CSCI 204 - Introduction to Computer Science II

Project 1: The Game of Life

Professor Wittie
Spring 2011

Objectives

1. Advanced Java: 2-dimensional arrays, interfaces, objects
2. Software Engineering: Teamwork, Version Control (Subversion)
3. Documentation: Specifications, Testplans, User Manual

Read through this entire document before you begin to work on this project.

Conway's Game of Life

Conway's Game of Life is an excellent example of deterministic cellular automata which model of growth. It simulates a petrie dish full of cells. As each generation passes, cells live or die based on how crowded their location is. In computer science, the Game of Life simulates artificially intelligent robots which interract with their neighbors.

You can read more about the Game of Life in the projects at the end of BJ Chapter 7 or on Wikipedia.

`http://en.wikipedia.org/wiki/Conway's_Game_of_Life`

Our Game of Life is played out on a 2-dimensional board full of squares. At the start, each square will determine whether or not it contains a live cell randomly based on the probability of life or by input from the user.

The game execution then consists of a sequence of generations, each of which is derived from the previous generation. 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 live 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 live if **exactly three** of its neighbors are live in this generation. The squares on the border line or at the corner will have fewer neighbors to check. But the same rule applies.

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. Test plans, and user manuals will be best created this way.

Subversion

To begin, import the GameOfLife project from the Class directory of Subversion. Then do a **Team** → **Disconnect** and **check the box to remove all SVN files** and hit ok. Then do a **Team** → **Share** and link the project to the Subversion directory for your project,

`https://svn.eg.bucknell.edu/csci204/s11/p1/yourteamname`

where `yourteamname` is replaced by the team name you gave me. The other team member will then import the project from Subversion.

As you work on the project, you and your partners will need to commit your changes to Subversion and do updates to get changes that others have made. As you do each commit, put a comment that says what work has been done (a class, a method, commenting, test plan, etc.). Part of your grade will depend on these commit comments so I expect to see all teammates equally contributing to the work.

GUI, Abstract Classes, and Interfaces

A Graphical User Interface (GUI - pronounced *gooey*) is a modern way to interact with software. It is the frame and any buttons or menus you see in software such as Firefox, Eclipse, Word, etc. GUIs also contain the ability to display information in chart or pictorial format.

We have provided a GUI for the Game of Life which controls the initial settings and the progression of generations. This GUI displays a 2-D grid with light squares representing dead cells and dark squares representing live cells. You must use the provided GUI in your project. In order to use the GUI, you must call its methods and provide it with certains methods and arrays. These requirements are specified in a set of classes, abstract classes and interfaces.

- **GameOfLifeGUI.java**: This is the top-level class in the Game of Life implementation. This class implements the actual GUI. You can see its public methods in the provided Javadoc.
- **TwoDPanel.java**: This class implements the portion of the GUI which draws the 2-D grid of cells. You can see its public methods in the provided Javadoc. The GameOfLifeGUI has a TwoDPanel instance field (member data).
- **Game.java**: This interface defines all of the methods that a Game of Life will need. The GameOfLifeGUI has a Game instance field (member data).
- **GameOfLife.java**: This class implements the Game interface. You will write this class and implement all of the methods in the interface.
- **TwoDBoard.java**: This interface defines all of the methods that a two-dimentional game board will need. Your GameOfLife has a TwoDBoard instance field (member data).
- **GameOfLifeBoard.java**: This class implements the TwoDBoard interface. You will write this class. Your GameOfLifeBoard will have a two-dimentional array of Cells instance field (member data).
- **Cell.java**: This interface defines all of the methods that a Cell in a game board will need. Each Cell has a a two-dimentional array of its neighboring Cells.
- **GameOfLifeCell.java**: This class extends the abstract Cell class. You will write this class and implement all of the abstract methods in the parent class.

Beginning the Game

To start the game, you will need to declare and instantiate a GameOfLife with some initial settings. The constructor will take a board size, an initial probability of life, and an initial number of generations to run. These settings can also be controlled from the GUI.

The board is square so only one dimension needs to be specified. We suggest using a size 15 board. The life probability should be a `double` value ranging from 0 and 1, where 0 means no life at all and 1 means every square is alive. We suggest using a probability of 0 and setting the actual probability from the GUI. The number of generations controls how long the Game of Life will run. We suggest using 0 generations and controlling this from the GUI.

After you have a GameOfLife object, you can call the GameOfLifeGUI static *run* method with your GameOfLife as the parameter.

```
public static void main(String[] args) {
    GameOfLife game = new GameOfLife(15, 0, 0);
    GameOfLifeGUI.run(game);
}
```

This will create the GUI and start the game.

In the GUI, set the board size, life probability, and max generations (Try 15, 0.25, and 5 respectively). Then hit reset to setup the board and play to watch it run. You can hit stop at any point to pause it. You could also hit next instead of play to control each generational update.

If you want to try out a specific combination of live cells, you can click on cells to change their status. This allows you to try out some of the glider guns mentioned on Wikipedia. It also allows you to easily test that your implementation is working.

Technical Specification

A technical specification explains all of the functionality of software and how it is assembled. This includes all of the public and private methods as well as any member data and algorithms that are used. This gets written before the software is implemented. You will be creating technical specifications for your project.

You will present this technical specification 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** → **GenerateJavadoc** and hit Finish. If the finish button is unavailable, make sure the command up top is `/usr/bin/javadoc`.

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; put this problem statement at the top of your GameOfLife class in a Javadoc comment.

2. A description of what the program needs to do; put this description in your GameOfLife class under the problem statement.

3. A Javadoc comment explaining the purpose of each other class you write; put this comment at the top of the class.

4. A Javadoc comment on each public method that you write. This comment will stating its purpose, inputs, outputs, and any (high level) algorithms it uses to accomplish the task. When you write these comments, your methods will not contain any code yet. Return dud values as needed so that your code compiles (A dud value would be 0, 0.0, false, or null). You can then generate Javadoc but not yet run your software.

5. A Javadoc comment on all instance fields in your classes. The comment should state what the field is used for. For example, the board size is used to make an NxN board.

6. A Javadoc comment on each private method that you think you will need.. You may add or remove these private methods later on as needed.

7. A comment inside each method listing the methods it will call. For example the changeState method in your GameOfLifeBoard will call the changeState method in your GameOfLifeCell.

Generate the Javadoc and add it to your Subversion repository.

Test plans

A test plan is a set of test cases created before you implement the software that you can run to see if your implementation is correct.

A board size ranging from 5x5 to 30x30 would be reasonable so you will test 5x5, 15x15, and 30x30. An initial life probability can range from 0 to 1 so you will test 0, .5, and 1. You can also consider testing 0.75 and 0.25. The generations could be 0 or more so you will test 0, 5, and 10 generations. You could also test one large number such as 100. Each cell updates dependant on its number of live neighbors. You will need to test the update for both live and dead cells, with all possible numbers of live neighbors. Some cells are on a border and have a different number of possible live neighbors. These border cells need seperate tests to make sure you do not have any boundary errors.

Here is the beginings of a test plan. You will need to complete it.

What is being tested?	**What to do**	**Expected results**
Board size	Set size to 5 in main	Starting 5x5 board in the GUI
	Set size to 15 in main	Starting 15x15 board in the GUI
	Set size to 30 in main	Starting 30x30 board in the GUI
	Set size to 5 in GUI, hit Reset	Board becomes 5x5
	Set size to 15 in GUI, hit Reset	Board becomes 15x15
	Set size to 30 in GUI, hit Reset	Board becomes 30x30
Life probability	Set probability to 0 in main	All cells start out dead
	Set probability to .5 in main	About half of the cells are alive
	Set probability to 1 in main	All cells start out alive
	Set probability to 0 in GUI, hit Reset	All cells become dead
	Set probability to .5 in GUI, hit Reset	
	Set probability to 1 in GUI, hit Reset	
Generation	Set generations to 0 in main	GUI runs 0 generations
	Set generations to 5 in main	
	Set generations to 10 in main	
	Set generations to 0 in GUI, hit Resret & Play	GUI runs 0 generations
	Set generations to 5 in GUI, hit Resret & Play	
	Set generations to 10 in GUI, hit Resret & Play	
Update	Give a dead middle cell 0 live neighbors, hit Next	cell stays dead
	Give a dead middle cell 1 live neighbors, hit Next	cell stays dead
	Give a dead middle cell 8 live neighbors, hit Next	cell becomes alive
	Give a live middle cell 0 live neighbors, hit Next	cell becomes dead
	Give a live middle cell 1 live neighbors, hit Next	cell becomes dead
	Give a live middle cell 8 live neighbors, hit Next	cell stays alive
	Give a left border dead cell 0 live neighbors, hit Next	cell stays dead
	Give a top border live cell 3 live neighbors, hit Next	cell stays alive
	Give a top left corner live cell 2 live neighbors, hit Next	cell stays alive

I've ommitted the expected results from some of the tests for you to fill in. I've also left out many of the Update tests for you to do. You need to complete the chart so it tests all cases that can occur. When you test your software, you will go down the list of test cases and try each one. Add your test plan to your subversion repository. It can be a text file.

User Manual

Your user manual describes how the program should be used and what are the valid inputs from a user. They tell the user what Operating Systems it runs on, how to launch and quit the software, and what buttons to hit to run it. Assume your user can be told to run Eclipse but does not know Java so they are just clicking on the buttons you tell them to hit.

Unless you find a nicer way to accomplish this, it can be a text file. Add this to your Subversion repository.

Important Deadlines

The day it is assigned

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

Phase 1

Design: The technical specification including Javadoc descriptions of how you plan to implement the classes, and your test plans.

Implementation: You must have completed at least half of the public methods. This means you are about half done. Do not wait for my input before continuing on. I will leave a comments file in your project when I grade it.

Phase 2

Handin using Subversion

- User manual
- Software

Team Assessment

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 2 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.