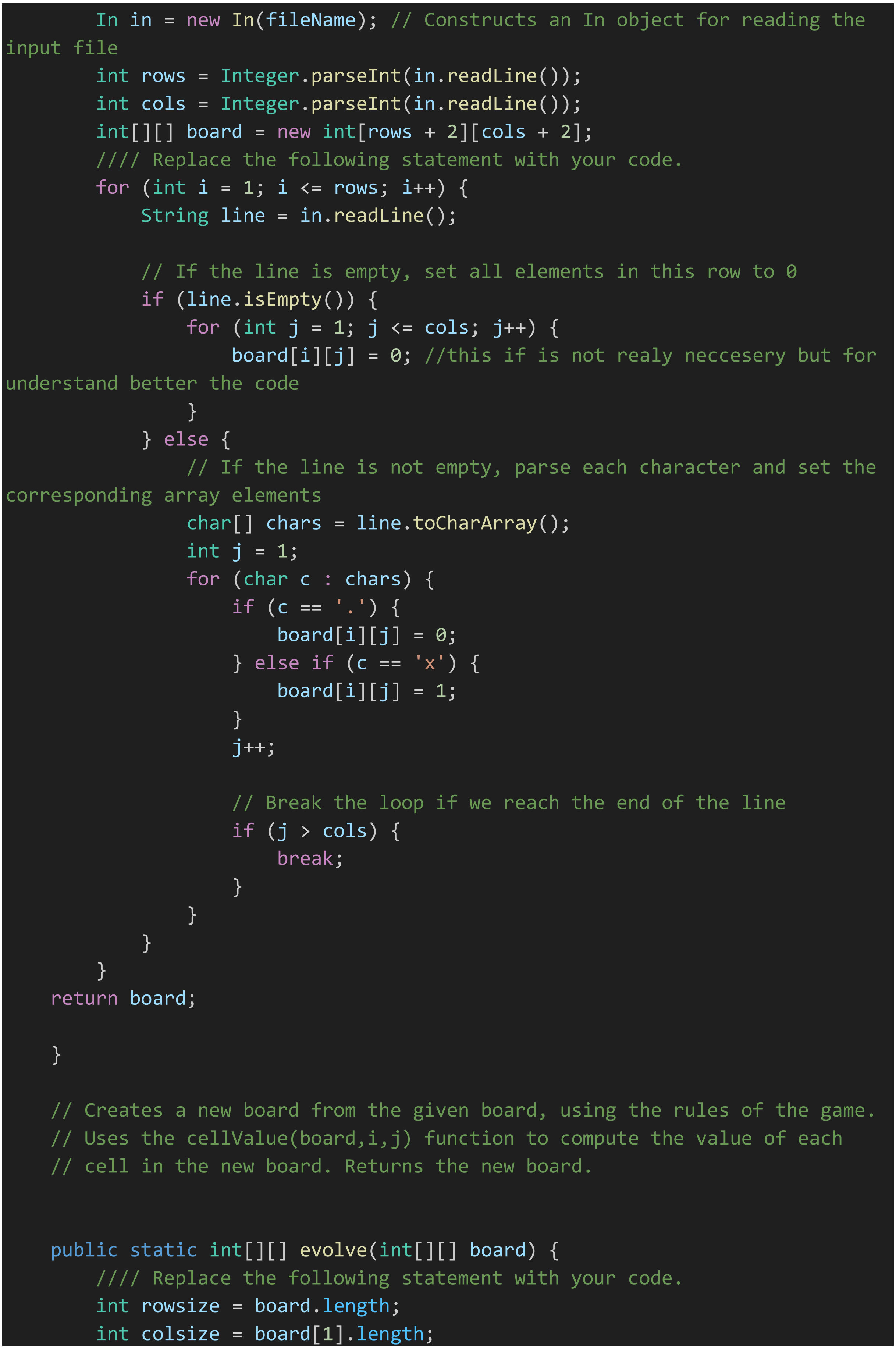
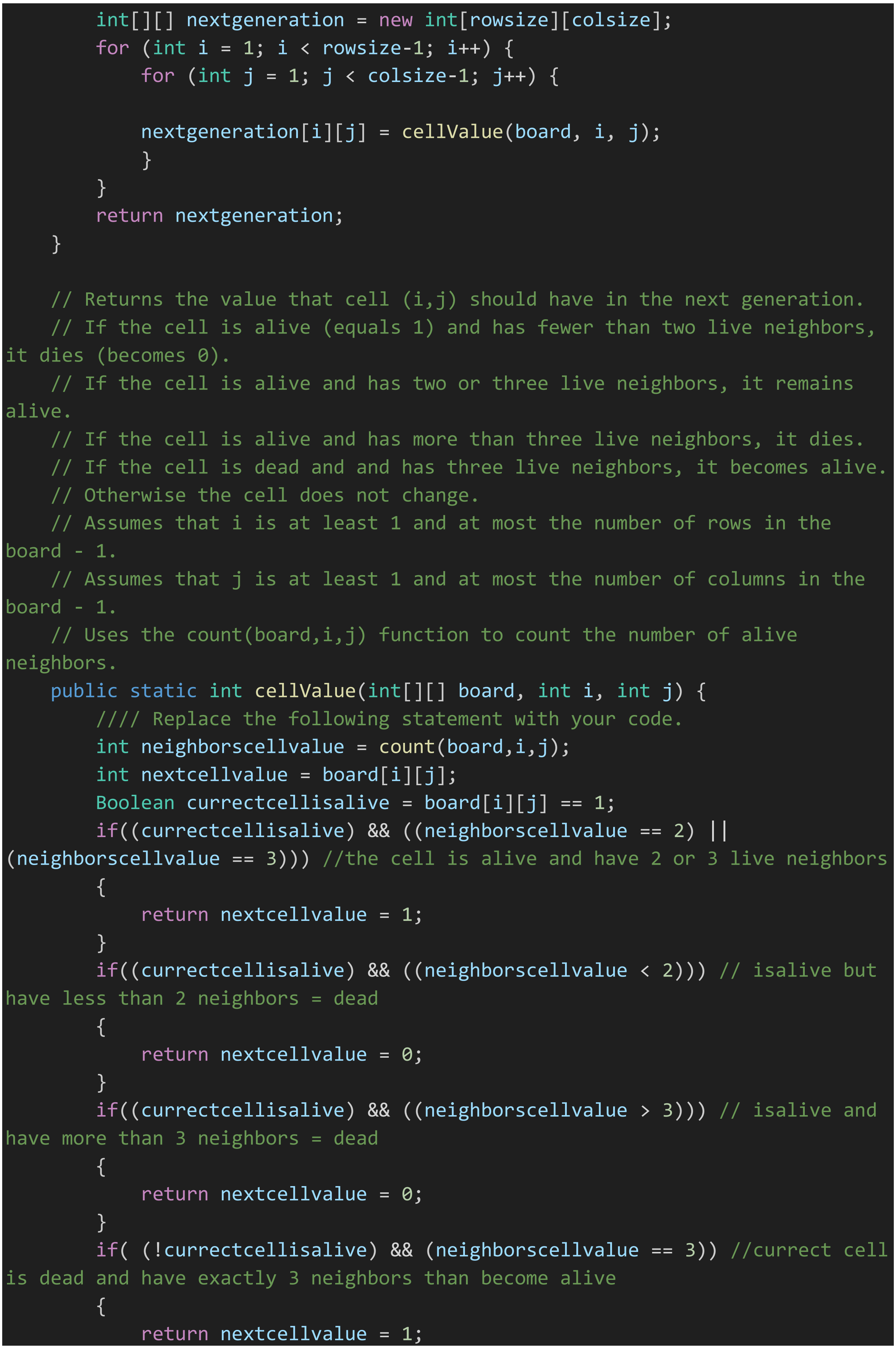
HW5 – Adi Lind

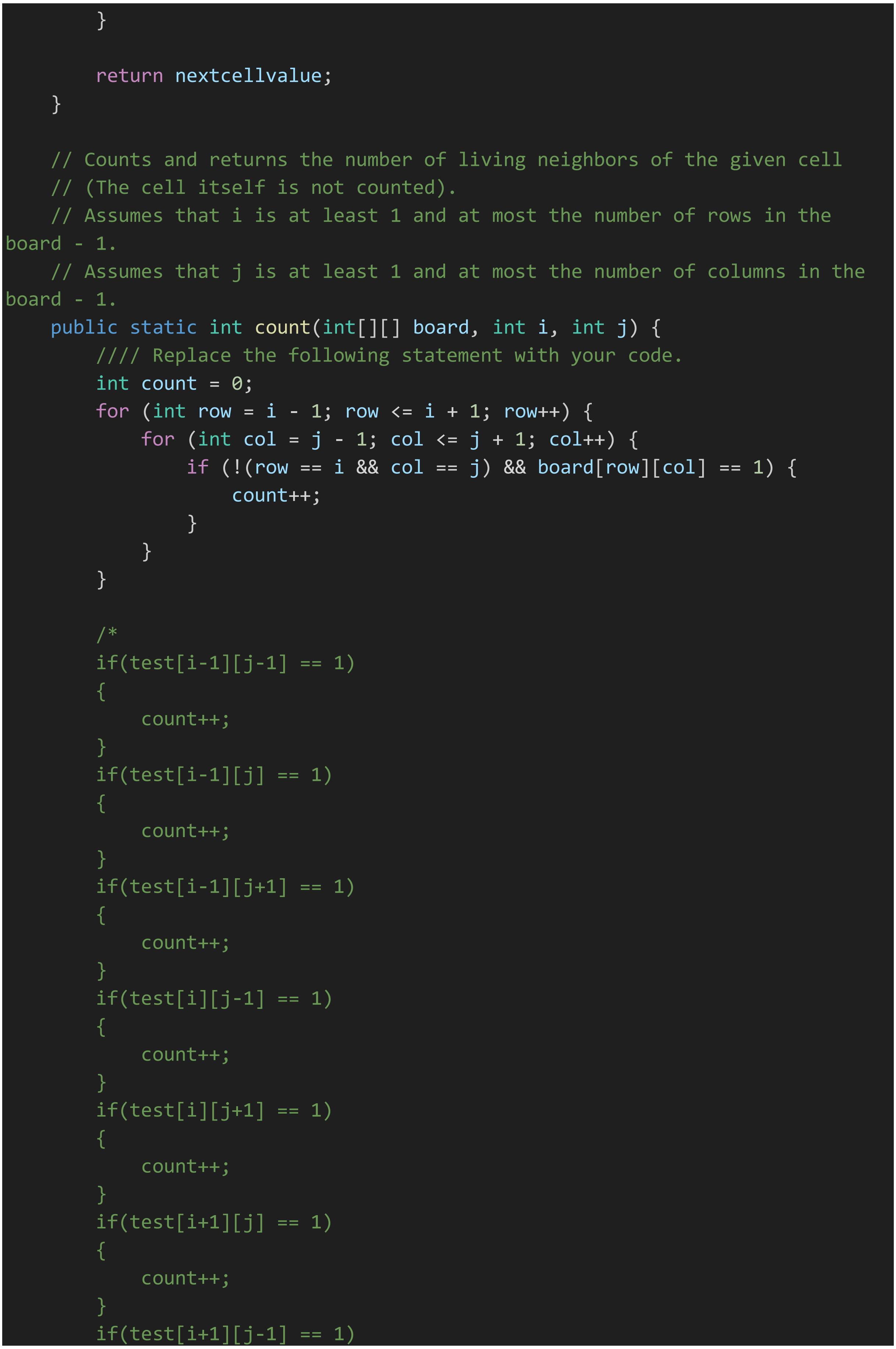
```
/**
 *  Game of Life.
 *  Usage: "java GameOfLife fileName"
 *  The file represents the initial board.
 *  The file format is described in the homework document.
 */

public class GameOfLife {

    public static void main(String[] args) {
        String fileName = args[0];
        //// Uncomment the test that you want to execute, and re-compile.
        //// (Run one test at a time).
        //// test1(fileName);
        //// test2(fileName);
         test3(fileName, 3);
        //// play(fileName);
    }

    // Reads the data file and prints the initial board.
    private static void test1(String fileName) {
        int[][] board = read(fileName);
        print(board);
    }

    // Reads the data file, and runs a test that checks
    // the count and cellValue functions.
    private static void test2(String fileName) {
        int[][] board = read(fileName);
        //// Write here code that tests that the count and cellValue functions
        //// are working properly, and returning the correct values.
        int rowsize = board.length;
        int colsize = board[1].length;
        //int[][] counttest = new int[rowsize][colsize];
        //int[][] cellvaluetest = new int[rowsize][colsize];
        //print counttest
        System.out.println("count test - ");
        for (int i = 1; i < rowsize-1; i++) {
            for (int j = 1; j < colsize-1; j++) {
            //System.out.print(arr[i][j] + " ");
            System.out.printf(count(board, i, j) + " ");
            }
            System.out.println();
            }
            //print cellvalue
            System.out.println("the cell value - ");
            for (int i = 1; i < rowsize-1; i++) {
                for (int j = 1; j < colsize-1; j++) {
                //System.out.print(arr[i][j] + " ");
                System.out.printf(cellValue(board, i, j) + " ");
                }
                System.out.println();
                }

    }

    // Reads the data file, plays the game for Ngen generations,
    // and prints the board at the beginning of each generation.
    private static void test3(String fileName, int Ngen) {
        int[][] board = read(fileName);
        for (int gen = 0; gen < Ngen; gen++) {
            System.out.println("Generation " + gen + ":");
            print(board);
            board = evolve(board);
        }
    }

    // Reads the data file and plays the game, for ever.
    public static void play(String fileName) {
        int[][] board = read(fileName);
        while (true) {
            show(board);
            board = evolve(board);
        }
    }

    // Reads the initial board configuration from the file whose name is
fileName, uses the data
    // to construct and populate a 2D array that represents the game board,
and returns this array.
    // Live and dead cells are represented by 1 and 0, respectively. The
constructed board has 2 extra
    // rows and 2 extra columns, containing zeros. These are the top and the
bottom row, and the leftmost
    // and the rightmost columns. Thus the actual board is surrounded by a
"frame" of zeros. You can think
    // of this frame as representing the infinite number of dead cells that
exist in every direction.
    // This function assumes that the input file contains valid data, and does
no input testing.
    public static int[][] read(String fileName) {
```



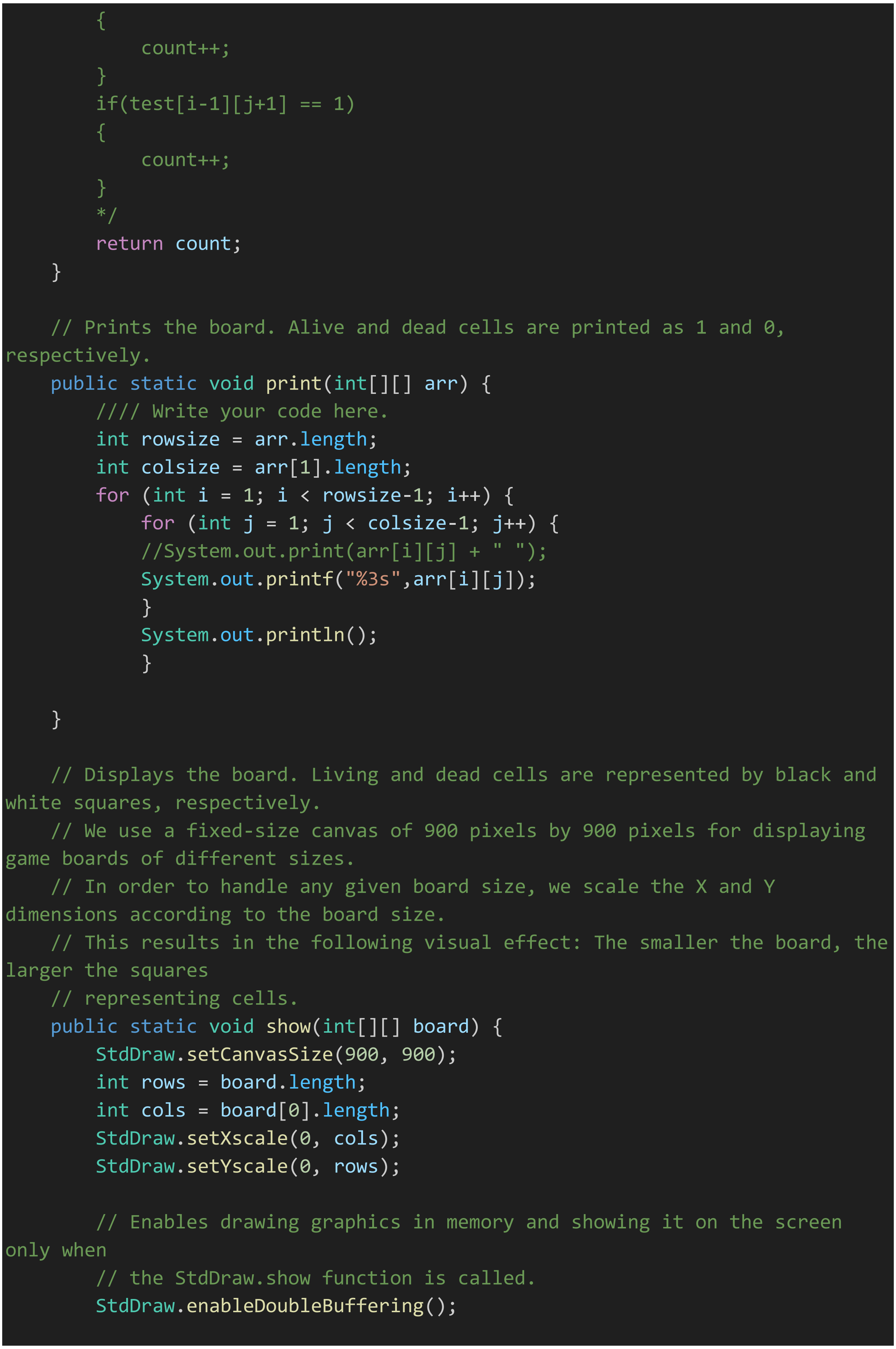
```java
        In in = new In(fileName); // Constructs an In object for reading the input file
        int rows = Integer.parseInt(in.readLine());
        int cols = Integer.parseInt(in.readLine());
        int[][] board = new int[rows + 2][cols + 2];
        //// Replace the following statement with your code.
        for (int i = 1; i <= rows; i++) {
            String line = in.readLine();

            // If the line is empty, set all elements in this row to 0
            if (line.isEmpty()) {
                for (int j = 1; j <= cols; j++) {
                    board[i][j] = 0; //this if is not realy neccesery but for understand better the code
                }
            } else {
                // If the line is not empty, parse each character and set the corresponding array elements
                char[] chars = line.toCharArray();
                int j = 1;
                for (char c : chars) {
                    if (c == '.') {
                        board[i][j] = 0;
                    } else if (c == 'x') {
                        board[i][j] = 1;
                    }
                    j++;

                    // Break the loop if we reach the end of the line
                    if (j > cols) {
                        break;
                    }
                }
            }
        }
    return board;

    }

    // Creates a new board from the given board, using the rules of the game.
    // Uses the cellValue(board,i,j) function to compute the value of each
    // cell in the new board. Returns the new board.


    public static int[][] evolve(int[][] board) {
        //// Replace the following statement with your code.
        int rowsize = board.length;
        int colsize = board[1].length;
```



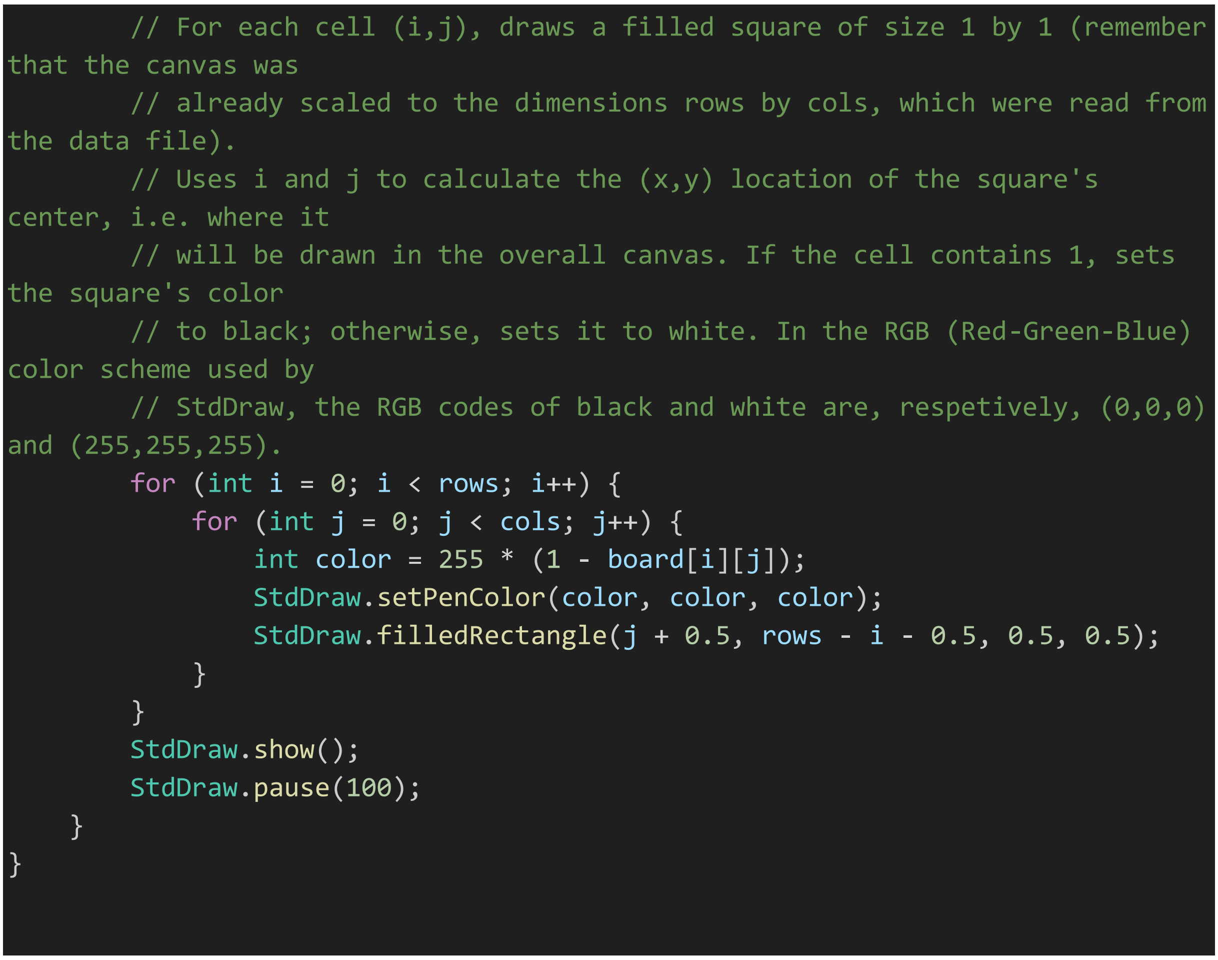
```
        int[][] nextgeneration = new int[rowsize][colsize];
        for (int i = 1; i < rowsize-1; i++) {
            for (int j = 1; j < colsize-1; j++) {

            nextgeneration[i][j] = cellValue(board, i, j);
            }
        }
        return nextgeneration;
    }

    // Returns the value that cell (i,j) should have in the next generation.
    // If the cell is alive (equals 1) and has fewer than two live neighbors,
it dies (becomes 0).
    // If the cell is alive and has two or three live neighbors, it remains
alive.
    // If the cell is alive and has more than three live neighbors, it dies.
    // If the cell is dead and and has three live neighbors, it becomes alive.
    // Otherwise the cell does not change.
    // Assumes that i is at least 1 and at most the number of rows in the
board - 1.
    // Assumes that j is at least 1 and at most the number of columns in the
board - 1.
    // Uses the count(board,i,j) function to count the number of alive
neighbors.
    public static int cellValue(int[][] board, int i, int j) {
        //// Replace the following statement with your code.
        int neighborscellvalue = count(board,i,j);
        int nextcellvalue = board[i][j];
        Boolean currectcellisalive = board[i][j] == 1;
        if((currectcellisalive) && ((neighborscellvalue == 2) ||
(neighborscellvalue == 3))) //the cell is alive and have 2 or 3 live neighbors
        {
            return nextcellvalue = 1;
        }
        if((currectcellisalive) && ((neighborscellvalue < 2))) // isalive but
have less than 2 neighbors = dead
        {
            return nextcellvalue = 0;
        }
        if((currectcellisalive) && ((neighborscellvalue > 3))) // isalive and
have more than 3 neighbors = dead
        {
            return nextcellvalue = 0;
        }
        if( (!currectcellisalive) && (neighborscellvalue == 3)) //currect cell
is dead and have exactly 3 neighbors than become alive
        {
            return nextcellvalue = 1;
```



```
        }

        return nextcellvalue;
    }

    // Counts and returns the number of living neighbors of the given cell
    // (The cell itself is not counted).
    // Assumes that i is at least 1 and at most the number of rows in the
board - 1.
    // Assumes that j is at least 1 and at most the number of columns in the
board - 1.
    public static int count(int[][] board, int i, int j) {
        //// Replace the following statement with your code.
        int count = 0;
        for (int row = i - 1; row <= i + 1; row++) {
            for (int col = j - 1; col <= j + 1; col++) {
                if (!(row == i && col == j) && board[row][col] == 1) {
                    count++;
                }
            }
        }

        /*
        if(test[i-1][j-1] == 1)
        {
            count++;
        }
        if(test[i-1][j] == 1)
        {
            count++;
        }
        if(test[i-1][j+1] == 1)
        {
            count++;
        }
        if(test[i][j-1] == 1)
        {
            count++;
        }
        if(test[i][j+1] == 1)
        {
            count++;
        }
        if(test[i+1][j] == 1)
        {
            count++;
        }
        if(test[i+1][j-1] == 1)
```



```
        {
            count++;
        }
        if(test[i-1][j+1] == 1)
        {
            count++;
        }
        */
        return count;
    }

    // Prints the board. Alive and dead cells are printed as 1 and 0,
respectively.
    public static void print(int[][] arr) {
        //// Write your code here.
        int rowsize = arr.length;
        int colsize = arr[1].length;
        for (int i = 1; i < rowsize-1; i++) {
            for (int j = 1; j < colsize-1; j++) {
            //System.out.print(arr[i][j] + " ");
            System.out.printf("%3s",arr[i][j]);
            }
            System.out.println();
            }

    }

    // Displays the board. Living and dead cells are represented by black and
white squares, respectively.
    // We use a fixed-size canvas of 900 pixels by 900 pixels for displaying
game boards of different sizes.
    // In order to handle any given board size, we scale the X and Y
dimensions according to the board size.
    // This results in the following visual effect: The smaller the board, the
larger the squares
    // representing cells.
    public static void show(int[][] board) {
        StdDraw.setCanvasSize(900, 900);
        int rows = board.length;
        int cols = board[0].length;
        StdDraw.setXscale(0, cols);
        StdDraw.setYscale(0, rows);

        // Enables drawing graphics in memory and showing it on the screen
only when
        // the StdDraw.show function is called.
        StdDraw.enableDoubleBuffering();
```



```
        // For each cell (i,j), draws a filled square of size 1 by 1 (remember
that the canvas was
        // already scaled to the dimensions rows by cols, which were read from
the data file).
        // Uses i and j to calculate the (x,y) location of the square's
center, i.e. where it
        // will be drawn in the overall canvas. If the cell contains 1, sets
the square's color
        // to black; otherwise, sets it to white. In the RGB (Red-Green-Blue)
color scheme used by
        // StdDraw, the RGB codes of black and white are, respetively, (0,0,0)
and (255,255,255).
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                int color = 255 * (1 - board[i][j]);
                StdDraw.setPenColor(color, color, color);
                StdDraw.filledRectangle(j + 0.5, rows - i - 0.5, 0.5, 0.5);
            }
        }
        StdDraw.show();
        StdDraw.pause(100);
    }
}
```