The Javadoc documentation for this project is completed at http://bill.kutztown.edu/~parson/javadoc/. The project consists of two packages, games and games.scrabble, both under directory games/. The games package contains game-neutral interfaces and classes, and the games.scrabble package contains Scrabble plugins and helper classes. Figure 1 shows the class diagram for these interfaces and classes. There is a new interface, TwoDimensionalBoardUI, that represents a user interface for any 2D game. Test driver GameMain can dynamically load and construct both a concrete TwoDimensionalBoardGame and a concrete TwoDimensionalBoardUI object and initiate a game. GameTtyUI is a terminal IO test driver UI component, refactored from our previous assignments.

ScrabbleSwingFrameUI is a Scrabble-specific implementation of TwoDimensionalBoardUI. It encodes direct dependence on class ScrabbleGame. ScrabbleSwingFrameUI derives from JFrame using implementation inheritance. Most of the work of ScrabbleSwingFrameUI does not occur within that class, but instead within the package-protected class ScrabbleSwingRootPaneHelper.

Figure 1: Class diagram for assignment 3.
Student work goes into class ScrabbleSwingRootPaneHelper.
The reason for this division of labor is that it should be possible to code a JApplet-derived counterpart to ScrabbleSwingFrameUI for assignment 5 with little additional work. That counterpart class, presumably called ScrabbleSwingAppletUI, will use ScrabbleSwingRootPaneHelper to perform most of its work as well.

The next page shows a screen shot of my working solution to assignment 4. Students can run this compiled code by performing the following steps on bill.kutztown.edu.

- Run putty in X11 mode.
- Run Exceed on the PC.
- Either “setenv CLASSPATH /export/home/faculty/parson/JavaLang/games.jar” (C shell)
- Or “export CLASSPATH /export/home/faculty/parson/JavaLang/games.jar” (Bash)
- java games.GameMain games.scrabble.ScrabbleGame games.scrabble.ScrabbleSwingFrameUI NUMPLAYERS \[ SEED \]

where NUMPLAYERS is the number of players, 2, 3 or 4, and SEED is an optional seed integer for the random number generator as before. Note after the GAME parameter comes a GameUI parameter, games.scrabble.ScrabbleSwingFrameUI in this case. Driver games.GameMain loads both the game and its UI and then interconnects them via operation TwoDimensionalBoardUI.playGame().

The version of ScrabbleSwingRootPaneHelper in JavaLang/games/scrabble/ is missing the code for the players. If you execute “gmake test” from within JavaLang/games, gmake first performs terminal IO tests that should pass as before. It then runs “gmake testgui” which runs the Swing GUI. Running that GUI shows the board of Figure 2 without any of the surrounding player panels or widgets. Your assignment is to write those panels and widgets, making your GUI look like Figure 2 or better!

File ScrabbleSwingRootPaneHelper.java has a number of comments, starting with the uppercase word STUDENT, that describe the detailed steps of this assignment. These steps occur in three places. First, in the field declarations for class ScrabbleSwingRootPaneHelper is this comment section.

// STUDENT 1: You must allocate array storage and objects stored in
// arrays playerpanels, playertiles, and score in the section marked
// "STUDENT 2" below. Each of these arrays holds the following data.
// Note that you can count on numplayers holding a value of 2, 3 or 4
// when your code runs. You do not have to set it.
//
// playerpanels must have numplayers elements. Each is a JPanel
// outside an edge of the board, starting at the SOUTH as explained
// in STUDENT 2 below.
//
// playertiles must have [numplayers][ScrabbleGame.TILESPERPLAYER] elements.
// Each element is a JButton holding the text of a player tile.
// You must initialize these JButtons to hold the correct text in
// the STUDENT 2 section below.
Figure 2: Screen shot of Swing GUI for Scrabble
As noted, you do not make any changes to these field declarations nor add any fields.

The next set of STUDENT changes are within method buildFrameOrApplet().

// STUDENT 2: The following steps relate to populating the arrays
// described in the STUDENT 1 comment above. Step 7c is another step.
// 5. Allocate arrays playerpanels, playertiles and score.
//    For each player
// 6. Allocate a panel for the player.
// 7. Set the layout manager for that panel.
// 8. That panel gets two sets of widgets, containing
//  8a. ScrabbleGame.TILESPERPLAYER (7) buttons for tiles in the
//       player's tray.
//       Add a new PlayerActionListener object for each button.
//       I have coded class PlayerActionListener below -- you need to
//       construct a new PlayerActionListener object for each button.
// 8b. The score label for the player.
// Place a blank JLabel between these two adjacent sets.
// 8c. Player 0 gets an additional button on the right side of
// the panel. Labeled "scores" it displays the full set of
// scores from invoking game.getScoreSheet(). It displays
// using JOptionPane.showMessageDialog() similar to doCommand()
// below. You need to write the ActionListener class for this
// button.
// 9. Add the panel to the SOUTH,WEST,NORTH,EAST of the enclosing
// RootPaneContainer, in that order (SOUTH for the first player).
// You can create any local variables that are necessary.
// Make sure to populate all arrays specified in STUDENT 1.

// This part comes after STUDENT 2.
updateDisplay();
The final set of changes are in method updateDisplay().

    // STUDENT 3: Apply the following foreground colors using
    // setForeground() to the JButtons on the board, for the
    // value in variable trim:
    //      2l      green
    //      3l      blue
    //      2w      pink
    //      3w      red
    //      others black

}  

}  

// THE FOLLOWING "IF" CAN GO AWAY AFTER STUDENT 2 IS WORKING.
// REMOVE THIS "if" construct (keep the for loop) when you are
// ready to test.
if (playerpanels != null && playertiles != null && score != null) {

The majority of the work occurs in the STUDENT 2 section, where you must construct panels and
low-level widgets, insert them into their containers, and attach them to their action listeners as
described in the STUDENT 2 comments. You can use the code that lays out the board JPanel,
JButtons and their action listener objects as examples. You do not need to write an action listener
class for player JButtons that represent the 7 tiles — simply use class PlayerActionListener that is
already completed with class ScrabbleSwingRootPaneHelper. You should not change this helper
class.

You will need to write your own action listener for the “scores” button as described in STUDENT
2 step 8c.

    // 8c. Player 0 gets an additional button on the right side of
    //      the panel. Labeled "scores" it displays the full set of
    //      scores from invoking game.getScoreSheet(). It displays
    //      using JOptionPane.showMessageDialog() similar to doCommand()
    //      below. You need to write the ActionListener class for this
    //      button.

See the Scores button in the lower right hand corner of Figure 2.

Testing will consist of running “gmake test,” which ensures the integrity of the non-GUI classes
that you should not change, followed by running an interactive game using the GUI. You can
initially run “gmake test” successfully, but your GUI will be missing the parts that you must add.
Changing CLASSPATH to point to my games.jar file as outlined at the start of this document
allows you to try a full, compiled GUI (no source) with no missing pieces.