

# COMP1406 - Assignment #6

(Due: Monday, March 3rd @ 11:30pm)

















In this assignment you will get the "Shiny Buttons" game working (from assignment 5).

## (1) Working On the Game:

Begin with your working code from assignment #5. Complete the game so that the following happens:

1. Add a **private score** integer attribute to the **ShinyButtons** model class. Create a **getScore()** method that returns the score.
2. When the user presses the **New Game** button, the score resets to zero and the game pieces are reset to a different set of random colors again.
3. When the user presses the **Quit** button, the application stops and the window closes.
4. When there is currently no *selected* game piece and then the user clicks on a game piece, that game piece's JButton becomes the "**selectedButton**". Each **JButton** can have two ImageIcon assigned to it: (1) the regular icon, and (2) the selected icon:

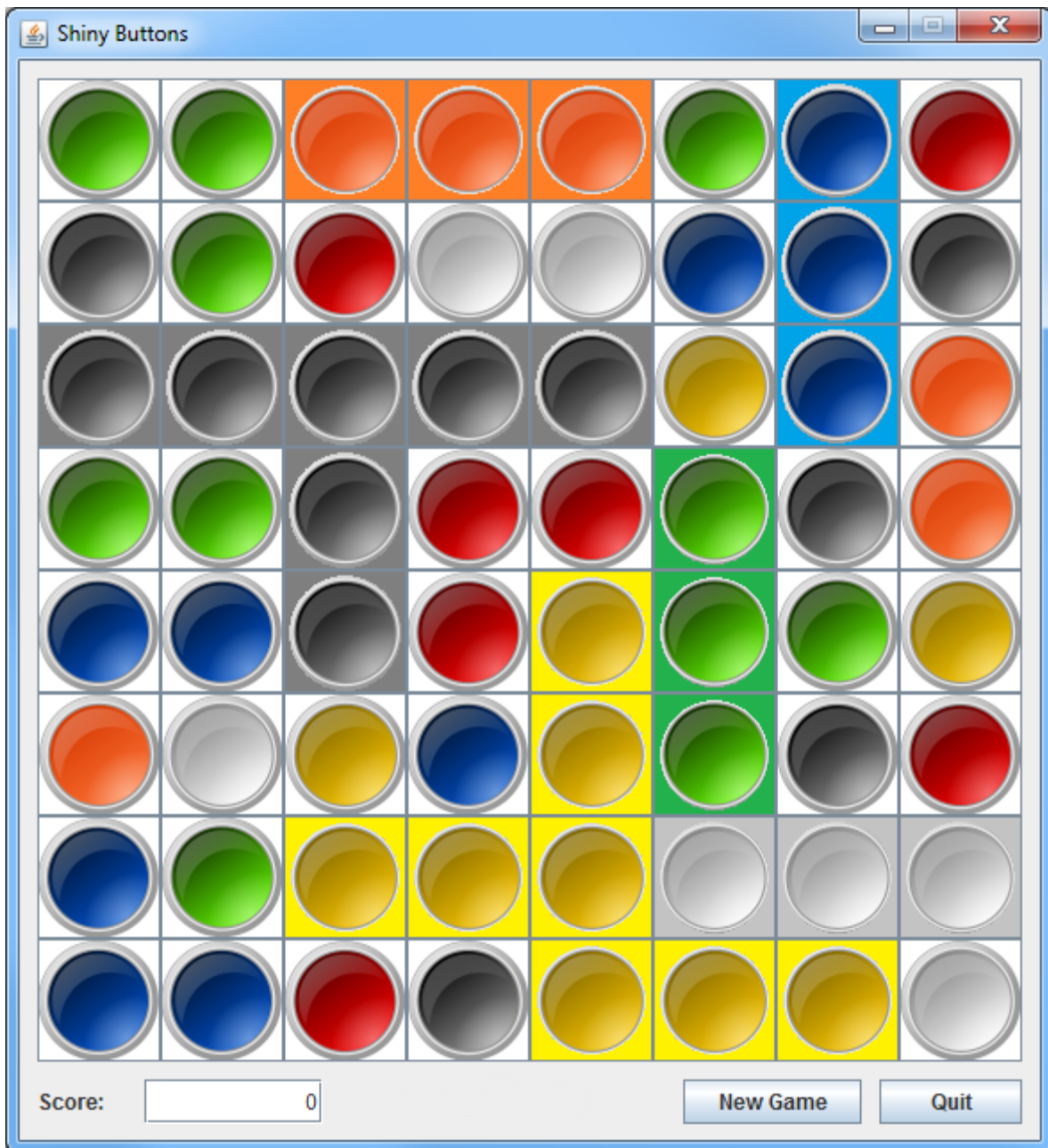
Regular Icons							
Selected Icons							

At this point, we have only used regular icons. You can also set a selected Icon for each JButton (i.e., a different image) by doing this: `aJButton.setSelectedIcon(new ImageIcon("..."))`; This just has to be set once, when you create the JButtons. Add the following array to your code and make sure that you download the additional 7 selected icon images (shown above). Use the array to set the appropriate *selected* icon for each JButton according to its current color.

```
public static ImageIcon[] selectedIcons = {  
    new ImageIcon("SelectedRedButton.png"),  
    new ImageIcon("SelectedOrangeButton.png"),  
    new ImageIcon("SelectedYellowButton.png"),  
    new ImageIcon("SelectedGreenButton.png"),  
    new ImageIcon("SelectedBlueButton.png"),  
    new ImageIcon("SelectedLightGrayButton.png"),  
    new ImageIcon("SelectedDarkGrayButton.png")  
};
```

Adjust your code so that all JButtons are **NOT** selected except for the **selectedButton**, which should be selected. You can set a JButton to be *selected* by saying: `aJButton.setSelected(true)`; JAVA will then show the appropriate image icon automatically. As a result, the background of the button image should be colored in to a non-white color.

5. Adjust your code so that when the user has already selected a game piece, and then an adjacent game piece is selected that lies beside (i.e., to its immediate left, right, up or down position) the selected game piece (i.e., the **selectedButton**) then this piece should swap positions with the **selectedButton**'s piece. Note that you should NOT swap **JButtons** ... just game pieces. However, you WILL need to ensure that the **JButtons**' regular and selected icons are properly swapped also. Just create an appropriate method in the **ShinyButtons** model class to swap two game pieces when given their rows and columns ... then **update()** the GUI accordingly. Once the swap is made, the **selectedButton** should be set to **null** so that no JButtons are selected anymore. You should see the swap happen visually on the window. If the game piece that was pressed was NOT beside the **selectedButton**'s piece, then no swap is made and the **selectedButton** should be set to **null**, so that no buttons remain selected.
6. Adjust your code so that it automatically highlights any horizontal or vertical sequences of 3 or more game pieces if they are the same color (see below).



In order to determine which game pieces to highlight (i.e., to select), you will need to write some code in the **ShinyButtons** class. Create the following attribute: `private boolean[][] selectionTable;`

Create a method in the **ShinyButtons** class called **processTable()** which resets the **selectionTable** to have all **false** values and then checks all rows for sequences of same color pieces and then checks the columns for sequences of same color pieces. Here is some pseudocode for checking the rows:

```
FOR each row r FROM 0 TO 7 DO {
  set matches to 0
  FOR each column c FROM 1 TO 7 DO {
    IF (piece at (r, c) has same color as piece at (r, c-1)) THEN {
      increase matches by 1
      IF (matches >= 2) THEN
        Select all three pieces at row r from column c-2 to column c.
    }
    ELSE set matches to 0
  }
}
```

Do something similar for the columns. Make sure that whenever a new game is started, then the **processTable()** method is called. Also, make sure that it is called whenever two pieces' positions are swapped (from step 5).

## (2) Completing Game:

Now it is time to complete the game

1. Write a **public void cleanTable()** method in the **ShinyGame** class that causes *all selected sequences* of game pieces to be removed from the game board. As a piece is removed from a row, all game pieces above it will drop down one row and a new game piece will appear at the top row of the board (within the same column that the removed piece was taken away). The method should work from the bottom row upwards and look for selected pieces to remove. Here is the pseudocode:

```
set newPieceAdded to FALSE
set r = 7
WHILE (r >= 0) DO {
  newPieceAdded = FALSE
  FOR each column c FROM 0 TO 7 DO {
    IF (piece at (r, c) is selected) THEN {
      // move the pieces down now ...
      FOR each row r2 FROM r TO 1 DO {
        selection of piece at (r2, c) = selection of piece at (r2-1, c)
        piece at (r2, c) = piece at (r2-1, c);
      }
      // Add a new piece
      set the piece color for piece at (0, c) to a valid random color index
      unselect piece at (0, c)
      set newPieceAdded to TRUE
    }
  }
  IF (newPieceAdded is FALSE) THEN
    decrease r by 1
}
```

To test this, add a **private Timer timer;** attribute to the **ShinyButtonsApp**. Add the following **Timer** and its event handler within the constructor:

```
timer = new Timer(500, new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    // Write code here to clean the table, process it, and then update
  }});
timer.start(); // This line starts the timer which will happen every 0.5 seconds
```

2. Finally, we need to work at getting the game to keep score. To record the score, you will need to update your **processTable()** method so that whenever the number of **matches** is 2 or more, you update the score. Make sure to update the score for both rows and columns, separately. Each time you find a sequence of 3 or more similar colored buttons, you will update the score as follows (where **S** is the size of the sequence):

$$\text{score} = \text{score} + 30 + \sum_{i=3}^{S-1} (i*30)$$

So, for example, here is the increase in score that should occur depending on the sequence size:

Sequence Size	3	4	5	6	7	8
Score Increase	30	120	240	390	570	780

Make sure to test your code carefully. However, it will be hard to test sequences of 6, 7 or 8 ... these can only occur by chance if new buttons are added to a row/column which happen to match the others in the row. However, you should be able to test sequences of size 3, 4 or 5. Please note, though, that this game never ends ... so the score will increase until the user quits.

Keep in mind that nothing special needs to happen with horizontal and vertical sequences that cross. For example, in the picture above you will notice a yellow vertical sequence of 4 (worth 120 points) and two horizontal sequences of 3 (worth 30 points each). Hence, for these 8 connected yellow pieces, the score will only be  $120 + 30 + 30 = 180$ . Similarly, the 5 piece horizontal sequence of dark gray buttons is worth 240 points and the crossing vertical sequence of 3 is worth 30 points.

---

**NOTE:** Submit all **.java** and **.PNG** files needed to run. You **MUST NOT use packages** in your code, **nor projects**. Submit ALL of your files in one folder such that they can be opened and compiled individually in JCreator. Some IDEs may create packages and/or projects automatically. You **MUST** export the .java files and remove the package declaration at the top if it is there. Do NOT submit JCreator projects either. **JUST SUBMIT the JAVA FILES and PNG files.** Note that if your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

**Please NOTE that you WILL lose marks on this assignment if any of your files are missing. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.**