Assignment 4 DESIGN.pdf

Julian Chop

February 14, 2023

1 Brief Description of Assignment

In this assignment, we are tasked with implementing John Horton Conway's 0-player game, The Game of Life. The Game of Life is played on a 2-dimensional grid of cells that represents a universe. Each cell is either dead or alive. The game progresses through generations and after each generation, there are 3 rules that determine the state of the universe:

1. Any live cell with 2 or 3 live neighbors survives to the next generation
2. Any dead cell with exactly 3 live neighbors becomes a live cell
3. All other cells die, either due to loneliness(not enough neighbors) or overcrowding (too many neighbors)

A neighbor is defined by any cell directly up, down, or diagonal to that cell. Our task will be to make Universes, populate these universes, and manipulate these universes through C.

2 Universe.c

This file will contain our Universe ADT or *abstract data type*. This ADT will provide the abstraction for a 2-dimensional grid of cells. This file will also contain the constructor, destructor, accessor, and manipulator functions for our ADT. The following sections will describe these functions and how the code may look like.

3 Universe *uv_create(uint32_t rows, uint32_t cols, bool toroidal)

This constructor function will create a Universe. The first 2 parameters will determine the dimensions of the universe. the third parameter determines if the universe is toroidal or not. Here is what the code may look like:

```
allocate memory for number of rows using calloc
for i in rows:
    allocate memory for matrix[i] which will be the columns
```

4 void uv_delete(Universe *u)

This destructor function will free any memory allocated for a Universe by the constructor function.

```
for i in rows:
    free memory of matrix[i]
free memory of grid
```

Basically, you are working backward from the constructor function, starting first freeing the columns and then the rows.

5 uint32_t uv_rows(Universe *u)

This accessor function will return the number of rows in the specified universe.

```
return u -> rows
```

the arrow basically grabs the data element 'rows' from universe u.

6 uint32_t uv_cols(Universe *u)

Does the same thing as uv_rows() but returns columns instead.

```
return u -> cols
```

7 void uv_live_cell(Universe *u, uint32_t r, uint32_t c)

This manipulator function will mark the cell at row r and column c as a live cell.

```
grid[r][c] = True
```

8 void uv_dead_cell(Universe *u, uint32_t r, uint32_t c)

This manipulator function will mark the cell at row r and column c as a dead cell.

```
grid[r][c] = False
```

9 bool uv_get_cell(Universe *u, uint32_t r, uint32_t c)

This function returns the value of the cell at row r and column c. If row and column is out of bounds, then false is returned.

```
return grid[r][c]
```

10 bool uv_populate(Universe *u, FILE *infile)

This function will populate the given universe will row and column pairs read in from the given infile. The first line of input from the file will determine the number of rows and columns. Every line after that will represent a coordinate of a living cell.

```
\\not 100% on how I will be implementing this yet
take first line to determine grid
for (i in infile - first line):
    grid[val 1][val 2] = True
```

11 uint32_t uv_census(Universe *u, uint32_t r, uint32_t c)

This function returns the number of lives neighbors adjacent to the cell at row r and column c. If the universe is toroidal, then all neighbors are valid. You just gotta wrap to the other side.

```
n_counter = 0
for i in range (-1,1):
    for j in range (-1,1):
        if( i != 0 || j != 0):
            if grid[i][j]:
                n_counter += 1
return n_counter
```

This code basically iterates a 3 by 3 box with the given cell in the middle. Then if that cell is living, then n_counter gets + 1.

12 void uv_print(Universe *u, FILE *outfile)

This function prints out the universe to the given outfile. A live cell is represented as the character 'o' and a dead cell is represented by the character '.'(a period).

```
for i in rows:
    for j in columns:
        if grid[i][j]:
            print 'o' to outfile
        else:
            print '.' to outfile
```

13 life.c

This c file brings everything together which will use the functions from universe.c to make The Game of Life. It will also include the command-line options and ncurses display.

```
create universe A and universe B with uv_create
populate universe A using uv_populate with given nfile

for i in generations:
    for rows in universe A:
        for rows in universe B:
            find num neighbors of current cell with uv_census
            if current cell alive:
                if num neighbors == 2 or 3:
                    cell is alive in universe B
                else:
                    cell is dead in universe B
            else:
                if num neighbors == 3:
                    cell is alive in universe B
                else:
                    cell is dead in universe B
    swap universes A and B

print universe A with uv_print to given outfile
delete universe A and B to free memory
```