Caterin Duarte Reynosa
cduarter@ucsc.edu

CSE13S Fall 2020
Assignment 3: The Game of Life
Design Document

Goal

The goal for this assignment is to create a program that mimics the game of life. Based on arguments parsed in the command line, we will be creating universes that progress through generations based on the state of the cells. Most importantly, the goal of this assignment is to learn about memory and how to allocate/free memory using the malloc function or the calloc function. We will also be developing text-based user interfaces with the universes that are created.

Summary + Rules

- Get the user input (parse arguments to the command line)
    - -t: means the user wants to play in a toroidal mode
    - -s: silence the ncurses (printing an interactive screen)
    - -n: sets the generation amount, if nothing entered default should be 100
    - -i: the file we should be reading form the list folder
    - -o: print out the final state of univer a
- With the user inputs, use fscanf() to read the number of rows and columns
- Using those rows x columns, create Universe A and Universe B
    - Universe A = my current universe
    - Universe B= the next “generation”
- With the inputs of rows x columns, populate universe A
- Set up the ncurses screen
- Set the number of generations
    - If -s was not parse, clear the screen and display the universe
    - display the state of the universe after each evolution
    - Swap universes to update universe a
- Close the screen
- Output universe A

Design Ideas + Comments on the functions I have to do

- Universe *uv_create(int rows, int cols, bool toroidal)
    - Takes in the rows, cols, and weather its toroidal or not
    - Allocate memory for a 2-d array using calloc
        - Note to self: look at the example from notes/slides
- void uv_delete(Universe *u)
    - Takes in a pointer (i.e the ones created in the above function)
    - Free memory
        - Note to self: when freeing, think first in last out
- int uv_rows(Universe *u)
    - Return the number of rows in the universe

- int uv_cols(Universe *u)
    - Returns the number of columns in the universe
- void uv_live_cell(Universe *u, int r, int c)
    - Takes in the row, column, and a pointer
    - First check to make sure it's in the bound
        - Then declare it as true since true == alive
    - If out of bound, continue
- void uv_dead_cell(Universe *u, int r, int c)
    - Takes in the row, column, and a pointer
    - First make sure it's in the bound
        - Then declare it as false since false == dead
    - If out of bound, continue
- bool uv_get_cell(Universe *u, int r, int c)
    - Checks if a cell (row x column in the parameters) is alive or dead
    - If out of bound, return false
    - Else return true
- bool uv_populate(Universe *u, FILE *infile)
    - With the pairs read from the file, it will populate the universe
    - loop that makes calls to fscanf() to read in all the row-column pairs in order to populate the universe.
    - If out of bound, return false
    - If it could populate, return true
    - If it could not populate return an error message
- int uv_census(Universe *u, int r, int c)
    - Counts the neighbors (North, south, east, west, adjacent)
    - If toroidal, have a loop that will circle and count neighbors alive
    - Else count the neighbors alive
- void uv_print(Universe *u, FILE *outfile)
    - Print out the content of the universe A
    - If a cell is alive, print a o
    - If a cell is dead, print a .
    - Always print out the flatten universe

Pseudocode

- Universe *uv_create(int rows, int cols, bool toroidal)
    - Allocate *u
    - Set the rows == r
    - Set the columns == c
    - Set the toroidal == bool
    - Since it’s a 2-d array, allocate for an array with number of rows and size of columns
    - Have a loop that will allocate each element in the array
    - Return what was allocated
- void uv_delete(Universe *u)
    - Have a loop to access the elements in the array

    - Free that element
    - Free the array
    - Free *u
- int uv_rows(Universe *u)
    - Return the pointer of row
- int uv_cols(Universe *u)
    - Return the pointer of cols
- void uv_live_cell(Universe *u, int r, int c)
    - If r is >0 and < pointer of rows and If c is >0 and < pointer of columns
        - Set array == true (similar to the rolls array for vampire.c)
- void uv_dead_cell(Universe *u, int r, int c)
    - If r is >0 and < pointer of rows and If c is >0 and < pointer of columns
        - Set array == false (similar to the rolls array for vampire.c)
- bool uv_get_cell(Universe *u, int r, int c)
    - If r and c are in bound
        - Return value of [r][c]
    - If out of bounce
        - Return false
- bool uv_populate(Universe *u, FILE *infile)
    - Loop until EOP
    - calls fscanf() to read in all the row-column pairs in order to populate the universe.
    - If out of bound,
        - return false
    - Else if it populated
        - return true
    - Else
        - Return an error message “Could not populate”
- int uv_census(Universe *u, int r, int c)
    - Counter for alive neighbors = 0
    - If teroidal == true
        - Calculate the ppl on the right/ left similar to the Midnight scenario
        - If neighbor == alive
            - Increment counter for alive neighbors
    - else :
        - If neighbor == alive
            - Increment counter for alive neighbors
- void uv_print(Universe *u, FILE *outfile)
    - Iterate through the file
    - If *u == alive
        - Print o
    - If *u == dead
        - Print .
    - **Always print out the flatten universe **

Updates + comments

While coding for this class I realized that this assignment was much more harder than what I originally expected. Even though I started my assignment early (I started thinking about what to do on tuesday, created my skeleton on wednesday, creating a more specific detailed version of my skeleton, and on friday I did most of the functions) I still didn't finish in time. I spent the whole weekend trying to debut it but I couldn't figure anything out. The hardest part of this assignment was the census function and the ncurses. While coding I didn't change a lot about my original design document. The only functions I changed were census and print. During print, I thought I would iterate over the file I was writing to, but I actually just had to iterate through rows, then columns and then I had to print out to the file. For my census function I wanted to do a loop but I ended up doing a bunch of if statements to circle around where I was. It wasn't the most efficient thing to do, but it was the only way I could understand.