Life in True and False: Design Process

Isabella Phung

January 2023

1 Introduction

The Game of Life by Conway is a well known concept that serves as a simple but effective example of simulations. While it may seem deceptively simple at first, this and many more cellular simulations can be used to represent the behavior of single celled organisms and their gradual evolution. One might initially envision a bunch of random cells but the reality is that, thanks to the simple rules provided, Conway's Game of Life produces complex patterns thanks to the cells interacting with one another.There are many more cellular automations besides Conway's Game of Life, but it's generally agreed upon that this is the most well known of them. Additional reading in regards to cellular automations can be found in the Resources section of this document.

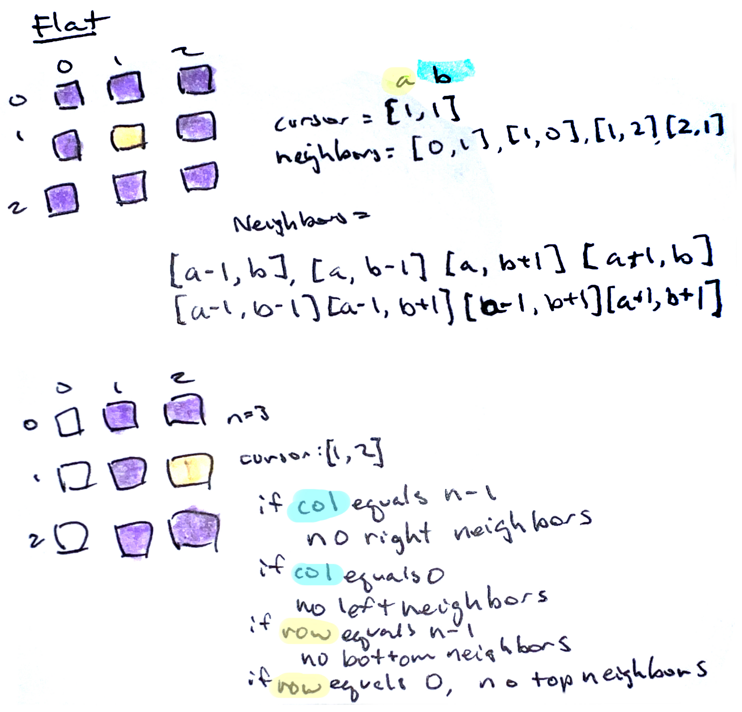
2 Pseudocode

Universe.h has been provided by the CSE13S instructors as a guideline for how to structure the Universe.c file. Life.c: Will require accepting string user input. In order to do this, fscanf will be required for the input, as will fprintf. These will handle file names for us without having to worry as much about things such as nonexistent files and the like. Handling the input and output will not be done with a set, but instead with a few variables since there aren't a lot of user input options.
Beyond that, this program will be the driver of this library. It will call the uv_populate function to create the universe, then run uv_census multiple times for the number of generations. It will also handle possible errors as the constructors will not focus on any error handling. It will also handle the ncursor output, which will require some fiddling to get working.
Universe.c:
Constructor: create matrix of indicated width and height of booleans to represent the universe. Bits will be allocated, in the event the allocation fails, I intend on using assert to crash the program.
Destructer: deallocate all boolean values and set pointers to null.
uv_rows: will return rows of universe.
uv_cols: will return number of columns of universe.

uv_live_cell: changes cell to true.
uv_dead_cell: changes cell to false.
uv_get_cell: returns state of cell
uv_populate: parses given file using infile and fscanf. First two items are row and height, can then call constructor to build matrix. Can then call uv_live_on following lines to populate the universe until the end of the file. Returns true if universe is successful. In the event that the population fails, I'm admittedly unsure if the constructor should handle that or if this function should handle it too. There are additional edge cases and errors that have to be addressed by this function. For example, what do we do if we have an extremely small toroidal function? Can a cell be its own neighbor? Nope, program will have to check if the neighbor cell is the same coordinates as the cursor cell. If the coordinate exceeds the size of the universe, then the program must say "Malformed Input". uv_census: This function is the critical aspect for the entire game to work. Firstly the rules indicated by the Conway are the following:

1. Live cells with 2-3 live neighbors survive.
2. Dead cells w/ 3 neighbors becomes live.
3. All other cells die.

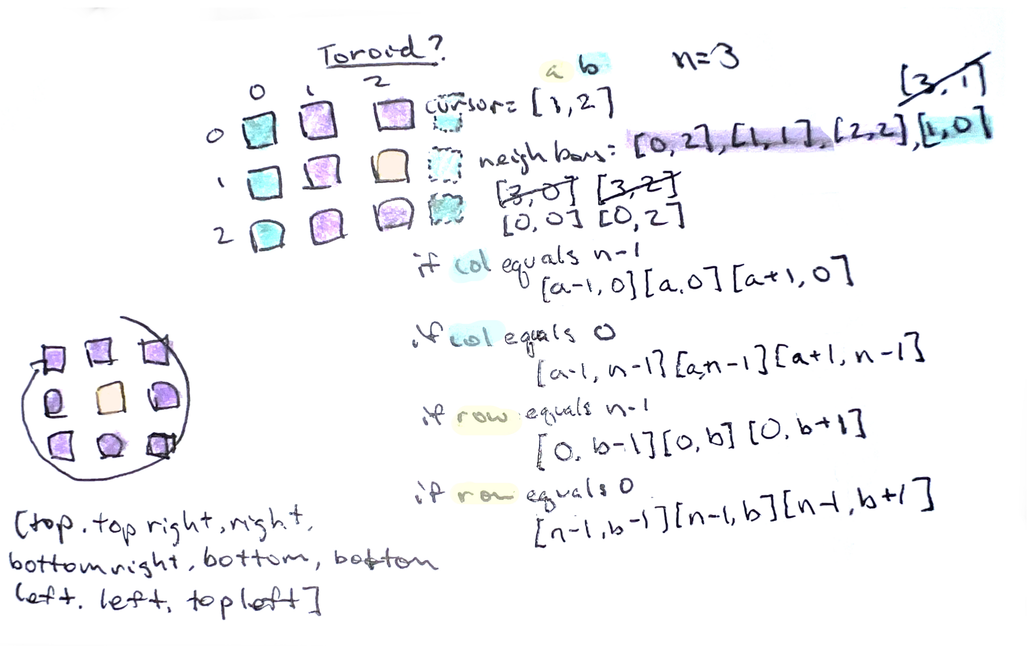
This function's role is to return the number of live cells surrounding a cell. First, check that the requested cell is a valid cell. For any cell with coordinates [a, b], its neighbors will follow the following format: top [a, b+1], top right [a+1, b-1], right [a+1, b], bottom right [a-1, b-1], bottom [a, b-1], bottom left [a-1, b+1], left [a-1, b], top left [a-1, b-1].
For a flat universe, if the cell is located at the edge of the universe, then the necessary neighbor will not be taken into account as it does not exist within the universe. The diagram below illustrates this concept:



Although a toroid universe may seem intimidating, it isn't too difficult. In the event the cell is located at the edge of the universe, then the corresponding neighbor follows this logic tree, assuming the cell has the coordinates [a, b] and the size of the universe is n by n.

- If column equals n-1 (right edge of universe), top right is [a-1, 0], right is [a, 0], top left is [a+1, 0].
- If column equals 0 (left edge of universe), top left [a-1, n-1], left is [a, n-1], bottom left is [a+1, n-1].
- If row equals n-1 (bottom edge of universe), bottom left is [0, b-1], bottom is [0, b], bottom right is [0, b+1].
- if row equals 0 (top edge of universe), top left is [n-1, b-1], top is [n-1, b], top right is [n-1, b+1].

This is illustrated in the following diagram.



uv_print: parses through universe, uses uv_get to check if cell is dead or alive and prints o for alive, . for dead.

3 Resources

Cellular Automation Wikipedia page
Stable Reflectors
Glider Gun Wikipedia page
Applications of Cellular Automations in Biology