6.831 User Interface Design & Implementation
Fall 2004

PS2: Output Models

This assignment explores the following topics related to GUI output:

- the component model (also called retained-object model);
- the stroke model (also called vector graphics);
- the pixel model (also called raster graphics);
- handling events sent from a model to a view.

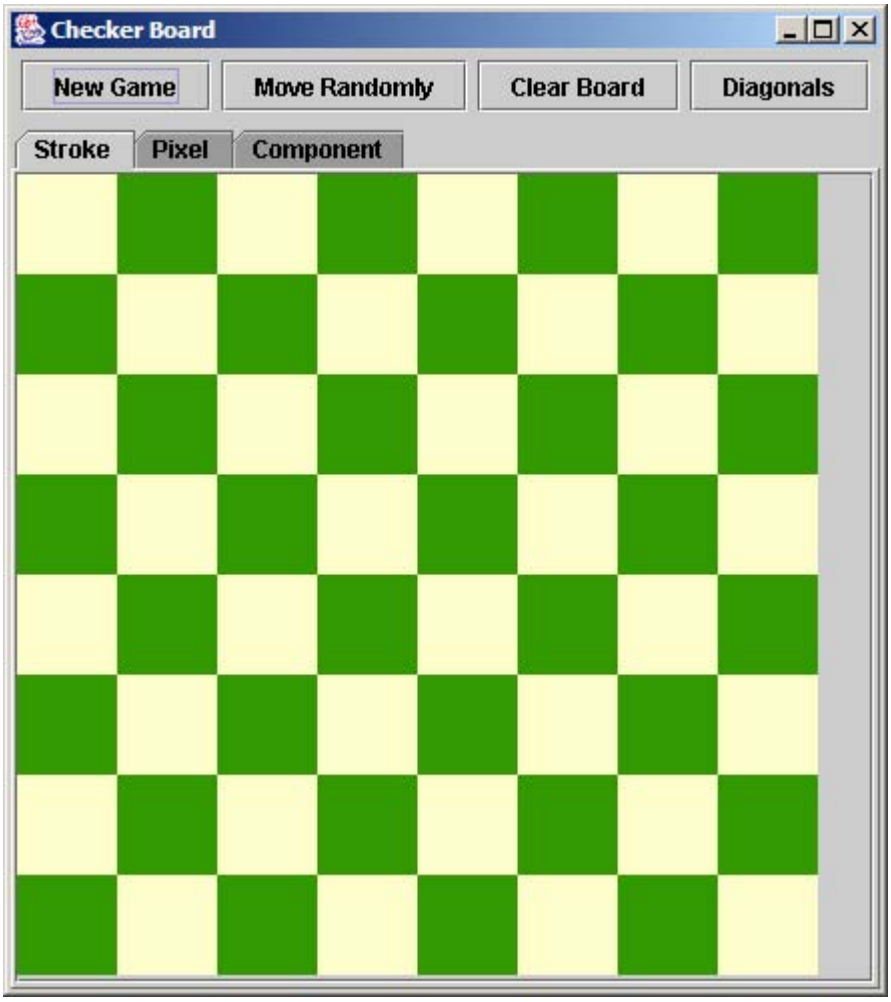
In this problem set, you will implement a view that displays the pieces of a checkers game. You'll implement the view three different ways, using component, stroke, and pixel techniques. In this problem set, you'll only be concerned with output. In the next problem set, you'll add input handling to your views, so that the user can move the mouse over checkers and drag them around.

Provided Resources

We provide you with a lot of existing code for this assignment. You can get it all at once here:

- ps2-provided.jar : a jar file containing source code, class files, and Javadoc documentation.

- You can run ps2-provided.jar (`java -jar ps2-provided.jar`) to see the BoardViewTester window pop up, displaying a checkerboard:



The board model actually has pieces on it, but you won't see them until you've implemented the pieces display.

You can unpack ps2-provided.jar (`jar xf ps2-provided.jar`) to find the source code and Javadoc documentation.

Feel free to change these classes as you see fit, or even replace them entirely.

Problem 1: Stroke Model

Fill in the skeleton of StrokeBoardView so that it displays all the checkers on the board using strokes drawn on Graphics. At a minimum, red checkers should be red circles and black checkers should be black circles. Feel free to make your checkers look better, if you like, but you may want to defer this until you've done all the problems.

Your StrokeBoardView must update correctly when the board changes. You can test this in BoardViewTester by clicking on the buttons, which produce various board configurations.

If you need help understanding the Swing painting model, you may want to look at Sun's article Painting in AWT and Swing. Particularly useful is the summary of Swing painting guidelines at the end of the article.

Problem 2: Pixel Model

Fill in the skeleton of PixelBoardView so that it displays checkers using pixel images. Two pictures are provided for you in the jar file (red.gif and white.gif, found in the ui directory), but you can replace them with different pictures if you prefer. These image files should be loaded as *resources*, not as files.

Like StrokeBoardView, your PixelBoardView must update correctly when the board changes.

Images in Java can sometimes be a pain, because Java is designed to allow loading images lazily from a network connection. You can avoid these headaches by using javax.swing.ImageIcon to load the image as a Swing icon, and then call getImage() to get an Image object that you can actually draw with.

Problem 3: Component Model

Fill in the skeleton of ComponentBoardView so that each checker is represented by its own JComponent object. The skeleton code includes an inner class CheckerComponent that represents a single checker. You'll have to fill in code to make this component draw itself, which it can do with either strokes or a pixel image. You'll also have to write code in ComponentBoardView for creating and managing CheckerComponent objects.

Like the other views, your ComponentBoardView must update correctly when the board changes. Warning: Swing has a bug (which Sun stubbornly refuses to fix) such that a container with no layout manager fails to repaint when a component is removed from it. When you remove a checker component, you'll have to call repaint manually.

Questions

Answer the following questions in readme.txt.

1. When a checker moves in ComponentBoardView, which checkers are repainted by your implementation? (Don't guess -- instrument your code to find out.) Why?
2. When a checker moves in StrokeBoardView, which checkers are repainted by your implementation? Why? Can you do better, and if so how?
3. Try a 100x100 checkerboard, by passing 100 as a command-line argument to BoardViewTester. How many checkers are requested from the model by each of your views when the view first appears? Why? Can you do better, and if so how?
4. For each of the following situations, which output models would be appropriate and which would not? Why?

a. The checkerboard has a million squares on each side, and its model is split up among many servers around the Internet.
b. The checkerboard is standard 8x8, but might be displayed on a variety of screen sizes: a desktop, a handheld, a watch, or a wall.
c. The game is chess instead of checkers.

What to Hand In

Package your completed assignment as a jar file, as described in PS0. Here's a checklist of things you should confirm before you hand in:

- ☐ all your Java source is included in your jar file (Javadoc documentation isn't necessary)
- ☐ the main class of your jar file is BoardViewTester
- ☐ all necessary third-party libraries are included, either inside your jar or as separate jars referenced by your jar's classpath
- ☐ all images used by your code are included in the jar and referenced as resources
- ☐ readme.txt is included, and it answers the questions above and credits anybody you discussed the assignment with

Before you submit your solution, put all the jar files you plan to submit in an empty directory and make sure you can run it:

```
java -jar yourfile.jar
```