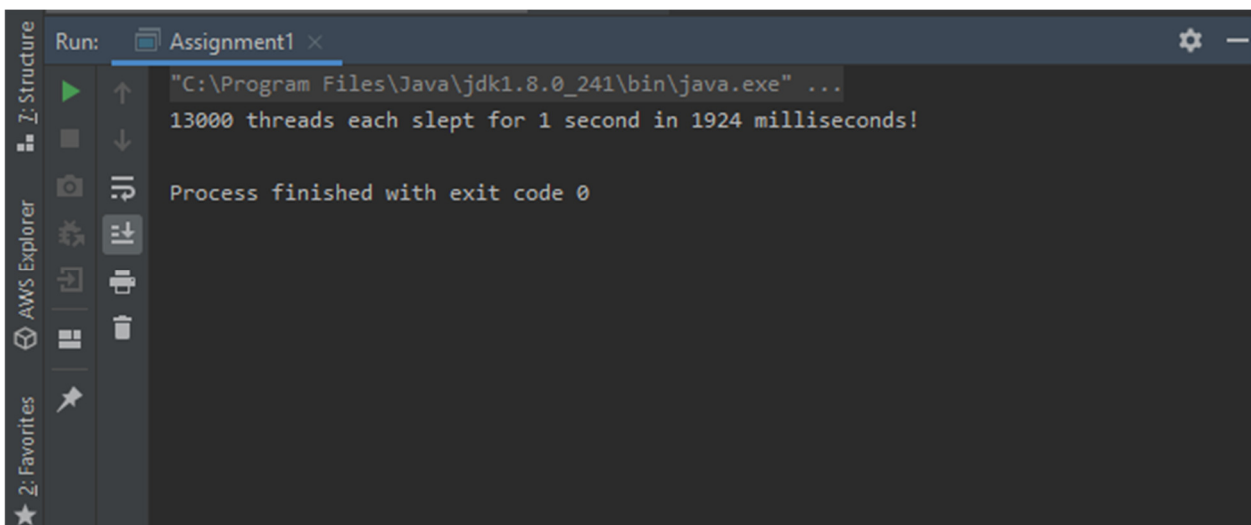


1. Everyone loves sleeping

Just as you would prefer to sleep whenever given the chance, so should programs do too when they have nothing better to do. If there is nothing useful for them to do, it's good practice telling them to sleep instead of wasting CPU cycles. Based on that knowledge, let's run an experiment.

Just as you want to cram as much sleep as possible in a day, so does your program. Let's assume that you are only allowed to sleep for 1 second at a time. How many seconds of sleep could you cram in 2 seconds of running your program? If you do not use multithreading, you will unfortunately not be able to even sleep for 2 whole seconds, as that would put you over the limit. Using more threads and setting them each to sleep for 1 second, should let you sleep for 1000s of seconds instead of just 1.

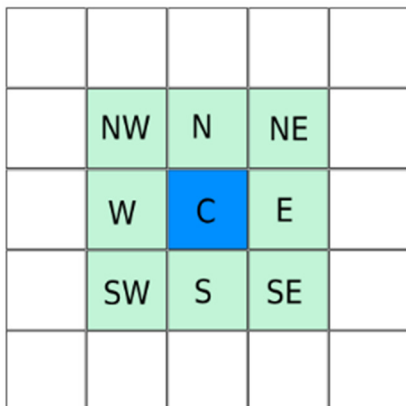
To do: How many seconds can you get your program to sleep in less than 2 seconds?



```
Run: Assignment1 x
"C:\Program Files\Java\jdk1.8.0_241\bin\java.exe" ...
13000 threads each slept for 1 second in 1924 milliseconds!
Process finished with exit code 0
```

2. Life is a game - Game of Life

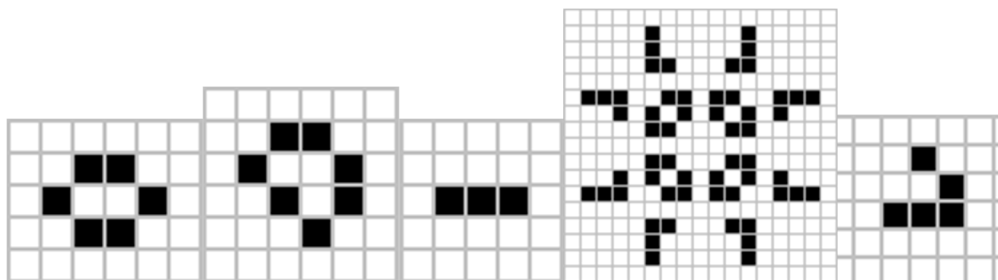
The game of life is a very well known game, more formally known as Conway's Game of Life, is what is known as a zero player game. You just give it some starting state as input and you get an output state. The game takes place in cells in a grid. Any cell that's "alive" has some marking on it: X, O or something else signifying it's alive. Any cell that's "dead" has nothing inside. Every cell has a number of neighbors, if the cell is located at an edge it can have fewer, but in general, it has 8 neighbors:



You can make the grid as big as you like, the rules for how the game progresses each round stay the same and are as follows:

- Any alive cell with less than 2 alive neighbors, dies
 - Any alive cell with 2 or 3 alive neighbors, lives
 - Any alive cell with more than 3 neighbors, dies
 - Any dead cell with exactly 3 neighbors, becomes alive
- Implementing the aforementioned rules, will result in patterns emerging, they can be still, oscillating (changing but staying place), or "moving" across the grid.

- Examples



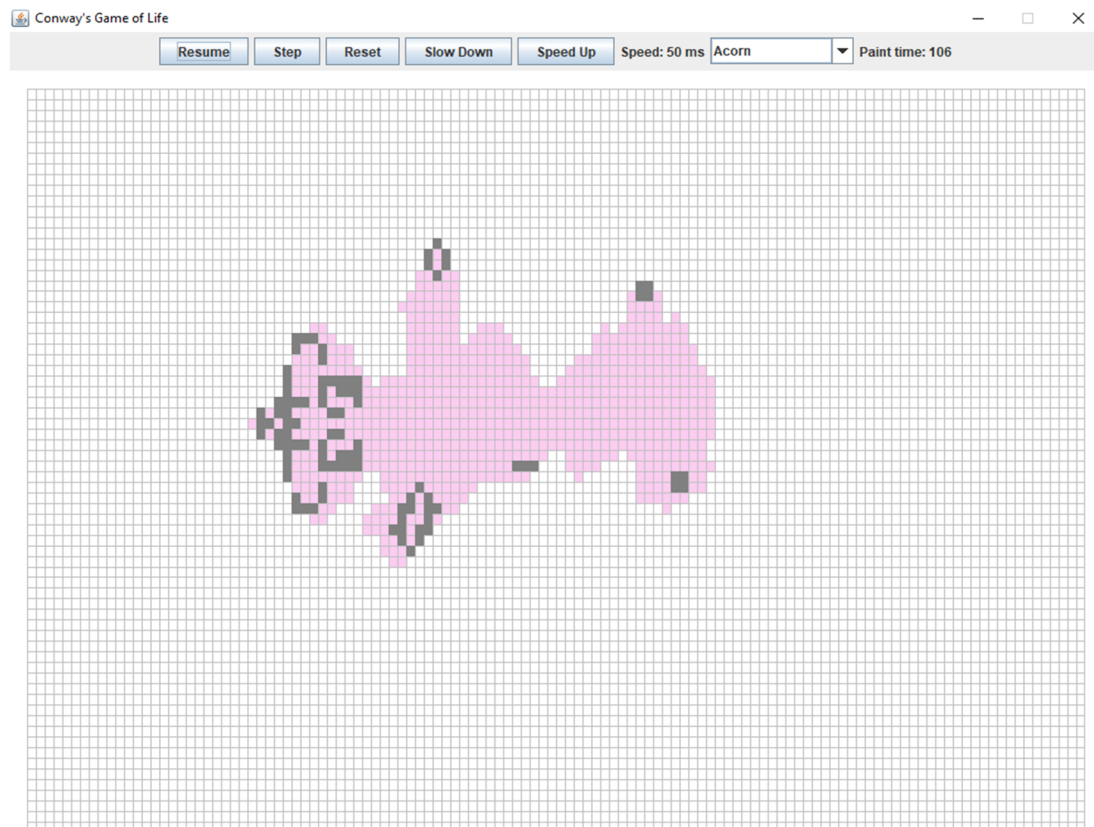
As you can probably tell, doing this on every cell sequentially could be resource consuming, especially as you are getting a bigger and bigger grid. So instead of just one thread for the whole thing, let's create one thread for each cell instead. The problem that rises now, is that you will need some way to share information between your threads (a buffer) and need to concern yourselves with advancing to the next generation ONLY after every cell in the grid is done, otherwise you will get different cells advancing at different speeds and getting wrong results.

You can display the grid in either a very simple GUI of your making or even in the terminal window. The size of the grid will be established by the input file. You can choose how to display the progress of the game.

The code attached as an example solution is largely taken from this Repository: <https://github.com/allforcode/Conway-Game-of-Life>

I have only removed some stuff that are of no concern to our course like testing and such. In this example, the starting patterns are hardcoded and the user chooses between them. White cells are considered dead, grey ones are considered alive, pink means that they have been alive in the past, but are dead now. It is still more complex than required for this exercise, but a good read nonetheless

Your goal is to make a working version of the game of life and to make each cell be its own thread.



GRADING AND SUBMISSION

Compress all the files, folders and subfolders into a **zip**, **rar**, **7z** file, and then upload it via the Assignment page on Canvas. Make sure that you submit the correct version of your project

and that you have compiled and tested your project before handing in. Be careful not to use any hard-coded file paths (for example path to an image file on your C-drive) in your source code. It will not work on other computers. Projects that do not compile and run correctly, or is done with poor code quality, will be returned for completion and resubmission.

After or before submitting your assignment to the module, show your assignment to your lab leader during the scheduled hours in the labs.