# CMPS 1053 – Program 1 John Conway's Game of Life

# Overview

The Game of Life was invented by John Conway (as you might have gathered). It consists of a collection of cells which, based on a few mathematical rules, can live, die or multiply. The "game" is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input from humans. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.

The universe of the Game of Life is an infinite two-dimensional [orthogonal](http://en.wikipedia.org/wiki/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*](http://en.wikipedia.org/wiki/Moore_neighborhood), which are the cells that are directly horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if caused by underpopulation.
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.

Below is an example of the first 3 generations from the following initial configuration (green = life).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Generation 0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  | |
| Generation 1 | Generation 2 | Generation 3 |

# Program

Your implementation will not be an infinite two-dimensional orthogonal grid of square cells, but a finite 2-dimensional array of integers*.* The maximum size of your world will be 78 columns by 24 rows (a size that fits nicely in a DOS window), but it could also be anything smaller. Because of this you will need to dynamically create a 2D array after your program starts to run.

The following function will do this for you:

//Necessary declaration!

int \*\*A;

//Function Call

make2dArray(A,MaxRows,MaxCols);

//Function Definition

bool make2dArray(int \*\* &x, int numberOfRows=24, int numberOfColumns=79){

// Create a two dimensional array.

   try {

        // create pointers for the rows

          x = new int \* [numberOfRows];

          // get memory for each row

          for (int i = 0; i < numberOfRows; i++)

              x[i] = new int [numberOfColumns];

          return true;

       }

   catch (bad\_alloc) {return false;}

}

**Generation 0**

How do we start our zero person game? We need to have an initial starting point (generation 0) for our game to be played. We are going to do this by reading in a file that will be formatted like below and named “***gen\_zero.txt***”:

8 10

0000000000

0000000100

0000011000

0000110000

0000100000

0001000000

0000000000

0000000000

The first two numbers (8 and 10) define the size of your world and the following numbers are the initial state of your world.

So, we have a 2D array to hold our world of 1’s and 0’s, how do we get from one generation to the next and figure out who lives, dies, or stays?

From the initial example above, let’s look at the transition from generation 2 to generation 3. We can’t just go and change values in the original array, like change a 1 to a zero to kill someone or vice versa, because it might affect whether another cell lives or dies! We have to keep the original world unchanged, figure out births, deaths, and who stays, and ***then*** make our changes all at once. Our technique is going to use a temporary array where certain values represent certain events:

* 1 = birth
* -1 = death
* 0 = no change

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| Generation 2 | Calculate Temporary Values | Both Arrays Added Together |

After we calculate the values in the temporary array, we simply traverse the original array, and “ADD” the original cell with the temporary cell. Doing this makes all of our changes at the same moment, and lets us go from one generation to the next.

A birth is a 1, because when you add a 1 to a 0, you get a 1 (amazing).

A death is a -1, because when you add a -1 to a 1, you get a 0 (more amazing).

No change is a 0, because …

**How do we decide?**

How do you decide if a cell lives or dies? You have to calculate what happens at each cell location for each generation. You will write a function to do this.

**Function Inputs:**

* WorldArray
* Width and Height of the WorldArray
* Cell coordinates of which cell to analyze

**Returns**:

* 1 for a birth
* 0 for no change
* -1 for a death

Look at the example below. The ‘x’ is the cell we are looking at. It’s 8 neighbors are referenced by adding or subtracting to the original coordinate pair of x.

|  |  |  |
| --- | --- | --- |
| -1,-1 | -1,0 | -1,+1 |
| 0,-1 | x | 0,+1 |
| +1,-1 | +1,0 | +1,+1 |

You must be careful to check and see if your on any bordering row or column in your array! You don’t want to be checking cells that are **not** actually in your array.

I will discuss the problem more thoroughly in class and have starter code for you. Below are some requirements that must be met!

We will also discuss when and how to terminate your simulation and what is expected in your output file.

# Pseudo Test Cases

1. Discussed In Class

**References**:

* Article that discusses the game: <http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_gamelife/ConwayScientificAmerican.htm>
* Wikipedia: <http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life>
* Game of life website: <http://www.bitstorm.org/gameoflife/>

Define an object like the following:

class LifeForm{

private:

int state; //[1=alive,0=dead];

int state\_transition; //[-1=dying,0=no-change,1=birth];

int x; //location on x axis

int y; //location on y axis

int neighbors; //number of neighbors

string last\_event; //birth, death, nothing

public:

LifeForm(){

state=0;

}

LifeForm(int life){

state=life;

}

LifeForm(int life, int InX, int InY){

State = life;

x=InX;

y=InY;

}

void SetState(int InState){

state = InState;

}

void SetXY(int InX, int InY){

x = InX;

y = InY;

}

void SetStateTransition(int transition){

state\_transition=transition;

}

int CountNeighbors(){

}

bool IsNeighbor(LifeForm Array[],int x,int y){

}

}