Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research

Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer, Sheila Tejada
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
{rogelio, amarshal, ascholer, tejada} @isi.edu

Gal Kaminka, Steven Schaffer, Chris Sollitto
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
galk@cs.cmu.edu
{srs3, cs3} @andrew.cmu.edu

ABSTRACT

This paper describes Gamebots, a multi-agent system infrastructure derived from an Internet-based multi-player video game called Unreal Tournament. The game allows characters to be controlled over client-server network connections by feeding sensory information to client players (humans and agents). Unlike other standard test-beds, the Gamebots domain allows both human players and agents, or bots, to play simultaneously; thus providing the opportunity to study human team behavior and to construct agents that play collaboratively with humans. The Gamebots system provides a built-in scripting language giving interested researchers the ability to create their own multi-agent tasks and environments for the simulation.

Keywords

Multiple Agent Systems, Agent Infrastructure, Multi-Agent Teamwork, Human-Agent Interaction, Virtual World Simulation, Video Game Bot.

1. INTRODUCTION

Standard AI test-beds have often been successfully used in the past to promote research, as they facilitate controlled empirical experimentation, comparative evaluations, and quantitative measurements. Problems like chess, and test environments like Phoenix [2] and RoboCup [6], have resulted in significant improvements to the sciences of artificial intelligence and multi-agent systems. Indeed, the usefulness of having a complex, dynamic multi-agent environment as a research infrastructure has been pointed out explicitly in [5] and [3].

However, most complex, dynamic, multi-agent research environments require considerable efforts to build and maintain, and therefore, there is a general scarcity of such infrastructures available for research use. Most existing infrastructures are designed to support specific tasks under a single environment, and rarely support human testing and comparison.

To address these difficulties, we have been developing a new type of research infrastructure. Gamebots is a project started at the University of Southern California's Information Sciences Institute, and jointly developed by Carnegie Mellon University. The project seeks to turn a fast-paced multi-agent interactive computer game into a domain for research in artificial intelligence (AI) and multi-agent systems (MAS). It consists of a commercially developed, complex, and dynamic game engine, which is extended and enhanced to provide the important capabilities required for research.

Gamebots provides several unique opportunities for multi-agent and artificial intelligence research previously unavailable with other proposed test-beds:

- It supports multiple tasks. The system is currently distributed with a series of multi-agent competitive tasks including Capture the Flag and King of the Hill derivatives.
- It can be extended with a built-in scripting language so agents can be faced with multiple, ever-changing tasks, to support continuous long-term research.
- It allows for the creation of multiple environments. In order to appeal to a wider user community we have developed a new environment that has a magical wizards theme (see figure 1).
- It supports humans-as-agents, thus allowing multi-agent researchers to study human problem solving, and investigate scenarios involving human-AI competitions and human-AI collaboration.
- It is publicly available both in the U.S. and overseas. http://www.planetunreal.com/Gamebots

This paper is organized as follows. Section 2 provides background and motivation for Gamebots. Section 3 describes the main design and implementation concepts in Gamebots. Section 4 describes several of our existing clients, and Section 5 concludes.

2. BACKGROUND AND MOTIVATION

Some of the earlier work on MAS infrastructures lead to ModSAF [1], a system for military training based on distributed simulations using computer generated military forces. The software agents, in addition to human participants, made up these forces (fixed or rotary wing pilots, tank drivers, etc.) and had to act in a coordinated fashion that involved team play, mission planning, and reactive behavior. Although commercially developed, this system is not broadly available which has prevented a larger participation by the research community.

More recently, the RoboCup initiative [7] has served as a growing software infrastructure for a wide variety of research in multi-agent systems. The main component of this framework is the RoboCup Soccer Server, a multi-agent environment that supports two teams of simulated soccer agents playing against each other in real time (with each agent running a client connected to the server) [6]. The rules of the game and the main task (score more goals than the opponent) cannot be changed, which can lead to the development of soccer-specific techniques that may not be reusable for other tasks or environments.

The agent "Quakebot" [9] is designed to play the popular first-person shooter video game Quake against human players. This agent (or "bot") is based on Soar [10], and uses dynamical hierarchical task decomposition to organize its knowledge and actions. It incorporates predictive capabilities and learning [8]. The Quakebot agent is currently limited to single agent tasks.

Figure 1: A screenshot of Gamebots' wizards and bubble wands.

At the Gamebots project, we are seeking to turn an interactive multi-player video game into a domain for a variety of research in artificial intelligence. The project aims to construct a new standard test-bed for research in multi-agent systems. When deciding on what particular video game implementation to use for the multi-agent framework, we required that the game be client/server based (allowing users be able to connect remotely) and that it support several players (either human or agent) playing individually or in teams. We needed the game to be easily modifiable in order to add different types of tasks or change the API, and that it have already a large user base. Our choice was a commercial game engine, Epic's Unreal Tournament (UT):

- UT consists of a fast, dynamic, and complex 3D simulation engine, widely available at little or no cost (a stand-alone server -- http://www.unrealtournament.com/downloads), and with a large user base.
- It is a robust environment, stress tested by thousands of people everyday, and under continued support at Epic. It is not uncommon for servers to stay online for extended periods (weeks or even months).
- It includes a broad set of game tasks, such as Domination and Capture the Flag, as well as 35 different world maps varied both in size and semblance [4]. Additionally, the active online gaming community has constantly added to this library with new maps and new game types. This gives researchers a wealth of environments for testing their agents.
- It provides a variety of ways for developing new game types and world objects, primarily through its integrated scripting environment UnrealScript. Agent developers can take advantage of this in order to create new and varied tasks for their agents.
- Unreal Tournament's wide popularity provides a familiar environment for students to explore agents and artificial intelligence research.

3. THE GAMEBOTS SYSTEM

The core of the Gamebots project is a module for UT that allows characters in the game to be controlled via network sockets connected to bot clients (see figure 2). The Gamebots server feeds sensory information for the characters over the network connections. Based on this information, the client (bot or human player) can decide what actions the character should take and issues commands back over the network to the game to have the character move, shoot, or talk. Agents must display advanced AI and MAS capabilities to play successfully, such as planning paths, learning a map of their 3D environment, using resources available to them, coordinating with their teammates, and engaging in strategic planning which takes their adversaries into account. Unlike other standard test-beds, the Gamebots system allows human players to play with the agents, thus providing opportunity to study human team behavior, and to construct agents that play collaboratively with humans.

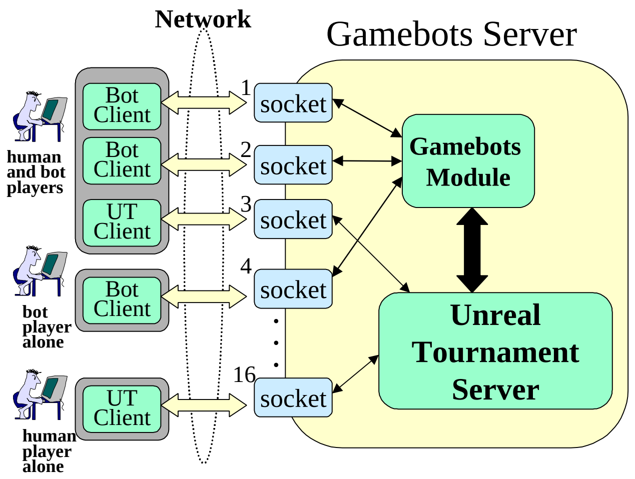


Figure 2: Gamebots architecture (note that human players connect directly to the UT server, and bots connect through the Gamebots Module).

3.1 Unreal Tournament System

Unreal Tournament falls into a category of video games known as first-person shooters, where all real time players (currently a maximum of 32) exist in a 3D virtual world with simulated physics and a variety of tools that give the players additional abilities. As implied by 'first person' in the genre's name, every player's senses are limited by their location, bearings, and occlusion within the virtual world.

Unreal Tournament comes with three core tasks, or game types. The simplest format is a free-for-all or team based game called Deathmatch where every players attempts to accumulate the highest points via 'kills' (bringing opponents health down to zero and thus resetting the player) up to a maximum score or within a time limit. In the game of Domination, players attempt to accumulate time controlling 'domination points' in a king-of-the-hill fashion. Capture the Flag mimics the game by the same name where two or more teams attempt to grab and carry an opponents flag back to the home base. Gamebots includes implementations for all three core game types.

Developers can extend the existing game types, or implement new ones using UnrealScript. UnrealScript is a C++ based scripting language that handles all game logic and object interaction under Unreal Tournament while the main game engine handles the hardcore work like rending scenes and simulating physics. UnrealScript can also be used to build small *mutator* scripts that implement small tweaks, such as adding or replacing world items, adjusting physics parameters, or changing item effects. Such mutators can be added at runtime to any existing game type, including those for Gamebots. While the inherent networked nature of Gamebots does not provide us with perfect repeatability, the built-in scripting language gives very precise control for potential experiment building.

The Unreal Tournament server implementations exist for Windows, Linux, and Macintosh. Additionally, the use of an open network protocol means that bot implementations are not bounded by platform and can often integrate easily with existing intelligence engines.

3.2 Bot Interaction Protocol

The Gamebots interaction protocol is a simple text based protocol of single-line messages sent over the network between the server and bots. The gamebots server sends sensory information messages to the bots containing the current state of the virtual world. The bots interact in the environment by sending action commands or player communication messages back to the server.

3.2.1 Sensory information

The server message loop is composed of three stages, the initial handshake, synchronous messages, and asynchronous messages. During the connection handshake, the server announces the game type and conditions for a win. Once connected, the server will alternate between the synchronous and asynchronous messaging modes. By default, synchronous message blocks are sent roughly ten times per second. This continues until the game ends and/or the connection is dropped.

Messages are composed of the message type with a list of attribute value pairs. Synchronous messages are bounded by one BEG ("begin") and one END message and represent the state of the world for that moment. Included in this block is the current game state such as the score, the game time, and any game specific attributes such as who has a flag. It also includes sensory information, a complete list of everything (items or players) within the agent's view at that moment, as well as the agent's own state (e.g. health and location).

Asynchronous messages occur as events happen between the synchronous messages. These can include immediate sensor updates, such as bumping into a wall, hearing a noise, or beginning to fall after reaching a ledge. Additionally, agents may receive messages for some dynamic and game critical items such as other players and flying projectiles.

Due to the underlying bot implementation in Unreal Tournament, all vision messages simplify a world object down to a set of location and rotation coordinates called a navigation point. All bots do not have a direct sense of the map architecture, such as walls and floors. Instead, the map architecture is conveyed implicitly through the set of navigation points. Navigation points have a "reachable" attribute on all vision messages. A bot may walk directly to any item marked as reachable. Items or points that are occluded to the bot are considered not reachable. Unreal Tournament maps often include no-item navigation points to assist bots in navigating the full extent of a map.

Preliminary tests with our Javabot implementation show that this minimal data is sufficient for navigating the rooms of most maps. Using one of Unreal Tournament's pre-built maps, we have been able to map a majority of the level using cached reachable data.

3.2.2 Bot action commands

Gamebots bots send commands in the same named attribute list format as server messages. Example action commands include:

STOP: stops all bot movement and rotation.

JUMP: causes the bot to jump.

RUNTO: turns towards and move directly to a destination that may be specified via either target or location argument. Target provides the unique id of a player, object, or navigation point (it must be visible when the command is received or the bot will do nothing), whereas location provides an (x,y,z) absolute position.

CHANGEWEAPON: switches to weapon specified as argument (if "best" is sent as argument, the server will pick the bot's best weapon that still has ammunition.)

In order to simplify the network protocol and decision-making, several messages allow attributes to be specified as named objects or targets, such as players, in the world. In these cases, the server handles the action command, like **RUNTO**, using the objects' location and a simple prediction algorithm for target movement, in order to minimize network latency issues.

3.2.3 Player communication

Players (humans and bots) can communicate with each other by having messages posted to either a global or team specific chat channel. Posting a message results in the server sending an asynchronous message to all appropriate listening players. Bots can also take advantage of a few enumerated messages sent by human players, such as "Defend the base", "Hold your position", and "Cover me".

3.3 Research Extensions

In order to turn this fast-paced multi-agent interactive computer game into an infrastructure for research in artificial intelligence (AI) and multi-agent systems (MAS), we have developed several extensions to facilitate research, such as, visualization tools, data collection tools, and standard tasks and environments.

3.3.1 *Visualization tools*

The Gamebots system provides three types of visualization tools for observing and analyzing multi-agent behavior. The first tool is the 3D virtual world itself. Playing as a character in the virtual world with the bots is an effective method for researchers to evaluate, diagnose and test their bots' behaviors. Players can also connect to the virtual world as spectators that can follow the movements of the characters throughout the environment (figure 1). This is another method for observing the bots' behavior without affecting the state of the environment or the behaviors of the players. In this 3D virtual environment, the game can be paused at any point and the simulation speed setting that can be changed.

The two other visualization tools are available from the Gamebots website -- the global and bot VizClients. Both VizClients offer a bird's eye view of the map and display the real-time movements of the bots. The global VizClient (figure 3) shows the activity of all the bots in the environment, as well as the team and individual bots scores and the location of all objects. It also includes graphics for the Stalwart series of world maps.

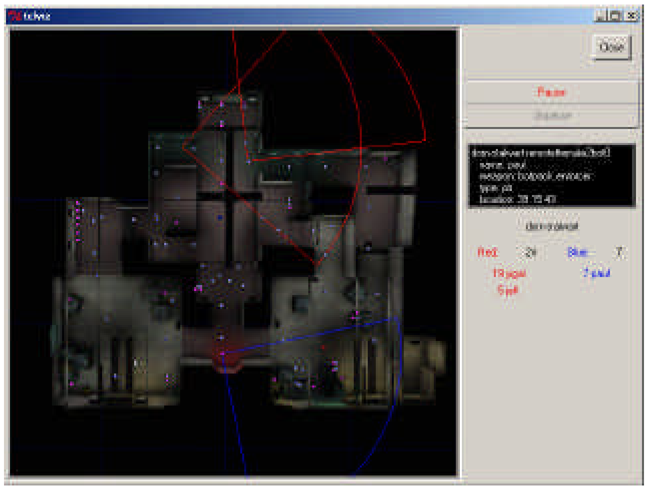
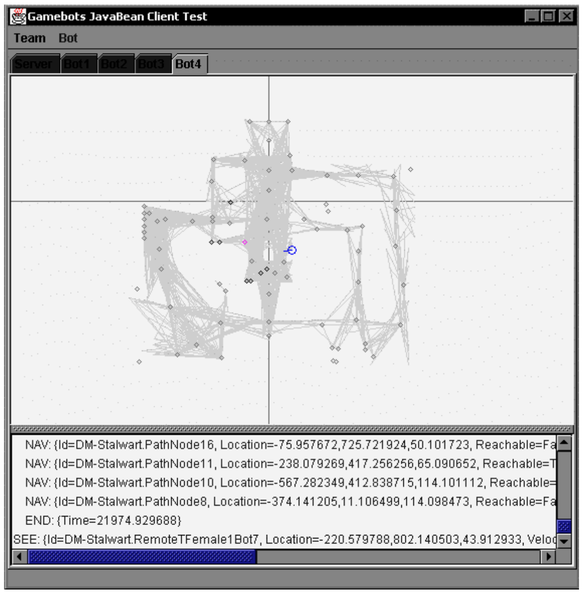


Figure 3: Global VizClient tool

While the global VizClient is used to observe team behavior and strategy, the bot VizClient aids in analyzing the behavior of individual bots. The bot VizClient displays the real-time movement of an individual bot. It draws the path of the bot as it moves in the environment as shown in figure 4, and displays the log of the bot's messages.

3.3.2 *Data collection tools*

Gamebots offers three built-in data collection tools that generate parsable text logs. First, the server-side has a configuration option for outputting all internal events to a log file. In this manner, researchers gain access to every detail of the system. From the bot side, there are two collection tools that log the global or localized views of the world by capturing redirected VizClient messages (figure 5). In addition, the VizClients can be used in conjunction with these logs to replay the bots actions in the game for further evaluation of their behavior.



```
NAV: {Id=DM-Stalwart.PathNode16, Location=-75.957672,725.721924,50.101723, Reachable=Fa
NAV: {Id=DM-Stalwart.PathNode11, Location=-238.079269,417.256256,65.090652, Reachable=T
NAV: {Id=DM-Stalwart.PathNode10, Location=-567.282349,412.838715,114.101112, Reachable=
NAV: {Id=DM-Stalwart.PathNode8, Location=-374.141205,11.106499,114.098473, Reachable=Fa
END: {Time=21974.929688}
SEE: {Id=DM-Stalwart.RemoteTFemale1Bot7, Location=-220.579788,802.140503,43.912933, Veloc
```

Figure 4: Bot VizClient tool

Besides collecting the sensory and action messages, a few task specific statistics are also collected, such as the teams' and bots' individual game scores. We are working on analysis tools for calculating further statistics from the log, like bot aiming accuracy or average bot speed, to aid in the evaluation of bot and team behavior.

Additionally, Unreal Tournament has support to save demo reels, or in-game video files, of the 3D environment for later replay and demonstration. We are currently developing a tool for Gamebots to take advantage of this feature. We would like the tool to actually be an agent that would explore the virtual world to capture the interesting actions and events.

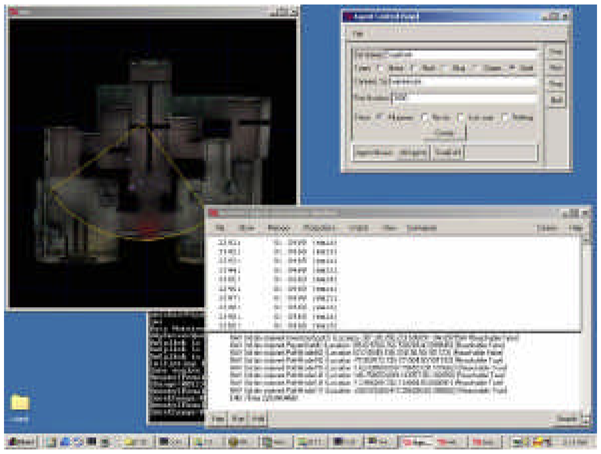


Figure 5: Log generated for SoarBot

3.3.3 *Multiple tasks and environments*

The three game types included in a minimal Gamebots installation provide a natural progression of tasks of increasing difficulty, to facilitate the development of robust and complex agents. Beginning with Deathmatch, agents only need to worry about low level issues such as resource management and enough spatial reasoning to track or avoid someone as necessary. These skills form a basis for the required skills in Domination, where mapping, path planning, and simple teamwork such as coordinating offensive and defensive roles, become important. And lastly, successful teams in Capture the Flag games will require coordinated maneuvers such as providing cover for a friendly flag carrier and managing efficient searches for an enemy flag carrier.

In addition to supporting the original Unreal Tournament environment, we have also developed a magical wizards environment (figure 1). The environment was created not only to demonstrate that the Gamebots system is able to support multiple environments, but also to appeal to a wider user/research community because the magical wizards environment does not include the graphic violence of the original UT environment. Also, since the gaming community is very active in creating new environments, more environments may be added in the future.

3.3.4 Uncertainty in the environment

We have modified the environment to include some noise, or uncertainty, in the bots' weapon aiming accuracy. In the original environment, bots had perfect aiming accuracy. The aiming accuracy is currently a function of the skill level setting for the bot. The bots designated as expert have better aim than the ones that are novices; furthermore, this setting can be changed at run-time. We have future plans for adding similar noise level controls for other bots sensors and actions.

3.3.5 Availability and support

Gamebots is publicly available both in the U.S. and overseas. The research extensions are provided in open-source format over the web at http://www.planetunreal.com/Gamebots. It is supported by an active and growing community of agent developers from around the world. Central to the community is a public mailing list where issues such as agent design and platform development are discussed.

4. EXAMPLE BOT IMPLEMENTATIONS

We have developed sample bots (Javabot and Soarbot) for the Gamebots project. Most of them are currently simple and their functionality is limited; nevertheless, they illustrate how more complex and intelligent agents can be built to communicate with the Gamebots server.

The Javabot includes a bot API, visualization client API, and an application for launching and managing the running clients written to those APIs. Both APIs abstract the network and protocol into a simple message object queue, simplifying the task for agent developers. Both the example bot and example visualization client make use of a top-down mapping library, providing developers with rudimentary feedback.

SoarBot is built with TCL/TK and Soar (figure 5). Soar provides a general cognitive architecture for developing systems that exhibit intelligent behavior [10]. SoarBot is an example of how Gamebots can integrate with existing AI systems for its bot implementations.

In addition to these bots, new bots are in the works both with our group and abroad. There is a TclBot, which is a very simple bot built in TCL. While it does not attempt to show intelligent behavior, it is a well-documented minimal example of a Gamebots agent. The CMU Gamebots team is implementing a new C/C++ based bot; once they have a solid skeletal example, it will be added to the collection of existing bot examples. Moreover, there is a group in England working on a Delphi/C++ bot, and another group working on a PHP-based bot.

5. CONCLUSIONS AND FUTURE WORK

We have described the Gamebots project, an infrastructure for multi-agent systems research that supports different platforms and is widely available. It is built as an environment separate from the agents that allows agents to sense and act in the environment. This supports the common convention of thinking about agents as situated in an environment. Gamebots also allows for human-AI competitions and collaboration, and unlike other complex MAS infrastructures (e.g., RoboCup), it supports multiple tasks and environments.

For future work on the Gamebots project, we are currently developing analysis tools for the server and client logs. In addition, we are continuing to develop our skeletal bot clients and associated libraries, as well as exploring how to scale beyond our current limit of 16 players in the world, to the 32 players supported by UT. Another important future extension is expanding the use of uncertainty in the environment to add different levels and types of noise to the agents' sensory information and actions.

ACKNOWLEDGMENTS

We would like to express our gratitude to Yigal Arens and Paul Rosembloom for their continued support of Gamebots and the use of ISI facilities. We also thank Manuela M. Veloso for her generous support of this project, and key insights on its uses for research. In addition, we are very grateful to Patrick F. Riley and Lauren Lamonica for their creative design contributions. Finally, we would like to thank Epic Software, the authors of Unreal Tournament, for their help and for kindly sharing their expertise with our group.

REFERENCES

[1] Calder, R. B., Smith, J.E., Courtemarche, A.J., Mar, J.M.F., Ceranowicz, A.Z.. ModSAF Behavior Simulation and Control. Proceedings of the Second Conference on Computer Generated Forces and Behavioral Representation, STRICOM-DMSO, July 1993.

[2] Cohen, P.R., Greenberg, M.L., Hart, D.M., and Howe, A.E. Trial by Fire: Understanding the design requirements for agents in complex environments. AI Magazine, 10(3):32-48.

[3] Gasser, L. MAS Infrastructure Definitions, Needs, and Prospects. Working Paper LG-2000-07, 5/12/2000, in Proceedings of the Workshop on Scalable MAS Infrastructure, Barcelona, Spain, 2000.

[4] Gerstmann, J. Unreal Tournament: Action Game of the Year, 1999, GameSpot. http://www.gamespot.com/features/1999/p3_01a.html

[5] Hanks, S., Pollack, M. E., Cohen, P., Benchmarks, Test Beds, Controlled Experimentation, and the Design of Agent Architectures, AI Magazine vol.14, p17-42, 1993

[6] Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., and Asada, M. The RoboCup Synthetic Agent Challenge 97. Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997.

[7] Kitano H., Asada M., Kuniyoshi Y., Noda I., Osawa E. Robocup: The Robot World Cup Initiative, Proceeding of the first International Conference on Autonomous Agents, Marina del Rey, CA, 1997, 340-347.

[8] Laird, J. E. It Knows What You're Going to Do: Adding Anticipation to a Quakebot. AAAI 2000 Spring Symposium Series: Artificial Intelligence and Interactive Entertainment, March 2000: AAAI Technical Report SS00 -02.

[9] Laird, J.E. and van Lent, M. Human-level AI's Killer Application: Interactive Computer Games. AAAI Fall Symposium Technical Report, North Falmouth, Massachusetts, 2000, 80-97.

[10] Rosenbloom, P., Laird, J.E., and Newell, A. The Soar Papers: Readings on Integrated Intelligence, MIT Press, 1993.