Design.pdf

Githika Annapureddy

February 2023

1 universe.c

All of the following functions are in universe.c

2 uv_create

uv_create accepts rows, cols, and the boolean for torodial. it returns a pointer to a Universe object.
It does this by first allocating memory for the Universe using calloc. size is sizeof(Universe). number is 1.
The pointer to this Universe is myUni.
We are now making a matrix of booleans.
Then it allocates memory for the rows using calloc. size is sizeof(pointer to a boolean). number is rows.
The pointer to the rows is called matrix. It is a pointer to a pointer of booleans.
Then, for each row, it allocates memory for the columns using calloc. size is sizeof(boolean). number is cols.
Use a loop to go through each row.
The pointer to each row of columns is matrix[r], where r is the row number.

myUni -¿ grid = matrix makes the grid of the Universe equivalent to the memory we allocated for the matrix.
Then, we initialize each square to false, by creating a nested for loop that loops through each element in the matrix.
myUni -¿ rows = rows
myUni -¿ cols = cols
myUni -¿ toroidal
These 3 lines assign the variables of the universe to the specified input.
Finally, we return myUni, the pointer to the Universe object.

3 uv_delete

uv_delete accepts a pointer to a Universe object. It returns nothing. It deletes all the memory allocated to create the universe.
if the universe == null, we return because it does not have any allocated memory.
Get the number of rows and columns using the uv_rows() and uv_cols() functions.
check if universe-¿ grid is null, if it is return.
make a for loop to loop through each row and free universe-¿grid[r], where r is each row.
then free the grid: free(universe-¿grid)
finally free the universe: free(universe)

4 uv_rows

uv_rows accepts a pointer to a Universe object. It returns an integer, the number of rows in the Universe using u-¿rows.

5 uv

uvaccepts a pointer to a Universe object. It returns an integer, the number of columns in the Universe using u-¿ cols.

6 uv_live_cell

uv_live_cell accepts a pointer to a Universe object, rows and columns. It returns nothing.
It first tests that the specified cell is within bounds by checking if the row and column are less than the uv_rows() and uv_cols() respectively. If so, it makes the specified cell at u-¿grid [r][c] true.

7 uv_dead_cell

uv_dead_cell accepts a pointer to a Universe object, rows and columns. It returns nothing.
It first tests that the specified cell is within bounds by checking if the row and column are less than the uv_rows() and uv_cols() respectively. If so, it makes the specified cell at u-¿grid [r][c] false.

8 uv_get_cell

uv_get_cell accepts a pointer to a Universe object, rows and columns.
It first tests that the specified cell is within bounds by checking if the row and column are less than the uv_rows() and uv_cols() respectively. If so, it returns a boolean, true if the cell is alive and false is the cell is dead.
Otherise, it returns false.

9 uv_populate

uv_populate accepts a pointer to a Universe object and a pointer to a file. It returns nothing. It reads coordinate points from that file. The first set of coordinate points, specified in the format 'x y' are the size of the Universe. Each following pair of coordinate points are on a new line. These points need to be marked as alive (true). If one of the pairs is not within the bounds of the Universe, the execution of the whole game is terminated and "Malformed input." is printed out.
first the file is opened using fopen()
then, a while loop is used to go through each line of the input (fscanf is used to read the files.
for every row except the first, make that coordinate point true on the grid
close the file
return true if successful

10 uv_census

uv_census accepts a pointer to a Universe object, rows and columns. It returns an integer, which is the number of alive cells that are the specified cell's ([r][c])'s neighbors. Neighbors can be diretly next to or diagonal from a cell. Each cell on a torodial Universe has 8 neighbors. In a non torodial universe, if a cell is on the border of the Universe, it will have less than 8 neighbors.
uv_census calls the function HowManyNeighbor which takes the arguments universe pointer, rows, cols, and a boolean for toroidal.
uv_census returns aliveNeighbors, which is returned from HowManyNeighbor
In HowManyNeighbor:
a variable aliveNeighbors is set to 0
A nested loop is used. The outer loop loops from r-1 to r+1 where r is rows. The inner loop loops from c-1 to c+1 where c is cols.
if r and c do not equal rows and cols, we call the function isNeighbor.
if isNeighbor returns true, then aliveNeighbors += 1.
HowManyNeighbor returns the of aliveNeighbors
isNeighbor takes a universe pointer, r for a specified row, c for a specified column, and a boolean for toroidal.
The function returns true if the cell is alive.
uv_rows() and uv_cols() are used to get the of rows and columns

if not toroideal,
the point (r,c) is checked to be in bounds (¿= to 0 and ¡ of rows and columns, respectively)
if it is in bounds, the boolean value of the cell is returned.
if it is out of bounds false is returned.
if toroidal,
if (r,c) == of rows or cols respectively, they become equal to 0
if r,c) == -1, they become equal to the of rows or cols respectively
the boolean value of the cell (r,c) is returned

11 uv_print

uv_print accepts a pointer to a Universe object and a pointer to a file. It returns nothing. It writes to the file. It prints out the universe with the specified dimensions. Each dead cell is represented with a '.'. Each alive cell is represented with a 'o'.
First, the file is opened with fopen with the mode "w" for write
uv_rows() and uv_cols() are used to get the of rows and columns
A nested for loop is used to go through each cell (outer loop is for the rows and inner loop is for the columns)
the state of the cell is checked using uv_get_cell()
if the cell is alive (true), fputs enters "o" to the file
if the cell is dead (false), fputs enters "." to the file
Finally, the file is closed

12 life.c

life.c contains one function, the main function.
first, we declare variables.
inputname is a string containing "stdin"
outputname is a string containing "stdout"
generations = 100
ncurses = true
toroidal = false
It reads user input using
while ((opt = getopt(argc, argv, OPTIONS)) != -1)
A switch is used to read each input letter
for case't', toroidal = true;
for case's': ncurses = false;
for case'n': generations = the number provided by user;
for case 'i': inputname = the name provided by user;
for case 'o': outputname = the name provided by user;
input file is opened for reading
the first line is read using fscanf. the input will be in the format x y

these values become rows and columns
Universe A is created using uv_create(rows, cols, toroidal)
Universe B is created using uv_create(rows, cols, toroidal)
if uv_populate returns false, print the error message "Malformed input" and return to terminate the execution
for g in generations

if ncurses is true:
initscr;
curs_set(FALSE);
for r in rows
for c in cols
clear screen
check if each cell is true using uv_get_cell(UniA, r, c)
if true: mvprintw(r, c, "o");
refresh screen;
sleep for 50000 ms
exit loops and endwin();

now, we change the input, and go to the next generation
for r in rows
for c in cols
n = uv_census(UniA, r, c)
if cell is true AND (n == 2 OR n == 3)
do nothing
else if cell is false AND n == 3
make cell alive
else
make cell dead
outside of all of the loops,
Universe *temp = UniA
UniA = UniB
UniB = temp

outside generation loop:
open output file for writing
uv_print(UniA, outfile);
uv_delete(UniA);
uv_delete(UniB)
return 0;