Assignment 4 - DESIGN.pdf

Description of the program

The game of life. In the design document, I will be focusing on the implementation of the 0 player John Conway's game - Game of Life.

<u>What is the Game of Life:</u>
Game of Life is played on a grid(the universe) of cells, in which the cells are represented by two states - alive or dead. The game is played within steps(also called generations), during which all cells are checked, and their state is adjusted according to the rules below. Nothing else is required, only from the player, an initial input of what cells are alive before the first step is taken.

This game should be a simple mathematical representation of how life functions.

<u>Game of Life rules:</u>
1. Any live cell with 2 or 3 live neighbors <u>survives</u>
2. Any dead cell with exactly 3 live neighbors <u>becomes a live cell</u>
3. All other cells <u>die</u>, either due to loneliness or overcrowding

Files to be included

<u>Source and header files:</u>
1. **universe.c** implements the Universe ADT
2. **universe.h** specifies the interface to the Universe ADT. This file is provided and may not be modified
3. **life.c** contains main() and may contain any other functions necessary to complete your implementation of the Game of Life

<u>Additional Files:</u>
1. **Makefile:** formats all source code, including the header files.
2. **README.md:** Description of how to use my program and Makefile. It also includes any command-line option that my program accepts. Any false positives

reported by scan-build should be documented and explained here as well. Note down any known bugs or errors in this file as well for the graders.

3. **DESIGN.pdf (*This file*):** The design document describes the preliminary design and design process for my program with sufficient detail for potential replication
4. **WRITEUP.pdf:** Analysis and description of the produced program, as well as graphs generated and/or information, gathered from outputs

Pseudocode/Structure:

<u>Universe.h</u>

This file is just s declaration of a new universe. This is created as a struct called *Universe.* Below is the code in C -

```
struct Universe {
        uni32_t rows;
        uni32_t cols;
        bool **grid;
        bool toroidal;
}
```

<u>Universe.c</u>

This file will work as a generation of a new universe, and the options that determine its characteristics. We will be making a normal and toroidal universe.

```
uv_create (rows: int, cols: int, isToroidal: bool):
        Matrix = an array size of rows
        For (r = 0, r < rows, r += 1) :
                Matrix[r] = input an array size of column (this will add it into each row)
uv_delete (u: Universe):
        Matrix = []
    +   I have to free up memory that was used by the matrix and cells in it
uv_rows(u: Universe):
        Return Matrix.count (this will return us the number of rows in the universe)
```

uv_cols(u: Universe):

Return Matrix[n].count (this will return us the number of columns in the universe)

uv_live_cell(u: Universe, r: int, c: int):

This function marks the cell of row -‘r’ , and column - ‘c’ as living

Matrix[r][c] = true

uv_dead_cell(u: Universe, r: int, c: int):

This function marks the cell of row -‘r’ , and column - ‘c’ as dead

Matrix[r][c] = false

uv_get_cell(u: Universe,r: int, c: int):

Returns the state of a cell of row -‘r’ , and column - ‘c’

Return Matrix[r][c]

uv_populate(u: Universe, infile, FILE):

This is the caller function and will populate the Universe with row-column pairs read from the ‘infile’.

First-line should consist of the number of rows and columns

Subsequence lines should consist of the cells that are alive

Return the function if the universe is successfully populated

uv_census(u: Universe, r: rows, c: columns):

This function returns the number of live neighbors adjacent to the cell as row ‘r’ and column ‘c’. This takes into consideration the state of the universe - whether it is flat, or toroidal

uc_print(u: Universe, outfile: FILE):

Prints the universe in an ‘outfile’. Live cells are denoted by th character ‘o’ while dead cells are denoted with the character ‘.’.

<u>Life.c</u>

This file will provide all the instructions on how the game will be conducted. It will implement

- GetOpt option commands
- Putting the input to ncurses - so we will see how it is payed
- Steps
    - Check cells one by one, to evaluate whether their state should change

- Generate the output
    - Show the cells in a visual interface

Command-line options:

-t : toggles that the Game of Life is played on a *toroidal* universe

-s: *Silences ncurses.* Enabling this means that nothing should be displayed by ncurses

-n [generations]: Specifies the number of *generations*/*steps* that the universe will go through until the program will terminate. The default is 100 *generations*.

-i [file]: Specifies the input file to read, in order to get the coordinates of where should the live cells be located, on the grid of the universe, before the game starts.

-o [file]: Specifies the output file to put the resulting universe in. If the file is not specified the universe will be printed out in the command line

Credit

- I took the pseudocode from the assignment 4 paper written by prof. Darrel Long
- I have watched the recording of Eugenes Lab section on 1/28/22, uploaded on Yuja
- I had an understanding of the functioning of Game of Life before, therefore I needed not gain a deeper understanding through external research