DESIGN

Wenyu Liang

February 13, 2023

1 Program Description

This program will implement the Game of Life which is a zero-player game, which means the game doesn't involve any players and it will proceed once the initial state is determined. The game will be on a finite 2-D grid of cells. Users can provide option to decide if the grid is flat or a torus. The program require users to provide the live cells to get the game started. Input can be a file or standard input and the program will display the generations of the game.

2 Files to be included in directory asgn4

1. **universe.c**: implements the Universe ADT.
2. **universe.h**: specifies the interface to the UniverseADT.
3. **life.c**: contains main() and other functions to implement the Game of Life.
4. **Makefile**: This file is provided and directs the compilation process of this program.
5. **README.md**: Describe how to use the script and Makefile.
6. **DESIGN.pdf**: Describes design for the program with pseudocode and visualization.
7. **WRITEUP.pdf**: Describes how the program works in detail, codes included.

3 Pseudocode

1. **universe.c**

```
// First we need to explicitly define structure Universe
struct Universe {
    uint32_t rows;
```

```
    uint32_t cols;
    bool **grid;
    bool toroidal;
};

*uv_create(rows, cols, toroidal){
    //dynamically allocate momery to a pointer of pointer using calloc
    **grid = calloc(rows, sizeof(uint32_t *));
    //use for loop to allocate memory to every array in the matrix
    for (r in range(0, col) {
        grid[r] = calloc(cols, sizeof(uint32_t));
    }
    assign attributes to the structure
}

*uv_delete(*u){
    free the memory from inside to outsize
    which is just the reverse process of uv_create
}

uv_rows(*u){
    return u->rows;
}

uv_cols(*u){
    return u->cols;
}

uv_live_cell(*u, r, c){
    if (!toroidal && IN_BOUND(r, c, u->rows, u -> cols))
    {
        u->grid[r][c] = true;
    }
    else if (u->toroidal){
        u->grid[toroidalIndex(x,y)] = true;
    }
}

uv_dead_cell(*u, r, c){
    u->grid[r][c] = false;
}
uv_dead_cell(*u, r, c){
    if (!utoroidal && IN_BOUND(r, c, u->rows, u -> cols))
    {
        u->grid[r][c] = false;
    }
```

```
    else if (u->toroidal)
    {
        u->grid[toroidalIndex(x,y)] = false;
    }
}

uv_get_cell(*u, r, c){
    if (toroidal=true){
        if(u->grid[toroidalIndex(x,y)){
            return true;
        }
        else {return false;}
    }
    else{
        if (index out of bound){
            return false;
        }
        else {return true;}
    }
}

uv_populate(*u, *infile){
    FILE *infile;
    uint16_t x, y;
    uint32_t lines = 0;
    int error;
    char buffer[20];
    while (!feof(infile)) {
        if (fgets(buffer, sizeof(buffer), infile) != NULL){
            if (lines == 0) {
                if (sscanf(buffer, "%" SCNu16 " %" SCNu16 " ",
                &(u->rows), &(u->cols)) == 2){continue}

            if (lines > 0) {
                if (sscanf(buffer, "%" SCNu16 " %" SCNu16 " ",
                &x, &y) == 2){u->grid[x][y] = 1;
        lines + =1;
    }
    fclose(infile);
}

uv_census(*u, r, c){
    num_t = 0;
    num_f = 0
    if (u->toroidal){
        for (neighbor in neignbors){
```

```
            if alive:
                num_t++
        }
        return num_t;
    else {
        for (neighbor in neignbors inside the grid){
            if alive:
               num_f++
        }
    return num_f;
}

uv_print(*u, *outfile){
    if (outfile == stdout){
        for (row in rows)
        {
        for (col in cols)
        {
        printf("%s", u->grid[i][j] == true ? "o" : ".");
        }
    }
    }
    else {
        for (row in rows)
        {
        for (col in cols)
        {
        fprintf(outfile, "%s", u->grid[i][j] == true ? "o" : ".");
        }
    }
    }

}
```

2. **universe.c**

```
swap(Universe **a, Universe **b){
    Universe *temp = *a;
    *a = *b;
    *b = temp;
}
update(Universe *a, Universe *b){
    for(i in row)
        for (j in col)
            check neighbor of a[i][j]
            update b based on rules in PDF
```

```
}
display(Universe *u){
    curs_set(FALSE);
    for(i in row):
        for (j in col):
            if (u[i][j] live):
                mvprintw(i, j, "o");
    refresh();
    usleep(DELAY);
    clear();
}

input -> rows, cols
uv_create(rows, cols) -> *u1, *u2
uv_populate(u1, input)
u1->u2
for g in generations:
    update(u1, u2);
    swap(&u1, &u2);
    display(u1)
uv_print(u1, output);
uv_delete(u1);
uv_delete(u2);
```