Assignment 4 DESIGN.pdf

Ning Jiang

Description of Program: For this assignment, we mainly need to creat a ADT(abstract data type) called Universe using struct in universe.c and life.c contains main() and may contain any other functions necessary to complete the implementation of the Game of Life.

Files to be included in directory "asgn4":
1. universe.c implements the Universe ADT.
2. universe.h specifies the interface to the Universe ADT. This file is provided and may not be
modified.
3. life.c contains main() and may contain any other functions necessary to complete the implementation of the Game of Life.
4. Makefile
5. README.md
6. DESIGN.pdf

Pseudocode / Structure:

```
struct Universe {
     uint32_t rows;
     uint32_t cols;
     bool **grid;
     bool toroidal;
 }; //The universe will be abstracted as a struct called Universe.

Universe *uv_create(uint32_t rows, uint32_t cols, bool toroidal)
allocating a matrix of uint32_ts // do it like Page3 in asgn4

void uv_delete(Universe *u):
for (r  in range(0, u -> rows):
          free(u->grid[r])
          // remember to use -> to point to the grid
          r += 1
free(u->grid);
free(u);
// free everything

uint32_t uv_rows(Universe *u):
       return(u->rows)
```

```
uint32_t uv_cols(Universe *u):
      return(u->cols)

void uv_live_cell(Universe *u, uint32_t r, uint32_t c):
      u->grid[r][c]= true
//marks the cell at row r and column c as live

void uv_dead_cell(Universe *u, uint32_t r, uint32_t c):
      u->grid[r][c] = false
//marks the cell at row r and column c as dead

bool uv_get_cell(Universe *u, uint32_t r, uint32_t c)
// returns the value of the cell at row r and column c

bool uv_populate(Universe *u, FILE *infile):
    while (fscanf(infile,"%d%d", &r, &c) != EOF)
       //use fscanf to scan the file to get the coordinate of live
       // cells.
       if ((r < 0 || c < 0) || (r >= u->rows || c >= u->cols))
       // if the live cell coordinate out of grid
             return false
        Else:
             Set the cell to live
   Return true

uint32_t uv_census(Universe *u, uint32_t r, uint32_t c):
    int64_t r1 = r; //reset the r and c to signed
    int64_t c1 = c;
    int count = 0;
    int64_t i, j;
    int64_t h, g;
    for(i in range(r1 - 1, r1):
       For (j in range(c1 - 1, c1):
             if(i == r1 and j == c1){
                  continue;
             // the point is (r,c) itself instead of its neighbors
             else if (grid is (toroidal) and (i or j is out of the
             grid)):
             Do modular arithmetic
              h = (i + (rows)) % (rows);
              g = (j + (cols)) % (cols);
             Check if grid[h][g] == 1:
             If true:
                Count = count + 1
```

```
            else if (grid is (not toroidal) and (i or j is out of
              the grid)):
                Continue to find the next point.
            Else:
                Count = count + 1
Return count


void uv_print(struct Universe *u, FILE *outfile):
       for(i in range(0, rows):
          if grid[i][j] = True:
             fputc('o', outfile);
          Else:
             fputc('.', outfile);
        fputc('\n', outfile)
   // give the output to the file

EXTRA FILE TO TEST uv_sensus:
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "universe.h"
#include <ncurses.h>
#include <unistd.h>

int main(){
        uint32_t rows, cols;
        bool toroidal = false;
        FILE *input;
        input = fopen("test.txt", "r");
        fscanf(input, "%d%d", &rows, &cols);
        Universe *A = uv_create(rows, cols, toroidal);
        uv_populate(A, input);
        fclose(input);
        for(uint32_t r = 0; r < rows; r++){
            for(uint32_t c = 0; c < cols; c++){
                int count = uv_census(A, r, c);
                printf("%d\n", count);
            }
        }
        return 0;
}

// I created it to test whether uv_census works
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "universe.h"
#include <ncurses.h>
#include <unistd.h>

#define DELAY 50000
#define OPTION "tsn:i:o:"

int main(int argc, char **argv){
    uint32_t rows, cols;
    int generations = 0;
    FILE *input = stdin;
    FILE *output = stdout;
    bool ncurses = true;
    bool toroidal = false;
    int opt = 0;
    while ((opt = getopt(argc, argv, OPTION)) != -1) {
        switch (opt) {
            case 't': toroidal = true; break;
            case 's': ncurses = false; break;
            case 'n': generations = (uint32_t) strtoul(optarg, NULL,
                                    10); break;
            case 'i': input = fopen(optarg, "r"); break; // fopen
            case 'o': output = fopen(optarg, "w"); break;
        }
    }
    fscanf(input, "%d%d", &rows, &cols)
    // read the rows and cols of the grid
    Universe *A = uv_create(rows, cols, toroidal);
    Universe *B = uv_create(rows, cols, toroidal);
    uv_populate(A, input);
    fclose(input);
    initscr();
    curs_set(FALSE);
    while(generations){

        // ncurses.
        if(ncurses){
            clear();
            for(uint32_t i = 0; i < rows; i++){
                    for(uint32_t j = 0; j < cols; j++){
                            if(uv_get_cell(A, i, j)){
```

```
                                printw("o");
                        }else{
                                printw(".");
                        }
                }
                printw("\n");
        }
        refresh();
        usleep(DELAY);
  }
 //end ucurses
// use ncurses to print the each generation

//Perform one generation.
       for(r in range(0, rows):
            for(c in range(0, cols):
                int count = uv_census(A, r, c);
                bool state = uv_get_cell(A, r, c);
                if(state){
//Any live cell with two or three live neighbors survives.
                    if((count == 2) || (count == 3)){
                        uv_live_cell(B, r, c);
                    } else{
                        uv_dead_cell(B, r, c);
                    }
                }else{
  // Any dead cell with exactly three live neighbors becomes
    a  live cell.
                    if(count == 3){
                        uv_live_cell(B, r, c);
                    } else{
                        uv_dead_cell(B, r, c);
                    }
                }
            }
        }
// swap pointer a->b, b->a

        // do the next generation
        generations = generations - 1;
    }
    endwin();
    uv_print(A, output);
    fclose(output);
```

```
        uv_delete(A);
        uv_delete(B);
        return 0;
    }
```

Notes:
1. Remember to use malloc to allocate a memory to u.
Universe *u = (Universe *) malloc(sizeof(Universe));
2. Use -> pointer to point to each elements in Universe. Like
u->rows = rows;
u->cols = cols;
u->toroidal = toroidal;
3. use modular arithmetic for toroidal grid.
 h = (i + (rows)) % (rows);
 g = (j + (cols)) % (cols);
3. Remember to free everything.
4. In ncurses, instead of using printf, we use printw

Error Handing:
1. I use a lot of time to find the error in uv_census. After I write my own testig file, I find out that since r is unsigned integer, when the testing point is (0,0), we want to get its neighbour by using r - 1 which is invalid. As a result, I use another signed integer variable r1 = r and do the same to c: int64_t c1 = c.
2. I my fscanf don’t work and cause: Segmentation fault (core dumped). I set up valgrind to find what’s going on. I found that the reason why this error appears is because I was scan the char instead of FILE.

Credit:
1. I learned how to make an ADT by using struct.
2. I learned how to use ncurses.
3. I learn how to do a terminal grid.
4. I learned how to debug.
5. I learned how to use fscanf.