ONLINE VIRTUAL TEAM COLLABORATION PLATFORM WITH 3D GRAPHICS

-RESCUE-

INITIAL DESIGN REPORT

INCREDIBLES

Salih Ahi

Kamil Nematlı

Abdulkadir Yazıcı

Mustafa Önder

1. Introduction

This report is written by Ceng490 students for an overview of design of the project Rescue. A detailed explanation of the scenario description, data flows, class and state diagrams are given yet they are not final and may be changed in the final design report.

1.1. Project Description

1.1.1. Purpose

The purpose of the project is:

- To simulate real world scenarios in a virtual environment.
- To train squad leaders each specialized on its own area and improve their skills in their areas.
- To improve collaboration between different squad leaders.

1.1.2. Scope

The project will be developed for leaders of squads mentioned below:

- police squad
- fire squad
- bomb defuse squad

In addition, this simulation could be useful for anyone who wants to increase their team collaboration skills.

1.1.3. Objectives

Project will have the following features:

- The scenarios are going to be designed in a realistic manner.
- There will be a facilitator who manages the simulation from the server and can intervene anything in the scenario.
- Facilitator will be able to choose among several different scenarios.
- Users can communicate with facilitator and other users via the voice chat.
- Facilitator can observe everything in the simulation from different view angles.
- Users follow the scenario from the first person view.
- Users will have limited resources and tools. These resources will be both equipments and people.
- The squad leader's subordinates are capable of accomplishing given tasks.

The program will run in two modes: passive and active mode. The features above will be same for both modes, however there will be some differences:

Passive mode:

- Users have restricted interaction with computer.
- Users manage their teams via the facilitator.

Active mode:

- Users have active interaction with computer.
- Users manage their teams using user interfaces on their own computers and via the facilitator.

1.2. Constraints

There is not much constraint given by the company. Also there is not any restriction to project platform or development environment. The constraints are:

- The project must have been finished by June 2008.
- The project must be done by 4 Metu-Ceng senior students.
- The project schedule must be synchronous with the course schedule.
- The trainee user interface must not be complex.
- Trainees' view perspective must be first person view.
- Facilitator must not affect the flow of scenario.

1.3. Scenario

Prologue : The scenario takes place in a large public building (like a big shopping center as Armada) in present time. In the building several minor bombs have exploded and as a result fire has started in some areas. And it has been reported that several unexploded bombs may still exist.

Main goal : Take all of the civilians to a safe area before they get hurt and defuse all of the bombs.

Teams : Firefighters, police squad and bomb defuse squad

Tools & resources :

Firefighters : Water is used as both tool and resource. They have two sources of water one of which is infinite and the other one is finite. Also they have a fire engine.

Police squad : A maul will be used as a tool for breaking the doors. Also a resource will be used for taking out the civilians from the high levels of the building.

Bomb defuse squad : A trained dog will be used as a tool for finding the bombs. Also detonators will be supplied as a resource for deactivating the bombs and exploding the doors.

Subgoals :

Firefighters: Their task is to prevent the fire to spread over the building. Also they must distinguish the fire in specific areas so that police and bomb defuse squads will be able do their task on these areas.

Police squad : Their task is to find and take out the civilians from the scene. Also they must help bomb defuse squad enter the rooms by breaking the doors with their maul.

Bomb defuse squad : Their task is to find and deactivate the bombs. Also they must help police squad enter the rooms by blowing up the doors with their detonator.

Colloboration Analysis:

If a team does not exist;

Fire fighters : The fire will spread over the building. As the result police and bomb defuse squad wil not be able to operate on areas which are under fire, also civilians will be hurt by the fire.

Police squad : Not all the civilians will be found and the found ones will be taken out of the scene in an unorganized way. Also bomb defuse squad may have some problems on entering some rooms since no doors will be broken by the police squad.

Bomb defuse squad : The bombs will not be defused and will damage the building and may damage the civilians in the evidence area. Also the police squad will not be able to enter some rooms since no door will be blowed up by the bomb defuse squad.

If a team isn't collobrating with the others;

Firefighters : Police and bomb defuse squad wil not be able to operate on areas which are under fire. And they may be trapped by the fire. Also civilians will be hurt by the fire.

Police squad : There will be time loss for bomb defuse squad for waiting the door to be broken. Also firefighters may try to distinguish fires in unnecessary places.

Bomb defuse squad : Police squad or civilians under their control may be hurt by bomb defuse squad's detonator. There will be time loss for police squad for waiting the door to be exploded. Also firefighters may try to distinguish fires in unnecessary places.

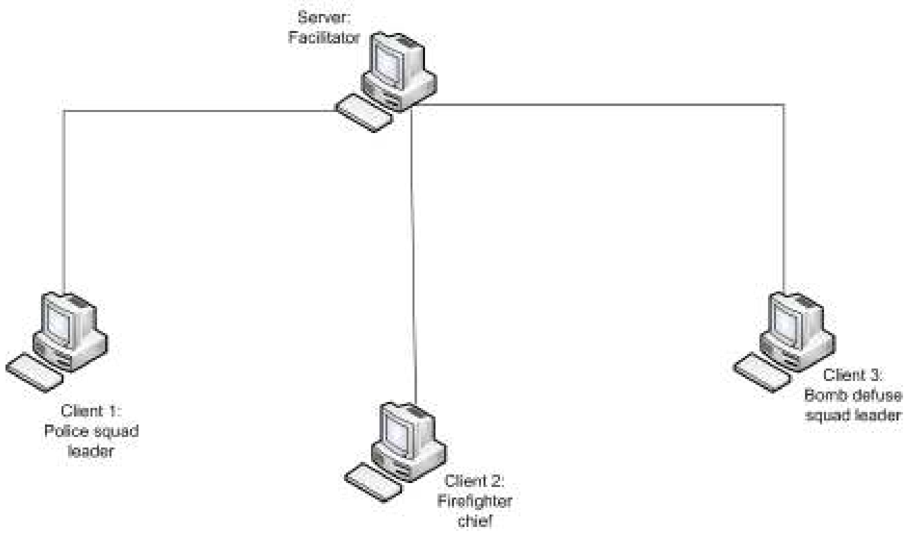
Scenario Specifications:

Bombs defuse squad is able to blow up any door with their detonators, but police squad is able to break only thin doors. Since bomb defuse squad has a limited number of detonators, they are not enough for blowing up all the doors. So they need police squad assistance for passing through the thin doors. Likewise police squad needs bomb defuse squad assistance for passing through the though doors.

2. Modules

2.1 Network Module

The project will be a real time online multi-user simulation. Users will connect to the system via network connection. This requires a good server/clients network architecture. There will be three clients and a server connected to each other. It is simply as follows:



There will be continuous data flow from server to clients and clients to server. This data will be sended and received as network packages. The network packages are converted into message objects. There will be two types of messages: voice packages for voice communication and all other messages. Every message object have tree main fields:

- message id: to determine the type of message,
- user id: to determine who send the message,
- message body: includes the message content.

Server is a user having extra privilages. He starts the session and waits for users to connect server as a client with their user id. After all connections are estaplished system is ready for bidirectional dataflow. Since the system have server-client architecture all the clients' messages are collected in server's message pool. Server processes them one-by-one and sends its own packages to related targets if necessary. Both clients and server have two main threads. First is for sending prepared messages to targets and second is for receiving and processing incoming messages. The simulation will be ended by the server.

2.2 Sound module

Voice chat is essential part of simulation. Users will be able to communicate with other user including facilitator via voice chat. When user wants to talk with someone he presses the button and then microphone start to record the speech. Actually recorded data agglomerates in the buffer, and when buffer becomes full data will be encoded into package before being sent to server. Data packages, received by server, are transferred to appropriate user. When client gets message it decodes the package and speaker will play the speech. There will also be two main threads running one for recording speech and second for decode and playing the received data packages.

2.3 Graphics Module

Graphics has always been one of the most important aspect of a program for the end-user. Since computer users usually don't have a detailed knowledge (and they don't need to actually) about the infrastructure and the architecture of a program, a well designed project may be thought as a useless one because of its hardly managable and complex user interface. Especially in a simulation program, making the user feel as if he is in the simulated environment is a hard task to achieve without decreasing the usability. The more the reality is achieved, the more successful and credible the results of the simulation are. Considering those facts, we try to present an environment with satisfying details level to the user. But this requirement should not overcome the usability of the system since the users are predicted to be trainees which are not so professional in managing complex GUIs.

Such goals bring the requirement of a well design of the graphics module. The general methodology is to define the module as seperate as possible from the main program to debug and test easier. Thus we designed a module, which consists of a single SceneManager. Our approach is basically as follows:

- Load all objects to the scene at the beginning of the simulation.
- As the simulation commences, if there is a change in the object's properties which will affect how it will be rendered, that change is reported to SceneManager.
- SceneManager receives the changes, recalculates, and draws to the screen.

2.4 Engine Module

The engine is the core of the simulation. It basically is the boss which manages all other modules, establishes the connection and controls the dataflow between other parts of the system. The engine also applies game logic rules, such as spreading of fire, exhaustion of resources, etc. Starting and terminating the triggers is another task done by the engine module. All map, character and environment object data is stored and their related properties are transferred to related modules by the engine. The engine is composed of several classes.

2.5 AI Module

AI is the brain of a system. The better the AI of a program, the better it can simulate human actions and present a more realistic system. In our project, since we are not advanced in aspects of developing a complex AI, simple tasks like pathfinding and decision making will be done by this module. For these two subtasks, two classes will be implemented as described in the following sections. This module contains three classes which are PathFinder, BehaviourDecider and a ScriptEngine powered by Python.

2.6 Physics Module

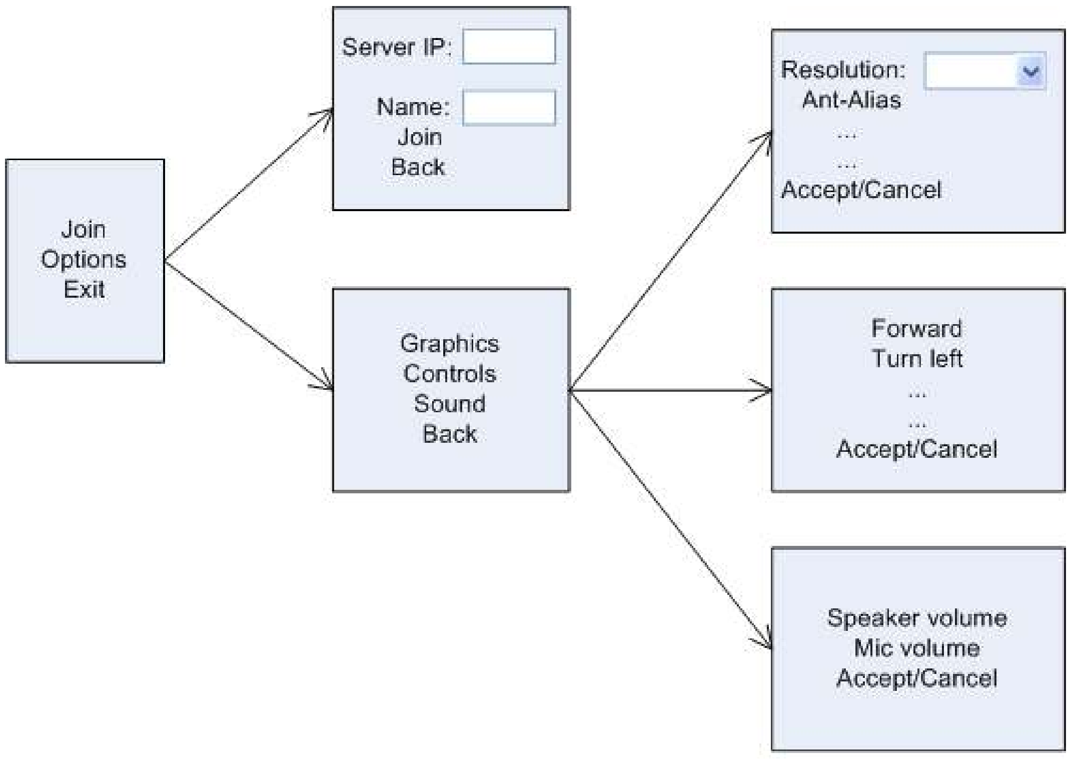
With just stunning graphics, fast network protocols and a genious AI, a simulation may be complete but the objects will fall when placed in the air! This is just one task the physics module should overcome. The system we are planning to develop will be able to calculate basic physical properties of enviromental objects and characters.

3 User Interface

The user interface, an mentioned above, is designed to be simple and useful for the trainees to navigate easily. Two seperate menus are designed for client and server.

3.1 Client Menu

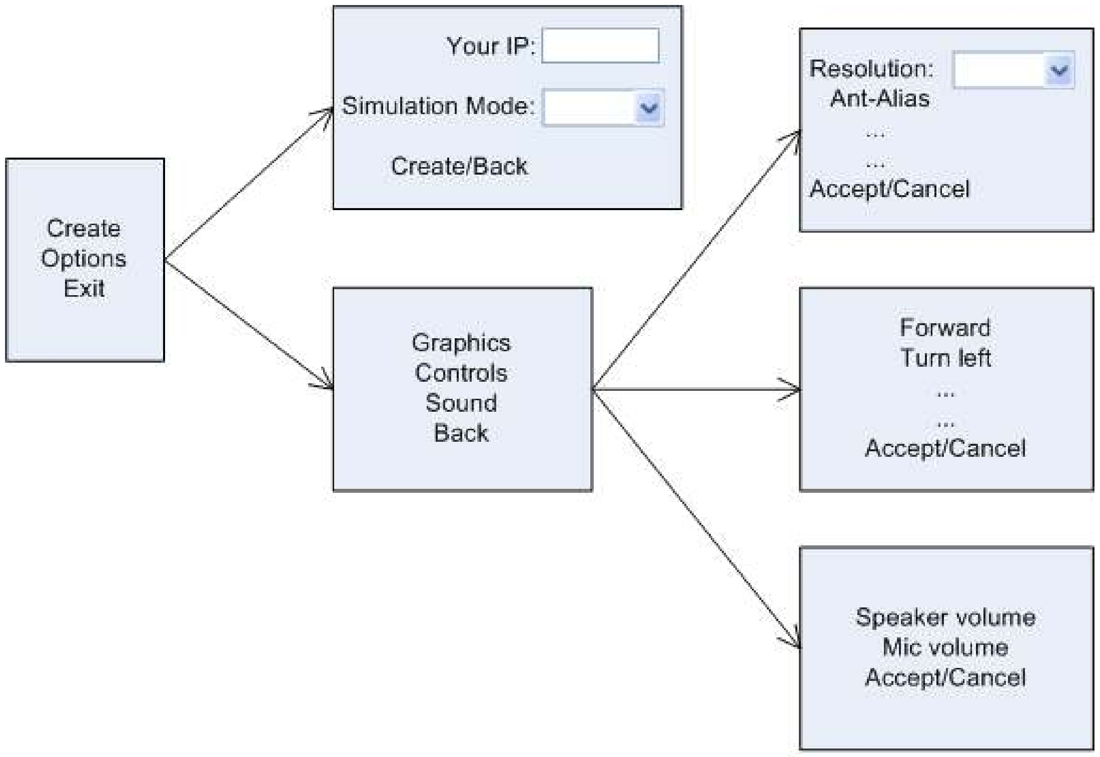
- From the main menu, user can connect to a server, adjust simulation settings or exit the system.
- In the join menu, after typing the server's IP address and specifying a name for the user, the simulation begins.
- From the options menu, user can change graphics settings (e.g. resolution, texture details..), control settings (e.g redefine movement keys, camera control keys,..), or adjust volume settings.



Client Menu

3.2 Server Menu

- In the main menu, facilitator can host a new session, adjust game settings or exit the system.
- From the create menu, IP address will be shown for the clients to join and the facilitator will choose whether the simulation will run on active or passive mode.
- From the options menu, user can change graphics settings (e.g. resolution, texture details..), control settings (e.g redefine movement keys, camera control keys,..), or adjust volume settings.

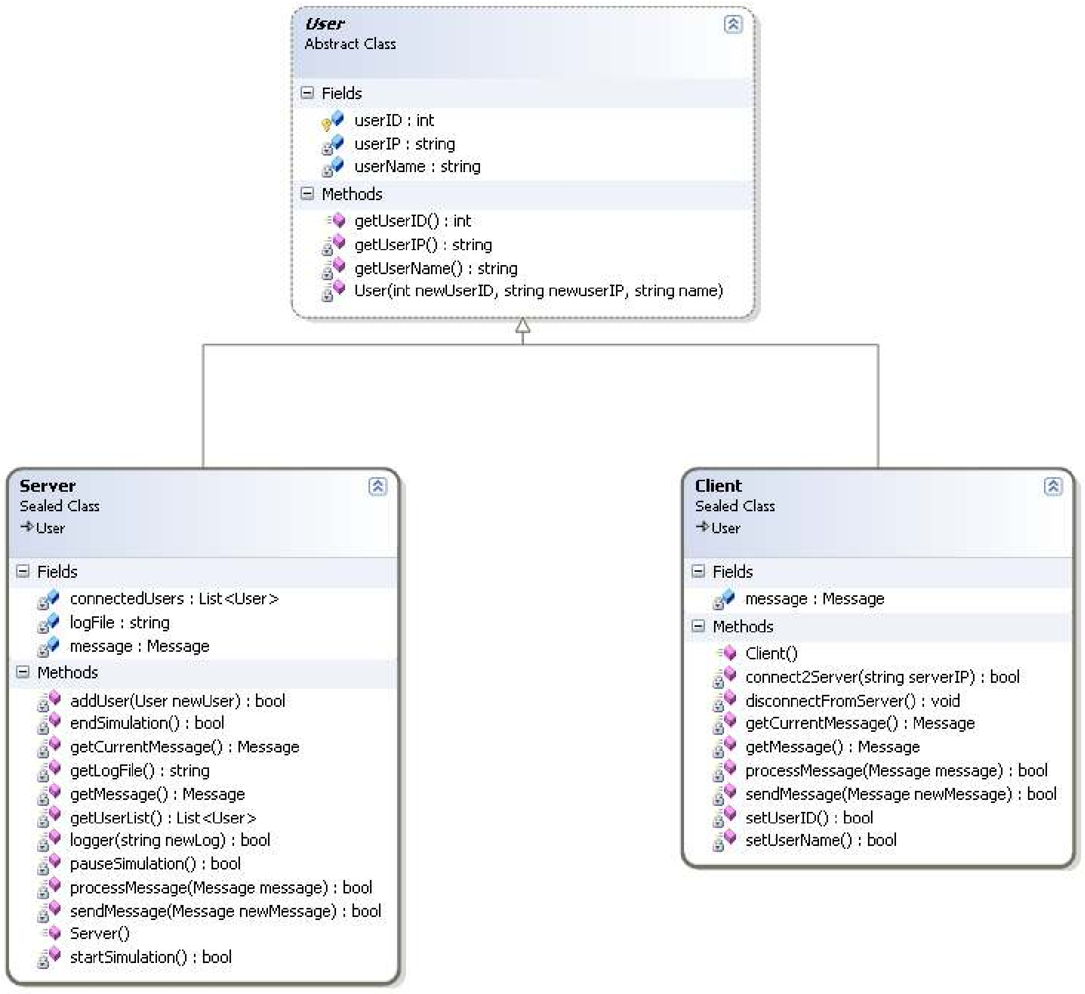


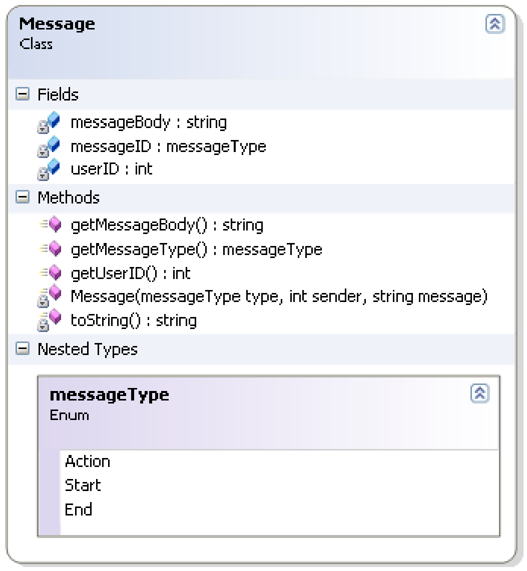
Server Menu

4 Class Definitions

4.1 Network

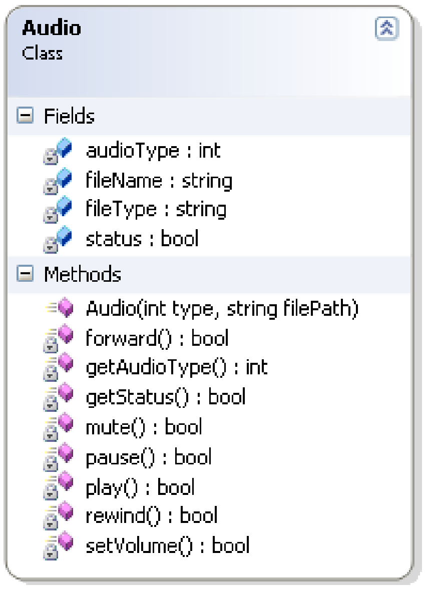
Network module consists of three classes. Two classes for client and server network modules which inherit the user class and one common Message class. Message types are to be defined in detail in the final report. We will use XML for message bodies which will make handling messages across the network.


User
Abstract Class
Fields
userID : int
userIP : string
userName : string
Methods
getUserID() : int
getUserIP() : string
getUserName() : string
User(int newUserID, string newuserIP, string name)
Server
Sealed Class
User
Fields
connectedUsers : List<User>
logFile : string
message : Message
Methods
addUser(User newUser) : bool
endSimulation() : bool
getCurrentMessage() : Message
getLogFile() : string
getMessage() : Message
getUserList() : List<User>
logger(string newLog) : bool
pauseSimulation() : bool
processMessage(Message message) : bool
sendMessage(Message newMessage) : bool
Server()
startSimulation() : bool
Client
Sealed Class
User
Fields
message : Message
Methods
Client()
connect2Server(string serverIP) : bool
disconnectFromServer() : void
getCurrentMessage() : Message
getMessage() : Message
processMessage(Message message) : bool
sendMessage(Message newMessage) : bool
setUserID() : bool
setUserName() : bool


Message
Class
Fields
messageBody : string
messageID : messageType
userID : int
Methods
getMessageBody() : string
getMessageType() : messageType
getUserID() : int
Message(messageType type, int sender, string message)
toString() : string
Nested Types
messageType
Enum
Action
Start
End

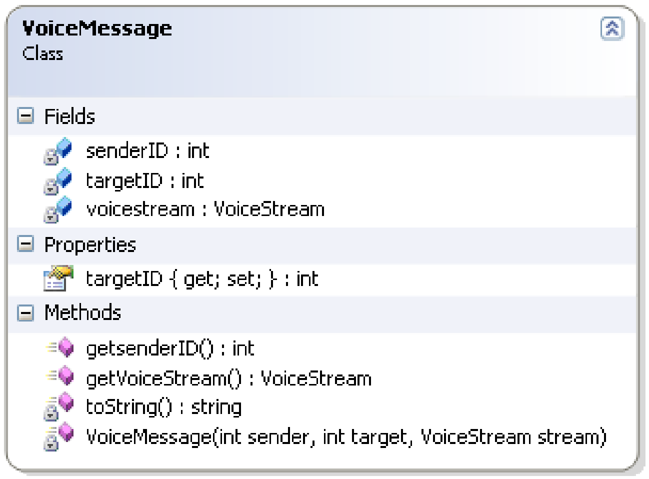
4.2 Sound

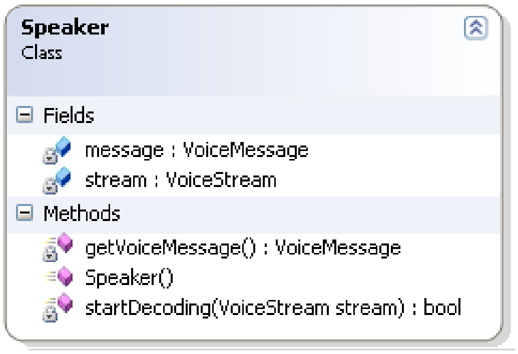
This class is used for playing environment sound, menu music and any audio files throughout the simulation.



4.3 Voice

VoiceMessage is separated from normal Message class since it has a different structure. Speaker and Microphone classes are used for encoding and decoding audio streams.

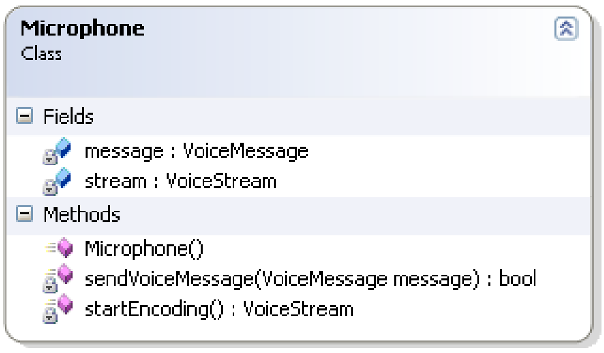

VoiceMessage
Class
Fields
senderID : int
targetID : int
voicestream : VoiceStream
Properties
targetID { get; set; } : int
Methods
getsenderID() : int
getVoiceStream() : VoiceStream
toString() : string
VoiceMessage(int sender, int target, VoiceStream stream)



Speaker
Class

Fields
- message : VoiceMessage
- stream : VoiceStream

Methods
- getVoiceMessage() : VoiceMessage
- Speaker()
- startDecoding(VoiceStream stream) : bool



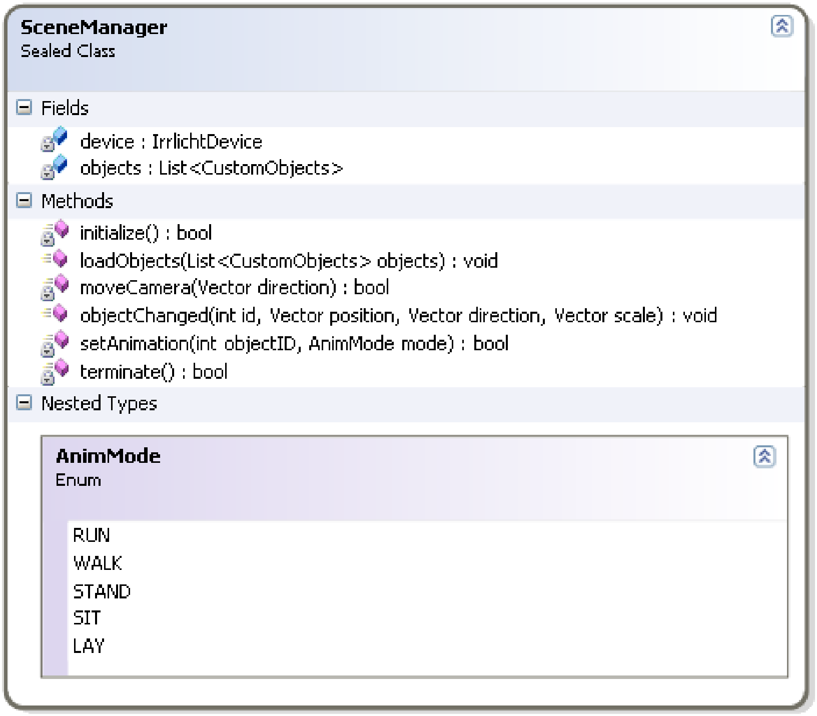
Microphone
Class

Fields
- message : VoiceMessage
- stream : VoiceStream

Methods
- Microphone()
- sendVoiceMessage(VoiceMessage message) : bool
- startEncoding() : VoiceStream

4.4 Graphics

This module consists of a single SceneManager class. Objects are loaded into the class and changes are reported in as they occur. Rendering details are handled inside the class.



SceneManager
Sealed Class

Fields
- device : IrrlichtDevice
- objects : List<CustomObjects>

Methods
- initialize() : bool
- loadObjects(List<CustomObjects> objects) : void
- moveCamera(Vector direction) : bool
- objectChanged(int id, Vector position, Vector direction, Vector scale) : void
- setAnimation(int objectID, AnimMode mode) : bool
- terminate() : bool

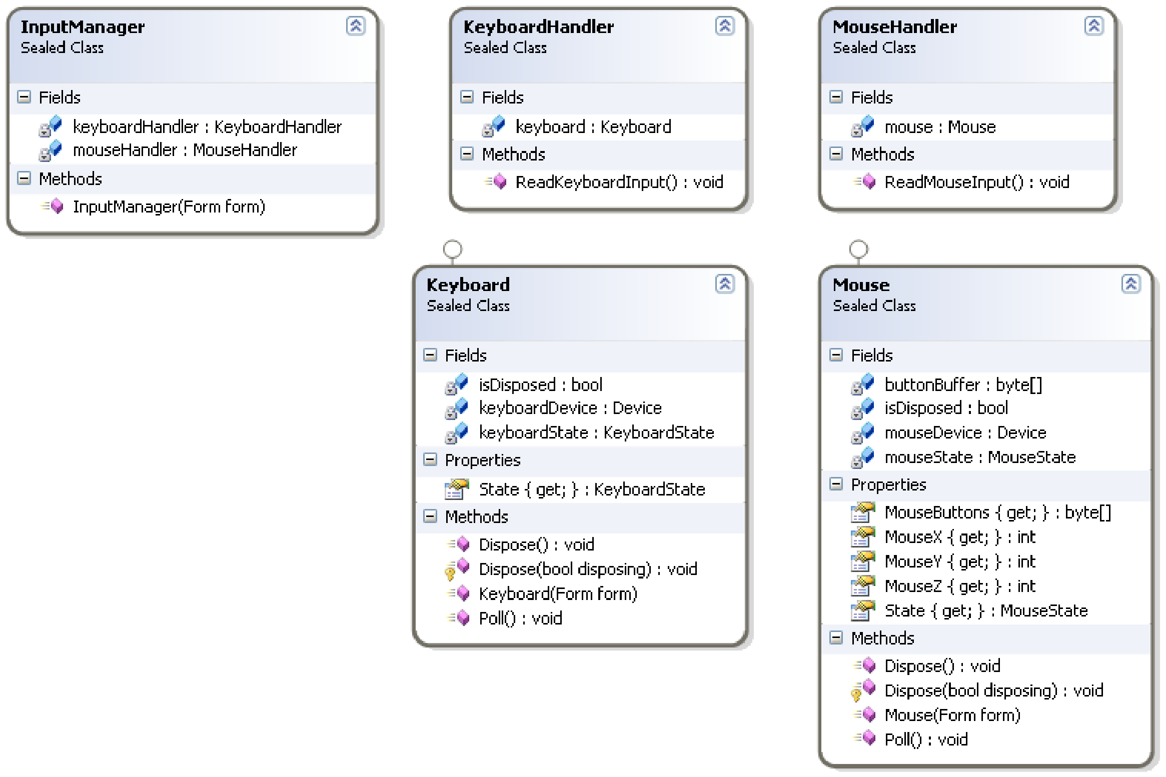
Nested Types

AnimMode
Enum

- RUN
- WALK
- STAND
- SIT
- LAY

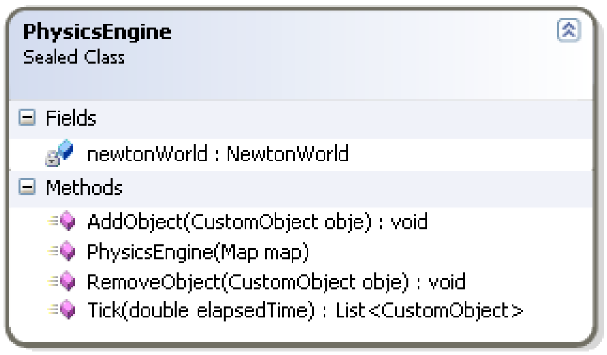
4.5 Input

InputManager class handles keyboard and mouse events with its KeyboardHandler and MouseHandler members. These handlers access Keyboard and Mouse classes which have event polls and translate events to InputManager.



4.6 Physics

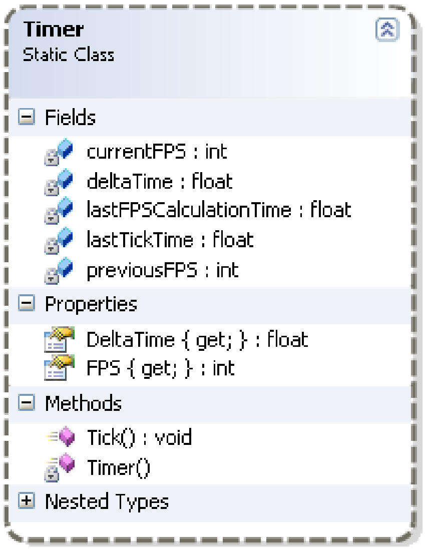
This module consists of a single Physics class. Physics class handles movement and interaction of objects with its newtonWorld


PhysicsEngine
Sealed Class
Fields
newtonWorld : NewtonWorld
Methods
AddObject(CustomObject obje) : void
PhysicsEngine(Map map)
RemoveObject(CustomObject obje) : void
Tick(double elapsedTime) : List<CustomObject>

4.7 Engine

4.7.1 Timer

This class helps control game speed of the simulation.

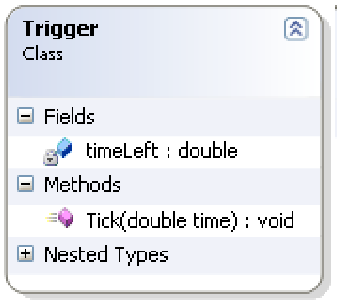


Timer
Static Class

Fields
currentFPS : int
deltaTime : float
lastFPSCalculationTime : float
lastTickTime : float
previousFPS : int
Properties
DeltaTime { get; } : float
FPS { get; } : int
Methods
Tick() : void
Timer()
Nested Types

4.7.2 Trigger

This is the class for triggering events during the simulation.



Trigger
Class

Fields
timeLeft : double
Methods
Tick(double time) : void
Nested Types

4.7.3 BaseSquad

This is the main squad class. All other squads inherit this class.

4.7.4 PoliceSquad

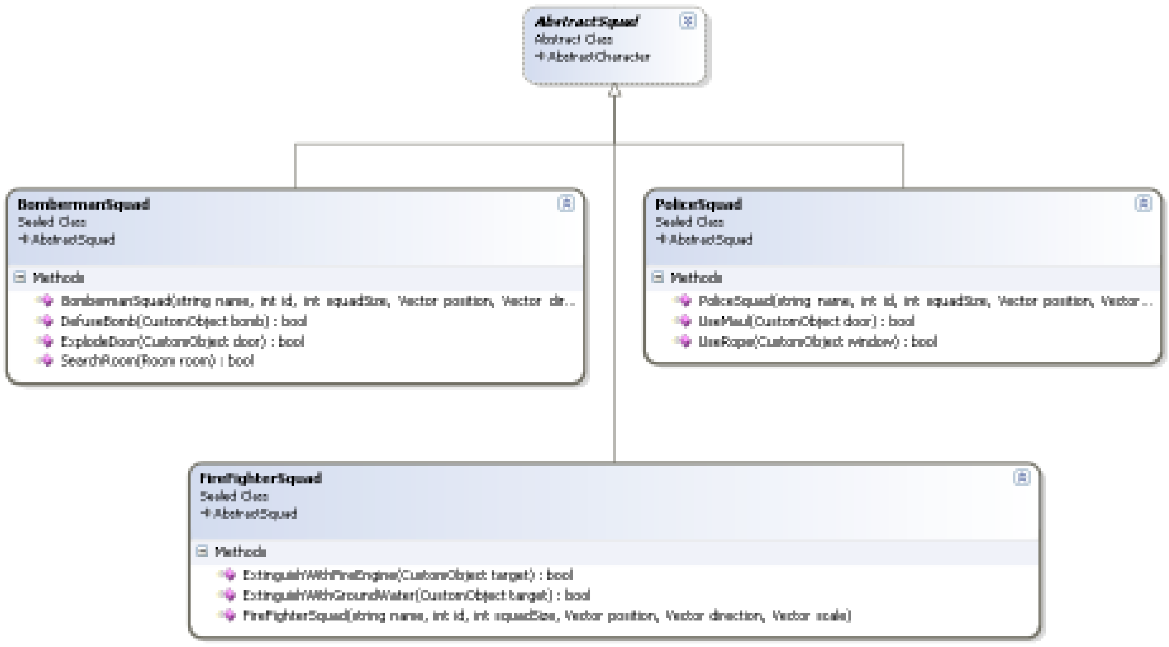
This is the police team class. It composes of many PoliceOfficer classes and controls them according to the orders given by PoliceLeader

4.7.5 BombermanSquad

This is the bomb defuse team. It composes of many Bomberman classes and controls them according to the orders given by BombermanLeader

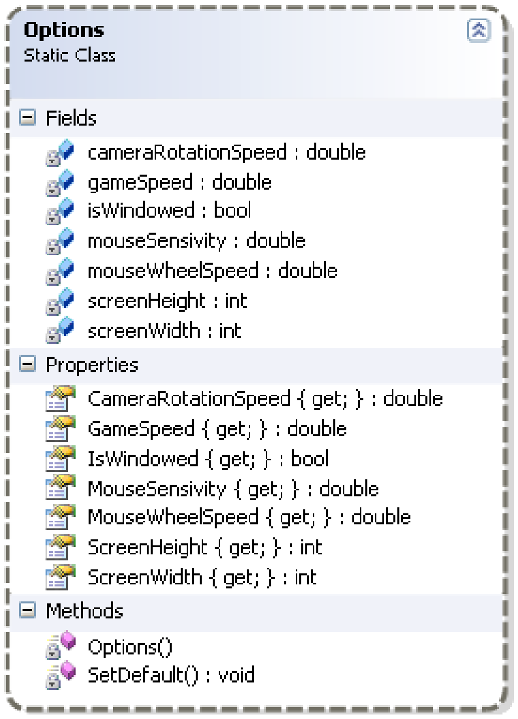
4.7.6 FirefighterSquad

This is the firefighter team. It composes of many Firefighter classes and controls them according to the orders given by FirefighterLeader


AbstractSquad
Abstract Class
AbstractCharacter
BombermanSquad
Sealed Class
AbstractSquad
Methods
BombermanSquad(string name, int id, int squadSize, Vector position, Vector dir...
DefuseBomb(CustomObject bomb) : bool
ExplodeDoor(CustomObject door) : bool
SearchRoom(Room room) : bool
PoliceSquad
Sealed Class
AbstractSquad
Methods
PoliceSquad(string name, int id, int squadSize, Vector position, Vector ...
UseMaul(CustomObject door) : bool
UseRope(CustomObject window) : bool
FireFighterSquad
Sealed Class
AbstractSquad
Methods
ExtinguishWithFireEngine(CustomObject target) : bool
ExtinguishWithGroundWater(CustomObject target) : bool
FireFighterSquad(string name, int id, int squadSize, Vector position, Vector direction, Vector scale)

4.7.7 Options

This is the class that hold the settings about the simulation.



Options
Static Class

Fields
cameraRotationSpeed : double
gameSpeed : double
isWindowed : bool
mouseSensivity : double
mouseWheelSpeed : double
screenHeight : int
screenWidth : int

Properties
CameraRotationSpeed { get; } : double
GameSpeed { get; } : double
IsWindowed { get; } : bool
MouseSensivity { get; } : double
MouseWheelSpeed { get; } : double
ScreenHeight { get; } : int
ScreenWidth { get; } : int

Methods
Options()
SetDefault() : void

4.7.8 Officer

This is the main NPC class. All other NPCs inherit this class.

4.7.9 PoliceOfficer

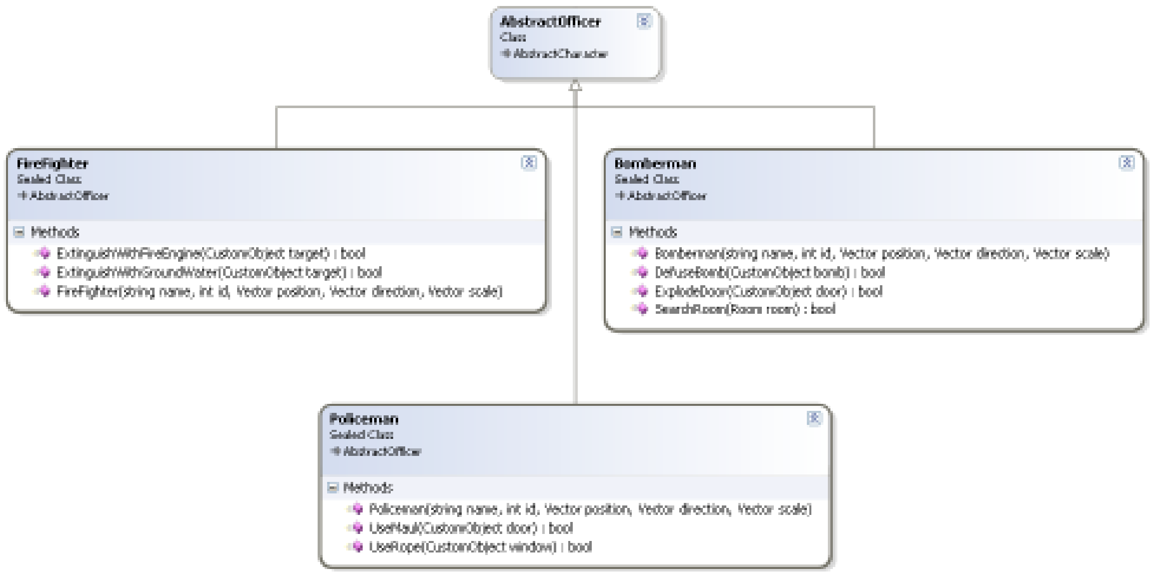
This is the NPC police class. These follow the orders given by the squad.

4.7.10 Bomberman

This is the bomb defuse NPC class. These follow the orders given by the squad.

4.7.11 Firefighter

This is the firefighter NPC class. These follow the orders given by the squad.


AbstractOfficer
Class
AbstractCharacter
FireFighter
Sealed Class
AbstractOfficer
Methods
ExtinguishWithFireEngine(CustomObject target) : bool
ExtinguishWithGroundWater(CustomObject target) : bool
FireFighter(string name, int id, Vector position, Vector direction, Vector scale)
Bomberman
Sealed Class
AbstractOfficer
Methods
Bomberman(string name, int id, Vector position, Vector direction, Vector scale)
DefuseBomb(CustomObject bomb) : bool
ExplodeDoor(CustomObject door) : bool
SearchRoom(Room room) : bool
Policeman
Sealed Class
AbstractOfficer
Methods
Policeman(string name, int id, Vector position, Vector direction, Vector scale)
UseMaul(CustomObject door) : bool
UseRope(CustomObject window) : bool

4.7.12 CustomObject

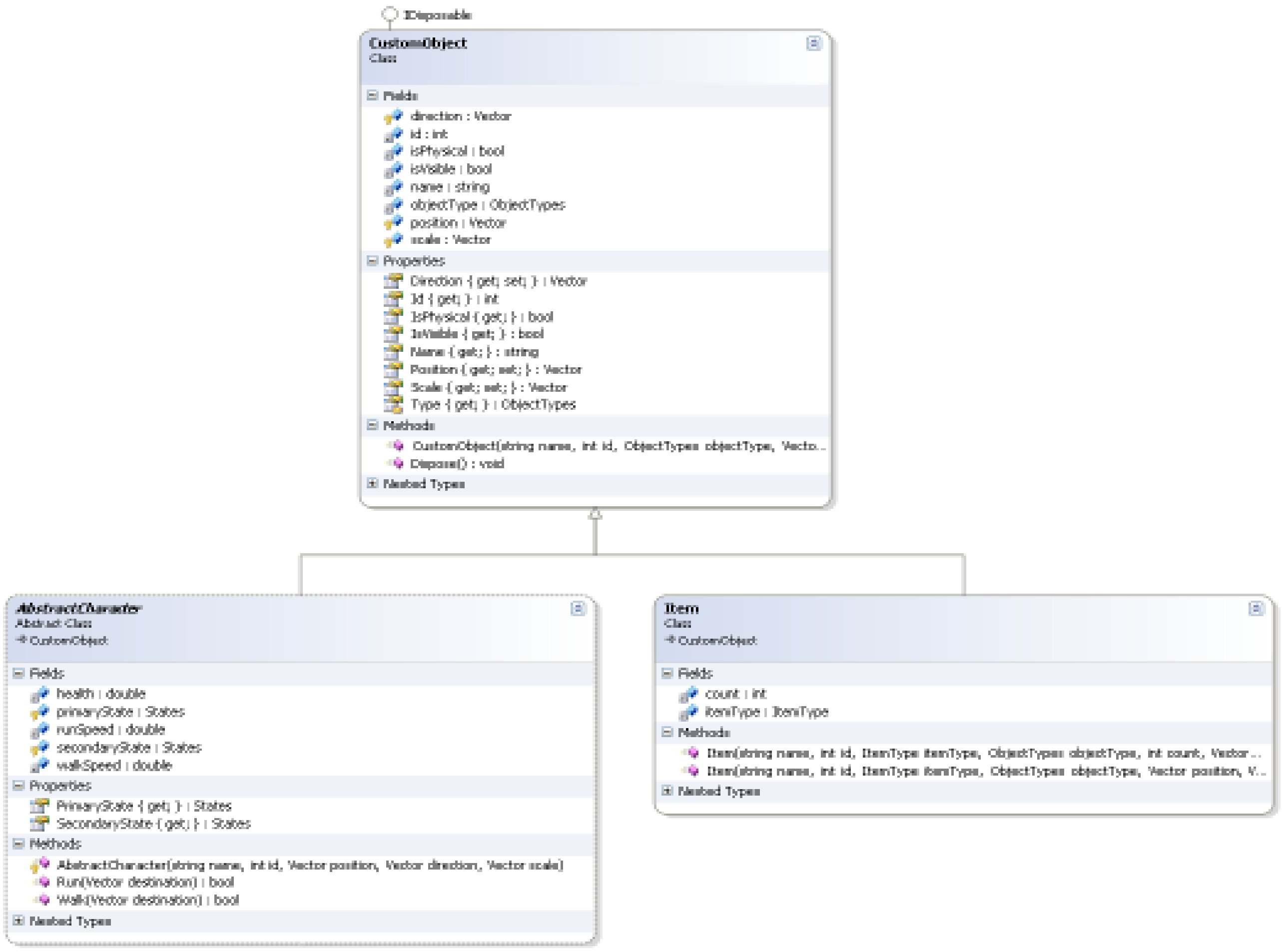
This is the main object class. All other objects inherit this class.

4.7.13 Item

This class is used for environment objects and items carried by players and NPCs like maul, door, etc.

4.7.14 BaseCharacter

This is the main character class. All other characters inherit this class.


IDisposable
CustomObject
Class
Fields
direction : Vector
id : int
isPhysical : bool
isVisible : bool
name : string
objectType : ObjectTypes
position : Vector
scale : Vector
Properties
Direction { get; set; } : Vector
Id { get; } : int
IsPhysical { get; } : bool
IsVisible { get; } : bool
Name { get; } : string
Position { get; set; } : Vector
Scale { get; set; } : Vector
Type { get; } : ObjectTypes
Methods
CustomObject(string name, int id, ObjectTypes objectType, Vecto...
Dispose() : void
Nested Types
AbstractCharacter
Abstract Class
CustomObject
Fields
health : double
primaryState : States
runSpeed : double
secondaryState : States
walkSpeed : double
Properties
PrimaryState { get; } : States
SecondaryState { get; } : States
Methods
AbstractCharacter(string name, int id, Vector position, Vector direction, Vector scale)
Run(Vector destination) : bool
Walk(Vector destination) : bool
Nested Types
Item
Class
CustomObject
Fields
count : int
itemType : ItemType
Methods
Item(string name, int id, ItemType itemType, ObjectTypes objectType, int count, Vector...
Item(string name, int id, ItemType itemType, ObjectTypes objectType, Vector position, V...
Nested Types

4.7.15 Map

This is the class holds other environment data. It is divided into sub regions.

4.7.16 Region

This is the class for a specific area of the map. Regions may be buildings, open areas, etc.

4.7.17 Building

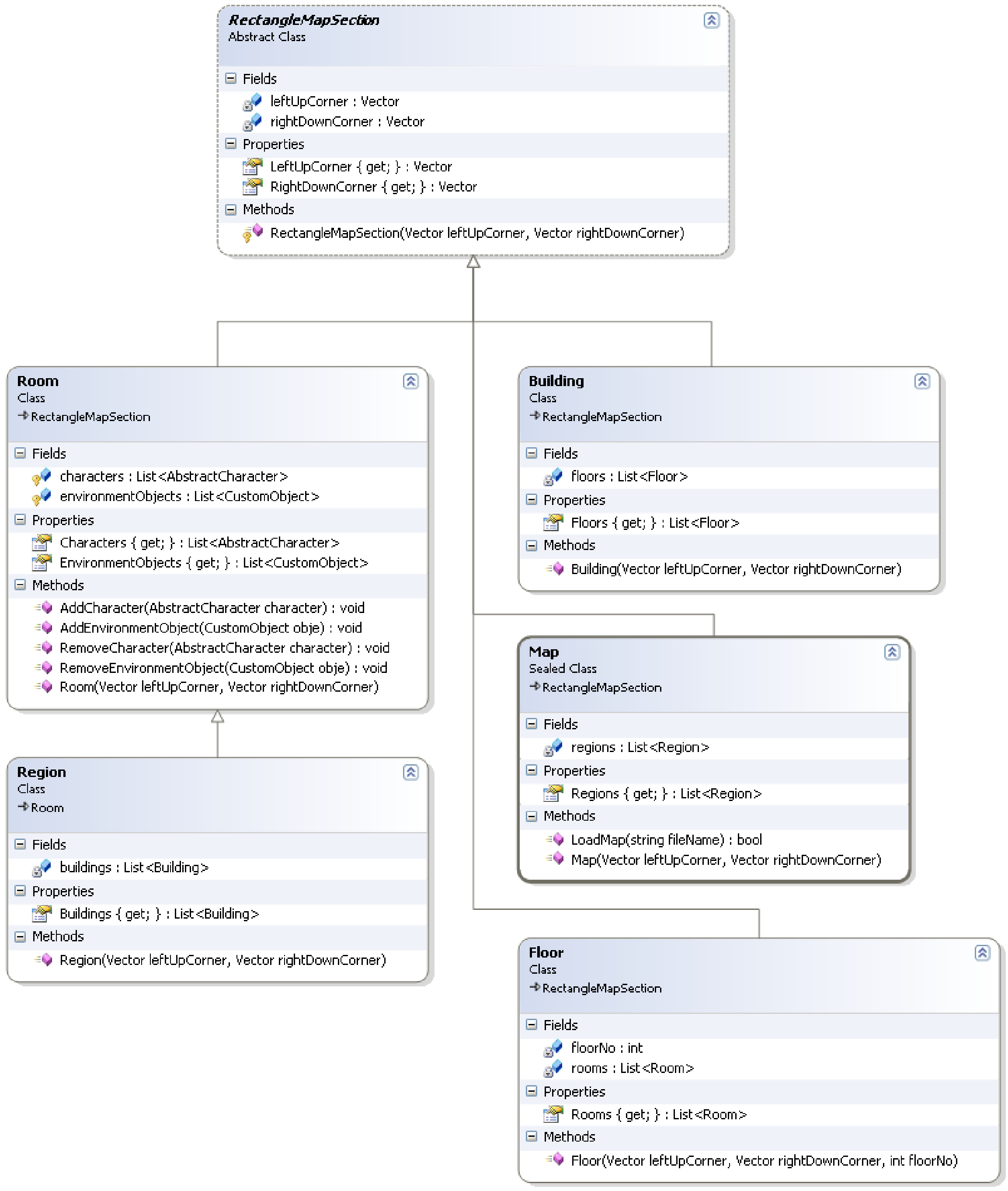
This is the class for buildings in a region. It is divided into floors.

4.7.18 Floor

This is the class for a floor in a building. It is divided into rooms

4.7.19 Room

This is the class for rooms in a floor of a buildings.


RectangleMapSection
Abstract Class
Fields
leftUpCorner : Vector
rightDownCorner : Vector
Properties
LeftUpCorner { get; } : Vector
RightDownCorner { get; } : Vector
Methods
RectangleMapSection(Vector leftUpCorner, Vector rightDownCorner)
Room
Class
RectangleMapSection
Fields
characters : List<AbstractCharacter>
environmentObjects : List<CustomObject>
Properties
Characters { get; } : List<AbstractCharacter>
EnvironmentObjects { get; } : List<CustomObject>
Methods
AddCharacter(AbstractCharacter character) : void
AddEnvironmentObject(CustomObject obje) : void
RemoveCharacter(AbstractCharacter character) : void
RemoveEnvironmentObject(CustomObject obje) : void
Room(Vector leftUpCorner, Vector rightDownCorner)
Building
Class
RectangleMapSection
Fields
floors : List<Floor>
Properties
Floors { get; } : List<Floor>
Methods
Building(Vector leftUpCorner, Vector rightDownCorner)
Map
Sealed Class
RectangleMapSection
Fields
regions : List<Region>
Properties
Regions { get; } : List<Region>
Methods
LoadMap(string fileName) : bool
Map(Vector leftUpCorner, Vector rightDownCorner)
Region
Class
Room
Fields
buildings : List<Building>
Properties
Buildings { get; } : List<Building>
Methods
Region(Vector leftUpCorner, Vector rightDownCorner)
Floor
Class
RectangleMapSection
Fields
floorNo : int
rooms : List<Room>
Properties
Rooms { get; } : List<Room>
Methods
Floor(Vector leftUpCorner, Vector rightDownCorner, int floorNo)

4.7.20 Leader

This is the main player class. All leaders inherit this class.

4.7.21 PoliceLeader

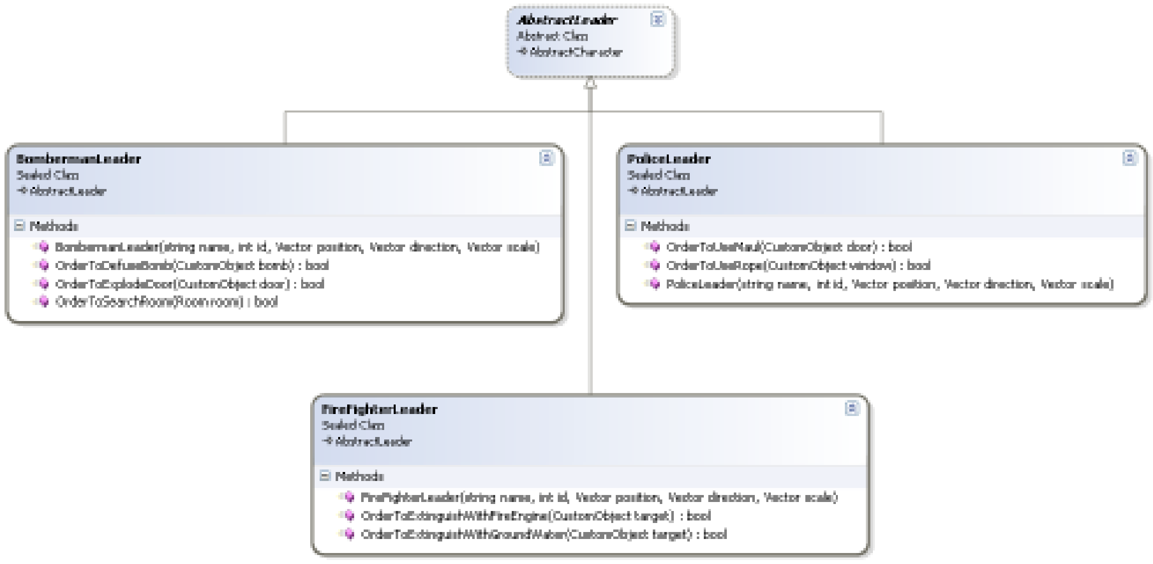
This is the police player's class. It has a squad member and can give orders through this squad class to PoliceOfficers

4.7.22 BombermanLeader

This is the bomb defuse player's class. It has a squad member and can give oerders through this squad class to Bombermans.

4.7.23 FirefighterLeader

This is the firefighter player's class. It has a squad member and can give orders through this squad class to Firefighters.

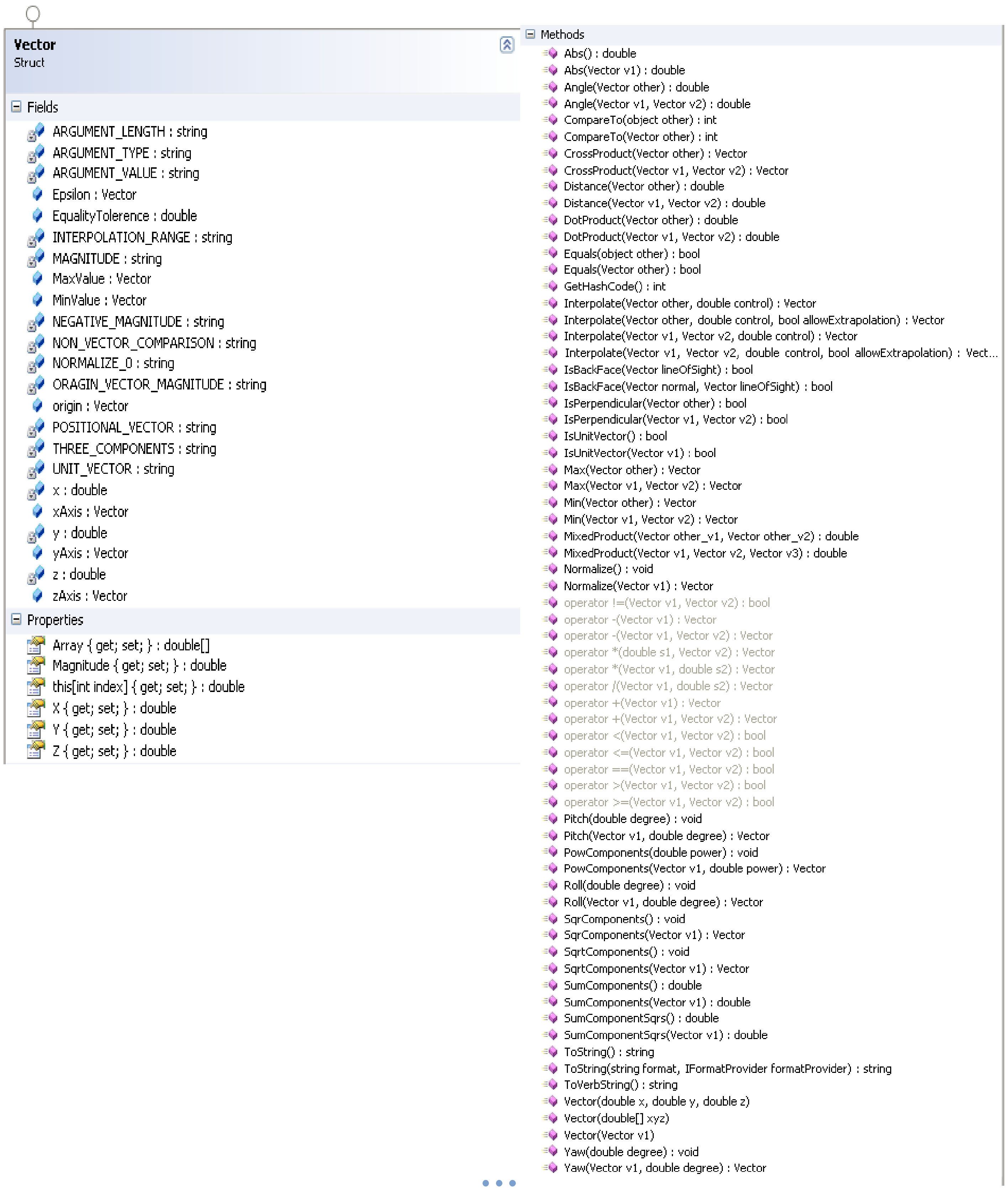

AbstractLeader
Abstract Class
AbstractCharacter
BombermanLeader
Sealed Class
AbstractLeader
Methods
BombermanLeader(string name, int id, Vector position, Vector direction, Vector scale)
OrderToDefuseBomb(CustomObject bomb) : bool
OrderToExplodeDoor(CustomObject door) : bool
OrderToSearchRoom(Room room) : bool
PoliceLeader
Sealed Class
AbstractLeader
Methods
OrderToUseMaul(CustomObject door) : bool
OrderToUseRope(CustomObject window) : bool
PoliceLeader(string name, int id, Vector position, Vector direction, Vector scale)
FireFighterLeader
Sealed Class
AbstractLeader
Methods
FireFighterLeader(string name, int id, Vector position, Vector direction, Vector scale)
OrderToExtinguishWithFireEngine(CustomObject target) : bool
OrderToExtinguishWithGroundWater(CustomObject target) : bool

4.7.24 Civilian

This class is used for civilians to be rescued in the building.

4.7.25 Vector

This is the main class for defining a position, direction and size on the map. It also has methods to calculate mathematical operations.



Vector
Struct

Fields

- ARGUMENT_LENGTH : string
- ARGUMENT_TYPE : string
- ARGUMENT_VALUE : string
- Epsilon : Vector
- EqualityTolerence : double
- INTERPOLATION_RANGE : string
- MAGNITUDE : string
- MaxValue : Vector
- MinValue : Vector
- NEGATIVE_MAGNITUDE : string
- NON_VECTOR_COMPARISON : string
- NORMALIZE_0 : string
- ORAGIN_VECTOR_MAGNITUDE : string
- origin : Vector
- POSITIONAL_VECTOR : string
- THREE_COMPONENTS : string
- UNIT_VECTOR : string
- x : double
- xAxis : Vector
- y : double
- yAxis : Vector
- z : double
- zAxis : Vector

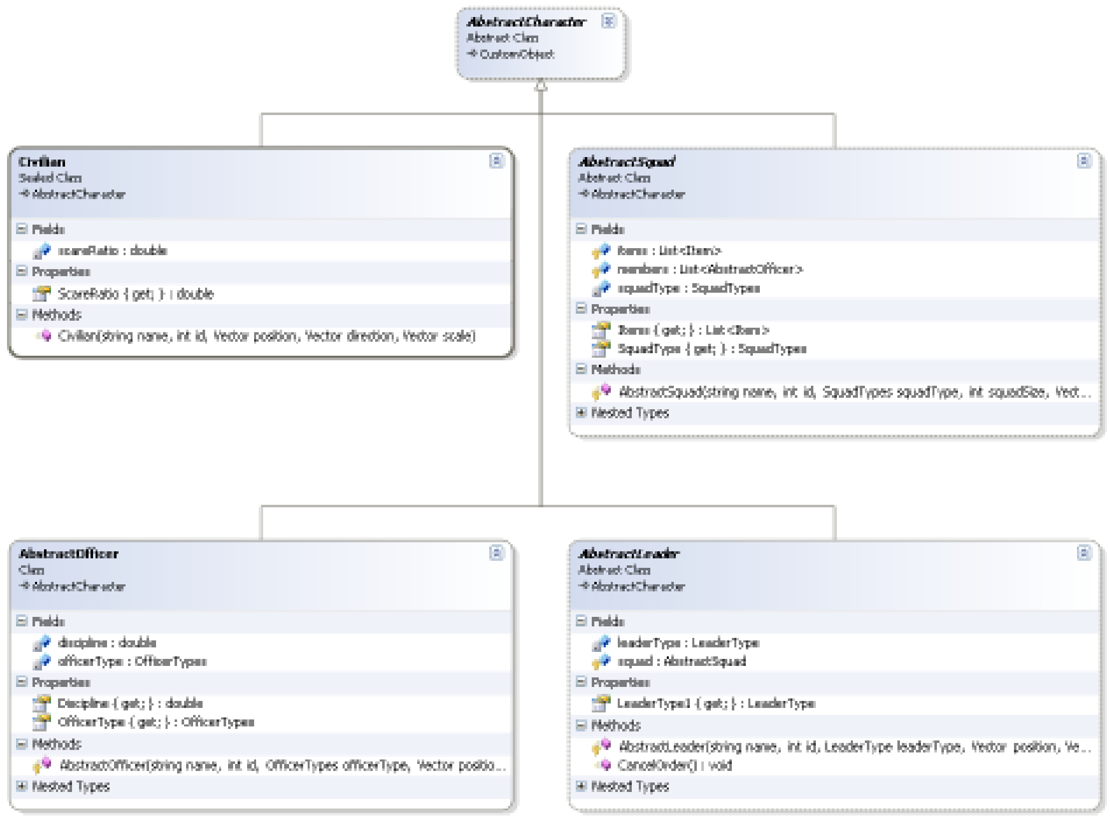
Properties

- Array { get; set; } : double[]
- Magnitude { get; set; } : double
- this[int index] { get; set; } : double
- X { get; set; } : double
- Y { get; set; } : double
- Z { get; set; } : double

Methods

- Abs() : double
- Abs(Vector v1) : double
- Angle(Vector other) : double
- Angle(Vector v1, Vector v2) : double
- CompareTo(object other) : int
- CompareTo(Vector other) : int
- CrossProduct(Vector other) : Vector
- CrossProduct(Vector v1, Vector v2) : Vector
- Distance(Vector other) : double
- Distance(Vector v1, Vector v2) : double
- DotProduct(Vector other) : double
- DotProduct(Vector v1, Vector v2) : double
- Equals(object other) : bool
- Equals(Vector other) : bool
- GetHashCode() : int
- Interpolate(Vector other, double control) : Vector
- Interpolate(Vector other, double control, bool allowExtrapolation) : Vector
- Interpolate(Vector v1, Vector v2, double control) : Vector
- Interpolate(Vector v1, Vector v2, double control, bool allowExtrapolation) : Vect…
- IsBackFace(Vector lineOfSight) : bool
- IsBackFace(Vector normal, Vector lineOfSight) : bool
- IsPerpendicular(Vector other) : bool
- IsPerpendicular(Vector v1, Vector v2) : bool
- IsUnitVector() : bool
- IsUnitVector(Vector v1) : bool
- Max(Vector other) : Vector
- Max(Vector v1, Vector v2) : Vector
- Min(Vector other) : Vector
- Min(Vector v1, Vector v2) : Vector
- MixedProduct(Vector other_v1, Vector other_v2) : double
- MixedProduct(Vector v1, Vector v2, Vector v3) : double
- Normalize() : void
- Normalize(Vector v1) : Vector
- operator !=(Vector v1, Vector v2) : bool
- operator -(Vector v1) : Vector
- operator -(Vector v1, Vector v2) : Vector
- operator *(double s1, Vector v2) : Vector
- operator *(Vector v1, double s2) : Vector
- operator /(Vector v1, double s2) : Vector
- operator +(Vector v1) : Vector
- operator +(Vector v1, Vector v2) : Vector
- operator <(Vector v1, Vector v2) : bool
- operator <=(Vector v1, Vector v2) : bool
- operator ==(Vector v1, Vector v2) : bool
- operator >(Vector v1, Vector v2) : bool
- operator >=(Vector v1, Vector v2) : bool
- Pitch(double degree) : void
- Pitch(Vector v1, double degree) : Vector
- PowComponents(double power) : void
- PowComponents(Vector v1, double power) : Vector
- Roll(double degree) : void
- Roll(Vector v1, double degree) : Vector
- SqrComponents() : void
- SqrComponents(Vector v1) : Vector
- SqrtComponents() : void
- SqrtComponents(Vector v1) : Vector
- SumComponents() : double
- SumComponents(Vector v1) : double
- SumComponentSqrs() : double
- SumComponentSqrs(Vector v1) : double
- ToString() : string
- ToString(string format, IFormatProvider formatProvider) : string
- ToVerbString() : string
- Vector(double x, double y, double z)
- Vector(double[] xyz)
- Vector(Vector v1)
- Yaw(double degree) : void
- Yaw(Vector v1, double degree) : Vector

Below is shown an inheritance map among the abstract classes in the Engine module.


AbstractCharacter
Abstract Class
CustomObject
Civilian
Sealed Class
AbstractCharacter
Fields
scareRatio : double
Properties
ScareRatio { get; } : double
Methods
Civilian(string name, int id, Vector position, Vector direction, Vector scale)
AbstractSquad
Abstract Class
AbstractCharacter
Fields
items : List<Item>
members : List<AbstractOfficer>
squadType : SquadTypes
Properties
Items { get; } : List<Item>
SquadType { get; } : SquadTypes
Methods
AbstractSquad(string name, int id, SquadTypes squadType, int squadSize, Vect...
Nested Types
AbstractOfficer
Class
AbstractCharacter
Fields
discipline : double
officerType : OfficerTypes
Properties
Discipline { get; } : double
OfficerType { get; } : OfficerTypes
Methods
AbstractOfficer(string name, int id, OfficerTypes officerType, Vector positio...
Nested Types
AbstractLeader
Abstract Class
AbstractCharacter
Fields
leaderType : LeaderType
squad : AbstractSquad
Properties
LeaderType1 { get; } : LeaderType
Methods
AbstractLeader(string name, int id, LeaderType leaderType, Vector position, Ve...
CancelOrder() : void
Nested Types

4.8 AI

4.8.1 Path Finder

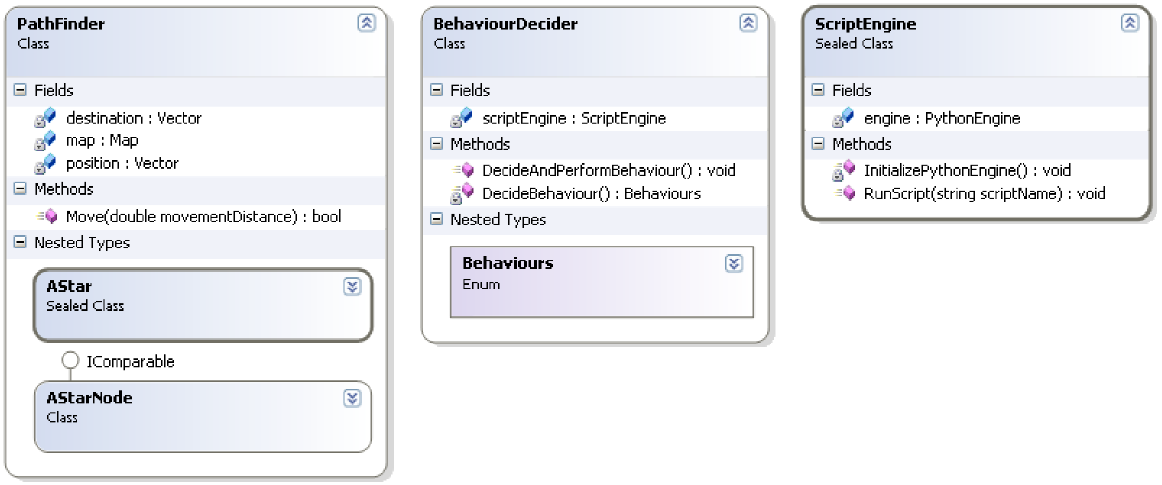
Given the start and end points, this class makes the necessary calculations to find a path between two poins on the map.

4.8.2 Behaviours Decider

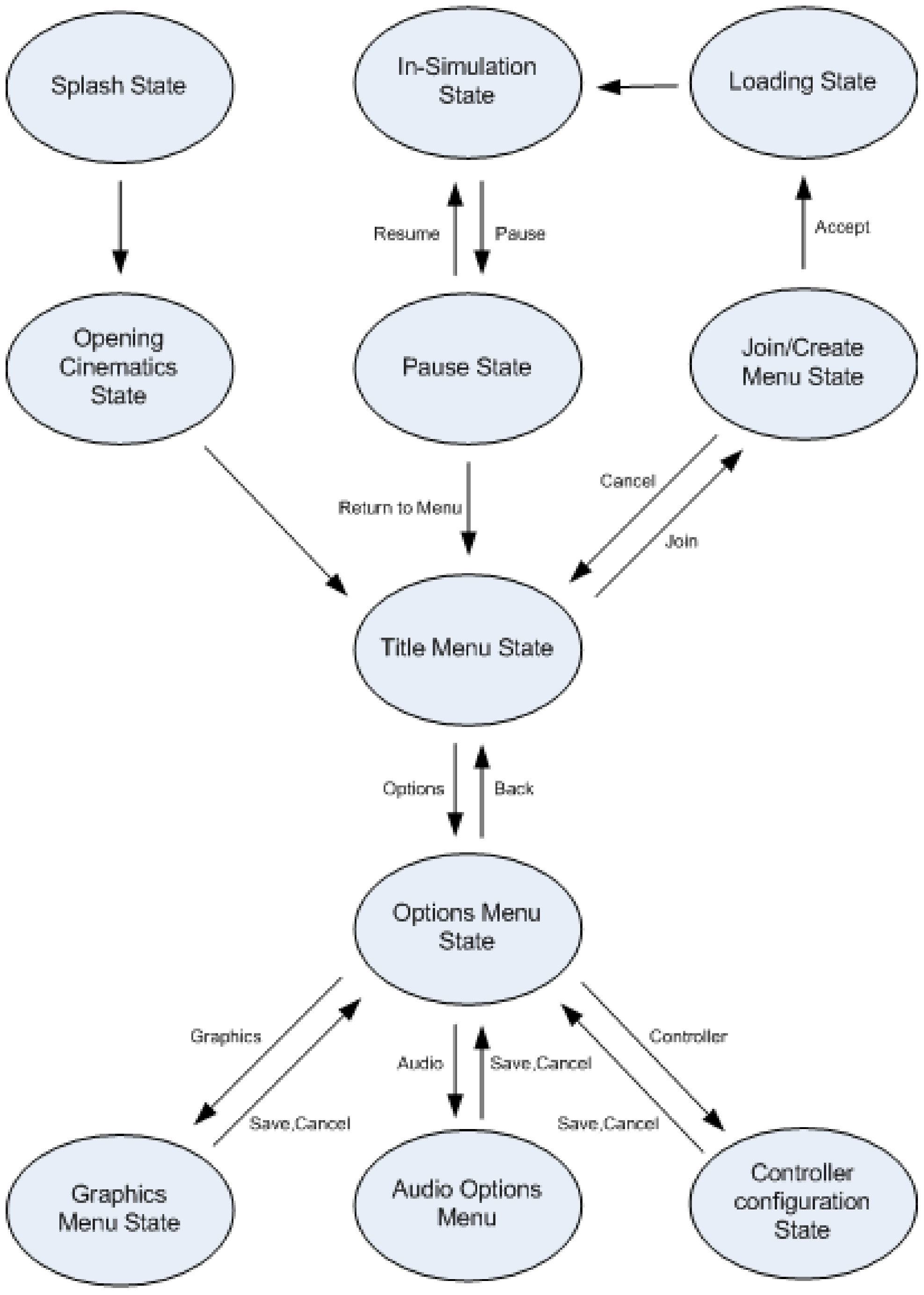
Using a script engine and looking at the objects' behavior, this class determines how the object should act in the simulation

4.8.3 Script Engine

This class is used by the behavior decider.



5 State Transition Diagram



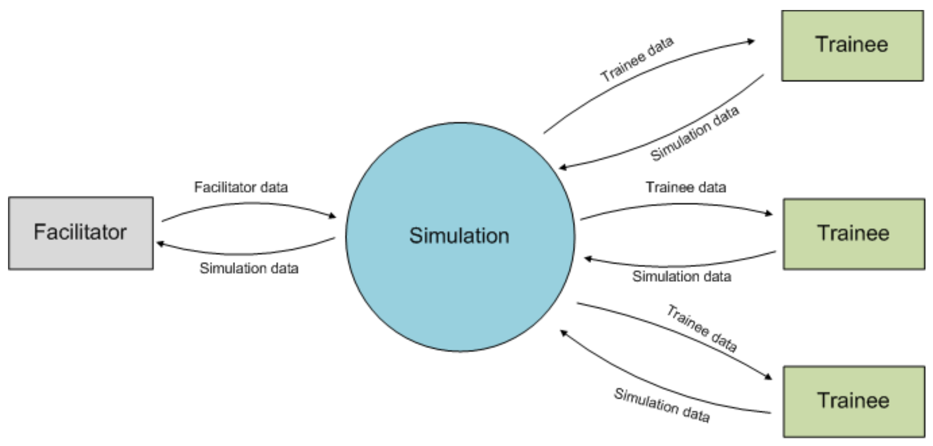
State Transition Diagram

6 System Analysis

6.1 Data Flow Diagram

6.1.1 DFD Level-0

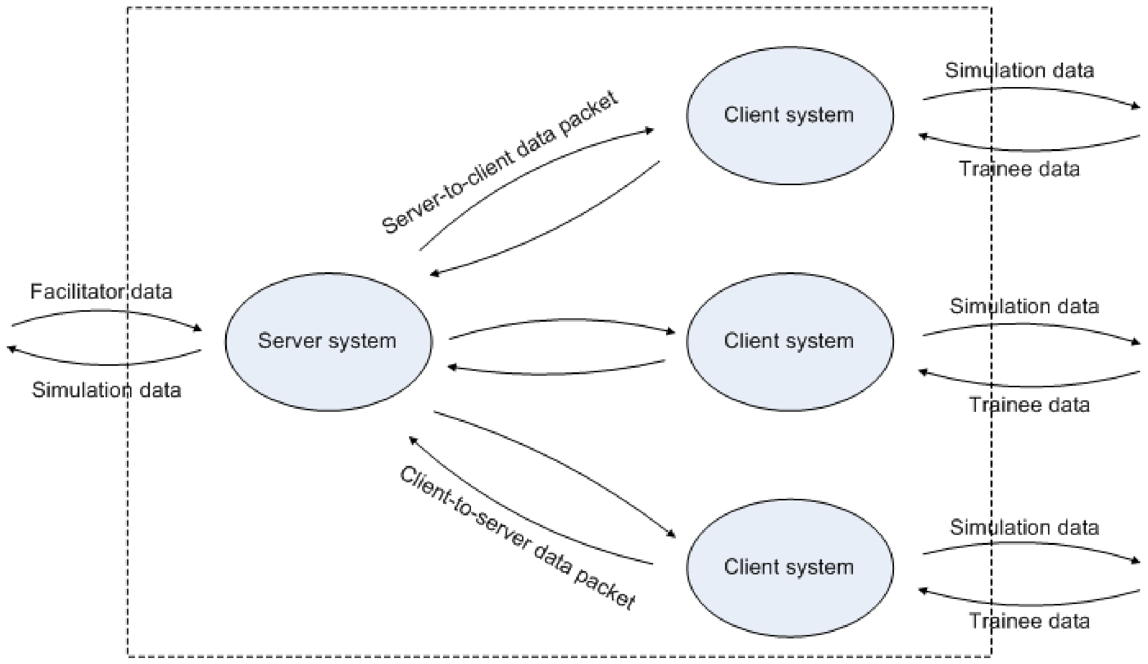
This diagram explains Data Flow Level-0 of the simulation. Facilitator and trainees sends commands via mouse, keyboard and microphone. They receive graphical representations of the simulation environment and listen to other users from their speaker.



DFD: Level 0

6.1.2 DFD Level-1

This diagram explains Data Flow Level-1 of the simulation. Client-to-server data packets include information such as user's outgoing voice and user commands. Server-to-client data packets include information such as other users' incoming voice, environment objects' modified properties.



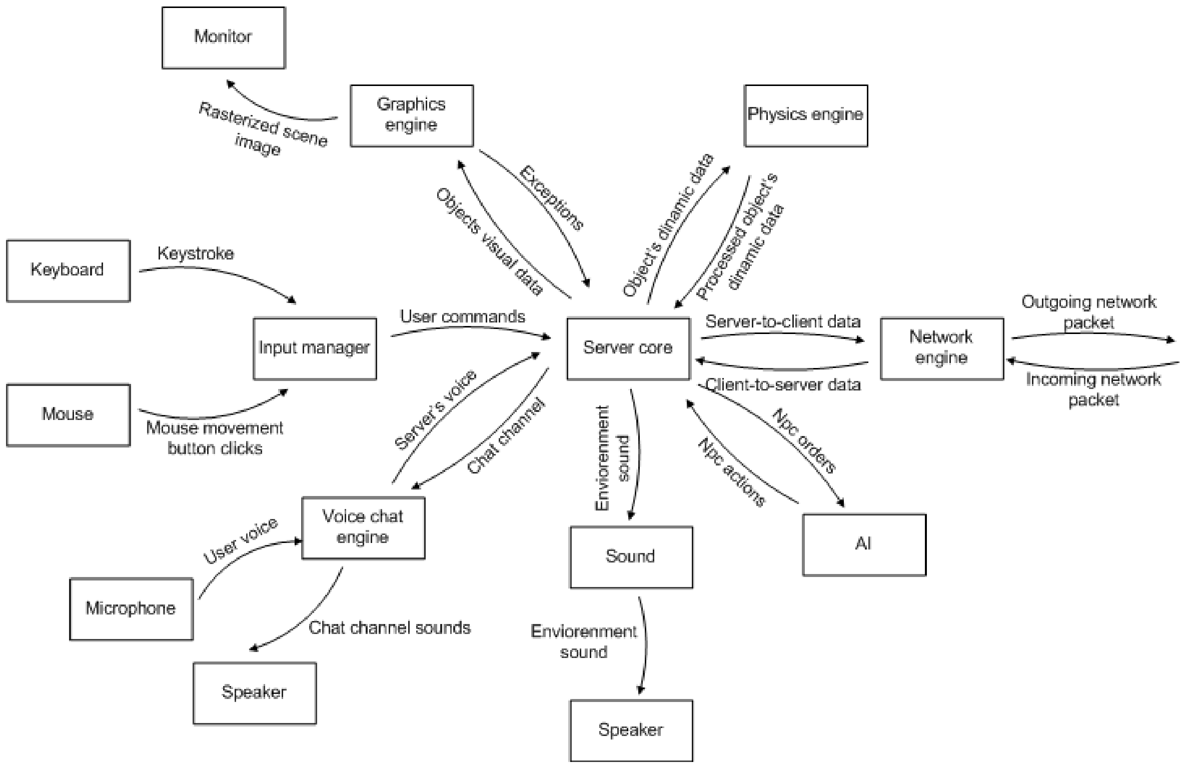
DFD: Level 1

6.1.3 DFD Level-2

These diagrams explain Data Flow Level-1 of the simulation.

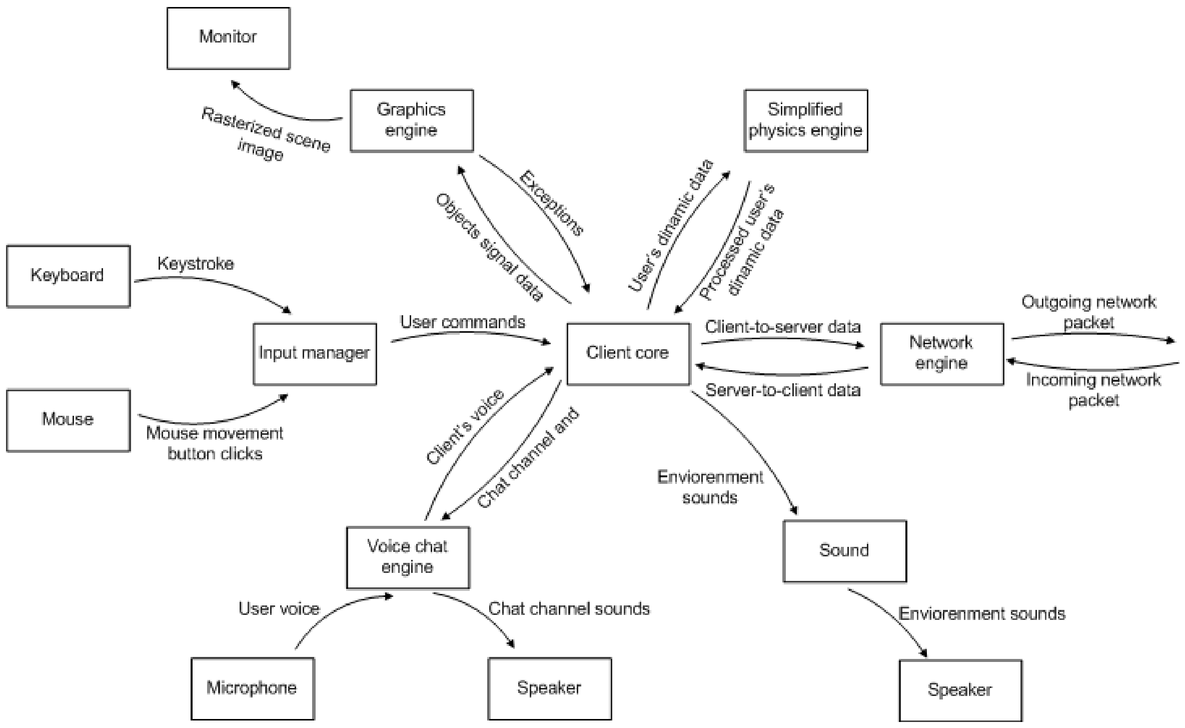
6.1.3.1 Server Core

- Object's visual data contains visible objects information.
- Graphics engine returns only exceptions to server core.
- Object's dynamic data contains physical object information.
- Processed object's dynamic data consists of physical object's modified information.
- NPC orders consist of given to non-playing characters.
- NPC actions consist of non-playing characters' reaction to given orders.
- User commands are commands generated by the user inputs.



6.1.3.2 Client Core

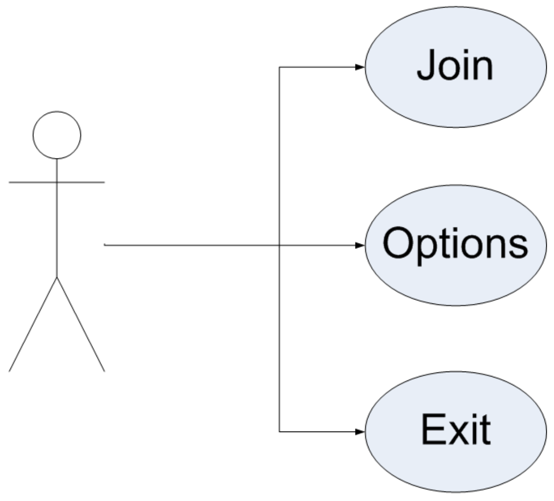
- Object’s signal data contains visible objects information.
- Graphics engine returns only exceptions to server core.
- User’s dynamic data contains user’s character’s physical information.
- Processed user’s dynamic data consists of physical object’s modified information
- User commands are commands generated by the user inputs.



6.2 Use Case Diagrams

6.2.1 Client: Menu State Use Case

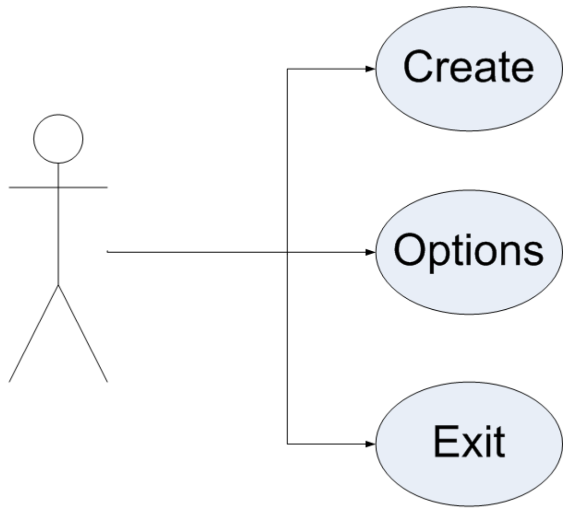
This is the main screen the client will face when starting the program. The screen has 3 buttons.
Join: Connect to the server at the given IP address.
Options: Set program options such as graphics, audio and controller configurations.
Exit: Terminates the program.



Client: Menu state

6.2.2 Server: Menu State Use Case

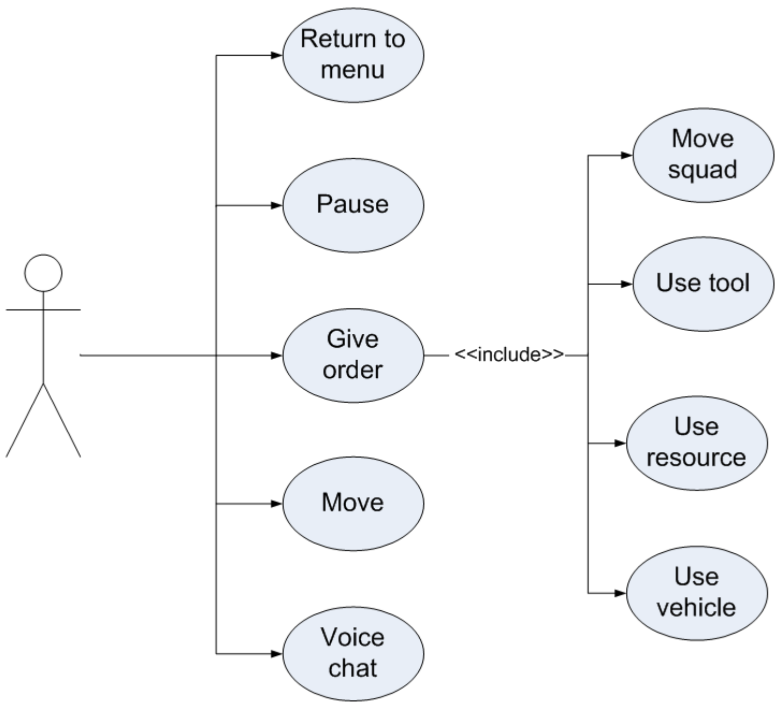
This is the main screen the server will face when starting the program. The screen has 3 buttons.
Create: After clicking this button and choosing the simulation mode, the simulation will initialize. The clients will then be able to connect the server by entering the server's IP.
Options: Set program options such as graphics, audio and controller options.
Exit: Terminates the program.



Server: Menu state

6.2.3 Client In-Simulation State Use Case

This diagram explains what the client can do while the simulation is running.
Return to Menu: Selecting this will switch the current state to the user menu state.
Pause: Selecting this will send a pause request to server and the simulation's current state will be switched to pause state.
Give Order: The client will be able to give commands such as: Move squad, use tools, use resources and use vehicle. If the simulation is at "Passive Mode", these commands shall be given to the facilitator via voice chat and are executed by the facilitator. If the simulation is at "Active Mode", these commands can be given by either voice chat or user interface.
Move: Moves the user on the map using the keyboard and the mouse.
Voice chat: Users will be able to communicate with the facilitator and other users by voice chat.

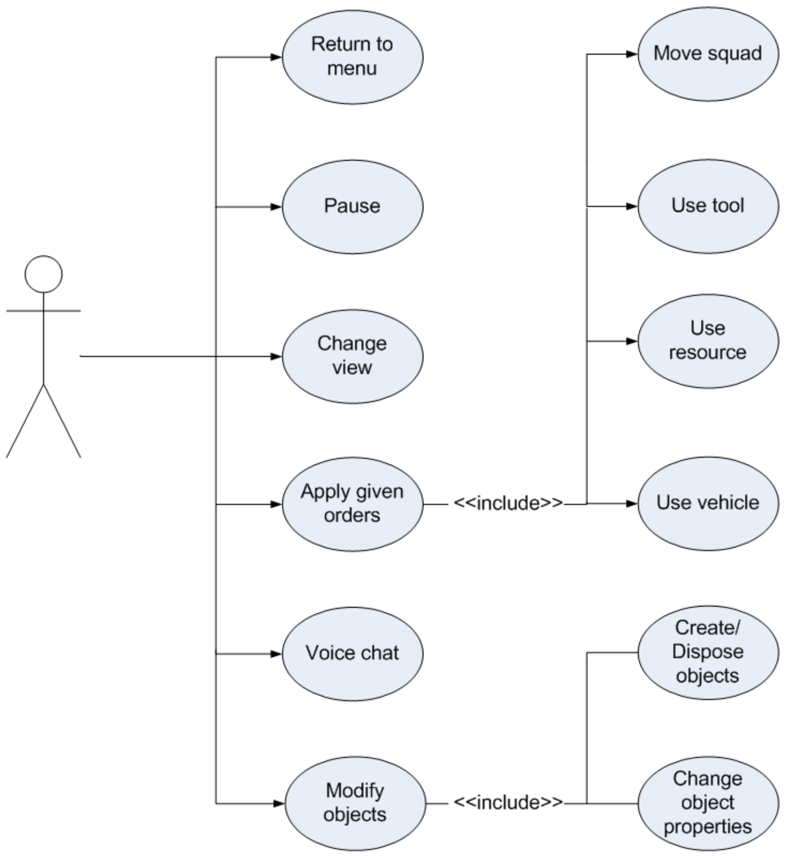


Client: In-simulation state

6.2.4 Server: In-Simulation State Use Case

This diagram explains what the server can do while the simulation is running.
Return to Menu: Selecting this will switch the current state to the server menu state.
Pause: The simulation's current state will be switched to pause state.
Change View: By default, facilitator will begin at "Free View", in which he can move in any direction without any constraint. He also can view the scene directly from any client's view at "User View". Another view mode is the "Map View", in which the facilitator sees the clients as little symbols on a full screen map.

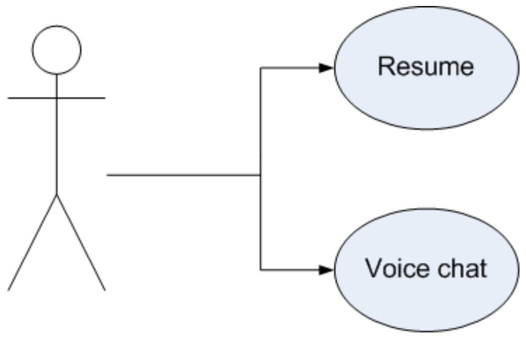
Apply given orders: Facilitator can, at any time, execute orders such as: Move squad, use tools, use resources and use vehicle. At "Passive Mode", these orders may be executed only by the facilitator.
Voice chat: Facilitator can communicate with the clients via voice chat.
Modify objects: Depending on the flow of the scenario, facilitator can create or dispose objects, or modify attributes of an object.



Server: In-simulation state

6.2.5 Server and Client: Pause State Use Case

This diagram explains available actions for the server and the user when the simulation is paused.
Resume: Selecting this will switch simulation's current to in-simulation state.
Voice chat: During the Pause State, users will be able to voice chat.



Server and Client: Pause state

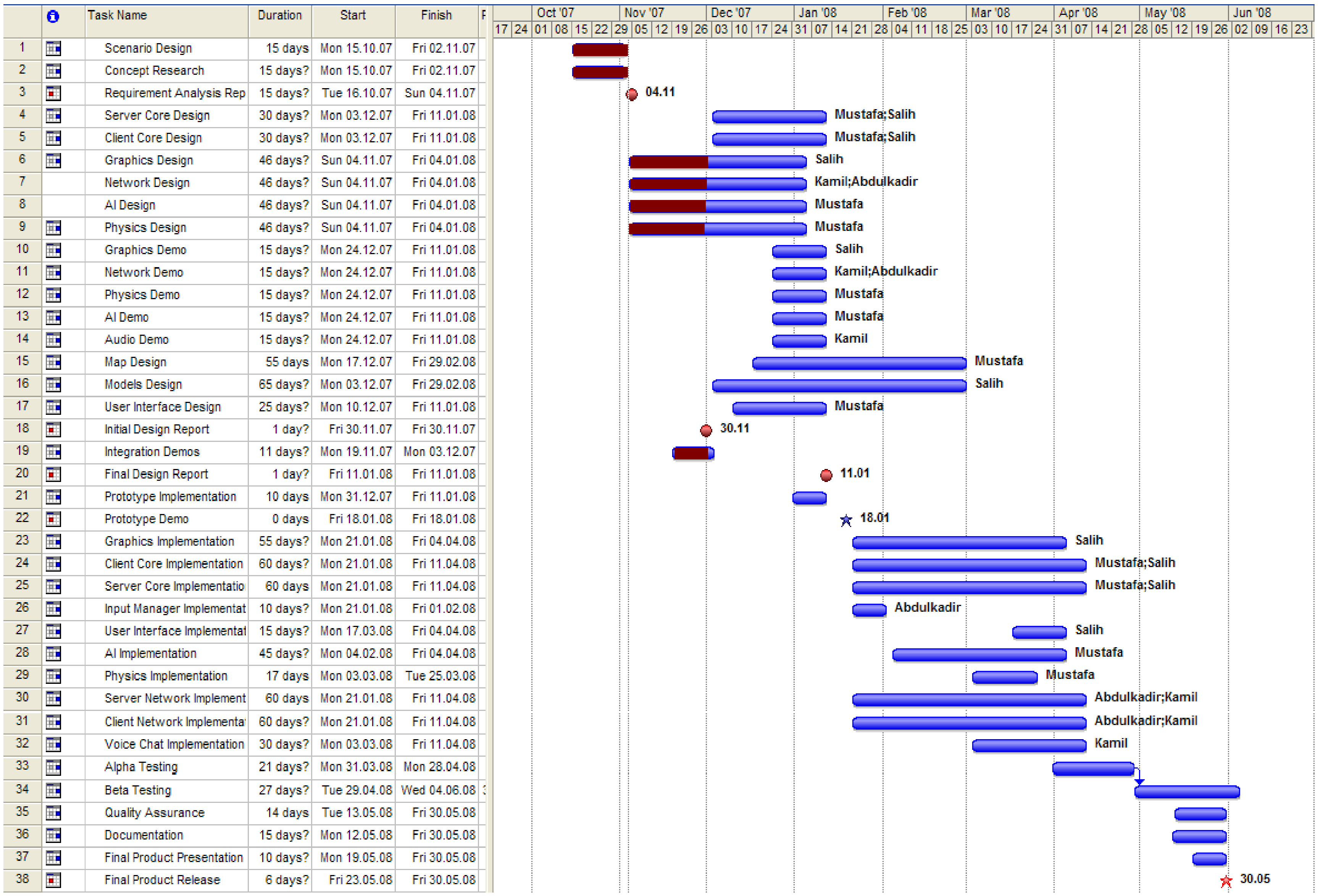
7 Tools

- The system will run on Windows XP operating system. This is preferred because of its large usage and that other tools were designed to run on XP.
- .NET will be used as programming platform throughout the project since the group members are relatively experienced in C#.
- Irrlicht game engine will be used for graphics. The easiness of using this engine and wide support on developer's site were important facts on our choosing this tool as graphics renderer.
- Textures and images will be edited on Paint Shop Pro.
- For designing 3D objects and animations, 3DSMax will be used

8 Future Works

In this initial design report the overall project is designed with some partially designed modules. Until the final design report all modules are planned to be fully designed with all elements fully designed and only implementation process will be left to complete to next term. Also before the due date of final report a prototype demo will be presented to feature some basic functionalities of the final product and the integration between the modules will be demonstrated. But before this final prototype, module demos will be presented separately to our assistant.

9 Schedule



	Task Name	Duration	Start	Finish	
1	Scenario Design	15 days	Mon 15.10.07	Fri 02.11.07	
2	Concept Research	15 days?	Mon 15.10.07	Fri 02.11.07	
3	Requirement Analysis Rep	15 days?	Tue 16.10.07	Sun 04.11.07	04.11
4	Server Core Design	30 days?	Mon 03.12.07	Fri 11.01.08	Mustafa;Salih
5	Client Core Design	30 days?	Mon 03.12.07	Fri 11.01.08	Mustafa;Salih
6	Graphics Design	46 days?	Sun 04.11.07	Fri 04.01.08	Salih
7	Network Design	46 days?	Sun 04.11.07	Fri 04.01.08	Kamil;Abdulkadir
8	AI Design	46 days?	Sun 04.11.07	Fri 04.01.08	Mustafa
9	Physics Design	46 days?	Sun 04.11.07	Fri 04.01.08	Mustafa
10	Graphics Demo	15 days?	Mon 24.12.07	Fri 11.01.08	Salih
11	Network Demo	15 days?	Mon 24.12.07	Fri 11.01.08	Kamil;Abdulkadir
12	Physics Demo	15 days?	Mon 24.12.07	Fri 11.01.08	Mustafa
13	AI Demo	15 days?	Mon 24.12.07	Fri 11.01.08	Mustafa
14	Audio Demo	15 days?	Mon 24.12.07	Fri 11.01.08	Kamil
15	Map Design	55 days	Mon 17.12.07	Fri 29.02.08	Mustafa
16	Models Design	65 days?	Mon 03.12.07	Fri 29.02.08	Salih
17	User Interface Design	25 days?	Mon 10.12.07	Fri 11.01.08	Mustafa
18	Initial Design Report	1 day?	Fri 30.11.07	Fri 30.11.07	30.11
19	Integration Demos	11 days?	Mon 19.11.07	Mon 03.12.07	
20	Final Design Report	1 day?	Fri 11.01.08	Fri 11.01.08	11.01
21	Prototype Implementation	10 days	Mon 31.12.07	Fri 11.01.08	
22	Prototype Demo	0 days	Fri 18.01.08	Fri 18.01.08	18.01
23	Graphics Implementation	55 days?	Mon 21.01.08	Fri 04.04.08	Salih
24	Client Core Implementation	60 days?	Mon 21.01.08	Fri 11.04.08	Mustafa;Salih
25	Server Core Implementatio	60 days	Mon 21.01.08	Fri 11.04.08	Mustafa;Salih
26	Input Manager Implementat	10 days?	Mon 21.01.08	Fri 01.02.08	Abdulkadir
27	User Interface Implementat	15 days?	Mon 17.03.08	Fri 04.04.08	Salih
28	AI Implementation	45 days?	Mon 04.02.08	Fri 04.04.08	Mustafa
29	Physics Implementation	17 days	Mon 03.03.08	Tue 25.03.08	Mustafa
30	Server Network Implement	60 days	Mon 21.01.08	Fri 11.04.08	Abdulkadir;Kamil
31	Client Network Implementa	60 days?	Mon 21.01.08	Fri 11.04.08	Abdulkadir;Kamil
32	Voice Chat Implementation	30 days?	Mon 03.03.08	Fri 11.04.08	Kamil
33	Alpha Testing	21 days?	Mon 31.03.08	Mon 28.04.08	
34	Beta Testing	27 days?	Tue 29.04.08	Wed 04.06.08	
35	Quality Assurance	14 days	Tue 13.05.08	Fri 30.05.08	
36	Documentation	15 days?	Mon 12.05.08	Fri 30.05.08	
37	Final Product Presentation	10 days?	Mon 19.05.08	Fri 30.05.08	
38	Final Product Release	6 days?	Fri 23.05.08	Fri 30.05.08	30.05