Criterion B – Design

Section Overview

User Interface

The program’s main layout will have five sections (see Figure B.1), map, options, country ranking, and play statistics scrollbar.

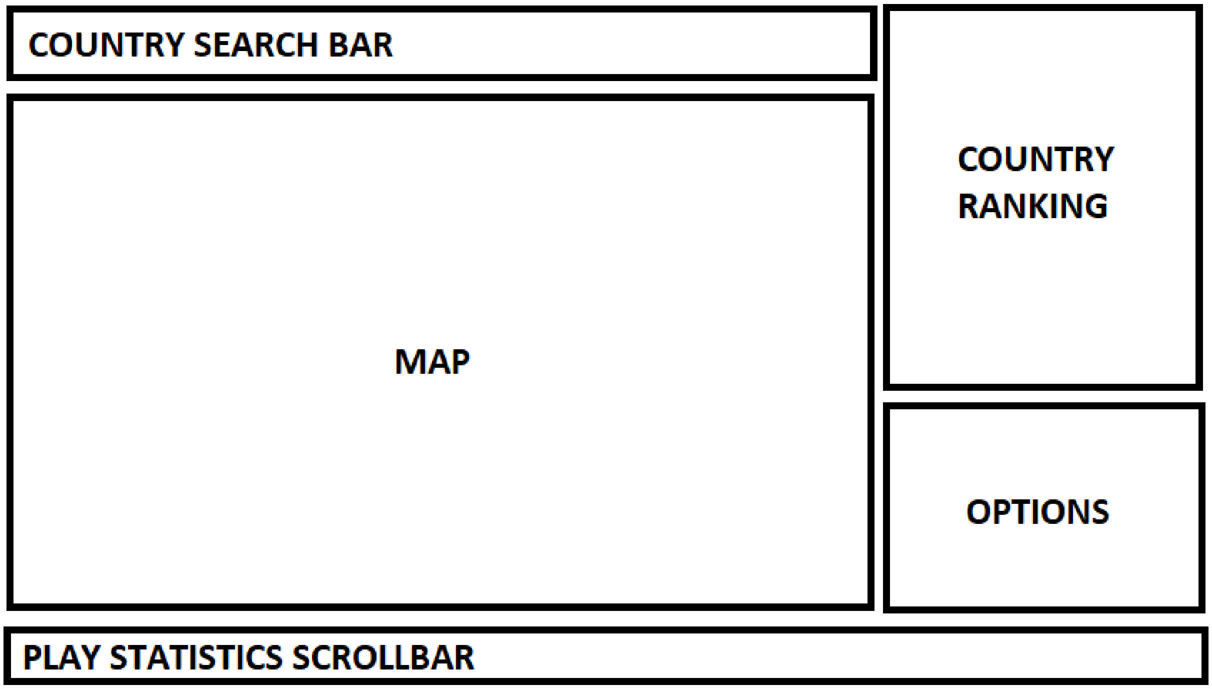


Figure B.1: initial layout of main page of the proposed Coronavirus Analyzer program.

Application Component	**Features**
Map	Displays a colour-coded world map by country. The colours will represent the magnitude of some data such as darker shades for countries with relatively more new cases. This can be algorithmically completed using SVG files.
Within the Map	The map will be interactive, allowing the user to click on countries to display more information about the country as an alternative to searching the name.
Options	Allows the user to select different data they would like to display such as cumulative cases instead of new cases. This is simply implemented using radio buttons.
Country Ranking	Displays the countries in highest magnitude to lowest of some statistics. Sorting is required here, perhaps quick sort because of its efficiency, average of O(n log n).
Statistics Scrollbar	Allows the user to scroll through time. For example, the user can scroll from January 2020 to March 2021 to view the daily changes in the colours in the map and also view the change in rankings.
Play Button on Play Statistics Scrollbar	The program automatically scrolls the scrollbar. Simple action listener.
Country Search Bar	Search for a country the user would like to view more statistics on. After the user clicks the result of the search, a pop-up window about more statistics such as a graph of the cumulative cases will appear. Perhaps use searching algorithms like binary search to efficiently find country in O(log n) time.

Table B.2: initial proposed solution and features plan.

More specifically, the search bar will have two portions. One section is a JTextField which allows the user to input a country. Another section is a “GO!” button, which performs the action of the search. This is shown in Figure B.2 below.

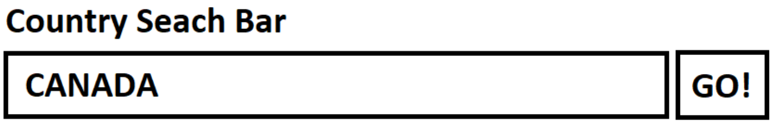
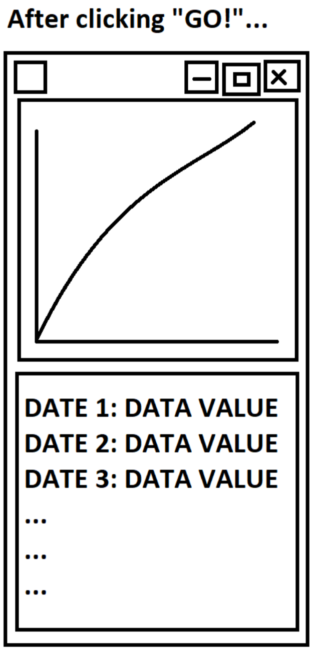


Figure B,2: initial proposed country search bar.

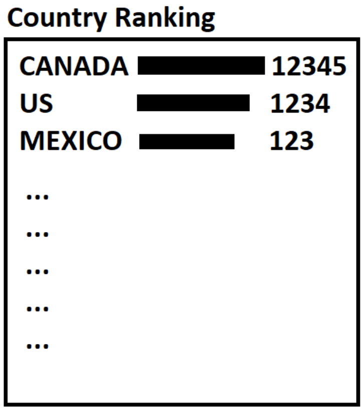
After the user selects go there could be a pop-up window showing more information such as a graph of cumulative data and a table specific to the country in the search bar.



DATE 1: DATA VALUE
DATE 2: DATA VALUE
DATE 3: DATA VALUE
...
...
...

Figure B.3: popup window design that displays data for a specific country in the search bar.

Figure B.4 shows the country ranking. There can be three piece of information for every row. The country, a bar and a value.



CANADA	12345
US	1234
MEXICO	123
...	
...	
...	
...	
...	

Figure B.4: initial country ranking design.

The options tab should allow the user to toggle between which type of data to display. See figure B.5.

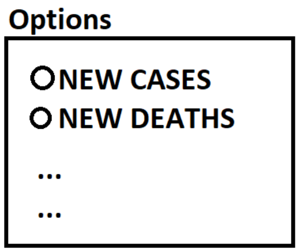


Figure B.5: options design.

The play statistics scrollbar should have a play button and a draggable video-playing-like slider. See Figure B.6.

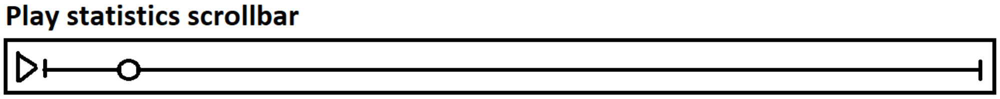


Figure B.6: scrollbar design.

The world map should be colour-coded and shows the coronavirus severity. See Figure B.7.

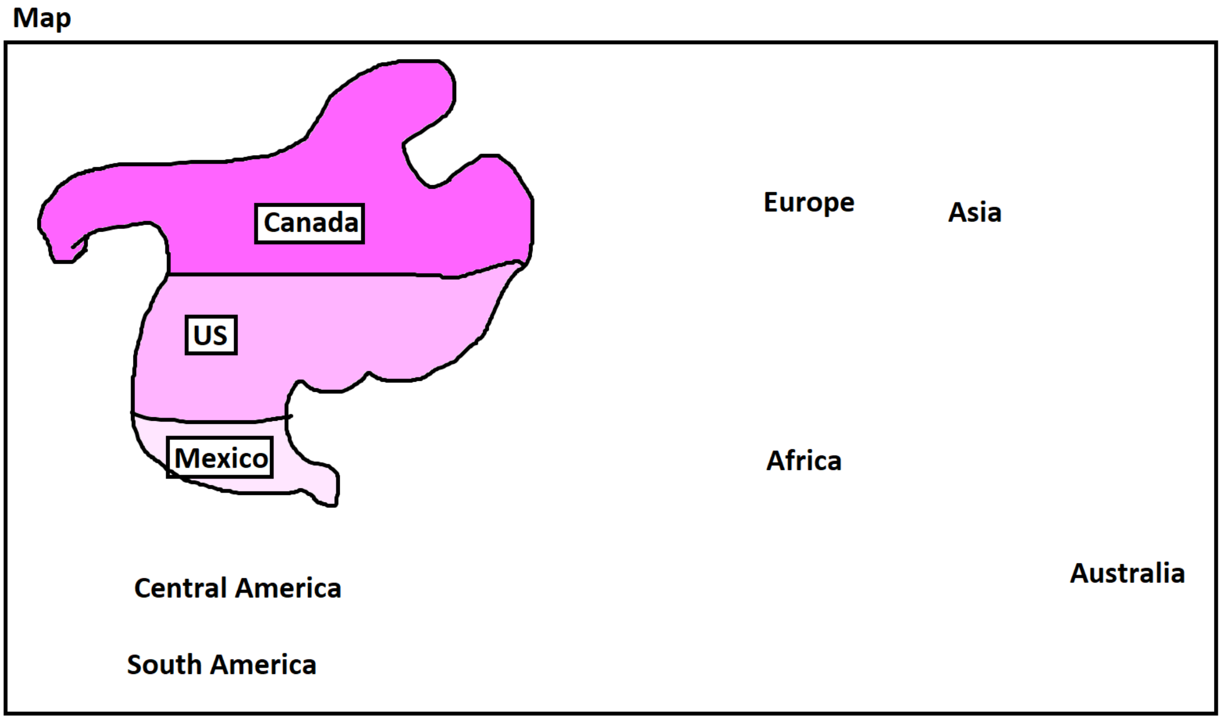


Figure B.7: example map showing countries are coloured.

Flowchart

Figure B.8 shows a simplified flowchart of the setup processes as well as the user interaction processes.

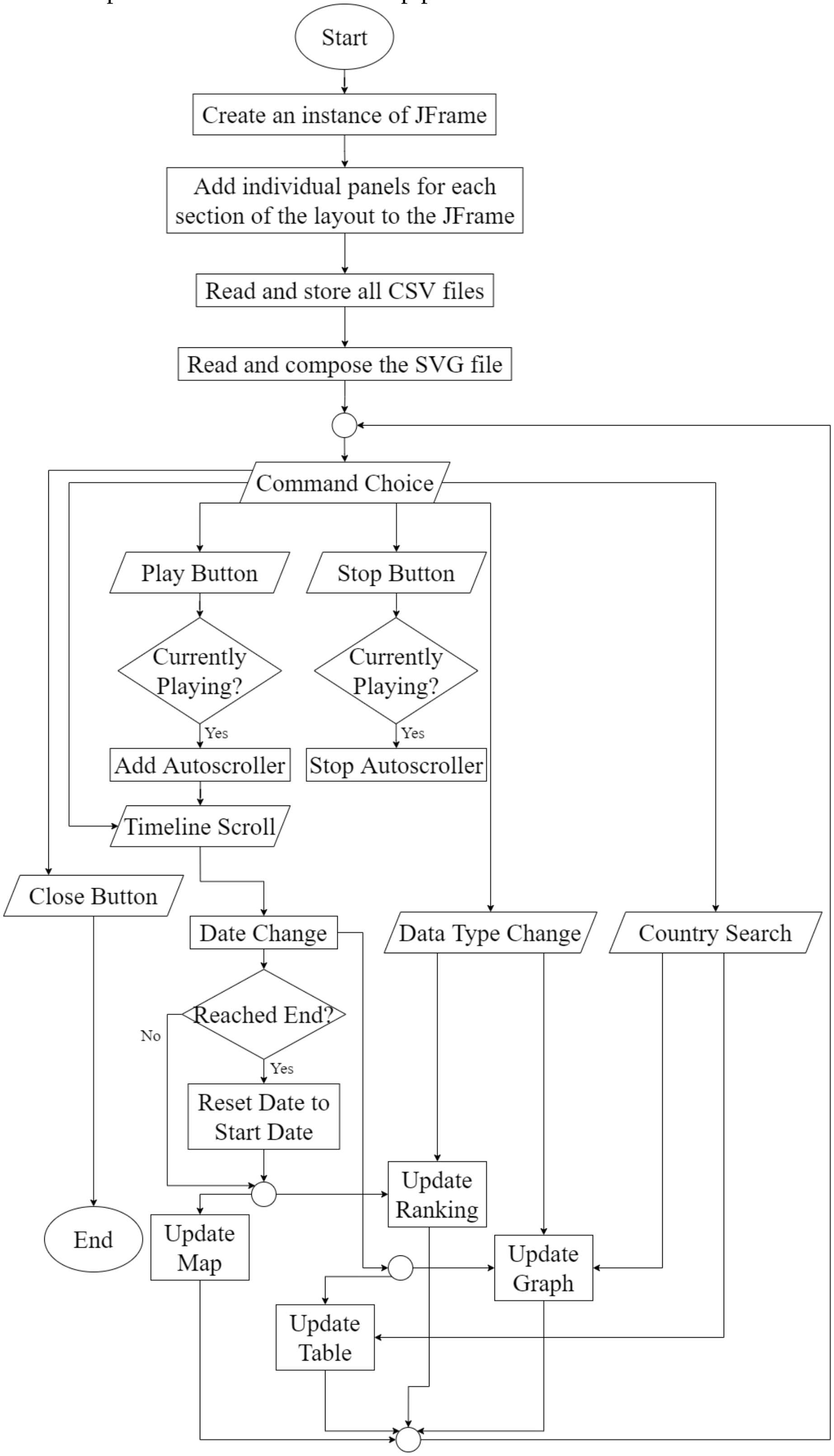


Figure B.8: simplified user setup and user interaction flowchart diagram of the program.

Class

Class Relationship

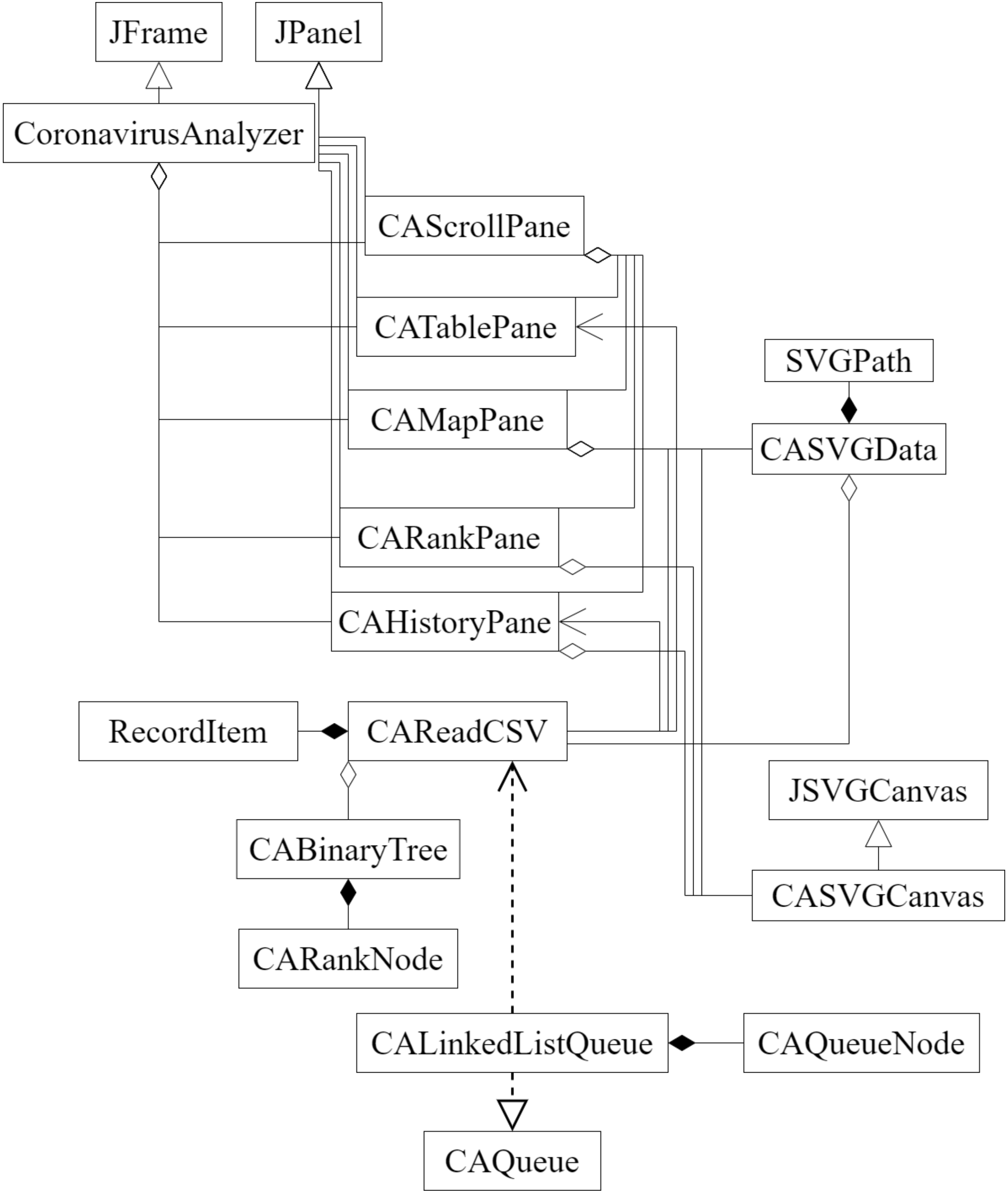


Figure B.9: simplified UML class diagram showing the relationship between classes. The CoronavirusAnalyzer class is the main class that starts executing the program by creating a JFrame instance, thus it inherits JFrame. Each constituent JPanel (i.e., CAScrollPane, CATablePane, CAMapPane, CARankPane, and CAHistoryPane) are added to the CoronavirusAnalyzer JFrame. Each

custom class inherits JPanel and is responsible for containing its respective data. Additionally, there are two main clusters, the CSV cluster and the SVG cluster. The CSV cluster (i.e., CAReadCSV, RecordItem, CABinaryTree, CARankNode, CALinkedListQueue, CAQueueNode, and CAQueue) are all responsible for reading and processing the CSV files. The SVG cluster (i.e., CASVGData, CASVGPath, CASVGCanvas, and JSVGCanvas) are responsible for processing the SVG interactions. Details of each class is presented in full UML class diagrams in the next section.

The relationships between each class is justified in Table B.3 below.

Classes	**Relationship**	**Explanation**
RecordItem, CAReadCSV	Composition	RecordItem contains the statistical values which makes CAReadCSV and RecordItem completely dependent on each other, so it is a composition relationship, that is, CAReadCSV is partially made up of RecordItem.
CABinaryTree CAReadCSV	Aggregation	The binary tree is contained within the CAReadCSV as a private CABinaryTree instance.
CARankNode CABinaryTree	Composition	CABinaryTree is made up of CARankNode's. Without each other they cannot exist.
CALinkedListQueue CAReadCSV	Dependency	The definition of CALinkedListQueue can cause changes to CAReadCSV CALinkedListQueue is temporarily useful to CAReadCSV.
CALinkedListQueue CAQueue	Implementation	CALinkedListQueue is an implementation of the CAQueue interface. The benefit of this system is it allows potential for further development and easy maintenance should the future user wish to make changes.
CAQueueNode CALinkedListQueue	Composition	CALinkedListQueue is composed with CAQueueNode, without it, CALinkedListQueue cannot exist.
CAReadCSV CASVGData	Aggregation	CASVGData holds instances of CAReadCSV while both classes can exist should one class be deleted.
CASVGData CAMapPane	Aggregation	Likewise, CAMapPane holds instances of CASVGData and can also exist without the presence of each other so this relationship is also aggregation.
SVGPath CASVGData	Composition	CASVGData is entirely made up of SVGPath's.
CASVGCanvas &(CAMapPane, CARankPane, CAHistoryPane)	Aggregation	Since the SVG needs to be composed in the map, ranking, and history graph panels the CASVGCanvas and CAMapPane, CARankPane, and CAHistoryPane are all aggregation relationships. The classes are necessary to function together but deletion of one can still allow the other to function.
CASVGCanvas JSVGCanvas	Inheritance	CASVGCanvas inherits JSVGCanvas because CASVGCanvas has custom features.
CAReadCSV &(CATablePane, CAHistoryPane)	Association	CATablePane, CAMapPane, and CAHistoryPane are all associated CAReadCSV because they require the CSV processed data to process and finally compose the SVG.
CAReadCSV & CAMapPane	Aggregation	CAReadCSV and CAMapPane are needed to function together but can be independent if one is deleted.
(CATablePane, CAMapPane, CARankPane, CAHistoryPane	Aggregation	A change in the scrollbar should update the history, rank, map, and table. CAScrollPane holds a collection of instances of CATablePane, CAMapPane, CARankPane, and CAHistoryPane.

) & CAScrollPane		
(CAScrollPane, CATablePane, CAMapPane, CARankPane, CAHistoryPane) & JPanel	Inheritance	Each pane is a special type of JPanel which allows content to be added on to it. Here, the use of OOP because very beneficial as it is much easier and clearer to organize the Panes as a collection of JPanel's.
(CAScrollPane, CATablePane, CAMapPane, CARankPane, CAHistoryPane) & CoronavirusAnalyzer	Aggregation	CoronavirusAnalyzer is the main class that holds the collection of instances of CAScrollPane, CATablePane, CAMapPane, CARankPane, and CAHistoryPane. Additionally CoronavirusAnalyzer is a JFrame that contains the JPanels of each of the separate panes.
CoronavirusAnalyzer JFrame	Inheritance	CoronavirusAnalyzer is a special type of JFrame that will contain all the panes that are special types of JPanels.

Table B.3: justification of UML class diagram relationships.

Class Structure – UML Class Diagrams

CoronavirusAnalyzer
- frame: CoronavirusAnalyzer - scrollBar: CAScrollPane - map: CAMapPane - ranking: CARankPane - dataTable: CATablePane - history: CAHistoryPane - bottom: JPanel - tfCountry: JTextField - WINDOW_WIDTH: int - WINDOW_HEIGHT: int - LABEL_TOTAL_CASES: String - LABEL_NEW_CASES: String - LABEL_TOTAL_DEATHS: String - LABEL_NEW_DEATHS: String
+ CoronavirusAnalyzer() + main(args: String[]): void - run(): void - updateDetail(): void - updatePresentationType(type: int): void

UML Diagram 1: Main class to run the Coronavirus Analyzer program.

CAScrollPane
- JSlider timeline - CAMapPane map - CATablePane dataTable - CAHistoryPane history - CARankPane ranking

- SCHEDULER: ScheduledExecutorService - handleOfPlay: ScheduledFuture<?>
+ CAScrollPane() + setDisplay(map: CAMapPane, table: CATablePane, history: CAHistoryPane, ranking: CARankPane): void + play(): void + stop(): void

UML Diagram 2: Organizes the scrolling of the timeline. Overall the timeline affects the ranking, map, history graph, and data table.

CAMapPane
- svgCanvas: CASVGCanvas - svg: CASVGData + covid: CAReadCSV - currentDay: int - presentationType: int
+ CAMapPane() - loadDocument(): Document + getCovidNumberOfDays(): int + createComponents(): void + updateDate(value: int): void - refresh(): void + updatePresentationType(type: int): void

UML Diagram 3: Loads and processes the SVG world map file and composes SVG map when requested.

CARankPane
- svgCanvas: CASVGCanvas - ranking: List<String> - rankingType: String - PANEL_WIDTH: int - PANEL_HEIGHT: int - COUNTRIES: int - BAR_WIDTH: int - BAR_HEIGHT: int - BAR_SCALE: int - TOP_MARGIN: int - STATISTIC_ALIGN: int - BAR_ALIGN: int - COUNTRY_ALIGN: int
+ CARankPane() + updateDate(covid: CAReadCSV): void - refresh(): void

UML Diagram 4: Sets up the ranking panel on the left. This includes updating and composing SVG to draw.

CATablePane
- tableModel: DefaultTableModel - table: JTable - scrollPane: JScrollPane
+ CATablePane() + updateDate(covid: CAReadCSV, country: String): void + updateDate(covid: CAReadCSV): void - scrollToRow(row: int): void

UML Diagram 5: Sets up the table at the bottom right of the frame. Autoscroll to current day entry.

CAHistoryPane
- svgCanvas: CASVGCanvas - history: TreeMap<String, Integer> - country: String - currentDay: String - GRAPH_WIDTH: double - GRAPH_HEIGHT: double
+ CAHistoryPane() + updateDate(covid: CAReadCSV, country: String): void + updateDate(covid: CAReadCSV): void - refresh(): void

UML Diagram 6: Sets up the graph and updates according to changes in day, desired data type, and country.

CASVGCanvas
- svgScale: short - svgPadding: int - type: int
+ CASVGCanvas() + setType(type: int): void # calculateViewingTransform(svgElementIdentifier: String, svgElement: SVGSVGElement): AffineTransform + setSvgScale(svgScale: short): void + setSvgPadding(svgPadding: int): void

UML Diagram 7: Sets up scaling and ratios for SVG drawing.

SVGPath
+ id: String + name: String + cls: String

+ d: String

UML Diagram 8: Class to organize SVG data.

CASVGData
- mPaths: ArrayList<SVGPath>
+ addPath(): int + setId(i: int, id: String): void + setD(i: int, d: String): void + setClass(i: int, cls: String): void + setName(i: int, name: String): void + getSize(): int + getId(i: int): String + getName(i: int): String + getD(i: int): String + getClass(i: int): String

UML Diagram 9: Stores extracted SVG data.

CAReadCSV
+ data: TreeMap<String, TreeMap<String, RecordItem>> + currentEntry: Entry<String, TreeMap<String, RecordItem>> - rankOfConfirmed: CABinaryTree + rankList: List<String> + mPresentationType: int
- getFilenames(): CALinkedListQueue - loadFileFromURL(filename: String): void + loadDataFiles(): void + getCountryName(country: String): String + getCurrentDate(): String + updateRankOfConfirmed(nDay: int, presentationType: int): void + getCountryData(country: String, cData: int[]): boolean + refreshTable(model: DefaultTableModel, country: String): void + refreshHistory(history: TreeMap<String, Integer>, country: String): void + quickSort(data: List<String>, start: int, end: int): void

UML Diagram 10: Read, process and store CSV file data. Data manipulation features such as generating ranking upon request by sorting data.

RecordItem
~ confirmed: int ~ increased: int ~ deaths: int ~ newDeaths: int
+ RecordItem(c: int, d: int)

+ setPrevious(c: int, d: int): void + getValue(presentationType: int): int

UML Diagram 11: Organizes and calculates the four data types to be stored.

CABinaryTree
- rootPointer: CARankNode
+ CABinaryTree() + insert(country: String, rank: int): boolean + CARankNode search (country: String)

UML Diagram 12: Implements a Binary Tree to organize country and data ranking.

CARankNode
- country: String - rank: int - left: CARankNode - right: CARankNode
+ CARankNode(country: String, rank: int) + setLeft(left: CARankNode): void + setRight(right: CARankNode): void + getLeft(): CARankNode + getRight(): CARankNode + getCountry(): String + getRank(): int

UML Diagram 13: Node class for CABinaryTree.

«interface» CAQueue
+ isEmpty(): boolean + dequeue(): Object + enqueue(item: Object): boolean

UML Diagram 14: Interface for a custom queue.

CALinkedListQueue
- front: CAQueueNode - rear: CAQueueNode
+ CALinkedListQueue() + isEmpty(): boolean + dequeue(): Object + enqueue(item: Object): boolean

UML Diagram 15: Implementing the interface CAQueue to create a linked list queue.

CAQueueNode
- value: Object - next: CAQueueNode
+ CAQueueNode(value: Object) + setValue(value: Object): void + setNext(next: CAQueueNode): void + getValue(): Object + getNext(): CAQueueNode

UML Diagram 16: Node class for the CALinkedListQueue class.

Testing Plan

Success Criteria	**Testing Method**
Intuitive and minimalistic single-window layout displaying coronavirus data quantitatively and qualitatively.	Run the program and click the play button to show that all components are functional and updates accordingly. Check that the layout contains clearly labelled panels to show intuitive and minimalistic. Check if table loaded which is quantitative data. Check that the map, graph, and ranking loaded for qualitative data.
Presentation of four types of daily and national coronavirus data. Total cases, new cases, total deaths, and new deaths.	Click the four radio buttons one by one to switch presentation type. Check if panels respond to daily changes to show daily covid data. Check if map colours change and countries are solid colours to show national covid data.
Colour-coded, titled, and date-labelled map showing the severity of coronavirus in each country.	Click the play button. Check map colour change. Check more severe countries in map are darker colors. Check for successful title and date-label creation. These combine to form the severity in each country.
The map should update based on a change in date.	Click, drag, and drop the slider to a new date. Or, press the play button. Check that the map changes.
Date changing should be quick and automatable.	Click, drag, and drop with speed to show the date changing is quick. Click the play button to demonstrate automatability.
The map should display all four types of data individually and updates accordingly.	Click the four radio buttons one by one to switch between the four types of data. Check the map updates and is different for all four button clicks.
Program contains a ranking that shows the top 20 most severe countries of a data type.	Click the play button. Compare the color on the map with the top 20 countries in the ranking to verify that data is consistent, thus demonstrating a top 20 ranking is present.
This ranking should be titled and labelled in terms of which of the four types of data is currently on display.	Click all four radio buttons one by one. Check that the label under rankings changes. Check if there is a ranking title.

For each entry in the ranking, there should be three pieces of information displayed. Country name. Statistical value corresponding to the currently displayed data. Bar that represents the relative severity between countries in the top 20 rankings.	Click the play button. Check if country, statistical value, and a bar is present for the top 20 countries. Click the stop button. Check if the bar lengths are relatively accurate by calculating the ratio between the statistical values then estimating the approximate length ratio of the bars.
More details about a country should be accessible should the user wish to look more into a country. Specifically, there should be a graph of the all-time historical values of a data type for a country.	Click the play button or scroll to any place in the timeline. Check if the top right corner contains the graph of the historical values for a country for a specific data type. Click each of the four radio buttons. Check if the graph updates.
There should be some indication of where the current day on the graph is.	Click the play button to check if the current-day-circle moves along the graph of the data. Or, click, drag, and drop the slider and check if circle changes position.
There should also be a quantitative aspect in which the numerical data for a country is presented in a table.	Click the text field. Type any country. Press enter or click details button. Check if table exists in bottom right corner.
Like the graph, there should be some indication of the present-day value in the table.	Click the play button. Check for a row highlight to verify a change in present-day value. Scroll to anywhere in the table to check if the highlight jumps back to current day. Click anywhere in the table to check if the highlight jumps back.
As a result, there should be a place to search for a country by its name.	Click the text field. Type another country. Press enter or click the details button. Check if rankings and table updates to see that this feature works.

Table B.4: testing plan for the program. See Criterion A for success criteria in indented bullet format.