Implementation (Approximately 3/4 of the total points)

Best practice violations (Approximately 1/2 of the total points)

Your reimplementation should reflect 4 proposed fixes of the violations of best practices (either your own or from the possible answers). If the proposed fix has already been implemented, simply add an internal
comment in the appropriate source code to document that.

1. One of my proposed fix was the lack of MVC architecture which is now implemented(said it here since there's no specific source code I can write this comment in)
2. Modularity(Modularity was applied throughout the whole project, one internal comment isn't enough to indicate so)
3. Type Safety: enumerated game status("Player 1 wins!", "Player 2 wins!", etc…), whose player it is("1" or "2"), and the Playerpiece("X" or "O" or "Empty"). Respectively in RowGameStatusView, RowGameModel, and RowBlockModel, also commented to indicate that they were enumerations for type safety
4. Readability and understandability: Made proper indentation, and when modularizing RowGameController, made proper java documentation(proper java documentation on some methods on other java files as well)

You should also generate the javadoc (contained in the jdoc folder) and commit it.

Design patterns (Approximately 1/4 of the total points)

Composite design pattern For the MVC architecture pattern, The RowGameGUI class represents the View. For the Composite design pattern, here is the proposed Component class.

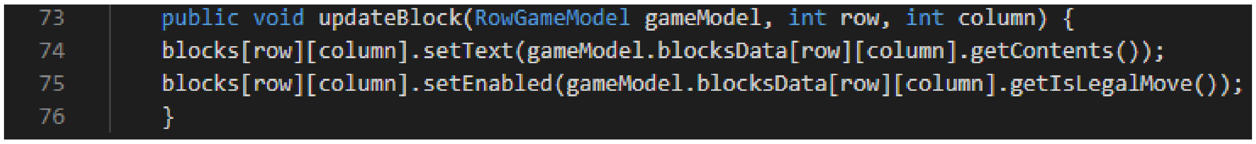
```
package view;

import model.RowGameModel;

public interface RowGameView
{
public void update(RowGameModel gameModel);
}
```

You should first briefly describe how to decompose the RowGameGUI class into the following two classes:

1. RowGameBoardView (i.e. Component A)
RowGameBoardView would have fields: 2d array representing the board
Update Method would update what's on the board
2. RowGameStatusView (i.e. Component C)
RowGameStatusView would have fields: a boolean that stores whose turn it is, a string that stores either "X" or "O".
Update method would either change whose turn it is or who is winning

For each class, which fields? What does the update method do?
You should then briefly describe how to modify the RowGameGUI class to be a Composite class that uses the above two classes.
Have the RowGameUI contain the above two classes, and have the update method of the RowGameUI call the update method of the two classes.

Observer design pattern For the original application, the Observer design pattern is being applied for the relationship between a Java Swing View and its Controller (in particular for Component A and Component B).
• Identify one field in the original application that corresponds to an Observable.
RowGameModel
• For that Observable, identify the Java Swing class that corresponds to its Observer.
JPanel
• For that Observer, identify the implementation of the update method.
In RowGameGui.java



```
public void updateBlock(RowGameModel gameModel, int row, int column) {
blocks[row][column].setText(gameModel.blocksData[row][column].getContents());
blocks[row][column].setEnabled(gameModel.blocksData[row][column].getIsLegalMove());
}
```