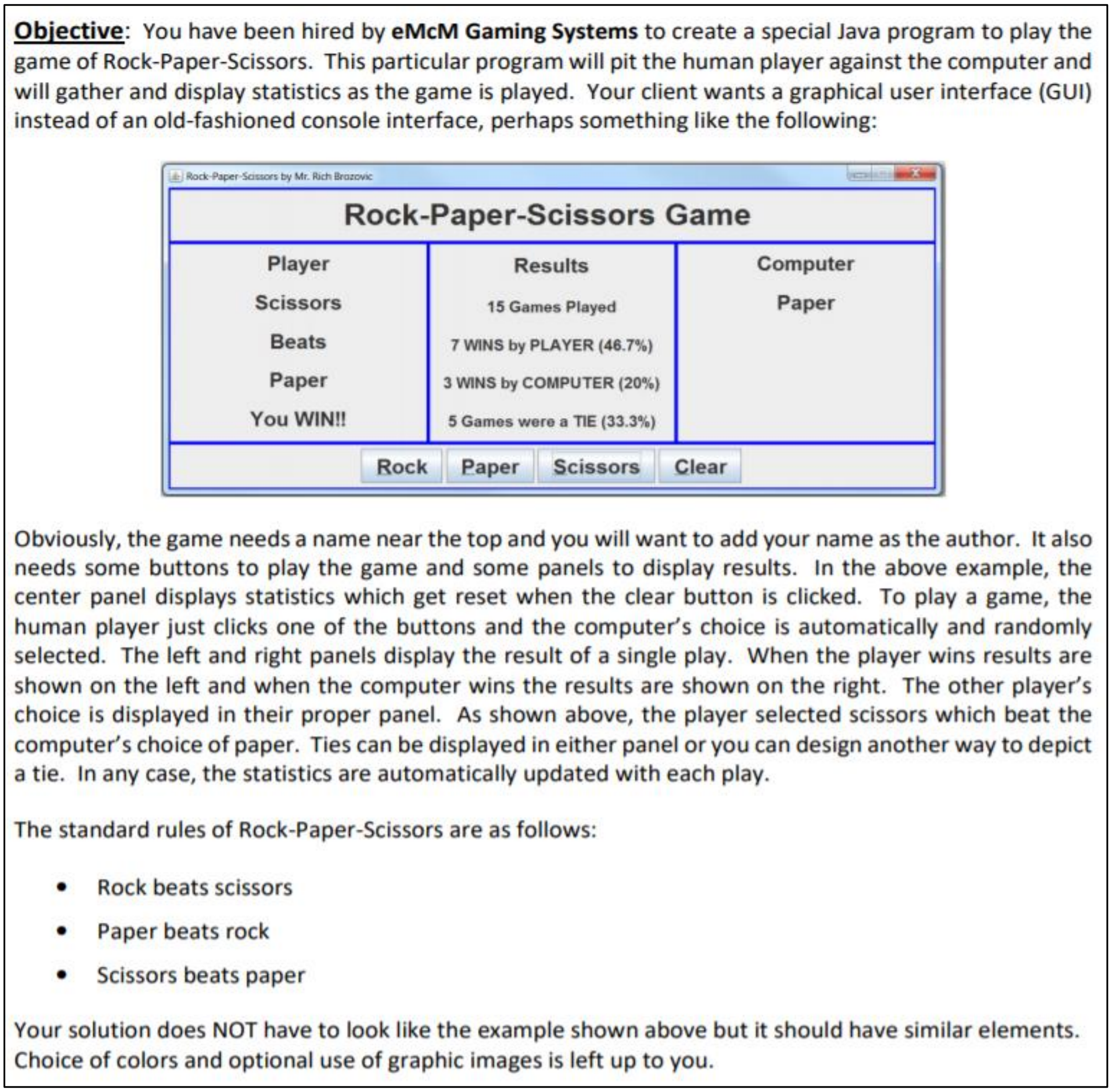
GUI in Java – Lydia Clarke

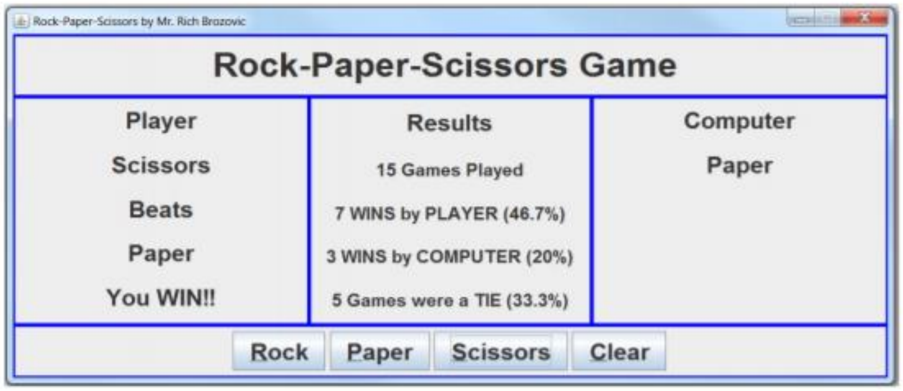
Overview:

During my applications programming class at McMurry University, I was assigned by my teacher an additional project to complete outside of class in order to earn honors credit hours. The assignment would focus on coding a GUI in Java using eclipse which was a topic not taught during the course. This project was meant to be worked on outside of class and required me to research and self-teach the techniques and aspects I needed to execute to successfully complete the assignment. I was given one month to complete the GUI and give a demonstration to the class as well as discuss the code behind it.

<u>Here is the original assignment:</u>



<u>Objective</u>: You have been hired by **eMcM Gaming Systems** to create a special Java program to play the game of Rock-Paper-Scissors. This particular program will pit the human player against the computer and will gather and display statistics as the game is played. Your client wants a graphical user interface (GUI) instead of an old-fashioned console interface, perhaps something like the following:



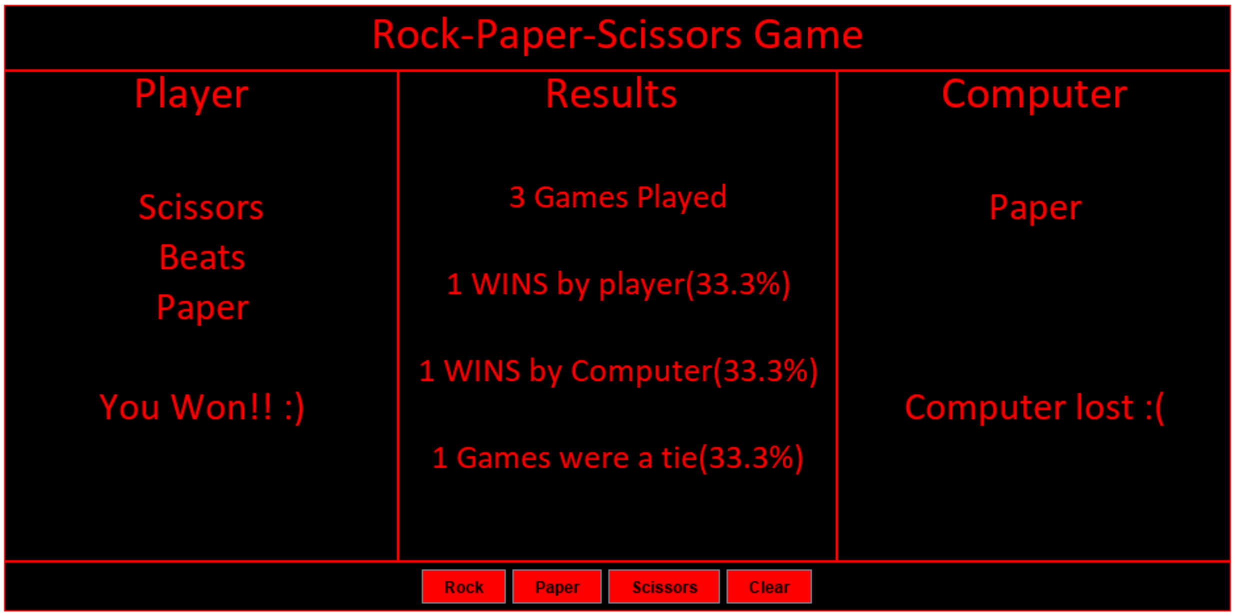
Obviously, the game needs a name near the top and you will want to add your name as the author. It also needs some buttons to play the game and some panels to display results. In the above example, the center panel displays statistics which get reset when the clear button is clicked. To play a game, the human player just clicks one of the buttons and the computer's choice is automatically and randomly selected. The left and right panels display the result of a single play. When the player wins results are shown on the left and when the computer wins the results are shown on the right. The other player's choice is displayed in their proper panel. As shown above, the player selected scissors which beat the computer's choice of paper. Ties can be displayed in either panel or you can design another way to depict a tie. In any case, the statistics are automatically updated with each play.

The standard rules of Rock-Paper-Scissors are as follows:

- Rock beats scissors
- Paper beats rock
- Scissors beats paper

Your solution does NOT have to look like the example shown above but it should have similar elements. Choice of colors and optional use of graphic images is left up to you.

My Final Result:



How I approached this project:

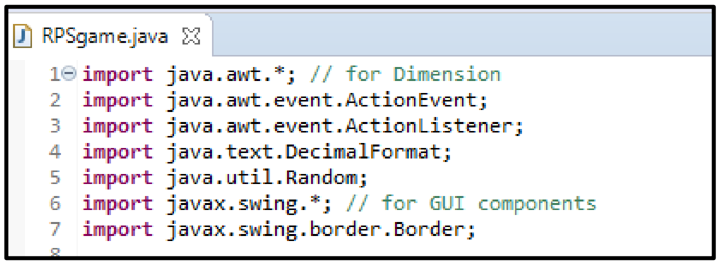
<u>Beginning Process</u>

Being unfamiliar with programming GUIs in Java, I first needed to research how to get started and the basic aspects I needed to include. When I receive any sort of programming assignment, I always start with the simplest parts of my program then build up to the more complex ones. Going piece by piece, helps me clearly understand what I am doing and allows me to quickly locate any errors as I am only changing/adding things one at a time. I always say to myself and when tutoring students at school, start with the easiest parts you know how to do or can figure out quickly and get those working before turning your focus on the harder parts. This helps makes a project easier to approach, and I have noticed also makes it easier for students to get started who may feel overwhelmed by an assignment or project.

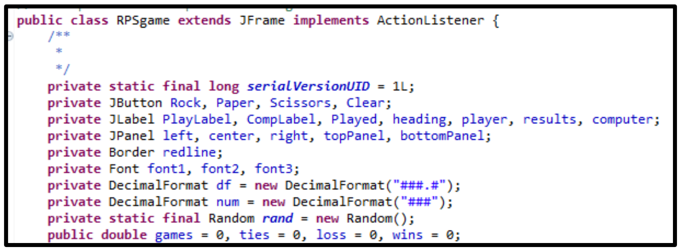
Getting Started:

<u>Importing Classes:</u>

It was important that I import the correct classes into my program for my GUI to run correctly. I needed each of these classes in order for the buttons to function and execute the data correctly when clicked, the percentage results to display the right decimal format, allow the computer to randomize game play, how the objects are positioned on the screen, and the overall appearance of the interface.



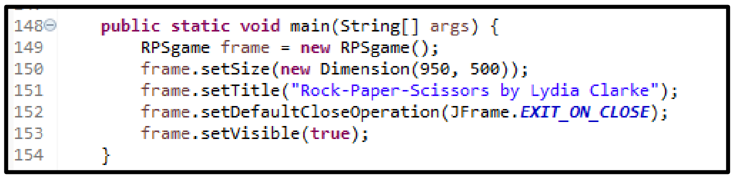
```java
import java.awt.*; // for Dimension
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.DecimalFormat;
import java.util.Random;
import javax.swing.*; // for GUI components
import javax.swing.border.Border;
```



```java
public class RPSgame extends JFrame implements ActionListener {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private JButton Rock, Paper, Scissors, Clear;
    private JLabel PlayLabel, CompLabel, Played, heading, player, results, computer;
    private JPanel left, center, right, topPanel, bottomPanel;
    private Border redline;
    private Font font1, font2, font3;
    private DecimalFormat df = new DecimalFormat("###.#");
    private DecimalFormat num = new DecimalFormat("###");
    private static final Random rand = new Random();
    public double games = 0, ties = 0, loss = 0, wins = 0;
```

Creating JFrame:

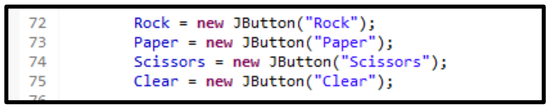
This was my first initial step which was creating the window with its title and including the exit button in the upper left-hand side of the window.



```java
    public static void main(String[] args) {
        RPSgame frame = new RPSgame();
        frame.setSize(new Dimension(950, 500));
        frame.setTitle("Rock-Paper-Scissors by Lydia Clarke");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
```

Once I successfully created the window, it was time to start adding the titles, columns and buttons to my GUI. I focused on making sure the layout of the game looked how it was supposed to before adding any further gameplay content.

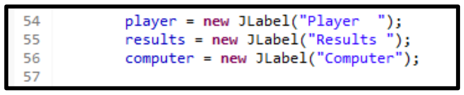
JButtons:



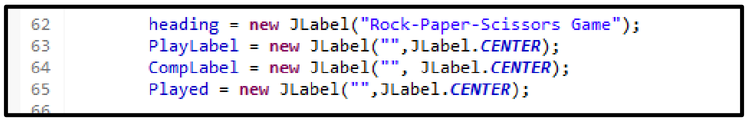
```java
        Rock = new JButton("Rock");
        Paper = new JButton("Paper");
        Scissors = new JButton("Scissors");
        Clear = new JButton("Clear");
```

The first part I started with was the game buttons : “Rock”,”Paper”,”Scissors”, and “Clear”. I was mainly focused on getting them to display on screen with the right button text. I wasn’t going to worry about them doing anything until they appeared the way I wanted. Creating the buttons was one of the easier pieces to do. Even though they were easy to create, positioning them on the bottom of the screen wasn’t as easy as I thought it was once I started to add JLabels.

JLabels:



```java
        player = new JLabel("Player  ");
        results = new JLabel("Results ");
        computer = new JLabel("Computer");
```

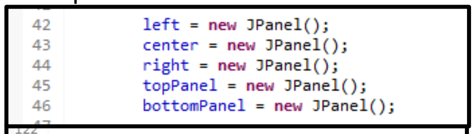


```java
        heading = new JLabel("Rock-Paper-Scissors Game");
        PlayLabel = new JLabel("",JLabel.CENTER);
        CompLabel = new JLabel("", JLabel.CENTER);
        Played = new JLabel("",JLabel.CENTER);
```

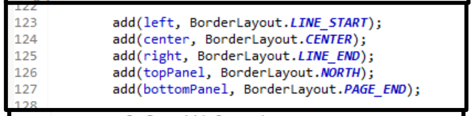
For the game title and the “Player”, “Results”, and “Computer” headers in each column I created JLabels. These lines of code simply tell the computer what the text will say but does not position it at a specific part of the screen. Creating the labels was easy but putting them into a layout became rather tricky. This was where I had one of my first major setbacks. There were so many different ways I could try to code the positions of my titles and headers, but pieces of my GUI overlapped or things would show up in the wrong place or not even show up at all. This was where I started to learn that choosing the right layout to format my labels was very important. I couldn’t just give things a direction and them show up where I wanted them to. After doing some research, I discovered I needed JPanels to contain different parts of my interface, and BorderLayout to position each part correctly.

JPanels & BorderLayout:

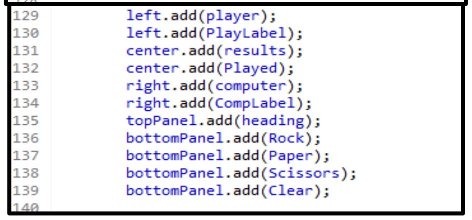
So to position my title, buttons, and column headers I needed to create JPanels. I made a panel for each quadrant within my GUI then added my Labels and Buttons to the Panels I needed them in. I once again ran into a layout problem. I didn’t immediately identify border layout as what I needed. I thought I would simply containerize my labels into the panels and then just give them a direction. Once again, things didn’t show up and the objects that did, were misplaced. I really got stuck at this part trying all different types of layouts. At this point, I consulted my professor and he recommended border layout.



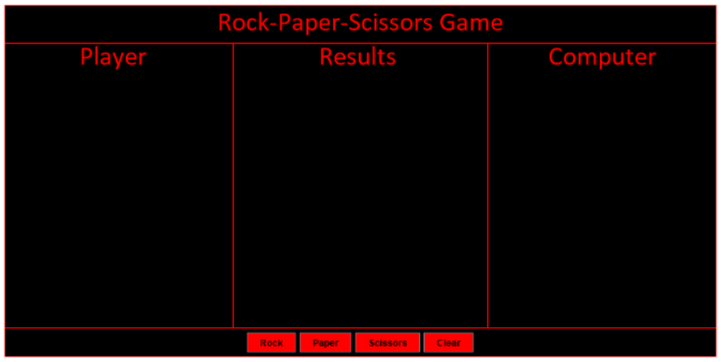
```
        left = new JPanel();
        center = new JPanel();
        right = new JPanel();
        topPanel = new JPanel();
        bottomPanel = new JPanel();
```



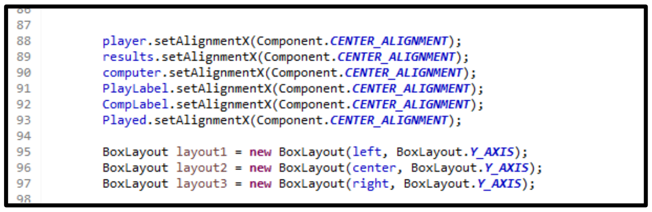
```
        add(left, BorderLayout.LINE_START);
        add(center, BorderLayout.CENTER);
        add(right, BorderLayout.LINE_END);
        add(topPanel, BorderLayout.NORTH);
        add(bottomPanel, BorderLayout.PAGE_END);
```



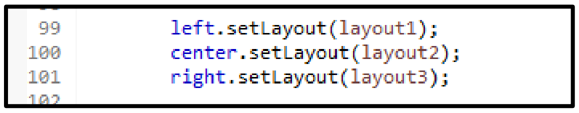
```
        left.add(player);
        left.add(PlayLabel);
        center.add(results);
        center.add(Played);
        right.add(computer);
        right.add(CompLabel);
        topPanel.add(heading);
        bottomPanel.add(Rock);
        bottomPanel.add(Paper);
        bottomPanel.add(Scissors);
        bottomPanel.add(Clear);
```



Border Layout helped get the panels in the right place but my middle three panels still weren’t equal size, and when the text was too long it didn’t move the overflow text to the next line. So I first had to center the text based on the X-axis, and then to show the middle three columns horizontally, I needed boxlayout to set them across the Y-axis. Now that I had the layout looking how I needed, it was time to make the buttons work so the game would be playable.



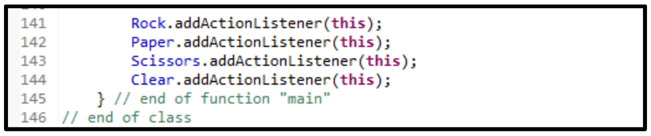
```
        player.setAlignmentX(Component.CENTER_ALIGNMENT);
        results.setAlignmentX(Component.CENTER_ALIGNMENT);
        computer.setAlignmentX(Component.CENTER_ALIGNMENT);
        PlayLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        CompLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        Played.setAlignmentX(Component.CENTER_ALIGNMENT);

        BoxLayout layout1 = new BoxLayout(left, BoxLayout.Y_AXIS);
        BoxLayout layout2 = new BoxLayout(center, BoxLayout.Y_AXIS);
        BoxLayout layout3 = new BoxLayout(right, BoxLayout.Y_AXIS);
```



```
        left.setLayout(layout1);
        center.setLayout(layout2);
        right.setLayout(layout3);
```

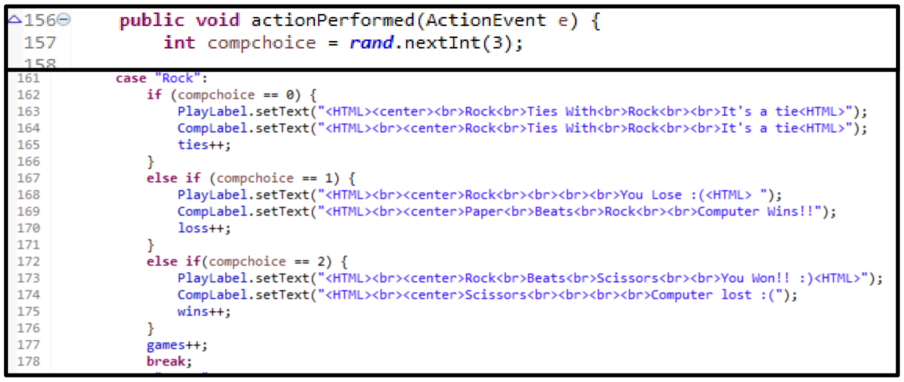
Action listener & Action event

My next step was for the player and computer columns to show what each side played. For my actionPerformed class to work, I needed to add actionListeners to each of my buttons so the computer will react to the button being clicked.

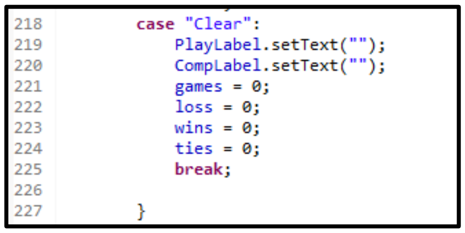


```
        Rock.addActionListener(this);
        Paper.addActionListener(this);
        Scissors.addActionListener(this);
        Clear.addActionListener(this);
    } // end of function "main"
// end of class
```

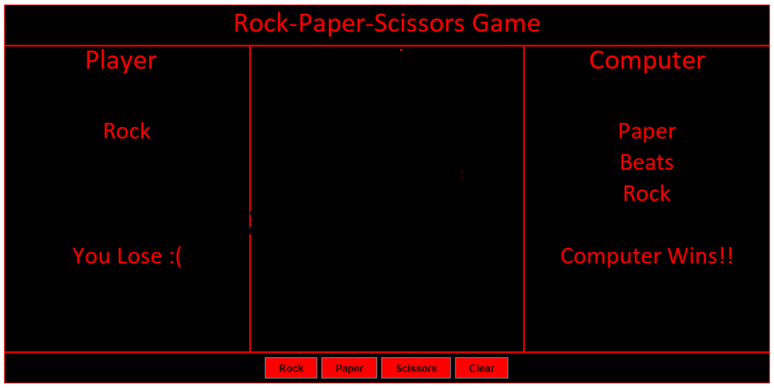
Next, I focused on making sure the player and computer choice displayed first; I wasn’t going to worry about calculating the game averages and percentages yet. I created my actionPerformed class that would tell the computer what to do when a gameplay button has been clicked. This part took a lot of trial and error because I wasn’t sure if I wanted to use nested loops or a switch statement. It took a few iterations of the nested loops not quite working that I changed it to a switch statement with four cases “Rock”, “Paper”, “Scissors” and “Clear”. When the player makes their selection, the computer randomizes a 0,1, or 2. Based on the player choice that case is executed then the program goes through an if-else based on the computer’s random number to determine what message will be displayed in the “player” column and the “computer” column. I also spent extra time trying to get the game results to line up right, I was testing different layouts, but I realized using HTML code in my print statements was much simpler and gave me the best results.



```
    public void actionPerformed(ActionEvent e) {
        int compchoice = rand.nextInt(3);

        case "Rock":
            if (compchoice == 0) {
                PlayLabel.setText("<HTML><center><br>Rock<br>Ties With<br>Rock<br><br>It's a tie<HTML>");
                CompLabel.setText("<HTML><br><center>Rock<br>Ties With<br>Rock<br><br>It's a tie<HTML>");
                ties++;
            }
            else if (compchoice == 1) {
                PlayLabel.setText("<HTML><br><center>Rock<br><br><br><br>You Lose :(<HTML> ");
                CompLabel.setText("<HTML><br><center>Paper<br>Beats<br>Rock<br><br>Computer Wins!!");
                loss++;
            }
            else if(compchoice == 2) {
                PlayLabel.setText("<HTML><br><center>Rock<br>Beats<br>Scissors<br><br>You Won!! :)<HTML>");
                CompLabel.setText("<HTML><br><center>Scissors<br><br><br><br>Computer lost :(");
                wins++;
            }
            games++;
            break;
```

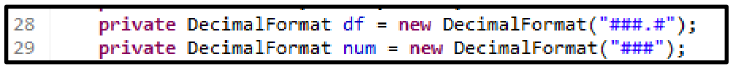


```
        case "Clear":
            PlayLabel.setText("");
            CompLabel.setText("");
            games = 0;
            loss = 0;
            wins = 0;
            ties = 0;
            break;

        }
```

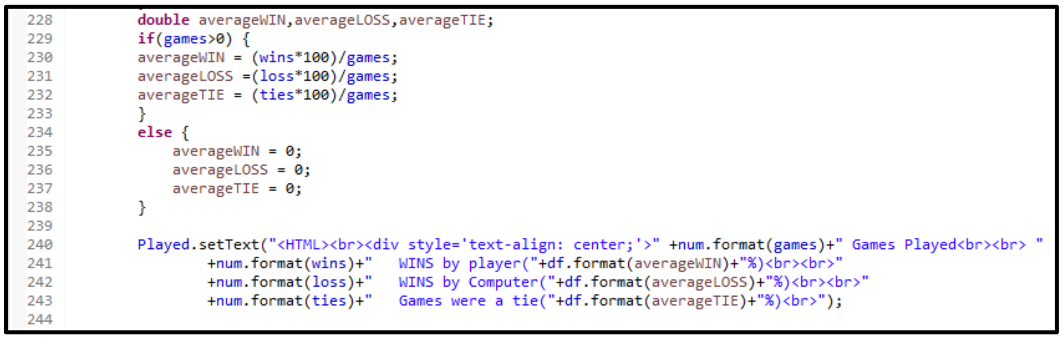


<u>Calculating Averages:</u>

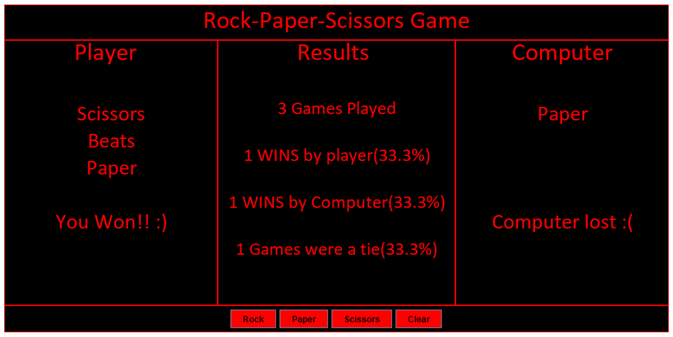
Now that everything was showing up in the correct place, buttons were working, and gameplay results were displaying, I needed to calculate how many games played and the percentage of games won, lost, or tied then have that display in the "Results" column. I used a simple if-else based on the counter variables in the switch statement cases to calculate the averages. This step was simple; however, displaying the correct text in the "Results" column was not. I was able to get the text to show up right, but all my percentages and the game count were zero. I could get the correct numbers if the decimal place was the full 7 places long but I need it to only show one place after the decimal. I finally figured out that if I did not create a decimal format specifying the number of places, all my results would be zero. Now that I finally had my Rock, Paper, Scissors game fully functioning and running correctly, I wanted to change the default view and add some color as well as make the font a little bigger.



```
    private DecimalFormat df = new DecimalFormat("###.#");
    private DecimalFormat num = new DecimalFormat("###");
```



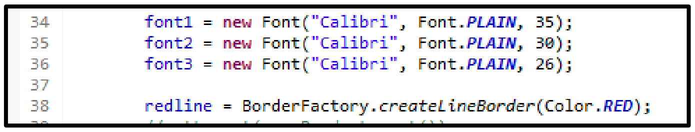
```
        double averageWIN,averageLOSS,averageTIE;
        if(games>0) {
        averageWIN = (wins*100)/games;
        averageLOSS =(loss*100)/games;
        averageTIE = (ties*100)/games;
        }
        else {
            averageWIN = 0;
            averageLOSS = 0;
            averageTIE = 0;
        }

        Played.setText("<HTML><br><div style='text-align: center;'>" +num.format(games)+" Games Played<br><br> "
                +num.format(wins)+"   WINS by player("+df.format(averageWIN)+"%)<br><br>"
                +num.format(loss)+"   WINS by Computer("+df.format(averageLOSS)+"%)<br><br>"
                +num.format(ties)+"   Games were a tie("+df.format(averageTIE)+"%)<br>");

```



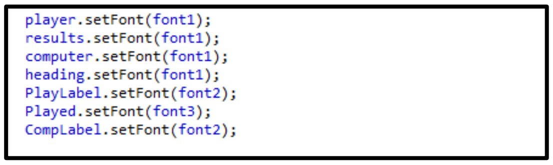
```
Scissors
Beats
Paper

You Won!! :)

3 Games Played

1 WINS by player(33.3%)

1 WINS by Computer(33.3%)

1 Games were a tie(33.3%)

Paper

Computer lost :(
```

Fonts, Borders, & Colors:

To use the font style and size I wanted I simply made three different font styles, and anywhere I wanted to use that font I did ".setFont".

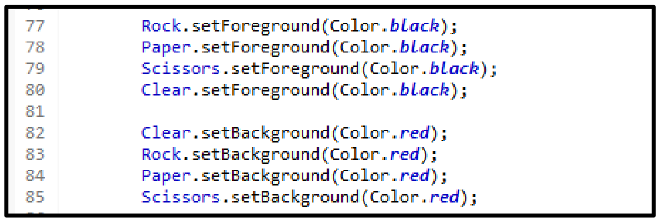


```java
        font1 = new Font("Calibri", Font.PLAIN, 35);
        font2 = new Font("Calibri", Font.PLAIN, 30);
        font3 = new Font("Calibri", Font.PLAIN, 26);

        redline = BorderFactory.createLineBorder(Color.RED);
```

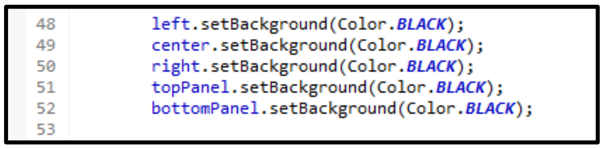


```java
player.setFont(font1);
results.setFont(font1);
computer.setFont(font1);
heading.setFont(font1);
PlayLabel.setFont(font2);
Played.setFont(font3);
CompLabel.setFont(font2);
```

For font color, I only just changed the foreground and background colors. So the foreground color set the color of the text.

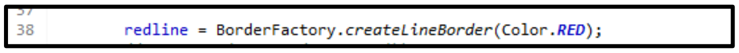


```java
        Rock.setForeground(Color.black);
        Paper.setForeground(Color.black);
        Scissors.setForeground(Color.black);
        Clear.setForeground(Color.black);

        Clear.setBackground(Color.red);
        Rock.setBackground(Color.red);
        Paper.setBackground(Color.red);
        Scissors.setBackground(Color.red);
```

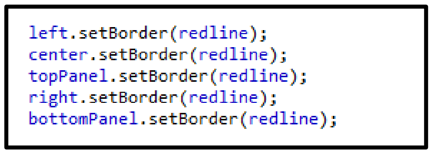


```java
        left.setBackground(Color.BLACK);
        center.setBackground(Color.BLACK);
        right.setBackground(Color.BLACK);
        topPanel.setBackground(Color.BLACK);
        bottomPanel.setBackground(Color.BLACK);

```

For my red borders throughout the GUI, I simply created one style of border called "redline', then used ".setBorder" to apply it to whatever object I wanted to have a red border.



```java
        redline = BorderFactory.createLineBorder(Color.RED);
```



```java
left.setBorder(redline);
center.setBorder(redline);
topPanel.setBorder(redline);
right.setBorder(redline);
bottomPanel.setBorder(redline);
```

Takeaways:

This project was in no way a walk in the park. I faced many setbacks and difficulties throughout my time working on it. Some days I would have a lot of progress, while other days I would be completely stuck. This GUI program really challenged me as it was something I had never done or been taught before. While I know GUIs aren't commonly programmed this way in Java anymore, programming this improved my skills of being able to find the answers to new and difficult challenges while also making me feel more confident in my abilities as a programmer. This project is a great reflection of my ability to self-teach and approach programming techniques/topics I am not experienced with. I was also able to finish within the time limit of one month on top of tutoring in the library and my work for other courses which shows that I can efficiently manage and balance my time.