Project - Game of Life

CSE/IT 113L

NMT Department of Computer Science and Engineering

"Testing leads to failure, and failure leads to understanding."

— Burt Rutan

"A language that doesn't affect the way you think about programming is not worth knowing."

— Alan J. Perlis

"Some of the best programming is done on paper, really. Putting it into the computer is just a minor detail."

— Max Kanat-Alexander

Contents

1 Conway's Game of Life

Your project is to write a program that simulates Conway's Game of Life. Your program will print a series of character matrices to your terminal window. You have been provided a print function to do this that requires no change, simply use this function for output. Your goal is to work on the backend of the game of life code, not to worry about the output.

Conway's game is a cellular automata (`http://en.wikipedia.org/wiki/Cellular_automaton`) game invented in 1970. The game takes place on an infinite 2-d grid, in which each space of the grid represents a cell. A set of rules is applied to each cell of the current generation in order to determine what cells live or die in the next generation.

For more information on the Game of Life see `http://www.conwaylife.com/wiki/Conway%27s_Game_of_Life` and `http://www.conwaylife.com` for general information. If you are unfamiliar with the game, reading this material will help you greatly in programming this project.

1.1 Rules of the Game of Life

From conwaylife.com, the rules of the game (really a simulation) are:

"The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead. Every cell interacts with its eight neighbors, which are the cells that are directly horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbors dies, as if by needs caused by under population.
- Any live cell with more than three live neighbors dies, as if by overcrowding.
- Any live cell with two or three live neighbors lives, unchanged, to the next generation.
- Any dead cell with exactly three live neighbors cells will come to life.

The initial pattern constitutes the 'seed' of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed – births and deaths happen simultaneously, and the discrete moment at which this happens is sometimes called a tick. (In other words, each generation is a pure function of the one before.) The rules continue to be applied repeatedly to create further generations."

2 Modeling the game of life

To model the game of life, you first create two grids (multidimensional arrays or a matrix) of size $width \times height$, where width is the width of the screen (in pixels), and height is the height of the

screen (in pixels) (see below).

It is important to note that your matrix is defined in the opposite way of how you would normally think: the width of the screen is stored in the rows of the matrix, while the height of the screen is stored in the columns of the matrix. The program will seg fault if you create a matrix with dimensions $height \times width$**. You have been warned.**

At any given time, one grid, say A, contains the current generation and the other grid, say B, contains the next generation.

To start the model, you initialize A with a pattern of cells that are alive. The initial pattern determines how quickly the ecosystem lives or dies. Some initial patterns produce ecosystems that last a number of generations, while other patterns die off quickly.

To calculate the generations, you loop through the current generation grid (A) determining if each cell is going to live or die in the next generation based upon the status of its 8 neighbors. You write the results to the next generation grid (B). At the next time step (generation) the roles of the grids reverse. You now scan grid B for who lives and dies and write to A. You continue on this manner indefinitely, until the game of life has reached some equilibrium or the pattern dies out.

2.1 Determining the grid

Obviously, you cannot model an infinite two-dimensional grid with a machine. Your grid is based on three factors: your screen dimensions, the spacing required to print a monospace font character, and how you model the boundaries of the grid.

The screen dimensions are the width and height of a 2-d grid in pixels. For example, 640 x 480, 800 x 600, 1024 x 768, 300 x 400, etc. Note: these are (width x height dimensions). The width and height determine the size of the rectangle you can draw to.

As pixels are too small to see, you will use characters, printable monospace ASCII symbols, to model cells on the screen. You are provided with a print function for your Game of Life output.

You create a matrix of $w_s \times h_s$ To hold the cells. **Important: you are creating a matrix with** w_s **rows and** h_s **columns and not the other way around.** To allocate space for the matrix you use an array of pointers to pointers. For example, to create a matrix of integers with dimension $rows \times cols$:

```
1  /* create a multi-dimensional array */
2  int **init_matrix(int rows, int cols)
3  {
4          int i;
5          int j;
6          char **a;
7          /* allocate rows */
8          a = malloc(rows * sizeof(char *));
9          if(!a)
10                 return NULL;
```

```
        for(i = 0; i < rows; i++) {
                /* allocate cols for each row */
                a[i] = malloc(cols * sizeof(char));
                if (!a[i]) {
                        for (j = 0; j < i; j++)
                                free(a[j]);
                        free(a);
                        return NULL;
                }
        }
        return a;
}
```

You ultimately need to create two matrices of size $w_s \times h_s$ to hold the alternating generations.

Finally, you have to determine the behavior of cells at the edge of the screen. **You will code three types of edges: the hedge, the torus, and the Klein bottle.**

1. **The hedge** – you add one row to the top and bottom and one column to the left and right side of your grid. All cells are dead in the newly added rows and columns. This allows the cells at the edges of your screen to be calculated the same way as interior cells. This doesn't lead to interesting patterns as cells die quickly along the edges, but is great for testing/development. **This edge type is the only one your GOL MUST implement successfully.**
2. **The torus** – you fold the left and right edges of your grid together and the top and bottom edges together to create a torus (`http://en.wikipedia.org/wiki/Torus#Flat_torus`. The top row of cells use the bottom row of cells in its calculations and vice versa; and similarly cells in the left and right edges you each other in their calculations. This leads to a much more interesting patterns as the cells move across screen edges. **Implementing this edge type successfully will gain you 10 bonus points.**
3. **The Klein bottle** – The Klein bottle joins the top and bottom edges of your grid (`http://en.wikipedia.org/wiki/Klein_bottle` –**read this**) and then the left edge is twisted before being joined to the right edge. Essentially, this means the upper left edge is joined to the lower right edge, and the lower left edge is joined to the upper right edge. **Implementing this edge type successfully will gain you 15 bonus points.**

The easiest way to model the torus is through modular arithmetic. If you find you are writing complex if logic to do this, come talk to me or a TA as there is a better way. And to create a Klein bottle, think about how you can model its behavior via a torus and a reflection about a horizontal line through the center of the screen.

2.2 Initializing the Grid - File I/O

You will initialize the grid by reading in a ConwayLife's Life 1.06 formatted file. You should be able to place the pattern anywhere on the screen. So you also need to set an initial coordinate for the

pattern. The initial coordinate (x_0, y_o) is relative to the upper left hand corner of the screen, which is considered (0, 0) in computer graphics. Unlike math, computer graphics flips the direction you move for a positive y-value – you move downward. For instance the point (10, 20) is located by starting at the origin moving 10 pixels to the right and 20 pixels down. A negative number for the y-value, would mean that you would move upward. For the point (10, -20), you would move 10 pixels to the right and 20 pixels up, which would be a point off the screen. The notation may seem strange at first, but notice how the coordinate system corresponds nicely with array index notation used in C and other programming languages for multi-dimensional arrays.

You must be able to read in the ConwayLife's file format Life 1.06 for the life pattern. The file format is available online at `http://www.conwaylife.com/wiki/Life_1.06`. Place all cells relative to the initial coordinate and movements mimic computer graphic conventions. A value of 0 0 in the file corresponds to the initial point x_o, y_o. Remember, a positive x-value moves right and a negative x-value moves left. A positive y-value moves down and a negative y-value moves up.

For the torus and Klein bottle, your cells should "wrap around" as you might choose a pattern and a initial coordinate that places the pattern outside the size of your screen. For example, if you are modeling a torus edge and a cell is outside the left edge of your screen, place it on the right side of the screen.

2.3 Command Line Arguments

Your program will get input from a number of command line arguments. This makes the program much more user-friendly. You will implement the following command line options:

```
usage: life -w 80 -h 24 -f glider_106.lif
-o 150,200 -e hedge

-w width of the screen argument 640, 800, 1024, etc.

-h height of the screen argument 480, 600, 768, etc.

-e type of edge. Values are hedge, torus, or klein (strings)

-f filename, a life pattern in file format 1.06.

-o x,y the initial x,y coordinate of the pattern found in the file. No
space between the x and y.

-H help, print out usage information and a brief description of each option. Note
↪  the capitalization, non standard, but -h is already taken.
```

If you want some extra credit on the basic hedge edge, implement the following command line options (see below):

```
-P filename, a life pattern in file format 1.06 \textbf{2 points}

-Q filename, a life pattern in file format 1.05 \textbf{2 points}

-p x,y the initial coordinate of pattern P \textbf{2 points}

-q x,y the initial coordinate of pattern Q \textbf{2 points}
```

As Linux is a (mostly) POSIX compliant system, you can use the function `getopt` found in `unistd.h`. This will simplify command line processing. For details, read `man 3 getopt`.

Make sure you have default values and error check all argument values. For instance, a user might enter 500 for width. You can fix this without causing the program to terminate. The program should work without any options if you have set all options to a default value.

2.4 gl.c, gl.h, Makefile, conwaylife.tar.gz

You are given the following files in `cse113_life.tar.gz`. You need to use these files in your project.

`gl.c` – this is the main game. This is where you place your game of life (initialization, calculation of living and dead cells, etc).

`gl.h` – the header file for gl.c.

`gl-demo.c` – this is a functional array-syntax implementation of Conway's Game of Life. This should help you get started with the basic hedge implementation.

`Makefile` – the makefile

`getopt_example.c` – provides an example for how to utilise compile line argument parsing in C.

`conwaylife.tar.gz` – a tar file of Life patterns in Life 1.05 and Life 1.06 format.

2.5 Valgrind/Leaks and freeing memory

Make sure you free all memory you allocated. To do this you add free calls to match all malloc calls.

To check for memory leaks with Valgrind/Leaks you need to run it with the following command options.

```
$ valgrind --show-reachable=yes --leak-check=full ./life
```

```
$ leaks --atExit -- ./life
```

Scan the results for errors in function calls you wrote. You are looking for errors in the code you wrote, not in the print function provided to you.

3 Comments and Doxygen

You will use Doxygen to generate HTML pages for your comments.

You will need to install Doxygen for your system and read the manual at:

`http://www.doxygen.nl/manual/`

The getting started section tells you how to quickly get up and working with Doxygen. You will create HTML pages. Make sure you set `EXTRACT_ALL` to `YES` in your `Doxyfile`.

Binaries are available for Ubuntu and other distros.

As you have been writing Doxygen style comments (@-tags) since the beginning of class, there is nothing special you need to do as far as commenting the code except to set Doxygen up correctly.

Use the following Doxygen commands for the header of each source code (*.c) file:

```
@file
@brief
@details
@author
@date
@todo
@bug
@remark (if needed)
```

Only the `@remark` section is optional. For functions, make sure you include a brief description and document all parameters and non-void return types.

For the header files (*.h), the Doxygen commands for the header are:

```
@file
@brief
@author
@date
```

Do not document the function declarations in the header file.

3.1 Doxygen Milestone

It is recommended that you have a TA look at your Doxygen setup and output before turning it in. Please make use of the lab time during each class to have your Doxygen checked.

4 Project Milestones

1. Implement the hedge version of the game of life **without** command line options. You should be able to manipulate the width, height, filename, and initial location all by manually changing variables and recompiling. Gliders are a good initial pattern to use for testing. You will notice a *glider* option in the demo code provided to you.
2. Implement the command line options.
3. Implement the torus, and Klein bottle (in that order) versions of game of life to achieve bonus points. Implement the ability to read and process ConwayLife's file format 1.05.

5 Pro Tip

Since this project is very long, it helps to break it into smaller pieces. For example, you could first make a 2-dimensional array and functions to compute whether something is alive or not, and test this by printing the arrays in the terminal. This is a very long lab, so start in advance! If you wait until the week before, you will struggle to finish in time!

6 File Format 1.05

Add the ability to read in Conway:life's file format Life 1.05. Details are at `http://www.conwaylife.com/wiki/Life_1.05`. Files in 1.05 format have the string "105" in their filename. You just need to add logic to the file command line options to distinguish between "105" and "106" files and process accordingly. **Implementing this will earn you 5 bonus points.**

7 Submission Guidelines

Submit a tarball of your project files, game of life seed files, and Doxygen files named `cse113_firstname_lastname_life.tar.gz` to Canvas before the deadline.