Requirements Specification

COMP SCI 2XB3, Lab 02, Group 05

March 1, 2019

1 The Domain

This project aims to track historic crime data of major cities and generate commuting routes that will avoid high crime areas while reducing trip durations. Areas avoided will consists of crimes that are frequent, violent, and severe. The goal for our implementation is to provide users with a more informed perspective of travel methods so that they can take necessary precautions as they see fit. We will implement our project first as a web application, later transitioning into a mobile version once fundamental operations are optimised.

In the early stages of our application we aim to focus strictly on the Greater Toronto Area. Once an efficient model has been developed for this geographic region we plan to expand to other major cities across North America.

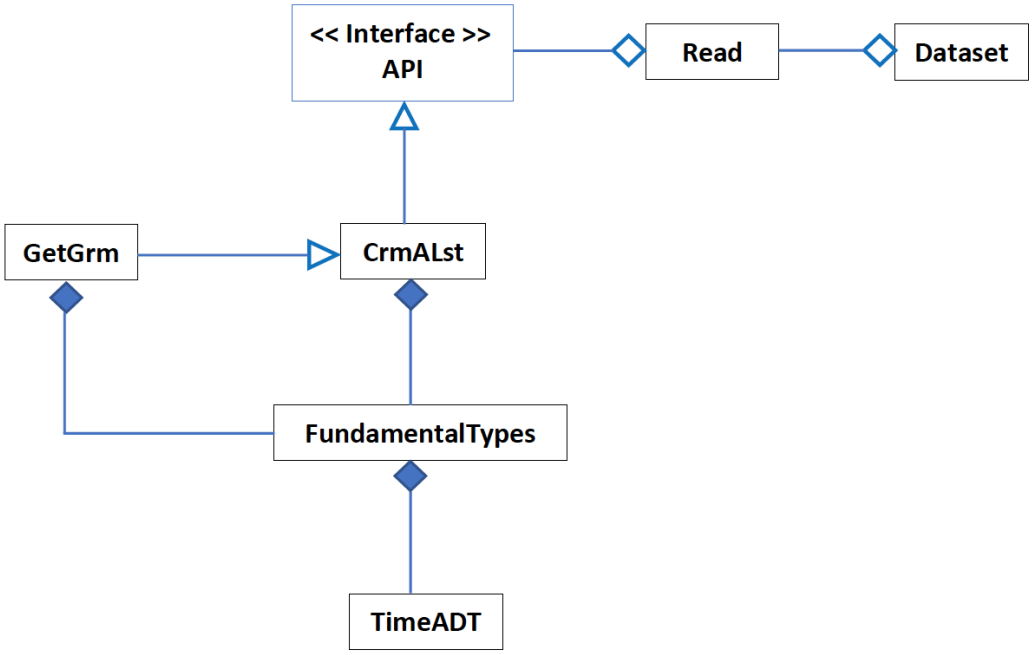
Stakeholders for our application consist mainly of individuals in the regions covered by our application. Those who choose to use our product will have a fuller understanding of their environment giving them the opportunity to keep themselves and others out of harm's way. Those who don't use our product might notice a change in volume of traffic in given areas of their commute.

Our product may give users information that encourages them to change their commuting patterns. Entities that may be affected by this change consist of local transit services and businesses. If a given area sees a significant increase or decrease in traffic, local transit services might need to consider modifying the resources they offer to reflect new commuter preferences. This shift in commuting preferences might also have an affect on store traffic for businesses established near an area that has been deemed a crime hotspot.

2 Functional Requirements

Step safe will take an input of a starting location and destination, and output directions between the two locations. Formulation of this output will be done by considering the fastest route first, and then making additional adjustments to avoid areas of high crime.

UML Diagram



Fundamental Types Module

Module

FundamentalTypes

Description

This module consists of fundamental supporting data types for crime information.

Uses

TimeADT(s, h)

Syntax

Exported Constants

None

Exported Types

CoordinateT = tuple of (x : float , y : float) where x is Latitude, y is Longitude
MCI = {assault, breakAndEnter, robbery}
Premise = {outside, house, commercial, apartment, other}
Hood = tuple of (id : $\mathbb{N}$, area : String)

CrimeInfoT = tuple of (time: TimeADT(s, h), place: Premise,
coordinate: CoordinateT, type: MCI, neighbourhood: Hood)

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Occurrence Time ADT Module

Template Module

TimeADT(s, h)

Description

The module provides the abstract data type for the occurrence time of crimes.

Uses

None

Syntax

Exported Constants

None

Exported Types

TimeADT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new SeqADT	s : string, h : string	TimeADT	InvalidArguments
getYear		int	
getMonth		int	
getDay		int	
getHour		int	
toWeekDay		WeekDays	
compareTo	TimeADT	int	
toString		string	

Semantics

State Variables

year: integer
month: integer

day: integer
hour: integer

State Invariant

$month \in [1..12]$
$hour \in [0..24]$
output of $compareTo \in [-1, 0, 1]$

Assumptions

The input string s should be the format: $yyyy - mm - dd$ where year, month and day are separated by dash.

Access Routine Semantics

new TimeADT(s, h):

- transition: $year$:= extract year information from input s and convert to integer
  $month$:= extract month information from input s and convert to integer
  day := extract day information from input s and convert to integer
  $hour := (integer)h$
- output: out := self
- exception: (day not in the range of the month ⇒ InvalidArgument)

getYear():

- output: $out := year$
- exception: none

getMonth():

- output: $out := month$
- exception: none

getDay():

- output: $out := day$

- exception: none

getHour():

- output: $out := hour$
- exception: none

toWeekDay():

- output: $out :=$ day of a week
- exception: none

compareTo:

- output: $i, out :=$ -1, 0, 1 respectively if t1 is before, equal, after t2
- exception: none

toString():

- output: $out :=$ string representation of the time in the format: $yyyy-mm-dd : hh$
- exception: None

Local Types

WeekDays = {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}

Crime Data Association List Module

Module

CrmALst

Description

The module provides an abstract object to store crime records.

Functionalities

add - add a crime record to the sequence
remove - remove a crime record from the sequence
elm - check if a crime record exists in the sequence
$info$ - returns the detailed information of a crime
count - count the number of crimes which satisfy the property given by the input function

Uses

FundamentalTypes
TimeADT(s, h)
GetCrmRate

Syntax

Exported Constants

None

Exported Types

FundamentalTypes

Exported Access Programs

Routine name	In	Out	Exceptions
init			
add	string, CrimeInfoT		KeyError
remove	string		KeyError
elm	string	$\mathbb{B}$	
info	string	CrimeInfoT	KeyError
count	CrimeInfoT $\rightarrow \mathbb{B}$	integer	
sort		sequence of string	

Semantics

State Variables

s: sequence of CrimeT

State Invariant

None

Assumptions

CrmALst.init() is called before any other access program.

Access Routine Semantics

init():

- transition: $s := \{\}$
- exception: none

add(id, i):

- transition: $s := s \cup \{\langle id, i \rangle\}$
- exception: $(\langle id, ? \rangle \in s \Rightarrow \text{KeyError})$

remove(id):

- transition: $s := s - \{\langle id, i \rangle\}$ where $\langle id, i \rangle \in s$
- exception: $(\langle id, i \rangle \notin s \Rightarrow \text{KeyError})$

elm(id):

- output: $out := \langle id, i \rangle \in s$
- exception: none

info(id):

- output: $out := i$ where $\langle id, i \rangle \in s$
- exception: ($\langle id, i \rangle \notin s \Rightarrow$ KeyError)

count(f):

- output: $out := (+i : \text{CrimeInfoT} | i \in CrimeInfoT \wedge f(i) : 1)$
- exception: None

sort:

- output: $out :=$ a sequence of crime IDs which is sorted by ascending order based on the crime of the around area gotten by applying $getCrime$ function on its CrimeInfoT
- exception: None

Local Types

CrimeT = tuple of (id: string, info: CrimeInfoT)

Get Crime Module

Module

GetCrm

Description

The module provides the function to calculate the number of crimes in the around area of a coordinate.

Uses

FundamentalTypes
TimeADT(s, h)
CrmALst

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
getCrime	id : string, $r : \mathbb{R}$	integer	
getCoordinate	id : string	CoordinateT	

Semantics

Environment Variables

s : sequence of CrimeT, generated by CrmALst

State Variables

None

State Invariant

None

Assumptions

The argument r is the radius of the circle which represents the range of the around area. It's specified by unit *kilogram*.

Access Routine Semantics

getCoordinate(id):

- output: $out := i.coordinate$ where $\langle id', i\rangle \in s \wedge id' = id$
- exception: none

getCrime(id, r):

- transition: $s := (+(id', i) : \text{CrimeT}|\text{getCoordinate}(id') \in circle(c, r) : 1)$ where $c =$ getCoordinate(id)
- exception: none

The function *getCrime* do searching on s and count the number of crimes happened in the around area.

Local Functions

circle: CoordinateT $\times \mathbb{R} \rightarrow$ set of CoordinateT
circle (c, r) $\equiv \{(x, y) : CoordinateT|(x - x_0)^2 + (y - y_0)^2 \leq r^2 \ : (x, y)\}$ where $c = (x_0, y_0)$

Read Module

Module

Read

Description

The module provides functions to read data from dataset.

Uses

CrmALst

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
load_crime_data	s : string		

Semantics

Environment Variables

crime_dataset: csv file containing all crime data

State Variables

None

State Invariant

None

Assumptions

The input file will match the given specification.

Access Routine Semantics

load_crime_data(s)

- transition: read data from the file crime_dataset associated with the string s. Use this data to update the state of the CrmALst module. Load will first initialize CrmLst (CrmLst.init()) before filling CrmALst with crime records that follows the types in CrimeT.

  The input csv file contains the following fileds:
  x: lantitude , y: longitude , $event\_unique\_id$: id of the crime , $occurrencedata$: crime occurrence date , $premisetype$: premise type , $offence$: the short stand for MIS (e.g. B&E for BreakAndRnter), MIS: MIS , $Hood\_ID$: hood id , $Neighbourhood$: the area name

- exception: none

Graph Module

Module

Graph

Description

Graph a sequence of coordinates on the map.

Uses

FundamentalTypes

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
graphSeq	xs : sequence of CoordinateT		

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

graphSeq(xs):

- transition: display path connected by xs on the map
- exception: none

API Module

Module

API

Description

The module provide application programming interface for the project.

Uses

FundamentalTypes
CrmALst
GetCrm
Graph

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
navigate	*start* : string, *end* : string		WrongLocation

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

negivate(start, end):

- transition: display the best route between starting point and destination.
- exception: none

Local Functions

convertToCoordinade: string $\rightarrow$ CoordinateT
convertToCoordinade(l): returns the coordinate of the location given by string l

3 Non-Functional Requirements

The software should have a dynamic and reasonable trade-off between efficiency and crime avoidance to encourage reliability. The product will originally be available to users in the Greater Toronto Area and accessed through a web application. Individual user information and location will remain hidden from other users to promote security. To increase accuracy of results, the directions outputted will use a function to optimize the crime score and the time to reach the inputted destination. This analysis as well as the sorting and searching algorithms would need to be as efficient as possible to provide a good user experience. Techniques for decreasing speed should be considered to minimize total run time of direction processing. The user interface should be simple and consist of two inputs for the starting and final destinations and one output of the directions. Multiple choices of directions may be presented with different variables considered (such as total distance, toll roads, etc). Outputted directions would provide the users with distances and road names with directions for each turn.

4 Requirements on Development and Maintenance Process

The three main algorithms that StepSafe uses includes a search algorithm, a sort algorithm and a graphing algorithm. Of the three algorithms, the search algorithm has the highest priority because the data generated will be used in the graphing algorithm. The

sort algorithm is needed to filter out relevant data to be used.

Quality control procedures will include several steps of testing: unit testing, integration testing, system testing, and acceptance testing. All testing, developing, and designing of code will seek to ensure that modules, and ultimately the web application are verifiable, correct, reliable, usable, efficient, and as robust as possible. A unique module will be created for the purpose of facilitating testing. The functions in the module will have meaningful names relating to what is being tested. After running all test cases in the module, there will be a summary of the tests that were run, including number of passed and failed tests. During the development stage, unit testing, both from a blackbox and whitebox perspective will be performed after modification/creation of a function or property of the module, to ensure that the individual units of code work as intended. Integration test will be regularly used, when possible to ensure that new and updated code does not cause any problems. Following the completion of preliminary modules, System testing can be done to ensure the compatibility of our code as a web application. Lastly, acceptance checking will be done prior to web application launch to ensure all requirements are met, and that the application is suitable to be used by the public. This will include testing of the application's reliability and usability by having our programmers use directions given by the app to traverse areas in the target city. Further testing will also be done by non-programmers outside of the project to gain objective feedback about the web application,that will be considered for update and change.

After initial establishment of the web applications, possible future improvements of the application could include different filtering for methods of transportation, and additional features such as time estimation from destination to arrival, options for path optimization for a sequence of destinations, and exporting direction instructions to a mobile phone.

In order to ensure that our application is up-to-date with current technology, features and methods will be modularized properly with information hiding so that the application can be easily modified. By doing this, identifying and fixing bugs in code will also be more efficient. Other anticipated changes include possible porting to new hardware or software platforms such as a mobile application version, that can perform the same functionality with more convenience.