Scripted Artificially Intelligent Basic Online Tactical Simulation

Jesse D. Phillips[+*] Roger V. Hoang[+*]
Joseph D. Mahsman[+*] Matthew R. Sgambati[+*] Xiaolu Zhang[+]
Sergiu M. Dascalu[+] Frederick C. Harris, Jr.[+*]

Department of Computer Science and Engineering[+]
University of Nevada, Reno
Reno, NV 89557

CAVCaM[*]
Desert Research Institute
Reno, NV 89512

{jdp, hoangr, mahsman, sgambati, zhang, dascalus, Fred.Harris}@cse.unr.edu

Abstract

For many years, introductory Computer Science courses have followed the same teaching paradigms. These paradigms utilize only simple console windows; more interactive approaches to Computer Science education are possible. Just as scientific visualization aides the researcher in drawing new conclusions from data, a visual tool that allows the visualization of the execution of code and the effects of that execution would, at the very least, aide understanding of the concepts and, at most, provide for better retention of the concepts. This paper presents details of the idea, specification, design, and functionality of the Scripted Artificially Intelligent Basic Online Teaching Simulator (SAI-BOTS), an interactive game that helps reinforce what is taught in class. Students script vehicles to fight each other in a three dimensional environment. In this environment users can play with other people and learn the basic programming techniques involved in artificial intelligence. The users navigate through the 3D world using a third-person camera or blind mode. This allows the user to take techniques from lecture and apply them, resulting in strong and immediate reinforcement of skills and concepts.

Keywords: scripting, AI

1 Introduction

Interactive teaching techniques have long since been acknowledged as a highly effective teaching method in today's educational landscape. For many years introductory Computer Science courses have followed the same teaching paradigms. However, in a field as inherently innovative as Computer Science, we feel that a great deal of improvement can be made in the way it is taught. Typical problems experienced by beginning programming students involve the aridity of the material, and frustration when a complex program fails to perform as desired. With that in mind, this paper presents details of the idea, specification, design, and functionality of the Scripted Artificially Intelligent Basic Online Teaching Simulator.

SAI-BOTS is an interactive video game in which users can write scripts to control vehicles in a three dimensional environment. It can be used by both beginning and advanced programmers, although its main utility lies in reinforcing the concepts taught in programming classes. Students can immediately use their newly learned skills and concepts in creative ways, and observe the results of their own programming reflected on screen. Interaction is also possible through cooperation or competition between vehicles scripted by multiple programmers. The advantage of this approach to teaching lies in its ability to teach basic programming techniques, and artificial intelligence scripting in an immersive and interesting manner.

We intend for SAI-BOTS to be not only a useful educational tool, but also a program that can be used in the budding area of neuroevolutionary research in artificial intelligence. Neural networks may be used to evolve more efficient and advanced scripts, and the user may write their own fitness functions with in the program. The implementation of Blind Mode helps users develop their skills by giving them only computer interpretable information with which to write their scripts. The ability to control both individual entities and entire squads of entities lends depth to the tactical side of the game.

The remainder of this paper is structured as follows: Section 2 delves into the aspects of video games that influenced SAI-BOTS, and previous work on the synthesis of games and teaching tools. Section 3 presents our design and implementation of SAI-BOTS. Section 4 details our prototype of the product and the inner workings of the code and presents the results of our experimentation. Section 5 concludes with our results and potential developments.

2 Background

Once the decision was made to create a game, we derived inspiration for the features we intended to incorporate from three game engines: APOCALYX [2], CryENGINE2 [5], and Unreal Engine 3 [9]. Some key features include collision detection, pathfinding algorithms [13], and networking subsystems for multiplayer support. Other features we wanted to integrate into our software, but which existing game engines do not support well, included team based AI and the ability to edit scripts on the fly. Finally, the Unreal Engine 3 provides a well-rounded user interface editor and extensive content creation capabilities which inspired us in the creation of our own interface.

An existing game that comes close to our vision is called GUN-TACTYX [3], developed by the creator of the APOCALYX engine. The premise of GUN-TACTYX is for the player to write scripts that dictate the actions of the bots in the game. In writing these scripts the player has access to sensor data within the game, and also basic functions that enable features such as pathfinding. The player is allowed to change a few parameters during the execution of the game, however they cannot change the scripts themselves.

Another existing game that provided inspiration for features was Crysis [8], developed in unison with CryENGINE2 by Crytek [6]. The basic terrain used in Crysis is a heightmap system that determines the elevation of the world at certain points. Along with the heightmap, there is an advanced voxel system that allows for the creation of both interior areas and unusual shapes that may be placed in the world. This allows for efficient creation of extremely detailed worlds in which players of the game may engage in combat. The highly detailed nature of the world allows for the user to get immersed in the game while also trying to perform simple manuevers, such as taking cover behind a building. This type of world can, unfortunately, provide increased difficulty in designing a sophisticated artificial intelligence scheme. The artificial intelligence in Crysis was completely designed to be as realistic and believable as possible. By using some pathfinding algorithms to get around the world, the artificial intelligence also uses tactical manuevers to attack enemies. Working as squads to flank enemies as well as hiding in the world to ambush someone is a common element Crytek was seeking to acheive. When not in combat, the artificial intelligence soldiers exhibit scripted life-like behavior including smoking, yawning, talking, and a few other actions. The user interface that Crysis uses is the base version built into CryENGINE2.

Currently, most artificial intelligence is scripted for the sake of simplicity. The benefits of using a scripting language are the ability to change how the artificial intelligence reacts and the simplification of not needing to recompile the engine source code after modifications. Unfortunately, most games implement a scripting system in such a way that scripts are not able to be updated while playing the game. Updating scripts is a beneficial feature. Since scripts do not need to be compiled, this allows for users to see what new changes to the scripts do while playing the game.

Since SAI-BOTS is intended primarily to be a teaching tool in the same vein as Alice [4], certain features normally associated with video games are not necessary, although in the future they may be implemented. Instead of expending resources in the development of immersive visual and audio detail, the bulk of our work revolved around the creation of a flexible, and powerful script creation interface. The user should be allowed access to the resources they need to develop and evolve their bots. The field of neuroevolution is a growing research field within artificial intelligence. The impressive results of the techniques involved in neuroevolution provided a vision for some of the best features that should ultimately be implemented in SAI-BOTS. Although these techiques are not yet utilized, the goal is to eventually incoroporate them.

Neuroevolution uses genetic algorithms to evolve artificial neural networks rather than having a person expend the time to manually train the artificial intelligence. Genetic algorithms use a fitness function to optimize the output of the neural networks based on what the user wants to accomplish. This idea allows for trainable agents that learn what they are supposed to do faster through offline simulations. The simulations save the highest rated neural networks so the user can test them out in for future use. Some systems that use neuroevolution include ANNEvolve [1], an artificial neural net engine that has been used to train virtual sailboat navigation systems, and NERO [12], a game which allows the user to modify parameters of their own AI agent to combat other neuroevolving agents.

3 Design

Unlike typical games, SAI-BOTS does not revolve around wanton destruction. The ability of a player to load scripts in order to control the behavior of entities in the game makes it an ideal educational tool for artifical intelligence courses. A simple use of the game would be for instructors to arrange competitions in which students on opposing teams write scripts. Each script controls a team of robots, and a subsequent battle between the teams would determine which group of students wrote the most intelligent script. Realizing this vision required the following items to be implemented.

3.1 Squad Control

Fundamental to multiplayer games is the ability to communicate and coordinate maneuvers with teammates. In a typical tactical game a player may send simple commands, and allies are allowed a limited number of options in response. We propose that users be allowed to send commands to other units. Commands may be broadcast to an entire team, over specific team channels, or vocalized so that units in the vicinity may hear them. Certain units may be designated solely as receivers that can act on the intelligence provided. Broadcast commands can be used to form squads. This adds depth to the tactical side of the game and also adds complexity to the design. We have decided to allow the user to directly control any of their vehicles. This will expedite the response should a vehicle encounter a problematic situation such as an enemy encounter; it is also useful in a situation requiring precision, such as maze navigation.

3.2 Neuroevolution

This growing area of research within the artificial intelligence field uses genetic algorithms to evolve artificial neural networks. Genetic algorithms use a fitness function to optimize the output of the neural network based on the goals of the user. The highest-rated neural networks are saved so that the user may test their efficacy against other neural networks in the future, thereby evolving even better systems. Neuroevolution may be used for creating a plethora of advanced artificial intelligences; however, we found that previous game engines were typically limited by the use of only one fitness function. Our idea is to enable developers to write fitness functions in Python, thereby allowing a greater amount of ideas to be developed, as well as enhancing the ability to evolve certain units as commanders, and others as operatives.

3.3 Sensors

Because neural networks take their input from sensor data, the use of neural networks to control the vehicles in SAI-BOTS necessitates the implementation of various sensors. Basic sensors detect the vehicle's distance from walls, allies, and adversaries; simpler sensors that relate to radar are also an option. With access to sensor data, users now wield more tools with which to evolve their scripts. The user is also allowed direct access to all the sensor input, to better decide how he wishes the script to react.

3.4 Scripting

The advantage of using scripts lies in their ease of use. We have chosen Python as the language of our implementation. Since scripts written in Python do not require compilation, the effects of minute changes in a script can be immediately evinced. The user garners feedback quickly as they are playing the game. Our cross-platform user interface allows the programmer to change the artificial intelligence of the application without requiring them to exit and recompile the program, thereby preserving their immersive game experience. Scripts are also flexible in that they may be as simple or complex as the user chooses them to be - entire games may be written using scripts. An additional feature of ours which most scriptable engines do not allow is the ability to control an entity themselves. This feature will lend the user a deeper understanding of how the entity is controlled, and improve their scripting skills.

3.5 Blind Mode

In Blind Mode the user is shown only a small radar and sensors - in other words they are given no information beyond that which is interpretable by the computer itself. Although this seems like a great limitation, it is entirely possible that, for instance, a script may be written which will navigate a maze knowing only the distance of the object from the walls. This will enhance the user's ability to write scripts based on information provided by the sensors. Due to the complexity of implementing more sensors to each vehicle we are currently developing more efficient ways of mapping access to all sensors.

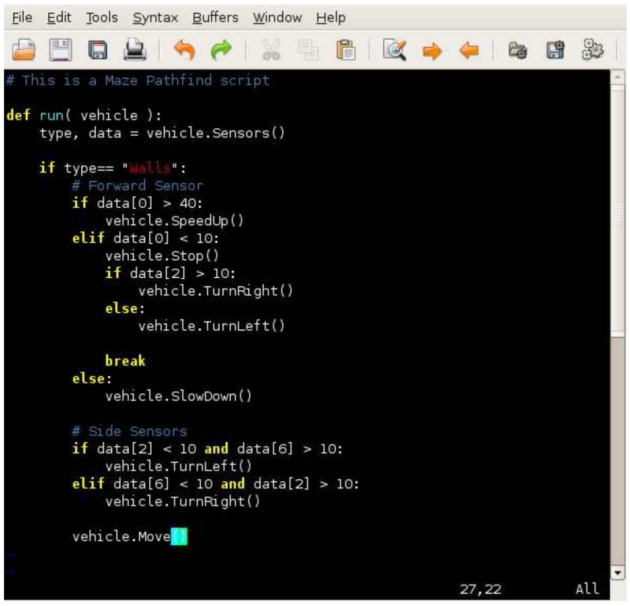
3.6 Terrain

There are two different types of terrain in video games. Deformable terrain is typically used when the environment is small, and allows the terrain to be changed or destroyed. Level of Detail terrain, such as the ones described in [7] and [10], is usually employed in large environments, and is unchangeable. SAI-BOTS supports two different types of Level of Detail terrain. The first is created from a simple black and white bitmap of a maze, the second is created from reading in a digital elevation model of real world data.

Both terrains are generated using a memory efficient rendering method. First, data is read into a vertex buffer object. Indices for the data to be rendered are then set, and a 256 x 256 grid is built, with extra points on one side. In the rendering process the index array is loaded and rotated so that extra side points are created. This is to allow for different levels of detail in the terrain to be merged. In the interest of speed, the rendering is executed entirely on the GPU.

4 Implementation

A prototype of SAI-BOTS was developed using Python as both the implementation and the scripting language. The user interface was created using wxWidgets [14] and is divided into two primary components: a scripting interface and a game client. The scripting interface allows the user to manipulate scripts in a number of text editors (see Figure 1). Once a script is ready for use, it can be executed through this interface; additionally, the interface allows for the user to switch the game client view between an omniscient commander view and a more representative blind mode.



```
# This is a Maze Pathfind script

def run( vehicle ):
    type, data = vehicle.Sensors()

    if type== "Walls":
        # Forward Sensor
        if data[0] > 40:
            vehicle.SpeedUp()
        elif data[0] < 10:
            vehicle.Stop()
            if data[2] > 10:
                vehicle.TurnRight()
            else:
                vehicle.TurnLeft()

            break
        else:
            vehicle.SlowDown()

        # Side Sensors
        if data[2] < 10 and data[6] > 10:
            vehicle.TurnLeft()
        elif data[6] < 10 and data[2] > 10:
            vehicle.TurnRight()

        vehicle.Move()
```

Figure 1: An example script.

Figure 2 depicts the possible paths a user can take while running SAI-BOTS. One path allows the user to start a single player game, in which he can enter a maze game or a combat game. In the maze game the player must navigate the tank through the maze, while in a combat game he faces off against an AI opponent. The other path is to start a multiplayer game where

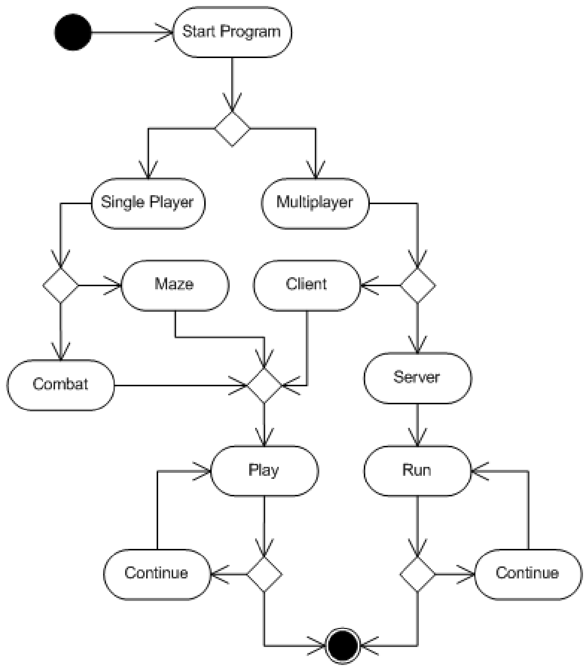


Figure 2: System activity diagram.

he can start either a client version or a server version.

The game client drives the simulation and visualization of SAI-BOTS in a separate window. The basic layout of the client system is shown in Figure 3. User-created scripts are given access to an instance of the Vehicle class. During any invocation, the script is able to move the vehicle, fire its weapons, and communicate with other entities through a message-passing interface. To allow for experimentation with ad hoc networking and information sharing, the broadcast range for messages can be adjusted.

Depending on the selected client view, the user is presented with different representations of the game. Using the commander mode, a 3D rendering of the game world is displayed with OpenGL. Visualization of the maze terrain is done by loading a maze image such as the one in Figure 4 as a heightfield texture and using it as input to a GPU-accelerated LOD rendering system (see Figure 5). This mode allows free reign over the camera system, giving the user the ability to view any part of the battlefield at any time. In contrast, blind mode presents the user with a minimalistic representation of the game world. The clairvoyance of the commander mode is replaced with a visualization more akin to what the scripted entity would see: simple sensor locations and numerical values.

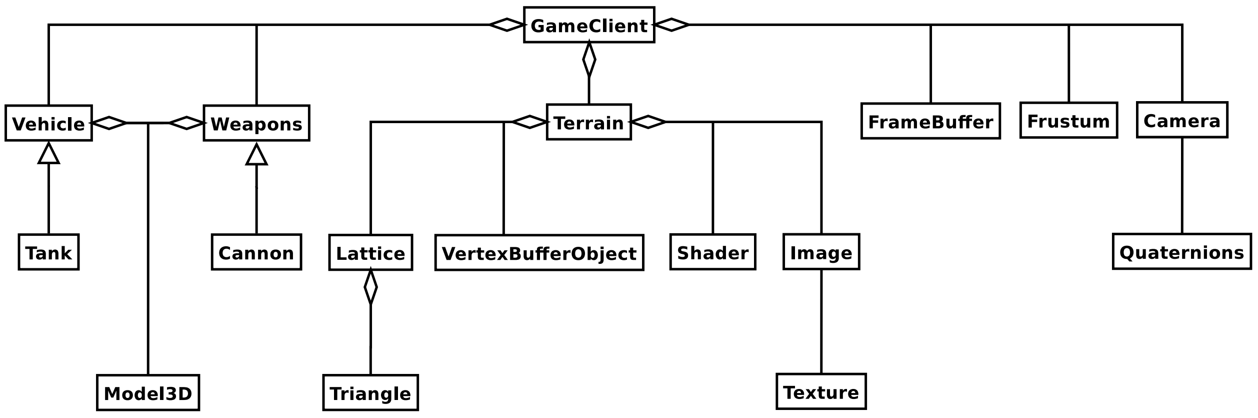


Figure 3: Game client structure.

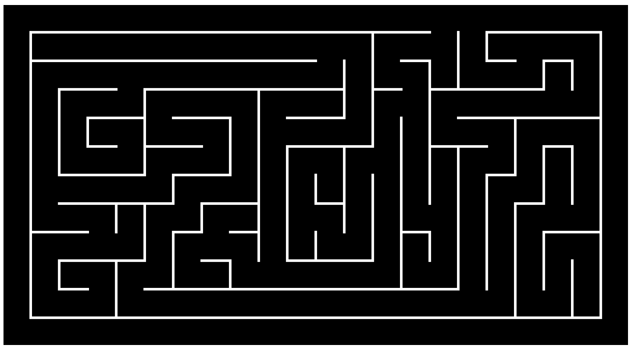


Figure 4: A maze created by MazeMaker [11].

5 Conclusions and Future Work

5.1 Conclusions

Due to the success and entertainment value of video games in today's society, it was inevitable that video games would be used to help teach students Computer Science concepts. In this paper we discussed elements in current video games and engines and identified a gap in the bridge between education and video games.

We presented a prototype of SAI-BOTS, a scriptable game in which the intelligence governing the entities in the game is immediately mutable. Our prototype allows for scripts to be loaded during execution, making it applicable for teaching. It provides the functionality to allow for customized fitness functions to be used, creating a more evolved AI compared to a static fitness function. The squad control allows for the improvement of network programming skills and more complex AI schemes. Introducing various types of sensors presents a neuroevolution problem in determining which sensors are of the greatest use to the AI.

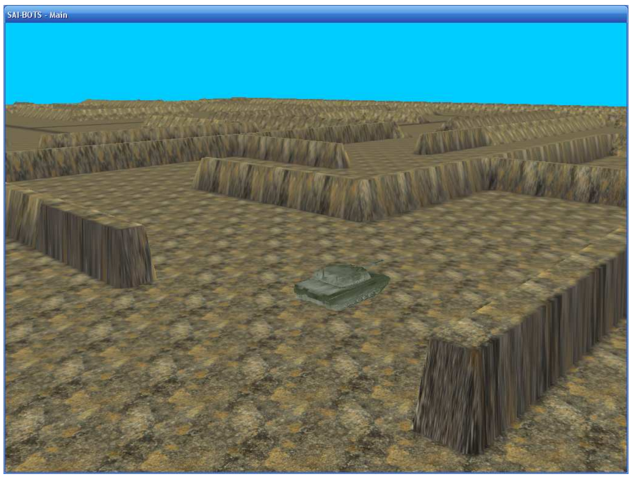


Figure 5: Maze rendered in commander mode.

5.2 Future Work

Although SAI-BOTS presents a proof-of-concept, several improvements must be made in order to make this an effective teaching tool. To begin with, as SAI-BOTS was implemented in Python, the speed of the engine itself can be considerably increased through the use of a compiled language such as C or C++. Python or any other interpretted language can still be used as a scripting language for SAI-BOTS, as long as an interface exists for the script to access game elements such as the vehicles.

In addition to the maze walls, other obstacles such as buildings and chasms should be added. This will not only introduce new pathing and line-of-sight problems but also allow AI programs to possibly evolve sophisticated strategies that take advantage of these features. These improvements can be further enhanced by causing entity actions to actively modify the game world. Blasting craters into the terrain can not only make it difficult for other vehicles to traverse the game world but also provide tactical hiding points. The dynamic nature of the world will also test an AI's ability to adapt to a range of variable situations.

To further increase the potential complexity of the user AI, the ability to script the behaviors of the sensors can be added. Limitations added to the sensors such as broadcasting power and limited energy resources present new challenges like determining the importance of particular information and creating efficient communication pathways to relay this data to the appropriate entities.

From an educator's stance, several pieces of functionality would better facilitate SAI-BOTS use in a classroom setting. One of these would be an observer mode that would allow spectators to study a game without actively participating in the event. Additionally, a replay feature would allow the instructor to critique the effectiveness of students scripts.

References

[1] ANNEvolve. ANNEvolve :: Evolution of Artificial Neural Networks. `http://annevolve.sourceforge.net/`. Accessed March 10th, 2008.

[2] Leonardo Boselli. APOCALYX. `http://apocalyx.sourceforge.net/`. Accessed March 10th, 2008.

[3] Leonardo Boselli. GUN-TACTYX. `http://gameprog.it/hosted/guntactyx/`. Accessed March 10th, 2008.

[4] S. Cooper, W. Dann, and R. Pausch. Teaching objects first in introductory computer science, 2003.

[5] CRYTEK. CryEngine 2 and Sandbox 2 Tutorials. `http://www.cryengine2.com/`. Accessed March 10th, 2008.

[6] CRYTEK. Welcome to Crytek. `http://www.crytek.com/`. Accessed March 10th, 2008.

[7] E. Danovaro, L. De Floriani, E. Puppo, and H. Samet. Multi-resolution out-of-core modeling of terrain and geological data. In *In Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems*, pages 200–209, Bremen, Germany, November 2005.

[8] EA and CRYTEK. EA : Crysis. `http://www.ea.com/crysis/`. Accessed March 10th, 2008.

[9] Epic Games. Powered By Unreal Technology. `http://www.unrealtechnology.com/html/technology/ue30.shtml`. Accessed March 10th, 2008.

[10] William E. Brandstetter III. Multi-Resolution Deformation in Out-of-Core Terrain Rendering. Master's thesis, University of Nevada, Reno , December 2007.

[11] John Lauro. Maze Maker. `http://hereandabove.com/maze/mazeorig.form.html`. Accessed March 10th, 2008.

[12] Austin University of Texas. nerogame.org. `http://nerogame.org`. Accessed March 10th, 2008.

[13] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley, third edition, 1993.

[14] wxWidgets. wxWidgets. `http://www.wxwidgets.org/`. Accessed March 10th, 2008.