CSE 13S Winter Quarter 2022

Assignment 4: The Game of Life

Description of the Program:

The goal of this assignment is to simulate Conway's zero-player game of life. The game of life is to be played on an infinite grid and determines the life and death of a given cell by certain conditions. We are to replicate this.

Files to be included in the "asgn4" directory

- universe.h
    - The header file for universe.c and is given in the resources file
- universe.c
    - For us to create the commands of universe and declare the abstract data type universe
- life.c
    - contains main and prints out the generations of the game of life, contains the getopt loop
- Makefile
    - Makefile to compile and format code
- README.md
    - short synopsis and how to run the code along with any known errors
- DESIGN.pdf
    - This file

Layout/Structure:

- We create a universe.c program to keep track of your universe and easily alter it
- Include the header file universe.h in our main life.c function to be able to use the functions we created in universe.c in life.c
- We then link both programs together to run the life.c function.

Pseudocode:

Universe.c creates various useful functions for us to use when we eventually manipulate the universes we created

```
- def uv_create():
    - allocate memory for universe
    - return universe
- def uv_delete():
    - delete memory for universe
- def uv_rows():
    - return num of rows for universe
    - return universe.rows
- def uv_cols():
    - return num of cols for universe
    - return universe.cols
- def uv_live_cell(row, col):
    - if row<uv_rows and col<uv_cols:
        - Universe[row][col] = true
    - else:
        - return false
- def uv_dead_cell(row,col):
    - if row<uv_rows and col<uv_cols:
        - Universe[row][col] = false
    - else:
        - return false
- def uv_get_cell(row,col):
    - if row<uv_rows and col<uv_cols:
        - return Universe[row][col]
    - else:
        - return false
- def uv_populate(Universe, file):
    - open file
    - for row,col in range(file.length()):
        - Universe[row][col] = true
- def uv_census(row,col):
    - counter = 0
    - if toroidal:
        - r_min = (r + rows - 1) % rows
        - r_max = (r + 1) % rows
        - c_min = (c + cols - 1) % cols
        - c_max = (c + 1) % cols
        - check all eight possibilities
            - if uv_get_cell(u, r, c):
                - counter += 1
    - else:
        - for r in range(row-1, row+1):
            - for c in range(col-1, col+1):
                - if r < 0 or c < 0 or r >= row or c >= col:
                    - continue
                - if Universe[r][c] = true:
                    - counter += 1
            -
    - return counter
```

life.c

life.c works by using the functions we made in universe.c to create a Universe and then simulate the game of life with the given data.

```
    - def swap(A, B):
        -   swap A and B
-   def next_gen(A, B, row, col):
        -   for r in range(row):
                -   for c in range(col):
                        -   if A[r][c] is alive and census == 2 or 3:
                                -   B[r][c] is alive
                        -   if A[r][c] is dead and census == 3:
                                -   B[r][c] is alive
                        -   else:
                                -   B[r][c] is dead
-   def main():
        -   initialize toroidal as false;
        -   initialize generation number to 100
        -   initialize in and out file as stdin and stdout
        -   use getopt to get arguments
                -   if t:
                        -   toroidal = true
                -   if s:
                        -   turn off ncurses
                -   if n:
                        -   generation number = input
                -   if i:
                        -   file in = input
                -   if o:
                        -   file out = input
        -   create Universe A and B get dimensions from file in
        -   close file in
        -   initialize screen
        -   for i in range(generation number):
                -   next_get(A, B, row, col)
                -   clear screen
                -   print A onto screen
                -   swap universe A and B
                -   delay screen 50000 microseconds
        -   output Universe B to out file
        -   return 0
```

- ## Notes:
    - allocate memory for grid using for loop and malloc
    - delete Universe by using free
    - return rows and cols by using r->rows and r->cols respectively
    - open file by using file = fopen(“file”, “r”)
    - check all eight possibilities by using
        - r_min c_max

        - r_min c
        - r_min c_min
        - r c_min
        - r c_max
        - r_max c_min
        - r_max c
        - r_max c_max
    - def swap is done by using double pointers
    - getting dimensions from file
        - use fscanf
    - close open files using fclose(file)
    - output Universe B to outfile using uv_print function

Credit:

- A lot of credit goes to Eugene for the syntax on opening files, getting their inputs, and just the blueprint for creating a struct albeit his was in only one dimension.