Safety Net

Version 1.4

McMaster University
COMPSCI 2XB3 L03

Group 2: Suleyman Kiani, Sunny Bhatt, Nikola Milanovic, Senan Gohar

April 13, 2020

Summary

NYC Safety Net for Tourists and Visitors is a desktop application that maps crime data to different regions of New York City, using NYPD arrest data of approximately four million crimes. Users can filter for specific crimes in designated areas of NYC to learn about safety with respect to certain crimes. The application assists tourists travelling to NYC by informing them safe areas to stay, ultimately helping them save money by staying in safe areas outside of Manhattan. It also provides valuable information to Real Estate Investors, as they can analyze crime data in areas and evaluate likelihood of investments to flourish due to gentrification, etc. Features of the application include comparing crime frequency and types of crime between the bureaus of NYC, comparing crime during different time periods, getting crime data at a specific address, and finding the safest path between two areas of the city.

Revisions

Revision history

1.1

Switched to using OpenCSV to parse the crime dataset, improving performance.

1.2

User interface improvements, now displaying frequency of crimes and limiting results to top 5 or 10 with an option to display all results.

1.3

Introduction of the location lookup feature, utilizing a geocoding API from LocationIQ.

1.4

Introduction of the safest path feature, implementing Dijkstra's shortest path algorithm.

Consent

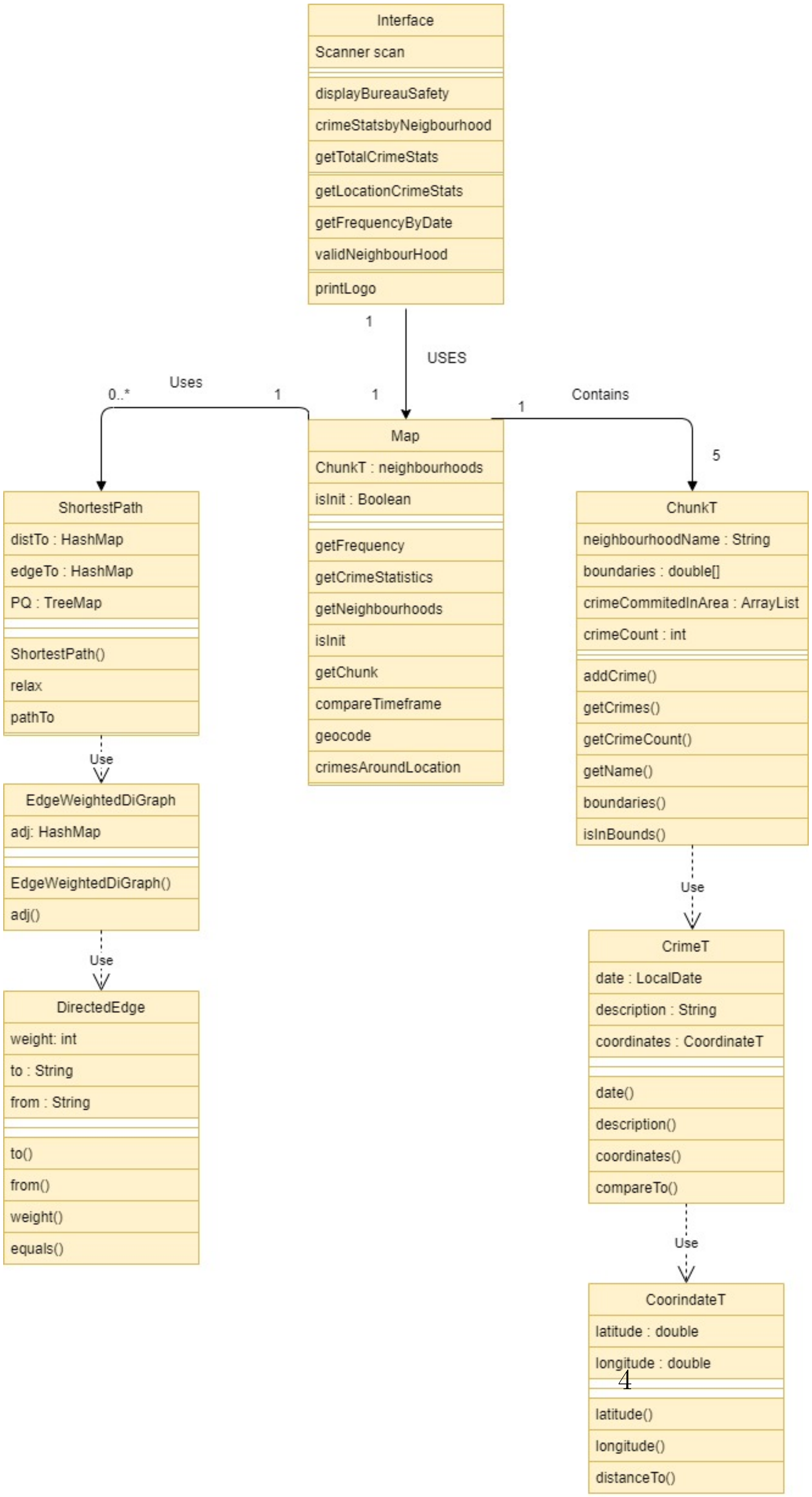
By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through CS2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Sunny Bhatt: 400190373
Suleyman Kiani: 400189668
Senan Gohar: 400131114
Nikola Milanovic: 400171310

Contributions

Name	Role(s)	Contributions
Sunny Bhatt	Developer Tester	CSV parsing, location crime lookup, unit tests, Map MIS
Nikola Milanovic	Team leader Developer Maintainer	ADT's, Map, Interface, ShortestPath, UML, Design
Suleyman Kiani	Developer	Module design, Application Design, MIS
Senan Gohar	Developer	User interface, Java documentation, architecture design

Application level UML diagram



Coordinate ADT Module

Module

CoordinateT

Uses

N/A

Syntax

Exported Constants

None

Exported Types

CoordinateT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
CoordinateT	$\mathbb{R}$, $\mathbb{R}$	CoordinateT	
latitude		$\mathbb{R}$	
longitude		$\mathbb{R}$	
distanceTo	CoordinateT	$\mathbb{R}$	

Semantics

State Variables

lat: $\mathbb{R}$
lon: $\mathbb{R}$

Considerations

The constructor CoordinateT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

CoordinateT(lat, lon):

- transition: $lat, lon := latitude, longitude$
- output: $out := self$
- exception: None

latitude():

- output: $out := lat$
- exception: None

longitude():

- output: $out := lon$
- exception: None

distanceTo(c):

- output: Great-circle distance from self to c using the haversine formula
- exception: None

Crime ADT Module

Template Module

CrimeT

Uses

CoordinateT

Syntax

Exported Types

CrimeT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
CrimeT	LocalDate, String, CoordinateT	CrimeT	
date		LocalDate	
description		String	
coordinate		CoordinateT	
compareTo	CrimeT	$\mathbb{Z}$	

Semantics

State Variables

$date$: LocalDate
$description$: String
$coordinates$: CoordinateT

Access Routine Semantics

CrimeT$(d, desc, c)$:

- transition: $date, description, coordinates := d, desc, c$
- output: $out := self$
- exception: None

date():

- output: $out := date$
- exception: None

description():

- output: $out := description$
- exception: None

coordinate():

- output: $out := coordinates$
- exception: None

compareTo(that):

- output: $out :=$ date().compareTo(that.date())
- exception: None

Chunk ADT Module

Generic Template Module

ChunkT

Uses

CrimeT

Syntax

Exported Types

ChunkT = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
ChunkT	name, $\mathbb{R}$, $\mathbb{R}$, $\mathbb{R}$, $\mathbb{R}$	ChunkT	
addCrime	CrimeT		
getCrimes		seq of CrimeT	
getCrimeCount		$\mathbb{Z}$	
getName		String	
boundaries		seq of $\mathbb{R}$	
isInBounds	CrimeT	$\mathbb{B}$	

Semantics

State Variables

neighbourhoodName: String
boundaries: seq of $\mathbb{R}$
crimesCommitedInArea: seq of CrimeT
crimeCount: $\mathbb{Z}$

Access Routine Semantics

ChunkT($name, left, right, up, down$):

- transition: $neighbourhoodName := name$
$boundaries[0], boundaries[1], boundaries[2], boundaries[3] = left, right, up, down$
- output: out := $self$
- exception: None

addCrime($crime$):

- transition: $crimesCommitedInArea.add(crime)$
$crimeCount++$
- exception: None

getCrimes():

- transition: None
- output: $out := crimesCommitedInArea$
- exception: None

getCrimeCount():

- transition:
- output: $out := crimeCount$
- exception: None

getName():

- transition: None
- output: $out := neighbourHoodName$
- exception: None

boundaries():

- transition:
- output: $out := boundaries$

- exception: None

isInBounds(crime):

- transition: None
- output: Returns true if crime.coordinates() is located within the four bounds of this chunk
- exception: None

Map Abstract Object Module

Abstract Object

Map

Uses

ChunkT, CrimeT, CoordinateT

Syntax

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
init			
getFrequency	$\mathbb{B}$	seq of String	
getCrimeStatistics	$\mathbb{B}$	map of String to Integer	
getCrimeStatistics	$\mathbb{B}$, String	map of String to Integer	
getCrimeStatistics	$\mathbb{B}$, seq of CrimeT	map of String to Integer	
getNeighbourhoods		seq of ChunkT	
isInit		$\mathbb{B}$	
getChunk	String	ChunkT	
compareTimeframe	LocalDate, LocalDate	seq of CrimeT	
assignCrimeToChunk	CrimeT		
geocode	String	CoordinateT	
crimesAroundLocation	CoordinateT, $\mathbb{R}$	seq of CrimeT	

Semantics

State Variables

$isInit$: $\mathbb{B}$
$neighbourhoods$: seq of ChunkT

Access Routine Semantics

init():

- transition: Initializes NYC's five bureaus to chunks and parses a CSV file of crimes into those chunks.
- exception: None

getFrequency(descending):

- output: Returns a sequence of neighbourhood names sorted by the number of crimes in the corresponding chunks. If descending is true, the sequence is sorted in descending order, else ascending order.
- exception: None

getCrimeStatistics(descending : $\mathbb{B}$):

- output: Returns a map of crime description mapped to frequency of the crime across all neighbourhoods. If *descending* is true, the sequence is sorted in descending order of frequency, else ascending order.
- exception: None

getCrimeStatistics(descending : $\mathbb{B}$, neighbourHoodName : String):

- output: Returns a map of crime description mapped to frequency of the crime in the neighbourhood *neighbourhoodName*. If *descending* is true, the sequence is sorted in descending order of frequency, else ascending order.
- exception: None

getCrimeStatistics(descending : $\mathbb{B}$, crimes : seq of CrimeT):

- output: Returns a map of crime description mapped to frequency of the crime in a given sequence of crimes. If *descending* is true, the sequence is sorted in descending order of frequency, else ascending order.
- exception: None

getNeighbourhoods():

- output: out := neighbourhoods
- exception: None

isInit():

- output: out := isInit
- exception: None

getChunk(neighbourHoodName):

- output: Returns the chunk with name *neighbourHoodName*.
- exception: None

compareTimeframe(start, end):

- output: Returns a sequence of crimes that occured between the dates *start* and *end*.
- exception: None

assignCrimeToChunk(crime):

- transition: Adds *crime* to the chunk its coordinates lie within.
- exception: None

geocode(address):

- transition: Returns a GPS coordinate given *address*, a string description of the location.
- exception: None

crimesAroundLocation(location, radius):

- transition: Returns a sequence of crimes with coordinates within *radius* kilometers of *location*
- exception: None

ShortestPath ADT Module

Template Module

ShortestPath

Uses

EdgeWeightedDiGraph

Syntax

Exported Types

ShortestPath = ?

Exported Access Programs

Routine name	In	Out	Exceptions
ShortestPath()	EdgeWeightedDiGraph, String		
relax	EdgeWeightedDiGraph, String		
pathTo	String	Stack	

Semantics

State Variables

$distTo$: map of String to $\mathbb{R}$
$edgeTo$: map of String to DirectedEdge
PQ: map of $\mathbb{R}$ to String

Access Routine Semantics

ShortestPath(EdgeWeightedDiGraph G, String source):

- transition: Initializes all the state variables, sets all the distances to positive infinity except the source and starts Djkstra's algorithm to find the shortest path.
- output: out := none
- exception: None

relax(EdgeWeightedDiGraph G, String vertex):

- transition: Edge relaxation for the given vertex.
- output: out := none
- exception: None

pathto(String v):

- output: out := A stack of directed edges that represents the path from the source vertex to the provided vertex v if such a path exists.
- exception: None

EdgeWeightedDiGraph ADT Module

Template Module

EdgeWeightedDiGraph

Uses

DirectedEdge

Syntax

Exported Types

EdgeWeightedDiGraph = ?

Exported Access Programs

Routine name	In	Out	Exceptions
EdgeWeightedDiGraph()			
adj		map of String to (seq of DirectedEdge)	

Semantics

State Variables

adj : map of String to (seq of DirectedEdge)

Access Routine Semantics

EdgeWeightedDiGraph():

- transition: Creates all the edges for our graph with all the neighbourhoods.
- output: out := none
- exception: None

adj():

- output: out := adj
- exception: None

DirectedEdge ADT Module

Template Module

DirectedEdge

Uses

None

Syntax

Exported Types

DirectedEdge = ?

Exported Access Programs

Routine name	In	Out	Exceptions
DirectedEdge()	String, String, $\mathbb{Z}$		
to		String	
from		String	
weight		$\mathbb{Z}$	
equals	DirectedEdge	$\mathbb{B}$	

Semantics

State Variables

to : String
from : String
weight : $\mathbb{Z}$

Access Routine Semantics

DirectedEdge(to, from ,weight):

- transition: to, from, weight := to, from , weight
- output: out := none
- exception: None

to():

- output: out := to
- exception: None

from():

- output: out := from
- exception: None

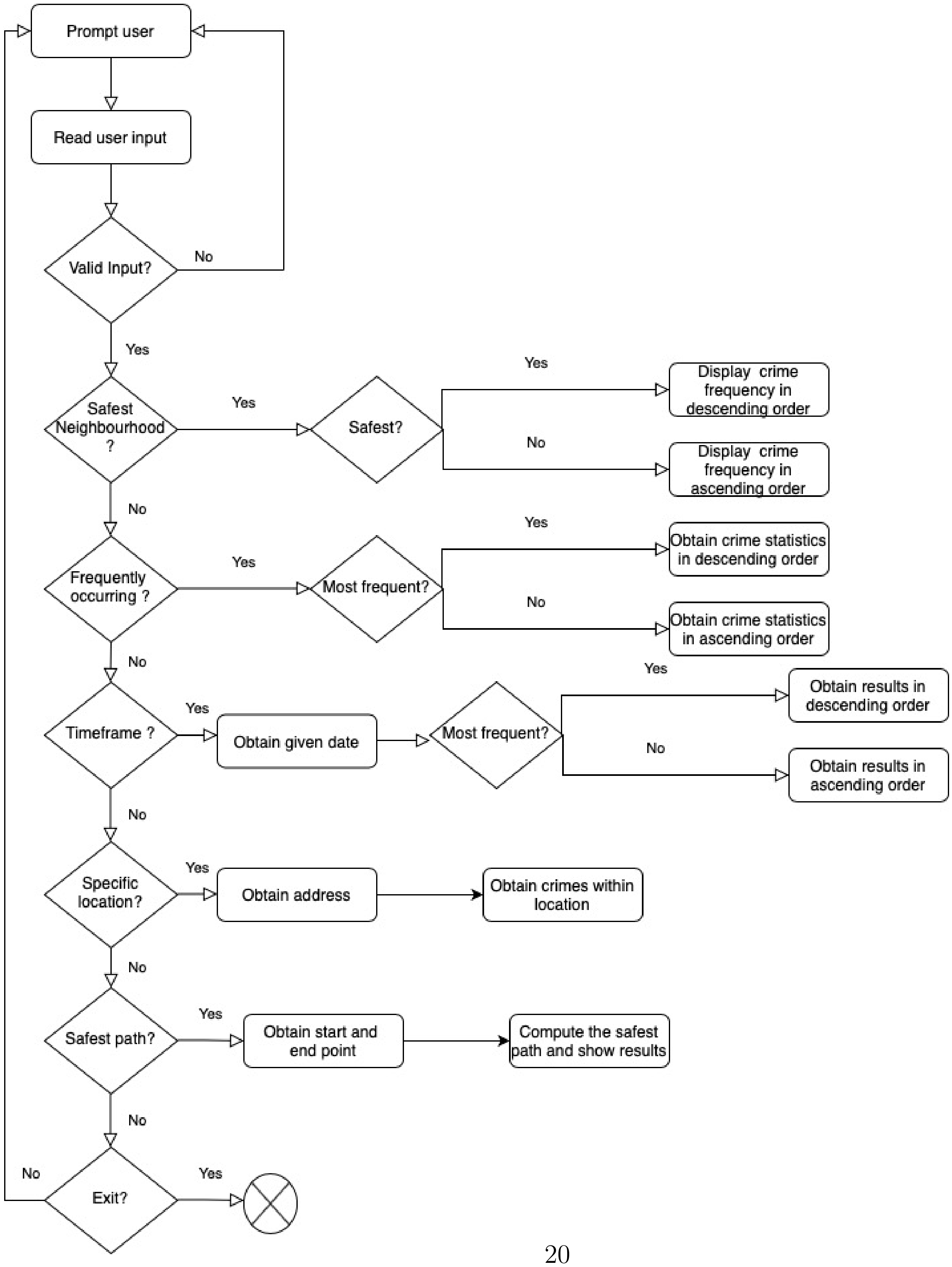
weight():

- output: out := weight
- exception: None

equals(DirectedEdge other):

- output: out := $to = other.to() \wedge from = other.from$
- exception: None

UML state diagram for Interface


Prompt user
Read user input
Valid Input?
No
Yes
Safest Neighbourhood ?
Yes
Safest?
Yes
Display crime frequency in descending order
No
Display crime frequency in ascending order
No
Frequently occurring ?
Yes
Most frequent?
Yes
Obtain crime statistics in descending order
No
Obtain crime statistics in ascending order
No
Timeframe ?
Yes
Obtain given date
Most frequent?
Yes
Obtain results in descending order
No
Obtain results in ascending order
No
Specific location?
Yes
Obtain address
Obtain crimes within location
No
Safest path?
Yes
Obtain start and end point
Compute the safest path and show results
No
Exit?
No
Yes

Module descriptions

Interface

The interface module is equivalent to the view and controller module in the model-view-controller design. The program user uses this class as an interface to interact with Map and Shortest Path in order to perform a variety of queries.

Map

This module functions as the model where all the logical processes occur. It provides methods for all the possible queries that a user can search for using the interface module to interact with it. A high-level abstraction of the methods available are:

- Obtain the safest/least safe neighbourhoods according to the number of crimes committed within that neighbourhood.
- Obtain the most/least occurring crimes within NYC, within a specific neighbourhood or within a given time period.
- Obtain the crimes committed within a certain radius of a specified address.
- Obtain the safest path from one neighbourhood to another.

ShortestPath

Given a starting point and a destination, obtain the safest path from the two points according to the number of crimes committed in each neighbourhood.

EdgeWeightedDiGraph

An edge-weighted directed graph with all the neighbourhood connections. The vertices are neighbourhoods and the edges are paths from one neighbourhood to another with the edge-weight being the number of crimes committed within the next vertex.

DirectedEdge

Standard class representing a directed edge. The vertices are Strings and the weight is an Integer.

ChunkT

ChunkT represents a neighbourhood and all the information contained within. Chunks are specified by their four boundaries and their name. Chunks also contain information about all the crime statistics contained within their boundaries.

CrimeT

Represents a crime and contains information about the date of occurrence, crime description, and location. Two crimes are compared according to their date of occurrence.

CoordinateT

A module for representing coordinates according to latitude and longitude.