Virtual Environment For Education

by

Atul Thapliyal

A report submitted in partial fulfillment

of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

**\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_**

Dr. Nick Flann Dr. Curtis Dyreson Dr. Dan Watson

Major Professor Committee Member Committee Member

UTAH STATE UNIVERSITY

Logan, Utah

2012

# 

# ABSTRACT

Virtual Environment for understanding science

by

Atul Thapliyal, Master of Science

Utah State University, 2012

Major Professor: Dr. Nicholas Flann

Department: Computer Science

Simulations provide an environment to experiment safely, openly, and repeatedly for learning mastery. However, many simulation environments experienced within a classroom fail to include the automated assessment components and also the automated data collection. Even when a measure of standard assessments is included, often it fails to account for the unpredictable nature of decision-making within the complex, 3D, open-ended simulation environment. Embedding assessments within a virtual simulation environment poses several issues. First, the program must provide assessments that will fulfill educational requirements that will not take the learner cognitively “away” from their activities. Second, the program must provide an engaging game-like experience for educational purposes. Third, it must provide assessments that maximize the unique capability inherent within digital deliveries, that allows for geographically disparate and asynchronous schedules between instructor and learner. This report addresses each of the above concerns through an integration of the class room requirements and simulation affordance. Created within an educational curriculum, this game is designed to function as a stand-alone module to teach and evaluate understandings about core concepts.

# ACKNOWLEDGEMENTS

First, I would like to thank Utah State University, Computer Science department for providing me with an opportunity to pursue my Masters degree. I am grateful to Dr Nicholas Flann for his interest and guidance in this project.

I would like to thanks Dr. Curtis Dyreson and Dr. Dan Watson for both their input and interest in this report. Dr. Curtis Dyreson introduced me to good database principles which helped me in designing the data collection application. I am grateful to Dr. Brett Shelton for supporting and guiding me throughout this project. His domain knowledge of instructional technology and his clear vision were vital for the success of this project.

Last but not the least, I would like to thank Anuj, Ashish and my parents for their support and encouragement in all my educational pursuits.

Atul Thapliyal

# TABLE OF CONTENTS

[ABSTRACT i](#_Toc247980224)

[ACKNOWLEDGEMENTS ii](#_Toc247980225)

[TABLE OF CONTENTS iv](#_Toc247980226)

[CHAPTER 1 INTRODUCTION 1](#_Toc247980227)

[1.1. Design Goals 3](#_Toc247980228)

[1.2. Software Engineering Practices 6](#_Toc247980229)

[1.3. Climate Vs West Nile Virus Data 7](#_Toc247980230)

[1.4. Technical Enhancements 8](#_Toc247980231)

[1.5. New Features 8](#_Toc247980232)

[1.6. Report Contents 9](#_Toc247980233)

[Chapter 2 MICROMAPS DESIGN DECISIONS 10](#_Toc247980234)

[2.1. Why LMSERVER.java? 10](#_Toc247980235)

[2.2. Sorting and Missing Data in Climate Application 11](#_Toc247980236)

[2.3. Design of Vertical Scale in Climate Data for Plotting Time Series Plot 13](#_Toc247980237)

[2.4. Design of Horizontal Scale in Climate Data 13](#_Toc247980238)

[2.5. Scaling Data Values for Accurate Plot 13](#_Toc247980239)

[2.6. West Nile Virus Logarithmic Scale 13](#_Toc247980240)

[2.7. Adaptive Drop Down Options 14](#_Toc247980241)

[Chapter 3 TECHNICAL ENHANCEMENT 17](#_Toc247980242)

[3.1. SWING Vs AWT 17](#_Toc247980243)

[3.2. Java Web Start vs. Applet 18](#_Toc247980244)

[Chapter 4 MICROMAPS ARCHITECTURE CHANGES 20](#_Toc247980245)

[4.1. Presentation Layer 20](#_Toc247980246)

[4.2. Java Web Start 21](#_Toc247980247)

[4.3. Domain Logic Layer Changes 21](#_Toc247980248)

[4.4. Data Source Layer Changes 21](#_Toc247980249)

[4.5. Unified Modeling Language (UML) Class Diagrams 22](#_Toc247980250)

[Chapter 5 NEW FEATURES 25](#_Toc247980251)

[5.1. Missing Data 25](#_Toc247980252)

[5.2. Logarithmic Scale 26](#_Toc247980253)

[5.3. Time Series Plot of Climate Data 28](#_Toc247980254)

[5.4. Nice Numbers for Graph Labels and Leland Wilkinson’s algorithm 32](#_Toc247980255)

[5.5. Switching Gears! 33](#_Toc247980256)

[Chapter 6 SUMMARY AND LESSONS LEARNED 34](#_Toc247980257)

[6.1. Lessons Learned 34](#_Toc247980258)

[6.2. Pollution Data 35](#_Toc247980259)

[6.3. Testing 36](#_Toc247980260)

[REFERENCES 38](#_Toc247980261)

[APPENDIX A 41](#_Toc247980262)

[A.1. System Context & Scope 42](#_Toc247980263)

[A.1.1 Scope of the System 42](#_Toc247980264)

[A.1.2 Problems with Current System 42](#_Toc247980265)

[A.1.3 Use Case List 43](#_Toc247980266)

[A.2. FUNCTIONAL REQUIREMENTS 43](#_Toc247980267)

[A.2.1. Functional Requirements 43](#_Toc247980268)

[A.3. Look, Feel and Use Requirements 43](#_Toc247980269)

[A.3.1. User Interface Requirements 43](#_Toc247980270)

[A.3.2. Ease of Use Requirements 44](#_Toc247980271)

[A.4. Proposed Solutions 44](#_Toc247980272)

[A.4.1. Solutions Proposed Based on Our Discussions 44](#_Toc247980273)

[A.5. Future Requirements 47](#_Toc247980274)

[APPENDIX B 48](#_Toc247980275)

[B.1. System Context & Scope 49](#_Toc247980276)

[B.1.1. Scope of the System 49](#_Toc247980277)

[B.1.2. Problems with Current System 49](#_Toc247980278)

[B.1.3. Use case List 49](#_Toc247980279)

[B.2. Functional Requirements 50](#_Toc247980280)

[B.3. Look, Feel and UI Requirements 50](#_Toc247980281)

[B.3.1. User Interface Requirements 50](#_Toc247980282)

[B.4. Proposed Solutions 51](#_Toc247980283)

[B.4.1. Solutions Proposed Based on Our Discussions 51](#_Toc247980284)

[B.5. Future Requirements 51](#_Toc247980285)

[APPENDIX C 52](#_Toc247980286)

[APPENDIX D 62](#_Toc247980287)

[APPENDIX E 68](#_Toc247980288)

**LIST OF FIGURES**

[**Figure 1 A Time Series Plot of Climate Data 4**](#_Toc247980207)

[**Figure 2 Sorting and Clustering of West Nile Virus Infection Data according to missing value 5**](#_Toc247980208)

[**Figure 3 New transparent SWING based tool tip compared to old opaque AWT based tool tip 8**](#_Toc247980209)

[**Figure 4 Applet fires twenty one queries (one for each station) and gets thirty one (one for each day in month) results per query via LMSERVER\_CLIMATE for the above plot. Climate Stations with missing values are sorted at the bottom of graph. 12**](#_Toc247980210)

[**Figure 5 The initial view of the drop down menu for selecting options. 14**](#_Toc247980211)

[**Figure 6 The drop down menu when West Nile Virus is chosen as a data group. 15**](#_Toc247980212)

[**Figure 7 The drop down menu when Climate is chosen as the data group. 16**](#_Toc247980213)

[**Figure 8 High level architectural design of Micromaps application 22**](#_Toc247980214)

[**Figure 9 UML Class diagram for client side classes for plotting graphs 23**](#_Toc247980215)

[**Figure 10 Missing data sorted and clustered according to a rank button that is active (Sorted by reference year 2003). 25**](#_Toc247980216)

[**Figure 11 Normal Scale for Infection Rate for year 2005 and 2002 26**](#_Toc247980217)

[**Figure 12 An example of logarithmic scale for infection rate for year 2003 and 2005 27**](#_Toc247980218)

[**Figure 13 Time series plot of maximum temperature for Utah County in Utah for March 2008. 28**](#_Toc247980219)

[**Figure 14 Snowfall in Denali county of Alaska for February 2004 (Leap year). The Horizontal scale adjusts according to number of days in a month. 29**](#_Toc247980220)

[**Figure 15 Minimum temperatures in North Slope county Alaska December 2007. 30**](#_Toc247980221)

[**Figure 16 Precipitation in Santa Barbara county California 2006. The Vertical scale captures even the small precipitation values. 31**](#_Toc247980222)

[**Figure 17 Nice scales ranging from -37.5 to +37.5 with a step size of 12.5. 32**](#_Toc247980223)

# CHAPTER 1 INTRODUCTION

This project was developed as a part of National Science Foundation Discovery Research K-12 Grant (NSF 09-602). Development of interactive simulation systems for traditional education has been increasing. However, assessing student learning results in a 3D environment has been a challenge for educators. This project is focused on designing an enhanced educational tool with embedded assessment and data collection in a 3D environment.

This project also provides a space for safe experimentation by students which has been a major concern in school laboratories. Eighth grade science teachers from across

the Western U.S. were brought together in a focus group environment to discuss desired aspects to be covered by a simulation environment, as well as the type, style, and functionality of the embedded assessment to meet their educational needs. The creative team used teacher suggestions in designing both the environment of the game play and the embedded assessments. This environment prototype was then presented to two classes of high school students in a rural charter high school for pilot testing of the environment and the assessment feature. This research design engaged elements of observational

study (Creswell, 2008) and grounded theory (Corbin & Strauss, 1990). Data was collected through observation, data mining, focus group feedback, and a metacognitive

awareness inventory (MAI) survey instrument attached to the gaming environment. The teachers themselves, in connection with the university, were considered the

subject matter experts (SMEs) and advisors in creating the assessments and ensuring adequate levels of complexity in the nature of the simulated environments.

The use of multiple SMEs ensured both accuracy

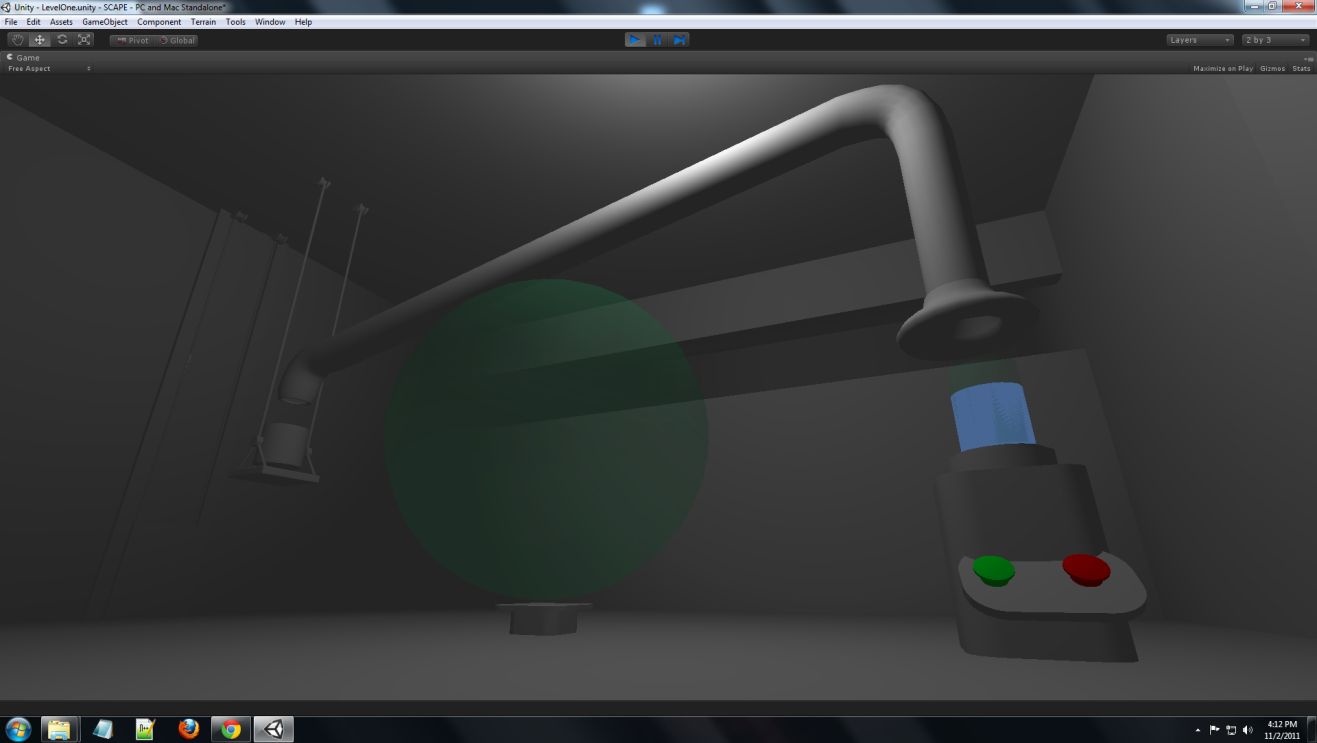
This game is designed from a first-person shooter game perspective (i.e., the student/player is directly seeing and interacting with the environment) to teach *Utah state science standard one* from the core standards.

* This game and standard is arranged to teach eighth grade students the nature of changes in matter.
* Four sequential levels engage the four objectives and subsequent sub-goals under each objective for this standard.
* This computer based interactive environment, in connection with classroom instruction, will cover the components of standard one of the eighth grade core.

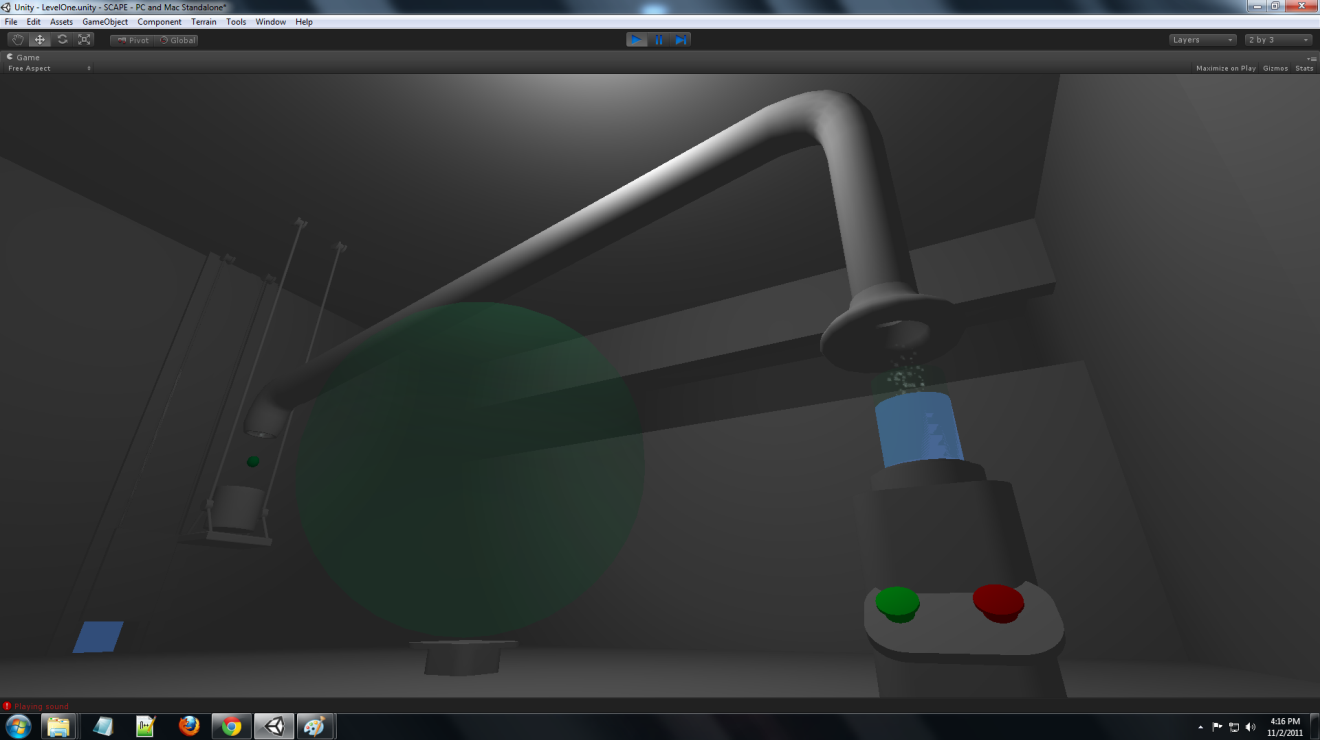
The student/player begins by finding themselves in a single room with a single locked door. The student/player has the option of both overhead and in-line view modes. The player, through narration, suspects that he or she has been abducted by aliens that are experimenting on them to observe the problem solving capabilities of humans. The shadows on the room from an overhead perspective indicate that they are being observed. The student/player must solve a series of tasks and come to understand what is happening to the properties of water in order to open the door and escape. Within each environment there are hotspots the player can click on to bring up a magnification window showing what is happening at the molecular level. As students solve the various issues and come to understand the principles involved they are released from the room, only to find themselves in another room where they need to solve another issue to progress through the four levels of the game.

Level 1:

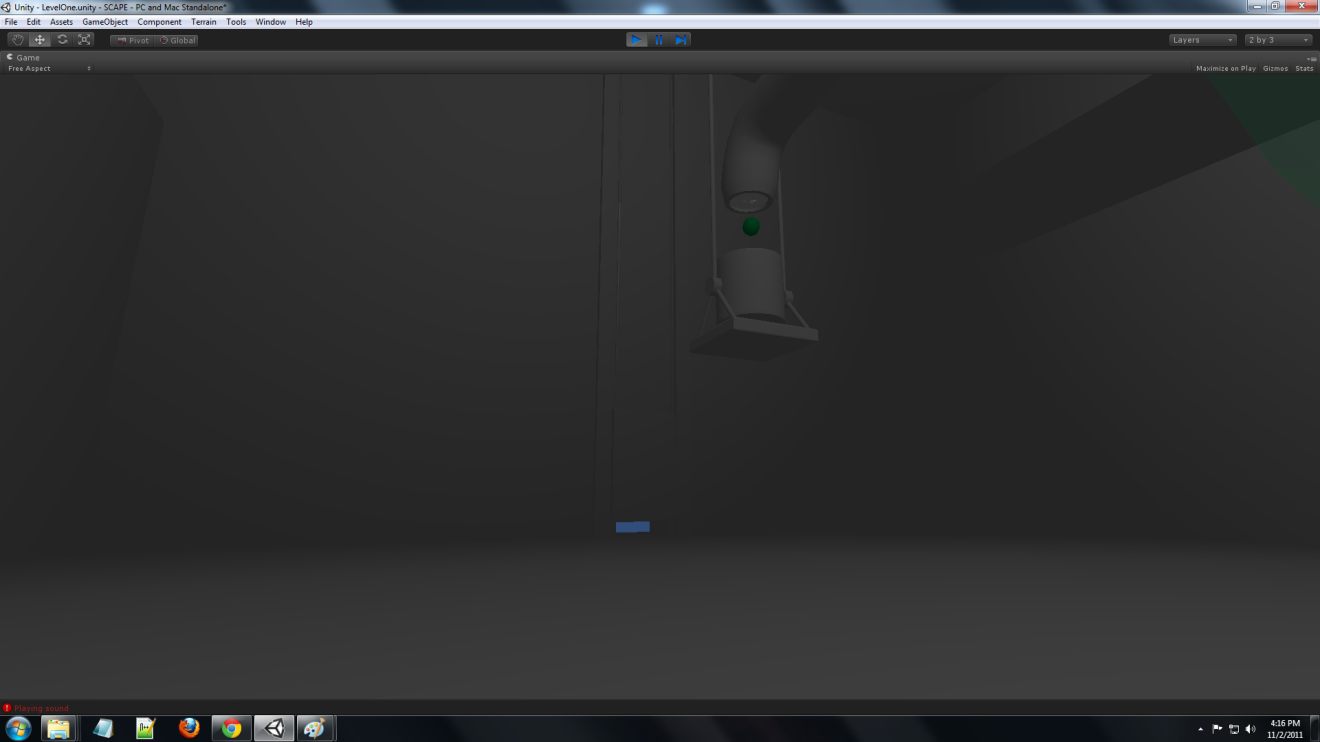
Beginning in the first level of the game, student/players wake up in a simple room filled with various elements from which they can work. This first room is designed to teach students the physical properties of water. The student player must determine the methods to create ice, standard water, and steam (gas) from provided elements in the room that will allow them to move to the next level. How this looks in the environment is visualized below within early development imagery:



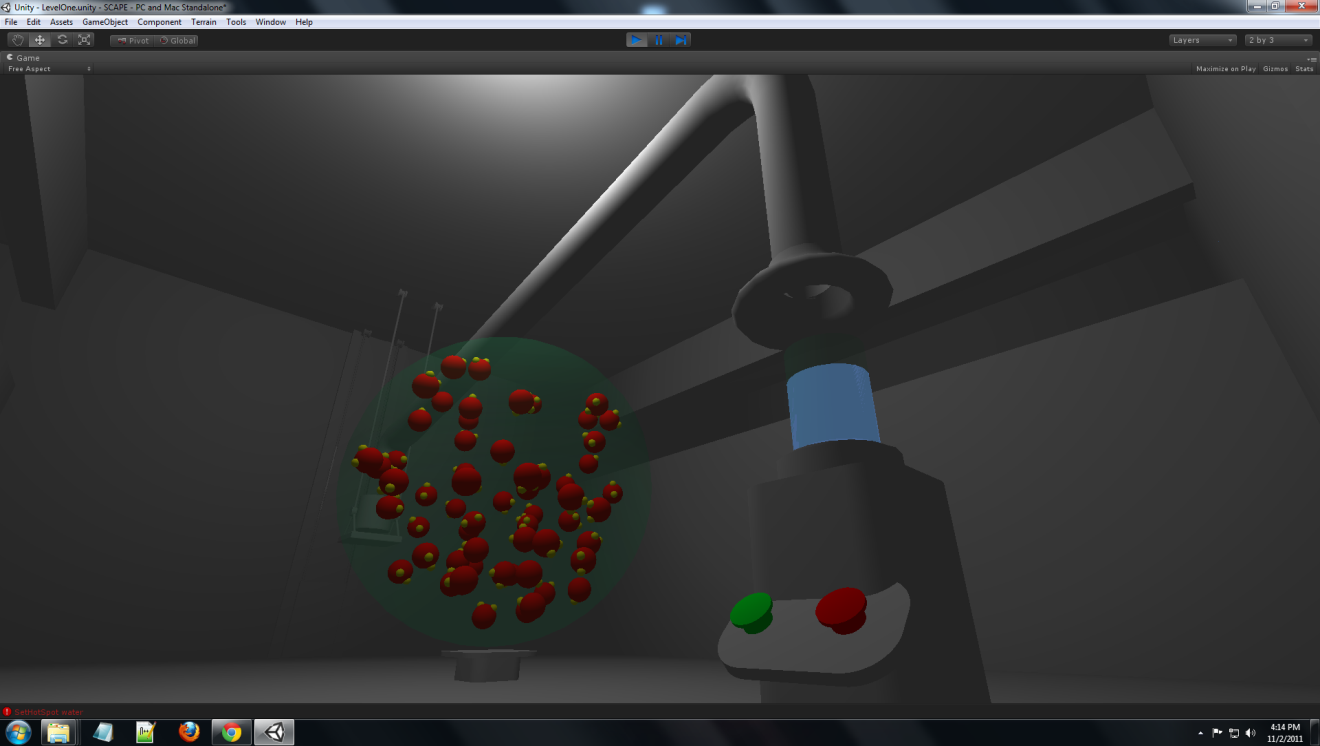
Students wake up in this room and can explore as a single shooter style game. In the above gallery, shadows will indicate that students are being watched by unknown figures. Within the environment are various hotspots for student observation and interaction. Should the students click on the red button, they will observe the following two changes:



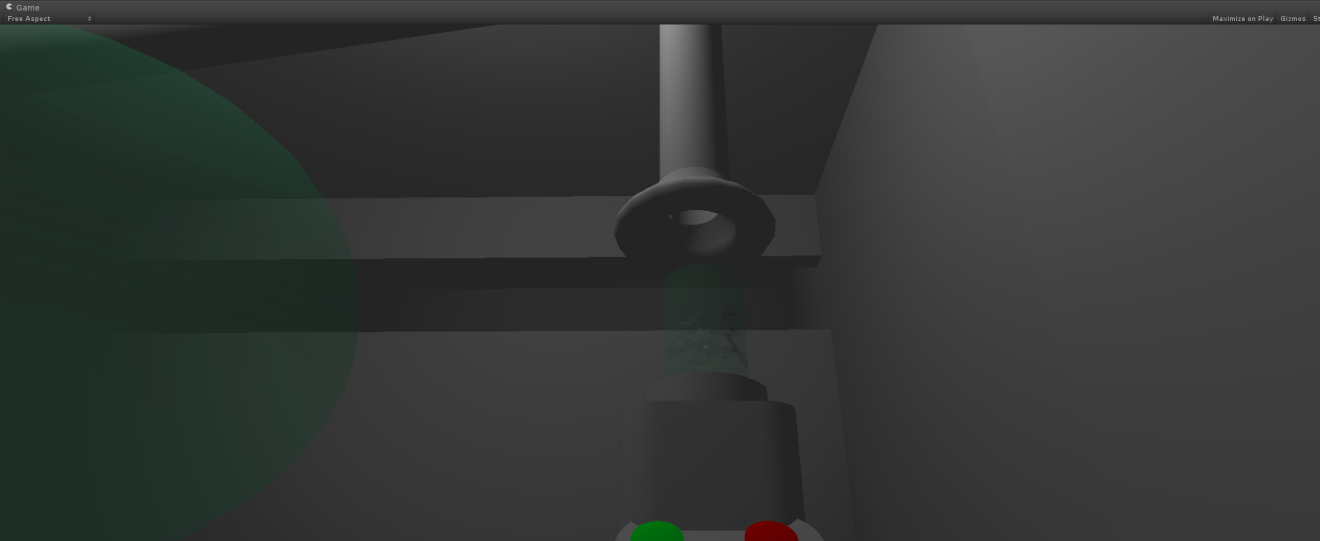
First, you will note that the blue basin of water has begun to boil and there are particles rising into the hood.



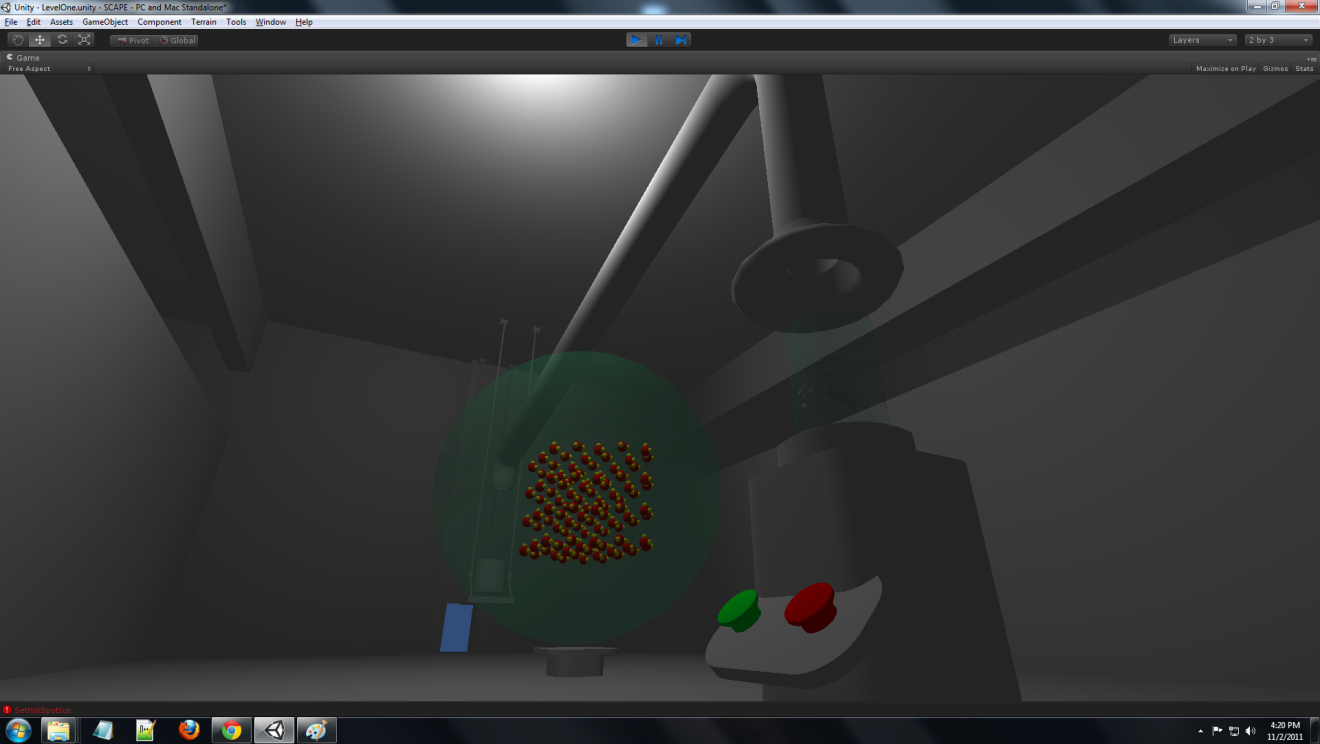
Second, students will be able to observe the droplets (condensed water vapor ) falling into the bucket, and that this process is slowly raising the door in the wall. This proves crucial to students being able to progress in the game. To help students in understanding what is occurring at a molecular level, students can click on the hood, the area of where the activity is collecting, and the molecular view appears in the large central orb, as viewed below.



On a similar score, students may also click the green button. The steam will stop, the bucket will stop filling and the bucket of water will begin to turn clear as below.



When students click on the bucket, again the area of molecular activity, they again activate the molecular view in the orb.



Note as well that the alignment of the molecules has also mimicked the nature of ice. The essential reason students are to understand and view these molecular alignments relates to how students will be required to apply the information. When students travel through the door they will be required to answer a series of questions related to the molecular nature of the water in demonstrating competency of the concepts before they can move on to level two. They may continue to explore and work on this level as long as it takes for them to understand the concepts and develop what they need.

LIST OF ASSETS FOR LEVEL ONE

(2 bucket (one with liquid, other to collect), platform, hood piping, furnace with two buttons and hot plate, orb (globe) with platform, ropes and pulleys (two), door,) (room itself includes galleries on either side with individual doors, tile floors, and chipped and pealing paint as part of the art of the environment).

Design of the Assessment-

Assessment in the interrogation level when students pass the required tasks within the experimentation room are structured within the replay function. Students will watch a recording of their actions from the previous room and be asked questions regarding either the interaction occurring or the intent of the student in creating that interaction. Currently the model is pausing at each time we have the hotspots for the molecules appear, which would indicate student input reached a certain level of interaction for the elements. At those molecular level interactions we have a question posed on a separate GUI left of the replay for the students to answer questions with hierarchical questions answers (good, better, best, or utilizing misconceptions identified within the Utah Core) to be determined with content experts. Recording will automatically occur within the simulation from the moment the person wakes up in the environment until they enter the assessment room.

Curriculum Standard links level one:

These activities connect to

* objective one,
* sub-goal A (differentiating differences in physical properties),
* sub-goal B (classifying substances based on physical properties), and
* sub-goal C (Investigate and report on physical properties of a particular substance);
* objective two,
  + sub-goal A (identifying observable evidence of physical change;
* objective three,
  + sub-goal A (Identify kinds of energy as substance undergoes physical change),
  + sub-goal B (Relate amount of energy in the motion of molecules in substance),
  + sub-goal C (Measure and graph relationship between states of water and temperature),
  + sub-goal E (Plan and conduct experiment, and report effects on physical changes); and
* objective 4,
  + sub-goal A (Identify reactants and products in a chemical change and describe the presence of atoms in both reactants and products).

Level 2:

From the second room the student/player finds himself/herself in another room just like the previous rooms. This third room requires students/players to explore the physical and chemical properties of an element; in this case the process of cooking an egg. To escape from this room requires the student/player to understand the physical properties of raw and cooked eggs to complete the level and flee to their final room and challenge.

As before the game includes the giant orb which displays the molecular level of what is happening to the materials in the bucket, which in this case includes an egg and one of three options for the liquid material the egg is floating in. Attached to the burner of level one we have a table with three buttons below, and on top, a variety of tubes and containers like a mad scientist lab. Three buttons correspond to three distinct containers within the web of tubes on the table. One contains water, another vinegar, and the last nitrogen. On the burner from level one the same two buttons for hot and cold are attached and another button for shake is included. Players entering this room must pick from the liquids, and either of the three preparation methods to cook the egg. This time when the heat button is pushed the burner turns hot as a result of a laser attached to the wall warming the burner. Depending on the combination the player selects the egg will either explode, crack and ooze, cook, or freeze. When the player has either destroyed the egg or feels it is ready, they may press a button on the wall which will drain the bucket and egg and feed the creature blocking entrance to the interrogation room. If the creature accepts the cooked egg, it will depart and the light will come on in the adjoining room to allow the player to enter assessment. If the egg was not properly prepared the beast spews bits of it into the room and growls at the player. This level is less about sequence as it is about properly performing the experiment.

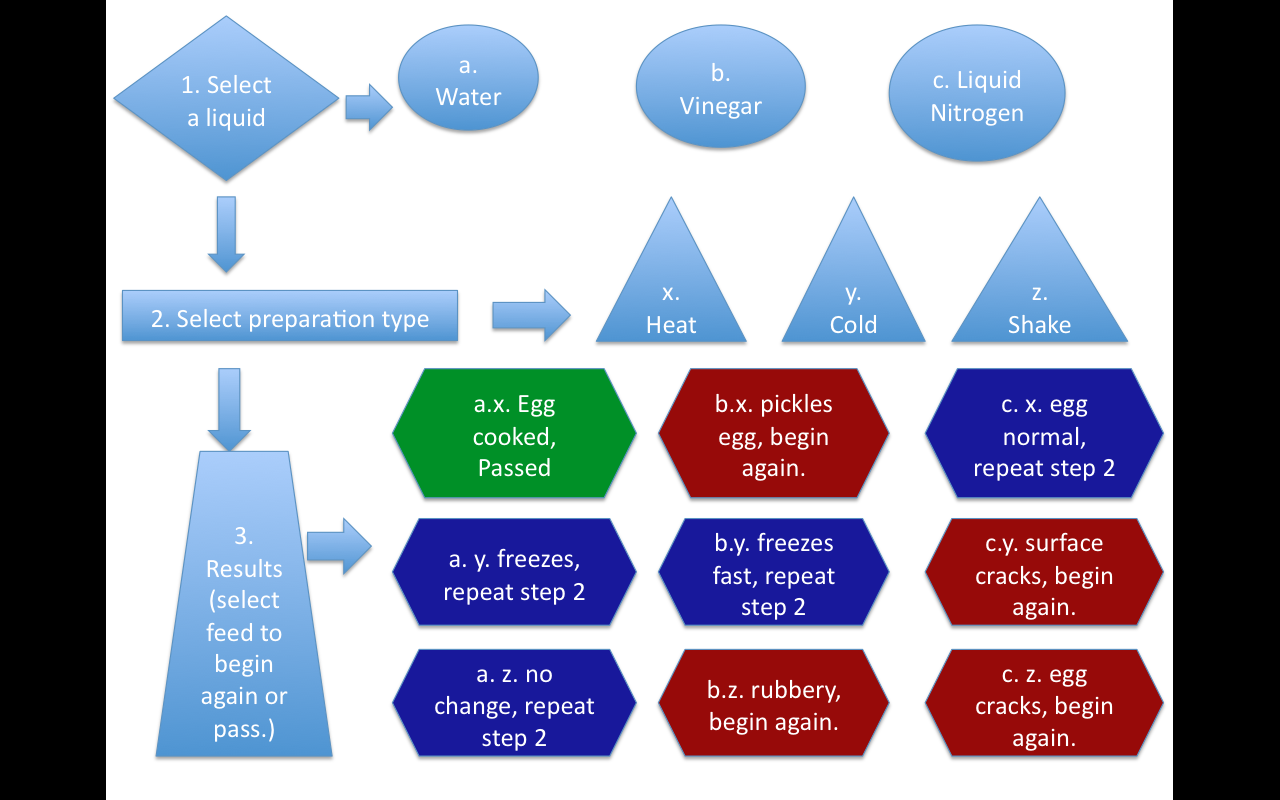


Figure : Decision tree model for level three options. (Green=success; Red=failure; Blue=may attempt another preparation method.)

List of Assets for Level 3: (1 bucket (one with egg), platform, hood piping which attaches to the ceiling, furnace with three buttons and hot plate, orb (globe) with platform, creature in doorway (visible with angry eyes and cruel rows of teeth), three buttons on a table, artistic environment still (like fermentation still), three jars filled with different colored liquids. Tubes running to the bucket, button on wall for feeding) (room itself includes galleries on either side with individual doors, tile floors, and chipped and pealing paint as part of the art of the environment).

Curriculum Standard links level three:

These activities connect to

* objective one,
  + sub-goal A (Differentiate between chemical and physical properties),
  + sub-goal B (Classify substances based chemical and physical properties), sub-goal C (Investigate and report on chemical and physical properties of a particular substance);
* Objective two,
  + sub-goal A (Identify observable evidence of a physical change),
  + sub-goal B (Identify observable evidence of a chemical change),
  + sub-goal D (Investigate the effects of chemical and physical properties);
* objective three,
  + sub-goal A (Identify kinds of energy change in a substance when undergoing chemical or physical change),
  + sub-goal B (Relate the amount of energy in a substance to the motion of molecules in the substance),
  + sub-goal D (Cite evidence showing heat change during a chemical change),
  + sub-goal E (Plan and conduct and experiment, and report the effect of adding or removing energy on chemical and physical changes);
* objective four,
  + sub-goal A (Identify the reactants and products in a chemical change and describe the presence of atoms in both reactants and products),
  + sub-goal B (Cite examples of common significant chemical reactions in daily life), and
  + sub-goal E (Research and report on how scientist have applied principles of chemistry to an application in daily life).

Embedded Assessment:

Within the structure of the game exists a dynamic back channel of information from the decision-tree algorithms outlined here as three distinct assessment variables for review by the teacher and the student. This embedded assessment model follows the work of (OUR AERA PAPER LAST SUMMER, 2010), and includes the following variables:

1. Accuracy, as assessed by recording the order of actions produced and comparing them to an existing “correct” decision tree logic mechanism. This correct model is predetermined by the content expert and the instructional designer
2. Completeness, as assessed by using the decision tree logic to determine if all activities in a task are performed
3. Timing, as assessed by measuring actual time on task with time parameters assigned to the activities. This time assignment is also related in the decision tree file

Clearly the imbedded nature of the assessment variables are focused on the performance of the players. Carolan et al. (2005) reported that the user is assessed on task completion, if the task is done with accuracy, done within a timely manner and if the preceding actions are done within the correct order. Squire (2003) stated “advances in assessment, such as peer-based assessment or performance-based assessment provide learners multiple sources of feedback based on their performance in authentic contexts” (p. 4). The value of the automated assessment is the instant feedback the user receives during the simulation.

# 

# CHAPTER 1 INTRODUCTION

Micromaps are an efficient tool for showing statistical summary of data with a geographic context. It could be seen as a data-mining tool that can search a huge database, according to options provided; in addition, Micromaps graphically represent data, along with respective geographic locations on a map. Domain experts could use Micromaps to see either changing trends or patterns in data that might not be obvious otherwise.

Carr B. Daniel [1] defines a Linked Micromap as a template that includes parallel sequences of Micromap, label and statistical summary graphics panels; Daniel also defines the logical groupings, sorting and linking for these components. Both web-based Micromaps and the use of Java as a preferred language are described in [2]. An idea to visualize West Nile Virus infection using Micromap is given by [3]. Linked Micromap was adopted at Utah Climate Center in 2005 showing West Nile Virus infection data from state cancer profiles ([http://statecancerprofiles.cancer.gov/Micromaps/](http://statecancerprofiles.cancer.gov/micromaps/)) [4]. A way to visualize pollution data using Micromap was suggested by [5]. The idea of web-based Micromap and its utilities are further described in [6] with reference to Micromap application at the Utah Climate Center. The initial development of Micromap to show West Nile Virus infection was done by Gopi Krishna Chapala [7]. Ravi Kiran Surampudi [8] started working on time series plot of climate data and Vineela Vallabhaneni [9] extended his work resolving over plot of climate stations in county maps.

This project involves enhancing some of the existing features and developing new ones. Some of the new features include accurate clustering of data according to missing values, an accurate logarithmic scale and inclusion of climate data with new time series plotting. A new scaling algorithm was designed for horizontal as well as vertical scaling of data so that time series plots are neat and accurate.

This project also resolved some of the errors with previous implementations. We implemented Leland Wilkinson’s [10] algorithm for generating Nice Numbers for graph labels. Our vertical scale in time series plot uses this algorithm to generate Nice Numbers as labels given minimum and maximum value for a dataset. The climate database is behind a firewall we had to de-couple the access mechanism so that Applets running on a client machine could query it.

Another aspect of this project is following good software engineering principles. We used a hybrid of water fall and iterative software development methodology. The entire project was carried out in two phases. First phase involved West Nile Virus data group and second phase involved climate data group. Each phase started with a requirement analysis sub-phase. All requirements where captured in two requirement analysis document [APPENDIX A, B] which were used as guiding documents during the development. These two documents kept changing during the course of development and now serve as the documentation for understanding the design decisions on our project. Before start of any coding, Subversion was integrated with Net Beans integrated development environment. Once requirements were finalized, coding phase involved implementing them. An iterative approach was followed, and we kept enhancing our applications in steps. There were many intermediate versions of application until final beta version of application was deployed. All these versions of application are maintained in our subversion repository.

## Design Goals

The main design goals were to enhance the original application by making it more accurate and by adding new features. The predecessor of this application did not have an accurate logarithmic scale; in addition, the predecessor was not able to switch back and forth between logarithmic and normal scale. The original interface for selecting different data groups efficiently was absent; however, the drop down options in the new version changes according to the selected data group.

Figure 1 a very accurate time series plot is shown with horizontal scaling according to the number of days in a month, as well as vertical scaling according to data values. Climate stations are clustered according to missing values and sorted alphabetically. Climate stations for which there are no observations recorded for entire month are sorted alphabetically at bottom of graph. With source code provided by Dr. Leland Wilkinson; we implemented an efficient algorithm by him [10] for generating Nice Numbers for horizontal scale.

Figure 2 shows clustering and sorting of West Nile Virus Data according to missing values. States having data values for both reference and comparison year are sorted by reference year; states with reference year missing are sorted by comparison year; states with comparison year missing are sorted by reference year and states with missing data for both years are sorted alphabetically.

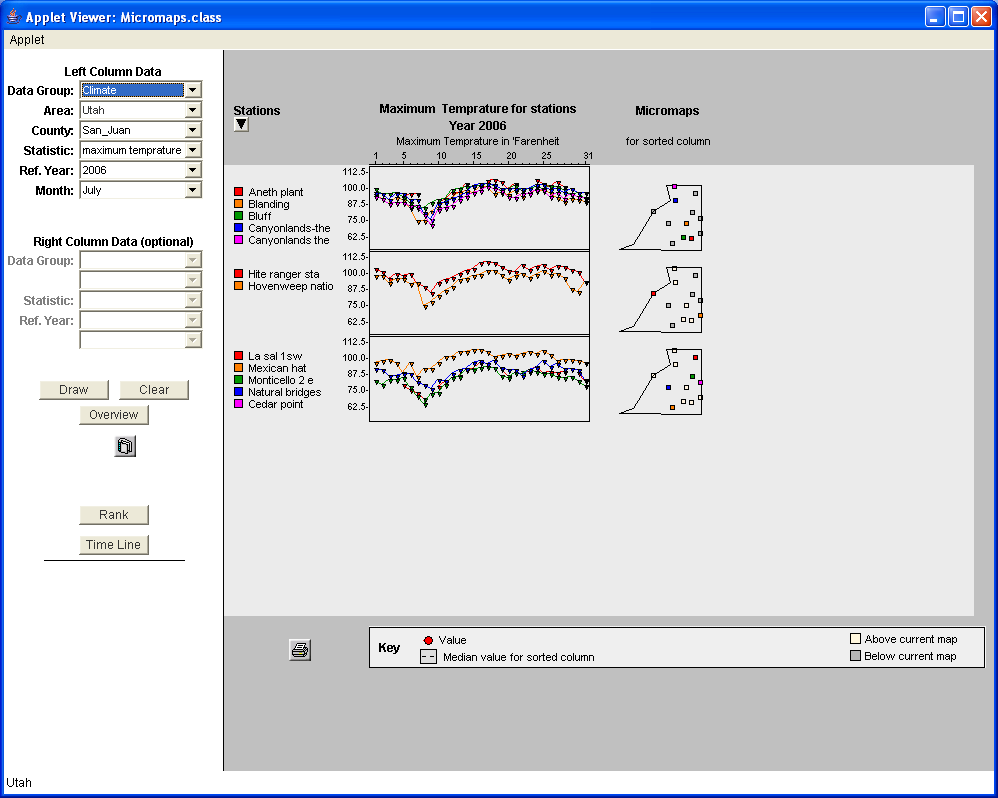


Figure A Time Series Plot of Climate Data

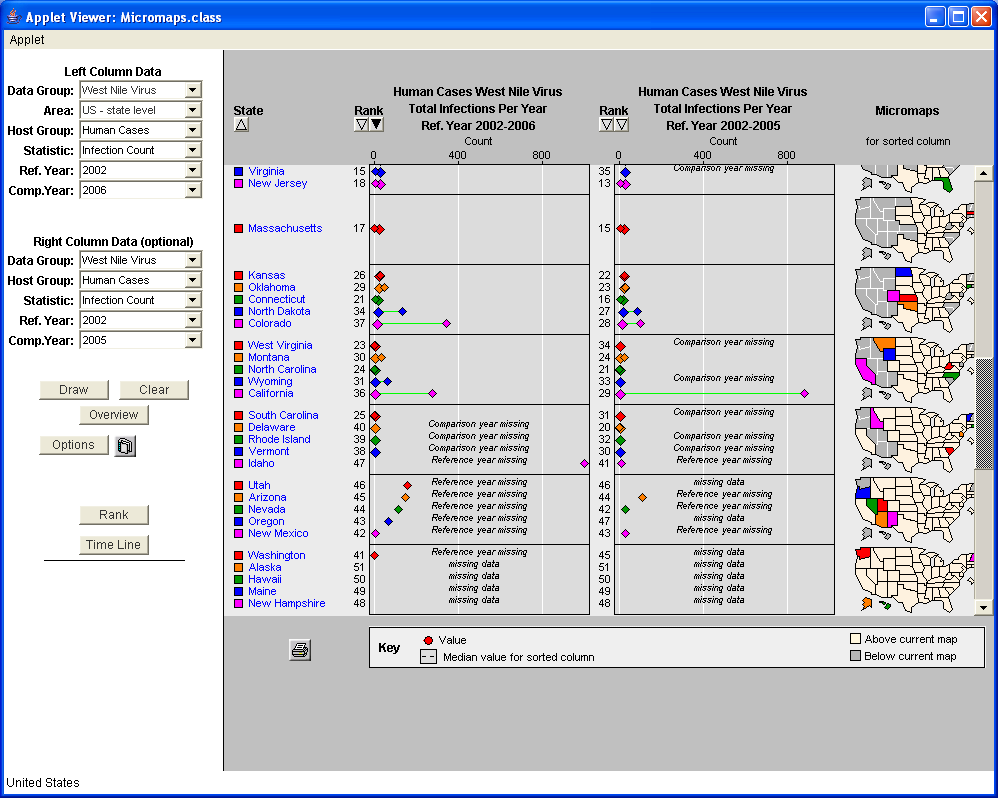


Figure Sorting and Clustering of West Nile Virus Infection Data according to missing value

## Software Engineering Practices

One of the important features of this project is optimal software engineering practices. This project followed a hybrid software development model which was a combination of water fall and iterative methodology. First phase was enhancement of West Nile Virus application. After a requirement analysis document [APPENDIX A] was finalized for this phase we started an iterative coding phase. Application was enhanced in steps one feature at a time and changes were made to original document as new ideas emerged and old ideas were discarded. Development of each new feature marked the end of single iteration of our software development process. After completion of each feature our requirement analysis document was reviewed and necessary changes were made as required. At the end of iteration, source code was committed to repository creating new version of application.

Second phase involved adding capability to show time series plot of climate data. An approach similar to first phase was followed and all requirements were first finalized [APPENDIX B].This phase was also carried out in steps implementing one requirement at a time. Nice Number algorithm and sorting functionality were added in last iteration of the project. Unit testing was also introduced for self documenting and fast debugging of application. A beta version of application is now live and is available has been made available for user acceptance testing.

Version management tool Subversion was integrated with the Net Beans integrated development environment before the start of coding phase. This helped in reverting back any changes to a current version or even switching to an older version of source code.

## Climate Vs West Nile Virus Data

We now have capability to show both Climate and West Nile Virus data using our Micromap application. These two datasets differ a great deal because West Nile Virus data is available at the US-state level and can be drilled down to county level; on the other hand, climate data is only available at individual climate station in a particular county. The graph for climate data is a time series plot of data for a specific month while for West Nile Virus it is a dot plot that contains a single or at most two points (for reference and comparison year ) connected by a line. For West Nile Virus data we have a logarithmic scale and a user can switch between normal and logarithmic scale. This scale helps in analyzing small data values, especially in the case of infection rate.

A logarithmic scale is not required for climate data. The time series plot of climate data has both vertical and horizontal scales. The horizontal scale represents time by the number of days in a month. The vertical scale is based on a dataset minimum and maximum. The vertical scale is generated using Leland Wilkinson’s algorithm for generating Nice Numbers. Given a data minimum and maximum value, the algorithm returns a new minimum and new maximum, as well as step size and the number of ticks for a scale.

## Technical Enhancements

The old Java Abstract Window Toolkit (AWT) based implementation was converted to new and lighter Java SWING based implementation. Main Applet class was converted to a Japplet (SWING based implementation of Applet class). Many other old AWT based components where converted to their newer SWING based counterparts. The result of this conversion creates more high quality graphics. An example of this can be seen in Figure 3; the figure shows compares old and new tool tips. The older tool tip has an opaque background while the new one is almost transparent. The next step would be to convert the Java Applet to a Java Web Start application

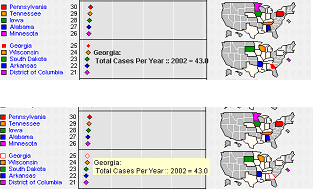


Figure New transparent SWING based tool tip compared to old opaque AWT based tool tip

## New Features

New features that were absent in the older version include sorting by both reference year and comparison year for West Nile Virus infection data. A new basic alphabetic sort is now added in time series plot of climate data. Data in that case is sorted alphabetically with climate station with all data missing sorted at bottom. I designed a new choice menu that changes itself according to the data group selected. The highlight of this project is with nice graph labels on both the horizontal and vertical scale of an accurate time series plot.

## Report Contents

This report will discuss the following topics: Micromaps design decisions in Chapter 2 , which describes thought process behind certain decisions; Technical Enhancements Chapter 3 which describes conversion of old AWT (Abstract Window Toolkit) to new SWINGS based implementation; Architectural Changes Chapter 4 talks about design changes to access firewall protected databases and include complex sorting functionality; New Features Chapter 5 elaborates some of the new features and Future Work Chapter 6 gives an overall summary and lessons learned from this project. Appendix A, B contains requirement analysis documents while Appendix C, D and E contain climate database schema, source code of important new functions and Subversion setup guide respectively.

# MICROMAPS DESIGN DECISIONS

This chapter covers certain design decisions and why they were taken. One decision was to decouple our database access so that we can query a firewall protected database. As an Applet runs on client machine a direct JDBC call to a firewall protected database is blocked. This problem was solved by using LMSERVER as mediator between Applet and database details of which are given in section 2.1. To introduce sorting capability in climate application a new class OneClimateStation.java was designed. A list of objects of this class represent climate stations with different data values and this can be sorted based on any attribute of OneClimateStation class.

Vertical Scale was generated using Nice Number algorithm based on dataset minimum, maximum and horizontal scale was generated based on number of days. The drop down menu was made more intuitive and options change dynamically according to data group (West Nile Virus or Climate).

## Why LMSERVER.java?

One of the factors that had a significant effect on the design was firewall protected databases. As an Applet runs on a client machine it cannot make a direct JDBC (JDBC is application programming interface for Java programming language that defines how clients may access databases) connection to a protected database. This problem is also known as a Java sandbox security problem. To counter this we had to decouple our architecture. We created a LMSERVER\_CLIMATE that ran on our tomcat server. The Tomcat server machine (Apache Tomcat is a pure Java based http web server) was given access rights to the database. Our Applet is connected to this program using sockets (an end point of communication flow across an Internet Protocol based computer network) and sends SQL (Structured query language is a database computer language for managing and retrieving data in a database) queries to it. This program in turn makes a JDBC connection to database and returns results of query back to Applet. Figure 4 explains the number of queries fired for a particular case and number of results returned.

## Sorting and Missing Data in Climate Application

Another design challenge was to get a sorting functionality into the climate application. It required creating new classes such as OneClimateStation.java (a class that holds the attributes of a Climate Station) and OneCounty.java (a class that holds the attributes of a single county in a state). In the climate dataset we consider a case as missing data if all the values for thirty day are missing. A variable was added to OneClimateStation.java to indicate missing data so that all such climate stations could be grouped and sorted alphabetically at bottom of graph.

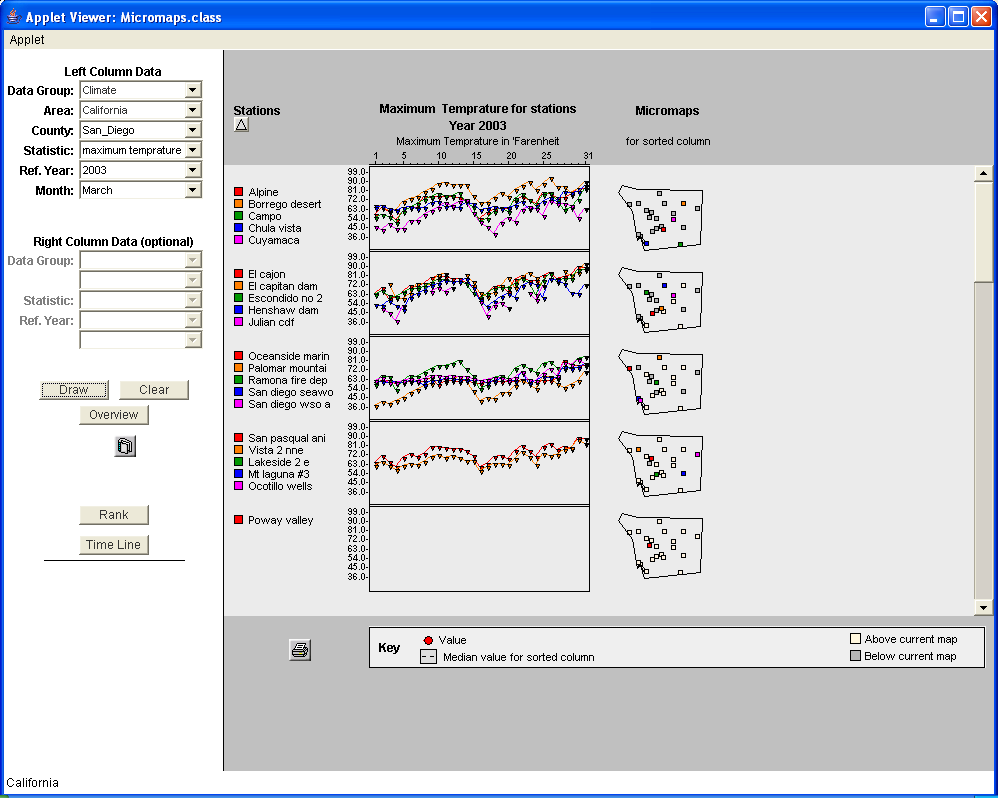


Figure Applet fires twenty one queries (one for each station) and gets thirty one (one for each day in month) results per query via LMSERVER\_CLIMATE for the above plot. Climate Stations with missing values are sorted at the bottom of graph.

## Design of Vertical Scale in Climate Data for Plotting Time Series Plot

The vertical scale of the time series plot presents nice graph labels, according to the minimum and maximum values of a dataset. These nice scales are generated using Leland Wilkinson’s algorithm for generating nice numbers on graph labels. For example if the minimum value for a dataset is 34 and the maximum value is 74, then our algorithm generates 30,45,60,75 as nice numbers for horizontal scale.

## Design of Horizontal Scale in Climate Data

The horizontal scale of the time series plot presents time as days in a month. It changes according to the number of days in a month. For example, if it is a leap year, the time scale will automatically show 29 days for February, rather than 28 days. It similarly adjusts to 30 or 31 days according to the month of a year.

## Scaling Data Values for Accurate Plot

The most complex part of the project is scaling the climate data to generate accurate time series plot. This involves both horizontal and vertical scaling. Horizontal scaling is based on the number of days in a month, while vertical scaling is based on the new nice data minimum and data maximum generated by Leland Wilkinson’s algorithm.

## West Nile Virus Logarithmic Scale

One of the important developments in the West Nile Virus application is an accurate logarithmic scale. The main issue with this scale was data values of zero. Insofar as log 0 is not defined, it was hard to show it on log scale. We instead decided to show missing values in case of zero data values. All the detailed discussion was captured in the requirement analysis document [APPENDIX A].

## Adaptive Drop Down Options

Another challenge was to switch between Climate and West Nile Virus data set smoothly. A new adaptive dropdown menu was designed which changed according to the data group selected. Figure 5, Figure 6 and Figure 7 capture this adaptive menu.

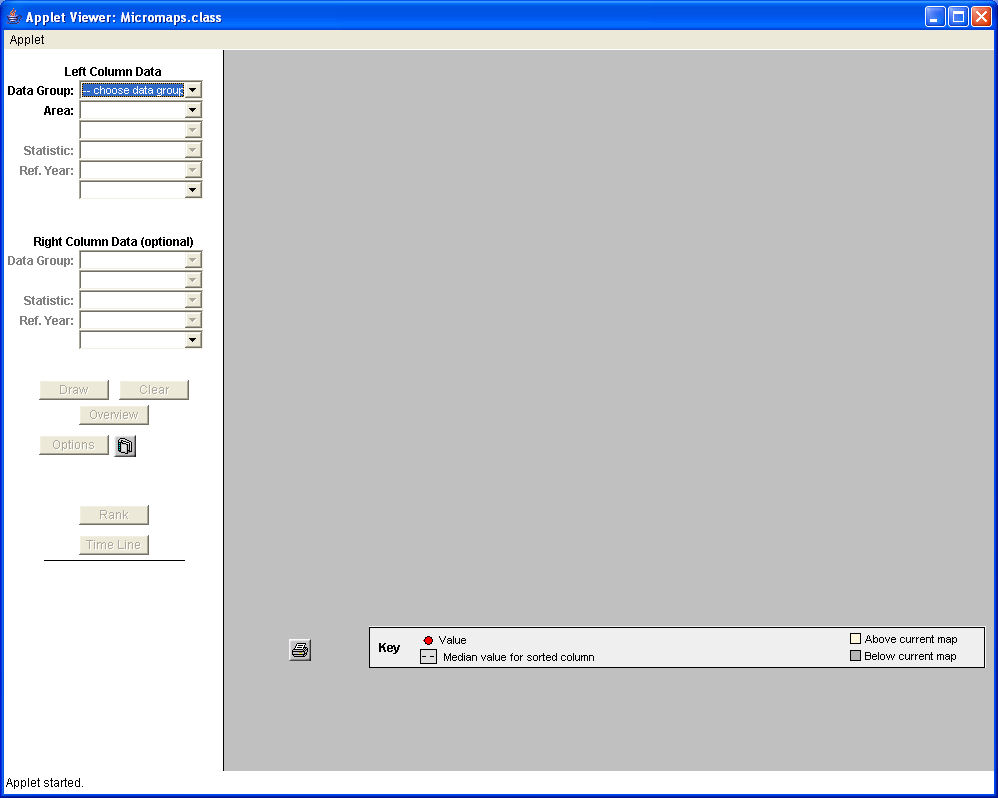


Figure The initial view of the drop down menu for selecting options.

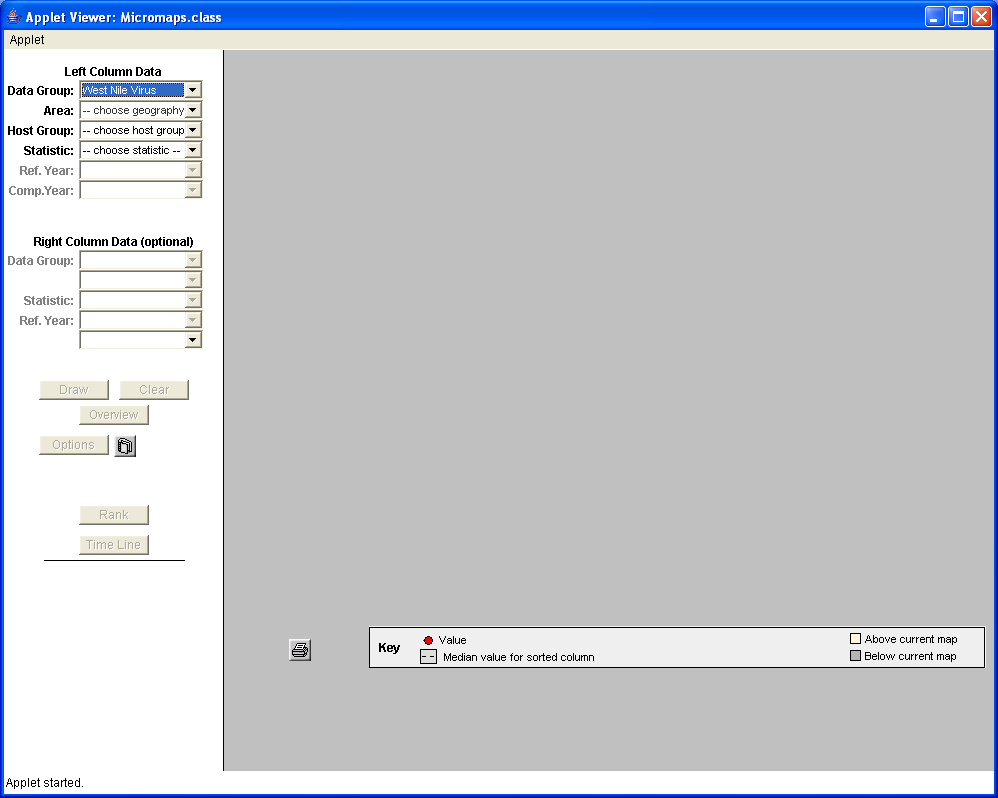


Figure The drop down menu when West Nile Virus is chosen as a data group.

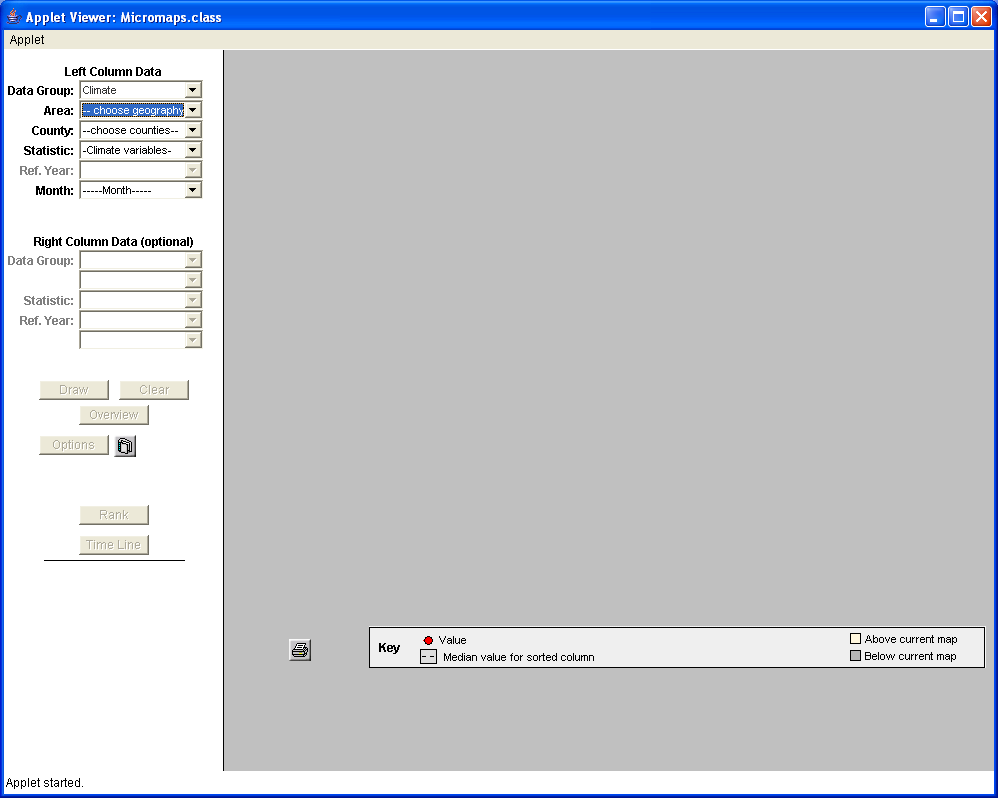


Figure The drop down menu when Climate is chosen as the data group.

# TECHNICAL ENHANCEMENT

###### 

This chapter covers technical enhancements of the Micromaps application. The Micromaps application was initially designed and developed in 2003. In internet years, this is a long time and most of the API’s used by it were deprecated. We carried out conversion from old AWT based implementation to new SWING based implementation. A Java Applet itself has several drawbacks, like heavy components and initial load time. A Java Applet also has some browser dependency. A better alternative is Java Web Start. We converted our Applet to web start application but it is at present only working with Java 1.5. Some more work needs to be done before we can work seamlessly with all Java versions.

## SWING Vs AWT

When developing Java program it is important to select the appropriate Java Graphical User Interface (GUI) components. There are two groups of components called the Abstract Window Toolkit (AWT) and SWING. Both of these groups of components are part of the Java Foundation Classes (JFC).

There are, of course, both pros and cons to using either set of components from the JFC in your Java applications. AWT is fast because of native peers, has good Applet portability because most web browsers support Applet classes (except some versions of Internet Explorer) and AWT components closely reflect the look and feel of the operating system they run on. On the downside AWT components suffer from portability issues due to the use of native peers, as supported by operating system. SWING components, on the other hand, are more portable, support rich features, have vendor support and have both a pluggable look and feel. SWING classes suffer from Applet portability on web browsers and require Java plug-in and are not as efficient as their AWT counterparts since they are written in pure Java and do not use native operating system support.

The conversion to SWING in our case has shown better graphics. The most visible enhancement is transparent tool tips. Components like drop down boxes are still based on AWT, insofar as their conversion was very tedious and would have not shown an appreciable improvement.

Both SWING and AWT are good for desktop applications; however, they suffer serious performance issues in a pure web application. JavaScript is a much lighter client side option, as compared to heavy Java classes such as SWING and AWT.

## Java Web Start vs. Applet

Java Web Start is a framework developed by Sun Microsystems; Java Web Start allows users to start the application software for a Java platform directly from the Internet using a web browser. Unlike Java Applets, the Java Web Start application does not run inside the browser; in addition, the sandbox in which they run needs to have as many restrictions, although this can be configured. Java Web Start eliminates browser dependency; consequently, it does not have issues with different Java virtual machine (JVM) versions for a browser’s Java plug-in.

Java Network Launching Protocol (JNLP) is sometimes used interchangeably with the term Web Start. The JNLP protocol defined with an XML schema specifies how to launch Java Web Start applications. JNLP files include information such as the location of the jar package file and the name of the main class for the application, in addition to any other parameters for the program. A properly configured browser passes JNLP files to a Java Runtime Environment (JRE) which in turn downloads the application onto the user's machine and starts executing it.

I converted our Applet to Java Web Start using the easy migration process provided by Sun Microsystems. Some compatibility issues still exist which need to be resolved.

.

# MICROMAPS ARCHITECTURE CHANGES

###### 

This chapter details the architecture changes of the Micromaps project. This project is implemented with a three layered architecture. The three layers are the presentation layer, the domain logic layer and the data source layer. The presentation layer now has two options: Java Applet or Java Web Start. The domain logic layer consists of the Java classes (including LMSERVER.class) and the Java server. The data source layer consists of a MySQL West Nile Virus Database and a PostgreSQL climate database.

## Presentation Layer

When the user accesses the Java Applet through the web browser, the Applet is connected with the Java server through sockets. The user can query the interested data through the server. The Applet processes the data received from the server and generates Micromaps for all states of the US, along with statistics that consist of ranks based on the mortality rate or mortality count, or on the infection rate or infection count of the West Nile Virus. Another option is the Climate data group which can generate a time series plot of climate data (maximum temperature, minimum temperature, snow fall and precipitation) of a particular month for all climate stations in a particular county in United States.

## Java Web Start

When the Applet gets loaded from the web server, the Applet will allow anyone to select data for any state, for any host group (humans, animals, birds, mosquitoes), for any statistic (mortality rate, mortality count, infection rate, infection count) and for any year to generate Micromaps. The rates or counts can be compared between two years. The scale can be changed, as an option.

## Domain Logic Layer Changes

In this layer, we have the Java files necessary for generating Micromaps. The Java server listens for connections from users on a particular port. The server will always have a persistent JDBC connection with both West Nile Virus and Climate databases. When the user sends a query for data, the server forwards the query to the database. The server returns the data received from the database to the user through sockets.

## Data Source Layer Changes

In this layer, the new application has two databases. The West Nile infection database is maintained by Utah Climate Center and the Climate database maintained by the Space Dynamics Laboratory. Climate data is maintained on a PostgreSQL database, even while climate data is maintained on a MySQL database. The Climate database is huge and has climate data as far back as the late 1800s. The West Nile Virus database is relatively small and has data since 2002.

## Unified Modeling Language (UML) Class Diagrams

This section contains new UML diagrams capturing architecture changes and new classes like OneCounty.java, OneClimateStation.java and LeScale.java. These diagrams are in addition to UML diagrams in previous reports [7][8][9] related to Micromaps application development.

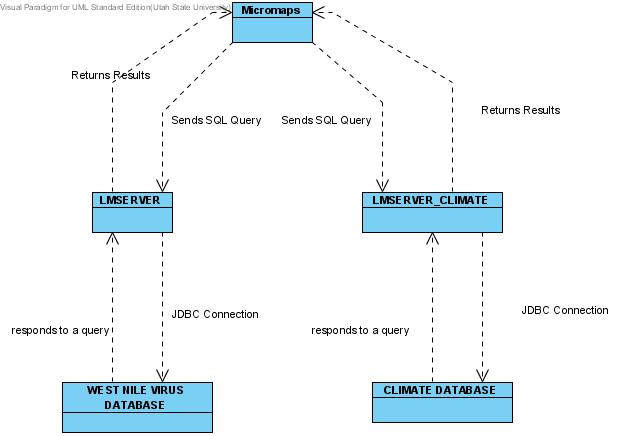


Figure High level architectural design of Micromaps application

Figure 8 shows a high level architecture of de-coupled database access architecture to overcome security issues when accessing the firewall protected database. Both the Climate and West Nile Virus databases both are behind a firewall. Tomcat web server machine at Utah Climate Center has access permissions through the firewall. Micromap application has both LMSERVER and LMSERVER\_CLIMATE that run on the web server. Instead of making direct a JDBC call to the climate database from a client machine; our Applet connects to the LMSERVER and LMSERVER\_CLIMATE on the web server, even while it passes SQL queries. These processes query the database and send back results to the front-end Applet.

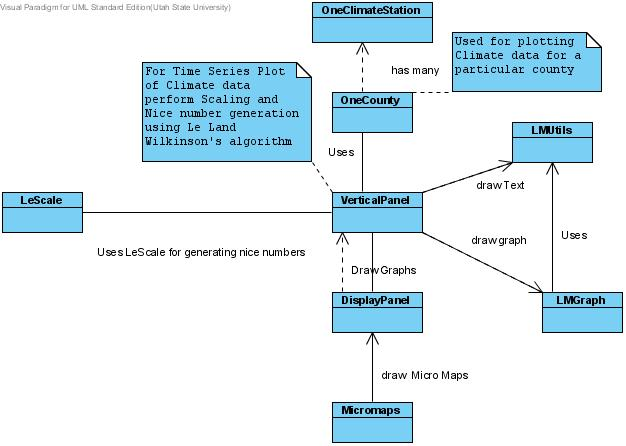


Figure UML Class diagram for client side classes for plotting graphs

Figure 9 shows detailed design of our application. Three new classes have been added by this project. LeScale.java is implementation of nice number generation algorithm. OneCounty.java has all the attributes of a county such as name, longitude, latitude, co-ordinates for generating map and a list of climate stations (OneClimateStation.java). OneClimateStation.java has attributes of a climate station which include its name, data values (minimum temperature, maximum temperature, snowfall or precipitation) and co-ordinates for generating maps. These classes help in sorting and grouping of climate stations and counties according to the desired criteria.

# NEW FEATURES

This chapter takes a look at the development of new features. We look at some of the accurate results of new development.

## Missing Data

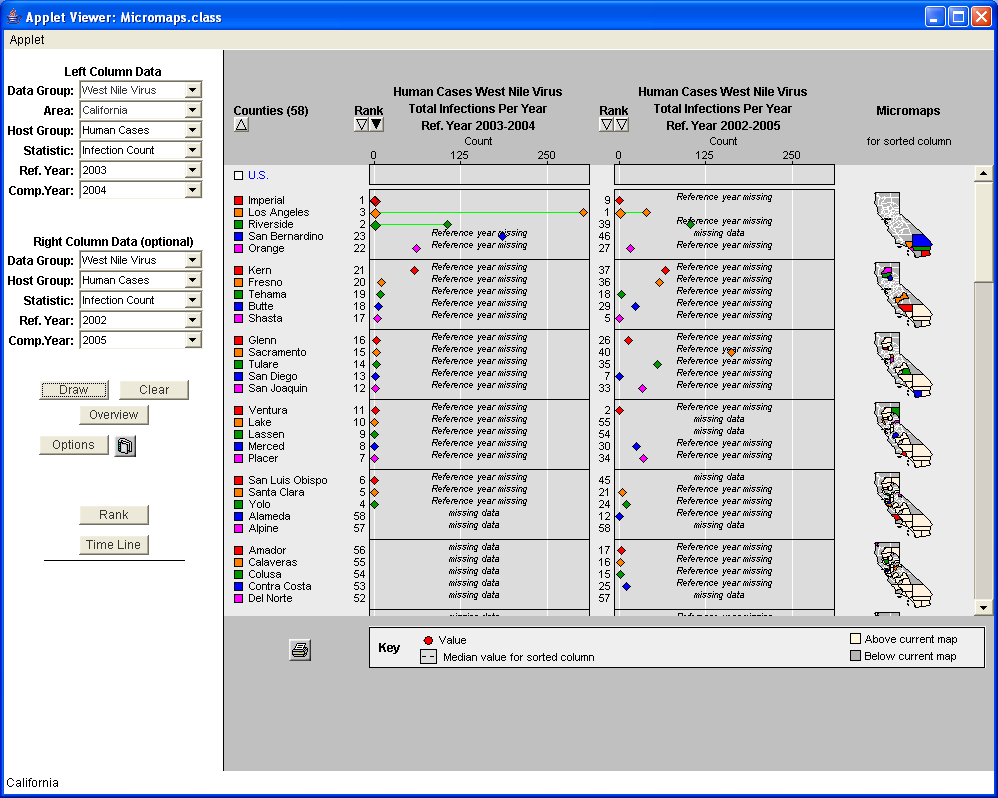


Figure Missing data sorted and clustered according to a rank button that is active (Sorted by reference year 2003).

Figure 10 shows the clustering and sorting of data according to missing value details for which are given in section 4 of APPENDIX A [A.4].

## Logarithmic Scale

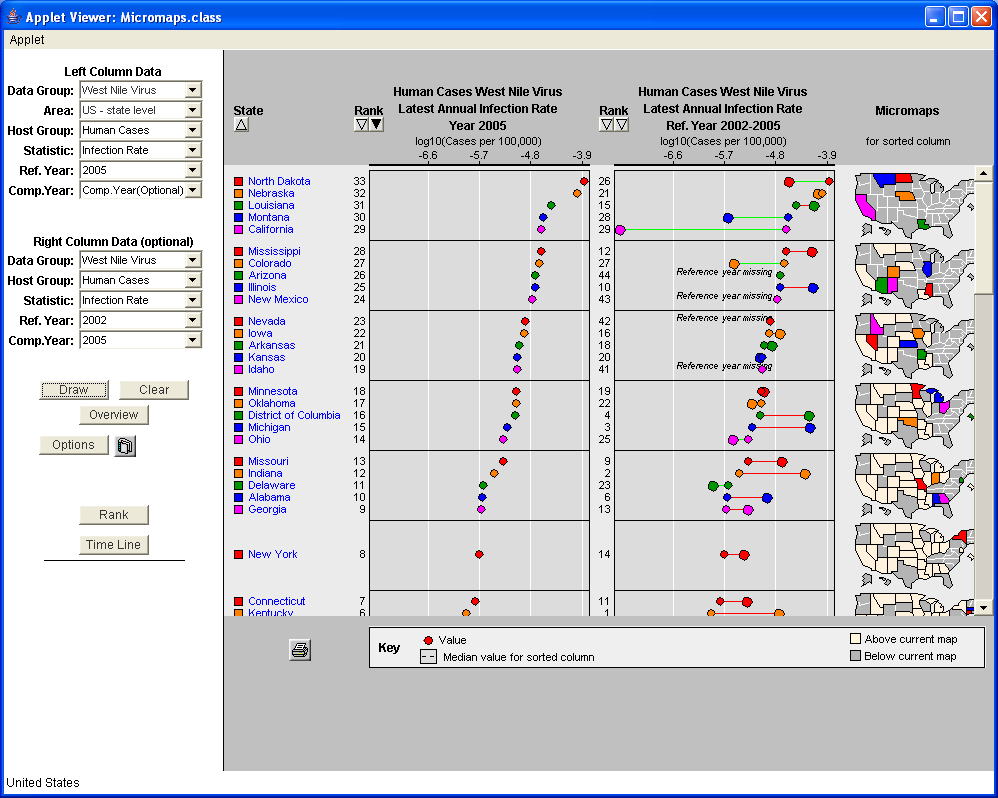


Figure Normal Scale for Infection Rate for year 2005 and 2002

Figure 11 and Figure 12 shows the logarithmic scale which treats zero values as missing data. Insofar as log (0) is not defined, it is hard to represent zero values on a logarithmic scale. So a design decision was made to treat zero values as missing values. This lead to the clustering and sorting of data according to which value is missing. Details of this clustering are given in APPENDIX A.

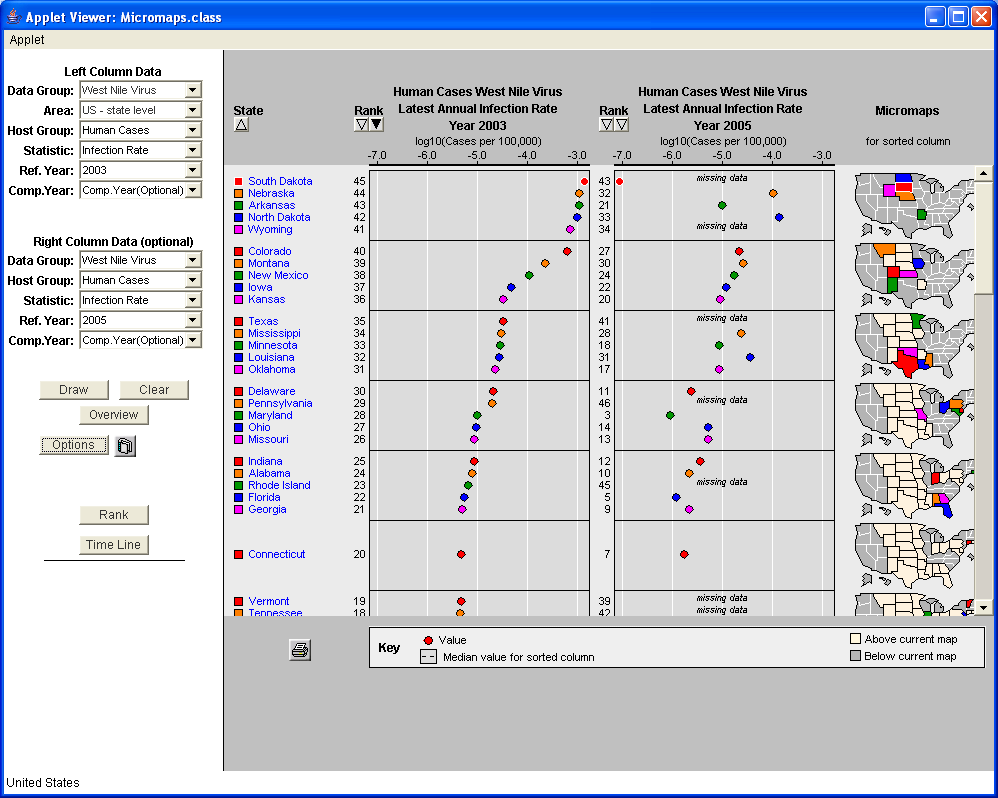


Figure An example of logarithmic scale for infection rate for year 2003 and 2005

## Time Series Plot of Climate Data

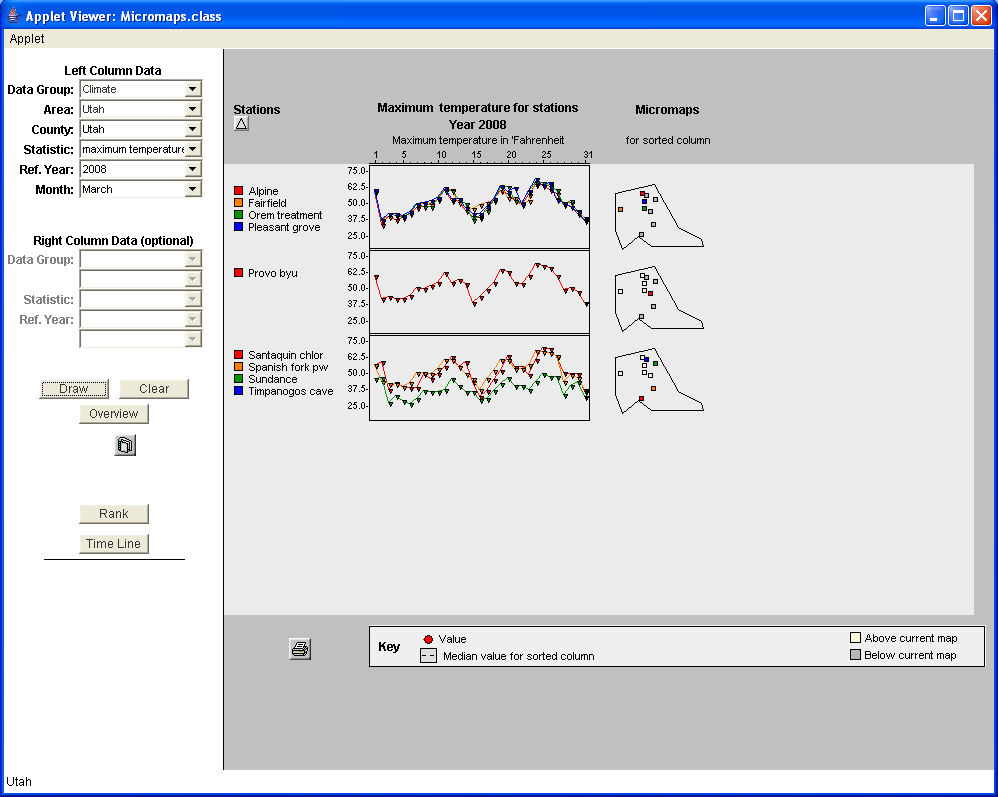


Figure Time series plot of maximum temperature for Utah County in Utah for March 2008.

Figure 13 shows a time series plot for the month of March 2008 for Utah County in Utah. Climate Stations are sorted alphabetically with stations having all values missing at the bottom of graph. The horizontal scale is time represented by the number of days in March, while the Vertical scale is generated based on minimum and maximum data for the entire dataset. Maximum temperature values are in degrees of Fahrenheit.

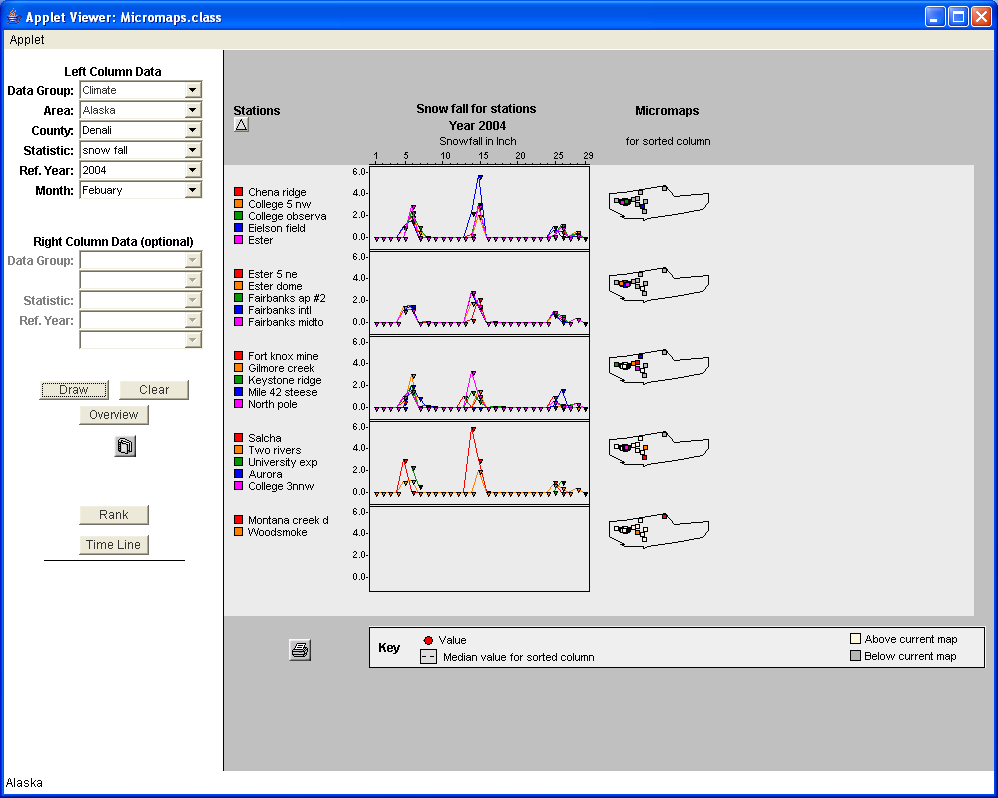


Figure Snowfall in Denali county of Alaska for February 2004 (Leap year). The Horizontal scale adjusts according to number of days in a month.

Figure 14 shows the scaling capability of the time series plot. The horizontal scale has changed to 29 days because 2004 was a leap year. It also shows stations with missing data sorted alphabetically at the bottom.

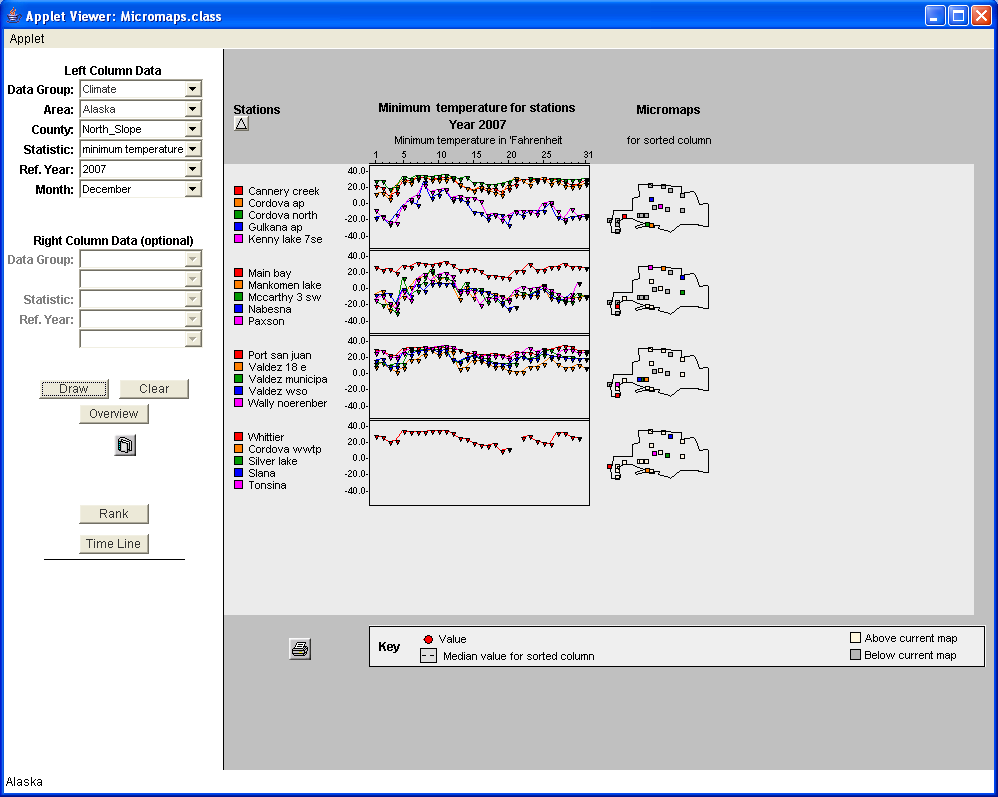


Figure Minimum temperatures in North Slope county Alaska December 2007.

Figure 15 shows minimum temperatures in Fahrenheit for December 2007 in North Slope county Alaska. This demonstrates the capability handling negative values on a vertical scale. The nice numbers start with negative value of -40 and ends at 40 on a vertical scale with a step size of 20.

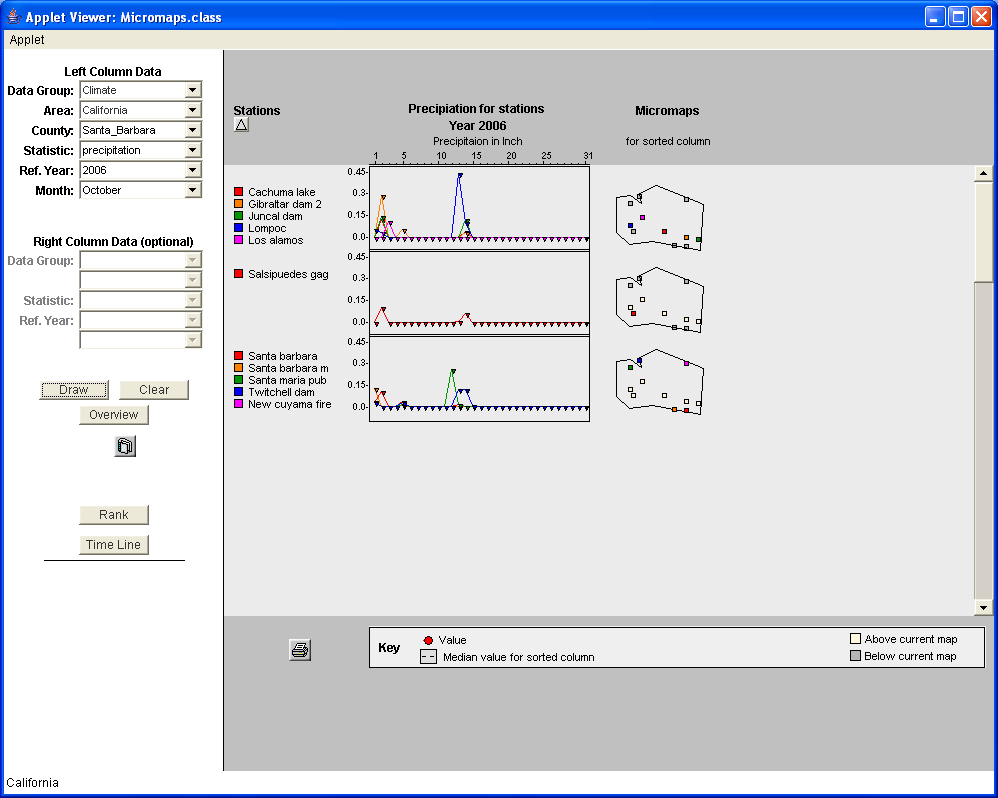


Figure Precipitation in Santa Barbara county California 2006. The Vertical scale captures even the small precipitation values.

Figure 16 represents small precipitation values in inches. The Nice numbers algorithm generates a nice scale form 0.0 to 0.45 with a step length of 0.15

## Nice Numbers for Graph Labels and Leland Wilkinson’s algorithm

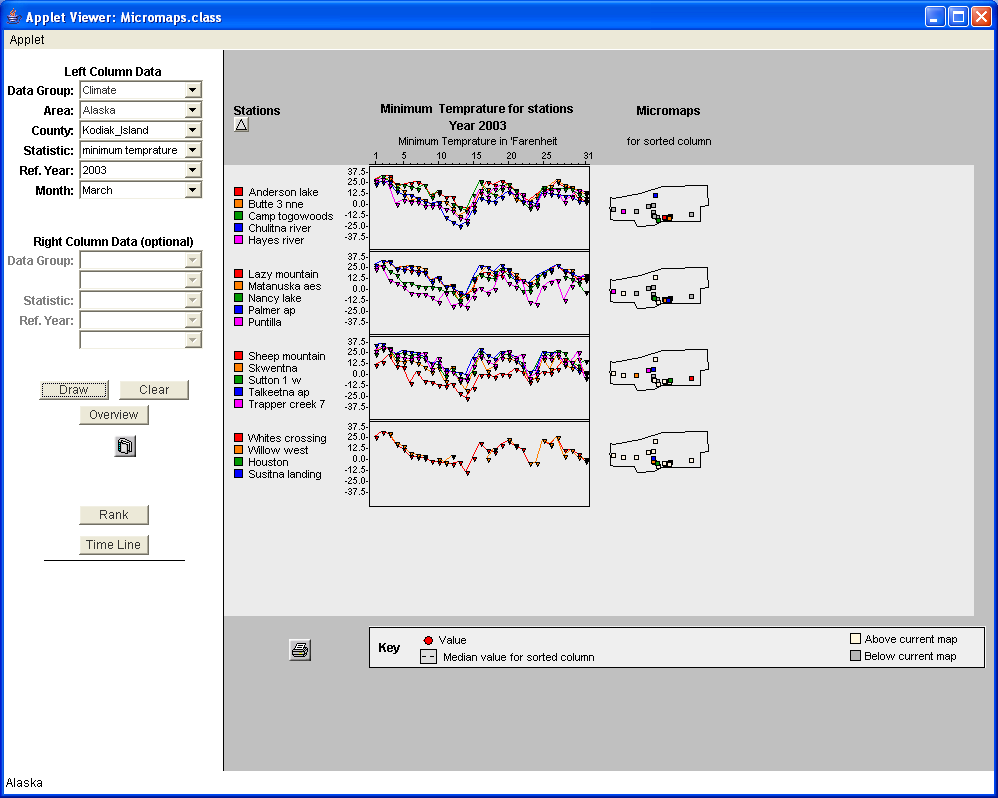


Figure Nice scales ranging from -37.5 to +37.5 with a step size of 12.5.

Figure 17 shows nice numbers generated by Leland Wilkinson’s algorithm [10] on vertical scales for the time series plot of minimum temperature. This algorithm generates a large number of scales depending upon the number of tick marks on the scale (between 3 and 8 in our application), the minimum and maximum of the dataset and a set of desired steps between two tick marks. It then picks up the best scale for which the average of three constants of granularity, coverage and roundness is less than zero.

## Switching Gears!

As Micromaps application now shows both Climate and West Nile Virus data it is important that there is a smooth transition between these two as a user changes options in drop down menu. A lot of functions have been added to facilitate this. Now user options are more adaptable; however, there is always room for improvement.

# SUMMARY AND LESSONS LEARNED

###### 

This chapter explains future enhancements that could be made to the Micromap code.

## Lessons Learned

This project is a good example of challenges faced during software development.

The initial challenge was to generate clear sets of requirements and set long term goals for the project. A thorough requirement analysis document was made capturing all our discussions. This document guided us through the project and is one of the keys to the success of this project. It is recommended that any future enhancements should first be captured in a requirement analysis document and then the document should be updated as requirements change during the course of project.

Another important lesson would be to test the deployment of the project as soon as some functionality is complete. This would help identify any deployment considerations early on and would save both time and effort. An example from this project was the security problem we faced while deploying the climate application. We carefully restructured and de-coupled our entire design in order to overcome that problem. An early deployment and testing process is highly recommended, based on this experience.

In the fast changing world of web technologies, an Applet seemed to be a technology of the past. The initial load time, Java plug-in and browser dependency make Applets less attractive. An alternative to Applet would be Java Web Start which removes browser dependency but still have initial load time. The new world of Web 2.0 involves new rich Internet technologies; consequently, RESTful architecture Applet seems to be bit outdated.

Amazing JavaScript libraries such as Prototype and Jquery are much lighter and can be used to get remarkable, dynamic content in a web page. Ongoing development needs to consider slowly and steadily enhancing the Micromaps application in this fast changing world of internet technologies. A good start would be to look at other alternatives; for instance, Ruby on Rails supports inbuilt testing, Ajax and the latest JavaScript libraries such as Prototype. The aim would be better performance, clear design and good cross platform portability.

Another important lesson is following Test Driven Development strategy which would involve writing unit test cases as we develop new functionality. Unit test cases are self-documenting and help in both fast error resolution and debugging. I have already started using the Junit testing framework built into Net Beans IDE and started using Junit testing to generate test cases for all the classes. I have to continue completing those initial test cases and keep writing new ones as we add more functionality.

## Pollution Data

The next application of Micromaps would be to show pollution data. Pollution data has lot of different variables like ppm concentration and uv-index. The idea is to show the traffic lights on Micromaps that indicate pollution. Green would mean clean air, orange would mean mild pollution and red would raise the alarm for degraded air quality. Right now we can implement both sorting and clustering with different criteria such as mean, average, above average and below average in the climate application.

## Testing

Software testing is a means of verifying and validating that a program meets business and technical requirements, as well as expected work. Different software development methodologies focus on testing at different times during the process. A traditional process would start testing after requirements analysis and coding phase while newer approaches like Extreme Programming (XP) and Agile software development would start testing early in the process. Developers use new methods of software engineering that employ testing upfront in the development process.

Tests are either grouped into testing levels according to when they are added to the software development process or grouped by level of specificity of test. Unit testing refers to testing that verifies the functionality of specific sections of code. It generally involves testing both Micromaps functions as standalone entities and whether they give desired results or not. Integration testing on the other hand tests the integration and interfaces between various software modules as defined in software architecture. System testing tests a completely integrated system to verify that it meets its requirements. Regression testing focuses on finding defects after major code changes. Generally, customers perform acceptance testing in their lab environment on their own hardware. Alpha testing is either simulated or is an actual Operational Testing performed either by a potential customers or an independent team at the developer’s site. Beta testing comes after alpha testing. Versions of the software known as beta versions are released to a limited audience outside the programming team.

As the project grows in size it becomes wiser to include unit test cases. In my case, it would be better late than never. I plan to implement some basic unit test cases using Junit testing framework. These would help in the long run with both quick error resolution and bug fixing.

# REFERENCES

1. Carr DB, Olsen AR, Pierson SM, Courbois JP, 2000. Using Linked Micromap Plots to Characterize Omernik Ecoregions. *Data Mining and Knowledge Discovery 4*: 43–67.
2. Wang X, Chen JX, Carr DB, Bell BS, Pickle LW, 2002. Geographic Statistics Visualization: Web-based Linked Micromap Plots. *Computing in Science & Engineering* *4(3)*: 90–94.
3. Symanzik J, Gebreab S, Gillies, RR, Wilson J, 2003. Visualizing the Spread of West Nile Virus. *2003 Proceedings, American Statistical Association*, Alexandria, VA. (CD)
4. Carr DB, Bell BS, Pickle L, Zhang,Y., Li Y, 2003. The State Cancer Profiles Web Site and Extensions of Linked Micromap Plots and Conditioned Choropleth Map Plots. *dg.o2003* *Proceedings, Digital Government Research Center (DGRC)*. (Web).
5. Symanzik J, Axelrad DA, Carr DB, Wang J, Wong D, Woodruff TJ, 1999. HAPs, Micromaps and GPL - Visualization of Geographically Referenced Statistical Summaries on the World Wide Web. *Annual Proceedings (ACSM-WFPS-PLSO-LSAW 1999 Conference CD,) American Congress on Surveying and Mapping*.
6. Symanzik, J., Carr, D. B. (2008): Interactive Linked Micromap Plots for the Display of Geographically Referenced Statistical Data, In: Chen, C., Härdle, W., Unwin, A. (Eds.), *Handbook of Data Visualization*, Springer, Berlin/Heidelberg, 267–294 & 2 Color
7. Chapal, G.K. (2005): Development of rich features for web-based interactive Micromaps, Utah State University, Logan.
8. Surampudi, R. (2006) : Micromaps: Development of additional key attributes, Utah State University, Logan
9. Vallabhaneni, V. (2007): Visualization of Climate Data, Utah State University, Logan.
10. Wilkinson, L. (2005).Grammar of Graphics, Second Edition. Springer, Berlin/Heidelberg, 217–218

# APPENDIX A

Requirements Specification for Micro maps

West Nile Virus Application

# System Context & Scope

## 

## Scope of the System

The micro maps application at Utah State University was adapted from micro maps application of state cancer profiles:

([http://statecancerprofiles.cancer.gov/Micromaps/](http://statecancerprofiles.cancer.gov/micromaps/))

The application was customized over time to show West Nile Virus infection rates and count at both state and county level. Many additional options have been added. Our primary focus in this document would be the normal and log scales on which we show the statistical summary of data. This can be considered as the core of application.

## Problems with Current System

The current version of application has many flaws due to which we cannot represent data correctly on the provided scales. We list some of the main problems that exist in our system.

1. If we toggle between normal scale and log scale the current implementation changes our data due to log-antilog conversion and mapping of log0(as log0 is not defined) to a certain constant value
2. The other problem is a count value of 1 when mapped to log scale will show no data available on log scale as log 1 is 0. Thus a 0 value on log scale should not show “no data available”. We should keep track of our values between scales for which we show “no data available”.
3. If we keep log scale as our scale and change the reference or comparison year, our scale goes out of order
4. All the “no data available” should come last always regardless of the sort order.
5. We lack sorting capabilities so that we can sort by both reference year and difference between reference and comparison year
6. We don’t distinguish cases when both the values of reference year and comparison year is zero or either of the one is zero. Our mapping of zero to a constant value gives us erroneous output. In our current implementation we assume that a count or rate of zero represents missing data.

## Use Case List

Following are the desired functionalities that should be added to current system.

1. Enhance our sorting capabilities.
2. Find a way to deal with missing values on log scale.
3. Make our data representation more intuitive.
4. Remove all the errors that have been identified in our application

# FUNCTIONAL REQUIREMENTS

## 

## Functional Requirements

Following are the desired functionalities that should be added to current system.

1) Sorting all values according to reference year

1. Smooth toggle between log and normal scale
2. The mapping of a zero value to some constant value should be removed and all zero values should be treated as missing data. We should show appropriate message on the scale so that all the cases of zero values are addressed.
3. If log scale option is kept checked on the options menu and reference or comparison year is changed then the values for those years should be fetched from the database and only be displayed after converting them into log values. A functionality that is missing in the current version.
4. Whenever new data is queried from the database all related variables should be immediately calculated, e. g, log, difference between reference and comparison year, etc. Also whenever we change any parameter (reference year, comparison year, data group, etc) on the main display, our application should sense this change and update the graphs. This implies that we may not need to hit the draw button every time or draw button may be removed altogether from the application.
5. We must discard the functionality of converting back to normal scale from log scale by taking antilog as it prone to errors.

# Look, Feel and Use Requirements

# 

## User Interface Requirements

1. We need to change our default sort order to be by reference year.
2. A mechanism (button, radio button) should be added in the display panel that may sort our data by both reference year and difference.
3. We may have to be able to toggle between these two displays for which we may add a radio button.

## Ease of Use Requirements

An integrated help should be attached to our application which explains various options available to a user and also explains to him the interpretation of various results.

# Proposed Solutions

## Solutions Proposed Based on Our Discussions

Following are a list of solutions that have been proposed for our application.

:

1. We would group our data according to following four categories

Grouping:

* 1. If both reference year and comparison year data is available ( both are non zero value)
  2. If both reference year and comparison year data is not available(both values are zero)
  3. Reference year value is available but comparison year value is not available(reference year value is non zero but comparison year value is zero)
  4. Reference year value is not available but comparison year value is available (reference year value is zero but comparison year value is non zero)

1. We want graph to plot the values according to above specified group
2. Sorting should also be based on type of groups and should be local to a group
3. Following sort orders have to be taken care of

4.1) By name

    A) Alphabetically

    B) Inverse alphabetically

1. **By Rank (Reference Year)**

    A) Regular Scale

         1) Highest to lowest according to reference year

         2) Lowest to highest according to reference year

3) Missing data always comes at bottom in alphabetical order

   B) Log Scale

       1) Highest to lowest according to reference year

       2) Lowest to highest according to reference year

3) Missing data always comes at bottom in alphabetical order

1. **By Difference (Reference year – Comparison year)**

    A) Regular Scale

         1) Highest to lowest according to difference

         2) Lowest to highest according to difference

3) Missing data always comes at bottom in alphabetical order

   B) Log Scale

       1) Highest to lowest according to difference

  2) Lowest to highest according to difference

3) Missing data always comes at bottom in alphabetical order

1. **Special Cases in Sorting**:

1. We follow the official numbering scheme for 50 states as given in link <http://www.itl.nist.gov/fipspubs/fip5-2.htm> .
2. Also “Washington, DC” means “District of Columbia” and therefore is sorted in between “Delaware” and “Florida”
3. Following cases can arise when sorting with reference year:
   1. If reference year value is same and comparison year value is available we decide the order of such values by comparison year value.
   2. If reference year value is same and comparison year value are also same then sort alphabetically.
   3. If reference year value is same and comparison year value is missing we decide the order by the alphabetic order of states.
   4. If both values are missing, all such values occur last and are sorted alphabetically according to states.
4. Following cases can arise when sorting with difference(reference year – comparison year)
5. If difference is same for two years we order them by reference year
6. If reference year value of data is available but comparison year is unavailable, they are ordered by reference year, though missing data is displayed on graph
7. If reference year value of data is same and difference of values is also same then sort alphabetically.
8. If reference year value of data is same and comparison year unavailable, sort alphabetically.
9. If comparison year value is same and reference unavailable, sort alphabetically.
10. If comparison year is available but reference year is unavailable ,we plot by comparison year, though missing data is displayed on graph
11. If both values are missing, all such values occur last and are sorted alphabetically according to states.

1. If log scale option is kept and user changes reference or comparison year and hits draw, new values from the database should be fetched and converted to log before they are plotted. Whenever new data is queried from the database all related variables are immediately calculated, e. g, log, difference between reference and comparison year, etc.
2. Also whenever we change any parameter (reference year, comparison year, data group, etc) on the main display, our application should sense this change and update the graphs. This implies that we may not need to hit the draw button every time or draw button may be removed altogether from the application
3. The version management tool Subversion is a freely available and most widely used version management**. http://subversion.tigris.org/.**

We may first implement version management and then go for further development.

We need to include version management into our software so that it is easy to keep track of development at different phases.

## 

# Future Requirements

We may eliminate draw button altogether and application should automatically plot graph when all input parameters are provided. This will make it sensitive to change in any parameter and the graph will re-plot after sensing any change in input parameter.

# APPENDIX B

Requirements Specification for Micro maps as applied to Climate data

# System Context & Scope

## Scope of the System

Micro maps have been used at Utah State University to show West Nile Virus infection.

<http://webcat.gis.usu.edu:8080/index.html>.

This concept can be further extended to show climate data. Climate data may include following variables that can be plotted in a time series graph.

1. Maximum temperature
2. Minimum temperature
3. Precipitation
4. Pollution
5. UV-light index

This can give a good statistical overview of climate conditions in an area. The application can be further enhanced to plot climate data on one panel and West Nile Virus infection on other so that interesting relations can be drawn between climate conditions and Wet Nile infection in a particular region.

## Problems with Current System

In the current scenario we have two different applications, one that connects only to West Nile Virus database and the other that has interface to both West Nile and Climate database.

The application that connects only to West Nile Virus data was the one that is now enhanced to increase the sorting capabilities, correct the log scale and remove all the critical bugs.

The other application that connects to both the databases is full of bugs and does not have a clear user interface to climate data. Even the plots for both the databases are flawed. Hence this application will only be used as a reference from time to time.

In the current development we will be enhancing the application that connects only to West Nile Virus database to connect to climate data as well with a much clearer user interface.

## Use case List

The current use case list will have following functionalities that need to be developed

1. Clear interface to climate database
2. Correcting the time series plot for climate data
3. Adding more climate variables the can be plotted (uv-light index, pollution).
4. Having capability to plot both Climate and West Nile Virus data side by side in two different panels so that any relation between them can be derived.

# Functional Requirements

We want following functionalities in our database

1. Data Group should be the first selection in when the Applet gets loaded so that application connects to the desired database according to Data Group.
2. The other major requirement would be correcting the scales for climate data time series plot.
3. As climate data is at station level for each county we will have to add additional drop down boxes for county whose climate station data we want to plot.
4. Other requirement would be to plot appropriate map .One important design issue is the maps that are plotted for both the databases are different.

West Nile Virus is plotted at state level which can be further drilled down to county level. When state level data is drawn we map of USA is drawn with corresponding state highlighted and for county level data map of state is drawn with corresponding county highlighted

In climate data, we draw map of county with corresponding climate stations as highlighted. As map is shared for both our panels we will have to decide which map will be plotted when we are plotting climate data on one panel and West Nile Virus data on another panel.

1. The other major design issue would be that climate data is only at climate station level for each county, while West Nile Virus data is at county level and state level. This will only come into picture when we want to plot different data on different panels i.e. West Nile Virus on one and Climate on another.

# Look, Feel and UI Requirements

## User Interface Requirements

User interface of current application will have to be redesigned so that access to multiple data sources is easy while keeping in mind further enhancements.

This may involve disabling and enabling various components according to data selected and adding additional components.

The priority at this stage would be to correct the scale for time series plot for climate data once we have easy interface to access it.

# Proposed Solutions

## 

## Solutions Proposed Based on Our Discussions

At this stage with initial discussion the following solutions have been proposed

1. Making the interface to climate data much more clearer and intuitive
2. Correcting the scale of time series plot for climate variables.

Once these two requirements are met we can move to address the problem when we want to plot different data on different panels.

# Future Requirements

Future requirements may involve showing aggregates of station level data at county level and aggregates of county level data at station level for climate data.

This may even resolve our problem of plotting climate data and West Nile Virus data side by side.

# APPENDIX C

**Database schema for Climate Database**

Table "coop.records"

|  |  |  |
| --- | --- | --- |
| Column | Type | Modifiers |
| index | integer | not null default nextval('coop.records\_index\_seq'::regclass) |
| station | integer | not null |
| day | date |  |
| temp | integer | not null |
| pcpn | integer | not null |
| et0 | real |  |
| source\_id | integer | not null |

Indexes:

"records\_index\_pk" PRIMARY KEY, btree ("index")

"records\_uniq" UNIQUE, btree (station, "day")

"index\_records" btree (station, "day") CLUSTER

"pcpn\_idx" btree (pcpn)

"records\_date\_year\_idx" btree (cli\_date\_year("day"))

"records\_day\_idx" btree ("day")

"records\_scd\_year\_idx" btree (station, cli\_date\_year("day"))

"records\_source\_id\_idx" btree (source\_id)

"temp\_idx" btree ("temp")

Foreign-key constraints:

"records\_pcpn\_fkey" FOREIGN KEY (pcpn) REFERENCES coop.pcpn\_sets("index")

"records\_source\_id\_fk" FOREIGN KEY (source\_id) REFERENCES coop.data\_source(source\_id)

"records\_station\_fkey" FOREIGN KEY (station) REFERENCES coop.stations(id)

"records\_temp\_fkey" FOREIGN KEY ("temp") REFERENCES coop.temp\_sets("index")

Triggers:

et0update BEFORE INSERT OR UPDATE ON coop.records FOR EACH ROW EXECUTE PROCEDURE et0update()

Index "coop.records\_date\_year\_idx"

|  |  |
| --- | --- |
| Column | Type |
| pg\_expression\_1 | date |

btree, for table "coop.records"

Index "coop.records\_day\_idx"

|  |  |
| --- | --- |
| Column | Type |
| day | date |

btree, for table "coop.records"

Index "coop.records\_index\_pk"

|  |  |
| --- | --- |
| Column | Type |
| index | integer |

primary key, btree, for table "coop.records"

Sequence "coop.records\_index\_seq"

|  |  |
| --- | --- |
| Column | Type |
| sequence\_name | name |
| last\_value | bigint |
| increment\_by | bigint |
| max\_value | bigint |
| min\_value | bigint |
| cache\_value | bigint |
| log\_cnt | bigint |
| is\_cycled | boolean |
| is\_called | boolean |

Index "coop.records\_scd\_year\_idx"

|  |  |
| --- | --- |
| Column | Type |
| station | integer |
| pg\_expression\_2 | date |

btree, for table "coop.records"

Index "coop.records\_source\_id\_idx"

|  |  |
| --- | --- |
| Column | Type |
| source\_id | integer |

btree, for table "coop.records"

Table "coop.stations"

|  |  |  |
| --- | --- | --- |
| Column | Type | Modifiers |
| id | integer | not null |
| name | character varying(30) |  |
| latitude | real |  |
| longitude | real |  |
| elevation | integer |  |
| active | boolean | default true |
| state\_approved | boolean | default false |
| first\_record | date |  |
| last\_record | date |  |
| state | character varying(2) |  |
| time\_offset | double precision |  |
| booldst | boolean |  |

Indexes:

"stations\_pkey" PRIMARY KEY, btree (id)

"stations\_state\_idx" btree (state)

Index "coop.stations\_pkey"

Table "coop.temp\_sets"

|  |  |  |
| --- | --- | --- |
| Column | Type | Modifiers |
| index | integer | not null default nextval(('coop.temp\_sets\_index\_seq'::text)::regclass) |
| tmax | real | default 9999 |
| tmin | real | default 9999 |
| tmid | real | default 9999 |
| tobs | real | default 9999 |
| flags | integer | not null default 1 |

Indexes:

"temp\_index\_pk" PRIMARY KEY, btree ("index")

"temp\_sets\_uniq" UNIQUE, btree (tmax, tmin, tobs, flags)

"tmid\_idx" btree (tmid)

"tmin\_idx" btree (tmin)

"tobs\_idx" btree (tobs)

Foreign-key constraints:

"temp\_sets\_flags\_fkey" FOREIGN KEY (flags) REFERENCES flag\_sets("index")

Sequence "coop.temp\_sets\_index\_seq"

|  |  |
| --- | --- |
| Column | Type |
| sequence\_name | name |
| last\_value | bigint |
| increment\_by | bigint |
| max\_value | bigint |
| min\_value | bigint |
| cache\_value | bigint |
| log\_cnt | bigint |
| is\_cycled | boolean |
| is\_called | boolean |

Index "coop.temp\_sets\_uniq"

|  |  |
| --- | --- |
| Column | Type |
| tmax | real |
| tmin | real |
| tobs | real |
| flags | integer |

unique, btree, for table "coop.temp\_sets"

Index "coop.tmid\_idx"

|  |  |
| --- | --- |
| Column | Type |
| tmid | real |

btree, for table "coop.temp\_sets"

Index "coop.tmin\_idx"

|  |  |
| --- | --- |
| Column | Type |
| tmin | real |

btree, for table "coop.temp\_sets"

Index "coop.tobs\_idx"

|  |  |
| --- | --- |
| Column | Type |
| tobs | real |

btree, for table "coop.temp\_sets"

Table "coop.pcpn\_sets"

|  |  |  |
| --- | --- | --- |
| Column | Type | Modifiers |
| index | integer | not null default nextval(('coop.pcpn\_sets\_index\_seq'::text)::regclass) |
| pcpn | real | Default9999 |
| snow | real | Default9999 |
| snowdepth | real | Default9999 |
| flags | integer | not null default 1 |

Indexes:

"pcpn\_index\_pk" PRIMARY KEY, btree ("index")

"pcpn\_sets\_uniq" UNIQUE, btree (pcpn, snow, snowdepth, flags)

"snow\_idx" btree (snow)

"snowdepth\_idx" btree (snowdepth)

Foreign-key constraints:

"pcpn\_sets\_flags\_fkey" FOREIGN KEY (flags) REFERENCES flag\_sets("index")

Sequence "coop.pcpn\_sets\_index\_seq"

|  |  |
| --- | --- |
| Column | Type |
| sequence\_name | name |
| last\_value | bigint |
| increment\_by | bigint |
| max\_value | bigint |
| min\_value | bigint |
| cache\_value | bigint |
| log\_cnt | bigint |
| is\_cycled | boolean |
| is\_called | boolean |

Index "coop.pcpn\_sets\_uniq"

|  |  |
| --- | --- |
| **Column** | **Type** |
| pcpn | real |
| snow | real |
| snowdepth | real |
| flags | integer |

unique, btree, for table "coop.pcpn

# APPENDIX D

**Important Functions and Their Source Code**

**a) Vertical Scaling for Time Series.**

// Get the data from database and return vertically scaled data.

public Vector<Vector<Double>> setTimeSeries(Vector <Vector<Double>> data)

{

sRank=new int[mapplet.iStationCount];

int i,j=0,k,l=0,map=0;

int no\_data = 9999;

double diff=0;

high = JDBC\_Climate.MAX;

low =JDBC\_Climate.MIN;

LeScale.computeNumericScale(low, high); //Calculate Nice Scale

high = LeScale.sMax; //Get new high from LeScale

low= LeScale.sMin; //Get new low from LeScale

double lstep = LeScale.sDelta; //Get step from LeSacle

for(i=0;i<iStationCount;i++)

{ // Function for drawing scales

if(border[i]==1)

{

if(high!=low )

diff=65.0/(high-low);

if( i!=0)

{

for(k=l;k<i;k++)

{

for(j=0;j< data.get(k).size();j++)

{ //for temperature values of a month

if(data.get(k).get(j)== no\_data)

//Missing data set to Double Max

data.get(k).set(j, Double.MAX\_VALUE);

else{

**//Main Vertical Scaling equation**

data.get(k).set(j, (double)(((7+((map-1)\*85)))+(int)((high -(data.get(k).get(j)))\*diff)) ); //The Magic Equation

}

}

}

dPanel.drawScale(low,high,lstep,85\*(map-1)); //Draw the graph Scale Labels

}

l=i;

map++;

}

}//end for i loop

if(high!=low )

{

diff=(float)65.0/(float)(high-low);

dPanel.drawScale(low,high,lstep,85\*(map-1)); //Draw the graph scale Label

}

for(int m=l;m<iStationCount;m++)

{

System.out.println("New Station");

for(j=0;j<data.get(m).size();j++)

{

if(data.get(m).get(j)== 9999)

data.get(m).set(j, Double.MAX\_VALUE);

else

{

data.get(m).set(j, (int)((7+85\*(map-1)))+(double)((high-data.get(m).get(j))\*diff) );

}

}

}

return data; //return vertically scaled data.

}

**b) Horizontal Scaling for Time Series.**

public void TplotPoint(Color fore,int top, boolean diff, Vector scaled\_values)

{

int i = 0;

if (g==null) return;

double days = scaled\_values.size();

double MAXDAYSINMONTH = 31;

double VERTICALSCALE = MAXDAYSINMONTH/days; //Factor for scaling according to number of days

int lineY = 0;

Color t1 = g.getColor();

int k = 0, count = 0, totaltemp = 0;

// double difftemp = 0, prevdifftemp = 0;

///Checking if all the data values for a climate Station are missisng

boolean missingData = true;

for(i=0;i< scaled\_values.size() ;i++)

{

prevdifftemp = difftemp;

Object temp = scaled\_values.get(i);

difftemp = (Double)temp;

if(difftemp != Double.MAX\_VALUE){

missingData = false;

break;

}

}

//////////////////////////////////////////////////////////////////////

if(missingData)

{

//Draw a string showing appropirate missing data message

int x = getPixelX(VERTICALSCALE);

g.setColor(fore);

// g.drawString(" ---missing data--- ", x, lineY);

}

else{ //Else plot the lines

for(i=0;i< scaled\_values.size() ;i++)

{

prevdifftemp = difftemp;

Object temp = scaled\_values.get(i);

difftemp = (Double)temp;

int x = getPixelX(i\*VERTICALSCALE);

if(difftemp != Double.MAX\_VALUE && prevdifftemp != Double.MAX\_VALUE){

g.setColor(fore);

if (i != 0)

{

g.drawLine(x+ (int)(7\*(i-1)\*VERTICALSCALE), lineY+(int)prevdifftemp, x+(int)(7\*(i)\*VERTICALSCALE), lineY +(int) difftemp);

LMUtils.fillMyShape(g, x+(int)(7\*(i-1)\*VERTICALSCALE), lineY+(int)prevdifftemp, iShapeSize/2, shapeType, Color.black);

LMUtils.fillMyShape(g, x+(int)(7\*(i)\*VERTICALSCALE), lineY + (int)difftemp, iShapeSize/2, shapeType, Color.black);

}

} //End of if if(difftemp != Double.MAX\_VALUE && prevdifftemp != Double.MAX\_VALUE)

}//End of for loop

}//End of else missingdata

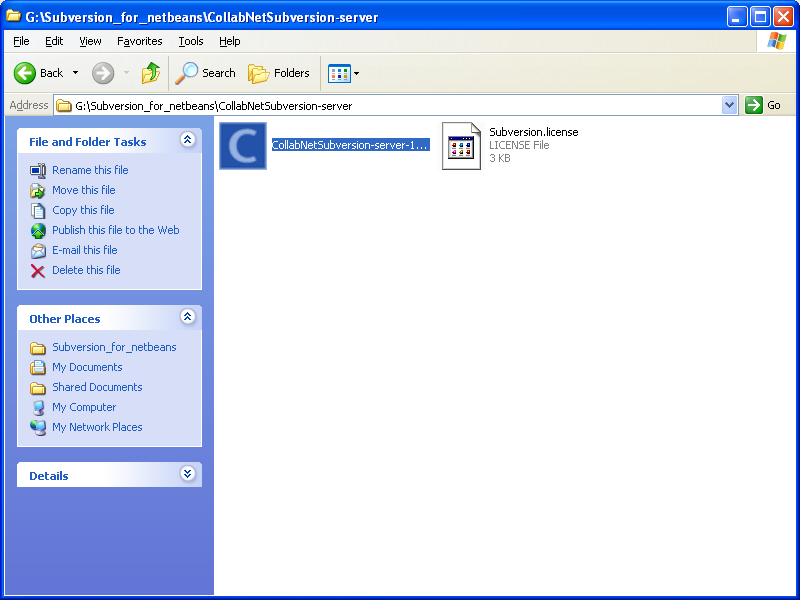
}

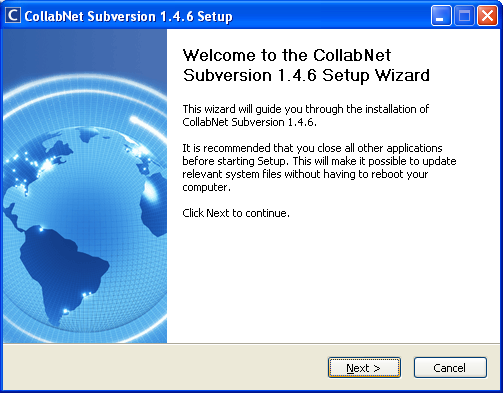
# APPENDIX E

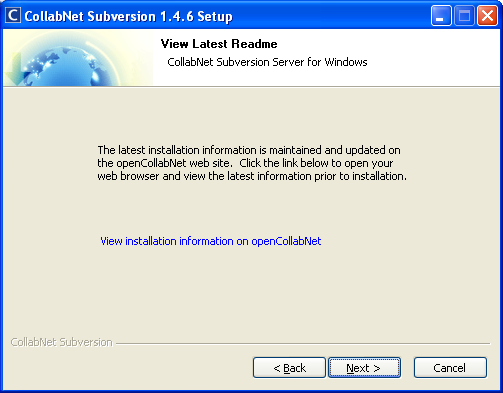
**Subversion Installation and Configuration with Net beans IDE**

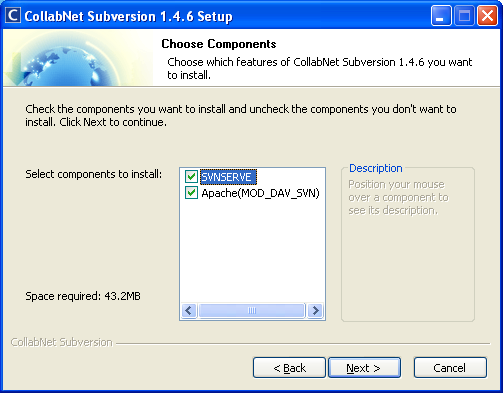
1) Download Subversion server from [**http://subversion.tigris.org/**](http://subversion.tigris.org/)**.**

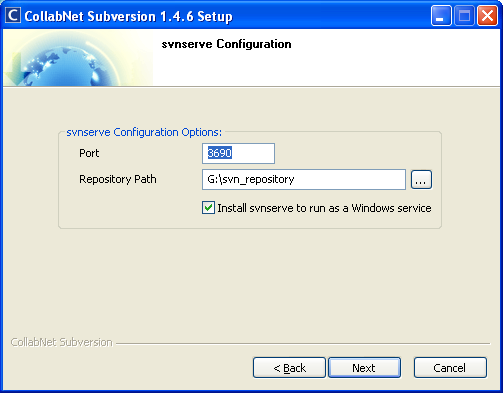
These are the steps for configuring and installing version management tool with our development environment at the start of the project. Different versions of source code have been maintained in the Subversion repository. These steps can be used to recreate Subversion repository in case of a hardware failure of any other unprecedented loss of data.



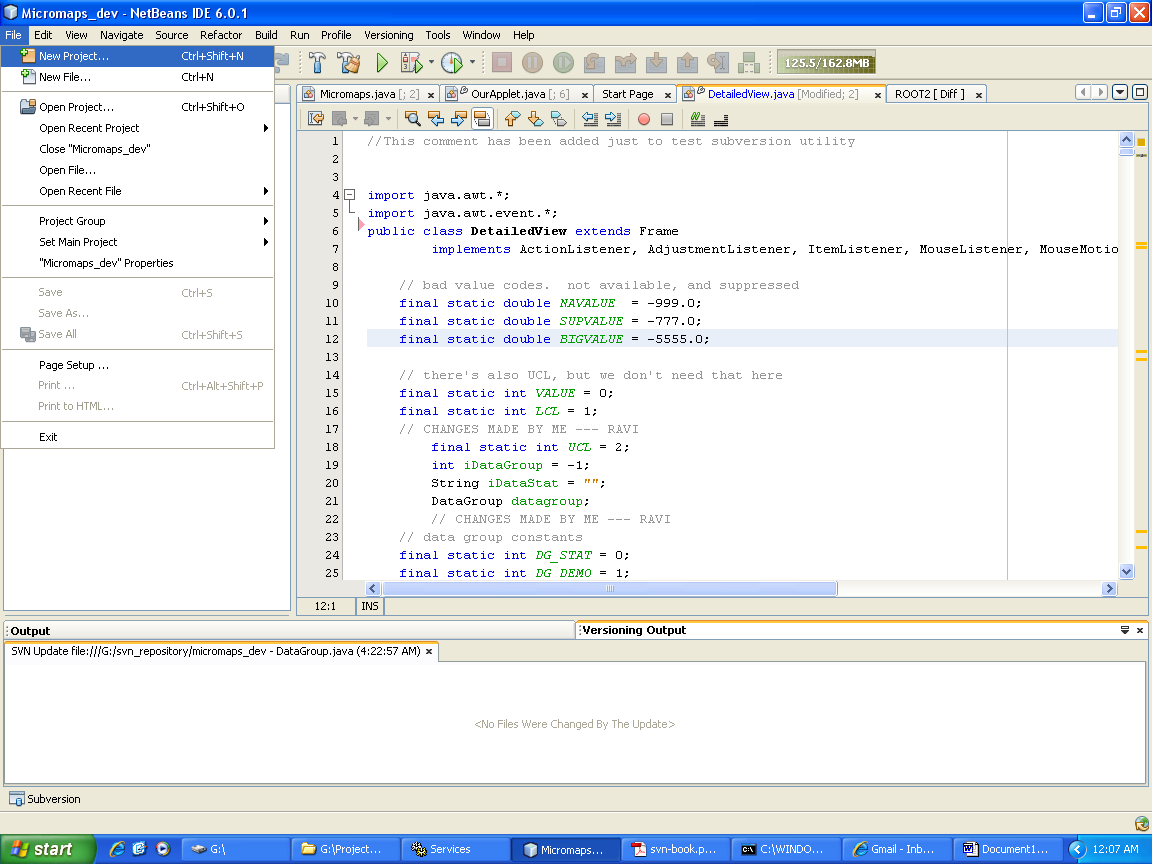




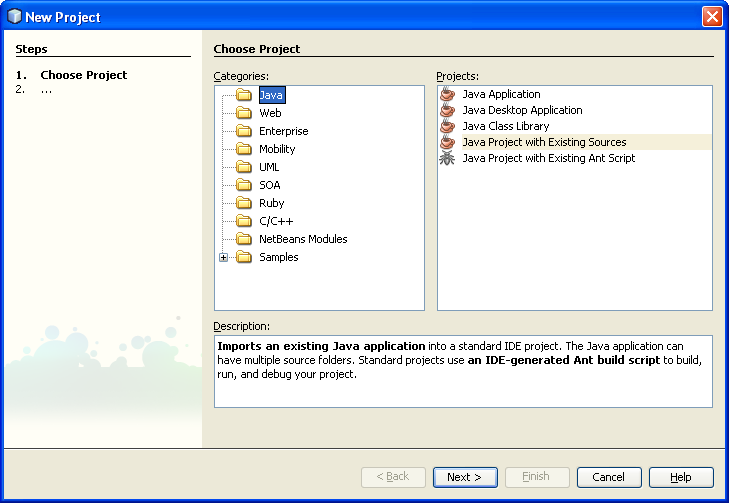


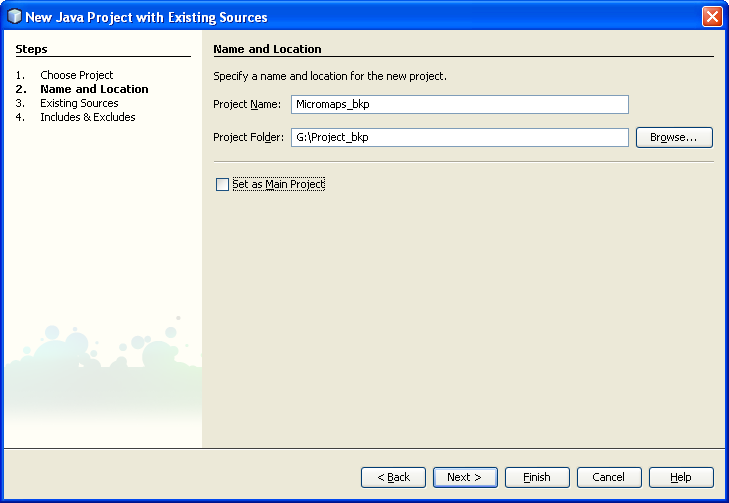


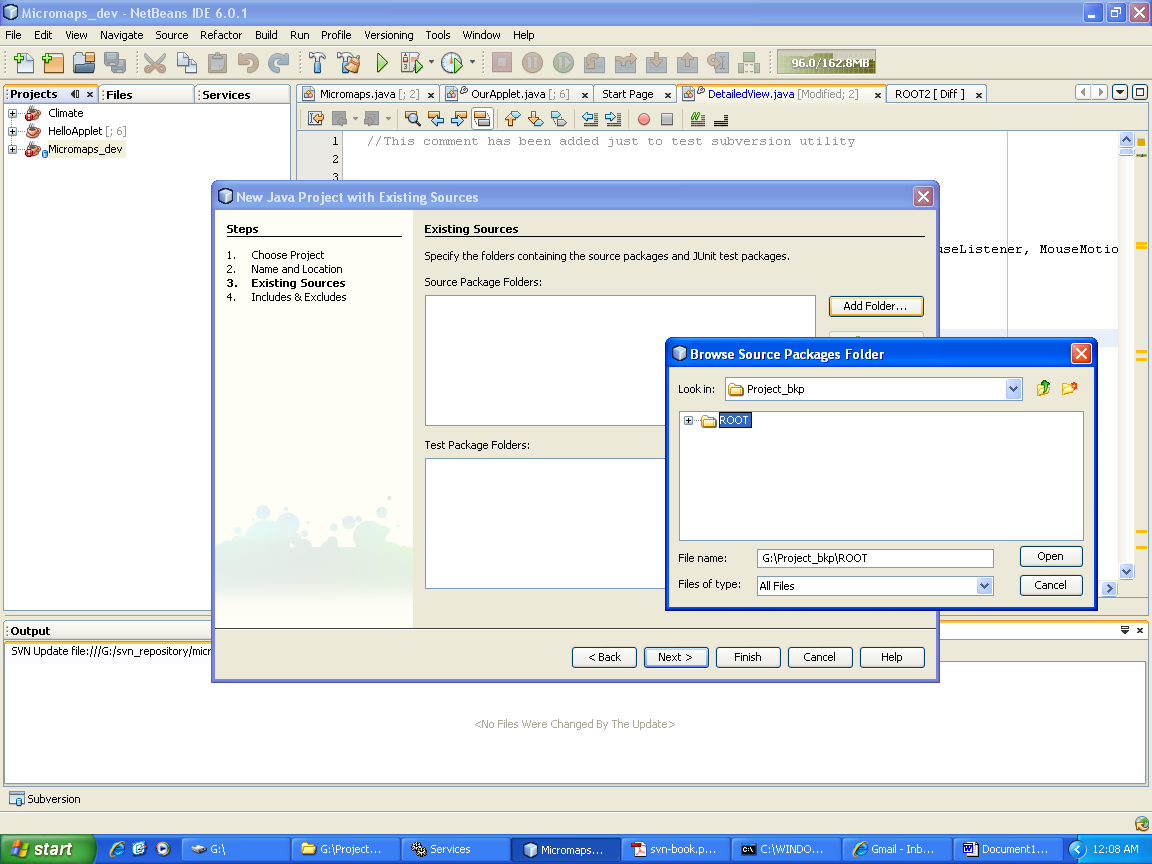
1. **Configuring Net beans project with Subversion**



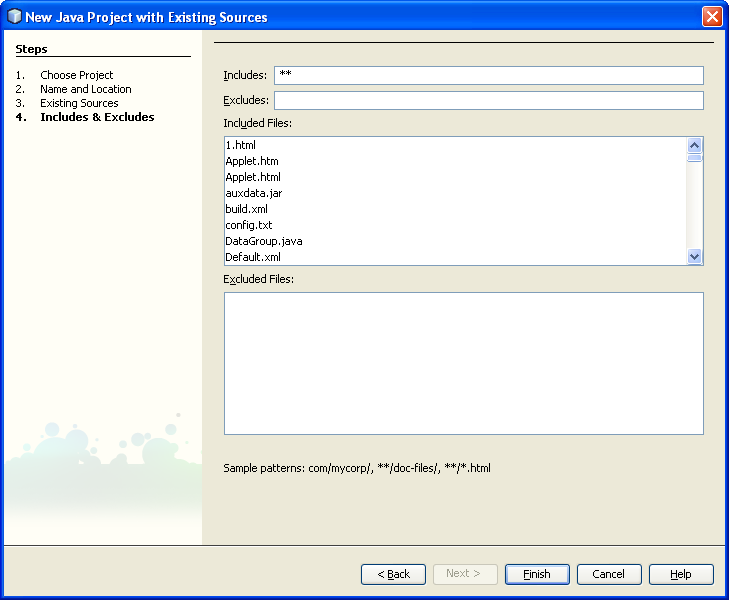
Create a New Project



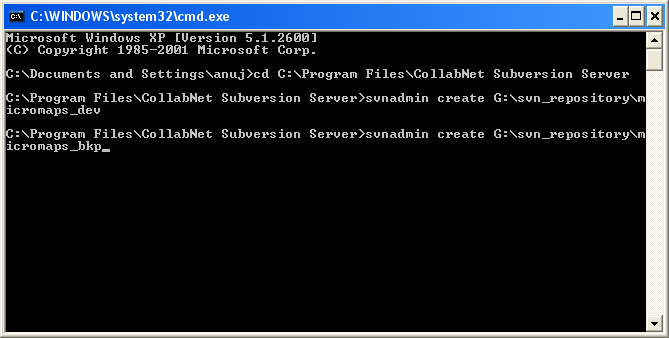




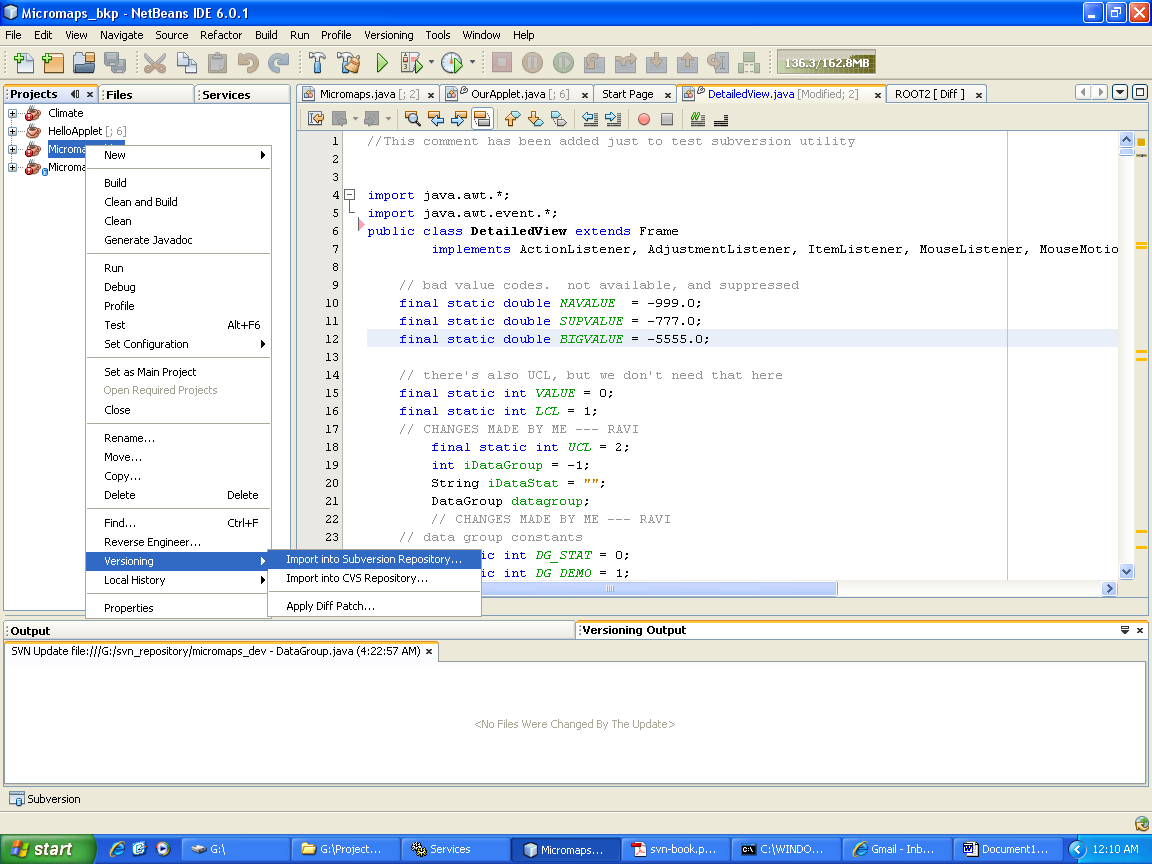
Add Source Code to project



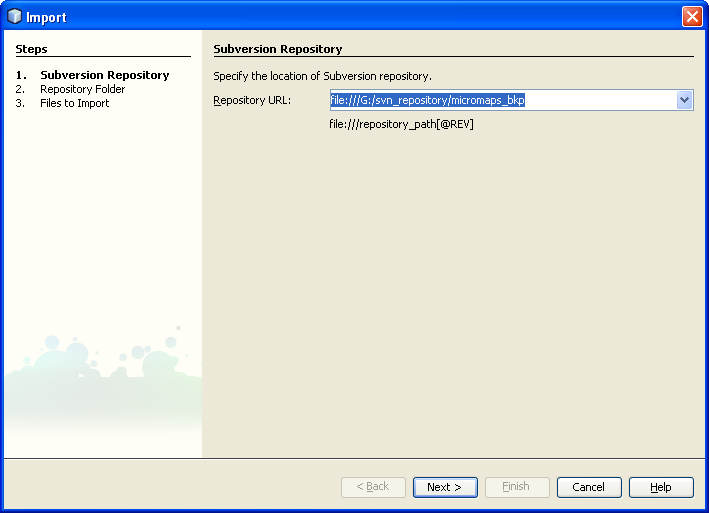
Click on finish.



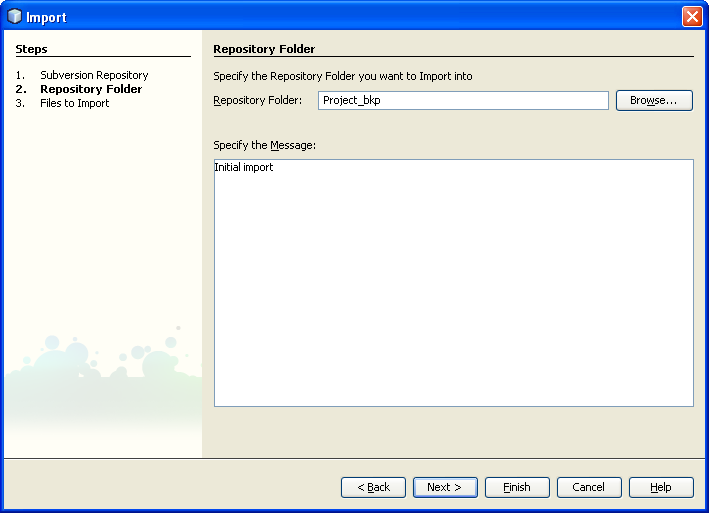
Create a new Subversion repository by using svnadmin.



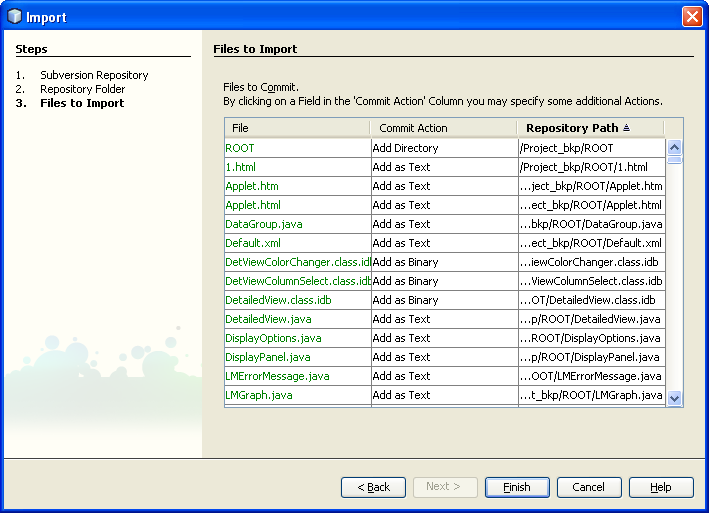
Right Click on newly created project 🡪 versioning 🡪 import to Subversion repository.



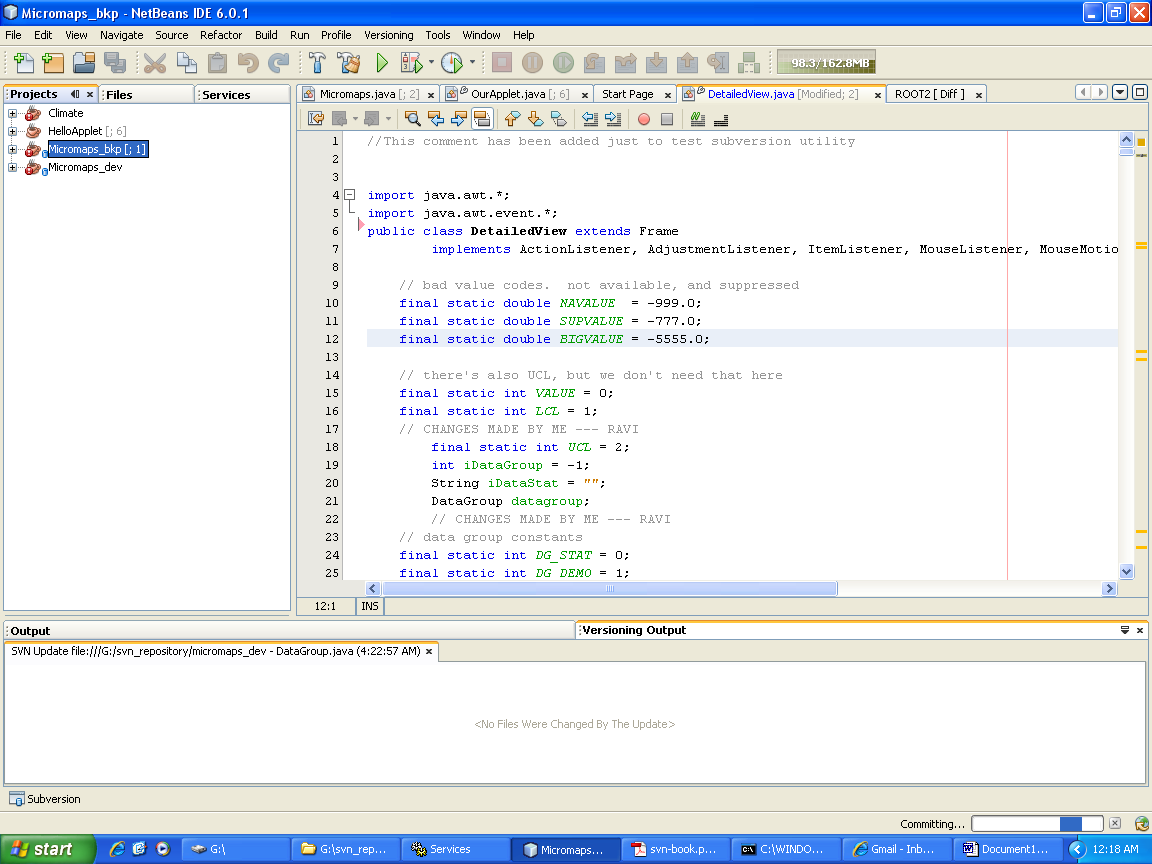
Give the repository path.



Enter Repository Folder name



List of files to be imported to repository is generated



You can see committing in progress at bottom right.

Subversion is now implemented!!!!