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 | 23 March, 2020 | 1 |
|  |  |  |
|  |  |  |

**Group Member and Roles**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student Number | Role in the Project | Responsibility |
| Kan Hailan | 400207974 | client & tester |  |
| Sembakutti Kalindu | 1046206 | researcher & programmer |  |
| Tao Haoyang | 400171589 | designer & programmer | Module decomposition, implement ADT modules. |
| Ye Fang | 400273067 | project leader & programmer | Manage the project, produce the prototype, implement sort and controller and view modules |

**Contribution**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Role(s)** | **Contributions** | **Comments** |
| Kan Hailan | client & tester |  |  |
| Sembakutti Kalindu | researcher & programmer |  |  |
| 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; finish MIS (14 modules) and UML state machine diagrams |  |

**Executive Summary**(Sembakutti Kalindu)

An abstract about the project not more than 200 words.

**Table of Contents**

# 1 Overall Design description (Sembakutti Kalindu)

# 2 Module decomposition and UML, view of uses relationship

* 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 have 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 CityPosT 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 and UML state machine diagrams

# There are totally 19 modules for this system. We have the MISs for all except MCVDemo modules in the following section. We also include two state machine diagrams for Sort and MCVDemo classes.

## 3.1 MIS

* for each class, a description of the interface (public entities), and make sure that there is a description of the semantics (behaviour) of each pub lic method in the class, as well as a description of the syntax;;
* for each class, a description of the implementation (private entities), including class variables - include enough detail to show how the class variables are maintained by the methods in the class; you should include two UML state machine diagrams for two most interesting classes in your implementation;

(Sembakutti Kalindu): finish the MIS of this three class: CSVreader, RedBlackBST, Earthquake T, Queue

(Fang Ye) finish the MIS of following modules and two UML state machine diagrams:

CityPostT, CityT, PointT,CityGraph, , GeoCollection, Edge, GeoCollection, DisplayByMagnitude, DisplayByDistance, DisplayInterface, DIsplayManager, RiskAssessement, SearchEarthquake, Sort

## 3.2 state machine diagrams

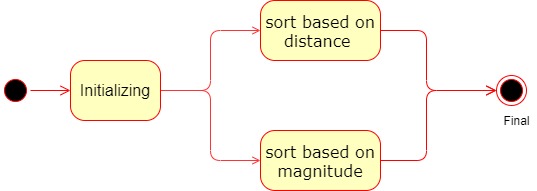


Figure 1 state machine diagram for Sort class

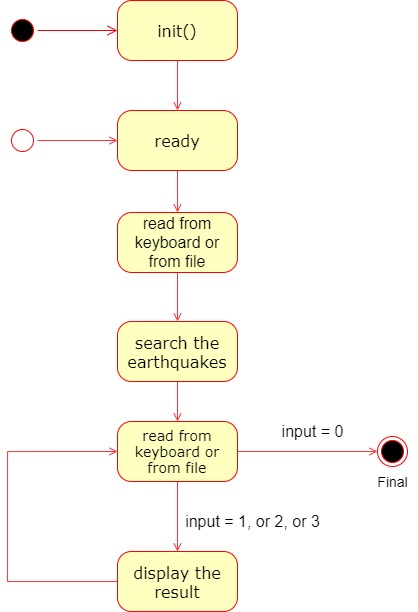


Figure 2 state machine diagram for MCVDemo class

**4 internal review/evaluation(**Kan Hailan**)**

|  |  |  |
| --- | --- | --- |
| **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% accurancy)* 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% accurancy*) |
| 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 |  |

* include a trace back to requirements in each class interface;
* an internal review/evaluation of your design.