I want to Big Bang myself

dlee181

February 2023

1 Introduction

In this assignment we will be taking John Horton Conway's game of life and trying making that game on our terminals. The way the game works is that it is played out on a theoretically infinite 2-d grid that represents a universe. Each cell will either be dead or alive. And the game will progress through multiple generations. The three rules are:

1. Any live cell with two or three live neighbors survives.
2. Any dead cell with exactly three live neighbors becomes a live cell.
3. All other cells die, either due to loneliness or overcrowding.

The catch with this assignment is that we must also make it either toroidal or plain flat

2 Deliverables

1. universe.c: Will implement the Universe ADT
2. universe.h: Specifies the interface of the universes ADT.
3. life.c contains main and will also implement ncurses inorder to bring game of life.. to life.
4. Makefile
5. README.md
6. DESIGN.pdf

3 Making My Universe.c

Universe *uv_create(uint32_t rows, uint32_t cols, bool toroidal)
This will be the constructor for the assignment that creates the Universe pointer (Universe *u). Takes in two parameters r/c which will be the dimensions of my 2d bool matrix. If bool toroidal is true, then the universe is toroidal. Use calloc and make a dynamically allocate Universe pointer. Then using the example create a 2d Bool matrix.

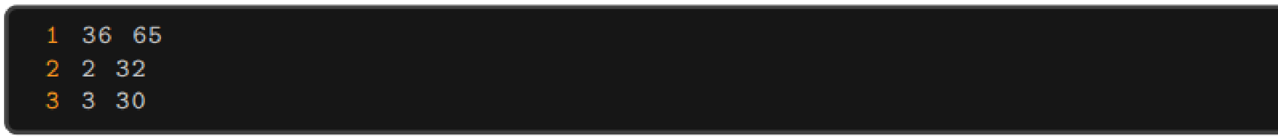
void uv_delete(Universe *u)
Destructor. Frees any memory allocated by the constructor. for every row in the grid free the row of the grid. Then after all that, free the grid.

uint32_t uv_rows(Universe *u);
Literally return $u->$ rows

uint32_t uv_cols(Universe *u);
Literally return $u->$ cols
void uv_live_cell(Universe *u, uint32_t r, uint32_t c)
if (out of bounds) return false; else u[r][c] = true;
void uv_dead_cell(Universe *u, uint32_t r, uint32_t c);
u[r][c] = false;

bool uv_get_cell(Universe *u, uint32_t r, uint32_t c);
if (out of bounds) return false; else return u[r][c];

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



```
36 65
2 32
3 30
```

The first line of a valid input file for this program consists of the number of rows and the number of columns. Above 36 will be the number of rows to be read, and 65 is the number of columns to be read. The values 2 and 32 are the rows and columns of the live cell. So are 3 and 30. Meaning each subsequent row after the 1st line are the rows and columns of live cells.

In order to go line by line the input file, we need to create a for loop that makes calls to fscanf() which allows us to read in all r-c pairs in order to use uv_populate(). Of the lines read, if a pair lies beyond the dimensions of the universe, return false, implying the universe failed to populate. Else return true if the universe successfully populated.
If you fail to populate the universe use fscanf() stderr to give an error message

and the game should fail.
Our "universe" will be in the shape of a torus, which is essentially a donut.
uint32_t uv_census(Universe *u, uint32_t r, uint32_t c)
example of the top right of a non-toroidal) first I check if it's not toroidal then if it isn't I will check the neighbors of the surrounding 8 squares to see if they are dead or alive. I do this by finding the top-left, top-middle, top-right, middle-left, middle-right, bottom-left, bottom-middle, and bottom-right. The math here is just adding or subtracting by a row or column per square.
For a toroidal shape, the same goes, but instead of only adding and subtracting, we will be implementing modular arithmetic. We will do –using the top left as an example– (row + the length of the universe's row) mod universe row, and the same for the column but swap row with the column. This works because when we go over the edge in a toroidal, the value will go back to the 0th index and thus repeat the toroidal again.

void uv_print(Universe *u, FILE *outfile); Prints out an outfile of the universe. The file will contain either 'o's or '.'s. In order to print to the specified outfile, use fputc() or fprintf(). And to print a torus, we will simply print out a 2d grid instead.

4 Life.c

We will be dealing with ncurses which is basically going to allow us to display our game of life. Basically everytime that I print something, I will first make the universe, and because I am using an ADT, I will be using function calls inorder to access things like rows and columns. Next I will loop in range of the rows, then the range of the columns. Each iteration will be checking the cell at the iteration and printing out 'o' or '.' based on the return of the uv_census function.

5 Credit

1. Credit Lev going over the matrix and how to do uv_create
2. Credit Ben explaining the Universe functions
3. Credit Darrel long for psuedo code for the assignment.
4. Credit Dev for the example on how to do uv_populate.