G-Bots: Intelligent Agents in a Complex Simulated Environment

Eman M. EL-SHEIKH
Department of Computer Science, University of West Florida
Pensacola, FL 32514, USA

and

Joel R. BECKER
Department of Computer Science, University of West Florida
Pensacola, FL 32514, USA

ABSTRACT

The G-Bots project focuses on the development of an intelligent agent-based architecture for a real-time simulated environment. The agent architecture interacts with a Newtonian physics-based environment, given a set of sensors and actuators. The environment consists of small circular planets and other objects in two-dimensional space. In each game episode, each intelligent agent must try to eliminate the others by causing objects, such as rocks, to hit them, while minimizing hits to itself. After receiving a certain number of hits, an agent is eliminated from the episode. An agent can perform actions such as driving on a planet's surface, picking up an object, throwing an object, and launching itself at a given angle and velocity. The planets exert conflicting gravitational pulls that the agents must account for in their predictions of object motion. Artificial neural networks (ANNs) will be used to improve the agents' performance, including better aiming accuracy and trajectory prediction.

Keywords: Artificial intelligence, intelligent systems, agent-based architectures, simulation and modeling, interactive game environments.

1. INTRODUCTION

This article describes the development of an intelligent agent-based architecture for a real-time simulated environment. The environment is unbounded two-dimensional space containing circular planets, and other objects. The planets exert gravity on other objects. The agent architecture controls multiple simulated robots in this environment. These G-Bots interact with the environment using a set of sensors and actuators.

The goal of the G-Bots project is twofold. The first goal is to create a generic real-time environment simulator that allows physical interaction between objects of various shapes in two-dimensional space, as well as the presence of multiple sources of gravity. The second goal, which is the greater focus of the project, is to experiment with creating intelligent agents that can operate sufficiently in such an environment, particularly to be able to tackle the challenges brought by the presence of multiple gravitational bodies.

The performance of the agents is tested within the framework of a game. In each game episode, each intelligent agent's goal is to eliminate other agents by causing objects, such as rocks, to hit them, while minimizing hits to itself. An agent can perform actions such as driving on a planet's surface, picking up an object, throwing an object, and launching itself at a given angle and velocity. The planets exert conflicting gravitational pulls that the agents must account for in their predictions of object motion. Artificial neural networks will be used to improve the agents' performance, including better aiming accuracy and trajectory prediction.

The rest of this article describes the development and use of the agent-based architecture and environment simulator. The next section provides a brief review of related literature and section three presents an overview of the system. Section four describes the environment simulator and section five describes the intelligent agent-based architecture. The article concludes with the results and potential uses of this project and plans for future work.

2. LITERATURE REVIEW

Since rather recently, much work is being done to do for the AI of simulations and games what OpenGL and DirectX did for graphics: to create application middleware that provide fundamental AI components, and to define a standard interface for such a middleware. Some of these typical AI components include finite state

machines, goal-oriented action planning (GOAP), steering, and path finding. The Artificial Intelligence Interface Standards Committee (AIISC), organized in 2002 by the International Game Developers Association, is leading the standardization of AI middleware interfaces [1]. Although this will be helpful for agents in typical virtual environments, it will not be able to meet the needs of virtual environment AI nearly as widely as has OpenGL and DirectX were able to for 3D rendering.

The AI requirements for the G-Bots environment cannot use the traditional implementations that would be provided by these middleware for path finding, steering, or even prediction of object motion, due to the non-uniform gravitational field. Conversely, the G-Bots project's implementation of these AI components will not be useful to enough applications to justify designing them with the new standard interfaces. The G-Bots project introduces a unique environment that will demand more from the typical tasks of virtual environment AI.

The environment and the agents" actuators resemble a game application. While AI techniques utilized by commercial game applications have been getting increasingly complex, they still only overlap the AI techniques of academia in a limited way, as observed in [2, 3]. However, using computer game environments for AI research has become more common [4, 5]. This is a rather logical progression, considering that the modern game environment is essentially a simulation of some real-world environment. Such a simulation is, of course, ideal for testing AI techniques for robots that act in such environments.

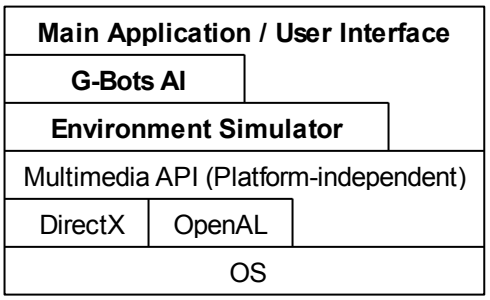

Figure 1: G-Bots system architecture. Each component directly interacts only with those

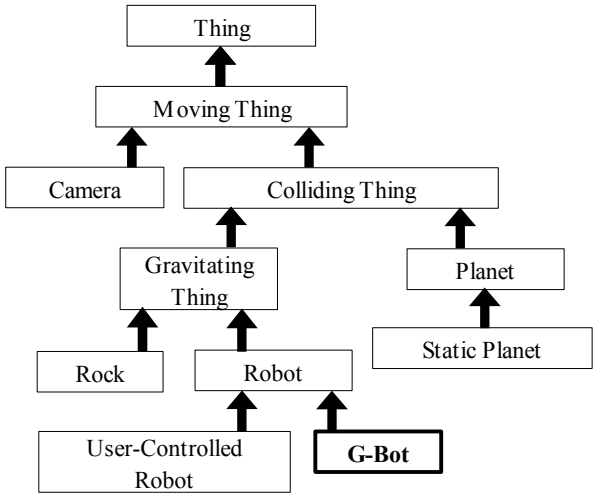

Figure 2: Virtual environment object hierarchy

3. SYSTEM OVERVIEW

Two main components comprise the system: the virtual environment simulator, and the G-Bots intelligent agent-based architecture. The main application integrates these two components to provide interfacing and AI testing for the system. The project was implemented in C++ and compiled for the Windows platform with Microsoft Visual Studio 6.0. The system uses a multimedia Application Programming Interface (API) previously created by one of the authors. This API provides the needed functionality for graphics, sound, user input, GUI, and resource management. It is in turn built on the Win32, DirectX, and OpenAL APIs. It provides a platform-independent programming interface so that the programs using it can be built on any other platform that the API is ported to in the future. The system architecture is shown in figure 1.

The real-time simulation system defines a number of generic objects that exist in the virtual environment. The most generic of these is called a *Thing*, which can be a *MovingThing* or a *StaticThing*. A *MovingThing* can be a *CollidingThing*, which in turn could be a *Planet*, or a *GravitatingThing*, which gravitates toward planets. The simulation system's classes of *Thing*'s may be further extended into more specific types of objects, such as rocks and bots. The G-Bot agent architecture does just this, extending the generic *GravitatingThing* into a "thinking" robot. The object hierarchy of the virtual environment is shown in figure 2.

4. THE ENVIRONMENT SIMULATOR

In this section, we describe the development and use of the environment simulator for the system, including the graphical rendering and the Newtonian Physics-based simulation.

4.1. Environment Specification

The environment being modeled is an unbounded two-dimensional space, containing planets and other objects. Each object is circular with a given radius, and is also assigned a material. A material is a set of physical

properties, including mass density, elasticity, kinetic friction, and static friction. Planets exert gravitational force on other objects, but not on each other. Thus, the environment is a type of simplified model of planets in space, where the planets are unrealistically small and close together, yet they exert gravity as if they were full-sized planets. This simplified model allows practical experimentation with a non-unified gravitational field. The gravitational constant remains constant for all program runs. All objects may be rotated. The system handles shapes generically to allow additional support for objects of other shapes, such as convex and concave polygons.

4.2. Rendering

The visible rendered shapes are handled generically and independently from their physical shape. That is, an object may be assigned, for example, a circular "felt" shape, while it is assigned a square "seen" shape. This allows us to render a picture of a circular-looking object on a simple square, which can be rendered with standard graphics hardware as a strip of two triangles. Rendering a circular object on a circle rather than a simple square would require the use of a triangle fan (A circular "seen" object might be rendered as a triangle fan (like a pie sliced into many thin pieces). Rendering all of these triangles, as well as transforming all of their vertices, would be superfluous for small objects, and would slow down the rendering. Thus, allowing circular objects to be rendered as squares not only increases flexibility, but also improves rendering performance.

The system renders the environment from the perspective of a camera object. The camera object is not visible, and has a position, rotation angle, and view depth (or "zoom"), which can be controlled from a keyboard, mouse, or game pad. The system runs at a fixed-frame rate when running simulations in real-time. A fixed-frame rate simplifies a number of calculations for the Physics simulation and Artificial Intelligence implementation.

4.3. Physics Simulation

Moving objects may be assigned position, velocity, acceleration, angle, angular velocity, and angular acceleration. In addition, torque and directional force may be exerted on an object at a given moment. The simulator operates in discrete time steps of equal length (thirty frames per second). Object orientations, velocities, and other changing quantities are updated each time step. Mass and inertia are assigned automatically to each object at object initialization, according to the material type and physical shape assigned to it.

The objects interact physically through collision detection and handling. A collision is detected by checking each pair of objects each time step for overlap of their physical shapes. When a collision between two objects is detected, a reaction is simulated. The calculation of the objects' new movements as a result of the collision is based on classic Newtonian physics. It accounts for point of contact, angle of tangent surfaces, resulting bounce velocities and spins, mass, inertia, and surface friction. The result is a realistic simulation of the physical world.

Due to gravity, an object may end up being on top of another object. To prevent the collision handling system from dealing with what it perceives as continual collisions at every frame between two such objects, a special state of physical contact between the objects is recorded, along with information on the nature of their contact. This also allowed the simulation of objects rolling along the ground. A rock that is initially in space, for example, will fall to a nearby planet's surface, bounce a few times picking up spin due to surface friction at contact, and then roll to a stop. Certain objects, such as the G-Bots, have stabilizing mechanisms that prevent them from such rolling and flipping upon landing. For this, a flag may be set on the object that indicates such an ability.

5. THE INTELLIGENT AGENT ARCHITECTURE

The environment simulator provides a highly interactive, complex environment for intelligent agent experimentation. One of the most notable complexities is that of the conflicting gravitational pulls from multiple planets. The environment is not realistic in the sense that the planets are very close together, and are extremely small, having a diameter as small as only eight times that of a G-Bot. The goal, though, is not to achieve complete realism, but to create an environment in which more than simple calculations must be used to accomplish physical tasks. The goal of the intelligent agent architecture is to allow the G-Bot agent to successfully accomplish such tasks.

To motivate the tasks and performance measures, the G-Bots play a game. The main goal of the game is to eject the other participants from the environment and be the last to remain. A participant is ejected after k hits from objects thrown by the other players.

The G-Bot is equipped with a number of actuators which allow it to interact with the environment. When on an object, usually a planet's surface, it can drive left or right, and can launch from the planet at a given angle and velocity. The launch actuator must be charged with energy over a period of time before use. The longer it is charged, the higher the launch velocity will be. It has a maximum charge capacity, which is reached after about a full second. While in space, the G-Bot can activate its four-way thrusters to accelerate in any direction. A G-Bot

may also pick up an object that is next to it. To throw the object, it rotates its arms to any angle, and extends its arm back to increase the initial velocity upon throwing. This is similar to the charging and boosting of the launch actuator.

Since it is not the goal of the project to address the challenges of partially observable environments, the G-Bots are equipped with sensors that allow it to perceive the environment as fully observable. It can perceive all object positions and orientations. Also, the G-Bots all use the same intelligent agent architecture and knowledge base.

5.1. Goal-Based Approach

During the first stage of design and development, the authors anticipated the need for a goal-based agent architecture for the G-Bot agents. This would provide the agent with its main goal, and the prerequisites to meeting that goal. All sub-goals needed to meet those prerequisites would also be defined, along with their respective prerequisites. The agent then considers the environment state, and plans a sequence of sub-goals that

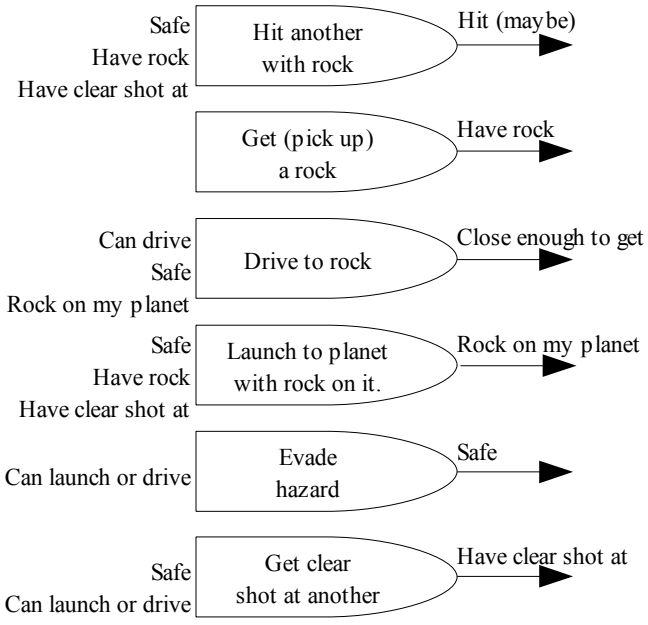

Figure 3. Goals and their prerequisites and results

will ultimately lead to fulfilling the main goal. The goals and their prerequisites and resulting actions are shown in figure 3.

5.2. Reduction to Extended State Machine

Most goal-based agent applications include goals with multiple outcomes. It is evident from figure 3, though, that the game played by the G-Bots generates goals with

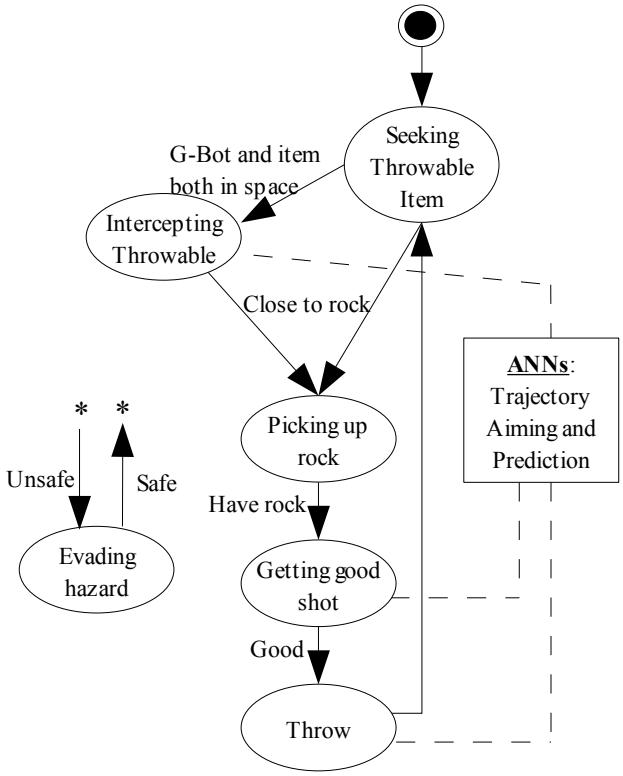

Figure 4: Extended state machine

only one state outcome. Furthermore, it results in the same continual sequence of actions, with the exception of evading a hazard. Thus it may be reduced to an extended state machine, with a sub-state for evading hazard, as illustrated in figure 4. If an object thrown by another agent looks like it will intersect the G-Bot's current path, then it swithes to the “evading hazard” state until it is no longer at risk. This condition is checked continually during any state.

The bulk of the AI strategies are embedded in the various services employed by certain states, such as finding the nearest item for throwing, or predicting/choosing a point at which such an item may be intercepted in its path through space. Using a finite state machine has the advantage of simplicity, and also of faster performance than other methods as [6] points out.

5.3. Uses for the Artificial Neural Networks

The agent architecture allows the G-Bots to play the game quite well. As expected however, they do not aim well nor predict trajectory movement well in the midst of the complex gravitational field. Accurately predicting a trajectory in this environment cannot be solved in a closed-form equation. There are also other variables that need to be considered, such as air resistance and obstruction by the planets. A practical solution would be

to train and use a set of artificial neural networks (ANNs). Neural networks can be used effectively for a wide range of applications, ranging from pattern recognition to stock price prediction. They are especially useful machine learning tools because of their robustness, fault tolerance, and universal function approximation capability [7].

We are currently working on developing and utilizing the ANNs. A neural network will be trained on choosing an angle and initial velocity for an object, given a target location and each planet's position, mass, and radius. A second will be trained to predict an item's position at a given time, given its current position, velocity, and acceleration, and once again each planet's position, mass, and radius. The neural networks will be tested with different data samples to determine the most effective settings for proper aiming of objects and prediction of trajectory movement.

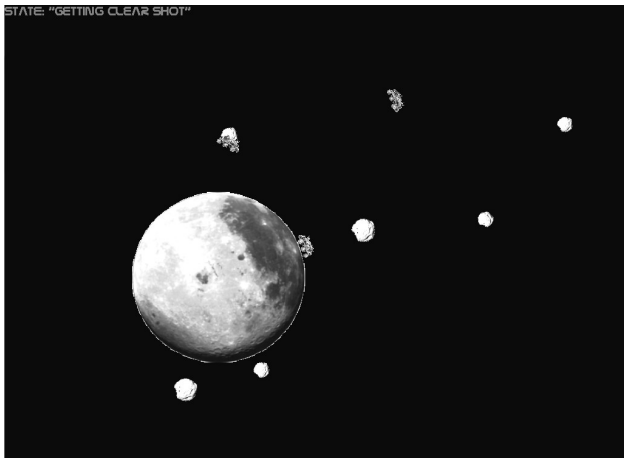

Figure 5. Sample Screen Shot

6. CONCLUSIONS

The G-Bot agents behave quite successfully in their environment. A sample screen shot of the application is shown in figure 5. Since all of the agents share the same AI-based implementation techniques, the agents consistently perform at the same level. . Originally, it was expected that a goal-based architecture would be necessary to provide effective decision-making for the environment. However, the problem could be reduced to a simpler state machine, due to the sequential nature of the problem. Continual checks must be made during execution of the current state, though, since there are other agents changing the state of the environment. The evolution of the system emphasizes the importance of identifying the simplest design that solves the problem, a fundamental principle of applying AI techniques to complex problems.

Several enhancements can be made to the simulator to improve robustness. One of these is to handle an object being on two other objects at the same time. Keeping track of only one causes such objects to "jiggle" as they oscillate which object they are on between the two objects underneath it.

This first version of the system will provide a good base agent to test future enhancements against. This will simply require deriving improved versions from the existing G-Bot class of figure 2. It will be of special interest to test the performance of agents with the specialized neural networks.

The system is quite flexible and modular. To allow it to be easily expanded or used for other purposes. For example, the system can be used to test other AI techniques or other physics simulations, since the system can be easily reduced to gravity in only one direction or to zero gravity. Other potential system enhancements include extending the architecture to include a three-dimensional simulator, using the system for a variety of computer games, and incorporating interactive physics for educational use.

7. REFERENCES

[1] The Artificial Intelligence Interface Standards Committee (AIISC), http://www.igda.org/ai/. Accessed: May 20, 2008.

[2] A. Nareyek, "AI in Computer Games," **ACM Queue**, volume 1, issue 10, pp. 58-65, 2004.

[3] D. E. Diller, W. Ferguson, A. M. Leung, B. Benyo and D. Foley, "Behavior Modeling in Commercial Games," Proceedings of 2004 Behavior Representation in Modeling and Simulation (BRIMS) Conference, Arlington, VA, May 17-20, 2004.

[4] J. E. Laird, "Using a Computer Game to Develop Advanced AI," **Computer**, vol. 34, no. 7, pp. 70-75, 2001.

[5] R. Adobbati, A. N. Marshall, A. Scholer, S. Tejada, G. A. Kaminka, S. Schaffer, and C. Sollitto, "Gamebots: A 3D Virtual World Test-Bed for Multi-Agent Research," presented at **Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS**, Montreal, Canada, 2001.

[6] A. Bartish and C. Thevathayan, "BDI Agents for Game Development," **AAMAS 2002**, pp.668-669, 2002.

[7] S. Haykin, **Neural Networks, a Comprehensive Foundation**, Macmillan Publishing Co., 1994.