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

March 23, 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 project 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 5**](#_Toc37514678)

[**2 Module Decomposition 5**](#_Toc37514679)

[**3 MIS 6**](#_Toc37514680)

[3.1 MIS 6](#_Toc37514681)

[3.2 state machine diagrams 7](#_Toc37514682)

[**4 Internal Review and Evaluation 8**](#_Toc37514683)

# 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 densit**y**.

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](FinalProj_UML_V7.png)(🡨click on the hyperlink to view in full resolution)

A close up of a map

Description automatically generated

# 3 MIS

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

## 3.1 MIS

## 3.2 state machine diagrams

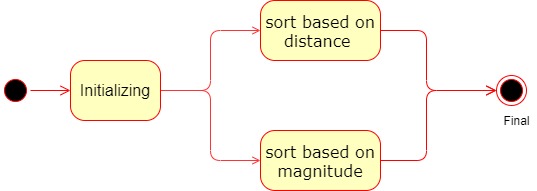


Figure 1 state machine diagram for Sort class

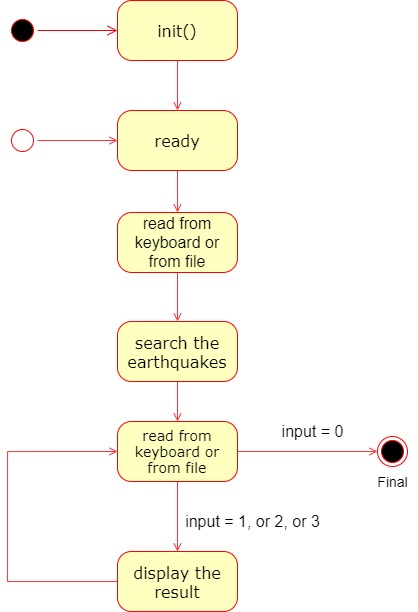


Figure 2 state machine diagram for MCVDemo class

# 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.