NEA TECHNOLOGIES

FINAL DESIGN REPORT

Authors

Erdal Sivri
Ersegün Şentürk
H. Hüseyin Topçu
Ömer Yanar

19.01.2009
# Table of Contents

1. **Introduction** ................................................................................................................................................. 3  
   1.1 Purpose.................................................................................................................................................. 3  
   1.2 Problem Definition ................................................................................................................................. 3  
   1.3 Design Constraints ................................................................................................................................. 4  
      1.3.1 Portability Constraints .................................................................................................................. 5  
      1.3.2 Performance Constraints ............................................................................................................. 5  
      1.3.3 Time Constraints .......................................................................................................................... 6  
      1.3.4 Financial Constraints .................................................................................................................. 6  
      1.3.5 Hardware and Software Constraints .......................................................................................... 6  
2. **Use Case Analysis** ............................................................................................................................................ 7  
   2.1 User Management ................................................................................................................................. 7  
   2.2 Team Management ............................................................................................................................... 8  
   2.3 Geographic Information Gathering ..................................................................................................... 9  
   2.4 Activity Planner ................................................................................................................................. 10  
   2.5 Simulation ......................................................................................................................................... 11  
   2.6 Reporting ..................................................................................................................................... 12  
3. **Data Design** ................................................................................................................................................ 13  
   3.1 Entity Relationship Diagram ............................................................................................................. 13  
4. **Architectural Design** .................................................................................................................................... 14  
   4.1 Detailed Module Explanation .............................................................................................................. 14  
      4.1.1 User Management ....................................................................................................................... 14  
      4.1.2 Team Management .................................................................................................................. 15  
      4.1.3 Geographic Information Gathering .......................................................................................... 15  
      4.1.4 Activity Planner ....................................................................................................................... 16  
      4.1.5 Simulation ............................................................................................................................... 17  
      4.1.6 Reporting .............................................................................................................................. 18  
   4.2 Module Design ................................................................................................................................... 19  
      4.2.1 User Management ...................................................................................................................... 20  
      4.2.2 Team Management .................................................................................................................. 22  
      4.2.3 Geographic Information Gathering .......................................................................................... 23  
      4.2.4 Activity Planner ....................................................................................................................... 26  
      4.2.5 Simulation ............................................................................................................................... 36  
      4.2.6 Reporting .............................................................................................................................. 39  
   4.3 Data Flow Diagrams ............................................................................................................................. 40  
      4.3.1 Level 0 – Context Level .............................................................................................................. 40  
      4.3.2 Level 1 – Climb Plan .................................................................................................................. 40  
      4.3.3 Level 2 ................................................................................................................................... 42  
      4.3.4 Data Dictionary ........................................................................................................................ 49  
5. **Procedural Design** ...................................................................................................................................... 50  
   5.1 State Diagram .................................................................................................................................... 50  
   5.2 Sequence Diagrams ............................................................................................................................. 51  
6. **User Interface Design** ................................................................................................................................. 54  
7. **Project Schedule** ...................................................................................................................................... 68  
   7.1 Schedule of the Project ....................................................................................................................... 68  
   7.2 Gantt Chart of the Project ................................................................................................................... 70
1. Introduction

CLIMBPLAN Software is intended to be used by mountaineering clubs or organizations, which need careful planning for their mountaineering activities. Climb Planner is expected to help mountaineers organize their climbing activities more effectively and make their planning more accurately and easier.

1.1 Purpose

This document depicts how CLIMBPLAN Software will be structured to satisfy the requirements identified in the Software Requirements Specification document prepared by NEA Technologies in 14.11.2008.

Requirements Specification document determines software, hardware, functional and non-functional requirements decided to be satisfied and gives a general idea how the system will work. This document covers the details and different aspects of the project in a comprehensive way and conceptualizes the overall product that will be formed accurately.

In the design process, it is intended to design an effective and modular product that will satisfy the needs and constraints of the project. It is also aimed to explain the functional, structural and behavioral features of the system by using specific types of UML diagrams such as class, sequence, use case and dataflow diagrams.

1.2 Problem Definition

As the technology evolves, usage of computers in daily life has been increased remarkably. From science to entertainment, healthcare to military, education to social life; people benefits from computational power of these machines. That means, whenever the complex calculations and complicated problems arise, computers involve in.

Mountain climbing, being a dangerous sport activity, requires careful planning and preparation, which is a complicated problem that may need a complex solution. There are plenty of unknowns to be considered such as the route to be followed, expected duration of the climbing, required equipments and resting locations before deciding to climb. Determining these unknown variables is very important to mountaineers as they play a significant role in providing safety.
Limited characteristic of traditional tools available for mountaineers is another issue. For example, geographical maps are main assistants to mountaineers as a visual tool demonstrating the climbing area. Detail level and accuracy of these maps are crucial for mountaineers in order to make a climbing plan and compute the route to be followed. However, conventional maps printed on papers have a restricted capability of indicating the area from only one aspect, usually from bird’s eye view. They have also a restriction of detail level because of the trade-off between accuracy and map dimensions. Being an outdoor activity, large dimensions of these maps is a problem for mountaineering due to lack of mobility.

On the other hand, computer programs can visualize a three-dimensional map from any aspect desired. They can indicate not only the width and length information about the geographic constitutions, but also height or depth values of them. There are traditional maps which are capable of demonstrating the depth or height information, usually by different colors at different altitudes, but they do not have a realistic view compared to images generated by computers. Furthermore, they still lack of ability to show an area from different views.

In addition, mobility is not a problem for computer programs; it is only restricted by the mobility of the computer they run. If it is taken account that mobile computers (laptops, PDA’s…) are becoming smaller and smaller, it can be foreseen that the importance of mobile computer applications will increase. Moreover, digital maps prepared for computers do not have a trade-off between accuracy and map dimensions if the program which will use them has zooming capability. The maps can be prepared in the largest dimensions possible and user can decide the level of detail needed.

As a result, it is a formidable task to plan a mountain climbing activity because it requires careful examination of multiple variables. Although humans can handle planning a mountaineering activity, with the help of computers better results can be achieved faster. Moreover, graphical capabilities of the computers can help climbers get familiar with the environment before physically going there.

1.3 Design Constraints

In the design process of a software development, it is inevitable to confront with some constraints which affect this process remarkably. Most important and restrictive ones are listed below.
1.3.1 Portability Constraints

CLIMBPLAN Software is aimed to be used in different kinds of operating systems running on different kinds of computers. Therefore, portability is an important constraint that affects design process.

There is a limited number of programming languages that have portability feature. Most of these languages are also restricted by some proprietary libraries, which mean portability level will be decreased if these libraries are used. Java has a special place in the programming languages whenever the portability issue involves. Many think that it has the most comprehensive portability level by having the ability of running everywhere that has Java Virtual Machine.

As a result, CLIMBPLAN Software will be developed in Java programming language. It is also decided to follow Sun Microsystems Code Conventions in order to form a more extensible product by agreeing with this standard.

1.3.2 Performance Constraints

Performance is not an essential constraint for CLIMPLAN Software because once a simulation or report obtained from the system; it can be used at other times. In other words, there is no need to make recalculation with the same map and parameters. However, obtaining this first simulation or report should not take much time.

The most complex calculations made by the system will be in the activity planner module, which should compute the requested path and generate the required outputs for the climbing activity in a reasonable amount of time. Since this is an interactive program, users should not be waited for a long time. As a recommendation, the path analyzing computation shall take at most twenty to forty seconds.

Simulation module is also important according to performance point of view, which has real time requirements. The system should display the simulation in the suggested operating hardware without feeling any sluggishness in motion. The details of the hardware constraints are specified in the Hardware Constraints.
1.3.3 Time Constraints

The schedule of the project is determined beforehand and from now on, it will take about six months to finalize the project successfully. Working on the prototype and preparation of it will be handled concurrently and it should be accomplished in one and a half month. Until now, all group members have been working together in harmony. Since the main implementation of the project will be held in the second term, a task distribution and the needed time duration for these tasks are stated in the project schedule.

The detailed project schedule and Gantt chart of the project is given in the Project Schedule section.

1.3.4 Financial Constraints

CLIMBPLAN Project does not need any embedded system or other device required to be developed. Therefore, it remains only that some developing software will be necessary.

Since Java SDK will be used as a compiler and it is free of charge, there is no financial constraint according to compiler part. MySQL Database Server will be used for database applications, which means there is also no expenditure for database part because MySQL is also free for non-commercial products. There are also plenty of open-source and free GIS software in the market which have sufficient quality if the big prices of commercial counterparts are taken into account. Thus it is decided that uDig and NASA WorldWind will be used, both of which are open-source and free.

1.3.5 Hardware and Software Constraints

Although the system is expected to run at every device that has Java Virtual Machine installed, the following hardware technologies are recommended for good performance:

- Windows 2000, XP or later
- Intel Pentium 3, 1 GHz, or AMD Athlon or higher
- 256 MB of RAM
- 3D Graphics Card
- 2 GB of disk space

It is also required that the device should have following software installed:

- Java SE Runtime Environment 1.5 or higher
- MySQL Database Server 5.0 or higher
2. Use Case Analysis

2.1 User Management

- Organizer login to the system by clicking “Click to login as an organizer” hyperlink. A new window appears and asks username and password. Organizer clicks “OK” button for confirmation or “Cancel” button for cancellation.

- Organizer registers a new user who can be either mountaineer or organizer by clicking “File / New / Mountaineer” menu item. A new tab appears and organizer fills the required fields. Organizer clicks “Save” button to complete registration or “Close” button to cancel registration and exit.

- Organizer edits the properties of a selected user by editing textboxes of the current user tab and clicking “Save” button.

- Organizer displays properties of registered users by clicking over the “Mountaineers” tab. A new tab appears and shows the properties of the current user.

![User Management Use Case Diagram]

Figure 1. User Management Use Case Diagram
2.2 Team Management

- Organizer creates a team by clicking “File / New / Team” menu item. A new tab including registered users in a combo box appears. Organizer selects a user and clicks “Add” button. The selected user is added to a list showing members. If a particular user is added to the team, it cannot be selected anymore. Organizer clicks “Save” button to complete creation or “Close” button to cancel creation and exit. If fewer than two users are added and “Save” button is clicked, an error message is displayed.

- Organizer edits an existing team by editing textboxes of the current team tab and clicking “Save” button.

- Organizer displays a particular team by clicking over the “Team” tab. A new tab appears and shows the members of the selected team.

![Team Management Use Case Diagram](image)

**Figure 2. Team Management Use Case Diagram**
2.3 Geographic Information Gathering

- Mountaineer takes map from either a local map file or from internet using Web Map Service or Web Feature Service by clicking “File / New / Map” menu item. A map wizard appears and allows user to select map source. The selected map is stored to GIS Database automatically.

Figure 3. Geographic Information Gathering Use Case Diagram
2.4 Activity Planner

- Mountaineer sets the team participated to the activity by clicking “Activity / Set Team” menu item. A new window including registered users in a combo box appears. Mountaineer selects team and clicks “OK” button.
- Mountaineer sets the map indicating the location of the activity by clicking “Activity / Set Map” menu item. A new window including current maps in a combo box appears. Mountaineer selects map and clicks “OK” button.
- Mountaineer sets the parameters of the activity by clicking “Activity / Set parameters” menu item. A new window appears and allows user to set parameters of the activity.
- Mountaineer simulates the current activity by clicking “Activity / Simulate” menu item. A new “Simulation” tab appears and allows user to simulate activity with current team, map and parameters.

Figure 4. Activity Planner Use Case Diagram
2.5 Simulation

- Mountaineer starts the simulation by clicking “Start |>” button. An arrow showing the computed path is drawn slowly.
- Mountaineer pauses the simulation by clicking “Pause ll” button. Mountaineer resumes the simulation by clicking “Pause ll” button again.
- Mountaineer zooms in or out by clicking “+” or “-” buttons, respectively. The current window showing the map acts accordingly.
- Mountaineer adjusts speed by changing the value of “Speed” text box. Simulation speed adjusted automatically.
- Mountaineer takes screenshot of the current window by clicking “Take Screenshot” button. The current window is loaded to the memory and a new window asking the location where the file is saved appears. Mountaineer selects a folder and clicks “OK” button.

![Simulation Use Case Diagram](image)

Figure 5. Simulation Use Case Diagram
2.6 Reporting

- Mountaineer chooses the file type (doc, pdf or txt) from the File Format combo box and clicks “Export” button. A new window appears and asks the location where the file is saved. Mountaineer selects a folder and clicks “OK” button.

![Diagram showing the process of exporting a file](image)

Figure 6. Reporting Use Case Diagram
3. Data Design

3.1 Entity Relationship Diagram

Figure 7. Entity Relationship Diagram
In our system each organizer is also assumed to be a mountaineer. It is represented by an ISA hierarchy. Organizer has the responsibility of creating user and team by login into CLIMBPLAN system. A team can be created by exactly one organizer and a team should involve at least one mountaineer.

For storing maps, we’ll use DBMS (Database Management System) with GIS and spatial information extension like MySql, PostgreSql. It is stored by its name with spatial geometric information.

4. Architectural Design

4.1 Detailed Module Explanation

4.1.1 User Management

User Management Module basically deals with the users of the system, namely mountaineers and organizers. Mountaineers are the members of the mountaineering clubs and organizers are the administrators of the system who arrange climbing teams, choose mountaineers for a specific climbing activity and register new mountaineers or organizers to the system. Registration process is always performed by organizers. Organizers will store all the information about mountaineers to the system. Organizers can register themselves to the system. There can be more than one organizer in the system and each organizer has their username and password. The system authenticates organizers by requesting user names and passwords of the organizers.

Mountaineers have many characteristics and separate skills according to each other. Our aim is to identify each mountaineer, analyze their skills and to keep information about them. All the information will be kept in a relational database. User Management Module categorizes the users of the system by their roles: Organizers and Mountaineers. An organizer is the leader for a team and mountaineers are regular users who can plan a climbing plan for a pre-defined team. However, only organizers can create/register users and manage teams.

User Management Module stores the following characteristics of the mountaineers to the database:

- Identity information of mountaineers
The system stores first name, last name, gender, birth date, e-mail and address of the users/mountaineers.

- Speed of the mountaineers in meter per hour
  
  *Mountaineers have two types of speed, namely walking speed and climbing speed.*
  
  *Speed of a mountaineer is inversely proportional to the slope of the area.*

- Resting time of mountaineers in minutes

### 4.1.2 Team Management

Team Management Module enables organizers to create teams by selecting mountaineers and shows the characteristics of the existing teams textually so that users can select a team for climb activity planning easily. Organizers determine a name for the teams they create since a unique name is required to reference a team later.

Information of the each team such as team name and members of the team will be stored in a relational database. When required, team speed is calculated as the speed of the slowest team member. Similarly, resting time of a team is calculated according to the longest resting time need of the team members.

Team Management Module presents each team’s characteristics, the characteristics of members. Organizers can consider this information while deciding on which team to select for a climbing activity.

### 4.1.3 Geographic Information Gathering

Geographic maps are the main inputs of the project. The system uses the maps with DTED, DEM and GEOTIFF file formats as inputs. The system will use two kinds of sources in order to get maps. Local sources and internet services are the main sources of the system while getting geographic maps as inputs.

In GIS applications, there are two standards which are mostly used, namely Web Map Service and Web Feature Service. There are another standards being used in GIS applications such as Catalogue Service and Coordinate Transformation Service.

Web Map Service uses a HTTP interface for requesting map images from distributed geospatial databases. It sends the outputs in raster formats. The response to the request is map images in JPEG, PNG and etc. file formats that can be displayed in a browser. A Web Map Service can basically do the following:
• Produces maps
  
  *As a picture, as a series of graphical elements or as a packaged set of geographic feature data*

• Response basic queries about the content of the map

By using Web Map Service, the users of the system can get graphical and textual data.

Web Feature Service uses the GML (Geography Markup Language) format. Web Feature Service allows a client to perform data manipulation operations on a set of geographic features. Data manipulation operations have ability to:

• Get or query features
  
  *Based on spatial and non-spatial constraints*

• Create a new feature

• Delete a feature

• Update a feature

Database is very important for the Geographic Information Gathering Module, since the system stores and fetches the maps from the relational database. The system supports raster data formats, because it will use Web Map Service Standard. A raster dataset can be served directly with WMS capabilities.

### 4.1.4 Activity Planner

Activity Planner Module is the main part of the system, all the computation operations will be held by this module. AI (Artificial Intelligence) and optimization algorithms will be used in this module. Activity Planner Module basically does the following operations:

• Determine the required equipments for the climbing activities

  *Equipment list can be flexible according to the climbing activity. Weather conditions, team's overall characteristics and climbing area are such factors affect the required equipments. Activity Planner will have a built-in logic to generate equipment list. In other words, equipments will not be stored in a database but will be hard-coded in the Activity Planner module.*

• Determine the camping locations of the climbing activity
Activity Planner decides the camping locations and camping durations of the climbing activity. Choosing camping locations and specifying the resting times at these locations is an important part of the activity plan.

- Calculate the duration of the climbing activity

Activity Planner computes the resting times at camping locations and the time required reaching to destination location. As a result, it calculates the overall duration of the climbing activity in hours.

- Compute route

Activity Planner Module computes a route according to criteria determined by the user: fastest/safest scale. This scale is in the range 1 to 10. A value of 10 means fastest possible route will be computed. Fastest route computation tries to minimize the duration of the activity considering the walking and climbing speed of the team (the slowest member determines these parameters). Safest route, however, tries to minimize the height difference between consecutive points on the route.

4.1.5 Simulation

Simulation Module is important for visual presentation. The computed path, check points and camping locations will be included in the 3D simulation.

Simulation Module basically does the following operations:

- Computed route is displayed on the map.
- Check points are highlighted on the map.
- Camping locations on the routes are shown with animated arrows.
- Zooming functionality is provided.
  
  *Users can zoom in or zoom out by using the mouse or keyboard.*

- Users will be able to view the map with different perspectives.
  
  *There will be different camera positions and from these camera positions users can view the map.*

- The speed of the simulation can be adjusted in a scale of [1-10].
- The simulation can be paused and stopped upon user request.
4.1.6 Reporting

The system provides a textual presentation of climbing activity plan. Reporting Module allows user to specify the format of the report. PDF (Portable Document Format), DOC and text formats are the primary formats supported. Users of the system can choose any of these formats and save the report file to his/her computer.

Reporting Module reports the following information:

- Computed route is displayed on a 2D image of the map.
- Required equipments of the climbing activity
  According to the climbing and weather conditions, the required equipments and their amounts can change.
- Camping locations
  Camping location names, coordinates and camping duration, that is, resting times at each camp location are reported by this module.
- Duration of the climbing activity
  Reporting Module reports the starting and finishing times of a climbing activity and the duration of the climbing activity.
4.2 Module Design

In module design process, Object Oriented Approach is employed. Each module has its own classes and these classes together with the relations with each other are shown in the class diagrams. Variables, methods, and the parameters of these methods and the return types can be seen from these class diagrams. Some functions are also depicted in pseudocode format in order to guide implementation process.

One of the most vital classes is DataAccess class which is not a part of only one module. It is used in User Management, Team Management and Geographic Information Gathering module. The class diagram is depicted below.

Data Access Class Diagram
4.2.1 User Management

User Management Module is responsible for registering and managing users of the system. It performs basic database operations such as adding, deleting and updating entries. Some of these entries are just supplying personal information about the user whereas some of them will be used by other modules in the calculations.

```
function addToUserDatabase(Mountaineer mountaineer)
    if userDatabase.canAppendRow()
        userDatabase.appendRow()
        sendToDatabase()
    else
        errorMessage("Database Error")
```
function **updateUserDatabase**(Mountaineer mountaineer)
    boolean confirmed = askConfirmation("Are you sure you want to update?")
    if(confirmed)
        userDatabase.find(mountaineer.getID)
        userDatabase.update()
    else
        return

function **deleteFromUserDatabase**(Mountaineer mountaineer)
    boolean confirmed = askConfirmation("Are you sure you want to delete?")
    if(confirmed)
        userDatabase.find(mountaineer.getID)
        userDatabase.delete()
    else
        return

void **changePassword**(String newPassword)
    boolean confirmed = askConfirmation("Are you sure you want to change your password?")
    if(confirmed)
        userDatabase.update()
4.2.2 Team Management

This module is used for creating new teams and managing them. Teams created by this module are used by the Simulation and Reporting modules to provide main features of the system.

```
function addMountaineer(int id)
    if(userList.contains(id))
        errorMessage("User is already in the team")
    else
        userList.add(id)
        teamDatabase.update()

function deleteMountaineer(int id)
    if(!userList.contains(id))
        errorMessage("No such user in the team")
    else
        userList.remove(userList.indexOf(id))
        teamDatabase.delete()
```

Team Management Class Diagram
4.2.3 Geographic Information Gathering

GeographicInformation Class represents the Geographic Information Gathering Module of the system. It heavily uses the GeoTools library as a GIS library. There exist class diagrams of the GeographicInformation Class and World Coordinate Class below.

Geographic Information and World Coordinate Class Diagram

In GeographicInformation class, the basic and vital operation is storing a map in the database. In order to store the map, the below method is defined. The Map which is encapsulated by this class will be stored with the given name.

```java
function storeMap(String name)
if (Map != null )
    db = new DataAccess()
    db.insertMap(Map)
```

The stored map can be updated by the below method with the name of map and object itself.

```java
function updateMap(String name)
    if (Map != null)
        db = new DataAccess()
        db.updateMap(name, Map)
```

The following four methods will be able to load the map into system from different sources namely local sources (e.g. Database, Local path), WMS and WFS.

```java
function loadMapFromDB(String name)
    db = new DataAccess()
    Map = db.getMap(name)

function loadMapFromLocalPath(String localPath)
    file = new File(localPath)
    reader = new mapReader(file) // e.g. GeoTiffReader in GeoTools
    Map = reader.read() // returns GridCoverage2D

function loadMapFromWMS(String url, String name)
    // WMSDataStore in GeoTools
    connection = WMSDataStore.getWMSConnection(url)
    if (connection = successful)
        Map = connection.read(name)

function loadMapFromWFS(String url, String name)
    connection = WFSDataStoreFactory().createNewDataStore(url)
    if (connection = successful)
        FeatureReader = connection.getFeatureReader()
        Feature = FeatureReader.read(name)
        Map = Transform(Feature)
```

There are some coordinate systems which introduce their own mathematical rules. So this method enables user to change the coordinate system. CoordinateReferenceSystem class in GeoTools enables the developer to work with CoordinateSystem types (Horizontal, Vertical, Geocentric and etc.)
function `changeCoordinateSystem` (CoordinateReferenceSystem `cs`)

```java
CoordinateSystem = cs
```

This method will return the height in world coordinate system. So the map (GridCoverage2d type in GeoTools) needs to be transform from its coordinate reference system to world coordinate system.

```java
function double `getHeight`(double `x`, double `y`)
if (Map != null && CoordinateSystem!=Null)
    mathtransform = Map.getGridGeometry().getGridToCRS(CoordinateSystem)
    point = GeometryFactory().createPoint( new Coordinate( (double) x, (double) y ) );
    worldPoint = Transform(point,mathtransform) // JTS in GeoTools
    return worldPoint.y
```

This method will return the world point of selected point of 2D map in world coordinate system.

```java
function WorldCoordinate `getWorldCoordinateFromMap`(double `x`, double `y`)
if (Map != null && CoordinateSystem!=Null)
    mathtransform = Map.getGridGeometry().getGridToCRS(CoordinateSystem)
    point = GeometryFactory().createPoint( new Coordinate( (double) x, (double) y ) );
    worldPoint = Transform(point,mathtransform) // JTS in GeoTools
    return worldPoint
```
4.2.4 Activity Planner

This module has five main classes responsible for the correct operation of the module. There are, of course, other classes, but they are generally used to hold data to be transferred across main classes of this module and the classes of other modules.

Main classes are:

- Activity Planner
- Path Planner
- Timing Planner
- Camping Planner
- Equipment Planner

Each of the classes except Activity Planner implements interfaces to provide an extensible architecture. Activity Builder class provides default implementations of these interfaces via static methods so that Activity Planner can plan the climbing activity using these classes. More complex algorithms can be used in the module by changing Activity Builder class or by implementing a dynamic loading behavior to this class. Following figures depict the diagrams of all the classes utilized in this module along with pseudo code that describes how the methods of the classes work.
4.2.4.1 Activity Planner

```python
function activity_plan(team, map, criteria)
    route = path_plan(team, map, criteria)
    timing_info = timing_plan(team, map, route)
    camping_info = camping_plan(map, route)
    equipments = equipment_plan(team, route)
    return {route, timing_info, camping_info, equipments}
```

Activity Planner Class Diagram
4.2.4.2 Path Planner

```
function path_plan(team, map, criteria)
    route.check_points = criteria.check_points
    route.map = map
    check_points = sort_check_points
        (criteria.check_points, criteria.start_point, criteria.end_point);
    for i from 0 to check_points.length by 1 % find a path between each two check points
        points += A*(check_points[i], check_points[i+1], team, map, criteria)
    route.coordinates = points
    return route

function sort_check_points(check_points, start_point, end_point)
    % sort the points using a library routine with a custom comparator
    sort(check_points, comparator)
    % insert start and end points to front and back respectively
```
% to have all the points in an array
check_points.insert_front(start_point)
check_points.insert_back(end_point)
return check_points

function comparator(point_a, point_b)
return euclidean_dist(point_a, start_point) > euclidean_dist(point_b, start_point)

function euclidean_dist(point_a, point_b)
    x_distance = point_b.x_coord - point_a.x_coord
    y_distance = point_b.y_coord - point_a.y_coord
    z_distance = point_b.z_coord - point_a.z_coord
    return sqrt(x_distance^2 + y_distance^2 + z_distance^2)

function calculate_walking_speed(team)
return get_minimum(team.mountaineers, get_walking_speed)

function calculate_climbing_speed(team)
return get_minimum(team.mountaineers, get_climbing_speed)

function get_minimum(elements, get_value)
    min_value = infinity
    foreach element in elements
        if (get_value(element) < min_value)
            min_value = get_value(element)
    return min_value

function get_walking_speed(mountaineer)
return mountaineer.walking_speed

function get_climbing_speed(mountaineer)
return mountaineer.climbing_speed

function heuristic_estimate_of_distance(point_a, point_b, team, map, criteria)
x_distance = point_b.x_coord - point_a.x_coord
y_distance = point_b.y_coord - point_a.y_coord
z_distance = point_b.z_coord - point_a.z_coord
xy_distance = sqrt(x_distance^2 + y_distance^2)
% heuristic estimate of distance is the time required to
% walk/climb the distance between the given points
% Note that this is an under estimation as it is the shortest
% distance between any two points in 3-dimensional space
return (xy_distance / walking_speed) + (z_distance / climbing_speed)
function dist_between(point_a, point_b, fastest_safest)
  x_distance = point_b.x_coord - point_a.x_coord
  y_distance = point_b.y_coord - point_a.y_coord
  z_distance = point_b.z_coord - point_a.z_coord
  xy_distance = sqrt(x_distance^2 + y_distance^2)
  distance = sqrt(xy_distance^2 + z_distance^2)
  if (z_distance / xy_distance > SAFE_SLOPE_VALUE)
    climbing_speed = climbing_speed * (fastest_safest / 10.0)
  end
  return (xy_distance / walking_speed) + (z_distance / climbing_speed)

function A*(start, goal, team, map, criteria)
  closedset := the empty set % The set of nodes already evaluated.
  openset := set containing the initial node % The set of tentative nodes to be evaluated.
  g_score[start] := 0 % Distance from start along optimal path.
  h_score[start] := heuristic_estimate_of_distance(start, goal, team, map, criteria)
  f_score[start] := h_score[start] % Estimated total distance from start to goal through y.
  while openset is not empty
    x := the node in openset having the lowest f_score[] value
    if x = goal
      return reconstruct_path(came_from, goal)
    end
    remove x from openset
    add x to closedset
    foreach y in neighbor_nodes(x)
      if y in closedset
        continue
      end
      tentative_g_score := g_score[x] +
        dist_between(x, y, criteria.fastest_safest)
      tentative_is_better := false
      if y not in openset
        add y to openset
        h_score[y] := heuristic_estimate_of_distance
          (y, goal, team, map, criteria)
        tentative_is_better := true
      elseif tentative_g_score < g_score[y]
        tentative_is_better := true
      end
      if tentative_is_better = true
        came_from[y] := x
        g_score[y] := tentative_g_score
        f_score[y] := g_score[y] + h_score[y]
      end
    end
  end
  return failure

function reconstruct_path(came_from, current_node)
  if came_from[current_node] is set
\[ p = \text{reconstruct\_path(} \text{came\_from, came\_from[} \text{current\_node}]\text{)} \]
\[ \text{return } (p + \text{current\_node}) \]

\text{else}
\[ \text{return the empty path} \]

“A*” and “reconstruct\_path” methods are modified from Wikipedia article named "A* search algorithm”. The complete URL of the article is \text{http://en.wikipedia.org/wiki/A*}.

4.2.4.3 Timing Planner

```
function timing_plan(team, map, route)
    timing_info.expected_duration = calculateExpectedDuration(team, route)
    resting_locations = CampingPlanner.calculate_camping_locations(map, route)
    {timing_info.resting_durations, timing_info.resting_times} =
        calculate_resting(expected_duration)

    return timing_info
```
function calculateExpectedDuration(team, route)
    walking_speed = PathPlanner.calculate_walking_speed(team)
    climbing_speed = PathPlanner.calculate_climbing_speed(team)
    expected_duration = 0.0
    foreach point in route.coordinates
        x_distance = point_b.x_coord - point_a.x_coord
        y_distance = point_b.y_coord - point_a.y_coord
        z_distance = point_b.z_coord - point_a.z_coord
        xy_distance = sqrt(x_distance^2 + y_distance^2)
        distance = sqrt(xy_distance^2 + z_distance^2)
        if (z_distance / xy_distance > WALKING_SLOPE_VALUE)
            speed = climbing_speed
        else
            speed = walking_speed
        expected_duration += distance / speed
    return expected_duration

% A simple rule-of-thumb is used for calculating resting times/durations
% The following information is from ODTU Dagcilik Klubu Web Site:
% Genelde 50 dakikalık bir yürüyüşten sonra 10 dakikalık uzun bir
% mola verilmelidir. Çok dik ve zorlu etaplarda yarım saatte bir,
% 2 dakikalık bir nefes molası da verilebilir.

function calculate_resting(ref expected_duration)
    LONG_REST_TIME = (50*60)
    LONG_REST_DURATION = (10*60)
    SMALL_REST_TIME = (30*60)
    SMALL_REST_DURATION = (2*60)
    for i from 0 to expected_duration by 10*60
        if ((i+1) % SMALL_REST_TIME)
            resting_durations.add(SMALL_REST_DURATION)
            expected_duration += SMALL_REST_DURATION
        if ((i+1) % LONG_REST_TIME)
            resting_times.add((i+1) * LONG_REST_TIME)
            resting_durations.add(LONG_REST_DURATION)
            expected_duration += LONG_REST_DURATION
    return { resting_durations, resting_times }
4.2.4.4 Camping Planner

```java
function camping_plan(timing_info, route)
    resting_times = timing_info.resting_times
    walking_speed = PathPlanner.calculate_walking_speed(team)
    climbing_speed = PathPlanner.calculate_climbing_speed(team)
    i = 0
    current_time = 0.0
    foreach point in route.coordinates
        x_distance = point_b.x_coord - point_a.x_coord
        y_distance = point_b.y_coord - point_a.y_coord
        z_distance = point_b.z_coord - point_a.z_coord
        xy_distance = sqrt(x_distance^2 + y_distance^2)
        distance = sqrt(xy_distance^2 + z_distance^2)
        % if the slope is greater for walking then the distance
        % should be climbed
        if (z_distance / xy_distance > WALKING_SLOPE_VALUE)
            speed = climbing_speed
        else
            speed = walking_speed
        current_time += distance / speed
        if (current_time ~= resting_times[i])
            i++
            campLocations.add(point) % this is a camp location
    camping_info.campLocations = campLocations
    return camping_info
```
4.2.4.5 Equipment Planner

function equipment_plan(team, route)
    % equipment list is arranged according to the list
    % provided by the "Istanbul Bilgi Universitesi Dagcilik Klubu"
    % the web site can be reached from:
    % http://www.bilgidak.org/dagbilgi/malzeme.htm
    climbing_required = climbing_required(route)
    equipments += walking_camp_equipment(team, climbing_required)
    equipments += clothing_equipment(team, climbing_required)
    % other categories are processed accordingly...
    % equipments += kitchen_equipment(team)
    % equipments += accessory_equipment(team)
    % equipments += other_equipment(team)
    return equipments
function climbing_required(route)
    foreach point in route.coordinates
        $x$ = point_b.x_coord - point_a.x_coord
        $y$ = point_b.y_coord - point_a.y_coord
        $z$ = point_b.z_coord - point_a.z_coord
        $xy$ = sqrt($x^2 + y^2$)
        $distance$ = sqrt($xy^2 + z^2$)

    if ($z / xy > WALKING_SLOPE_VALUE$)
        return true
    else
        return false

function walking_camp_equipment(team, climbing_required)
    % select equipment required for walking
    % even though team may have to climb using tools
    % they need walking/camping equipments,
    % i.e. they are general equipments
    % walking and camping equipment list is available on
    % http://www.bilgidak.org/dagbilgi/malzeme.htm
    % under "Yuruyus ve Kamp Malzemeleri" header
    % for instance:
    walking_camp_equipments.add("Cakmak", team.mountaineers.count)
    % ...
    if (climbing_required = true)
        % select equipment required for climbing
        % climbing equipment list is available on
        % http://www.bilgidak.org/dagbilgi/malzeme.htm
        % under "Teknik Malzemeler" header
        % for instance:
        walking_camp_equipments.add("Tirmanis Ipi", team.mountaineers.count)
        % ...
    return walking_camp_equipments

function season_info()
    % provides simple season info via checking system date/time
    % later can be replaced by live weather information from a web service
    month = get_system_date()
    if (month in { December, January, February } )
        return Winter
    else if (month in { March, April, May } )
        return Spring
    else if (month in { June, July, August } )
        return Summer
    else
        return Fall
function clothing_equipment(team, season)
    % clothing list is available on
    % http://www.bilgidak.org/dagbilgi/malzeme.htm under "Giyim" header
    % select clothes required regardless of weather
    % for instance:
clothing_equipments.add("Pantalon", team.mountaineers.count)
    % ...
    if (season = Winter || season = Fall)
        % select clothes appropriate for cold weather
        % for instance:
clothing_equipments.add("Bere", team.mountaineers.count)
        % ...
    else
        % select clothes appropriate for hot weather
        % for instance:
clothing_equipments.add("T-Shirt", team.mountaineers.count)
        % ...
    return clothing_equipments

4.2.5 Simulation

The main class of the simulation module is SimulationEngine. The required simulations, animation, rendering and highlighting tasks are handled in this module.

Nasa World Wind Java SDK basically provides a simulation infrastructure to the SimulationEngine Class. CampingInfo, TimingInfo, and Route classes are used in the SimulationEngine Class. Moreover, Nasa World Wind camera classes support the camera operations that we want to implement.
On the above picture, the class diagram of the SimulationEngine Class is available.

Function \textit{adjustSpeed}(int speed) :
\begin{verbatim}
if(speed <= 10 && speed > 0)
    simulationSpeed = speed;
else if(speed == 0)
    stop();
else
    cout << “Speed out of range!” << endl;
\end{verbatim}
function start() :
    if(simulationSpeed == 0)
        startSimulationProcess(); // This function will start the simulation
    else
        cout << " It is already simulating now!" << endl;

function pause() :
    if(speed <= 10 && speed > 0)
        pauseSimulationProcess();  // This function will pause the simulation

function stop() :
    if(speed <= 10 && speed > 0)
        stopSimulation();  // This function will stop the simulation

function showCheckPoints() :
    for(int I = 0;I < route.checkPoints.size();I++)
        highlightCheckPoints(route.checkPoints[I]); // highlights the
        // check points one by one

function showRoute() :
    for(int I = 0;I < route.coordinates.size();I++)
        renderCoordinates(route.coordinates[I]); // renders the Coordinates

function showCampingLocations() :
    for(int I = 0;I < campingInfo.campLocations.size();I++)
        // animate camp locations
        animateCampLocations(campingInfo.campLocations[I]);

function render() :
    showRoute();
    showCampingLocations();
    showCheckPoints();
4.2.6 Reporting

Reporting Class works as in the same manner of the SimulationEngine Class. While SimulationEngine is handling 3D simulation, animation and highlighting operations, Reporting Class reports the information that kept in the Equipment, Route, TimingInfo and CampingInfo classes. In the below picture, it can be seen the class diagram of the Reporting Class.

```
function exportFile(string fileName, string fileFormat) :

    reportCampingInfo(campingInfo);
    reportTimingInfo(timingInfo);
    reportEquipmentList(equipments);
    reportRouteInfo(route);

    if(fileFormat == "doc")
        exportFileInDocFormat(fileName);
    else if(fileFormat == "pdf")
        exportFileInPdfFormat(fileName);
    else if(fileFormat == "txt")
        exportFileInTextFormat(fileName);
```
4.3 Data Flow Diagrams

4.3.1 Level 0 – Context Level

This diagram shows the operating environment of the CLIMBPLAN system by considering the program as one process which has three external entities: User, Local Sources and Web Service. The user supplies the required input such as his/her identity information, climbing criteria and preferences. If required, the system requests for map files from pre-defined web services or local sources. In other words, if the requested map does not exist in the database, the system looks for local sources or web sources. The system generates a simulation and an activity plan report as a result.

4.3.2 Level 1 – Climb Plan

This level one DFD diagram depicts the main system modules and the flow of data between them. There are six processes displayed in the diagram: User Interaction, User Management, Team Management, Geographic Information Gathering, Activity Planning and Reporting & Simulation.
User Interaction: This process is responsible for gathering user input and passing this data to appropriate processes. Identity data, climbing criteria, login info and preferences of user are the inputs of the process. It generally passes the data to other processes without any change with only exception being the preferences data which is divided into several objects and passed to appropriate processes.

User Management: User identity data, username, password and climbing criteria, taken from the User Interaction process, are stored in the database and password is queried from the database for user authentication. User authentication is required only for organizers because organizers have the privilege to manage teams whereas regular users, mountaineers, can only use existing teams to plan a climbing activity.

Team Management: Climbing teams are created and stored considering the user selection inputs and existing team information. Selected team data is passed to Activity Planning Process.

Geographic Data Gathering: Map data is gathered from the external world (local sources or web sources), stored to the database and passed to Activity Planning Process when requested.

Activity Planning: Map data and selected team, which are determined by user, are taken from Geographic Information Gathering Process and Team Management Process respectively and they are processed subject to user constraints. The final results are passed to Reporting & Simulation Process.

Reporting & Simulation Process: The products of Activity Planning are displayed textually and graphically according to file format specification and simulation speed parameter of the user.
4.3.3 Level 2

4.3.3.1 User Interaction
This process has three sub-processes: Get User Data, Handle User Preferences and Display. Get User Data interacts with the user and takes user input and forwards them to other
processes except the preferences data, which is passed to Handle User Preferences process and divided into its building blocks and passed to other processes by the Handle User Preferences process. The last process, Display process, is responsible for displaying the specified data to the user.
4.3.3.2 User Management
This process consists of three sub processes. User Registration stores identity data of users to the database. Domain Specific Data Registration takes mountaineering related information from the users and stores to the database. User Authentication provides security for the other modules which require permission to operate certain functions.

4.3.3.3 Team Management
This process creates teams and selects among them. According to user preferences, the created teams are stored to the database by the Create Team Process. Select Team Process
gets team data from the database and passes the data of selected team which is selected according to user’s team preference.

### 4.3.3.4 Geographic Information Gathering

Geographic information is gathered via two means. Get Local Sources Process gets physical map data from the local sources. On the other hand, Get Web Sources gathers data from web map services. Supply Geographic Data Process provides other modules with map data from local sources, web sources or from preloaded maps from the database. Note that map data from web sources are first stored to the database and then used which enables offline use of downloaded map data.
4.3.3.5 Activity Planning

This process has two sub-processes. Get Activity Input gets the data from other modules and external world and pass to the Compute Path Process, which does the actual job of computing the route. Compute Path Process has a built-in logic to decide on the equipments required for the planned activity. In other words, equipments required for mountaineering are hard-coded to it. Considering user constraints, Activity Planning generates the required equipment list and route, camping and timing information.
4.3.3.6 Reporting & Simulation
A text report and a graphical simulation are prepared by this process. Compile Data Process compiles the required data for the Prepare Report File and Simulate Processes. Simulation Engine Process generates a simulation based on the simulation speed parameter. Prepare
Report File gets the format of the report file from the user and passes the climb activity report file to the user. The format of the report file is specified in the Data Dictionary.
## 4.3.4 Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity data</td>
<td>Mountaineer object</td>
<td>Part of the Mountaineer object that holds the data related to identity of the user: first name, last name, gender, birth date, address and e-mail address</td>
</tr>
<tr>
<td>climbing criteria</td>
<td>Mountaineer object</td>
<td>Part of the Mountaineer object that holds the data related to climbing criteria of the user: walking speed, climbing speed and resting time</td>
</tr>
<tr>
<td>username &amp; password</td>
<td>A tuple of two Strings</td>
<td>Holds user name and user password</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>A string that holds username, a unique string for each member</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>A string that holds password</td>
</tr>
<tr>
<td>preferences</td>
<td>Composite data</td>
<td>A composite data containing user preferences. Which mountain, mountaineers and team to select, what simulation parameters and user constraints to consider, and what the format of the report is</td>
</tr>
<tr>
<td>mountaineer selection</td>
<td>String</td>
<td>The username of the mountaineer to be selected</td>
</tr>
<tr>
<td>team preference</td>
<td>String</td>
<td>The name of the team that is unique for each team</td>
</tr>
<tr>
<td>activity plan report</td>
<td>String</td>
<td>Name of a file that contains the details of the computed climbing activity plan</td>
</tr>
<tr>
<td>simulation</td>
<td>Display of visual simulation</td>
<td>The graphical simulation of the generated activity plan</td>
</tr>
<tr>
<td>report format</td>
<td>String</td>
<td>A string that holds the format of the activity plan report: DOC, PDF or TXT (plain text)</td>
</tr>
<tr>
<td>simulation speed</td>
<td>Integer</td>
<td>An integer number, between one and ten (ten is the fastest), that determines the speed of the simulation</td>
</tr>
<tr>
<td>user criteria</td>
<td>UserCriteria object</td>
<td>Criteria defined by the user: start and ending points, fastest/safest route scale [1-10] (10 means fastest route) and check points (the locations the route must pass)</td>
</tr>
<tr>
<td>mountain selection</td>
<td>String</td>
<td>The name of the mountain on which the climbing activity will be planned</td>
</tr>
<tr>
<td>required equipments</td>
<td>Array of Equipment objects</td>
<td>The list of equipments required for the climbing activity</td>
</tr>
<tr>
<td>camping</td>
<td>CampingInfo object</td>
<td>Contains camping locations</td>
</tr>
<tr>
<td>timing</td>
<td>TimingInfo object</td>
<td>Contains timing information: the expected duration of the activity, resting times and durations</td>
</tr>
<tr>
<td>route</td>
<td>Route object</td>
<td>Holds an array containing coordinates that define the route of the activity (a coordinate array)</td>
</tr>
<tr>
<td>selected team</td>
<td>Team object</td>
<td>An object containing team name and team members. It is a data structure for the database entity named Team</td>
</tr>
<tr>
<td>team data</td>
<td>Team object</td>
<td>Structurally equivalent to selected team</td>
</tr>
<tr>
<td>map data</td>
<td>Map object</td>
<td>An object containing Geographic data for a map. It is a data structure for the database entity named Map</td>
</tr>
<tr>
<td>compiled data</td>
<td>Composite data</td>
<td>A tuple consisted of the following items: Array of Equipment objects, CampingInfo, TimingInfo and Route objects used for passing data to sub-processes “Prepare Activity Report” and “Simulate”</td>
</tr>
</tbody>
</table>
5. Procedural Design

5.1 State Diagram

This diagram is used to show the behavior of the system describing all of the possible states. CLIMBPLAN will have several modules such as User Management, Team Management, Geographic Information Gathering, Activity Planner, Simulation and Reporting. Each module has distinct responsibility. The diagram shows a top-level view of system which combines the responsibilities of all modules defined.
5.2 Sequence Diagrams

In order to model the flow of logic within our system, we use sequence diagram which is a dynamic modeling technique. For System-level view of CLIMBPLAN, Creating New User, Creating Team and Simulation, following sequence diagrams are defined. The communication between actors is done by messaging.

Creating New User

This diagram depicts a sequence diagram for the “Register New User” use case in Figure 1 User Management Use Case diagram. The “Organizer” as an actor directly interacts with user interface. Furthermore, the “User Database” actor is accessed by an instance of DataAccess Class.
Creating Team

This diagram depicts a sequence diagram for the “Create Team” use case in Figure 2 Team Management Use Case diagram. The “Organizer” as an actor directly interacts with user interface. “MountaineerInfo” parameter within the “AddUserToTeam” message is provided from “Select User” use case in Figure 2 Team Management Use Case diagram. The “User Database” actor is accessed by an instance of DataAccess Class.

Simulation
This diagram depicts a sequence diagram for the “Start” use case in Figure 5 Simulation Use Case diagram. The “Mountaineer” which can also be an organizer directly interacts with user interface. An instance of Activity Planner computes the desired path and the coordinates of the computed path to Simulation Engine which renders the path in 3D.

**System-Level**

This diagram depicts a system-level approach where the interactions between the actors and the system are shown modeling flow of logic within the system. The actor here is “Organizer” who has the responsibility of creating user and team. The database is not depicted in this diagram, but it is accessed by an instance of DataAccess class.
6. User Interface Design

Users interact with the CLIMBPLAN System via a Graphical User Interface. This part explains how users interact with this interface or, in other words, what functionalities this interface present to users. The below figure, Figure 7, is a screenshot taken from a typical usage of the program. As is shown there are three main views:

**Top Left View:**

This area displays the currently created activities, which are objects that hold the necessary data to represent a climbing activity namely the name of the map, the name of the team that is planning to climb to the mountain and the criteria of the activity determined by the user. Users can create, edit, rename and delete activities via the File menu. However, it is noteworthy that these activity objects are not permanent objects; they are created for each session by the user. Actually, one activity object suffices for most cases; nevertheless, the program provides with such flexibility. It is often annoying to warn users by giving messages to them not to do something. Therefore, creating more than one activity is permitted.

**Bottom Left View:**

This area contains three entities: maps, mountaineers and teams. These are actually entities stored in a database and retrieved from that database during the load of the program. This area presents the required information in an easily-accessible and user-friendly way.

**Right View:**

This view is the most important part of the program because it both presents the selected entities from the left views in detail and allows users to create/display climbing activity plan report and to perform a simulation. It should be noted that this view is very dynamic. In other words, the tabs are opened and closed as needed which allows a more compact control of program. It is preferable over the classic “go to next form” method and allows users to do all their operations in a single window. As can be seen from the figure, there are several tabs each displaying an entity. The active tab views a mountaineer entity. Users can select a mountaineer from the bottom left view to inspect his/her further or update his/her information. All the tabs opened in this figure are enough to depict the functionalities of the whole graphical user interface. In addition, each tab is displayed separately following Figure 7.
This screenshot displays the overall functionalities of the user interface but it focuses on detailed Mountaineer view. All the stored data related to mountaineers are displayed here. In addition, if an organizer logs in to the system, he/she can change and update the data by clicking the Save button.

Organizers can register a new mountaineer to the system from the File menu (Details of the menu will be explained later). A new view will open on the right view displaying empty text areas to be filled by the organizer. After clicking the Save button, the new mountaineer will be added to the Mountaineers list in the bottom left view.
This screenshot shows a sample view of a map entity. Double clicking one of the items of the Maps tab from the bottom left view opens up a new view on the right view displaying the selected map. The start and end points of the route and any check points; that is any user-defined points that the route must pass, are determined by clicking on the map. To ease the user interaction, undo and redo mechanisms are provided via Edit menu, details of which will be explained later. It should be noted that the first point clicked will be the start point and the last point clicked will be the end point, most likely the peak of the mountain.
Similar to map and mountaineer entities, team entity can also be viewed in detail on the right view area via clicking a team name on the bottom left view. As a result of double clicking on the item named “A Takimi”, a new tab named “A Takimi [Team]” is opened as depicted in the above figure. The owner of the team can make changes to the team such as changing the name of the team and adding/removing mountaineers. A mountaineer can be selected from the combo box and clicking the Add button will add the selected user to the team. Note that the Add button is not visible because of the combo box items.
This figure displays an organizer, who is also a mountaineer but has additional attributes: user name and password. The figure below shows how an organizer can change his/her password. Clicking on the link button “Change Password?” will show two additional text boxes in which the organizer can enter his/her new password. A new organizer can be registered to the system by another organizer via the File menu. Clicking the File -> New -> Organizer item, a new view identical to the above view except that the text areas will be empty is opened in the right view and after entering the required data, clicking Save button will register the new organizer to the system.
Figure 11. Cem Yilmaz [Organizer] – Change Password

Figure 12. Main Screen – Hasan Dagi’na Yolculuk [Report]
The climbing activity plan report is displayed in the above view. The computed path is displayed on the map and a summary of the report is placed above the map. The complete report can be exported to a file with the specified format. Clicking the Export button opens up a file save dialog which is depicted on the next figure.

![Figure 13. Main Screen – Hasan Dagi’na Yolculuk [Report] – Export File](image)

This figure displays the file save dialog used to export the climbing activity plan report. As is seen from the screenshot, a default file name is generated by the program for the user.
The simulation is viewed on the right view area. Its speed can be adjusted using the numeric up down control. Control buttons are provided to start, pause or stop the simulation. Take Screenshot button is used to take a screenshot while the simulation is on process. Zooming functionality is supplied through the Zoom In and Zoom Out menu items in the Simulation menu. Mouse wheel can also be used to perform zooming.
Organizers are required to login to the system because they have the privileges normal mountaineers do not posses such as registering mountaineers and creating teams.

Figure 16. Bottom Left View - Maps, Mountaineers and Teams

This figure displays the bottom left view in detail. Entities in this view is displayed with minimum amount of information and double clicking on any entity opens up a detailed view of that entity on the right view.
In order to add a new map to the system, users can click on File -> New -> Map menu items. The above screen opens allowing users to select a source to import a map from.

Selecting “Local Files” and clicking on the Next button displays the above open file dialog, which lets users select a map file in a supported format from the file system.
These two separate dialog boxes are displayed when “Web Map Server” and “Web Feature Server” are selected from the New Map screen respectively. Both of the dialogs request a URL from the user to get map data or feature content from the web.

Figure 19. New Map Screen – Web Map Server and Web Feature Server

Figure 20. Set Criteria Dialog Box
Criteria dialog box opens when users double click a Criteria item on the Activities view or click Activity -> Set Criteria menu item. This dialog lets users determine the fastest/safest scale of the climbing activity.

![Figure 21. Menu - File](image)

File menu has eight items:

- **New**: Creates a new Activity, Map Mountaineer, Organizer and Team
- **Save**: Saves modified data on the focused tab of the right view. For instance, if an organizer adds a new member to a team he/she can update the team data by clicking this item.
- **Save All**: Saves all the modified tabs on the right view.
- **Rename**: Renames the selected item on any of the tabs on the left view. For example, clicking this item when an activity is selected changes the name of that activity.
- **Delete**: Deletes the selected item on any of the tabs on the left view. For example, clicking this item when an activity is selected deletes that activity.
- **Close**: Closes the focused tab of the right view.
- **Close All**: Closes all the tabs of the right view.
- **Exit**: Quits the program.
Edit menu contains six items:

- **Undo**: Rollbacks the change made on a map. As explained before when a user clicks on a map to specify start and end points and check points for the route, any unwanted clicks can be undone via this menu item.
- **Redo**: Redoes what undo operation rolls back.
- **Cut**: Cuts the selected text.
- **Copy**: Copies the selected text.
- **Paste**: Pastes the text on the clipboard to the text area
- **Select All**: Selects all the text on a text area.

Activity menu contains six items:

- **Create**: Creates a new activity by inserting a new item to the Activities view.
• **Set Map**: Assigns the selected map on the Maps tab on the bottom left view to the selected activity on the Activities view, which is on the top left area of the window.

• **Set Team**: Works the same way as the “Set Map” item to assign a team to the selected activity.

• **Set Criteria**: Opens the Criteria dialog introduced before.

• **Simulate**: Opens a new tab on the right view to display graphical simulation of the activity.

• **Create Report**: Opens a new tab on the right view to display climbing activity plan report.

Simulation menu has items that works as the duplicates of the simulation control buttons introduced before in the Simulation tab of the right view. In addition, Zoom In and Zoom Out menu items are used to perform zoom in and zoom out operations respectively.

Help menu contains two items: Contents and About. Contents item displays the user manual and About item opens the about dialog box of the program.
7. Project Schedule

7.1 Schedule of the Project

<table>
<thead>
<tr>
<th>Deadline Dates of Tasks</th>
<th>Description of Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Nov 17, 2008</td>
<td>Investigating GIS software</td>
</tr>
<tr>
<td>2 Nov 21, 2008</td>
<td>Investigation on raster and vector data</td>
</tr>
<tr>
<td>3 Nov 28, 2008</td>
<td>Examining GIS libraries, GDAL and GeoTools</td>
</tr>
<tr>
<td>4 Dec 5, 2008</td>
<td>Examining GIS Standards, WMS (Web Map Service) and WFS (Web Feature Service)</td>
</tr>
<tr>
<td>5 Dec 12, 2008</td>
<td>Examining GIS servers, Geo Server and Map Server</td>
</tr>
<tr>
<td>6 Dec 15, 2008</td>
<td>Initial Design Report</td>
</tr>
<tr>
<td>7 Dec 26, 2008</td>
<td>Analyzing uDig and NASA World Wind</td>
</tr>
<tr>
<td>8 Jan 9, 2009</td>
<td>Analyzing GIS Databases (MySQL, PostgreSQL)</td>
</tr>
<tr>
<td>9 Jan 16, 2009</td>
<td>Detailed Design Report</td>
</tr>
<tr>
<td>10 Jan 23, 2009</td>
<td>Prototype Demo</td>
</tr>
<tr>
<td>Deadline Dates of Tasks</td>
<td>Description of Tasks</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>11 Feb 7, 2009</td>
<td>Graphical User Interface Design</td>
</tr>
<tr>
<td>12 Feb 14, 2009</td>
<td>Mountaineer, Organizer and UserManagement Classes Implementation</td>
</tr>
<tr>
<td>13 Feb 20, 2009</td>
<td>Team and TeamManagement Classes Implementation</td>
</tr>
<tr>
<td>14 Mar 4, 2009</td>
<td>GeographicInformation and WorldCoordinate Classes Implementation</td>
</tr>
<tr>
<td>15 Mar 10, 2009</td>
<td>Completion of DataAccess Class Implementation</td>
</tr>
<tr>
<td>16 Mar 25, 2009</td>
<td>Route, CampingInfo, TimingInfo, Equipment and UserCriteria Classes Implementation</td>
</tr>
<tr>
<td>17 April 1, 2009</td>
<td>EquipmentPlanner and TimingPlanner Classes Implementation</td>
</tr>
<tr>
<td>18 April 21, 2009</td>
<td>CampingPlanner, PathPlanner, ActivityPlannerBuilder and ActivityPlanner Classes Implementation</td>
</tr>
<tr>
<td>19 April 30, 2009</td>
<td>SimulationEngine Class Implementation</td>
</tr>
<tr>
<td>20 May 5, 2009</td>
<td>Reporting Class Implementation</td>
</tr>
<tr>
<td>21 May 18, 2009</td>
<td>Integration of Modules</td>
</tr>
<tr>
<td>22 May 25, 2009</td>
<td>Preparation of Help Documentation</td>
</tr>
<tr>
<td>23 May 27, 2009</td>
<td>Optimization of the project</td>
</tr>
<tr>
<td>24 May 30, 2009</td>
<td>Prototype Demo</td>
</tr>
<tr>
<td>25 June 8, 2009</td>
<td>Testing</td>
</tr>
<tr>
<td>26 June 10, 2009</td>
<td>Final Package</td>
</tr>
</tbody>
</table>

The schedule of the project has changed according to the detailed design of the project. By specifying of many classes and the update on the deadlines of the tasks, the schedule changed.
## 7.2 Gantt Chart of the Project

### Gantt Chart - First Term

<table>
<thead>
<tr>
<th>Task</th>
<th>Q4-2008</th>
<th>Q1-2009</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>September</td>
<td>October</td>
</tr>
<tr>
<td>Group Creation and Teaming Up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Topic Selection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Topic Research</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposal Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Examining GIS in detail</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creating and Designing Modules of the Project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement Analysis Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detailed User Interface Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detailed Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype Demo</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Gantt Chart - Second Term

<table>
<thead>
<tr>
<th>Task</th>
<th>Q1-2009</th>
<th>Q2-2009</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>January</td>
<td>February</td>
</tr>
<tr>
<td>Graphical User Interface Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of User Management Module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of Team Management Module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of Geographic Information Gathering Module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of Activity Planner Module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of Simulation Module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation of Reporting Module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration of Modules</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preparation of Help Documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimization of the Project</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype Demo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Package</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>