Middle East Technical University

Department of Computer Engineering

CENG 491
Computer Engineering Design I

Initial Design Report

CodeSchbeke Software Solutions

Sıla Arslan
Çağla Okutan
Hatice Kevser Sönmez
Bahar Pamuk
Ebru Doğan

FALL 2005

Contents

1. INTRODUCTION

1.1 Purpose of the document

This document is intended to introduce the system design considerations that will be basic guideline for the final design specification and prototype implementation. The document gives an initial motivation on how to design the system to meet the requirements previously defined in the Requirement Analysis Report.

1.2 Scope of the document

After giving a general overview, the document specifies the requirements of the system. Graphical user interface of the system is shown and system functionalities are described with the help use case and activity diagrams. System modules along with their classes are described in detail giving the relationships between them. Lastly, a dynamic Gantt chart is presented.

1.3 Abbreviations and Definitions

2D: Two Dimensional
3D: Three Dimensional
AVI: Audio Video Interleave
DTD: Document Type Definition
GUI: Graphical User Interface
MP3: MPEG Audio Layer 3
ODE: Open Dynamics Engine
OSG: OpenSceneGraph
WAV: Waveform Audio
XML: Extended Markup Language

2. SYSTEM OVERVIEW

2.1 System Description

The system TraffEdu is a 3D editor for designing 3D traffic environment and preparing traffic case simulation on it. It provides the user to prepare any traffic case which includes creating road map, inserting traffic lights or traffic signs, environmental objects, audio or text, any type of vehicles and determining behaviors of the vehicles in the traffic. According to these specifications a XML formatted file will be produced.

Animation will be constructed after user presses the animation button by reading data from that XML file. This animation will not be in the format of AVI file which does not allow user interaction. It will enable the user to change view of the case by translating or rotating the camera.

2.2 Functional Requirements

Functional requirements of the TrafEdu System are mentioned in the Software Requirements Analysis report. Functionalities below are the ones that are changed or added to the system after the analysis phase.

1. In the analysis report it is mentioned that TraffEdu Editor shall provide the user to define traffic lights' synchronization. Instead, in the timeline user sets behavior of the traffic light (red, yellow, or green) for each key frame and system adjusts synchronization accordingly.
2. In the analysis report it is mentioned that while constructing the road map of the environment, user will see 2D symbolic representations of the real 3D objects in 2D environment. Instead user can see different projections (orthogonal projection from top, orthogonal projection from front, orthogonal projection from side and perspective projection by a user defined angle) of the 3D environment in four sub windows.
3. TraffEdu system shall provide the user to hide or display sub windows mentioned above.
4. TraffEdu system shall provide the user to hide or display menus in the GUI.
5. Functionality of choosing the hours of the day will not be supported.
6. Functionality of creating an executable animation file will not be supported.
7. Functionality of changing behaviour of a vehicle during the animation phase will not be supported.

8. While user is constructing roadmap, adding traffic signs or constructing special case, the user's conflicting actions (e.g. locating the car to a position outside of its path) will be prevented by giving a warning message to direct the user to correct her/his fault.

2.3 System Requirements

TraffEdu System requires Microsoft Windows 98/2000/NT/XP Operating System and DirectX support on the user side.

On the developer side the requirements are listed below:

- Microsoft Windows XP Professional
- Microsoft Visual Studio .NET 2003
- OpenSceneGraph 0.9.9
- Open Dynamics Engine 0.5
- DirectX SDK
- 3D Studio Max
- Adobe Photoshop
- LaTEX

3. INTERFACE DESIGN

3.1 Sample Graphical User Interfaces

The following screen capture is what TraffEdu Editor will look like when a new file is opened. The Window is mainly divided into four areas which is toolbar (or menu bar) at the top, Toolbox Window at the left, main window(OSG windows) at the middle and Objects and Properties Windows on the right of the editor window. The events occurred in OSG windows will be handled by `EventHandleOSG` class and events occurred in GUI windows will be handled in `GUIEventHandler`. User will use the orthogonal view (which is the top left view in the following picture) for creating his/her traffic environment. The addition of objects to the environment is supported via Toolbox Window on the left of the window. A grid platform is provided for better placement of objects into the area.

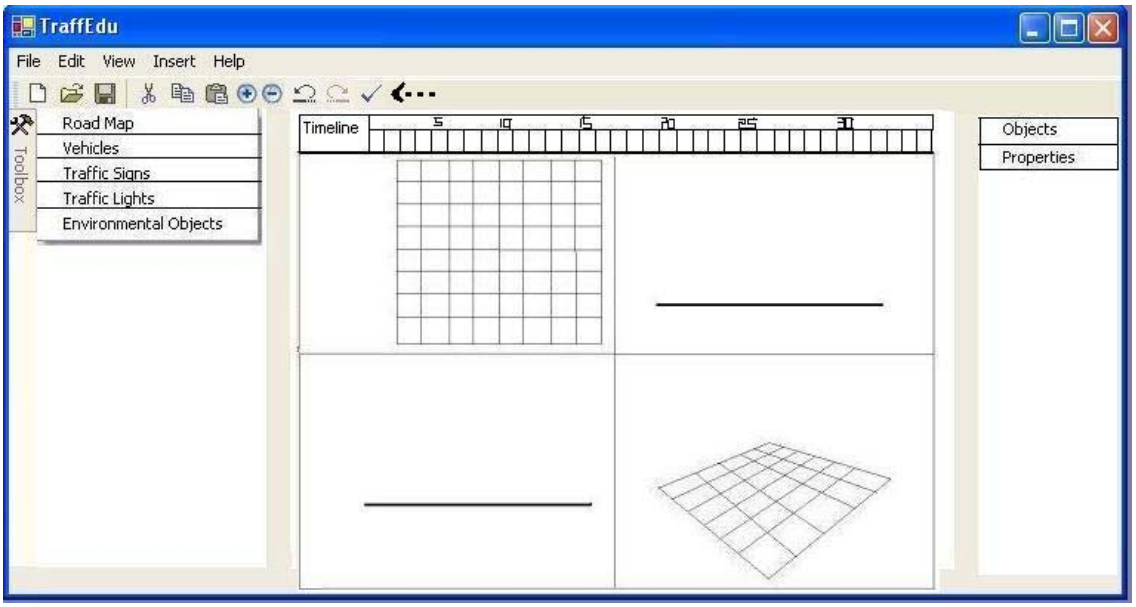


Figure 3. 1

Creation of a traffic case starts with placing appropriate road type onto the grid platform. This is provided in the Road Map section of Toolbox Window. The user will drag & drop the needed road types to the platform. If an inappropriate kind of road is placed that cannot be joint to the neighbor road type, this is prevented by the system.

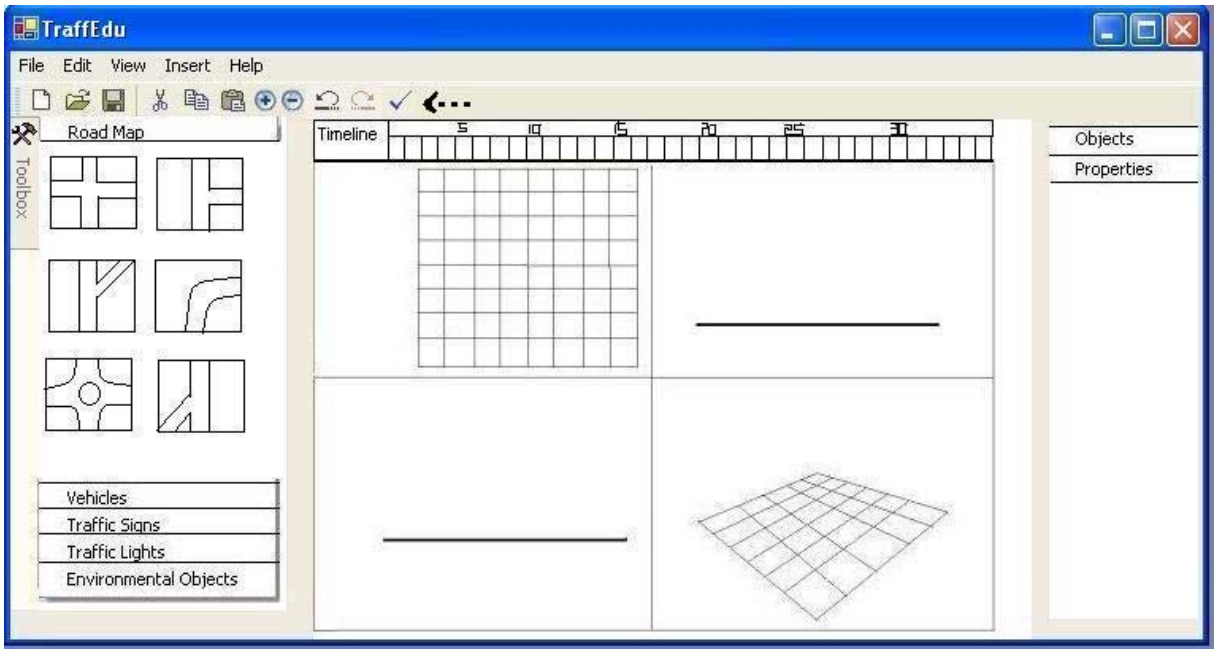


Figure 3. 2

For placement of Vehicle objects user can drag and drop from Vehicles section of toolbox. Here there will be types of vehicles that the user can select for using in the traffic case being designed.

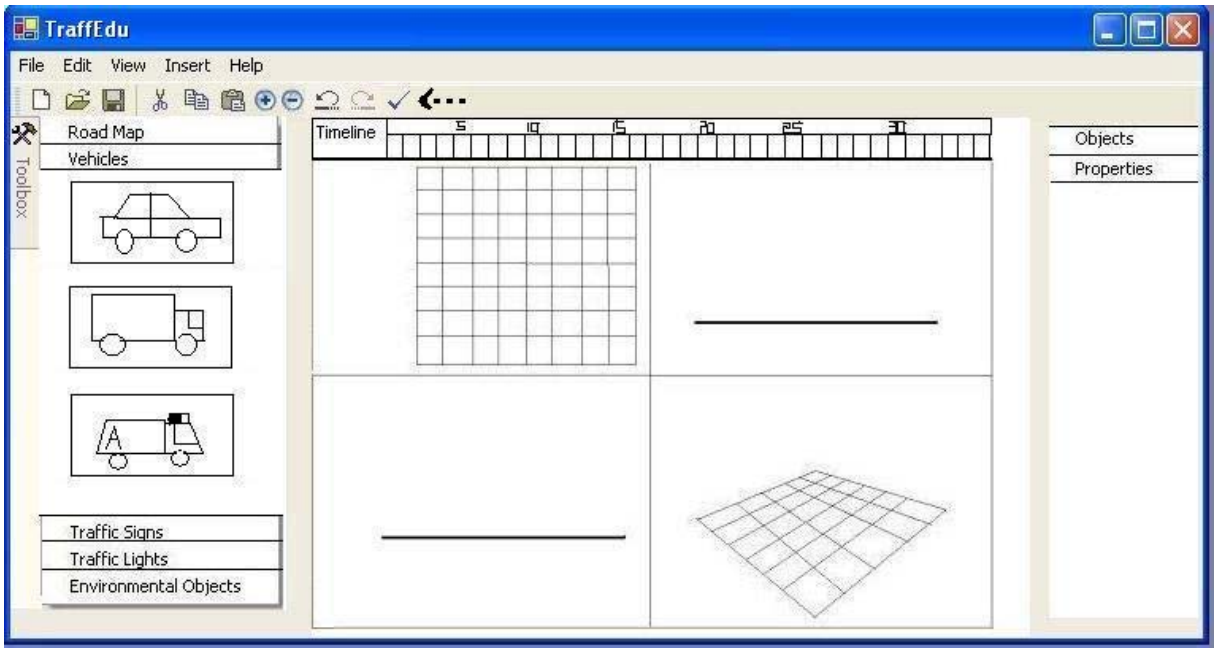


Figure 3. 3

For addition of the traffic signs and lights again the toolbox is used and the user has the opportunity to add them to the appropriate places in the road map designed.

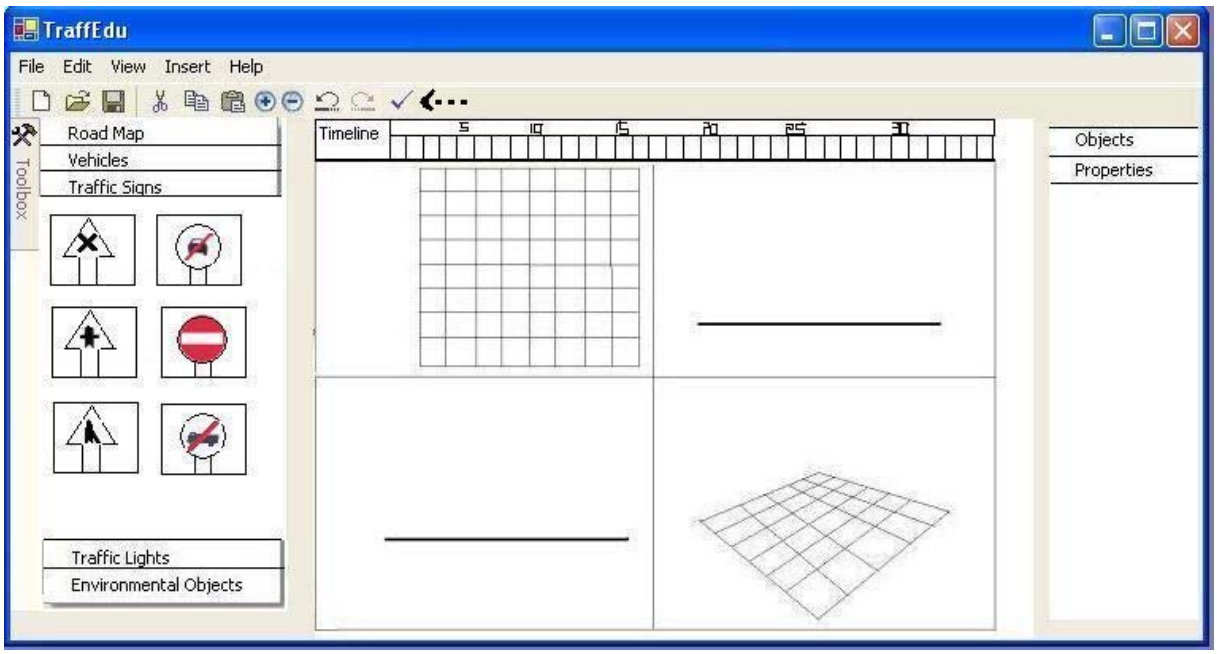


Figure 3. 4

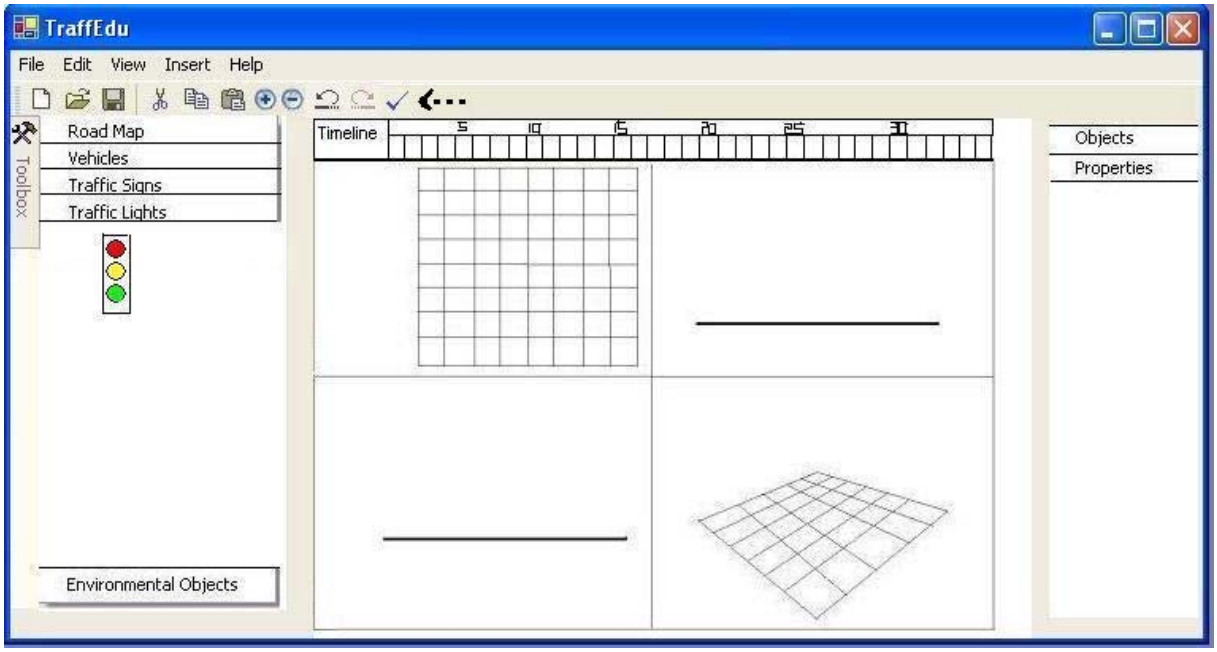


Figure 3. 5

The user can insert environmental objects house and trees from the Environmental Objects section of the toolbox.

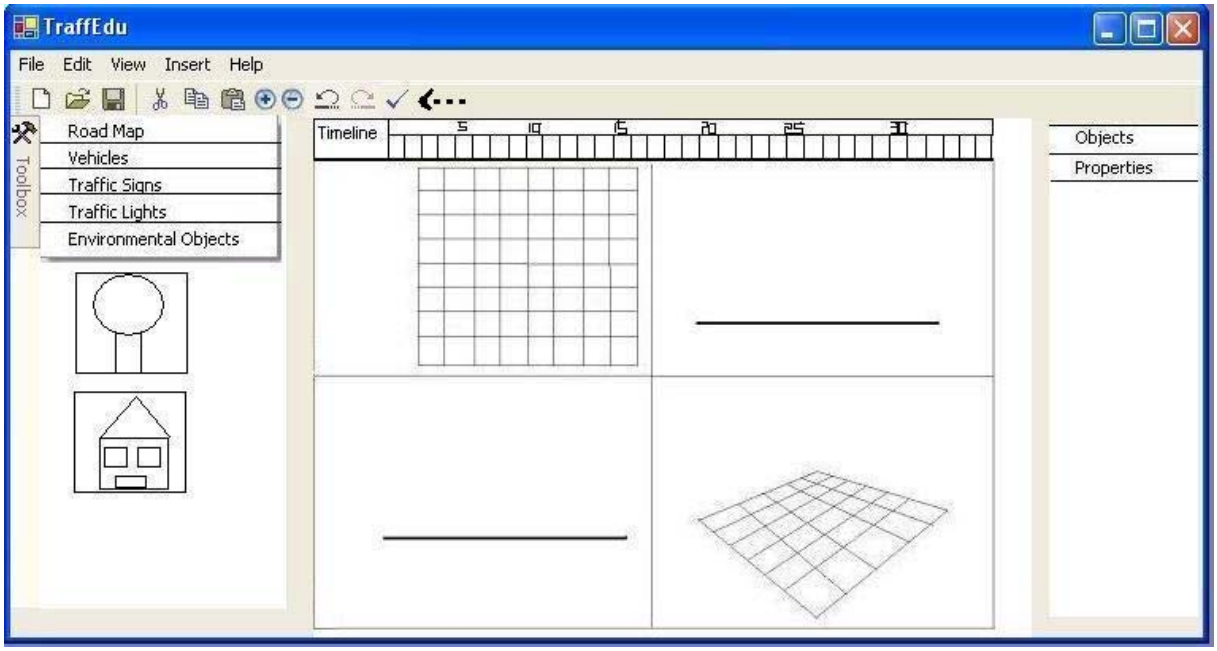


Figure 3. 6

The current objects in the environment can be seen from the Objects Window on the right panel of the window.

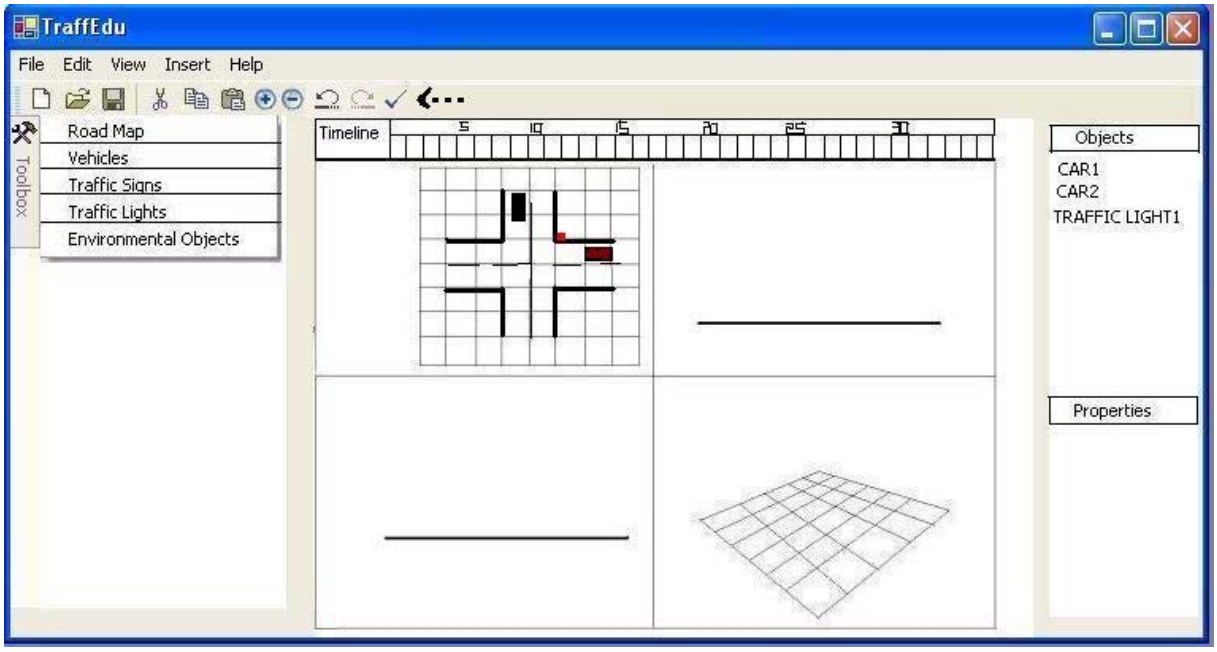


Figure 3. 7

Clicking on an object listed in the Objects Window shows the current selected object's properties in Properties Window below.

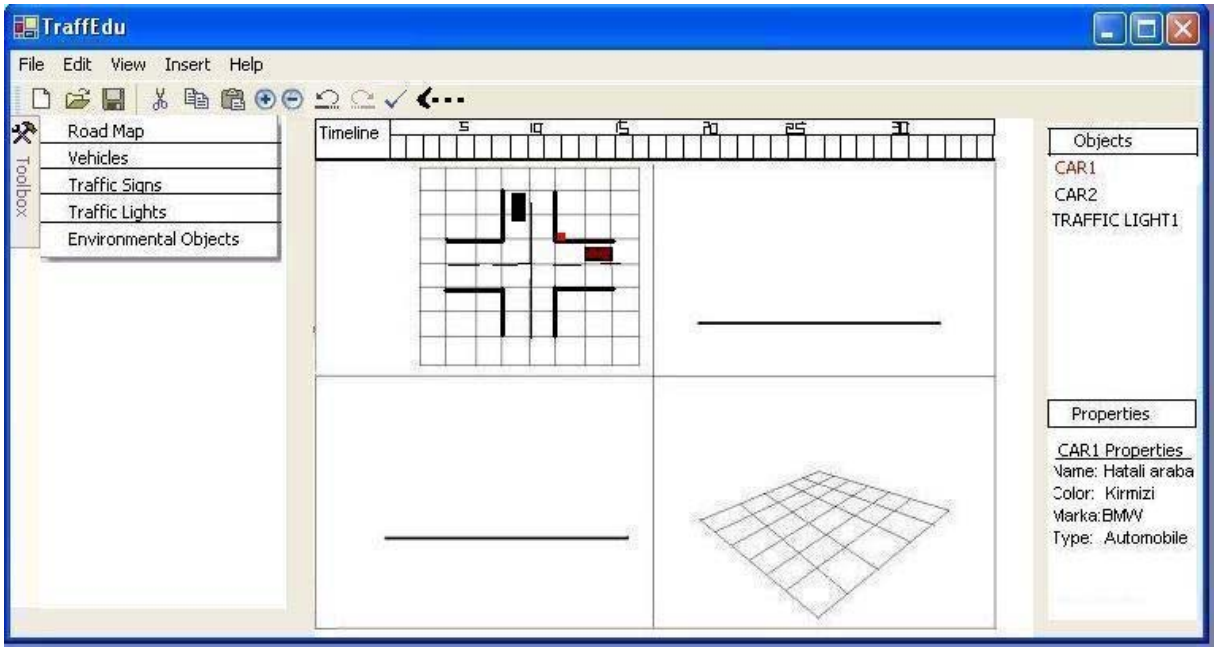


Figure 3. 8

The File menu presents the following actions to the user. Current Files is for viewing the currently opened files.

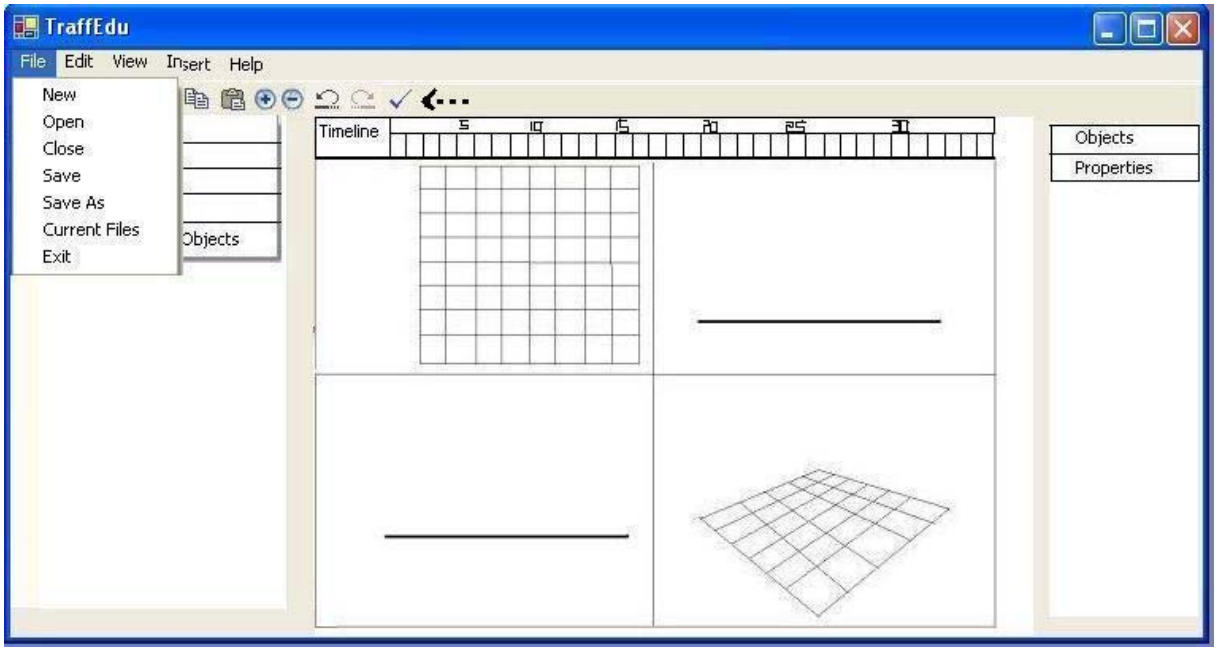


Figure 3. 9

Under edit menu the following actions may be performed. These actions will be active when an object is selected. Only Zoom In/Zoom out property does not require selection of an object which means zooming with respect to the current camera direction. When an object is selected zooming action is done centering that object selected.

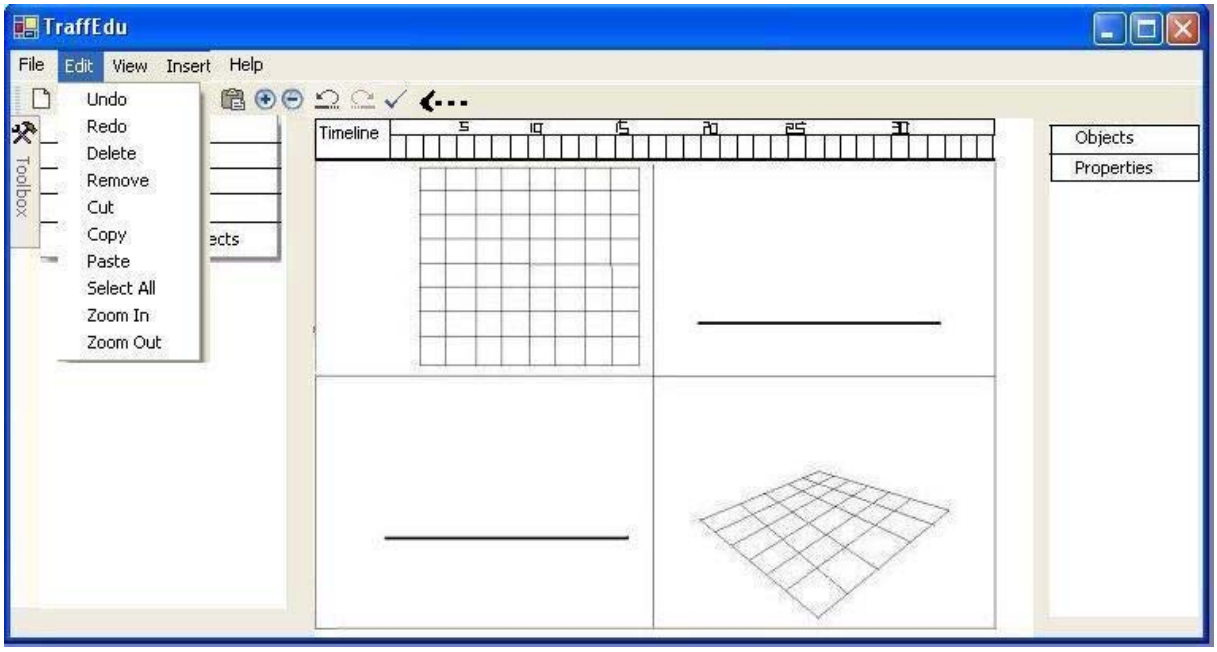


Figure 3. 10

Removing a class of objects or all objects can be done via Edit->Remove.

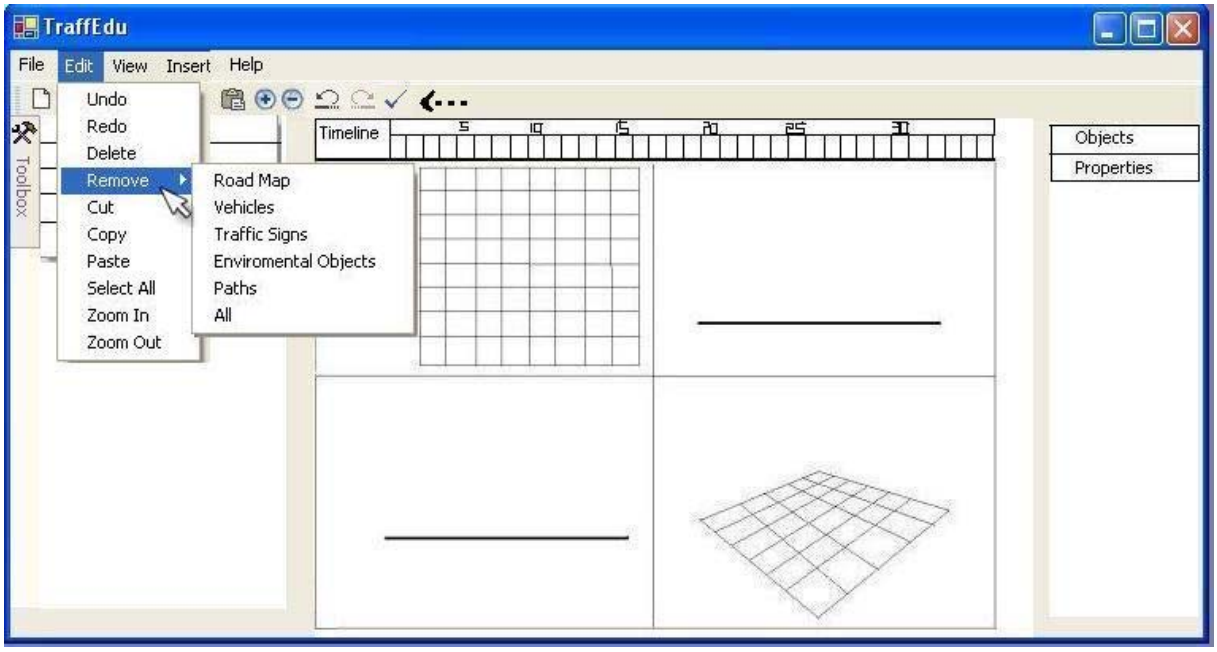


Figure 3. 11

View menu gives user the chance of viewing/not viewing objects or panels on the environment and the Editor window.

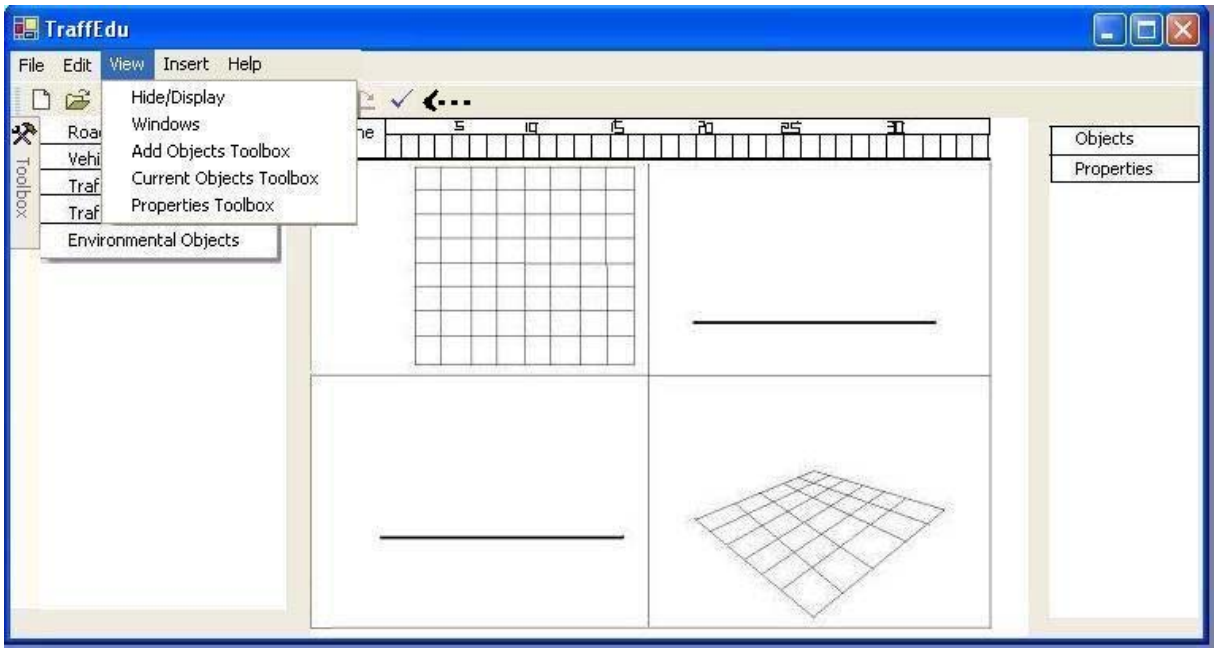


Figure 3. 12

The user will be able to hide or display objects as shown below.

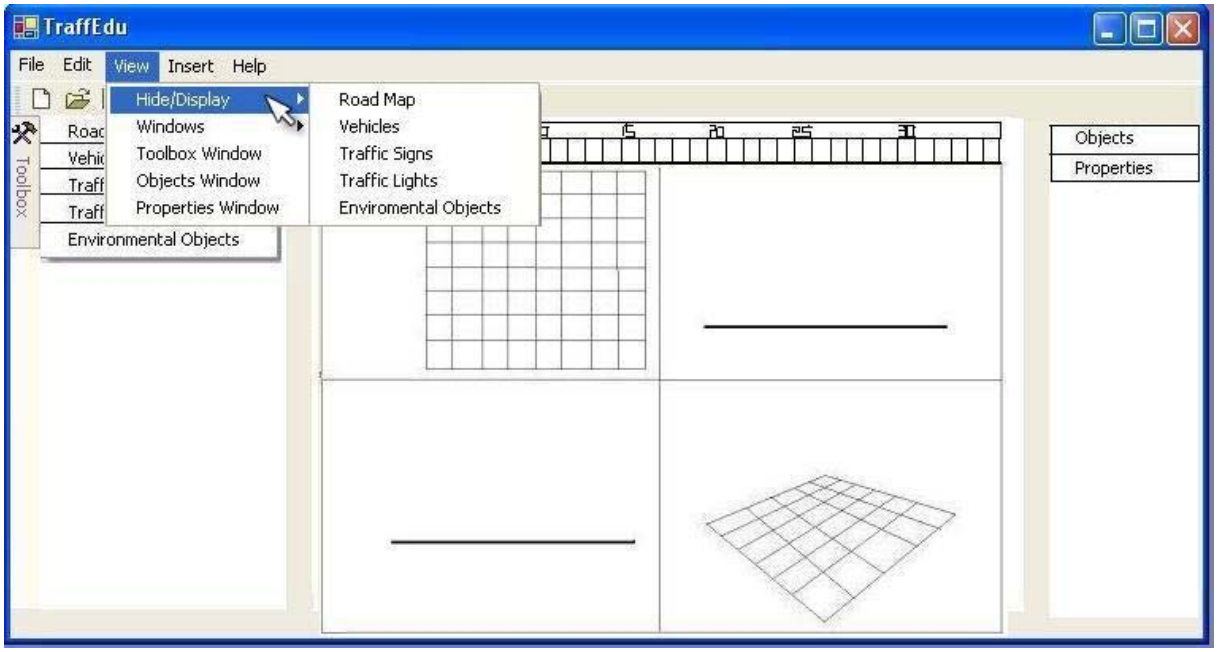


Figure 3. 13

The user will have the opportunity to see the environment prepared, with Top View, Front View, Side View and 3D View options. The checked ones will be provided in the main area of the editor.

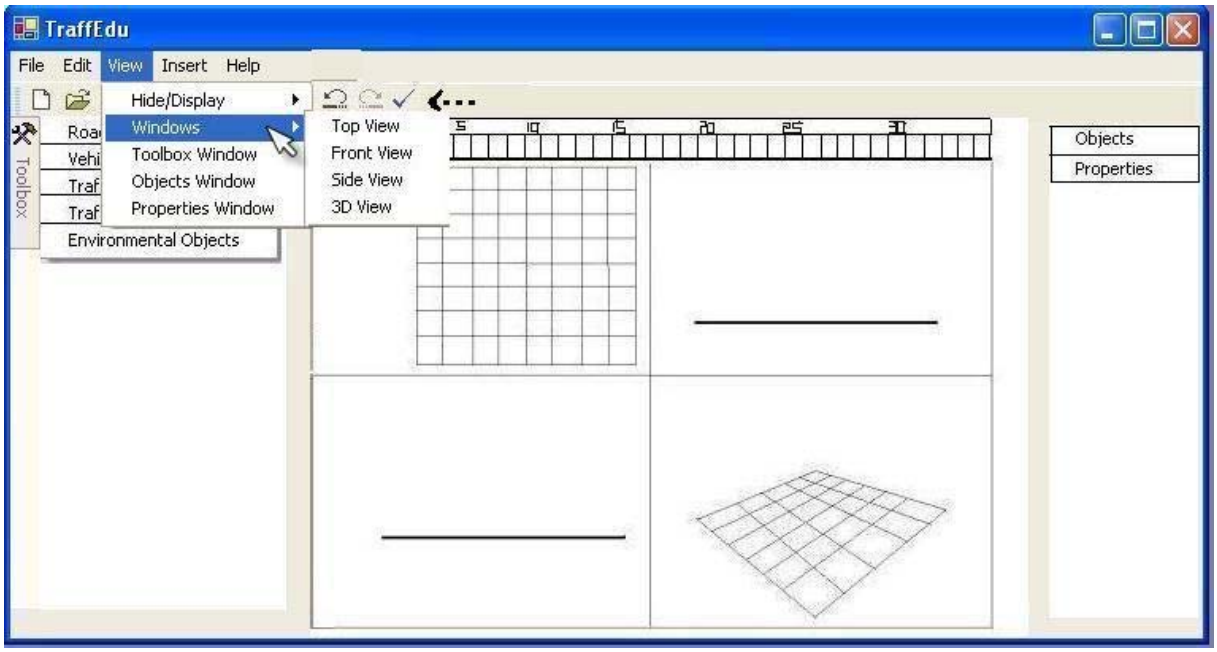


Figure 3. 14

The user will be able to insert objects, audio and text to the case within this Insert Menu and also save the currently formed environment as a frame by selecting Frame. When Text is clicked, a pop up window will appear allowing the user to insert any text into the current frame. In the same way the user will be able to insert any audio file into the frame by clicking Audio menu and selecting an audio file of her/his choice.

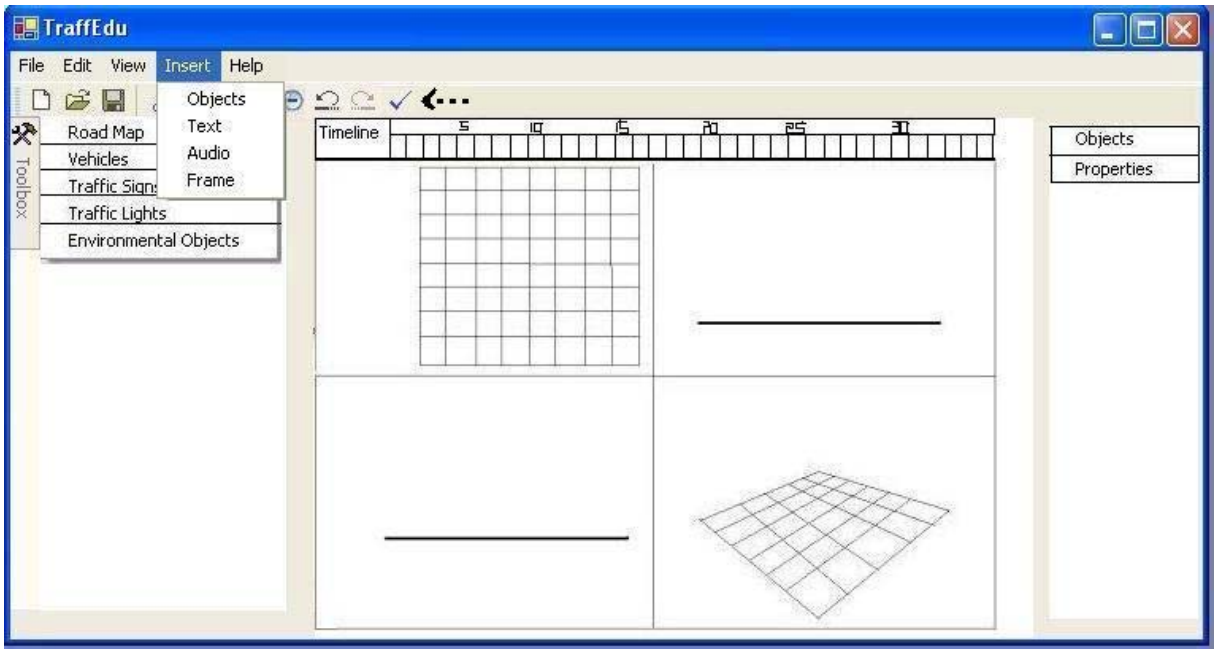


Figure 3. 15

Help menu presents the manual for TraffEdu and also some pre-created sample maps with explanations for easing program usability.

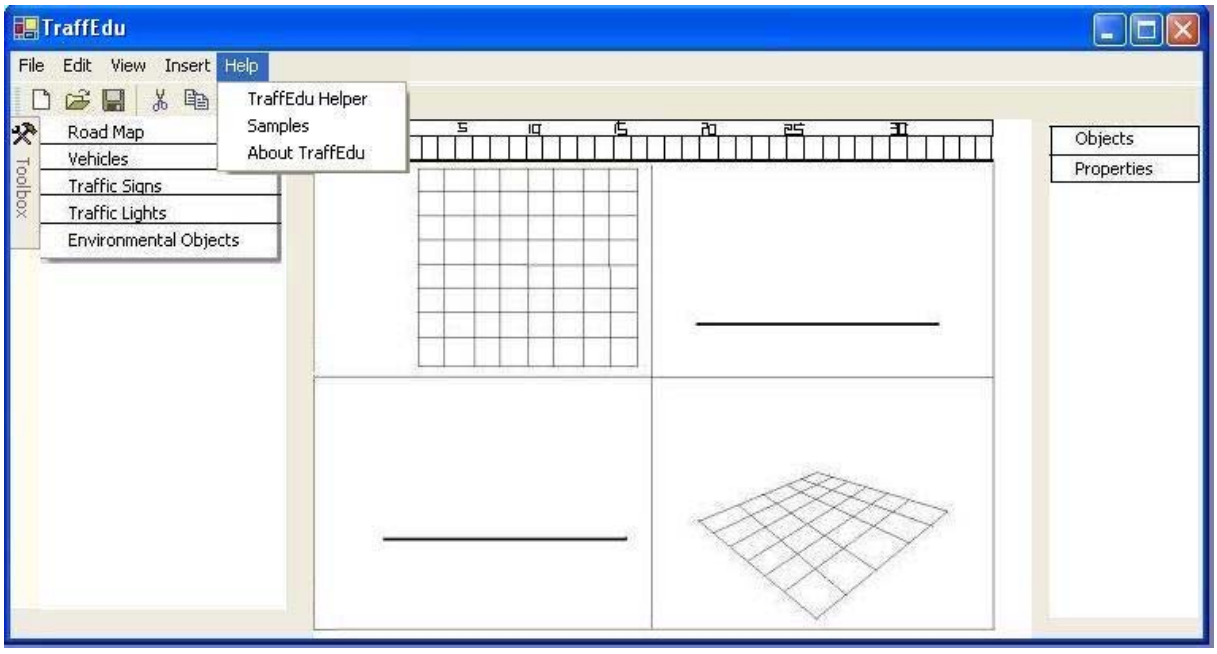


Figure 3. 16

For adding a path to the vehicles a toolbar icon is provided. Also the user can right click on the car object to draw its path or do some actions related with the car like cut, copy, etc.

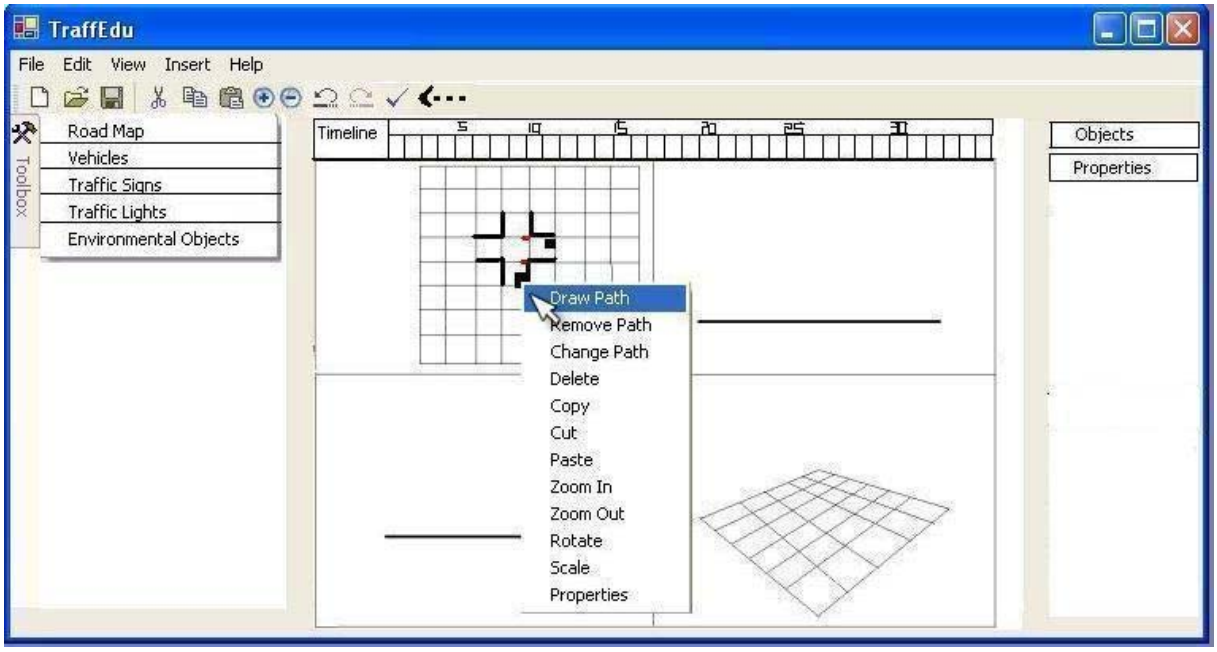


Figure 3. 17

Drawing path of the vehicle is done by joining the small squares that is presented as soon as the Draw Path icon from toolbar is clicked or the user right-clicked the vehicle object and chose Draw Path. For changing a path the user will hold the arrow in front of the path line and drop to the target place. The user will stay in this mode until it clicks the icon again. Removing of the path may be done by again right-clicking on the vehicle and clicking Remove Path.

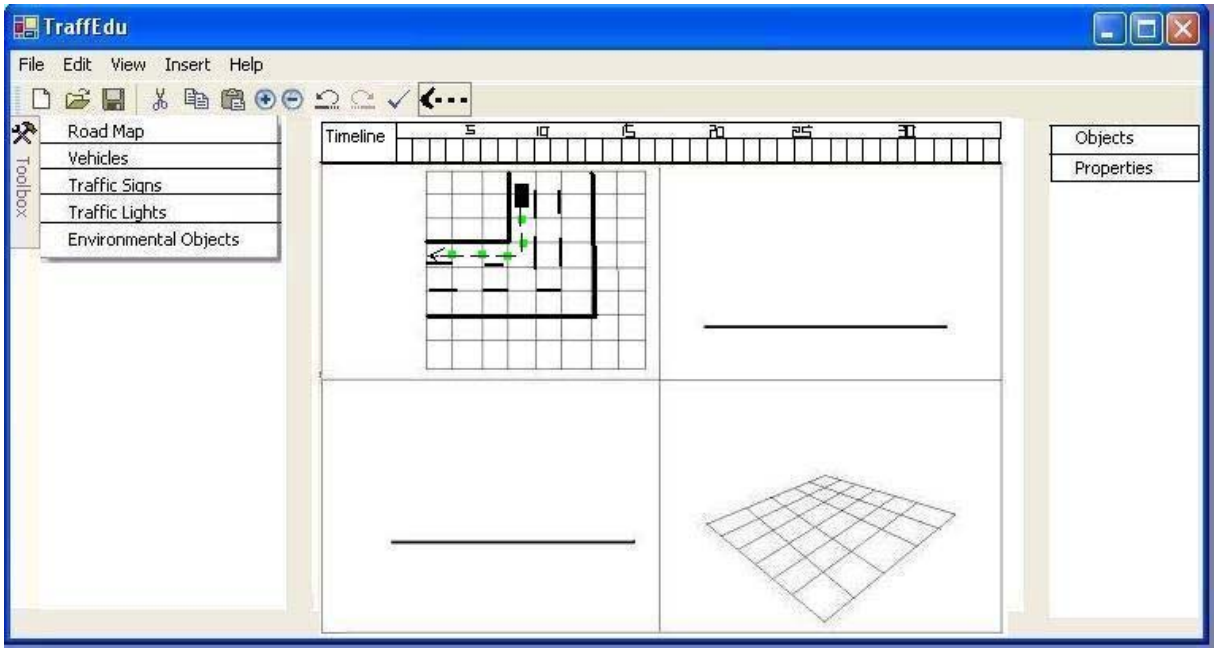


Figure 3. 18

Right-click action on a road object presents the following menu.

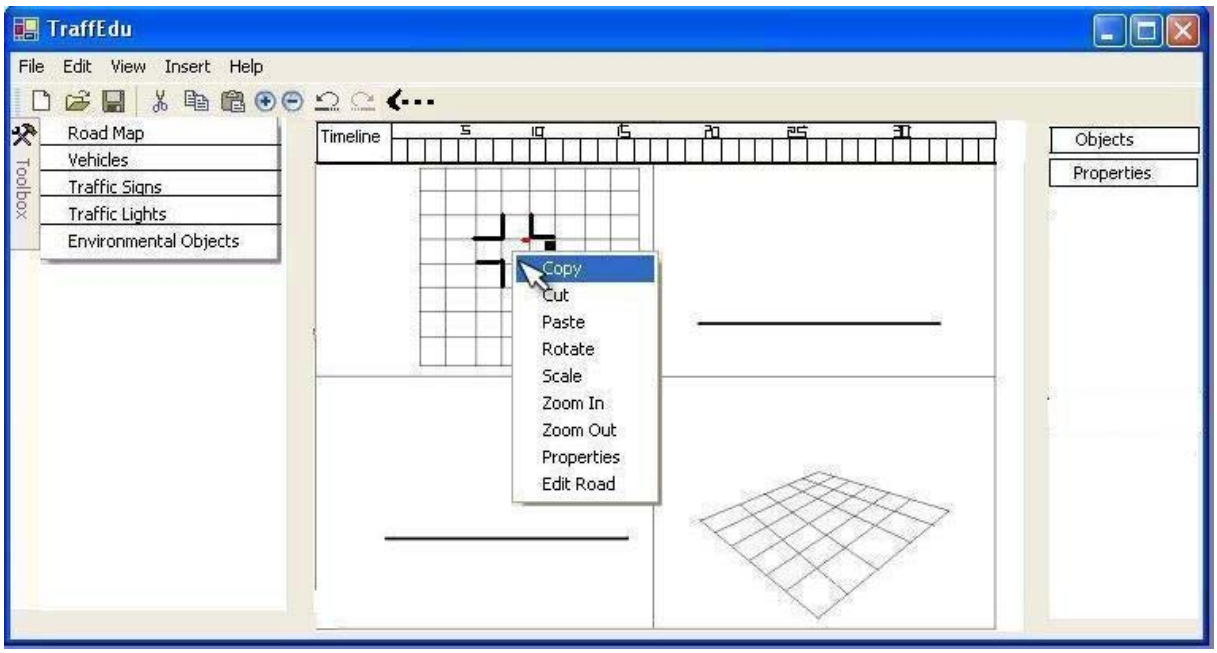


Figure 3. 19

If the user clicks on Edit Road the following menu will pop-up and the user will have the chance of determining road properties like count of lanes and direction of lanes. Also the type of the road lines will be determined. In later steps if another road object is added to the environment and has a join with this road object, the directions of the lanes will be checked.

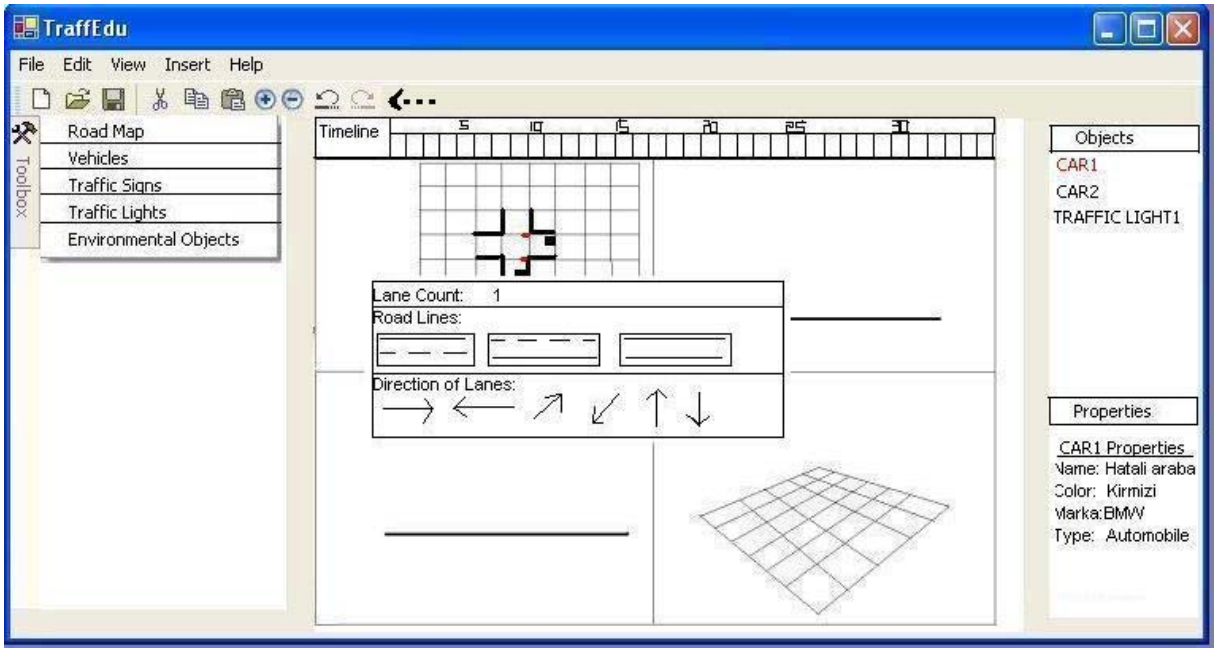


Figure 3. 20

Right clicking on an empty grid shows the following menu.

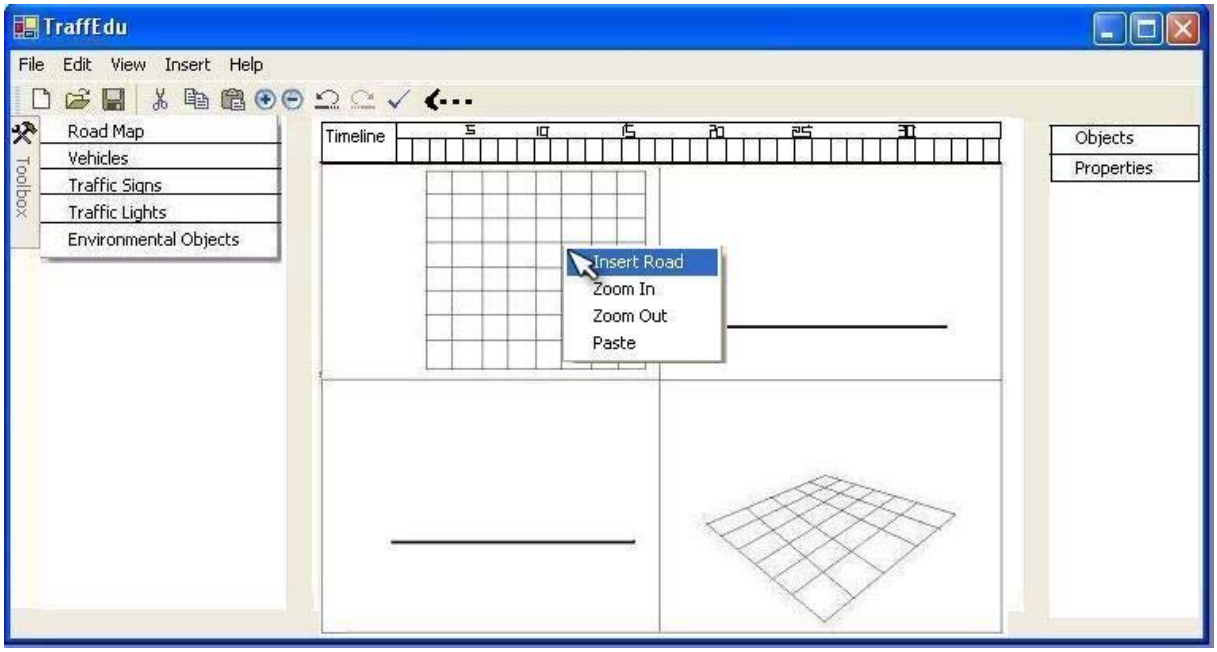


Figure 3. 21

If the user places two car objects on top of each other, a warning message is presented in order to make certain that the user wants an accident indeed.

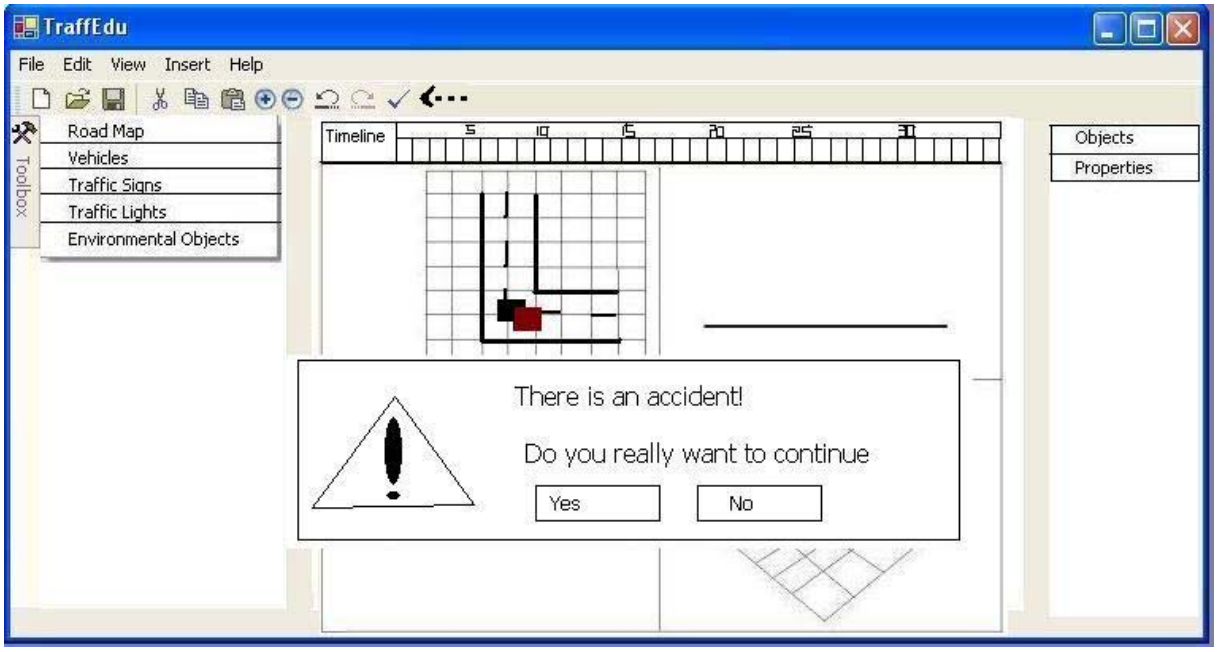


Figure 3. 22

User will first click on a frame from the timeline and then starts designing the case. When the user finishes with the current frame, either right clicking the selected frame on the timeline or by Insert->Frame menu, the frame will be saved in the system. Removal of a frame is done again by a right-click action on the frame.

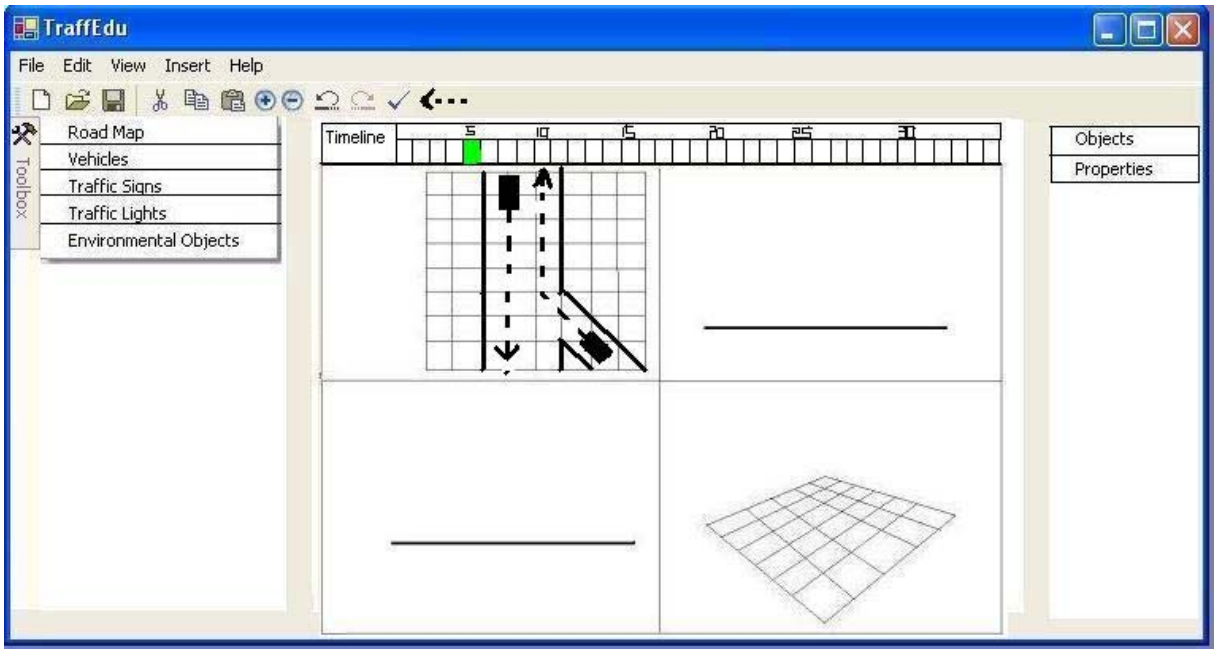


Figure 3. 23

The system forces placement of vehicle objects on their path not anywhere else, in each frame.

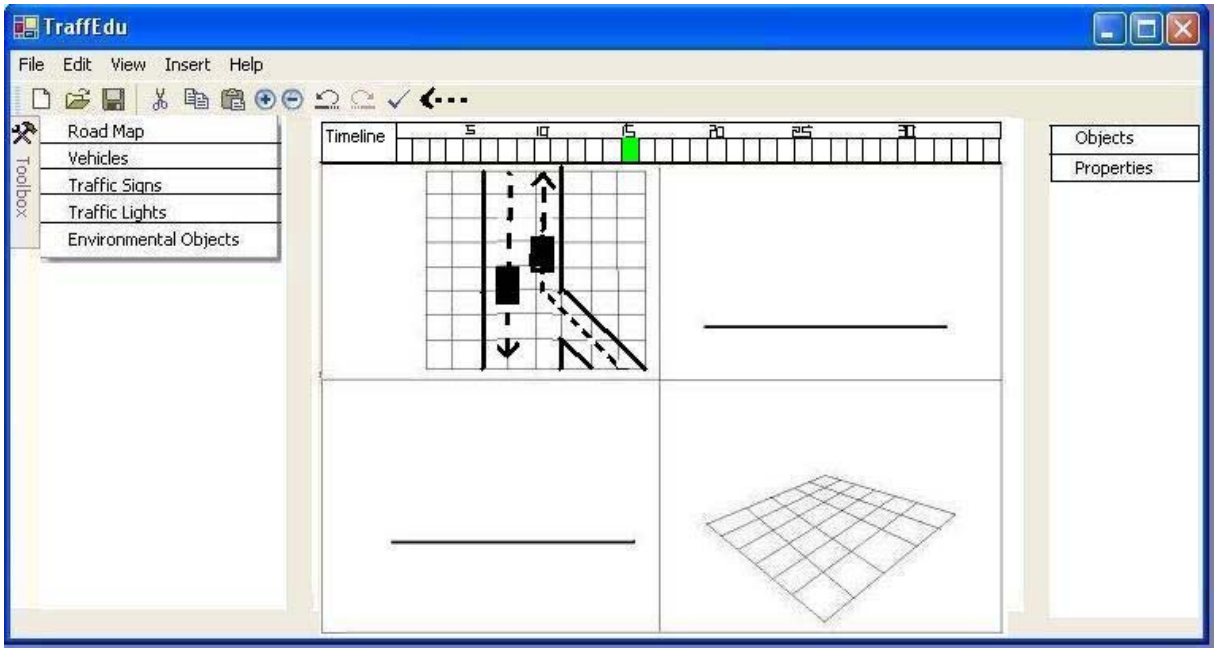


Figure 3. 24

3.2 Reviewed Use Case Diagrams

The functionalities of the system and the actions that can take place by the user are depicted clearly in the GUI of the program. However there exist some actions that must be done sequentially and dependently to some other actions. These dependencies that are not dictated in the GUI are represented with use case diagrams in Figure 3.25.

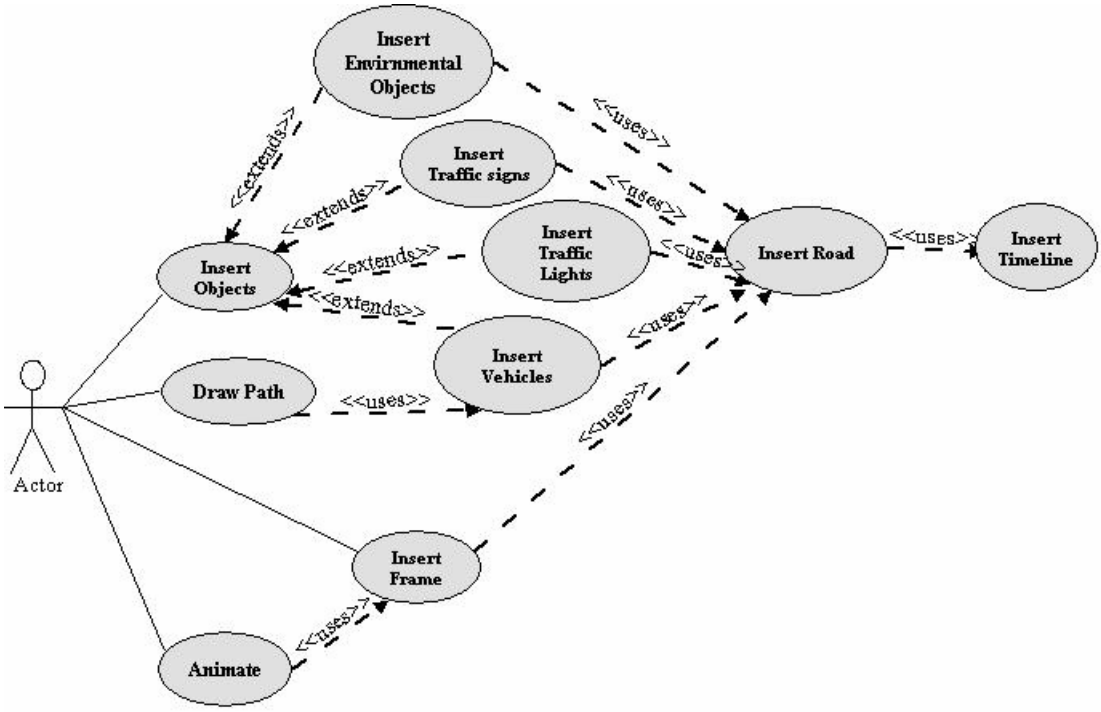


Figure 3. 25

If the user wants to insert an object (i.e. traffic sign, traffic light or vehicle) into the map, he should first insert a road on which the other objects must be located or he should insert the objects on an already inserted road. There could not be any object residing on the grids of the editor except for the road and the environmental object.

If the user presses on animate button, he should have inserted any frame to the system that he plans to constitute.

For a frame to be inserted there must be any road located on the grids previously to be displayed in the animation phase.

Lastly, all the insertions should be done after inserting a timeline. For the system to be informed about the starting, ending and characteristic times of the cases, the timelines of each case should be inserted before preparing the positions of the objects.

3.3 Activity Diagram

In TraffEdu system, from opening a new file to construct a new traffic case to pressing animation button and watching the simulation, several sequences can be followed. Activity Diagram below shows one of those sequences to make working mechanism of the system clearer.

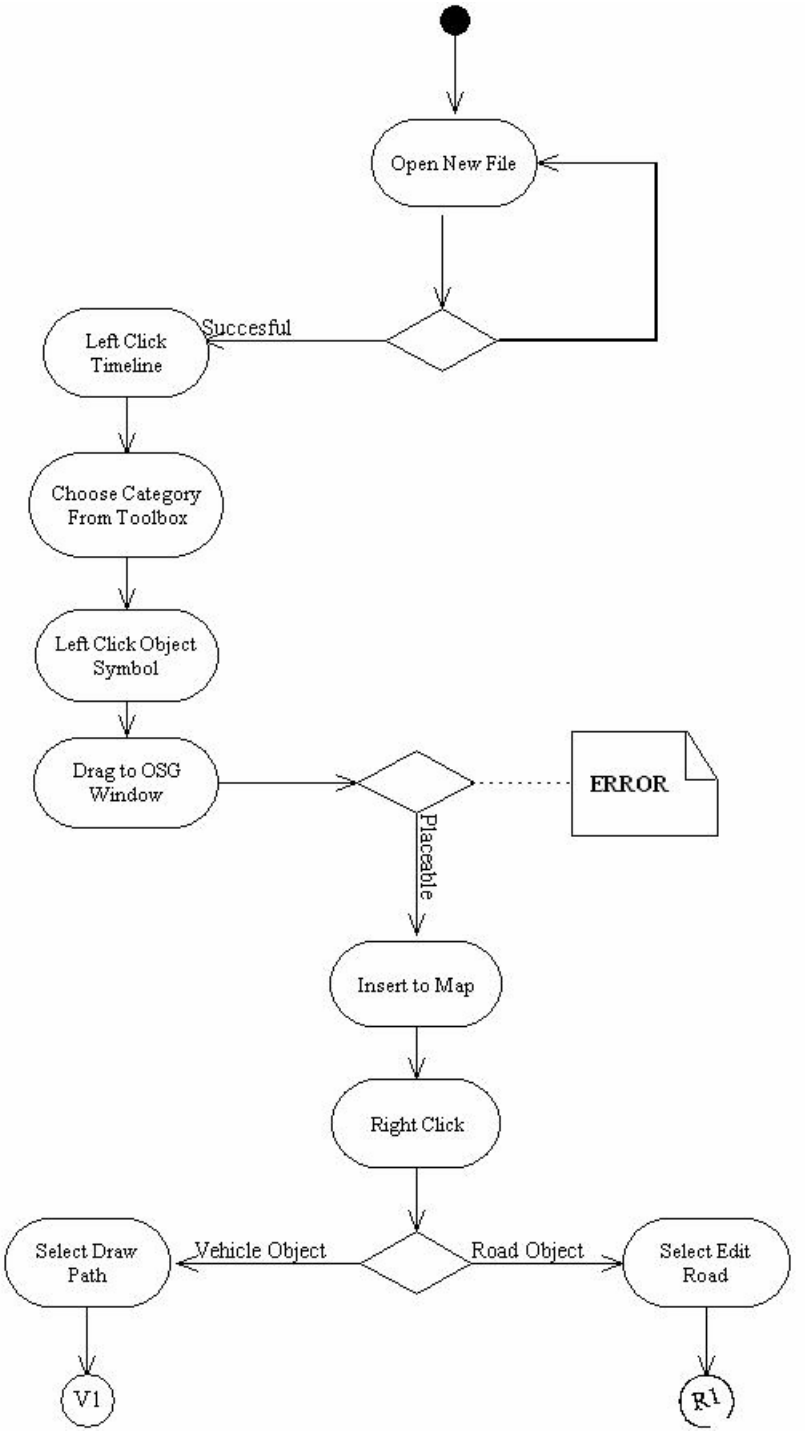

Open New File
Succesful
Left Click Timeline
Choose Category From Toolbox
Left Click Object Symbol
Drag to OSG Window
ERROR
Placeable
Insert to Map
Right Click
Select Draw Path
Vehicle Object
Road Object
Select Edit Road
V1
R1

Figure 3. 26

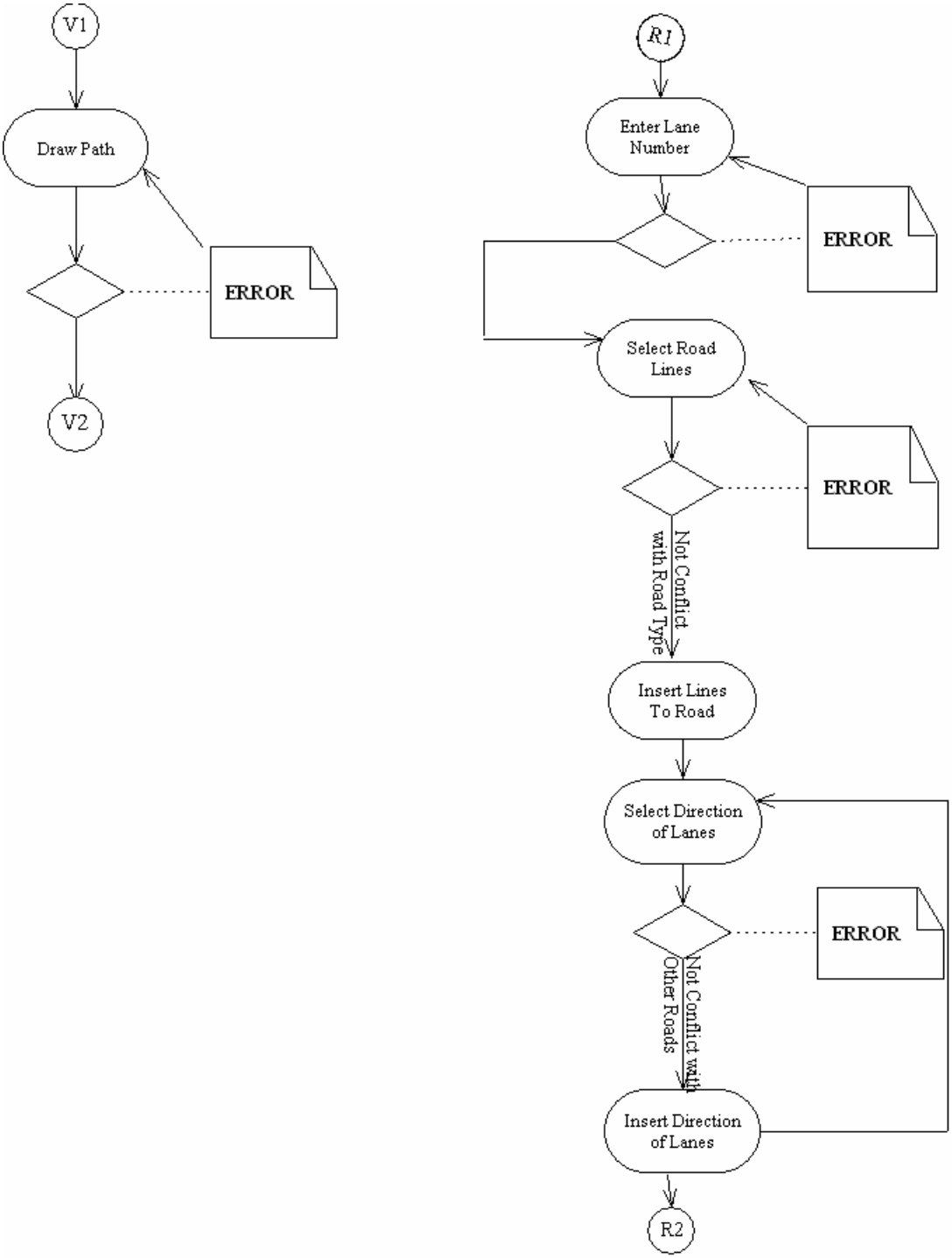

V1
Draw Path
ERROR
V2
R1
Enter Lane Number
ERROR
Select Road Lines
ERROR
Not Conflict with Road Type
Insert Lines To Road
Select Direction of Lanes
ERROR
Not Conflict with Other Roads
Insert Direction of Lanes
R2

Figure 3. 27

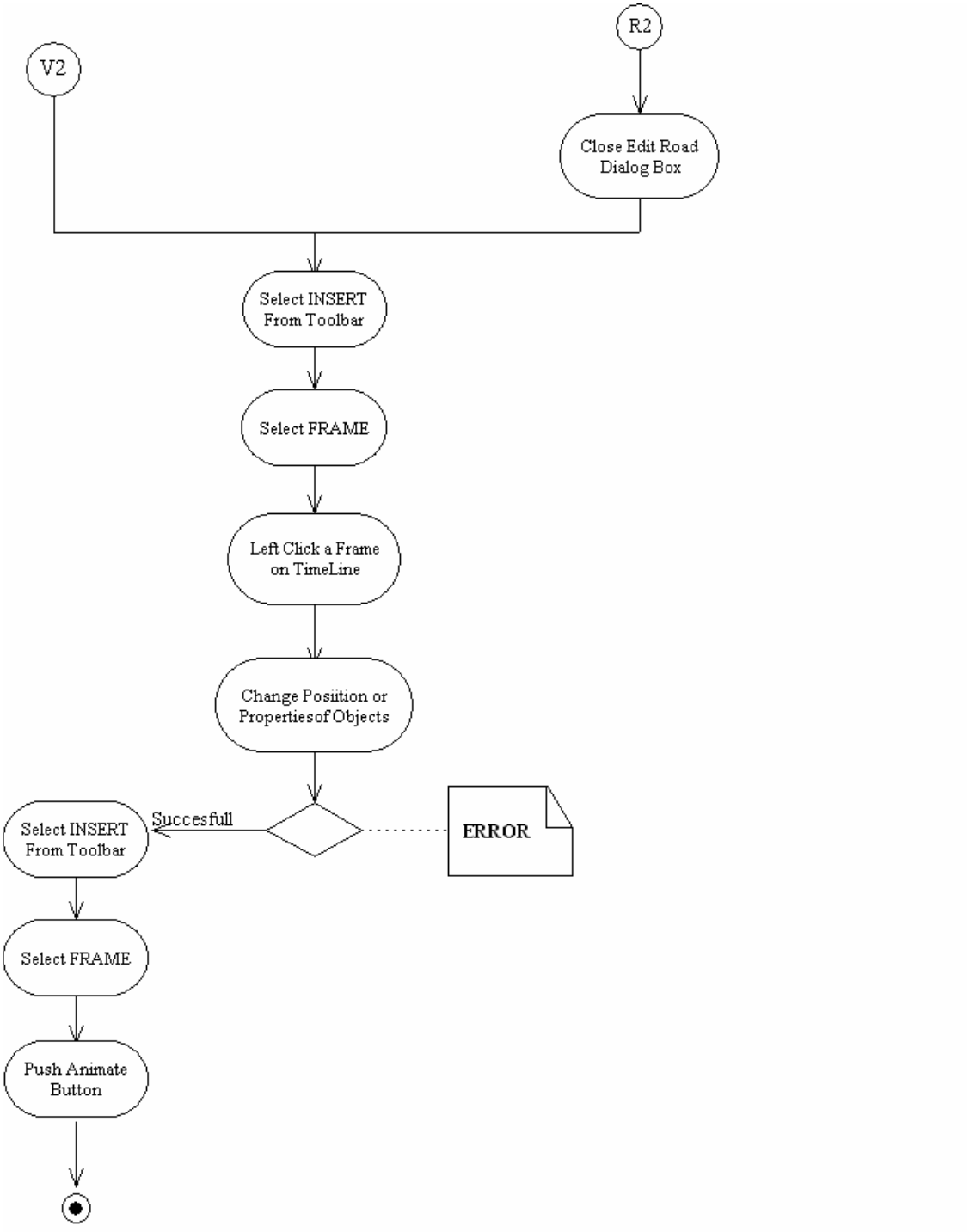

R2
V2
Close Edit Road Dialog Box
Select INSERT From Toolbar
Select FRAME
Left Click a Frame on TimeLine
Change Posiition or Propertiesof Objects
Succesfull
ERROR
Select INSERT From Toolbar
Select FRAME
Push Animate Button

Figure 3. 28

4. SYSTEM DESIGN

4.1 System Data Structures

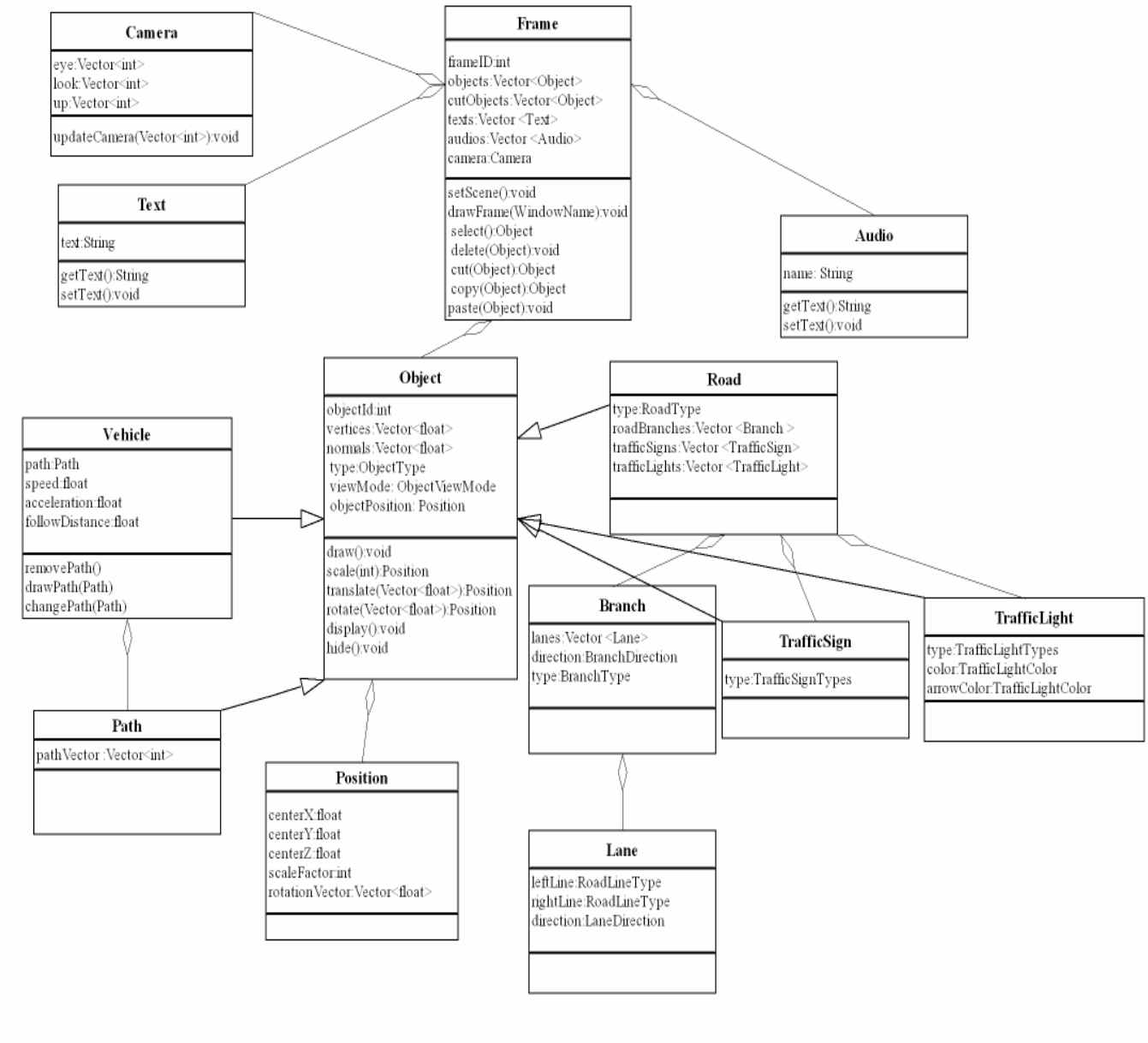


Figure 4. 1

Frame Class:

`Frame` class is an aggregate class that contains all elements specific to a frame. Objects of scene, such as vehicles, roads, traffic lights, traffic signs; camera, text, and audio are all inside this class.

`setScene` method will arrange the camera position that is inside the frame.

`drawFrame` method calls draw methods of all the objects in the objects vector. Other methods `select`, `cut`, `copy`, `delete` or `paste` an object in the frame. These methods change the object vectors in the `Frame` class. Whenever an object is cut it is removed from `objects` vector and put in `cutObjects` vector, so it is not drawn in the scene. But whenever an object is copied it is put in `cutObjects` without removing it from the original list. To paste an object last element of the `cutObjects` is added to `objects` vector. Deleting an object is simply removing it from the `objects` vector.

Camera Class:

Whenever a keyboard callback related to camera positioning happens, `EventHandlerOSG` class request an update for the camera position, the camera object in the current frame is then updated using the `updateCamera` function of this class. This function resets the point where the camera is, the point where it looks and the up vector of the camera.

Audio and Text Class:

These are the classes of inserted text and audios to the frame. `Text` class' string field keeps the text that user inputs and `Audio` class' string field keeps the name of the audio file. These are both saved to the related frame part of XML file. Whenever the frame is rendered, the `Frame` object will call `Audio` module's `Audio` class' (this is not the `Audio` class of `Frame` class) `playSound` function giving the audio name as the argument.

Object Class:

`Object` class is the parent of classes that denotes the objects to be drawn in 3D environment. Since vehicles, roads, traffic signs and traffic lights along with the environment objects such as buildings and trees all have similar properties; the functions manipulating these will be implemented in the `Object` class. Every `Object` has a position so any information related to `Object's` position is kept in object, as 'hasa' relationship in `Object` and `Position` classes.

`draw` function will use the OSG functions to render the object at the given position.

`scale` function will change the scaling factor in the `Position`.

translate function will change the center coordinates of the object in the Position.

rotate function will change the rotation vector in the Position.

display function sets the object's viewMode to DISPLAY.

hide function sets the object's viewMode to HIDE.

Position Class:

This class is related to the object class with 'hasa' relationship, since every object should have attributes like its center coordinates, its rotation vector and its scale factor.

Vehicle Class:

Vehicles have properties such as acceleration, speed, path and followDistance that are specific to vehicle objects.

drawPath method takes a Path object and copies it to Vehicle's path attribute. By this way the path and the corresponding vehicle is related.

removePath method deletes the path attribute of a Vehicle object.

changePath method reloads path attribute of a Vehicle object.

Path Class:

Path class has only one attribute which is an integer vector that keeps the indexes of grids where the path is on. These indexes are the grids vector indexes which is an attribute of OSGWindows class.

Road Class:

Road class also inherits from Object class with some specific attributes like its type, its branches and vectors of TrafficSign and TrafficLight that are on the road. The reason why Road class has 'hasa' relationship with TrafficSign and TrafficLight classes is, they are static objects and their existence depends on the road type and existence of the road. Whenever a road is removed from the map they are also removed and a road should already exist wherever a traffic sign or traffic light is placed and their consistency should be checked with that road by the EditorChecker class.

Road objects are composed of branches and branches should be accessed from Road since the user can manipulate each Branch by setting its attributes such as increasing the number of lanes, assigning direction to the lanes which the vehicles should follow and determining road lines.

4.2 GUIEventHandler Class

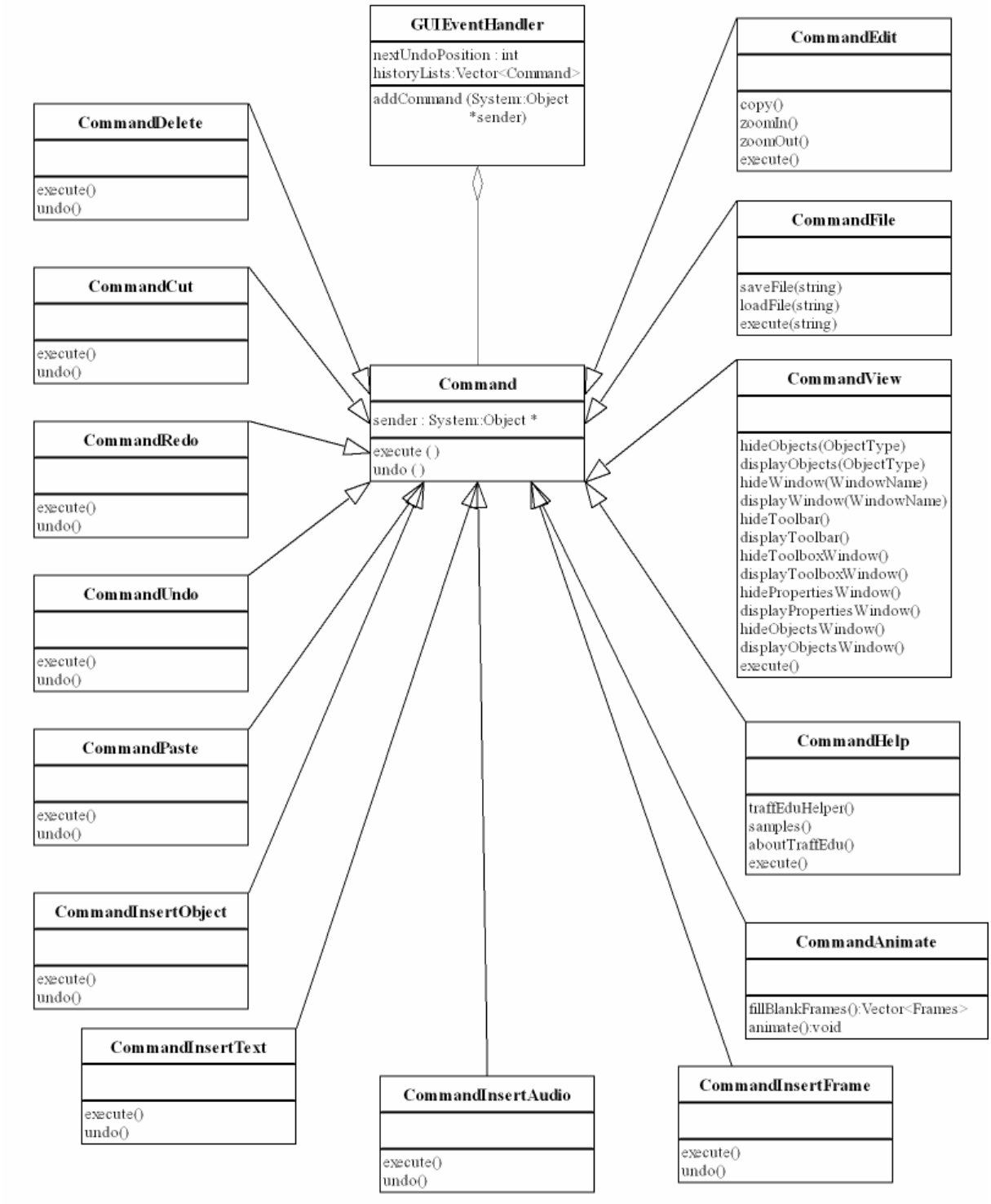


Figure 4. 2

`GUIEventHandler` class is designed by obeying rules of "command design pattern" which encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

`GUIEventHandler` class meets events coming from GUI, creates command objects, and adds them to the `historyLists` vector (`addCommand`) which contains constructed command objects. By traversing on the `historyLists` vector, undo and redo operations are performed.

Each inherited class of `Command` class, is a command, used for handling GUI callbacks and calling related classes of other modules. Descriptions of those classes are below:

CommandDelete Class:

`CommandDelete` class handles GUI delete operation either coming from the toolbar or from the menu which opens when the user clicks right button of the mouse on an environment object.

CommandCut Class:

`CommandCut` class handles GUI cut operation either coming from the toolbar or from the menu which opens when the user clicks right button of the mouse on an environment object.

CommandRedo Class:

`CommandRedo` class handles GUI redo operation coming from the toolbar.

CommandUndo Class:

`CommandUndo` class handles GUI undo operation coming from the toolbar.

CommandPaste Class:

`CommandPaste` class handles GUI paste operation either coming from the toolbar or from the menu which opens when the user clicks right button of the mouse on an environment object.

CommandInsertObject Class:

`CommandInsertObject` class handles GUI insert object operation either coming from the toolbar or from the toolbox which is used for constructing traffic case.

CommandInsertText Class:

`CommandInsertText` class handles GUI insert text operation coming from the toolbar.

CommandInsertAudio Class:

`CommandInsertAudio` class handles GUI insert audio operation coming from the toolbar.

CommandInsertFrame Class:

`CommandInsertFrame` class handles GUI insert frame operation coming from the toolbar.

CommandEdit Class:

`CommandEdit` class handles GUI edit operations coming from the toolbar, except those added to the `historyLists` vector. Since undo and redo operations can not be applied to this class, constructed `CommandEdit` object is not added to the `historyLists` vector.

CommandFile Class:

`CommandFile` class handles GUI file operations coming from the toolbar. Since undo and redo operations can not be applied to this class, constructed `CommandFile` object is not added to the `historyLists` vector. `save` and `load` methods call related classes in the `FileHandler` module.

CommandView Class:

`CommandView` class handles GUI file operations coming from the toolbar. Since undo and redo operations can not be applied to this class, constructed `CommandView` object is not added to the `historyLists` vector. Functionality of its methods is hiding or displaying objects, windows, toolbar, toolbox window, properties window or objects window.

CommandHelp Class:

`CommandHelp` class handles GUI help operations coming from the toolbar. Since undo and redo operations can not be applied to this class, constructed `CommandHelp` object is not added to the `historyLists` vector.

CommandAnimate Class:

`CommandAnimate` class handles GUI animate operations coming from the animate buton in the toolbar. This class has two functions; first one generates the frames that are not specifically designed by the user. Second function starts the simulation loop. Since undo and redo operations can not be applied to this class, constructed `CommandAnimate` object is not added to the `historyLists` vector.

4.3 EditorChecker Class

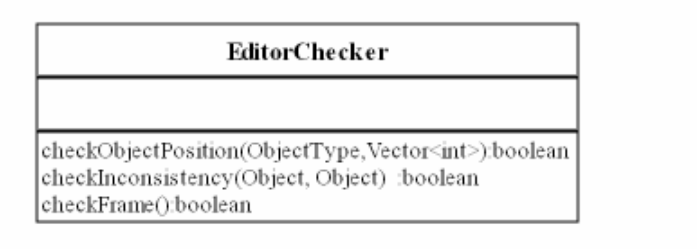


Figure 4. 3

`EditorChecker` class, as the name implies, checks the operations done in TraffEdu Editor. Whenever an event occurs on OSG Windows such as inserting a new object to the map, before taking that action, `EditorChecker` object checks the consistency of the operation.

`checkObjectPosition(ObjectType, Vector<int>)`

This function checks whether the object with object type denoted with the first argument can be inserted at the coordinates denoted with the second argument. It first selects the nearest object in that coordinates, and according to the type of that object it returns true or false meaning it is possible to insert new object on top of existing one. For example a vehicle may be inserted on road but a road can not be inserted on other road.

`checkInconsistency(Object, Object):boolean`

This function checks whether there is an inconsistency between given two objects and returns true if they are not consistent, otherwise it returns false. For example if a traffic sign is to be put on a road, this function checks whether that type of traffic sign is consistent with the road type.Traffic sign with type `ANAYOL` can not be put on a road with type `TALIYOL`. Samely, the consistency of directions of roads that are next to each other, the consistency of traffic signs on the same road and whether the vehicle is moved on its path is checked with this function.

`checkFrame():boolean`

Whenever a frame is attempted to save this function is called whether the frame is set completely, for example if there is any vehicle without a path this function will return false and a warning will be created since the user will not be able to change positions of the vehicles in the next frames.

4.4 OSGWindows Class

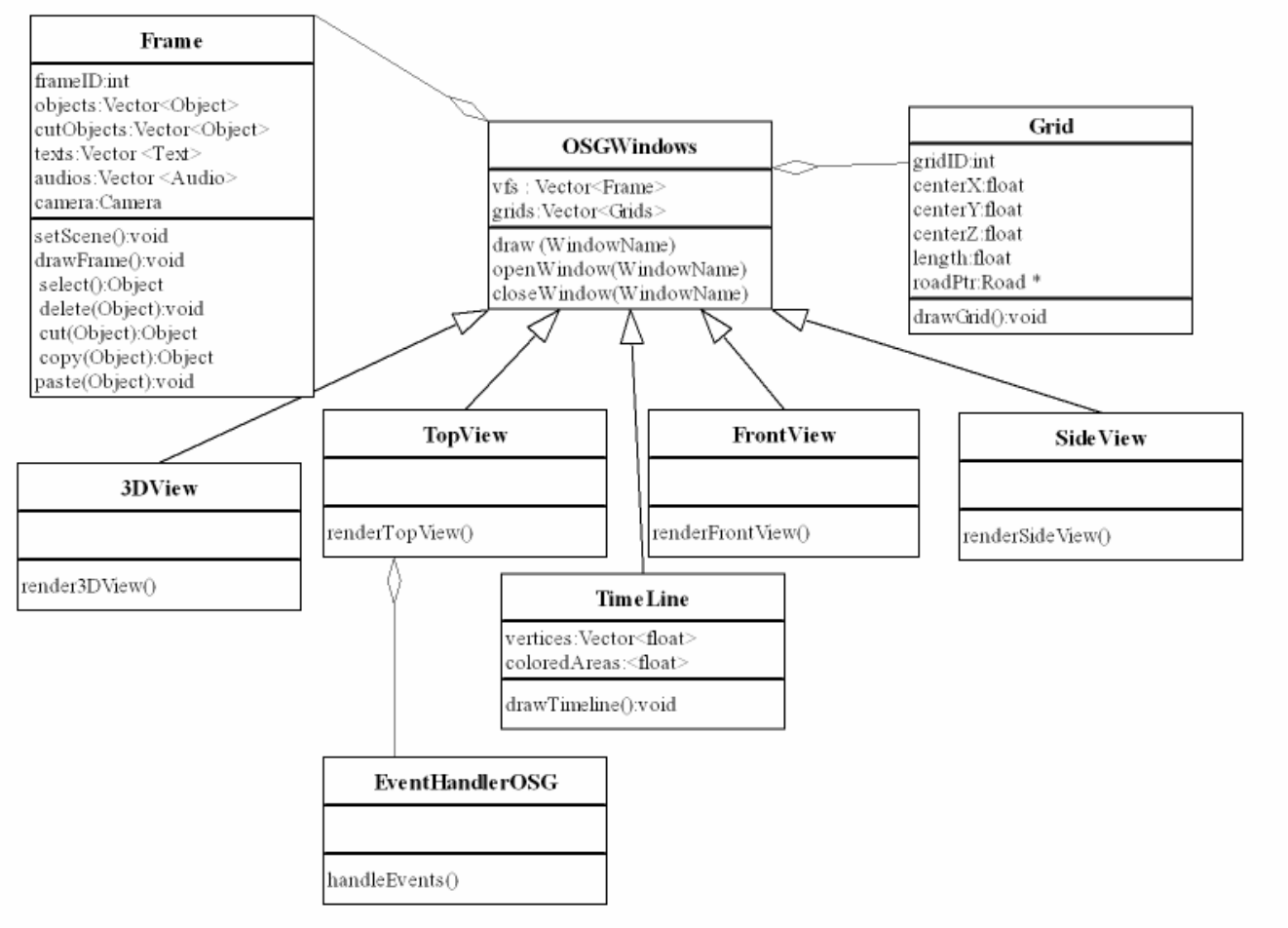


Figure 4. 4

`OSGWindows` class represents the area where the user designs his/her own environment. Windows created in OSG are reached by means of a pointer from the window created by .NET.

It contains `Frame` class vector that includes created environment objects and `Grid` class vector which represents the ground that the environment settles on. Grid class has 'hasa' relationship with `OSGWindows` class since it is drawn in every window except for `TimeLineWindow` and it does not change frame to frame.

There are four subwindows displaying the environment from different projections and a `TimeLineWindow` for inserting frames in the OSG window. Classes inherited from `OSGWindows` class represents those sub windows. Description of those classes are as follows:

3Dview Class:

`3Dview` class represents the sub window displaying perspective projection of the environment.

TopView Class:

`TopView` class represents the sub window displaying orthogonal projection of the environment from top view. Interaction with the user will be handled in this window by `EventHandleOSG` class.

FrontView Class:

`FrontView` class represents the sub window displaying orthogonal projection of the environment from front view.

SideView Class:

`SideView` class represents the sub window displaying orthogonal projection of the environment from side view.

TimeLineView Class:

`TimeLineView` class represents the timeline window displaying a timeline by using the vertices array of that class. Interaction related to choosing frame from timeline is handled by `EventHandleOSG` class.

4.5 PhysicsEngine Class

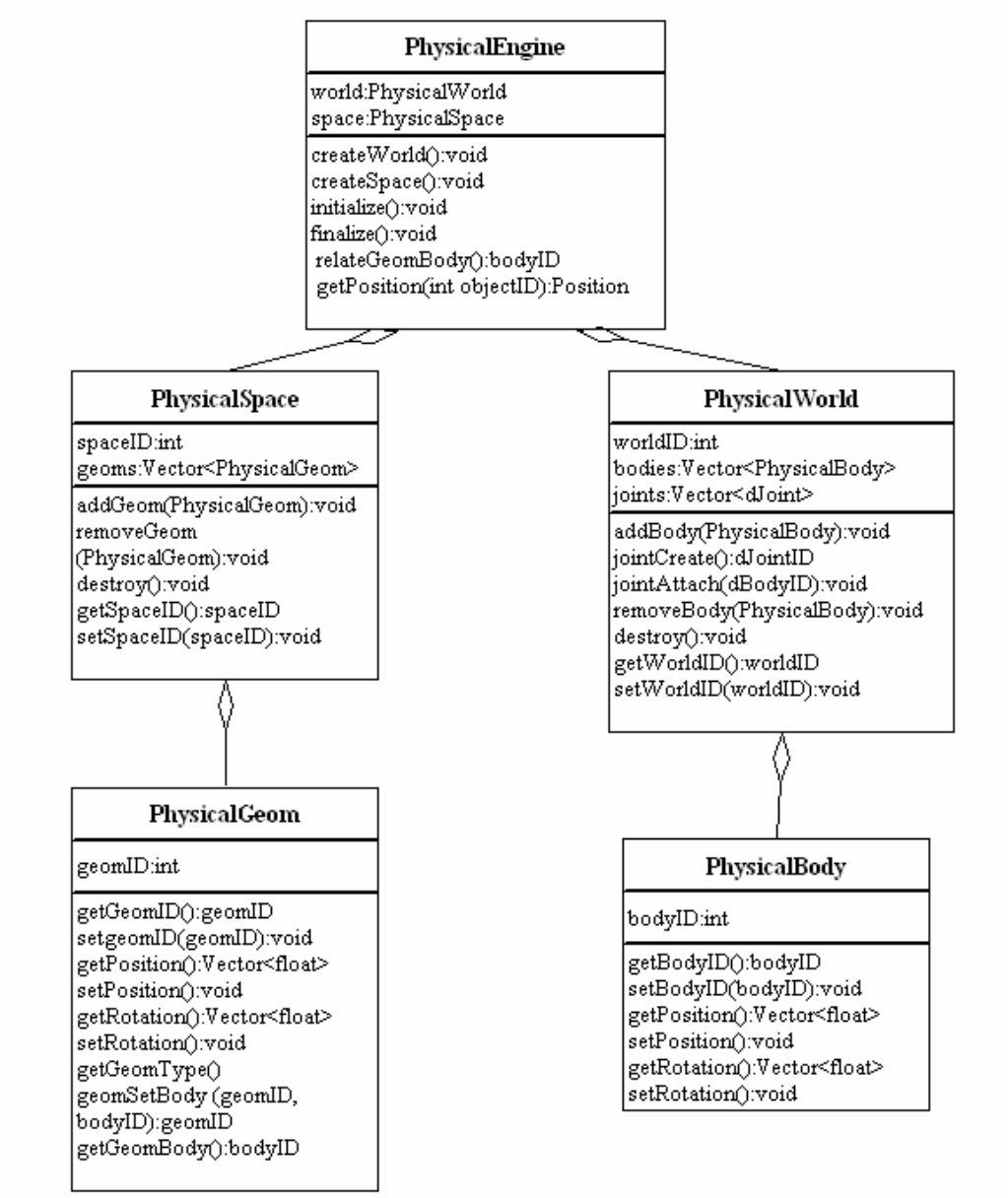


Figure 4. 5

The `Physics Module` is used for simulating vehicles in the system and apply the essential physics to the bodies in the environment so that the resulting animation becomes more realistic. This module receives the positional properties of the vehicles from `Frame` class and responds with the physics that is required for realistic animation applied to the vehicles. It is decided to use Open Dynamics Engine (ODE) library for this purpose which supplies the collision detection and velocity control of the bodies in the created world. The hierarchy of physics module is shown in Figure 4.5.

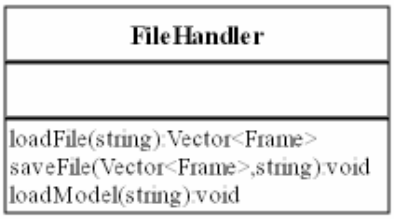
`PhysicalEngine` class makes the necessary ODE initializations to constitute the environment of a typical simulation in animation phase of TrafEdu system. Firstly, it is in involved to create a world with `createWorld` function to embed the bodies of the objects. Then, a physical space is created with `createSpace` function to handle the collision detections.

For each vehicle model a rigid body to which the force is transmitted to gain the linear velocity in the desired direction, should be created. These bodies are added to the corresponding world object in `PhysicalWorld` class. The bodies gain these forces via a geom object which is of dGeom data type in ODE. Therefore, each body should have a geom object associated with it and each geom should be created in a space. For this purpose a `PhysicalSpace` class is created in `PhysicalEngine` class to provide the collision detection environment in which the geom objects in accordance with the bodies are put. Geoms are distinct from rigid bodies in that a geom has geometrical properties (size, shape, position and orientation) but no dynamical properties (such as velocity or mass). A body and a geom together represent all the properties of the simulated object.

The `PhysicalBody` class read the corresponding position and rotation properties from the `Frame` class in which the calculated properties of models can be found. That's why the body object can follow the actual model during the animation. In `PhysicalGeom` class there are also the position and rotation functions in the same manner and the `geomSetBody` function which associates the geom to the rigid body.

In a vehicle model body objects should be created for each part of the car (such as the wheels and the body of the car). Because when the car is leaded to a certain position all the parts should also go along the same path following one another to provide the combination. This task is done by the joints (dJoint data type) in ODE which combine bodies to each other. In `PhysicalWorld` class joints are created after bodies by `JointCreate` function and attached to the corresponding bodies via `JointAttach` function and form a rigid body that acts like a car in the same world during the animation.

4.6 FileHandler Class

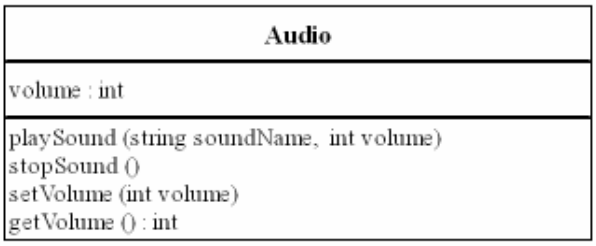


This module has three main functionalities; one of which is to read the vertices data of models to be inserted. One other functionality is to read from a XML file. The last one is to write to an XML formatted file and so involve in the reconstruction of the `Frame` object for the traffic case prepared previously. The module consists of one class, `FileHandler`, to manage these functionalities.

We will use premeditated 3ds max models in TraffEdu. To export models from 3ds max to OpenSceneGraph, we will use OSGExp which is an open source exporter, actually a plug-in to be installed on top of 3ds max. The .max models will be translated once and will be stored in the directory hierarchy of our project TraffEdu in .osg format under TraffEdu/Models directory. As an extra feature, it is still considered to give the user the opportunity of adding new models. In implementing the file functions, both the functions supported by the OpenSceneGraph in osgDB library and the input/output functions of C++ will be used according to their ease of use. The osgDB library provides support for reading and writing scene graphs, providing a plugin framework and file utility classes. The plug-in framework is centered around the osgDB::Registry, and allows plugins which provide specific file format support to be dynamically loaded on demand. osgDB provides handy functions not only for managing files but also managing directories. To give an example, the directory content or a given file's type in a given directory can easily be accessed.

The function `loadModel` will load the vertices and normal vector variables of `Object` class instances, with the vertices and normal vector data read from the appropriate model files. The function `saveFile` simply takes two arguments one of which is for the `Frames` to be saved and the other for the name of the file to be produced. `loadFile` is the opposite of `saveFile`. It takes an argument for the name of the file to be loaded and returns a vector of `Frame` class instances. The parsing of the XML file will be done via Apache Xerces C++ XML Parser. The DTD schema and a hierarchy of tags for the XML file is given in A.2.

4.7 Audio Class



TraffEdu will have some audios like speaking of the user, effects in the animation, or warning sound effects in the preparation of the traffic case. `Audio` class will be implemented to control the sounds. Functions in Direct Sound Library of DirectX will be used to implement this module. Audio will interact with physics engine while playing crash effects and with `GUIEventHandler` if user makes a wrong attempt while constructing the environment. All kinds of audio files will be saved as mp3 or wav files.

`Audio` is the class that operates in `Audio` module. Description of the methods of this class is explained below:

playSound:

`playSound` creates a new audio stream for the audio whose name is given. Volume is adjusted according to volume argument. This function returns a `streamID` used to pause or stop the sound stream.

pauseSound:

`pauseSound` pauses the audio stream according to the given `streamID`.

stopSound:

`stopSound` stops the audio stream according to the given streamed.

4.8 Class Diagrams Overview

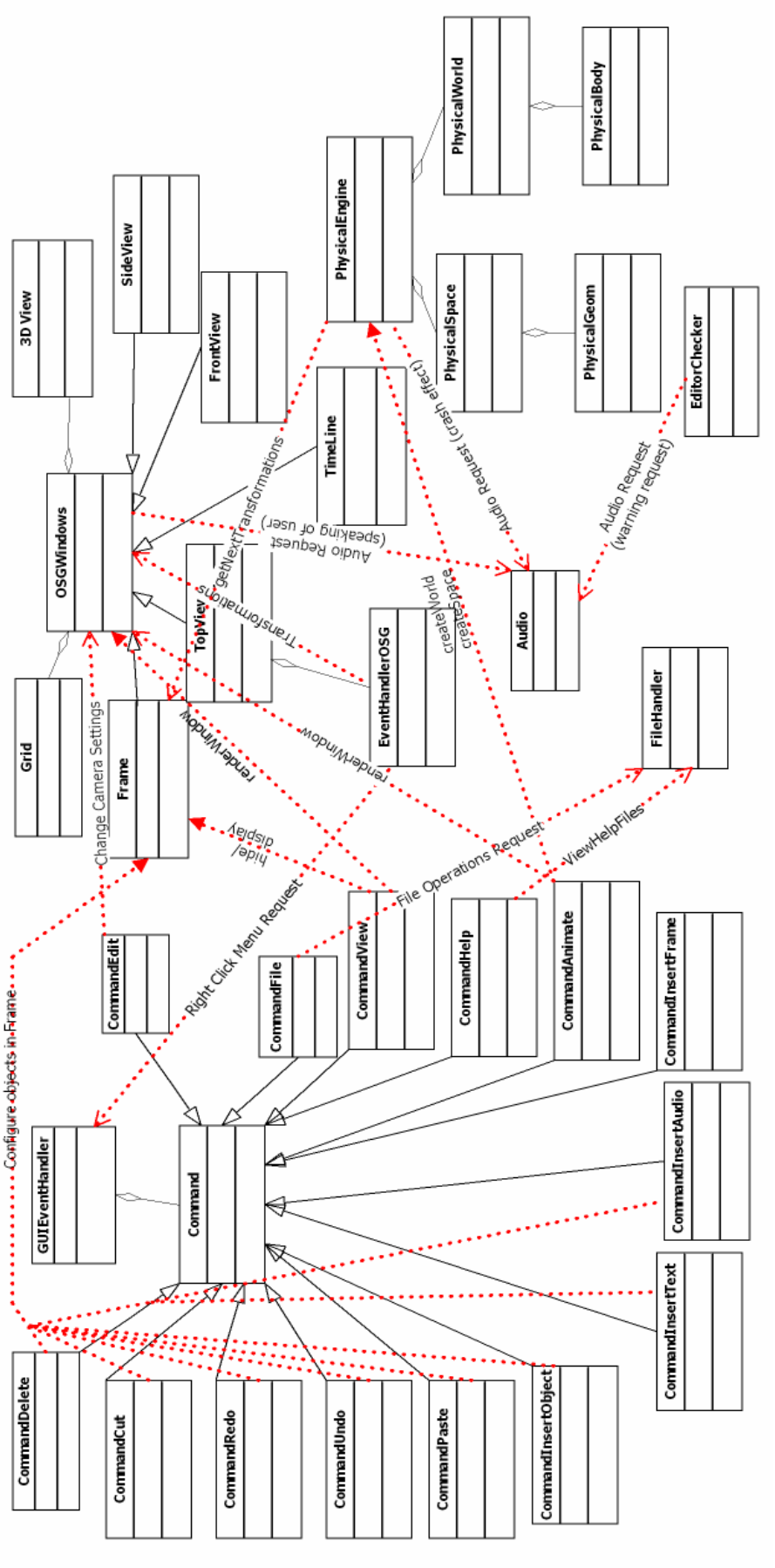

PhysicalWorld
PhysicalBody
PhysicalEngine
SideView
3D View
FrontView
PhysicalSpace
PhysicalGeom
EditorChecker
TimeLine
OSGWindows
Audio Request (Crash effect)
Audio Request (warning request)
Audio Request (Speaking of user)
getNextTransformations
TopView
Transformations
createdWorld
createdSpace
EventHandlerOSG
Audio
FileHandler
Grid
Frame
renderWindow
renderWindow
Change Camera Settings
hide/display
File Operations Request
ViewHelpFiles
Right Click Menu Request
Configure objects in Frame
CommandEdit
CommandFile
CommandView
CommandHelp
CommandAnimate
CommandInsertFrame
GUIEventHandler
Command
CommandInsertAudio
CommandInsertText
CommandDelete
CommandCut
CommandRedo
CommandUndo
CommandPaste
CommandInsertObject

Figure 4. 6

A. APPENDIX

A.1 Enumeration Types

```
Enum WindowName{
     TOP_VIEW,
     SIDE_VIEW,
     FRONT_VIEW,
     3D_VIEW,
     TIMELINE
}
Enum DrawMode{
     DRAW_VEHICLE,
     DRAW_PATH,
     DRAW_ROAD,
     DRAW_ROAD_BRANCH,
     DRAW_LANE,
     DRAW_TRAFFIC_SIGN,
     DRAW_TRAFFIC_LIGHT,
     DRAW_TREE,
     DRAW_HOUSE
}
Enum VehicleType{
     OTOMOBILE,
     AMBULANCE,
     TRACK
}
Enum ObjectType{
     VEHICLE,
     PATH,
     ROAD,
     ROAD_BRANCH,
     LANE,
     TRAFFIC_SIGN,
     TRAFFIC_LIGHT,
     HOUSE,
     TREE
}
Enum BranchType{
     ANAYOL,
     TALIYOL,
     KAVSAK
}
Enum RoadType {
     DUZ_YOL,
     TALI_ YOL,
     KAVSAKLI_YOL,
     ADA_YOL,
     BOLUNMUS_YOL,
     VIRAJLI_YOL,
     DARALAN_YOL,
     IKI_YONDEN_DARALAN_YOL
```

```
}
Enum RoadLineType {
     TEKLI_KESIKLI_CIZGI,
     CIFTLI_KESIKLI_CIZGI,
     TEKLI_DEVAMLI_CIZGI,
     CIFTLI_DEVAMLI_CIZGI
}
Enum LaneDirection {
     DOGU_BATI,
     BATI_DOGU,
     KUZEY_GUNEY,
     GUNEY_KUZEY,
     KUZEYDOGU_GUNEYBATI,
     KUZEYBATI_GUNEYDOGU,
     GUNEYDOGU_KUZEYBATI,
     GUNEYBATI_KUZEYDOGU
}
Enum BranchDirection{
     DOGU_BATI_DOGRULTUSU,
     KUZEY_GUNEY_DOGRULTUSU,
     KUZEYDOGU_GUNEYBATI_DOGRULTUSU,
     KUZEYBATI_GUNEYDOGU_DOGRULTUSU
}
Enum TrafficSignTypes{
        SAGA_TEHLIKELI_VIRAJ,
        SOLA_TEHLIKELI_VIRAJ,
        SAGA_TEHLIKELI_ DEVAMLI_VIRAJ,
        SOLA_TEHLIKELI_DEVAMLI_VIRAJ,
        IKI_TARAFTAN_DARALAN_KAPLAMA,
        SAGDAN_DARALAN_KAPLAMA,
        SOLDAN_DARALAN_KAPLAMA,
        KAYGAN_YOL,
        ISIKLI_ISARET_CIHAZI,
        IKI_YONLU_TRAFIK,
        DIKKAT,
        KONTROLSUZ_KAVSAK,
        ANAYOL_TALIYOL_KAVSAGI,
        SAGDAN_ANAYOL_TALIYOL_KAVSAGI,
        SOLDAN_ANAYOL_TALIYOL_KAVŞAGI,
        SAGDAN_ANAYOLA_GIRIS,
        SOLDAN_ANAYOLA_GIRIS,
        DONEL_KAVSAK_YAKLASIMI,
        TEHLIKELI_VIRAJ_YON_LEVHASI,
        YOL_VER,
        DUR,
        TASIT_GIREMEZ,
        TASIT_TRAFIGINE_KAPALI_YOL,
        MOTOSIKLET_HARIC_MOTORLU_TASIT_TRAFIGINE_KAPALI_YOL,
        MOTORLU_TASIT_GIREMEZ,
        TASIT_GIREMEZ,
        SAGA_DONULMEZ,
```

```
        SOLA_DONULMEZ,
        U_DONUSU_YAPILMAZ,
        ONDEKI_TASITI_GECMEK_YASAKTIR,
        AZAMI_HIZ_SINIRLAMASI,
        BUTUN_KISITLAMALARIN_SONU,
        HIZ_KISITLAMASI_SONU,
        GECME_YASAGI_SONU,
        SAGA_MECBURI_YON,
        SOLA_MECBURI_YON,
        ILERI_MECBURI_YON,
        ILERI_SAGA_MECBURI_YON,
        ILERI_SOLA_MECBURI_YON,
        SAGA_SOLA_MECBURI_YON,
        ILERIDE_SAGA_MECBURI_YON,
        ILEIDE_SOLA_MECBURI_YON,
        SAGDAN_GIDINIZ,
        SOLDAN_GIDINIZ,
        HER_IKI_YANDAN_GIDINIZ,
        ADA_ETRAFINDA_DONUNUZ,
        MECBURI_ASGARI_HIZ,
        MECBURI _ASGARI_HIZ_SONU,
        GIRISI_OLMAYAN_YOL_KAVSAGI,
        ILERI_CIKMAZ_YOL,
        ANAYOL,
        ANAYOL_BITIMI,
        BOLUNMUS_YOL_ONCESI_YON_LEVHASI
}

Enum TrafficLightTypes{
          NORMAL_ISIK,
          SUREKLI_YANIP_SONEN_ISIK,
          SAGA_OKLU_ISIK,
          SOLA_OKLU_ISIK
}
Enum TrafficLightColor{
     KIRMIZI,
     SARI,
     YESIL
}
Enum ObjectViewMode{
     DISPLAY,
     HIDE
}
```

A.2 DTD Schema

```
<?xml encoding="ISO-8859-1"?
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE TrafficCase [
<!ELEMENT TrafficCase (Frames*, Background)>
<!ELEMENT Frames (Frame+)>
<!ELEMENT Frame (Object+, Audio, Text)>
<!ATTLIST Frame id CDATA #REQUIRED>
<!ELEMENT Object (Vehicle+))>
<!ELEMENT Text (#PCDATA)>
<!ELEMENT Audio (#PCDATA)>
<!ELEMENT Background (Road+,Light*,Sign*)>
<!ELEMENT Road (Position, Branch+)>
<!ATTLIST Road type CDATA #REQUIRED>
<!ATTLIST Road id CDATA #REQUIRED>
<!ELEMENT Position>
<!ATTLIST Position centerX CDATA #REQUIRED>
<!ATTLIST Position centerY CDATA #REQUIRED>
<!ATTLIST Position centerZ CDATA #REQUIRED>
<!ATTLIST Position scaleFactor CDATA #REQUIRED>
<!ATTLIST Position rotationVector CDATA #REQUIRED>
<!ATTLIST Position translationVector CDATA #REQUIRED>
<!ELEMENT Branch (Lane)+>
<!ATTLIST Branch direction CDATA #REQUIRED >
<!ATTLIST Branch type CDATA #REQUIRED>
<!ELEMENT Lane>
<!ATTLIST Lane leftLine>
<!ATTLIST Lane rightLine>
<!ATTLIST Lane direction>
<!ELEMENT Vehicle (Position)>
<!ATTLIST Vehicle type CDATA #REQUIRED>
<!ATTLIST Vehicle id CDATA #REQUIRED>
<!ATTLIST Vehicle path CDATA #REQUIRED>
<!ATTLIST Vehicle speed CDATA #REQUIRED>
<!ATTLIST Vehicle acceleration CDATA #REQUIRED>
<!ELEMENT Light (Position)>
<!ATTLIST Light type CDATA #REQUIRED>
<!ATTLIST Light color CDATA #REQUIRED>
<!ATTLIST Light id CDATA #REQUIRED>
<!ATTLIST Light arrowcolor CDATA #IMPLIED >
<!ELEMENT Sign (Position)>
<!ATTLIST Sign type CDATA #REQUIRED>
<!ATTLIST Sign id CDATA #REQUIRED>
]>


The XML structure can be seen in a more hierarchical way below:

<TrafficCase>
        <Background>
                <Road type id>
                        <Position centerX centerY centerZ scaleFactor
                                rotationVector translationVector>
                        <Branch direction type>
                                <Lane leftLine rightLine direction>
                </Road>
                <Road>
                        ...
                </Road>
                <Light type id color arrowcolor>
```

```
                        <Position centerX centerY centerZ scaleFactor
                                rotationVector translationVector>
                </Light>
                <Light>
                        ...
                </Light>
                <Sign type id>
                        <Position centerX centerY centerZ scaleFactor
                                rotationVector translationVector>
                </Sign>
                <Sign>
                        ...
                </Sign>
        </Background>
        <Frames>
                <Frame id>
                        <Object>
                                <Vehicle id type path speed acceleration>
                                <Position centerX centerY centerZ scaleFactor
                                        rotationVector translationVector>
                                </Vehicle>
                                <Vehicle>
                                        ...
                                </Vehicle>
                        </Object>
                        <Object>
                                ...
                        </Object>
                </Frame>
                <Frame>
                        ...
                </Frame>
        </Frames>
</TrafficCase>
```

A.3 Coding Standards

We decided on some coding standards to make our code more readable. Below are the standards for variables, function names, type names and enumeration types.

Variable names begin with lower case characters and if it is a composite word each new word begins with a capital letter. (e.g. `word1Word2Word3`)

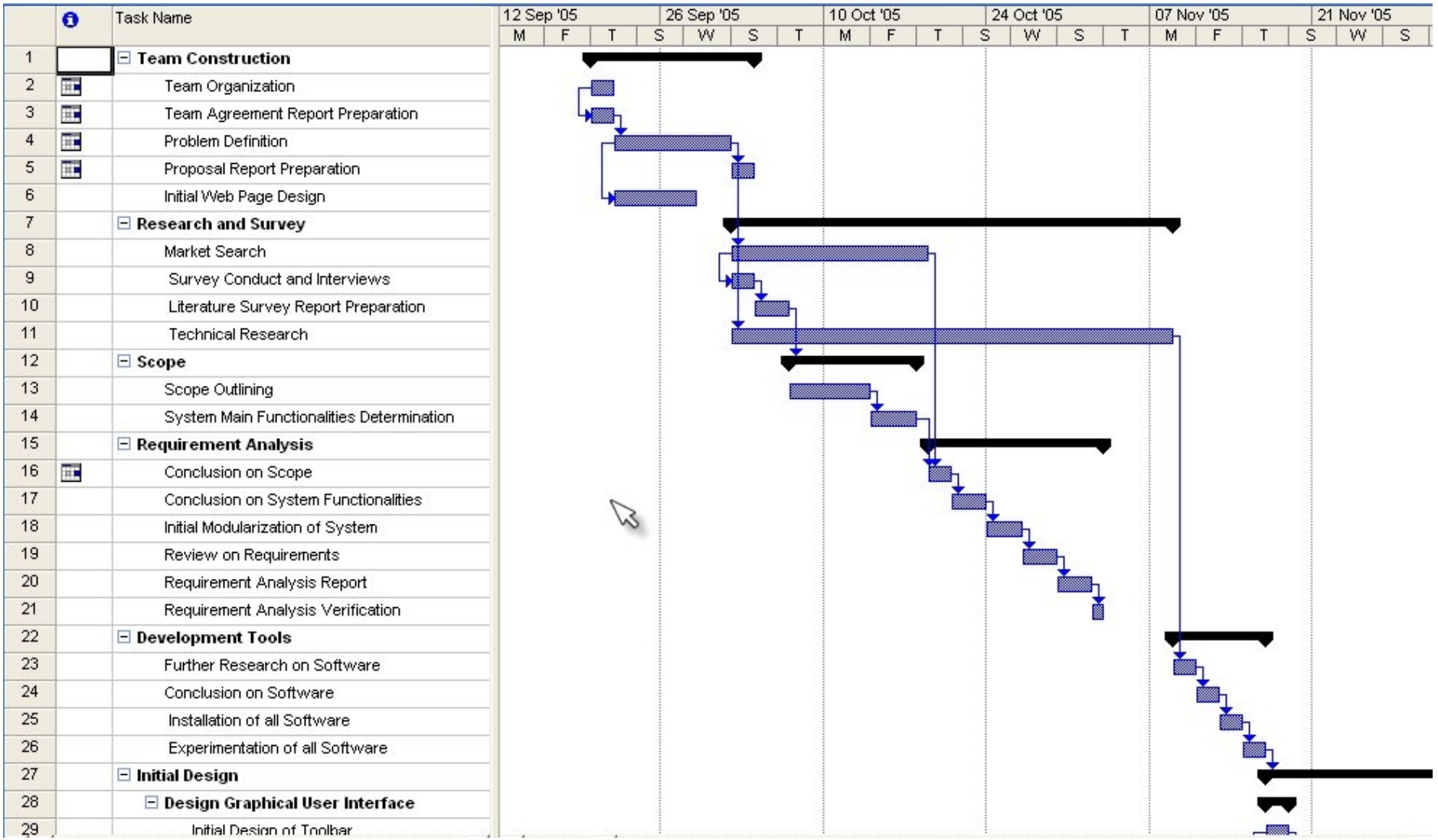
Function names begin with lower case characters and if it is a composite word each new word begins with a capital letter. (e.g. `word1Word2Word3`)

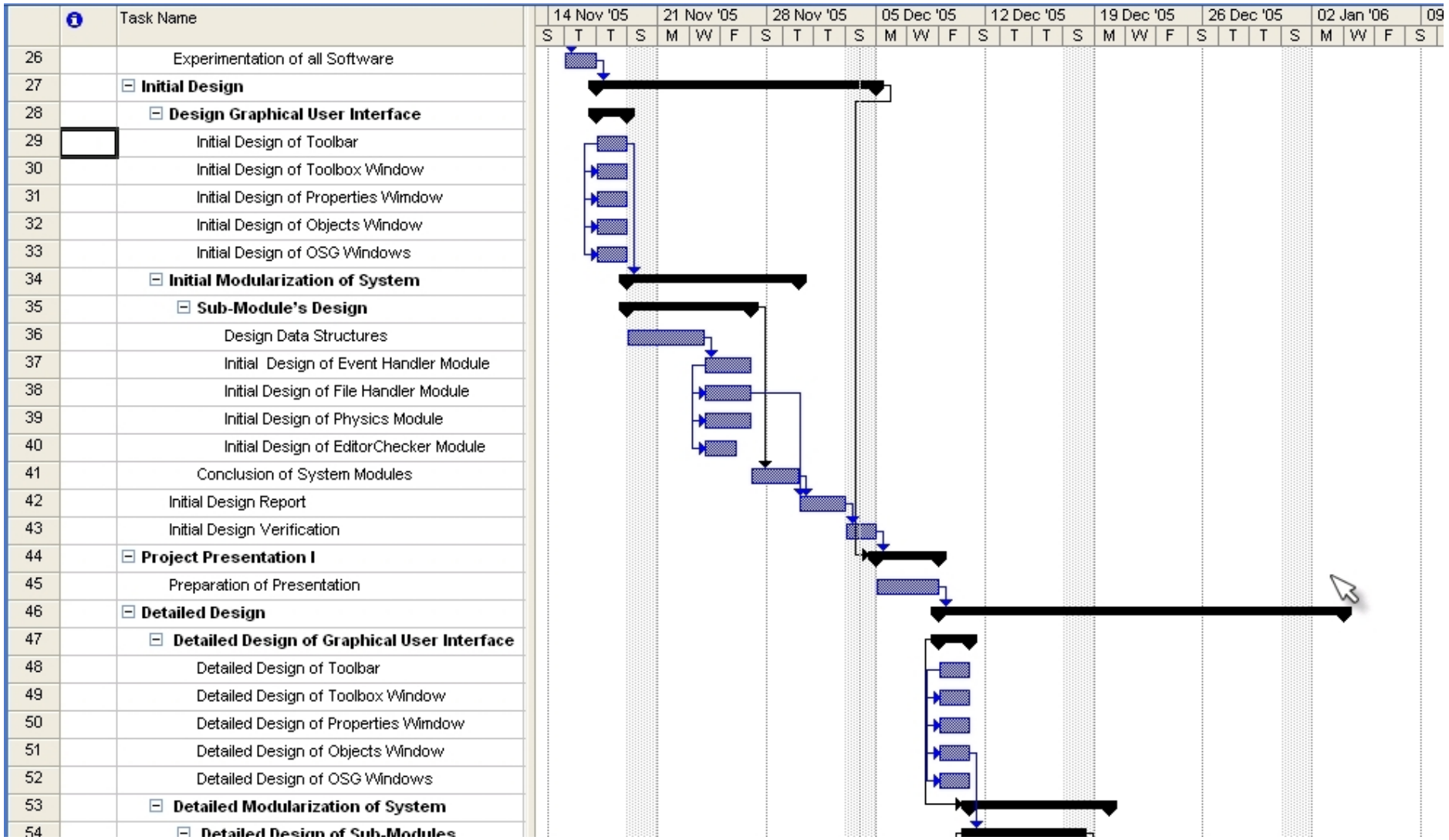
Type names such as class, enumeration and user defined types begin with upper case characters and if it is a composite word each new word begins with a capital letter. (e.g. `Word1Word2Word3`)

Enumerated types are all written in capital letters.

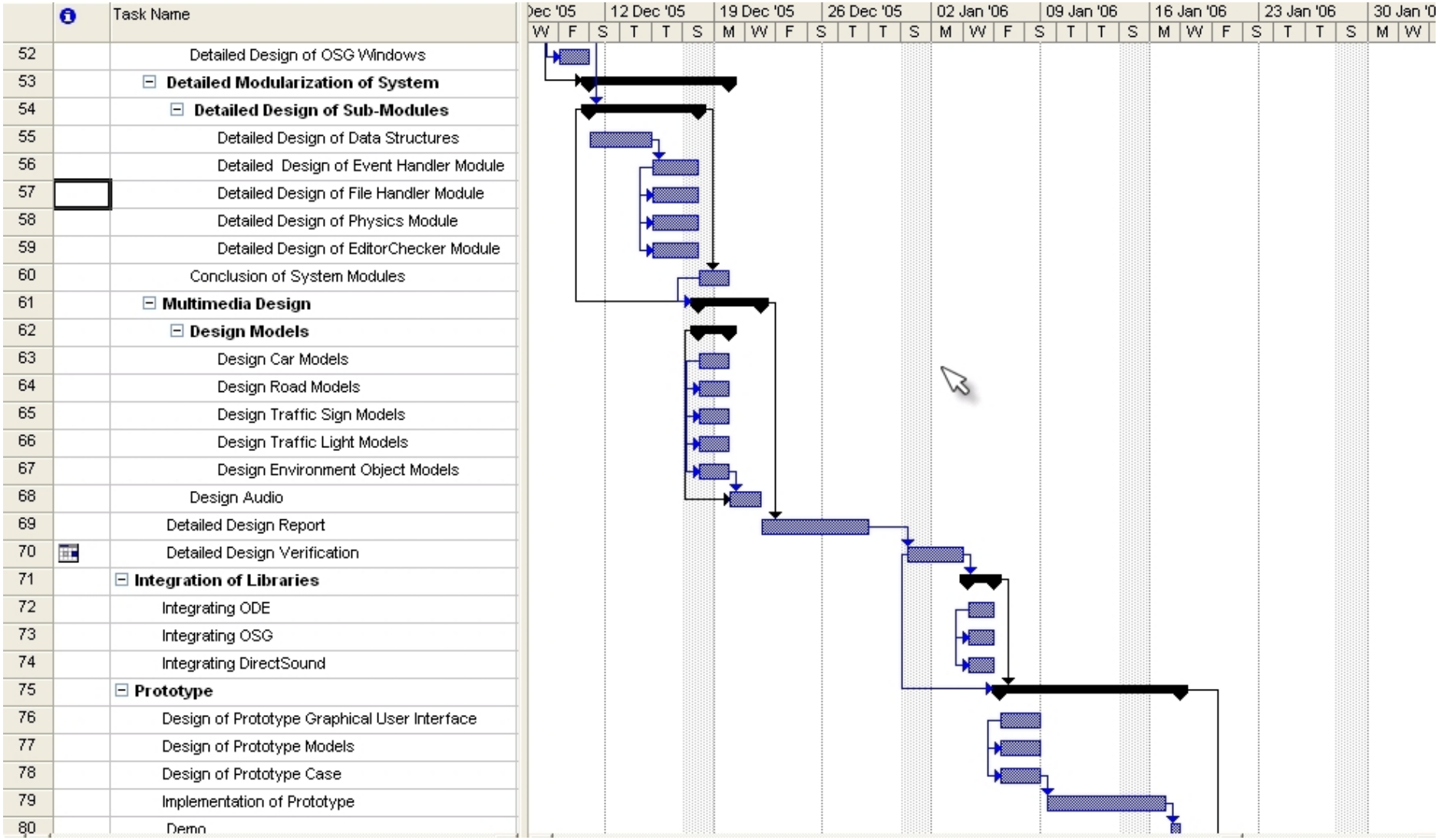
Vector typed variables' are written in plural form.

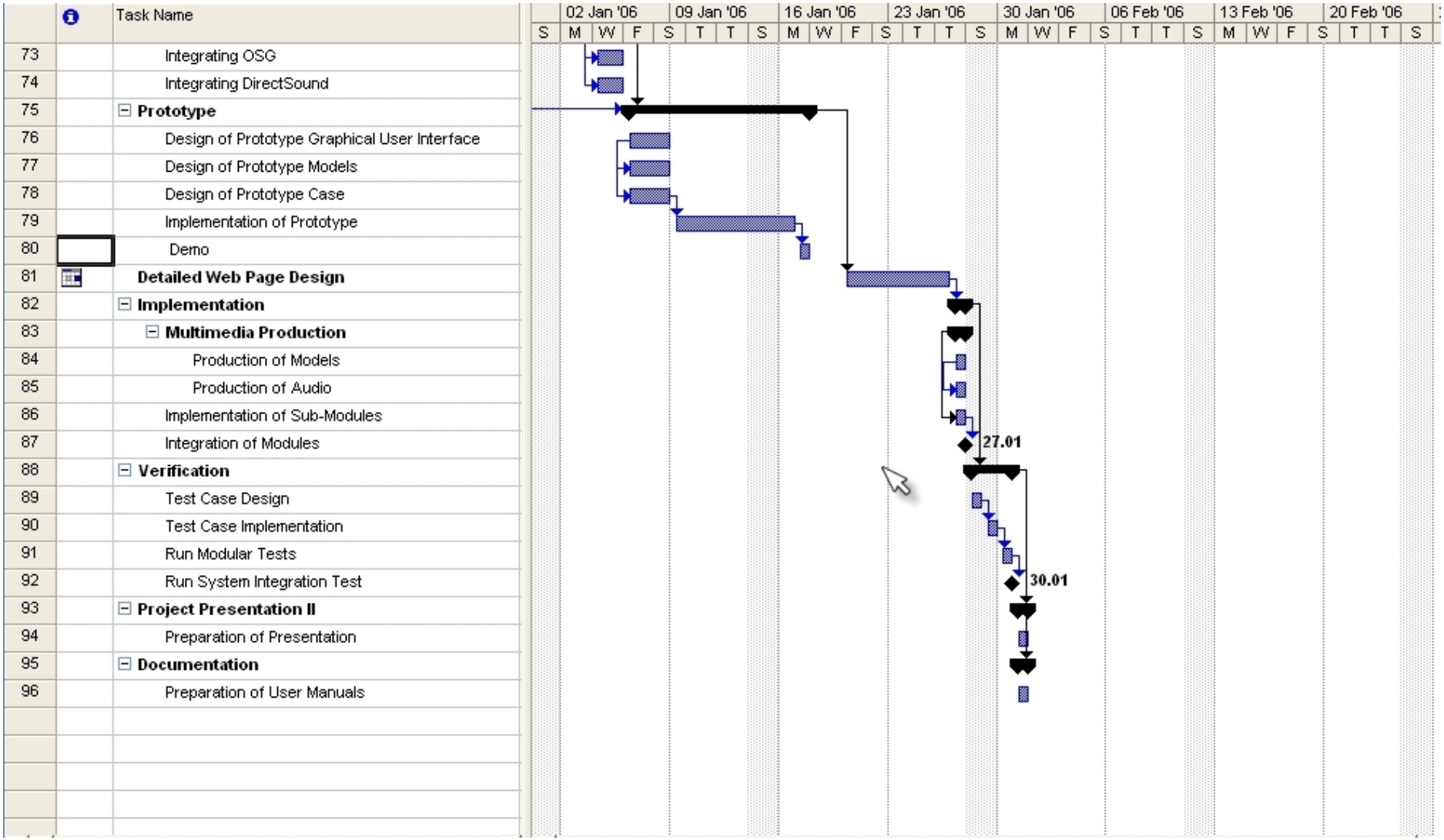
A.4 Gantt Chart


Task Name
12 Sep '05
26 Sep '05
10 Oct '05
24 Oct '05
07 Nov '05
21 Nov '05
M F T S W S T M F T S W S T M F T S W S
1 Team Construction
2 Team Organization
3 Team Agreement Report Preparation
4 Problem Definition
5 Proposal Report Preparation
6 Initial Web Page Design
7 Research and Survey
8 Market Search
9 Survey Conduct and Interviews
10 Literature Survey Report Preparation
11 Technical Research
12 Scope
13 Scope Outlining
14 System Main Functionalities Determination
15 Requirement Analysis
16 Conclusion on Scope
17 Conclusion on System Functionalities
18 Initial Modularization of System
19 Review on Requirements
20 Requirement Analysis Report
21 Requirement Analysis Verification
22 Development Tools
23 Further Research on Software
24 Conclusion on Software
25 Installation of all Software
26 Experimentation of all Software
27 Initial Design
28 Design Graphical User Interface
29 Initial Design of Toolbar



	Task Name
26	Experimentation of all Software
27	**Initial Design**
28	**Design Graphical User Interface**
29	Initial Design of Toolbar
30	Initial Design of Toolbox Window
31	Initial Design of Properties Wimdow
32	Initial Design of Objects Window
33	Initial Design of OSG Windows
34	**Initial Modularization of System**
35	**Sub-Module's Design**
36	Design Data Structures
37	Initial Design of Event Handler Module
38	Initial Design of File Handler Module
39	Initial Design of Physics Module
40	Initial Design of EditorChecker Module
41	Conclusion of System Modules
42	Initial Design Report
43	Initial Design Verification
44	**Project Presentation I**
45	Preparation of Presentation
46	**Detailed Design**
47	**Detailed Design of Graphical User Interface**
48	Detailed Design of Toolbar
49	Detailed Design of Toolbox Window
50	Detailed Design of Properties Wimdow
51	Detailed Design of Objects Window
52	Detailed Design of OSG Windows
53	**Detailed Modularization of System**
54	**Detailed Design of Sub-Modules**


Task Name
12 Dec '05
19 Dec '05
26 Dec '05
02 Jan '06
09 Jan '06
16 Jan '06
23 Jan '06
30 Jan '0
52 Detailed Design of OSG Windows
53 Detailed Modularization of System
54 Detailed Design of Sub-Modules
55 Detailed Design of Data Structures
56 Detailed Design of Event Handler Module
57 Detailed Design of File Handler Module
58 Detailed Design of Physics Module
59 Detailed Design of EditorChecker Module
60 Conclusion of System Modules
61 Multimedia Design
62 Design Models
63 Design Car Models
64 Design Road Models
65 Design Traffic Sign Models
66 Design Traffic Light Models
67 Design Environment Object Models
68 Design Audio
69 Detailed Design Report
70 Detailed Design Verification
71 Integration of Libraries
72 Integrating ODE
73 Integrating OSG
74 Integrating DirectSound
75 Prototype
76 Design of Prototype Graphical User Interface
77 Design of Prototype Models
78 Design of Prototype Case
79 Implementation of Prototype
80 Demo


Task Name
02 Jan '06
09 Jan '06
16 Jan '06
23 Jan '06
30 Jan '06
06 Feb '06
13 Feb '06
20 Feb '06
S M W F S T T S M W F S T T S M W F S T T S M W F S T T S M W F S T T S
73 Integrating OSG
74 Integrating DirectSound
75 Prototype
76 Design of Prototype Graphical User Interface
77 Design of Prototype Models
78 Design of Prototype Case
79 Implementation of Prototype
80 Demo
81 Detailed Web Page Design
82 Implementation
83 Multimedia Production
84 Production of Models
85 Production of Audio
86 Implementation of Sub-Modules
87 Integration of Modules
88 Verification
89 Test Case Design
90 Test Case Implementation
91 Run Modular Tests
92 Run System Integration Test
93 Project Presentation II
94 Preparation of Presentation
95 Documentation
96 Preparation of User Manuals
27.01
30.01