McMaster University, Department of Computing and Software
CS/SE 2XB3 2020 Final Projects Instructor: Dr. Reza Samavi

Design Specification

Version: 1

Project Name: Earthquake Risk Assessment
Group: 01
Member: Kan Hailan, Sembakutti Kalindu,
Tao Haoyang, Ye Fang

April 11, 2020

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.

Revision History

Name	**Date**	**Version**
Design Specification	11 April, 2020	1

Group Members and Roles

Name	Student Number	Role in the Project	Responsibility
Kan Hailan	400207974	client & tester	Implement unit tests, debug and verify code
Sembakutti Kalindu	1046206	researcher & programmer	Cleaning and extracting data, writing documents, data structure algorithms
Tao Haoyang	400171589	designer & programmer	Module decomposition, implement ADT modules.
Ye Fang	400273067	project leader & programmer	Manage the project, produce the prototype, implement sort, risk assessment, controller and view modules

Contribution

Name	Role(s)	Contributions
Kan Hailan	client & tester	• Write test cases for 12 modules • Debug or change test cases according to test results • Review and verify MIS • Finish the domain part in SRS • Finish the trace back to requirements and internal review in Design Specification
Sembakutti Kalindu	researcher & programmer	• Outlining initial SRS functional and non-functional requirements • Implementing CSVreader module • Implementing data structures (EarthquakeBag, RedBlackBST, Queue) • Modified datasets to have correct province names, and non-empty cells • The MIS of 4 modules (CSVreader, RedBlackBST, Earthquake T, Queue) • Review and editing documents
Tao Haoyang	designer & programmer	• Create UML class diagram for all initial 21 modules and keep updating to current version. • Idea discussions with project leader. • Implementation of ADT modules (EarthquakeT, CityT, PointT) • Implementation of GeoCollection module • SRS part 3.4: Requirements on the development and maintenance process • Design Specification part 2: Module decomposition and UML, view of uses relationship
Ye Fang	project leader & programmer	• Manage the program to meet all milestones • Finish the MIS of 14 modules • Implement 12 modules (CityPostT, Edge, Graph, SearchEarthquakes, Sort, RiskAssessement, ViewRisk, ViewList, DisplayByMagnitude, DisplayByDistance, Controller, MCVDemo) • Prepare the outline of SRS and Design Specification • Finish and update the Overall Description in SRS • Design Specification part 3: MIS (14 modules) and UML state machine diagrams

Executive Summary

The earthquake risk assessment is vital for numbers of different reasons. The main reason is that human lives are endangered unnecessarily by living in a geologically active area that poses a significant risk of a damaging earthquake in the future. There are places considered to be geographically beautiful, but they have an underlying earthquake risk. First, the people living in these places should become aware of this risk, in order to, make informed decisions for future relocations.

The other aspect is that cities should adjust their building codes to be more suitable to the geological activities. The cities can pass new construction codes into by-laws, based on our earthquake risk assessment, so the building developers who are more profit-oriented do not endanger lives unnecessarily by over-doing construction projects in a geologically active place.

The last aspect is insuring the buildings that were developed in the past, but now face a significant risk due to known geological activities. These buildings can be Canadian heritage sites that may require significant costs to repair if damaged by an earthquake. With the knowledge of our risk assessment in hand, all stakeholders can take necessary steps to properly insure these buildings.

Table of Contents

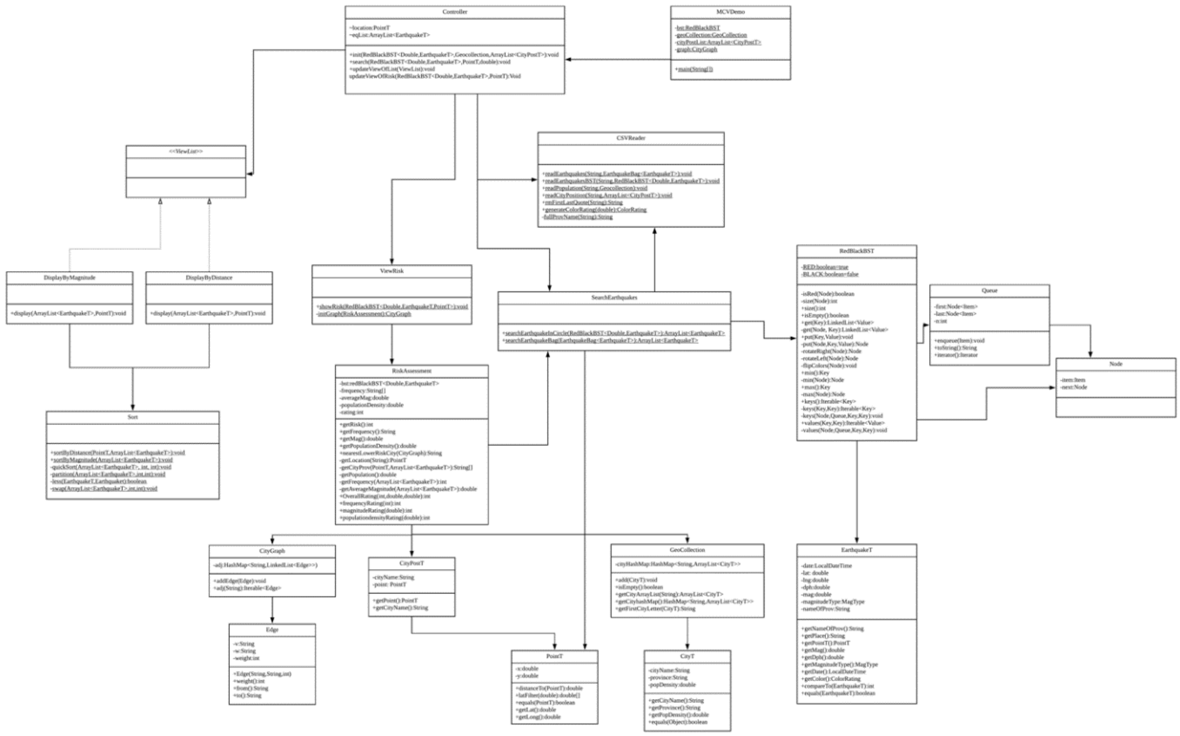
1 Overall Design Description

The project reads from a csv file all the earthquake information, one earthquake at a time, storing individual earthquake information in an earthquake object, and those objects themselves in a RedBlackBST data structure. The project allows the user to provide input latitude, longitude, and a radius to search earthquakes within the radius. It produces a list of earthquakes sorted based on the magnitude or distance from the user location. It also shows the risk rating for a specific location, based on earthquakes and population data within a 100 km radius. It calculates the overall risk rating, based on earthquake frequency, average magnitude, and population density.

In addition, the project uses a city coordinate dataset which has a specific location of a city in terms of longitude and latitude. The program iterates through all the earthquakes within a 100 km radius, finds the closest earthquake, and accesses its location in order to determine a geographical name of the current location. It, then, use a CityGraph created by the city coordinate dataset to determine the closest city by finding the adjacent edge with the lowest weight from the current location. When iterating through the adjacent cities, it also determines if the closest city also has a lower risk rating than the current location. Once, it finds the closest city with a lower risk rating, it outputs that location.

2 Module Decomposition

- Our product is following the MVC design pattern that is, we have a controller module and view modules. At the very beginning of our prototype, the performance was unsatisfiable, since we are using EarthquakeBag, a linked list data structure to keep everything in order.
- Then, we decide to add a method “latFilter” in PointT ADT. This method will be given a radius and make the current PointT object as the center and produce a new pair of latitude to “filter” out points that outside of the given radius. Now the performance has increased a little.
- We finally decide to use RedBlackBST to improve the performance, discard using EarthquakeBag. By making this decision, the time complexity for insertion has increased, but when it comes to searching and sorting, the time taken is greatly reduced.
- All classes and ADTs are well reused. For example, PointT is a point consist of x and y coordinates representing latitude and longitude respectively, and CityPostT uses PointT as part of its state variable. By dividing those classes and methods, we are trying to balance low coupling, code reusability, and maintainability at the same time.
- UML class diagram(←click on the hyperlink to view in full resolution)



3 MIS

There are totally 19 modules for this system. We have the MIS**e**s for all modules except MCVDemo in the following section. We also include the state machine diagrams for Sort and MCVDemo classes respectively.

3.1 MIS

CSVreader Module

Module

CSVreader

Uses

CityPostT, CityT, EarthquakeT,
EarthquakeT.ColorRating, EarthquakeT.MagType,
EarthquakeBag, GeoCollection, RedBlackBST

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
readEarthquakes	String, EarthquakeBag		IOException
readEarthquakesBST	String, RedBlackBST		IOException
readPopulation	String, GeoCollection		IOException
readCityPosition	String, seq of CityPostT		IOException
rmFirstLastQuote	String	String	
generateColorRating	$\mathbb{R}$	ColorRating	
fullProvName	String	String	

Semantics

Environment Variables

A file listing Earthquake Information.
A file listing Population Densities.
A file listing City Coordinates.

State Variables

None

State Invariant

None

Assumptions

The input file by filename will match the specific format required by the read method.

Access Routine Semantics

readEarthquakes(filename, bag):

- transition: Read each line of the earthquake csv file and convert to a EarthquakeT object, which is stored in a EarthquakeBag.
- exception: If the file by filename does not exist, produces IOException.

readEarthquakesBST(filename, bst):

- transition: Read each line of the earthquake csv file and convert to a EarthquakeT object, which is stored in a RedBlackBST.
- exception: If the file by filename does not exist, produces IOException.

readPopulation(filename, geoCollec):

- transition: Read each line of the population csv file and convert to a CityT object, which is stored in a GeoCollection HashMap.
- exception: If the file by filename does not exist, produces IOException.

readCityPosition(filename, cityPostList):

- transition: Read each line of the city coordinates csv file and convert to a CityPostT object, which is stored in a list of cities.
- exception: If the file by filename does not exist, produces IOException.

rmFirstLastQuote(cell):

- transition: Remove first and last double quotations from a string value.
- exception: None

generateColorRating(cell4):

- transition: Generate an enum ColorRating type based on the magnitude of earthquake.
- exception: None

fullProvName(nameP):

- output: a new province name similar to the following table.

	nameP =	*out* :=
$\lvert\text{nameP}\rvert = 2$	ON	Ontario
	QC, PQ	Quebec
	NS	Nova Scotia
	NB	New Brunswick
	MB	Manitoba
	BC	British Columbia
	PE	Prince Edward Island
	SK	Saskatchewan
	AB	Alberta
	NL	Newfoundland and Labrador
	NU	Nunavut
	NT	Northwest Territories
	YT	Yukon
	AK	Alaska
	WA	Washington
	default	UNLOCATED
$\lvert\text{nameP}\rvert \neq 2$	VANCOUVER ISLAND	British Columbia
	SOUTHERN QUEBEC	Quebec
	default	UNLOCATED

- exception: None

Considerations

There are a number of different variations of geolocation names in the earthquake csv file, for these an appropriate province name should be assigned. For any that could not be matched to a province name, UNLOCATED should be assigned.

Point ADT Module

Template Module

PointT

Uses

N/A

Syntax

Exported Types

PointT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new PointT	$\mathbb{R}, \mathbb{R}$	PointT	RuntimeException
getLat		$\mathbb{R}$	
getLong		$\mathbb{R}$	
distance	PointT	$\mathbb{R}$	
latFilter	$\mathbb{R}$	$< \mathbb{R}, \mathbb{R} >$	
equals	PointT	$\mathbb{B}$	

Semantics

State Variables

$x : \mathbb{R}$
$y : \mathbb{R}$

Access Routine Semantics

PointT($lat, long$):

- transition: $x, y := lat, long$
- output: $out := self$
- exception: $exc := ((lat > 90 \vee lat < -90) \Rightarrow \text{IndexOutOfBoundsException})$

getLat():

- output: $out := x$

getLong():

- output: $out := y$

distanceTo(that):

- output: $out :=$ d such that d is the distance(in km) between current point and that

latFilter(radius):

- output: $out :=< minLat, maxLat >$ such that $\forall (p : PointT | distanceTo(p) \leq radius : p.getLat() \geq minLat \wedge p.getLat() \leq maxLat$

equals(that):

- output: $out := (x = that.getLat()) \wedge (y = that.Long())$

City ADT Module

Template Module

CityT

Uses

N/A

Syntax

Exported Types

CityT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new CityT	String, String, $\mathbb{R}$	CityT	
getCityName		String	
getProvince		String	
getPopDensity		$\mathbb{R}$	
equals	CityT	$\mathbb{B}$	

Semantics

State Variables

$cityName$: String
$province$: String
$popDensity$: $\mathbb{R}$

Access Routine Semantics

CityT($city, pro, pop$):

- transition: $cityName, province, popDensity := city, pro, pop$
- output: $out := self$

getCityName():

- output: $out := cityName$

getProvince():

- output: $out := province$

getPopDensity():

- output: $out := popDensity$

equals(that):

- output: $out := (cityName = that.getCityName()) \wedge (province = that.getProvince()) \wedge (popDensity = that.getPopDensity())$

City Position ADT Module

Template Module

CityPostT

Uses

PointT

Syntax

Exported Types

CityPostT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new CityPostT	String, $\mathbb{R}, \mathbb{R}$	CityPostT	
getPoint		PointT	
getCityName		String	

Semantics

State Variables

$cityName$: String
$point$: PointT

Access Routine Semantics

CityT($city, lat, lon$):

- transition: $cityName, point := city, newPointT(lat, lon)$
- output: $out := self$

getPoint():

- output: $out := point$

getCityName():

- output: $out := cityName$

CityT Collection Module

Template Module

GeoCollection

Uses

CityT

Syntax

Exported Types

GeoCollection = ?

Exported Access Programs

Routine name	In	Out	Exceptions
add	CityT		
getCities	String	seq of CityT	
getAllCities		set of tuple of(String, seq of CityT)	
isEmpty		$\mathbb{B}$	

Semantics

State Variables

s : set of tuple of (String, seq of CityT)

Access Routine Semantics

add(city):

- transition: $s := \{< str, cities >:< String, \text{seq of CityT} > \ | \ < str, cities > \in s : (str = getFirstCityLetter(city) \Rightarrow < str, cities||[city] > \ | \ true \Rightarrow < str, cities >)\}$

isEmpty():

- output: $out := (|s| = 0 \Rightarrow true \mid true \Rightarrow false)$

getCities(firstLetter):

- output: $out := \{< str, cities >:< String, \text{ seq of CityT} > \ | \ < str, cities > \in s \wedge str = firstLetter : cites\}$

getAllCities():

- output: $out := s$

Local Functions

getFirstCityLetter : String $\rightarrow$ String
getFirstCityLetter($city$) $\equiv city[0]$

Earthquake ADT Module

Template Module

EarthquakeT

Uses

LocalDateTime, PointT

Syntax

Exported Types

EarthquakeT = ?

ColorRating = { NOCOLOR, ZERO, PURPLE, BLUE, GREEN, YELLOW, ORANGE, RED }

MagType = { M5, mb, MB, Mb, MC, Mc, mc, ML, MLSn, MN, MS, MW, Ms, Mw, BLANK }

 # EarthquakeT implements Comparable(EarthquakeT)

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
new EarthquakeT	String, String, LocalDateTime, $\mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R}$, MagType, ColorRating	EarthquakeT	
getNameOfProv		String	
getPlace		String	
getPointT		PointT	
getMag		$\mathbb{R}$	
getDph		$\mathbb{R}$	
getMagitudeType		MagType	
getDate		LocalDateTime	
getColor		ColorRating	
compareTo	EarthquakeT	$\mathbb{Z}$	
equals	EarthquakeT	$\mathbb{B}$	

Semantics

State Variables

place: String
nameOfProv: String
date: LocalDateTime
lat: $\mathbb{R}$
lng: $\mathbb{R}$
dph: $\mathbb{R}$
mag: $\mathbb{R}$
magnitudeType: MagType
color: ColorRating

State Invariant

None

Assumptions

Two earthquakes are not the same if they happened to have two different dates or two different places recorded.

Access Routine Semantics

EarthquakeT(place, prov, date, lat, lng, dph, mag, mgT, color):

- transition:
  lat, lng, place, nameOfProv, date, dph, mag, magnitudeType, color :=
  lat, lng, place, prov, date, dph, mag, mgT, color
- output: *out* := *self*
- exception: None

getNameOfProv():

- output: *out* := nameOfProv
- exception: None

getPlace():

- output: *out* := place
- exception: None

getPointT():

- output: *out* := PointT(lat, lng)
- exception: None

getMag():

- output: *out* := mag
- exception: None

getDph():

- output: *out* := dph
- exception: None

getMagitudeType():

- output: *out* := magnitudeType
- exception: None

getDate():

- output: out := date
- exception: None

getColor():

- output: out := color
- exception: None

compareTo(eq):

- output: out := an integer value according to the following table.

	$out :=$
$this$.mag < eq.mag	-1
$this$.mag > eq.mag	1
$this$.mag = eq.mag	0

- exception: None

equals(that):

- output: out := (sameDate $\wedge$ samePoint $\wedge$ samePlace $\wedge$ sameDepth $\wedge$ sameMagValue $\wedge$ sameMagType $\wedge$ sameEqClass) $\Rightarrow True | True \Rightarrow False$
- exception: None

Local Functions

sameDate: EarthquakeT $\rightarrow \mathbb{B}$
sameDate(d) $\equiv$ (d.date) = ($this$.date)
 # Returns true if the given EarthquakeT object has the same date as the current.

samePoint: EarthquakeT $\rightarrow \mathbb{B}$
samePoint(d) $\equiv$ (d.Point) = ($this$.Point)
#Returns true if the given EarthquakeT object has the same Point as the current.

samePlace: EarthquakeT $\rightarrow \mathbb{B}$
samePlace(d) $\equiv$ (d.place) = ($this$.place)
#Returns true if the given EarthquakeT object has the same place as the current.

sameDepth: EarthquakeT $\rightarrow \mathbb{B}$
sameDepth(d) $\equiv |d.\mathrm{dph} - this.\mathrm{dph}| < 0.0000001$
#Returns true if the given EarthquakeT object has the same depth value as the current within the tolerance.

sameMagValue: EarthquakeT $\rightarrow \mathbb{B}$
sameMagValue(d) $\equiv |d.\mathrm{mag} - this.\mathrm{mag}| < 0.0000001$
#Returns true if the given EarthquakeT object has the same magnitude value as the current within the tolerance.

sameMagType: EarthquakeT $\rightarrow \mathbb{B}$
sameMagType(d) $\equiv$ (d.magnitudeType) = ($this$.magnitudeType)
#Returns true if the given EarthquakeT object has the same magnitude type as the current.

sameEqClass: EarthquakeT $\rightarrow \mathbb{B}$
sameEqClass(d) $\equiv$ (d.color) = ($this$.color)
#Returns true if the given EarthquakeT object has the same earthquake class as the current.

Edge ADT Module

Template Module

Edge

Uses

N/A

Syntax

Exported Types

Edge = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new Edge	String, String, $\mathbb{Z}$	Edge	
weight		$\mathbb{Z}$	
from		String	
to		String	

Semantics

State Variables

v : String
w : String
$weight$: $\mathbb{Z}$

Access Routine Semantics

Edge($from, to, w$):

- transition: $v, w, weight := from, to, w$
- output: $out := self$

weight():

- output: $out := weight$

from():

- output: $out := v$

to():

- output: $out := w$

City Graph Module

Template Module

CityGraph

Uses

Edge

Syntax

Exported Types

CityGraph = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new CityGraph		CityGraph	
addEdge	Edge		
adj	String	seq of Edge	

Semantics

State Variables

adj : set of tuple of (String, seq of Edge)

Access Routine Semantics

CityGraph():

- transition: $adj := \{\}$
- output: $out := self$

addEdge(e):

- transition: $adj := \{< str, edges >:< String, \text{ seq of Edge } > \ | \ < str, edges > \in adj : (str = e.from() \Rightarrow < str, edges || [e] > \ | \ true \Rightarrow < str, edges >)\}$

adj(v):

- output: $out := \{< str, edges >:< String, \text{ seq of Edge } > \ | \ < str, edges > \in adj \wedge str = v : edges\}$

EarthquakeBag Module

Template Module

EarthquakeBag is seq of EarthquakeT

 # Used only for testing time performance against RedBlackBST module.
 # This module is not part of the actual application.

Generic Queue Module

Generic Template Module inherits Iterable(T)

Queue(T)

Uses

None

Syntax

Exported Constants

None

Exported Types

Queue = ?

Internal Types

Node = ?

 # Internal Node type has a link to next item in the queue.

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Queue		Queue	
isEmpty		$\mathbb{B}$	
enqueue	T		
toString		String	
start			
next		T	NoSuchElementException

Semantics

State Variables

first: Node
last: Node
$n : \mathbb{N}$
s: seq of T

 # For simplification, the linked-node structure is represented by seq of T.
 # s[1] is the first Node.
 # s[n] is the last Node.

State Invariant

None

Assumptions

None

Access Routine Semantics

Queue():

- transition: first, last, n := null, null, 0
- output: out := self
- exception: none

isEmpty():

- output: $out := (n = 0) \Rightarrow True | True \Rightarrow False$
- exception: None

enqueue(item):

- output: $out := s||$item
- exception: None

toString():

- output: $out := out||(\forall\, i : \mathbb{N}|i \in [1..n] : s[i])$
- exception: None

<u>Iterator Methods</u>:

$i : \mathbb{N}$

start():

- transition: $i := 0$
- exception: none

next():

- transition-output: $i, out := i + 1, s[i]$
- exception: $(i > n) \Rightarrow$ NoSuchElementException

Considerations

When an instance of Queue is iterated in a loop, an iterator consisting of these two methods is returned, and the start() method is call initially, and for the successive iterations next() method is call.

Generic RedBlackBST Module

Generic Template Module

RedBlackBST(T with Comparable(T), V)

Uses

Queue

Syntax

Exported Types

RedBlackBST = ?

Internal Types

Node = ?
State Variables of Node:
key: Key, lst: seq of V, left: Node, right: Node, color: B, size: N

 # Internal Node type was modified to store a seq of V.

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
RedBlackBST		RedBlackBST	
size		$\mathbb{N}$	
isEmpty		$\mathbb{B}$	
get	T	seq of V	
put	T, V		
min		T	
max		T	
keys		seq of T	
keys	T, T	seq of T	
values	T, T	seq of V	

Semantics

State Variables

root: Node
RED: $\mathbb{B}$
BLACK: $\mathbb{B}$
s: set of $\langle T, V \rangle$

 # For simplification, the linked-node structure is represented by set of $\langle T, V \rangle$.

State Invariant

RED = True
BLACK = False

Assumptions

None

Access Routine Semantics

RedBlackBST():

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

size():

- output: $out := root.size$
- exception: None

isEmpty():

- output: $out := (root = null) \Rightarrow True | True \Rightarrow False$
- exception: None

get(key):

- output: $out :=$ L where $\langle x, L \rangle \in s \wedge (x.key = key)$

- exception: None

put(key, val):

- transition: $s := \{\langle x, L\rangle : \langle T, V\rangle | \langle x, L\rangle \in s : (x.key = key \Rightarrow \langle x, L || [val]\rangle | \text{True} \Rightarrow \langle x, L\rangle)\}$
- exception: None

min():

- output: $out :=$ smallest key in s
- exception: None

max():

- output: $out :=$ largest key in s
- exception: None

keys():

- output: $out := out || (\forall \langle x, L\rangle : \langle T, V\rangle | \langle x, L\rangle \in s : x.key)$
- exception: None

keys(lo, hi):

- output: $out := out || (\forall \langle x, L\rangle : \langle T, V\rangle | \langle x, L\rangle \in s \wedge lo \leq x.key \leq hi : x.key)$
- exception: None

values(lo, hi):

- output: $out := out || (\forall \langle x, L\rangle : \langle T, V\rangle | \langle x, L\rangle \in s \wedge lo \leq x.key \leq hi : L)$
- exception: None

Search Earthquakes Module

Module

SearchEarthquakes

Uses

RedBlackBST, PointT

Syntax

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
searchEarthquakeInCircle	RedBlackBST, PointT, $\mathbb{R}$	seq of EarthquakeT	

Semantics

State Variables

N/A

Access Routine Semantics

searchEarthquakeInCircle(bst, location, radius):

- output: $out := \{e : EarthquakeT \mid e \in bst \wedge location.distanceTo(e.getPointT()) \leq radius : e\}$

Sort Module

Module

Sort

Uses

PointT

Syntax

Exported Types

N/A

Exported Access Programs

Routine name	In	Out	Exceptions
sortByDistance	PointT, seq of EarthquakeT		
sortByMagnitude	seq of EarthquakeT		

Semantics

State Variables

N/A

Access Routine Semantics

sortByDistance(location, eqList):

- transition: $eqList := eqList$ such that $\forall(i : \mathbb{N} \mid i \in [0..|eqLisi| - 2] :$ $location.distanceTo(eqList[i].getPointT()) < location.distanceTo(eqList[i{+}1].getPointT()))$

sortByMagnitude(eqList):

- transition: $eqList := eqList$ such that $\forall(i : \mathbb{N} \mid i \in [0..|eqLisi| - 2] : eqList[i] >$ $eqList[i + 1]$

Risk Assessment Module

Template Module

RiskAssessment

Uses

SearchEarthquakes, GeoCollection, CityGraph, CityPostT

Syntax

Exported Types

RiskAssessment = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new RiskAssessment	RedBlackBST, PointT		
getRisk		$\mathbb{Z}$	
getCity		String	
getFrequency		$\mathbb{Z}$	
getMag		$\mathbb{R}$	
getPoplationDensity		$\mathbb{R}$	
nearestLowerRiskCity	CityGraph	String	

Semantics

State Variables

$earthquakeTree$: RedBlackBST $< \mathbb{R}$, EarthquakeT $>$
$cityProv$:$<>$
$frequency$: $\mathbb{Z}$
$averageMag$: $\mathbb{R}$
$populationDensity$: $\mathbb{R}$
$rating$: $\mathbb{Z}$

Access Routine Semantics

RiskAssessment(bst, location):

- transition: earthquakeTree := bst,
  cityPov := getCityProv(location, SearchEarthquakes.searchEarthquakeInCircle(bst, location, 100)),
  frequency :=getFrequency(),
  averageMag :=getAverageManitude(),
  populationDensity := getPopulation() ,
  rating := OverallRating(frequency, averageMag, populationDensity)

getRisk():

- output: $out := rating$

getCity():

- output: $out := cityProv[0]$

getFrequency():

- output: $out := frequency$

getMag():

- output: $out := averageMag$

getPoplationDensity():

- output: $out := populationDensity$

nearestLowerRiskCity(graph):

- output: $out := min.to()$ such that $min \in graph.adj(getCity) \wedge RiskAssessment(earthquakeTree, getLocation(min.to()).getRisk() < rating \wedge \forall(e : Edge \mid e \in graph.adj(getCity) \wedge RiskAssessment(earthquakeTree, getLocation(e.to()).getRisk() < rating : e.weight \geq min.weight)$

Local Functions

getLocation : String $\rightarrow$ PointT
getLocation($city$) $\equiv$ $cityPost.getPoint()$ such that $\forall(c : CityPostT \mid c \in$ seq of CityPostT $\wedge$ $c.getCityName = city : cityPost = c)$

getCityProv : PointT, seq of EarthquakeT $\rightarrow$ tuple of (String, String)
getCityProv($location, eqList$) $\equiv < eq.getPlace(), eq.getNameOfProv() >$ such that $\forall(e : EarthquakeT \mid c \in eqList : location.distanceTo(e.getPointT()) \geq location.distanceTo(eq.getPointT()))$

getPopulation :
getPopulation $\equiv$ $city.getPopDensity()$ such that $\forall(c : CityT \mid c \in$ seq of CityT $\wedge$ $c.getCityName = cityProv[0] \wedge c.getProvince = cityProv[1] : city = c)$

getFrequency : seq of EarthquakeT $\rightarrow \mathbb{Z}$
getFrequency(s) $\equiv |s|$

getAverageMagnitude : seq of EarthquakeT $\rightarrow \mathbb{R}$
getAverageMagnitude(s) $\equiv +(e : EarthquakeT \mid e \in s : e.getMag())/|s|$

frequencyRating : $\mathbb{R} \rightarrow \mathbb{Z}$
frequencyRating($frequency$) $\equiv$ $(frequence < 1 \Rightarrow 0 \mid frequence \geq 1 \wedge frequence < 10 \Rightarrow 1 \mid frequence \geq 10 \wedge frequence < 100 \Rightarrow 2 \mid frequence \geq 100 \wedge frequence < 1000 \Rightarrow 3 \mid frequence \geq 1000 \Rightarrow 4)$

magnitudeRating : $\mathbb{R} \rightarrow \mathbb{Z}$
magnitudeRating($averageMag$) $\equiv$ $(averageMag < 1 \Rightarrow 0 \mid averageMag \geq 1 \wedge averageMag < 4 \Rightarrow 1 \mid averageMag \geq 4 \wedge averageMag < 6 \Rightarrow 2 \mid averageMag \geq 6 \wedge averageMag < 7 \Rightarrow 3 \mid averageMag \geq 7 \Rightarrow 4)$

populationdensityRating : $\mathbb{R} \rightarrow \mathbb{Z}$
populationdensityRating($populationdensity$) $\equiv$ $(populationdensity < 1000 \Rightarrow 0 \mid populationdensity \geq 1000 \wedge populationdensity < 5000 \Rightarrow 1 \mid populationdensity \geq 5000 \Rightarrow 2)$

overallRating : $\mathbb{Z}, \mathbb{R}, \mathbb{R} \rightarrow \mathbb{Z}$
overallRating(f, a, p) $\equiv$ $frequencyRating(f) + magnitudeRating(a) + populationdensityRating(p)$

View Interface Module

Interface Module

ViewList

Uses

PointT

Syntax

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
display	seq of EarthquakeT, PointT		

display by magnitude Module

Module inherits ViewList

DisplayByMagnitude

Uses

PointT, Sort

Syntax

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
display	seq of EarthquakeT, PointT		

Semantics

State Variables

N/A

Access Routine Semantics

display(eqList, location):

- print(e.getMag(), e.getColor(), e.getDate().getYear(), e.getPlace())
  for all e $\in$ Sort.sortByMagnitude(eqList)

display by distance Module

Module inherits ViewList

DisplayByDistance

Uses

PointT, Sort

Syntax

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
display	seq of EarthquakeT, PointT		

Semantics

State Variables

N/A

Access Routine Semantics

display(eqList, location):

- print(location.distanceTo(e.getPointT()), e.getMag(),e.getColor(), e.getDate().getYear(), e.getPlace()) for all e $\in$ Sort.sortByMagnitude(location, eqList)

view risk assessment Module

Module

ViewRisk

Uses

RiskAssessment, RedBlackBST, PointT

Syntax

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
showRisk	RedBlackBST, seq of CityPostT, PointT, CityGraph		

Semantics

State Variables

N/A

Access Routine Semantics

display(bst, loc, s, graph):

- print(ra.getRisk(), ra.getCity(),ra.getFrequency(), ra.getMag(), ra.getPoplationDensity(), ra.nearestLowerRiskCity(graph)) such that ra = RiskAssessment(bst, loc)

Local Functions

initGraph : $RiskAssessment, seqofCityPostT, CityGraph \rightarrow CityGraph$
initGraph(ra, s, graph) $\equiv$ graph.addEdge(e) such that e = Edge(ra.getCity(), cityPost.getCityName, ra.getCity().getPoint().distanceTo(cityPost.getCityName.getPoint())) for all cityPost $\in$ s and ra.getCity().getPoint().distanceTo(cityPost.getCityName.getPoint()) $<$ 100

Controller Module

Module

Controller

Uses

CSVreader, SearchEarthquakes, ViewList, ViewRisk

Syntax

Exported Types

Controller = ?

Exported Access Programs

Routine name	In	Out	Exceptions
init	RedBlackBST, GeoCollection, seq of CityPostT		
search	RedBlackBST, PointT, $\mathbb{R}$		
updateViewOfList	ViewList		
updateViewOfRisk	RedBlackBST, PointT, seq of CityPostT, CityGraph		

Semantics

State Variables

location : PointT
eqList : seq of EarthquakeT

Access Routine Semantics

init(bst, geoCollection, cityPostList):

- transition:
  The states of bst, geoCollection, cityPostLists are modified by accessing the routes of readEarthquakesBST, readPopulation, and readCityPosition in CSVreader module.

search(bst, loc, radius):

- transition: location := loc,
  The states of variable eqList is modified by accessing the route of searchEarthquakeInCircle in SearchEarthquakes module.

updateViewOfList(view):

- print the list of earthquakes by accessing the route of display in ViewList module.

updateViewOfRisk(bst, loc, s, graph):

- print the the risk assessment result by accessing the route of showRisk in ViewRisk module.

3.2 state machine diagrams

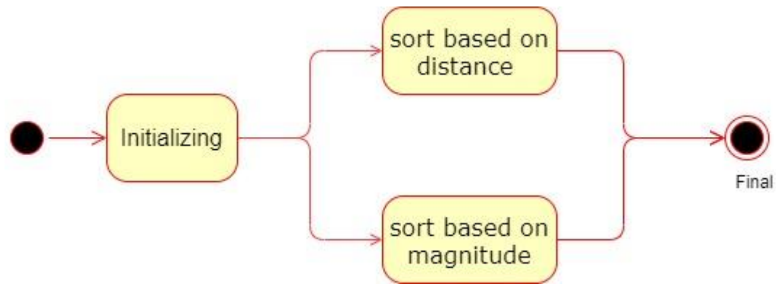


Figure 1 state machine diagram for Sort class

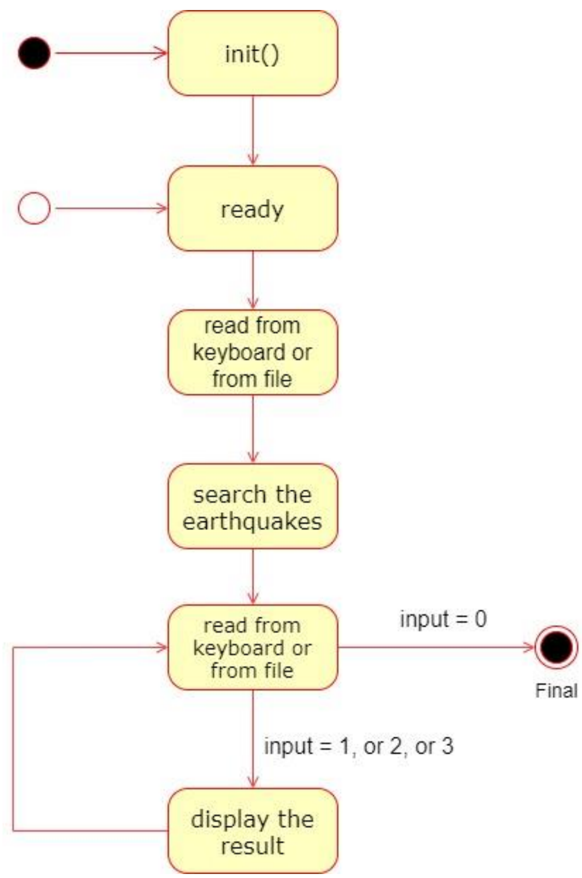


Figure 2 state machine diagram for MCVDemo class

4 Internal Review and Evaluation

class	requirements	Details(uses)
CSVreader	Read earthquakes	Read data to generate color rating then construct *EarthquakeT*, store them in *EarthquakeBag*
	read earthquakeBST	Read data to generate color rating then construct E*arthquakeT*, store them in *RedBlackBST*
	read population	Read data to construct *CityT*, store them in *GeoCollection*
	read city position	Read data to construct *CityPostT*, store them in a sequence
CityPostT	Accurately represent geographical location of a city	Use *PointT* to represent city position
CityT	Accurately represent city data	
EarthquakeT	Accurately represent earthquake data	Use *PointT* to represent earthquake position, use *LocalDateTime* to represent earthquake time
GeoCollection	Data structure to store cities	Store *CityT*
RedBlackBST	Data structure to store something	Store T using *Queue*
PointT	Accurately represent geographical location	
Edge	Represent distance between cities as weight	Represent connections between two cities
CityGraph	Represent cities using a directed weighted graph	Construct a graph made of *Edge*
EarthquakeBag	Data structure to store earthquakes	Store *EarthquakeT*
Queue	An implementation of queue	LinkedList representation of queue
SearchEarthquakes	Search earthquakes within a given radius of input location	Given a position(*PointT*) and radius, search required *EarthquakeT*s in earthquake database(*RedBlackBST*)
Sort	Sort earthquakes based on magnitude	use *quick sort* to sort *EarthquakeT* by magnitude
	Sort earthquakes based on distance	Use *insertion sort* to sort *EarthquakeT* based on distance between earthquake location and given location(*PointT*)

RiskAssessment	Calculate a risk rating using both earthquake and population data	Given a position(*PointT*) and earthquake database(*RedBlackBST*), assess based on earthquake *frequency*, assess based on earthquake *magnitude*, assess based on *population density(60% accuracy)* of the city of nearest earthquake found in *GeoCollection*, then add up the results
	find the nearest city that has a lower risk	Given a *CityGraph*, find the nearest city with a lower earthquake risk
ViewList	Display earthquakes	Display a sequence of *EarthquakeT*s based on given location(*PointT*) within ***2*** seconds(*100% accuracy*)
DisplayByMagnitude	Display earthquakes by magnitude in descending order	*Sort* by magnitude, then displays *EarthquakeT* information in descending order of magnitude given a sequence of earthquakes and a *PointT*
DisplayByDistance	Display earthquakes by distance in ascending order	*Sort* by distance between a position and earthquakes, then displays *EarthquakeT* information in ascending order of distance given a sequence of earthquakes and a *PointT*
ViewRisk	Display risk and earthquake information and nearest lower risk city	Given *RedBlackBST* and *PointT*, Display earthquake risk of the point within 4 seconds, display historical earthquake information and population information, display nearest lower risk city got from *RiskAssessment*
Controller	Read data	Use *CSVreader*
	search earthquake	Use *SearchEarthquakes*
	display earthquake	Use *ViewList*
	display risk	Use *ViewRisk*
Demo	A sample run	
ExperimentsSearch	Compare which algorithm is better	

Our project design follows the MVC design pattern. The model is completely separated from view and controller. The model is separated into 5 parts: ADT, search, sort, graph, and riskAssessment. Search and sort use different ADTs, and riskAssessment uses graph and ADTs. The ADT part contains ADTs for position (PointT), city (CityT), city position (CityPostT), and earthquake (EarthquakeT). The search module searches earthquakes within a radius using RedBlackBST. The sort module sorts earthquakes using insertion sort and quick sort. The graph module implements a directed weighted graph between cities. The risk assessment module assesses the risk for a position and finds the nearest city with a lower risk. The view displays earthquake information and risk assessment result using sort and riskAssessment. The controller reads data from csv files, searches earthquakes and updates view. In these modules, only methods that are called by other modules are public, otherwise they are private. Generally speaking, each module has distinct functionalities, while in each module, components are closely connected.

Overall, our program is doing great in encapsulation and modularity. We achieved good reusability and verifiability of code. The quality is ensured by following MIS strictly. The use of BST and quick sort improve the performance, and the use of MVC and strategy design pattern improve the quality and maintainability. The program is not required to be robust, but it is required to be correct and reliable. We do not achieve the goal of making a GUI, but we will work on it in the future to make our program reliable.