6.831 User Interface Design & Implementation
Fall 2004

PS3: Input Models

This assignment explores the following topics related to GUI input:

- event handling;
- hit detection;
- dragging;
- feedback.

In this problem set, you will implement input handling for the checkerboard you created in PS2, so that the user can pick up checkers with the mouse and drag them around.

Provided Resources

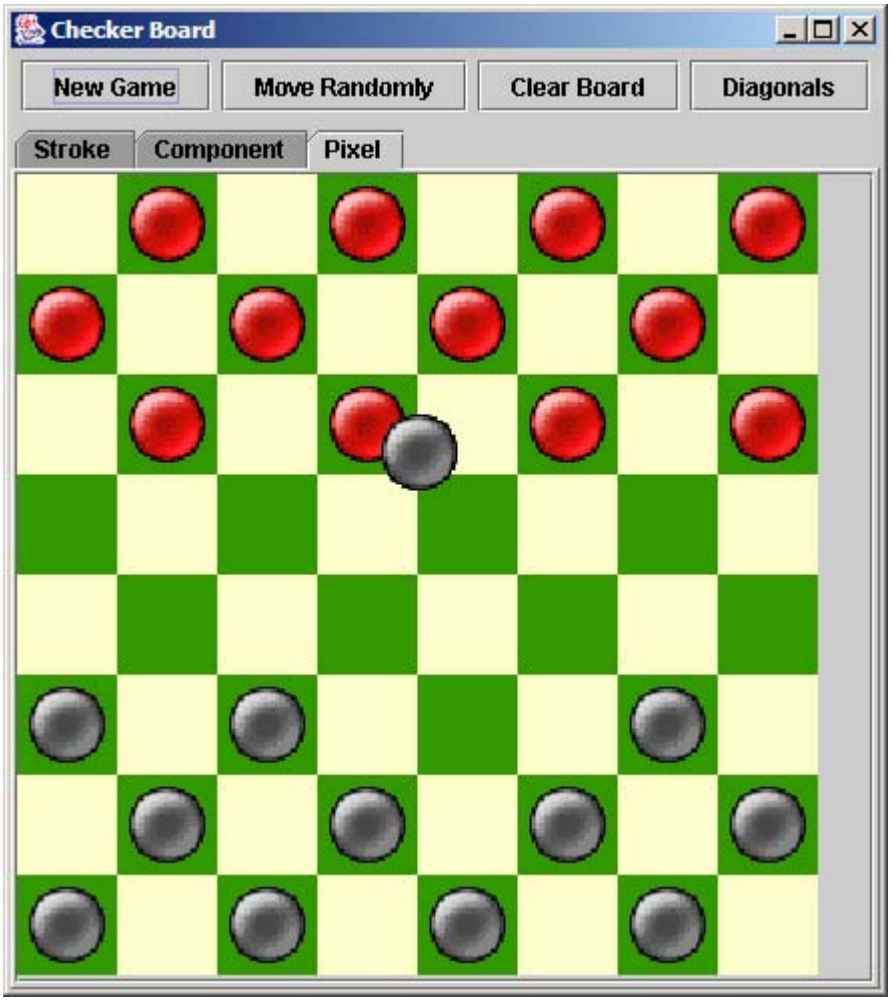
This assignment builds on top of the code you wrote for PS2, plus a new class:

- `MoveInteractor` : skeleton for a generic dragging interactor. Use this only for Problem 2, not for Problem 1.

Problem 1: Dragging in Stroke and Pixel Models (30%)

Add input handling to your StrokeBoardView and PixelBoardView, so that the user can click and drag checkers around.

Pressing and dragging in a square containing a checker should pick up the checker from the board. The checker should not make any abrupt jumps -- neither when the user presses the mouse button, nor when the user starts moving the mouse. The checker should move smoothly with the mouse pointer, hovering over the other checkers on the board:



If the user releases the mouse button when the mouse pointer is over an empty square, the checker should be moved to that square using Board.move(). The board model should not be changed until then. If the mouse pointer is over a filled square or off the checkerboard entirely, then the dropped checker should be put back where it was taken from.

If the mouse pointer leaves the checkerboard during the drag operation, the dragged checker may either follow it (becoming invisible because of clipping), or it may stop at the edge of the checkerboard --- the behavior is up to you. But if the mouse pointer moves back into the checkerboard, the checker should resume following the mouse.

A word of advice: you may notice that your stroke and pixel model implementations are very similar, differing only in how a single checker is painted. Before you add input to either class, you might want to restructure your code so that one class extends the other, or so that both classes extend a single abstract class, with a method called something like `paintChecker()` that can be overridden with the appropriate behavior for both models. That way you can just add input handling once, to the superclass, and save yourself a lot of redundant implementation.

Problem 2: Dragging in Component Model (30%)

We could add input handling to ComponentBoardView in the same way, but instead we're going to build a solution more in the spirit of the component model: a generic controller.

Create a generic, reusable *move interactor* that is capable of dragging *any* components around inside their container, not just checker components in your ComponentBoardView. We have provided a skeleton interface for MoveInteractor for you to fill in. Your implementation should be generic, not referring to any classes specific to the checkerboard problem.

MoveInteractor should listen to the draggable components themselves (the checkers) for mouse events. It should *not* listen to their container (i.e., the checkerboard view).

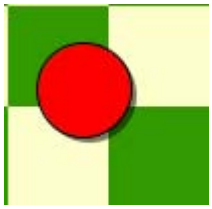
Use MoveInteractor to add input handling to your ComponentBoardView. Checker dragging should behave the same way as specified in Problem 1. Since MoveInteractor is generic, you need to customize it for the checkerboard, which you can do by creating a subclass of MoveInteractor that overrides drop(), in order to decide whether it should allow the checker to be dropped, and if so, change the board model.

Hints:

- Use MouseEvent.getComponent() to figure out which component is being dragged.
- Be aware of coordinate systems. Unlike Problem 1, which you probably solved by adding listeners to the entire checkerboard view, MoveInteractor listens to each individual checker component. That means that the mouse events it receives will not be in the checkerboard's coordinate system, but in the checker's coordinate system. MoveInteractor.pickUp() and MoveInteractor.drop() are also specified to use the checker's coordinate system.
- Be aware of z-order, since you want the dragged component to hover on top. Swing actually draws child components in reverse order, so child 0 is higher in z-order (drawn on top of) child 1. Also, Swing has no simple method for bringing a component to the top. You have to remove the component from its parent container and add it back as child 0.

Problem 3: Drop Shadows in Stroke Model (10%)

Modify your StrokeBoardView so that it provides more feedback about the dragging operation. When a checker is being dragged, a shadow should appear underneath it. This shadow should appear as soon as the mouse button is pressed, and persist until the mouse button is released:



Hint: to draw a nice translucent drop-shadow, use black with 50% alpha, offset by a few pixels vertically and horizontally. The Color class has constructors that take an alpha value in addition to RGB values.

Problem 4: Automatic Scrolling in Component Model (10%)

When the checkerboard is large enough to require scrollbars, it becomes hard to move a checker to a square that's scrolled offscreen. Modify your MoveInteractor so that it automatically scrolls to keep the dragged component entirely in view as the user moves it around.

This is easier than you might think, because Swing has built-in support for it. Look at JComponent.scrollRectToVisible() and JComponent.setAutoscrolls().

Questions (20%)

Answer the following questions in readme.txt.

1. Does Swing coalesce mouse move events when the user drags a checker? (Don't just guess: instrument your code to find out.)
2. What other feedback could you add to checker dragging to make it more usable?
3. How would each of your views behave if a checker is removed from the model while the user is currently dragging it?
4. How could you add drop shadows to the generic MoveInteractor, so that they are displayed automatically for any component that you drag?

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
```