Synthetic Adversaries for Urban Combat Training

Robert E. Wray

Soar Technology, Inc.
3600 Green Court Suite 600
Ann Arbor MI 48105
wray@soartech.com

John E. Laird, Andrew Nuxoll, Devvan Stokes, Alex Kerfoot

University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109-2110
laird, anuxoll, dstokes, akerfoot@umich.edu

Abstract

This paper describes requirements for synthetic adversaries for urban combat training and MOUTBots, a prototype application. The MOUTBots use a commercial computer game to define, implement and test basic behavior representation requirements and the Soar architecture as the engine for knowledge representation and execution. We describe how these components aided the development of the prototype and present an initial evaluation against competence, taskability, fidelity, variability, transparency, and efficiency requirements.

Introduction

Offensive urban combat is one of the most difficult tasks soldiers perform. Urban combat is characterized by building-to-building, room-to-room fighting. Frequent training is an essential element in reducing casualties. However, training in urban environments is costly and restricted to physical mockups of buildings and small towns. The Office of Naval Research's Virtual Training & Environments (VIRTE) program is developing immersive virtual trainers for military operations on urbanized terrain (MOUT). In this trainer, four-person fire teams of U.S. Marines will be situated in a virtual urban environment and tasked to clear a building, possibly containing enemy soldiers. Virtual opponents are required to populate the environment and challenge the trainees.

This paper describes the general requirements for opponents and our development of synthetic adversaries to meet them. The agents are built using the Soar cognitive architecture and they interface to a commercial game engine that served as an initial development environment. In order to simplify the behaviors that needed to be encoded in the prototype, we have initially focused on MOUT behaviors within a building.

Requirements for Synthetic Adversaries

This application has six major high-level requirements:

1. **Competence:** The adversaries must perform the tactics and missions humans perform in this domain. For this application, the adversaries' goal is to defend a small multi-storied building in teams of 2-5 using assault weapons and grenades. The agents must move through the environment, identify tactically relevant features (such as escape routes), and communicate and coordinate with other agents.
2. **Taskability:** The agents must be able to be assigned new missions for different training scenarios and they must change their objectives during an exercise.
3. **Observational fidelity:** The agents do not have to model accurately all aspects of human behavior. Instead, they must model those aspects of human behavior that are observable and relevant to training.
4. **Behavior variability:** The agents must not be predictable so that the trainees are exposed to many diverse experiences. By experiencing many different situations and opponent behaviors, unexpected situations will be minimized in real life.
5. **Transparency:** To allow "after action review," opponents should be able to explain why they went into a particular room, or retreated at some juncture in the scenario. Explanations are much easier to generate in systems that use transparent representations that map directly to understood terms.
6. **Minimal computational footprint:** This application is targeted for personal computers, with a majority of the computational processing reserved for graphics and physics modeling. Thus, synthetic adversaries will be allocated only a small time slice in system execution.

Related Work

Possibly the most advanced examples of embedding synthetic agents in military simulations are TacAir-Soar and RWA-Soar. These systems emulate the behavior of military personnel performing missions in fixed-wing and rotary-wing aircraft and have been used in large-scale military training exercises (Jones et al., 1999; Tambe et al., 1995). TacAir-Soar and RWA-Soar agents are completely autonomous, making decisions based on their awareness of the surrounding environment, commands received from other entities, and their extensive knowledge of military doctrine and tactics. They have the ability to pursue mission objects alone, or they can participate in larger groups made up of other synthetic agents or even human teammates participating via simulators (Laird, Jones, & Nielsen, 1998). The research described here builds on that prior research and extends it to the specific requirements of the creating adversaries for MOUT training. The four most important areas of differentiation are:

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

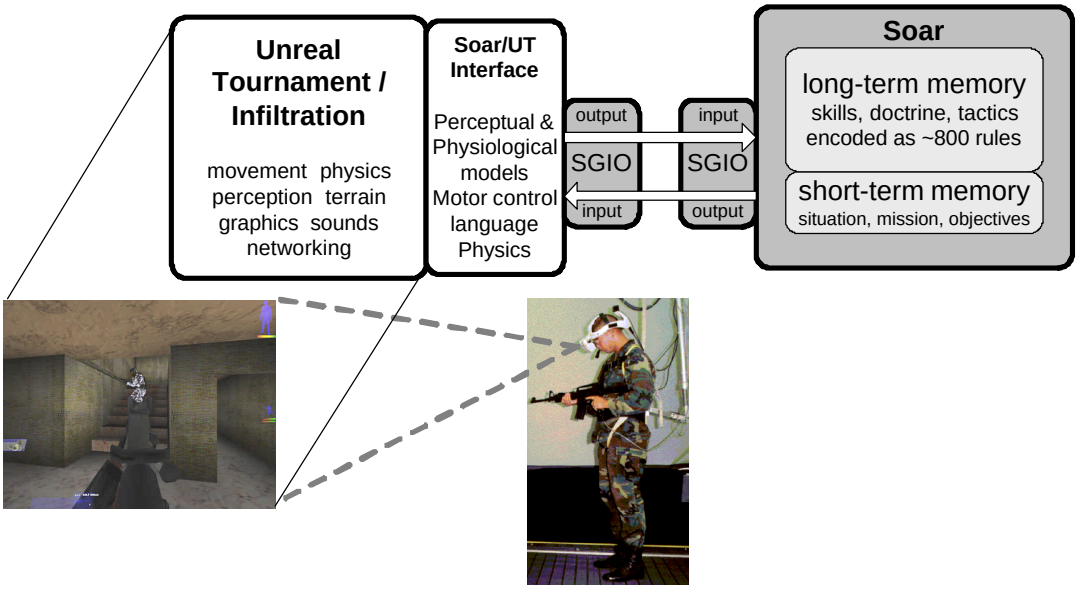

Figure 1. The MOUTBot System Architecture

1. **Compressed time scale**: In TacAir-Soar and RWA-Soar, the decision cycle could vary from .25 to .5 seconds without significant degradation in observable behavior. For MOUT, reaction times must be on the order of .1 seconds to support realistic behavior (e.g., avoid running into walls). Missions in TacAir-Soar and RWA-Soar were on the order of an hour, while for the MOUT domain, scenarios transpire in minutes.
2. **Loose mission structure, teamwork and coordination:** Missions could be very complex in the air domain with multiple phases to each mission. In addition, a complex hierarchical command structure, set by military doctrine, coordinated teamwork. For some missions, over 30 aircraft could be flying together. In the MOUT domain, the missions consist mostly of defending specific rooms, with some individual missions. Only small numbers of agents work together and there is little or no time for complex communication and preplanned coordination. In fact, one surprise was that the types of coordination required of the adversaries was so minimal, that using the coordination schemes developed for the air agents (Tambe, 1997) was unnecessary.
3. **Indoor spatial reasoning:** In the air domain, fine-grained spatial reasoning about terrain is unnecessary. For indoor combat, agents must understand room geometry of as well as the total topology of a building in order to set up attacks, ambushes, and retreats.
4. **Behavior variability:** In the air domain, the overall complexity of the virtual environment provided sufficient variability so that the behavior of TacAir-Soar agents was not predictable. However, this application requires a training environment where the trainees can be exposed to the same general situation many times. As a result, variability in behavior is much more important than in TacAir-Soar.

Another implementation of synthetic adversaries has been developed using ACT-R (Best, Lebiere, & Scarpinnatto, 2002). Their emphasis was to demonstrate the feasibility of using ACT-R for controlling synthetic characters and fine-grained spatial representations; our application was directly geared to the requirements of deployment.

Many commercial computer games simulate military combat (America's Army, Operation Flashpoint, Full Spectrum Warrior). The adversaries in these games are usually scripted or based on simple finite-state machines. Thus, they have limited autonomy, little ability to reason on the spatial aspects of tactical situations, and can be too predictable, possibly leading to negative training. However, more autonomous individual adversaries have also been developed for first-person perspective interactive computer games (Adobbati, Marshall, et al 2001; Laird, 2001a). These "bots" play "deathmatches", where the rules and weapons are significantly different than MOUT engagements. They do not meet the list of requirements listed above (no individual missions, limited coordination, behavior not constrained by real-world military doctrine and tactics). However, there is significant overlap in some of the underlying capabilities, enough so that the system described here was based in part on a bot developed in Soar to play Quake (Laird, 2001a, 2001b).

Overall System Design

This section introduces *MOUTBots*, the autonomous, intelligent opponents developed to meet the requirements.

Here, we briefly review supporting technologies and describe the overall system architecture.

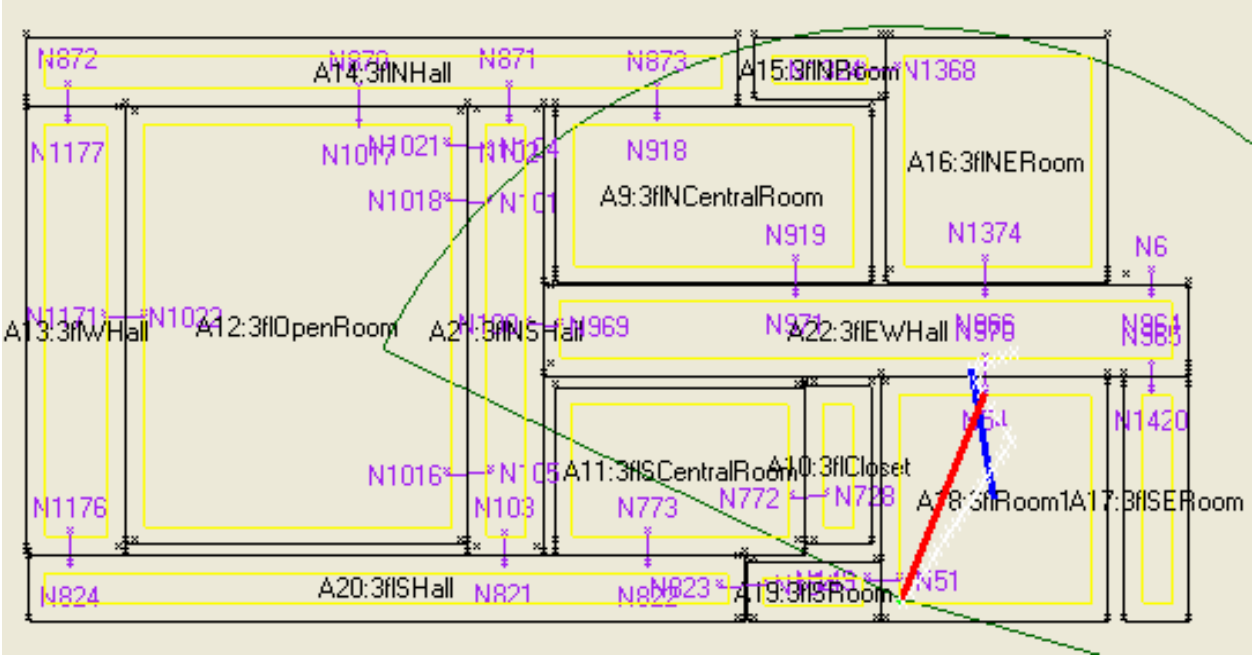

Figure 2. Internal spatial representation in the MOUTBots

Simulation Environment: Unreal Tournament

Figure 1 presents the system architecture. The simulation environment is realized in an extension of the commercial computer game Unreal Tournament (UT) called Infiltration[1]. The UT-Soar-Interface simulates the perception and motor actions of our agents in the environment and is connected to Soar via Soar General Input/Output (SGIO). Each agent is run as an independent copy of Soar, the cognitive component of a MOUTBot. Soar knowledge is encoded as rules. All of the agents are run on a single computer, with the human players running on their own computers, networked together.

Unreal Tournament is an off-the-shelf extendable 3D game engine that supports networked play. Infiltration provides graphics and models of modern weapons (e.g., M-16 rifles), more sophisticated models of the impact of wounds, and representations of modern army uniforms and equipment. All game physics and the game interface are coded in an internal scripting language. Using UT's environment editor, we created a synthetic training environment (a 3 story building with a variety of rooms and hallways) that presented the breadth of training situations as recommended by our subject matter experts.

Perception and Motor Control Module

We extended Infiltration to include more complete models of perception and motor actions. Visual perception is challenging because of the difficulty of sensing walls (and doors) in simulation environments. Processing the internal Unreal data structures to detect walls, doors and other environmental features would be computationally prohibitive. To circumvent these challenges, we introduced a number of simplifications based on an annotated map.

Figure 2 illustrates an example map. Each agent receives an annotated map with nodes that define the boundaries of each room. This allows an agent to determine the room in which it is located. Nodes are also placed in the map at connections (doors, windows) between areas. The MOUTBots use the nodes to build up a topological map of the building. The interface supports the motor actions required by adversaries, including walking, running, shooting (aiming, firing, reloading), grenade throwing, as well as postures other than standing (crouching, kneeling).

Cognitive Architecture: Soar

The cognitive component of the synthetic adversaries is implemented in the Soar cognitive architecture (Laird, Newell, & Rosenbloom, 1987; Newell, 1990). In Soar, all long-term knowledge is encoded as production rules, while the current situation (perception, situational awareness, mission information, and goals) is encoded in declarative data structures comprising its state. The rules in Soar match against the state and propose operators to apply to the current situation. Primitive operators send commands to the motor system. Complex operators are dynamically converted to subgoals that are then pursued by once again having rules select and apply operators in the subgoals, eventually resulting in operators that execute primitive

[1] Unreal Tournament was developed by Epic Games. Infiltration was developed by Sentry Studios.

actions. Recent versions of Soar now also include belief and goal maintenance mechanisms (Wray & Laird, 2003a). These mechanisms improve robustness and simplify knowledge development, contributing to our ability to encode a complex behavior system quickly.

Meeting the Requirements

The MOUTBot consists of over 25 major data structures and 800 production rules (which implement 120 operators). This section outlines some of the knowledge encoded in the MOUTBots and how this knowledge is applied to generate behavior, addressing each of the above requirements.

Competence

To create competent adversaries for MOUT training, we studied available literature (field manuals and historical accounts) and interviewed experts. We discovered there is little written doctrine for urban combat defense, so we relied heavily on human subject matter experts (SMEs). We developed synthetic adversaries that could play many different roles: defending a room, acting as a sentry (by watching through a window), defending a hallway, and acting as the leader of the group – giving commands to reposition, retreat or attack. As the project progressed, the training application began to focus on scenarios of only a single fire team of four trainees. Because attackers prefer at least a 3-to-1 force ratio for urban attack, these scenarios require only two adversaries working together to provide an appropriate training experience. However, our implemented system also supports larger scenarios with teams of five to eight adversaries defending a building.

In order to perform the required missions, the MOUTBots required the following capabilities:

- Situational awareness
  - Categorize situation: available weapons, ammo, enemy, health, level of incoming fire, etc.
  - Record/manage information about threats and friendly units
  - Identify tactically relevant topology of building
- Movement
  - Move within room, to door ways, and through doorways (run and walk)
  - Compute paths between rooms; determine visibility between rooms, note types of rooms, exits, etc.
  - Explore buildings and create an internal map
- Weapons handling & management
  - Reload, unjam weapon
  - Choose weapon based on situation
- Perform specific mission
  - Change mission on command from leader
  - Abandon mission
- Appropriate tactical and mission response
  - Atack with gun and grenade (and rocks)
  - Retreat/Hide and pop out
  - Defend a room
  - Roam
  - Watch for enemy entry (sentry)
  - Surrender
- Communication & coordination via realistic messages

This knowledge is represented as elaborated objects in memory and operators (implemented via production rules).

Objects. The agent uses object data structures to maintain situation awareness during execution of a scenario. For example, the **threat** data structure is used to maintain awareness about enemy (i.e., trainee) contacts. The threat structure includes information about the status of the threat and current or last known location. Additional situation-specific information may also be elaborated. For example, for a visible threat, the threat's body position, its direction of movement, whether or not it is firing, its egocentric location with respect to the agent (near, far; left, right; etc.) will be asserted. In some situations when the agent cannot directly observe a known threat, it may make (simple) predictions about the possible movement of the enemy which are stored with the threat information.

The building map is the most complex object maintained by the MOUTBot. The MOUTBot creates its map by exploring the building before an engagement. It uses map nodes to build an internal representation of the location and extent of rooms, doors, and windows. Figure 2 shows a visualization of the MOUTBot's internal representation of the map. The bot is in the lower right corner facing northeast. The semi-circle shows the bots potential visible areas (it can not see through walls). Nodes (represented as asterisks) are embedded in the map. The agent uses these nodes to define walls (black lines) and connections between rooms. The agent determines navigable areas within rooms (gray lines). During exploration, the MOUTBot also analyzes the map to determine paths between rooms, which are critical for some of its tactical analysis of threats and safe retreat paths. The MOUTBot can engage an enemy without a full map; however, because many of the tactics are dependent on map information, its behavior will be severely degraded without it. Thus, the image in Figure 2 is built up by the agent from its own exploration of the building environment.

Other important objects include **mission**, a description of the current mission and the achievement of objectives; and **self**, the current assessment of the agent's capabilities including health, available weapons, as well as any customizations that differentiate this agent from others.

Operators. Operators are used to take action in the world, establish goals, and record persistent facts about the current situation. For example, the **reload** operator performs an action in the environment. The MOUTBots do not count rounds as they are expended (in contrast to Dirty Harry, but in agreement with our experts). **Reload** is proposed after the agent attempts to fire and receives an error message from the simulator. This error message corresponds to the "click" of a trigger to an empty chamber. When reload is selected, a rule sends the **reload** command to the simulator. The simulator will respond by playing an animation

representing the reloading of the weapon and providing additional rounds to the weapon (assuming the agent has additional rounds available). During this time, the agent can do additional reasoning, but it cannot perform any other actions in the world. The reload error message can trigger other actions beside **reload**. For example, if a threat is near, other operators will be proposed so that the agent might choose to seek cover before reloading.

Record-threat is an example of an operator that maintains situation awareness by internally recording information, in this case, about other agents it has observed. It is invoked whenever a threat is encountered that the agent has not previously encountered, when it is loses contact with an agent (e.g., if the agent passes by a doorway), and when it encounters an agent it has encountered previously. The action of the **record-threat** operator is to add to or update the **threat** object with information about the enemy agent. In the current MOUTBot, the agent discriminates between individual enemies and makes no mistakes due to misclassification of a threat (e.g. fratricide), although introducing human error would require relatively simple elaborations of the model.

The **Retreat** operator requires many primitive actions spread out over time. When **retreat** is selected, a subgoal is automatically created (because there are not rules to directly apply it). In the subgoal, there are additional operators that perform the necessary actions, including determining where the greatest threat is (shown as the red line in Figure 2), picking a door to retreat through (away from the threat), abandoning the current objective, moving to a new room and then determining what objective should be pursued in that room (such as defending the room or further retreating). Some of these actions require subgoals as well (such as moving to a new room), and a subgoal stack is dynamically created. The dynamic context switching required by the MOUTBots did not require elaborate knowledge representations because Soar now includes mechanisms that dynamically capture dependencies between goals and performs the context switching via architectural mechanisms (Wray & Laird, 2003a).

Taskability

All agent knowledge (and consequently behavior) is independent of the specific mission and scenario; the same set of rules is used for all scenarios and all agents. In order to define specific scenario with unique roles for different MOUTBots, a user creates explicit mission tasking within a single "mission specification" production rule. This mission specification allows the user to indicate teammates, fire teams, and a commander(s), the type of mission (defensive/offensive), each MOUTBot's role in the mission (defend area/roam/sentry), initial areas to defend, places to which to retreat, and the type and direction of expected threats. Moreover, the leader MOUTBot can issue a limited set of commands to change the missions of other bots. For example, if there is an agent performing the sentry mission, once it sights an enemy approaching the building and informs others of the approaching threat, it might be instructed to move to another area and defend that area, causing it to terminate the sentry objective and initiate a defend objective.

Observational Fidelity

We did not attempt to make the MOUTBots as realistic as possible. The MOUT environment has many behavioral elements to consider for representation, among them doctrine and tactical knowledge, spatial and temporal knowledge, coordination and communication with other entities, courage and cowardice, indecision, leadership, startle responses, reaction to light, noise, smoke, debris, emotion, mood, physiological moderators, etc. The observational fidelity requirement provided a critical guide for determining what to represent among this imposing list. We concentrated on those elements observable to trainees as well as fundamental behaviors such as tactically appropriate movement, weapons handling, and communication. This focus allowed us to simplify non-observable behaviors. For example, trainees should not generally be able to observe opponents at-ease in a MOUT situation; thus, MOUTBots "at-ease" simply stand without moving, fidgeting, etc. Observational fidelity also allowed us to avoid detailed internal models when the behavioral role of the model is minimal. For example, although we implemented a fairly sophisticated ray-tracing model of peripheral vision, a plane-based model of visual perception was indistinguishable at the level of behavior from the ray-tracing model. Although it did not fully represent the complexities of human peripheral vision, the simpler model required significantly less computational resources.

Observational fidelity also mandates that any behavior that is observable should be represented. Thus, agents must not "cheat" (as in common in commercial computer games). Agents are limited by the terrain; they do not transport to other locations, disappear in firefights, etc. Additionally, they must coordinate as actual human combatants do, by shared, common knowledge, observation, and communication when necessary. This requirement also means that their behavior can be disrupted in the same way human behavior is and it means that the agents will make mistakes. For example, if one member of a MOUTBot fire team decides to exit after observing an enemy, it is possible that his partner may have not sighted the enemy and may miss his retreat, unless these events are communicated.

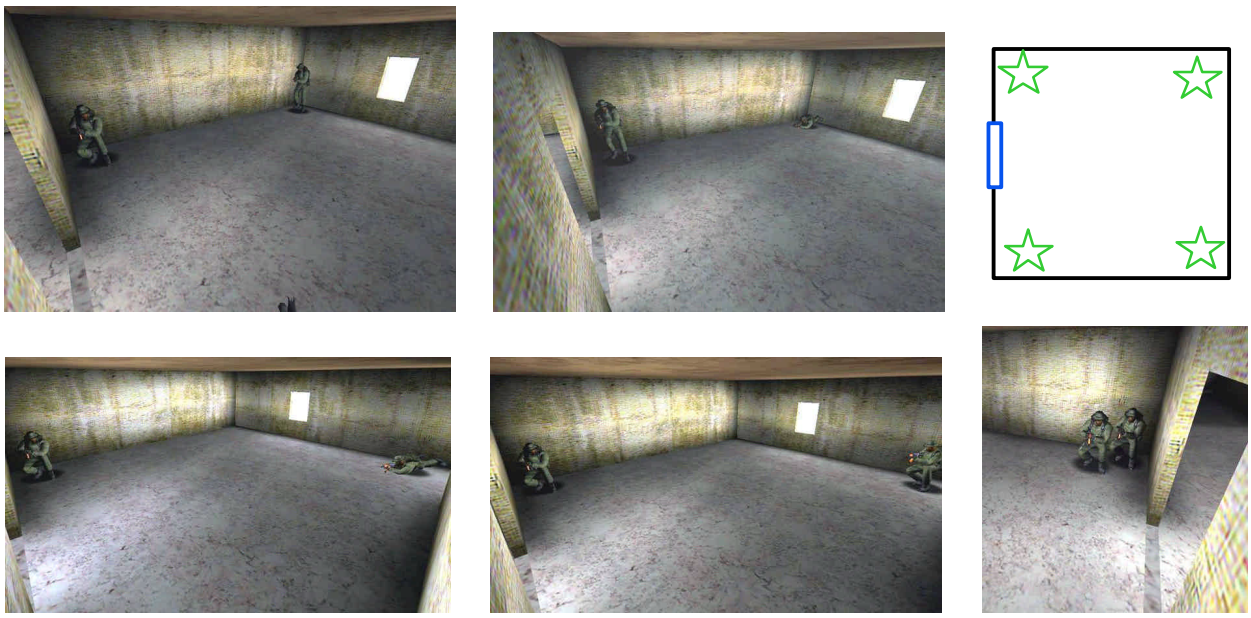

Figure 3. An example of variability in the MOUTBots: Choosing a location from which to defend

Behavior Variability

A difference in observed behavior when entities are placed in essentially the same situations is the essence of behavior variability. Capturing variability in synthetic entities is important because it prepares the trainee for the variability inherent in human behavior in the real world. As project SMEs stressed, it is critical to expose trainees to the breadth of skill levels in the opponent forces. Untrained forces may behave in ways that are non-optimal and even dangerous for themselves; however trainees must be prepared to quickly respond to such behavior with appropriate tactics, even if they would not use them against a highly trained opponent.

For this project, we focused on behavior variability in individual agents. If a single agent has little or no variability in its behavior, it becomes predictable, and a trainee may attempt to "game" an opponent inappropriately. Rather than modeling detailed knowledge differences that lead to different choices or potential sources of variability, our goal was to create mechanisms that better support variability in decision making within the agent architecture. Our hypothesis is that, long-term, it will be less expensive to introduce factors that influence the decision making process that can be generalized over many applications rather than attempting to program (or have an agent learn) knowledge differences. These *variability parameters* can be used to generate within-subject and across-subject variability without having to model the sources of variability explicitly. This hypothesis assumes that there are human behavior moderators that lead to variability, even when the knowledge of human participants is (more or less) the same.

Normally, we would encode the best or good choices for a specific situation. For example, one might only use a grenade at dynamic tactical junctures, or when faced with overwhelming firepower. Following this approach, multiple agents in the simulation and across multiple runs of the simulation all would exhibit similar behaviors. They would not use grenades until tactically appropriate situations. In reality, soldiers make different choices.

We extended Soar's basic knowledge representation and modified the Soar decision process to support more varied option selection. Productions that propose and prefer operators now include a numeric value, indicating a "probability" of selection when compared to other, equally preferable choices. When the options available are all equally preferable, the values for each option are averaged and then a random choice made from the normalized probability distribution of the averaged values.

This new selection mechanism requires a broader knowledge base than is strictly necessary. When variability is desired, the knowledge engineer must identify a range of options rather than one. For example, in the MOUTBots, target selection was initially based on proximity, which is a valid, realistic algorithm for selecting targets. To improve variability, we need to encode multiple target selection strategies and define simple probability distributions among these different strategies. In the long-term, agent development may focus on much more comprehensive efforts to describe and codify behaviors across many classes of subjects.

We implemented this simple within-subject approach to greater variation in decision making. Figure 3 shows an example of the resulting variability in the MOUTBots. In this scenario, two MOUTBots move to corner positions in the room. Agents also choose a body position (standing, crouched, prone) appropriate for the position. For example,

at positions near the covered doors, agents will stand or crouch, but not go prone. The figure shows some of the possible positions that the agents will take, given repeated execution of the identical scenario. Variability also plays a role in more dynamic decision making. For example, the agents have knowledge to recognize some tactically appropriate situations in which to use a grenade. The grenade selection knowledge was also encoded for regular engagements with a low probability. Thus, the user can be surprised when the MOUTBots unexpectedly throw a grenade, increasing unpredictability in the user experience.

This new mechanism is still being evaluated and, at this point, we can not claim that this design for variability in decision making provides human-like decision making. This is an empirical question and will require both data on human variability as well as experimentation to determine how it provides/fails to provide human-like variability.

Transparency

Because Soar is a symbolic architecture, its knowledge can be readily understood by non-technical users. For example, a MOUTBot goal might be to defend a room; a user could find a "defend-room" goal among the Soar agent's active goals. Although this representation facilitates transparency, alone it does not achieve it because some knowledge of the run-time aspects of the architecture is needed to extract this information. For the MOUTBot development, we used an application specific visualization tool that provides users the ability to see an agent's map of its terrain, where it believes other friendly and enemy agents are located, its desired movement and targeting goals, and a fine-grained view of its near-field environment (e.g., whether it was adjacent to a wall). Figure 2 showed a representation of the map portion of the tool. For a fielded application, we could couple the Soar agents with a tool specifically designed to bridge the gap between the transparency of Soar's representation and the extraction of this information from the agents themselves (Taylor et al, 2002), making it much easier for non-technical users to understand the decisions and behavior of Soar agents.

Minimal Computational Footprint

A significant direction Soar development over the past decade has been improving its performance and it has run as many as twenty simple bots in UT while maintaining the game's frame rate.(Laird et al., 2002). For the MOUTBot application, as mentioned previously, we used observational fidelity as a criterion for simplifying the models of perceptions and actions that would increase computational overhead. Another significant element of the MOUTBot's efficiency is the SGIO interface. SGIO provides both a socket interface for development and an option to compile Soar directly into other applications. Using the compiled connection, we were able to run 5 independent agents while maintaining a game frame rate of 20 Hz on a 1GHz Pentium III laptop with ½ GB RAM. While we did not perform more specific stress tests, this performance was more than acceptable for the target application, because only 1 or 2 MOUTBots were needed.

Limitations and Future Work

The MOUTBot is a prototype, a first approximation of the tremendous behavior space of this domain. It has a number of limitations, four of which are outlined below.

1. **Integration with higher-fidelity simulations**: Some behavior simplifications were dictated by Unreal Tournament's simulation limitations, rather than fundamental limits in the approach. For example, UT represents entity bodies as cylinders, rather than as articulating components. While there are some animations that show specific body movements (aiming, reloading, walking), it was not feasible to simulate realistically ones that were not (such as turning the head or communicating with gestures). When MOUTBots are integrated with a simulator that provides more fine-grained representations of the body, they will need to be extended or combined with other, existing technologies to capture the kinematics of body movement and to make control decisions over these features.
2. **Robust models of spatial reasoning**: The MOUTBots are dependent on the nodes embedded in the Unreal Tournament maps. As described previously, the agents need representations of walls and doors. In simulators where these representations are available, more general models of space and the ability to comprehend and predict concealment and cover locations will be required.
3. **Fault tolerant execution**: Extensions to the agent are needed so that it can better handle failures itself. In particular, the current agent lacks common sense knowledge that would allow it to reflect about its situation when it found itself unable to execute its desired task. Such models exist (e.g., Nielsen et al, 2002), so this problem is more of a knowledge integration challenge than a basic research problem.
4. **Requirements analysis & behavior validation**: Even with the behavior variability solutions described above, competent behavior covers only a narrow part of the overall behavior space. Fanaticism, "less" competent behaviors (possibly representing poorly trained guerillas or novices), additional errors in judgment and perception, etc. all could be modeled. In order to justify their cost, these extensions require a thorough understanding of user requirements in specific training applications. We attempted to obtain some of this information by presenting the MOUTBots at various stages during development to subject matter experts. These demonstrations resulted in valuable feedback (e.g., early demos led us to concentrate on variability as an overall requirement). However, we recognize the need for more formal validation techniques that would allow knowledge engineers to evaluate agent behaviors

more directly and quickly. Some recent progress has been made towards this goal, using the current MOUTBot prototype as a testbed (Wallace & Laird, 2002).

Conclusions

Achieving intelligent opponents that are fully autonomous, believable, and interactive, while exhibiting a variety of behaviors in similar circumstances, will require much additional research and development. However, we made significant progress towards these goals. Using computer game technology allowed us to develop the prototype MOUTBot entities without a high fidelity simulator. The Soar cognitive architecture provided efficient execution, allowing a full complement of opponents to run with the simulation on standard PC hardware. Soar also facilitated knowledge reuse, resulting in greater breadth, depth, and realism in behaviors. We also developed a general framework for supporting behavioral variability, and implemented portions of this framework in Soar. Thus, the MOUTBots, by combining the strengths of human behavior representation and computer game technology, demonstrate that realistic, autonomous, embodied intelligent agents can meet the requirements of interactive, virtual training.

Acknowledgments

This work was sponsored by the Office of Naval Research, VIRTE Program, contracts N00014-02-C-003 and N00014-02-1-0009. Kelly McCann and K.C. Pflugler provided expert opinion about urban combat behaviors. Portions of this work were reported previously (Wray, Laird, et al 2002; Wray & Laird 2003). The authors thank Randolph Jones and Jens Wessling for help in the development of the MOUTBot and associated software.

References

Adobbati, R., Marshall, A. N., Kaminka, G., Scholer, A., & Tejada, S. (2001). *Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research.* Paper presented at the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada.

Best, B. J., Lebiere, C., & Scarpinnatto, K. C. (2002). *Modeling Synthetic Opponents in MOUT Training Simulations Using the ACT-R Cognitive Architecture.* Paper presented at the 11th Conference on Computer Generated Forces and Behavior Representation, Orlando.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P. G., & Koss, F. V. (1999). Automated Intelligent Pilots for Combat Flight Simulation. *AI Magazine, 20*(1), 27-42.

Laird, J. E. (2001a). *It Knows What You're Going To Do: Adding Anticipation to a Quakebot.* Paper presented at the Fifth International Conference on Autonomous Agents (Agents 2001), Montreal, Canada.

Laird, J. E. (2001b). Using a Computer Game to Develop Advanced AI. *Computer, 34*(7), 70-75.

Laird, J. E., Assanie, M., Bachelor, B., Benninghoff, N., Enam, S., Jones, B., Kerfoot, A., Lauver, C., Magerko, B., Sheiman, J., Stokes, D., & Wallace, S. (2002, March). *A Test Bed for Developing Intelligent Synthetic Characters.* Paper presented at the AAAI 2002 Spring Symposium: Artificial Intelligence and Interactive Entertainment.

Laird, J. E., Jones, R. M., & Nielsen, P. E. (1998). *Lessons learned from TacAir-Soar in STOW-97.* Paper presented at the Seventh Conference on Computer Generated Forces and Behavioral Representation, Orlando, Florida.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*(3), 1-64.

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

Nielsen, P., Beard, J., Kiessel, J., & Beisaw, J. (2002). *Robust Behavior Modeling.* Paper presented at the 11th Computer Generated Forces Conference.

Tambe, M. (1997). Towards Flexible Teamwork. *Journal of Artificial Intelligence Research (JAIR), 7*, 83-124.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F. M., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. B. (1995). Intelligent Agents for Interactive Simulation Environments. *AI Magazine, 16*(1), 15-39.

Taylor, G., Jones, R. M., Goldstein, M., Fredericksen, R., & Wray, R. E. (2002). *VISTA: A Generic Toolkit for Agent Visualization.* Paper presented at the 11th Conference on Computer Generated Forces and Behavioral Representation, Orlando.

Wallace, S. A., & Laird, J. E. (2002). *Toward Automatic Knowledge Validation.* Paper presented at the Eleventh Conference on Computer Generated Forces and Behavioral Representation, Orlando, Florida.

Wray, R. E., & Laird, J. E. (2003a). An architectural approach to consistency in hierarchical execution. *Journal of Artificial Intelligence Research, 19*, 355-398.

Wray, R. E., & Laird, J. E. (2003b, May). *Variability in Human Behavior Modeling for Military Simulations.* Paper presented at the Conference on Behavior Representation in Modeling and Simulation, Scottsdale, AZ.

Wray, R. E., Laird, J. E., Nuxoll, A., & Jones, R. M. (2002). *Intelligent Opponents for Virtual Reality Training.* Paper presented at the Inter-service/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando.