Proceedings of the First
Artificial Intelligence and Interactive
Digital Entertainment Conference
(AIIDE-2005)

Quagents: A Game Platform for Intelligent Agents

Chris Brown and **George Ferguson** and **Peter Barnum** and **Bo Hu** and **David Costello**
University of Rochester
Rochester, NY 14627

Abstract

The Quagents system provides a flexible interface to the functionality of a game engine. The goal is to make interactive games a useful research and teaching vehicle for academics. Quagents is freely available, and runs under Unix/Linux, Windows, and Mac OS X. Intelligent agent controllers may be programmed in any language that supports sockets. A communications protocol between the controller and the bot resembles that between the high-level software and low-level controller of a mobile robot. More complex APIs may be built on the protocol that support complex interactions like trading. We describe the Quagent architecture and research and teaching applications.

Motivation

Interactive Digital Entertainment (IDE) will increasingly involve artificial, intelligent agents and related technology. Some of the agents will be managing things behind the scene, while others will be represented explicitly by graphic renditions. Ultimately, hardware embodiments are possible as well. Games and entertainment agents will soon hit a ceiling of performance possible with *ad hoc* approaches to intelligence and will need to incorporate state-of-the-art AI. Luckily future hardware can support the increased demands. Our goal in creating Quagents was to give researchers access to an entertainment-style simulated universe, and thus for IDE to reap the benefits of research carried out in relevant domains.

The *Quagents* system reported here is meant to help bridge the gap between the concepts and interfaces of commercial products and the typically unconstrained approach of academic AI research. It is designed for use in research and teaching. It smoothly supports multiple agents. It is a robotic simulator that uses the underlying physical and graphical capabilities of a commercial game engine. The result is a system that is freely distributable, runs in UNIX and Windows environments, has the look and feel of a first-person game, and in which AI algorithms written in any language that supports sockets can control the agents through a protocol or user-written APIs. One is not restricted to the game-supplied "mods", so any research-grade software can be used for agent control.

This work is inspired by many pioneers who have used and are using computer games to support AI research, and working to import AI into games (e.g. (Buckland 2002)). Quagents are a form of *animat:* that is, a simulated organism (*e.g.*, (Terzopoulos & Rabie 1997)). Quake III has been used as a testbed for neural models (Noelle & Fakhouri 2004).

The main animat advantages are: no hardware, quick prototyping, known ground truth, ability to create wide variety of (physically possible or not) agents, sensors and environments. Laird and co-workers have used the Soar production system to simulate various human reasoning characteristics and skills in a Quake "game AI" (*e.g.*, (Laird 2004)): automating opponents to the human player.

Our system uses the freely-available Quake II engine, which handles all the details of collision detection and environment management so researchers can concentrate on exploring agent behaviors and interactions in a fairly complex world. We had two basic requirements: The "agent" code should build and run under the Linux/Unix operating system regardless of where the game server was running. There must exist a mechanism to build "agents" (that is, their intelligent controllers) external to the game itself so that external resources of the agents would not have to be linked directly into the game. This is a brief version of a technical report (Brown *et al.* 2004). which has many more details and references.

Quake II Basics

UR-Quake is a modified version of Id Software's Quake II (Id 2004). Dynamically, Quake II is divided into a server and a client. Statically, Quake II consists of an executable and two DLLs. The *game engine* executable has both the server and client code. The client part is connected to the server over a network, or locally if on the same machine. The server updates the world state and sends state update network messages to the clients at about 10Hz; The client updates, interpolates fast event effects, renders, tells the server what the player is up to, and manages the user interface (keyboard, mouse, game pad, etc.). A video refresh library provides basic graphics abilities. The game library has various *monsters*, whose innate behaviors we have replaced with our Quagent control protocol. Thus they become *bots*.

Quake II is accompanied by PAK-files, which contain one or several maps that define the Quake II world, including

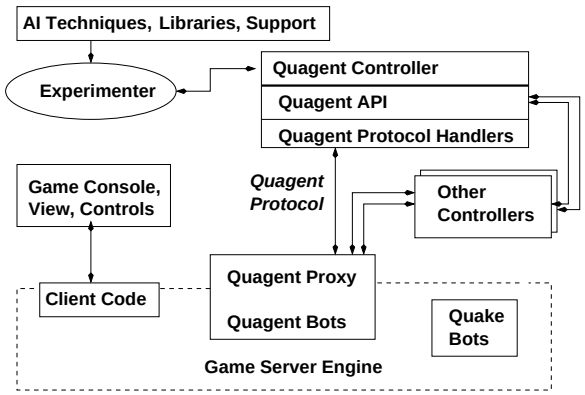

Figure 1: Basic Quagent Architecture

information about where entities are placed (spawned) in the world.

All entities in the game use a "think" function to execute calculations or behavior. The server loop, which is single threaded, provides each entity a slice of time to execute instructions from its current "think" function. The monsters in Quake II "think" with limited, game-inspired hacks; *e.g.*, they often navigate by following invisible breadcrumb-like paths. Our modification is that Quagents "think" by following orders from a user-written controller distinct from the game.

We considered other game platforms for Quagents, such as Quake 3 and DI-Guys. Our reasoning is reported in detail in (Brown *et al.* 2004), but source-code accessibility and operating system independence figured largely.

Quagent Architecture

The Quagent code and the communications protocol are a kind of specialized proxy server (Fig. 1). The user calls upon whichever AI disciplines are relevant, uses whatever language is best, and writes a Quagent controller. Via a socket connection this exterior controller substitutes for the original "game AI" of the erstwhile monster.

Quagents and Bots

In our ontology, the bots in the Quake II world are strictly hardware. They are controllable with a finite set of commands and they have reflexes that prompt them to inform the controller about certain events. All the intelligence therefore is off-board, outside the game, in user-written programs, which communicate with the game and *vice versa* through the protocol or user-written APIs using the protocol. The Quagent is thus the combination of the bot in the game and the user-written controlling software. If multiple bots are spawned, each has its own "private line" to its controller.

Quagent Ontology

Time: the quagent's world advances in 1/10 second intervals called "ticks". **The Client** (the erstwhile first-person shooter): the user controls this entity and sees through his eyes. He can move around and interact with bots. **Walls and windows:** things the quagent cannot walk through. **Items:** box, gold, tofu, battery, data, kryptonite, head. User can spawn these things at specified locations. **Other quagents:** perhaps controlled by other controllers. A Quagent has an internal state influenced by the world and its actions. **Position:** (X,Y,Z, Roll, Pitch, Yaw), **Inventory:** (Items in quagent's possession), **Wellbeing:** Health, Wealth, Wisdom, Energy, Age.

Quagent Physics and Biology

Something like normal physical laws apply in Quake II, but there are many tricks being used so some information one thinks should be available is either not there or hard to get. Default Quagents obey certain mechanical and biological laws (which are very easy to eliminate, change, or extend.) We present here a representative subset of current default Quagent characteristics. Presently we provide about a dozen different bot choices (corresponding to different monsters in the original game), so there is a fairly wide choice of appearances and certain hard-wired innate characteristics. We have removed violence from the bots' behavioral repertoire.

A bot is spawned upon connection to the game. Often this happens in the bot constructor function. Disconnecting kills the bot. All items but boxes can be walked over; boxes block walking. All items can be picked up. Picking up (Dropping) Tofu increases (decreases) Health. There are similar laws for picking up other items, and kryptonite is dangerous to pick up or even be near. Increased Wisdom increases perceptual abilities. Age increases every tick. When lifetime is exceeded or health drops to zero, the bot dies. Energy decreases by a small amount at every tick. When energy goes to zero, bot stops moving (everything else still works, including Pickup, so another bot could give it batteries.) There are several parameters settable by the user or the system that govern the physical and physiological constants in the Quagent world.

Usually the user sets up an experimental situation with two types of configuration file: One type sets up the world and populates it with items in specified locations. The second type initializes state levels and thresholds for bots to be spawned.

Arbitrarily complex communication and protocols (trading, auctions, deducing plans, etc.) between Quagent controllers (not bots) is trivial if they are all running in the same program. We have also implemented non-trivial communication between bots that may be spawned by different controllers or players (see below).

Quagent Protocol

Quagent controller code communicates with its bots in the game using a simple text protocol. The controller can send commands for the bot to do things (`DO`) or ask it questions (`ASK`). The bot confirms command receipt (`OK`), sends error messages (`ERR`), or sends back data in response. Bots can also asynchronously report events (`TELL`). The following is a quick synopsis of the main parts of the current Quagent protocol.

Actions

WALKBY *distance*: Command bot to start walking in current direction. Bot will start walking in the right direction and silently stop when distance is reached. Also RUNBY and STAND (stop moving).

TURNBY *angle*: Bot turns in place.

PICKUP *item*: Pickup item. Also DROP.

Perception

ASK *distance*: Bots reports list of objects and their positions within given distance. Also ASK RAYS.

CameraOn, CameraOff: Normally the terminal shows the view from the (first-person) client. These move the camera to the bot and back.

LOOK: Uploads image buffer to the quagent controller, showing the world as seen from the bot's point of view.

Proprioception and Asynchronous Condition Messages

The following query the bot's internal state and set certain bot parameters for experimental purposes: GetWhere (position), GetInventory, Get-Wellbeing, Set-EnergyLowThreshold, SetAgeHighThreshold.

The bot reports certain events asychronously, using the TELL message. Among those currently generated are:

STOPPED: Motion stops before desired WALKBY or RUNBY distance attained

LOWENERGY: Bot's energy fallen below a threshold; paralysis looms

NOTLOWENERGY: Energy has risen from below to above threshold.

OLDAGE: Bot's age is above threshold: death looms;

STALLING Bot is out of energy, alive but unable to move.

Quagent APIs

A key design goal for the Quagents platform is that it be readily usable in undergraduate AI courses and by AI researchers (see the next two sections, respectively). We have implemented an initial API in the first release of the platform, and are developing a more appropriate one for the next release.

The initial Quagent API, in Java, hides the details of socket connections, session initialization, and stream management from the user code. The details of creating and parsing messages are still left to user code, where it is both time-consuming to write and error-prone to run.

For the next release of the platform, we are implementing a more sophisticated API as another layer above the basic API. This "agent abstraction layer" hides all aspects of the Quagent protocol from user code. Instead, it provides abstractions such as Sensors and Behaviors that abstract the protocol details (asynchronous events are handled with a standard Listener paradigm). This allows agent developers to concentrate on the cognitive design of their agents rather than on talking with a game server.

Importantly, in this API, we can also develop "virtual" sensors and behaviors that extend and augment the "real" ones provided by the protocol. Two examples are a directional scanner that filters the results of a protocol ASK RAYS request, and an agent communication behavior that allows agents to exchange messages without any protocol interactions. The agent abstraction layer is designed to be extensible, allowing new real or virtual sensors and behaviors to be added easily.

To support research into agent teams, we have already implemented an inter-agent communication API dubbed TeamQuagent. This extension to the system adds unique (identifiably different) versions of items and bots, makes some items non-pickupable, and makes some game engine modifications. The TeamQuagent package contains low-level routines for Quagent functioning, including a Java class to extend and a separate facilitator for communication between TeamQuagents.

A message between quagents can contain an arbitrary collection of Java Objects, wrapped with the TeamMessage class. By using the Quagent-to-Controller and Quagent-to-Quagent methods, a fairly sophisticated bot can be constructed without using low-level system or networking commands. Also, the multi-thread safe nature of TeamQuagent allows for robust, real-time applications to be constructed without extra checks. We use these capabilities to implement auctions and trading for role-swapping (see below).

Quagents in Teaching

We wanted to take advantage of the fact that many undergraduate CS majors like games, that a fair amount of work is going on to improve AI in games, and that there is increasing awareness that standard AI topics can be coupled to entertainment (*e.g.*, (Buckland 2002)). In keeping with the tagline "The Intelligent Agent Book" of our text (Russell & Norvig 2003), with the recent introduction of a software engineering course based on such interactive games, and with our own experience with physical mobile robot courses at the graduate and undergraduate level, we created Quagent versions of all five programming exercises for our main undergraduate AI course (Rochester 2004a).

State-Space Problem Solving: Write general state-space search algorithms (depth first, breadth first, A*. etc.) for a maze-solving application in which the maze is represented by matrix of 0 and 1. Then transfer the solution to the Quagent world and have the Quagent execute the path found. Variations on the problem include implementing on-line problem-solving algorithms.

Production Systems: Students learn about production systems and use current commercial-grade AI software, usually to code a subsumption-style controller (Fig. 2). We use Jess (Sandia 2004) (our code distribution includes a Jess example.) The exercise also introduces the possibility of quagent-quagent communication.

Learning: As in the problem-solving project, students work in a small grid-world (as in the exercises in the text), and then must export and apply their policies and models to the Quake world. Again, on-line learning is also a possibility.

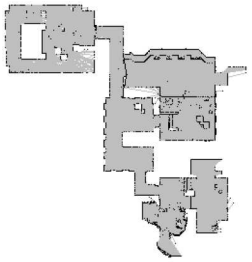

Figure 2: Map made under production system control (student project, 2003.)

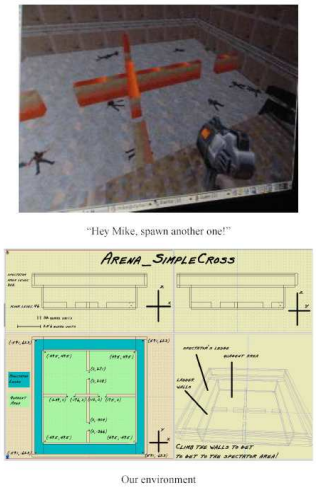

Figure 3: View of the situation and ground truth for visual learning and exploration project: some bots have expired of old age while exploring (student project 2003).

Natural Language Understanding: Command the bot (or get reports back from it) in natural language. As well as the text, we provide chapters from NLU texts on chart parsing, semantic grammars and recursive descent parsers.

Vision: Put some classic computer vision problems into the Quagent world. Low-level vision software is provided. Possible topics include segmentation, line-finding, object recognition, tracking (calls on previous chapters on Kalman filters), and mapping (Fig. 3.)

Quagents in Research

We want to use Quagents to study strategies for flexible, self-organizing cooperation between possibly heterogeneous intelligent agents, especially with minimal communication and methods of commanding and communicating with such teams.

In general, we are most interested in distributed coordination of intelligent agent teams (in general heterogeneous agents with different physical and computational abilities) using constrained communication. We assume that re-planning and re-assignment of roles can be based on higher-level re-tasking or local intelligence, that agents can defect or fail, that communication rates can vary, and that visual information can sometimes substitute for communications (this raises the "vision problem", however). We would like to allow individual agents to make their own decisions regarding how best to carry out their tasks.

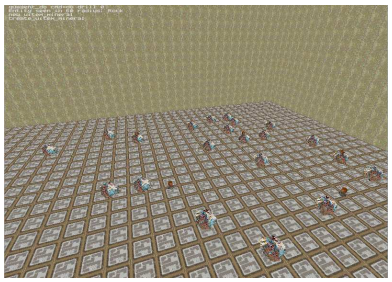

Figure 4: Agents in action. Operating surfaces are textured to help humans gauge distance. Rocks are the larger objects, host drilling bots appear smaller and dark.

Distributed Cooperation

As a first exercise in agent team control and complex communication, we implemented role-assignment, role-swapping and resource-allocation methods that approach optimal with good communications and degrade gracefully as communications degrade. Each robot is made responsible for a subset of the team's work, but while pursuing their goals, the robots attempt to trade goals with others that are more capable, thereby improving team performance. As communications degrade trading becomes less frequent, but every goal that can be achieved ultimately is achieved. We used the TeamQuagent protocol to implement trading. We modified the Quake code to add a protocol to trade items, money, and information. A trade is a transfer of any or all of the three between two quagents. Only items are handled in the actual Quake program: money and information, being new to the Quake ontology, are handled in the quagent controllers. Our communication protocol (Barnum, Ferguson, & Brown 2004) is similar to the contract net (*e.g.*, (Davis & Smith 1983)), with some significant differences. One is that once a goal is transfered to another host, the first host does not have to monitor its progress to make sure it is completed. Thus there is no communication while the team is working except when goals are reassigned. The quagents are not really subcontracting; they transfer the full goal. There are no thus layers or chains of subcontracting.

In the simulation, there is a large group of randomly positioned rocks that have to be drilled for samples (Fig. 4). We varied the number of agents in the team, tested different levels of communication uncertainty, and used several randomly generated groups of rocks, etc. We quantified team effectiveness, message traffic, *etc.* (*e.g.*, Fig. 5). On average, as more agents are added and as communications are more reliable, the faster the problem gets solved. Our experience confirms the sentiment that first-choice strategies such as auctions tend to work reasonably well (*e.g.*, (Dias & Stentz 2003).)

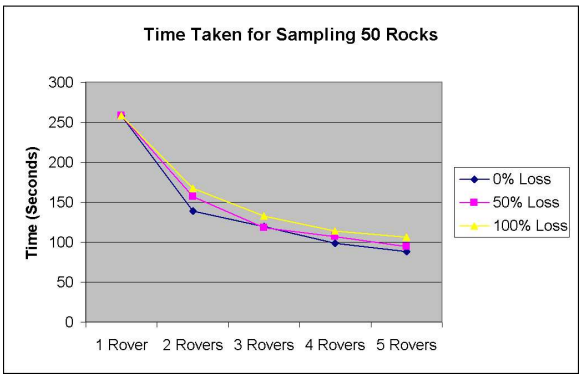

Figure 5: Task completion time vs. number of rovers for three values of message loss.

Robust Agent Teams

Interesting problems often have to be solved by teams. Sometimes a team of many agents can get a job done better or faster even if the agents are all the same (homogeneous teams). Other times a team is needed because the solution of the problem requires a set of skills that no one agent alone possesses (heterogeneous teams). In either case, an important issue is how to make the team robust to failures of individual members. That is, how do we ensure that the team meets its objectives even though individual members may fail to meet theirs (through communications or hardware failure or defection).

The Quagents platform is ideal for exploring the design and implementation of robust agent teams. There is much work in this area, from classic (*e.g.*, (Smith 1980)) to more recent (*e.g.*, (Modi *et al.* 2003)). Our current research uses the language of Joint Intention theory (Cohen & Levesque 1990) to describe shared goals, and uses voting and auction mechanisms to let agents agree on what the goals are and who ought to achieve them. We have implemented and analyzed a distributed version of Borda counting, which nicely takes account of agents' preferences and which is resistant to manipulation by individuals. Also for reasons of robustness we are considering Vickrey (sealed bid, second-price) auctions.

For our initial implementation of these ideas, we are developing a Mars exploration scenario using the Quagents platform. This includes specifying a new spatial layout, creating new graphics for various agents, and most importantly, using and extending the Quagent APIs for agent behavior and inter-agent communication implementing the robust agent teams.

In the future, one of the most promising directions is to refine the structure of the agents themselves. Although hybrid automata allow for complex functionality, hierarchical decision making, (*e.g.*, (Tambe 1997)) could allow for much more robust single-agent functionality even within teamwork systems. Another improvement would be to upgrade the auctioning process used for a goal to migrate from one Quagent to another. Combinatorial auctions have been successful in many agent-based and planning approaches (*e.g.*, (Hunsberger & Grosz 2000; Likhodedov & Sandholm 2004)), and give better performance than simple greedy searches.

We should like to pursue approaches that suggest performance can be guaranteed for robots that observe their environment and see what their teammates are doing instead of explicitly communicating (*e.g.*, (Wie 1997; Balch & Arkin 1997; Various 2004)). To reduce further the need for communication, elements from swarm robotics, mechanism design, and stigmergic theory could be added or substituted (Reynolds 1987; Steels 1990). Even though the goals move from agent to agent, it may be possible to make the interactions more sophisticated and attain greater functionality, even in uncertain environments.

Human Management of Intelligent Agents

Intelligent agents are increasingly capable (whether in the real world, such as various robots, or in a cyber-world of software agents). However, little attention has been paid to the fact that these agents will almost always have to interact with people. At the very least, people originate the problems that agents are intended to solve. People are also often the beneficiaries (or victims) of agents' actions, meaning that human control and delegation of authority are crucial. And once we get beyond the simplest reactive, hard-coded agents, they will need to interact with people during their operation, for example to clarify requirements, obtain information or authorization, identify problems, and so on.

A promising approach to this type of collaborative problem solving is to develop an intelligent assistant that helps a person solve problems. In some such systems (*e.g.*, (Ferguson & Allen 1998)), this assistant communicates using spoken natural language. Regardless of modality, the idea is that the assistant will coordinate and manage a team of autonomous agents running on the Quagents platform. The assistant's job is to help the human supervisor understand the situation and achieve their objectives. Quagents provide an ideal level of abstraction for this research. It is realistic enough that we believe the results would transfer to real world huma and/or robotic agents, but abstract enough that we can concentrate on developing the assistant rather than the agents.

Use of the Quagents platform in this scenario suggests some unique possibilities for evaluation. First, one could compare the human-assistant-quagents team with a team of expert human players each controlling a single character in the game. Presumably the team of humans would easily outperform our team (but it's a nice challenge). Our intuition is that the main reason the humans would do well is that they would talk to each other offline (outside the game). So we could make a fairer comparison by putting them in separate rooms. Or we could try putting a team of players in a room together, against a human with computer assistant controlling the same number of Quagents. One can imagine that, with some attention to scaling in the assistant system, the team of humans couldn't possibly coordinate themselves as well as the system could (or to the extent that they can, we learn something about agent coordination). In any case, the Quagents platform provides an ideal environment in which

to develop collaborative assistants for human management of intelligent agents.

Conclusion

Interactive Digital Entertainment needs access to the latest AI research but industries cannot give their source code away to academics. AI communities need powerful real-time simulations to develop AI for robotic systems and for games and entertainment, but they must be flexible, use existing AI code, be instrumentable, and in short be open, sharable, research vehicles. *Quagents* is an agent development environment embedded in an interactive digital environment. The goal is to have a sophisticated, game-like simulation available to academic (and individual) investigators and teachers using Linux/Unix as well as Windows platforms. The intelligence component is independent from the game (world) controller: the two communicate through a standard interface (sockets) that can be enhanced by APIs designed by the investigator, not provided by the game. Ultimately, we want to give AI researchers the luxury of a complex, real-time environment for intelligent agents, and we hope that progress and technology will then flow the other way from the research into the commercial communities. In parallel with IDE applications, we believe AI will enter the real world of transport, home, health care, and military applictions as well, and flexible and powerful simulators will speed that process. The Quagent code is freely distributable and available at (Rochester 2004b), which also has the full version of this paper with many implemention and documentation details.

References

Balch, T., and Arkin, R. 1997. Behavior-based formation control for multi-robot teams. Submitted for Publication (citeseer.ist.psu.edu/balch99behaviorbased.html).

Barnum, P.; Ferguson, G.; and Brown, C. 2004. Team coordination with multiple mobile agents. Technical Report URCS TR Draft, Dept. CS, U. Rochester.

Brown, C.; Barnum, P.; Costello, D.; Ferguson, G.; Hu, B.; and Wie, M. V. 2004. Quake ii as a robotic and multi-agent platform. Technical Report 853, University of Rochester Computer Science Department. In DSpace: http://hdl.handle.net/1802/1042.

Buckland, M. 2002. *AI techniques for game playing*. Cincinnati, OH: Premier Press.

Cohen, P., and Levesque, H. 1990. Intention is choice with commitment. *Artifical Intelligence* 42:213–261.

Davis, R., and Smith, R. 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 20(1):63–109.

Dias, M. B., and Stentz, A. 2003. Traderbots: A market-based approach for resource, role, and task allocation in multirobot coordination. Technical Report CMU-RI-TR-03-19, CMU Robotics Institute.

Ferguson, G., and Allen, J. F. 1998. TRIPS: An integrated intelligent problem-solving assistant. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 567–573.

Hunsberger, L., and Grosz, B. 2000. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, 151–158.

Id, S. 2004. id software downloads. http://www.idsoftware.com/business/techdownloads/.

Laird, J. E. 2004. John laird's artificial intelligence and computer games research. http://ai.eecs.umich.edu/people/laird/gamesresearch.html.

Likhodedov, A., and Sandholm, T. 2004. Methods for boosting revenue in combinatorial auctions. In *Proceedings on the National Conference on Artificial Intelligence (AAAI)*, 232–237.

Modi, P.; Shen, W.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization.

Noelle, D., and Fakhouri, T. 2004. Modeling human prefrontal cortex in quake iii arena. http://www.vuse.vanderbilt.edu/∼noelledc/resources/VE/IDED/home.htm.

Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics* 21(4):25–34.

Rochester, U. 2004a. Artificial intelligence course assignments and student projects. http://www.cs.rochester.edu/u/brown/242/assts/assts.html.

Rochester, U. 2004b. Quagent main page. http://www.cs.rochester.edu/ research/quagents/.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence A Modern Approach (2nd Ed)*. New Jersey: Prentice Hall.

Sandia, N. L. 2004. Jess programming system. http://herzberg.ca.sandia.gov/jess/.

Smith, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers* C-29(12):1104–1113.

Steels, L. 1990. Cooperation between distributed agents through self-organization. *Decentralized A.I.* 175–196.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.

Terzopoulos, D., and Rabie, F. 1997. Animat vision: Active vision in artificial animals. *VIDERE* 1(1). Electronic Journal: http://www-mitpress.mit.edu/e-journals/Videre/.

Various. 2004. Various. In *Modeling Other Agents from Observations*. (Columbia U. Workshop at Int. Joint Conf. on Autonomous Agents and Multi-agent systems.

Wie, M. V. 1997. Establishing joint goals through observation. In David P. Casasent, C. M. U., ed., *Intelligent Robots and Computer Vision XVI:Algorithms, Techniques, Active Vision, and Materials Handling*. Pittsburgh, PA, USA: The International Society for Optical Engineering.