

# Assignment 1: Game of Life

## 1 Objectives

1. Use version control in a team (Subversion)
2. Review Java
3. Use 2-dimensional arrays
4. Write documentation (user's manual and technical specification)
5. Design and execute a testplan (see cs203 handout)
6. Revise your work
7. Comment your code

## 2 Eclipse

You will be using Eclipse for the project. When you need to create text files (not Java classes), you can do so by selecting **File** → **New** → **File**, clicking on the destination folder, and entering a filename. CRC cards, test plans, and test runs will be best created this way.

You will be creating technical specifications for your project. One way to present them to others is using a tool called Javadoc. This produced the webpages seen in the Java API. To use Javadoc, you will need to use special comments which begin with `/**` and end with `*/`. There is a reference guide on Javadoc linked from the course website. There is also a quick example of Javadoc. I suggest you look at the quick example first and then the reference guide. To create the Javadoc webpages, select **Project** → **Generate Javadoc** and hit Finish. If the finish button is unavailable, make sure the command up top is `/usr/bin/javadoc`. For this project, try Javadoc and use a text file only if it fails. In the future, you will use Javadoc.

## 3 Subversion

You will need to create a new Eclipse Project. Using the steps in the Subversion lab, link it to the Subversion directory for your project,

```
http://svn.eg.bucknell.edu/csci204/f08/p1/yourteamname
```

where *yourteamname* is replaced by the team name you gave me. You and your partners will need to both commit and update to share the project files.

You can see the project and team directories on the web by looking at <http://svn.eg.bucknell.edu/csci204> and p1. You will only be able to access your own files.

## 4 Assignment

In this assignment, you will write a program to model the Game of Life. You will specify a set of initial conditions for the game. The program then will carry out the game from one generation to the next, following the game's rules. Provide a display of the generations as they are produced.

Your program should ask the user to specify the following initial conditions of the game. At this point, your program doesn't have to verify the validity of the input. You can assume the user input is reasonable.

1. How big the board should be. The board should be a square so only one number needs to be specified. We ask you to test for a board size of 15 by 15. But the user could specify any positive integer in a reasonable range (e.g. 5 to 30). **You must use a 2D array for your board.** Why? You will need arrays and 2D arrays all throughout this term and this proves you know how to use them.
2. The probability of a life square in initial setting. This should be a double value ranging from 0 and 1, where 0 means no life at all (thus no game), 1 means every square is a life (thus no game). When testing your program, try for a couple of different values to make the game reasonably manageable. (0 and 1 are not sufficient on their own.)
3. The number of generations to run the game. Ask the user to enter the limit.

Before implementing the program, you are required to write a technical specification for the software including

1. A problem statement describing the assignment in your own words;
2. A description of what the program needs to do;
3. A description in the form of CRC cards of major objects with name, responsibilities, knowledge, and collaborator(s); Write your CRC cards in a file in Eclipse so you can add it to the Subversion repository.
4. For each of the objects identified above, convert the CRC description into a list of attributes (data members) and actions (methods);
5. A UML diagram for each of the objects. For now, draw your UML by hand. In the future, you will learn to use a software tool to draw it.
6. A list of major algorithms (and the algorithm, not just its name)
  - (a) how to produce successive generations of life (see game rules below);
  - (b) how to determine if a cell will be live or dead in the next generation;
  - (c) how to determine if the condition is met for the game to end.
7. A list of cases that need to be tested to prove your software functions correctly. See the CSCI 203 testplan handout for help.

You will then implement and test the program from your design. When testing, save the test results. Your methods, member data, and classes must all have appropriate comments.

After the implementation, write a report that contains two parts, one documents all the test cases and results, the other is a user manual which describes how the program should be used and what are the valid inputs from a user. Unless you find a nicer way to accomplish these, they can be text files. Add both of these to your Subversion repository.

#### **4.1 The Rules for the *Game of Life***

The game of life is a model of growth. The game is played out on a 2-dimensional board. At the initial setting, each square will randomly decide whether or not it has a live cell based on the probability of life entered by the user. The game execution then consists of a sequence of generations, each of which is a world derived from the previous one. The presence of life in a square in the next generation depends on its immediate environment in the current generation. This environment consists of the square's neighbors. A life square will survive the next generation if and only if two or three of its at most eight neighbors contained life in this generation. A dead square will become life if exact three of its neighbors are life in this generation. The squares on the border line or at the corner will have fewer neighbors to check. But the same rule applies.

## 5 Some Helping Information

Here we provides some programming examples that may be useful to your program. These code segments are available on the course web site next to the description of the assignment.

- `livingNeighbors.java`: a method counting how many *live* neighbors a cell has.
- `TestRandomArray.java`: an example how to generate a two-dimensional array with random values in it.

## 6 Testing

You must develop and execute a testplan which shows that the game is updated correctly with each generation. Think of all the situations that a cell can be in and what the cell will look like in the next generation.

Carefully explain all testplans and show the actual results. See the CSCI 203 testplan guide for help.

## 7 Important Deadlines

### 7.1 Phase 0 - The day it is assigned

**You must work in a team of two.** Why? One of the goals for this course is to introduce you to teamwork.

If there are an odd number of students in the class, I will allow one team of three people. The person left without a partner gets first dibs on being in that team of three. Make sure your name and your partners name(s) are included (typed) with all submitted documents. Give me one clearly written piece of paper with the usernames of all people in your team and your team name.

### 7.2 Phase 1

The typed technical specification including CRC cards and UML diagrams are due. Hand drawn UML diagrams should be handed in. The rest must be committed to Eclipse.

Technical specifications include all method info ( pre conditions, post conditions, inputs, output, and purpose) and all major data structures (including 2D arrays), and any special algorithms (find number of living neighbors). Much of this can be done using Javadoc.

Do not wait for my input before beginning the next phase.

### 7.3 Phase 2

You must have completed and tested at least one of the three major algorithms. I can access your files via Subversion.

Do not wait for my input before beginning the next phase.

### 7.4 Phase 3

Handin

- User manual (via Subversion)
- Technical specification, CRC (via Subversion)
- UML
- Test runs (via Subversion)
- Code (via Subversion)
- Your grading sheet from previous phases

## **7.5 Phase 4**

If you worked in a team, you must individually email me an assessment of your and your partner's contribution to the assignment. The assessment should be a percentage of how the work was split. It should add to 100. In a perfect team, you would both score a 50. If you are not writing me 50, Bob 50, I'd like a sentence or two saying why. If I decide the work split was very unfair, the grades will be adjusted accordingly. Your email must have the subject 204 project: Life. This email must arrive within 1 class day of the phase 3 deadline.

If you send me an email with a subject other than the one seen here, I may choose not to count it (and I may not find it in the many incoming emails until after projects are graded). If you do not email me, I will assume you agree with your partner(s) evaluation of you. If nobody on your team emails me, I will assume everything went fine.