# Conway's Game of Life Exercise

## Goal

The goal of this exercise is to calculate the next generation of Conway's game of life given any initial state. Take a look at the following for some background on Conway's game of life:

http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

## Background

The playing field for Conway's game of life consists of a two dimensional grid of cells. Each cell is identified as either alive or dead. For this exercise, let's assume the playing field is an 8x6 grid of cells (i.e. 8 columns, 6 rows).

The challenge is to calculate the next state of the playing field given any initial grid state. To find the next state, follow these rules:

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with more than three live neighbors dies, as if by overcrowding.
3. Any live cell with two or three live neighbors lives on to the next generation.
4. Any dead cell with exactly three live neighbors becomes a live cell.
5. A cell's neighbors are those cells which are horizontally, vertically or diagonally adjacent. Most cells will have eight neighbors. Cells placed on the edge of the grid will have fewer.

## Input / Output

Design your program to accept an initial 8x6 grid state where each cell is identified as alive or dead. Your program should output a new state by following Conway's game of life rules. Your program should display the new state of the playing field. You may choose the data model for representing your grid and how to display the state of the grid. Here is a very simple command line output example:

| Initial state | Updated state |
|---|---|
| .....O.<br>OOO...O.<br>.....O.<br>.....<br>...OO...<br>...OO... | .O.....<br>.O...OOO<br>.O.....<br>.....<br>...OO...<br>...OO... |

## Implementation

You may choose the language for your exercise implementation. Java and JavaScript are popular choices. Please include unit tests with your solution. It is important that your unit tests demonstrate that your software works as designed.

Please design and develop a solution to the exercise on your own. You will be expected to explain your design choices and talk about implementation details after the exercise is complete.

## Delivery

Once you are finished with the exercise, zip up your project and email it to your Cardinal audition contact. Please include instructions describing how to build, test and run your software.

Your Cardinal audition contact will give you a due date for your solution. While a simple solution meeting the provided requirements is all that is needed, feel free to add additional details or features if you like. Please document any additional features that you add.

## Additional Feature Ideas

- Loop through states: updated state becomes initial state and recalculate
- Support arbitrarily sized grids
- Emphasis on performance
- Improved result display