COMP 201 – Fall 2021

Assignment 3: Game of Life

KOÇ UNIVERSITY

Assigned 8 November 2021 23:59, Due: 25 November 2021 23:59

Mustafa Orkun Acar (*macar20@ku.edu.tr*) is the lead person for this assignment.

1 Introduction

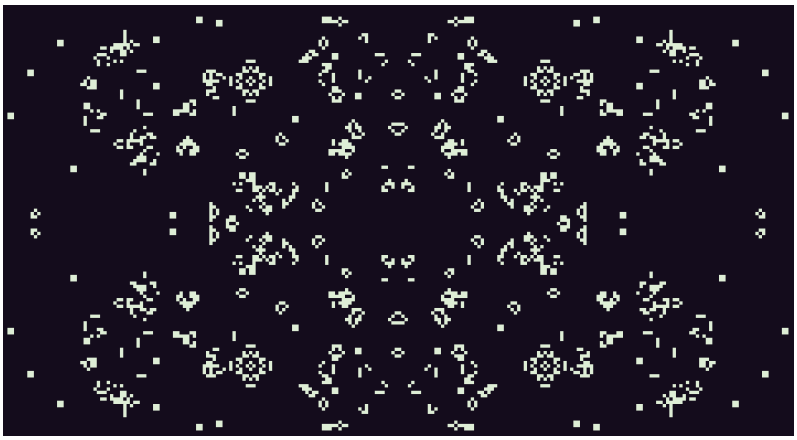

Figure 1: A view of a state from the Game of Life

The purpose of this assignment is to provide you with experience in working with pointers and dynamic memory allocation.

The Game of Life is a cellular automaton devised by the British mathematician John Horton

Conway in 1970. It is the best-known example of a cellular automaton. The "game" is actually a zero-player game, meaning that its evolution is determined by its initial state, needing no input from human players. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live (X) or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. A square that is on the border of the grid has fewer than eight neighbors.

In the following example, the red colored middle location has four "live" neighbors:

	X	
X		
X	X	

At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbours dies, as if by needs caused by under population.
- Any live cell with more than three live neighbours dies, as if by overcrowding.
- Any live cell with two or three live neighbours lives, unchanged, to the next generation.
- Any dead cell with exactly three live neighbours cells will come to life.

The initial pattern constitutes the seed of the system. The next generation is created by applying the above rules simultaneously to every cell in the seed - births and deaths happen simultaneously. The rules continue to be applied repeatedly to create further generations [Taken from Wikipedia].

To illustrate these two patterns below should alternate forever:

Table 1: Initial states of cells:

X	X	X

Table 2: After the first iteration:

	X	
	X	
	X	

In the initial state (Table-1) the cells on the left and on the right have only one neighbour which is the middle cell, so they die. The middle cell has two neighbours, left and right, so it stays alive. The top and bottom cells have 3 neighbours (middle, right and left) so they become alive. All the decisions about live or die of cells are taken at the end of the iteration, so the evaluation order of cells does not matter. After the first iteration the cells look like in Table-2. Running one more iteration turns the cell states in initial state (Table-1). So, if we start the game with this configuration, the game enters an infinite loop, namely a *blinker*.

There are also stationary patterns called still lifes which do not change from a generation to the next.

Table 3: A still life example, *Block*

	X	X	
	X	X	

2 Programming Task

2.1 How to start

- Accept the GitHub Classroom assignment using the link: **https://classroom.github.com/a/MQI-5jvG**
- Clone the GitHub repository created for you to a Linux machine in which you plan to do your work (Using Linux servers [linuxpool.ku.edu.tr]. See Section 5 for details is suggested.)

```
$ git clone https://github.com/COMP201-Fall2021/assignment-3-USER.git
(Replace USER with your GitHub username that you accept the assignment).
```

2.2 Task:

You are going to implement a version of the game of life. The checkerboard in Conway's game of life can be represented as a two-dimensional matrix. Given an initial pattern, the game simulates the birth and death of future generations using simple rules.
Below, we described the specifications of the simulation and input parameters.

2.2.1 Input Parameters

This section describes the input parameters of your code. You will get the initial state of the grid from one of the provided input text files, which follow a particular format.

In each input file, the first three lines will contain integers, i, r and c, representing the number of iterations, number of rows and columns in the grid, respectively. The next lines of the file will contain the grid itself, a set of characters of size r x c with a line break (\n) after each row. Each grid character will be either a "-" for an empty dead cell, or a "X" for an alive cell.

The input files are in inputs folder which is in the same working directory as your program. For example, the following text might be the contents of an example file ex1.txt, a 10x10 grid with 9 live cells:

Input :
1 // number of iterations
10 // number of rows
10 // number of columns
```
----------
---XX-----
----X-----
----------
----------
---XX-----
--XX------
-----X----
```

```
----X-----
----------
```

2.2.2 Output

This section describes the output of the task. Your code should print the death count in the step, born cell count in the step and state of the grid. There must be an output file for each executed step of the game. A sample output file ex1_out_iteration_1.txt can be seen below:

Output:
3
4
```
----------
---XX-----
---XX-----
----------
----------
--XXX-----
--XX------
---XX-----
----------
----------
```

Note: After an update step, it is possible that some cells outside of the grid dimensions become alive. Note that we are only interested in cells inside the initial grid's borders.

2.2.3 Run

Your could should take two command line arguments: input file path and the output folder path, in which you should generate output files.
./game_of_life diamond/diamond.txt output/

2.2.4 Bonus Task

There are some initial patterns called "oscillators" which returns to its original state, in the same orientation and position, after a finite number of generations. The bonus task is to implement a boolean function, IsOscillator(), given the initial pattern and maximum number of steps to check, determines whether the pattern is an oscillating object. **Hint:** You can simulate the game for given maximum steps and collect the results in a 3D dynamically allocated array structure. Using this "game history", it is possible to check for duplicate game states.

3 Evaluation

You are not allowed to use static arrays. Your implementation must utilize dynamic memory allocation.

Your score will be computed out of a maximum of 100 points based on the following distribution:

- **80** Correctness points
- **10** Effective use of version control points
- **10** Style points

- **20** Bonus task

Correctness points: Your code will be evaluated based on a set of given input parameters and should result in correct numbers for: 1. Death count at each step 2. Born count at each step. 3. *.txt files that demonstrate each step's status using '-' and 'X' characters.

Effective use of version control points: You are required to push your changes to the repository frequently. If you only push the final version, even if it is implemented 100% correctly, you will lose a fraction of the grade because you are expected to learn to use Version Control Systems effectively. You do not have to push every small piece of change to GitHub, but every meaningful change should be pushed. For example, each of the functions coded and tested can be one commit. **For each function, there should be at least one commit (with proper commit message) that includes just modifications on that function.**

Style points: Finally, we've reserved 10 points for a subjective evaluation of the style of your solutions and your commenting. Your solutions should be as clean and straightforward as possible. Your comments should be informative, but they need not be extensive.

4 Handin Intructions

Same as previous assignments, we use GitHub for the submissions as follows. Note that we want you to get used to using a version management system (Git) in terms of writing good commit messages and frequently committing your work so that you can get the most out of Git.

- Commit all the changes you make:

```
$ git commit -a -m "commit message"
(Note: please use meaningful commit messages.)
```

- Push your work to GitHub servers:

```
$ git push origin main
```

5 Advice

- Keep your GitHub repository by frequently committing the changes.
- Don't forget GDB and Valgrind since they can save a lot of time in debugging.
- Use linuxpool.ku.edu.tr Linux servers to test your code in order to avoid compatibility issues.
- You can write a makefile for your own project that helps you compile and run your code faster.

6 How to use linuxpool.ku.edu.tr linux servers

- Connect to KU VPN (If you are connected to the KU network, you can skip this step.) See for details:
- Connect to linuxpool.ku.edu.tr server using SSH (Replace USER with your Koc University username):

```
$ ssh USER@linuxpool.ku.edu.tr
(It will ask your password, type your Koc¸ University password.)
```

- When you are finished with your work, you can disconnect by typing:

```
$ exit
```

Your connection to the server may drop sometimes. In that case, you need to reconnect.

We advise you to watch the following video about the usage of SSH, which is used to connect remote servers, and SCP, which is used to transfer files between remote servers and your local machine:

7 Academic Integrity

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the web material as everything on the web has been written by someone else. See Koç University - Student Code of Conduct.

8 Late Submission Policy

You may use up to 7 grace days (in total) over the course of the semester for the assignments. That is, you can submit your solutions without any penalty if you have free grace days left. Any additional unapproved late submission will be punished (1 day late: 20 % off, 2 days late: 40 % off) and **no submission after 2 days will be accepted.**

9 Regulations

- **Blackboard:** This is an individual assignment, and all hand-ins are electronic. Any clarification or correction will be announced on the Blackboard.
- **Cheating:** We use automated plagiarism detection to compare your assignment submission with others and also the code repositories on GitHub and similar sites. Moreover, we plan to ask randomly selected 10% of students to explain their code verbally after the assignments are graded. And one may lose full credit if he or she fails from this oral part.

10 Sources

- Inventing Game of Life (John Conway) - Numberphile (on Youtube)