```java
import java.time.Year;

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).
        // read(fileName);
        // 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);
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[i].length; j++) {
                System.out.printf("%3s", cellValue(board, i, j));
            }
            System.out.printf("%n");
        }
    }

    // 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.
    private static void play(String fileName) {
        int[][] board = read(fileName);
        while (true) {
            show(board);
            board = evolve(board);
        }
    }

    // Reads the data from the given fileName, uses the data to construct the
    // initial board,
    // and returns the initial board. Live and dead cells are represented by 1 and
    // 0, respectively.
    private static int[][] read(String fileName) {
        StdIn.setInput(fileName);
        if (StdIn.isEmpty()) {
            System.out.println("The input file is empty. Please enter other file");
            System.exit(0);
        }
        int rows = Integer.parseInt(StdIn.readLine());
        int cols = Integer.parseInt(StdIn.readLine());
        int[][] board = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            String row = StdIn.readLine();
            for (int j = 0; j < row.length(); j++) {
                if (i == 0 || i + 1 == rows) {
                    board[i][j] = 0;
                } else if (j == 0 || j + 1 == cols) {
                    board[i][j] = 0;
                } else if (row.charAt(j) == 'x') {
                    board[i][j] = 1;
                } else {
                    board[i][j] = 0;
                }
            }
        }
        return board;
    }

    // Creates a new board from the given board, using the rules of the game.
    // Returns the new board.
    private static int[][] evolve(int[][] board) {
        int[][] nextStageBoard = new int[board.length][board[0].length];
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[i].length; j++) {
                nextStageBoard[i][j] = cellValue(board, i, j);
            }
        }
        return nextStageBoard;
    }

    // Returns the value that cell (i,j) should have in the next generation.
    private static int cellValue(int[][] board, int i, int j) {
        if (board[i][j] == 1) {
            if (count(board, i, j) < 2) {

                return 0;
            } else if (count(board, i, j) > 3) {
                return 0;
            } else if (count(board, i, j) == 2 || count(board, i, j) == 3) {
                return 1;
            }
        } else if (count(board, i, j) == 3) {
            return 1;
        }
        return 0;
    }

    // Counts and returns the number of living neighbors of the given cell.
    private static int count(int[][] board, int i, int j) {
        int counter = 0;
        for (int row = -1; row <= 1; row++) {
            for (int col = -1; col <= 1; col++) {
                if (row != 0 || col != 0) {
                    if ((i + row) < 0 || (i + row) >= board.length) {
                        break;
                    } else if ((j + col) < 0 || (j + col) >= board[i].length) {
                        break;
                    } else if (board[i + row][j + col] == 1) {
                        counter++;
                    }
                }
            }
        }
        return counter;
    }

    // Prints the board. Alive and dead cells are printed as 1 and 0, respectively.
    private static void print(int[][] arr) {
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[i].length; j++) {
                System.out.printf("%3s", arr[i][j]);
            }
            System.out.printf("%n");
        }
    }

    // Displays the board. Living and dead cells are represented by black and white
    // squares, respectively.
    private 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);
        StdDraw.show(100); // delay the next display 100 miliseconds
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                int grey = 255 * (1 - board[i][j]);
                StdDraw.setPenColor(grey, grey, grey);
                StdDraw.filledRectangle(j + 0.5, rows - i - 0.5, 0.5, 0.5);
            }
        }
        StdDraw.show();
    }
}
```