Département SE / 3A (S5)
Programmation Structurée

Project: the Game of Life

1 Preamble

Conway's *Game of Life*[1], is a process that simulates the life and death of cells. The game's universe is a two-dimensional grid of square cells. Each cell has two possible states: *live* or *dead*. Every cell interacts with its eight neighbours (horizontally, vertically, and in diagonal). At each step of the game, also called *generation*, cells live or die according to the following rules:

- Any cell with one or zero live neighbours dies;
- Any cell with two or three live neighbours lives on;
- Any dead cell with exactly three live neighbours becomes live;
- Any cell with four or more neighbours dies.

2 Functionnalities

In this project you will develop a program that implements the Game of Life. Your program must provide the following base functionnalities:

1. *Initialize*: load the initial pattern, also called *seed*, from a specified input file;
2. *Display*: display the current state of the game;
3. *Evolve*: compute the next generation;
4. *Animate*: display the successive generations of the game. For more convenient visualization, we recommend clearing the screen between each generation, using the function call `system(clear)`. You must provide two animation modes:
   (a) *Interactive*: the next generation is displayed whenever the user strikes a key;
   (b) *Automatic*: the next generation is displayed automatically after a given amount of time elapses (see the manual of function `sleep`, by typing `man 3 sleep` in your terminal).

Once the base functionnalities have been developped, implement the following extensions:

- *Random seed*: the seed is generated randomly. The probability for a cell to be live is specified as a percentage by the user;
- *Periodic universe*: consider that the left and right edges of the grid are stitched together, and the top and bottom edges too;
- *Custom life/death rules*: let the user specify what number of live neighbours cause a cell to live or die (try for instance the *Highlife* variant[2]).

[1] `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`
[2] `https://en.wikipedia.org/wiki/Highlife_(cellular_automaton)`

3 Resources

We provide an Application Programming Interface (API) for loading a seed. It is available in directory `~jforget/Cours/PS/Projet`. The directory contains:

- `src/inout.c`, `src/inout.h`: these are the main API files. You do not need to understand the implementation in `inout.c`, you only need to read the function documentation in `inout.h`;
- `src/Makefile`: this file automates the compilation process. To compile your program, simply type `make` in your terminal. Your usual compilation command (`clang/gcc`) *will not work*;
- `src/life.c`: a simple usage example. Write you code in this file;
- `seeds`: a directory with some example seeds.

4 Evaluation

You must work as a pair with another student (binômes). No trio is allowed. This student must be in the same group as you (TP group). This implies that some students will work on their own. Evaluation will be more generous to them.

We will evaluate your work based on an archive that must respect the following constraints:

- The archive must be in either format `.zip` or `.tgz`;
- Store the archive on the NextCloud website of the University: `https://nextcloud.univ-lille.fr/`
- Share the file (via nextcloud) with:
  `julien.forget@polytech-lille.fr`
  `laurent.grisoni@polytech-lille.fr`
  `tanguy.navez@inria.fr`
- The deadline for the deposit is 26-01-2023 (any time of the day).

The archive must contain:

- Your source code (**no compiled files**);
- A report, maximum 10 pages, that presents your work and details:
  - Your main algorithms;
  - What is implemented and what is not implemented;
  - The limitations of your program;
  - Any comment/explanation that helps understanding your work.

The following criteria will be taken into account to evaluate the quality of your code:

- Time/space complexity;
- The program compiles without warnings with option `-Wall`;
- The program executes correctly, and causes no segmentation fault;

- Code is correctly indented;
- No magic numbers (check the web in case you do not know this term);
- Variable and procedure names are explicit;
- Difficult points are (briefly) commented;
- Code is structured in procedures;
- No code duplication.