SAI-BOTS: Scripted Artificial Intelligent Basic Online Tank Simulator

William E. Brandstetter, Michael P. Dye, Jesse D. Phillips, Jason C. Porterfield,
Frederick C. Harris Jr., Brian T. Westphal

Department of Computer Science and Engineering
University of Nevada, Reno
1664 N. Virginia St.
Reno, NV 89557, USA
{brandste, mdye, jdp, fredh, westphal}@cse.unr.edu

Abstract *This paper presents details of the specification, design and functionality of the Scripted Artificial Intelligent Basic Online Tank Simulator (SAI-BOTS), an interactive virtual environment that allows users to script tanks to fight each other. SAI-BOTS provides a 3D environment and allows the user to play with other people and learn basic programming and AI Scripting. The users can navigate through the 3D world using a first-person view, third-person view, or its "Blind Mode." As this project is in its development phase, the current status and future work are included.*

Keywords: 3D modeling, Lua, software specification, real-time interaction, artificial intelligence, scripting, graphics.

1 Introduction

The Scripted Artificially Intelligent Basic Online Tank Simulator (SAIBOTS) is a networked tank game that emphasizes game-play through both predefined and custom scripts. The program is intended to teach students the basics of artificial intelligence through an interactive gaming environment. This environment can include computer controlled tanks based on either predefined or custom scripts as well as other players connected through a network. The player is allowed to manually control a tank in a series of games including a free-for-all death-match style game, and a squad-based elimination game.

The focus of the program is its ability to use and generate custom scripts to dictate the behavior of computer controlled tanks. Players are able to create and customize scripts that control the behavior of tanks through a tank API. This API will allow the player to dictate the behavior of the tank. This includes but is not limited to, the tanks speed, offensive and defensive postures, aggressiveness and general combat tactics such as the use of higher ground or cover. The user can generate custom scripts describing actions to be taken during combat and when in need of health.

The free-for-all mode is setup as a tutorial that allows the player to create, change, and apply scripts to all the tanks and monitor the results both as an impartial observer and by facing tanks in combat. In squad-based combat the player will have the ability to command the tanks under his control by setting scripts for each tank. This includes defending as well as attacking a target.

An in-game editor allows one to customize and create scripts on the fly and apply scripts to tanks. This allows players to immediately see the results of newly created scripts and change them accordingly. In order to flatten the learning curve for scripting, users will be able to see how their tanks work against a series of sensors. In one mode, a user will be able to see a 3D world with their tank in it. They can visually see what is happening in real-time and play as if it were a normal game.

In another mode, the user can completely eliminate the 3D world and only display the tanks sensors (position, velocity, collision, health, radar, etc). This helps one get a feel for how a script controls a tank only "seeing" sensors. Once a user feels comfortable controlling the tank only by its sensors, one will be able to start writing scripts with Lua to control tanks.

The remainder of this paper is structured as follows: Section 2 describes the functional and non-functional software requirements for SAI-BOTS, Section 3 presents the use case diagram and a sample of use cases of SAI-BOTS, Section 4 describes SAI-BOTS' high-level architecture, Section 6 reports on SAI-BOTS' current status and points to future work, and Section 7 contains closing remarks.

2 Requirements Specification

Using the notations presented in [3], the requirements for SAI-BOTS, structured as functional and non-functional requirements, are specified in the next two subsections.

2.1 Functional Requirements

The main functional requirements of SAI-BOTS are the following:
1. SAI-BOTS shall provide 3D graphics with being able to look completely around.
2. SAI-BOTS shall provide a real-time scripting environment allowing players to script while playing the game.
3. SAI-BOTS shall provide different game modes including a tutorial and free-for-all deathmatch.
4. SAI-BOTS shall provide the ability for the user to play over a network with other people.
5. SAI-BOTS shall provide a GUI.
6. SAI-BOTS shall provide real-time terrain deformations and textures updated for damage

2.2 Non-Functional Requirements

The most important non-functional requirements for SAI-BOTS are the following:
1. SAI-BOTS shall run on Windows machines.
2. SAI-BOTS shall be written in C++
3. SAI-BOTS scripts shall be written in Lua.
4. SAI-BOTS shall limit scripting to Tank API calls.
5. SAI-BOTS shall limit the number of tanks to eight per team.

3 Use Case Modeling

As part of the formal modeling process, the functionality of SAI-BOTS has been defined using use cases and scenarios. The entire functionality of SAI-BOTS is captured with a high level of abstraction in the use case diagram shown in Figure 1.

Several parts of the use case diagram that describe SAI-BOTS functionality are presented in separate use case diagrams shown in Figures 2 and 3. Examples of specific ways of using the software are provided as use case scenarios in Figures 4, 5, and 6.

3.1 Use Case Diagram

The use case diagram displayed in Figure 1 shows the interaction between a player and SAI-BOTS as well as a server admin and SAI-BOTS.

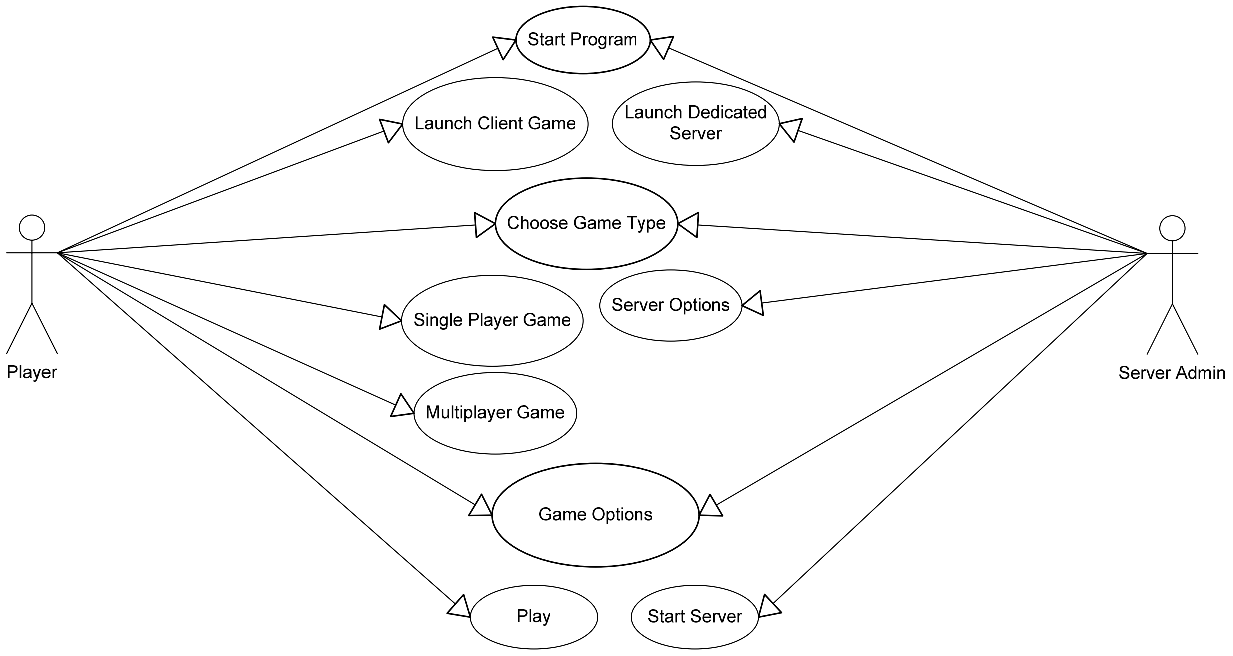

Figure 1. Use Case Diagram for SAI-BOTS

Because of space limitations and the large number of use cases in our diagram, only two detailed use cases and three examples of use case scenarios are presented in this paper. Figure 2 shows the "Single Player Game" use case, where the player chooses to learn how to play the game and script his/her tanks. Figure 3 depicts the "Multiplayer Game" use case, in which the player chooses to connect to a game on another server or host a server through the client.

Use Case: Single Player Game
Use Case ID: UC04
Actor: Player
Preconditions: 1. Client game chosen (UC03)
Flow of Events: 1. Select game type (UC05) 2. Choose or generate tank names (UC06) 3. Play game (UC07)

Figure 2. "Single Player Game" Use Case

For the "Single Player game" use case shown in Figure 2, multiple scenarios are possible. For illustration purposes, Figure 4 presents the primary scenario and Figure 5 shows one of the many possible secondary scenarios of this use case.

Use Case: Multiplayer Game
Use Case ID: UC08
Actor: Player
Preconditions: 1. Client Game chosen (UC03)
Flow of Events: 1. Set up game for multiplayer action 1.1 Choose to host server (UC09) 1.2 Choose to join existing server (UC10)

Figure 3. "Multiplayer Game" Use Case

For the "Multiplayer Game" use case shown in Figure 3. Figure 6 shows one of the many possible secondary scenarios of this use case.

3.2 Requirements Traceability Matrix

Using the format presented in [3], the Requirements Traceability Matrix shown in Figure 7 depicts the mapping between the use cases and the functional requirements of SAI-BOTS.

Use Case: Single Player Game
Primary Scenario: Tutorial
Use Case ID: UC04
Actor: Player
Preconditions: 1. Client game chosen (UC03)
Primary Scenario: 1. The use case begins when the Player selects “Single Player Game” 2. Choose Tutorial 3. Choose easy level of difficulty 4. Set number of tanks in command of to 1 5. Set number of tanks against to 1

Figure 4. Primary Scenario of the “Single Player Game” Use Case

Use Case: Single Player Game
Secondary Scenario: “Deathmatch”
Use Case ID: UC04
Actor: Player
Preconditions: 1. Client game chosen (UC03)
Secondary Scenario: 1. The use case begins when the Player selects “Single Player Game” 2. Choose “Deathmatch” 3. Set Round Limit to 5 minutes 4. Set Kill Limit to 10 kills 5. Set Number of Rounds to 3 rounds 6. Set Game Options

Figure 5. Secondary Scenario of the “Single Player Game” Use Case

4 Architectural Design

The system architecture is made up of several subsystems which communicate through a central subsystem, as shown in Figure 8. Each subsystem is composed of a library, serving as an interface, and a DLL, serving as an implementation of the interface. In this way, updating a particular subsystem's DLL will not interfere with the rest of the system. The five subsystems are the scene subsystem, input subsystem, network subsystem, physics subsystem, and audio subsystem.

Use Case: Multiplayer Game
Secondary Scenario: Host Server
Use Case ID: UC08
Actor: Player
Preconditions: 1. Client game chosen (UC03)
Secondary Scenario: 1. The use case begins when the Player selects “Multiplayer Game” 2. Host Server 3. Choose “Deathmatch” 4. Set Round Limit to 5 minutes 5. Set Kill Limit to 10 kills 6. Set Number of Rounds to 3 rounds 7. Set Server Options

Figure 6. Secondary Scenario of the “Multiplayer Game” Use Case

	Use Cases						
	UC01	UC02	UC03	UC04	UC05	UC06	UC07
F01	X		X				X
F02							X
F03			X	X	X		
F04			X				
F05	X		X				
F06						X	
F07							X

Figure 7. Requirements Traceability Matrix (Partial)

5 Detailed Design

The structure of SAI-BOTS was designed using an object-oriented approach. The organizations of SAI-BOTS into a class diagram and system activity chart are given in subsections 5.1 and 5.2, respectively.

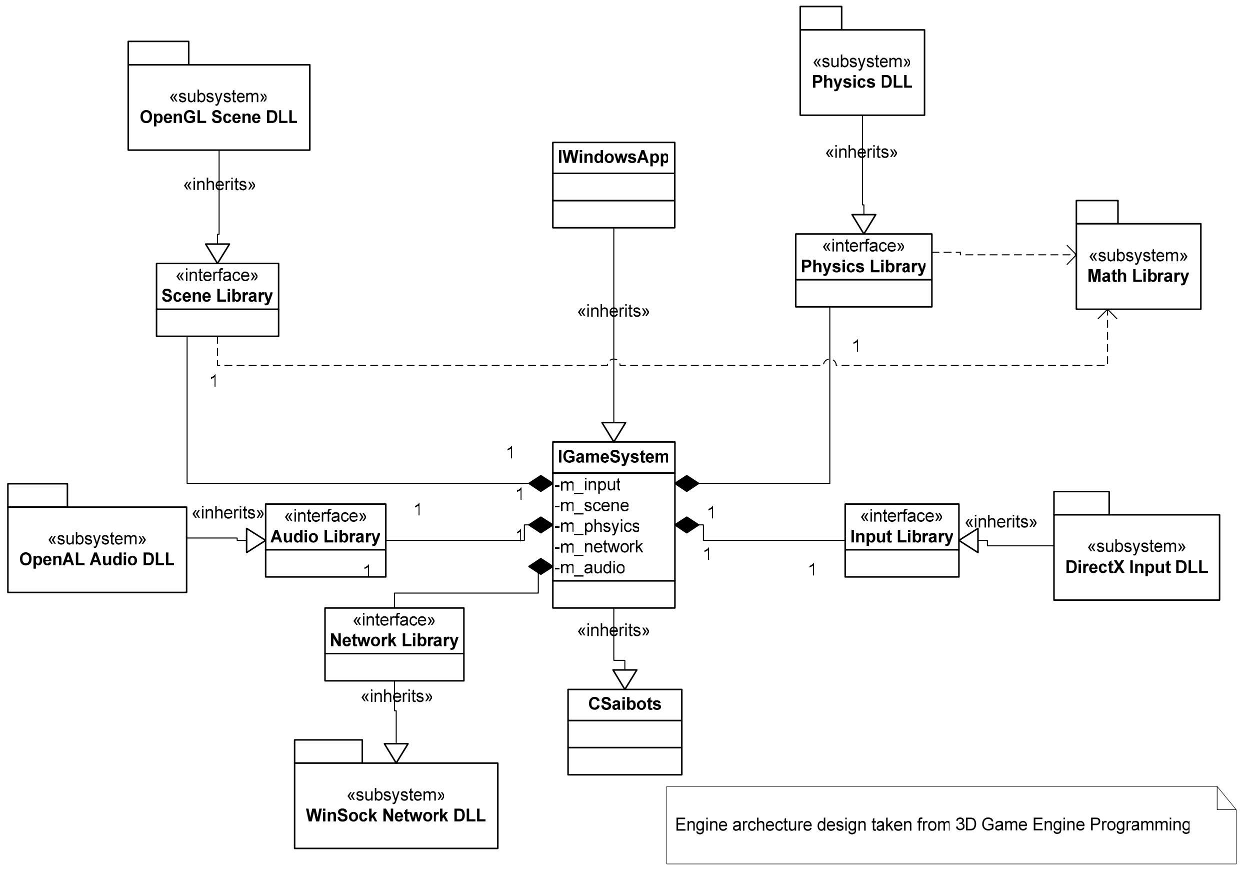

Figure 8. SAI-BOTS system architecture

5.1 Class Diagram

A class diagram of SAI-BOTS, showing the modularization of the system into object classes is presented in Figure 9. The diagram also includes details of relationships, multiplicity constraints, and visibility for each class. Complete class and method descriptions can be found at [8].

5.2 System Activity Chart

In order to thoroughly cover the design of SAI-BOTS, various diagrams were created as part of its software model, including system activity charts, state charts, and flow charts. A system activity chart of SAI-BOTS is presented in Figure 10.

6 Current Status and Future Work

The specification and design phases for SAI-BOTS have already been completed. The main part of the game engine has already been completed and allows us to render 3DS models on a terrain. There are four other components currently under development: the AI interface, audio subsystem, network subsystem, and physics and particle engine.

The Audio subsystem uses OpenAL and gives realistic sound to the game. The networking subsystem allows for multiple players to play each other over a local area network. Finally, the physics and particle

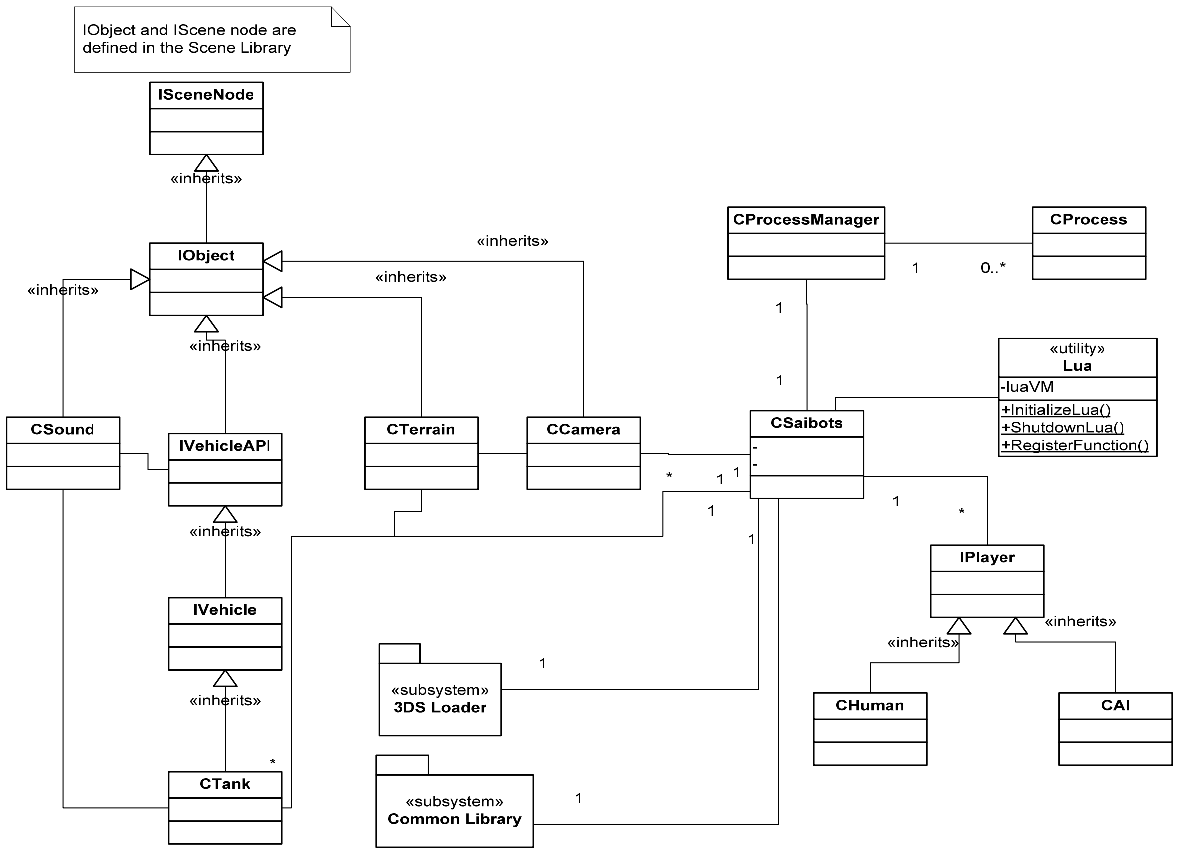

Figure 9. SAI-BOTS class diagram

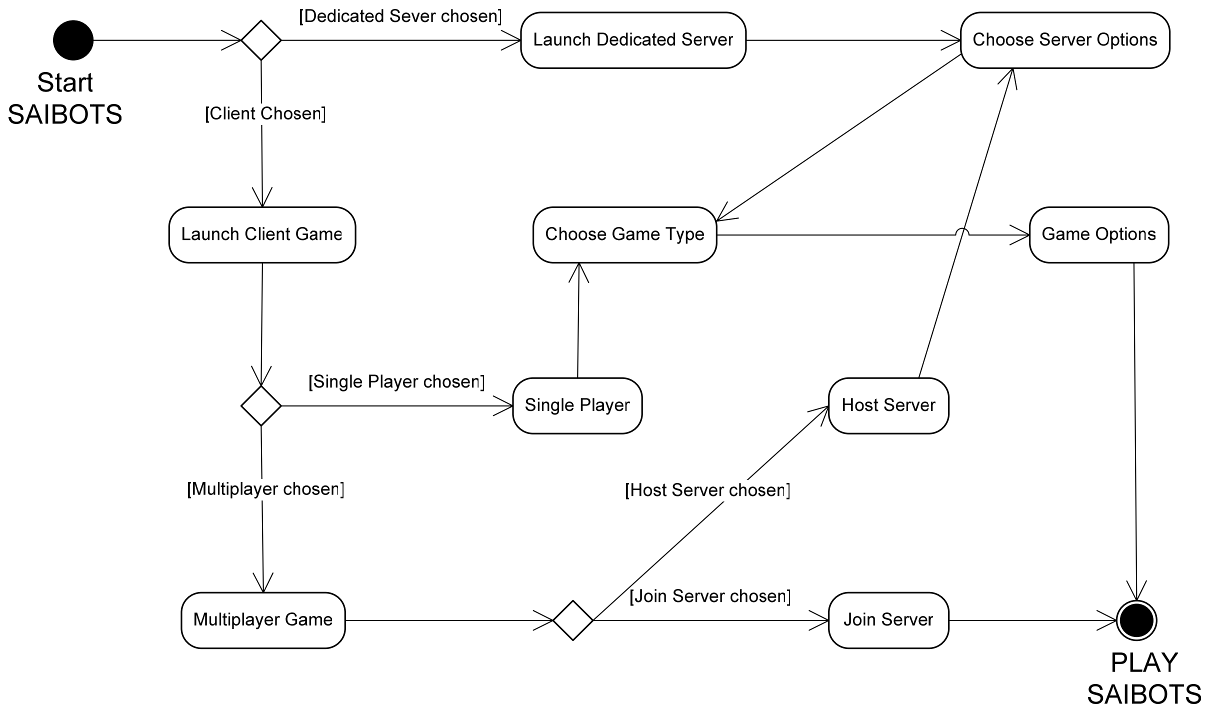

Figure 10. SAI-BOTS system activity chart

engine allows for the tanks to drive across the terrain, collide with trees, fire weapons, be hit, and destroy the terrain. Along with the work that is already completed and currently under development, there are many things we want to add to SAI-BOTS in the future.

Improvements to SAI-BOTS include: more game modes such as Capture-the-Flag, King of the Hill, and a Squad-based mode. In Addition, different vehicles and power-ups such as planes, shields, and new weapons are planned as future work.

7 Conclusion

SAI-BOTS will be a useful artificial intelligence learning tool. The application is presented in the form of an interactive game, letting the user learn artificial intelligence concepts and techniques in an entertaining way. Combining this "easy and fun" philosophy with an expandable and diverse set of game-play modes allows the user to learn and program to a wide variety of situations. By combining a comprehensive and complete vehicle API with a powerful scripting language Lua, able to interface with the applications native to C++, the user will be able to control virtually all aspects of vehicle operation. These operations include but are not limited to: vehicle movement and rotation as well as individual object movement and rotation (where allowed by the specific vehicle). By allowing real-time script modifications users can instantly see the results of a script. SAI-BOTS is a powerful and adaptive learning tool for learning artificial intelligence in an entertaining and interactive way.

References

[1] Zerbst and Düvel, *3D Game Engine Programming*, Muska & Lipman/Premier, 2004.

[2] Ierusalimschy, *Programming In Lua*, R. Ierusalimschy, 2003

[3] Arlow and Neustadt, *UML and the Unified Process: Practicle Object-Oriented Analysis & Design*, Addison-Wesley, 2002.

[4] Enginuity, accessed on January 31st, 2005 at http://www.gamedev.net

[5] An Introduction to Lua, accessed February 15th, 2005 at http://www.gamedev.net

[6] Real-Time Dynamic Level of Detail Terrain Rendering with ROAM, accessed January 31st, 2005 at http://www.gamasutra.com/features/20000403/turner_01.htm

[7] ROAM Implementation Optimizations, accessed January 31st, 2005 at http://www.flipcode.com

[8] SAI-BOT Project, accessed February 28th, 2005 at http://www.cse.unr.edu/~jdp

[9] Karel J. Robot, accessed March 31st, 2005 at http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html