Course: CS 162 – Introduction to Computer Science II
Professor: Terry Rooker
Date: April 7, 2014

Design Document, Game of Life
by Benjamin R. Olson
April 7, 2014

Discovering Requirements:

Task: Implement Conway's Game of Life.
Game of Life Setup, Rules, and Execution:

Setup:

80 X 22 grid of cells, where each cell is an char that denotes toggle between two states – live ('*') or dead (' '), starting configuration of cells determined from array hard-coded into the program – user chooses from a list of a few/several configurations).

Rules:

1. Each cell defined as having 8 neighbor cells (edge and corner cells' neighbors assumed to be ' ' (dead) if those neighbors are not within the grid boundaries).
2. If a live cell x has exactly 0 or 1 live neighbor cells...OR if it has more than 3 live neighbors, cell x will die in the next generation.
3. If a dead cell x has exactly 3 live neighbor cells, cell x will become live in the next generation.
4. When a cell is evaluated for whether it should change state, the state of that cell does not actually change until the next generation.

Execution:

After each generation is rendered, the user decides to render the next generation by hitting <Enter/Return> on the keyboard, or quit the game execution by pressing any key and return to the main menu.

Designing a possible solution:

Data representation:

Data #1 (d1): A single dimensional array of chars (' ' = dead, '*' = live) representing the current generation, a grid of cells (use modulus arithmetic on index value to navigate to the correct row): 80 columns by 22 rows = 1760 array elements (cells), where
row 1: cells 0 – 79, row 2: cells 80 – 159, ...where last cell has index 1759.

Data #2 (d2): A single dimensional array if chars, similar to #1, representing the next generation to be rendered.

Data #3 (d3): A 2d (jagged?) array representing starting configurations, where each sub-array (2nd dimension) is the indexes of all live cells. For example, data #3[3] would represent the 4th configuration and evaluate to an array of ints which are the indexes of all live cells for the 4th configuration. The user decides from data #3 which configuration to start the game with.

Functions descriptions (helper functions may be added during implementation, in addition to these functions described here):

Function #1 (f1): takes 2 arrays by reference (current generation, data #1 and next generation, data #2) and modifies data #2 based on the rules applied to data #1. Part of this function is determining how many live cells surround the current cell being considered, and storing that number in a local counter.

Function #2 (f2): displays data #2, the next generation.

Function #3 (f3): sets data #1 to equal data #2, to prepare for the next call to function #1.

Function #4 (f4): Helper to function #1. This function takes an int representing the number of live surrounding cells that surround cell x and evaluates that number based on the 4 rules to return true (cell x will be live in next generation) or false (cell x will be dead in next generation).

Flow of execution (refer to descriptions of data and functions above):

Step a. Data items declared: d1 (current generation grid), d2 (next generation grid), d3 (configurations)

Step b. Data items initialized: d3

Step c. User asked to choose from a list of starting configurations or quit program.

Step d. Do:

{f1 (d1, d2) {...f4...d2 modified}, f2 (d2), f3 (d1, d2), f3 (d1, d2)}
while user hits only enter and no characters.

Step e. User is returned to "Step c."

Example program flow based on the above design but with smaller grid 5 rows X 10 columns (This smaller, oscillating configuration could be an initial test case):

User chooses configuration represented by array {14, 24, 34}.
Configuration is loaded into d1 array, charted below, and displayed (where "14v1" means d1 index 14 with value of '*', and "15v0" means d1 index 15 with value ' '):

0v0	1v0	2v0	3v0	4v0	5v0	6v0	7v0	8v0	9v0
10v0	11v0	12v0	13v0	**14v1**	15v0	16v0	17v0	18v0	19v0
20v0	21v0	22v0	23v0	**24v1**	25v0	26v0	27v0	28v0	29v0
30v0	31v0	32v0	33v0	**34v1**	35v0	36v0	37v0	38v0	39v0
40v0	41v0	42v0	43v0	44v0	45v0	46v0	47v0	48v0	49v0

Displayed as:

```
|       |                              |       |
|   *   |                              |       |
|   *   | → then user hits enter →     |  ***  | → user hits 'q',<Enter> → user hit 'q',<Enter> → exit(0)
|   *   |  (data is evaluated and      |       |
|       |    changed)                  |       |
```