Design Specification

Project: Threat Detect

Group 2: Harsh Patel, Michael Barreiros, Ben Li,

Mustafa Choueib, Torja Istiaque

McMaster University, CAS Department

11971: SFRWENG 2XB3

April.13.19

Table of Contents

Revision History & Team Data

Date	Revisions
Mar.21.19	Created Document Template
Mar.25.19	Updated group roles
April.11.19	Rest of report completed

Name & Student Number	Role(s)
Harsh Patel (400132010)	Designer, Developer, Debugger, Meeting Minutes
Michael Barreiros (400124043)	Designer, Leader, Developer
Mustafa Choueib (400135076)	Designer, Developer, Debugger
Ben Li (400156109)	Designer, Developer
Torja Istiaque (400081773)	Designer, Developer, Project Logger/Log Admin

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Contributions & Summary

Name	Role	Contribution
Harsh Patel (400132010)	**Designer, Developer, Debugger, Meeting Minutes**	- Completed States and Incidents ADT - Meeting Minutes for first 4 meetings - Code for functions buildIncidents(), buildBST(), SortIncidents(), filterIncidents(), and reading from User Input of ReadAllocation.java - Debugging ReadAllocation.java for issues arising when reading from Dataset such as commas within strings, unknown characters in dataset, and incidents with missing data - Design Spec components of Module Decomposition and Class Descriptions(Incidents, States, and QuickSort) section
Michael Barreiros (400124043)	**Designer, Group Leader, Developer**	- Completed Quicksort ADT, and RedBlackTree ADT - Design Spec components of Abstract, and Class Descriptions(ReadAllocation, and RedBlackTree) sections - Code for function read() in ReadAllocation.java and added while loop for continuous prompt for user input till user terminates with a 0 - Debugging ReadAllocation.java for user input in the main function, and dealing with random blank lines in the dataset
Mustafa Choueib (400135076)	**Designer, Developer, Debugger**	- Completed the graphical user interface (GUI) code - Implementation of code of the GUI to work with the main function of ReadAllocation.java and add functionality to components of GUI such as buttons, input box, output box, and user input - Note: GUI was implemented but not used as it was too slow and not efficient as the console operation, and the time constraint to test and make changes to GUI was tight.

		- Design Specification component of the Internal Review section. - Meeting minutes for the entirety of week 4.
Ben Li (400156109)	**Designer, Developer**	- Redesigned Requirement specification - Helped with some logic behind program code - Contributed to initial idea and discussion of project
Torja Istiaque (400081773)	**Designer, Developer, Project Logger/Log Admin**	- Geocoding functions and API from Google to convert string address to longitude and latitude - Project logger - Code responsible for writing to the out text file in the main() function of the ReadAllocation.java module. - All UML sections and Traceability matrix of the design spec - All administrative work for group

Abstract

Incidents of gun violence in the United States has been increasing throughout the years. Over the past 5 years, from 2013-2018, there have been over 250,000 unique incidents of gun violence in the United States alone. Finding a safe place to move to is a necessity for many Americans. Threat Detect puts the power of knowledge in the hands of Americans. American users can input a desired location, for example the location of a home to be moved into, and see how dangerous the surrounding area is. After the desired location is inputted, that location is given an appropriate “Danger Level” based upon number of incidents, injuries, and killed. The user will also be shown all the incidents with the appropriate details/information, including a total of kills, injuries, and incidents. Threat detect takes advantage of a dataset containing over 250,000 gun violence incidents to give reliable information to the user.

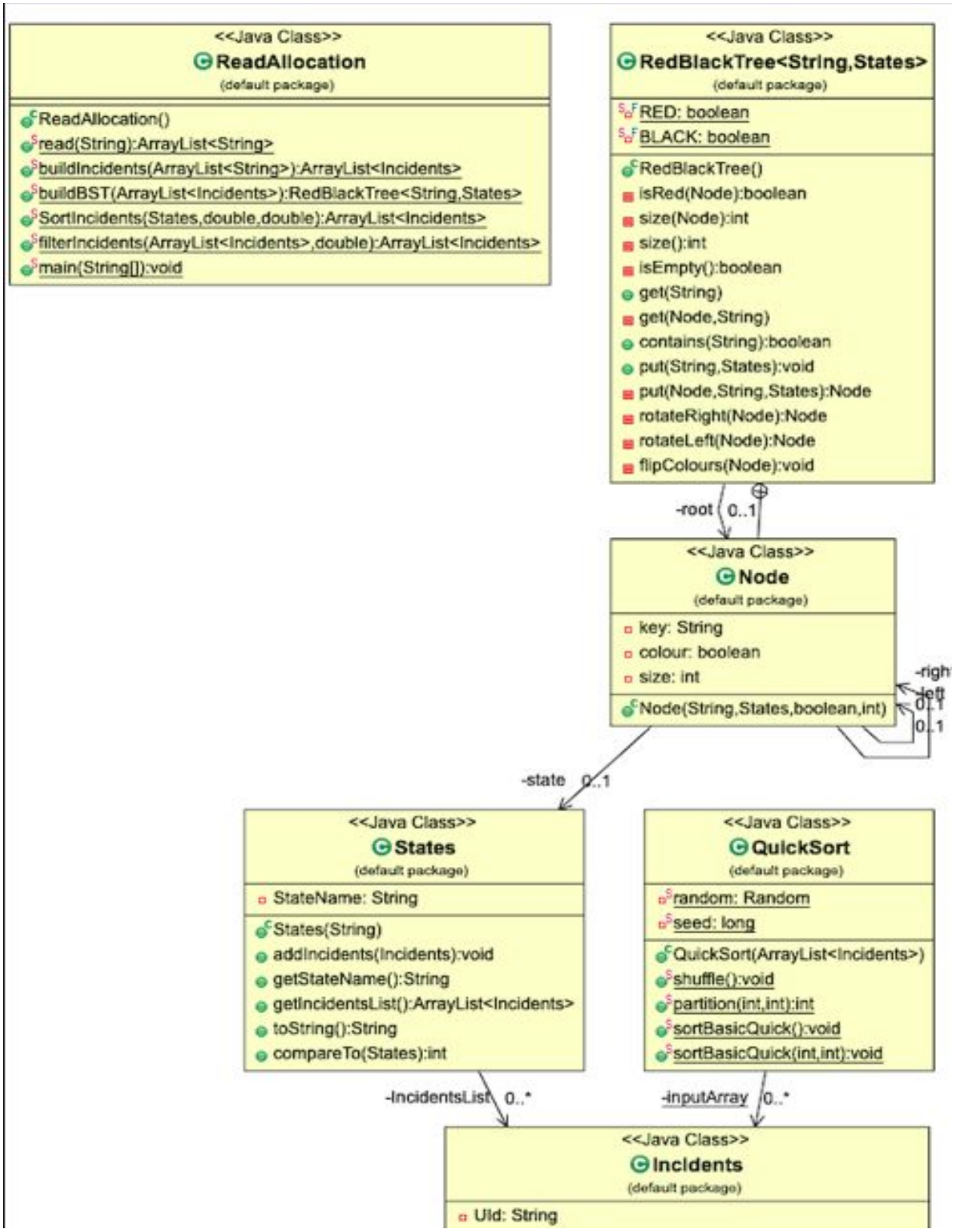
Module Decomposition

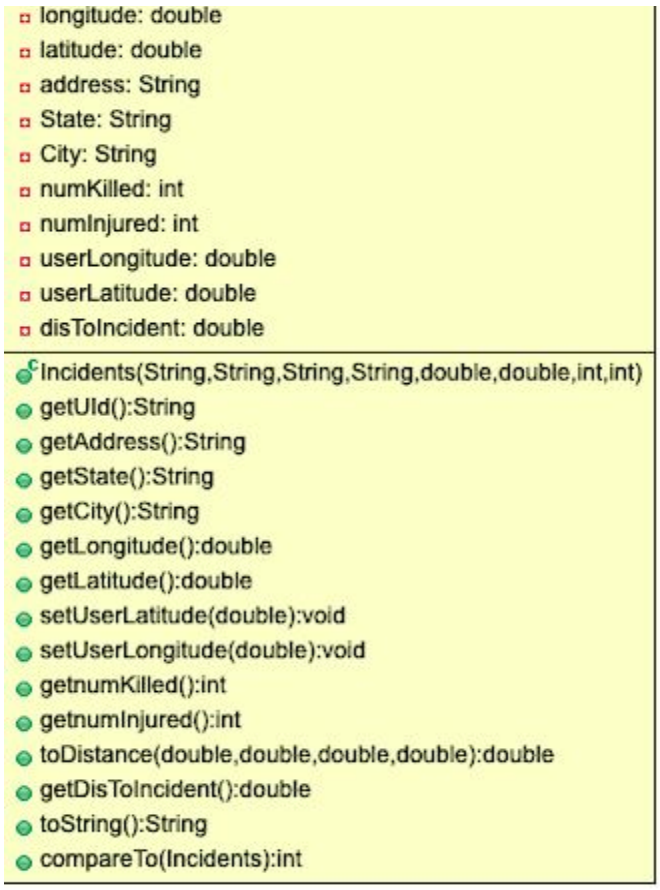
Note: Geocoding package decomposition not provided as it is taken from Google Maps API

Module	Reasoning
ReadAllocation	This module includes the functions/methods to read from the dataset, and initialize the Red-Black Balanced BST storing all information about US States and their corresponding incidents. There are also methods that search, sort, and filter incidents based on user input of a reference location and the distance constraint set by the user to only view gun incidents within a certain range of this reference location. This module includes the main function that executes the functionalities of the application and prompts for user input.
Incidents	This module includes the Incident ADT which stores relevant data related to the gun violence incident from the CSV file dataset for each incident that has occured.
RedBlackTree	This module includes the ADT for a balanced Binary Search Tree which stores a State ADT as a node, organized lexicographically in the graph-tree structure for efficient searching and inserting time.
States	This module includes the State ADT which stores the name of one of the 50 States in the USA and an Array List of the corresponding gun incidents for the State.
QuickSort	This module is solely responsible for sorting the gun incidents based on distance from the reference location set by the user to the location

	that the incident took place from closest to farthest in kilometers.

Static UML Diagram





*Note that Incidents class diagram spans two pages

Traceability Matrix

Threat Detect - Traceability Matrix

Project Name: Threat Detect
Group Number: 2

Purpose: To manage requirements throughout the software development lifecycle. The Traceabilty Matrix ensures that requirements are captured in the design implemented in the code, and verified by testing.

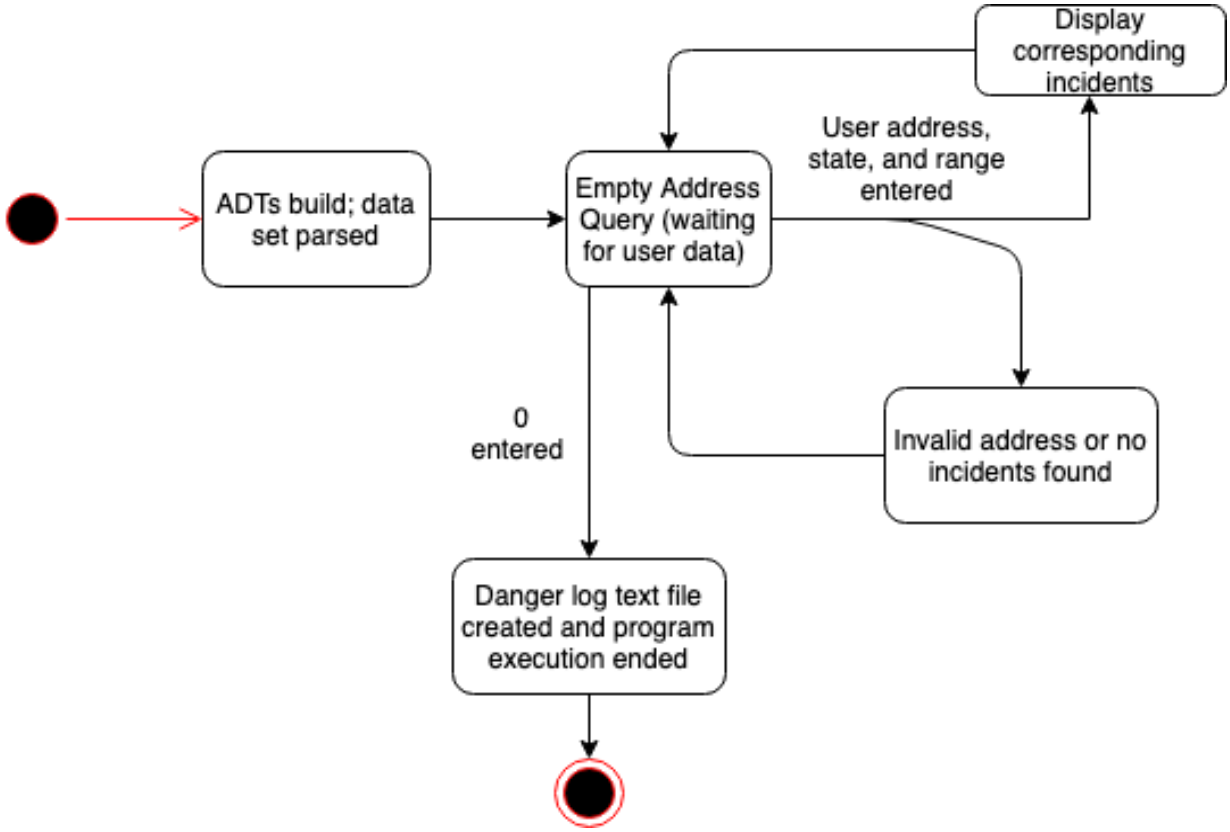
Use Case	Requirement Number	Requirement Description	Status	Release	Test Case	Design Components / Modules
Reading and storing of input data	1	Read and store incidents of gun violence from a file. The relevant parameters for each incident will contain city name, longitude/latitude, number of wounded, and number of casualties.	Verified	Final	Compared input paramaters of csv file to those filtered by program and handeling of no file	ReadAllocation.java, Incidents.java, ReadBlackBST.java, States.java
Filter data via user's location	2	Take a location, determined by the user, as input and determine the region to which the location belongs.	Verified	Final	Compared address entered by user and filtered location output to google maps ensuring all outputs are geographically correct, also tested for addresses not inside data set	ReadAllocation.java, States.java
Filter data via user's selected range	3	Determine which of the incidents are found within a certain distance (which the user should also specify) of the location.	Verified	Final	Compared filtered ouputs with google map distances to ensure correct proximity to user's location, ensured that 0 km range gives no output, compared sorted list of outputs with previous correctly sorted list	ReadAllocation.java, Incidents.java, ReadBlackBST.java, States.java, QuickSort.java

Traceability Matrix Cont..

Determine Threat Levels for specific locations	4	Calculate a severity index with a formula based number of killed and number of injured using the values of incidents within range of the desired location. Assign the threat level that corresponds to the calculated severity index based on the determined range of threat levels.	Verified	Final	Compared threat levels to general preceived reviews of users location based on online resources, ensured accurate caculations by comparing it to previous hand calculated ones	ReadAllocation.java
Relevant data is displayed to user	5	Display the list of incidents and threat level corresponding to the area around the desired location. Also, display the associated threat level.	Verified	Final	Ensured data corresponding to output is correct by comparing it to original input data file, ensured correct display to console and text file (correct formatting), tested for case of no incidents and immediate termination	ReadAllocation.java

UML State Machine Diagrams

ReadAllocation.java



Incidents.java

Class Descriptions

ReadAllocation.java		
Name	**Type**	**Description**
public		
main(String args[])	void	Prompts user to enter input in form of address, US State, and range which initializes the RedBlackTree and gets the ArrayList of incidents, sorts the list, filters the ones within the given range and outputs associated information.
buildIncidents(ArrayList< String> data)	ArrayList<Incidents>	Given an ArrayList of strings build a new ArrayList of Incidents that are initialized with the correct information from the data ArrayList.
buildBST(ArrayList <Incidents> incidents)	RedBlackTree<String, States>	Builds a RedBlackTree from the incidents list that originates from the dataset.
SortIncidents(State state, double userLong, double userLat)	ArrayList<Incidents>	Searches for a state in the BST and updates the contents of its state value with a sorted version of this based on the distance from user location.
filterIncidents(Array List<Incidents> InRangeIncidents, double Range)	ArrayList<Incidents>	Filters an incoming ArrayList of Incidents by a given range. A new filtered list gets returned.

Incidents.java		
Name	**Type**	**Description**
private		
UID	String	Unique string Id associated with each gun violence incident
longitude	double	Longitude coordinate of the gun violence incident location
latitude	double	Latitude coordinate of the gun violence incident location
address	String	Specific address of the gun violence location
State	String	US State of the gun violence incident location
City	String	City of the gun violence incident location
numKilled	int	Number of people that were killed in the gun violence incident
numInjured	int	Number of people that were injured in the gun violence incident
userLongitude	double	Longitude coordinate of reference location provided by user input
userLatitude	double	Latitude coordinate of reference location provided by user input

disToIncident	double	Distance from reference location provided by user input to the
public		
Incidents(String UId, String address, String State, String City, double longitude, double latitude, int numKilled, int numInjured)	Incidents	Incidents constructor using gun violence incident address, US State location, longitude coordinate, latitude coordinate, number of people killed, and number of people injured
getUId()	String	Returns the unique ID of the associated with the incident
getAddress()	String	Returns the address location of the incident
getState()	String	Returns the US State location of the incident
getCity()	String	Returns the City location of the incident
getLongitude()	double	Returns the longitude coordinate location of the incident
getLatitude()	double	Returns the latitude coordinate location of the incident
setUserLatitude(double userLatitude)	void	Sets the latitude coordinate location of the reference address from the user input
setUserLongitude(double userLongitude)	void	Sets the longitude coordinate location of the reference address from the user input
getnumKilled()	double	Returns number of people killed in the incident

getnumInjured()	double	Returns number of people injured in the incident
toDistance(double latitude, double longitude, double userLatitude, double userLongitude)	double	Returns the calculated distance from incident to the reference location from the user input in kilometers and assigns it to the variable disToIncident
getDisToIncident()	double	Returns the value stored in the disToIncident variable
toString()	String	Returns the string format of the incident information to be read by the user

RedBlackTree.java		
Name	**Type**	**Description**
private		
RED	boolean	A boolean flag to tell us that a node is a red branch
BLACK	boolean	A boolean flag to tell us that a node is a black branch
root	Node	A node that is the root of the tree
Node class	Node	An ADT that has a String as its key, a States object as its value, a boolean colour to signify what colour branch it is, and a size to tell us how many children it has
isRed(Node x)	boolean	Returns whether or not the branch of node x is red

size(Node x)	int	Returns the size associated with a Node x
size()	int	Returns the size associated with the root Node, this is the size of the whole tree
isEmpty()	boolean	Returns whether or not the tree is empty
get(Node x, String key)	States	Returns the state found when searching for a key from a starting node, if not found return null
put(Node h, String key, States state)	Node	Puts the <String, State> pair onto the BST in the proper position. Fixes the tree after a node is place
rotateRight(Node h)	Node	Rotates the node right, used to rebuild the tree to be a left-leaning red black tree
rotateLeft(Node h)	Node	Rotates the node left, used to rebuild the tree to be a left-leaning red black tree
flipColours(Node h)	void	A function to flip the colours of a node, used to rebuild the tree to be a left-leaning red black tree
public		
RedBlackTree()	RedBlackTree	Constructor to initialize an empty symbol table
get(String key)	States	Returns a state when given a key
contains(String key)	boolean	Returns whether a key is in the tree

put(String key, States state)	void	Puts the <String, state> pair into the BST

States.java		
Name	**Type**	**Description**
private		
StateName	String	Name of the US State
IncidentsList	ArrayList<Incidents>	List of incidents associated with the this US State based on location
public		
States(String StateName)	States	States constructor using the State name
addIncidents(Incidents incident)	void	Adds incidents of the corresponding State to the IncidentsList
getStateName()	String	Returns name of the US State
getIncidentsList()	ArrayList<Incidents>	Returns list of incidents to the corresponding US State location
toString()	String	Returns string format of the State name
compareTo(States that)	int	Returns an integer to represent if the State name is less than, greater than, or equal to another State's name alphabetically when comparing.

QuickSort.java		
Name	**Type**	**Description**
private		
inputArray	ArrayList<Incidents>	List of Incidents ADT
random	Random	Creates a random object to shuffle the array
seed	long	Number to be used when producing the random
public		
QuickSort(ArrayList<Incidents> inputArray)	QuickSort	QuickSort constructor initialized with input Array of Incidents ADT
shuffle()	void	Shuffles the list of Incidents ADT
partition(int lo, int hi)	int	Returns an index to partition at for the list of Incidents ADT when calling Quicksort
sortBasicQuick()	void	Shuffles the array first then calls the recursive sortBasicQuick function that uses 0 as its lower bound and the size of the (array - 1) as its upper bound
sortBasicQuick(int lo, int hi)	void	A recursive basic quicksort that partitions an element then performs quicksort on sub-array to the left of the partitioned element and then to the left

Internal Review

The overall feasibility of the program was achieved and implemented correctly. The accuracy and efficiency of the program certainly stands out. Through a series of different tests, we were able to ensure that the reliability, robustness, and correctness of the algorithm were optimized. Given a certain input, the program will produce the same respective output after each execution, it is also able to handle unusual input through exception handling and code that operates as intended, and also provides the correct output for any properly given input. These three aspects are vital for the success of the program, thus, maintaining these properties was a priority we upheld. Another aspect in which we managed to enhance was the performance of the program. The runtime of the program should be reasonable respective to the size of the output that is being displayed. Through the use of efficient sorting and searching algorithms, we managed to return the correct output (smaller and larger output) in mere seconds. Having high speed performance is essential to maintain user satisfaction.

One aspect that could have been improved was the implementation of the graphical user interface. A graphical user interface (GUI) was created, however, we found that the interface was much slower in terms of run time. Despite all our efforts to make the GUI as quick as running the program on the console, we were unable to do so. In essence, we came to a decision to abandon the GUI as we held the speed/performance of our program to high standards, and wanted to ensure the user could receive the output that was desired in a few seconds. Through collaboration, we decided that we could write the respective output to a text file that would be available to the user.

The connectivity and communication amongst a group is vital to success of a project. Our group managed to communicate at a high degree in order to ensure the project would be completed. Each individual member completed their respective parts in a timely manner and maintained the created schedule. The inclusion and teamwork of the group allowed for each member to excel and perform to the best of their abilities. Hence, the atmosphere was a huge benefit when completing the project.

The entirety of the program was successful. We managed to implement a fully functioning algorithm that met all the functional and nonfunctional requirements that we were looking to prioritize. The idea behind Threat Detect is also very relevant and useful. We wanted to ensure that the overall project would be one of use, opposed to a algorithm that is essentially pointless. Hence, we made sure we all provided effort that goes above and beyond the common expectation in order to create something fulfilling.