CS211 | **Computers and Programming** | **Handout 3**

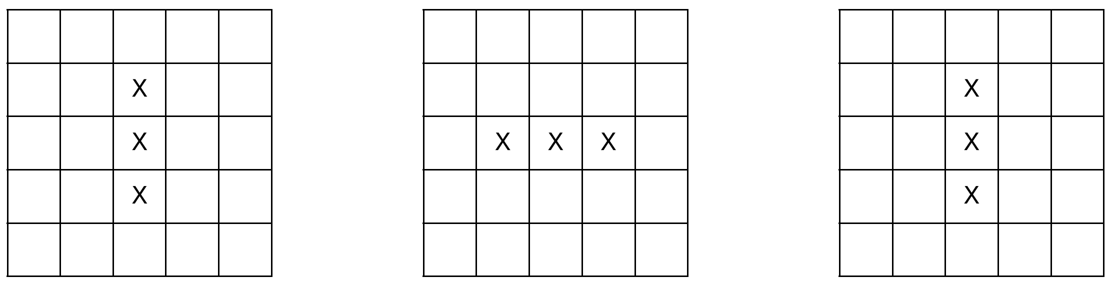
ASSIGNMENT 2: Object-oriented Programming

Fall 1999 | **September 9, 1999.**

DUE DATE: September 23.

Read the ***whole*** *assignment before attempting the implementation.*

Conway's Life



Initial generation — Generation no.1 — Generation no.2

Figure 1

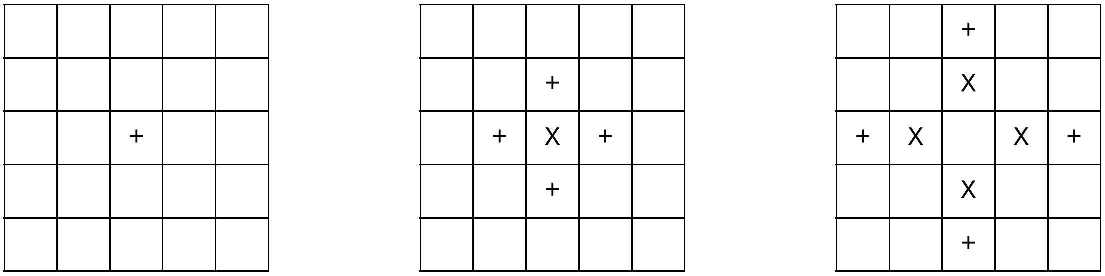
The Life Automaton is run by placing a number of filled cells on a 2-D grid. Each generation then switches cells on or off depending on the state of the cells that surround it. The rules are defined as follows. All eight of the cells surrounding the current one are checked to see if they are on or not. Any cells that are on are counted, and this count is then used to determine what will happen to the current cell.

Rules:

1. *Death*: if the count is less than 2 or greater than 3, and the current cell is on, the current cell is switched off.
2. *Survival*: if (a) the count is exactly 2, or (b) the count is exactly 3 and the current cell is on, the current cell is left unchanged.
3. *Birth*: if the current cell is off and the count is exactly 3, the current cell is switched on.

Note that the number of neighboring cells depends on the location of the current cell in the grid. A cell on the grid edge will have fewer than eight neighbors, but the above rules still apply.
Generations shown in Figure 1 are based on the above rules.

Variation on Life



Initial generation   Generation no.1   Generation no.2

X indicates a live cell which survived from the previous generation
+ indicates a live cell which is born in the present generation

Figure 2

Figure 2 shows a variation on the standard Life Automaton. The first difference is that the neighbors are now only the *horizontally and vertically* adjacent cells. The diagonally adjacent cells are excluded from the neighbor count. Secondly, some of the rules rely on the number of generations a cell has been on.

Rules:

1. *Death*: if the current cell has already survived 2 generations, the current cell is switched off.
2. *Survival*: if the current cell has survived exactly 1 generation, the current cell is left unchanged.
3. *Birth*: if the current cell is off and the count is exactly 1, the current cell is switched on.

The Assignment

Implement a program that allows the user to play her choice of the game of Life, as described below.

Design Decisions

Using Interfaces

The major differences between the two variations of game of Life are the three rules concerning death, survival and birth of a cell. Any cell that implements these rules could be used to play a game of Life. You should specify an interface `ILifeRules` which captures this design decision.

```
interface ILifeRules {
    boolean deathRule();
    boolean survivalRule();
    boolean birthRule();
    ...
}
```

The Grid
The grid can be implemented as a *matrix*, (i.e. as an array of array), of cells implementing `ILifeRules`:

```
ILifeRules[][] grid;
```

The Cell
A cell can keep track of how many generations it has been on. This information can be used to deduce whether it is on in the *present* generation.
Since a cell needs to know the number of neighbors it has, this information can also be a part of the state of a cell.
When computing the *next* generation, the current status of a cell cannot be changed immediately because this would impact those cells that have *this* cell as a neighbor. For this reason a cell could have a variable to indicate what happens to it in the next generation. When the next generation has been computed for *all* cells, the status of all cells can be updated.

Calculating the Next Generation
You should implement a method `step` which takes `ILifeRules[][]` as parameter and calculates the next generation.

The User Interface

We will not implement any graphical user interface (GUI) in this assignment. Interaction will be via the command line and by commands given at the terminal. Isolate the operations for the user interaction as much as possible so that they can be, for example, implemented with a graphical user interface at a later stage.

The Size of the Grid
The size of the grid is specified on the command line when the program is started. The format is `nxm`, there `n` and `m` are the number of rows and columns respectively in the grid.
Example of a session using a 10 by 12 grid:

```
java LifePlayer 10x12 ...
```

Choice of Game
The program should read the version of the game to play from the command line when the program is started. The options `C` and `V` respectively indicate Conway's version and the variation outlined above.
Example of a session using a 10 by 12 grid to play by Conway's rules:

```
java LifePlayer 10x12 C ...
```

The Initial Generation
The initial generation should be read from a file whose name is specified on the command line when the program is started.
Example of a session using a 10 by 12 grid to play by Conway's rules, and read the initial generation from a file named `initLife.data`:

```
java LifePlayer 10x12 C initLife.data
```

Each cell which is on is specified as a pair `(i,j)` on a separate line in the file. For example, the input file for the initial generation for Figure 1 would be as follows:

```
1 2
2 2
3 2
```

The User Interaction

The user interaction will be via simple commands given at the terminal, *with the program printing the current generation after each command*.

`+`	Do one generation.
n	Do `n` generations, there `n > 1`.
`q`	Quit

The current generation can be printed as a grid, *but without the ruling around the cells*. Example of the output for the second generation in Figure 2 is shown below.

```
Generation no. 2
---------------
       +
       X
 +  X     X  +
       X
       +
---------------
```

What to hand in:

Hand in design documentation (preferably class diagrams and collaboration diagrams) and the code (preferably documented with javadoc comments), and the sample printout of any 3 generations computed by running the two versions of Life on the initial inputs shown in Figure 3.

	x						
	x	x	x				
	x		x				
			x				

(a)

		x							
		x							
		x							
			x	x	x				
		x							
		x							
		x							

(b)

Figure 3