Tu-An Nguyen
tunhnguy@ucsc.edu
01/31/2021

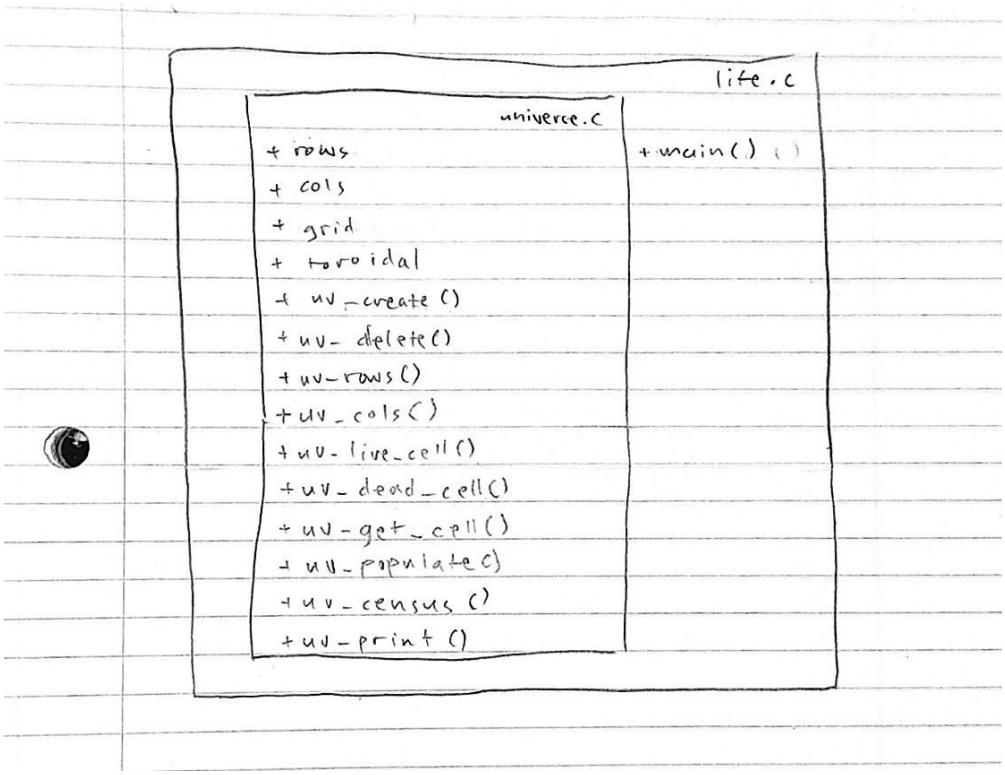
CSE 13S Spring 2020
Assignment 3: The Game of Life
Design Document

# Purpose

The purpose of this lab is to simulate the *Game of Life*, invented by John Horton Conway. The overarching idea is that we have a universe that contains a two-dimensional grid of cells, and each cell follows three rules for each generation of the universe: any live cell with two or three neighbors survives, any dead cell with exactly three live neighbors becomes a live cell, and all other cells die due to either loneliness or overcrowding. The universe can be toroidal, meaning the finite grid will have the top and bottom edges and the right and left edges connect without any half-twists.

# Structure/Layout

Here is a simplified sketch of the layout of the program:



# Pseudocode

```
life.c

main(void):
    boolean flags for -t and -s command-line options → tor, silence
    int for -n command-line option → gens
    FILE for -i and -o command-line option → input, output

    while there are options:
        set options

    scan input for number of rows and columns → r, c

    initialize Universe A
    initialize Universe B

    populate Universe A with the remainder of input

    if not sil:
        initialize screen
        hide cursor

    for i in [0, gens):
        if not sil:
            clear screen
            display Universe A
            refresh screen
            sleep for 50000 microseconds
        for each cell in Universe A:
            if census of cell == 3:
                corresponding cell in Universe B = true
            if census of cell != 2 or cell != 3:
                corresponding cell in Universe B = false
        swap Universe A and B

    if not sil:
        close screen

    output Universe A into output file using uv_print()
```

universe.c

```
uv_create(int rows, int cols, bool toroidal):
    allocate memory for Universe u
    set u.rows, u.cols, and u.toroidal
    allocate memory for u.grid

uv_delete(Universe u):
    frees allocated memory in u

uv_rows(Universe u):
    return u.rows

uv_cols(Universe u):
    return u.cols

uv_live_cell(Universe u, int r, int c):
    u.grid[r][c] = true

uv_dead_cell(Universe u, int r, int c):
    u.grid[r][c] = false

uv_get_cell(Universe u, int r, int c):
    return u.grid[r][c]

uv_populate(Universe u, FILE infile):
    scan second line of infile and onwards → int r, c:
        if nrows > r >= 0 and ncols > c >= 0:
            u.grid[r][c] = true
        else:
            return false
    return true

wrap(int pos, int n):
    return (pos + n - 1) % n

uv_census(Universe u, int r, int c):
    int count for number of alive adjacent cells

    iterate int rows through r - 1 to r + 1:
        iterate int cols through c - 1 to c + 1:
            skip step when looking at grid[r][c]
```

```
            if u is toroidal:
                apply wrap() to rows and cols
            if grid[rows][cols] is true and rows and cols in range of grid:
                count += 1

    return count

uv_print(Universe u, FILE outfile):
    get the number of rows and columns from u → nrows, ncols
    iterate through rows r < nrows:
        iterate through cols c < ncols:
            if cell at r, c is alive:
                write 'o' to outfile
            else write '.' to outfile
```

# Design Process

1. I read the Assignment 3 specifications to get a good idea of the subject matter and my task. I also referred to the supplemental readings outlined at the end of the specifications.
2. Afterwards, I began writing pseudocode for the `Universe` ADT first, because it seemed simpler than dealing with `<ncurses.h>` to me. After studying the `<ncurses.h>` man pages and playing around with the "Short `ncurses` example" in section 4 of the assignment specifications, I wrote pseudocode for the `main()` function.
3. Now, it was time to start implementing my pseudocode. First, I tried to get everything to compile. I made a `README`, a `Makefile`, `life.c`, and `universe.c` files. I left all of the functions inside `universe.c` blank, only returning the required types. As for `life.c`, I left `main()` empty, only returning 0. I made sure to include "`universe.h`" in both files.
    a. However, I did follow Sahiti's lab section to help me build the constructor of Universe. The `calloc()` function tripped me up a bit, but I'm more comfortable with it now after reading its man pages.
    b. After a missing semicolon here and a missing `<stdlib.h>` header there, I finally got the program to compile without errors.
4. Next, I implemented the `main()` function in `life.c` to ensure a way to test my functions.
    a. I first implemented and tested the command-line options before implementing the rest of the function
5. Then, I implemented the `uv_delete()`, `uv_rows()`, and `uv_cols()` functions.
6. Afterwards, I implemented `uv_populate()`.
7. At this point in time, I wasn't testing my code too well. Everytime I ran the program, I got the same output, as I haven't implemented uv_census yet. Therefore, I decided to skip the most

complicated function (in my opinion), `uv_census()` and go straight to `uv_print()`. I think this function will help the most with debugging.

8. After implementing `uv_print()`, I began to test the function. Once the function runs with no errors, I can be assured that my `uv_create()` and `uv_populate()` functions are working.
    a. I was hitting a segmentation fault whenever I tried to print the output into a file. It took me a while, but I found out that I was missing a ':' at the end of my OPTIONS.
    b. The output only had dead cells. I realized I didn't need to scan the infile again to get the number of rows and columns for the universe when the universe has already been passed into the function. This was a small oversight in my pseudocode.
9. I also realized that I wasn't using the accessor and modifier functions that I made and instead accessed `Universe u`'s members and modifying them directly in `uv_populate()` and `uv_print()`. I made sure to change this before moving forward.
10. The last function to implement is `uv_census()`. My initial approach to it was too messy. I basically just hardcoded in a bunch of if statements, and it looked terrible. I used Eugene's 1/29/2021 section to help me clean up my implementation.
    a. The outputs were printing wrong, so I spent a lot of time debugging it. I finally traced it down to `Universe b` having residual live cells after swapping it with `Universe a`. I cleared `Universe b` after the swap, which fixed the problem.
11. Also, I made sure to check the cursor display for each generation, and there were minimal issues.
12. Lastly, I added error handling for my command-line options.