Jeffrey Schachtsick

CS 162: Intro CS II

April 6, 2015

Assignment #1: Conway’s Game of Life

**Understanding:**

In this assignment, I will be creating a program that emulates the zero player game of John Horton Conway’s Game of Life. This game has the user create a configuration of cells and see how the configuration changes over time or from generation to generation. In general the configuration is made of a bunch of cells that are either alive or dead. Based on the rules, the configuration’s time of existence depends upon the rules of the game. These sets of rules, help to evolve/devolve the configuration over its generations.

In this program I will need to follow the set of rules established by the game as follows:

1. If an occupied cell has zero or one neighbor, it will die of being alone;
2. If an occupied cell has more than three neighbors, it will die of overcrowding;
3. If an empty cell has exactly three occupied neighbor cells, there is a birth of a new cell to replace the empty cell;
4. Births and deaths are instantaneous and occur at the change of generation.

In creating this program, I will need to create a visible display with a 40 by 20 cells. I will need to signify in some way how a cell is alive and how a cell is dead. The display of the 40 by 20 cells, will be similar to a world or sphere. As in, the right vertical border of cells are neighbors to the left vertical border. Same goes with the upper horizontal border is neighbor with the lower horizontal border.

The user will be asked which of three configurations to choose from: oscillator pattern; glider pattern; and glider gun or cannon pattern. The oscillator pattern changes between generations as 3 vertical or horizontal alive cells. The glider, is a configuration of about 5 alive cells moving through the display like an arrow. The cannon configuration, looks like two entities moving back and forth and creating gliders.

The purpose of this assignment, is to hopefully achieve a better understanding of software design and testing. Refining the ability to analyze the problem by decomposing big problems into smaller and solvable problems that can be turned into psuedocode and eventually actual code. Also, write a testing document where I can go back to test my code and get the results I need.

**Testing Plan:**Design some tests you can perform to verify that your program meets the given specifications.  They should be representative enough that if your program passes them, you would have a high level of confidence that your program is ready to be submitted.  Create a table with two columns: one for the description of each test and one for the expected results of each test.

**Name Entry Criteria Exit Criteria**

|  |  |  |
| --- | --- | --- |
| General Oscillator Test | 1. Choose Oscillator  2. Choose 5 Generations  3. Choose origin x-coordinate of 5  4. Choose origin y-coordinate of 5 | For the first generation, 3 ‘X’s appear vertically, with middle (origin) at 5,5 location.  Second generation is again 3 ‘X’s, but horizontally. The middle X is still at 5,5 location.  Third generation appears as the first generation.  Fourth appears as the second generation.  Fifth appears as the first generation. |
| General Glider Test | 1. Choose Glider 2. Choose 10 Generations 3. Choose origin x-coordinate of 5 4. Choose origin y-coordinate of 5 | For the first generation, should see a figure appear like below  . . X .  . . . X  X X X X  For the Second generation and onward, should see about 5 X figures, but move over each generation toward bottom right. |
| General Cannon Test | 1. Choose Cannon 2. Choose 50 Generations | For the first generation there should be a block toward the top and a block at the bottom of the world. Two live cell configuration entities appear between the two blocks. After each generation, the entities will come together and create a glider. The two entities will go apart until hitting the blocks and come together to start to make another glider. |
| World Bounds | 1. Choose Glider 2. Choose 200 Generations 3. Choose origin x-coordinate of 5 4. Choose origin y-coordinate of 5 | For the first generation, should see a figure appear like below  . . X .  . . . X  X X X X  For the Second generation and onward, should see about 5 X figures, but move over each generation toward bottom right.  When it reaches the edge, entities should wrap to the other side of the world. |

**Design:**Describe or draw out your design for how the program should behave using pseudocode or flowcharts, following the given guidelines for those methods.

Pseudocode

Class World

private:

* char world [rows][columns]

public:

- Default Constructor

- Print the world

Default Constructor Method

Loop through each row

Loop through each column

apply a ‘.’ for each index.

Print World Method

With the world object

Loop through each row

Loop through each column

Print the index

Create Oscillator Method

Parameters include x and y coordinates of starting point

Add live cell for starting point and to the cells above and below

(Out of bounds conditions)

If cell is to be -1, place cell at row 39

If cell is to be 40, place cell at row 0

Create Glider Method

Parameters include x and y coordinates for origin point

Add live cells for the other 4 starting points

Create conditions for out of bounds, to wrap around the world.

Create Cannon Method

Add live cells to specified location from a reference design on how to start out the Cannon.

Game Rules Method

1. If an occupied cell has zero or one neighbor, it will die of being alone;
2. If an occupied cell has more than three neighbors, it will die of overcrowding;
3. If an empty cell has exactly three occupied neighbor cells, there is a birth of a new cell to replace the empty cell;
4. Births and deaths are instantaneous and occur at the change of generation.

Parameters include x and y coordinates of starting point

Loop through each cell in the world.

Go through each neighbor cell

If the neighbor cell is alive

count toward number of alive neighbor cells

If neighbor cells count is less than 2

Cell must be dead, per Rule #1

If there are more than 3 alive neighbor cells

Cell must be dead, per Rule #2

If cell is dead and has 3 neighbors

Cell must be alive, per Rule #3

If cell is dead and has 2 neighbors

Cell must be dead

Else

Cell is alive.

Oscillator Function

* Asks user how many generations to see pattern
* Asks user starting location.
* World with creating Oscillator method
* Loop through each generation after 0, until last generation
  + Apply rules to each cell in the world
  + Print the world

Glider Function:

* Asks user how many generations to see pattern
* Asks user starting location.
* Set the generations
* Set the starting location
* Set the pattern
* Play board function

Cannon Function:

* Asks user how many generations to see pattern
* Asks user starting location.
* Set the generations
* Set the starting location
* Set the pattern
* Play board function

Main

* Welcome the user to the program
* Create the World class
* Do while loop starts here, until quit selected
  + Ask the user which pattern to use or quit program: oscillator pattern; glider pattern; or cannon pattern.
  + User answer question going to a switch/case
    - Oscillator Function
    - Glider Function
    - Cannon Function
    - Quit program
* Exit program