CENG 491 FINAL DESIGN REPORT

MIDDLE EAST TECHNICAL UNIVERSITY COMPUTER ENGINEERING DEPARTMENT

CLIMB PLANNER

aiDSoft

TEAM NAME: AIDSOFT

PROJECT: CLIMBPLANNER

PROJECT ASSISTANT: ALİ ORKAN BAYER

PROJECT ENDORSER: ASELSAN

PREPARED BY:

İsmail PARLAK 1395409

Mustafa DİKİCİ 1394956

Ozan KEÇECİOĞLU 1395185

Yetkin SAKAL 1395425

Contents

1 INTRODUCTION 5
1.1 Motivation 5
1.2 Project Definition 5
1.3 Project Features 6
1.4 Purpose of Document 7
1.5 Design Constraints 8
2 USER INTERFACE DESIGN 9
2.1 Introduction 9
2.2 Main Window 9
2.3 File Menu 11
2.4 Create Project Tab 16
2.5 Weather Conditions & Constraints Tab 20
2.6 View Report Tab 22
2.7 Animation Menu 23
2.8 Add to DB Menu 25
2.9 Edit Climbers and Equipment Menu 26
3 GENERAL PROGRAM FLOW 27
4 ARCHITECTURAL DESIGN 28
4.1 Overall System Architecture 29
4.2 Level 0 Dataflow Diagram 30
4.3 Level 1 Dataflow Diagrams 32
4.4 Main Sequence Diagram 38
5 DETAILED DESIGN 39
5.1 Visualization Module Design 39
5.1.1 Data Structures 40
5.1.2 Module Design 42
5.2 AI Module 45
5.3 Map Reader Module 47

6 GANNT CHART 50
7 CONCLUSION 51

1 INTRODUCTION

1.1 Motivation

As a group of four senior computer engineering students, the AidSoft team aims to build a graphical tool to facilitate the analysis and preparation process of mountaineering clubs & societies.

Classified as a professional sports branch, a hobby or a profession, mountaineering embraces the aim to reach the top of a mountain regardless of its height. It is also noteworthy that the type of land the climbing action is performed varies remarkably as a result of different land & weather conditions. Hence, the discipline must be capable of overcoming the potential risks at analysis period in which all the precautions for the climb to be successful are taken.

The AidSoft team considers it of utmost importance to come up with a robust solution for mountaineers to input the conditions on which their climbing plan depends. The expected output of the software would then be an estimate on how long the climb will take and a number of potential climb paths where camping is available.

1.2 Project Definition

As the project name suggests, Climb Planner aims to provide the users with the ability to prepare their custom climbing environment and view the climbing path in a 3D visualization scheme.

In order for the mountaineers to adopt the system of analyzing a specific climbing path, the software possesses a number of instructions to help the

users do exactly what they want. In addition, the users are encouraged to navigate through the user interface to see what options they have and how wide the list of available input parameters they can add is. This would be a key factor in benefiting from the software as much as one can.

The system is made up of modular components to allow new maps to be added into the program. This feature not only enhances the capabilities of the software but also provides the users with the option to create custom maps and examine the subtle differences between the climbing paths generated.

1.3 Project Features

Features that will be included in the project can be given as:

- *3D Graphics:* To provide most realistic environment program will use 3D images in both visualization of mountain and animation.

- *Save & Load:* For user not to repeat same process again and again program have Save & Load function

- *Textual Report:* By providing textual report users will be able to have a printable form of whole climbing activity.

- *User Friendly GUI:* Since all mountaineers are not computer experts and program will obtain inputs in a wide range, designing an easy to use and functional GUI as much as possible is a key feature.

- *Easy Project Creating:* Users will be able to create their projects from ClimbPlanner database by just clicking. Additionally every climber and equipment defined by users will be saved in this database.

- *Easy Terrain Control:* User will be able to define starting point, end point and camping points by just clicking on the mountain image derived from loaded maps by user. Additionally since we are not expecting totally perfect mountain maps user will have a chance to modify this image again just by clicking.

1.4 Purpose of Document

Purpose of this document is to express design strategies necessary to accomplish the design activities associated with the project ClimbPlanner. This document explains how project is designed to meet requirements specified in requirement analysis report.

This document contains the following design decisions:

- GUI Design
- General Program Flow
- Architectural Design
- Detailed Design

1.5 Design Constraints

The project will operate under stress and tolerate unpredictable or invalid input. It is also designed with resilience to low memory conditions; compared to similar type of programs, to implement robustness.

Reliability is also an important constraint on software. Project is designed to execute deterministic. That is, for stated conditions and stated inputs during the execution, the program always executes same.

Designed modular components of the system capture the essence of functionality expected from them and no more or less. This design renders the components reusable wherever there are similar needs.

Due to strictly set deadlines for this project it becomes undeniable to make excellent scheduling analysis. Also our heavy course load is another factor that takes us under pressure. While preparing necessary reports for our project, we also have to work for on the prototype. However in order to complete this project we know we should meet the strict deadlines that were determined at the beginning of the term.

Since software is going to read multiple maps which these maps can be unexpectedly large, process with a large amount of data and most importantly support 3D graphics and animation performance will be a very important constraint for us. After testing the program there may be some minor changes to make the program to run better.

2 USER INTERFACE DESIGN

2.1 Introduction

ClimbPlanner software helps mountaineers to plan, simulate and animate their climbing session. Since not every mountaineer is a computer expert, user interface must be easy to manage so that mountaineers can easily use the software.

Description of GUI is below.

2.2 Main Window

This is the main window of Climb Planner. It contains:

- File, Animation, Add to DB, Edit Climbers and Equipment menus
- Create Project, Weather Conditions and Constraints and View Report tabs
- Play, Pause and Stop buttons
- Animation viewport

These components will be explained later.

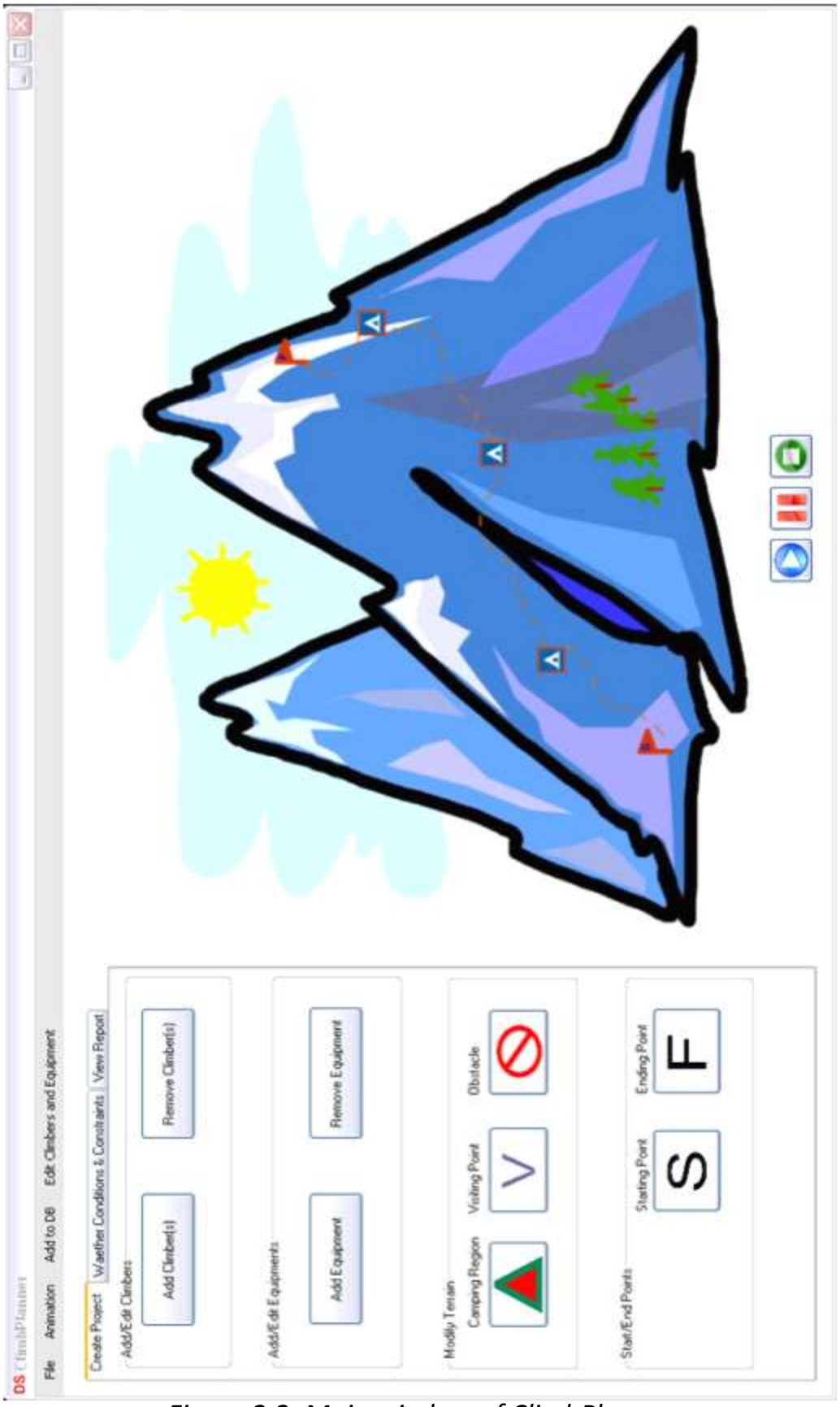


Figure 2.2: Main window of ClimbPlanner

2.3 File Menu

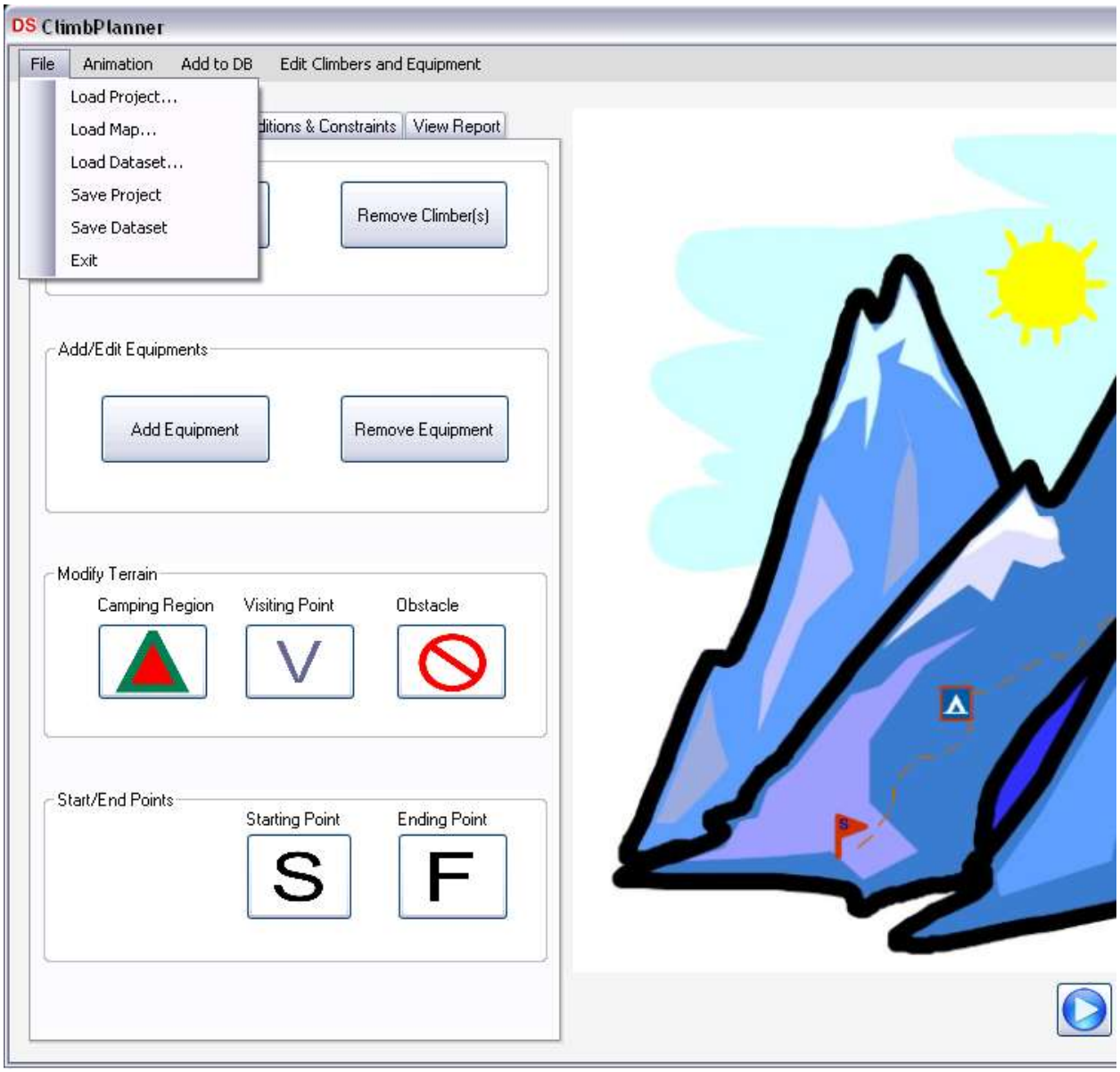


Figure 2.3.1: File Menu of ClimbPlanner

File menu contains the following components:

- Load Project…

User can load a previously created project by clicking on "Load Project…" and selecting the project from Load Project window.

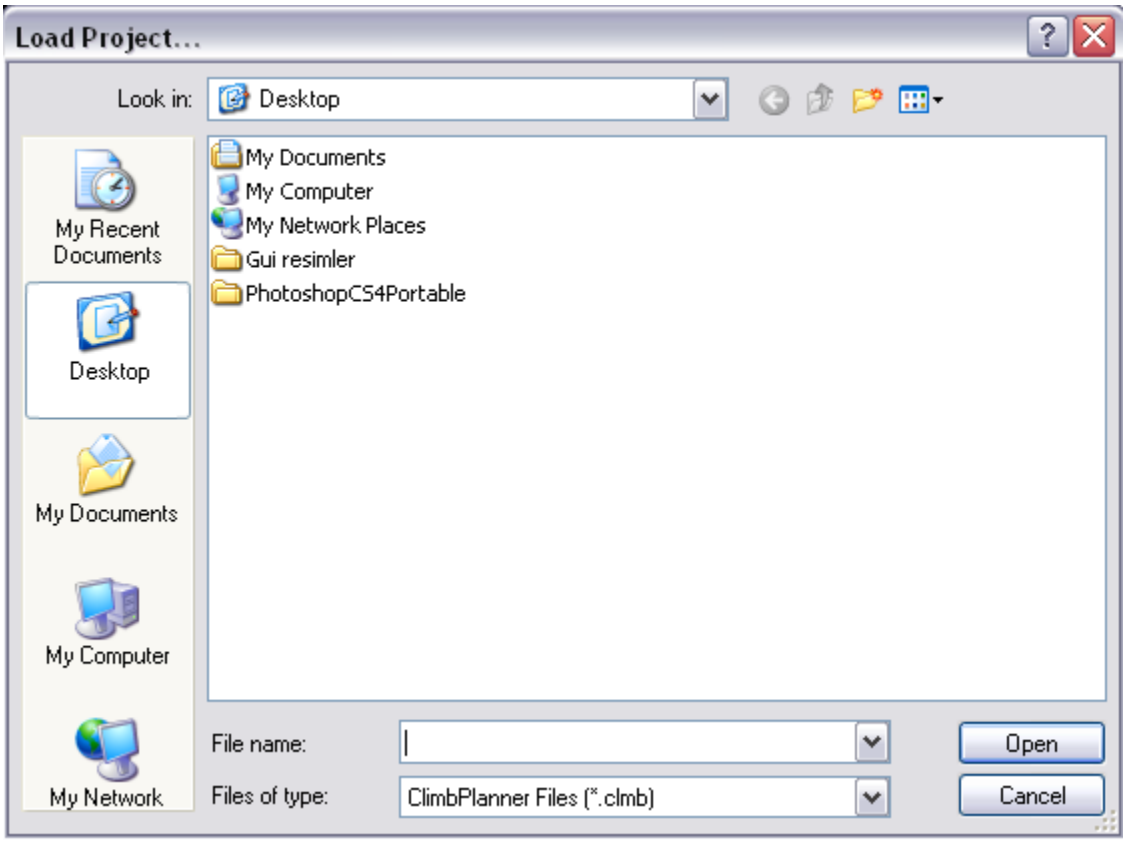


Figure 2.3.2: Load Project window of ClimbPlanner

- Load Map…

User can load one or several map(s) by clicking on "Load Map…" and then selecting map(s) from Load Map window.

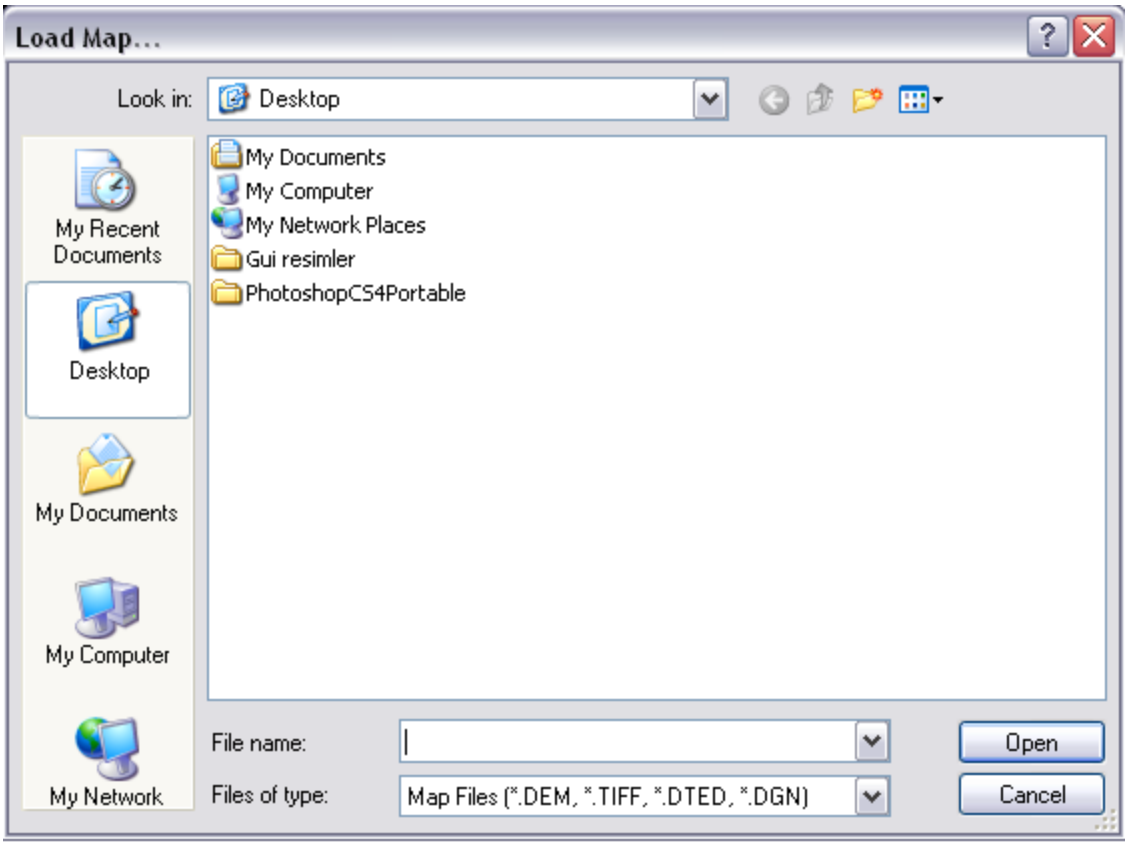


Figure 2.3.3: Load Map window of ClimbPlanner

- Load Dataset…

User can load a dataset by clicking "Load Dataset…" and then select a dataset from Load Dataset window. A dataset is previously recorded combination of several maps.

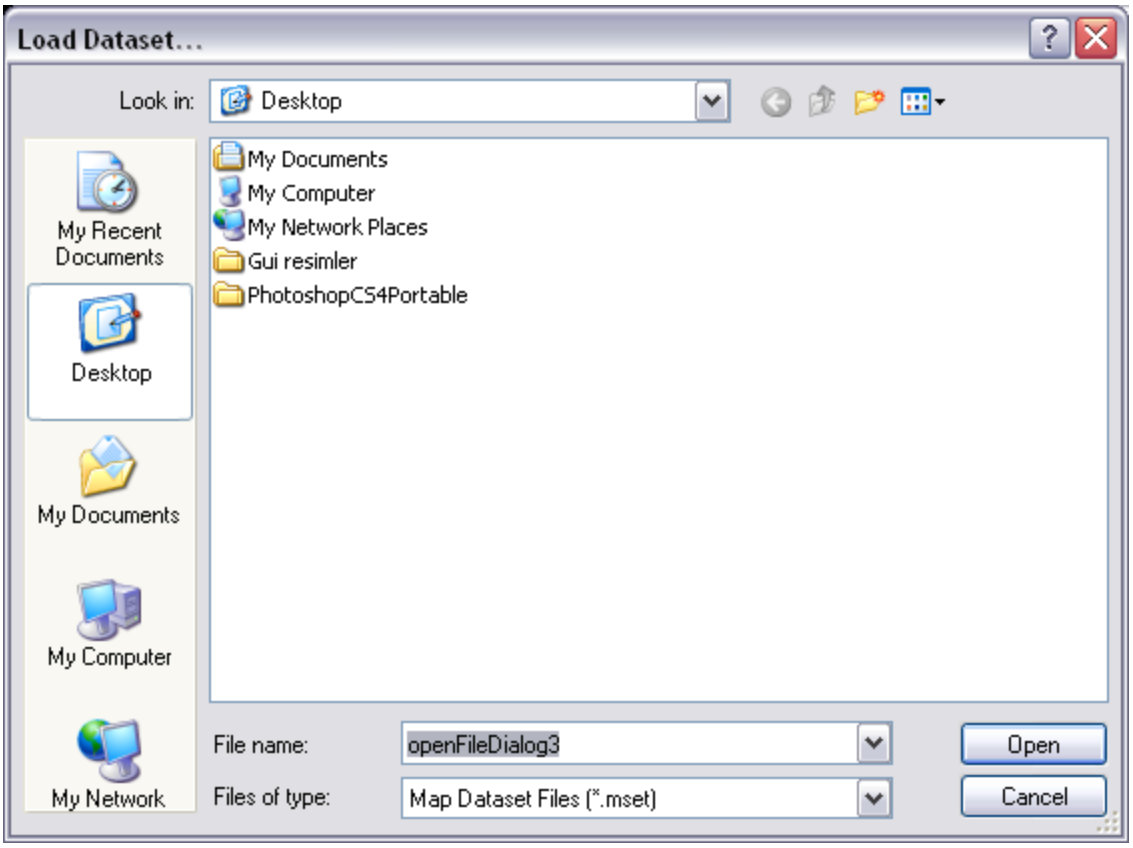


Figure 2.3.4: Load Dataset window of ClimbPlanner

- Save Project

User can save his/her project by clicking "Save Project" and then by typing the project's name into Save Project Window and select its location from it. Saving projects makes it easier when a user wants to retain and/or edit a previously created project easily.

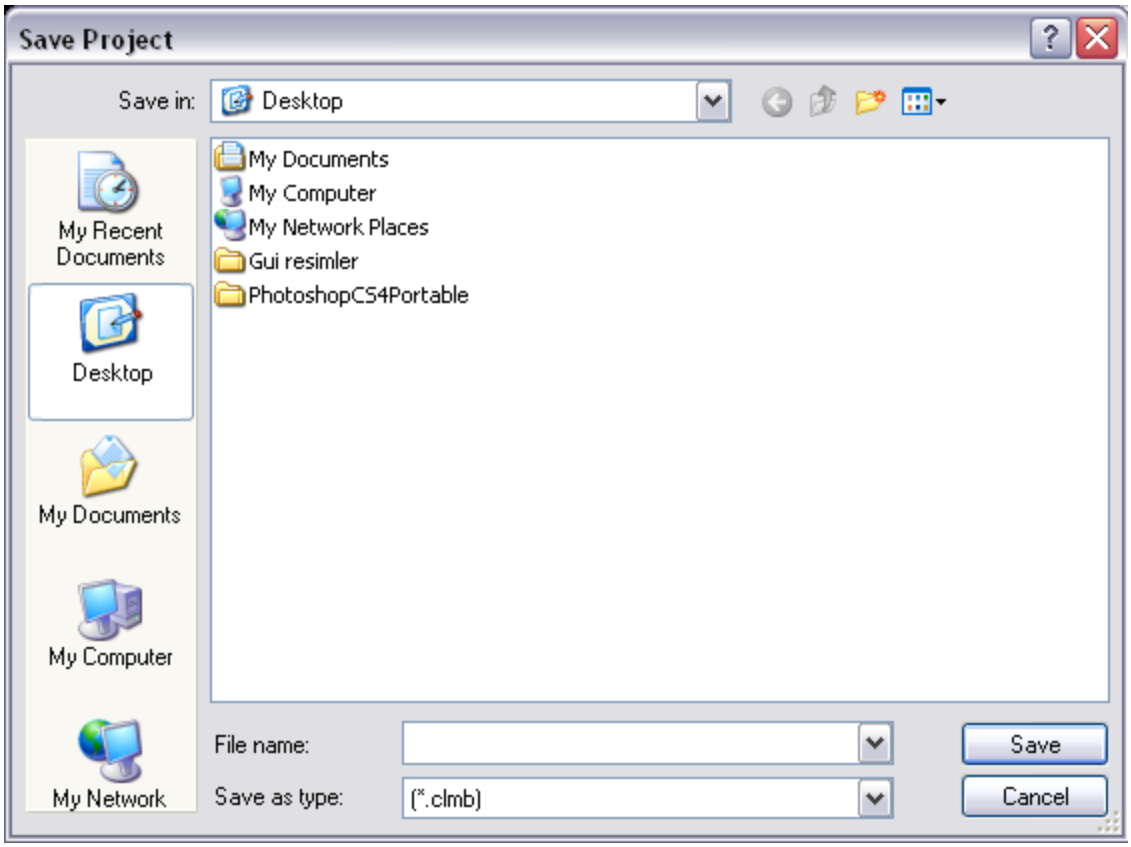


Figure 2.3.5: Save Project window of ClimbPlanner

- Save Dataset

User can save a combination of all load maps into a single map by clicking "Save Dataset" and then by choosing its location from Save Dataset window.

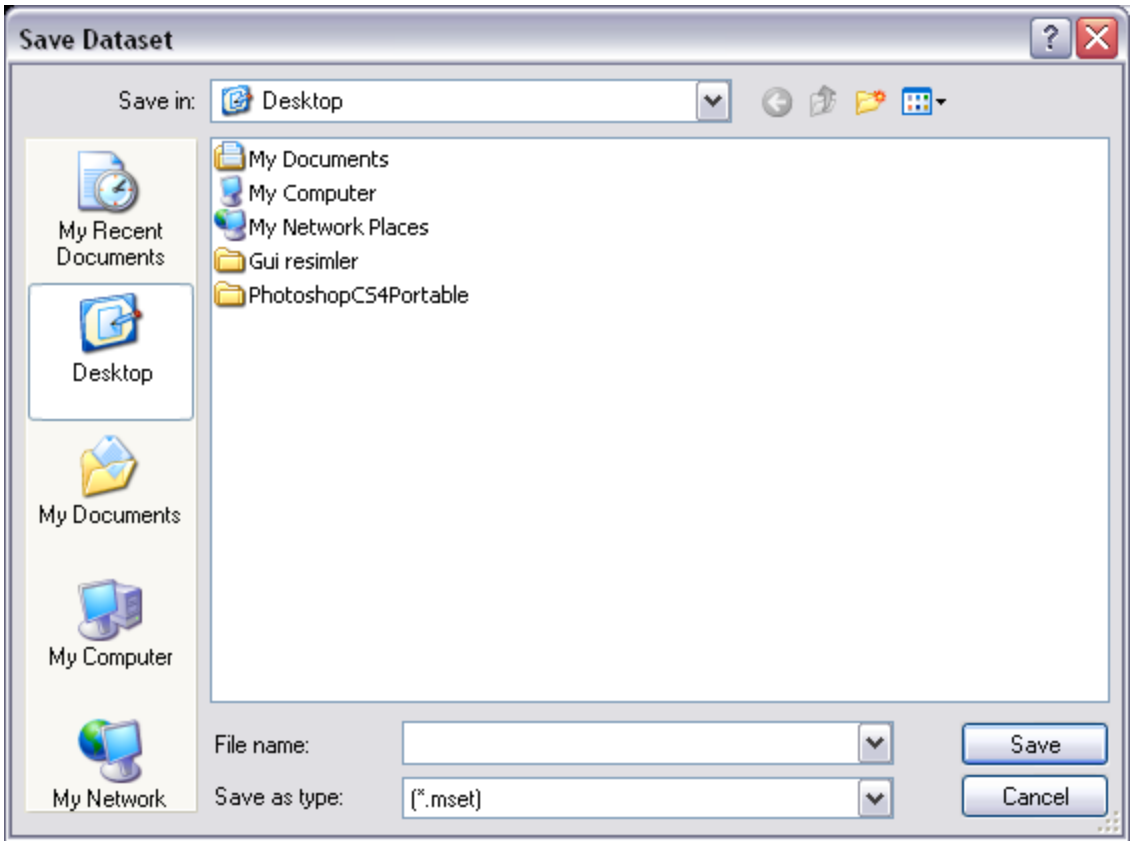


Figure 2.3.6: Save Dataset window of ClimbPlanner

- Exit

By clicking "Exit" user can terminate ClimbPlanner.

2.4 Create Project Tab

Within this tab, user can create a climbing project from scratch or edit it. Within the scope of this tab user can add or remove climbers and equipments to a project, modify terrain by inserting camping regions, visiting points or obstacles and define starting and ending points of climbing process. To modify

terrain, user must first click on one of the camping region, visiting buttons or obstacle button, then click on the map to insert one of these modifications. Starting and ending points can be inserted as just the same way. Camping region, visiting points or obstacle areas are not mandatorily expected from user but once any of them is inserted on the climbing terrain and submitted, AI module works accordingly.

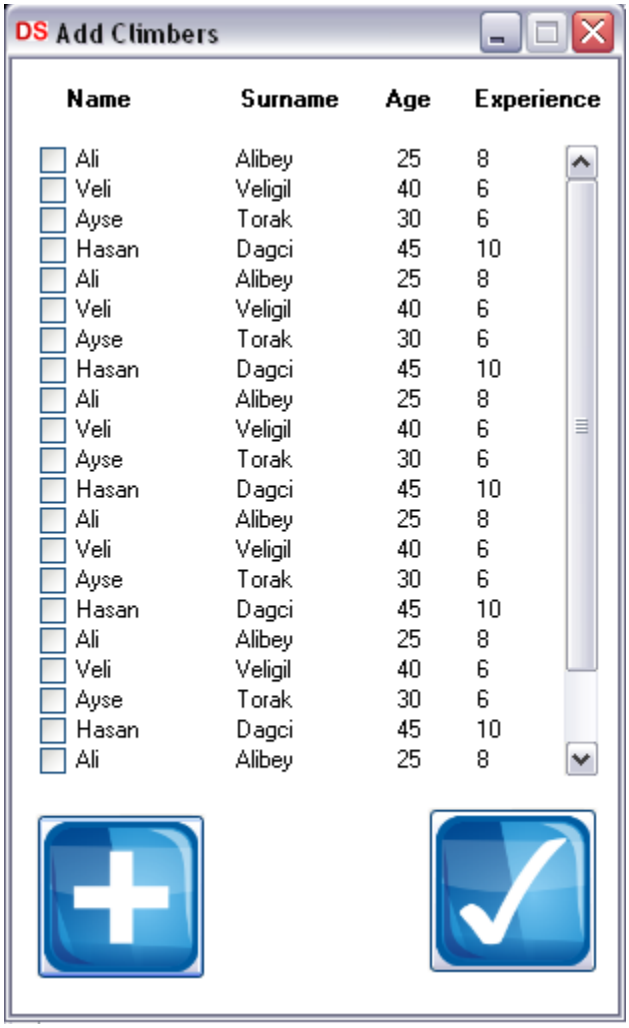


Figure 2.4.1: Add Climbers window of ClimbPlanner

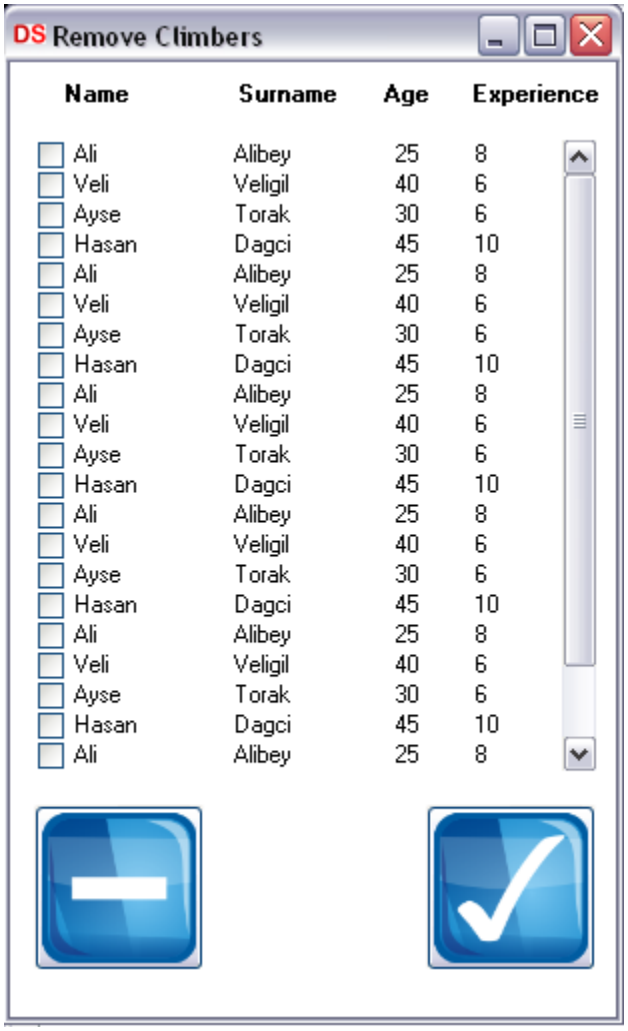


Figure 2.4.2: Remove Climbers window of ClimbPlanner

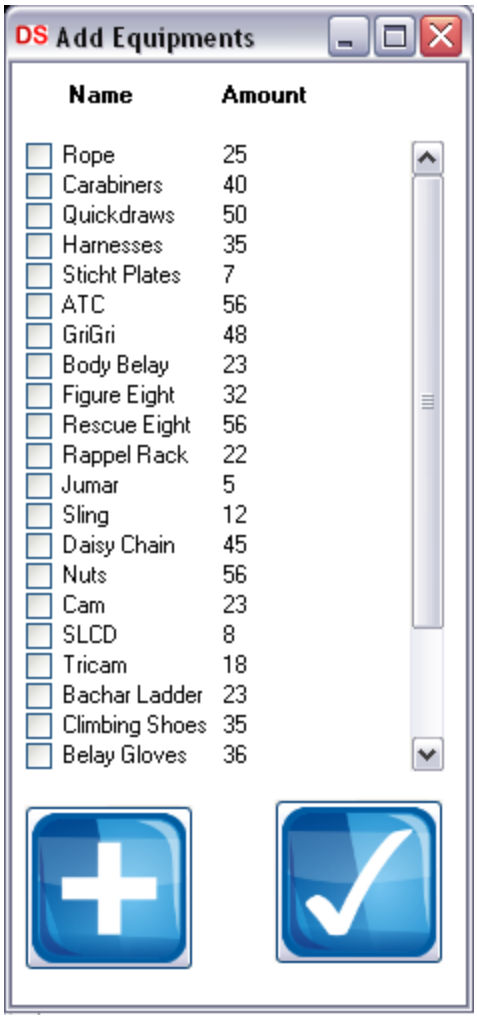


Figure 2.4.3 Add Equipments window of ClimbPlanner

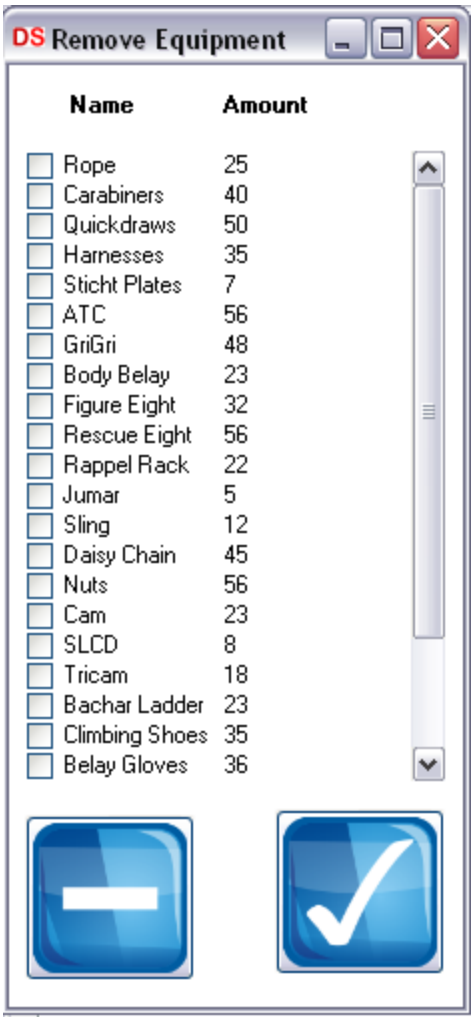


Figure 2.4.4: Remove Equipment window of ClimbPlanner

2.5 Weather Conditions & Constraints Tab

Over this tab, user can inform ClimbPlanner about the weather condition during the climbing process. This is not mandatory but makes the simulation and climbing calculations more realistic.

In addition to weather conditions, user can also enter the constraints for climbing process through this tab. Starting time box is the starting time of the

climbing activity and must be entered by the user. Max available time is the total time that the user spares for climbing activity. This can be left blank to point that user has enough time for climbing process whatever it takes to finish. Resting time and period is the preferred resting options by the user.

If user wants to climb on cliffs if any, or user wants to climb also at night time, user must check these cliff climbing and night climbing boxes.

If user wants to obtain the fastest route to aim point, he/she must check fastest route box. Easiest point box is just the similar except that the easiest route is the lowest inclined path between starting and the ending points.

When user is done s/he must click the submit button to pass the information to ClimbPlanner system.

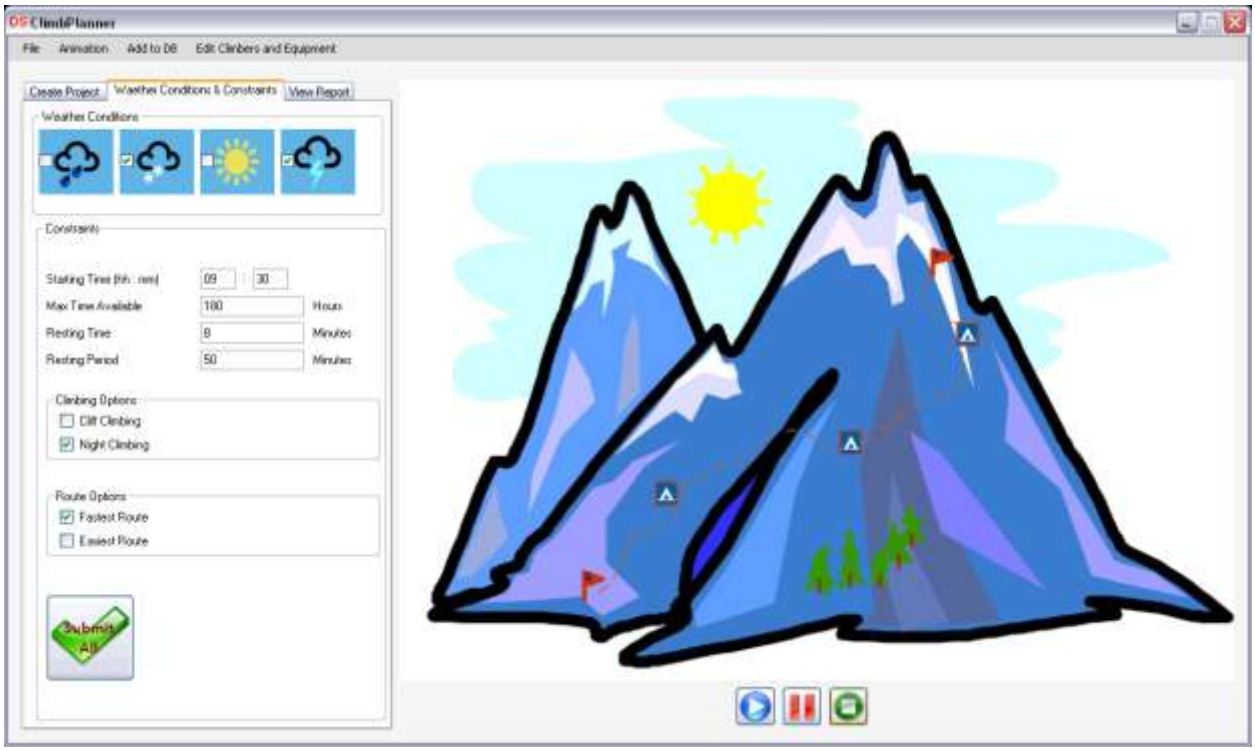


Figure 2.5: Weather Conditions & Constraints tab of ClimbPlanner

2.6 View Report Tab

When user is done with entering information about climbing process and submits it, a report of the whole climbing process appears on this tab. If user lacks of equipment needed for climbing activity, ClimbPlanner prepares a list of needed equipments and displays them on this tab textually. From view report tab user can also see the starting and ending times of climbing activity, suitable camping coordinates and the coordinates of the path s/he must follow.

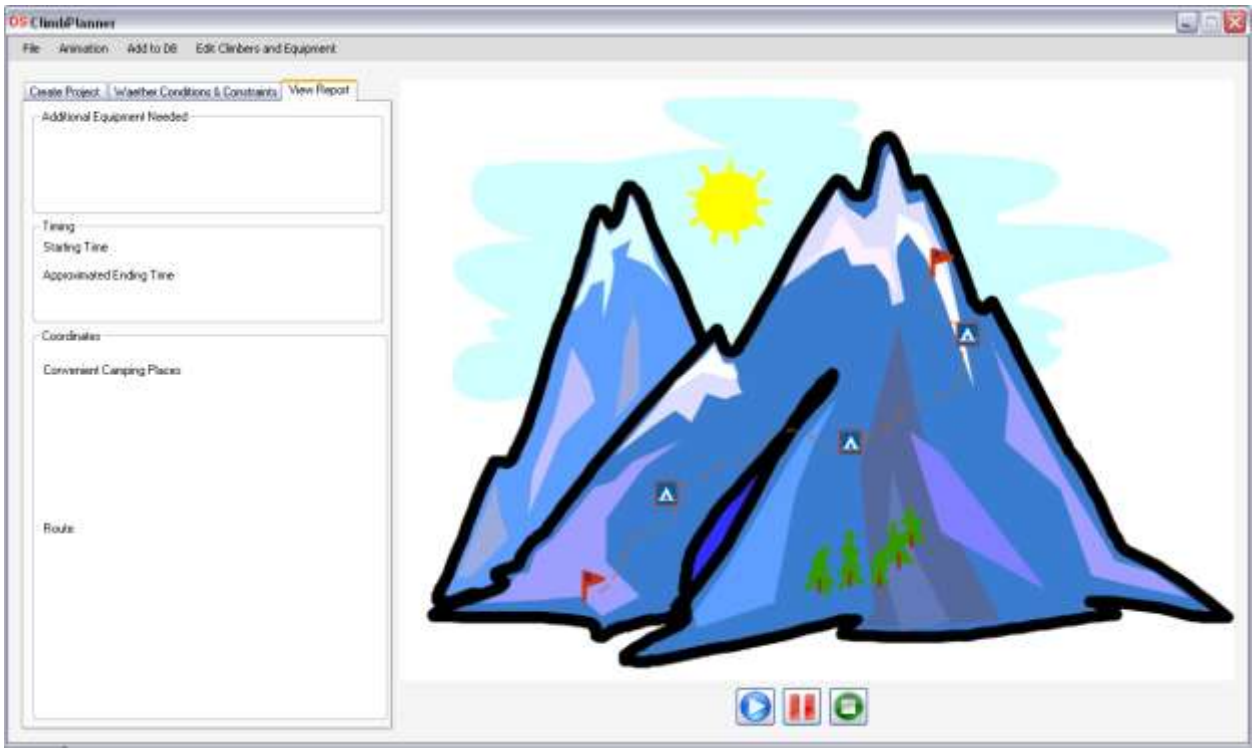


Figure 2.6: View Report tab of ClimbPlanner

2.7 Animation Menu

In addition to speed selection, animation menu has the same features as the buttons below of the main window, which are Play, Pause and Stop. User is able to select the animation speed by clicking on 1x, 2x, 4x, 8x. Once user clicks on “Play”, animation of the calculated climbing is illustrated on the right hand side of the GUI.

Picture on animation viewport here is obviously caricaturized just to show where the animation will take place. Actual animation will be rotatable and scalable by using mouse and it also will illustrate movement of the sun, starting, ending points of the route, land type, calculated route, North, South, East, West directions and camping points.

If the user clicks on pause button during animation, animation will be paused. In “clicking stop” case, the animation will stop and return to the beginning.

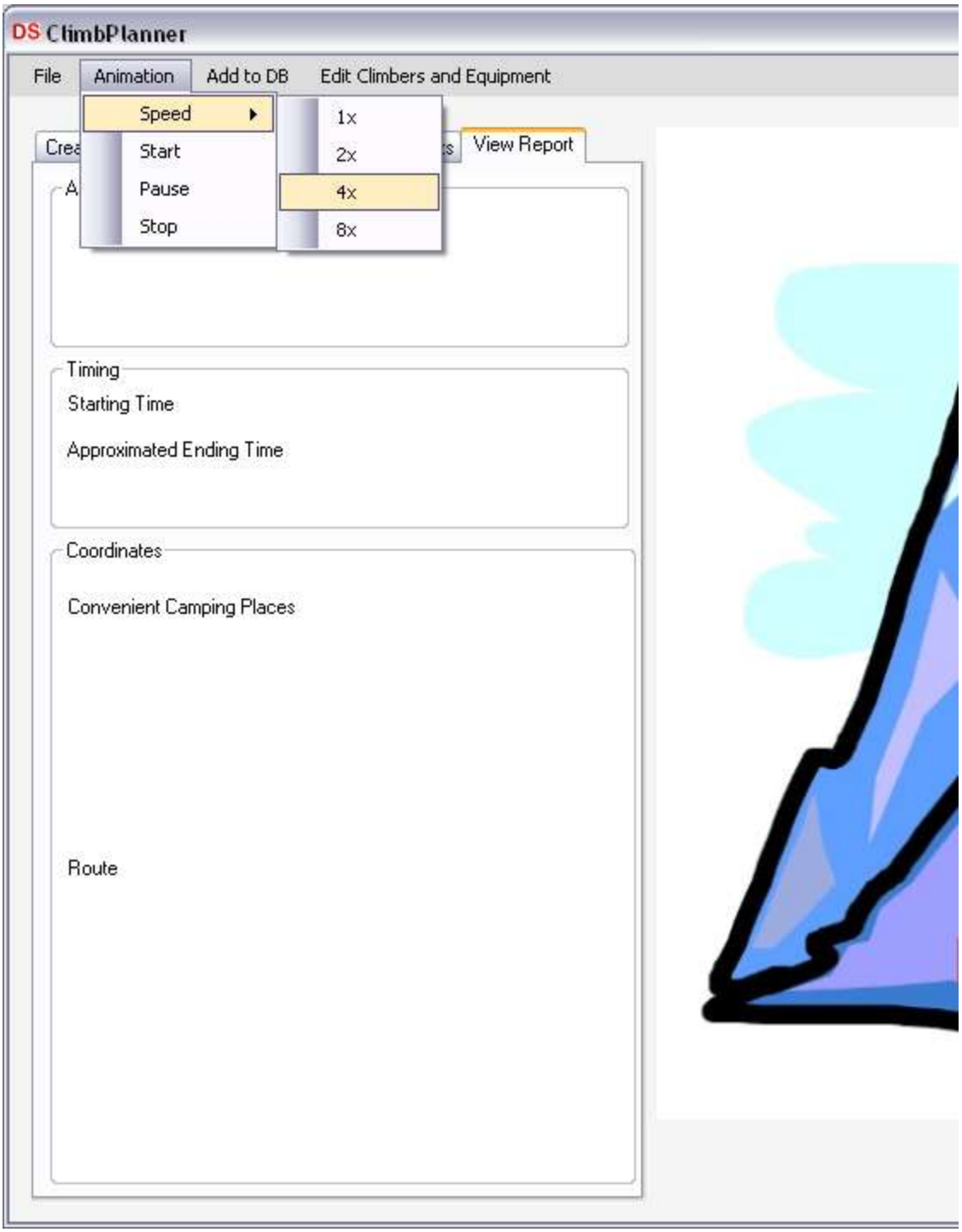


Figure 2.7: Animation Menu of ClimbPlanner

2.8 Add to DB Menu

This menu is used for adding climbers or equipments to the climbing clubs database. If user wants to add a new climber to climber database, s/he must click “Climber(s)” from this menu. After clicking “Climber(s)” a pop window is activated and user can enter the information about new climber through this window.

Figure 2.8.1: Add New Climber window of ClimbPlanner

If user wants to add new equipment to club’s database, s/he must click “Equipment(s)” and then enter the needed information in text boxes.

Figure 2.8.2: Add New Equipment window of ClimbPlanner

2.9 Edit Climbers and Equipment Menu

Though this menu, user can edit or delete climbers or equipments which are retrieved from club's database. Usage of this menu is similar to "Add to DB" menu.

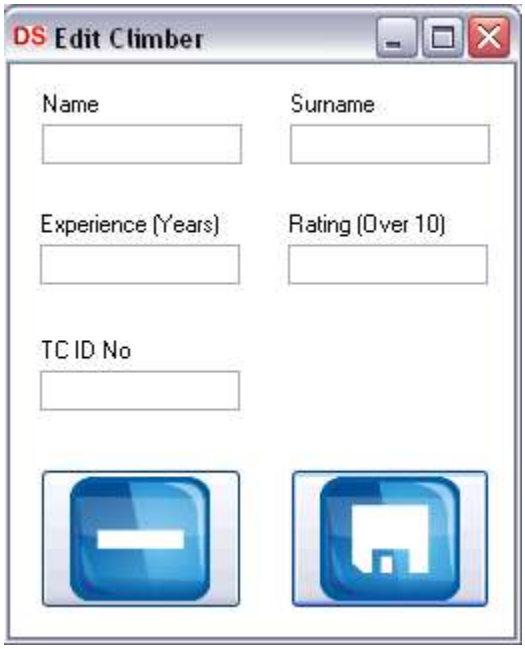


Figure 2.9.1: Edit Climber window of ClimbPlanner

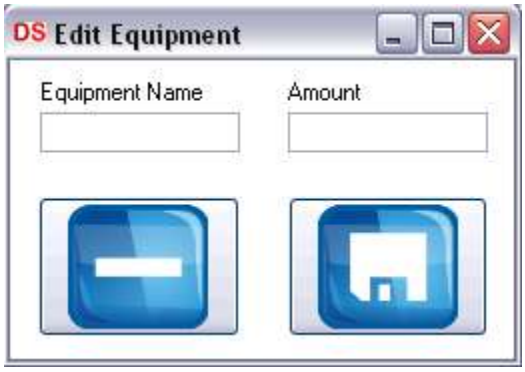


Figure 2.9.2: Edit Equipment window of ClimbPlanner

3 GENERAL PROGRAM FLOW

After ClimbPlanner software begins, user can create a new climbing project or load predefined climbing projects. In the case of that user selects new project, user can do the followings:

- To load desired map(s) or predefined datasets created by using earlier maps. After map loading process, program aligns all loaded maps (raster map and other vector maps) and creates visualization of terrain. On this terrain user can create land covers, obstacles, select visiting and camping points and select starting and finishing points.
- To create team by selecting climbers from climber database or by defining new climber(s).
- To load a team from team database.
- To select climbing equipments.
- To define climbing constraints such as:

- Resting time
- Resting frequency
- Night climbing
- Rock climbing
- Longest range per day
- Climbing range per day
- Fastest route
- Easiest route
- Weather information

If user loads previously created project, program imports all data related to selected project. After loading, user can modify previously recorded data (maps, team, equipment, constraints).

After user completes climbing project, user can do the followings:

- To start climbing animation in 3D environment with these options:
  - Adjusting animation speed
  - Pausing, Replaying, Stopping
- To obtain climbing report textually.
- To save climbing report and/or whole project to desired location.

4 ARCHITECTURAL DESIGN

In this section, reader will be informed about Climb Planner's architectural design, first by a general perspective, afterwards by a detailed description of subcomponents of the software.

4.1 Overall System Architecture

The application consists of several subsystems each of which is responsible for a different task. It is designed in a way so that it has a modular structure which makes it easier to maintain. Main principles of object-oriented approach are considered in the design phase. Each module is aimed to do only its job and its entire job.

ClimbPlanner is composed of four main parts to keep the minimal requirements of a possible climbing simulation. These are a simple and clear GUI, a 3D visualization module, an AI module to analyze and process the climbing route, a map reader module to maintain the environment data to the visualization and AI modules. The roles of these modules are defined to satisfy a strict interface through the data flow inside the application.

4.2 Level 0 Dataflow Diagram

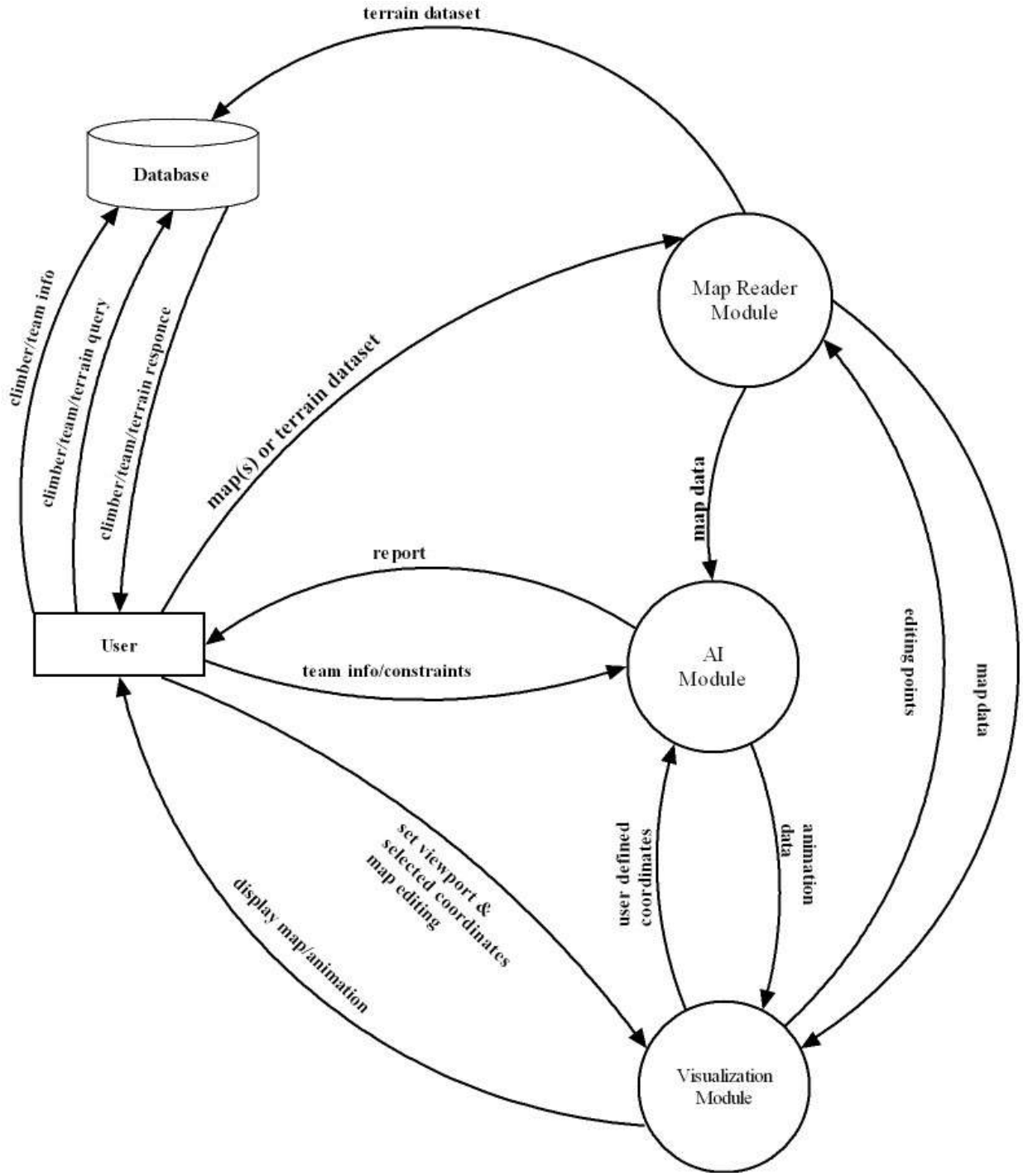


Figure 4.2: Level 0 Dataflow Diagram

Climb planner is composed of four main parts to keep the minimal requirements of a possible climbing simulation. These are a simple and clear GUI, a 3D visualization module, an AI module to analyze and process the climbing route, a map reader module to maintain the environment data to the visualization and AI modules. The roles of these modules are defined to satisfy a strict interface through the data flow inside the application. Above, data flow diagram level 0 illustrates the communication between modules and database where GUI is represented as user.

Map(s): User selects which mountain map is to be simulated. These maps are sent from GUI to map reader module as operating system dependent logical file locations to be opened and handled directly.

Team Data: Team data is composed of team members' information and the equipments that team owns.

Constraints: These are climbing speed, resting time, resting frequency, weather conditions, starting time, night climbing choice, cliff climbing choice, climbing range per day, longest range per day and desired path type.

Terrain Data: Terrain data is a pointer to file that is keeping information about previously constructed map data sets.

Map Data: Map data is gridded data of combined maps which includes land cover type, height, latitude and longitude coordinates and user defined points.

Animation Data: Animation data is made up of user defined points, calculated path coordinates, user-defined constraints and timing information. It is simply a combination of map data and user specific information.

Report: Report is the textual illustration of climbing process.

Map Editing: This is inserting or deleting geographical constraints on mountain map by user.

Viewport information: Since the user will be allowed to wander inside the 3D environment, position and orientation information of the viewing camera will be gathered by mouse and keyboard inputs and sent to the visualization module to be handled in real time. Visualization module will process the viewport according to these inputs.

Selected coordinates: This data group represents the information of coordinates which user wants to be processed through the simulation. These are starting and ending points of the mountaineering activity and desired fixed visiting coordinates.

This process is handled by visualization module to unify the screen output and AI output since it would not be nice if the location of data is not the same with which user sees. Visualization module will translate these mouse data to real coordinates, display them in screen and send them directly to the AI module.

Editing points: These are the points edited by user on mountain map image. It can be selected coordinates or added or deleted land covers and obstacles.

4.3 Level 1 Dataflow Diagrams

The main role of the map reader module is to achieve an abstraction between GIS data formats and map data which will be used by AI and visualization modules. This module works in accordance with what kind of files was loaded and this accordance is provided by **Format Factory**. In the case of that user loads new maps simply it reads the raw map data (this is not shown in the dfd for the sake of convenience) and converts it to per-grid data. Doing this, it will make use of a common data structure - called **MapData** - which will be realized and used by the whole application without any further needs of

transformation. Through the lifetime of every single simulation, this module is called for one time, at the beginning. It will be capable of calculating intergrid slopes, which will be needed by AI module in the determination of the elevation weighted shortest path. On the other hand if user loads one of the predefined dataset (called **terrain dataset**) which is dataset that created from earlier maps, then map reader module just reads the file with a special extension and containing terrain information and then creates **MapData**. A birds-eye view of this module can be seen in DFD level 1, below.

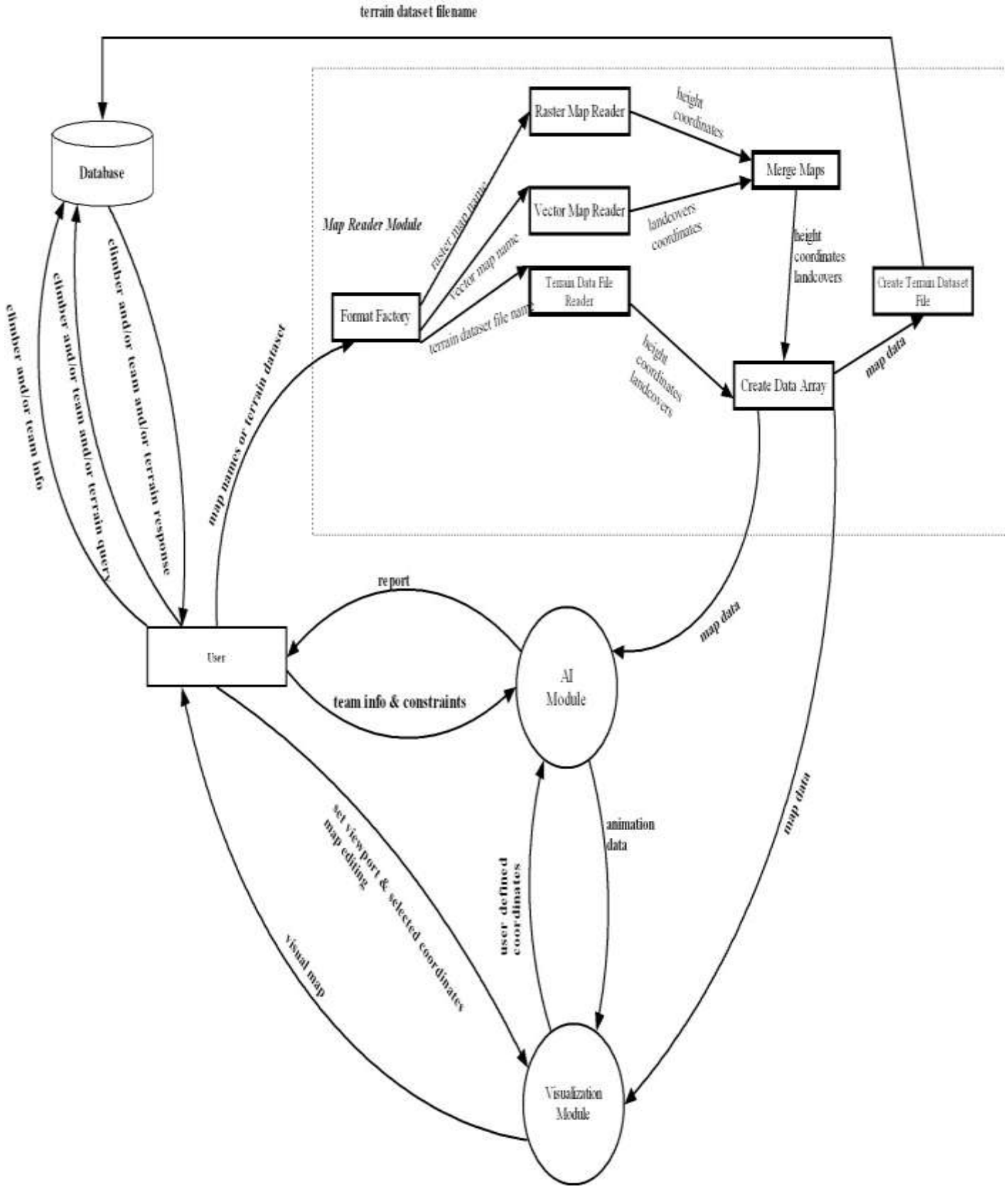


Figure 4.3.1: Level 1 Dataflow Diagram for Map Reader Module

AI module abstracts its internal structure with the other parts of the application totally. It gathers user specific inputs as states to be used in the route determination of grid based map data. Its output is time-location-progress oriented animation data to visualization module.

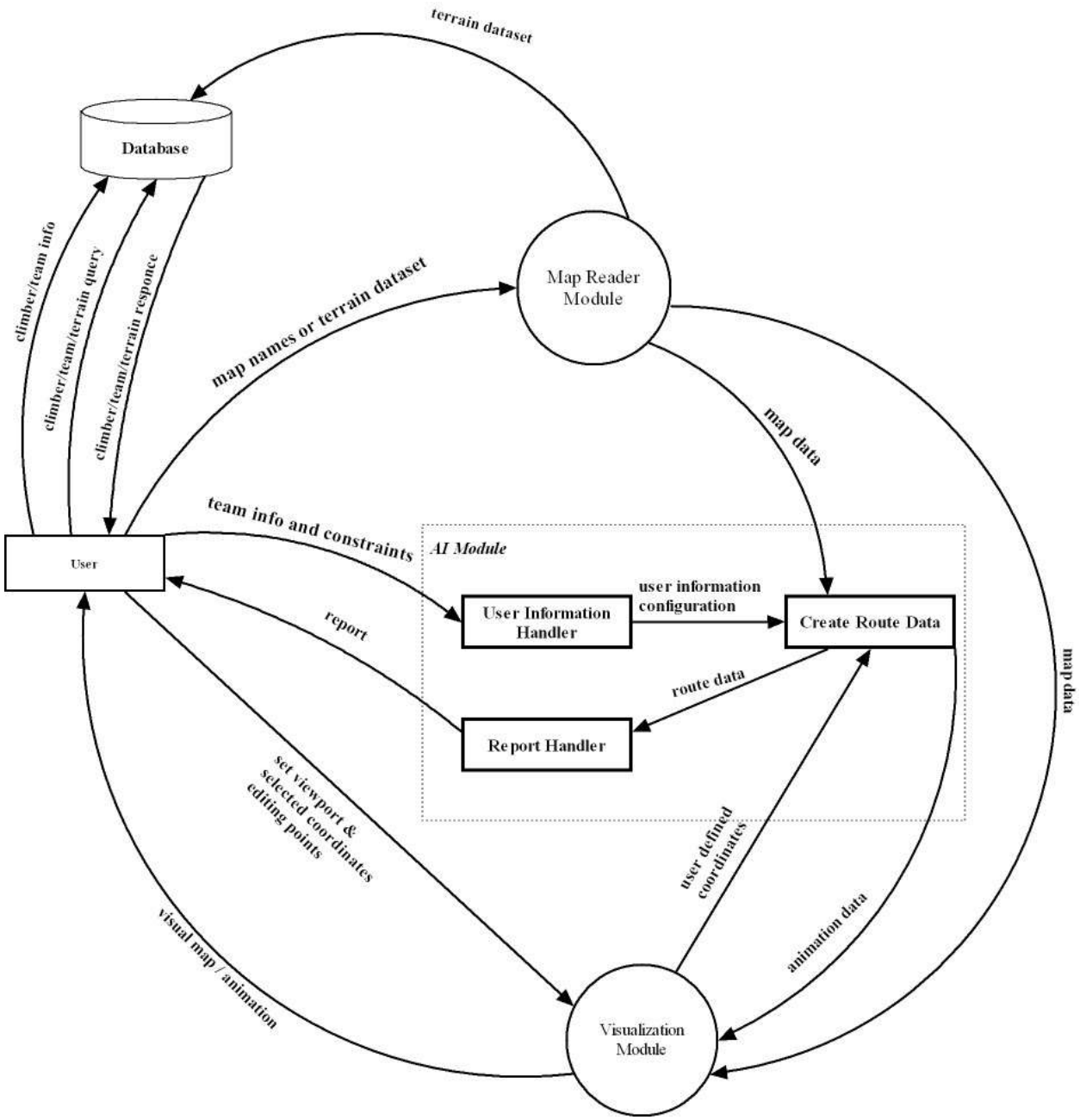


Figure 4.3.2: Level 1 Dataflow Diagram for AI Module

Visualization module will be responsible of building the 3D representation of the mountain using the 2.5D height map data, mainly. It constructs its own polygonal data and uses it through the simulation. Also module combines the user defined coordinates, which are mentioned above, with this polygonal data and animates the climbing route which it takes from AI module. More specific representation of the visualization module can be seen in DFD level 1.

Visualization module has three functionalities called animation process, selected coordinate handler and 3D visualize. Animation process gets the animation data from AI module and gives the next animation point to 3D visualize. By getting map data, next animation point, selected coordinates and viewport settings, 3D visualize returns the visual map to user. Selected coordinates handler gets the desired coordinates from user and deliver them to 3D visualize and AI module.

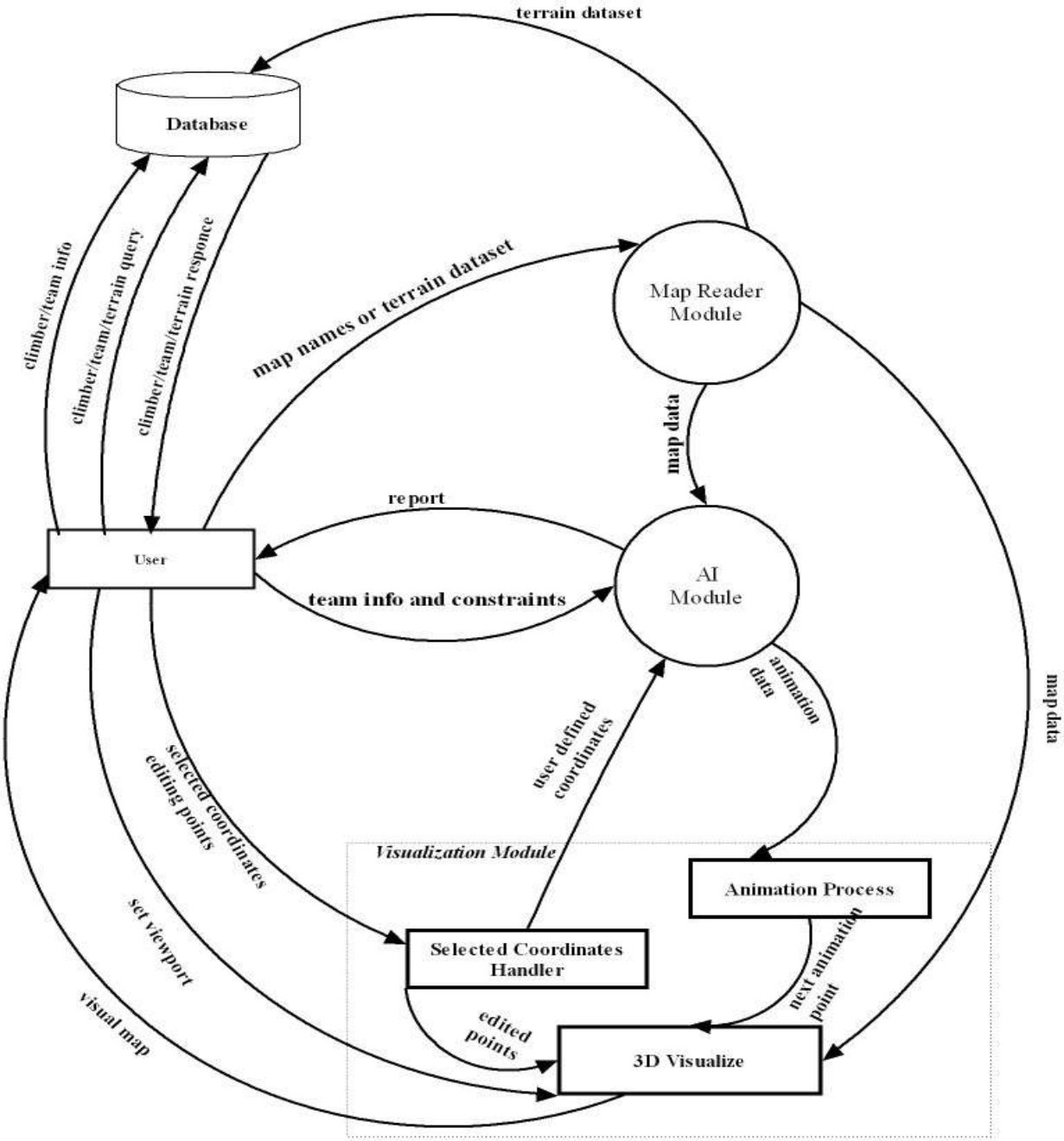


Figure 4.3.3 Level 1 Dataflow Diagram for Visualization Module

4.4 Main Sequence Diagram

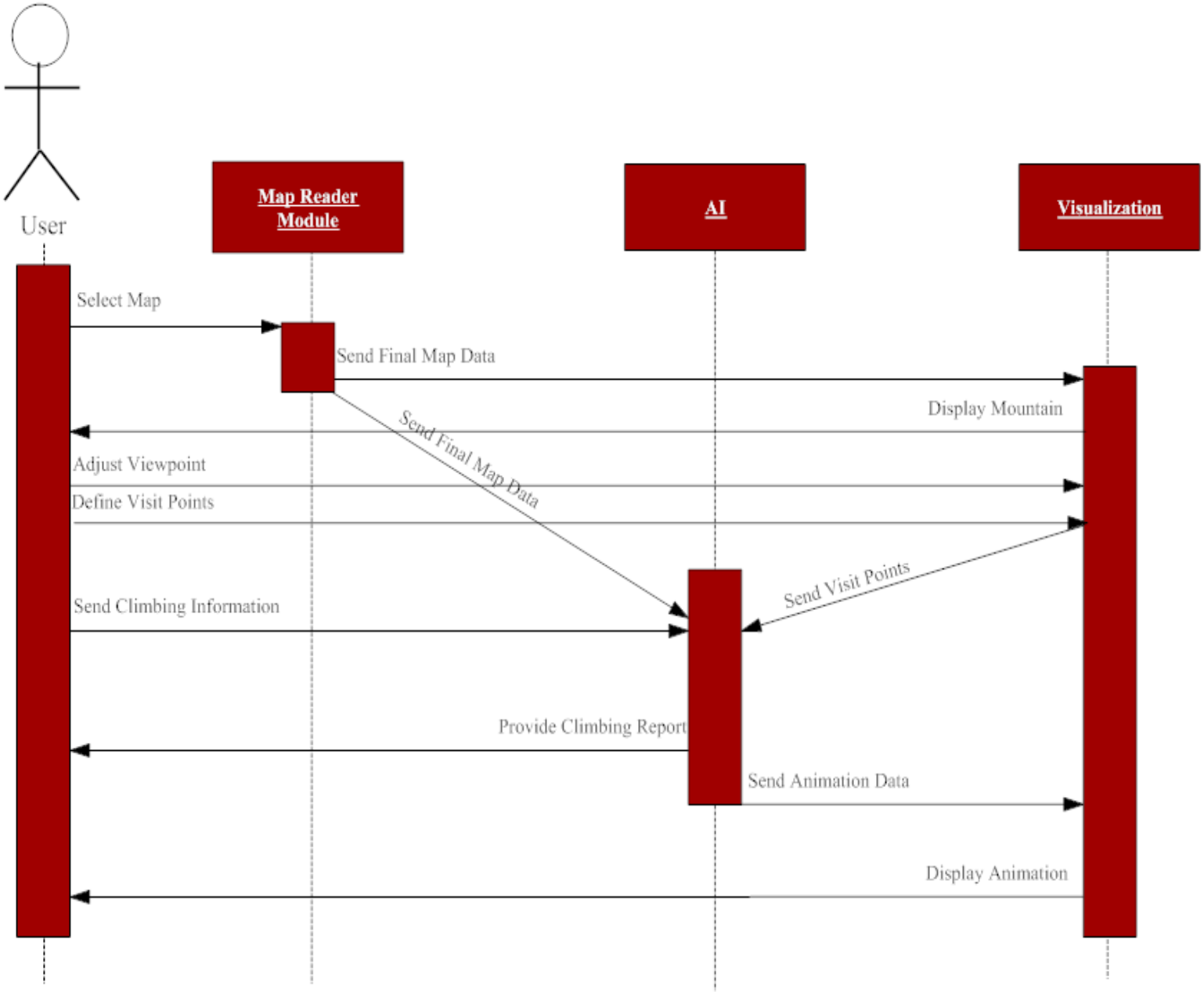


Figure 4.4: Main Sequence Diagram

User selects map(s) (or data set) from GUI File menu. Map reader module reads the map(s) and sends final map data to both visualization and AI modules. Once Visualization module gets the final map data, it begins to

display the selected mountain to user via GUI viewport. Now user can adjust the viewport by mouse and define the visit points. Now the defined visit points go to visualization module and then to AI module. After AI module gets the user specific information (climbing information), it calculates the path and returns the climbing report to user and returns animation data to visualization module. After visualization module gets the animation data, it displays the animation to user.

5 DETAILED DESIGN

5.1 Visualization Module Design

The main design aspect of visualization module is its necessity of performance and realism. This module is designed to satisfy such goals. Also expandability has been tried to be kept for future purposes.

To define the performance limits, simple measurements have been conducted:

- Intel Core2 Duo T5550
- ATI Radeon HD 3470
- 1280 by 800 screen resolution
- No culling algorithms (excluding OpenGL's basic polygon culling)

With this setup above, the map of Hasandağı can be modeled with an average frames per seconds of 55. Although this level of performance is sufficient for a simulation environment, larger maps can be problematic in the means of resolution. Since Hasandağı can be considered relatively small map (671 by 440 grid resolution, it is modeled with 1200K triangles), it is decided that visualization module should consist an open area oriented culling algorithm.

Considering 2.5D static terrains, one of the most effective and contemporary culling algorithms is octree implementation and its derivatives.

Quadtrees, a derivative of octree, makes use of image pyramids to cull any non visible terrain. In this algorithm, every node is composed of 4 child nodes dividing its terrain into 4 equal parts, where the root node stands for the whole terrain. With such a hierarchy, whenever a node remains outside of the viewing frustum, since it will be culled, there is no necessity of further computation by controlling its children if they are inside the viewing frustum. When this operation is carried out recursively until a predefined extend, non visible section of the terrain can be estimated roughly and cheaply in cost.

Using quadtrees enables another open area simulation technique called level of details (lod). Combined with quadtrees, lod technique can decrease the polygon count to be drawn for each frame by reducing the detail level of distant portions of the terrain to the camera. Like in image pyramids, every node of the quadtree keeps its level of detailed model, since it can be expensive to calculate it for each frame. When any node stands away from the camera by a predefined distance, its level of detailed model is drawn while rendering process is stopped before its children are drawn. The disadvantage of this algorithm is that it is costly for memory since approximately two copies of the terrain data is kept in memory through the simulation. Considering its success on rendering performance, this can be sacrificed easily.

These two algorithms will compose the basis of the visualization module design.

5.1.1 Data Structures

The main data structures that will be used by visualization module can be seen below. It should be noted that these structures will be used only by culling tests. Rendering calculations will be carried out by OpenGL environment.

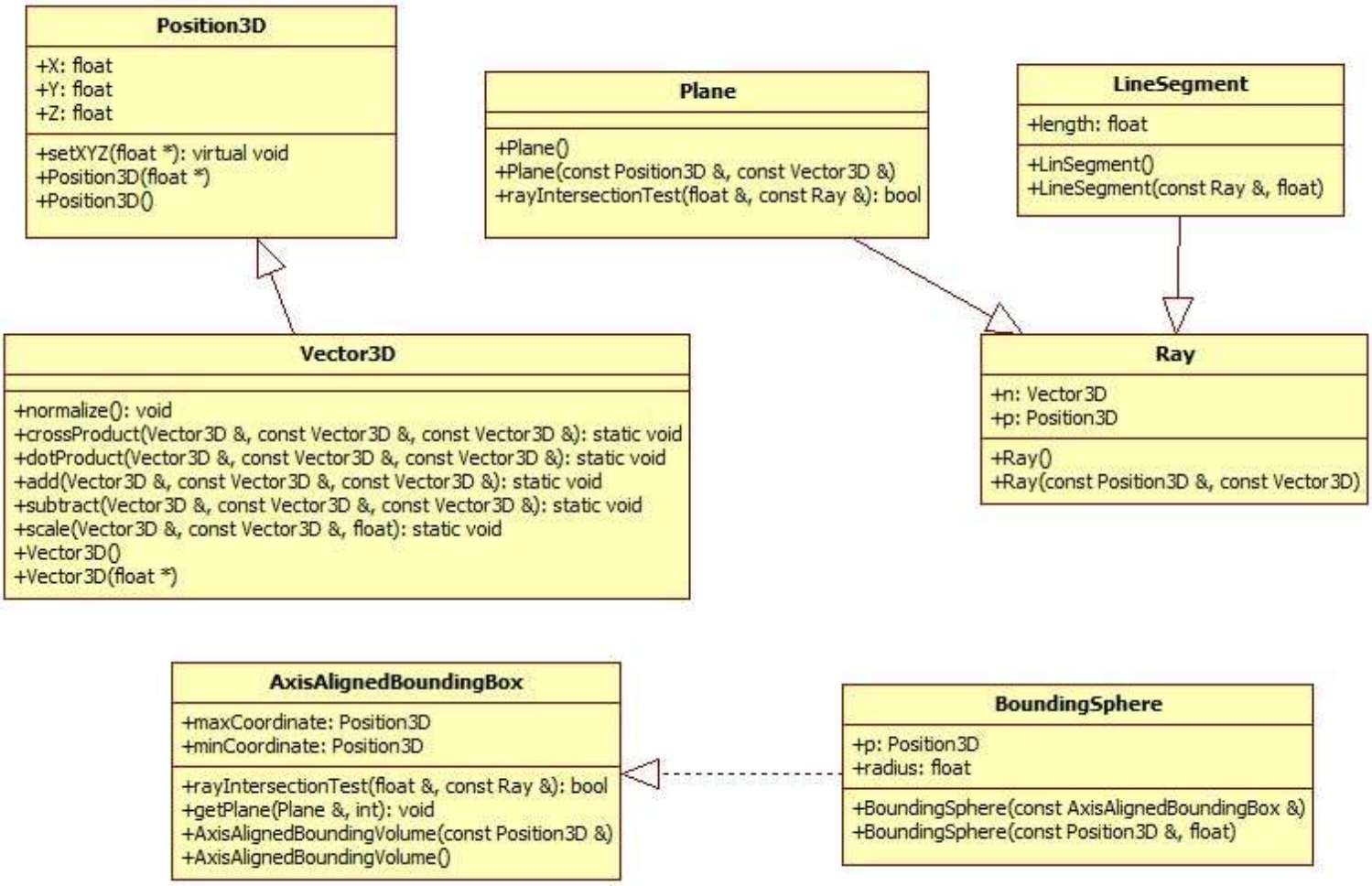


Figure 5.1.1: Visualization Data Structures

Position3D and **Vector3D** are representatives for a coordinate in 3D. Vector3D class includes basic geometry skills between two vectors.

Ray and **LineSegment** classes stand for rays and line segments as it can be forecasted. These classes will be used by intersection tests by bounding boxes of quadtrees. Mouse coordinates, which comes as a screen space input, can be translated to real 3D coordinates easily by using these data types.

View frustum will be composed of **Planes**. This class will be used in quadtree culling. **Plane::RayIntersectionTest** will be used by ray casting of mouse coordinates.

AxisAlignedBoundingBox and **BoundingSphere** will be the main components that will be used by intersection tests. Both of them will represent a quadtree node by wrapping it minimally in size. Culling operation will be carried out in two steps. First BoundingSphere vs. ViewCone intersection test will be executed. If node passes this pass, then the second, AxisAlignedBoundingBox vs. ViewFrustum test, will be controlled. If quadtree node passes both tests, then it will be respected as visible and its children will be tested. If node fails in one of them, then it is assured that this node is outside the viewing frustum.

The main goal of using two tests is that BoundingSphere vs. ViewCone test is very cheap considering AxisAlignedBoundingBox vs ViewFrustum test. But it is less accurate then the latter. Using both, an optimum solution can be achieved.

5.1.2 Module Design

The main roles of visualization module is to display the map terrain, selected coordinates as start point, end point and visited points, the mountaineer team, a skybox(not indicated in the class diagram), with animation capabilities using light and textures. To achieve the compatibility

with the other modules, all positions will be kept in real coordinates inside the visualization module.

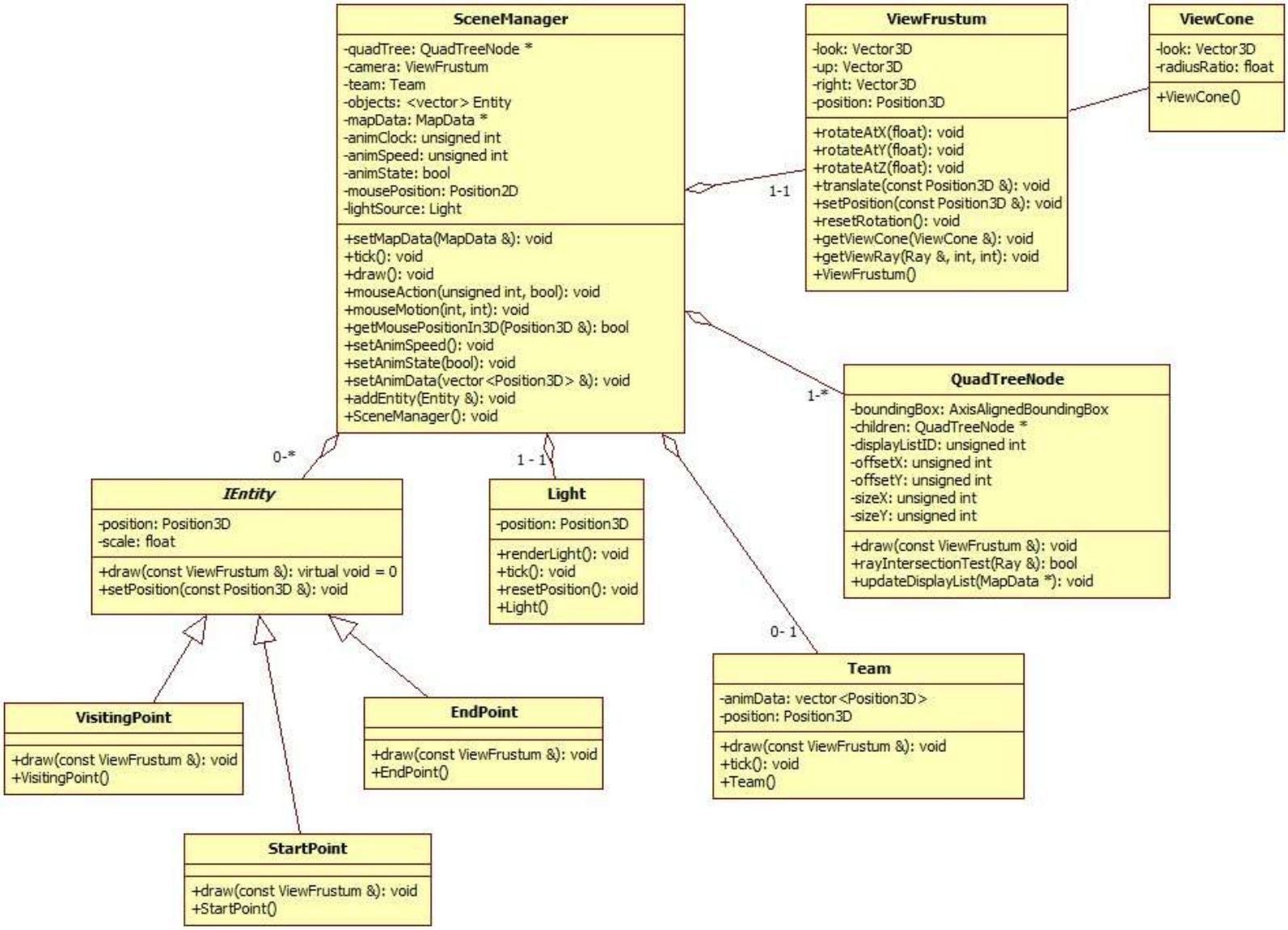


Figure 5.1.2: Visualization Main Module

Design is centered around the **SceneManager** class. It is tried to keep one interface for this module, which is the SceneManager. All the inputs are delivered by this class through the module. Also drawing commands and ticks are taken from SceneManager.

QuadTree root is kept under SceneManager as **SceneManager::quadTree**. It is constructed by **SceneManager::setMapData()** instantly to prepare the application for the visualization. Recursively to a predefined level, **QuadTreeNode** class constructs the tree appropriately by calling **QuadTreeNode::updateDisplayList()** for its children, where display list stands for pre compiled OpenGL display lists. By using display lists, terrain model data will be compiled for once and stored in memory through the application (When a modification occurs in MapData, SceneManager will handle it by calling SceneManager::updateDisplayList() again). QuadTreeNode class has also the capability of ray intersection test for mouse coordinate to real coordinate translation. Culling test will be executed inside **QuadTreeNode::draw()**.

SceneManager class has a **ViewFrustum** attribute representing camera. ViewFrustum and ViewCone will be used for culling operations. **ViewCone** has not a direct access for rotation and translations. Rather, it will be constructed from ViewFrustum class when it is needed.

Team, **StartPoint, EndPoint** and **VisitingPoint** classes are considered to be 2D sprites. Except for team, all are inherited from **IEntity** class to achieve expandibility. It is possible to design more entities like tree to represent forest.

Team class has an animation capability. To achieve this, it keeps the animation data as **Team::AnimData**. By calling **Team::tick()**, SceneManager provides the animation of this class. In every call of this function, SceneManager::team attribute changes its position appropriately.

SceneManager keeps track of the animation by its **SceneManager::AnimClock**, **SceneManager::AnimState** and **SceneManager::AnimSpeed** attributes. These will be modified by GUI using related accessor functions.

Light class represents sun light if it is noon, moon light otherwise. It is animatable like Team class. SceneManager calls **Light::tick()** function to achieve this goal.

5.2 AI Module

The main design goal of AI Module is to detect the most optimal path between user given start and end points, using AI states that are taken from GUI module.

Considering graph search algorithms, A* shortest path algorithm is a strong candidate for this application since it uses a distance-plus-cost heuristic function. Comparing with single cost shortest path algorithms like Djikstra, A* algorithm considers not only the cost of the path to be detected, but also the distance to the end node by using 2 functions for each purpose. So A* is modifiable according to needs. Since application presents the possibility that user can requests different path processing modes, AI module will make use of it.

Using grid based maps makes easier to implement A* algorithm. If each grid of the heightmap is considered as a node for the graph, then distances between neighbor grids constitutes the edges between nodes. In this perspective, each node has 8 neighbors and total of 8 edges to them, since each grid has 8 (except the boundary grids). AI module will follow this pattern through its design and implementation.

The main processing unit in AI module is **AIMap**. It keeps all the states, graph data and priority queue to be used. Before any path processing, it takes all its input from GUI. So it is the only interface to other modules.

AIMap class keeps the main map data in **AIMap::mapData** attribute, which has all the height map and terrain information. But this member is not modified during AIMap's execution, it stands only for providing information about the main map. It is set by the **AIMap::setMapData()** function.

AIMap class uses **AIMap::mapNodes** for its graph related processing. This member keeps all the information about each graph node (or other saying grids) in its **Node** class instances through the mid-processing. A Node keeps track of for which grid it stands by **Node::xRef** and **Node::yRef**, the current cost by **Node::currentCost**, visiting state by **Node::visitingState** (alternatives are not visited, visited or currently visiting), a priority queue reference for update operations and last node which is followed by(to detect the final route).

PriorityQue class is a little different of std's by its **PriorityQue::update()** function. There is no such alternative in std::priorque class. This function allows to modify PriorityQue data without reordering.

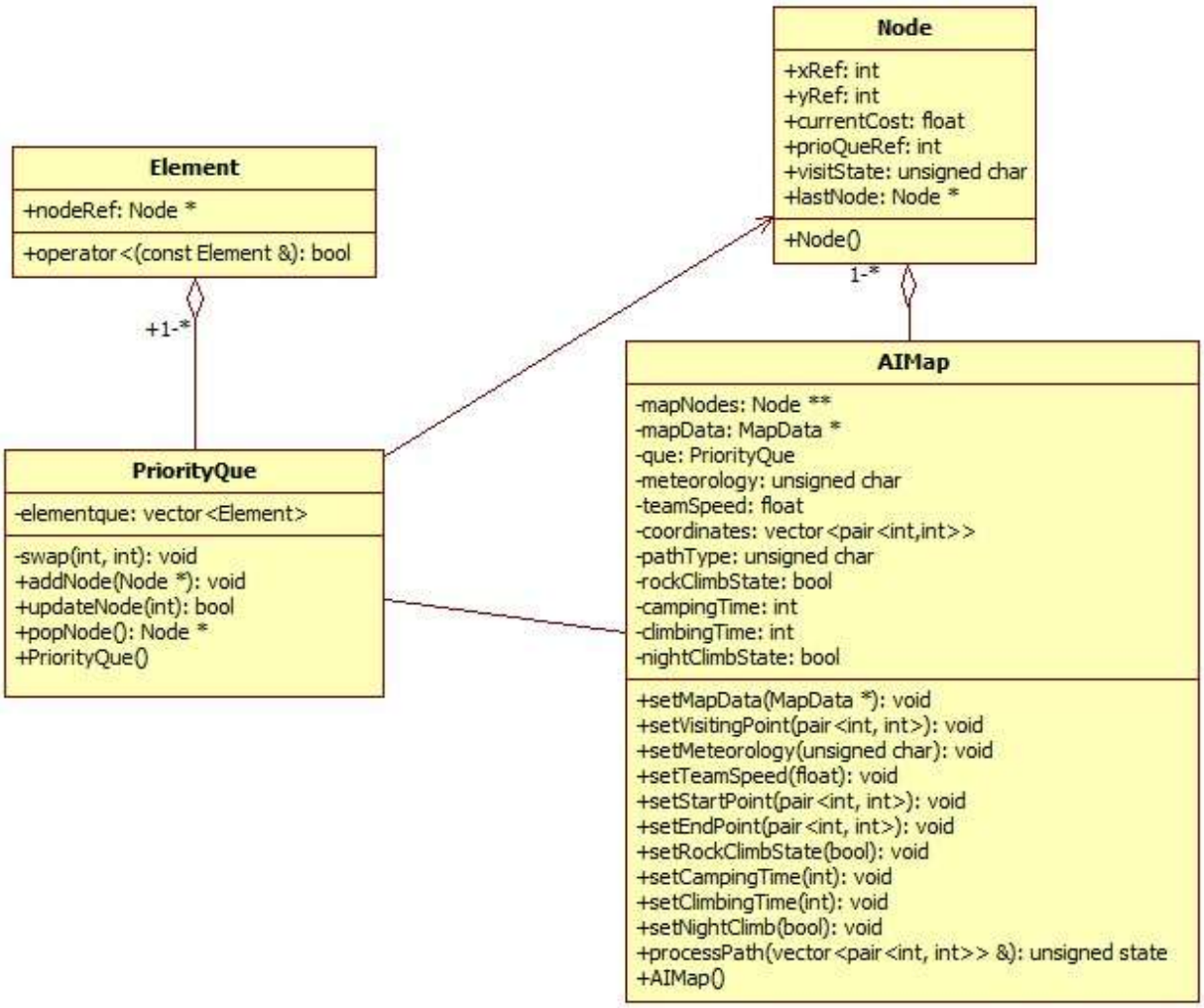


Figure 5.2: AI Module Class Diagram

5.3 Map Reader Module

The main role of MapReader module is to get the mountain maps in several GIS formats, process them and present them in a common data structure to the use of the other parts of the application. There will be at least 1, at most 4 types of maps which will be used in the application. These are a heightmap

(Possibly in raster format) which its existence is a must, a rocky terrain map, one map of the existing rivers and bridges and a map of forests. Class diagram of the module can be seen below.

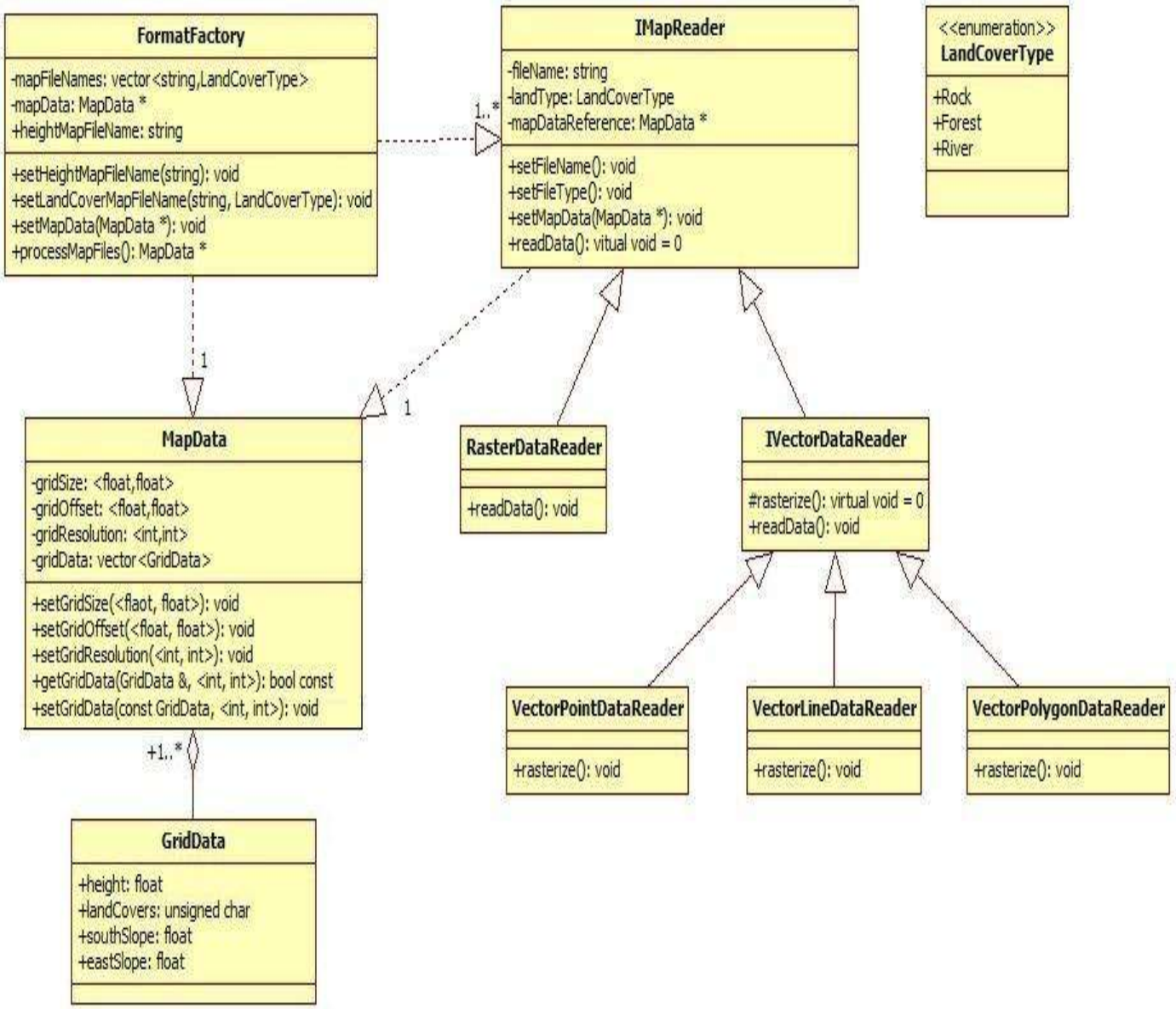


Figure 5.3: MapReader Module Class Diagram

MapData is the main data structure which will be used by all the modules through the application. Because using grids makes the implementation of the shortest path algorithm much easier and efficient, MapData has been designed to keep data in grids. So any possible vector formats should be rasterized and converted to grids. For the sake of simplicity, all maps should be processed in the same resolution by AI module. At this point heightmap data becomes important to keep the grid size and count unique since it will define the resolution of the MapData for the analysis (In case of a vector formatted heightmap is used, resolutions should be pre-defined. We will modify the design in this situation.).

MapData stores the environment information per grid in its GridData array MapData::gridData. GridData is composed of a height value, a hexadecimal group of LandCoverType values called GridData::landCovers, and its slope to its south and east neighbors. MapData instance will also keep its resolution, offset (in latitudes and longitudes) and grids' size (in latitude and longitude).

Because there are variety of GIS formats, format specific properties and projection types, the most critical design aspect of MapReader module is its abstraction of map data. To satisfy a single interface to this module, OOP design pattern “Factory” is used. Called FormatFactory, not only this class has the ability to recognize any pre-defined GIS format and it creates an appropriate IMapReader object to read the data, but also it can manage IMapReader objects to keep the data integrity and process the map data to calculate the intergrid slopes itself.

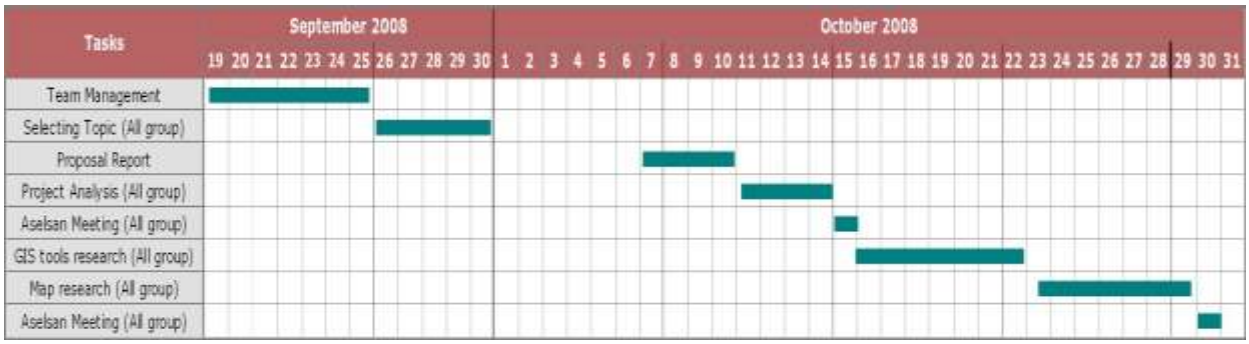
When a FormatFactory instance is created, it will be fed with the map file names which user wants to be simulated. FormatFactory keeps these file names in mapFileNames vector structure. When all the file names are ready, (User may not define file names for all map types, in this situation a default terrain type will be used for that map type.)

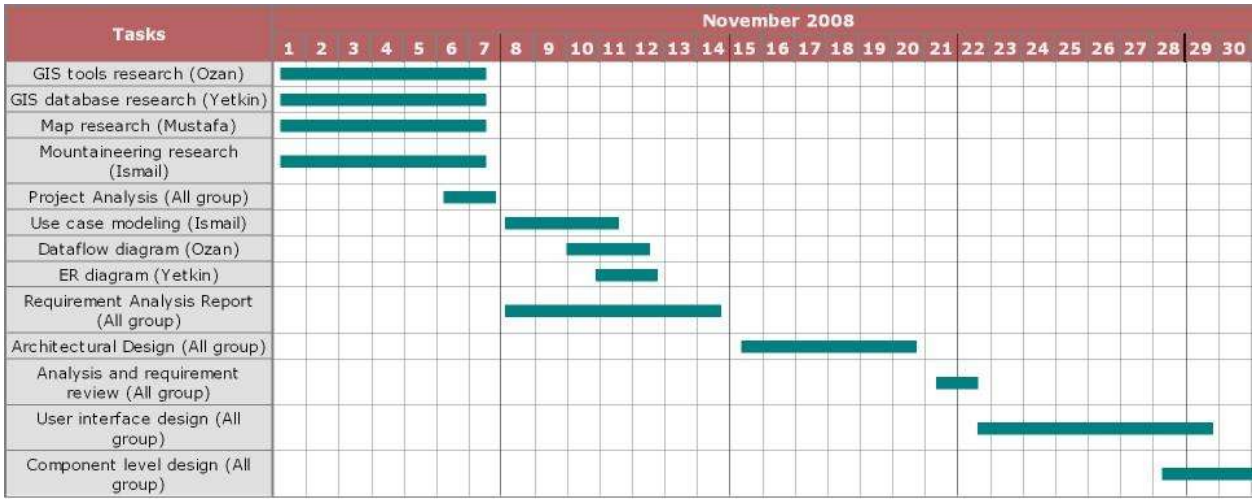
FormatFactory::processMapFiles() is called to start a transformation between any possible GIS formats to MapData grid structure. FormatFactory realizes the formats of the given maps and creates appropriate IMapReader

objects, which will handle the reading and ,if the map is in vector format, rasterizing process of the maps. Firstly the heightmap should be translated to define the resolution of the MapData. After all the conversion, FormatFactory will calculate the intergrid slopes and return a complete MapData object that is ready to be analyzed and visualized.

In the implementation phase, MapReader will make use of OGR and GDAL libraries heavily to translate GIS formats. These libraries present a well defined abstraction of projection and GIS formats.

6 GANNT CHART





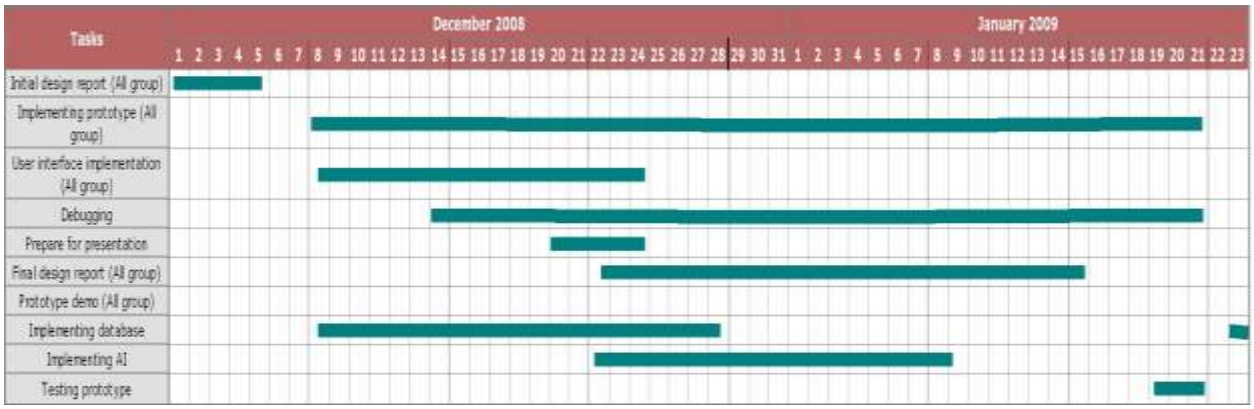


Figure 6: GANNT Chart

7 CONCLUSION

We benefit much from initial design report so when we started final design report we were aware of the importance and benefits of final design report. We are quite aware of the importance of good design so we give our best to design our project and prepare this document. We changed some of design choices since requirement analysis report so we believe that writing a final design report will reflect our project much better than initial design report.

We were not sure about how testing will be done until we start writing this document. Now we determine not only how we will do the test but also what kind of tests we will do, when we will do tests and how we will do the tests.

Modules were determined in requirement analysis report, we also determined classes of each module with their member functions in initial design report and we believe that it will make our job easy in the coming months.

To summarize we believe that aiDSoft team will get benefits of final design report in the coming weeks. This report will guide us throughout the next term.