ECE École d'Ingénieurs Paris, France
Department of Embedded Systems

Advanced C Programming
by Aakash SONI

Lab : Game of Life

Overview

The Game of Life, invented by John Conway in 1970, is an example of a zero-player "game" known as a cellular automaton. It consists of a two-dimensional world, divided into "**cells**". extending infinitely in all directions.
Each cell is either "**dead**" or "**alive**" at a given "**generation**". The game has a set of rules that describe how the cells evolve from generation to generation.
Based on these rules the **state of a cell in the next generation** is calculated as a function of its **neighboring cells' states in the current generation**.
In a 2 dimensional world, a cell can have upto 8 neighboring cells : vertically, horizontally, and diagonally adjacent to that cell.

The set of rules described by Conway are summarized as:

1. A living cell will die if it has fewer than two living neighbors. (case of solitude)
2. A living cell will also die if it has more than three live neighbors. (case of overcrowding)
3. A living cell will survive if it has exactly two or three living neighbors.
4. A dead cell will become alive if it has exactly three living neighbors.

In this lab, we will be implementing Conway's Game of Life, with a restriction that our world is a 2-D finite space. In this case, the neighbors of a cell on the "edge of the world" that would be beyond the edge are assumed dead.

For more information about the game, refer to Wikipedia at ae
http://en.wikipedia.org/wiki/Conway's_Game_of_Life
and youtube at
https://www.youtube.com/watch?v=CgOcEZinQ2I

Part A: Implementing Evolution (In-Lab)

In this part, we will focus on implementing the rules needed to evolve the world from one generation to the next. To assist you, I provide several functions you should use to access and modify the current and next state of the world. These functions are declared in the header file lifegame.h and implemented in the file lifegame.c.
Also, I have provided a skeleton describing **what you need to do for this part in lab1a.c.**

Before getting started, you should download all these files from the campus (*pedago-ece.campusonline.me*) into your working directory.
For compilation, you need to compile ***lab1a.c*** **and** ***lifegame.c*** **together to generate a single executable file** (let's call it lab1a.o) with all the code in it (otherwise, you'll get "undefined reference" errors).

Example for compiling this code:

```
gcc -g -O0 -Wall lab1a.c lifegame.c -o lab1a.o
```

Start by examining the contents of lifegame.h and lab1a.c.

Your Job :
You need to fill in a few lines in ***main()*** and complete the functions ***next_ generation()***, ***get_next_state(x,y)***, and ***num_neighbors(x,y)***.

Note : There is **no need to modify the files lifegame.h or lifegame.c** in this part.

- Think about a way to initialize the world for the Game of Life? Write the appropriate code in the ***main()*** function.
- How will you output the final state of the world once all the evolutions are done? Write the appropriate function call in ***main()***.
- The ***main()*** function calls ***next_generation()*** for each generation to handle the evolution process. Your code should set each cell's state in the next generation according to the rules specified in the Overview of this lab. Once the states of all the cells have been set for the next generation, calling ***finalize_evolution()*** will set the current world state to the next generation and reset the next generation state. Your code should make use of the ***get_next_state(x,y)*** function to compute the next state of each cell.
- Write the code for ***get_next_state(x,y)***, so the function returns the next state (ALIVE or DEAD) of the cell at the specified coordinates using the number of live neighbors (returned by the ***num_neighbors(x,y)*** function) and the Game of Life rules.
- Fill in the function ***num_neighbors(x,y)***, so it returns the number of live neighbors (cells vertically, horizontally, or diagonally adjacent) for the specified cell. Since our world is finite, adjacent cells which are beyond the edge of the world are presumed DEAD.

Now that you're done, compile and run the program. Feel free to change the definition of *NUM_GENERATIONS*, but when you're ready to be checked off, make sure *NUM_GENERATIONS = 50* and ask me to check your program's output.

At this point, I consider that you completely understand how the code works. Feel free to modify ***initialize_worldfunction()*** in **lifegame.c** file to try different patterns and observe their evolution over generations.

Part B: The World in a File

In the first part of this lab, the initial state of the world was hard-coded into *lifegame.c* and the final state of the world was output to the console. In this part, we will modify the code so that the initial state of the world can be read from a file, and the final state is output to a file.
First, let's examine *lifegame.c*.
Notice the functions you need to implement: ***initialize_world_from_file(filename)*** and save world to ***file(filename)***.

- The first of these, ***initialize_world_from_file(filename)***, reads the file specified by ***filename*** and initializes the world from the pattern specified in the file. Basically, the file is a matrix of characters, **'*'** to specify a living cell, and ' ' (space) to specify a dead cell. The ith character of the jth line (zero-indexed) represents the initial state of the cell located at (i,j). If the line doesn't contain enough characters or the file doesn't contain enough lines, the unspecified cells are presumed dead. Fill in the function to read the file and initialize the world. Don't forget to reset all the next generation states to DEAD. Use appropriate error handling.
- The other function, ***save_world_to_file(filename)***, saves the final state of the world to the file ***filename*** in the same format as used in the initialization function: the ith character of the jth line (zero-indexed) represents the state of the cell located at (i,j) in the world.
- Fill in the contents of ***lab1b.c*** using the code from Part A ***(lab1a.c)*** and modifying it to call these functions. The name of the file to load will be specified in the first command line argument passed to the executable. If no file is specified, you should default to initializing the world to the hard-coded default "glider" pattern. Save the final output to the file "***world.txt***".

To finish, write a brief (5 pages max.) lab report describing your experience completing this lab, what challenges you faced and how you addressed them, and what you learned from this lab. Turn in a **zip file** containing **all your code *(lab1a.c, lab1b.c, lifegame.h,* and *lifegame.c)*,** and **your lab report**, using the online submission system on the campus.

Note: Do not copy your code in the report. Only pseudo-code, algorithm and flow diagrams are allowed.

Congratulations, you're done!

Important

You are welcome to discuss assignments and laboratory projects with other students, provided that all work turned in must be your own. If you do discuss your work with other students on assignments, don't forget to give them a credit by listing their names in the acknowledgement part of your report.
In summary, when you are turning in an assignment with your name on it, what you turn in must be your work, and yours alone. Cheating will not be tolerated.

Credits: Daniel Weller and Sharat Chikkerur at MITOCW