Pre-Project NMCGJ 2025-2026: Conway's Game of Life

Conway's Game of Life is a very simple life simulator. The rules of the game were invented by the British mathematician J. H. Conway in 1970.

Rules of the Game

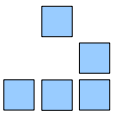
The universe of the Game of Life is an infinite two-dimensional grid of square cells (this is the game board). Any of these cells can be in one of two possible states: alive or dead. A step in the game can be seen as the creation of a new generation of cells. Each cell interacts with its eight neighboring cells, and its state changes according to the following rules:

1. Any live cell with two or three live neighbors lives on to the next generation.
2. Any dead cell with exactly three live neighbors becomes a live cell (*reproduction*).
3. Any live cell with fewer than two live neighbors dies (*under-population*).
4. Any live cell with more than three live neighbors dies (*over-population*).

These simple rules often create funny behavior of a population, i.e., a group of live cells. For example, take the following configurations and check how they evolve:

1. The *Block* is a static population:

2. The *Glider* travels around the board:



3. The *Blinker* is a small oscillator:

Even though the rules are simple, it is not so easy to predict the behavior of this game: it creates a so-called chaotic system.

Problem

This project aims to implement Conway's Game of Life.

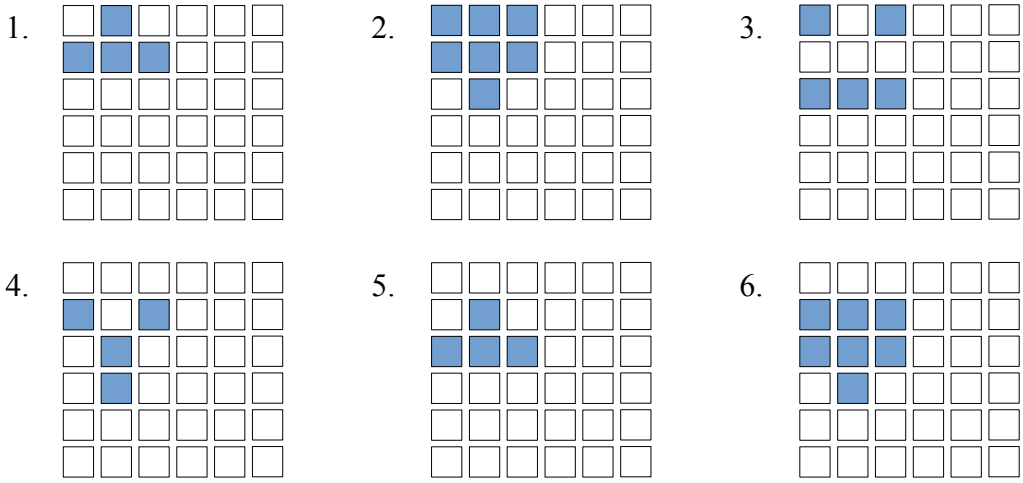
Game board

Instead of working with an infinite board, we restrict ourselves to a finite board of an a priori chosen size. On each position of the board there can be a cell, alive or dead. This means that the board can be represented internally as a matrix of boolean values.

Boundary rules

We follow the 4 rules described above to generate a new generation of cells. However, since we are working with a finite board, we still need to specify the behavior at the boundary of the board. We assume that there is nothing alive outside the board, and there cannot be created new life outside the board as well.

Example evolution:



Input/Output

In order to start the game, there must be live cells present on the board (life cannot be created out of nothing). One possibility could be to ask the user via the console the coordinates of all initial live cells. However, this is a quite cumbersome procedure, especially when the initial configuration has to be repeated. It would be easier if the coordinates were read from a file.

The class `CoordinateIO` is developed for such a reading job. It provides functionality to read a sequence of coordinates from a text file and to store them into a list. It can also write a list of coordinates to a text file. This class makes use of some other classes as `Coordinate` and `File`. The class `File` belongs to the library `java.io.*`, and consists of an abstract representation of a file. The class `Coordinate` is a container of an integer x-coordinate and an integer y-coordinate. This class can be used to represent the position of the live cell, or also to represent the dimension of the board.

For example, a *Blinker* configuration could be stored in a text file as follows.

```
20 40 // dimension of board
5 5   // position of cell 1
5 6   // position of cell 2
5 7   // position of cell 3
```

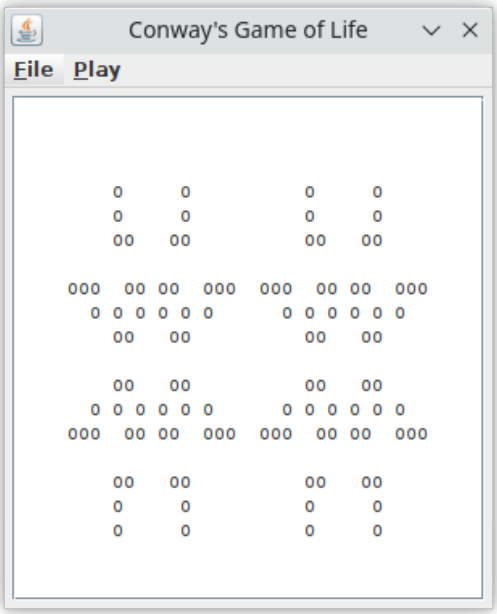
Each line in the text file must start with two integer values (x-coordinate and y-coordinate), separated by a space character. After them, some comments can be added as long as they are separated from the second integer by a space character.

The class `CoordinateIO` has the following methods:

setFile(File file)	Sets the text file for coordinate I/O
read(): ArrayList<Coordinate>	Reads the coordinates from the text file and puts them in a list.
write(ArrayList<Coordinate> list)	Writes a list of coordinates to the text file.

Visualization

The evolution of the Game of Life can be visualized with a text output. For example, a dead cell can be represented by means of the space character " ", and a life cell can be represented by means of the letter "o".



In order to simplify the visualization task, the class `GameOfLifeFrame` is already partially prepared. It is a subclass of the class `TextFrame` that we used before. Hence, it provides a large text area, on which the cells can be represented. To steer the game through the menu bar of the window, the following methods should be implemented.

clearText()	Clears the game board (called by the menu File > New).
loadTextFile(File file)	Loads the content from a text file to the game board (called by the menu File > Open).
saveTextFile(File file)	Saves the content from the game board to a text file (called by the menu File > Save).
play()	Creates and visualizes the next generation of the cell population (called by the menu Play).

Since the class `GameOfLifeFrame` is a subclass of the class `TextFrame`, the following useful methods are available.

start()	Visualizes the frame.
close()	Closes the frame.
setResizable(boolean resizable)	Indicates whether the frame is resizable by the user.
setTextEditable(boolean editable)	Indicates whether the text field is editable or not.
getText(): String	Gets the content of the text field.
setText(String text)	Sets the content of the text field.
setTextDimension(int rows, int cols)	Sets the preferred dimension of the text field.

Notes

- A good Java program is not just a program that produces "the right result"; it should also be designed properly. In a good program design, every class should be responsible for a single well-defined job. For example, the class `GameOfLifeFrame` is mainly responsible for the graphical user interface (GUI) of the game. Therefore, this class should not take care of the computations involved in the game, but should delegate this functionality to another class.
- Do NOT modify the pre-defined classes `TextFrame`, `CoordinateIO`, and `Coordinate`. Of course, you must update the class `GameOfLifeFrame`.

Practical Information

The deadline for the pre-project is **November 30, 2025**. It is not mandatory to make the pre-project, but it is strongly recommended.

The only way to master a programming language is by practicing a lot. This pre-project offers an opportunity to make an exercise on a larger scale. It is considered as part of the learning process, and therefore there is no formal evaluation (as part of the exam). Of course, the next project will be evaluated!

If you would like to get feedback, then turn in your report of the pre-project before the deadline. This can be done electronically by sending an email to **speleers@mat.uniroma2.it**. Such a report should include:

1. the source code of your program;
2. a class diagram of your program;
3. an overview of your program design decisions.

Good luck and have fun!

Hendrik Speleers