May 2020 - Challenge

<< April | May

On Saturday, April 11, 2020, the mathematical genius John Horton Conway passed away.
This month's challenge is dedicated, with deep love, to his memory.

Conway researched many topics in mathematics. One of his most famous inventions is the Game of Life.
In the standard game, each cell has eight neighbors; in our version, each cell has only four neighbors (only those cells with adjacent edges).
The standard rules can be formulated as 000100000;001100000 (if the cell is empty, it is born if it has exactly three live neighbors; if the cell is alive, it stays alive if it has two or three live neighbors).

In our version of the game, the rules 01100;00010 mean that a cell is born if it has one or two neighbors, and stays alive if it has three. If we start with a single cell in the middle of an 11x11 torus board, then after 15 generations, you will have an alternating chess-like pattern, and after 16 steps, just the four corners.
Your task, this month, is to find rules for our version of the game and an initial input on an 11x11 torus board that will lead, after at least 100,000 generations, to a 72-long cycle.

Typical of many IBM Ponder This challenges, this is month's challenge asks the solver to find solution(s) in an intractably large search space.

Below is my initial solution and its submission to the IBM site.

Dear Ponder This,

Using the rules **01001;00101**

The board below (1 = alive; 0 = dead) generates 480,738 distinct generations before leading to a 72-long cycle.

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 0 0 0
0 0 0 0 **1 0 1** 0 0 0 0
0 0 0 0 **1 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 0 0 0 0 0 0 0 0 0

Thank you for considering!
Charles Joscelyne

After submitting the above it occurred to me that the challenge would most likely also include a tribute to J. Conway. Indeed (surely not a coincidence) there is also a solution with the initial C formed by the live cells in the board's center.

Using the same rules **01001;00101**

The letter C board below (1 = alive; 0 = dead) generates 162,813 distinct generations before leading to a 72-long cycle.

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 0 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 **1 1 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 0 0 0 0 0 0 0

On how I arrived at a solution:

After building the supporting Java code to create and explore an 11x11 torus board, all 1024 rule sets for various simple configurations were tested for cycles. As my intuition grew by repeated experimentation the initial solution described above emerged.

The Java class IBMConway code along with the experimentation/test code is included below:

```
/**
* IBMConway code to explore/solve May 2020 IBM Ponder This Challenge
*/
 import java.util.*;
public class IBMConway
{
   //11X11 Torus Board
   private int[][] board = new int[11][11];

   //Returns the board
   public int[][] getBoard(){
     return  board;
   }
  //Utility to update the board at a given cell and value
    public void setBoard(int row, int col, int val){
      board[row][col] = val;
  }

   //Initialize board to an all off state
   public IBMConway()
   {
      for (int row = 0; row <board[0].length; row ++){
          for (int col = 0; col < board[0].length; col++){
              board[row][col] = 0;
          }
       }
   }

   //Reset the board to an off state
   public void clear(){
     for (int row = 0; row <board[0].length; row ++){
          for (int col = 0; col < board[0].length; col++){
              board[row][col] = 0;
          }
       }
   }

  //Utility to display board for testing
  public void displayBoard(){
     System.out.println();
     for (int row = 0; row <board[0].length; row ++){
          for (int col = 0; col < board[0].length; col++){
              System.out.print(board[row][col]+" ");
          }
          System.out.println();
       }
      System.out.println();
   }

   //Returns the number of on states in a Von Neumann neighborhood of a cell
  public int getAlive(int row, int col){
     int ct = 0;
     if(board[(row+1)%board.length][col] == 1)
         ct++;
```

```
   if((row -1 == -1) && board[board.length-1][col] == 1)
      ct++;
    if((row -1 >=0) && board[row-1][col] == 1)
      ct++;
   if(board[row][(col+1)%board.length] == 1)
      ct++;
   if((col -1 == -1)  &&  board[row][board.length-1] == 1)
      ct++;
  if((col -1 >=0)  &&  board[row][col-1] == 1)
      ct++;
   return ct;
 }

 //Given a birth and death length 5 Strings of 0 and 1 updates the board
 //according to the birth and death rules. This is best explained here:
 //http://www.research.ibm.com/haifa/ponderthis/challenges/May2020.html
public void nextGen(String birth, String death){
   //buffer array to store the next board generation
   int[][] temp = new int[11][11];

  for (int row = 0; row <board[0].length; row ++){
       for (int col = 0; col < board[0].length; col++){
          temp[row][col] = 0;
       }
   }

 for (int row = 0; row <board[0].length; row ++){
       for (int col = 0; col < board[0].length; col++){
         if(board[row][col] == 0 && birth.substring(getAlive(row,col),getAlive(row,col)+1 ).equals("1")){
            temp[row][col] = 1;
          }
         else if(board[row][col] == 1 && !death.substring(getAlive(row,col),getAlive(row,col)+1).equals("1")){
              temp[row][col] = 0;
          }
         else{
           temp[row][col] = board[row][col];
          }
    }
  }

 //update board using the buffer array
 for (int row = 0; row <board[0].length; row ++){
       for (int col = 0; col < board[0].length; col++){
         board[row][col] = temp[row][col];
       }
    }

}

//Returns a 121 length String version of  the board
//Used below in the getPeriod() method
public String getStr(){

  String s = "";
```

```
    for (int row = 0; row <board[0].length; row ++){
          for (int col = 0; col < board[0].length; col++){
            s+=board[row][col];
   }
  }
  return s;
 }

 //Given a 121 length String of 0(s) and 1(s) this produces the unique board
 //corresponding to the String. This is not used by any method nonetheless
 //the method was used in testing.
 public void strToBoard(String s){
      for (int row = 0; row <board[0].length; row ++){
          for (int col = 0; col < board[0].length; col++){
          System.out.print(s.substring(row+col,row+col+1));
          board[row][col] = Integer.parseInt(s.substring(row+col,row+col+1));
          //System.out.print(Integer.parseInt(s.substring(row+col,row+col+1)));
   }
 }
 }

 //Starting from some initial board configuration, the board evolves
 //into distinct configurations until a certain configuration appears and
 //subsequently appears again in latter generations.
 //This method returns a 2 element array containing the index of the last distinct board
 //configuration and the length of the ensuing period.
 //This is best explained here:
 //http://www.research.ibm.com/haifa/ponderthis/challenges/May2020.html
 public int[] getPeriod(String birth, String death){

    int ct = 0;
    int[] indxs = new int[2];

    //The ArrayList is used to track position.
    ArrayList<String> ls = new ArrayList<String>();

    //The HashSet detects duplicates.
    HashSet<String> hs = new HashSet<String>();
    //Detects  a duplicate
    boolean flag = true;

    while(flag){
       //Generate next board configuration
       nextGen(birth,death);
         //if able to add to the HashSet add and also add to the ArrayList
         if(hs.add(getStr())){
            ls.add(getStr());
            ct++;
           }
           else{
            flag = false
       }
     }
```

```
    indxs[0] = ls.indexOf(getStr());//index of last distinct configuration
    indxs[1] = ct - ls.indexOf(getStr());//length of the period
    return indxs;//return two element array
  }
}


/**
 * Explore all 1024 rules for a given board configuration
 */
import java.util.*;
public class IBMConWayTester
{
  public static void main(String args[]){
    //Code to set and display the board being explored
    //In this case the C Conway board configuration
     IBMConway r = new IBMConway();
          r.setBoard(4,4,1);
          r.setBoard(4,5,1);
          r.setBoard(4,6,1);
          r.setBoard(6,4,1);
          r.setBoard(6,6,1);
          r.setBoard(6,5,1);
          r.setBoard(5,4,1);
     r.displayBoard();
     /**
      * Try all 1024 possible board rules for the C Conway board configuration.
      */
     for(int i = 0; i <32; i++){
       for(int j = 0; j <32; j++){
          String b = String.format("%5s", Integer.toBinaryString(i)).replace(' ', '0');
          String d = String.format("%5s", Integer.toBinaryString(j)).replace(' ', '0');
          IBMConway bd = new IBMConway();
          bd.setBoard(4,4,1);
          bd.setBoard(4,5,1);
          bd.setBoard(4,6,1);
          bd.setBoard(6,4,1);
          bd.setBoard(6,6,1);
          bd.setBoard(6,5,1);
          bd.setBoard(5,4,1);

          int[] arr = bd.getPeriod(b,d);

          int t1 = arr[0];
          int t2 = arr[1];

          //If a rule set is found display
          if(t1>=100000 && t2 == 72)
          {
            System.out.println(t1+" "+t2+" "+b+" "+d);

          }}}}}
```