# Assignment 4: "The Game of Life" WRITEUP.pdf

Vincent Liu

February 13, 2023

## 1 Introduction

Assignment 4, titled "The game of life", requires students to replicate a simulation called "Conway's game of life". This simulation has 3 simple rules: If a cell is alive and has 2 or 3 neighbors, it lives. If a live cell has greater than 4 or less than 2 neighbors it dies. If a dead cell has exactly 3 neighbors, it becomes alive. As a result, the simulation's cells (usually) live and die many times over various numbers of iterations. Additionally, the simulation can also be bounded or toroidal. For the universe to be toroidal it means that if something moves in a certain direction and reaches a boundary, it will be moved to the border of the opposite side it was at. For clarity, this is similar to a flat map of the world; If one were to walk to the border of the map, they would not fall of the edge or be blocked. Instead they would be found on the opposite side of the map. When the simulation is not toroidal, the simulation will continue with borders. In this assignment students will be programming this simulation using specified functions in conjunction with the header file provided from the resources repository. This assignment requires students to become familiar with memory allocation, pointers, dereferencing and other fundamental concepts of C covered in previous assignments. Students will also be using ncurses to draw the simulation/game on a screen when running/executing the main file.

## 2 Functions used in this assignment

This assignment requires students to create two '.c' files in conjunction with a Makefile. The functions included in universe.c and life.c will be explained below.

### 2.1 universe.c

- Universe *uv_create(uint32_t rows, uint32_t cols, bool toroidal)
  Creates a universe and allocates memory for the universe structure
- void uv_delete(Universe *u)
  Frees the memory previously allocated for a specified universe. This is done to prevent any memory leaks. In short, it 'destroys' the galaxy.
- uint32_t uv_rows(Universe *u)
  This function returns the number of rows in a specified galaxy
- uint32_t uv_cols(Universe *u)
  This function returns the number of columns in a specified galaxy
- void uv_live_cell(Universe *u, uint32_t r, uint32_t c)
  This function sets a specific coordinate in a universe to be "true". In other words, it sets a specified coordinate cell to be "alive".
- void uv_dead_cell(Universe *u, uint32_t r, uint32_t c)
  This function sets a specific coordinate in a universe to be "false". This means that a specified coordinate cell will be set to "dead".
- bool uv_get_cell(Universe *u, uint32_t r, uint32_t c)
  This function returns whether or not a specified coordinate cell in the specified universe is 'alive' or 'dead'

- bool uv_populate(Universe *u, FILE *infile)
  This function takes an input file of coordinates and plots them in the specified universe. The function returns true or false depending on whether it was able to successfully plot all points. If a coordinate is outside the boundary of a universe, it will return false. If no file is specified for the input, it will default to using "stdin"

- uint32_t uv_census(Universe *u, uint32_t r, uint32_t c)
  This function returns the number of neighbors a specified cell has. Additionally, this function takes into account whether the specified universe is toroidal.

- void uv_print(Universe *u, FILE *outfile)
  This function prints data of a specified universe struct into an output file. If there is no specified output file, it will automatically be output to "stdout"

### 2.2 life.c

- void usage(char *exec)
  This function prints out an error message when necessary. It is called when an argument requiring additional input is unsatisfied, if the user inputs -h and requires a synopsis, or if an error occurs.

- void update(Universe *thing1, Universe *think2)
  This function was created by me to be used as an updater for swapping and updating the information between universe(s). The design for the output to be printed was that two universes were to be created and used one at a time. As a result, information had to be swapped from one universe to another while also being updated, after each iteration.

- int main(int argc, char **argv) This is the main function. It serves multiple purposes such as:
  - Print a synopsis when required
  - Determining what to do upon specific user inputs upon execution of executable file (Case wise)
    * If a user decides to mute ncurses, it will do so
    * If a user decides to use a specific input file, it will accept if the appropriate file exists and is compatible with the program.
    * If a user decides to use a specific output file, it will either update an existing file of the same name or create a new one. In both cases, if the appropriate file type is stated and is compatible with the program, it should work.
    * If a user decides to use a specific number of generations/iterations of the simulation, it abides
    * If the user specifies that it wishes for the created universe to be toroidal, it will do so.
  - Determining whether or not to use "stdin" "stdout" or a specified file
  - Creating the two universes necessary for the simulation to work
  - Executing the program either with or without ncurses depending on the user input
  - Calling the necessary functions to properly run the simulation.

## 3 UNIX commands used

- ./life: Used to run the executable main file. Expects a passed argument. Arguments are as follows:
  - -t : Specify that the Game of Life is to be played in a toroidal universe.
  - -s : Silence ncurses. Enabling this option means that nothing should be displayed by ncurses.
  - -n generations : Specify the number of generations that the universe goes through. The default number of generations is 100.

  - -i input : Specify the input file to read in order to populate the universe. By default the input should be stdin
  - -o output : Specify the output file to print the final state of the universe to. By default the output should be stdout.

- make / make all : Used to create executable files as well as compile the necessary '.c' files into objects files (".o" files)
- make clean : Used to remove the executable main file as well as remove the associated object files
- make format : Used to properly format all '.c' files.

# 4 Conway Game of Life - Visuals

In the following images, I will be showing an example of this program's simulation outputs iteration by iteration. The input file used will be "101.txt", which was provided in the resources repository. This simulation's universe is toroidal, meaning that the universe is infinite and loops back around to itself. This example consists of 18 iterations before all cells in the example universe are dead. The command used within the terminal for these outputs was:
"./life -i "101.txt" -t -s -n X -o "example.txt"
*("X" is the specified iteration number)
*("example.txt" was changed to the correlating iteration number for clarity in the directory)

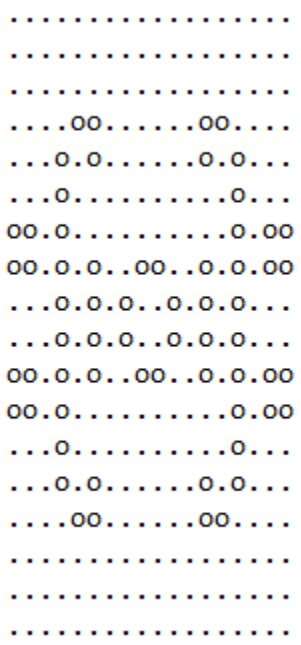



Figure 1: Iteration 0 shows the initial state of the universe. Coordinates are taken from an input '*.txt*' file and plotted onto a grid within the universe. The function that does this is "*uv_populate*" in universe.c

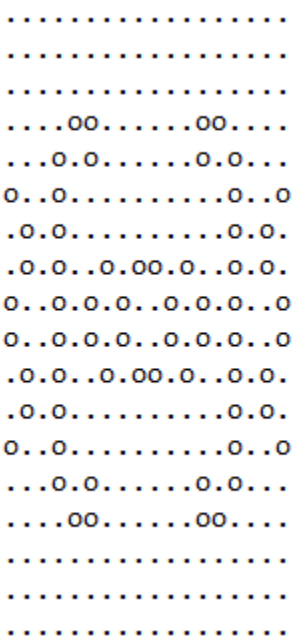
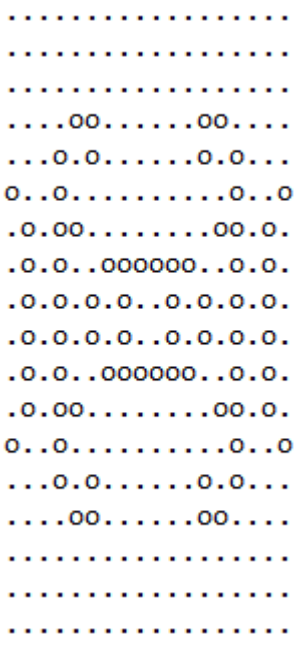
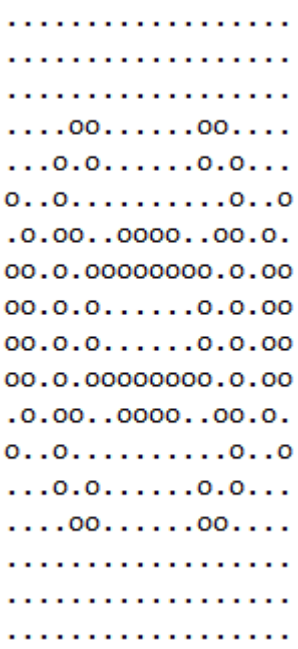


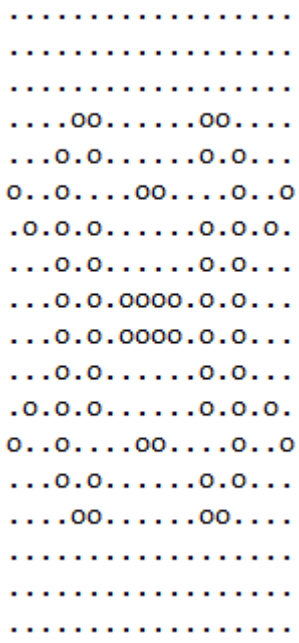

Figure 2: Iteration 1 of the universe. As can be seen, cells abide by the rules of the game



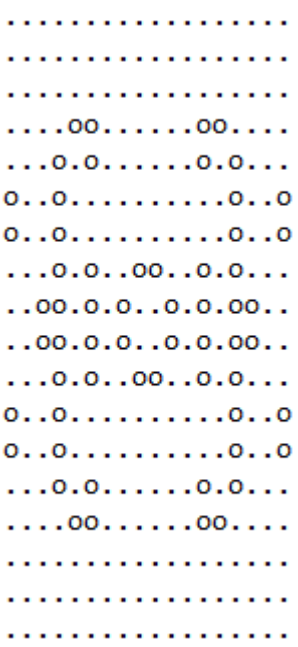
(a) Iteration 2



(b) Iteration 3

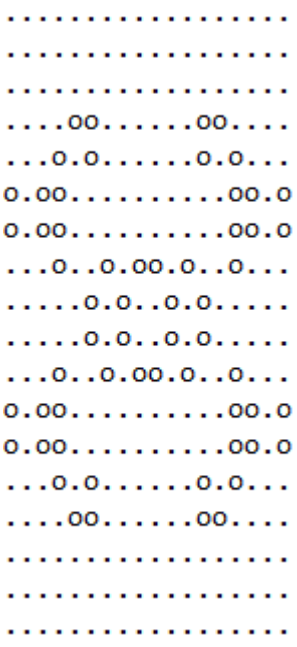


(c) Iteration 4

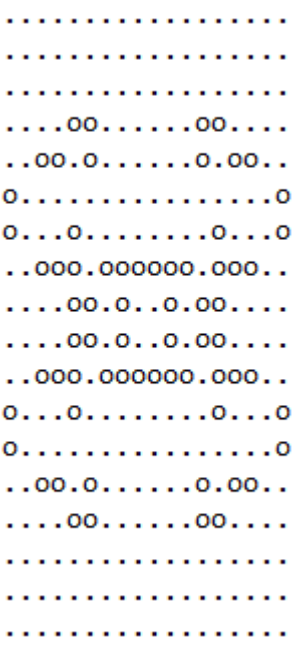


(d) Iteration 5

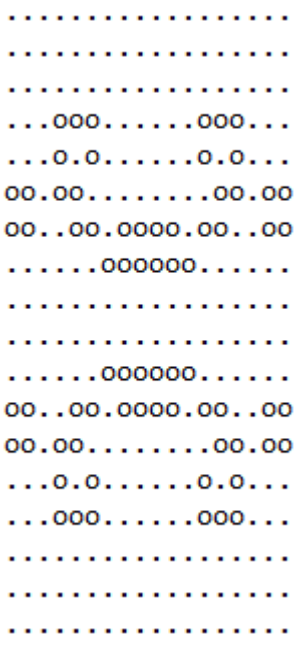
Figure 3: Iterations 2 - 5



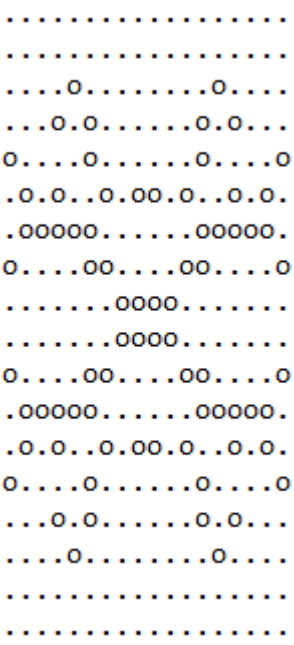
(a) Iteration 6



(b) Iteration 7

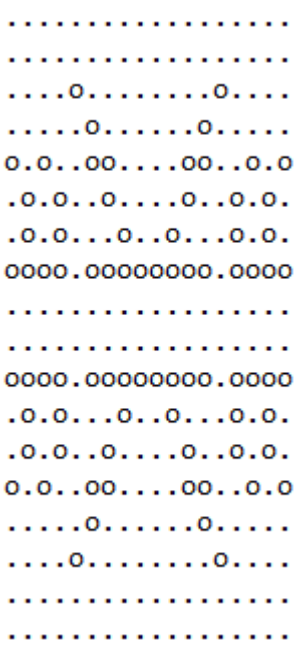


(c) Iteration 8

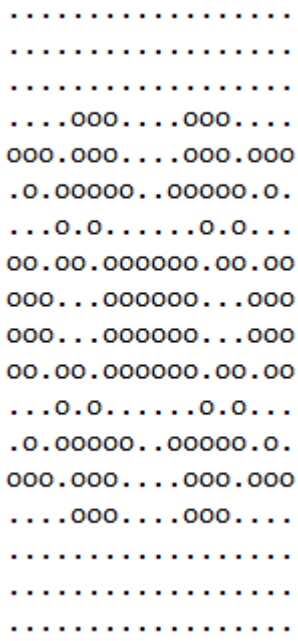


(d) Iteration 9

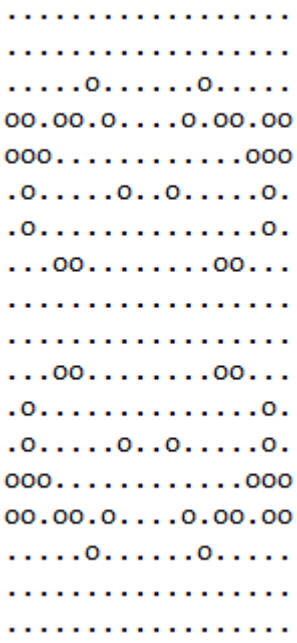
Figure 4: Iterations 6 - 9



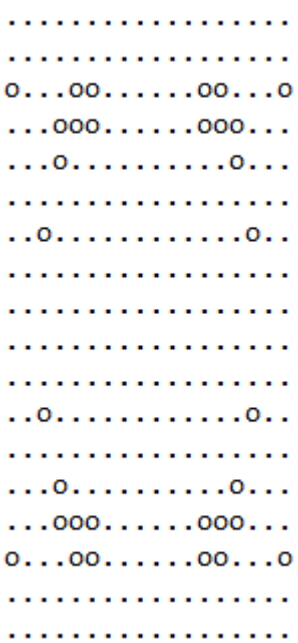
(a) Iteration 10



(b) Iteration 11

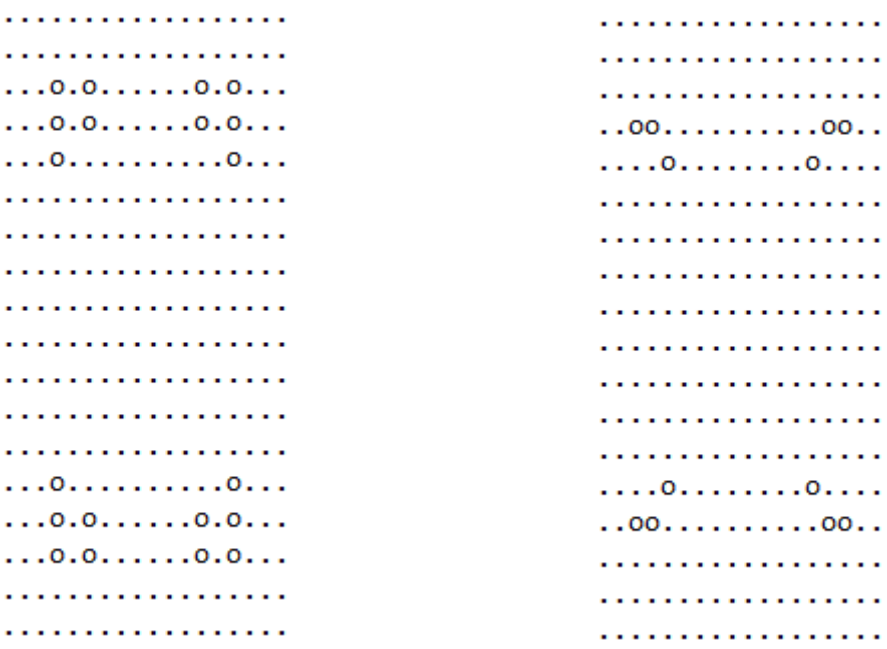


(c) Iteration 12



(d) Iteration 13

Figure 5: Iterations 10 - 13



(a) Iteration 14

(b) Iteration 15

Figure 6: Iterations 14 - 15

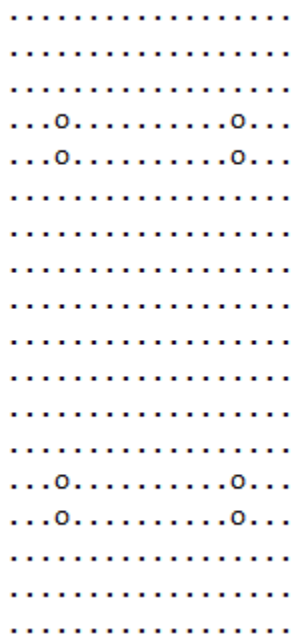


Figure 7: Iteration 16 - The last iteration of this universe that contains any live cells

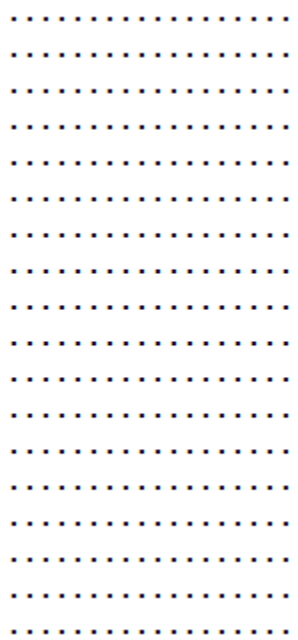


Figure 8: Iteration 17 - Death of all life in the Universe, as all remaining cells from iteration 16 had an insufficient number of neighbors to remain alive

# 5 Files in this assignment's directory

- universe.h
  - specifies the interface to the Universe ADT. This file is provided and may not be modified.
- universe.c
  - implements the Universe ADT.
- life.c
  - contains main() and may contain any other functions necessary to complete implementation of the Game of Life.
- Makefile
  - Used to compile all '.c' files into '.o' object files, create an executable file, or remove all object files and executables from the directory. Is used via commands such as "make', 'make all', 'make format', 'make clean'
- README.md
  - Text file in Markdown format that describes how to build and run the program.
- WRITEUP.pdf
  - Describes what the program does, what the assignment is about, gives insight on the results found, UNIX commands used, and what I learned as a result of completing this assignment
- DESIGN.pdf
  - Describes the design for the program thoroughly with pseudocode

# 6 What I learned

From this assignment I learned what Conway's game of life was, and how to implement it into a C program. I was able to gain more experience and knowledge on working with pointers in C, structures, header files, as well as debugging with tools such as valgrind. Valgrind is a tool most often utilized by programmers for memory leak detection and memory debugging. I found how useful debugging with valgrind was, especially when getting errors such as segmentation faults and memory leaks - issues which were problems I faced quite often towards the completion of this assignment. I found it to be quite interesting how structures work in C, and how useful they can be when actually programming. Especially after working with structures for this assignment and assignment 3, I was able to better understand their potential and practicality. Though I initially understood how they worked (during asgn3), I found myself struggling to grasp how and access them, which led to errors as memory was not allocated properly. By the end of it I was able to overcome this obstacle by reading and understanding the manual for functions such as malloc() and calloc() (via "man malloc()" and "man calloc()"). As I worked on assignment 4 I was able to use these functions properly in conjunction to valgrind. I believe assignment 3 and 4 really helped me learn a lot about structures in C, as well as memory allocation and debugging.

# 7 Credit

- TA John
  - For providing insight and guidance on: uv_delete, uv_dead_cell