AI Magazine Volume 26 Number 3 (2005) (© AAAI)

Synthetic Adversaries for Urban Combat Training

Robert E. Wray, John E. Laird, Andrew Nuxoll, Devvan Stokes, and Alex Kerfoot

■ This article describes requirements for synthetic adversaries for urban combat training and a prototype application, MOUTBots. 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. The article describes how these components aided the development of the prototype and presents an initial evaluation against competence, taskability, fidelity, variability, transparency, and efficiency requirements.

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 and 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 are situated in a virtual urban environment and tasked to clear a building that possibly contains simulated enemy soldiers. Virtual opponents are required to populate the environment and challenge the trainees.

This article describes the general requirements for virtual MOUT opponents and our development of synthetic adversaries to meet these requirements. The agents use the Soar cognitive architecture for reasoning and execution, and they interface to Unreal Tournament, a commercial game engine that provided an initial development environment. In order to simplify the behaviors for the prototype, we focused initially on MOUT behaviors within a building.

Requirements for Synthetic Adversaries

Six high-level requirements drive the implementation of intelligent synthetic adversaries for training: (1) competence, (2) taskability, (3) observational fidelity, (4) behavior variability, (5) transparency, and (6) minimal computational footprint.

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 multistoried building in teams of two to five 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.

Taskability: The agents must be able to be assigned new missions for different training scenarios, and they must change their objectives during an exercise. Behavior is not scripted or specific to a particular mission, terrain, or operational setting, providing flexibility for operational use.

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. It also may not be necessary to create realistic models of, for example, startle responses, as long as the observed behavior to a startling event is congruent with human behavior.

Behavior variability: Trainees should be exposed to many diverse experiences. By experiencing many different situations and opponent behaviors, the likelihood of encountering unexpected situations in actual performance is minimized. For example, a player in a first-person perspective computer game may learn that enemies always appear in similar places within rooms and can target those locations before new enemies arrive. This behavior improves game play but does not transfer to the real world; it does not reflect the unpredictability that actual human opponents would present. Diversity in actual human behavior arises from many sources, and attempts to model the opponents should account for the sources of that diversity systematically. Randomization or "noise" parameters are an insufficient solution to this problem.

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 domain terms.

Minimal computational footprint: Realistic behavior requires performance in real time. A majority of the computational processing is reserved for graphics and physics modeling. Thus, the synthetic adversaries are allocated only a small portion of the computational resources. There is a difficult trade-off between minimizing the computational footprint and other requirements for behavioral realism.

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 (Tambe et al. 1995, Jones et al. 1999). 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 through simulators (Laird, Jones, and 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: (1) compressed time scale; (2) loose mission structure, teamwork, and coordination; (3) indoor spatial reasoning; and (4) behavior variability. We explore each of these areas of differentiation in the following paragraphs.

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 (for example, to 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.

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, more than 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.

Indoor spatial reasoning: In the air domain, fine-grained spatial reasoning about terrain is unnecessary to meet the requirements. For indoor combat, agents must understand room geometry as well as the total topology of a building in order to set up attacks, ambushes, and retreats.

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 adver-

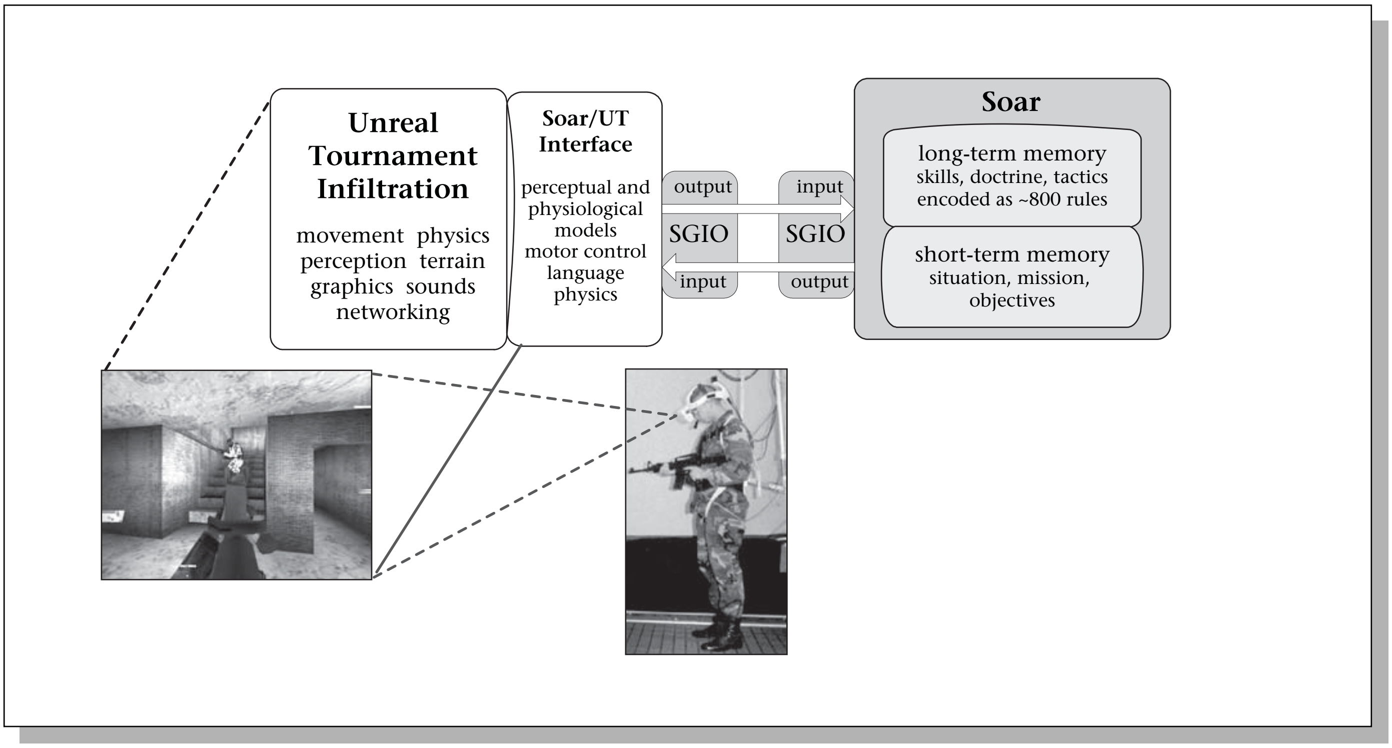

Figure 1. The MOUTBot System Architecture.

saries has been developed using ACT-R (Best, Lebiere, and 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 (for example, 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, have 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 et al. 2001, Laird 2001a). These "bots" play "deathmatches," where the rules and weapons are significantly different than in MOUT engagements. They do not meet the list of requirements listed previously (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 an agent 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 introduced previously. We briefly review supporting technologies and describe the overall system architecture.

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 Soar/UT Interface simulates the perception and motor actions of Soar agents in the environment and is connected to Soar through Soar general input/output (SGIO). SGIO is a high-level interface that provides two low-level interfaces: a socket-based interface (allowing network connection across separate computers or across separate threads on the same computer) and a C-language-based interface that connects all simulation components within a single process. The user can select the desired low-level interface at run time. This configurability allows a developer access to development and run-time tools during the agent development process while allowing the most efficient, within-process interface to be used in the embedded application.

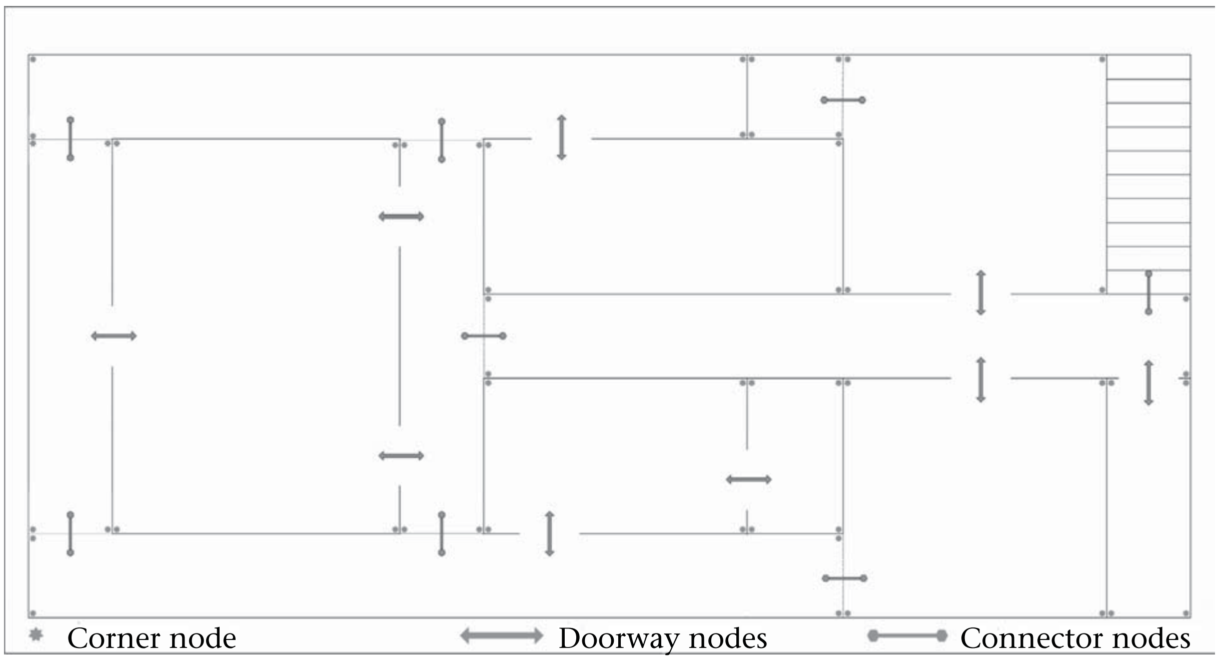

Figure 2. Internal Spatial Representation in the MOUTBots.

Each agent is implemented as an independent instance of Soar, the cognitive component of a MOUTBot, regardless of the interface selected.

Unreal Tournament is an off-the-shelf extensible three-dimensional (3D) game engine that supports networked play. Infiltration provides graphics and models of modern weapons (for example, 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 three-story building with a variety of rooms and hallways) that presented the breadth of training situations as recommended by 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 Tournament data structures to detect walls, doors, and other environmental features in real time is 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 ("corner nodes"). These nodes allow an agent to determine the room in which it is located. "Connector" nodes connect adjacent areas that do not have a doorway. The MOUTBots use these nodes for navigation between rooms. Unlike many computer game agents, movement is not restricted to nodes alone; MOUTBots can move freely within a room. Agents use the nodes to recognize the location of doors, windows (not shown in figure 2), connectors, and corners and can move directly through them when desired (for example, when moving from one room to another).

Each agent has the ability to comprehend map nodes and build its own annotated internal map of the building in which it is located. We generally provided a complete map to agents when developing other behaviors, representing a situation in which the occupants of a building already understood its spatial layout. However, this was simply a convenience; agents are not dependent on having a complete map when a scenario begins. The compiled map is actually generated by an agent that fully "explores" an Unreal Tournament level and records all node and link annotations in the map. Thus, the image in figure 2 is not necessarily provided by the simulation, but is constructed by an agent itself, from its own exploration of the building environment.

Grounding agents in their environment requires physiological models that can be used to influence behavior. Physiological variables may include temperature, fatigue, and hunger, among many others. Models of agent physiology are being developed for a role-playing computer game called Haunt2, also implemented in this software environment (Laird et al. 2002). As suggested in figure 1, UT has been extended so that agents have a primitive model of physiological responses to the environment and to their internal processing. The physiological models are implemented in UT and serve as input into the intelligent agents. For example, Haunt agents are aware of internal temperature and level of fatigue. Changes in the attributes can affect others, so that a significant drop in body temperature can increase fatigue. Physiological effects that have been implemented in Haunt include temperature, exertion, fatigue, sleepiness, hunger, and thirst (Laird et al. 2002). These physiological factors have not been included in the implemented MOUTBots, in part, because it was unclear what physiological factors were most important to model for the MOUT domain. However, the existing infrastructure provides the necessary tools for moderating behavior through physiological states. The functional role of the physical drives in MOUTBots is to provide increased realism and contribute to behavior variability.

Opponents move and act using controls similar to those used by a human game player (turn left and right; thrust forward, backward, left, and right). This style of control is more challenging for behavior development than directly moving to nodes or objects, but it provides more flexibility in controlling the character during the scenario. The interface supports the motor actions required by adversaries, including walking, running, shooting (aiming, firing, reloading), and grenade throwing, as well as standing, crouching, and kneeling postures.

Cognitive Architecture: Soar

The cognitive component of the synthetic adversaries is implemented in the Soar cognitive architecture (Laird, Newell, and Rosenbloom 1987; Newell 1990). All long-term knowledge (doctrine, tactics) 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 actions.

Meeting the Requirements

The MOUTBot consists of more than 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 previously mentioned requirements.

Competence

To create competent adversaries for MOUT training, we studied available literature (field manuals and historical accounts) and interviewed experts. 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 three to one 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 capabilities depicted in figure 3.

The knowledge to accomplish the abilities in figure 3 is represented as elaborated objects in memory and operators (implemented through production rules).

Objects. The agent uses object data structures to maintain situation awareness during execution of a scenario. Some of these are represented in figure 3. For example, the threat data structure is used to maintain awareness about enemy (that is, trainee) contacts. The threat structure includes information about the status of the threat and the 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; and so on) are 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. These predictions are stored with the threat information.

The building map is the most complex object maintained by the MOUTBot. As described previously, the MOUTBot can create the complete map by exploring the building before an engagement. It uses the nodes to build an internal representation of the location and extent of rooms, doors, and windows. 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 is severely degraded without it.

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 and 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 filmdom's 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 responds 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 are 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 loses contact with an agent (for example, 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 (for example 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 no rules to directly apply it). In the subgoal, there are additional operators that perform the necessary actions, including determining the location of the greatest known threat, 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

Situational Awareness
- Categorize situation: Available weapons, ammo, enemy, health, level of incoming fire, and so on
- Record or manage information about threats and friendly units
- Identify tactically relevant topology of building

Movement
- Move within room, to doorways, and through doorways (run and walk)
- Compute paths between rooms; determine visibility between rooms, note types of rooms, exits, and so on
- Explore buildings and create an internal map

Weapons Handling and 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
- Attack with gun and grenade (and rocks)
- Retreat/hide and pop out
- Defend a room
- Roam
- Watch for enemy entry (sentry)
- Surrender

Communication and Coordination

Figure 3. Examples of Objects in Agent Memory.

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 through the use of architectural mechanisms (Wray and Laird 2003a). These mechanisms improve robustness and simplify knowledge development, contributing to our ability to encode a complex behavior system quickly.

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 a 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 or commanders, 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 agents. 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. Thus, the agents are not only taskable by human users, they can also be tasked by other agents in the scenario.

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, and so on.

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 nonobservable 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, and so on. 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 is common in commercial computer games). Agents are limited by the terrain; they do not transport to other locations, disappear in firefights, and so on. Additionally, they must coordinate as actual human combatants do, by shared, common knowledge, observation, and communication when necessary. For example, in figure 4, when these agents initially encounter the enemy, they are both standing near the door. Each recognizes the possibility of fratricide given the angle to the threat, and the MOUTBot closest to the attacker crouches, while the agent in the rear shifts slightly to obtain a clearer line of fire. In this case, coordination occurs with no explicit communication between the entities; they coordinate out of shared, common knowledge of the domain. Because agents do not always explicitly communicate their actions, agents sometimes make mistakes. For example, if one member of this MOUTBot fire team decided to exit after first observing the attacker, it is possible that his partner may have not sighted the enemy and may miss his teammate's departure unless the sighting and the decision to retreat are explicitly communicated.

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 nonoptimal 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 better trained opponent.

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 is less expensive to introduce factors that influence the decision-making process and that can be generalized over many applications rather than attempt 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 use a grenade only 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 is 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, the MOUTBots' target selection algorithm 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.

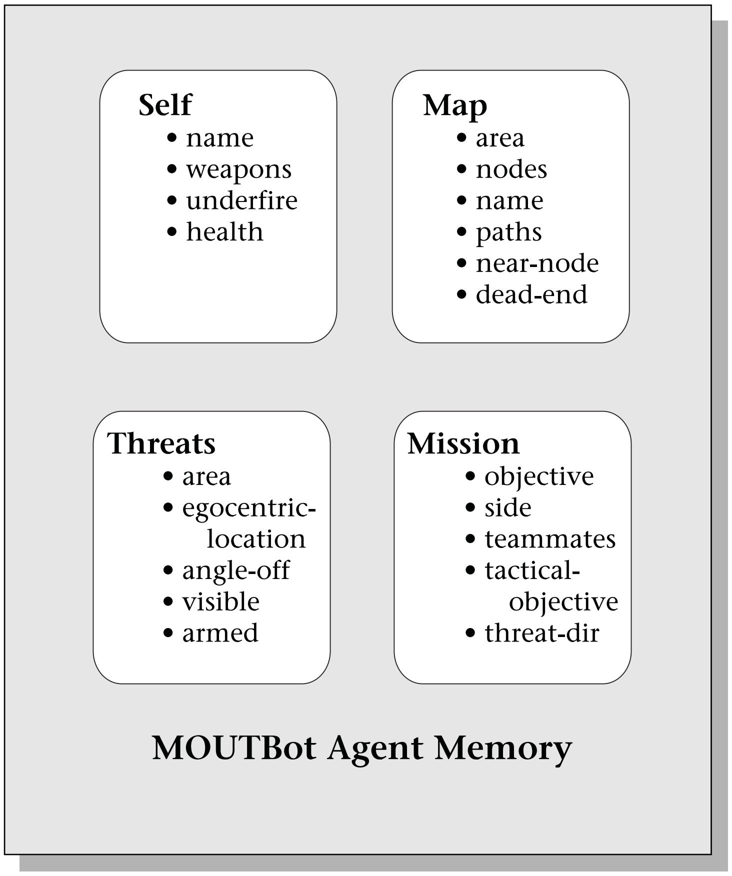

Figure 4. Coordination in the MOUTBots.

We implemented this simple within-subject approach to greater variation in decision making. Figure 5 shows an example of the resulting variability in the MOUTBots. In this scenario, two MOUTBots move to corner positions in the room, which are the most tactically appropriate positions to take within the room. Agents also choose a body position (standing, crouched, or prone) appropriate for the position. For example, at positions near the covered doors, agents stand or crouch, but do not go prone. The figure shows some of the possible positions that the agents can 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.

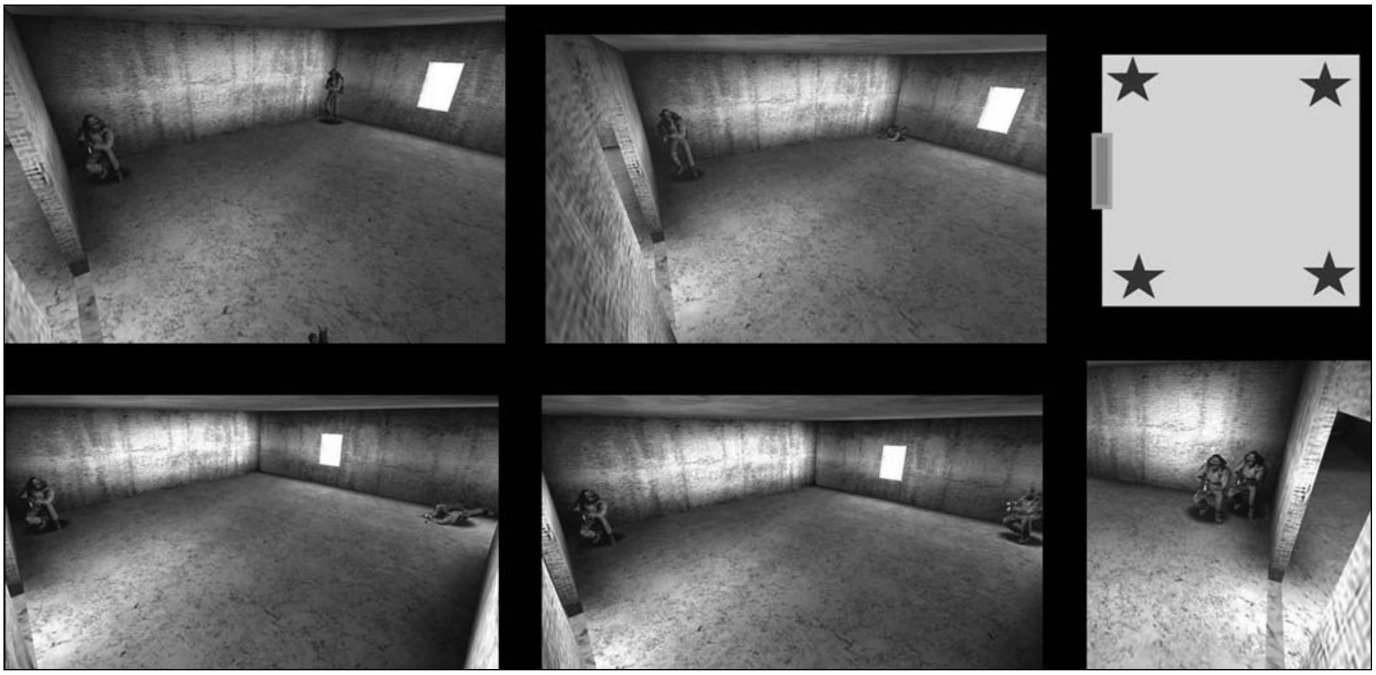

Figure 5. An Example of Variability in the MOUTBots: Choosing a Location from Which to Defend.

This new mechanism is still being evaluated, and at this point we cannot claim that this design for variability in decision making provides humanlike decision making. This is an empirical question and requires both data on human variability as well as experimentation to determine how it provides or fails to provide humanlike variability. Recent research is also exploring reinforcement learning as a mechanism to allow an agent to tune the weights of its actions with experience (Nason and Laird 2004).

Transparency

Because Soar is a symbolic architecture, its knowledge can be readily understood by nontechnical 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 knowledge of the run-time 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 (for example, whether it was adjacent to a wall). Figure 2 illustrated 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 nontechnical users to understand the decisions and behavior of Soar agents.

Minimal Computational Footprint

A significant focus of Soar development over the past decade has been the improvement of its performance. Soar has run as many as 20 simple bots in UT while maintaining an acceptable game 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. Using the compiled connection, we were able to run 5 independent agents while maintaining a game frame rate of 20 hertz on a 1-gigahertz Pentium III laptop with .5 gigabytes of random-access memory. 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 (1) integration with higher-fidelity simulations; (2) robust models of spatial reasoning; (3) fault-tolerant execution; and (4) requirements for analysis and behavior validation. These limitations are outlined in greater detail in the following paragraphs.

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.

Robust models of spatial reasoning: The MOUTBots are dependent on the nodes embedded in the Unreal Tournament maps. As described previously, the agents require better representations of walls and doors and of space more generally. 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.

Fault-tolerant execution: Extensions to the agent are needed so that it can better handle failures itself. In particular, the current agent lacks commonsense knowledge that would allow it to reflect about its situation when it found itself unable to execute its desired task. Models exist that provide this kind of capability (for example, Nielsen et al. 2002). In addition to the basic commonsense reasoning research challenges, we must also develop better techniques for integrating these models with existing domain knowledge (such as the MOUTBots' knowledge representations covering doctrine and tactics).

Requirements for analysis and behavior validation: Even with the behavior variability solutions described previously, 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, and so on, 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 (for example, early demos led us to concentrate on variability as an essential 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 MOUTBot prototype as a testbed (Wallace and Laird 2003).

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 within 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 et al. 2002, Wray and Laird 2003b). The authors thank Randolph Jones and Jens Wessling for help in the development of the MOUTBot and associated software.

Note

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

References

Adobbati, R.; Marshall, A. N.; Kaminka, G.; Scholer, A.; and Tejada, S. 2001. Gamebots: A 3D Virtual World Test-Bed for Multi-Agent Research. Paper presented at the Second International Workshop on Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, Montreal, Canada, May 28–June 1.

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

Jones, R. M.; Laird, J. E.; Nielsen, P. E.; Coulter, K. J.; Kenny, P. G.; and 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. In *Proceedings of the Fifth International Conference on Autonomous Agents* (Agents 2001), 385–392. New York: Association for Computing Machinery.

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.; and Wallace, S. 2002. A Test Bed for Developing Intelligent Synthetic Characters. In Artificial Intelligence and Interactive Entertainment: Papers from the AAAI Spring Symposium. Technical Report SS-02-01, American Association of Artificial Intelligence, Menlo Park, CA.

Laird, J. E.; Jones, R. M.; and 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, FL.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An Architecture for General Intelligence. *Artificial Intelligence* 33(3): 1–64.

Nason, S., and Laird, J. E. 2004. Soar-RL: Integrating Reinforcement Learning with Soar. Paper presented at the International Conference on Cognitive Modeling, Pittsburgh, PA, July 30–August 1.

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

Nielsen, P.; Beard, J.; Kiessel, J.; and Beisaw, J. 2002. Robust Behavior Modeling. Paper presented at the Eleventh Conference on Computer Generated Forces and Behavior Representation. Orlando, FL, May 7–9.

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.; and 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.; and Wray, R. E. 2002. VISTA: A Generic Toolkit for Agent Visualization. Paper presented at the Eleventh Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, May 7–9.

Wallace, S. A.; and Laird, J. E. 2003. Behavior Bounding: Toward Effective Comparisons of Agents and Humans. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Amsterdam: Elsevier Science Publishers.

Wray, R. E., and Laird, J. E. 2003a. An Architectural Approach to Consistency in Hierarchical Execution. *Journal of Artificial Intelligence Research (JAIR)* 19: 355–398.

Wray, R. E., and Laird, J. E. 2003b. Variability in Human Behavior Modeling for Military Simulations. Paper presented at the 2003 Conference on Behavior Representation in Modeling and Simulation, Scottsdale, AZ, May 12–15.

Wray, R. E.; Laird, J. E.; Nuxoll, A.; and 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, FL., Dec. 2–5.

Robert E. Wray is chief scientist at Soar Technology. He earned a Ph.D. in computer science from the University of Michigan where he developed novel techniques for maintaining logical consistency in agent reasoning systems. His innovations are incorporated in the Soar architecture. At Soar Technology, he leads R&D projects in many areas of artificial intelligence, including agent-based systems and agent architectures, machine learning, cognitive science, and knowledge representation and ontology.

John E. Laird is a professor of electrical engineering and computer science at the University of Michigan. He received his B.S. from the University of Michigan in 1975 and his Ph.D. in computer science from Carnegie Mellon University in 1983. He is one of the original developers of the Soar architecture and leads its continued development and evolution. He is a Fellow of AAAI.

Andrew Nuxoll is a Ph.D. candidate in the Electrical Engineering and Computer Science department at the University of Michigan in Ann Arbor. He received his B.S. in computer science from Rose-Hulman Institute of Technology in Terre Haute, Indiana, in 1991 and subsequently spent six years in industry as a professional software engineer before arriving at the University of Michigan. He is currently researching task independent episodic memory for AI agents.

Devvan Stokes has assisted John Laird for four years at the University of Michigan's Advanced Technology Laboratory. He has integrated the Soar cognitive architecture with several computer games and simulation environments and has worked as well on the software interface to Soar.

Alex Kerfoot worked on computer-game environments and Soar applications in those environments while an undergraduate student at the University of Michigan. He now works at Electronic Arts.