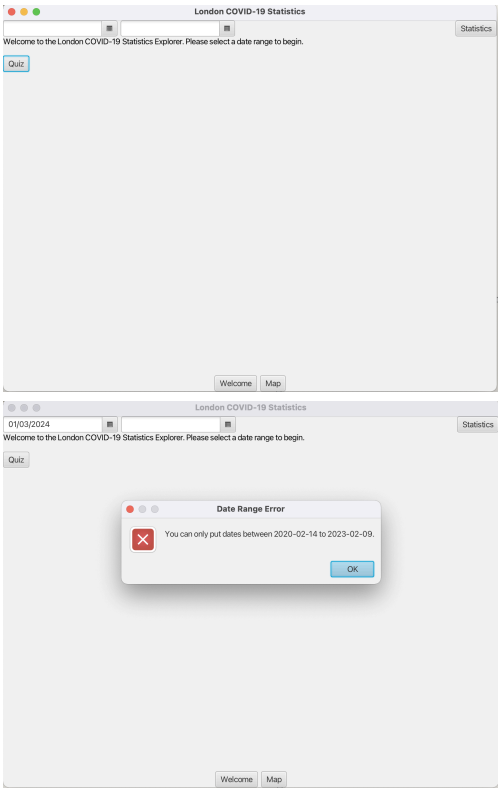
Covid Data Controller - Coursework 4

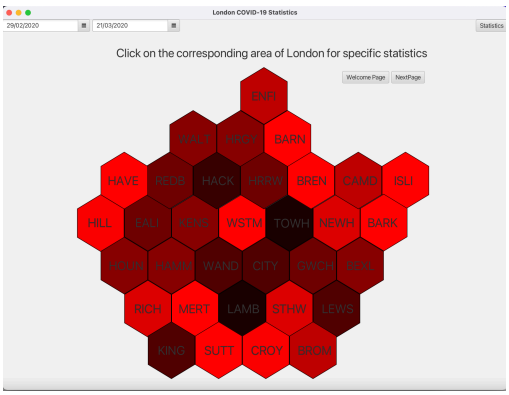
By: Eduardo Sanchez Morales (k23025983), Marc Mot (k23040798), Saif Al Dhaheri (k21210342)

In this final project, we created a graphical user interface application that reads data from a .CSV file with covid-related data, such as deaths, cases, locations… The application has multiple features and panels, which will be discussed into more detail later. The main features are the main welcome page, where the user can input certain dates for which they want to retrieve data, and, according to the selected dates, the other panels change. So, for example, Panel Nº2 has a map that showcases each borough in London, and changes color according to the death rate. Then, the statistics panel (Nº3) showcases some main statistics about everything, in the form of a slide show. And, finally, the last panel (Nº4), is a quiz that tests the user on their COVID-19 knowledge and how to prevent spreading / getting covid. Subsequently, I will explain the GUI for each panel, and how we completed it.

Panel	**Screenshot**	**Important code used**

Welcome		The Welcome panel is the first one. It is initialized by the Interface App, and has the following buttons: Date pickers, Statistics, Welcome (back button), and the Map button. The Welcome page is the hub. It redirects and accesses all the other panels for the user.
		
		When the dates are picked, the Cove DataLoader class filters all the data from the dates the user has chosen. However, the dates have to be within the range of the provided data. So, we created an Alert that pops up when the data is not within range by using this if statement:
		
		if (!isDateWithinRange(fromDate)) {
		showAlert();
		datePickerFrom.setValue(startDate);
		
		The alert is shown (we learnt this in Week 9 of this term when we were taught the different messages we could throw to the user). Then, if the dates are not within range, the dates are automatically filled out with the minimum date and the most recent date. We used the LocalDate library to do this because this way we can parse the dates in:
		private final LocalDate startDate = LocalDate.parse("2020-02-14");
		private final LocalDate endDate = LocalDate.parse("2023-02-09");

Map



This is the Map, which can be accessed by clicking the Map button on the bottom center of the Welcome Page.

The design was made using SceneBuilder, and we put all the polygons together, naming each one of them with their corresponding borough name. Each polygon has a button inside that then provides specific data stats on the borough, but that is for the MapData part.

As you can see, the colors change according to their death rate. To do this, we imported the data from the Covid Data Loader. The death rate is calculated by dividing totalNewDeaths by totalDeaths and multiplying by 100 to get a percentage. This formula gives the proportion of new deaths relative to the total number of deaths.

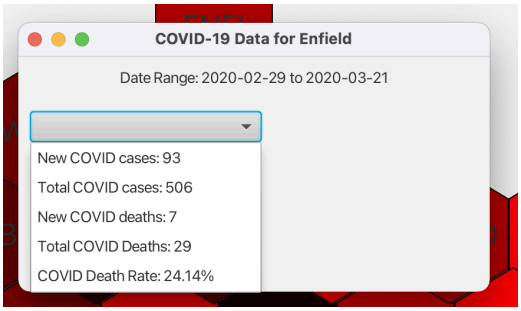
```
private double calculateDeathRate(List<CovidData> boroughData) {
    int totalNewDeaths = boroughData.stream().mapToInt(CovidData::getNewDeaths).sum();
    int totalDeaths = boroughData.stream().mapToInt(CovidData::getTotalDeaths).sum();
    return totalDeaths > 0 ? (double) totalNewDeaths / totalDeaths * 100 : 0;
}
```

Then, the intensity is calculated by taking the minimum of 1 and the death rate divided by 10. This scales the death rate to a 0-1 range suitable for color intensity. According to the intensity, If the intensity is high, the color will be a brighter red. If it's low, the color will be closer to black.

```
//between red shades.  If it's low, the color will be closer t
private Color getColorForDeathRate(int deathRate) {
    double intensity = Math.min(1, deathRate / 10.0);
    return Color.color(intensity, 0, 0);
}
```

Finally, we just set the color of the polygon by using polygon.setFill(color);

Map data

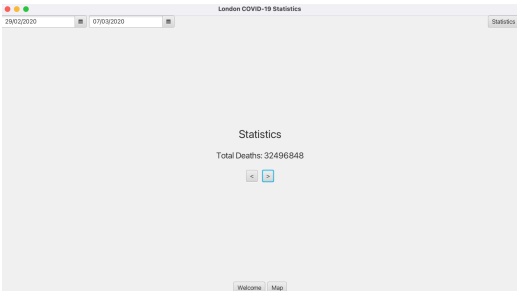
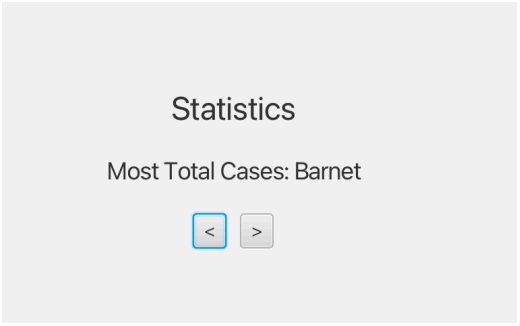


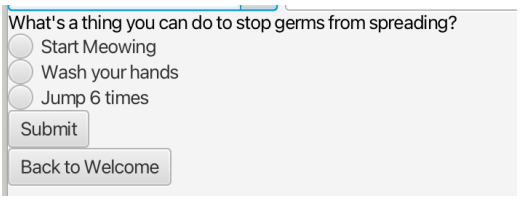
Once we are in the map, we can click on the polygons to find more specific data about the boroughs and covid. So, if we click on Enfield, we can get the New Covid cases, Total cases, New and Total deaths, and finally, the death rate. To do this, we simply made an integer variables, and iterated through all the data from the Covid Data class, and added them to the integer counter. This data is the displayed

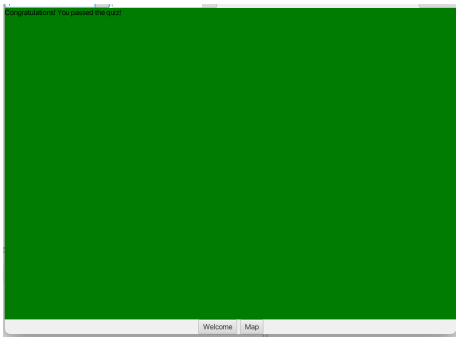
```
// Calculating totals by iterating and adding
for (CovidData data : boroughData) {
    totalNewCases += data.getNewCases();
    totalCases += data.getTotalCases();
    totalNewDeaths += data.getNewDeaths();
    totalDeaths += data.getTotalDeaths();
}
```

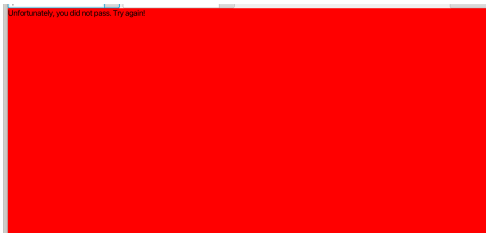
The data is then displayed in a combo box, which is a mix between a list and a text for the menu-like representation of the data, which is what we were asked for.
https://docs.oracle.com/javase%2Ftutorial%2Fuiswing%2F%2Fcomponents/combobox.html

```
// Creating the labels with the covid data stats
Label newCasesLabel = new Label("New COVID cases: " + totalNewCases);
Label totalCasesLabel = new Label("Total COVID cases: " + totalCases);
Label newDeathsLabel = new Label("New COVID deaths: " + totalNewDeaths);
Label totalDeathsLabel = new Label("Total COVID Deaths: " + totalDeaths);
Label deathRateLabel = new Label(String.format("COVID Death Rate: %.2f%%", deathRate));
```

Statistics	
	 
	 
		The statistics panel is accessed through the statistics button in the welcome page or the map. It uses for loops on all the recorded data to get all the data needed. For the statistics, we have:
		- Average google mobility GMR
		- Total Deaths
		- Average Total Deaths per Borough
		- Borough with most Total Cases
		- Borough with most Total Deaths
		
		The screenshots are provided on the left.
		
		To get the data, we did the following code:
		
		`private double calculateAverageTotalCases() { List<CovidData> records = dataLoader.getFilteredRecords(); int totalCases = 0; for (CovidData record : records) { totalCases += record.getTotalCases(); } if (records.isEmpty()) { return 0; } else { return (double) totalCases / records.size(); } }`
		We make a list with all the filtered data from the Covid Data, and then use a for loop to get all the total cases (or deaths, google mobility ratio…). For the GetMaxCases for each borough, we do the same thing, but created a Map Key,Value pair that holds the key as the borough name and the value as the cases/deaths per borough. The one with more is the max borough and the key gets printed out.
		`String maxBorough = "Not available"; int maxCases = -1; for (Map.Entry<String, Integer> entry : casesByBorough.entrySet()) { if (entry.getValue() > maxCases) { maxCases = entry.getValue(); maxBorough = entry.getKey(); } } return maxBorough; }`
		
		Finally, toTo navigate, we created this system with a direction to navigate through the statistics like a list. It has the statistics at certain indexes and when you click the right button, the index goes up by 1, and the left one goes to -1. Then, there are stoppers if the index is at 0, you can't go further down.
		
		`if (currentStatisticIndex < 0) { currentStatisticIndex = 0`

Quiz		Finally, the quiz is the challenge task that tests the users knowledge. So, I created an array that holds each question and answer, and an integer that list that tracks all the answers.






```
// if the socre is more than 7 or 7, then the user passes.
if (score >= 7) {
    resultText.setText("Congratulations! You passed the qu
    this.setBackground(new Background(new BackgroundFill(C
} else { //if less, then they don't pass.
    resultText.setText("Unfortunately, you did not pass. T
    this.setBackground(new Background(new BackgroundFill(C
}

this.getChildren().add(resultText); // Show the resultText
}
```

```
// Array of questions and their possible answers
private String[][] questions = {
    {"What's a thing you can do to stop germs from spreading?", "Start Meowing", "Wa
    {"Where should you sneeze?", "Hand", "Person next to you", "Elbow"},
    {"Can animals get covid?", "Only cats", "Rarely but yes", "No"},
    {"How long should you wash your hands to kill 99.99999% of germs?", "Meow 10 tim
    {"What should you wear to help protect against covid?", "A wizard hat", "A face
    {"When is it okay not to wear a mask during the pandemic?", "While swimming", "I
    {"If you feel sick, what should you do?", "Go to Heaven", "Stay home", "Go to PP
    {"What are the chances of getting covid after being in a room with someone that
    {"What should you do with used tissues?", "Eat them", "Throw them in the bin", "
    {"Which of these is NOT a COVID-19 symptom?", "I start meowing too often", "Sore
};
private int[] answers = {1, 2, 2, 0, 1, 0, 1, 2, 1, 2}; // Correct answers indices
```

The buttons are made with a Radio type button that are round, and each button represents an answer. To load the questions, we used a for loop, to set each question with their corresponding answers and their actual answers (the integer list with answers). To check the answer, we compare the answer to the given question to the button selected, and if it's correct, the user gets score of +1. (options[answers[CurrentQuestionNum]].isSelected()).

The score is kept in a variable called score, and if it is higher than or equal to 7 by when the quiz is finished, then the user passes the test, making the background green and a text shows up for congratulations. If the score is less than 7, then the screen turns red.

Unit testing:

For unit testing, we used preventive and debugging measures in order to test our data. For example, we used the System.out.println("Retrieved " + boroughData.size() + " records for the borough: " + boroughName); to see if the data was actually received, and then the same for the filtered data: System.out.println("Filtered " + filteredRecords.size() + " records for date range.");.

Additionally, we used the try and catch method in order to debug and prevent errors in loading the data within the range that we want.The try-catch block catches any DateTimeParseException that might occur if the record.getDate() does not conform to the expected date format. This ensures that the program doesn't crash due to an unparseable date string. When an exception is caught, it prints an error message along with the problematic date string, and then the stack trace of the exception is printed for debugging purposes.

```
for (CovidData record : allRecords) {
    try {
        LocalDate date = LocalDate.parse(record.getDate(), formatter);
        System.out.println("Record date: " + date);

        if (!date.isBefore(startDate) && !date.isAfter(endDate)) {
            filteredRecords.add(record);
        }
    } catch (DateTimeParseException e) {
        System.err.println("Error parsing date: " + record.getDate());
    }
}
```

Completion:

- **Eduardo Sanchez:** Written report, and map class.
- **Marc Mot:** Welcome page and quiz panel.
- **Saif Al Dhaheri:** Welcome page, statistics panel.