

# *Sirius Software*

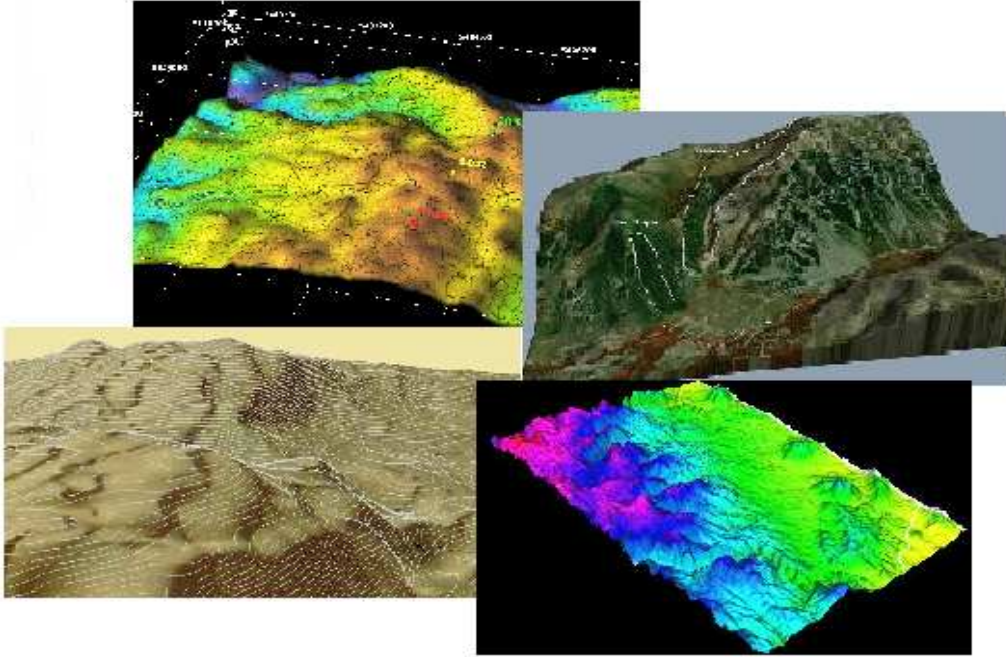
---

Duygu Yapa 1449222

Ayşe Turan 1449164

Duygu Altınok 1502095

Elif Kamer Karataş 1448836



## THE STRIDER

### *[Final Design Report]*

---

19.1.2009

## Contents

1.	INTRODUCTION .....	5
1.1.	Purpose of the Document .....	5
1.2.	Project Topic .....	5
1.3.	Project Definition .....	5
1.4.	Project Scope .....	5
2.	DESIGN CONSIDERATIONS .....	6
2.1.	Constraints and Limitations .....	6
2.1.1.	Time .....	6
2.1.2.	Performance .....	6
2.2.	Design Goals and Objectives .....	7
2.2.1.	Portability .....	7
2.2.2.	Usability .....	7
2.2.3.	Reliability .....	7
2.2.4.	Approach and Modeling .....	7
3.	CURRENT STATUS .....	8
3.1.	Current State .....	8
3.2.	Prototype Implementation .....	8
4.	LIBRARIES AND TOOLS .....	8
4.1.	Languages and Platforms .....	8
4.1.1.	Java .....	8
4.1.2.	Eclipse .....	9
4.1.3.	NetBeans .....	10
4.2.	Geographical Information Systems .....	10
4.2.1.	GeoTools .....	10
4.3.	Graphics Libraries .....	10
4.4.	Other Tools and Libraries .....	10
4.4.1.	XML .....	10
4.4.2.	MySQL .....	11

5.	USER INTERFACE DESIGN .....	11
5.1.	Main Window.....	13
5.2.	File Menu .....	13
5.3.	Terrain Menu .....	14
5.4.	Weather Menu.....	16
5.5.	Equipment Menu .....	16
5.6.	Help Menu .....	18
5.7.	Member Information Panel .....	19
5.8.	Constraints Panel.....	21
5.9.	3D Visualization of the Mountain Panel .....	22
5.10.	Get Activity Plan Window .....	23
6.	ARCHITECTURAL DESIGN .....	24
6.1.	Package and Class Design.....	24
6.1.1.	strider package .....	24
6.1.2.	strider.definedTypes package .....	25
6.1.3.	strider.gis Package .....	27
6.1.4.	strider.planner Package .....	27
6.1.5.	strider.gui Package.....	29
6.1.6.	strider.gui.helpers Package.....	29
6.1.7.	strider.gui.helpers.member Package .....	30
6.1.8.	strider.gui.helpers.equipment Package.....	31
6.1.9.	strider.gui.helpers.project Package .....	32
6.1.10.	strider.gui.helpers.weather Package .....	33
6.1.11.	strider.gui.helpers.visualization Package .....	34
6.1.12.	strider.gui.helpers.simulation Package .....	34
6.1.13.	strider.gui.windows Package .....	35
6.2.	Method and Class Interfaces .....	45
6.2.1.	strider package .....	45
6.2.2.	strider.gis package .....	45
6.2.3.	strider.planner package .....	45
6.2.4.	strider.gui.helpers.member package.....	46
6.2.5.	strider.gui.helpers.equipment package.....	47
6.2.6.	strider.gui.helpers.project package .....	47

6.2.7. strider.gui.helpers.weather package .....	48
6.2.8. strider.gui.helpers.visualization package .....	48
6.2.9. strider.gui.helpers.simulation package .....	49
6.2.10. strider.gui.windows package .....	50
6.3. Design of Expedition Plan .....	57
6.3.1. Considerations on maps .....	57
6.3.2. Considerations on route planning .....	59
6.3.3. Algorithmic design for route .....	61
6.3.4. Design of camping plan .....	72
6.3.5. Design of food&water planning .....	73
6.3.6. Design of time planning .....	73
6.3.7. Design of equipment planning .....	75
6.3.8. Design of carriage planning .....	78
6.4. Functional Modeling .....	79
6.4.1. Data Flow Diagrams .....	79
6.4.2. Data Dictionary .....	84
6.5. Data Design .....	92
6.5.1. Database Design .....	92
6.5.2. File and Folder Formats and Syntax .....	96
6.6. Behavioral Design .....	101
6.6.1. Strider General Behavior .....	101
6.6.2. Member Operations .....	102
6.6.3. Chosen Member Operations .....	105
6.6.4. Project Operations .....	108
6.6.5. Equipment Operations .....	112
6.6.6. Map Operations .....	115
6.6.7. Weather Operations .....	117
6.6.8. Visualization Operations .....	119
7. CONCLUSION .....	120
8. APPENDIX .....	121
9. REFERENCES .....	122

# **1. INTRODUCTION**

## **1.1. Purpose of the Document**

This report is written to show final design of Strider software by Strider project team. The headings and parts of this document are divided appropriately by grouping relevant subjects of whole system. Project group has tried to give the definition and design of the software system by dividing subsystems and showing the relations with each other. Also, the initial design report is detailed and the future work of that report has done in this report. In addition, the other necessary issues like implementation details and specifications are handled.

## **1.2. Project Topic**

The topic of this project is planning a climb for mountaineering clubs. Project name is Strider. Strider is being developed by a team of Sirius Software.

## **1.3. Project Definition**

The software basically finds a route on a terrain for mountaineers. It will provide the best activity plan for reaching a selected mountain peak by taking into account the specified constraints. In addition, it makes a visualization of the terrain and a simulation of the route. Software uses some kind of information from mountaineers, database and website to make all of these facilities.

## **1.4. Project Scope**

The scope of the project has been described in the Requirement Analysis Report. Here, a brief of the scope will be given.

The software should be used by instructor mountaineer, because only instructor can know about the capabilities of the mountaineers in a club or decide to the date of an activity

This guy has some responsibilities and abilities on the software;

- He should give the climbers' information in the club to the system any time and this information will be kept in system until an instructor delete or update them.
- Before running system for getting an activity plan, he should give terrain information to the system by giving a ".dted" or ".dem" format map, the region (ex; Ankara, Turkey) and optionally vector map and satellite image.
- He can give some constraints like safest route, time and distance constraints and checkpoints.
- He can mark the avalanche or rock falling risk places on the visualization of the terrain at our user-friendly graphical user interface.
- He can make a walk-through on terrain by useful buttons.

After giving a map, program will show a visualization of the terrain. Then he can run the program. The program gives an activity plan (optionally in a report) which includes climbing route, estimated duration of climbing, food and equipment list, emergency equipment list and camping plan. He can see a simulation of the route on the terrain, if he wants. The additional features will be described in the rest of this document.

## 2. DESIGN CONSIDERATIONS

### 2.1. Constraints and Limitations

#### 2.1.1. Time

The project should be completed until July 2009. In addition a demo of this project should be prepared until the end of January 2009. The detailed project schedule is given as a Gantt chart in Appendix.

#### 2.1.2. Performance

Performance is an important issue and makes us anxious, because finding an optimum way in a terrain with lots of constraints which have different priorities is a hard problem.

Hence we will try different solutions for this problem to reach the best performance. In addition, the visualization and simulation part we will use efficient ways.

## **2.2. Design Goals and Objectives**

### **2.2.1. Portability**

We have thought about this issue and decided to use portable languages and tools. In other words, this was a constraint to choose libraries and toolkits. So, we will try to make software portable on Windows and Linux platforms.

### **2.2.2. Usability**

We have tried to design user friendly software and correspond to mountaineers needs. In this context, we have made interviews with professional mountaineers and obtained their ideas about planning a climb. We have examined how much we make this software useful for mountaineers. At the end, we canceled some unnecessary and absurd parts from the project. For example, we canceled rafting or skiing experience of climbers because a climber never carries rafting or skiing equipments when he goes to climbing even if he has these kinds of experiences. We also added some extra features that make software better for users like visualization of the terrain.

### **2.2.3. Reliability**

As we mentioned before, we have made our studies in wide range. So, we have thought almost all aspects of the climbing issue. We have taken up references from the mountaineers Hasan Hüseyin BOĞAZ and Tunç FINDIK and the book of “Zirvelerin Özgürlüğü”. Besides we have searched lots of libraries, tools and algorithms for the implementation part and selected the most appropriate ones for our project.

### **2.2.4. Approach and Modeling**

Because of the complexity of our project, we have decided to use Object Oriented Approach. This approach models our system so makes the design more understandable and the Object Oriented Design elaborates the models to produce implementation

specifications. For representing these models, we use Unified Modeling Language (UML) that has a number of different notations for representing models.

## 3. CURRENT STATUS

### 3.1. Current State

We have started to use tools and libraries. We have used Geotools libraries and implement some features. We have read vector maps and visualize them. We have added some features to visualization part like zooming and moving. We have designed graphical user interface and database. On the other hand, when we designed the packages and classes of the project in Architectural Design part, we have determined the methods of classes and defined them.

### 3.2. Prototype Implementation

The prototype of Strider will contain an understandable graphical user interface with its main features. In addition to these, gui will be able to get map formats and show a 3D visualization of the terrain. For the visualization part the Geotools libraries will be used and this part will have futures. We considered that these features will be zooming in, zooming out on map and moving on it. There may be mouse events on map for showing coordinate information of any point of map, and geting start, end coordinates, checkpoint, avalanche/ rock falling risky places.

## 4. LIBRARIES AND TOOLS

### 4.1. Languages and Platforms

#### 4.1.1. Java

Java is a programming language originally developed by Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives



much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. The syntax of Java is largely derived from C++. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object oriented language. All code is written inside a class and everything is an object, with the exception of the intrinsic data types (ordinal and real numbers, boolean values, and characters), which are not classes for performance reasons. [1] We preferred Java for its high-level object orientation, its rich IDEs, wide developer support and documentation, wide library choices; but the our main reason was that we wanted to use GeoTools which is a Java library. Also for our project's visualization & simulation parts, Java was the best choice for its available libraries.

#### 4.1.2. Eclipse

Eclipse is a software platform comprising extensible application frameworks, tools and a runtime library for software development and management. It is written primarily in Java to provide software developers and administrators an integrated development environment (IDE). In its default form it is meant for Java developers, consisting of the Java Development Tools (JDT). Users can extend its capabilities by installing plugins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Language packs provide translations into over a dozen natural languages. It was easy for us to decide to use Eclipse as it's the most powerful Java IDE; it has dozens of plugins to support our work, it has automatic code generation as well as easy usage such as showing and explaining runtime, compile-time and code mistakes. Also Eclipse is easy to use with “svn” as well as it provides the Eclipse Workbench - views, editors, perspectives, wizards; Equinox OSGi - a standard bundling framework, Core platform - boot Eclipse, run plug-ins, the Standard Widget Toolkit (SWT) - a portable widget toolkit, JFace - viewer classes to bring model view controller programming to SWT, file buffers, text handling, text editors etc.[2]

#### **4.1.3. NetBeans**

A free, open-source Integrated Development Environment for software developers. The NetBeans IDE provides several new features and enhancements, such as rich PHP, JavaScript and Ajax editing features, improved support for using the Hibernate web framework and the Java Persistence API, and tighter GlassFish v3 and MySQL integration. [3] We used NetBeans for designing our GUI interfaces; as we use Java by far NetBeans is the best choice for interface design for its easy usage and developer-friendly.

### **4.2. Geographical Information Systems**

#### **4.2.1. GeoTools**

GeoTools is an open source (LGPL) Java code library which provides standards compliant methods for the manipulation of geospatial data, for example to implement Geographic Information Systems (GIS). [4] We decided to use GeoTools as we needed a GIS library and GeoTools provides all GIS capabilities. Moreover, as we use Java; GeoTools & Java combination would be powerful enough.

### **4.3. Graphics Libraries**

We decided to use GeoTools' graphic capabilities for our project's visualization and simulation parts, since we mostly use GeoTools for the other parts of our project. However, we may use NasaWorld Wind libraries for visualization parts because of its useful features.

### **4.4. Other Tools and Libraries**

#### **4.4.1. XML**

Extensible Markup Language (XML) provides a foundation for creating documents and document systems. XML operates on two main levels: first, it provides syntax for document markup; and second, it provides syntax for declaring the structures of documents. XML's simplicity is its key selling point, perhaps even its strongest feature. Our team have plenty of

reasons to use XML : simplicity , extensibility , interoperability. For the simplicity , XML's rigid set of rules helps make documents more readable to both humans and machines. XML document syntax contains a fairly small set of rules, making it possible for our team to get started right away. For the extensibility; XML is extensible in two senses. First, it allows developers to create their own DTDs, effectively creating 'extensible' tag sets that can be used for multiple applications. Second, XML itself is being extended with several additional standards that add styles, linking, and referencing ability to the core XML set of capabilities. XML complements Java, a force for interoperability, very well, and a considerable amount of early XML development has been in Java. A generic application programming interface (API) for parsers, the Simple API in XML (SAX), is freely available. [5]

#### 4.4.2. MySql

MySQL database is the world's most popular open source database because of its fast performance, high reliability, ease of use, and dramatic cost savings. Our team preferred MySQL for its easy usage and mostly for its Java compability. [6] Since MySQL is also a Sun Microsystems product ; not only Java has JDBC support for SQL , also the combination of Java & MySQL is a very powerful combination due to its high performance.

## 5. USER INTERFACE DESIGN

We have made some changes and additions to our user interface. Here is a list of those changes and additions:

- Since we are assuming that the users of our software will be instructor mountaineers who has the enough information about equipments, we have thought that they should be able to access the equipments in the system before making the plan. Here, accessing the equipments means, listing the equipments in the system, defining new equipments or removing out-dated or unused equipments from the system permanently. For this purpose, we have added an “Equipment Menu”.

- We have changed “Terrain Menu” a little bit. Instead of pressing “Add Shape File” to add a shape file and “Add Satellite Image” to add a satellite image etc, user will just press “Add Map” and he/she will be able to specify all map types. We have removed “Browse Data” menu item for this reason. Since specifying country and city information was in “Browse Data” window, we have moved them to “Constraints” panel in main window. During our research on GIS we have learned that the elevation data file or shape file should not be necessarily unique. There can be more than one elevation maps or shape files for the same terrain. Before learning this, we were assuming that there will be exactly one elevation map and at most shape file or satellite image for a terrain. Seeing the assumption we made was wrong, we changed this part also. Now, user can specify as many maps as he/she wants and he/she can remove any map (including elevation maps) from the project. However, if user tries to get activity without specifying any elevation map the software will not generate any an activity plan.
- We have moved choosing member for an activity option from “See all Members” window to main window as we thought this would be more convenient. If user presses “Choose Member” button in the Main Window, a new window listing the unchosen members in the system will appear instead of “See all Members” window. Here, user will be able to choose more than one member by selecting all and pressing “Ok” button.
- We have made changes in taking weather information part. At the beginning we were taking weather information from the internet only. We have added an option for user to specify the necessary information manually in case of lack of internet connection or lack of information. If user does not specify any information, we will use default values and warn the user for this situation.

Again, there may be some changes in the design of user interface at the next steps of the project. User screens may be designed to be more user-friendly as the need arises. However, the functionalities the screens serve will remain more or less stable.

In the following sections, we will explain the main points of our graphical user interface.

## 5.1. Main Window

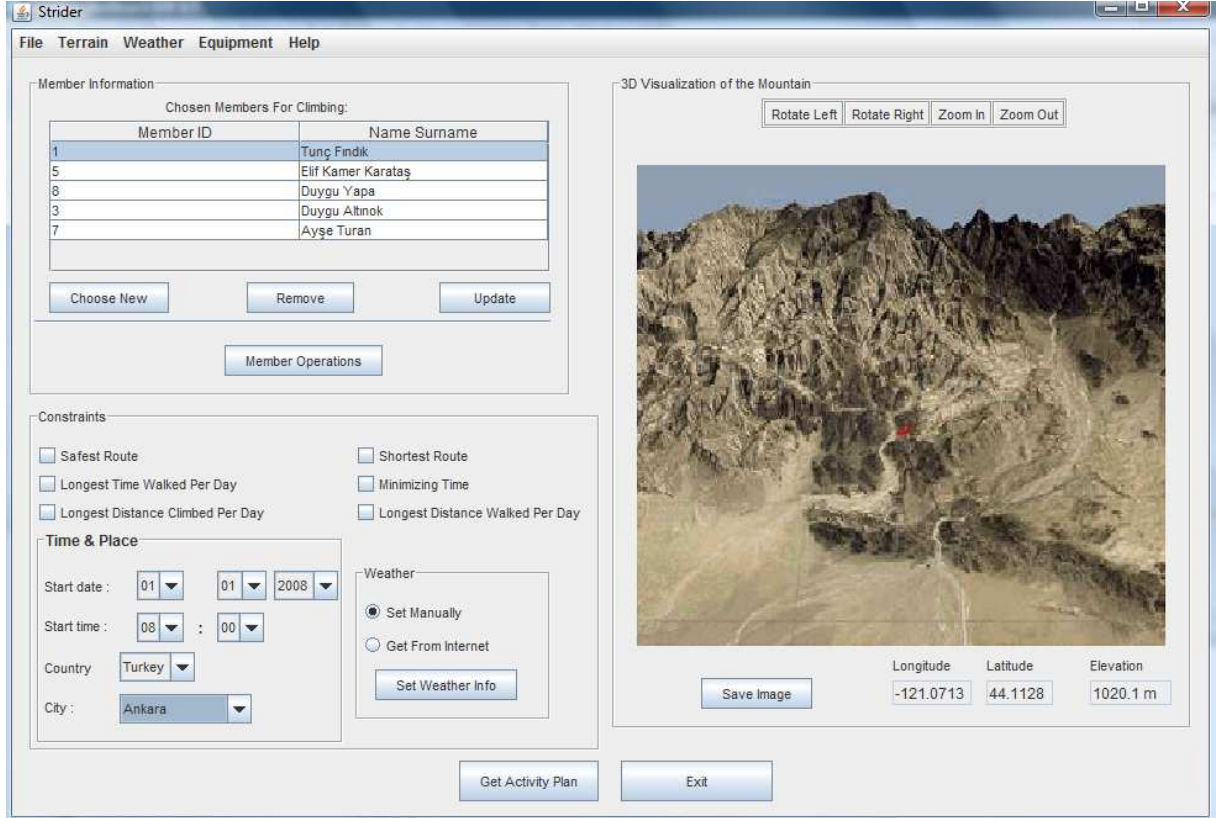


Figure 5.1. Main Window of Strider

In Figure 5.1, you see the main window of our software. It consists of a menubar including File, Terrain, Weather, Equipment and Help menus, Member Information, Constraints and 3D Visualization of the Mountain Panels and Get Activity Plan and Exit buttons.

## 5.2. File Menu

From File Menu, the following options can be chosen:

- **New Project:** If “New Project” is clicked; after closing the old one, a new project will be open with default values.

- **Open Project:** If “Open Project” is clicked after closing the current one, a file chooser for opening a “.srs” file will appear. Clicking the desired project with “.srs” extension, and “Open”, the project will be open.
- **Save Project:** If “Save Project” is clicked, a file chooser for saving the current project will appear and writing the name and clicking “Save” will save the project to the desired location.
- **Exit:** When “Exit” is clicked, if the current project has not saved recently, first an option pane that asks whether to save the project before exiting or not, will appear. According to chosen option, Strider will exit either with saving the project or without saving it.

### 5.3. Terrain Menu

From Terrain Menu, the following options can be chosen:

- **Add Map :** If this option is chosen, a filechooser which shows files with extensions “dted”, “dem”, “shp” and “geotiff” will appear as shown in Figure 5.2. Selecting a map and pressing “Add Map” will add the selected file to project. The effect of adding a map to the project will also be seen in visualization.

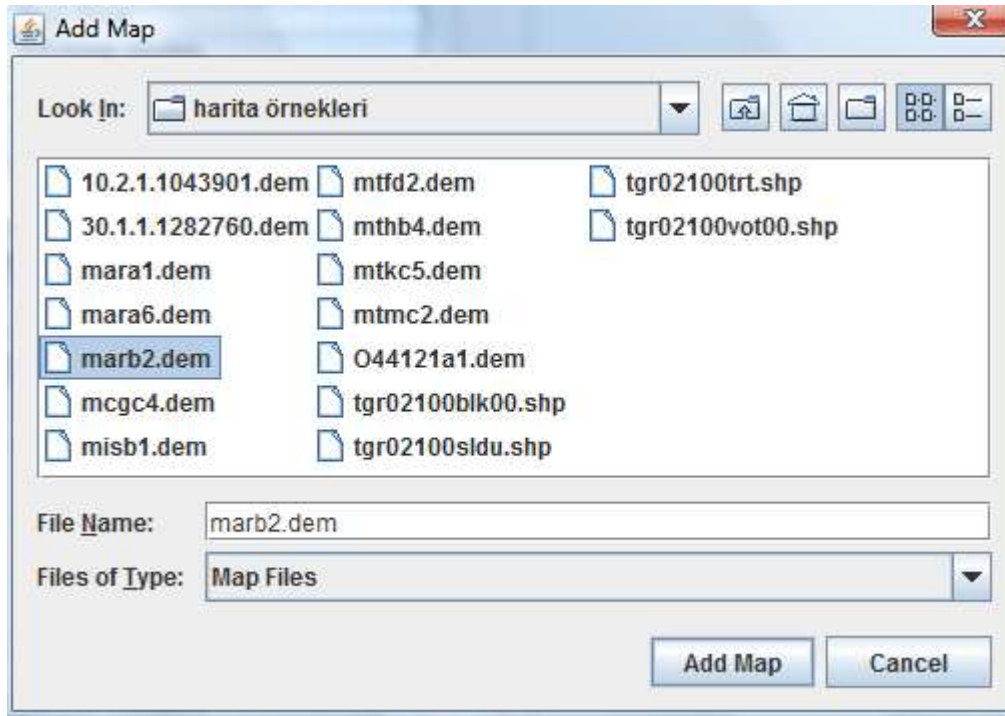


Figure 5.2.Add Map Window

- **Remove Map(s)** : If this option is chosen, a window including the list of project maps will appear (Figure 5.3). Selecting one or more maps from the list and pressing “Remove Selected Map(s)” will remove the specified maps from the project. The effect of removing map(s) from the project will also be seen in visualization.

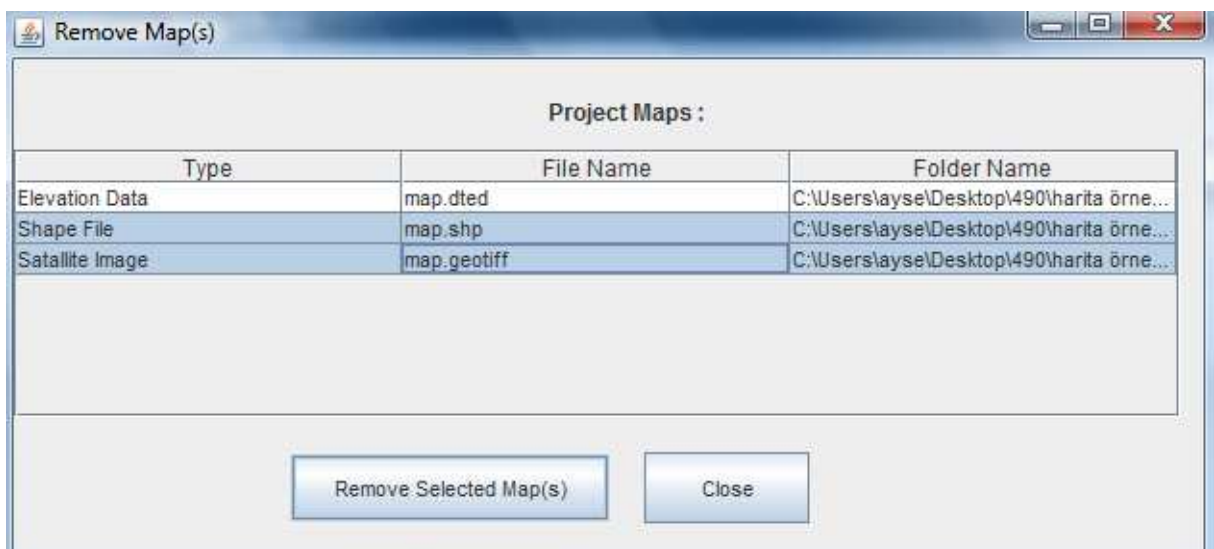


Figure 5.3.Remove Map(s) Window

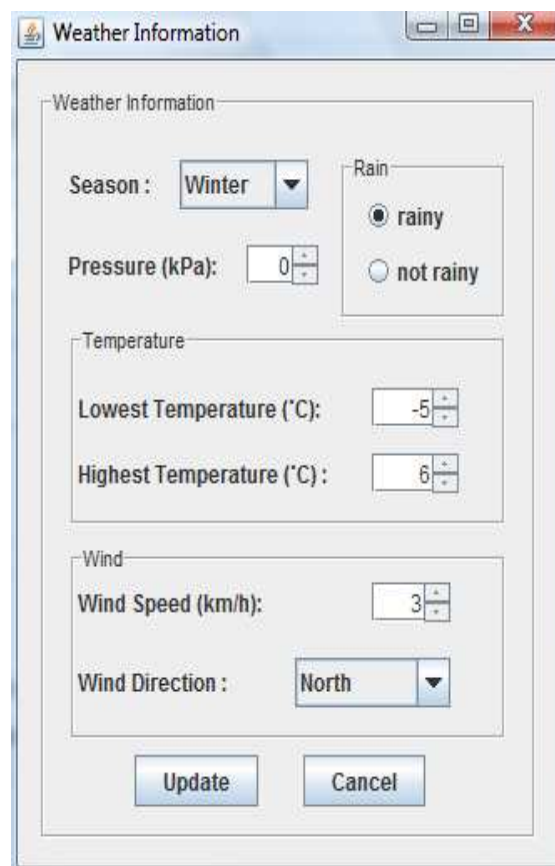
## 5.4. Weather Menu

In Weather Menu there is only one option : “See Weather Information”. If this option is chosen, then the current weather data of the project will be shown to the user (Figure 5.4). Pressing “OK” will redirect to the main window. (Figure 5.1)



Weather Information	
Season :	WINTER
Pressure (kPa):	102.3
Lowest Temperature (°C):	-5
Highest Temperature (°C) :	3
Wind Direction :	NorthEast
Wind Speed (km/h):	5
isRainy :	Yes
<input type="button" value="OK"/>	

Figure 5.4. See Weather Window



Weather Information	
Season :	Winter
Pressure (kPa):	0
Rain <input checked="" type="radio"/> rainy <input type="radio"/> not rainy	
Temperature	
Lowest Temperature (°C):	-5
Highest Temperature (°C) :	6
Wind	
Wind Speed (km/h):	3
Wind Direction :	North
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Figure 5.5. Update Weather Window

## 5.5. Equipment Menu

In this menu, there is only one option : “Equipment Operations”. Choosing that option triggers a window, containing the list of equipments in the system, opening (Figure 5.6).





Figure 5.6. Equipment Operations Window

From “Equipment Operations” window user can choose the following options:

- **Add New Equipment:** If this option is chosen, a window to define a new equipment will appear (Figure 5.7). Filling the fields and pressing “OK” will add the newly defined equipment to the system after doing necessary error checks, and redirect to “Equipment Operations” window.

Figure 5.7. Add Equipment Window

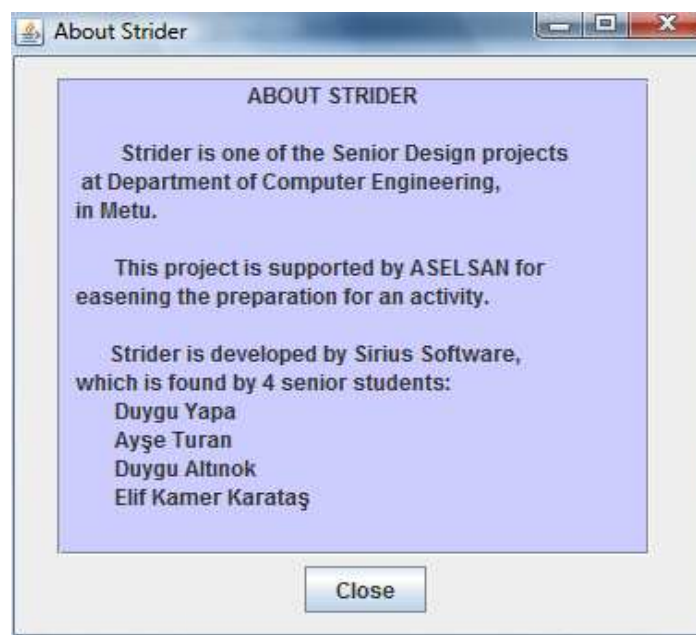
Figure 5.8. Update Equipment Window

- **Remove Equipment:** If this option is chosen, the selected equipment will be removed from the system after doing necessary error checks and confirmations. After removing an equipment, user will stay in “Equipment Operations” window.
- **Update Equipment:** If this option is chosen, a window for user to update the selected equipment will appear (Figure 5.8).
- **Close:** Choosing this option will close “Equipment Operations” window and redirect to main window (Figure 5.1).

## 5.6. Help Menu

From Help Menu, the following Options can be chosen:

- **Help Topics:** If Help Topics option is chosen, then the user manual file will be open without closing the main window.
- **About Strider:** If this option is chosen, then a short description and information about the developers of Strider will appear (Figure 5.9).

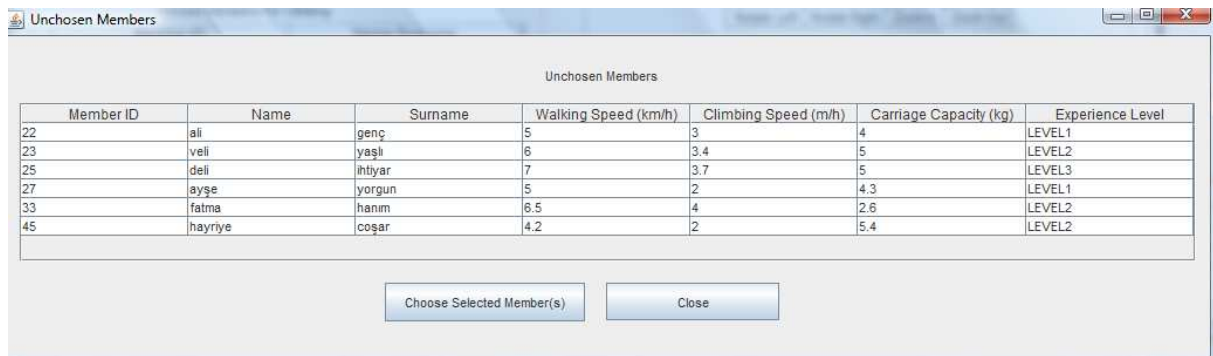


*Figure 5.9. About Strider Window*

## 5.7. Member Information Panel

In Member Information Panel, there is a list of members (their ids and names) that will go climbing.

- If user presses “Choose New” button to include member(s) in the activity then a window containing the list of all unchosen members with their whole informations will appear (Figure 5.10). User can select more than one member, and pressing “OK” button will add all selected members to the project’s chosen member list. Pressing “OK” or “Cancel” button will redirect to main window, reflecting the changes in the chosen member list to the chosen member panel, if any.



The screenshot shows a window titled "Unchosen Members". Inside the window, there is a table with the following data:

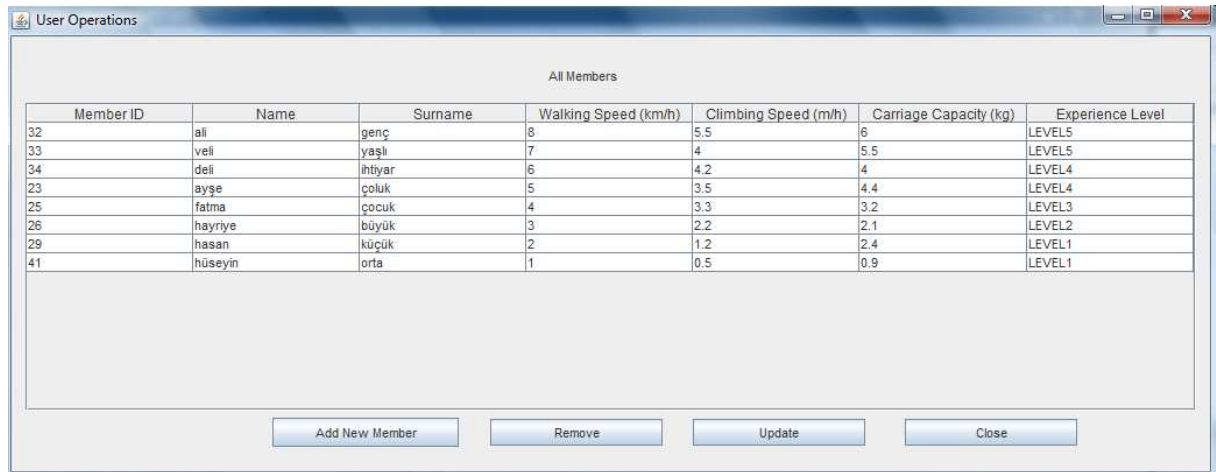
Member ID	Name	Surname	Walking Speed (km/h)	Climbing Speed (m/h)	Carriage Capacity (kg)	Experience Level
22	ali	genç	5	3	4	LEVEL1
23	veli	yaşlı	6	3.4	5	LEVEL2
25	deli	ihliyar	7	3.7	5	LEVEL3
27	ayşe	yorgun	5	2	4.3	LEVEL1
33	fatma	hanım	6.5	4	2.6	LEVEL2
45	hayriye	coşar	4.2	2	5.4	LEVEL2

Below the table, there are two buttons: "Choose Selected Member(s)" and "Close".

*Figure 5.10. Choose Member for Activity Window*

- If user chooses one of them and presses
  - “Remove Button”, then that member is removed from chosen members list.
  - “Update Button”, then a window (given in Figure 5.13) that will contain the information of the selected member is shown. Updating the necessary field and pressing “OK” will updates the information of the selected member.

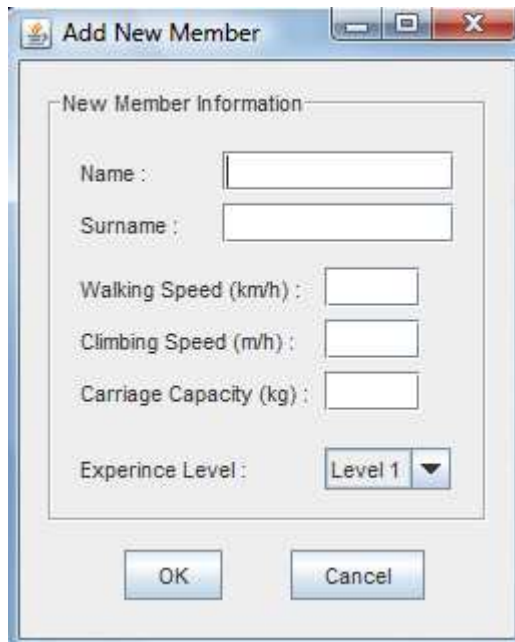
There is a button, User Operations Button, to see the information of all existing members (both chosen for climbing and not). If user presses this button, the window given in Figure 5.11 will appear.



*Figure 5.11. User Operations Window*

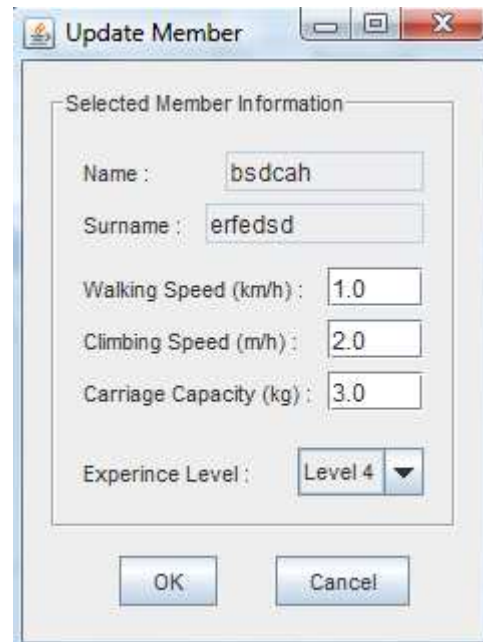
From User Operations Window, user can perform the following actions:

- **Add New Member:** If user presses “Add New Member Button”, then the window given in Figure 5.12 will appear. Filling the informations, and pressing “OK” gives the member a unique id and adds member to the database and “User Operations” window is updated accordingly.
- **Remove an Existing Member:** If user selects a member and presses “Remove Button”, then the selected member is removed from the database permanently and “User Operations” window is updated accordingly. If the removed user is in chosen members for climbing list, he will also be removed from that list.
- **Update Member Information:** If user selects a member and presses “Update Button”, then the information of the selected member will appear. (Figure 5.13) Changing the necessary fields and pressing “OK”, updates the information of the selected member.



The 'Add New Member' window contains a section titled 'New Member Information'. It includes five input fields: 'Name', 'Surname', 'Walking Speed (km/h)', 'Climbing Speed (m/h)', and 'Carriage Capacity (kg)'. Below these is a dropdown menu for 'Experince Level' currently set to 'Level 1'. At the bottom are 'OK' and 'Cancel' buttons.

Figure 5.12. Add New Member Window



The 'Update Member' window contains a section titled 'Selected Member Information'. It displays the same five input fields as the 'Add New Member' window, but with pre-filled values: 'Name' is 'bsdcah', 'Surname' is 'erfedsd', 'Walking Speed (km/h)' is '1.0', 'Climbing Speed (m/h)' is '2.0', and 'Carriage Capacity (kg)' is '3.0'. The 'Experince Level' dropdown is set to 'Level 4'. 'OK' and 'Cancel' buttons are at the bottom.

Figure 5.13. Update Member Window

- **Close Window:** If “Close” is pressed, User Operations Window will be closed returning back to the main window. (Figure 5.1)

## 5.8. Constraints Panel

In Constraints Panel, there are constraints that are taken into account when preparing the activity plan. User can choose more than one constraint. He also specifies the start date and time of the activity, country and city of the terrain that mountain is located. In addition he will specify the weather setting information. By choosing “Set Manually” option and pressing “Set Weather Info” a window showing the current weather information will appear (Figure5.5). Filling the necessary fields and pressing “OK” will update the weather information of the current project. If “Get from internet option” is chosen and “Set Weather Info” is pressed then after checking the internet connection and availability of the information, if everything seems okay, weather information will be taken from the internet. If any problem occurs, the software will give a warning to the user and use the current weather information not taken from internet.

## 5.9. 3D Visualization of the Mountain Panel

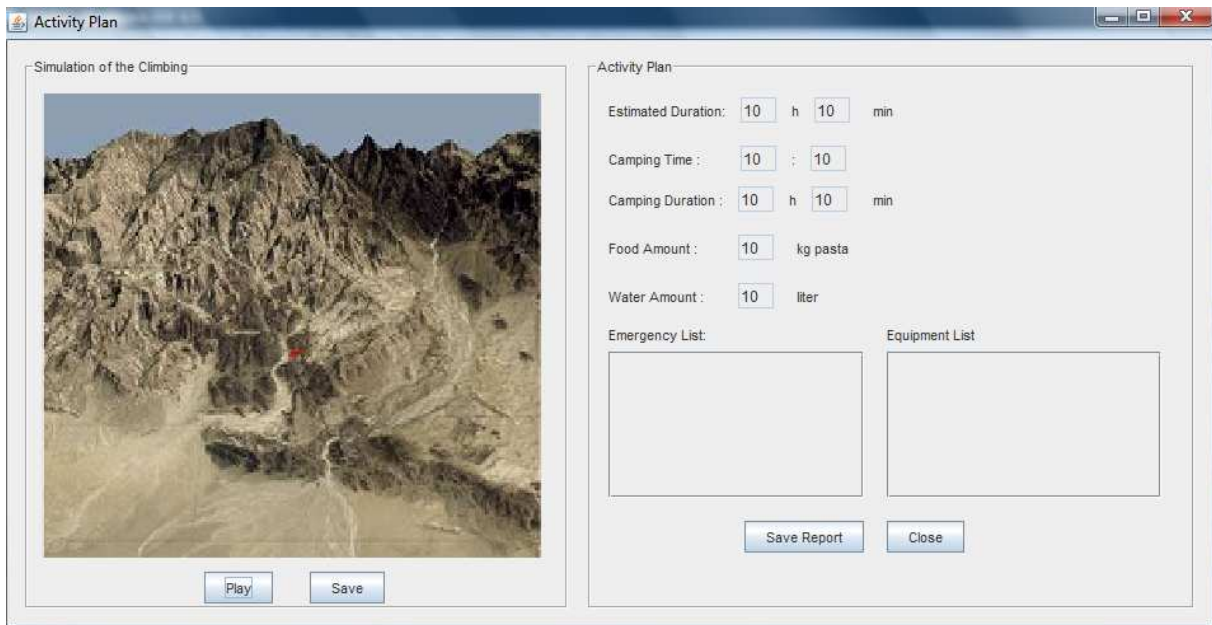
In this panel the 3D visualization of the mountain using all of the given maps will be shown.

- As user moves the mouse on the terrain, the longitude, latitude and elevation of the terrain will be shown in given boxes.
- If user clicks left mouse button then a popup menu will appear. In the popup menu the following options exist:
  - **Start coordinate:** If this option is chosen, then the clicked position will be the start coordinate. If there is a start point given before, that point will be removed since there can be only one start point.
  - **End coordinate:** If this option is chosen, then the clicked position will be the end coordinate. If there is an end point given before, that point will be removed since there can be only one end point.
  - **Checkpoint:** If this option is chosen, the clicked position will be added to the checkpoints list. During the preparation of the activity plan, passing through the given checkpoints will be taken into account as a constraint.
  - **Risk of avalanche:** By choosing this option, user can point the risk of avalanche.
  - **Risk of falling rocks:** By choosing this option, user can point the risk of falling rocks.
- If user clicks right mouse button, if there exists any points defined on that position, it will be removed.
- If user clicks the save image button, a file chooser will appear and pressing save, saves the image to a desired location.
- User can view the other sides of the mountain by pressing “Rotate Left” and “Rotate Right” buttons or keyboard’s left and right arrow keys.

- User can zoom in and out the terrain by pressing “Zoom In” and “Zoom Out” buttons or keyboard’s up and down arrow keys.

## 5.10. Get Activity Plan Window

When user presses “Get Activity Plan” button, after doing the error checks, the window shown in Figure 5.14 will appear.



The screenshot shows a software window titled "Activity Plan". It is divided into two main sections. The left section, titled "Simulation of the Climbing", contains a 3D rendered image of a rugged, rocky mountain terrain. Below this image are two buttons: "Play" and "Save". The right section, titled "Activity Plan", contains several input fields and two empty list boxes. The input fields are: "Estimated Duration:" with values "10" and "10" for hours and minutes respectively; "Camping Time:" with values "10" and "10" for hours and minutes; "Camping Duration:" with values "10" and "10" for hours and minutes; "Food Amount:" with a value of "10" and unit "kg pasta"; and "Water Amount:" with a value of "10" and unit "liter". Below these fields are two empty list boxes labeled "Emergency List" and "Equipment List". At the bottom of the right section are two buttons: "Save Report" and "Close".

*Figure 5.14. Activity Plan Window*

From the Activity Plan Window, user can simulate the climbing by pressing “Play Button”. If he presses “Save Button”, a file chooser appears and pressing “Save”, saves the simulation to the desired location.

In the Activity Plan Panel, there exists the generated output about the climbing. In case of more than one camping, all of the campings will be listed. If there is no camping, nothing will be displayed for camping.

If user presses “Save Report”, a file chooser for saving the activity plan will appear. Pressing “Save”, saves the report. If user presses “Close”, the Activity Plan will be closed without saving the activity plan report.

## 6. ARCHITECTURAL DESIGN

In this section, the architectural design of the project will be described. In the first part, package and class descriptions and class diagrams will be given. In the second part, the method and class interfaces of the packages will be described.

### 6.1. Package and Class Design

#### 6.1.1. strider package

“strider” package is the main package of the project (Figure 6.1). It consists of two classes namely “Main” and “DatabaseConnection” and 4 main packages namely “definedTypes”, “gis”, “planner” and “gui”.

- **Main Class** is the class that has the main method (Figure 6.2).
- **DatabaseConnection Class** is the class that is responsible for database connection (Figure 6.3).



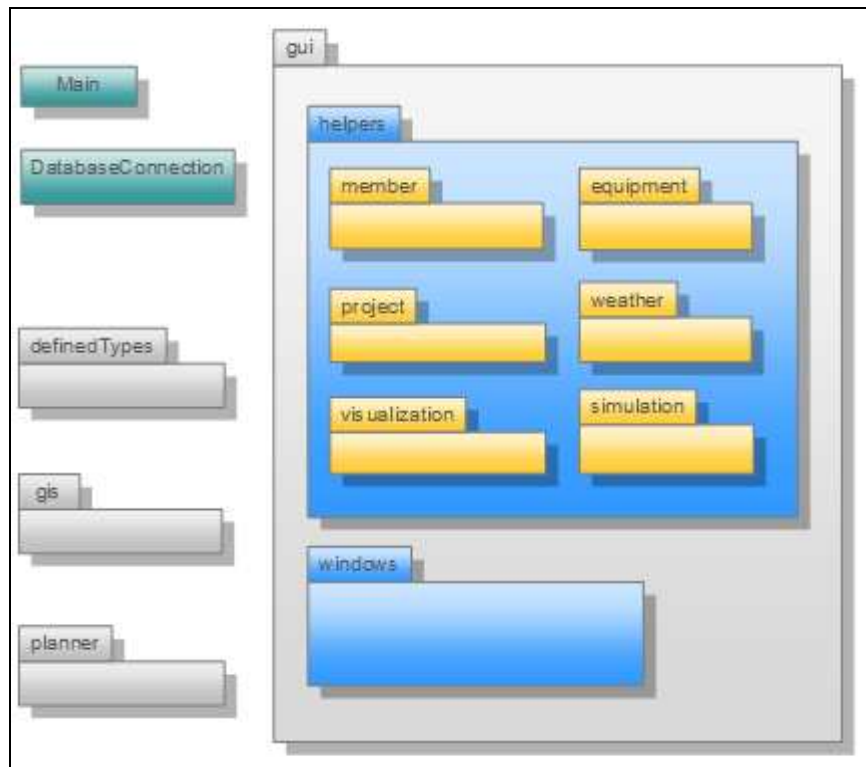


Figure 6.1. strider package

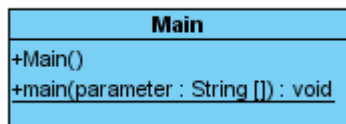


Figure 6.2. Main Class

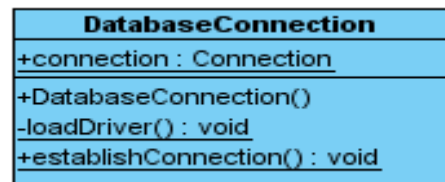


Figure 6.3. DatabaseConnection

### 6.1.2. strider.definedTypes package

This package contains the definition classes that will be used by other packages and classes, namely Season, Constraint, WindDirection, ExperienceLevel and Coordinate classes (Figure 6.4).

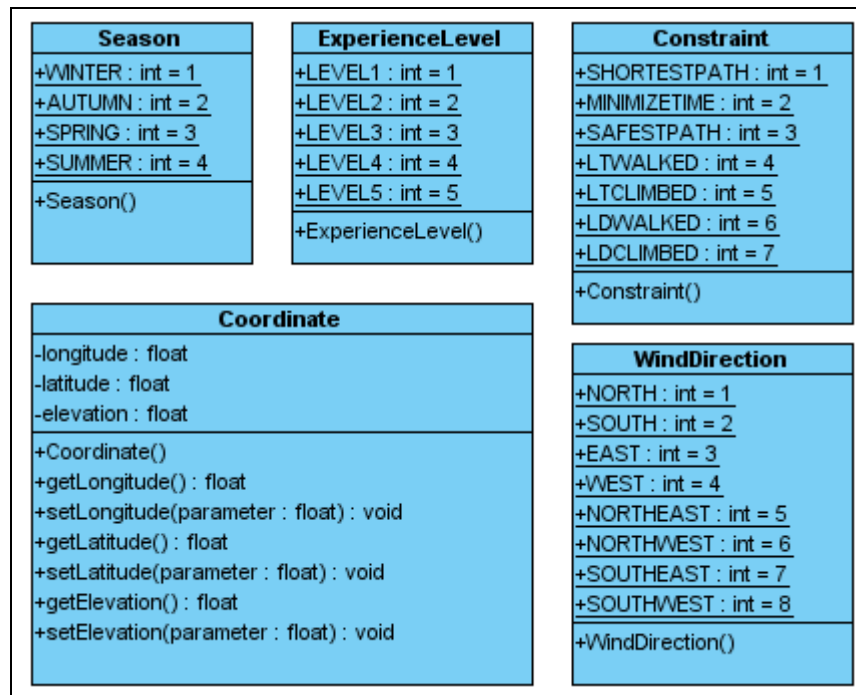


Figure 6.4. *strider.definedTypes Package*

- **Season Class** contains the season information. Season can be WINTER, AUTUMN, SPRING, SUMMER.
- **Constraint Class** contains the project constraint definitions. There are seven types of constraints defined.
- **WindDirection Class** contains the definition of wind direction.
- **ExperienceLevel Class** defines the experience level of a member.
- **Coordinate Class** defines a new type that is consist of longitude, latitude and elevation informations.

### 6.1.3. strider.gis Package

This is the package that extracts data from the map file paths specified by the user. This package is capable of getting features of both raster and vector maps. Below you can find the class diagram of this package. This package contains the class named “GIS” (Figure 6.5).

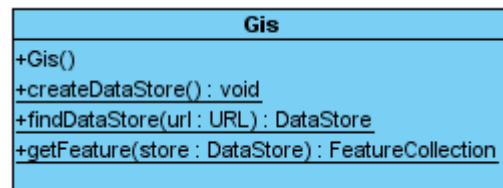


Figure 6.5. strider.gis package

- **GIS Class** is the class that is responsible for geographic information system side operations. The descriptions about methods will be given in the “Methods and Class Interfaces” section.

### 6.1.4. strider.planner Package

This is the package that the plan of the activity is prepared (Figure 6.6). There are three classes in this package: Camping class, Plan Class and Planner class.

The method makePlan() in the Planner class prepares the activity plan. Our algorithm will work in makePlan() function. Since we do not have a well-defined algorithm to prepare plan at this step, we will only give some information about the working principals of our planner algorithm.

Activity plan consists of a route, a camping plan including camping regions and camping duration, the optimized list of equipments that are required for the climbing and an optimized food list. Among these members of the plan, route is the one that dominates the others. So the algorithm should first find an optimized route for the activity meeting the constraints specified by the user.

The algorithm will make decision on paths considering the least experienced, slowest and weakest members of the group as a basis. The paths that require more experience than

the least experienced member of the group will not be chosen. Duration of the activity will be decided according to the speed of slowest member in climbing or walking.

Walking speed of the members will be reduced at nights, at snowy and rainy weathers. At rainy and snowy weathers walking will be preferred climbing, actually climbing will not be an option.

Safest route is the route that there is less or no avalanche and falling rocks risks. Also walking will be taken as safer than climbing. When minimizing time of the activity is a constraint, safest route will be considered as shortest route in duration, because the risky activities like climbing, passing rivers, are time consuming activities even if they are on the shortest path. This is one of the mountaineering knowledge obtained from reliable sources.

Camping is not mandatory. However, if necessary, the duration of the camping will always be minimized. Equipment list will be prepared according to the requirements of the path and will be optimized. Food list will be prepared according to the duration of the activity. The load of the group should be minimized.



Figure 6.6. strider.planner package

- **Camping Class** includes the definition of a camp.
- **Plan Class** includes the definition of a plan.
- **Planner Class** is the class that makes the activity plan.

#### 6.1.5. strider.gui Package

This is the package that handles the user interaction and commands. Inputs are obtained from the user to be used in other packages for planning. Some important features it provides to user can be listed as follows:

- User can manage the member database of the club
- User can see the 3D visualization of the terrain and can mark coordinates on the image for different purposes: start and end point specification, checkpoint specification, avalanche risk specification, falling rock risk specification.
- User can see the simulation of the prepared plan.

This is the largest package in the system. It consists of two packages namely “helpers” and “windows”.

#### 6.1.6. strider.gui.helpers Package

This is the package that contains helper packages for gui windows. It contains “MapFilter Class” (Figure 6.7) and 6 helper packages namely “Member”, “Equipment”, “Visualization”, “Project”, “Simulation” and “Weather”.

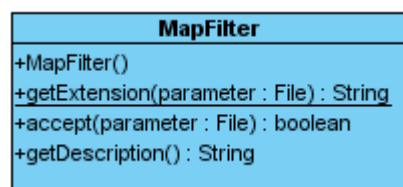


Figure 6.7. MapFilter Class

- **MapFilter Class** is extended from java’s `FileFilter` class and overrides its “accept” and “getDescription” methods. This class is used for browsing maps in the following formats : “dted”, “dem”, “geotiff”, “shp”.

### 6.1.7. strider.gui.helpers.member Package

This is the package that apply the changes and operations to the database of members. It contains member related information and operations. It consists of two classes namely “Member” and “MemberManager” (Figure 6.8).

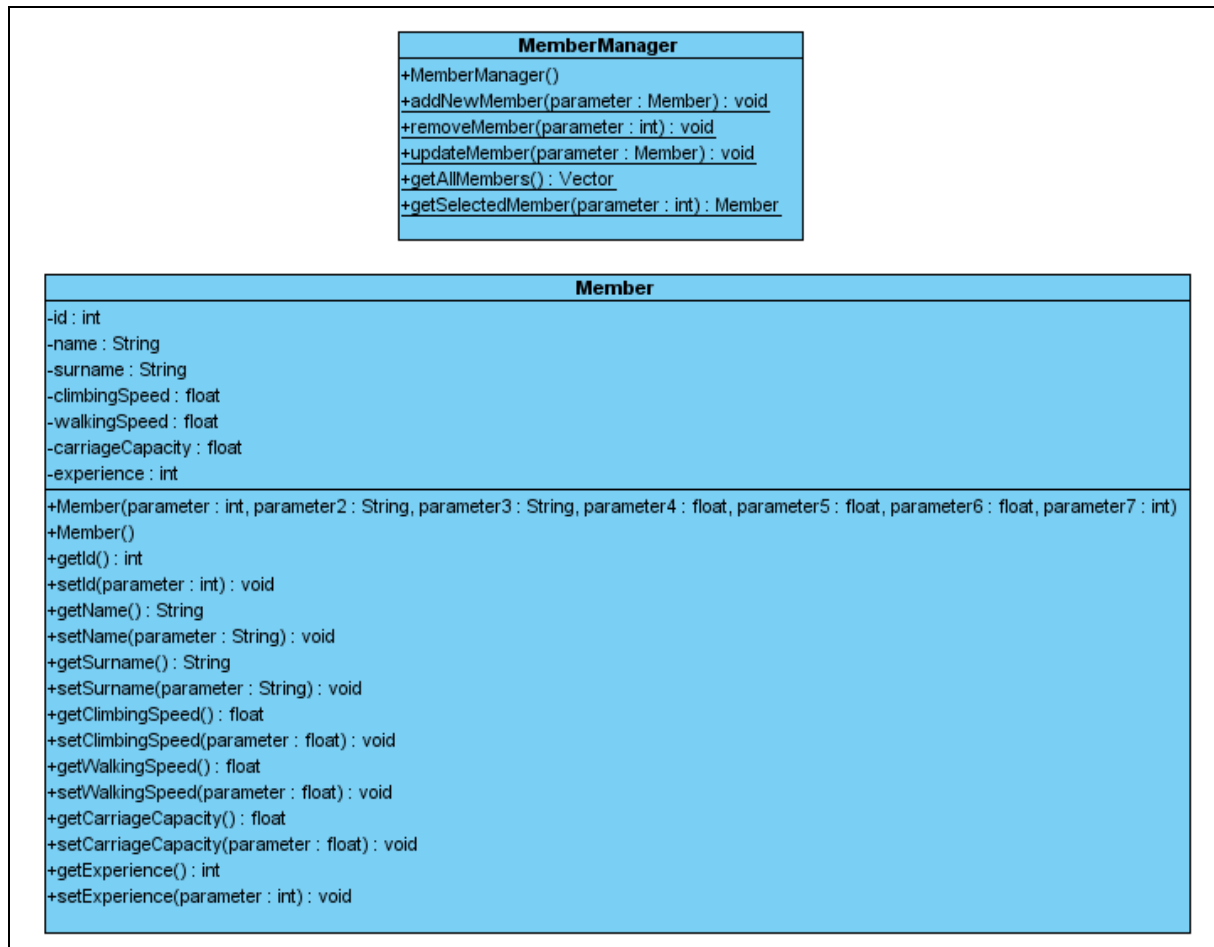


Figure 6.8. strider.gui.helpers.member Package

- **Member Class** is the class containing the attributes of a member and getter, setter methods for those attributes.
- **MemberManager Class** is the class responsible for handling member operations. Since it is the manager class, MemberManager Class has no constructor and all methods within this class are static. This class is the bridge class between the database and the software.

### 6.1.8. strider.gui.helpers.equipment Package

This package contains equipment related information and operations. It consists of two classes namely “Equipment” and “EquipmentManager” (Figure 6.9).

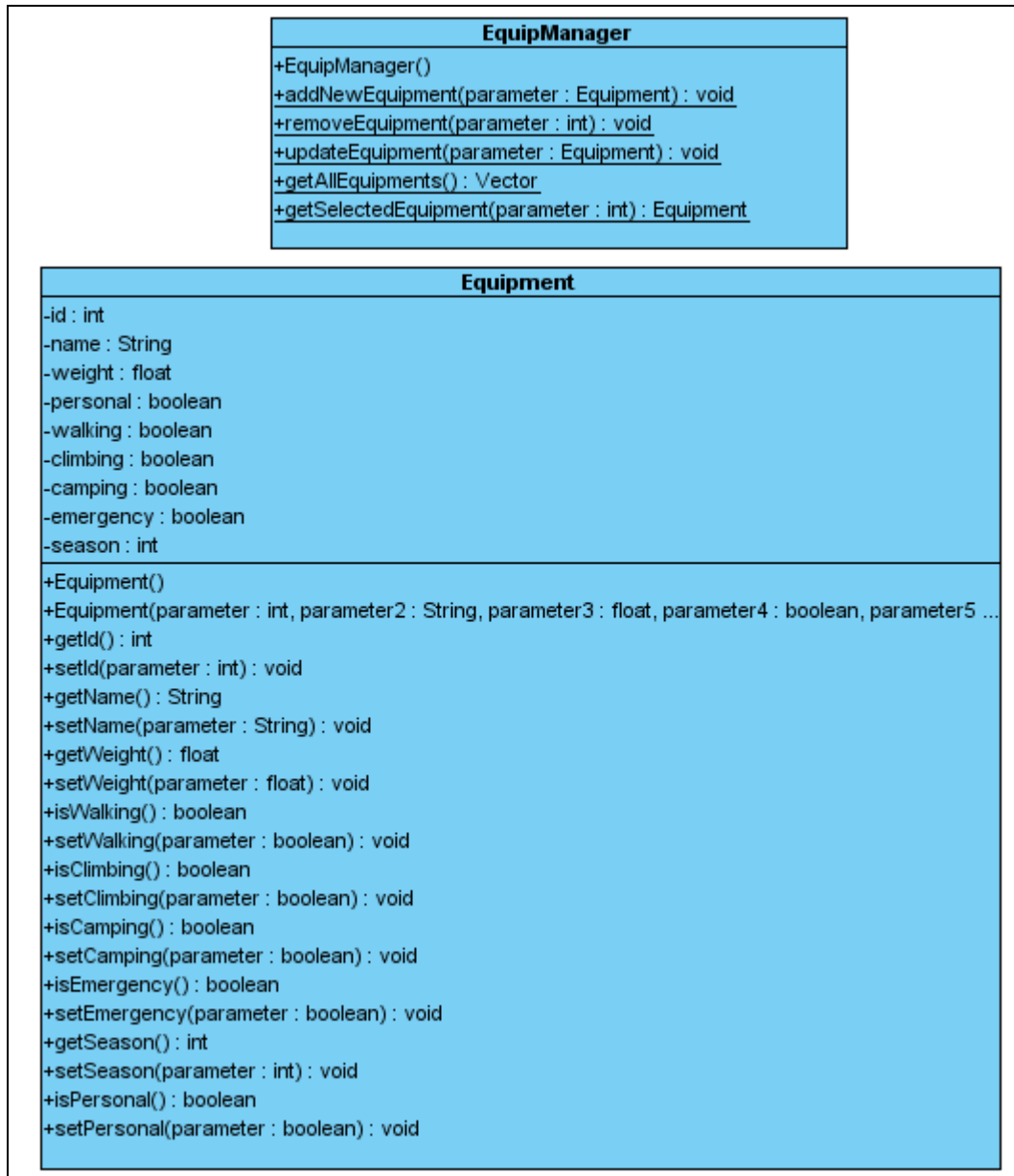


Figure 6.9. strider.gui.helpers.equipment Package

- **Equipment Class** is the class containing the attributes of an equipment and getter, setter methods for those attributes.

- **EquipmentManager Class** is the class responsible for handling equipment operations. Since it is the manager class, EquipmentManager Class has no constructor and all methods within this class are static. This class is the bridge class between the database and the software.

#### 6.1.9. strider.gui.helpers.project Package

This package contains project related information and operations. It consists of two classes namely “Project” and “Project Manager” (Figure 6.10 ).

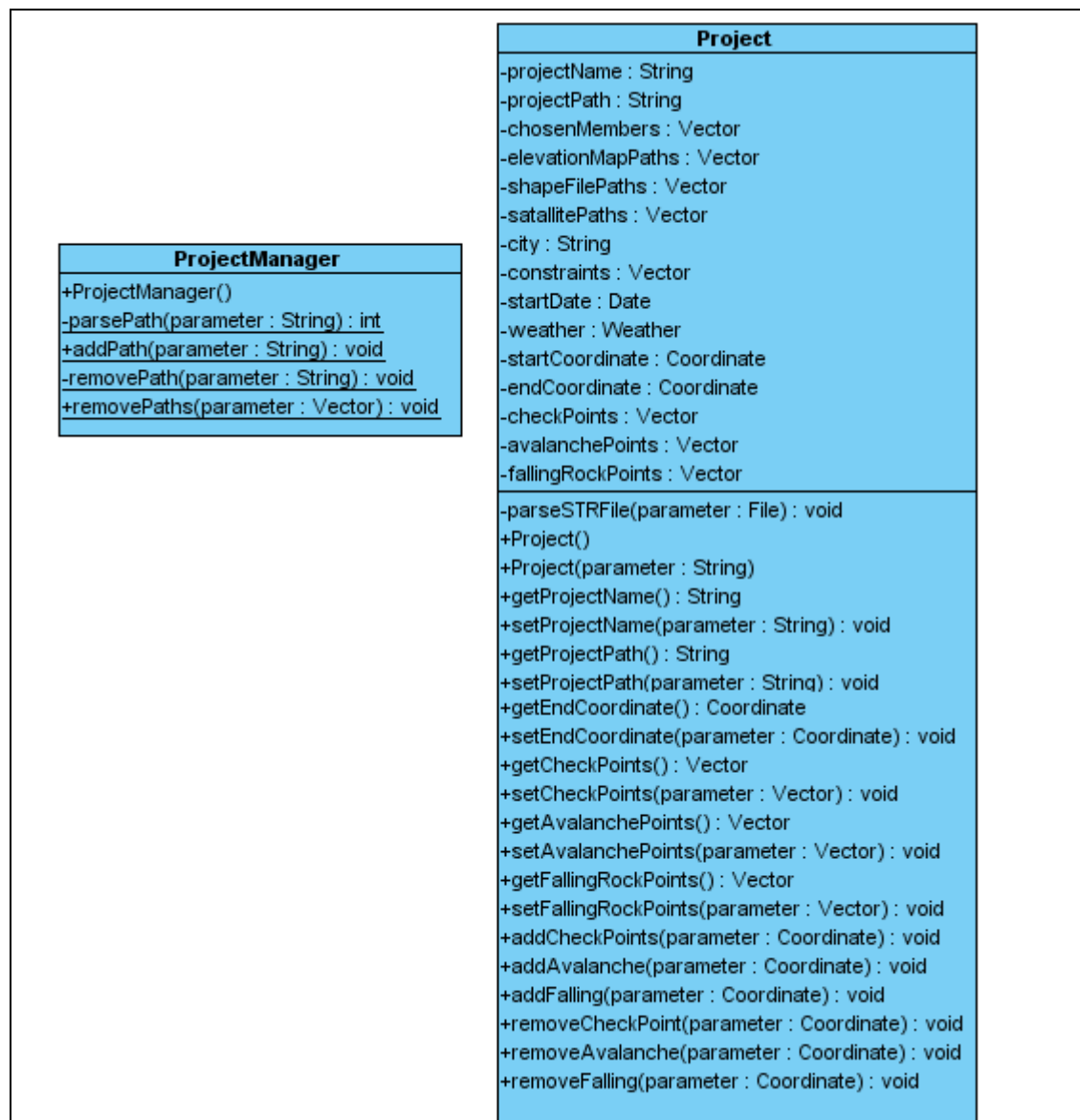


Figure 6.10. strider.gui.helpers.project Package



- **Project Class** is the class containing the attributes of a project and getter, setter methods for those attributes. In addition, it has methods to remove or add various coordinates and a method to parse “srs” file. Some of the getter/setter methods couldn’t be shown in the figure because of lacking space.
- **ProjectManager Class** is the class responsible for handling project operations. Since it is the manager class, ProjectManager Class has no constructor and all methods within this class are static. This class holds methods for adding or removing a map from the project and parsing path according to its extension.

#### 6.1.10. strider.gui.helpers.weather Package

This package contains weather related information and operations. It consists of two classes namely “Weather” and “WeatherManager”.

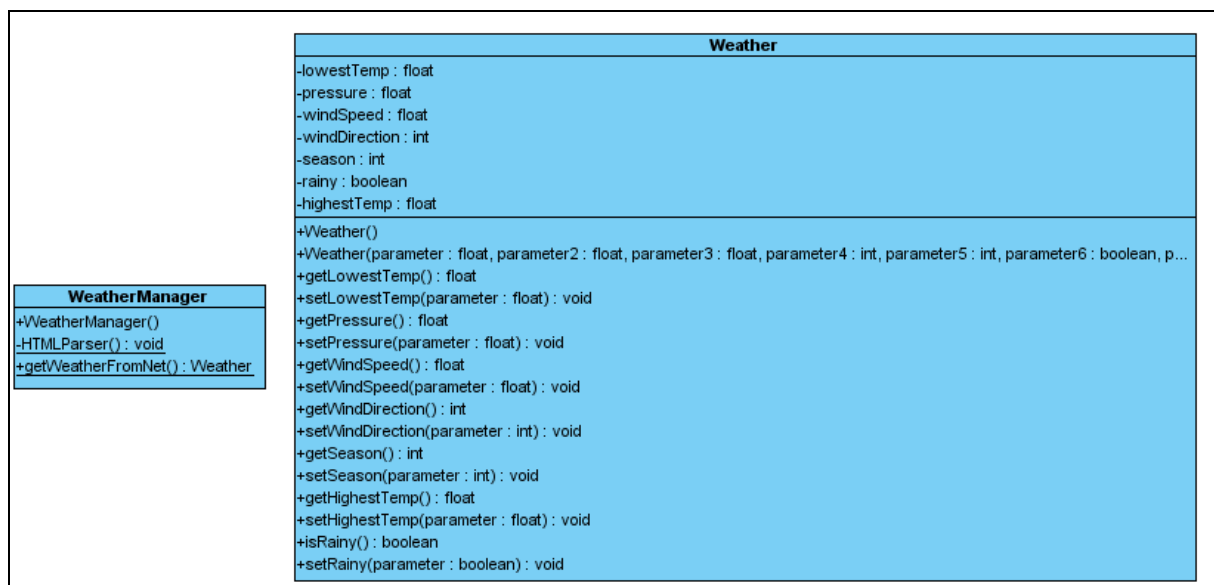


Figure 6.11. *strider.gui.helpers.weather Package*

- **Weather Class** is the class containing the attributes of weather information and getter, setter methods for those attributes.
- **WeatherManager Class** is the class responsible for handling weather operations. Since it is the manager class, WeatherManager Class has no constructor and all

methods within this class are static. This class is basically responsible for getting weather from internet and HTML parsing.

#### 6.1.11. *strider.gui.helpers.visualization Package*

This package contains only “Visualization” class (Figure 6.12).

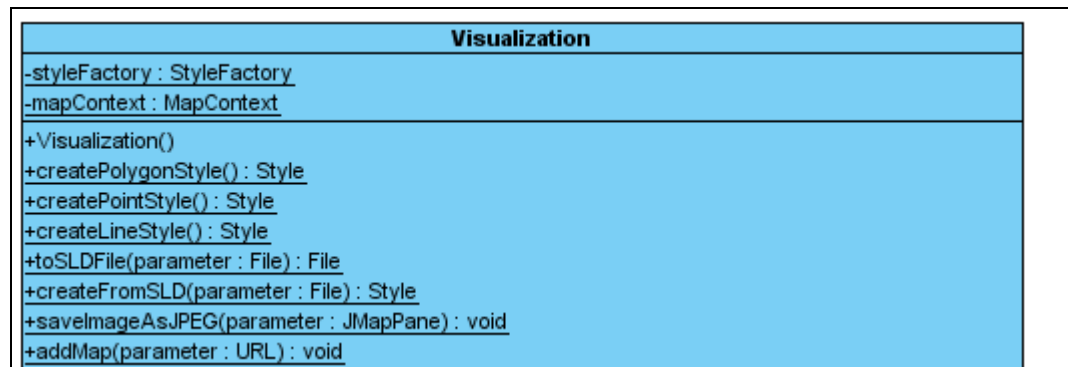


Figure 6.12. *strider.gui.helpers.visualization Package*

- **Visualization Class** is the class responsible for visualizing the terrain. It basically makes use of Geotools for this purpose.

#### 6.1.12. *strider.gui.helpers.simulation Package*

This package contains “Simulation” and “SimulationHelper” class.

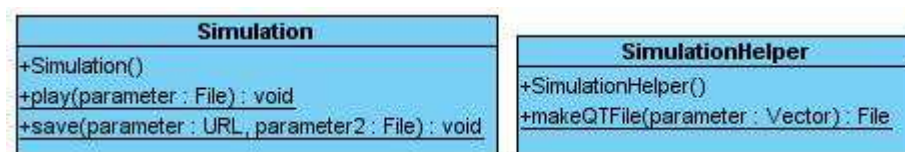


Figure 6.13. *strider.gui.helpers.simulation Package*

- **Simulation Class** is the class for making simulation of the planned route.

### 6.1.13. strider.gui.windows Package

This package is responsible for presenting the gui. The classes correspond the figures of “User Interface Design” part. Details about the methods in the classes will be given in “Class and Method Interfaces” section.

It contains the following classes:

- **MainWindow Class** is the class responsible for showing the “Main Window of Strider”. The class diagram of MainWindow is shown in Figure 6.14. Of course, there are much more attributes in this class, but because of space reasons we show some of them here.

MainWindow
+currentProject : Project -model : DefaultTableModel -aboutMenuItem : JMenuItem -addMapMenuItem : JMenuItem -buttonGroup : ButtonGroup -checkBox1 : JCheckBox -checkBox2 : JCheckBox -checkBox3 : JCheckBox -zoomOutButton : JButton
-showMaps() : void -showMouseEvents() : void -fillTable() : void -deleteRow(parameter : int) : void +MainWindow(parameter : String) -initComponents() : void -getActivityButtonActionPerformed(parameter : ActionEvent) : void -rotateLeftButtonActionPerformed(parameter : ActionEvent) : void -rotateRightButtonActionPerformed(parameter : ActionEvent) : void -openProjectMenuItemActionPerformed(parameter : ActionEvent) : void -newProjectMenuItemActionPerformed(parameter : ActionEvent) : void -removeChosenButtonActionPerformed(parameter : ActionEvent) : void -updateButtonActionPerformed(parameter : ActionEvent) : void -memberOpButtonActionPerformed(parameter : ActionEvent) : void -chooseButtonActionPerformed(parameter : ActionEvent) : void -setWeatherButtonActionPerformed(parameter : ActionEvent) : void -zoomInButtonActionPerformed(parameter : ActionEvent) : void -zoomOutButtonActionPerformed(parameter : ActionEvent) : void -exitButtonActionPerformed(parameter : ActionEvent) : void -exitMenuItemActionPerformed(parameter : ActionEvent) : void -saveProject(parameter : Project, parameter2 : String) : void -saveProjectMenuItemActionPerformed(parameter : ActionEvent) : void -addMapMenuItemActionPerformed(parameter : ActionEvent) : void -removeMapMenuItemActionPerformed(parameter : ActionEvent) : void -helpTopicsMenuItemActionPerformed(parameter : ActionEvent) : void -aboutMenuItemActionPerformed(parameter : ActionEvent) : void -seeWeatherMenuItemActionPerformed(parameter : ActionEvent) : void -equipOpMenuItemActionPerformed(parameter : ActionEvent) : void -saveImageButtonActionPerformed(parameter : ActionEvent) : void -mouseClickedActionPerformed(parameter : MouseEvent) : void -mouseMoveActionPerformed(parameter : MouseEvent) : void

Figure 6.14. *strider.gui.windows.MainWindow Class*

- **AboutWindow Class** is the class responsible for showing the “About Strider Window”. The class diagram of AboutWindow is shown in Figure 6.15.

AboutWindow
-TextArea : JTextArea -closeButton : JButton -scrollPane : JScrollPane
+AboutWindow() -initComponents() : void -closeButtonActionPerformed(parameter :(ActionEvent)) : void

Figure 6.15. *strider.gui.windows.AboutWindow Class*

- **AddNewEquipWindow Class** is the class responsible for showing the “Add Equipment Window”. The class diagram of AddNewEquipWindow is shown in Figure 6.16.

AddNewEquipWindow
-newEquip : Equipment -lastId : int -buttonGroup : ButtonGroup -campingCheckBox : JCheckBox -cancelButton : JButton -climbingCheckBox : JCheckBox -emergencyCheckBox : JCheckBox -equipInfoPanel : JPanel -groupRadioButton : JRadioButton -nameField : JTextField -nameLabel : JLabel -okButton : JButton -personalRadioButton : JRadioButton -seasonPanel : JPanel -summerCheckBox : JCheckBox -usedForPanel : JPanel -walkingCheckBox : JCheckBox -weightField : JTextField -weightLabel : JLabel -winterCheckBox : JCheckBox
+AddNewEquipWindow(parameter : int) -initComponents() : void -okButtonActionPerformed(parameter :(ActionEvent)) : void -cancelButtonActionPerformed(parameter :(ActionEvent)) : void +getNewEquipment() : Equipment +setNewEquipment(parameter : Equipment) : void

Figure 6.16. *strider.gui.windows.AddNewEquipWindow Class*

- **AddNewMemberWindow Class** is the class responsible for showing the “Add New Member Window”. The class diagram of AddNewMemberWindow is shown in Figure 6.17.

AddNewMemberWindow
-newMember : Member -lastId : int -cancelButton : JButton -carriageField : JTextField -carriageLabel : JLabel -climbingField : JTextField -climbingLabel : JLabel -experienceComboBox : JComboBox -experienceLabel : JLabel -memberInfoPanel : JPanel -nameField : JTextField -nameLabel : JLabel -okButton : JButton -surnameField : JTextField -surnameLabel : JLabel -walkingField : JTextField -walkingLabel : JLabel
+AddNewMemberWindow(parameter : int) -initComponents() : void -okButtonActionPerformed(parameter : ActionEvent) : void -cancelButtonActionPerformed(parameter : ActionEvent) : void +getNewMember() : Member +setNewMember(parameter : Member) : void

Figure 6.17. *strider.gui.windows.AddNewMemberWindow Class*

- **EquipOperationsWindow Class** is the class responsible for showing the “Equipment Operations Window”. The class diagram of EquipOperationsWindow is shown in Figure 6.18.

EquipOperationsWindow
~model : DefaultTableModel ~lastId : int -addEquipButton : JButton -allEquipsLabel : JLabel -allEquipsTable : JTable -closeButton : JButton -equipScrollPane : JScrollPane -removeMemberButton : JButton -updateButton : JButton
-fillTable() : void -insertRow(parameter : Equipment) : void -deleteRow(parameter : int) : void -updateRow(parameter : int, parameter2 : Equipment) : void +EquipOperationsWindow() -initComponents() : void -updateButtonActionPerformed(parameter : ActionEvent) : void -addEquipButtonActionPerformed(parameter : ActionEvent) : void -removeMemberButtonActionPerformed(parameter : ActionEvent) : void -closeButtonActionPerformed(parameter : ActionEvent) : void

Figure 6.18. *strider.gui.windows.EquipOperationsWindow Class*

- **GetActivityPlanWindow Class** is the class responsible for showing the “Activity Plan Window”. The class diagram of GetActivityPlanWindow is shown in Figure 6.19.

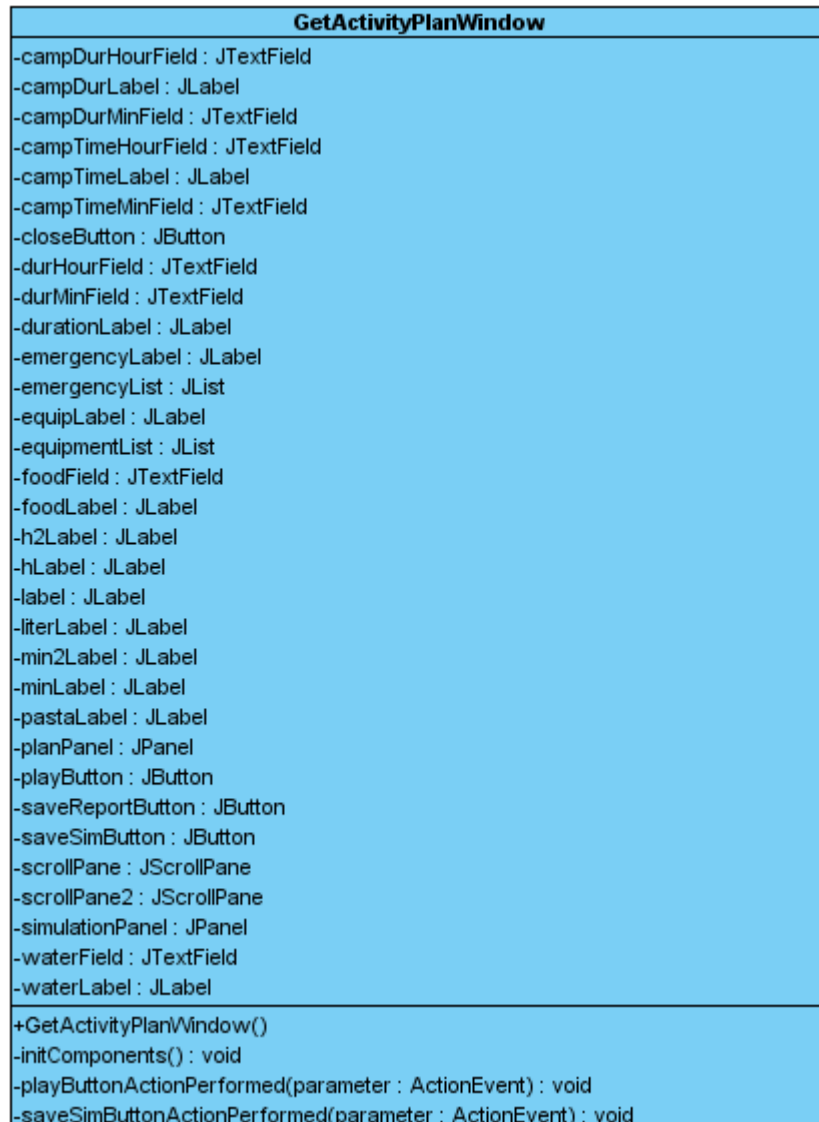


Figure 6.19. *strider.gui.windows.GetActivityPlanWindow Class*

- **RemoveMapWindow Class** is the class responsible for showing the “Remove Map(s) Window”. The class diagram of RemoveMapWindow is shown in Figure 6.20.

RemoveMapWindow
~model : DefaultTableModel -closeButton : JButton -mapsLabel : JLabel -mapsTable : JTable -removeButton : JButton -scrollPane : JScrollPane
-getFileName(parameter : String) : String -fillTable() : void +RemoveMapWindow() -initComponents() : void -closeButtonActionPerformed(parameter :(ActionEvent)) : void -removeButtonActionPerformed(parameter :(ActionEvent)) : void

Figure 6.20. *strider.gui.windows.RemoveMapWindow Class*

- **UnchosenMembersWindow Class** is the class responsible for showing the “Choose Member for Activity Window”. The class diagram of UnchosenMembersWindow is shown in Figure 6.21.

UnchosenMembersWindow
~model : DefaultTableModel -allMembersLabel : JLabel -chooseMemberButton : JButton -closeButton : JButton -membersScrollPane : JScrollPane -unchosenMembersTable : JTable
-fillTable() : void +UnchosenMembersWindow() -initComponents() : void -chooseMemberButtonActionPerformed(parameter :(ActionEvent)) : void -closeButtonActionPerformed(parameter :(ActionEvent)) : void

Figure 6.21. *strider.gui.windows.UnchosenMembersWindow Class*

- **UpdateEquipWindow Class** is the class responsible for showing the “Update Equipment Window”. The class diagram of UpdateEquipWindow is shown in Figure 6.22.



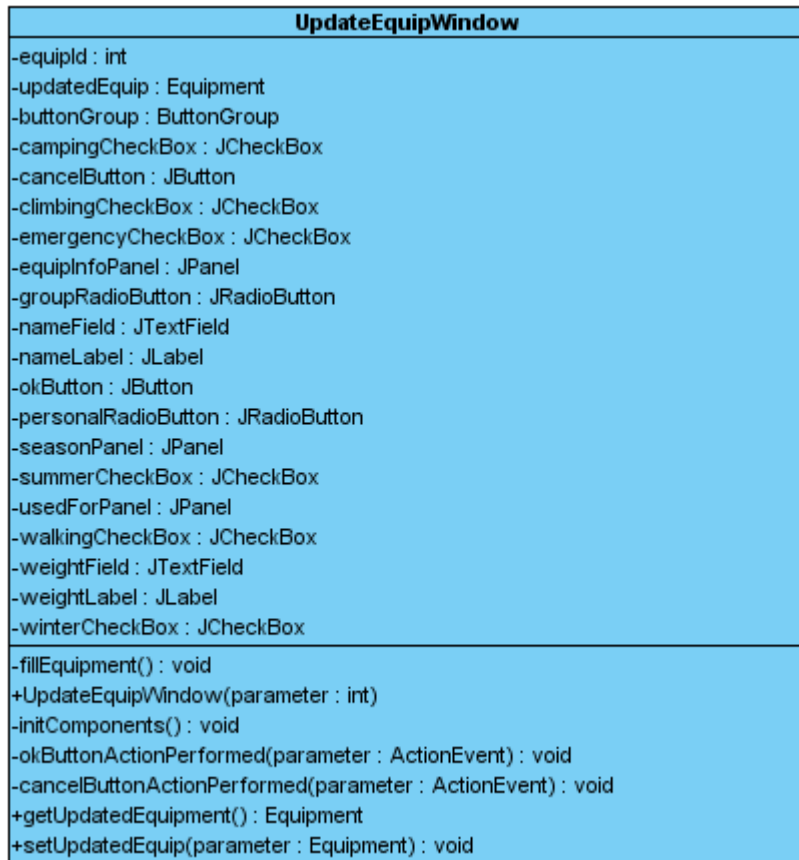


Figure 6.22. *strider.gui.windows.UpdateEquipWindow Class*

- **UpdateMemberWindow Class** is the class responsible for showing the “*Update Member Window*”. The class diagram of UpdateMemberWindow is shown in Figure 6.23.

UpdateMemberWindow
-memberId : int -updatedMember : Member -cancelButton : JButton -carriageField : JTextField -carriageLabel : JLabel -climbingField : JTextField -climbingLabel : JLabel -experienceComboBox : JComboBox -experienceLabel : JLabel -memberInfoPanel : JPanel -nameField : JTextField -nameLabel : JLabel -okButton : JButton -surnameField : JTextField -surnameLabel : JLabel -walkingField : JTextField -walkingLabel : JLabel
-fillMember() : void +UpdateMemberWindow(parameter : int) -initComponents() : void -okButtonActionPerformed(parameter :(ActionEvent)) : void -cancelButtonActionPerformed(parameter :(ActionEvent)) : void +getUpdatedMember() : Member +setUpdatedMember(parameter : Member) : void

Figure 6.23. *strider.gui.windows.UpdateMemberWindow Class*

- **UpdateWeatherWindow Class** is the class responsible for showing the “*Update Weather Window*”. The class diagram of UpdateWeatherWindow is shown in Figure 6.24.

UpdateWeatherWindow
-buttonGroup : ButtonGroup -cancelButton : JButton -highestTempLabel : JLabel -highestTempSpinner : JSpinner -lowestTempLabel : JLabel -lowestTempSpinner : JSpinner -notRainyRadioButton : JRadioButton -pressureLabel : JLabel -pressureSpinner : JSpinner -rainPanel : JPanel -rainyRadioButton : JRadioButton -seasonComboBox : JComboBox -seasonLabel : JLabel -temperaturePanel : JPanel -updateButton : JButton -weatherInfoPanel : JPanel -windDirectionComboBox : JComboBox -windDirectionLabel : JLabel -windPanel : JPanel -windSpeedLabel : JLabel -windSpeedSpinner : JSpinner
-fillWeatherInfo() : void +UpdateWeatherWindow() -initComponents() : void -updateButtonActionPerformed(parameter :(ActionEvent)) : void -cancelButtonActionPerformed(parameter :(ActionEvent)) : void

Figure 6.24. *strider.gui.windows.UpdateWeatherWindow Class*

- **UserOperationsWindow Class** is the class responsible for showing the “*User Operations Window*”. The class diagram of UserOperationsWindow is shown in Figure 6.25.

UserOperationsWindow
~model : DefaultTableModel ~lastId : int -addMemberButton : JButton -allMembersLabel : JLabel -allMembersTable : JTable -closeButton : JButton -membersScrollPane : JScrollPane -removeMemberButton : JButton -updateButton : JButton
-fillTable() : void -insertRow(parameter : Member) : void -deleteRow(parameter : int) : void -updateRow(parameter : int, parameter2 : Member) : void +UserOperationsWindow() -initComponents() : void -updateButtonActionPerformed(parameter : ActionEvent) : void -addMemberButtonActionPerformed(parameter : ActionEvent) : void -removeMemberButtonActionPerformed(parameter : ActionEvent) : void -closeButtonActionPerformed(parameter : ActionEvent) : void

Figure 6.25. *strider.gui.windows.UserOperationsWindow Class*

- **WeatherInfoWindow Class** is the class responsible for showing the “*Weather Information Window*”. The class diagram of WeatherInfoWindow is shown in Figure 6.26.

WeatherInfoWindow
-highestTempField : JTextField -highestTempLabel : JLabel -isRainyField : JTextField -isRainyLabel : JLabel -lowestTempField : JTextField -lowestTempLabel : JLabel -okButton : JButton -pressureField : JTextField -pressureLabel : JLabel -seasonField : JTextField -seasonLabel : JLabel -weatherInfoPanel : JPanel -windDirectionField : JTextField -windDirectionLabel : JLabel -windSpeedField : JTextField -windSpeedLabel : JLabel
-fillWeather() : void +WeatherInfoWindow() -initComponents() : void -okButtonActionPerformed(parameter : ActionEvent) : void

Figure 6.26. *strider.gui.windows.WeatherInfoWindow Class*

## 6.2. Method and Class Interfaces

### 6.2.1. strider package

#### **DatabaseConnection Class:**

*private static void loadDriver():* This method loads the driver. It is a private and static method and called within the “establishConnection” method to establish the connection.

*public static void establishConnection():* This method first loads the driver by calling loadDriver() method, then establishes the database connection. This method is called in the constructor of Main Class to establish the database connection at the beginning.

### 6.2.2. strider.gis package

#### **Gis Class:**

*public static void createDataStore():* This method creates DataStore for different map formats. DataStore is a defined type in Geotools holding the information about the maps. (Something like a database) Adding a map to the project causes this methods to be invoked.

*public static DataStore findDataStore(URL url):* This method finds the data store for a map format. Adding a map to the project causes this methods to be invoked.

*public static FeatureCollection getFeature(DataStore store):* This method is used for getting the features for a specified data store. FeatureCollection is a defined type in Geotools holding features about datastores.

### 6.2.3. strider.planner package

#### **Planner Class:**

*private static Priority makePriority(Weather weather , Season season , WindDirection windDirection, ExperienceLevel experienceLevel, DataStore dataStore, Constraint constraints):* This method determines the priorities up to the weather , terrain , season and team's stamina conditions. It is invoked within the makePlan().

*private static Vector<Coordinate> makeRoute(DataStore datastore, Vector<Coordinate> referencePoints, Priority priorities):* This method calculates the route between start & end coordinates with respect to the priorities and user given constraints. This method is invoked within the makePlan().

*private static Vector<Camping> makeCampPlan(Vector<Coordinate> route, Weather weather):* This method is used for making the camp plan. While preparing the activity plan, this method is invoked.

*private static float calculateWater(Weather weather):* This method calculates the necessary water amount. It is invoked within the makePlan().

*private static float calculateFood(int duration, int teamMembers):* This method calculates the necessary food amount. It is invoked within the makePlan().

*private static Vector<Equipment> calculateEquipment (Vector<Coordinate> route, int members, Season season, Weather weather):* This method determines the necessary equipments. It is invoked within the makePlan().

*public static Plan makePlan():* This method calls makePriority, makeRoute, makeCampPlan, calculateWater, calculateFood and calculateEquipment methods to make the activity plan. When the user hits “ Get Activity Plan” button from GUI , this method is invoked.

#### 6.2.4. strider.gui.helpers.member package

##### **MemberManager Class:**

*public static void addNewMember(Member newMember)* : This method will add the newMember to the Member table of the database.

*public static void removeMember(int memberId):* This method will remove the member with memberId from the database.

*public static void updateMember(Member member):* This method will update the given member in the database.

*public static Vector getAllMembers():* This method will retrieve all the members in the database and puts them in a Member vector and returns.

*public static Member getSelectedMember(int memberId):* This method will retrieve the member with given memberId from the database and returns it.

#### 6.2.5. strider.gui.helpers.equipment package

##### **EquipmentManager Class:**

*public static void addNewEquip(Equipment newEquip)* : This method will add the newEquip to the Equipment table of the database.

*public static void removeEquipment(int equipId):* This method will remove the equipment with equipId from the database.

*public static void updateEquipment(Equipment equip):* This method will update the given equipment in the database.

*public static Vector getAllEquipments():* This method will retrieve all the equipments in the database and puts them in an Equipment vector and returns.

*public static Equipment getSelectedEquipment(int equipId):* This method will retrieve the equipment with given equipId from the database and returns it.

#### 6.2.6. strider.gui.helpers.project package

##### **Project Class:**

*private void parseSRSFile(File file):* This method will be used to parse the srs file and sets the necessary fields accordingly. It will be called within the constructor of Project, so it is private.

##### **ProjectManager Class:**

*private static int parsePath(String path):* This method will parse the path by path's extension and returns its type. If it has a "dtd" or "dem" extension it will return 1, "shp"

extension it will return 2, "geotiff" extension it will return 3. This method will be called from addPath(path) or removePath(path) methods in the ProjectManager class, so it is private.

public static void addPath(String path) : This method will add the given path to the current project. It will first invoke parsePath(path) method, if the returned value is 1, the path will be added to elevationMapPaths list, if it is 2 it will be added to shapeFilePaths list, if it is 3 it will be added to satelliteImagePaths list. This process is necessary since geotools has different procedures to read different map formats.

private static void removePath(String path): This method will remove the specified path from the current project. It will first invoke parsePath(path) method, if the returned value is 1, the path will be removed from elevationMapPaths list, if it is 2 it will be removed from shapeFilePaths list, if it is 3 it will be removed from satelliteImagePaths list.

public static void removePaths(Vector<String> paths): This method will remove all given paths from the project by calling removePath method for each element. This method is public since more than one path can be removed from the project at the same time.

#### 6.2.7. strider.gui.helpers.weather package

##### **WeatherManager Class:**

private static void HTMLParser(): This method will parse the website for weather and retrieve necessary information from it.

public static Weather getWeatherFromNet() : This method will first parse the html for weather and creates a weather object with taken information and returns that object.

#### 6.2.8. strider.gui.helpers.visualization package

##### **Visualization Class:**

public static Style createPolygonStyle(): This method will create polygon style and adds that style to styleFactory (public static attribute of Visualization class).

public static Style createPointStyle(): This method will create point style and adds that style to styleFactory.



*public static Style createLineStyle():* This method will create line style and adds that style to styleFactory.

*public static File toSLDFile(File file):* This method will convert the file to geoTool's styled layer descriptor format and returns it.

*public static Style createFromSLD(File sld):* This method will call geoTools' SLDParser.readXML() method to parse the file. That method returns Style[], and createFromSLD method returns the first element of that array.

*public static Style createStyle(File file, FeatureType schema):* This method will first call toSLDFile(file) and creates a Style from the returned SLD file using createFromSLD(sld) method and returns that style.

*public static void saveImageAsJPEG(JMapPane mapPane):* This method takes a mapPane (one of the GeoTool types) and save that pane as "jpeg" format.

*public static void addMap(URL url):* This method will add the map information taken from url to the mapContext (static attribute of Visualization class) using geoTools' addLayer method.

#### 6.2.9. strider.gui.helpers.simulation package

##### **Simulation Class:**

*public void play(File file ):* This method will get the qt file of the simulation of planned route and play on the graphical user interface.

*public void save(File file, URL path ):* This method will called when user clicks the save button on the graphical user interface and it saves the simulation video as a qt file to the specified destination.

##### **SimulationHelper Class:**

*public File makeQTFile(vector<GridCoverage> grids )* : This method gets terrain data as GridCoverage which is a feature of Geotools . It collects the grids and converts to image format, than uses the Java QuickTime libraries to make a qt file.

#### 6.2.10. strider.gui.windows package

##### **MainWindow Class:**

*private void ShowMaps()*: This method will get the mapContext (public static attribute of Visualization class). Using mapContext, mapPane(static attribute of MainWindow class) will be initialized and the map will be rendered in the visualization panel of the main window.

*private void showMouseEvents()*: This method will retrieve the existing start, end, checkpoints, avalanche and falling rocks points from the current project (currentProject is the public static attribute of MainWindow. It is instantiated in the constructor of MainWindow), and shows different points with different icons on the map.

*private void createPopup(GeneralDirectPosition q)*: This method will creates a popup menu containing startCoordinate, endCoordinate, checkpoint, avalanche, falling rock as menu items at position p. Using geoTools' getMapCoordinate(g) method, the screen coordinates are transformed into real coordinates. Then that coordinate will be added to the project as the chosen items position. And to reflect the change on visualization showMouseEvents() method is called.

*private void fillTable()*: This method is called within the constructor. It fills the chosen member table. Chosen members are not related to database. This method first retrieves the chosen members from current Project then constructs a table with member id, name and surname columns and fills the table accordingly.

*private void deleteRow(int row)*: This method is called within the removeChosenButton's actionPerformed method. It basically deletes the row from the chosen list table. Changes are immediately seen on main window.

*public MainWindow(String path)*: This is the constructor method. It takes a project path as parameter. Within constructor, a database connection is established, currentProject is

initialized with path. Necessary DataStores are created for the specified project and fillTable() is called.

*private void getActivityButtonActionPerformed(ActionEvent evt)* : This is the method called when “Get Activitiy” button is pressed. In this method, the necessary error checks will be done, if any field is not specified by user, the default values will be used for preparing the plan and a confirmation message will shown to the user. If everything is okay, Planner’s makePlan method will be called. makePlan will return the activity plan and, GetActivityPlanWindow is created and returned plan is set in that window.

*private void rotateLeftButtonActionPerformed(ActionEvent evt)* : This method is called when “Rotate Left” button is pressed. It will set the state of mapPane to mapPane.RotateLeft.

*private void rotateRightButtonActionPerformed(ActionEvent evt)* : This method is called when “Rotate Right” button is pressed. It will set the state of mapPane to mapPane.RotateRight.

*private void zoomInButtonActionPerformed(ActionEvent evt)* : This method is called when “Zoom In” button is pressed. It will set the state of mapPane to mapPane.ZoomIn.

*private void zoomOutButtonActionPerformed(ActionEvent evt)* : This method is called when “Zoom Out” button is pressed. It will set the state of mapPane to mapPane.ZoomOut.

*private void openProjectMenuItemActionPerformed(ActionEvent evt)* : This method is called when “Open Project” menu item is pressed. Within this method, first whether to save the existing project will be asked to the user, after behaving accordingly, a browser will be opened with “srs” files filtered. If a project is selected to open, this method will call MainWindow constructor with the newly specified path.

*private void newProjectMenuItemActionPerformed(ActionEvent evt)* : This method is called when “New Project” menu item is pressed. Within this method, first whether to save the existing project will be asked to the user, after behaving accordingly, MainWindow constructor will be called with “default.srs”.

*private void saveProject(Project project, String path):* This method saves the project in specified path.

*private void saveProjectMenuItemActionPerformed(ActionEvent evt):* This method is called when “Save Project” menu item is pressed. Within this method a browser will be opened and saveProject method will be called with the currentProject and selected path.

*private void removeChosenButtonActionPerformed(ActionEvent evt)* : This method is called when “Remove” button is pressed. After confirmation, the method removes the selected chosen member from the current project by calling project’s removeChose(member) method. To reflect those changes on the window, the method calls deleteRow method also.

*private void updateButtonActionPerformed(ActionEvent evt)* : This method is called when “Update” button is pressed. After doing the necessary error checks UpdateMemberWindow will be created and shown to the user.

*private void memberOpActionPerformed(ActionEvent evt):* This method is called when “User Operations” button is pressed. The UserOperationsWindow will be created and shown to the user within this method.

*private void choseButtonActionPerformed(ActionEvent evt):* This method is called when “Choose New” button is pressed. The UnchosenMembersWindow will be created within this method.

*private void setWeatherButtonActionPerformed(ActionEvent evt):* This method is called when user presses “Set Weather Info” button. Within this method, if “Set Manually” option was chosen then UpdateWeatherWindow will be created and shown. If “Get from Internet” option was chosen then first internet connection is checked, then the availability of the information is checked. If everything is okay, the weather information is taken from the internet using WeatherManager class’ methods. If there is any problem, the old information will be used for the weather and warning will shown to the user. Finally, project’s weather information will be set accordingly.

*private void exitButtonActionPerformed(ActionEvent evt)*: This method is called when “Exit” button is pressed. After asking for saving and taking the necessary action the software will be closed.

*private void exitMenuItemActionPerformed(ActionEvent evt)*: This method is called when “Exit” menu item is pressed. The method will be same as the previous one.

*private void addMapMenuItemActionPerformed(ActionEvent evt)*: This method is called when “Add Map” menu item is pressed. Within this method a browser will be opened with filtered map formats (MapFilter class will be used here). Selecting a map and pressing “Ok” will add the specified path to one of the currentProject paths. ProjectManager’s addPath method will be used here. Gis’ createDataStore method will be called to create a datastore for newly added map and showMaps() method will be invoked to update the visualization.

*private void removeMapMenuItemActionPerformed(ActionEvent evt)*: This method is called when “Remove Map (s)” method is pressed. Within this method, RemoveMapWindow will be created and shown to the user.

*private void helpTopicsMenuItemActionPerformed(ActionEvent evt)*: This method is called when “Help Topics” menu item is pressed. It will open the UserManual file for the gui.

*private void aboutMenuItemActionPerformed(ActionEvent evt)* : This method is called when “About Strider” menu item is pressed. Within this method, AboutWindow will be created and shown to the user.

*private void seeWeatherMenuItemActionPerformed(ActionEvent evt)* : This method is called when “See Weather” Menu item is pressed. Within this method, WheatherInfoWindow will be created and shown to the user.

*private void equipOpMenuItemActionPerformed(ActionEvent evt)*: This method is called when “Equipment Operations” menu item is pressed. Within this method, EquipOperationsWindow is created and shown to the user.

*private void saveImageButtonActionPerformed(ActionEvent evt)*: This method is called when “Save Image” button is pressed. saveImageAsJPEG method will be called.

*private void mouseClickedActionPerformed.MouseEvent evt)*: This method is called when mouse is clicked. Within this method, if left button was clicked then createPopup method will be called and a popup menu will be created. If right button was clicked then if the pressed location contains any defined coordinate, that coordinate will be removed from the project.

*private void mouseMoveActionPerformed(MouseEvent evt)* : This method is called as the mouse moves. Within this method the longitude, latitude and elevation of the coordinate on the terrain is set.

#### **AddNewEquipWindow Class:**

*private void okButtonActionPerformed(ActionEvent evt)*: This method is called when “Ok” button is pressed in “New Equipment” window. Within this method, first necessary checks will be done, if everything is okay, then newly defined equipment is added to the database using EquipManager’s addNewEquipment method. Then this window is closed, returning to the “Equipment Operations” window. The newly added equipment should be seen in that window.

*private void cancelButtonActionPerformed(ActionEvent evt)*: If cancel button is pressed in “New Equipment” window, then the window will be closed returning to the “Equipments Operations” window.

#### **AddNewMemberWindow Class:**

*private void okButtonActionPerformed( ActionEvent evt)*: This method is called when “Ok” button is pressed in “New Member” window. Within this method, first necessary checks will be done, if everything is okay, then newly defined member is added to the database using MemberManager’s addNewMember method. Then this window is closed, returning to the “User Operations” window. The newly added member should be seen in that window.

*private void cancelButtonActionPerformed(ActionEvent evt):* If cancel button is pressed in “New Member” window, then the window will be closed returning to the “User Operations” window.

#### **EquipOperationsWindow Class:**

*private void fillTable():* This method is called within the constructor. It fills the equipments table. This method first retrieves the equipments from database using EquipmentManager’s getAllEquipments method, then constructs and fills the table.

*private void deleteRow(int row):* This method is called within the removeEquipButton’s actionPerformed method. It basically removes the row from equipments list table.

*private void updateButtonActionPerformed(ActionEvent evt):* This method is called when “update” button is pressed. Within this method UpdateEquipWindow is created and shown to the user.

*private void addEquipButtonActionPerformed(ActionEvent evt):* This method is called when “Add New Equipment” is pressed. Within this method AddNewEquipWindow is created and shown to the user.

*private void removeEquipButtonActionPerformed(ActionEvent evt):* This method is called when “Remove” button is pressed. Within this method after confirmation, the selected equipment will be removed using EquipManager’s removeEquipment method and deleteRow methods.

*private void closeButtonActionPerformed(ActionEvent evt):* This method is called when “Close” button is pressed in “Equipment Operations” window. It will just close the window, returning to the main window.

#### **UserOperationsWindow Class:**

*private void fillTable():* This method is called within the constructor. It fills the members table. This method first retrieves the members from database using MemberManager’s getAllMembers method, then constructs and fills the table.

*private void deleteRow(int row):* This method is called within removeMemberButton's actionPerformed method. It basically removes the row from members list table.

*private void updateButtonActionPerformed(ActionEvent evt):* This method is called when "update" button is pressed. Within this method UpdateMemberWindow is created and shown to the user.

*private void addMemberButtonActionPerformed(ActionEvent evt):* This method is called when "Add New Member" is pressed. Within this method AddNewMemberWindow is created and shown to the user.

*private void removeMemberButtonActionPerformed(ActionEvent evt):* This method is called when "Remove" button is pressed. Within this method after confirmation, the selected equipment will be removed using MemberManager's removeMember method and deleteRow methods.

*private void closeButtonActionPerformed(ActionEvent evt):* This method is called when "Close" button is pressed in "User Operations" window. It will just close the window, returning to the main window.

#### **GetActivityWindow Class:**

*private void playButtonActionPerformed( ActionEvent evt):* This method is called when "Play" button is pressed in the "Get Activity Plan" window. Within this method, Simulation's play method will be called.

*private void saveSimButtonActionPerformed(ActionEvent evt):* This method is called when "Save" button is pressed in the "Get Activity Plan" window. Within this method, Simulation's save method will be called.

*private void saveReportButtonActionPerformed(ActionEvent evt):* This method is called when "Save Report" button is pressed. Within the method, the activity plan report will be saved in a defined location.

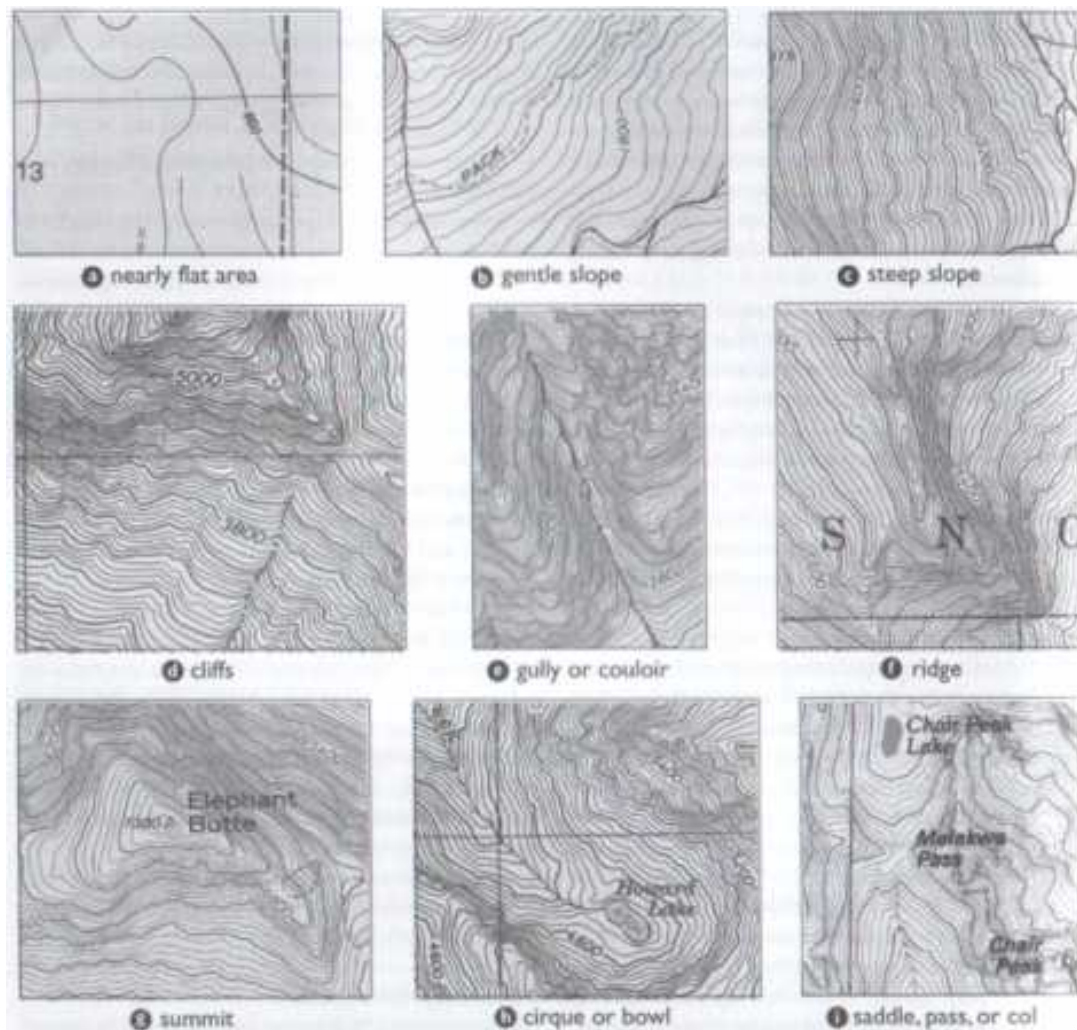


*private void closeButtonActionPerformed(ActionEvent evt):* This method is called when “Close” button is pressed. It closes the project.

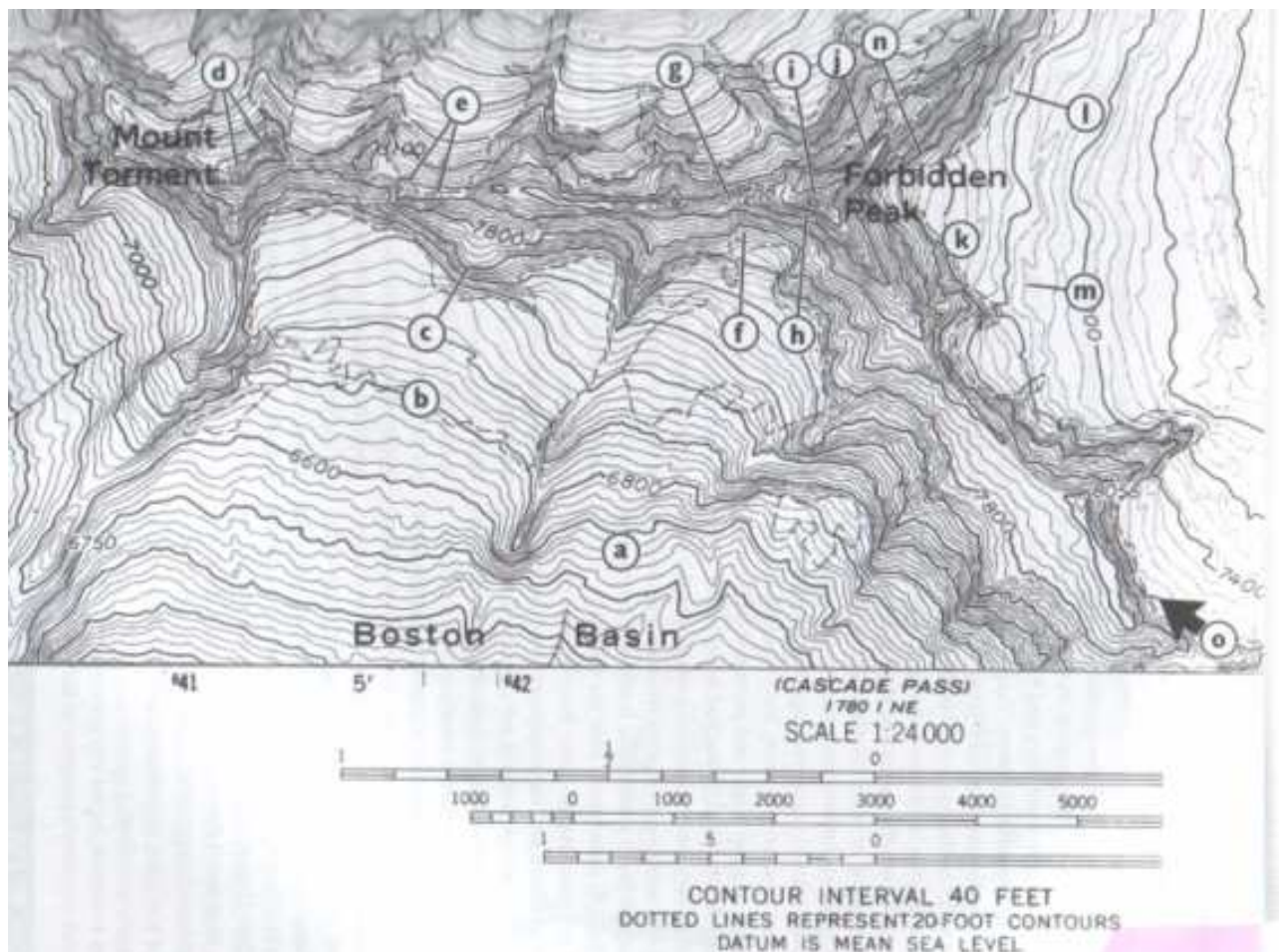
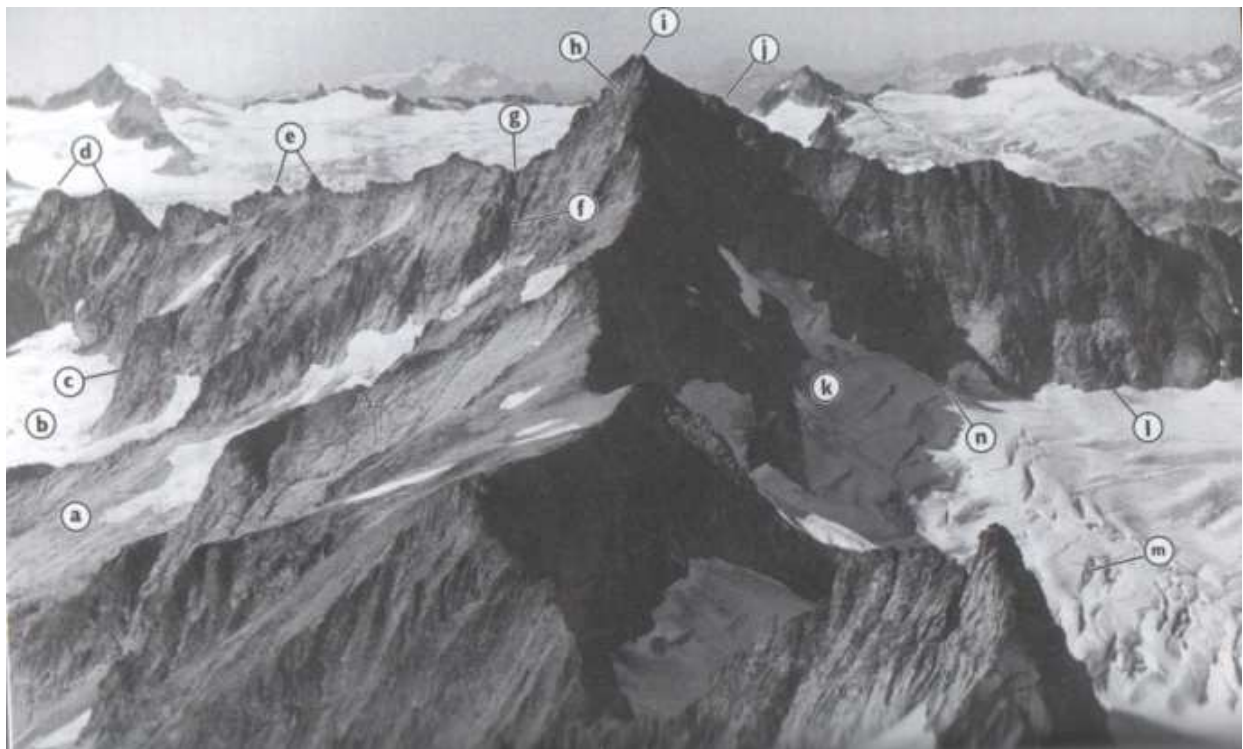
## 6.3. Design of Expedition Plan

### 6.3.1. Considerations on maps

The maps which are given from user are considered with respect to some rules. Here is some information on how maps are commented with respect to data which are got from map formats.



*Basic topographic features on map.*



Key of map;

- a. Basin: moderate slope, camp spots
- b. Snow or ice line: dashed line ends on cliffs, rock
- c. Buttress: change in features of wall may provide approach to ridge
- d. Twin summits
- e. Gendarmes, aiguilles, or pinnacles
- f. Gully or couloir
- g. Saddle, pass, or col
- h. Rock face
- i. Summit: highest point on map
- j. Ridge or arete
- k. East slope: note shadows and ice accumulation
- l. Moat
- m. Crevasses: indicated by irregular contours, not smooth as near buttress, c, above
- n. Bergschrund: not seen on map but possibility inferred when rock and snow are steep
- o. Photo taken from above this spot, looking in direction of arrow

### 6.3.2. Considerations on route planning

#### **Meanings of Levels for climbing;**

Level 1; Hiking.

Level 2; Scrambling, with possible occasional use of the hands. A rope might be carried.

Level 3; Climbing, often with exposure. A rope is often used. Typically, natural protection can be easily found.

Level 4; Where rock climbing begins in earnest. A fall on Level-4 climbing could be fatal. Climbing involves the use of a rope, belaying, and protection ( natural or artificial ) to protect the leader from a long fall.

Level 5; The realm of true experts; demands much training and natural ability and, often, repeated working of a route.

#### **Risk Factor of a climbing**

Risk = emphasis X probability X time X weather conditions (r)

Emphasis; indicates the level of emphasis when an accident occurs. Emphasis is computed by direct proportion with level of climbing.

Probability; indicates the level of probability of accident. As the level of climbing is higher, the probability of accident will increase. So, probability is computed by direct proportion with level of climbing.

Time; indicates “the distance of climbing stage” / “the minimum speed of group ( means the speed of slowest person in the group )”.

Weather conditions; indicates “high temperature” + “low temperature” + “snow” + “wind speed”

The risk count of temperature is;

- If temperature is higher than 25 C or lower than 5 C , it is considered as a risk factor.
- The risk count increases as the temperature increases. 1 C increase in temperature from 25 C increments the risk count by one.
- The risk count increases as the temperature increases. 1 C decrease in temperature from 5 C increments the risk count by one.

The risk count of wind speed is;

- If wind speed is higher than 40 km/h, it is considered as a risk factor.
- The risk count increases as the wind speed increases. 1 km/h decrease in wind speed from 40 km/h increments the risk count by one.

Note: If there is rain, the climbing can't be done.

The affect of risk factor on climbing of a stage.

The climbing of a stage can't be done, when risk factor is higher than below numbers with respect to level of group ( means minimum level climber's level in group )

- Level 2 ; 500 r
- Level 3; 1,500 r
- Level 4; 10,000 r
- Level 5; 50,000 r

Note: For level 1, there can't be any climbing activity.

### 6.3.3. Algorithmic design for route

#### **Ant Colony Optimization [8]**

We decided to take Ant Colony Optimization (ACO) as our starting point for our path-finding algorithm. There are lots of deterministic path-finding graph algorithms, but nature of our problem tends to NP-hardness because of the size of maps & images we have to process, we have to do optimization. There are lots of optimization algorithms, we decided to try ACO first since path-finding in a terrain is strongly connected to the “natural life” of ants.

Ant colony optimization (ACO) is a population-based meta-heuristic that can be used to find approximate solutions to difficult optimization problems.

In ACO, a set of software agents called artificial ants search for good solutions to a given optimization problem. To apply ACO, the optimization problem is transformed into the problem of finding the best path on a weighted graph. The artificial ants (hereafter ants) incrementally build solutions by moving on the graph. The solution construction process is stochastic and is biased by a pheromone model, that is, a set of parameters associated with graph components (either nodes or edges) whose values are modified at runtime by the ants.

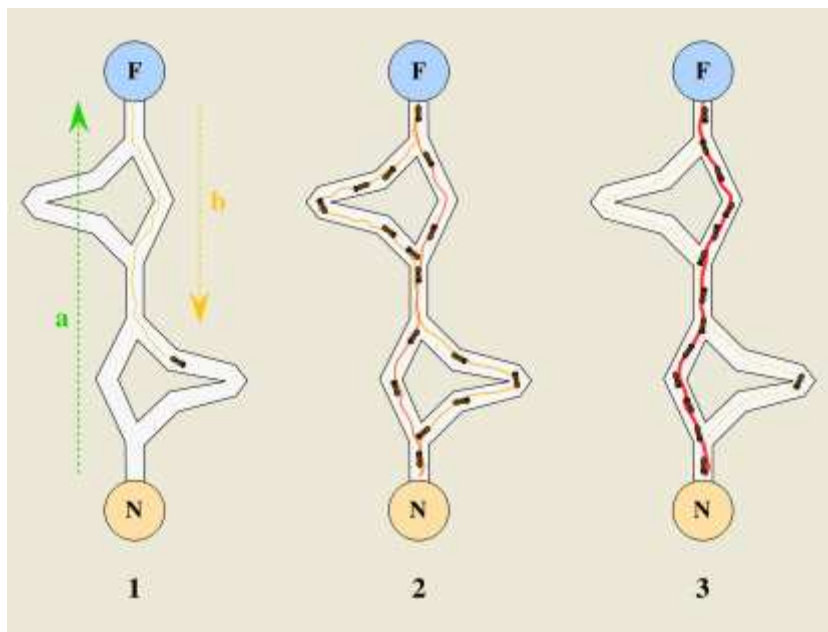
#### **Overview**

In the real world, ants (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food.

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over faster, and thus the pheromone density remains high as it is laid on the path as fast as it can evaporate. Pheromone evaporation has also the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of

the solution space would be constrained.

Thus, when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve.



The original idea comes from observing the exploitation of food resources among ants, in which ants' individually limited cognitive abilities have collectively been able to find the shortest path between a food source and the nest.

1. The first ant finds the food source (F), via any way (a), then returns to the nest (N), leaving behind a trail pheromone (b)
2. Ants indiscriminately follow four possible ways, but the strengthening of the runway makes it more attractive the shortest route.
3. Ants take the shortest route, long portions of other ways lose their trail pheromones.

In a series of experiments on a colony of ants with a choice between two unequal length paths leading to a source of food, biologists have observed that ants tended to use the shortest route. A model explaining this behavior is as follows:

1. An ant (called "blitz") runs more or less at random around the colony;



2. If it discovers a food source, it returns more or less directly to the nest, leaving in its path a trail of pheromone;
3. These pheromones are attractive, nearby ants will be inclined to follow, more or less directly, the track;
4. Returning to the colony, these ants will strengthen the route;
5. If two routes are possible to reach the same food source, the shorter one will be, in the same time, traveled by more ants than the long route will strengthen the route;
6. The short route will be increasingly enhanced, and therefore become more attractive;
7. The long route will eventually disappear, pheromones are volatile;
8. Eventually, all the ants have determined and therefore "chosen" the shortest route.

Ants use the environment as a medium of communication. They exchange information indirectly by depositing pheromones, all detailing the status of their "work". The information exchanged has a local scope, only an ant located where the pheromones were left has a notion of them. This system is called "Stigmergy" and occurs in many social animal societies (it has been studied in the case of the construction of pillars in the nests of termites). The mechanism to solve a problem too complex to be addressed by single ants is a good example of a self-organized system. This system is based on positive feedback (the deposit of pheromone attracts other ants that will strengthen it themselves) and negative (dissipation of the route by evaporation prevents the system from thrashing). Theoretically, if the quantity of pheromone remained the same over time on all edges, no route would be chosen. However, because of feedback, a slight variation on an edge will be amplified and thus allow the choice of an edge. The algorithm will move from an unstable state in which no edge is stronger than another, to a stable state where the route is composed of the strongest edges.

#### Formal Definition of a Combinatorial Optimization Problem [9]

The first step for the application of ACO to a combinatorial optimization problem (COP)

consists in defining a model of the COP as a triplet  $(S, \Omega, f)$ , where:

- $S$  is a search space defined over a finite set of discrete decision variables;

- $\Omega$  is a set of constraints among the variables; and
- $f : S \rightarrow R_0^+$  is an objective function to be minimized (as maximizing over  $f$  is the same as minimizing over  $-f$ , every COP can be described as a minimization problem).

The search space  $S$  is defined as follows. A set of discrete variables  $X_i, i=1, \dots, n$ , with values  $v_i^j \in D_i = \{v_i^1, \dots, v_i^{|D_i|}\}$ , is given. Elements of  $S$  are full assignments, that is, assignments in which each variable  $X_i$  has a value  $v_i^j$  assigned from its domain  $D_i$ . The set of feasible solutions  $S_\Omega$  is given by the elements of  $S$  that satisfy all the constraints in the set  $\Omega$ .

A solution  $s^* \in S_\Omega$  is called a global optimum if and only if:  $f(s^*) \leq f(s) \forall s \in S_\Omega$ . The set of all globally optimal solutions is denoted by  $S_\Omega^* \subseteq S_\Omega$ . Solving a COP requires finding at least one  $s^* \in S_\Omega^*$ .

#### The Ant Colony Optimization Metaheuristic [8]

In ACO, artificial ants build a solution to a combinatorial optimization problem by traversing a fully connected construction graph, defined as follows. First, each instantiated decision variable  $X_i = v_i^j$  is called a *solution component* and denoted by  $c_{ij}$ . The set of all possible solution components is denoted by  $C$ . Then the construction graph  $G_C(V, E)$  is defined by associating the components  $C$  either with the set of vertices  $V$  or with the set of edges  $E$ .

A pheromone trail value  $\tau_{ij}$  is associated with each component  $c_{ij}$ . (Note that pheromone values are in general a function of the algorithm's iteration  $t : \tau_{ij} = \tau_{ij}(t)$ .) Pheromone values allow the probability distribution of different components of the solution



to be modeled. Pheromone values are used and updated by the ACO algorithm during the search.

The ants move from vertex to vertex along the edges of the construction graph exploiting information provided by the pheromone values and in this way incrementally building a solution. Additionally, the ants deposit a certain amount of pheromone on the components, that is, either on the vertices or on the edges that they traverse. The amount  $\Delta\tau$  of pheromone deposited may depend on the quality of the solution found. Subsequent ants utilize the pheromone information as a guide towards more promising regions of the search space.

The ACO meta-heuristic is:

Set parameters, initialize pheromone trails

SCHEDULE\_ACTIVITIES

ConstructAntSolutions

DaemonActions {optional}

UpdatePheromones

END\_SCHEDULE\_ACTIVITIES

The meta-heuristic consists of an initialization step and of three algorithmic components whose activation is regulated by the Schedule\_Activities construct. This construct is repeated until a termination criterion is met. Typical criteria are a maximum number of iterations or a maximum CPU time.

The Schedule\_Activities construct does not specify how the three algorithmic components are scheduled and synchronized. In most applications of ACO to NP-hard problems however, the three algorithmic components undergo a loop that consists in (i) the construction of solutions by all ants, (ii) the (optional) improvement of these solution via the use of a local search algorithm, and (iii) the update of the pheromones. These three components are now explained in more details.

### ConstructAntSolutions

A set of  $m$  artificial ants construct solutions from elements of a finite set of available solution components  $C = \{c_{ij}\}, i = 1, \dots, n, j = 1, \dots, |D_i|$ . A solution construction starts with an empty partial solution  $s^p = \emptyset$ . Then, at each construction step, the current partial solution is extended by adding a feasible solution component from the set of feasible neighbors  $N(s^p) \subseteq C$ . The process of constructing solutions can be regarded as a path on the construction graph  $G_C(V, E)$ . The allowed paths in  $G_C$  are implicitly defined by the solution construction mechanism that defines the set with respect to a partial solution  $s^p$ .

The choice of a solution component from  $N(s^p)$  is done probabilistically at each construction step. The exact rules for the probabilistic choice of solution components vary across different ACO variants. The best known rule is the one of ant system (AS)

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{c_{kl} \in N(s^p)} \tau_{kl}^\alpha \cdot \eta_{kl}^\beta}, \forall c_{ij} \in N(s^p),$$

where  $\tau_{ij}$  and  $\eta_{ij}$  are respectively the pheromone value and the heuristic value associated with the component  $c_{ij}$ . Furthermore,  $\alpha$  and  $\beta$  are positive real parameters whose values determine the relative importance of pheromone versus heuristic information.

### Daemon Actions

Once solutions have been constructed, and before updating the pheromone values, often some problem specific actions may be required. These are often called *daemon actions*, and can be used to implement problem specific and/or centralized actions, which cannot be

performed by single ants. The most used daemon action consists in the application of local search to the constructed solutions: the locally optimized solutions are then used to decide which pheromone values to update.

### Update Pheromones

The aim of the pheromone update is to increase the pheromone values associated with good solutions, and to decrease those that are associated with bad ones. Usually, this is achieved (i) by decreasing all the pheromone values through *pheromone evaporation*, and (ii) by increasing the pheromone levels associated with a chosen set of good solutions  $S_{upd}$ :

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \rho \cdot \sum_{s \in S_{upd} | c_{ij} \in s} F(s),$$

Where  $S_{upd}$  is the set of solutions that are used for the update,  $\rho \in (0, 1]$  is a parameter called evaporation rate, and  $F : S \rightarrow R_0^+$  is a function such that

$$f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in S.$$

$F(\cdot)$  is commonly called the *fitness function*.

Pheromone evaporation implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Different ACO algorithms, for example ant colony system (ACS) or MAX-MIN ant system (MMAS), differ in the way they update the pheromone.

Instantiations of the update rule given above are obtained by different specification of  $S_{upd}$ , which in many cases is a subset of  $S_{iter} \cup \{s_{bs}\}$ , where  $S_{iter}$  is the set of solutions that were constructed in the current iteration, and  $s_{bs}$  is the *best-so-far* solution, that is, the best solution found since the first algorithm iteration. A well-known example is the AS-update

rule, that is, the update rule of ant system:  $S_{upd} \leftarrow S_{iter}$ .

An example of a pheromone update rule that is more often used in practice is the IB-update rule (where IB stands for *iteration-best*):

$$S_{upd} \leftarrow \arg \max_{s \in S_{iter}} F(s).$$

The IB-update rule introduces a much stronger bias towards the good solutions found than the AS-update rule. Although this increases the speed with which good solutions are found, it also increases the probability of premature convergence. An even stronger bias is introduced by the BS-update rule, where BS refers to the use of the best-so-far solution  $S_{bs}$ . In this case,  $S_{upd}$  is set to  $\{S_{bs}\}$ . In practice, ACO algorithms that use variations of the IB-update or the BS-update rules and that additionally include mechanisms to avoid premature convergence, achieve better results than those that use the AS-update rule.

### Pseudo codes

#### *Finding the Route*

Here's the overall idea of application of ACO to our problem: We divide our map into squares (number of squares are calculated according to the map's size & scale), these squares represent the “general” properties of their nearby land. Since we cannot make nodes from points (which are infinitely many), we decided to make a weighted graph from the map where the midpoints of the squares are the nodes, edges represent two half squares between two nodes and the weights are determined according to the user input (i.e. priorities). If the user states reference points, then we also take these points into account. We also calculate the number of ants to be released considering the map's size & scale based upon the ACO's performance statistics.

Here's the pseudocode for finding the route between two nodes:

FindRoute(Graph, start, end, Nturns, MAX\_TURNS, ants)

1. Initialize

```

2. turn = 0, turnsRemaining = Nturns + 1

3. Loop

4. Release a new set of ants from the starting point

5     Loop

6         turn = turn + 1

7         turnsRemaining = turnsRemaining -1

8         For each ant 'a' in the current set

9             If ant 'a' does not reach to target point

10                Move to the next grid point using random propositional rule

11            Else

12                Ant 'a' stops exploring

13     Until (turnsRemaining = 0)

14     Apply the global pheromone update rule using ants that reached to the target point

15     Update optimal path best so far

16     Remove the current set of ants from the civilization.

17.     turnsRemaining = Nturns + 1

18. Until (turn <= MAX_TURNS)

```

Ants perform a complete tour (in our case tour is defined as travelling from start point to the target point) by choosing the nodes according to a probabilistic state transition rule (random-proportional rule) which selects neighboring nodes that are closest to the target node and have a high amount pheromone. Once all ants have completed certain number of turns (Nturns) a global pheromone updating rule (global updating rule, for short)

is applied; a fraction of the pheromone evaporates on all edges (edges that are not refreshed become less desirable); each ant who were able to finish a complete tour, deposits an amount of pheromone on edges which belong to its tour in proportion to how short its tour was (in other words, edges which belong to many short tours are the edges which receive the greater amount of pheromone). After the global updating, current set of ants removed from the civilization, and another set of ants starts from the start point to explore the target point. The process is iterated until the number of turns reach to the maximum number of turns (MAX\_TURNS).

Note that, we set the parameter Nturns such that, most of the ants in the initial set were able to reach the target point.

Here's our random-proportional rule (probabilistic state transition rule) : (gives the probability with which ant k in node (r) chooses to the node (s) )

$$p_k(r,s) = \frac{[\tau(r,s)] \cdot [\eta(r,s)]^\beta}{\sum_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^\beta} \quad \text{if } s \in J_k(r) \\ 0 \quad \text{otherwise}$$

where  $\tau$  is the pheromone,  $\eta = 1/\delta$  is the inverse of the distance ( $\delta$ ) from the point s to the target point,  $J_k(r)$  is the set of neighbor points of r that remain to be visited by ant k positioned on the point r (to make the solution feasible), and  $\beta$  is a parameter which determines the relative importance of pheromones versus distance ( $\beta > 0$ ).

In ant system, the global updating rule is implemented as follows: Ants that were able to complete their tour within the number of allocated turns (Nturns ) allow to update pheromone levels of their visited edges according to

$$\tau(r,s) \leftarrow (1 - \rho) \cdot \tau(r,s) + \sum_k \Delta \tau_k(r,s)$$

where

$$\Delta \tau_k(r, s) = \begin{cases} 1 / L_k & , \text{ if } (r, s) \in \text{tour done by ant } k \text{ calculated.} \\ 0, & \text{otherwise} \end{cases}$$

$0 < \rho < 1$  is a pheromone decay parameter,  $L_k$  is the length of the tour performed by ant  $k$ , and  $m$  is the number of ants that were able to complete tour within the stipulated turns  $N_{\text{turns}}$ .

Since we may have reference points, we calculate the shortest paths between adjacent reference points and then we put together this “small” route to make our route. Here we take start & end point as reference points as well.

FindRoute (MapGraph, referencePoints):

1. smallRoute = null , route = null , points = # of reference points , referencePoint = start
2. Loop
3.     smallRoute = findRoute(MapGraph, referencePoint , next ( referencePoint ) , Nturns , NMAX)
4.     route += smallRoute
5.     points -= 1;
6.     referencePoint = next(referencePoint in referencePoints)
7. end if points == 1

Now we have to construct a weighted graph from the map, namely we have to form vertices and assign weights to the edges; we first divide our map into squares. We decided to make 1 km x 1 km = 1 kilometer squares to be represented by one square. Then we scale this value with map's scale & size to calculate how many squares there will be. The middle points of the squares will be nodes and the area, namely adjacent two half squares between two nodes will be the edges. Also we take user's reference points into account; while

dividing the map, the user reference points must represent a node. Now we use weather, terrain, climbing team's performance and other issues to assign priorities, then combine these priorities and risk factors with the “edge area” to assign weights to the edges. Here is how we form a weighted graph from a map:

1. dividedMap = DivideMapIntoSquares(graph , referencePoints, mapScale, mapSize)
2. unweightedGraph = makeEdges(dividedMap)
3. weightedGraph = assignWeightsToEdges(unweightedGraph , priorities , risks)
4. return weightedGraph

#### 6.3.4. Design of camping plan

The camping planning is up to the conditions such as duration of the activity, weather & terrain conditions. We plan the camping points after making the route and estimating the duration of the climbing & walking without camping. Here's the algorithm assuming we have the route and the expected duration:

1. numberOfCampPoints = expectedDuration ; daylight = terrain's duration of daylight
2. if numberOfCampPoints < 1 ; return  
referencePt = startPt; newCampLoc = null , campLocs = null , campDuration = 0
4. Loop
5. numberOfCampPoints -= 1
6. referencePt = newCampLoc
7. newCampLoc = referencePt + the distance travelled in ( numberOfCampPoints \* 24 + daylight) hours
8. campDuration = (24 - daylight ) + (if it rains , snows or “hard” windy : 24 )
9. Add (newCampLoc , campDuration) to campLocs return if numberOfCampPoints == 0

**Camp planning rules:**



- Camping is planned with respect to duration of expedition.
- Camping duration is planned with respect to user constraints, day or night durations of season, weather conditions and terrain conditions.
- Camping places should be land or rock.
- Camping places shouldn't be meadowy places. If it is mandatory, there isn't be stayed more than one or two nights. [10]

#### 6.3.5. Design of food&water planning [11]

Water is considered as an important parameter. It's amount is recommended for only first a few days of expedition, because of difficulty of carrying. For other days, climbers should find their waters in nature.

Food should be carbonhydrate based because of easy cooking and getting enery with using less oxygen. There is less oxygen at higher places. Food should also be solid food because of easy carrying.

Here we use the general conventions:

Amount of Food needed: 1.5 kg s of carbohydrate per day x # of days per person

Amount of Water needed:

if the temperature > 35 centigrade, 5 liters x 2 days per person

else 2.5 liters x 2 days per person

#### 6.3.6. Design of time planning

- The walking speed of group is considered as the climber who has the slowest walking in the group.
- The climbing speed of group is considered as the climber who has the slowest climbing in the group.
- At the end, time is calculated as "total distance" / "speed of group".

- At rainy days, the walking speed of climbers decreases in proportion of 20%.
- At nights, the walking speed of climbers decreases to half of it.
- At the undulating lands, the walking speed of climbers decreases to approximately half of it. [11]

#### **Typical speeds for an average party;**

- On a gentle trail, with a day pack: 3 to 5 km per hour.
- Up a steep trail, with a full overnight pack: 2 or 3 km per hour.
- Traveling cross-country up a moderate slope, with a day pack: 300 meters of elevation gain per hour.
- Traveling cross-country up a moderate slope, with a full overnight pack: 150 meters of elevation gain per hour. [10]

#### **Typical estimated duration of a trip; [10]**

<b>Trip Segment</b>	<b>Estimated Time</b>
Hike up the trail	2 hours
Cross-country approach	1 hour
The climb itself	4 hours
Time on the summit	1 hour
Descent time	2 hours
Return to the trail	1 hour
Hike out	1.5 hours
Total time estimated	12.5 hours
Contingencies	2 hours
Total time allowance	14.5 hours

### 6.3.7. Design of equipment planning

Equipment is considered with respect to season, climbing conditions of climbers, camping, duration of expedition, terrain conditions and number of climbers in the group.

#### **Sample Expedition Equipment List [10]**

##### ***Group Gear***

- Expedition-quality tent(s)
- Ground cloths
- Snow stakes and/or tent flukes
- Sponge and whisk broom
- Snow shelter construction tools: large snow shovel (for moving a lot of snow), small snow shovel (for delicate trimming), snow saw (for cutting blocks)
- Ropes
- Hardware: snow and ice gear (pickets, flukes, ice screws), rock gear (pitons, spring-loaded camping devices, chocks), carabiners, runners, daisy chains, fixed line, extra climbing equipment ( spare ice ax or tool, spare crampons, spare rescue pulleys)
- Stove gear: stove, windscreen and stove platform, fuel containers and fuel filter, matches and/or butane lighters, firestarter
- Cooking gear: pots, pot cozy, pot gripper, sponge/scrubber, dip cup, cooking spoon, snow sack (for collecting clean snow to melt for water)
- Food
- Water treatment: filter, chemicals
- Tent repair kit: pole splices, spare pole
- Stove repair kit
- Crampon repair kit: extra screws, connecting bars, straps
- Tape (duct, filament, fabric repair)
- Adhesive-backed repair cloth
- Seam repair compound

- Tools: slotted and Phillips screwdrivers; Allen wrenches; small pliers; small wire cutter/shears: file
- Sewing kit: assorted needles and thread; awl; assorted buttons, snaps, buckles, and D-rings; Velcro (hook and pile), fabric (Cordura, ripstop nylon), flat webbing
- Other: wire, accessory cord, pack buckle, extra ski-pole basket, patch kit for inflatable foam pads
- First Aid Kit\*
  - ✓ In addition to normal first aid items, the kit should include the following drugs, plus others recommended by a physician.
- Prescription drugs vary with the destination, but should include: antibiotics, strong analgesics, anti-diarrhetics, laxatives, and altitude medications ( acetazolamide, dexamethasone).
- Nonprescription drugs vary with the destination, but should include: cough suppressants, decongestants, mild analgesics (aspirin, ibuprofen).
- Wands
- Altimeter, map, compass
- Radio transceiver and extra batteries
- Mobile phone
- Latrine equipment

### ***Personal Gear***

- Synthetic-fabric underwear
- Insulating layers
- Down clothing
- Wind-protection and rain-protection garments (top and bottom)
- Extremities: hands (liner gloves, insulating gloves, mittens), feet (liner socks, insulating socks, vapor-barrier socks), head (balaclava, sun hat, face mask, wool hat)
- Other: bandannas, sun shirt, synthetic fill/down booties
- Sleeping bag
- Bivouac sack

- Vapor-barrier liner
- Inflatable foam pad or closed-cell foam pad
- Ice ax
- Second ice tool
- Seat harness
- Crampons
- Personal carabiners and slings
- Chock pick
- Belay device
- Rescue pulley
- Ascenders/prosiks
- Helmet
- Large-volume pack
- Pack cover
- Snowshoes
- Sled with associated hardware for pulling
- Duffel bag
- Avalanche transceiver
- Sunglasses and goggles
- Spare prescription glasses
- Pocketknife
- Headlamp
- Wide-mouth water bottles
- Personal hygiene: toilet paper, pee bottle, toothbrush, comb, chemical wash/wipes, sunscreen, lib balm, foot powder, earplugs
- Personal recreation: camera and film, books, journal, pen or pencil, personal stereo, playing cards

\*It is in the table of basic personal first aid kit. [10]

### **Basic Personal First Aid Kit**

Item	Use
Adhesive bandages	To cover small minor wounds
Butterfly bandages or Steristrips	To close minor lacerations
Sterile gauze pads	To cover larger wounds
Carlisle dressing or sanitary napkin	To absorb and control severe bleeding
Nonadherent dressings	To cover abrasions and burns
Self-adhering roller bandages	To hold dressings in place
SAM splint	To splint
Athletic tape	Multiple uses
Triangular bandages	To use as a sling or cravat (for splinting)
Moleskin or Molefoam	To cushion blister areas
Tincture of benzoin	To aid in adherence of adhesive tape; to protect skin
Providine iodine swabs	Antiseptic for surface wounds
Alcohol or soap pads	To cleanse skin
Thermometer	To measure body temperature
Sugar packets	To treat diabetes; for hypoglycemia intervention
Aspirin	To treat headache, pain; if the group includes children, bring acetaminophen tablets instead aspirin.
Anaphylaxis (epinephrine) kit (EpiPen)	To treat severe allergic reaction. Climbers should carry if known to have severe allergy.
Elastic bandage	To wrap sprains; for compression of injured area
Latex gloves	To serve as an infection barrier
Safety pins	Multiple uses
Tweezers	To remove splinters, ticks, wound debris
Plastic bag	To hold contaminated materials
Breathing barrier	To administer CPR, rescue breathing

#### 6.3.8. Design of carriage planning

The average carriage weight per person should be 18-20 kg. The maximum carriage weight for a person is 40-45 kg. However, the weight for a person shouldn't be higher than 25 kg. [11]

## 6.4. Functional Modeling

### 6.4.1. Data Flow Diagrams

- Context Level Data Flow Diagram

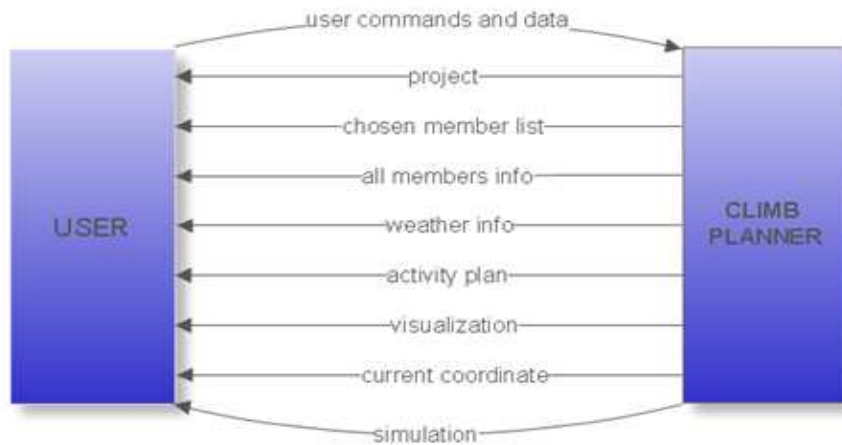


Figure 6.27. Context Level DFD

- **Level 1 Data Flow Diagram**

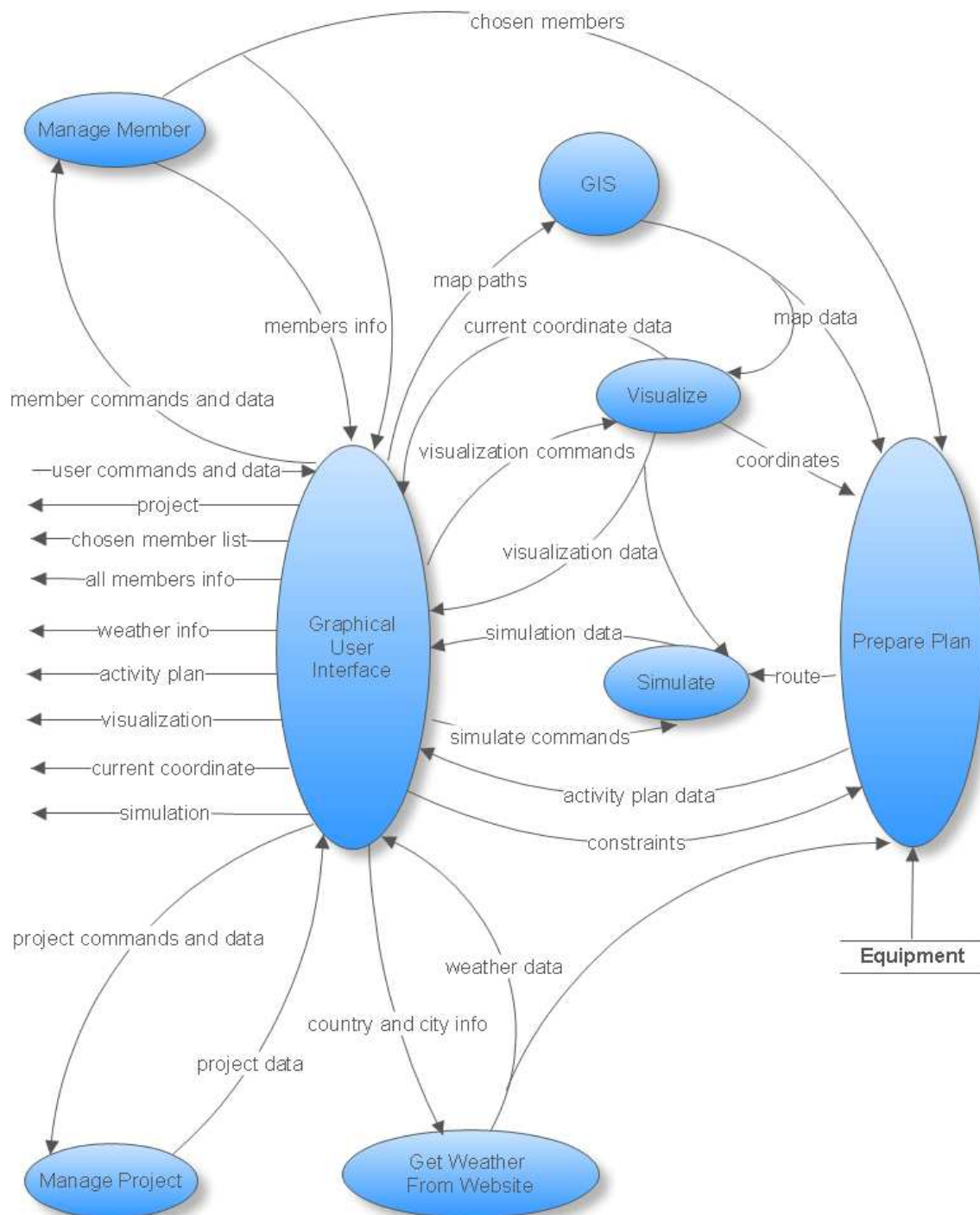


Figure 6.28. Level 1 DFD



- **Level 2 Data Flow Diagram : Visualize**

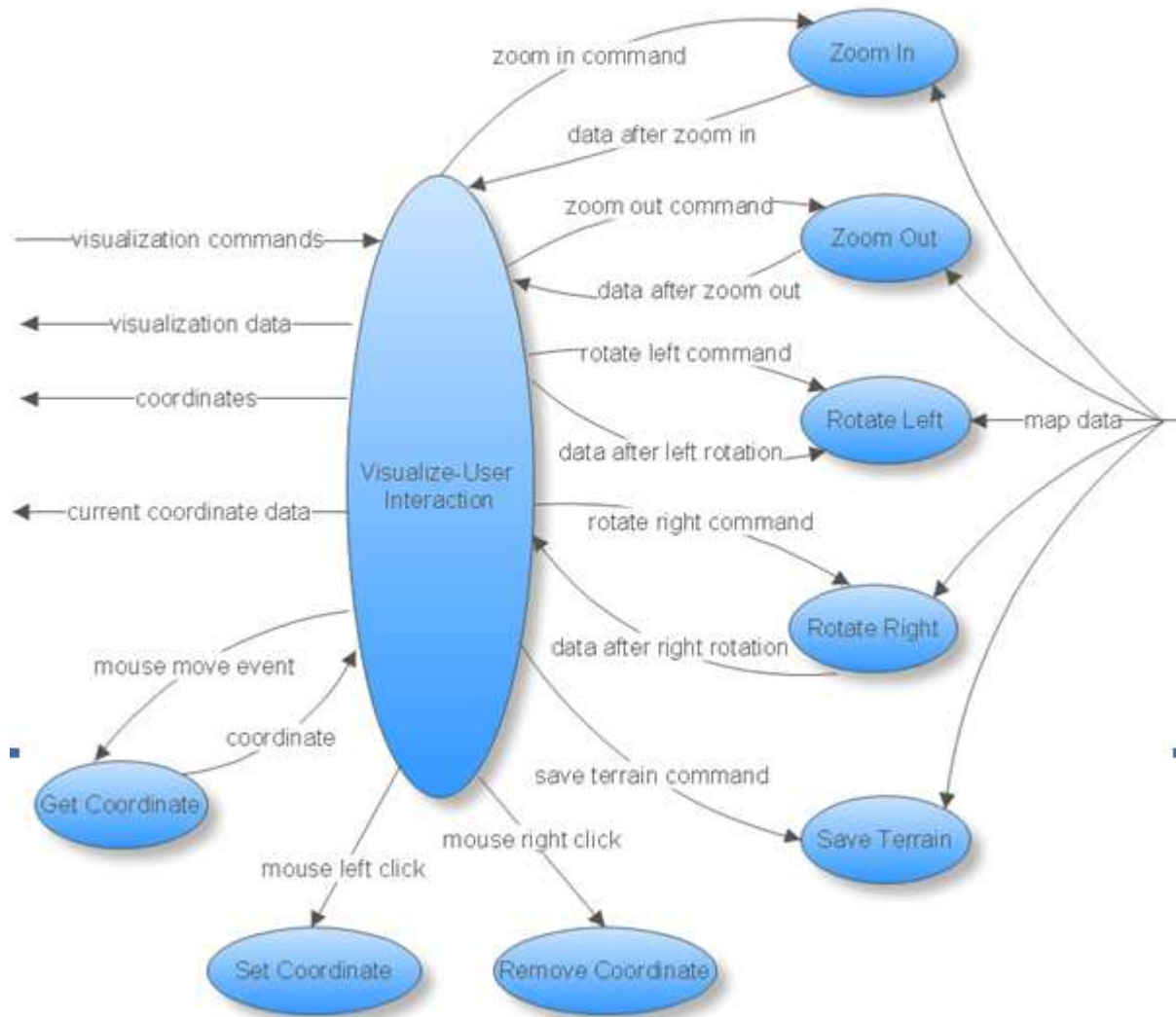
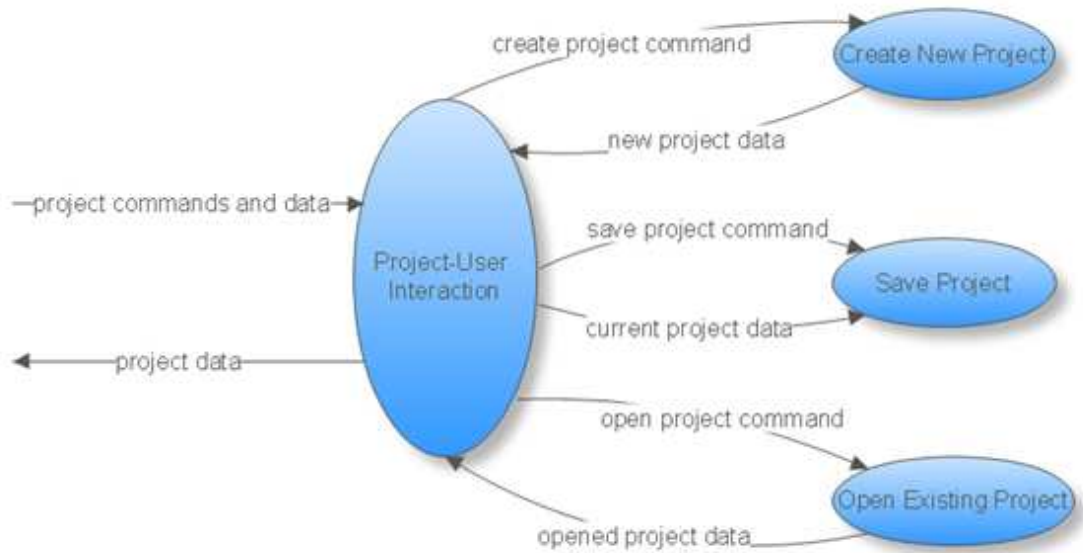


Figure 6.29. Level 2 DFD: Visualize

- **Level 2 DataFlowDiagram : Manage Project**



*Figure 6.30. Level 2 Data Flow Diagram: Manage Project*

- **Level 2 Data Flow Diagram : Manage Member**

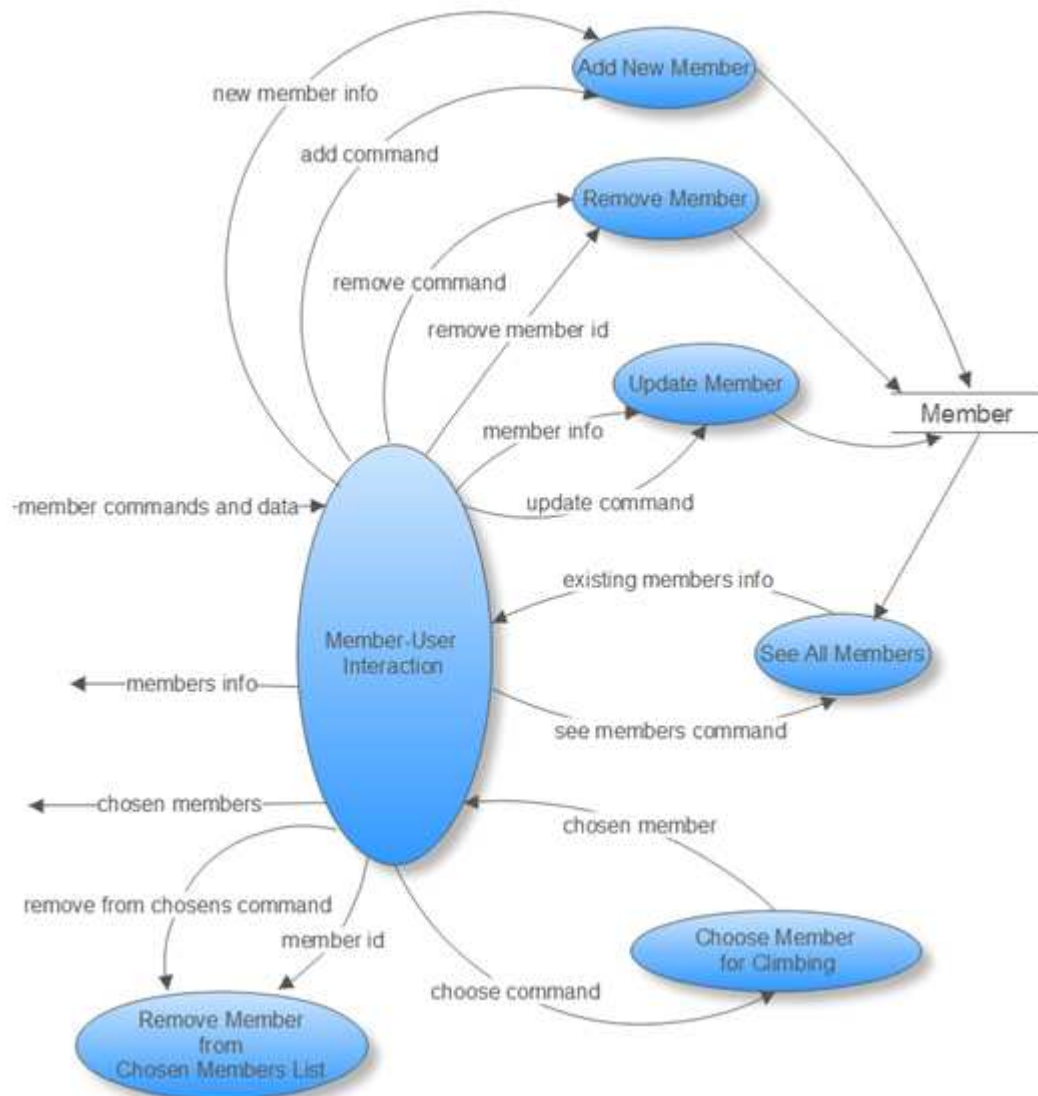


Figure 6.31. Level 2 DFD: Manage Member

### 6.4.2. Data Dictionary

<b>Name:</b>	<b>user commands and data</b>
<b>From:</b>	USER
<b>To:</b>	Graphical User Interface
<b>Description:</b>	Contains all commands and data taken from user

<b>Name:</b>	<b>member commands and data</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	Member-User Interaction
<b>Description:</b>	Contains all commands and data related to member

<b>Name:</b>	<b>add command</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Add New Member
<b>Description:</b>	The command for adding a new member to the system

<b>Name:</b>	<b>new member info</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Add New Member
<b>Description:</b>	Holds the information about the member that will be added to the system

<b>Name:</b>	<b>remove command</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Remove Member
<b>Description:</b>	The command for removing a member from the system

<b>Name:</b>	<b>remove member id</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Remove Member
<b>Description:</b>	The id of the member who will be removed from the system permanently

<b>Name:</b>	<b>update command</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Update Member
<b>Description:</b>	The command for updating an existing member

	in the system
--	---------------

<b>Name:</b>	<b>member info</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Update Member
<b>Description:</b>	The member whose information will be updated

<b>Name:</b>	<b>remove from chosens command</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Remove Member from Chosen Members List
<b>Description:</b>	The command for removing a member from the list that contains the members who are going to participate in the activity (chosen member list)

<b>Name:</b>	<b>member id</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Remove Member from Chosen Members List
<b>Description:</b>	The member id of the person who will be removed from the chosen members list (not from the system)

<b>Name:</b>	<b>choose command</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Choose Member for Climbing
<b>Description:</b>	The command for choosing a member to participate in the activity

<b>Name:</b>	<b>chosen member</b>
<b>From:</b>	Choose Member for Climbing
<b>To:</b>	Member-User Interaction
<b>Description:</b>	Holds information about the recently chosen member for climbing

<b>Name:</b>	<b>see members command</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	See All Members
<b>Description:</b>	The command for showing information about all members registered in the system

<b>Name:</b>	<b>existing members info</b>
--------------	------------------------------

<b>From:</b>	See All Members
<b>To:</b>	Member-User Interaction
<b>Description:</b>	Holds information about all members registered in the system

<b>Name:</b>	<b>members info</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Graphical User Interface
<b>Description:</b>	Holds information about all members to show them to the user

<b>Name:</b>	<b>chosen members</b>
<b>From:</b>	Member-User Interaction
<b>To:</b>	Graphical User Interface, Prepare Plan
<b>Description:</b>	Holds the information about the members chosen for climbing to show those members to the user and to help preparing activity plan

<b>Name:</b>	<b>chosen member list</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER
<b>Description:</b>	Holds the members chosen for the climbing to show those members to the user

<b>Name:</b>	<b>all members info</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER
<b>Description:</b>	Holds the information of all members existing in the system to show those members information to the user

<b>Name:</b>	<b>map paths</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	GIS
<b>Description:</b>	Contains all given map locations given by the user (it includes "dted", "dem", "geotiff" and "shp" file paths)

<b>Name:</b>	<b>map data</b>
<b>From:</b>	GIS
<b>To:</b>	Visualize, Prepare Plan

<b>Description:</b>	The map data gathered after reading the map files by GIS.
---------------------	---

<b>Name:</b>	<b>visualization commands</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	Visualize-User Interaction
<b>Description:</b>	commands to change the settings of visualization

<b>Name:</b>	coordinates
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Graphical User Interface
<b>Description:</b>	Holds the start and end coordinates, checkpoints and the points that has avalanche or falling rock risks to mark those point in the visualization of the terrain

<b>Name:</b>	<b>current coordinate data</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Graphical User Interface
<b>Description:</b>	As mouse moves on the terrain, the corresponding coordinate is shown. Current coordinate data holds longitude, latitude and elevation of the corresponding coordinate

<b>Name:</b>	<b>current coordinate</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER
<b>Description:</b>	Current coordinate shown to the user

<b>Name:</b>	<b>zoom in command</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Zoom In
<b>Description:</b>	The command for zooming in the terrain

<b>Name:</b>	<b>visualization data</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Graphical User Interface, Simulation
<b>Description:</b>	Holds the necessary information to visualize the terrain

<b>Name:</b>	<b>data after zoom in</b>
<b>From:</b>	Zoom In
<b>To:</b>	Visualize-User Interaction
<b>Description:</b>	Holds the visualization data after zooming in the terrain to show the effect of zooming in to the user

<b>Name:</b>	<b>zoom out command</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Zoom Out
<b>Description:</b>	The command for zooming out from the terrain

<b>Name:</b>	data after zoom out
<b>From:</b>	Zoom Out
<b>To:</b>	Visualize-User Interaction
<b>Description:</b>	Holds the visualization data after zooming out from the terrain to show the terrain with zoomed out to the user

<b>Name:</b>	<b>rotate left command</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Rotate Left
<b>Description:</b>	The command for rotating left in the terrain

<b>Name:</b>	<b>data after rotate left</b>
<b>From:</b>	Rotate Left
<b>To:</b>	Visualize-User Interaction
<b>Description:</b>	Holds the visualization data after rotating left in the terrain

<b>Name:</b>	<b>rotate right command</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Rotate Right
<b>Description:</b>	The command for rotating right in the terrain

<b>Name:</b>	<b>data after rotate right</b>
<b>From:</b>	Rotate Right
<b>To:</b>	Visualize-User Interaction
<b>Description:</b>	Holds the visualization data after rotating right in the terrain



<b>Name:</b>	<b>save terrain command</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Save Terrain
<b>Description:</b>	The command for saving the terrain in a user defined location

<b>Name:</b>	<b>mouse move event</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Get Coordinate
<b>Description:</b>	Action of mouse movement in the terrain to get the corresponding coordinate

<b>Name:</b>	<b>coordinate</b>
<b>From:</b>	Get Coordinate
<b>To:</b>	Visualize-User Interaction
<b>Description:</b>	Contains the coordinate of the corresponding point in the terrain

<b>Name:</b>	<b>mouse left click</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Set Coordinate
<b>Description:</b>	If user clicks the left button of the mouse on the terrain, the corresponding coordinate is set as one of the followings : start, end, checkpoint, avalanche risk, falling rock risk

<b>Name:</b>	<b>mouse right click</b>
<b>From:</b>	Visualize-User Interaction
<b>To:</b>	Remove Coordinate
<b>Description:</b>	If user clicks the right button of the mouse on the terrain, if there exists a predefined coordinate (start, end, etc.), that coordinate will be removed.

<b>Name:</b>	<b>project commands and data</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	Project-User Interaction
<b>Description:</b>	Contains all commands and data related to project

<b>Name:</b>	<b>project data</b>
<b>From:</b>	Project-User Interaction
<b>To:</b>	Graphical User Interface
<b>Description:</b>	Holds the current project data that are necessary when loading a project

<b>Name:</b>	<b>create project command</b>
<b>From:</b>	Project-User Interaction
<b>To:</b>	Create New Project
<b>Description:</b>	The command for creating a new project

<b>Name:</b>	<b>new project data</b>
<b>From:</b>	Create New Project
<b>To:</b>	Project-User Interaction
<b>Description:</b>	Holds the data of the newly created project

<b>Name:</b>	<b>save project command</b>
<b>From:</b>	Project-User Interaction
<b>To:</b>	Save Project
<b>Description:</b>	The command for saving the project

<b>Name:</b>	<b>current project data</b>
<b>From:</b>	Project-User Interaction
<b>To:</b>	Save Project
<b>Description:</b>	Holds the current project data to be saved

<b>Name:</b>	<b>open project command</b>
<b>From:</b>	Project-User Interaction
<b>To:</b>	Open Existing Project
<b>Description:</b>	The command for opening an existing project

<b>Name:</b>	<b>opened project data</b>
<b>From:</b>	Open Existing Project
<b>To:</b>	Project-User Interaction
<b>Description:</b>	Holds the data of the opened project

<b>Name:</b>	<b>project</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER

<b>Description:</b>	The data to show the current settings of the project to the user
---------------------	--

<b>Name:</b>	<b>simulation data</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	Simulate
<b>Description:</b>	The data combined with route and terrain

<b>Name:</b>	<b>simulate commands</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	Simulate
<b>Description:</b>	Contains playing and saving simulation commands

<b>Name:</b>	<b>country and city info</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	Get Weather From Website
<b>Description:</b>	Country and city of the mountain to get the weather conditions from a website

<b>Name:</b>	<b>weather data</b>
<b>From:</b>	Get Weather From Website
<b>To:</b>	Graphical User Interface, Prepare Plan
<b>Description:</b>	Weather data taken from the website to show the user and to use in preparing plan

<b>Name:</b>	<b>weather info</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER
<b>Description:</b>	Weather information shown to the user

<b>Name:</b>	<b>constraints</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	Prepare Plan
<b>Description:</b>	Holds all given constraints by user

<b>Name:</b>	<b>route</b>
<b>From:</b>	Prepare Plan
<b>To:</b>	Simulate

<b>Description:</b>	The route of the climbing that is shown to the user in simulation
---------------------	---

<b>Name:</b>	<b>activity plan data</b>
<b>From:</b>	Prepare Plan
<b>To:</b>	Graphical User Interface
<b>Description:</b>	Generated activity plan data to show the user and save the activity plan report

<b>Name:</b>	<b>activity plan</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER
<b>Description:</b>	The activity plan shown to the user including the duration of the climbing, camping location and times, food, equipment and emergency equipment list

<b>Name:</b>	<b>visualization</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER
<b>Description:</b>	Visualization of the terrain shown to the user

<b>Name:</b>	<b>simulation</b>
<b>From:</b>	Graphical User Interface
<b>To:</b>	USER
<b>Description:</b>	Simulation of the climbing shown to the user

## 6.5. Data Design

### 6.5.1. Database Design

Since the GIS tool that we are going to use (GeoTools) is capable of storing the map information in its own data store, we don't need to create any table for storing maps. So, the data that we will store is not complex. We have two basic tables for storing member information and equipment information.

#### 6.5.1.1. Tables

##### **Member Table**

Member Table holds the basic information about a member. As user adds, removes or updates a member, his information is saved to database by using this table. In the next steps of the project, new fields can be added to this table when needed.

Field Name	Data Type
<u>Id</u>	INTEGER
Name	VARCHAR(15)
Surname	VARCHAR(20)
ClimbingSpeed	FLOAT
WalkingSpeed	FLOAT
CarriageCapacity	FLOAT
ExperienceLevel	INTEGER

*Table 6.32. Member Table*

- *Id* is the unique id of the member.
- *Name* is the name of the member.
- *Surname* is the surname of the member.
- *ClimbingSpeed* is the climbing speed of the member in m/h.
- *WalkingSpeed* is the walking speed of the member in km/h.
- *CarriageCapacity* is the carriage capacity of the member in kg
- *ExperienceLevel* is the experience level of the member.

### **Equipment Table**

Equipment Table holds the basic information about an equipment. It also includes the equipments that are needed in case of emergency. This table is needed for preparing the activity plan. User will not be able to access Equipment Table. In the next steps of the project, new fields can be added to this table as needed.

Field Name	Data Type
Id	INTEGER
Name	VARCHAR(30)
Weight	FLOAT
IsPersonal	BOOLEAN
NeededforWalking	BOOLEAN
NeededforClimbing	BOOLEAN
NeededforCamping	BOOLEAN
Season	INTEGER
IsEmergency	BOOLEAN

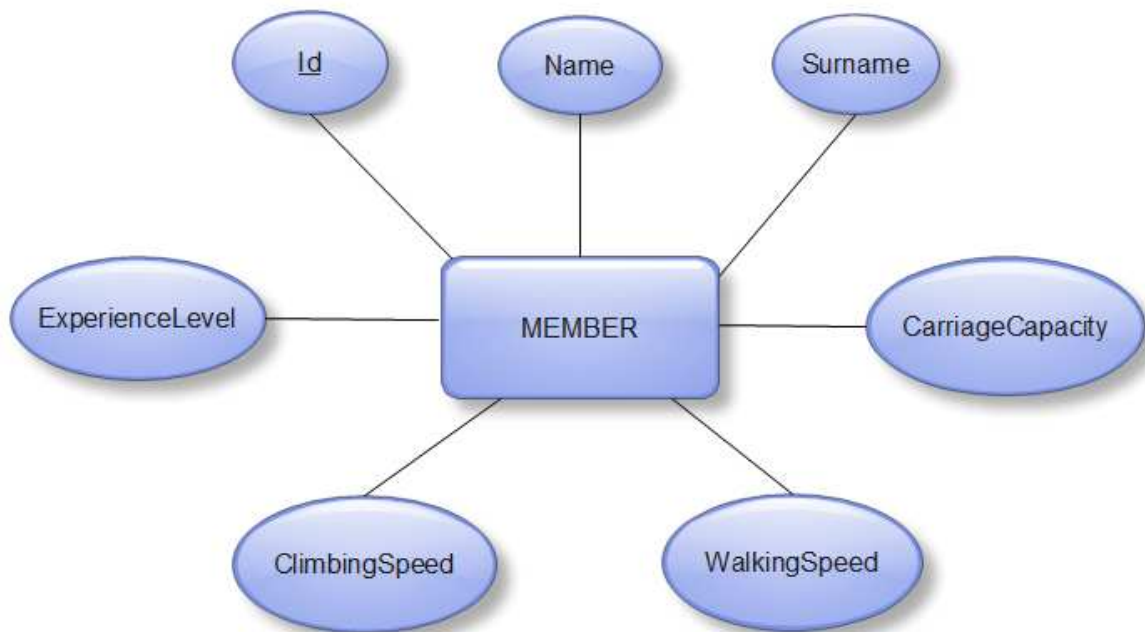
*Table 6.33. Equipment Table*

- *Id* is the unique id of the equipment.
- *Name* is the name of the equipment.
- *Weight* is the weight of the equipment.
- *IsPersonal* is true if the equipment is a personal one, false otherwise.
- *NeededforWalking* is true if the equipment is needed for a walking activity, false otherwise.
- *NeededforClimbing* is true if the equipment is needed for a climbing activity, false otherwise.
- *NeededforCamping* is true if the equipment is needed for camping, false otherwise.

- *Season* is the season in which the equipment is used, it can take three different values for winter, summer or both.
- *IsEmergency* is true if the equipment is an emergency equipment, false otherwise.

#### 6.5.1.2. ER Diagrams

Since we don't store any complex data and the data that we store are not related to each other, the ER diagrams are not complex, also. We have ER diagrams of only our database tables.



*Figure 6.34. ER Diagram of Member*



*Figure 6.35. ER Diagram of Equipment*

### 6.5.2. File and Folder Formats and Syntax

In this project, we use some kind of file formats and folder arrangement. We will mention about these formats under map files, image and video files and system files and folders headings.

#### 6.5.2.1. Map Files

We will use map files for getting terrain information in a format from user. We have thought the consistency of the file formats with our GIS tool, when searching map files. The “dted” and “dem” raster map formats are suggested to us for elevation information of the terrain by Aselsan. We have searched on them and we saw that they are usable and readable for our GIS tool. As satellite image we also think about the easy access from internet. Hence we have found the “geotiff” format. On the other hand, as vector file, we have decided to use “shp” shape file format because it is able to contain the necessary information about especially water wells, rivers and lakes.



### 6.5.2.2. System Files and Folders

The software will use some file and folder format specifications. We have designed a file format which contains all existing information of a project when it is saved. We will save the project file with a “srs” format because we want to use an original file format and this file will have the project’s information in XML format. Because XML is a useful format and Java has ready functions for reading, writing or parsing etc. for this file format.

The Strider’s sample workspace view is below.

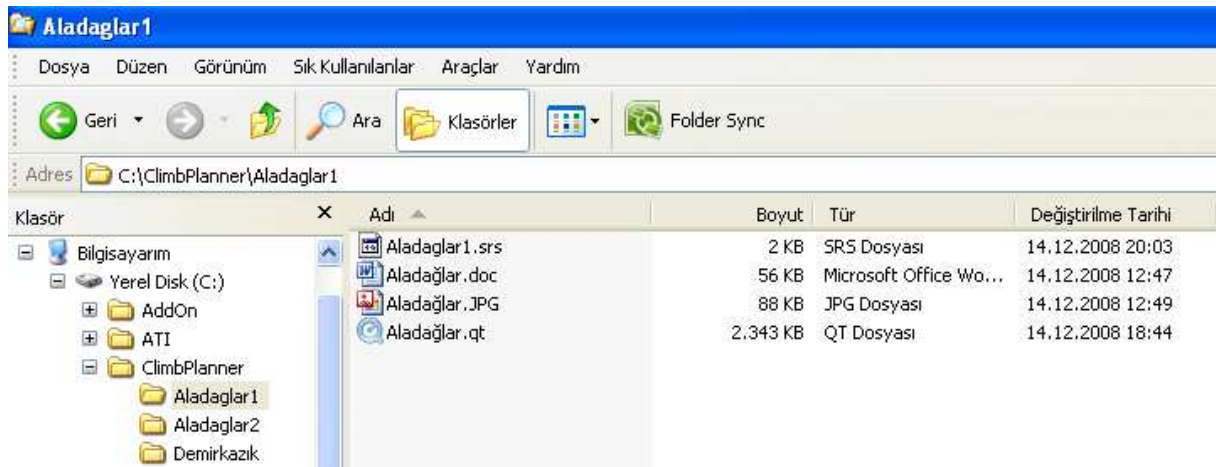


Figure 6.36. Strider’s workspace

As we mention above, the “srs” file contains the projects current state information. It is in XML format. The syntax of an “srs” file can be seen from the example of “Aladaglar1.srs”. “Aladaglar1.srs” file’s content is like that;

```
<?xml version="1.0" encoding="utf-8" ?>
<projectDescription>

  <projectName>Aladaglar1</projectName>
  <projectPath>C:\ClimbPlanner\Aladaglar1</projectPath>

  <climbers>
    <id>1449222</id>
    <id>1449164</id>
    <id>1502095</id>
    <id>1448836</id>
  </climbers>
</projectDescription>
```

```

</climbers>

<projectMaps>
  <elevationMaps>
    <elevationMap>C:\ClimbPlanner\aladag.dem</elevationMap>
  </elevationMaps>
  <shapeFiles>
    <shapeFile>C:\ClimbPlanner\aladag1.shp</shapeFile>
    <shapeFile>C:\ClimbPlanner\aladag2.shp</shapeFile>
  </shapeFiles>
  <satelliteImages>
    <satelliteImage>C:\ClimbPlanner\aladag.geotiff</satelliteImage>
    <satelliteImage>C:\ClimbPlanner\aladag2.geotiff</satelliteImage>
  </satelliteImages>
</projectMaps>

<time>
  <day>20</day>
  <month>6</month>
  <year>2009</year>
  <hour>8</hour>
  <minute>0</minute>
</time>

<place>
  <country>Turkey</country>
  <city>Ankara</city>
</place>

<weather>
  <lowestTemp>2.3</lowestTemp>
  <highestTemp>15.4</highestTemp>
  <pressure>102.3</pressure>
  <windDirection>NORTH</windDirection>
  <windSpeed>2.5</windSpeed>
  <season>WINTER</season>
  <rainy>true</rainy>
</weather>

<constraints>
  <constraint>SHORTESTPATH</constraint>
  <constraint>MINIMIZINGTIME</constraint>
</constraints>

<!-- Start Coordinate -->
<startCoord>
  <longitude />
  <latitude />
  <elevation />

```

```
</startCoord>

<!-- End Coordinate -->
<endCoord>
    <longitude />
    <latitude />
    <elevation />
</endCoord>

<!-- Check Points (if exist) -->
<checkPoints />

<avalanche>
    <longitude />
    <latitude />
    <elevation />
</avalanche>

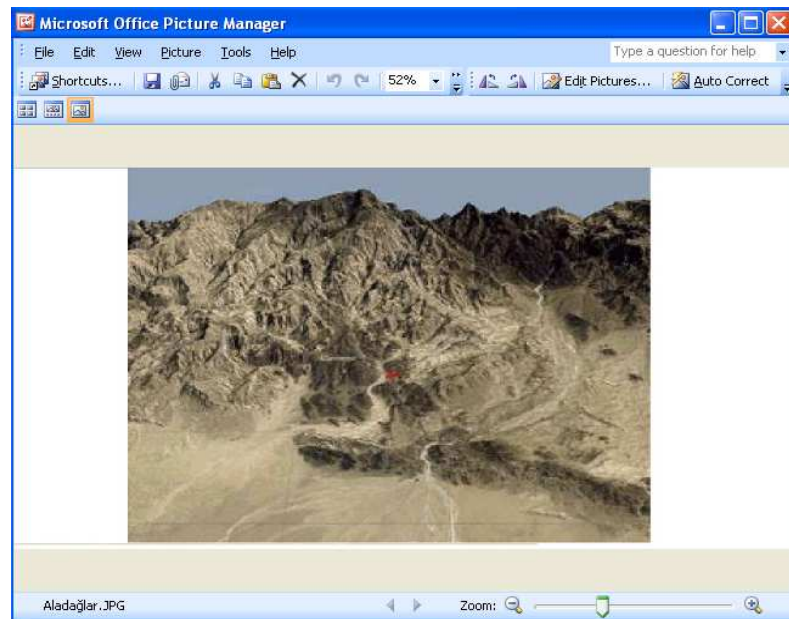
<fallingRock>
    <longitude />
    <latitude />
    <elevation />
</fallingRock>

</projectDescription>
```

#### 6.5.2.3. Image and Video Files

We will use the image and video files for saving the visualization and simulation of the terrain. For image file we will use a generic file format “.jpeg”. We think that this file format will be consistent with the systems of users. The user can save the visualization of terrain as an image, which is a screenshot of the visualization, anytime.

The “.jpeg” file of this image is like that;



*Figure 6.37. Visualization of the terrain*

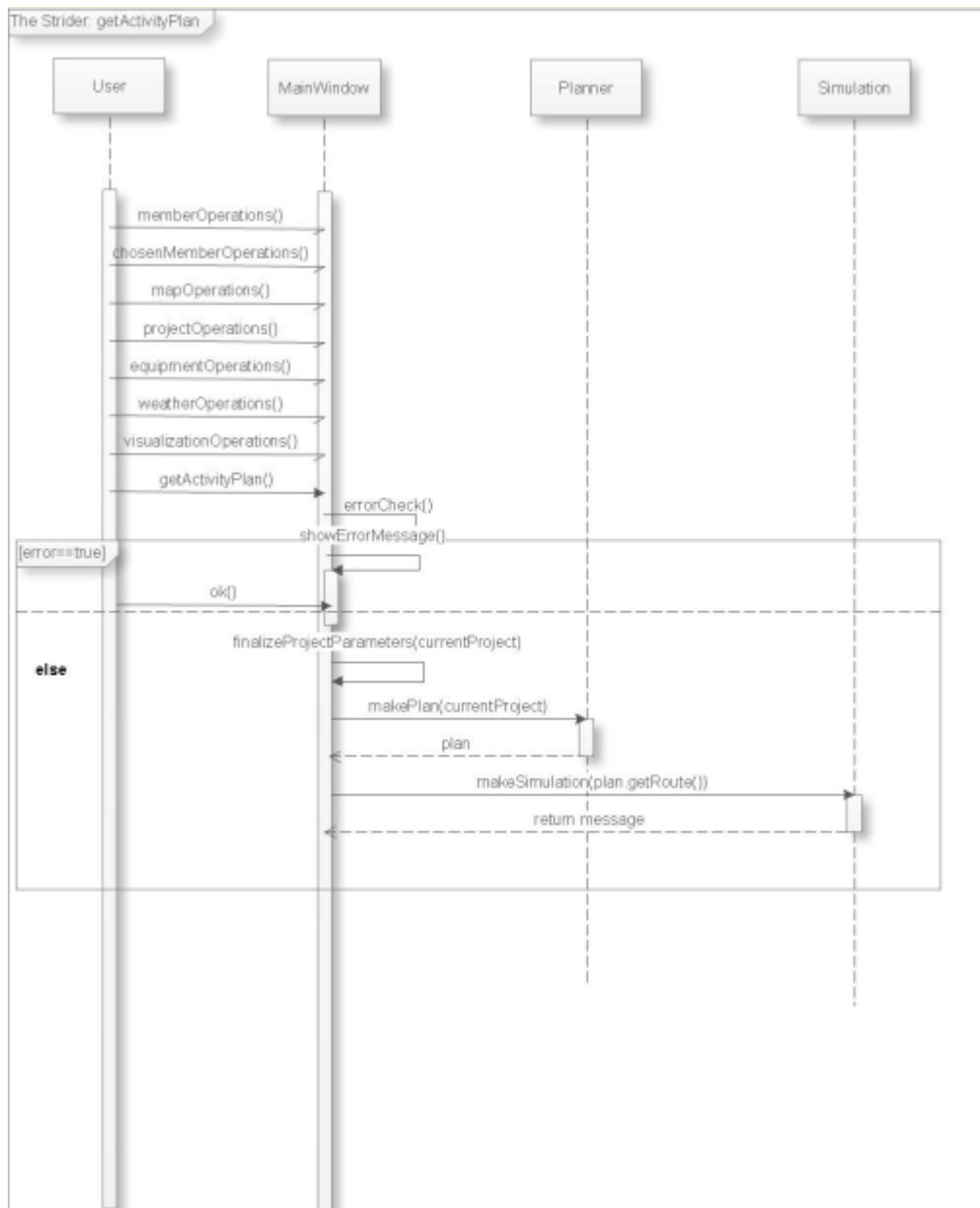
In the simulation part, user can save the simulation as a “qt” file that can be opened with QuickTime multimedia framework. A screenshot of this video is below. In this video, camera will follow the red colored route which is planned by Strider from the start coordinate to the end coordinate.



*Figure 6.38. Simulation of the route*

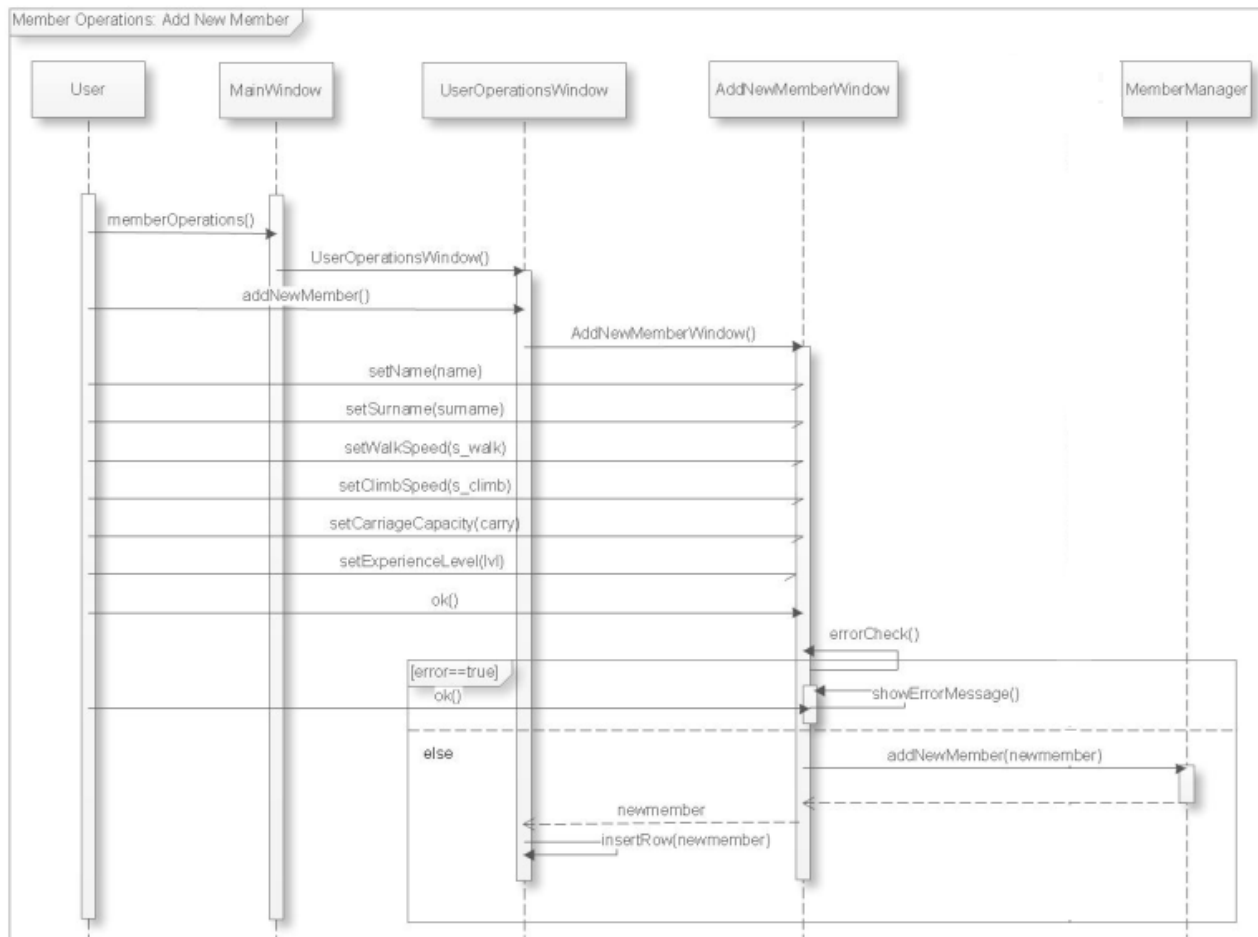
## 6.6. Behavioral Design

### 6.6.1. Strider General Behavior

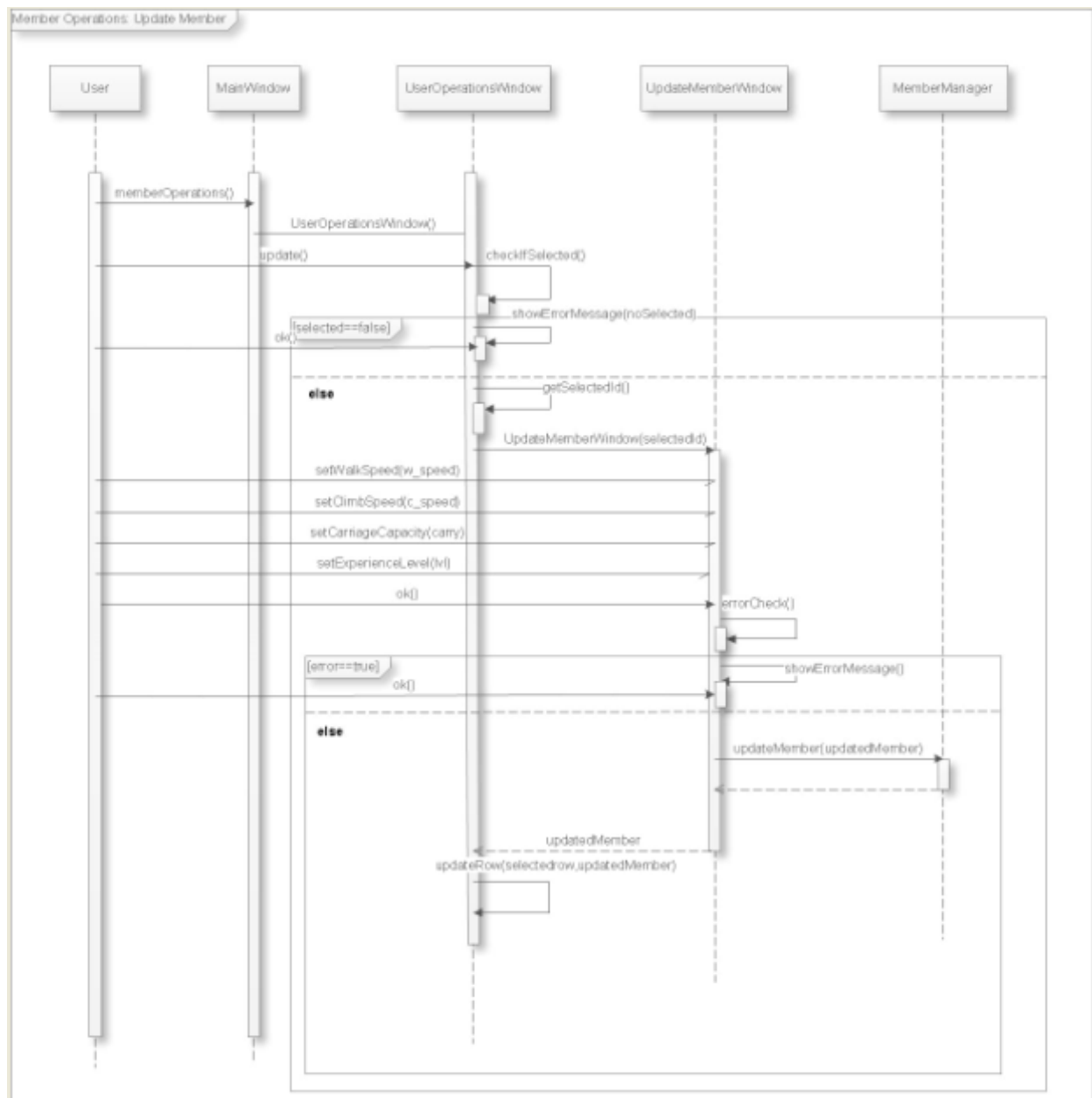


## 6.6.2. Member Operations

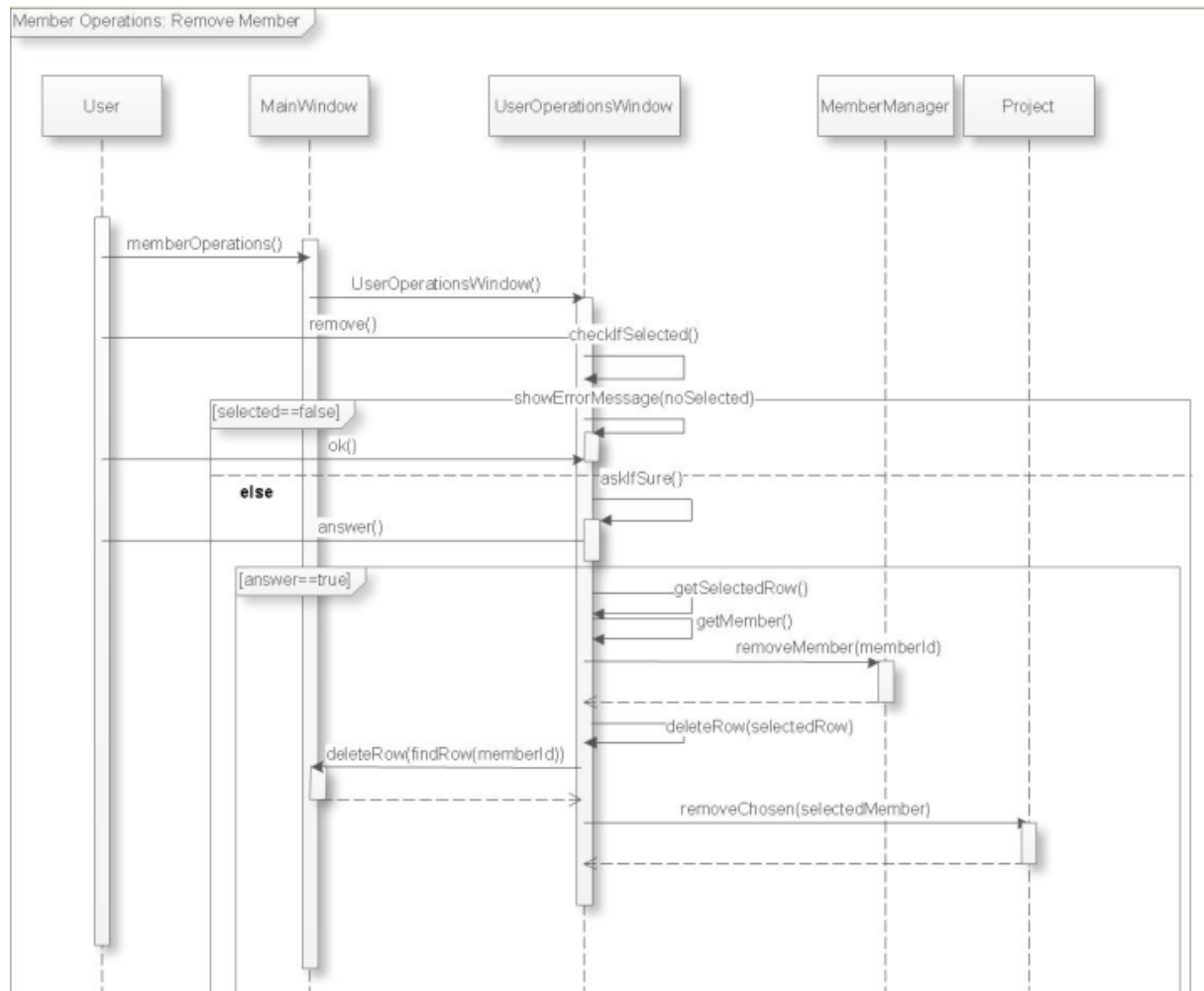
### 6.6.2.1. Add New Member



### 6.6.2.2. Update Member



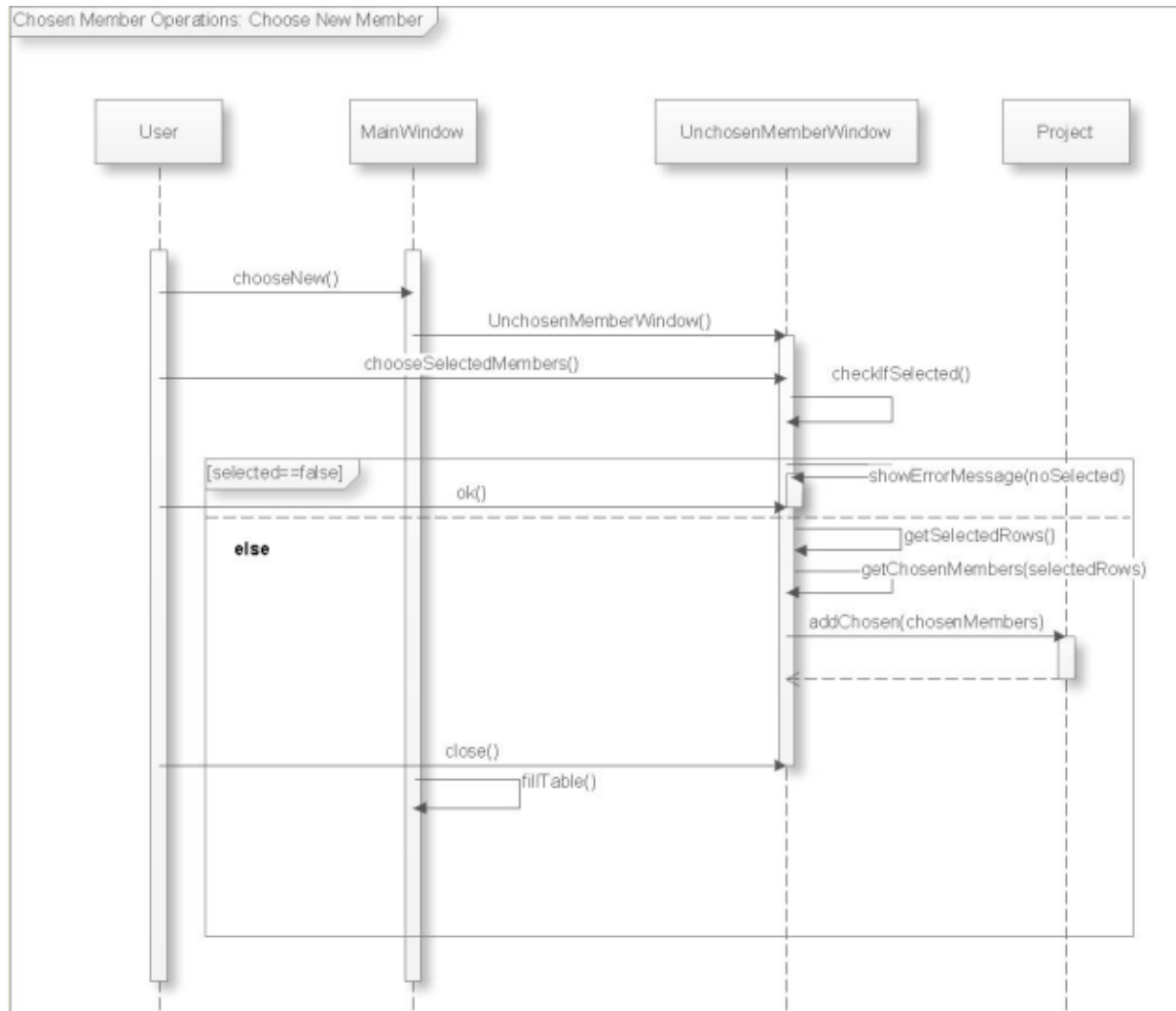
### 6.6.2.3. Remove Member



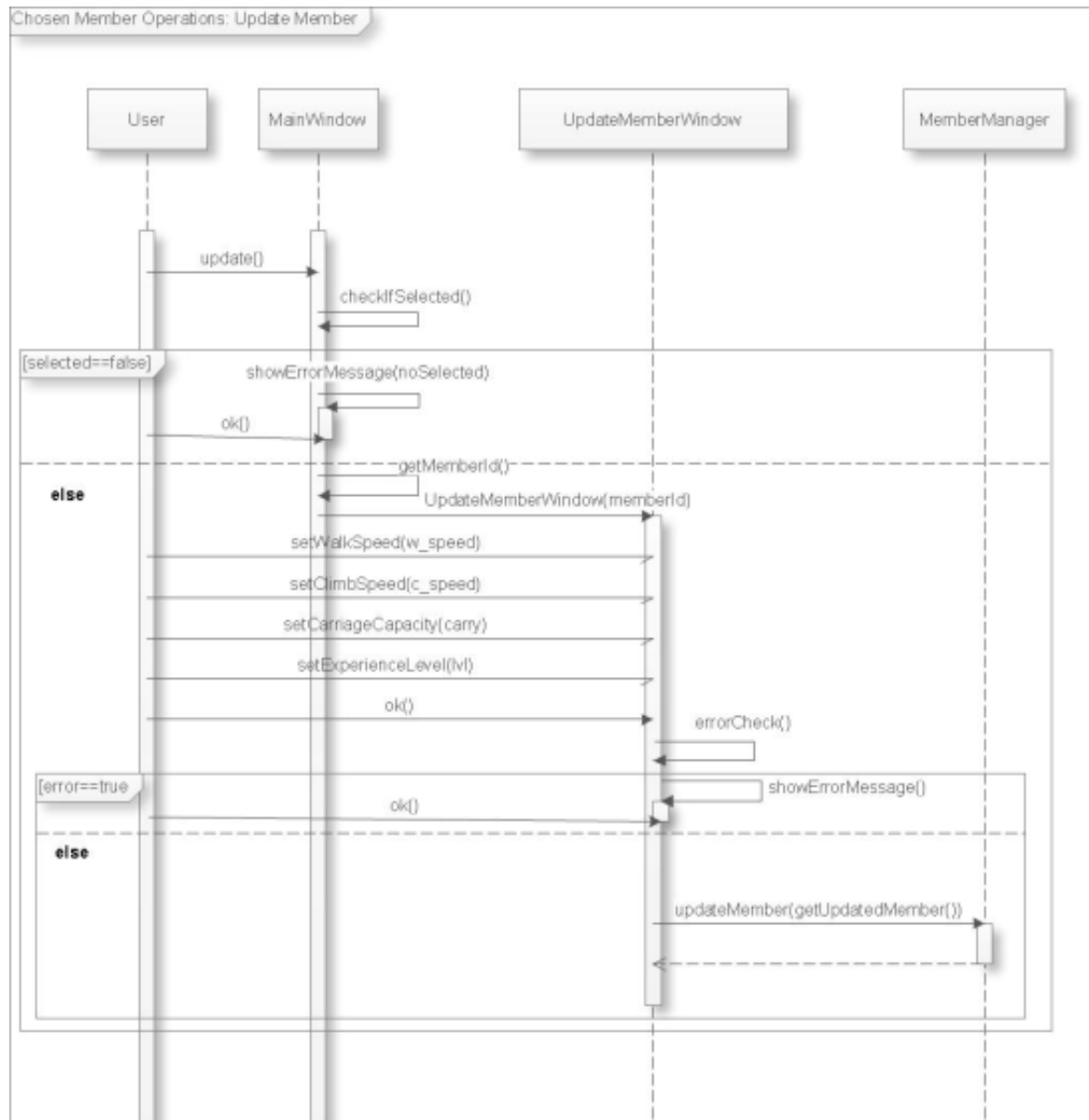


### 6.6.3. Chosen Member Operations

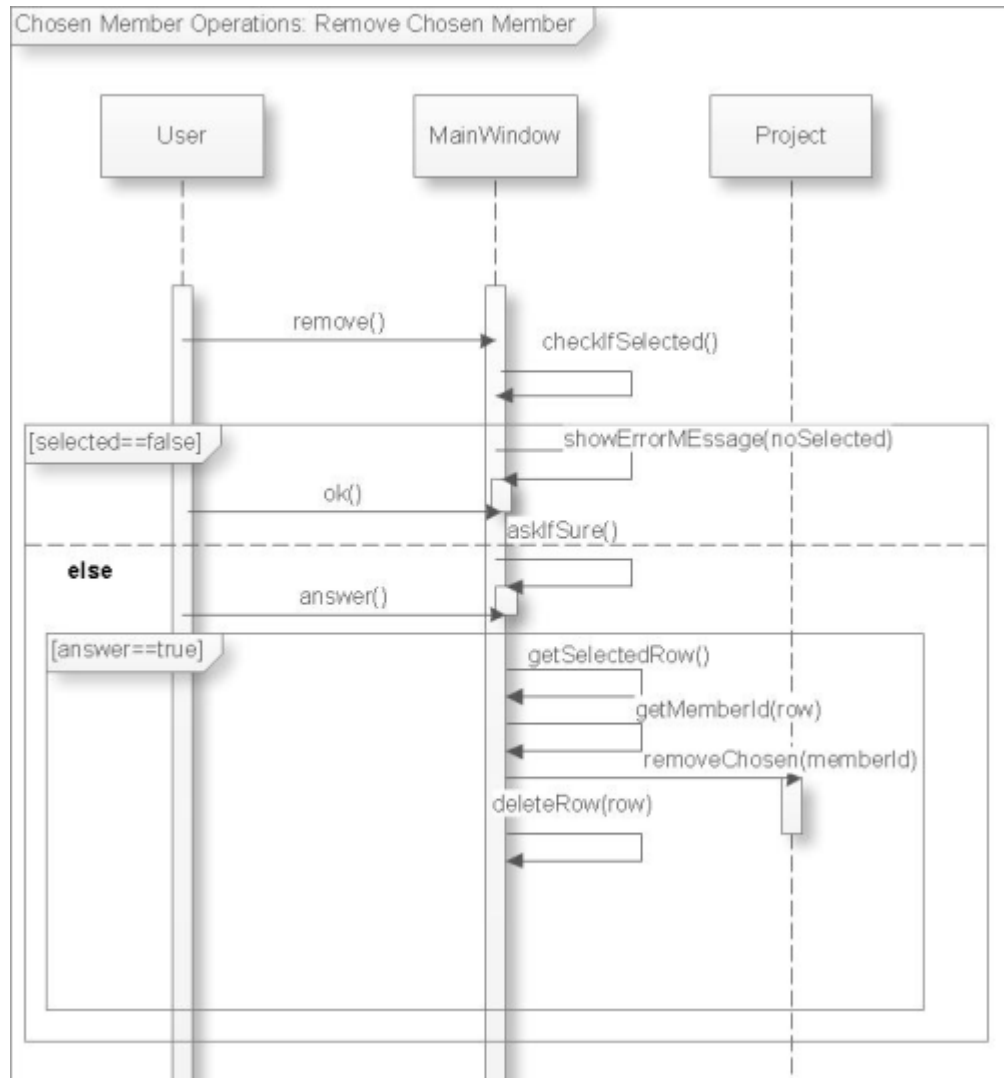
#### 6.6.3.1. Chose New Member



### 6.6.3.2. Update Member

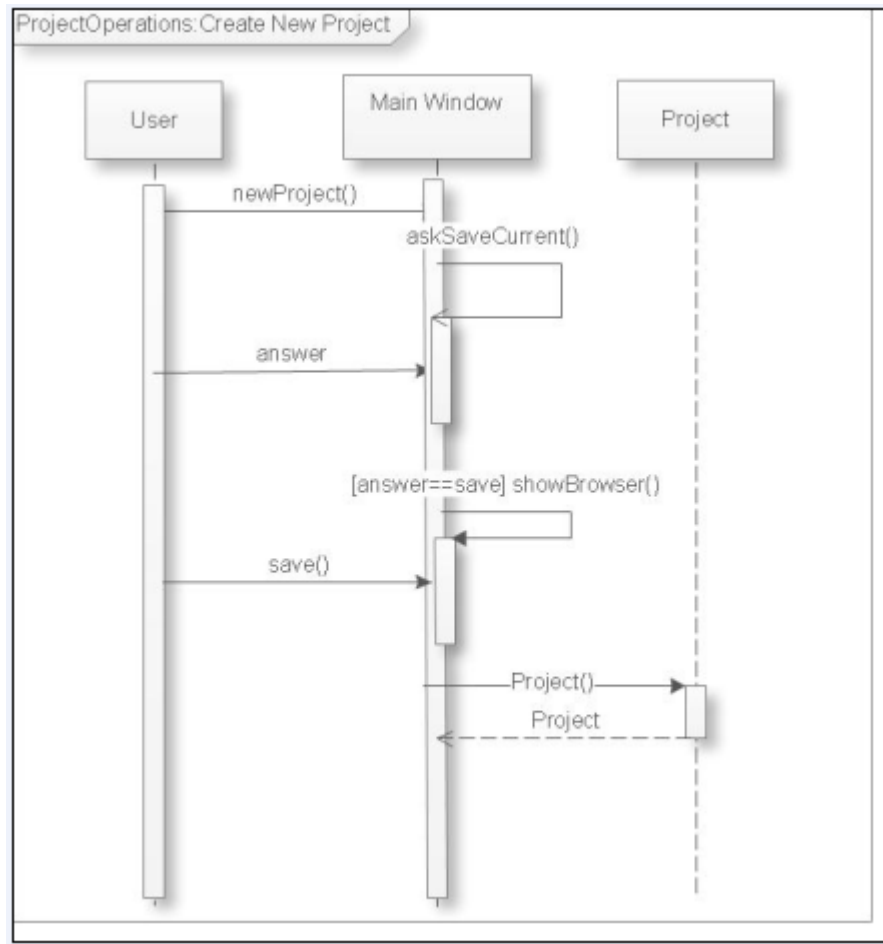


### 6.6.3.3. Remove Chosen Member

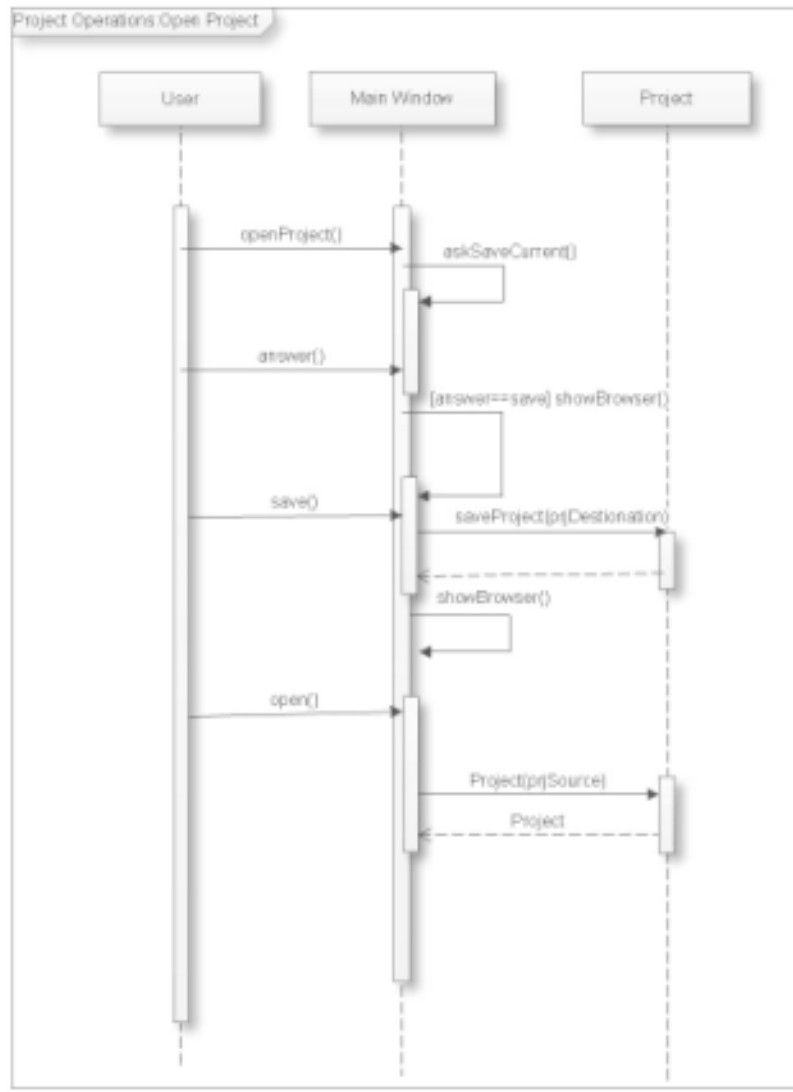


## 6.6.4. Project Operations

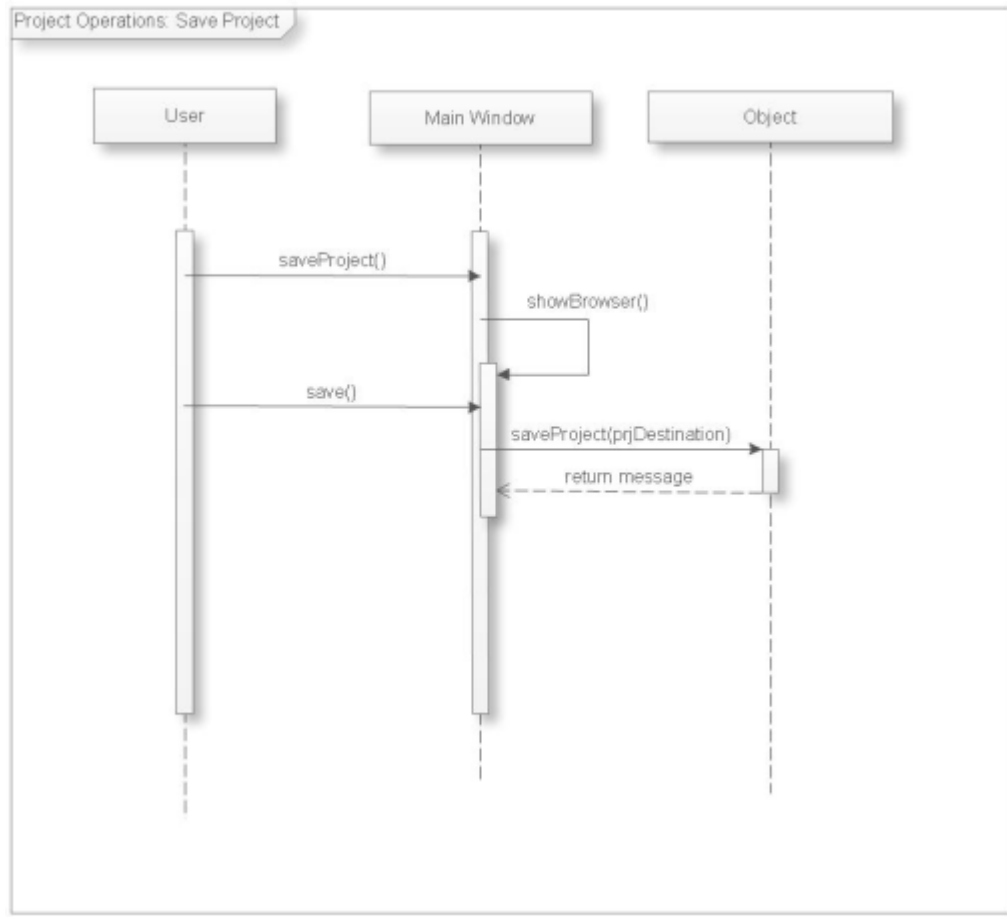
### 6.6.4.1. Create New Project



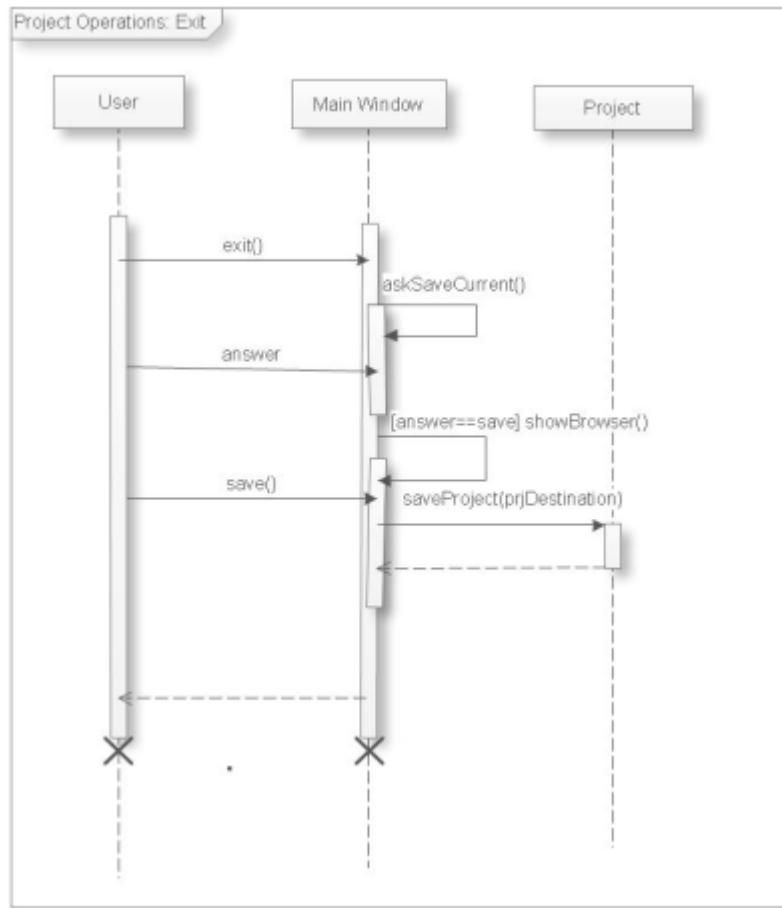
#### 6.6.4.2. Open Project



### 6.6.4.3. Save Project

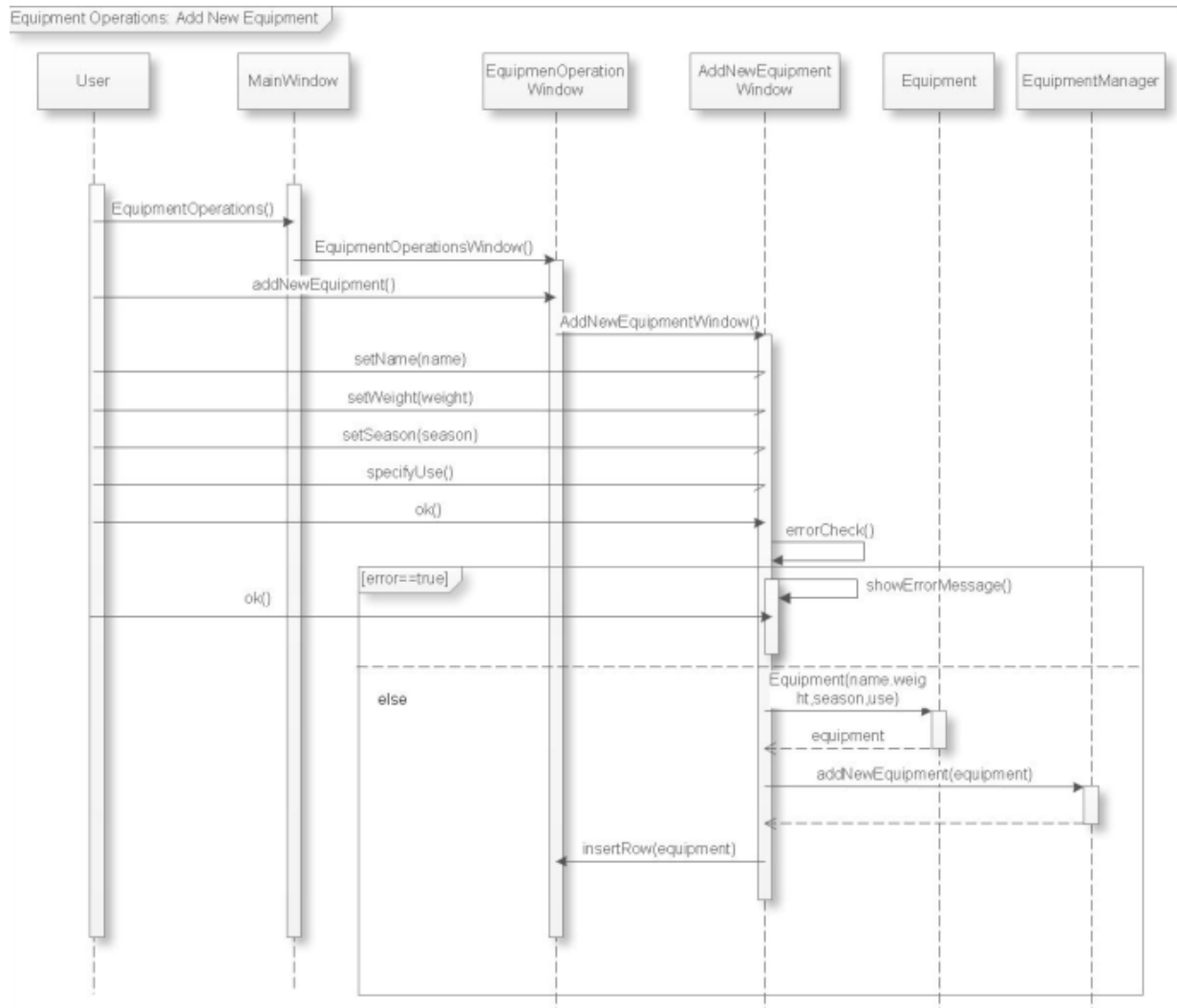


#### 6.6.4.4. Exit Project



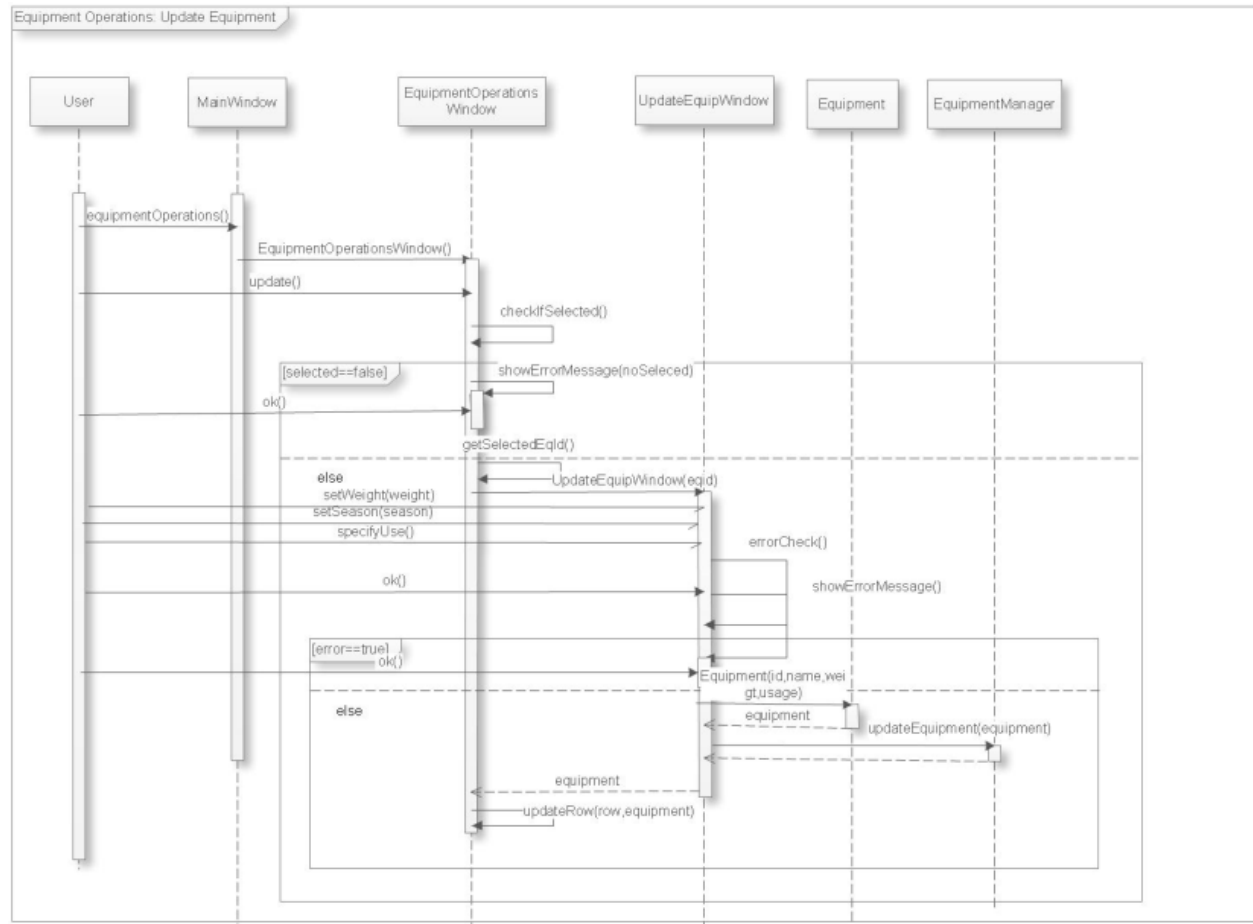
## 6.6.5. Equipment Operations

### 6.6.5.1. Add New Equipment

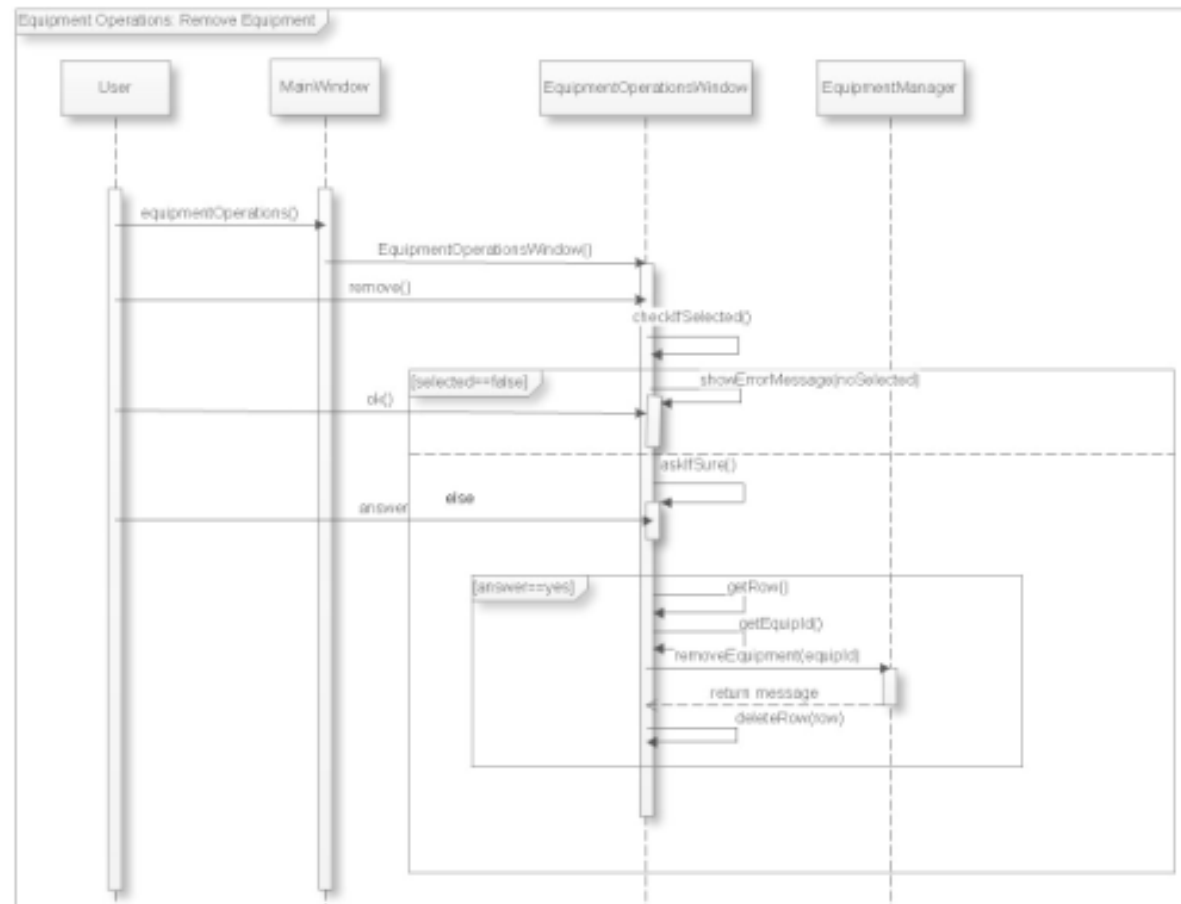




### 6.6.5.2. Update Equipment

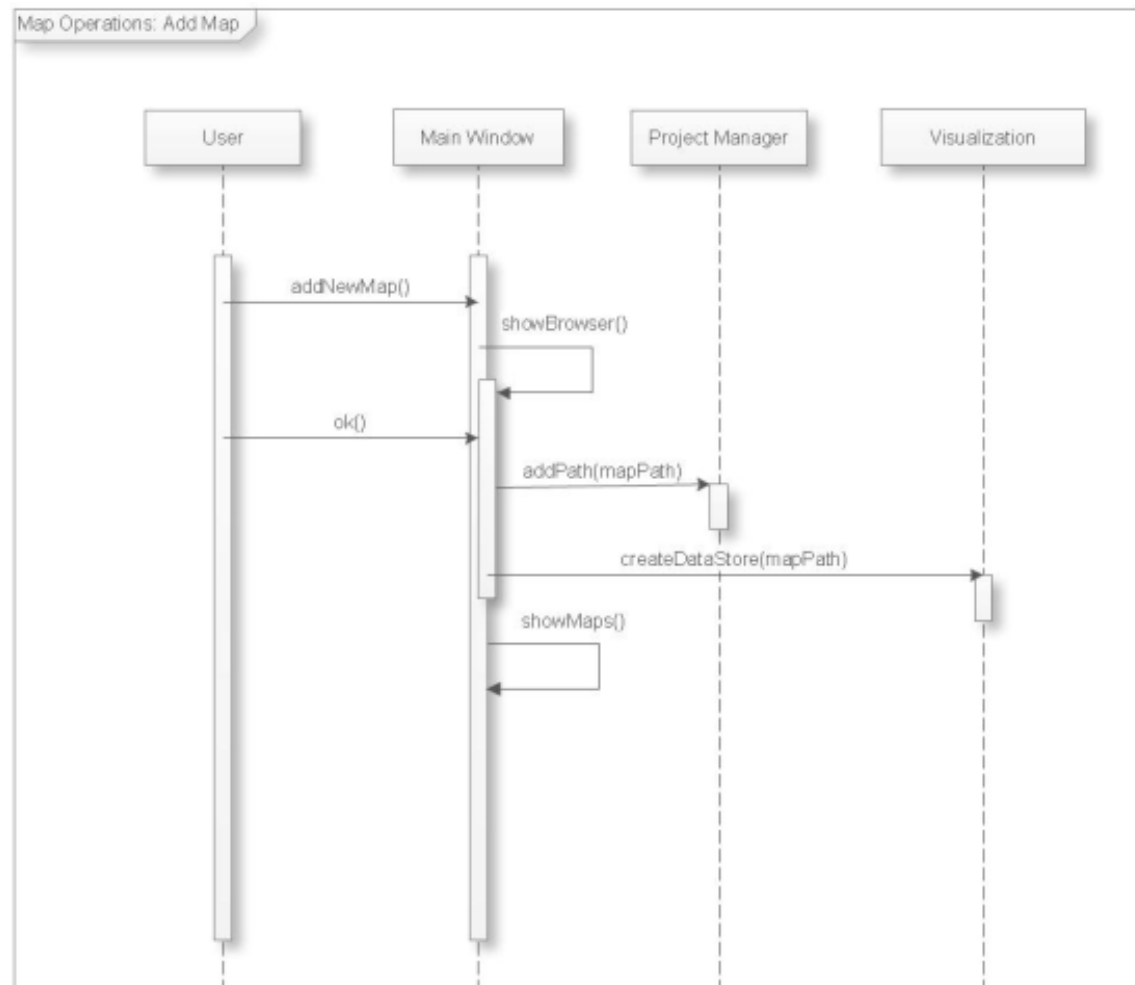


### 6.6.5.3. Remove Equipment

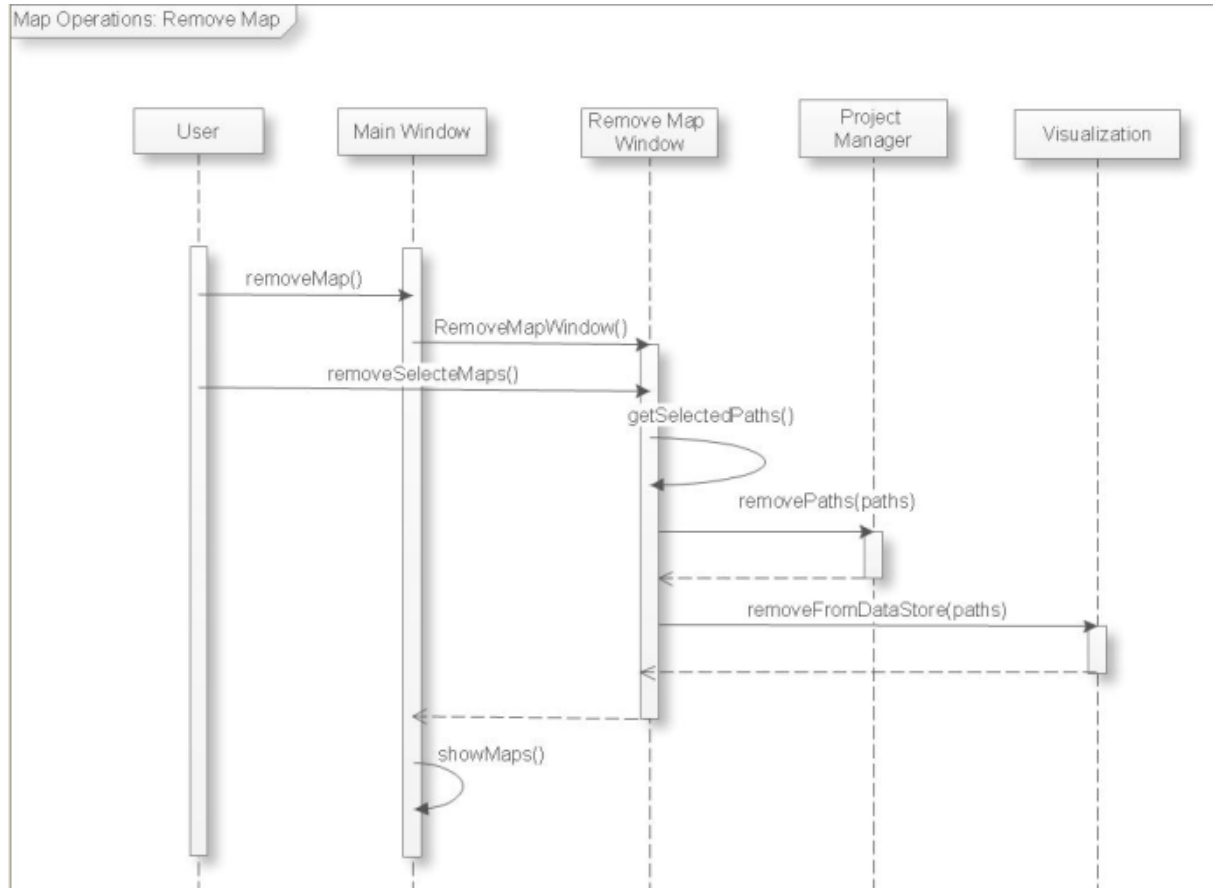


## 6.6.6. Map Operations

### 6.6.6.1. Add Map



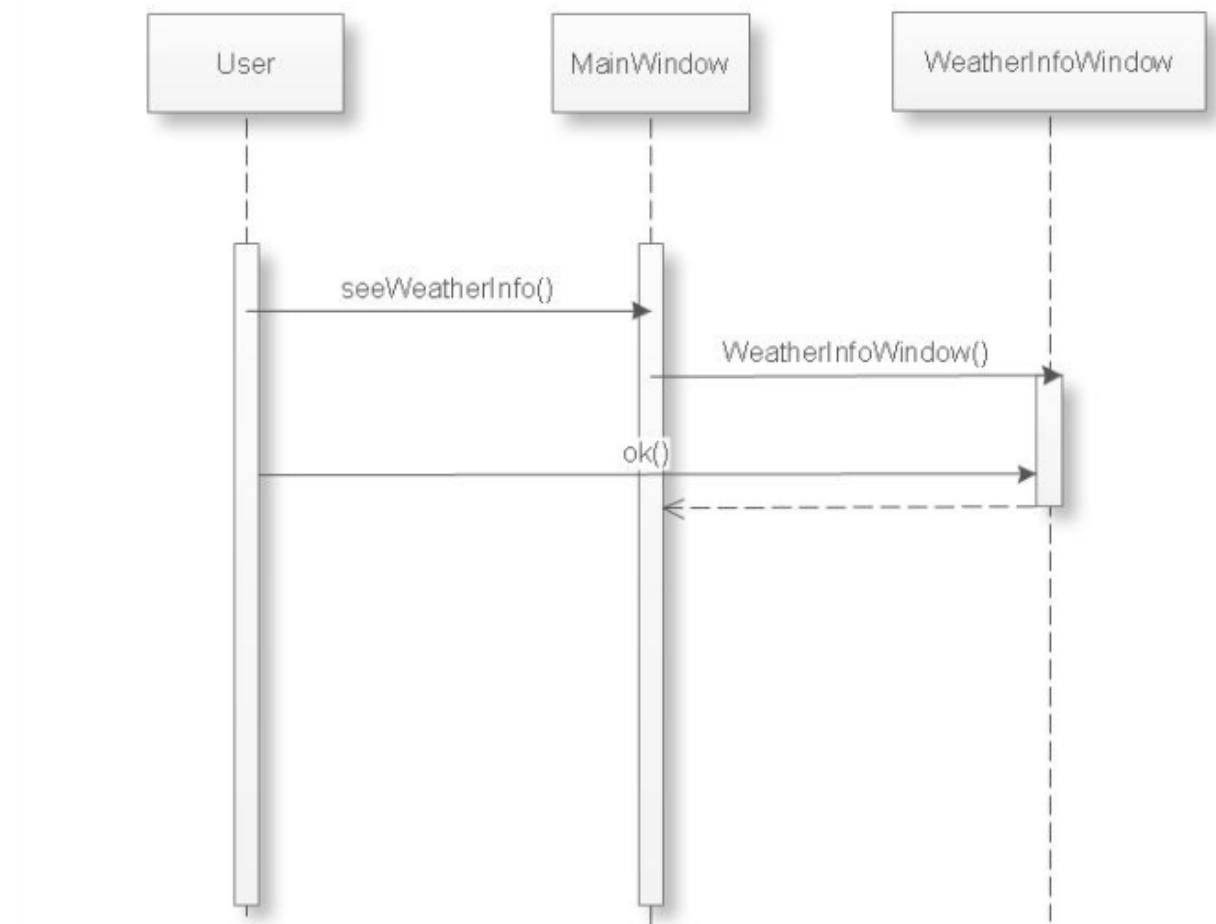
### 6.6.6.2. Remove Map



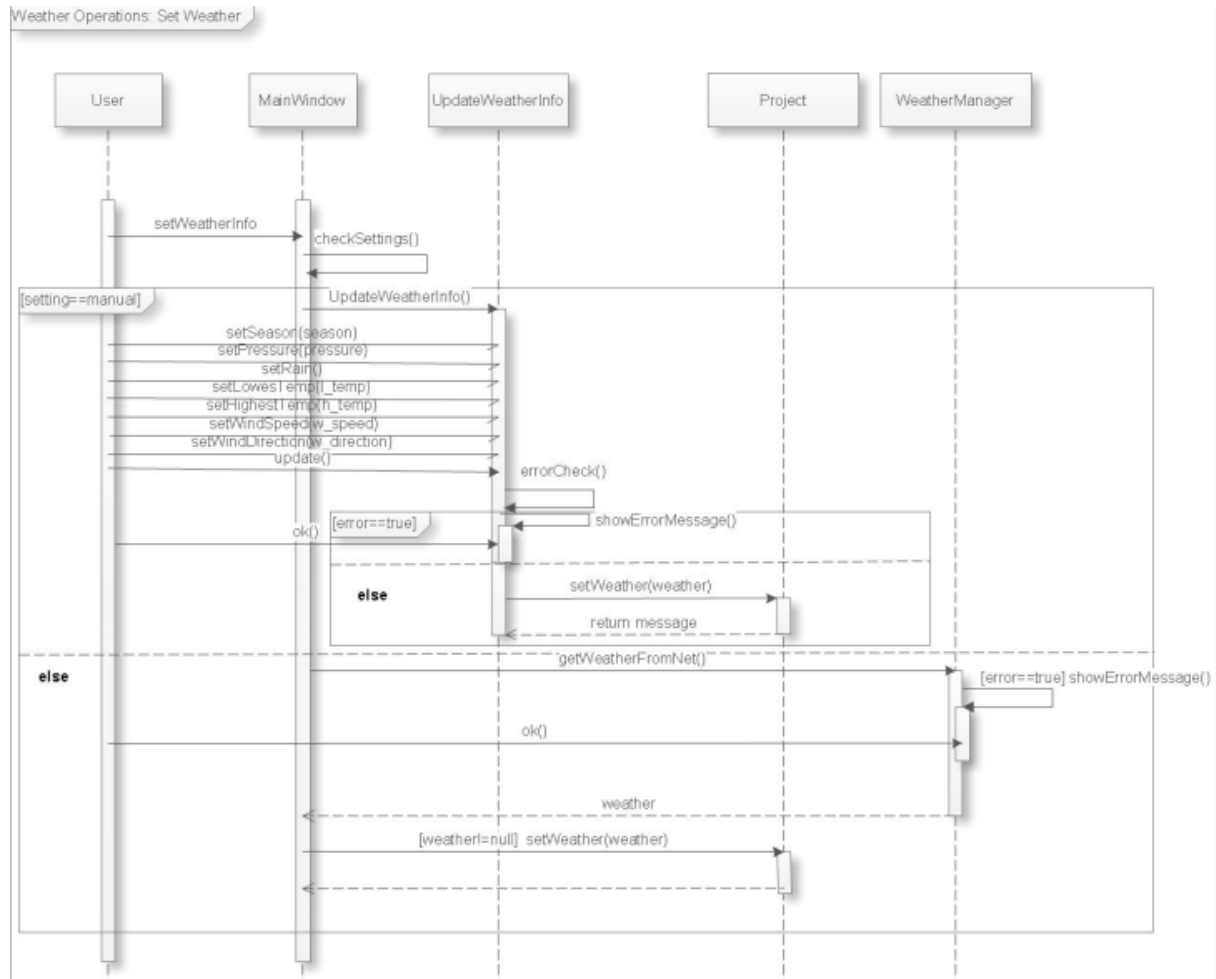
### 6.6.7. Weather Operations

#### 6.6.7.1. See Weather Information

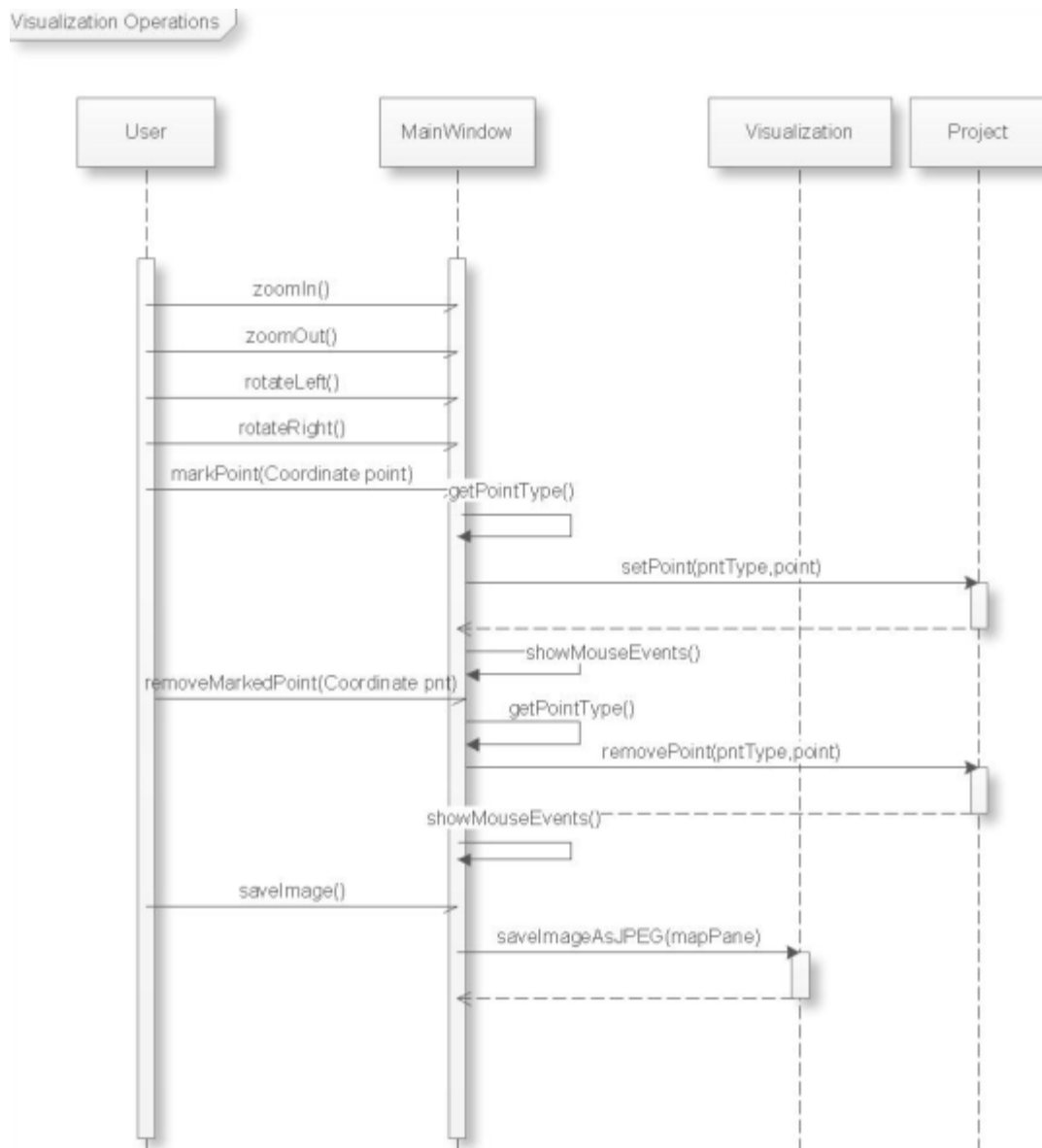
Weather Operations: See Weather Info



### 6.6.7.2. Set Weather Information



### 6.6.8. Visualization Operations



## 7.CONCLUSION

This document is a detailed summary of the design approach taken by the Sirius Software team for the Strider project. In this document, the revision of the project specifications and the changes to the architecture is stated. The details of the specifications and the algorithms are explained throughly with figures and pseudocodes. Moreover, this document provides quite large amount of elaboration on the technical design of the project. The architecture of the system is analyzed pointing out most of the details that concerns the programmers and system designers. The structure and the behavior of the system modules are discussed with detailed class and sequence diagrams. Other aspects of the project are also reviewed such as library and tool research.

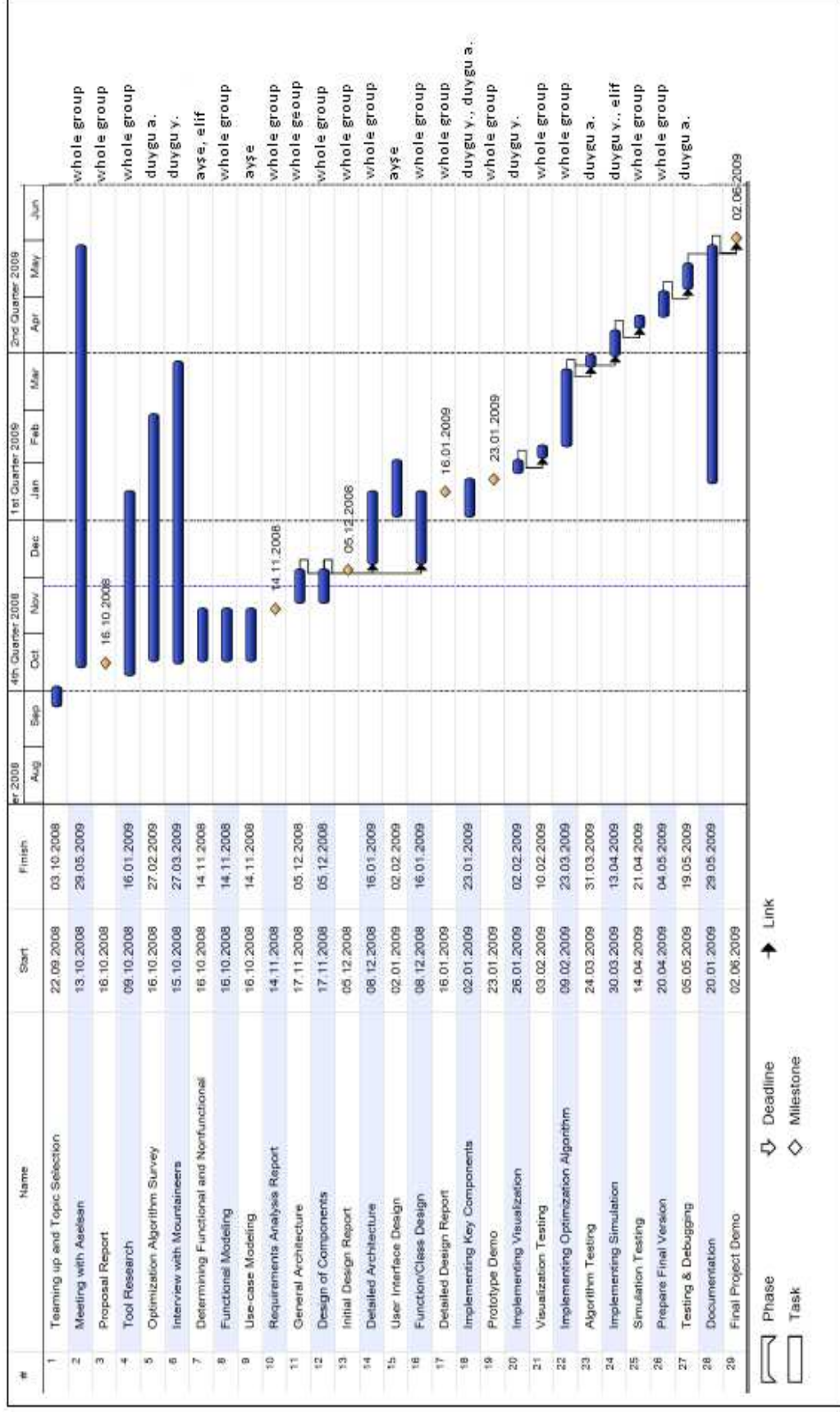
With the high level of detail shown in the document, the design of the system can be considered as almost finished. The next step will be starting the development phase of the project based on the design presented in this report. The report is expected to have only necessary changes during the development.

This report helped us to become familiar with software engineering and software design concepts and made us face the difficulties that a development team may have to overcome to make a sound and deep design of a large scale software project. Moreover, it serves as a starting point for the development phase since it provides an understanding of what we have to code and implement.

In conclusion, Sirius Software has designed the whole Project in detail in order to be ready for the implementation phase. In the following duration of five months our team will concentrate on the coding and testing process.



## 8. APPENDIX



## 9. REFERENCES

- [1]. Java Programming Language,  
[http://en.wikipedia.org/wiki/Java \(programming language\)](http://en.wikipedia.org/wiki/Java_(programming_language))
- [2]. Eclipse Platform, [http://en.wikipedia.org/wiki/Eclipse \(software\)](http://en.wikipedia.org/wiki/Eclipse_(software))
- [3]. Netbeans IDE, <http://www.netbeans.org/features/4>
- [4]. GeoTools, Open Source Java GIS Tool, <http://geotools.codehaus.org/>
- [5] Extensible Markup Language, <http://www.xml.com/>
- [6] MySQL, <http://www.mysql.com/>
- [7] User Interfacers Desing Tips, Techniques, and  
Principles: <http://www.ambyssoft.com/essays/userInterfaceDesign.html>
- [8] Ant colony optimization , Cambridge, Mass. : MIT Press, 2004 , Marco Dorigo, Thomas  
Stèutzle
- [9] Evolutionary Algorithms in Engineering Applications, Springer New York 2006, Dasputa,  
Michalewicz
- [10] “Zirvelerin Özgürlüğü” by Don Graydon, Kurt Hanson
- [11] Tunç FINDIK