : unit -> unit

    (* update_grid () -- Updates the current grid one "tick" by
       updating each cell simultaneously as per the CA's update
       rule. *)
    val update_grid : unit -> unit

    (* run_grid grid -- Initializes the current grid with 'grid' and
       repeatedly updates it using the automaton's update rule rule,
       rendering the grid state to the graphics window until a key is
       pressed. Assumes graphics has been initialized with
       'graphics_init'. *)
    val run_grid : grid -> unit
  end ;;

(*.....
Implementing automata based on a specification

Given an automata specification (satisfying 'AUT_SPEC'), the
'Automaton' functor delivers a module satisfying 'AUTOMATON' that
implements the specified automaton. *)

module Automaton (Spec : AUT_SPEC)
  : (AUTOMATON with type state = Spec.state) =
  struct
    (*.....
    COMPLETE THE IMPLEMENTATION OF THIS FUNCTOR. You'll undoubtedly

```

```
    find the code in 'life.ml' useful. Much of the needed code can be  
    cut and pasted from there, with appropriate modifications.  
    .....*)  
end ;;
```