

# Homework 7 – Color Matching Game

Authors: David Cornell, Allan Nguyen, Cindy Kwok, Sumit Choudhury

## Problem Description

This assignment will test your knowledge of JavaFX to create a basic GUI (Graphical User Interface) for a game.

You will be creating a color matching game! It will be a brain-teaser game that shows the player the text name of a color and an actual visual color (probably not matching each other). The player then needs to select, from a set of possible choices, a visual color that matches the text color name they read. It may sound easy, but studies have shown that it's more difficult than you may think. Our brains have a difficult time resolving the difference between the color text name and the visual color that you see. Every time the player picks a correct match, they get one more point. However, whenever they make a wrong choice, their score goes back to 0.

You will be designing a GUI for the game. The GUI contains a region for the player to enter their name to be shown. There is also a region containing a “question” box that has the name of a random color in it, and the box itself is visually colored with another random color. Below that are three “answer” boxes that are set up the same way, each also containing the name of a random color and the box itself visually colored another random color. **A “correct” answer is a selection of an answer box whose VISUAL color is the same as the COLOR TEXT NAME that is spelled out in the question box.** There is also a “None” answer box for when none of the other answer options are correct. For example, if the question box is visually green and says “Yellow,” then a correct option could be an answer box that is visually yellow but says “Red.” See the examples below for further clarification. The GUI also contains a “Reset” button for going back to the beginning state of the game.

## Requirements

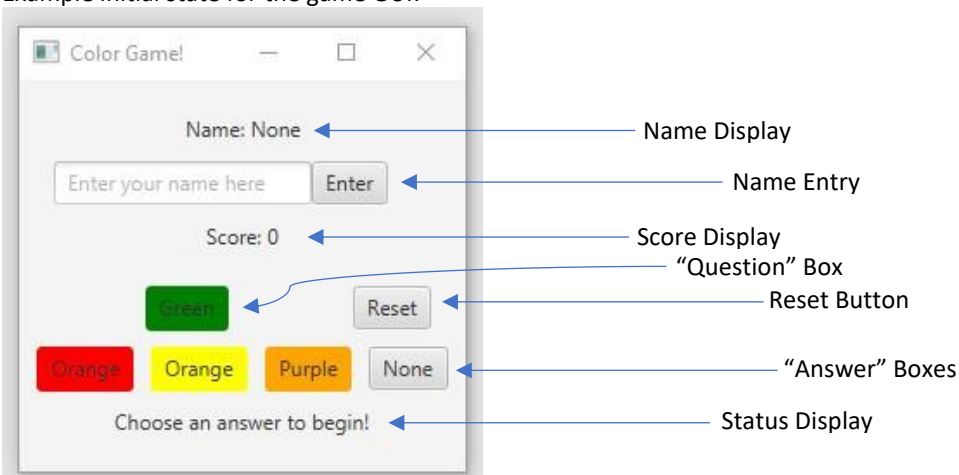
There are two type of requirements: Visual Requirements and Functionality Requirements.

### **Visual Requirements:**

These are the visual components that your GUI must have. For functionality details, see Functionality Requirements below.

- **“Question” Box:** There must be a “question” box that contains the name of a color and is visually colored some color.
- **“Answer” Boxes:** There must be THREE (3) “answer” boxes that each contain the name of a color and are visually colored some color. There must also be a fourth box for a “none” option for when no other answers are correct. There must be some way to select an answer from among the options.
- **Reset Button:** There must be a “reset” button that resets various aspects of the game GUI.
- **Name Entry:** There must be a text entry field that allows the player to type their name. There must also be an “enter” button to enter the name.
- **Name Display:** There must be some display of the name that the player entered. This must be separate from the name text entry field itself.
- **Score Display:** There must be some display of the current score.
- **Status Display:** There must be some display of the current status of the game. There are 3 statuses: (1) the initial state of the game, (2) when the player has just answered correctly, and (3) when the player has just answered incorrectly.

Example initial state for the game GUI:



### Functionality Requirements:

These requirements detail how the game must work and how the GUI must be updated to display things correctly. See the examples below for further clarification.

- **Initial State:** When the game window first opens (see example above for a possible initial state):
  - The Name Display should have “None” as its name.
  - The Status Display should show a brief instruction message on how to begin.
  - The Score Display should show a score of zero (0).
  - The question box and answer boxes should be generated for the first round (see Color Generation below).
- **Color Generation:** The question box and answer boxes (except for the “none” answer box) must randomly generate a color name to display AND a color to be applied as the box color for each round. The minimum list of colors should be: **Red, Orange, Yellow, Green, Purple**. Feel free to add more colors if you wish.
  - The generated colors do NOT have to be unique or distinct; for example, a yellow box that says “Yellow” is fine, and multiple boxes may randomly have the same colors for a round.
- **Answering:** The player must be able to submit an answer for each round. The options should be the three answer boxes and the “none” answer box. If a correct answer is chosen, the score should increase by one (1) in the Score Display, and the Status Display should change to a message that tells the player they were correct. If an incorrect answer is chosen, the score should be set to zero (0) in the Score Display, and the Status Display should change to a message that tells the player they were incorrect.
  - A correct answer is an answer box whose VISUAL color is the same as the COLOR NAME that is spelled out in the question box. See the examples below for further clarification.
  - There could be multiple correct answers for a round (since the answer options are randomly generated and do not have to be unique or distinct). In that case, any of them should be counted as a correct answer. Additionally, when there IS a correct color answer option available, the “none” answer should be counted as an incorrect answer. The “none” answer is only correct when none of the other answer options are correct.
- **Name Entry:** The player must be able to type a name into the Name Entry field and be able to click an “enter” button to enter it. When the “enter” button is pressed, the Name Display should change to display “Name: <player’s name>”. Alternatively, you may choose to have the player press the “Enter” or “Return” key on their keyboard instead of pressing a button on the GUI.
  - You do NOT need to worry about handling any edge cases for name entry (such as pressing the “enter” button when no name has been entered yet).
- **Reset Button:** The Reset Button must reset everything to match the Initial State. This means it must do ALL of the following:
  - Reset the name and Name Display to display “None” as its name

- Reset the Status Display to display the brief instruction message on how to begin
- Reset the score and Score Display to display zero (0)
- Reset the Name Entry field to be empty (or contain some prompt or default text, if you choose)
- Re-generate the question and answer boxes for a new round (see Color Generation above)

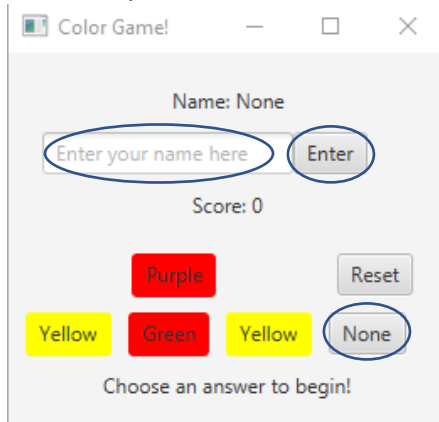
## ColorGame.java

This is where your overridden JavaFX Application `start` method must be located in order to run the JavaFX program. The game GUI must run when ColorGame is run as a JavaFX program. Other than that, implementation is entirely up to you! You may create methods and classes to help however you want. Feel free to have fun with it, add additional features and functionality, and make it look nice as much as you want! See Import Restrictions below for allowed imports. See the Tips section below for helpful tips on how to implement ColorGame.

## Example Game

Here is an example of what a few rounds of the game may look like. The first (top) image is the initial state, and successive rounds are shown in the following pictures, going downwards. **The blue ovals on each image show what the player does on that round; they are NOT part of the GUI.** To the right of each image is an explanation of what the player does for that round. Your GUI may not look exactly like this, and that's ok! This is just an example of a game GUI that fulfills the requirements.

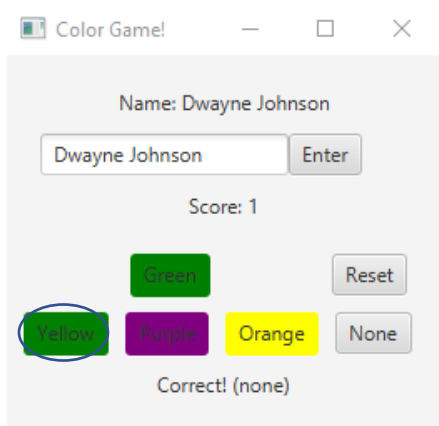
Initial State/Round One:



This is the initial state when the game GUI is first run. The Name Display says "None", the Score Display shows zero (0), and the Status Display shows a brief instruction message.

**Action:** The player enters the name "Dwayne Johnson", then presses the "Enter" button, and then chooses the "None" answer option.

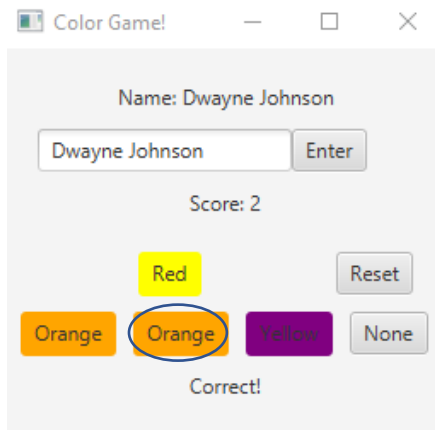
Round Two:



The Name Display now displays "Dwayne Johnson", the Score Display shows one (1) because the player chose a correct answer ("None") for Round One, and the Status Display shows a "correct" message.

**Action:** The player chooses the leftmost answer option.

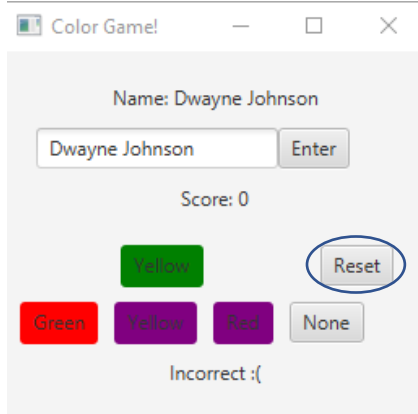
#### Round Three:



The Score Display now shows two (2) because the player chose a correct answer (a box with a green background) for Round Two, and the Status Display shows a “correct” message.

**Action:** The player chooses the middle answer option.

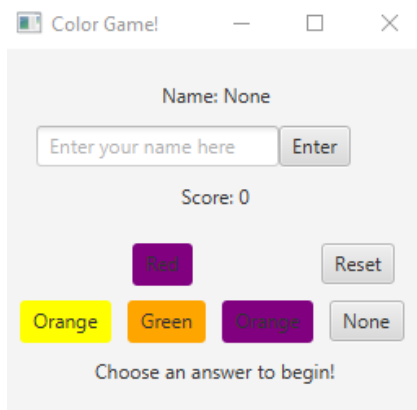
#### Round Four:



The Score Display now shows zero (0) because the player chose an incorrect answer (a box with a background color that is NOT red) for Round Three, and the Status Display shows an “incorrect” message.

**Action:** The player presses the “Reset” button.

#### Round Five:



Because the player chose to reset, the Score Display shows zero (0), the Name Display shows “None”, the Name Entry field is empty (or contains some prompt or default text), the Status Display shows a brief instruction message, and the “question” and “answer” boxes have been re-generated for a new round.

**Note:** there is no end or termination condition; the player can continue to try again (or reset) for as long as they wish. Also, there is no “round” counter; the round numbers given in the example are just for clarity.

## Optional Extra Credit Opportunity

Up to 20 points of extra credit on this assignment will be awarded for non-trivial improvements to the GUI and/or the game functionality. Up to 10 of these points can come from *functionality improvements or additions*, and up to 10 can come from *good design improvements*. These additional features should not affect any of the base functionality. If participating in extra credit, also submit a text file called **extra\_credit.txt** that briefly explains each additional feature included (just a few sentences of explanation per feature is sufficient).

Some functionality improvement examples include:

- Audio feedback for correct and incorrect answers
- Making each random color and/or answer choice distinct/unique
- Randomizing the color of the text itself in the question and answer boxes (be sure that the text color is not the same as the box color)
- Allowing the user to input their answers as key presses, as well as still being able to click on the GUI

Some design improvement examples include:

- Adding appropriate images and/or graphics that add to the user experience
- Improved feedback/response to the player
- A separate title or introduction screen

Feel free to be creative; these are just a few ideas!

## Tips

You can implement the game GUI however you want as long as it meets the requirements listed above. However, here are some tips and suggestions that might help:

- The Internet is your friend! JavaFX has lots of classes and functions, so feel free to look up how they work or how to do some *specific* thing (ex. "How do you center text in a Label in JavaFX?"). However, it is not allowed or helpful to take code directly from online sources; the same policies apply to this homework as have applied to every other assignment for this course. Online sources can help with the syntax of specific classes and method calls, but plagiarism is not allowed.
- Before thinking about implementation, think about what you want to lay out your components. Draw it out by hand and decide what kinds of panes you want the visual components in and how to nest those panes inside each other.
- Using a `BorderPane` as the outermost, or "root," pane is a simple way to put some things in the center of the screen and other things around those things. `VBox` and `HBox` panes are also very helpful.
- Creating an array instance variable with the possible color options (Red, Orange, Yellow, Green, Purple at minimum) may be helpful for randomly choosing a color for color generation.
- In order to update the information on the screen, having all of the Labels, Buttons, etc. (whatever has text or colors that need to be able to change) declared as instance variables in the `ColorGame` class could also be helpful. Then, if you want to change the text of a component, you can use the following:

```
o variableName.setText("New Text Example");
```

You can change pretty much anything about a component like this.

- To change the color of a component, one option is to use the `setStyle(String)` method. The parameter is a String that contains some CSS information for the component's style, but the only one we need to worry about is the background color. For example, if you have a Button called "b" that you want to make red, you can do the following:

```
o b.setStyle("-fx-background-color: Red");
```

There are other ways to change a component's color (and other attributes), such as using individual methods and classes like `setBackground(Background)` and `setFill(Color)`, which you may use if you desire. The `setStyle(String)` method is merely the most versatile.

- The `setText(String)` and `setStyle(String)` methods will immediately update the GUI, so using these on the instance variables mentioned above is an easy way to change what is being shown.

- To check the text of a component, you can use the `getText()` method. This returns a `String` containing the component's text.
- To check the color of a component, you can use the `getStyle()` method. This returns a `String` that contains the style information, formatted the same way as described above for `setStyle(String)`.
- To space things out, you can use `Insets`. After importing `javafx.geometry.Insets`, you can set a pane's padding by calling `paneName.setPadding(new Insets(10, 10, 10, 10));`. You can change the values to adjust the spacing.
- Methods are your friends. If you create methods to reset, submit an answer, create a new round, etc., you can easily call them (and they can directly access those instance variables mentioned above).

## Submitting

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `ColorGame.java`

If you are attempting extra credit points, be sure to also upload:

- `extra_credit.txt`

Make sure you see the message stating, "Homework 7 submitted successfully." We will only grade your last submission. If you choose to create additional files, be sure to **submit every file each time you resubmit**.

## Import Restrictions

To prevent trivialization of the assignment, you may only import:

- `java.util.Random`
- `java.util.ArrayList`
- Anything from `javafx`

If there is anything else you wish to import, feel free to post a private Piazza post asking if it is allowed for this assignment.

## Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

## Checkstyle and Javadoc

You must run Checkstyle on your submission. (To learn more about Checkstyle, examine our course's external website [Java Resources page](#) under "CS 1331 Style Guide".) The Checkstyle cap for this assignment is **20 points**. If you don't have Checkstyle yet, download it from Canvas → files → `checkstyle-8.28.jar`. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early! Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

## Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due. Only post code on Piazza in a **private** post.

## Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications
- Make sure to test your program manually based on the assignment instructions and expected outputs. The Gradescope autograder visible to you is **NOT** comprehensive.