Morgan, G. P., Ritter, F. E., Stevenson, W. E., Schenck, I. N., & Cohen, M. A. (2005). dTank: An environment for architectural comparisons of competitive agents. In L. Allender & T. Kelley (Eds.), Proceedings of the 14th Conference on Behavior Representation in Modeling and Simulation. 133-140. 05-BRIMS-043. Orlando, FL: U. of Central Florida. 05-BRIMS-043.

dTank: An Environment for Architectural Comparisons of Competitive Agents

Geoffrey P. Morgan
Frank E. Ritter
William E. Stevenson
Ian N. Schenck
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16801-3857
gmorgan@psu.edu, frank.ritter@psu.edu, wstevenson@ist.psu.edu, ins5000@psu.edu

Mark A. Cohen
Business Administration, Computer Science, and Information Technology
Lock Haven University
Lock Haven, PA 17745
mcohen@lhup.edu

Keywords: simulation environments, agent architectures

ABSTRACT: *We introduce dTank, a competitive environment, as useful for architectural comparisons of competitive agents and comparisons of human and agent behavior. dTank, a Java-based simulation, was designed to facilitate these forms of holistic comparison of emergent behavior. We present several models built using several cognitive and agent architectures (Java, Jess, and Soar), and compare them against a novice participant. We find that our novice participant used two strategies while playing: wandering and attacking. All three agents accurately modeled the novice's wandering behavior, while the Soar agent more accurately modeled the novice's attack behavior. These results demonstrate the usefulness of dTank as a teaching and modeling tool, and as a lightweight environment for testing cognitive models and architectures.*

1. Introduction

Often, modelers develop agents and build agent-environments to facilitate the development and isolation of particular cognitive phenomena. Modelers have rarely examined the emergent patterns of behavior of different models in the same environment. To our knowledge, very few modelers have examined models and agents built in different architectures as well as humans in the same environment, although the AMBR project is the definitive example of this form of analysis (Gluck & Pew, 2001) and the Sisyphus project, which used tasks such as the one documented by Yost and Rothefluh (1996), was similar in direction for expert system shells. We offer an environment, dTank, that facilitates this type of analysis for competitive environments.

We present dTank as a useful tool for inter-architectural comparison of models and agents against human measures as well in comparison with one another. dTank has been designed to present uniform capabilities to models, agents, and humans. dTank is a Java-based tool and runs well in both PC and Mac environments. It uses socket communication methods to provide uniform connections to all models.

First, we present the system design of dTank and discuss how this design facilitates parallel comparison of models, agents, and humans. Second, we introduce several dTank models and agents built using different cognitive and agent architectures. Third, we present a model-to-model and model-to-human comparison. Fourth, and finally, we conclude with a discussion of these comparisons, their implications, and future work with dTank.

2. dTank's Design

dTank was designed with two main criteria, that (a) all software models/agents should be able to use a universal interface for connection, and that (b) the human and software players had parallel capabilities available to them. There were several other criteria as

well, including that the system be interesting and easy to use, and that it logged the agents' behavior for later analysis and playback. Finally, the driving force was that we have been working on an approach to creating models that can explain themselves (Haynes, Ritter, Councill, & Cohen, Submitted). Such models need something to explain. While models within ModSAF would be appropriate, this simulation is far too large and unwieldy for most exploratory work, particularly in most students and by students.

The original design of dTank was inspired by Tank-Soar, and was intended to provide very similar functionality. Its implementation has drifted somewhat from Tank-Soar, the tank game developed by Mazin As-Sanie and included with Soar distributions, but not in spirit. It was also designed to provide a lightweight alternative to ModSAF.

2.1 dTank's Architecture and Interface

We chose a client-server architecture and a socket-based interface. A server is started up as a Java program. The server displays everything, and runs the simulation. A human user can choose to enter the environment from the server menu, and then the UI displays only the limited view parallel to agent vision. Agents, models, or humans that connect to the server are all given a tank on the board, and can then send commands to move the tank. The clients are given updates every 2 s of what is visible to them on the board. This update rate was chosen because this is an estimated scan rate of Navy pilots found in another study (Councill, Haynes, & Ritter, 2003). More details are provided in the manual, including how to create and use up to 100x100 square boards (Councill, Morgan, & Ritter, 2004).

Models receive information and pass commands in the form of text-strings that are converted into state representations conforming to the needs of the architecture. Converter files must be created on a per-architecture basis. Currently Lisp, Java, and Tcl/Tk files are available.

Figure 2.1 shows a detailed view of two tanks in the server window. Figure 2.2 shows the interface that a human user sees when his dTank has connected to a server, and a more limited view is provided.

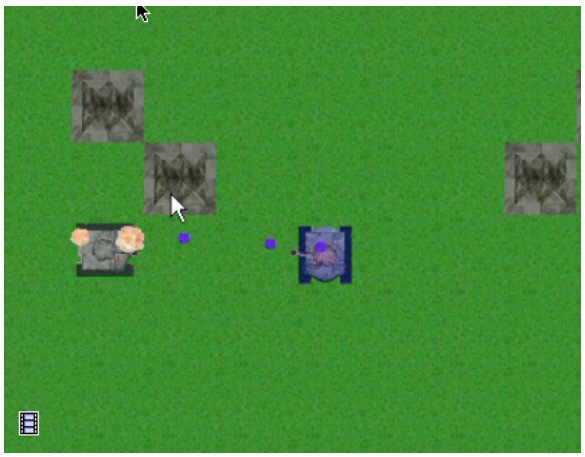

Figure 2.1. A detailed view of dTank from the server interface.

Models can use this state-representation differently, either acting directly on information as it is made available, or building internal state-representations. Models use this state information, and then decide upon an action that affects the environment. This action is sent to dTank in a formatted string and held in a buffer for time-synchronization purposes.

Aside from state information, which is sent at regular intervals, software agents are also informed of important events, such as being hit, 'dying', and 'health-low'. This is to allow modelers some flexibility in the behaviors of models in difficult situations.

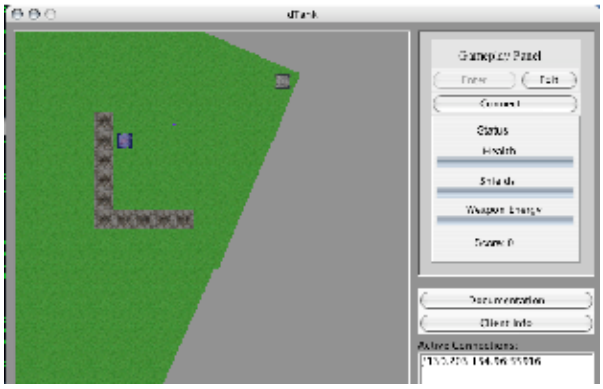

Figure 2.2. View of the dTank client being played by a human.

2.2 Parallelism in dTank

dTank was also designed to keep the capabilities of the human and software agent parallel. This proved more difficult than initially expected. In order to keep the capabilities parallel the vision of both humans and agents must be limited appropriately. Software agents

receive state information and event notices for events that occur within a 100 degree arc centered on their turret's current vector. To limit human vision similarly, a 3D interface was proposed and implemented, but proved difficult to optimize using the Java environment. Instead, a ray-casting algorithm is used to limit human vision to a 100 degree arc on a 2D (plan-view) display.

The use of the 2D display for humans is also appropriate because the first person view, while perhaps more interesting to some users, requires a more complex reasoning strategy than all models created so far. Agents view other agents and themselves as being placed on an overview map. If an agent were created that evaluated risk by examining distances (i.e. perception), then a 3D interface would be more parallel.

Because the plan-view display was originally designed to serve as the server interface for the dTank system, sensory filtering occurs at different stages between the software agents and human players, as shown in Figure 2.3. The human client has all the information, but is limited by its ray-casting algorithm to displaying only what the human agent should see. The agent is given only the information it can currently possess.

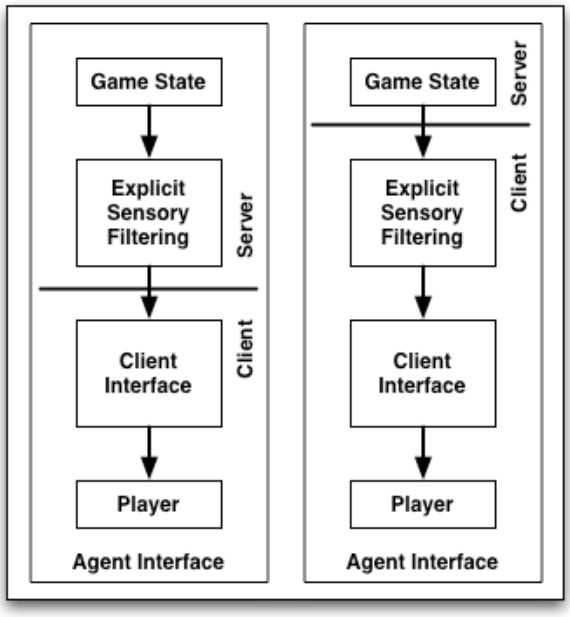

Figure 2.3 Sensory Filtering to create parallelism in dTank.

Aside from input, models and humans should be able to do the same things at the same pace. This has also proven difficult. Concrete actions, such as moving forward, firing, and turning, are performed similarly by humans and agents. More complex and abstract actions such as communicating with other tanks and scanning specific visible tanks are difficult to implement in a parallel fashion.

Forcing a theory of perceptual and motor skills upon software agents through a required time lag is not an acceptable option, and these capabilities are abstract and made use of only by highly refined models. Therefore, we have made the capabilities available to both software models and human users and currently rely on the modeler to maintain parallelism of what is available to models. This could be done by ignoring the socket-based interface system and playing the game with the human user interface using a tool like Segman (Amant & Riedl, 2001).

dTank has been used by two classes on cognitive modeling at Penn State (Cohen, Ritter, & Haynes, In Press), a class at Lock Haven and at a university in Brazil, and by an Army MURI to investigate the effect of communications upon teamwork (Sun, Councill, Fan, Ritter, & Yen, 2004).

Work remains with dTank and its analysis tools to make them more useful to the modeler. Moves into a wall or missed shots are ineffective. dTank does not differentiate between effective and ineffective actions when creating its log file—it does not record whether actions were successful, or if they were ignored. Currently, neither do the analysis tools. We believe that information about the number of effective and ineffective clicks are both very valuable to the debugging modeler, but we do not currently distinguish between them.

Also, because the system does not inform the human participant of ineffective clicks, the user can become frustrated by apparent lack of response by dTank. This could be fixed by providing immediate feedback (as is provided across the socket interface) of an action's effectiveness (such as a beep when an action is ineffective). Capturing ineffective keystrokes when we know that the user has been informed of the error will let us examine a participant's ability to learn the game and other valuable aspects of error.

Although the parallelism is still imperfect, dTank provides input and output to humans and agents in a manner sufficiently parallel to provide useful comparisons between humans and models and across models. We do this next.

3. dTank Models

To understand several architectures better, we have created dTank models in several different architectures.

We discuss here a Java, Jess, and Soar model and compare their behaviors and actions with a human novice player. We are currently working on models for ACT-R (Anderson & Lebiere, 1998), JACK (AOS, 2004), and COJACK (Norling & Ritter, 2004). Other agents have been written in the CAST architecture (Sun et al., 2004; Yen et al., 2001) but are not discussed here.

We feel that dTank actions, such as moving forward, turning, rotating the turret, and firing can be composed into chains that form strategies such as Wandering, Hunting, Chasing, and Attacking that are broadly applicable to both models and humans. Briefly, we define these strategies but note that Wandering and Attacking are the two most common strategies across models.

Wandering consists of moving in a random or semi-random fashion across the map while moving the turret. This strategy initiates when no target is visible and terminates upon acquiring a target. It is composed of a series of turret rotation, move forward, and turn commands.

Hunting is a specialized form of wandering. A tank moves to the border of the map and begins to circle the board while rotating the turret until a target is acquired. It is otherwise identical.

Chasing consists moving towards a known tank location. It is composed of a series of turn and move-forward commands.

Attacking consists of a rapid series of commands that aim and fire the turret. Usually this is composed of alternating series of aim and fire commands until the enemy can no longer be seen. This strategy initiates upon sight of an enemy tank and terminates when no enemy tank is visible.

3.1 JavaTank

Java is a object-oriented programming language (Arnold, Gosling, & Holmes, 2000). It has no implicit theory of cognition, perception, or action. However, our Java agent, JavaTank, takes a small step towards cognitive plausibility by waiting 50 ms before it will consider the environment or implement an action JavaTank is capable of moving forward, turning, rotating its turret, aiming at a target, and firing. An expert Java developer created the JavaTank in about five man-hours. It consists of 13 methods and takes advantage of two objects created to support further tanks built in Java.

JavaTank wanders normally for a period of time. After a period of time in failing to spot the enemy, JavaTank uses the Hunt strategy. Upon sight of an enemy tank, it attacks.

3.2 JessTank

Jess (Java Expert Systems Shell) is a Java-based rule engine derived from the CLIPS expert systems shell created and distributed by NASA (Friedman-Hill, 2003). The Jess agent, JessTank, also requires 50 ms before it will consider the environment or implement an action. Development time of this model was approximately one man-hour by an expert, but it is simpler than the other two. It is only capable of moving forward, turning, rotating its turret in the cardinal directions, and firing. This model also takes advantage of Java class objects built to support development of Java-compatible agents for dTank and has nine rules; an example rule is included here as Table 3.1

Table 3.1 Fire command for JessTank

```
(defrule attack
  "if I see a tank and we are in the same x location, fire!"
  (visual (x ?tx))
  (status (x ?x) (reloading false))
  (test (= ?tx ?x))
 =>
  (fire)
  (printout t "attacking..." crlf))
```

JessTank wanders until it spots an enemy tank. Upon sight of that tank, it chases that tank until it has reached the same horizontal row as the tank it has spotted. Once it has, it begins to attack. This unusual strategy was chosen by the JessTank developer. Unlike JavaTank or SoarTank, JessTank does not perform the required trigonometric equations to aim the turret at any enemy that can be seen.

3.3 SoarTank

Soar is a cognitive architecture (Newell, 1990). The SoarTank model included in the dTank distribution was the first model developed, and it was built by the original dTank developer in about three hours. It is capable of moving forward, turning, rotating its turret, aiming at a target, and firing.

SoarTank relies on a directed form of wandering that prevents it from merely rotating in place. After it has spotted an enemy, it begins attacking.

A further half-dozen models have been created in Soar, Herbal-Soar (a high level compiler for Soar, and Slip-Soar (a version with errors in its motor output).

Examples of these are included in the dTank web site (http://acs.ist.psu.edu/projects/dTank/).

4. Model to Human Comparisons

Model-to-Human comparisons are the major product of our work with dTank.

4.1 Methodology

We gathered five minutes of keystroke protocols from a novice player of dTank. The player was familiar with the concept of dTank, and had played computer games before. We provided the subject an instruction sheet that listed how to perform the basic dTank commands: move, fire, turn clockwise, turn counter-clockwise, rotate (the turret), and shield. We gave the subject five minutes to review the instruction information without access to the environment. We then logged his keystrokes while he played against the JavaTank agent for five minutes, producing a text log of more than a 1,000 attempted actions in sequence. For comparison, a text log was produced for each agent and model described above in a similar fashion.

After the player had finished, we asked him to explain his strategy and assisted the participant in constructing a flowchart of his actions, which the participant validated.

In order to validate that dTank is showing behavior similar to people, we used CaDaDis (Tor, Ritter, Haynes, & Cohen, 2004), a tool that automatically creates categorical data displays for cognitive models and intelligent agents. A software tool (available in the next release of dTank) was used to process the dTank text logs and create CaDaDis log files for each agent. The default CaDaDis displays were modified to display the action sequence. This action sequence focus allowed us to compare human and model sequences at the same level of granularity.

4.2 Analysis of Participant Strategy

The participant relied more on turret rotations and less on moving the tank, but still both rotated the turret and moved the tank. The participant did not attempt to use his shields at any point in the five minute log. With assistance, he then produced a flowchart that encapsulates his strategy.

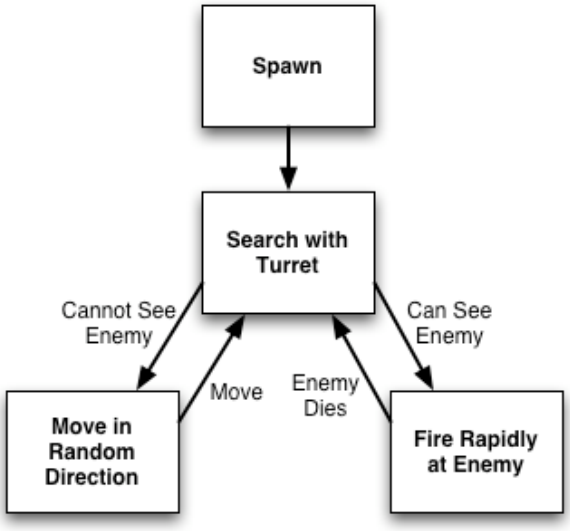

Figure 4.1 Participant strategy flow.

This strategy is equivalent to the Wander and Attack strategies discussed earlier. After being spawned, if he could not see an enemy, he wandered, with a preference for turret rotations. If he did see an enemy tank, he attacked. We think it is interesting that the participant did not explicate the actions of attacking, that is, aiming and firing actions in a long series, but instead focused on the high-level strategy of attacking.

4.3 Models to Subject Strategy Comparisons

This analysis will focus on each of the basic strategy types and examine the typical action chains that formed each core strategy in the agents as compared to the human version. For this reason, only Wander and Attack behaviors are compared to the subject data, because those are the behaviors that the subject engaged in.

A summary of the total attempted actions for the participant and each model is shown in Table 4.1. Not all actions proposed by either the subject or software modelers were accepted by the server (e.g., moving into a wall), although all actions are still analyzed here. A tank is able to turn in both a clockwise (Turn-C) and counter-clockwise (Turn-CC) direction.

Table 4.1 Attempted Actions for all Actors.

Actor	Fire	Rotate	Move	Turn-CC	Turn-C
Participant	350	569	529	42	8
JavaTank	693	193	310	35	38
JessTank	39	91	202	100	62
SoarTank	316	90	93	62	69

The Wandering strategy varied between human and the different models. The participant's wandering strategy

varied between turret-rotations followed by a series of move-forward and turn commands, as shown in Figure 4.2

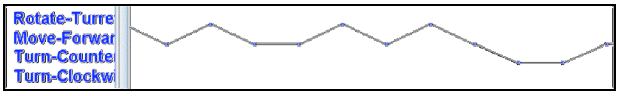

Figure 4.2 Participant wandering strategy from CaDaDis.

The models had different wandering strategies, as shown in Figure 4.3. JavaTank and JessTank alternates between rotating its turret and moving, either one or several actions. JavaTank typically moved only once before rotating its turret again. SoarTank, which builds internal memory, rotated its turret less frequently and continued with more maneuvering commands between turret rotations. All tanks still receive updated visual field information every two seconds.

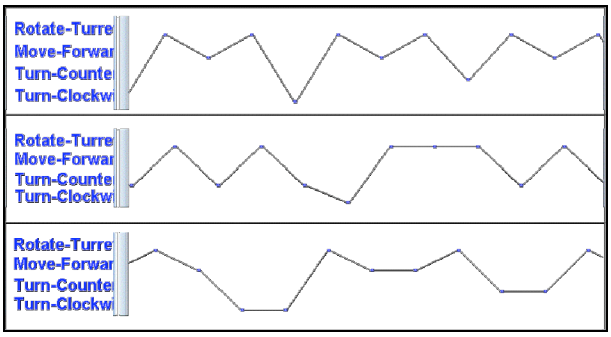

Figure 4.3 JavaTank (top), JessTank (middle), and SoarTank (bottom) wandering strategy, from CaDaDis.

The participant's attack strategy was very simple, but effective. Most often, the participant would re-aim the barrel and immediately fire. We believe that although it involved two separate actions, pressing the mouse-button to re-aim and pressing the space-bar to fire, the time lag between the two actions indicate that the participant viewed them as one action, with no time for thought between the two. Rotate Turret, of the 560 attempted actions, was followed by Fire 289 times (more than half the time), with an average latency between the two actions of 83 ms. This behavior is demonstrated in a CaDaDis trace shown in Figure 4.4.

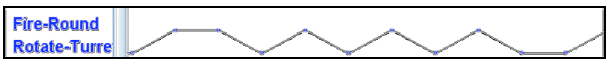

Figure 4.4 Participant attack strategy.

Most of the agents had attack strategies that were similar to the participant's strategy, as is shown in Figure 4.5. The major exception is JessTank, which must be on the same horizontal location as its enemy or it will not fire, which makes movement much more prominent in the behavior of this agent. SoarTank alternates turret rotations between attacks. JavaTank, once locked on to an enemy, fires many shots rapidly at the enemy.

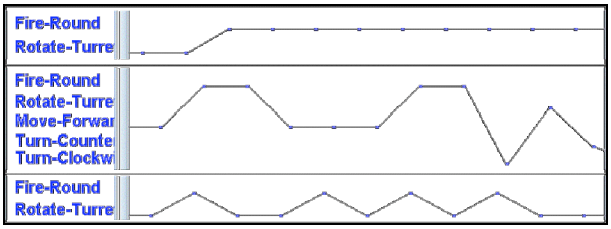

Figure 4.5 JavaTank (top), JessTank (middle), and SoarTank (bottom) attack strategies.

Once the model behavior is captured in CaDaDis, an audio representation can be synthesized that literally "plays" the actions of the agent. Such audio files, particularly compared against those of other models, can be very useful for analyzing model behavior differences, including innate differences between cognitive architectures. Example CaDaDis audio analyses can be found on the CaDaDis website, acs.ist.psu.edu/projects/CaDaDis.

5. Discussion and Future Work with dTank

The analysis shown here demonstrates the potential use of dTank as a platform for comparing agents in competitive environments. It will also be useful for examining specific cognitive phenomena such as learning and the effect of moderators on behavior.

A recent undergraduate class project, for example, showed that as slips increase, the win/lose ratio against the Java tank decreases linearly to 0.

Perhaps its greatest use is as a training environment for cognitive modelers. The environment is simple enough to work with, yet complex enough to provide some challenges. More importantly, dTank provides tools to support the analysis of information gathered in its environment.

Theories of social processes are testable in dTank, although we do not provide such an analysis here. dTank allows communication to pass between all the agents currently active in the environment. Work with CAST (Yen et al., 2001) has shown that the appropriate level of communication for agents varies by task complexity (Sun et al., 2004).

We believe that dTank has promise and use as a useful modeling environment for behavior representation, although we see many opportunities for improvement in dTank.

6. Acknowledgements

The development of this software was supported by ONR (contract N000140210021). It was also supported by the Director of Technology Development, Ministry of Defence, Metropole Building, Northumberland Ave, London WC2N 5BP and was carried out under the terms of Contract No RT/COM/2/007. We also thank Isaac Councill for his work on SoarTank and work on dTank.

7. References

Amant, R. S., & Riedl, M. O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies, 55*, 15-39.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.

AOS. (2004). *JACK intelligent agents*. Victoria Australia: Agent Oriented Software Pty. Ltd.

Arnold, K., Gosling, J., & Holmes, D. (2000). *The Java programming language* (3rd ed.). Boston: Addison-Wesley.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (In Press). Herbal: A high-level language and development environment for developing cognitive models in Soar. In *the 14th Conference for Behavioral Representation in Modeling Simulations (BRIMS)*.

Councill, I. G., Haynes, S. R., & Ritter, F. E. (2003). Explaining Soar: Analysis of existing tools and user information requirements. In F. Detje, D. Doerner & H. Schaub (Eds.), *Proceedings of the Fifth International Conference on Cognitive Modeling.* (pp. 63-68). Bamberg, Germany: Universitats-Verlag Bamberg.

Councill, I. G., Morgan, G. P., & Ritter, F. E. (2004). *dTank: A competitive environment for distributed agents* (Tech. Report No. 2004-1): Applied Cognitive Science Lab, School of Information Sciences and Technology, Penn State.

Friedman-Hill, E. (2003). *Jess in action: Rule-based systems in Java*: Manning Publications Company.

Gluck, K. A., & Pew, R. W. (2001). Lessons learned and future directions for the AMBR model comparison project. In *The 10th Computer Generated Forces and Behavioral Representation Conference (10TH-CGF-067)* (pp. 113-121). Orlando, FL: Division of Continuing Education, University of Central Florida.

Haynes, S. R., Ritter, F. E., Councill, I. G., & Cohen, M. A. (Submitted). Explaining intelligent agents. *Journal of Autonomous Agents and Multi-Agent Systems*.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Norling, E., & Ritter, F. E. (2004). A parameter set to support psychologically plausible variability in agent-based human modeling. In *The Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS04)* (pp. 758-765). New York, NY: ACM.

Sun, S., Councill, I. G., Fan, X., Ritter, F. E., & Yen, J. (2004). Comparing teamwork modeling in an empirical approach. In *International Conference on Cognitive Modeling*. Mahwah, NJ: Lawrence Erlbaum.

Tor, K., Ritter, F. E., Haynes, S. R., & Cohen, M. A. (2004). CaDaDis: A tool for displaying the behavior of cognitive models and agents. In *13th Conference on Behavior Representation and Simulation* (pp. 192-200). Orlando, FL: U. of Central Florida.

Yen, J., Yin, J., Ioerger, T. R., Miller, M. S., Xu, D., & Volz, R. A. (2001). CAST: Collaborative agents for simulating teamwork. In *Seventeenth International Joint Conference on Artificial Intelligence* (pp. 1135-1142). Seattle, WA.

Yost, G. R., & Rothenfluh, T. R. (1996). Configuring elevator systems. *International Journal of Human-Computer Studies, 44*, 521-568.

Author Biographies

GEOFFREY MORGAN is a research assistant in the Applied Cognitive Science Lab, School of Information Sciences and Technology (IST). His current research involves implementing behavioral moderators in cognitive architectures and models of teams.

FRANK RITTER is one of the founding faculty of the School of IST, an interdisciplinary academic unit at Penn State to study how people process information using technology. He works on the development, application, and methodology of cognitive models, particularly as applied to interfaces and emotions. He is an editorial board member of *Human Factors* and *AISB Journal*. His review (with others) on applying models in synthetic environments was published in 2003 as a HSIAC State Of the Art Report.

WILLIAM STEVENSON is a student in the School of Information Sciences and Technology at Pennsylvania State University. He is also a research assistant in the Applied Cognitive Science Lab. His current work focuses on modeling human vision and creating displays for explaining cognitive models. Stevenson received a BA/BS dual degree in Cognitive Science and Computer Science from Muhlenberg College in Allentown, PA and a MEng in Computer

Science and Engineering from Penn State. He is the current editor in chief of *ACM Crossroads*.

IAN SCHENCK is a research assistant in the Applied Cognitive Science Lab, School of Information Sciences and Technology (IST). His current research involves fitting models to data with a combination of search methods including gradient descent and genetic algorithms. He also maintains dTank and the dTank API's, and creates models in various architectures to play in dTank.

MARK COHEN is an instructor in the Business Administration, CS and IT Department at Lock Haven University, and a graduate student associated with the Applied Cognitive Science Lab in the School of IST at Penn State. His current research efforts include developing software that simplifies the creation and maintenance of cognitive models. He received an MS in CS from Drexel University and a BS EE from Lafayette College. He has over 10 years of experience developing health care and pharmaceutical software.