CS2112—Fall 2015
Assignment 6
Graphical User Interface Design

Due: Thursday, November 12, 11:59PM
Design Overview due: Wednesday, November 4, 11:59PM

In this assignment you will use the JavaFX API to build a graphical visualization of the critter world described in the project specification. This visualization will have a graphical user interface that permits you to advance time in the world simulation and also to more closely inspect one critter at a time.

The majority of the grades for this assignment will be for new functionality. However, in addition to implementing new functionality, you are expected to fix bugs in your code as necessary to ensure that the functionality implemented for Assignment 5 works correctly.

0 Changes

- 11/5: Java 8 is ok.
- 11/3: Fixed link to the A7 API.

1 Instructions

1.1 Grading

Solutions will be graded on design, correctness, and style. A good design makes the implementation easy to understand and maximizes code sharing. A correct program compiles without errors or warnings, and behaves according the requirements given here. A program with good style is clear, concise, and easy to read.

A few suggestions regarding good style may be helpful. You should use brief but mnemonic variable names and proper indentation. Keep your code within an 80-character width. Methods should be accompanied by Javadoc-compliant specifications, and class invariants should be documented. Other comments may be included to explain nonobvious implementation details.

We will evaluate your user interface on multiple grounds. The visual appearance and layout will be factors, as will the design of the controls. We are looking for an attractive and functional user interface, but we have specified neither its precise appearance and layout, nor exactly how the UI will allow the user to control the system. This is deliberate; we would like you to think through the design. A good idea is to storyboard and also to experiment with more than one UI design. See what works best. Do not get locked into design decisions too early in the process.

1.2 Final project

This assignment is the third part of the final project for the course. Read the Project Specification to find out more about the final project and the language you will be working with in this assignment.

1.3 Partners

You will work in a group of two students for this assignment. This should be the same group as in the last assignment.

Remember that the course staff is happy to help with problems you run into. Read all Piazza posts and ask questions that have not been addressed, attend office hours, or set up meetings with any course staff member for help.

1.4 Restrictions

Use of any standard Java libraries from the Java 8 SDK is permitted. If there is a third-party library you would like to use, please post to Piazza to receive a confirmation before using it. You may code the UI in JavaFX's XML, use a GUI builder, or hand-code your GUI. There is no restriction on design tools.

2 Design overview document

We require that you submit an early draft of your design overview document before the assignment due date. The Overview Document Specification outlines our expectations. Your design and testing strategy might not be complete at that point, but we would like to see your progress. This is also a good time to submit design sketches for the UI. Feedback on this draft will be given as soon as possible.

3 Version control

As in the last assignment, you must submit file `log.txt` that lists your commit history from your group.

Additionally, you must submit a file showing differences for changes you have made to files you submitted in Assignment 5. Version control systems already provide this functionality.

4 Requirements

Your program should be able to display the current state of the world. To initialize, the simulation should load the world files as done in Assignment 5. The critters will be controlled by programs using the simulation engine you have already built.

The user interface must allow the user to step the world state one time step at a time, or to start the world running continuously. In either case, the graphical display will be updated as the simulation progresses to show the new state of the world.

The total number of time steps taken must be displayed on the user interface, along with the total number of critters alive in the world. As in Assignment 5, the user should be able to create a new random world, load a world, or load a specified number of critters.

When loading a critter program file, the user should be able to either specify a number of critters to be randomly placed throughout the world or to select a particular hex in which to place a critter.

The user should be able to set the maximum rate at which the simulation advances (including 0). Regardless of how quickly the simulation is progressing, the graphical display should not be updated more often than 30 times per second. Thus, if the simulation is progressing very rapidly, the world state may not be displayed for some time steps.

Another part of the user interface will allow the user to inspect a single critter somewhere in the world. The user can click on the hex containing a critter to make it the currently displayed critter. The user interface will indicate which critter is currently displayed and will also display the state of the selected critter, corresponding to the 8 initial memory locations, along with the critter program and information about the most recently executed rule. As the simulation progresses, this information will be updated accordingly.

Remember, this UI will have to serve you for A7 as well, we plan to require some of the following features from your remote server, so your world will need to (soon but not now) support the listed operations. Please keep these in mind when fixing your world and building your UI so you will not need to make as many changes when supporting a server-client connection.

For more details about how the simulation of critters and other parts of the world work, consult the Project Specification.

5 Bootstrapping the graphical user interface

Your program must support the following command-line interface:

- `java -jar <your_jar>`
  Start the simulation with a world populated by randomly placed rocks. The program should automatically read the input file `constants.txt` and set the value of the various simulation parameters accordingly.

Note that there are no command line options. All user interaction should be done through the graphical user interface.

6 MVC reiterated

As detailed in class, this GUI should follow some MVC specification. Most importantly you will find it very helpful to keep your world and GUI as programmatically separate as possible. This will also make your reorganization and addition of code much easier for the final project, as the Model and View will be physically separated by a network connection. Also remember that your view will be required to operate given only the commands listed in the api.

7 Common UI Design Choices

UI design can be difficult, and rarely do rules *not* have exceptions. There are, however, a few strategies you can follow.

7.1 Buttons and Control

Users should not need to play a guessing game or to experiment in order to find out what buttons do. They should be clearly and succinctly labeled in order to best describe their function, maybe an icon could prove more useful than words in some cases?

Buttons should also be "easy to click". The meaning of this is intentionally vague, however it's important to consider at least two criteria: distance and size. A small button way off on the side of the screen is difficult to access compared to a large button near the focus of the window, and can be annoying to the user if they need to repeatedly use it. Consider the case where a user needs to click a small button off on the side of a window in order to insert a newline character in a typesetting program, or a large collection of small buttons all close to to one another (like a control panel) thus making it very easy to click the wrong button.

Common solutions to these problems are seen often as part of the UI we use every day. Consider the Windows red X in the top right of windows. When the window is placed on the upper right hand side of the screen the effective size of the button is large since no matter how far to the upper-right you move your cursor, it remains in the clickable area of the button. Since the button is effectively large, its distance does not matter (and, in fact, is beneficial so you don't accidentally close your workspace).

Having discussed infinite effective button size, but how about 0 effective button distance? You actually use 0 distance buttons every day, these range from key-shortcuts (where you don't need to move the cursor to initiate an action) to right-click menus (where the buttons appear wherever the cursor happens to be).

Your GUI should make use of these concepts to provide an intuitive and manageable interface to the user.

7.2 Color Schemes

As a general rule, avoid highly saturated colors and stick to monochromatic, adjacent, triad, or tetrade schemes (or just use this site: paletton.com) We would like to see some original, but usable, designs.

8 Written problems

Consider the following immutable data abstraction:

```
/** A TVBool is an immutable three-valued model of boolean algebra.
 *  A value is either true, false, or unknown, where unknown represents
 *  either true or false.
 *  Note: unknown ∧ unknown = unknown ∨ unknown = unknown
 *      true ∧ x = x = false ∨ x
 *      false ∧ x = false
 *      true ∨ x = true
 */
public class TVBool {
    /* Represents: true if positive, false if negative, unknown if zero. */
    int state;

    /** Creates: the unknown value */
    public TVBool() { }
    /** Creates: an object representing the same boolean as b. */
    TVBool(boolean b) {
        state = b ? 1 : -1;
    }

    /** Effects: none */
    public void normalize() {
        // Ensures state is one of -1, 0, or 1.
        if (state > 1) state = 1;
        else if (state < -1) state = -1;
    }
    /** Returns: conjunction of this and that (this ∧ that). */
    public TVBool and(TVBool that) {
        TVBool result = new TVBool();
        result.state = Math.min(this.state, that.state);
        return result;
    }
    /** Returns: disjunction of this and that (this ∨ that). */
    public TVBool or(TVBool that) { ... }
    /** Returns: logical negation of this. */
    public TVBool not() {
        TVBool result = new TVBool();
        result.state = -state;
        return result;
    }

    private String[] names = {"false", "unknown", "true" };
    @Override
    public String toString() {
        normalize();
        return names[state+1];
    }
    @Override
    public int hashCode() {
        return state;
    }
```

```
51      @Override
52      public boolean equals(Object o) { ... }
53  }
```

1. Give a concise implementation of `or()`.

2. The overview says that the data abstraction is immutable, so it is perhaps surprising that the `normalize()` and `not()` methods assigns to the instance variable `state`. Explain in 3–4 sentences why these two methods are in fact implemented correctly despite these changes to the representation of the object.

3. The `hashCode()` method is incorrect. Explain what can go wrong with this definition of the method, and provide a correct implementation of `hashCode()` and `equals()`.

9 Overview of tasks

Determine with your partner how to break up the work involved in this assignment. Here is a list of the major tasks involved:

- Implementing new components to display the state of the critter world. This will involves graphically rendering hexes and critters. It should be possible, at least, to distinguish critters of different species (as defined by the program), and to see the size and direction of a critter.
- Connecting the simulation engine from Assignment 5 to the display. This means allowing the display both to update as the simulation progresses, and to stop, start, and step the simulation.
- Implementing the loading of critter and world files, including placing critters into an existing world.
- Solving the written problems.

10 KARMA

KARMA questions do not affect your raw score for any assignment. They are given as interesting problems that present a challenge.

10.1 Extensions to the final project

Possible extensions include but are not limited to the following:

- Add a user interface that allows a user to control the selected critter and decide what moves it performs on each turn. A good way to start this is by storyboarding the user interface with sketches that show how the different components will be placed on the screen. Remember, not all components must be visible at all times.
  KARMA:

11 Tips

Take care not to entangle your model with this new interface; proper separation of the simulation and UI is something we will be looking for. The model should not depend on the user interface in any way, because such a dependency will interfere with a distributed implementation of the simulation in Assignment 7. We will also be looking for good documentation of your classes and their methods, using the methodology described in class.

GUI code can become quite long and you will likely have to make a more conscious effort to keep your code clean and readable than in previous assignments. In addition to organizing your classes in packages, think about how you will organize your external resources (constants file and any other files you add). Make sure your resources load properly if the project's location changes.

12 Submission

You should submit these items on CMS:

- `overview.txt/pdf`: Your final design overview for the assignment. It should also include descriptions of any extensions you implemented and of any **HARMA** problems you attempted. Additionally, you should document the different aspects of your GUI. Do not assume that all observable features of your GUI are observable to a new user. You should also indicate the operating system and the version of Java you use.
- A `zip` file containing these items:
  - *Source code*: You should include all source code required to compile and run the project. All source code should reside in the `src` directory with an appropriate package structure.
  - *Other Files*: You should include all other files needed for your project. For example, you might use image files or other data files to control appearance. Be sure to include these.

  Do not include any files ending in `.class`. Git users: to save space, exclude the hidden `.git` folder when zipping.
- `screenshots.pdf`: A PDF file containing 3–4 pages of screenshots showcasing the GUI.
- `log.txt`: A dump of your commit log from the version control system of your choice.
- `a6.diff`: A text file showing diff of changes to files that were submitted in the last assignment, obtained from the version control system.
- `written_problems.txt/pdf`: This file should include your response to the written problems.