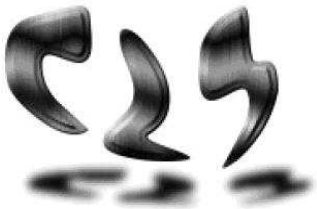
Middle East Technical University

Department of Computer Engineering

CENG 491
Computer Engineering Design I

Detailed 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 prototype implementation. The document is an improved version of the previous Initial Design Report in which design specifications were defined.

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 of use case and activity diagrams. System modules along with their classes are described in detail giving the relationships between them. Additionally, file hierarchy and a dynamic Gantt chart is attached.

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

Lane: A portion of a street or highway, usually indicated by pavement markings, that is intended for one line of vehicles.

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 an 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 TraffEdu 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 behavior 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.

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
- osgAL 0.4
- 3D Studio Max
- Adobe Photoshop

3. INTERFACE DESIGN

3.1 Sample Graphical User Interfaces

The TraffEdu Editor's window is mainly divided into four areas which is toolbar (or menu bar) at the top, Toolbox Window at the left, main window at the middle and Objects and Properties Windows on the right of the editor window. By default an orthogonal view of the 3D environment is provided for the user 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. Creating a traffic case, starts with placing appropriate road types onto the grid platform. This is provided in the Road Map section of Toolbox Window. The user may click on a type of road and then click on where he/she wants to place the road on the grid.

For placement of Vehicle objects, user can choose the type from Vehicles section of toolbox and then clicks on the place. Here there will be types of vehicles that the user can select for using in the traffic case being designed.

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.

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

The current objects in the environment can be seen from the Objects Window on the right panel of the window. Clicking on an object listed in the Objects Window shows the current selected object's properties in Properties Window below.

User will draw path to vehicle objects by clicking the path icon in the toolbar. After user is done with the current frame, he/she will click the check inconsistency button provided in the toolbar to see if the designed environment has any erroneous placement of objects.

Animation button in the toolbar is clicked after all the frames are inserted, to see the 3D animation of the traffic case designed. The animation is displayed in another window.

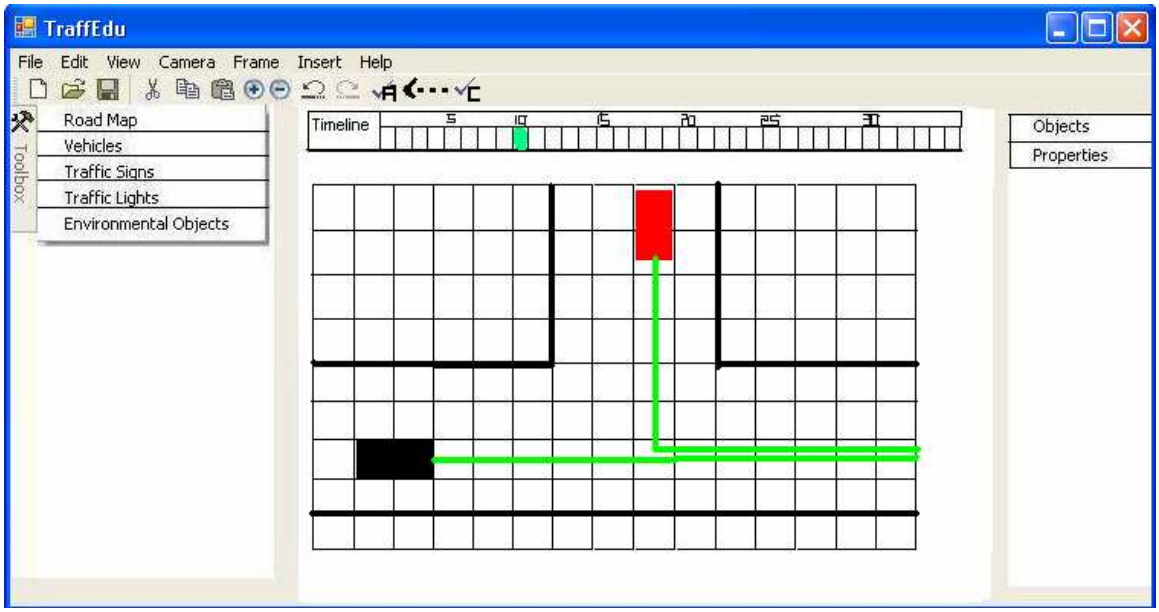


Figure 3.1

The File menu presents actions related with the file operations which are opening a new file, opening an existing file, closing the current file, saving the file with the same name or with a different name, displaying currently opened file names and finally exiting the program.

Camera menu adjusts he camera according to the needs of the user. Yaw means changing the look at point of the camera in left and right directions. Pitch means changing the look at point of the camera in up and down directions. Zoom In/Zoom out property of camera means zooming in/out to the current position of the camera reference point through the camera view direction. When an object is selected zooming action is done by

centering to the object that means changing camera reference point as the center of the selected object. The keyboard shortcuts will be displayed in the menu near each item.

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

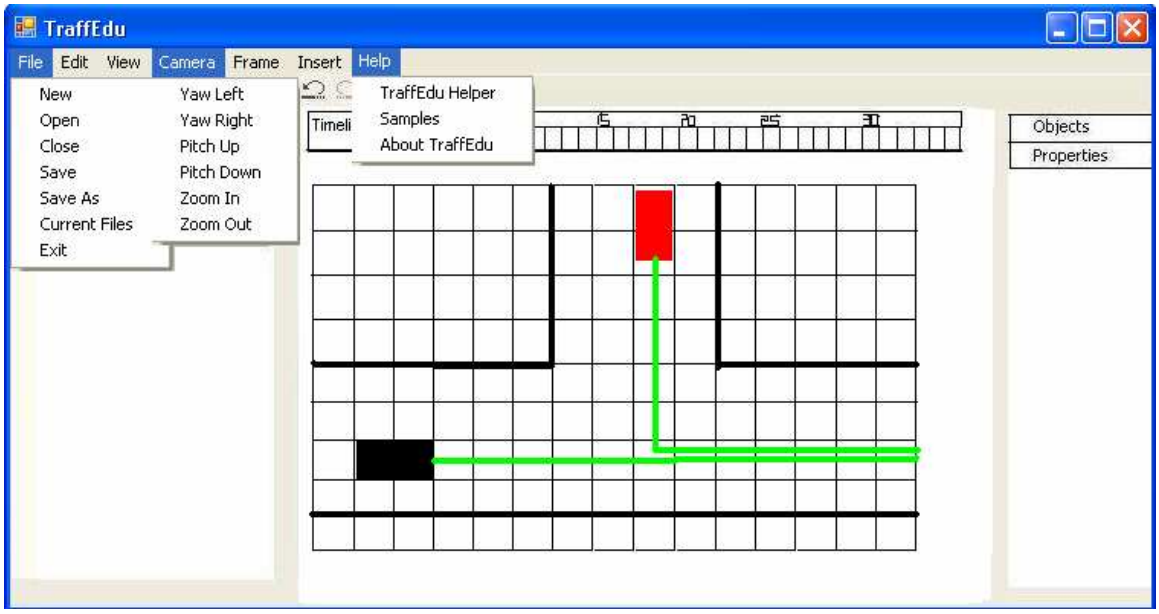


Figure 3.2

Under edit menu the following actions may be performed. Delete, cut, copy actions will be active when an object is selected.

Under frame menu, user will be able to save the currently formed environment as a frame by clicking Insert Frame or he/she will be able to delete the current frame by clicking Delete Frame.

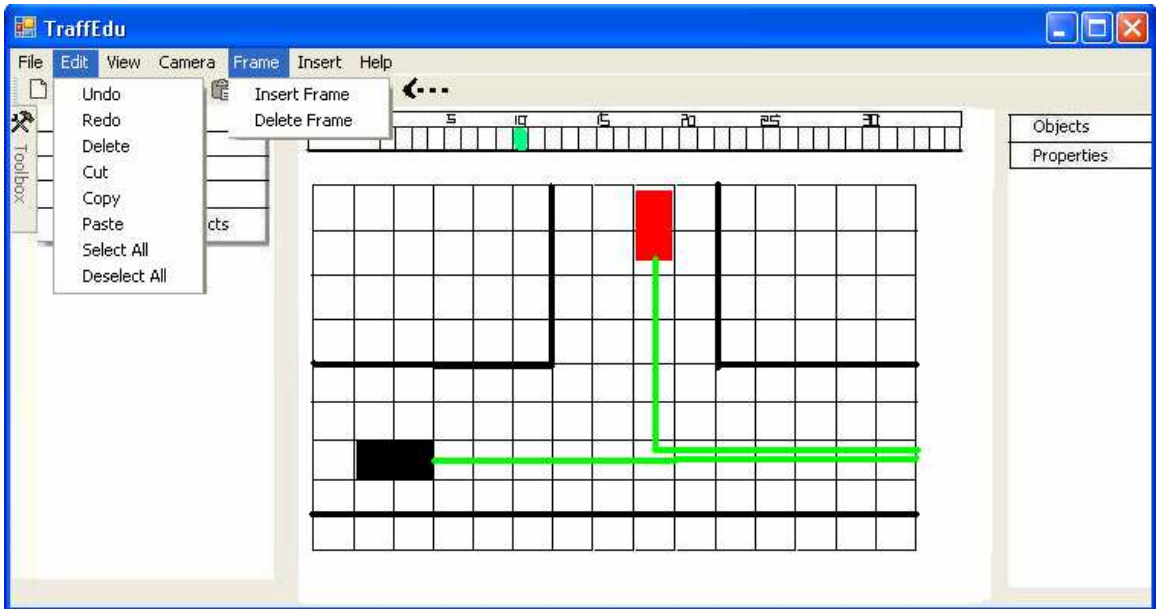


Figure 3.3

View menu gives user the chance of viewing/hiding objects or panels on the environment and the Editor window. Under Hide/Display item user will have the choice of viewing or not viewing Road Map, Vehicles, Traffic Signs, Traffic Lights, and Environmental Objects.

Under Windows 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. User will also choose which panels of the Editor (Toolbox, Objects and/or Properties) to be shown.

User will be able to insert objects, audio and text to the current frame within Insert Menu. When Text is clicked, a pop up window will appear allowing the user to insert any text. In the same way the user will be able to insert any audio file into the frame by clicking Audio item and selecting an audio file of her/his choice.

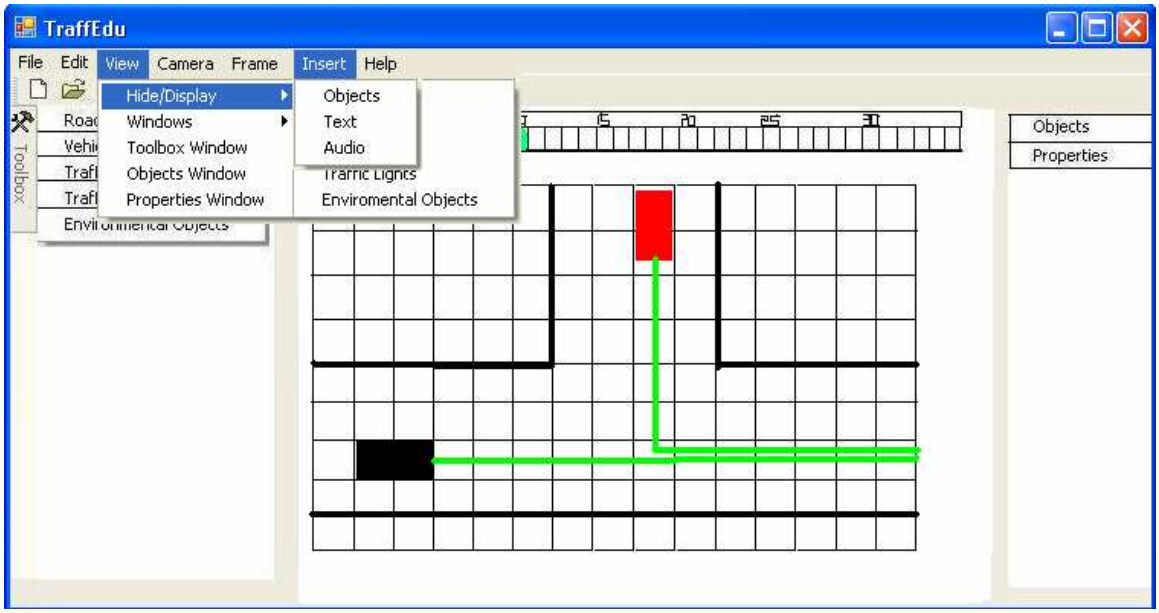


Figure 3.4

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 3.5.

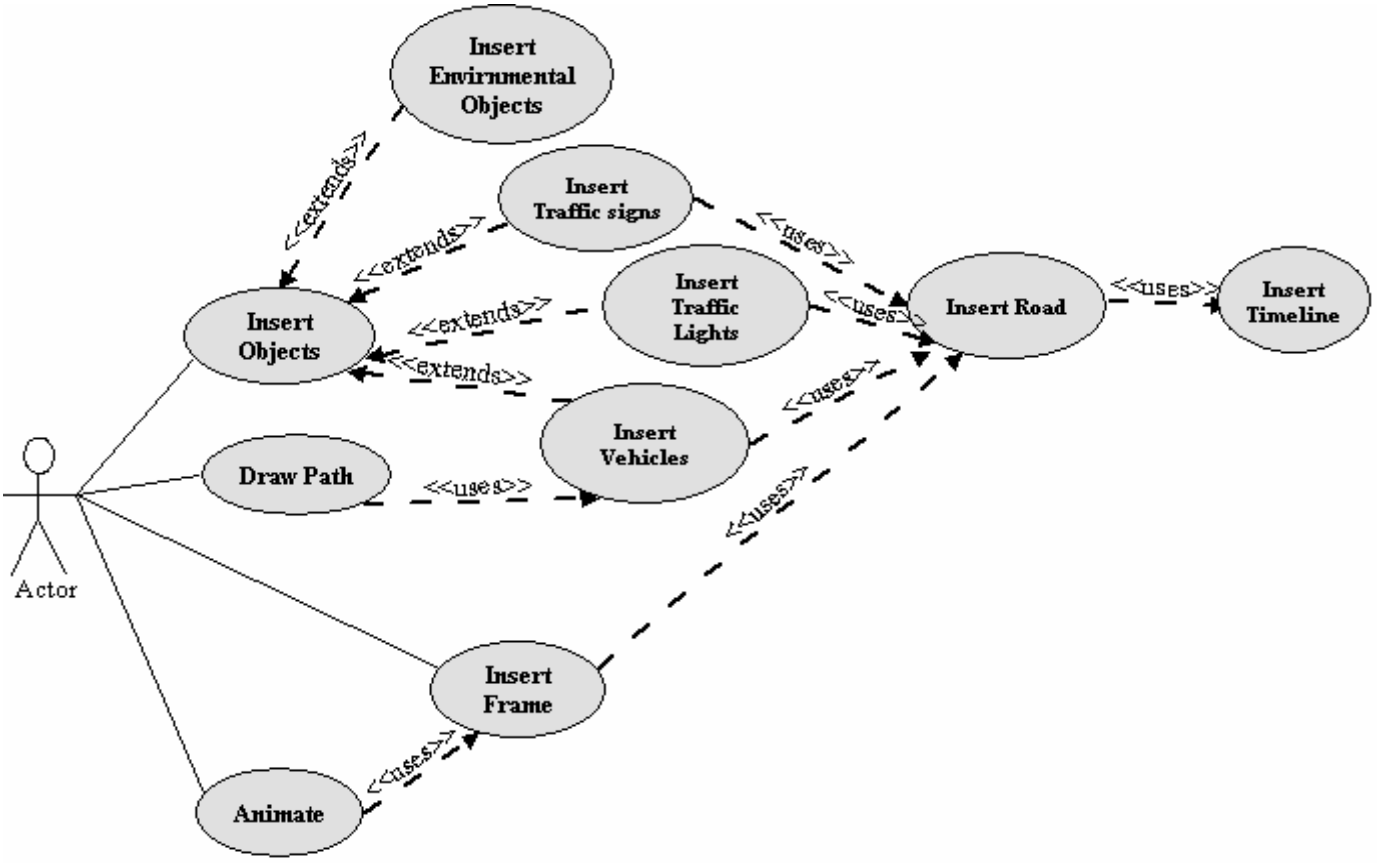


Figure 3.5

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.

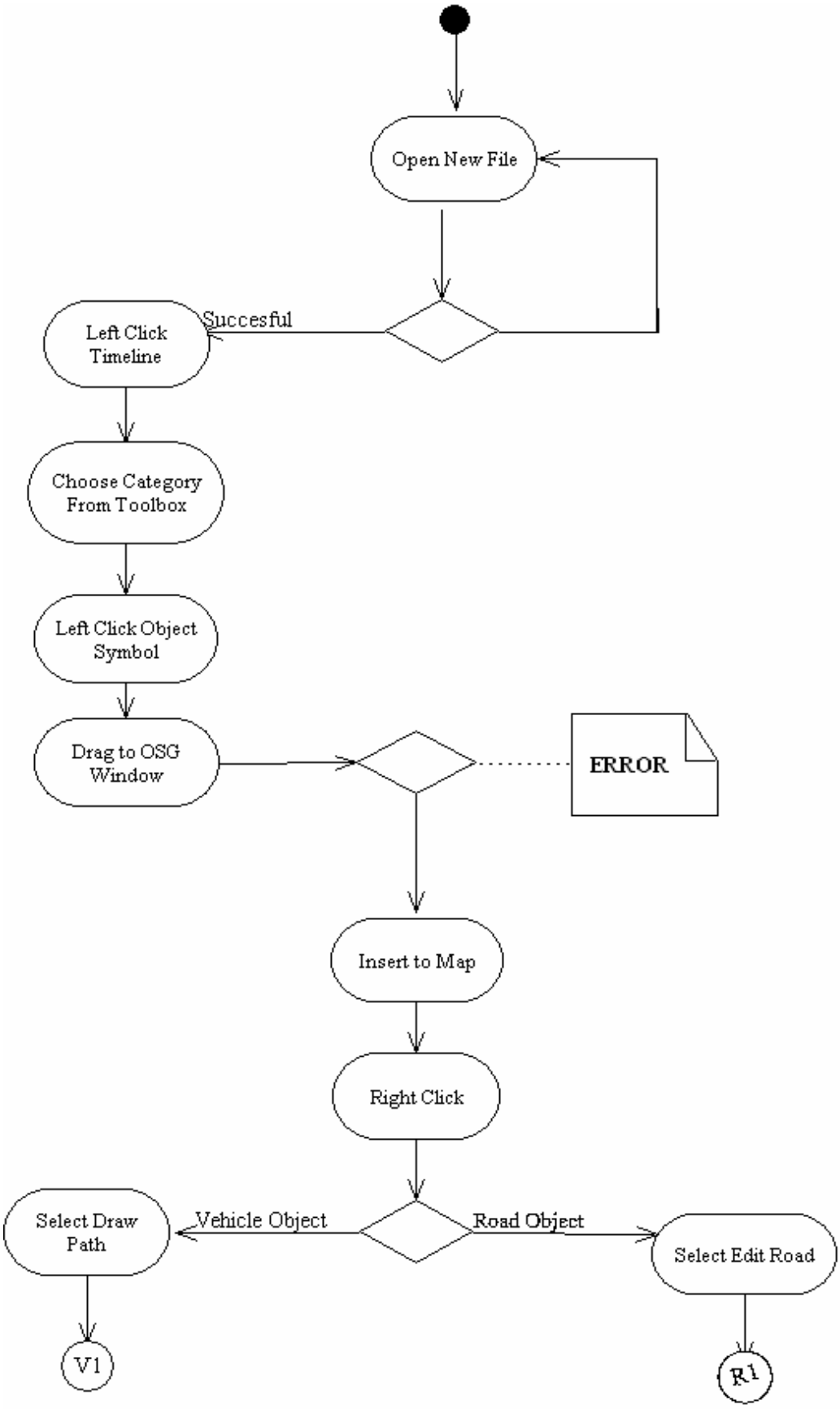

Open New File
Succesful
Left Click Timeline
Choose Category From Toolbox
Left Click Object Symbol
Drag to OSG Window
ERROR
Insert to Map
Right Click
Select Draw Path
Vehicle Object
Road Object
Select Edit Road
V1
R1

Figure 3.6

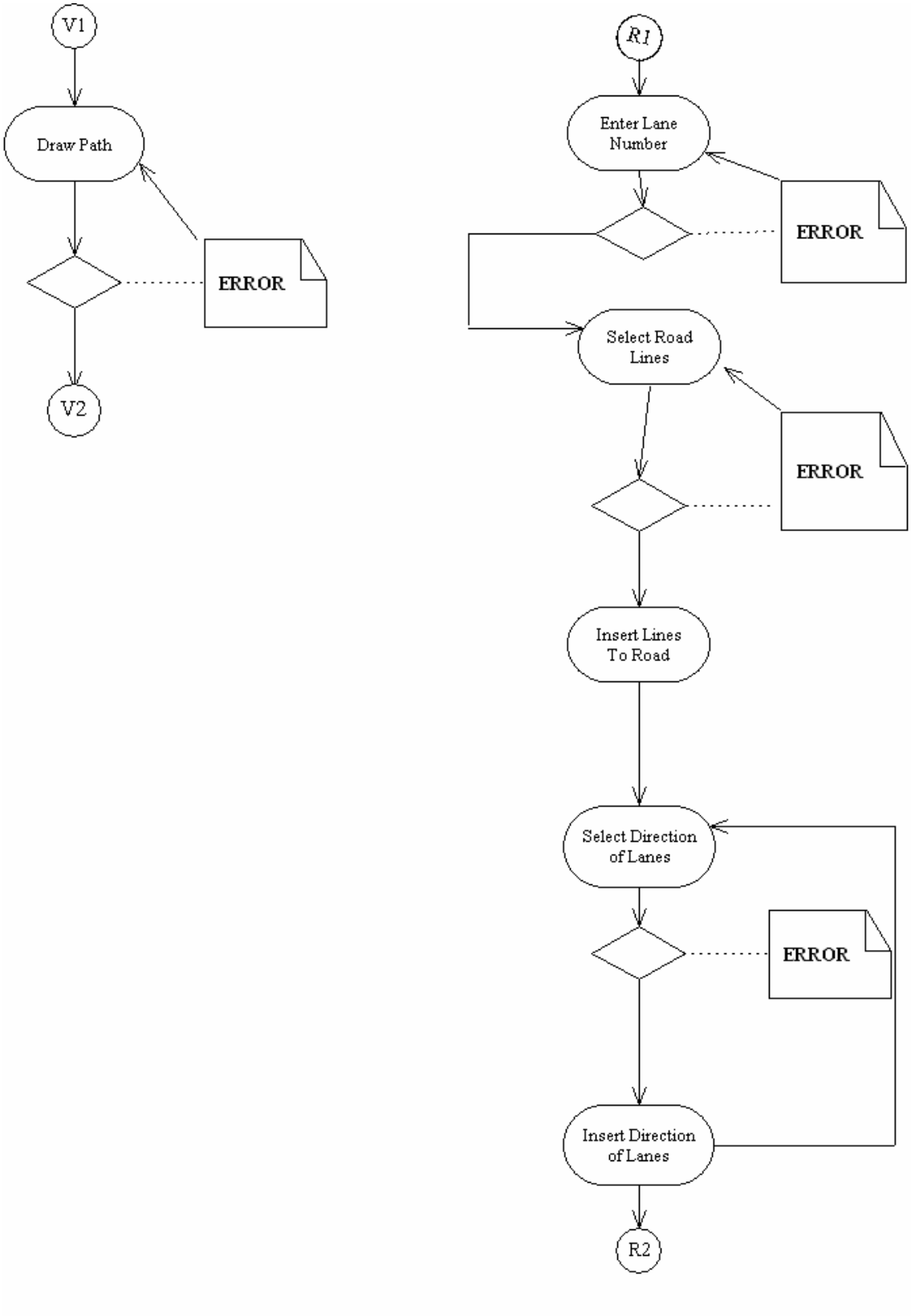

V1
Draw Path
ERROR
V2
R1
Enter Lane Number
ERROR
Select Road Lines
ERROR
Insert Lines To Road
Select Direction of Lanes
ERROR
Insert Direction of Lanes
R2

Figure 3.7

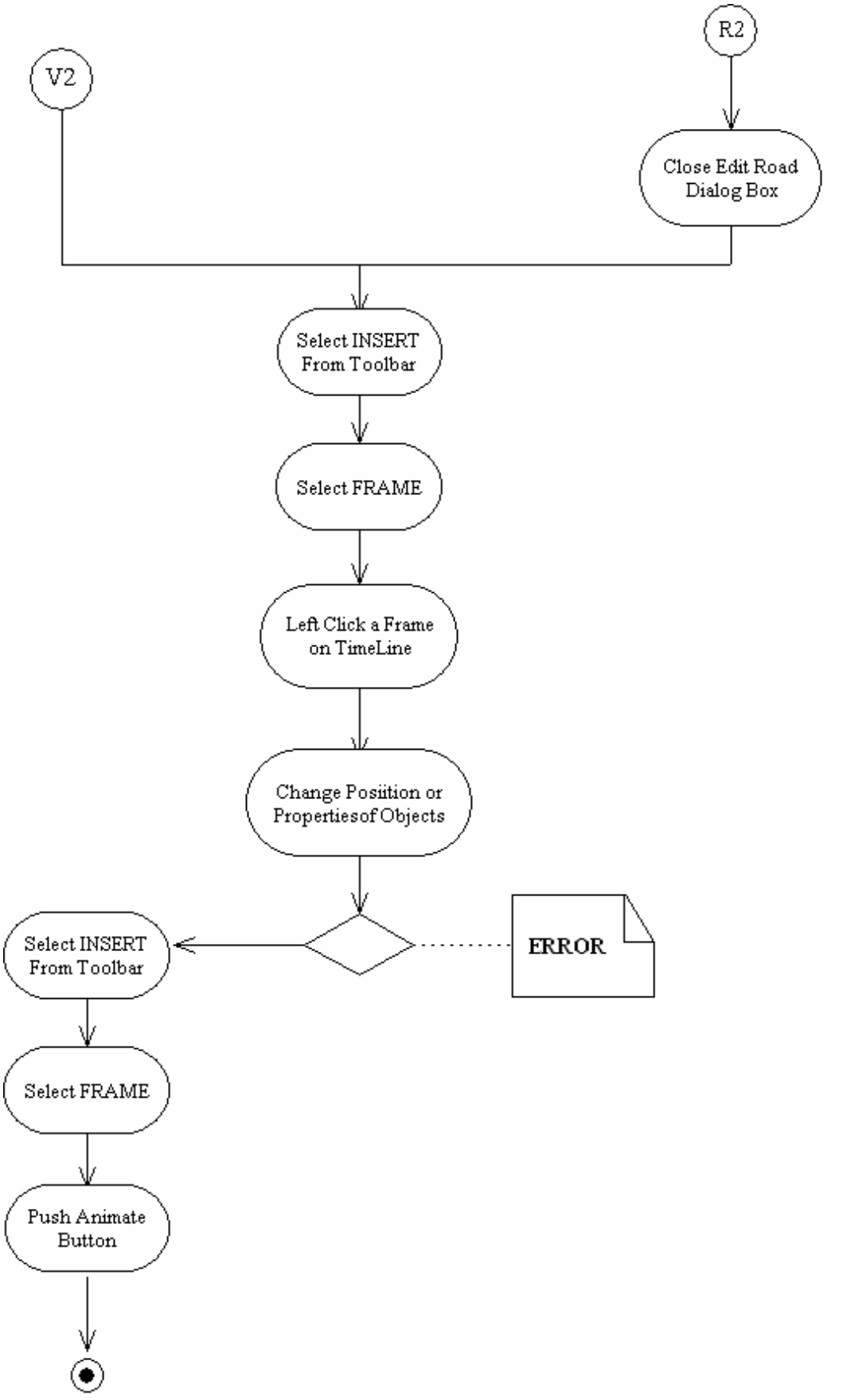

R2
V2
Close Edit Road Dialog Box
Select INSERT From Toolbar
Select FRAME
Left Click a Frame on TimeLine
Change Posiition or Propertiesof Objects
ERROR
Select INSERT From Toolbar
Select FRAME
Push Animate Button

Figure 3.8

4. SYSTEM DESIGN

4.1 System Data Structures

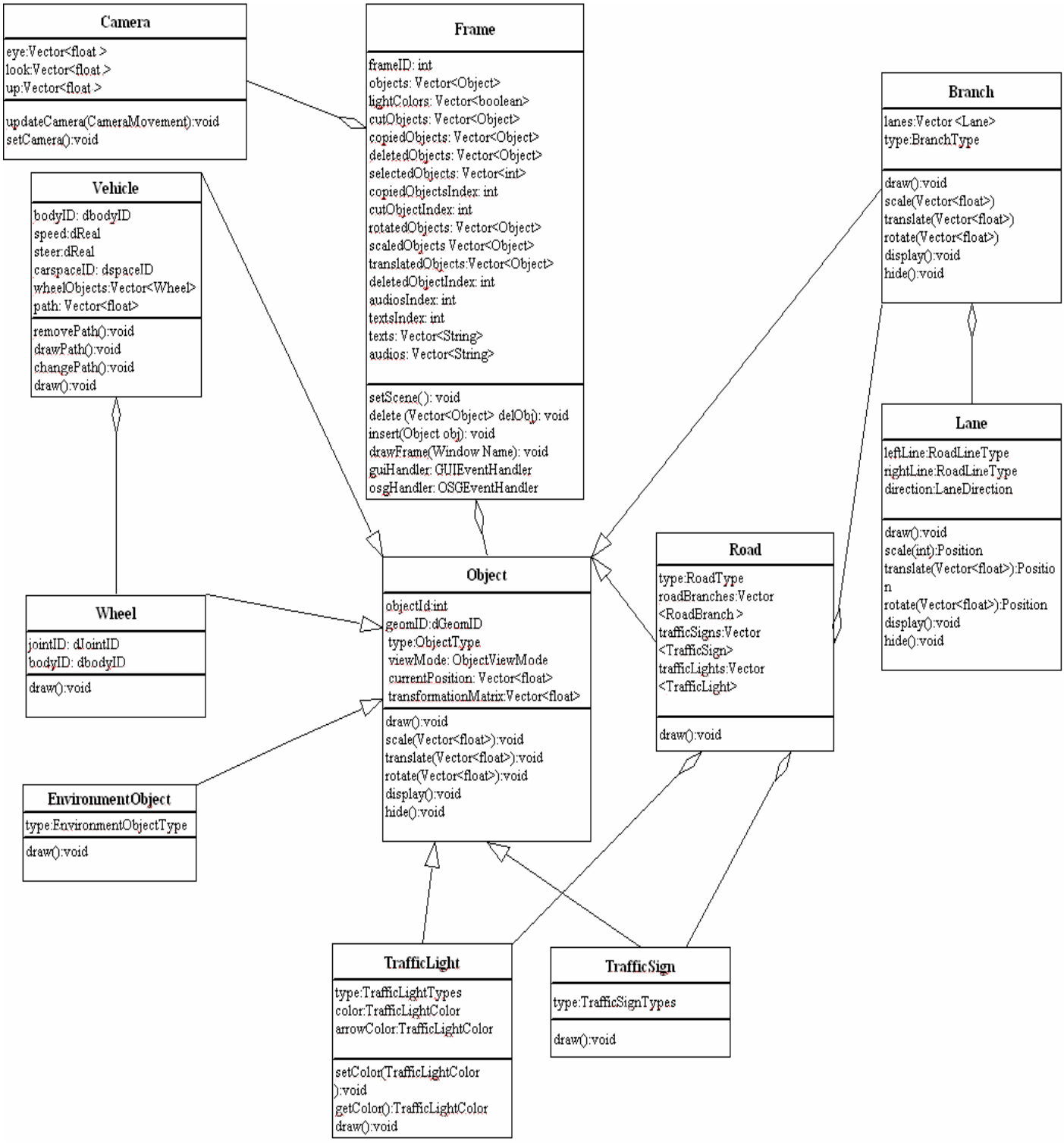


Figure 4.1

Frame Class

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

`frameID: int`

This member is an identification number for frames.

`objects: Vector<Object>`

The dynamic objects of the TraffEdu system that are possibly different in each frame is hold in this vector. These dynamic objects are instances of either `Vehicle` class or `TraficLight` class.

`lightColors: Vector<boolean>`

This vector holds the state information of traffic lights, if there exists a traffic light in the frame.

`cutObjects: Vector<Object>`

This vector holds the objects on which a cut operation is performed. The vector is adjusted by the `GUIEventHandler` class.

`copiedObjects: Vector<Object>`

This vector holds the objects on which a copy operation is performed. The vector is adjusted by the `GUIEventHandler` class.

`deletedObjects: Vector<Object>`

This vector holds the objects on which a delete operation is performed. The vector is adjusted by the `delete` function.

`selectedObjects: Vector<int>`

This vector holds the `objectId`s of objects that are currently selected. The vector is adjusted by the `GUIEventHandler` class.

`copiedObjectsIndex: int`

This is the index which points to an object in the `copiedObjects` vector whose `execute` function was called at the closest time. The value of the variable is adjusted by the `ComandCopy` class.

cutObjectsIndex: int

This is the index which points to an object in the `cutObjects` vector whose `execute` function was called at the closest time. The value of the variable is adjusted by the `ComandCut` class.

transformationObjects: Vector<Object>

This vector holds the objects on which a transformation operation is performed. The vector is adjusted by the `GUIEventHandler` class.

transformationObjectsIndex: int

This is the index which points to an object in the `transformationObjects` vector whose `execute` function was called at the closest time. The value of the variable is adjusted by the `CommandScale, CommandRotate and CommandTranslate` classes.

deletedObjectsIndex: int

This is the index which points to an object in the `deletedObjects` vector whose `execute` function was called at the closest time. The value of the variable is adjusted by the `ComandDelete` class.

audiosIndex: int

This is the index which points to an object in the `audios` vector whose `execute` function was called at the closest time. The value of the variable is adjusted by the `ComandInsertAudio` class.

textsIndex: int

This is the index which points to an object in the `texts` vector whose `execute` function was called at the closest time. The value of the variable is adjusted by the `ComandInsertText` class.

texts: Vector<String>

This vector holds the text for the frame.

audios: Vector<String>

This vector holds the audio file name for the frame.

setScene(): void

This function is responsible from camera settings like pitching, yawing and zooming operations.

delete(Vector <Object>): void

This function removes the objects from the `objects` vector according to the objects in the argument vector.

insert(Object): void

This function inserts the argument to the `objects` vector.

drawFrame(Window): void

This function is responsible from the rendering. It calls the `draw` methods of all objects in the `objects` vector.

guiHandler: GUIEventHandler

Each frame has a `GUIEventHandler` object.

osgHandler: OSGEventHandler

Each frame has a `OSGEventHandler` object.

Camera Class:

Whenever a keyboard callback related to camera positioning happens, `OSG EventHandler` class or `AnimationEventHandler` 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 looking at the parameter which tells the movement type of the camera. For example if it is `ZOOM_IN,` it will change the `eye` vector of the camera that keeps the position of the camera. Whenever a frame either in the animation or in the editor is rendered, `setCamera` method of the `Camera` class will be called to set the viewing position and viewing volume. This method will be the called from `drawFrame` method of `Frame` class using the `Camera` object of that frame in the editor, but in the animation this method will be called from the d`rawFrame` function of `Animation` Class using the `Camera` object of `AnimationWindow` class. `Camera` object is both involved in `Frame` class and `AnimationWindow` class. But the crucial point here is that, `Frame` objects that has `Camera` object is the ones in the editor, not in the animation phase. In the animation phase the user will not go backwards and see the previous frames, so we do not need to keep the camera positions of the previous frames, only one `Camera` object is enough for the animation. This is also one of the reasons why we need a separate event handler class for the animation phase. Whereas, while editing the environment in the editor,

user will also be able to update camera and this will be permanent since for each frame we had a separate `Camera` object.

Object Class:

Object class is the parent of classes that denotes the objects to be drawn in 3D environment. Since vehicles, wheels, roads, branches, 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.

`objectID:int`

This field will be used to identify objects. For example to be able to calculate the speed of a vehicle, we should find the difference in its position from one frame to another frame. Since each `Frame` object has its own `carObjects` vector, we should identify the same car by the help of its `objectID.`

`geomID:dGeomID`

Every object in the environment will have a geom object related to it and this field will keep the related `geomID,` and this field will be set for each object just before the animation starts by calling `createGeom` method from `initialize` method of `PhysicsEngine` class and assigning the returned dG`eomID` to object's `geomID`.

`type:ObjectType`

This field will be used to identify the model of the object to be rendered. To give an instance, If the `type` is `TRAFFIC_SIGN,` a predisposed traffic sign model will be used or if the `type` is `PATH` red lines will be rendered.

`viewMode:ObjectViewMode`

It identifies whether the object should be hidden or displayed in the screen.

`currentPosition: Vector<float>`

If a user clicks on one of `OSG Windows` after clicking on an icon in ToolBox Window or selecting insert from the Menu Bar, an object will be created at the clicked position. This will be the currentPosition of that object. Whenever a translation is done on that object, this will be updated. If the object is a `Vehicle,` this `currentPosition` should follow the path, means it should be on the path.

`transformationMatrix:Vector<float>`

This matrix will keep all the translations, rotations, and scaling on the object.

draw():void

If the object's viewMode is DISPLAY, this function will multiply the current matrix with the `transformationMatrix,` then it will draw the vertices of the object using the predisposed model according to the `ObjectType.` For the animation phase this `transformationMatrix` will be calculated using `getGeomPosition` or `getBodyPosition` giving the `geomID` (or `bodyID` if it is `Vehicle` or `Wheel` object) as parameter to these functions. The returned vector is the newly calculated `transformationMatrix.`

display():void

This function sets the object's viewMode to DISPLAY.

hide():void

This function sets the object's viewMode to HIDE.

translate(Vector<float>): void

`transformationMatrix` is recalculated multiplying it with the parameter vector.

scale(Vector<float>):void

`transformationMatrix` is recalculated multiplying it with the parameter vector.

rotate(Vector<float>):void

`transformationMatrix` is recalculated multiplying it with the parameter vector.

Vehicle Class:

`Vehicle` have properties such as, `speed`, `steer`, `path` vector of type float, and `wheelObjects` vector of type `Wheel`, car`spaceID` of type `dspaceID` and `bodyID` of type d`bodyID` that are specific to vehicle objects.

bodyID: dbodyID

Every `Vehicle` object in the environment will have a body related to it and this field will keep the related `bodyID,` and this field will be set for each `Vehicle` object just before the animation starts by calling `createBody` method from `initialize` method of `PhysicsEngine` class and assigning the returned dBody`ID` to vehicle's `bodyID`. `Vehicle` class has this field because the vehicle body and its `Wheel` bodies will be attached by joints.

speed:dReal

This is the desired speed of the motor that will be given to related hinge joints of the front wheels in the animation phase.

steer:dReal

This is the steering parameter of the motor that will be given to related hinge joints of the front wheels in the animation phase.

carspaceID: dspaceID

Every vehicle object will form a space with its wheels. In the `initialization` of the `PhysicsEngine` class after creating related geoms of the `Vehicle` and of its `Wheels`, we create a carspace calling `createCarSpace of PhysicsEngine` class.

wheelObjects: Vector<Wheel>

`Vehicle` and `Wheel` classes have hasa relationship since every `Vehicle` object strictly includes four `Wheel` object.

path: Vector<float>

This is a vector of float keeping the coordinates of the path's critical points. For example if the path is composed of two line segments, it will keep three coordinates denoting the beginning vertex of the path, the vertex in the middle and the ending point of the path.

drawPath():void

If the `path` vector of the `Vehicle` object is not empty, this function will draw lines using the coordinates in this vector. Whenever an object of type `VEHICLE` is drawn this method will be invoked.

removePath():void

This method deletes the path vector of a Vehicle object.

changePath():void

This method will be invoked whenever the path of the vehicle is needed to be changed. This will happen when the user clicks on path icon in the ToolBar or selects Change Path after right clicking on a selected vehicle. These two actions will set the `DrawMode` to DRAW_PATH, so when an event occurs in the `OSG Window` when the user left clicks, current selected vehicle object's changePath will be invoked and the mouse coordinates are added to path `vector and drawPath` method will be called after this.

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.

Wheel Class:

`Wheel` class has also inherits from `Object` class with some specific attributes like `jointID` and `bodyID.`

`jointID: dJointID`

Every `wheel` will be attached by joints to its vehicle, so for the related bodies of vehicles and wheels, we need to keep related joint ID. This field will be set for each `Wheel` object just before the animation starts by calling `createJoint` method from `initialize` method of `PhysicsEngine` class and assigning the returned d`JointID` to wheel's `jointID`.

`bodyID: dBodyID`

Every `Wheel` object in the environment will have a body related to it and this field will keep the related `bodyID,` and this field will be set for each `Wheel` object just before the animation starts by calling `createBody` method from `initialize` method of `PhysicsEngine` class and assigning the returned dBody`ID` to wheel's `bodyID`.

4.2 GUIEventHandler Class

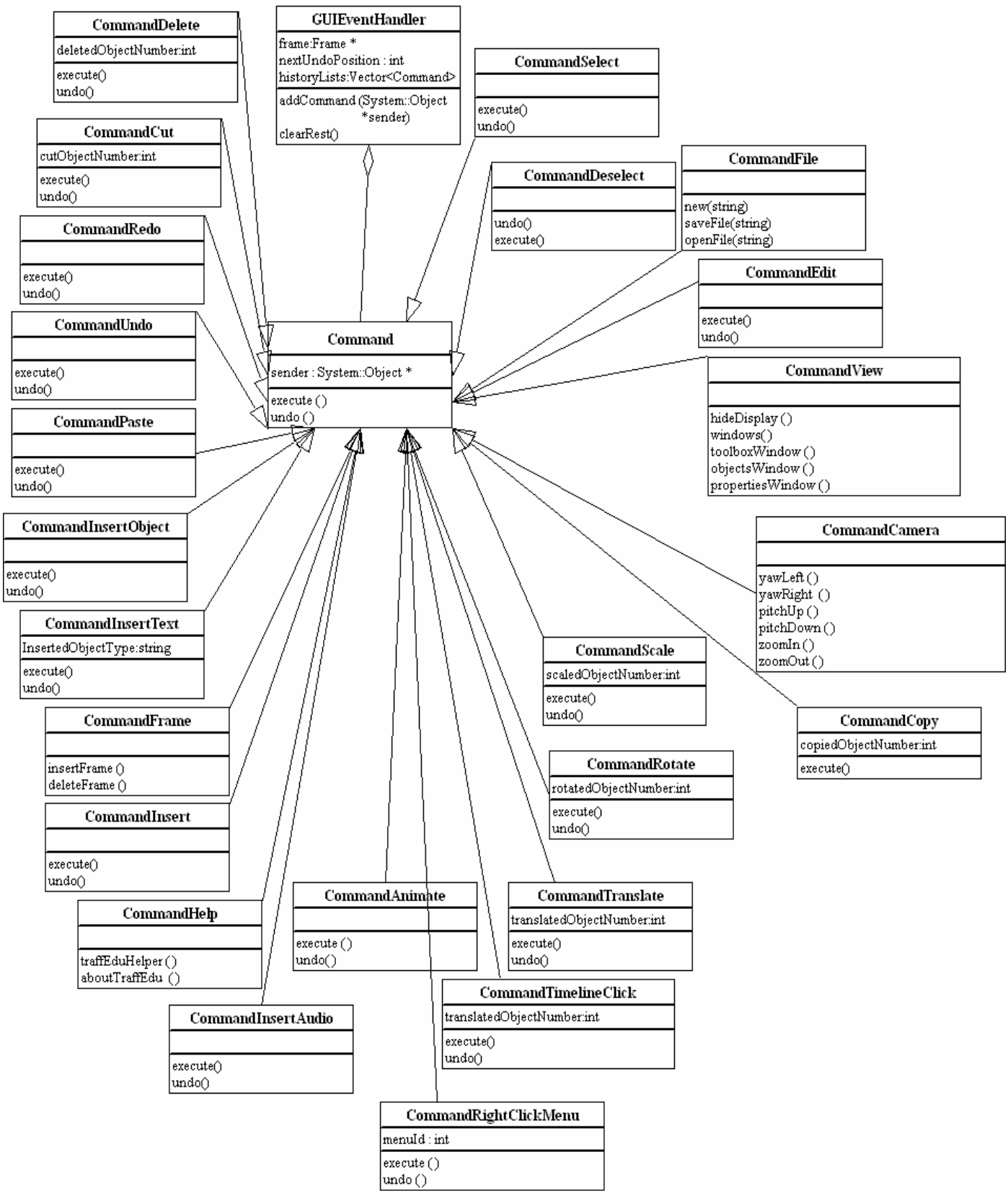


Figure 4.2

Main work-flow of TraffEdu System is, getting inputs from the user and operating accordingly. As a result, there are many kinds of events that should be handled in the system.

`GUIEventHandler` class is one of the event handler classes in the system that handles the requests coming from toolbar, toolbox window or `OSGEventHandler` class. Each `Frame` object has an instance of `GUIEventHandler` class. Whenever a `Frame` object is created, a `GUIEventHandler` object for that frame is also created automatically.

`GUIEventHandler` class is designed according to the rules of "command design pattern". "Command design pattern" encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations. Encapsulating each operation as a separate first-level object means that we can also easily support multilevel undo/redo operations.

The work sequence of this class is as follows:

Whenever a request comes from the frame, an object from the `Command` class is created. If the request is an undoable operation then the function, `addCommand,` adds the object for this request to the `historyLists` vector. Undo and redo operations are performed by traversing on the `historyLists` vector.

Detailed explanations of these data structures, requests and the response of the system are below:

`frame: Frame *`

The `frame` points to the `Frame` object which created this `GUIEventHandler` object.

`historyLists: Vector <Command>`

Each command operation that the user performs is stored in this vector.

`nextUndoPosition:int`

This variable holds the position of the next undoable `Command` object in the `historyLists` vector. Value of this variable is determined by the undo and redoes operations. The initial value for the variable is -1.

`addCommand(System Object *)`

This function deals with creation of the `Command` objects according to the argument `sender` and also addition of that object to the `historyLists` vector, if the operation is undoable. For example, if a cut request comes to the `GUIEventHandler`, `addCommand` function is activated. Then the function creates a `CommandCut` object and calls

`clearRest` function. After that, since cut operation is undoable, it increments the `nextUndoPosition` by one and adds the object to the `historyLists` vector. As a last step, it calls `execute` function of `CommandCut` object.

`clearRest()`

If the user performs a new command after some undo operations, this function clears the front of the `historyLists` vector to disable further redos.

It operates as follows;

Firstly `clearRest` function checks whether `nextUndoPosition` is equal to the size of the `historyLists` vector. If the condition holds, meaning there is no undo operation before the request, it returns. If not, meaning there has been done some undo operations before the request, it traverses `historyLists` vector from the last added command object to the object `nextUndoPosition` points. Each command object between these two points is checked.

For example, there is an instance of `CommandCut` class between those points. By using the information indicating how many objects were cut, `clearRest` function clears that number of objects from the front of the `cutObjects` vector, in the `Frame` object that `frame` points. After that, `CommandCut` object is popped out from the `historyLists` vector.

Other command objects between those two points are performed similarly.

Command Class

`Command` class is an abstract class used for gathering all kinds of command objects in the same structure. It has `undo` and execute functions that will be implemented in each inherited class of `Command` class.

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:

CommandSelect Class

In TraffEdu system user has opportunity to make single or multiple selections. User can select objects in the following ways:

- Select All Option:

  When this option is chosen by the user from the toolbar under `Edit` menu, the `CommandSelect` class constructor is called by the `addCommand` function, with the arguments indicating that a request to select all parameters has made. After that, all `objectId`s in the `frame` will be copied into the `selectedObjects` vector in the `frame` by the `execute` function of this `CommandSelect` class.

- Single Select Option:

  This option is chosen by the user by left-clicking on the object to be selected in the frame. This results in a request from `OSGEventHandler` to `GUIEventHandler` to create a `CommandSelect` object. After that, selected object's `objectId` in the `frame,` will be copied into the `selectedObjects` vector in the `frame` by the `execute` function of this `CommandSelect` class.

- Multiple Select Option:

  This option is activated when the CTRL key is pressed down and deactivated when the CTRL key is released up. If this option is active, user shall select the objects in the frame by left-clicking on them. This results in a request from `OSGEventHandler` to `GUIEventHandler` to create a `CommandSelect` object. After that, selected objects' `objectId`s (in the `frame),` will be copied into the `selectedObjects` vector (in the `frame)` by the `execute` function of this `CommandSelect` class.

The `undo` function of this class has a similar behavior with `execute` function of `CommandDeselect` class.

CommandDeselect Class

User can deselect objects in the following ways:

- Deselect All Option:

  This option can be chosen in two ways. One of them is by using the Edit menu in toolbar, and the other one is by left-clicking on an empty area in the `frame` if multiple selection option is not active. These results in a request from `OSGEventHandler` to `GUIEventHandler` to create a `CommandDeselect` object. After that, `selectedObjects` vector in the `frame` is emptied by the `execute` function of this `CommandDeselect` class.

- Single Deselect Option:

  This option can be chosen by left-clicking on an object if multiple selection option is active. This results in a request from `OSGEventHandler` to `GUIEventHandler` to create a `CommandDeselect` object. After that, `objectId` of the chosen object is removed from `selectedObjects` vector in the `frame.`

`The undo` function of this class has a similar behavior with `execute` function of `CommandSelect` class.

CommandDelete Class

In TraffEdu system user has opportunity to make single or multiple deletions. `CommandDelete` class handles GUI delete operations either coming from Edit menu in toolbar or from the RightClickMenu which is opened when the user clicks right button of the mouse on an object.

The `deletedObjectNumber` indicates how many objects are deleted from the frames. The value of this variable is set with the size of the `selectedObjects` vector.

The `execute` function carries all the objects whose `objectId`s exist in the `selectedObjects` vector from `objects` vector to the `deletedObjects` vector for the current frame and removes these objects from `objects` vector of following key frames.

`The undo` function copies `deletedObjectNumber` number of objects from the `deletedObjects` vector of current frame to the `objects` vector of this and all following key frames. Then `deletedObjectsIndex` is decremented according to the `deletedObjectNumber.`

CommandCut Class

In TraffEdu system user has opportunity to make single or multiple cut operations. `CommandCut` class handles GUI cut operations either coming from Edit menu in toolbar or from the RightClickMenu which is opened when the user clicks right button of the mouse on an object.

The `cutObjectNumber` indicates how many objects are cut from the frames. The value of this variable is set with the size of the `selectedObjects` vector.

The execute function carries all the objects whose objectIds exist in the selectedObjects vector from objects vector to the cutObjects vector for the current frame and removes these objects from objects vector of following key frames.

The undo function copies cutObjectNumber number of objects from the cutObjects vector of current frame to the objects vector of this and all following key frames. Then cutObjectsIndex is decremented according to the cutObjectNumber.

CommandCopy Class

In TraffEdu system user has opportunity to make single or multiple copy operations. CommandCopy class handles GUI copy operations either coming from Edit menu in toolbar or from the RightClickMenu which is opened when the user clicks right button of the mouse on an object.

The copiedObjectNumber indicates how many objects are copied from the frame. The value of this variable is set with the size of the selectedObjects vector.

The execute function copies all the objects whose objectIds exist in the selectedObjects vector from objects vector to the copiedObjects vector for the current frame. Then copiedObjectsIndex is incremented according to the copiedObjectNumber.

CommandPaste Class:

CommandPaste class handles GUI paste operations either coming from Edit menu in toolbar or from the RightClickMenu which is opened when the user clicks right button of the mouse on an object.

The execute function starts traversing the historyLists vector looking for a match either for a CommandCut or CommandCopy object. If it finds a match for CommandCut(CommandCopy), it copies cutObjectNumber (copiedObjectNumber) number of objects from cutObjects (copiedObjects) vector to the objects vector of the current and the following frames. Then cutObjectsIndex(copiedObjectsIndex) is decremented according to the cutObjectNumber(copiedObjectNumber).

The undo function starts traversing the historyLists vector from the index of CommandPaste object looking for a match either for a CommandCut or CommandCopy object. If it finds a match for CommandCut (CommandCopy), it removes

cutObjectNumber (copiedObjectNumber) number of objects from the objects vector of the current and the following frames according to the objectIds' of objects in cutObjects(copiedObjects)vector.

Then cutObjectsIndex (copiedObjectsIndex) is incremented according to the cutObjectNumber (copiedObjectNumber).

CommandUndo Class

When this option is chosen by the user from the toolbar under Edit menu, the CommandUndo class constructor is called by the addCommand function.

The execute function calls the undo function of the Command object which is pointed by the nextUndoPosition and steps backwards to the previous item by decrementing nextUndoPosition.

CommandRedo Class

When this option is chosen by the user from the toolbar under Edit menu, the CommandRedo class constructor is called by the addCommand function.

The execute function steps forward by incrementing nextUndoPosition and then calls the execute function of the Command object which is pointed by the nextUndoPosition.

CommandInsertObject Class

CommandInsertObject class handles GUI insert operation either coming from the insert menu in toolbar or from the toolbox by making drag & drop.

The insertedObjectType indicates which type of object will be created in the frame.

The execute function calls current frame's and the following frames' insert function. This results in creation of an object at the coordinates where the user determined.

The undo functions calls the delete function of the frame object and the object, whose objectId is given, is removed from the objects vector.

CommandInsertText Class

CommandInsertText class handles GUI insert text operation coming from insert menu in toolbar.

The execute function takes the text that the user inserted into the frame and sends it to the texts vector in the frame. TraffEdu System supports only adding one text for each frame so if there is a text inserted before, newly created text is written over the old one. Then textsIndex in the frame is incremented.

The undo function only decrements the textsIndex in the frame by one. If this value is zero, there will no text is written to the file while other parameters of the frame is being written to the file.

CommandInsertAudio Class

CommandInsertAudio class handles GUI insert audio operation coming from insert menu in toolbar.

The execute function takes the audio information that the user chose and sends it to the audios vector in the frame. TraffEdu System supports only adding one audio for each frame so if there is an audio inserted before, newly created audio information is written over the old one. Then audiosIndex in the frame is incremented.

The undo function only decrements the audiosIndex in the frame by one. If this value is zero, there will no audio information is written to the file while other parameters of the frame is being written to the file.

CommandScale Class

In TraffEdu system user has opportunity to scale single or multiple objects at the same time. CommandScale class handles GUI scaling operation coming from the RightClickMenu which is opened when the user clicks right button of the mouse on the selected area in the frame.

The scaledObjectNumber indicates how many objects are scaled in the frame. The value of this variable is set with the size of the selectedObjects vector.

The execute function applies scaling operation to each object in the selectedObjects vector by using the objectIds stored in the selectedObjects vector. Before multiplying current matrix of the object with the scale matrix, that matrix is stored in the transformationObjects vector in the frame and transformationObjectsIndex is incremented by one.

The undo function loads the matrixes which are stored in the transformationObjects vector over the current matrix of the related objects. Then transformationObjectsIndex is decremented by one.

CommandRotate Class

In TraffEdu system user has opportunity to rotate single or multiple objects at the same time. CommandRotate class handles GUI rotation operation coming from the RightClickMenu which is opened when the user clicks right button of the mouse on the selected area in the frame.

The rotatedObjectNumber indicates how many objects are rotated in the frame. The value of this variable is set with the size of the selectedObjects vector.

The execute function applies rotation operation to each object in the selectedObjects vector by using the objectIds stored in the selectedObjects vector. Before multiplying current matrix of the object with the rotation matrix, that matrix is stored in the transformationObjects vector in the frame and transformationObjectsIndex is incremented by one.

The undo function loads the matrixes which are stored in the transformationObjects vector over the current matrix of the related objects. Then transformationObjectsIndex is decremented by one.

CommandTranslate Class

In TraffEdu system user has opportunity to translate single or multiple objects at the same time. After selecting objects in the frame, user shall make drag & drop. This results in a request from OSGEventHandler to GUIEventHandler to create a CommandTranslate object.

The translatedObjectNumber indicates how many objects are translated in the frame. The value of this variable is set with the size of the selectedObjects vector.

The execute function applies translation operation to each object in the selectedObjects vector by using the objectIds stored in the selectedObjects vector. Before multiplying current matrix of the object with the translation matrix, that matrix is stored in the transformationObjects vector in the frame and transformationObjectsIndex is incremented by one.

The undo function loads the matrixes which are stored in the transformationObjects vector over the current matrix of the related objects. Then transformationObjectsIndex is decremented by one.

CommandFile Class

CommandFile class handles file operations coming from file menu in the toolbar. Since undo and redo operations can not be applied to this class, constructed CommandFile object is not added to the historyLists vector.

The new method opens a new OSG window to provide a blank environment to the user.

The open method calls loadFile method in the FileHandler class with giving file name as the parameter to the method.

The close method firstly checks if the current file is saved. If not, a pop-up menu appears to check if the user wants to save the current file or not. According to the user intent save method is called or current OSG window is deleted.

The save method calls saveFile method in the FileHandler class with giving file name as the parameter to the method.

The saveAs method works similar with the save method, only sent parameter name is changing according to the user request.

The currentFiles method is used to make all currently name of opened files appear.

The exit method is used for exiting from the system but before this it calls close method for each currently opened files.

CommandView Class

CommandView class handles view operations coming from view menu in the toolbar. Since undo and redo operations can not be applied to this class, constructed CommandView object is not added to the historyLists vector.

The hideDisplay method hides or displays the selected type of objects in the frame.

The windows method shows sub-windows which are not on the display or hides sub-windows which are on the display.

The toolboxWindow method hides or displays the toolbox window.

The objectsWindow method hides or displays the objects window.

The `propertiesWindow` method hides or displays the properties window.

CommandCamera Class

In TraffEdu system camera properties are adjusted either from camera menu in the toolbar or from the keyboard. If the user wants to change camera properties from the keyboard, a request is created by `OSGEventHandler` and sent to the `GUIEventHandler.`

The `yawLeft` method rotates the camera head to the left.

The `yawRight` method rotates the camera head to the right.

The `pitchUp` method rotates the camera head to the up.

The `pitchDown` method rotates the camera head to the down.

The `zoomIn` method zooms in to the current position of the camera reference point through the camera view direction. When an object is selected zooming action is done by centering to the object that means changing camera reference point as the center of the selected object.

The `zoomOut` method zooms out from the current position of the camera reference point through the camera view direction. When an object is selected zooming action is done by centering to the object that means changing camera reference point as the center of the selected object.

CommandFrame Class

As it is mentioned before, in TraffEdu system cut, copy, and paste operations are not supported. Only inserting a new frame or deleting a created frame is supported.

The `insertFrame` method firstly calls `checkInconsistency` method of the `EditorChecker` class by giving pointer of the temporary frame. By this way correctness of everything in the frame for the animation is guaranteed. Then `searchFrames` method in the `OSGWindows` class is called. This function returns if there is a frame in the `keyFrames` vector whose `frameId` is same as the temporary frame. If it returns true, means there is a frame whose `frameId` is same with the temporary frame, that frame is removed from the `keyFrames` vector. Finally temporary frame is inserted to the `keyFrames` vector.

The `deleteFrame` method firstly calls `searchFrames` method in the `OSGWindows` class. If it returns true, means there is a frame whose `frameId` is same with the temporary frame, that frame is removed from the `keyFrames` vector. If not, nothing is

done because temporary frame has not been inserted to the `keyFrames` vector in the `OSGWindows` class.

CommandHelp Class

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

The `traffEduHelper`, `samples`, and `aboutTraffEdu` are the methods of this class whose aim is to help the user while preparing the animation.

CommandAnimate Class

`CommandAnimate` class is created when the user clicks on the animate button on the GUI. The `execute` function of this class calls `animate` method of the `AnimationWindow` class. By this way prepared traffic case is animated on a newly created window.

CommandRightClickMenu Class

In TraffEdu system when the user right clicks on the OSG window, a request from `OSGEventHandler` to `GUIEventHandler` to open a right click menu is sent.

The `menuId` indicates which right click menu will be popped up when the user clicks right button of the mouse.

The `execute` function of this class opens a right click menu by using the `menuId`.

CommandTimelineClick Class

There are many cases when the user clicks on the timeline. These cases and how they are handled are as follows:

When the user clicks on the timeline,

- If `keyFrames` vector in the `OSGWindows` is empty a blank temporary frame is created.
- If `keyFrames` vector in the `OSGWindows` is not empty and there is no frame with the same `frameId` with the clicked area of the timeline, a temporary frame is created. This frame contains same environment objects with the last key frame, whose `frameId` is less than the `frameId` of the frame.

- If `keyFrames` vector in the `OSGWindows` is not empty and there is a frame with the same `frameId` with the clicked area of the timeline, that frame is loaded to the newly created temporary frame.

The `execute` function performs all the conditions by the help of `searchFrames` method in the `OSGWindows` class.

4.3 EditorChecker Class

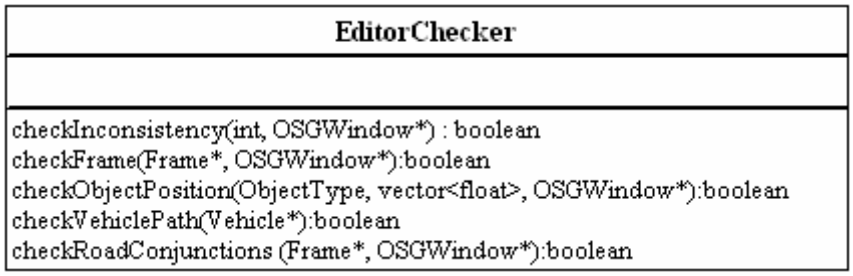


Figure 4.3

`EditorChecker` class, as the name implies, checks the operations done in TraffEdu Editor so that the user can be notified if he does conflicting operations. The consisitency in each frame is checked whenever check consistency button is pressed, that means `GUIEventHandler's commandCheckConsistency` object will create a `EditorChecker` object and call `checkInconsistency` method of that object, or the user inserts the current environment as a new frame, that means GUIEventHandler's `commandInsertFrame` object will create a `EditorChecker` object and call `checkInconsistency method`.

`checkInconsistency(int, OSGWindows *)`

This method calls `checkFrame`, for the frame whose `frameID` given as first argument, so a process of checking the frame for different types of consistencies will be started. The second argument is a pointer to the `OSGWindows` object that has that frame. We need this pointer to be able to reach static frame objects like roads, traffic signs and lights which are kept in `OSGWindows` object but not in `Frame` object, since these objects are not specific to a single frame.

checkFrame(Frame *, OSGWindows *):boolean

This function calls checkObjectPosition for each object in the frame which is pointed by the first argument. Then for Vehicle type objects it calls checkVehiclePath, and for Road type objects it calls checkRoadConjuctions.

checkObjectPosition (ObjectType, Vector<float>, OSGWindows *):boolean

This function checks whether the object placements are done appropriately. It checks whether the object with object type denoted with the first argument could 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. Following are the constraints about positioning of objects.

- A vehicle can only be inserted on road and should follow its path.
- A road can only be inserted on the grid.
- A path can only be inserted on road.
- Traffic lights and traffic signs can only be inserted on road.
- Environmental objects can not be inserted on the road.

checkVehiclePath(Vehicle *):boolean

If the vehicle whose pointer is given as argument has empty path vector, this function returns false, otherwise true. A warning message should be generated since the user will not be able to change position of the vehicle in the next frames unless it has a path associated with it.

checkRoadConjuctions(Frame *, OSGWindows *):boolean

The direction of the lanes of the two branches of two different roads should match if these branches are placed side by side. This means if a lane of a branch with DOGU_BATI direction meets a lane of another branch with BATI_DOGU direction this function will return false.

4.4 OSGWindows Class

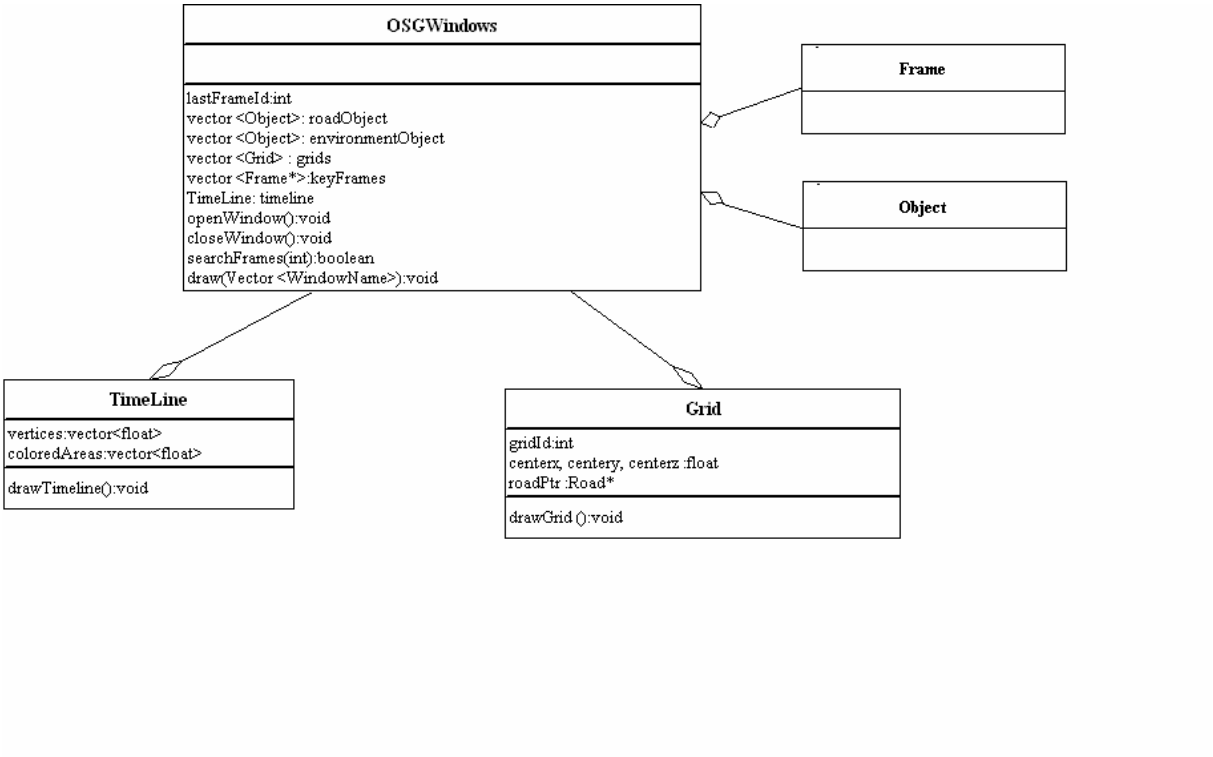


Figure 4.4

`lastFrameId:int`

`lastFrameId` is the variable that keeps the id of the last keyframe inserted. This variable is necessary for allocating space for the `animationFrames` vector of the `AnimationWindow` object. The size of the vector is calculated by using this variable.

`roadObject: Vector<Object>`

`environmentObject: Vector<Object>`

`grids: Vector<Grid>`

`Grid` objects, `Road` objects, the `TrafficSign` and `TrafficLight` object that the roads contains and `Environment` objects are static for each frame, their position and orientation does not change from frame to frame. That is why they are kept in this class rather than `Frame` class.

`keyFrames: Vector<Frame*>`

The key frames that the user inserted are pushed back to the `keyFrames` vector in the order the user inserted.

timeline: TimeLine

This object is drawn with each call to `draw` function.

openWindow()

This function initializes the main display area of the editor.

closeWindow()

This function clears the main display area of the editor.

searchFrames(int):boolean

This function returns true if the `keyFrames` vector of this class contains a frame with id that is same with argument and false otherwise.

draw(Vector <WindowName>)

This function calls `draw` functions for objects in `roadObject`, `environmentObject`, `grids`, `keyframe` vectors of this class. The `timeline` object is also displayed by setting a predetermined orthographic projection. The argument determines which of the views will be displayed in the main area. `3D_VIEW` mode displays perspective projection of the environment. `TOP_VIEW mode displays` orthogonal projection of the environment from top view. `FRONT_VIEW mode` displays orthogonal projection of the environment from front view. `SIDE_VIEW` mode displays orthogonal projection of the environment from side view. The viewports are adjusted according to the size of the argument vector which is the number of viewing modes. Each mode is drawn by adjusting the properties of the `camera` object of current frame.

TimeLine Class

vertices:vector<float>

This vector stores the vertices of each rectangle in the timeline.

coloredAreas:vector<float>

This vector keeps the indexes of the rectangles for which a frame is inserted.

drawTimeline()

This function draws the `vertices` vector as a sequence of rectangles.

Grid Class

`Grid` class is used to help the user visualize the environment better. Also grids help in consistency checking. The user is not allowed to put the cars on the grids on which no roads are located. Grids are fixed for all frames.

gridId:int

It is the identification number for each grid.

centerx, centery, centerz :float

They are x, y and z coordinates of each grid.

roadPtr :Road*

This pointer is set to the `Road` object if such a `Road` object is placed on this grid. This is needed in order to check if a `Vehicle` type object is placed on a `Road` type object or not.

drawGrid()

This function is responsible for drawing the grid on `OSGWindows`.

OSGEventHandler Class

`OSGEventHandler` class is one of the event handler classes in the system that handles the requests coming from the OSG windows. It manages the operations of the callback functions of the openscenegraph. Each `Frame` object has an instance of this `OSGEventHandler` class. Whenever a `Frame` object is created, an `OSGEventHandler` object for that frame is also created automatically.

The `mouseControl` function in `OSGEventHandler` considers the following possible conditions:

- ***Mouse Left-Click (with CTRL key released)***
  - On Empty Area

    There is no action taken against such a condition by the `OSGEventHandler` class.
  - On a Selected Object

    There is no action taken against such a condition by the `OSGEventHandler` class.
  - On a Deselected Object

    `OSGEventHandler` class leaves the management of this callback to `GUIEventHandler` class by calling the `addCommand` function with a parameter to indicate that a `CommandSelect` object should be created.

- ***Mouse Left-Click (with CTRL key pressed)***
  - On Empty Area

    There is no action taken against such a condition by the `OSGEventHandler` class.
  - On a Selected Object

    `OSGEventHandler` class leaves the management of this callback to `GUIEventHandler` class by calling the `addCommand` function with a parameter to indicate that a `CommandDeselect` object should be created.
  - On a Deselected Object

    `OSGEventHandler` class leaves the management of this callback to `GUIEventHandler` class by calling the `addCommand` function with a parameter to indicate that a `CommandSelect` object should be created.
- ***Mouse Right-Click (with CTRL key released)***
  - On Empty Area

    `OSGEventHandler` class leaves the management of this callback to GUIEventHandler class by calling the `addCommand` function with a parameter to indicate that an appropriate `CommandRightClickMenu` object should be created.
  - On a Selected Object

    `OSGEventHandler` class leaves the management of this callback to `GUIEventHandler` class by calling the `addCommand` function with a parameter to indicate that an appropriate `CommandRightClickMenu` for that selected object should be created.
  - On a Deselected Object

    There is no action taken against such a condition by the `OSGEventHandler` class.
  - On TimeLine

    `OSGEventHandler` class leaves the management of this callback to `GUIEventHandler` class indicating a `CommandTimelineClick` object should be created.

When met with a mouse motion callback on an object, the `OSGEventHandler` class calls the `addCommand` function of the `GUIEventHandler` indicating that a `CommandTranslate` object should be created.

updateCamera(float, float, float, float, float, float, float, float, float)

When a keyboard callback occurs, this function updates the camera properties which are position, reference point and the look up vector.

4.5 AnimationWindow Class

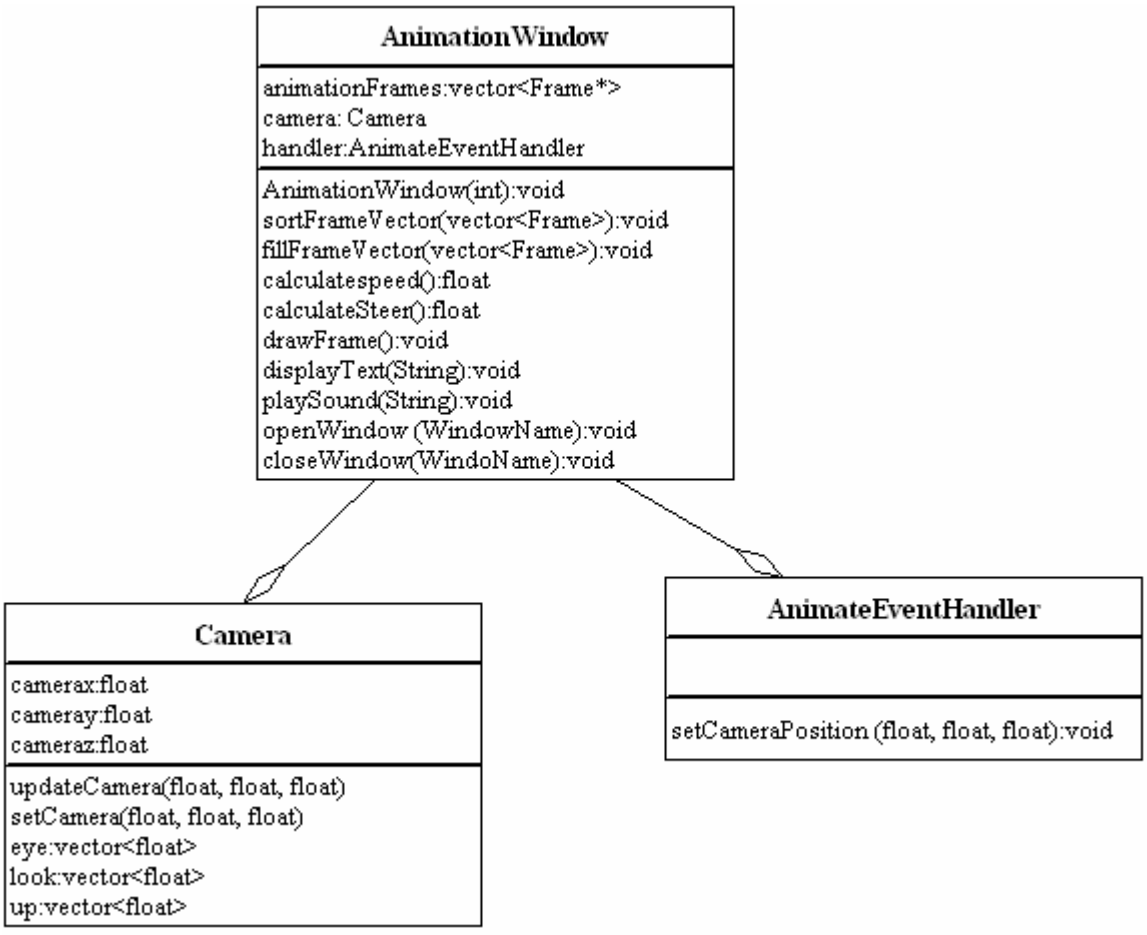


Figure 4.5

`AnimationWindow (last_frame_id:int)`

The size of `animationFrames` vector will be equal to `30*5*last_frame_id`. `last_frame_id` is the identification number of the frame that is inserted lastly in `OSGWindows` object. This number is multiplied by 30, which is the number of frames displayed per second in the animation, and 5. The user is allowed to insert frames into places in timeline which are multiple of 5 seconds--similar to the timeline of the Macromedia Flash Editor. So the `last_frame_id` must be multiplied by 5 to get how many seconds of animation is required in total.

camera: Camera

A camera object is created and located in a default position in the scene at the beginning of the animation. As the input comes from the keyboard, the callback function, `setCameraPosition`, of the `AnimateEventHandler` object locates the camera to the new location. As stated before in the explanation of `Camera` class, in the animation phase only one camera object is created instead of one camera object per frame, therefore a change in the position or orientation of the camera will be permanent for all frames until a new keyboard event occurs.

animationFrames:vector<Frame *>

When the animate button is pressed an object of this class is created. The in-between frames of the `frames` vector in `OSGWindows` object, are created and collected in the `animationFrames` vector by calculating and filling the fields of objects.

animate () : void

This function will call `sortFrameVector`, `fillFrameVector`, `calculateSpeed`, `calculateSteer`, `openWindow` functions of this call in succession.

sortFrameVector(vector<Frame>)

The `frames` vector in `OSGWindows` class is filled with frames in the order, the user inserts them. For calculations, two consecutive keyframes must be detected. In order to do so, the frames vector of `OSGWindows` is sorted by `sortFrameVector` function according to the frame ids.

fillFrameVector(vector<frame>)

`fillFrameVector` function fills the in-between frames by incrementally taking two consecutive frames from its argument vector. The necessary `speed` and `steer` properties of `Vehicle` objects and color properties of the `trafficLight` objects are adjusted.

calculateSpeed( )

After setting the corresponding values for objects in `animationFrames`, speed of `Vehicle` objects in each frame is calculated by `calculateSpeed` function using the position difference of Vehicles between frames. This `speed` property of the `Vehicle` objects will be used in setting the speed of each motor for each `worldstep` for making calculations in ODE.

`calculateSteer( )`

The steer of each `Vehicle` object is calculated for each frame according to the checkpoints of the path of the car in `calculateSteer` function. The checkpoints are the points where the angle of the path is changed. In these points the steer of the vehicle will be changed accordingly. For the vehicle to make the rotations realistic, the car should start steering before reaching the checkpoints. This steer property of each object will be used to set the steers of the cars in each worldstep for making calculations in ODE.

Steer and speed properties of the vehicles are not calculated during the animation but just before the animation while filling the in-between frames.

`displayText(String)`

This function displays the String object given as its argument.

`playAudio(String)`

This function plays audio by calling the playSound function of Audio class with its argument.

`openWindow( )`

After making the calculations for sorting the array, filling the array with necessary frame information, calculating the speed and steer of the each vehicle object, the `openWindow` function is called for passing to the animation window from the editor window. A new window is opened in the screen and the `drawFrame` function is called.

`drawFrame( )`

The objects vector in `OSGWindows` class keeps the positions static objects like roads, traffic lights and traffic signs. For current frame, the `RoadObject` and `EnvironmentObject` vectors are drawn since the elements in these arrays are stable during the animation. The color of light for each `TrafficLight` object is stored in `lightColor` vector of `animationFrames.` So the change in traffic lights for each frame is achieved by this way.

The necessary information for the `Vehicle` objects is collected in the `animationFrames` vector. This is the speed and steer information. At each frame we give the speed and steer to motors and step the world and get the positions of the bodies to update the position of vehicles and wheels, namely the dynamic objects.

Draw static environment

If time equals timeStep

then

increment currentFrame

run motor with speed and steer of currentframe

worldstep(timeStep)

for each vehicle objects

vehicle curren tposition = getBodyPosition

for each wheel object of vehicle

wheel curren tposition = getBodyPosition

draw vehicle and the wheels in their current position

In addition, the `animationFrames` vector's `texts` vector is checked if this frame contains textual information. If `textsIndex` variable of this frame is not `-1`, then `displayText` is called with argument `texts[textsIndex].`

For audio play, the `animationFrames` vector's audios vector is checked if this frame must be displayed with audio. . If `audiosIndex` variable of this frame is not `-1`, then `playAudio` is called with argument `audios[audiosIndex].`

`closeWindow()`

This method exits the animation window.

AnimateEventHandler Class

Although animation window is also a `OSGWindow,` a separate class is needed to handle this window events. The reason is due to the different responses of the animation window to the events coming from the keyboard or mouse. For example, a right click on other OSG windows will result in calling the `GUIEventHandler` class to create a GUI related to the current selected object, whereas in animation window all inputs coming from the user is related to camera positioning.

`setCameraPosition (CameraMotion)`

This function handles the inputs from the keyboard and adjusts the camera positions by calling the `updateCamera` function of the `AnimationWindow` class's `camera` object.

4.6 PhysicsEngine Class

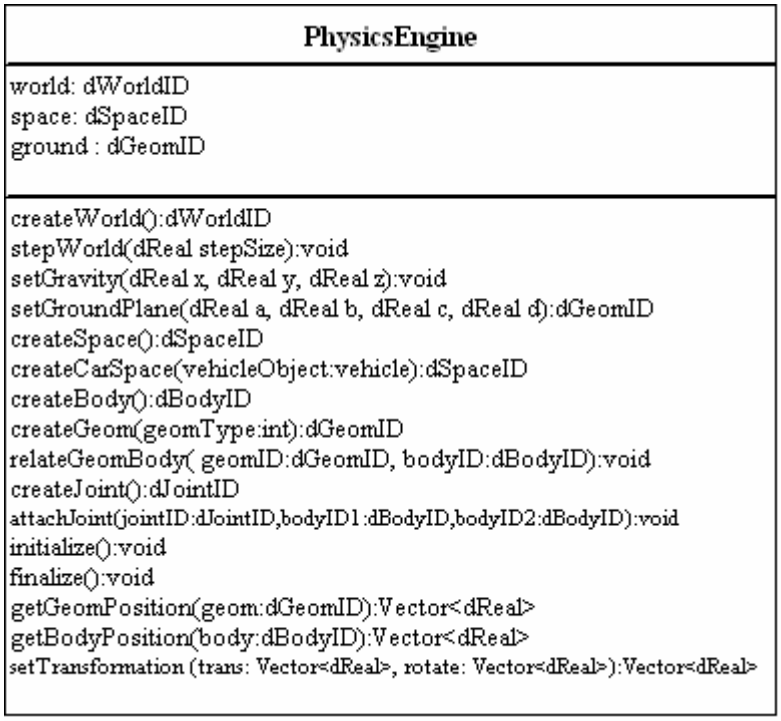


Figure 4. 6

The `Physics Module` is used for simulating vehicles in the system and applies the essential physics to the bodies in the environment so that the resulting animation becomes more realistic. All the objects in the environment should be exposed to gravitation, in addition to this dynamic `vehicle` objects should be exposed to friction. Motion with acceleration, deceleration or constant speed should be simulated for those dynamic objects. What is more, collision detection should be done and collisions should be simulated in consistency with the real world physics. It is decided to use Open Dynamics Engine (ODE) library for these purposes to supply the collision detection and velocity control of the dynamic bodies in the created world.

`PhysicalEngine` class makes the necessary ODE initializations with the call of `initialize` function to constitute the static environment of a typical simulation in animation phase of TraffEdu system. Firstly, it is invoked to create a world with `createWorld` function to embed the bodies of the `objects` and sets the gravity for that world. Then, a physical space is created with `createSpace` function to handle the collision detections. Finally the ground plane will be created in the space with `createPlane`

function to set the ground of the environment. Planes are non-placeable geoms, therefore they do not have an assigned position and rotation. In other words it is assumed that the plane is always part of the static environment and not tied to any movable object.

For each `vehicle` and for its four `wheel objects` a rigid body should be created and the corresponding rigid bodies should be attached with joints. To achieve this `createBody` function of the `PhysicalEngine` class should be called, this function will return the created dBodyID which will be recorded in the bodyID field of the `object.` To attach a joint to a `vehicle` object body and its `wheels'` bodies' `attachJoint` method will be used. Similarly for each `object` a geom should be created and should be related to its body if one exists. The former will be done by `createGeom and` the latter. What is more, for each `vehicle` and its `wheels` these geoms should be added to simple car space for collision handling.

Before rendering each frame the world should be stepped once by the `stepWorld` function so that the new dynamics can be calculated and the body or geom positions can be get in the correct place by calling the `getBodyRotation` /`getBodyPosition` or `getGeomPosition` /`getGeomRotation.`

The hierarchy of physics module is shown in Figure 4.6.

`world: dWorldID` Identifies the world

`space: dSpaceID` Identifies the space.

`ground : dGeomID` Identifies the ground geom.

`dWorldID :createWorld()`

Calls dWorldCreate() function of ODE and returns the created world ID

`void : stepWorld(dReal stepSize)`

Calls dWorldStep (world, stepSize) function of ODE

`void : setGravity(dReal x, dReal y, dReal z)`

Calls dWorldSetGravity (world ,x, y, z) so that the world's global gravity vector is set

`dGeomID:setGroundPlane(dReal a, dReal b, dReal c, dReal d)`

Calls dCreatePlane (space,a,b,c,d) to create a plane for ground with the equation ax+by+cz=d and returns the created plane geom ID.

`dSpaceID : createSpace()`

Calls dHashSpaceCreate (0) function of ODE and returns the created space ID

dSpaceID : createCarSpace(vehicleObject:vehicle)

Calls dSimpleSpaceCreate (space) function of ODE and adds the vehicle's and its wheels' geom to that space by calling dSpaceAdd (dSpaceID, dGeomID) function of ODE and returns the created spaceID so that the vehicle object can set its carSpaceID parameter.

dBodyID : createBody()

Calls dBodyCreate (world) and returns the created body ID.

dGeomID: createGeom(geomType:int)

Calls dGeomCreate (`geomType`) and returns the created geom ID.

void : relateGeomBody(geomID:dGeomID, bodyID:dBodyID)

Calls dGeomSetBody (geomID, bodyID) function of ODE.

dJointID : createJoint()

Calls dJointCreateHinge2 (world,0) and returns the created joint ID.

void:attachJoint(jointID:dJointID,bodyID1:dBodyID,bodyID2:dBodyID)

Calls dJointAttach (`jointID`, `bodyID1`, `bodyID2`)

void : initialize()

Calls `createWorld` ,`setGravity`, `createSpace`, `setGroundPlane` methods and for each object in the environment creates needed body, geom and joints and relates these as explained above.

void : finalize()

Destroys world, spaces, contact groups and geoms using the dWorldDestroy, dSpaceDestroy, dGeomDestroy, dJointGroupDestroy methods of ODE.

Vector<dReal>:getGeomPosition(geom:dGeomID)

It returns `setTransformation(`dGeomGetPosition(geom),dGeomGetRotation(geom))

Vector<dReal>:getBodyPosition(body:dBodyID)

It returns `setTransformation(`dBodyGetPosition(body),dBodyGetRotation(body))

Vector<dReal>: setTransformation (trans: Vector<dReal>, rotat: Vector<dReal>)

Calculates a 4 by 4 matrix from the given 4 by 3 matrix and returns the result.

4.7 FileHandler Class

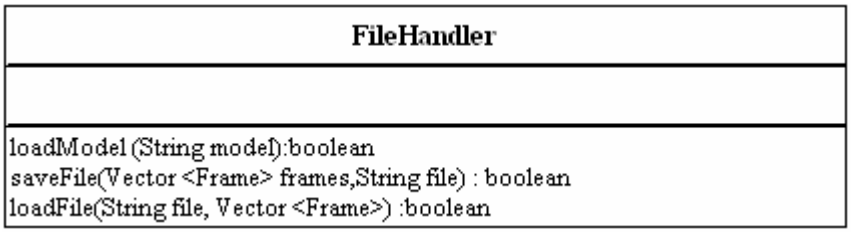


Figure 4.7

This class 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 an XML file and so involve in the reconstruction of the frames vector for the traffic case prepared previously. The last one is to write the prepared traffic scene to an XML formatted file.

In TraffEdu, premeditated 3ds max models will be used. 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. 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 on 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.

`boolean loadModel(String)`

The function will read the vertices and normal vector information of objects from the given argument file name. If read operation is successful (unsuccessful), it returns a true (false) boolean value.

`boolean saveFile(Vector <Frame>,String)`

This function 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. The function saves only the `objects, audios, texts, lightcolors` vectors for each frame and `roadObject, environmentObject` vectors in `OSGWindow` class. If save operation is successful (unsuccessful) it returns a true (false) boolean value.

`boolean loadFile(String, Vector <Frame> *)`

This function simply takes two arguments one of which is for the frames to be loaded in and the other for the name of the file to be loaded from. If save operation is successful (unsuccessful), it returns a true (false) boolean value.

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 are given in A.2.

4.8 Audio Class

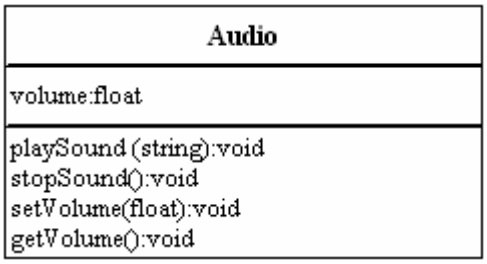


Figure 4.8

TraffEdu will have some audios like speaking of the user and effects in the animation. `Audio` class will be implemented to control the sounds. `Audio` will interact with `Physics` class to play crash effects if a traffic accident happens in the animation. Also it will interact with `AnimationWindow` class to play some music or effects inserted by the user in the 3D Editor part.

Functions in osgAL, toolkit for handling spatial (3D) sound in the OpenSceneGraph rendering library, will be used to implement this module.

- osg::Sample will be created to hold the .vaw sample file for each audio object.
- osgAL::SoundState will be created to hold the samples and its settings. Volume of the sound is one of them.
- osgAL::SoundNode will be created to hold a SoundState and this node will be inserted wherever an audio object is created.
- osgAL::SoundManager will be created to handle queued SoundStates and to store all SoundStates to make it possible to find them later on. For example, when a crash occurs in a frame and user has been inserted an `Audio` object into that frame in the animation preperation phase, both sound events coming from `AnimationWindow` class and `Physics` class will be pushed into the queue and played according to their priority.

The `volume` variable indicates the volume (gain) of the sound state.

The `playSound` method is used for creating a new audio stream according to the given sound name and playing the stream by using setPlay method of osgAL::SoundState setting "true" as a parameter.

The `stopSound` method is used for stopping the stream which is being played by using setPlay method of osgAL::SoundState setting "false" as a parameter.

The `setVolume` method is used for setting the volume of the stream by using setGain method of osgAL::SoundState.

The `getVolume` method is used for getting the volume of the stream by using getGain method of the osgAL::SoundState.

4.9 Class Diagrams Overview

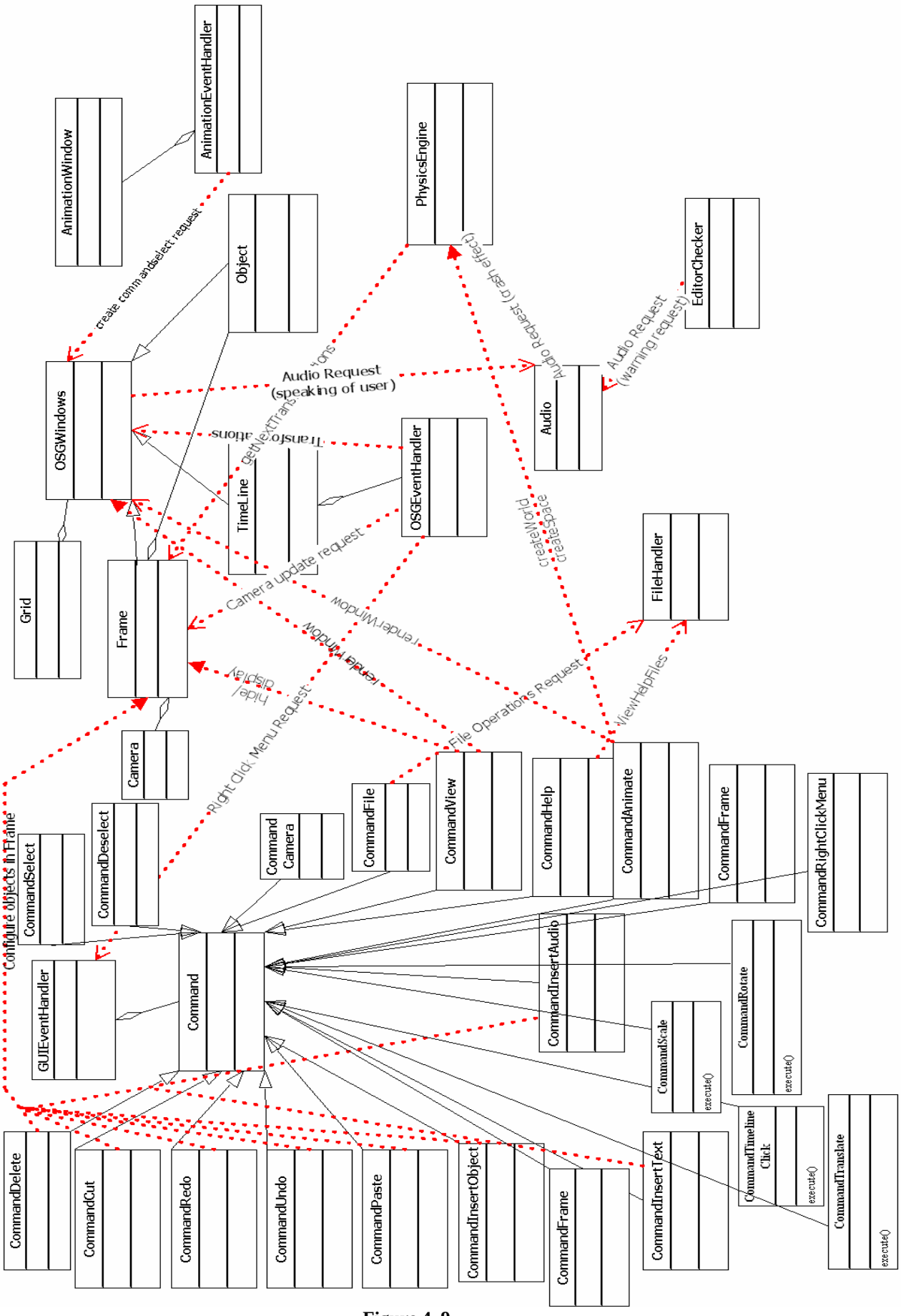

AnimationEventHandler
AnimationWindow
create commandselect request
Object
PhysicsEngine
EditorChecker
Audio Request
(warning request)
Audio Request
(speaking of user)
OSGWindows
Audio
TimeLine
OSGEventHandler
Grid
Frame
Camera update request
FileHandler
Camera
Command
Camera
CommandFile
CommandView
CommandHelp
CommandAnimate
CommandFrame
CommandRightClickMenu
CommandSelect
CommandDeselect
GUIEventHandler
CommandInsertAudio
CommandRotate
CommandScale
execute()
CommandTimeline
Click
CommandTranslate
CommandDelete
CommandCut
CommandRedo
CommandUndo
CommandPaste
CommandInsertObject
CommandFrame
CommandInsertText
Configure objects in Frame
Right Click Menu Request
File Operations Request
createWorld
createSpace

Figure 4. 9

A. APPENDIX

A.1 Enumeration Types

```
Enum WindowName{
     TOP_VIEW,
     SIDE_VIEW,
     FRONT_VIEW,
     3D_VIEW
}
Enum DrawMode{
     SELECT
     DRAW_VEHICLE,
     DRAW_PATH,
     DRAW_WHEEL,
     DRAW_ROAD,
     DRAW_ROAD_BRANCH,
     DRAW_LANE,
     DRAW_TRAFFIC_SIGN,
     DRAW_TRAFFIC_LIGHT,
     DRAW_TREE,
     DRAW_HOUSE
}

Enum ObjectType{
     VEHICLE,
     WHEEL
     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 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,
        ILERIDE_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
}
```

```
Enum CameraMotion{
     ZOOM_IN,
     ZOOM_OUT,
     PITCH_UP,
     PITCH_DOWN,
     ROW_LEFT,
     ROW_RIGHT
}
```

A.2 DTD Schema

```
<?xml encoding="ISO-8859-1"?
     <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
     <!DOCTYPE TrafficCase [
     <!ELEMENT TrafficCase (Background, Frames)>
     <!ELEMENT Background (Roads)>
     <!ELEMENT Roads (Road+))>
     <!ELEMENT Road (Position, Branches+, Lights*,Signs*)>
     <!ATTLIST Road type CDATA #REQUIRED>
     <!ATTLIST Road objectID CDATA #REQUIRED>
     <!ELEMENT Position>
     <!ATTLIST Position currentX CDATA #REQUIRED>
     <!ATTLIST Position currentY CDATA #REQUIRED>
     <!ATTLIST Position currentZ CDATA #REQUIRED>
     <!ATTLIST Position transformationMatrix CDATA #REQUIRED>
     <!ELEMENT Branches(Branch+))>
     <!ELEMENT Branch (Lanes)>
     <!ATTLIST Branch objectID CDATA #REQUIRED>
     <!ATTLIST Branch direction CDATA #REQUIRED >
     <!ATTLIST Branch type CDATA #REQUIRED>
     <!ELEMENT Lanes(Lane+)>
     <!ELEMENT Lane>
     <!ATTLIST Lane leftLine>
     <!ATTLIST Lane rightLine>
     <!ATTLIST Lane direction>
     <!ELEMENT Lights (Light+))>
```

```
<!ELEMENT Light (Position)>
<!ATTLIST Light type CDATA #REQUIRED>
<!ATTLIST Light objectID CDATA #REQUIRED>
<!ELEMENT Signs (Sign+))>
<!ELEMENT Sign (Position)>
<!ATTLIST Sign type CDATA #REQUIRED>
<!ATTLIST Sign objectID CDATA #REQUIRED>
<!ELEMENT Frames (Frame+)>
<!ELEMENT Frame (Objects, Text, Audio)>
<!ATTLIST Frame frameID CDATA #REQUIRED>
<!ELEMENT Objects (Vehicles, LightUpdates)>
<!ELEMENT Vehicles (Vehicle*)>
<!ELEMENT Vehicle (Position)>
<!ATTLIST Vehicle objectID CDATA #REQUIRED>
<!ATTLIST Vehicle type CDATA #REQUIRED>
<!ATTLIST Vehicle path CDATA #REQUIRED>
<!ELEMENT LightUpdates (LightUpdate *)>
<!ELEMENT LightUpdate>
<!ATTLIST LightUpdate objectID CDATA #REQUIRED>
<!ATTLIST LightUpdate color CDATA #REQUIRED>
<!ATTLIST LightUpdate arrowColor CDATA #IMPLIED >
<!ELEMENT Text (#PCDATA)>
<!ELEMENT Audio (#PCDATA)>
]>
```

The hierarchy of tags in the XML output files of TraffEdu can be seen below:

```
<TrafficCase>
    <Background>
        <Roads>
            <Road type objectID>
                <Position currentX currentY currentZ
                    transformationMatrix/>
                <Branches>
                    <Branch objectID direction type>
                        <Lanes>
                            <Lane leftLine rightLine
                                direction />
```

```
                                        <Lane>...</Lane>
                                    </Lanes>
                                </Branch>
                                <Branch>...</Branch>
                        </Branches>
                        <Lights>
                                <Light objectID type>
                                        <Position currentX currentY curentZ
                                                transformationMatrix/>
                                </Light>
                                <Light>...</Light>
                        </Lights>
                        <Signs>
                                <Sign objectID type>
                                        <Position currentX currentY curentZ
                                                transformationMatrix/>
                                </Sign>
                                <Sign>... </Sign>
                        </Signs>
                </Road>
                <Road> ... </Road>
        </Roads>
</Background>
<Frames>
        <Frame frameID>
                <Objects>
                        <Vehicles>
                                <Vehicle objectID type path>
                                        <Position currentX currentY currentZ
                                                transformationMatrix/>
                                </Vehicle>
                                <Vehicle>...</Vehicle>
                        </Vehicles>
                        <LightUpdates>
                                <LightUpdate objectID color arrowColor/>

                                <LightUpdate>...</LightUpdate>
```

```
                    </LightUpdates>
                </Objects>
                <Text>  </Text>
                <Audio>  </Audio>
            </Frame>
            <Frame>...</Frame>
        </Frames>
    </TrafficCase>
```

The directory hierarchy of system TraffEdu is as follows:

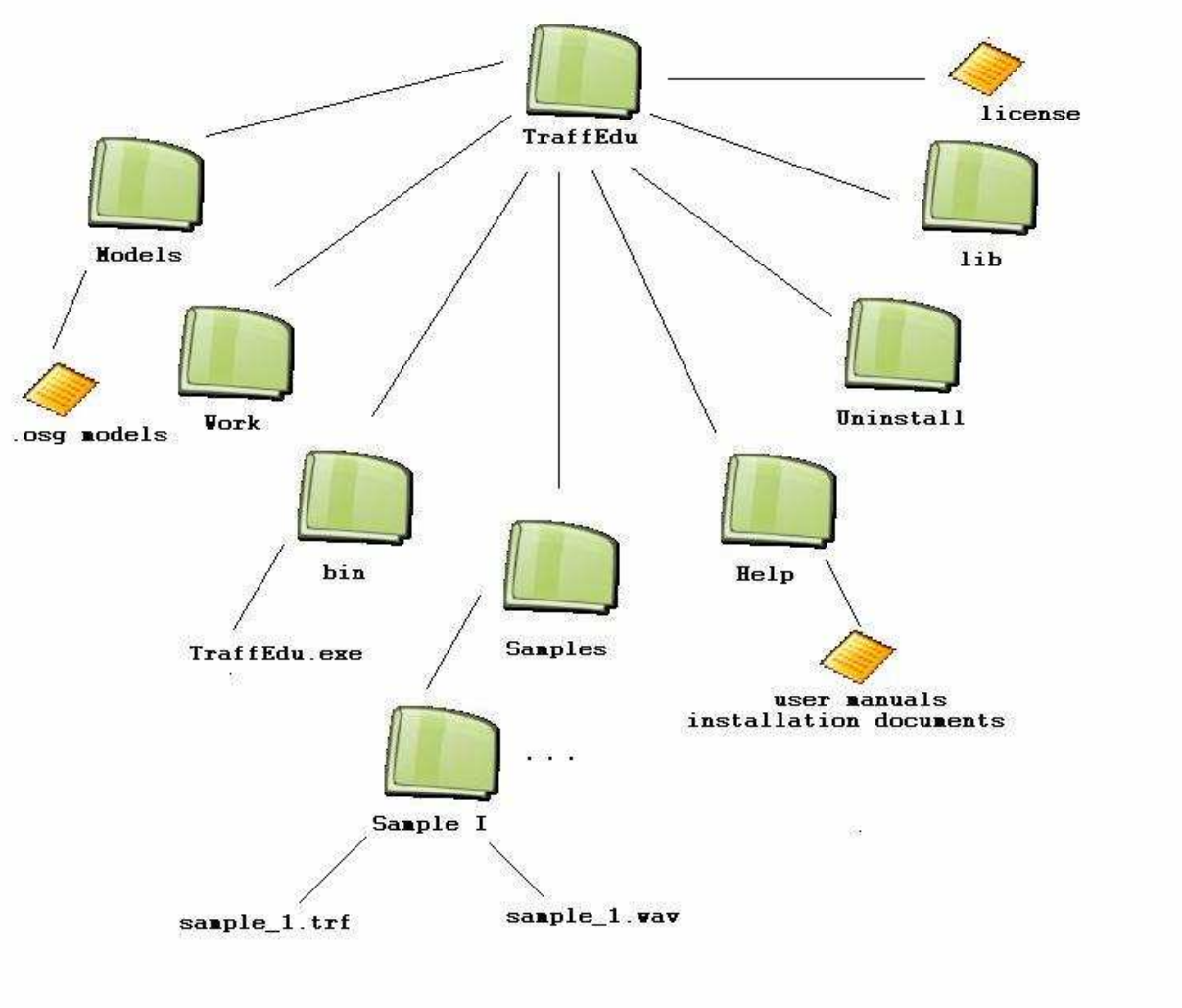


Figure A.1

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`)

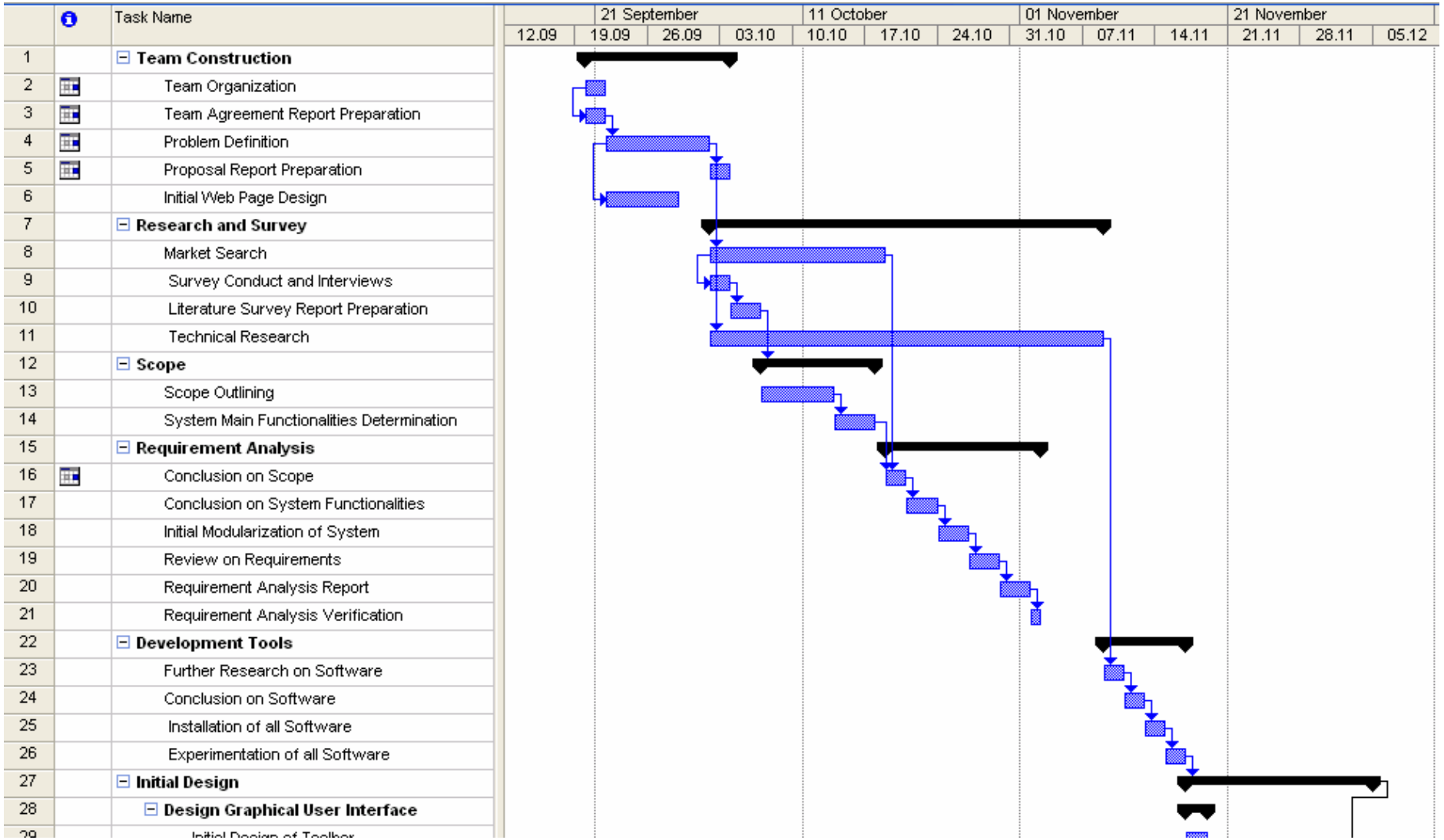
Function names begin with lower case characters and if it is a composite word each new word begins with a capital letter. (e.g. `word1Word2Word3`)

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`)

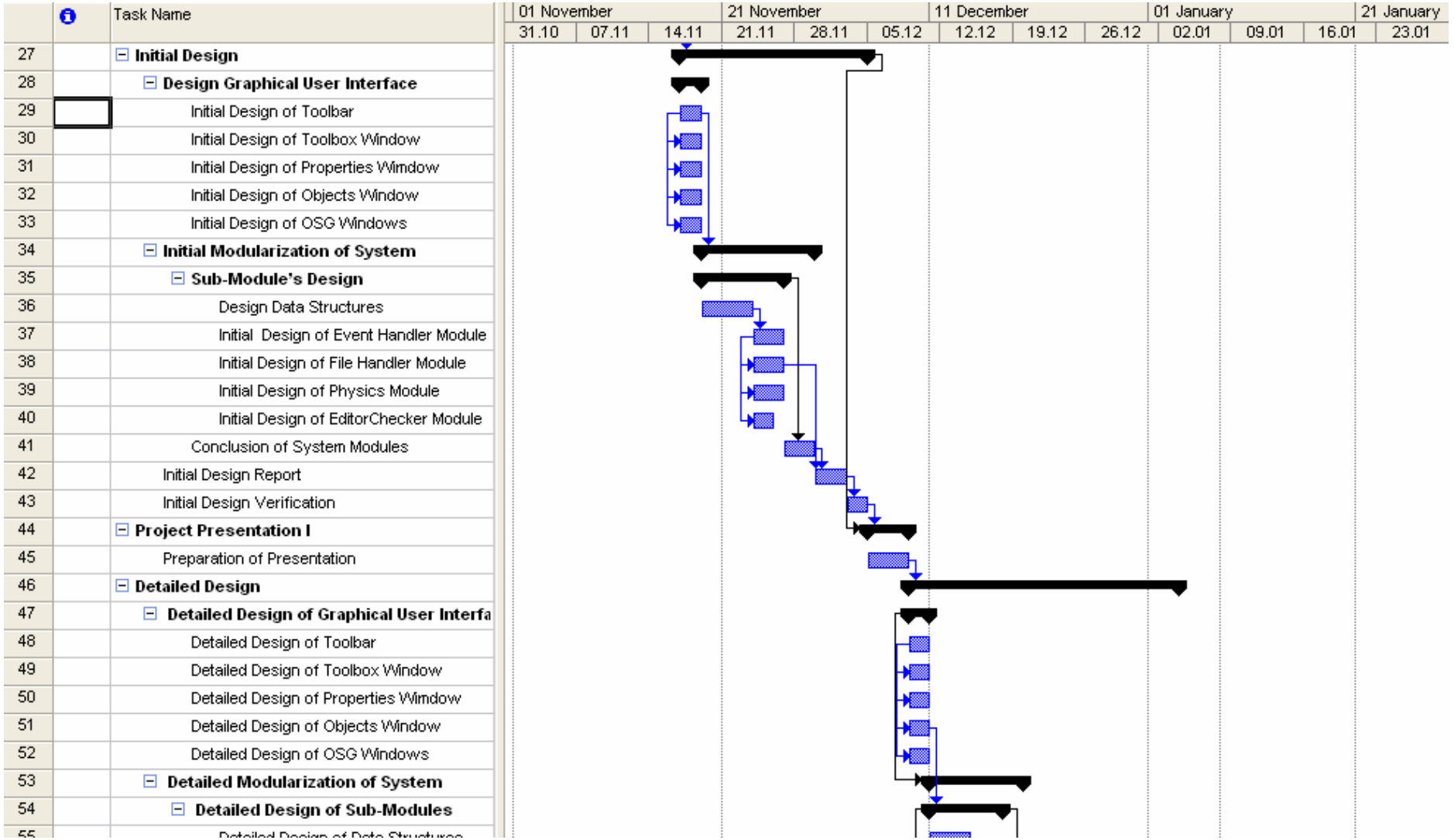
Enumerated types are all written in capital letters.

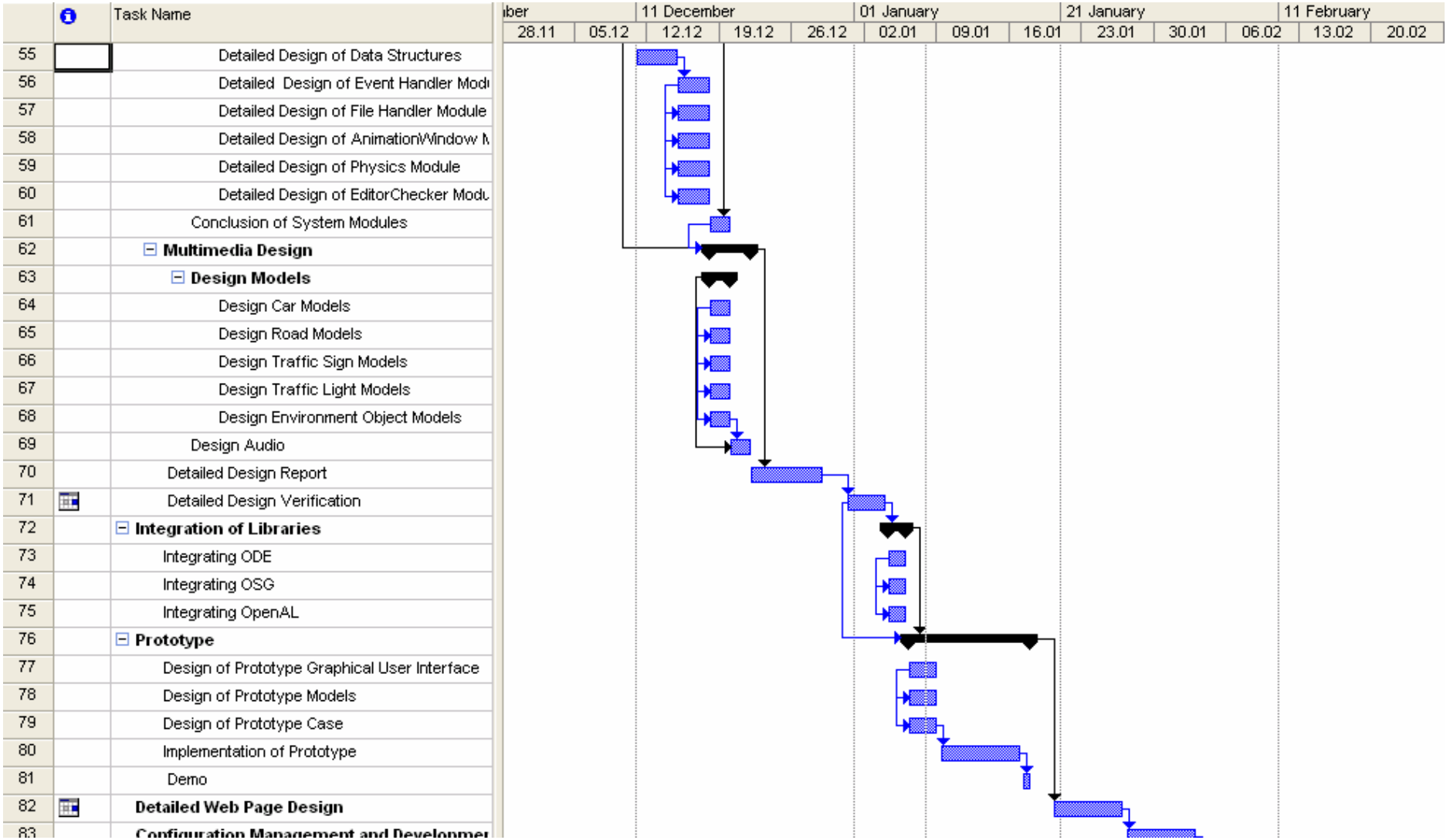
Vector typed variables' are written in plural form.

A.4 Gantt Chart

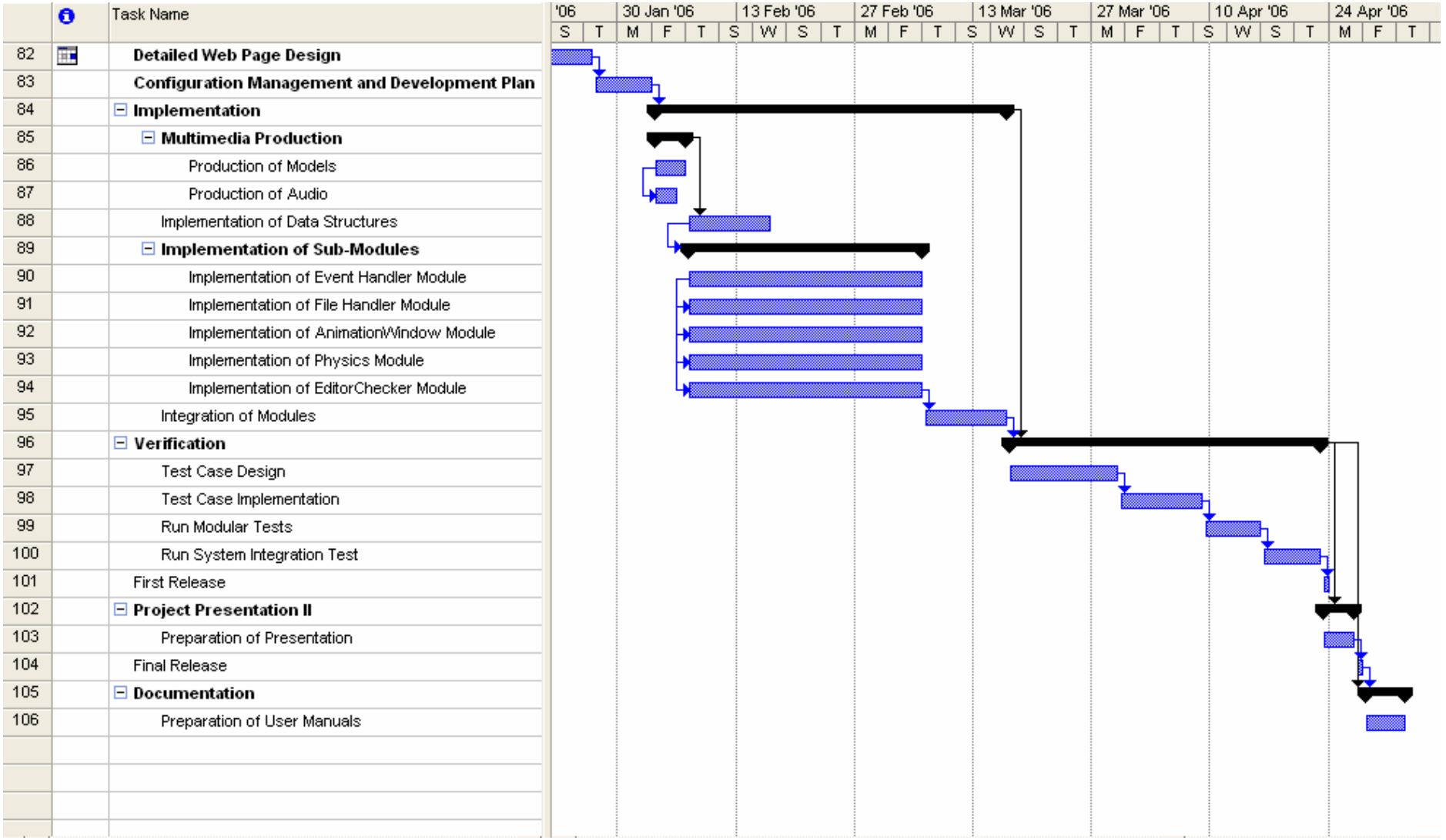


	Task Name
1	**Team Construction**
2	Team Organization
3	Team Agreement Report Preparation
4	Problem Definition
5	Proposal Report Preparation
6	Initial Web Page Design
7	**Research and Survey**
8	Market Search
9	Survey Conduct and Interviews
10	Literature Survey Report Preparation
11	Technical Research
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**


Task Name
01 November
21 November
11 December
01 January
21 January
31.10
07.11
14.11
21.11
28.11
05.12
12.12
19.12
26.12
02.01
09.01
16.01
23.01
27 Initial Design
28 Design Graphical User Interface
29 Initial Design of Toolbar
30 Initial Design of Toolbox Window
31 Initial Design of Properties Window
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 Interfa
48 Detailed Design of Toolbar
49 Detailed Design of Toolbox Window
50 Detailed Design of Properties Window
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
55	Detailed Design of Data Structures
56	Detailed Design of Event Handler Mod
57	Detailed Design of File Handler Module
58	Detailed Design of AnimationWindow M
59	Detailed Design of Physics Module
60	Detailed Design of EditorChecker Modu
61	Conclusion of System Modules
62	**Multimedia Design**
63	**Design Models**
64	Design Car Models
65	Design Road Models
66	Design Traffic Sign Models
67	Design Traffic Light Models
68	Design Environment Object Models
69	Design Audio
70	Detailed Design Report
71	Detailed Design Verification
72	**Integration of Libraries**
73	Integrating ODE
74	Integrating OSG
75	Integrating OpenAL
76	**Prototype**
77	Design of Prototype Graphical User Interface
78	Design of Prototype Models
79	Design of Prototype Case
80	Implementation of Prototype
81	Demo
82	**Detailed Web Page Design**
83	**Configuration Management and Developme**


Task Name
'06
30 Jan '06
13 Feb '06
27 Feb '06
13 Mar '06
27 Mar '06
10 Apr '06
24 Apr '06
S T M F T S W S T M F T S W S T M F T S W S T M F T S W S T M F T
82 Detailed Web Page Design
83 Configuration Management and Development Plan
84 Implementation
85 Multimedia Production
86 Production of Models
87 Production of Audio
88 Implementation of Data Structures
89 Implementation of Sub-Modules
90 Implementation of Event Handler Module
91 Implementation of File Handler Module
92 Implementation of AnimationWindow Module
93 Implementation of Physics Module
94 Implementation of EditorChecker Module
95 Integration of Modules
96 Verification
97 Test Case Design
98 Test Case Implementation
99 Run Modular Tests
100 Run System Integration Test
101 First Release
102 Project Presentation II
103 Preparation of Presentation
104 Final Release
105 Documentation
106 Preparation of User Manuals