Introduction to Computer Science, Homework 11:
Graphical Sudoku

assigned Dec. 6, due Dec. 14

Overview

In this assignment, you’ll create a graphical version of the sudoku game you wrote for homework 7. That’s really all there is to it, and if you turn in a working game with just that description, I’ll be happy. Everything that follows is intended to help you along the way, but you can ignore it if you want. I’d hoped you’d have two weeks to do this (and nothing else), but since you don’t, I understand if you don’t get though all of the event handlers.

Required steps

1. Copy the Sudoku class you created for homework 7 to one called GraphicalSudoku. When it starts, it should create a JFrame for the game to be displayed in. See TextWindow.java for an example.

2. Use getContentPane to get a container you can add elements to in the JFrame. Add a JLabel for displaying messages to the player. Change GraphicalSudoku so that instead of printing messages for the player, it calls JLabel.setText. After this step, the game board itself will still be output to the terminal, and player input still comes from the terminal.

3. Create a class NumComponent whose constructor takes a GameInterface as an argument. It should extend JComponent, overriding paintComponent to display a nine-by-nine grid of numbers obtained with GameInterface.getMark. Draw grid lines after the third and sixth row and the third and sixth column. See TextWindow.java for an example of how to center text in a component. This step is similar, just in a nine-by-nine grid. Draw changeable cells in a different color (or make them otherwise distinguishable from other cells).

4. Modify GraphicalSudoku so that it creates a NumComponent and adds it to the same container you added the JLabel to. Modify the main loop of GraphicalSudoku so that it calls repaint on the NumComponent after a player makes a move. After this step, all of the output will be graphical, but the input will come from the terminal. If you make it this far, you get full credit.
Optional steps

1. Modify `NumComponent` so that it keeps track of a selected cell in the grid and draws a border around it. This will be used to select a cell the player can make a mark in. Modify `paintComponent` so that it draws a border around the selected cell. Add `getSelectedRow` and `getSelectedColumn` methods so `GraphicalSudoku` can determine which cell is selected.

2. Add a `MouseListener` to `NumComponent` in its constructor. The added code will look something like this:

   ```java
   addMouseListener(new MouseAdapter() {
   @Override public void mousePressed(MouseEvent event) {
   int cellHeight = getSize().height / 9;
   int cellWidth = getSize().width / 9;
   selectedRow = event.getY() / cellHeight;
   selectedColumn = event.getX() / cellWidth;
   repaint();
   }
   });
   ```

3. Modify `GraphicalSudoku` so that instead of prompting the player for a cell to modify, it uses the selected cell.

4. Add a `KeyListener` to `NumComponent` similarly to the way you added a `MouseListener` (probably by extending `KeyAdapter`). Move all the code for handling player moves from `Sudoku` to the `keyTyped` method of the `KeyListener`. So, if cell (4,5) is selected and the player presses “7”, a seven should appear in cell (4,5) if that cell is changeable. Now, all player input will be through the graphical interface. Note that you’ll have to override `isFocusable` in `NumComponent` or else it will silently ignore key presses:

   ```java
   @Override public boolean isFocusable() {
   return true;
   }
   ```

   2