ŞahinSim: A Flight Simulator for End-Game Simulations

Özer Özaydın, D. Turgay Altılar
Department of Computer Science
ITU Informatics Institute
Maslak, Istanbul, 34457, Turkey
ozaydinoz@itu.edu.tr altilar@cs.itu.edu.tr

Keywords: flight simulator, visual simulation, end-game simulation

Abstract

In this paper, a new flight simulation framework, ŞahinSim is introduced. ŞahinSim is part of an on going academic project on proportional navigation guided missiles and aircrafts' practical evasive maneuvers against these missiles. ŞahinSim provides an easy to use and flexible 3D visual simulation environment, as well as an interface to an accurate flight dynamics model to this project. Open source projects JSBSim, SimGear, OGRE, SDL and OIS are used within ŞahinSim. Although ŞahinSim is intended to be used for end-game simulations such as air-to-air combat scenarios, it can be extended to be used for also other aerospace related issues.

1. INTRODUCTION

ŞahinSim is a flight simulation framework designed for use in research or academic environments. Aerospace researchers can benefit from ŞahinSim while studying aerospace studies such as aircraft models, tracking algorithms and auto pilot-control applications. ŞahinSim provides users a flight dynamics model, a fairly nice 3D graphical user interface and an easy to use input interface. Without dealing with lots of programming code, a user can concentrate on his own research topic while saving time and effort.

Written in C++ language, ŞahinSim can run on any Linux distribution that includes the dependency packages needed by the flight dynamics model (FDM) and the graphical engine. The FDM and the graphical engine are also written in C++ so that the FDM and graphic libraries can be natively used without requiring any conversion interface which generally causes performance degradation. Moreover, C++ is a widely used-well known programming language which achieves both flexibility and performance requirements of a flight simulation program.

In this project, five open source projects have been integrated; these are JSBSim, SimGear, OGRE, SDL and OIS. The flight dynamics engine is JSBSim [1]. SimGear is used with JSBSim to provide some geometric calculation functions and a convenient logging interface [2]. The graphics engine is OGRE and both OIS and SDL are used for keyboard and joystick input interface [3-5]. A diagram of the mentioned projects in ŞahinSim is shown in (Figure 1).

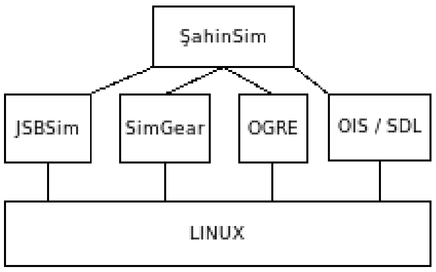


Figure 1. Projects in ŞahinSim

In the next section, related work and motivation are explained. The third section has details about the three main components of ŞahinSim. The fourth section gives information about the implementation details of the application. Two ways of implementing an end-game application is explained in section five, and the last section concludes the paper.

2. RELATED WORK AND MOTIVATION

ŞahinSim has been developed as the second generation visual end-game simulator to another on going project that investigates evasive maneuvers of an aircraft against proportional navigation missiles [6]. The first generation simulator, named VEGAS (Visual End-Game Simulation) was implemented as a complementary work to visualize the end-game between a missile and an aircraft. Akdag and Altilar worked on modeling an agile aircraft capable of moving high-g maneuvers and performing different evasive maneuvers [6]. Moran and Altilar implemented a missile model using proportional navigation techniques to track previously implemented aircraft model [7]. They used three degree of freedom (3-DoF) flight dynamics equations for both missile and aircraft models. The models were embedded in source code so that after changing any model the code had to be recompiled. Models could not be controlled via user inputs from keyboard or joystick; VEGAS could only run for predetermined scenarios which literally indicate that the simulation could run only in batch mode. For visualization, OpenGL was used with immediate mode commands to draw missile and aircraft objects, which caused performance degradation while running the simulation.

Compared to VEGAS, ŞahinSim uses a 6-DoF flight dynamics model. Unlike the embedded FDM in VEGAS, the flight dynamics parameters are configured by using configuration files so that no recompilation is required when FDM is edited. ŞahinSim also provides an easy use keyboard and joystick interface to fly airborne objects. The graphics engine provides better visuals and it even performs better than VEGAS.

At the beginning of this project, rather than writing a new simulation environment, we considered using existing flight simulators or simulation frameworks such as FlightGear and OpenEaagles [8-9]. Flightgear is an open source, multi-platform flight simulator that is designed for gaming and training purposes as well as academic use. After some investigation on Flightgear's code and documents, we realised that Flightgear was designed for only civilian flights. It required a lot of work to turn Flightgear into a combat simulator or to just modify it for using in our project, so we did not use Flightgear. OpenEaagles was another option. It is a comprehensive simulation framework to build simulation applications and it could be used as the framework to build an end game application but we did not use OpenEaagles because of the complexity of the framework. Both projects could be used in our solution but both required too much effort to achieve what we wanted, consequently we decided to build our own simulation environment. ŞahinSim is specially designed for end-game simulations; it only provides the most important capabilities while keeping the source code simple.

3. COMPONENTS OF ŞAHINSIM

Having designed in a modular fashion, ŞahinSim consists of three main components; the flight dynamics model, the graphical engine and the input interface. In this section, the components and their usage in ŞahinSim are explained.

3.1. Flight Dynamics Model

JSBSim is the flight dynamics model of ŞahinSim. JSBSim is an open source project under LGPL license, freely available for proprietary and public use. It's written in C++ and can be compiled by almost any C++ compiler. In 1996, JSBSim was conceived as a batch simulation tool for modeling flight dynamics and flight control. It was designed for use in aircraft design and control courses. Later, the author Berndt started to work in FlightGear project, which is a comprehensive flight simulator, and JSBSim integrated with FlightGear in 1999. Today JSBSim is the default flight dynamics model in FlightGear [10].

JSBSim provides ŞahinSim a mathematical model for rigid aircraft equations of motion. Aerodynamics of the aircraft is modeled using a component buildup method. All forces and moments on the aircraft are calculated by summing up all contributions to each force and moment axis. After calculating all the forces and moments, JSBSim returns next state of the aircraft in discrete time steps [11].

Propulsion system of an aircraft is also modeled in JSBSim. In order to provide a realistic perception of the propulsion system from pilots' point of view, several engine types such as piston, turbine, rocket and electric are defined. Although the models are not precise engineering models, they provide relatively accurate forces and moments on the aircraft [11].

Aerodynamic characteristics, propulsion system, control and automatic control systems (explained later) are described in configuration files which are written in XML format. Any change of the properties can be tried in the simulation without any code change and without recompiling the code.

ŞahinSim uses the libraries generated by JSBSim and it is compiled with the interfaces of JSBSim. Basically ŞahinSim sends the control inputs to JSBSim; JSBSim calculates the next state of the aircrafts and sends back the results to ŞahinSim. As ŞahinSim is an end-game simulation, it generally lasts around a minute or two. The result of the simulation is either a hit or miss of the missile. The simulation doesn't have to deal with takeoff and landing stages of a flight course. Consequently physical landing gear model of JSBSim is not used in ŞahinSim, all references and functions of the landing gear model are removed from JSBSim interface. Aircrafts start in the air and never expected to land with landing gears.

3.1.1. Flight Control and Autopilot Models

In modern aircrafts, either military or commercial ones, the aircraft is controlled through an electronic flight control system. Commands given by the pilot are processed in the control system (i.e., flight computer) and actual control commands to actuate the mechanical control system are produced by the flight computer. By using JSBSim, flight control and autopilot systems can be defined. Just like a real control system, the control system can be designed by connecting chains of control components each other. Some control components that are modeled in JSBSim are; filter (lag, lead-lag, second order, integrator, etc.), switch, gain and summer control blocks. Each component runs in the order of definition and calculates the output regarding its type.

Autopilot systems can also be defined by using the same control components available for a flight computer in JSBSim [12]. The main purpose of the flight computer is to actuate necessary parts(e.g., elevator, aileron, rudder) of the aircraft in order to perform the desired move (e.g., turn, dive). On the other hand, autopilot systems are generally designed to perform specific actions such as keeping the altitude, keeping the heading or automatically heading towards a specified angle or even landing the aircraft automatically. Automatic pilots can also be implemented as a part of the main flight control system depending on the design intend.

3.1.2. Scripted Flights

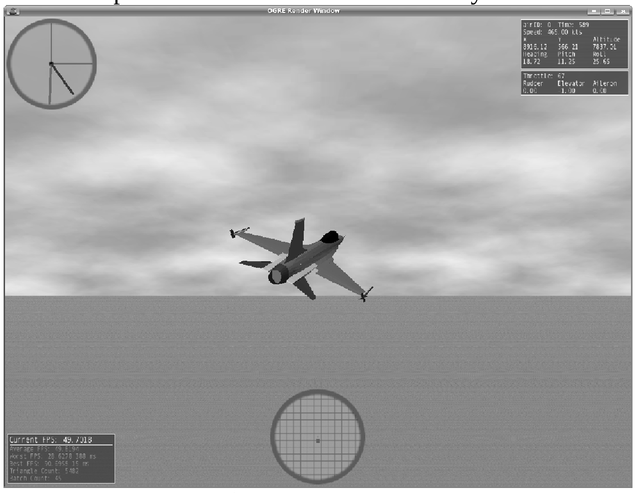
JSBSim can be used as a flight dynamics library or it can be run in batch mode. When run in batch mode, JSBSim controls the aircraft in the way that it is defined in a configuration script. Scripting allows users to define actions when any defined condition occurs. In a scripted flight “action” means setting a property of the aircraft (e.g., setting wing leveler autopilot switch on/off, moving the flight stick, adjusting the throttle, etc.). Any property can be set to a fixed number or set to an output of a function which is defined in the script. Conditions can depend on any property of the aircraft. Test operations "==", "!=",">",">=", "<" and "<=" can be used in conditions, also logical operators “AND” and “OR” can be used with nested condition checks. Simply, a scripted flight can be thought as an autonomous robot flying the aircraft by following the predefined movements in its program. The robot can start the engine at a time, advance the throttle, pull the flight stick when the aircraft reaches at a defined speed, head the aircraft to a location when the aircraft reaches a defined altitude and fly the aircraft to a location. Scripted flights provide exactly the same responses and actions repetitively are very useful in aircraft performance tests and control systems development [11].

3.2. Graphics System

The graphics engine OGRE provides a flexible, easy to use and fast graphical interface which runs on top of OpenGL graphics library [3,13]. Every texture and 3D model can be changed in the simulation without compiling the simulation code. In order to visually observe the details of the simulation ŞahinSim provides a fairly nice graphical interface (Figure 2).

The active aircraft is centered in the screen and the default camera is just at the back of the aircraft. At top-left of the screen, there is the tracking radar, at left-bottom there is the FPS information panel. At the bottom of the screen there is the overall radar and at top-right of the screen there is the flight information panel. Graphical interface can be easily modified using OGRE's configuration files.

Panels are just for basic usage and can be extended easily. Panel positions, panel backgrounds and even the text font in the panels can be modified without any need to



```
airID:  0   Time:  589
Speed:  465.00 kts
X           Y           Altitude
8916.12     566.21      7837.01
Heading     Pitch       Roll
18.72       11.25       25.65

Throttle: 67
Rudder      Elevator    Aileron
0.00        -1.00       0.00

Current FPS:  49.7018
Average FPS:  49.8194
Worst FPS:  28.6278 388 ms
Best FPS:  50.6958 15 ms
Triangle Count:  5482
Batch Count:  45
```

Figure 2. A capture of ŞahinSim

compile the application. Currently there are four panels used.

3.2.1. Tracking Radar

As its name indicates this radar is designed for tracking a specified target. It can be used in dog-fight scenarios as well as in regular escorting missions and end-game simulations. Without need to actually see the target, pilot can estimate the position of the target in three dimensions by just looking at this two dimensional radar (Figure 3).

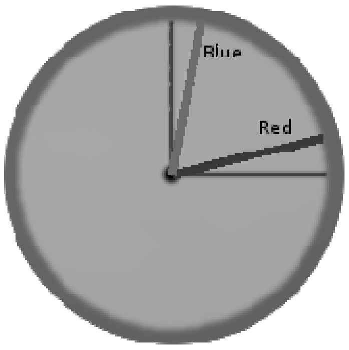


Figure 3. Tracking Radar

There are two needles in the radar. In order to track the target from behind, the pilot should keep the needles on their fixed origins, red needle on the horizontal fixed black line and blue needle on the vertical fixed black line. The red needle indicates the vertical position of the target. The length of the needle varies regarding the altitude difference between the aircraft and its target in a predefined range (longer needle means closer). The blue needle indicates the horizontal position of the target. Its length varies regarding the distance between the aircraft and the target on the plane which is parallel to simulation surface.

3.2.2. Overall Radar

This radar shows all aircraft instances in range of the active aircraft. The radar orientation is fixed and top of the radar is always aligned with the heading of the aircraft. All instances on the radar panel orientates around the aircraft as shown in (Figure 4).

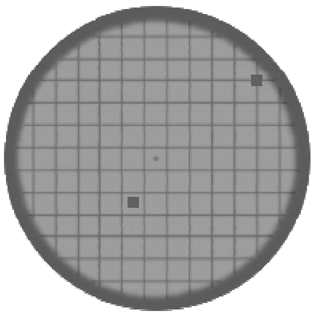
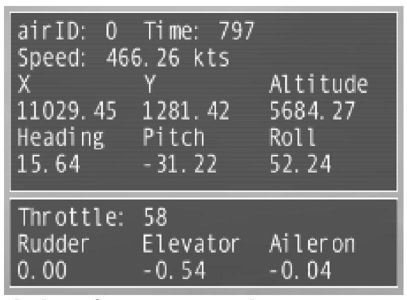


Figure 4. Overall radar

The range is calculated in three dimensions by using Euclidean distance.

3.2.3. Flight Information Panel

The panel shows some essential flight data about the simulation and the aircraft (Figure 5).



```
airID: 0  Time: 797
Speed: 466.26 kts
X         Y         Altitude
11029.45  1281.42   5684.27
Heading   Pitch     Roll
15.64     -31.22    52.24

Throttle: 58
Rudder    Elevator  Aileron
0.00      -0.54     -0.04
```

Figure 5. Flight Information Panel

3.2.4. Cameras

There are two camera types implemented in ŞahinSim. By default, camera is attached to the first aircraft defined in the configuration file. User can cycle through the cameras and through the aircrafts. Camera positions are saved for each camera and each aircraft so that when user selects the previous camera while cycling through the cameras, the camera doesn't reset its orientation, instead previous orientation is restored. Every camera is positioned at the back of its aircraft and parallel to surface of the simulation environment by default.

The first camera is a free-cam that can be positioned around the aircraft. The camera is always directed to the center of the aircraft model and always parallel to the surface. Movement axis of the camera can be thought as a sphere which has the aircraft in its center. In order to move the camera, joystick hat can be used. The second camera is a fixed-yaw camera which is always at the back of the aircraft. The camera's yaw angle always follows the aircraft's yaw angle. The orientation of the camera is automatically updated regarding aircraft's orientation.

3.2.5. Replay Engine

Another feature of ŞahinSim is its replay engine. The replay engine can be used to examine every movement details of the aircrafts for each simulation tick. If replay mode is enabled in the configuration file, the simulation can be paused at any time. After that, simulation enters into replay mode. In replay mode, simulation time can be controlled for forward and backward replays. Speed of the replay mode can also be adjusted.

For every simulation tick, every object's location, orientation, radar information, tracking radar information (if enabled) and flight panel information are saved; this information can be investigated in replay mode. When user quits the replay mode, the simulation time advances to the time the simulation is paused and the simulation keeps on running.

3.3. Input System

Input system of ŞahinSim enables an interactive simulation environment. In some scenarios the simulation may require to run in batch mode but for some other cases instead of automatically controlling the simulation, inputs from a keyboard or a joystick may be required. ŞahinSim uses both OIS (Object Oriented Input System) and SDL (Simple Directmedia Layer) input systems as its input system to provide both interactive and batch mode simulation.

OIS is a cross platform, simple and robust solution for using all kinds of input devices such as mouse, keyboard, joystick, etc. [4]. OIS is written in C++ and natively used by ŞahinSim for just keyboard events. OIS was intended to be used for also joystick events but it did not recognize the joystick that is used in ŞahinSim, consequently for joystick events SDL is preferred. SDL is also a cross platform library which provides low level access to not only input devices but also audio, 3D hardware via OpenGL and 2D framebuffer. SDL is written in C but it works natively with C++ [5]. Although it provides comprehensive interfaces to these media devices, ŞahinSim only uses the interface for the joysticks. Currently user can control any aircraft instance in the simulation from the keyboard or the joystick. Replay engine and cameras can be controlled by using the joystick.

4. IMPLEMENTATION DETAILS

From a high level view, ŞahinSim consists of five components (i.e., set of classes) (Figure 6). Simulation manager (SimMan) is the main component which creates, initializes and controls all other components. Input component is an interface to keyboard and joystick. It polls keyboard and joystick states at each iteration and keeps the state information internally. Input component has both SDL and OIS class instances which are used for polling the actual hardware. Graphics component creates the visual interface, loads the 3D environment, information-radar panels and 3D models. It is the interface to the graphical engine, OGRE. Radar component has position information of all objects in the environment. It is actually an internal class but represented separately to differentiate the component roles.

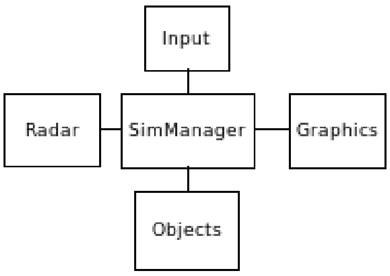


Figure 6. ŞahinSim components

Objects component covers all simulation objects in the environment. These objects are users of the input, graphics and radar components. Depending on the implementation, objects may use any of the simulation components; using all components is not mandatory. It is also intended to make each component as independent as possible so that any component can be edited easily without changing others.

4.1. Objects in ŞahinSim

In C++ terms, all object instances are derived from an abstract class named SimObject. SimMan calls all object instances' functions by using their down casted pointers to SimObject class type. SimObject can be regarded as the interface to ŞahinSim. Any kind of simulation objects can be created, initialized and run through this interface.

Currently all aircraft instances are derived objects from SimObject and aircraft class name is JSBSimPlane (Figure 7). JSBSimPlane object also uses other interfaces and other class instances to fully use the ŞahinSim simulation environment. Additionally, JSBSimPlane is derived from HUDUser and XMLReader classes and it has Model and FDMJSBsim class instances. HUDUser class is used for updating the head-up displays(HUD) of the aircraft. XMLReader class is used for decoding the XML configuration options. Model class is an interface to the graphical engine and hold 3D model information about the object. Finally, JSBSimPlane has an FDMJSBSim class instance to use the JSBSim flight dynamics model.

Different aircraft or missile objects can be implemented by using the model defined above. Regarding the requirements of a particular simulation scenario, ŞahinSim objects can be derived from other additional classes or may contain new classes. Main ŞahinSim structure will not be affected by these particular changes.

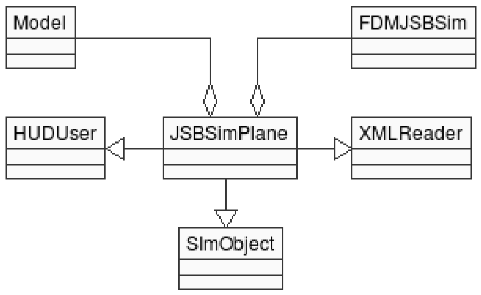


Figure 7. Some ŞahinSim classes

4.2. ŞahinSim Configuration

Simulation environment and aircrafts can be configured using a configuration file in XML format. Without losing time to configure the simulation from source code, ŞahinSim can easily be configured through its configuration file, sahin_conf.xml. The simulation configuration and aircraft configurations must be in context of main XML element named “sahin_conf”. Currently there are two main configuration elements within this context; those XML elements are named environment and aircraft. The “environment” element must be unique, however multiple aircraft instances can be defined by using multiple “aircraft” elements.

The “environment” XML element is used for configuring simulation environment options. The center of the simulation environment (in geodetic coordinates), simulation duration, simulation frame rate are some of the configurable simulation environment parameters.

Each aircraft instance is configured by using “aircraft” XML element. ŞahinSim parses the simulation configuration XML file and creates an aircraft instance for every “aircraft” element. Each aircraft is given a unique number starting from zero in the order they are parsed.

Initial location, orientation, and speed of the aircraft can be changed by just editing the relevant lines. Without losing time on compiling, the simulation can be run with different initial configurations immediately. Moreover, by just changing the related configuration parameters, any desired aircraft can be used under the same initial conditions. The use of aircraft configuration is very handy and makes it easy to run the simulation with different models or different conditions.

4.3. Initialization

Initialization steps of ŞahinSim are briefly illustrated in (Figure 8). SimMan class first parses the ŞahinSim configuration file and fills an internal configuration structure. After reading the configuration, considering simulation resource requirements, it creates the classes related to resource interfaces. First resource is the SDL library which is used for the joystick device. This initialization is mandatory before actually creating the joystick class and before initializing the graphical engine OGRE. If SDL is initialized successfully, next step is initializing the graphical engine. An empty user interface without the SimObjects is created at this step. The sky, terrain are loaded and lighting of the environment is configured. The last resource is the input interface. At this stage SDL joystick and OIS keyboard classes are created and initialized. After initializing the resources, all SimObjects are created and initialized regarding the configuration elements defined in the XML file. SimMan actually creates the SimObject instances and calls the init functions of the objects by passing the related XML element to each instance. SimObjects are responsible for initializing themselves.

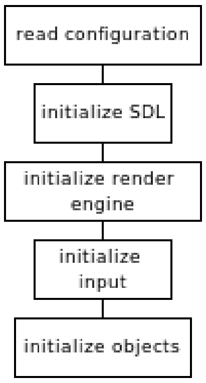


Figure 8. ŞahinSim initialization steps

4.4. Main Simulation Loop

Following the initialization stage, simulation enters an infinite while loop and runs until the stop time is reached or any escape condition defined in the code(e.g., user pressed 'escape' key, missile hit the target, etc..) occurs. Functions that are called in main simulation loop are in (Figure 9).

In each iteration, at the beginning of each loop, a timer to be used for frame regulation is reset. This timer is checked at the end of the loop and if the loop is finished before the expected time, simulation sleeps to synchronize to the required frame rate. After resetting the time, keyboard and joystick states are polled by the input object. Just after polling the input interfaces, simulation checks if any escape condition is occurred. Upon the occurrence of any escape condition, the cleanup method is called to terminate the simulation safely.

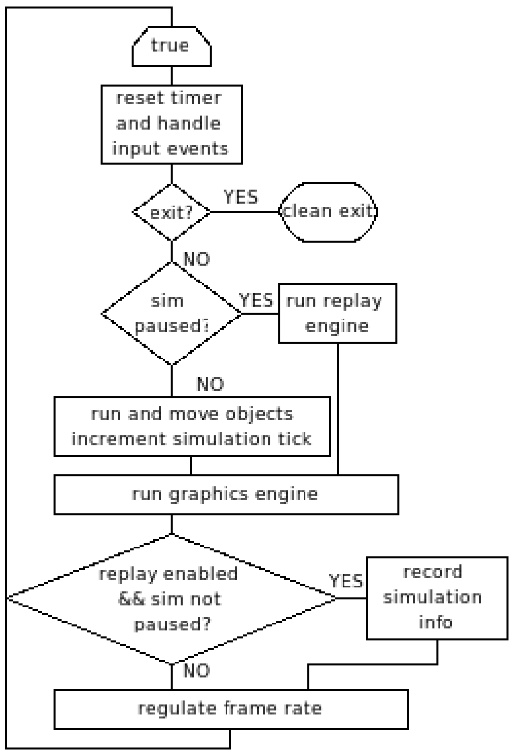


Figure 9. Main simulation loop

The simulation has two running modes; one is active mode and the other is paused mode. If the simulation is in the active mode, all SimObject instances' run and move methods are called. Basically run methods are used for calculating the next state of the object (i.e., orientation, location, speed, etc.) and move methods are used for moving the objects in the 3D environment (at this stage objects are not drawn, only state information in OGRE engine is updated) . After calling these methods, simulation tick counter is incremented.

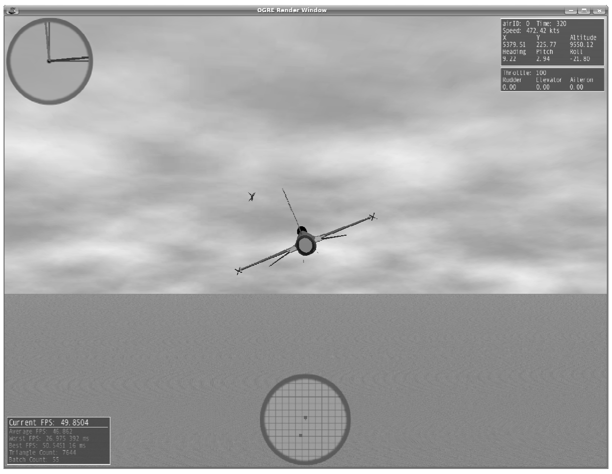
If the simulation is running in the paused mode, ReplayEngine's run method is used to view the state of the simulation at a time frame. In each iteration, if replay mode is enabled, each SimObjects' state information is recorded in ReplayEngine class. The main simulation loop ends with the frame regulating function.

5. IMPLEMENTATION OF AN END-GAME APPLICATION IN ŞAHINSIM

There are two approaches to implement an end-game application in ŞahinSim. The first approach is embedding an existing missile or aircraft model code into ŞahinSim. In this approach, all model classes are derived from SimObject abstract class and they implement all interface methods of SimObject class. Since the code is embedded into ŞahinSim, missile and aircraft classes are created by SimMan class manually (i.e., not from configuration file). Actually this is not the expected use of ŞahinSim because it will only benefit from the graphical and the input interfaces.

Second approach is using JSBSim models and ŞahinSim. Aircraft and missile models are defined in JSBSim model definition format. For the aircraft model, there already exist some aircraft definitions in JSBSim format in FlightGear and JSBSim repositories. The missile model probably has to be implemented initially, since there are no ready to use models found yet. JSBSim scripts can also be used with this approach in order to try different types of aircraft models performing same maneuvers at the same time. This approach requires more effort at the beginning; however it will have the advantages of using ŞahinSim which will save time during experiments.

As and end-game application, a similar scenario to VEGAS is implemented in ŞahinSim by using the second approach defined above. In VEGAS, a missile tries to hit an aircraft while the aircraft tries to evade the missile by performing evasive maneuvers. Initial location, speed, heading, and some other parameters of the missile and the aircraft are changed and the simulation is run again to investigate both performance of the missile's guidance algorithm and aircraft's evasive maneuver algorithms. It is intended to implement the same scenario with VEGAS but the missile model is still being work on so that another aircraft model is used to chase the targeted aircraft manually (Figure 10).



```
airID: 0  Time: 320
Speed: 472.42 kts
X          Y          Altitude
5379.51    225.77     9550.12
Heading    Pitch      Roll
9.22       2.94       -21.80

Throttle: 100
Rudder     Elevator   Aileron
0.00       0.00       0.00
```

```
Current FPS: 49.8504
Average FPS: 46.862
Worst FPS: 26.975 392 ms
Best FPS: 50.5451 16 ms
Triangle Count: 7644
Batch Count: 55
```

Figure 10. An aircraft chasing another one

The aircrafts start in the air with initial location and speed such that the escaping aircraft is in front of the chasing aircraft. Both aircrafts are JSBSim aircrafts; the one in the front is controlled by a JSBSim script and performs predefined maneuvers where the one in the back is controlled by either joystick or keyboard manually trying to chase the other. When the missile model is ready in JSBSim format, the chasing aircraft model will be replaced by the missile model.

6. CONCLUSION AND FUTURE WORK

ŞahinSim is an end-game simulation environment providing a 3D graphical interface, an input interface and an interface to a popular flight dynamics model. Multiple instances of manual or script controlled, JSBSim or user-created airborne objects can run in the same environment. It is easy to understand simple and straightforward architecture of the code, and the generic design of ŞahinSim allows it to be extended while considering specific requirements.

ŞahinSim application presented in this paper can be used in the missile-aircraft engagement scenarios which were simulated using VEGAS application [6,7]. However, there are some areas that will be improved to provide a more generic simulation environment. Network support is one of these areas. In a user controlled combat scenario, every user can control objects with a number of joysticks but it won't be feasible because only one camera will be active at a time in this version of ŞahinSim. In that case only one user can actually see his object and control it on the screen. A network interface will be defined to allow multiple users share the same simulation environment. Another improvement area is collision detection. Current implementation leaves collision awareness to objects in the environment. Every object must check that if it collides with another object itself. An overall collision detection technique which checks collisions out of object implementations will be introduced.

References

[1] JSBSim: http://jsbsim.sourceforge.net/
[2] SimGear: http://www.simgear.org/
[3] OGRE: http://www.ogre3d.org/
[4] OIS: http://sourceforge.net/projects/wgois
[5] SDL: http://www.libsdl.org/
[6] Akdag, R., Altilar, D.T., "A Comparative Study on Practical Evasive Maneuvers against Proportional Navigation Missiles", AIAA Guidance, Navigation and Control Conference, San Francisco, CA, 2005.
[7] Moran, I., Altilar, D.T., "Three Plane Approach for 3D True Proportional Navigation", AIAA Guidance, Navigation, and Control Conference, San Francisco, CA, 2005.
[8] FlightGear: http://www.flightgear.org/
[9] OpenEaagles: http://www.openeaagles.org/
[10] Berndt, J.S., "JSBSim: An Open Source Flight Dynamics Model in C++", AIAA Guidance, Navigation, and Control Conference, Providence, Rhode Island, 2004.

[11] Berndt, J.S., JSBSim Reference Manual,
http://jsbsim.sourceforge.net/JSBSimReferenceManual.pdf
[12] Berndt, J.S., “Automatic Flight in JSBSim”, JSBSim
Project Technical Report
http://jsbsim.sourceforge.net/AutomaticFlightInJSBSim.pdf
[13] OpenGL: http://www.opengl.org/