Agents that Learn to Explain Themselves

W. Lewis Johnson
USC / Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695
johnson@isi.edu

Abstract

Intelligent artificial agents need to be able to explain and justify their actions. They must therefore understand the rationales for their own actions. This paper describes a technique for acquiring this understanding, implemented in a multimedia explanation system. The system determines the motivation for a decision by recalling the situation in which the decision was made, and replaying the decision under variants of the original situation. Through experimentation the agent is able to discover what factors led to the decisions, and what alternatives might have been chosen had the situation been slightly different. The agent learns to recognize similar situations where the same decision would be made for the same reasons. This approach is implemented in an artificial fighter pilot that can explain the motivations for its actions, situation assessments, and beliefs.

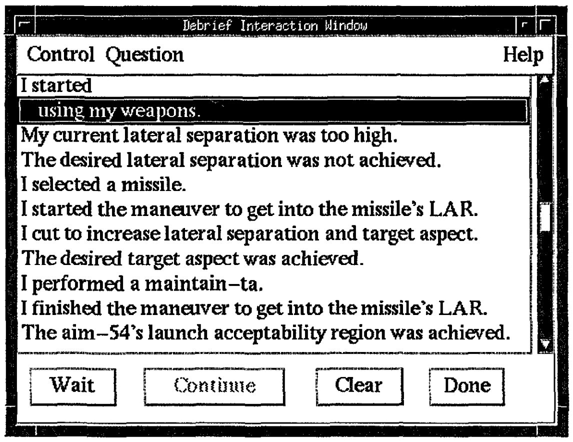
Introduction

Intelligent artificial agents need to be able to provide explanations and justifications for the actions that they take. This is especially true for computer-generated forces, i.e., computer agents that operate within battlefield simulations. Such simulations are expected to have an increasingly important role in the evaluation of missions, tactics, doctrines, and new weapons systems, and in training (Jones 1993). Validation of such forces is critical—they should behave as humans would in similar circumstances. Yet it is difficult to validate behavior through external observation; behavior depends upon the agent's assessment of the situation and its changing goals from moment to moment. Trainees can greatly benefit from automated forces that can explain their actions, so that the trainees can learn how experts behave in various situations. Potential users of computer-generated forces therefore attach great importance to explanation, just as potential users of computer-based medical consultation systems do (Teach & Shortliffe 1984).

Explanations based on traces of rule firings or paraphrases of rules tend not to be successful (Davis 1976; Swartout & Moore 1993; Clancey 1983b). They contain too many implementation details, and lack information about the domain and about rationales for the design of the system. More advanced explanation techniques encode domain knowledge and problem-solving strategies and employ them in problem solving either as metarules (Clancey 1983a) or in compiled form (Neches, Swartout, & Moore 1985). In the computer-generated forces domain, however, problem-solving strategies and domain knowledge representations are matters of current research. An intelligent agent in such a domain must integrate capabilities of perception, reactive problem solving, planning, plan recognition, learning, geometric reasoning and visualization, among others, all under severe real-time constraints. It is difficult to apply meta-level or compilation approaches in such a way that all of these requirements can be met at once.

This paper describes a system called Debrief that takes a different approach to explanation. Explanations are constructed after the fact by recalling the situation in which a decision was made, reconsidering the decision, and through experimentation determining what factors were critical for the decision. These factors are critical in the sense that if they were not present, the outcome of the decision process would have been different. Details of the agent's implementation, such as which individual rules applied in making the decision, are automatically filtered out. It is not necessary to maintain a complete trace of rule firings in order to produce explanations. The relationships between situational factors and decisions are learned so that they can be applied to similar decisions.

This approach of basing explanations on abstract associations between decisions and situational factors has similarities to the REX system (Wick & Thompson 1989). But while REX requires one to create a separate knowledge base to support explanation, Debrief automatically learns much of what it needs to know to generate explanations. The approach is related to techniques for acquiring domain models through experimentation (Gil 1993), except that the agent learns to model not the external world, but itself.



I started
using my weapons.
My current lateral separation was too high.
The desired lateral separation was not achieved.
I selected a missile.
I started the maneuver to get into the missile's LAR.
I cut to increase lateral separation and target aspect.
The desired target aspect was achieved.
I performed a maintain-ta.
I finished the maneuver to get into the missile's LAR.
The aim-54's launch acceptability region was achieved.

Figure 1: Part of a an event summary

Debrief is implemented as part of the TacAir-Soar fighter pilot simulation (Jones *et al.* 1993). Debrief can describe and justify decisions using a combination of natural language and diagrams. It is written in a domain-independent fashion so that it can be readily incorporated into other intelligent systems. Current plans call for incorporating it into the REACT system, an intelligent assistant for operators of NASA Deep Space Network ground tracking stations (Hill & Johnson 1994).

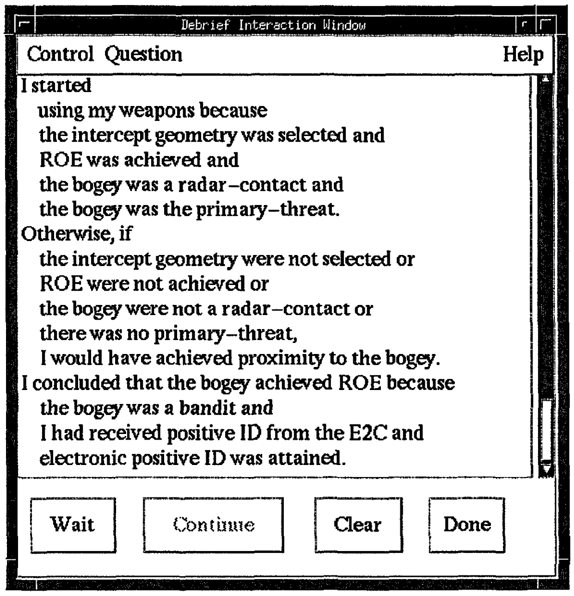
An Example

Consider the following scenario. A fighter is assigned a Combat Air Patrol (CAP) mission, i.e., it should fly a loop pattern, scanning for enemy aircraft. During the mission a bogey (an unknown aircraft) is spotted on the radar. The E2C, an aircraft whose purpose is to scan the airspace and provide information to the fighters, confirms that the bogey is hostile. The fighter closes in on the bogey, fires a missile which destroys the bogey, and then resumes its patrol.

After each mission it is customary to debrief the pilot. The pilot is asked to describe the engagement from his perspective, and explain key decisions along the way. The pilot must justify his assessments of the situation, e.g., why the bogey was considered a threat.

TacAir-Soar is able to simulate pilots executing missions such as this, and Debrief is able to answer questions about them. TacAir-Soar controls a simulation environment called ModSAF (Calder *et al.* 1993) that simulates the behavior of military platforms. TacAir-Soar receives information from ModSAF about aircraft status and radar information, and issues commands to fly the simulated aircraft and employ weapons. After an engagement users can interact with Debrief to ask questions about the engagement.

The following is a typical interaction with Debrief.



I started
using my weapons because
the intercept geometry was selected and
ROE was achieved and
the bogey was a radar-contact and
the bogey was the primary-threat.
Otherwise, if
the intercept geometry were not selected or
ROE were not achieved or
the bogey were not a radar-contact or
there was no primary-threat,
I would have achieved proximity to the bogey.
I concluded that the bogey achieved ROE because
the bogey was a bandit and
I had received positive ID from the E2C and
electronic positive ID was attained.

Figure 2: Explanations of the agent's decisions

Questions are entered through a window interface, by selecting a type of question and pointing to the event or assertion that the question refers to. The first question selected is of type Describe-Event, i.e., describe some event that took place during the engagement; the event chosen is the entire mission. Debrief then generates a summary of what took place during the mission. The user is free to select statements in the summary and ask follow-on questions about them.

Figure 1 shows part of a typical mission summary. One of the statements in the summary, "I started using my weapons," has been selected by the user, so that a follow-on question may be asked about it. Figure 2 shows the display at a later point in the dialog, after follow-on questions have been asked. First, a question of type Explain-Action was asked of the decision to employ weapons, i.e., explain why the agent chose to perform this action. The explanation appears in the figure, beginning with the sentence "I started using my weapons because the intercept geometry was selected and..." Debrief also lists an action that it did not take, but might have taken under slightly different circumstances: flying toward the bogey to decrease distance.

One can see that the agent's actions are motivated largely by previous assessments and decisions. The bottom of Figure 2 shows the answer to a follow-on question relating to one of those assessments, namely "ROE was achieved,"[1] Debrief lists the following fac-

[1]ROE stands for Rules of Engagement, i.e., the conditions under which the fighter is authorized to engage the enemy.

tors: the bogey was known to be hostile (i.e., a "bandit"), the bogey was identified through electronic means and confirmation of the identification was obtained from the E2C.

In order to answer such questions, Debrief does the following. First, it recalls the events in question and the situations in which the events took place. When summarizing events, it selects information about the intermediate states and subevents that should be presented, selects appropriate media for presentation of this information (the graphical display and/or natural language), and then generates the presentations. To determine what factors in the situation led to the action or conclusion, Debrief invokes the TacAir-Soar problem solver in the recalled situation, and observes what actions the problem solver takes. The situation is then repeatedly and systematically modified, and the effects on the problem solver's decisions are observed. Beliefs are explained by recalling the situation in which the beliefs arose, determining what decisions caused the beliefs to be asserted, and determining what factors were responsible for the decisions.

Implementation Concerns

Debrief is implemented in Soar, a problem-solving architecture that implements a theory of human cognition(Newell 1990). Problems in Soar are represented as goals, and are solved within problem spaces. Each problem space consists of a state, represented as a set of attribute-value pairs, and a set of operators. All processing in Soar, including applying operators, proposing problem spaces, and constructing states, is performed by productions. During problem solving Soar repeatedly selects and applies operators to the state. When Soar is unable to make progress, it creates a new subgoal and problem space to determine how to proceed. Results from these subspaces are saved by Soar's chunking mechanism as new productions, which can be applied to similar situations.

The explanation techniques employed in Debrief are not Soar-specific; however, they do take advantage of certain features of Soar.

- The explicit problem space representation enables Debrief to monitor problem solving when constructing explanations.
- Since Soar applications are implemented in production rules, it is fairly straightforward to add new rules for explanation-related processing.
- Learning enables Debrief to reuse the results of previous explanation processing, and build up knowledge about the application domain.

The current implementation of Debrief consists of thirteen Soar problem spaces. Two are responsible for inputing questions from the user, three recall events and states from memory, four determine the motivations for actions and beliefs, three generate presentations, and one provides top-level control. The following sections describe the system components involved in determining motivations for decisions and beliefs; other parts of the system are described in (Johnson 1994).

Memory and Recall

In order for Debrief to describe and explain decisions, it must be able to recall the decisions and the situations in which they occurred. In order words, the agent requires an episodic memory. Debrief includes productions and operators that execute during the problem solving process in order to record episodic information, and a problem space called Recall-State that reconstructs states using this episodic information.

The choice of what episodic information to record is determined by a specification of the agent's working memory state. This specification identifies the state attributes that are relevant for explanation, and identifies their properties, e.g., their cardinality and signature, and how the attribute values may change during problem solving. In order to apply Debrief to a new problem solver, it is necessary to supply such a specification for the contents of the problem solver's working memory, and indicate which operators implement decisions what should be explainable. However, it is not necessary to specify how the problem solver uses its working memory in making decisions—that is determined by Debrief automatically.

When the problem solver applies an operator that as marked as explainable, Debrief records the operator application in a list of events that took place during the problem solving. It also records all attribute values that have changed since the last problem solving event that was recorded.

Debrief then builds chunks that associate the state changes with the problem solving event. Once these chunks are built, the state changes can be deleted from working memory, because the chunks are sufficient to enable Debrief to recall the working memory state. During explanation, when Debrief needs to recall the state in which a problem solving event occurred, it invokes the Recall-State problem space. This space reconstructs the state by proposing possible attribute values; the chunks built previously fire, selecting the value that was associated with the event. Recall-State aggregates these values into a copy of the state at the time of the original event, and returns it. This result is chunked as well, enabling Debrief immediately to recall the state associated with the event should it need to refer back to it in the future. This process is an instance of data chunking, a common mechanism for knowledge-level learning in Soar systems (Rosenbloom, Laird, & Newell 1987).

Debrief thus makes extensive use of Soar's long term memory, i.e., chunks, in constructing its episodic memory. In a typical TacAir-Soar run several hundred such chunks are created. This is more economical than simply recording a trace of production firings, since over

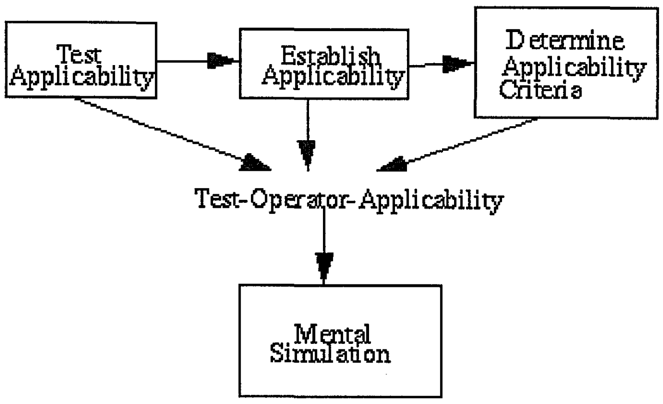


Figure 3: The process of evaluating decisions

6000 productions fire in a typical TacAir-Soar run. Since Soar has been shown be able to handle memories containing hundreds of thousands of chunks (Doorenbos 1993), there should be little difficulty in scaling up to more complex problem solving applications.

Explaining Actions and Conclusions

Suppose that the user requests the motivation for the action "I started using my weapons." Debrief recalls the type of event involved, operator that was applied, the problem space in which it was applied, and the problem solving state. In this case the event type is Start-Event, i.e., the beginning of an operator application, the operator is named Employ-Weapons, and the problem space is named Intercept. The situation was one where the agent had decided to intercept the bogey, and had just decided what path to follow in performing the intercept (called the intercept geometry).

Analysis of recalled events such as this proceeds as shown if Figure 3. The first step, testing applicability, verifies that TacAir-Soar would select an Employ-Weapons operator in the recalled state. An operator called Test-Operator-Applicability performs the verification, by setting up a "mental simulation" of the original decision, and monitoring it to see what operators are selected.

This initial test of operator applicability is important for the following reasons. State changes are not recorded in episodic memory until the operator has already been selected. The operator might therefore modify the state before Debrief has a chance to save it, making the operator inapplicable. This is not a problem in the case of Employ-Weapons, but if it were Debrief would attempt to establish applicability, which involves recalling the state immediately preceding the state of the event, and trying to find an interpolation of the two states in which the operator would be selected. But even when recalling the precise problem solving state is not a problem, verifying applicability is useful because it causes chunks to be built that facilitate subsequent analysis.

After a state has been found in which the recalled operator is applicable, the next step is to determine applicability criteria, i.e., identify what attributes of the state are responsible for the operator being selected. This also involves applying the Test-Operator-Applicability operator to construct mental simulations.

Mental simulation

Given the problem space Intercept, the recalled state, the operator Employ-Weapons, and the decision Start-event(Employ-Weapons), Test-Operator-Applicability operates as follows. It creates an instance of the Intercept problem space as a subspace, and assigns as its state a copy of the recalled state. The working memory specification described above is helpful here: it determines which attributes have to be copied. This state is marked as a simulation state, which activates a set of productions responsible for monitoring mental simulations. Test-Operator-Applicability copies into the simulation state the event and the category of decision being evaluated. There are three such categories: perceptions, which recognize and register some external stimulus, conclusions, which reason about the situation and draw inferences from it, and actions, which are operations that have some effect on the external world. Employ-Weapons is thus an action. The Intercept problem space is disconnected from external sensors and effectors (the ModSAF simulator), so that mental simulation can be freely performed. Execution then begins in the problem space. The first operator that is selected is Employ-Weapons. The monitoring productions recognize this as the desired operator, return a flag to the parent state indicating that the desired event was observed, and the mental simulation is terminated. If a different operator or event had been selected instead, Debrief would be checked to see if it is of the same category as the expected operator, i.e., another action. If not, simulation is permitted to continue; otherwise simulation is terminated and the a description of the operator that applied instead is returned.

Whenever a result is returned from mental simulation, a chunk is created. Such chunks may then be applicable to other situations, making further mental simulation unnecessary. Figure 4 shows the chunk that is formed when Debrief simulates the selection of the Employ-Weapons operator. The conditions of the chunk appear before the symbol $\longrightarrow$ and actions follow. Variables are symbols surrounded by angle brackets, and attributes are preceded by a carat ($\wedge$). The conditions include the expected operator, Employ-Weapons, the problem space, Intercept, and properties of the state, all properties of the bogey. If the operator is found to be inapplicable, a different chunk is produced, that indicates which operator is selected instead of the expected one.

```
(sp chunk-230 :chunk
  (goal <g1> ^operator <o1> ^state <s1>)
  (<o1> ^name test-operator-applicability
        ^expected-operator employ-weapons
        ^expected-step *none*
        ^problem-space intercept)
  (<s1> ^simulated-state <r1>)
  (<r1> ^local-state <l1>)
  (<l1> ^bogey <b1>)
  (<b1> ^intention known-hostile
        ^roe-achieved *yes*
        ^intercept-geometry-selected *yes*
        ^contact *yes*)
  (<l1> ^primary-threat <b1>)
-->
  (<s1> ^applicable-operator employ-weapons))
```

Figure 4: An example chunk

These chunks built during mental simulation have an important feature—they omit the details of how the operator and problem space involved is implemented. This is an inherent feature of the chunking process, which traces the results of problem solving in a problem space back to elements of the supergoal problem space state. In this case the state recalled from episodic memory is the part of the supergoal problem space state, so elements of the recalled state go into the left hand side of the chunk.

Determining the cause for decisions

At this point it would be useful to examine the chunks built during mental simulation in order to proceed to generate the explanation. Unfortunately, productions in a Soar system are not inspectable within Soar. This limitation in the Soar architecture is deliberate, reflecting the difficulty that humans have in introspecting on their own memory processes. It does not a serious problem for Debrief, because the chunks built during mental simulation can be used to recognize which attributes of the state are significant.

The identification of significant attributes is performed in the Determine-Applicability-Criteria problem space, which removes attributes one by one and repeatedly applies Test-Operator-Applicability. If a different operator is selected, then the removed attribute must be significant. If the value of a significant attribute is a complex object, then each attribute of that object is analyzed in the same way; the same is true for any significant values of those attributes. Meanwhile, if the variants resulted in different operators being selected, the applicability criteria for these operators are identified in the same manner. This generate-and-test approach has been used in other Soar systems to enlist recognition chunks in service of problem solving (Vera, Lewis, & Lerch 1993), and is similar to Debrief's mechanism for reconstructing states from episodic memory. Since the state representations are hierarchically organized, the significant attributes are found quickly.

If chunking were not taking place, Debrief would be performing a long series of mental simulations, most of which would not yield much useful information. But the chunks that are created help to ensure that virtually every mental simulation uncovers a significant attribute, for the following reason. Subgoals are created in Soar only when impasses occur. Test-Operator-Applicability instantiates the mental simulation problem space because it tries to determine whether the recalled operator is applicable, is unable to do so, and reaches an impasse. When chunks such as the one in Figure 4 fire, they assert that the operator is applicable, so no impasse occurs. Mental simulation thus occurs only in situations that fail to match the chunks that have been built so far. In the case of the Employ-Weapons operator, a total of seven mental simulations of variant states are required: two to determine that the bogey is relevant, and five to identify the bogey's relevant attributes.

Furthermore, even these mental simulations become unnecessary as Debrief gains experience explaining missions. Suppose that Debrief is asked to explain a different Employ-Weapons event. Since most of the significant features in the situation of this new event are likely to be similar to the significant features of the previous situation, the chunks built from the previous mental simulations will fire. Mental simulation is required for the situational features that are different, or if the operator was selected for different reasons.

Two kinds of chunks are built when Determine-Applicability-Criteria returns its results. One type identifies all of the significant features in the situation in which the decision was made. The other type identifies an operator that might have applied instead of the expected operator, and the state in which the operator applies. These chunks are created when mental simulation determines that an operator other than the expected one is selected. Importantly, the chunks fire whenever a similar decision is made in a similar situation. By accumulating these chunks Debrief thus builds an abstract model of the application domain, associating decisions with their rationales and alternatives. The problem solver's performance-oriented knowledge is reorganized into a form suited to supporting explanation.

Performing mental simulation in modified states complicates mental simulation in various respects. The result of deleting an attribute is often the selection of an operator in mental simulation to reassert the same attribute. Debrief must therefore monitor the simulation and detect when deleted attributes are being reasserted. The modified state may cause the problem solver to fail, resulting in an impasse. Mental simulation must therefore distinguish impasses that are a normal result of problem solving from impasses that suggest that the problem solver is in an erroneous state.

There is one shortcoming of the analysis technique described here. Chunking in Soar cannot always backtrace through negated conditions in the left hand sides of productions. Therefore if the problem solver opted for a decision because some condition was *absent* in the situation, Debrief may not be able to detect it. Developers of Soar systems get around this problem in chunking by using explicit values such as `*unknown*` to indicate that information is absent. This same technique enables Debrief to identify the factors involved.

Relationship to other exploratory learning approaches

The closest correlate to Debrief's decision evaluation capability is Gil's work on learning by experimentation (Gil 1993). Gil's EXPO system keeps track of operator applications, and the states in which those operators were applied. If an operator is found to have different effects in different situations, EXPO compares the states to determine the differences. Another system by Scott and Markovich (Scott & Markovich 1993) performs an operation on instances of a class of objects, to determine whether it has different effects on different members of the class. This enables it to discover discriminating characteristics within the class.

Some exploratory learning systems, such as Rajamoney's systems (Rajamoney 1993), invest significant effort to design experiments that provide the maximum amount of information. This is necessary because experiments can be costly and can have persistent effects on the environment. Debrief's chunking-based technique filters out irrelevant experiments automatically, without significant effort. Side events on the environment are not a concern during mental simulation.

Explaining Beliefs

Explaining beliefs, e.g., that ROE was achieved, involves many of the same analysis steps used for explaining decisions. Debrief starts by searching memory for the nearest preceding state in which the belief came to be held. It determines what operator was being applied during that state, and uses Establish-Applicability if necessary to make sure that the operator applies in the recalled state. If the belief had to be retracted in order to make Test-Operator-Applicability succeed, then the operator was responsible for asserting the belief. Such is the case for the belief that ROE is achieved, which is asserted by an operator named ROE-Achieved. Otherwise, Debrief would remove the belief and attempt mental simulation again; if the belief is asserted in the course of applying the operator, the operator is probably responsible for the belief.

Summary of the Effects of Learning

Learning via chunking takes place throughout the Debrief system. The following is a summary of the different types of chunks that are produced:

- Episodic memory recognition chunks: event + attribute value $\rightarrow$ recognition;
- State recall chunks: event $\rightarrow$ state;
- Mental simulation chunks: event + problem space + state $\rightarrow$ applicable or inapplicable + alternative operator;
- Applicability analysis chunks: event + problem space + state $\rightarrow$ significant state attributes; event + problem space + state $\rightarrow$ alternative operator + alternative state;
- Natural language generation chunks: case frame $\rightarrow$ list of words; content description $\rightarrow$ list of utterances;
- Presentation chunks: content description + user model $\rightarrow$ utterances + media control commands + user model updates.

The presentation mechanisms that yield the latter two types of chunks are described in (Johnson 1994). Altogether, these chunks enable Debrief to acquire significant facility in explaining problem solving behavior. These chunks result in speedups during the course of explaining a single mission. Future experiments will determine the transfer effects between missions.

Evaluation and Status

The implementation of Debrief comprises over 1700 productions; in a typical session these are augmented by between 500 and 1000 chunks. Debrief currently can describe and/or explain a total of 66 types of events in the tactical air domain. Its natural language generation component has a vocabulary of 259 words and phrases. Debrief can explain a range of one-on-one and one-on-two air-to-air engagements.

Formative evaluations of Debrief explanations have been performed with US Naval Reserve fighter pilots. These evaluations confirmed that explanations are extremely helpful for validating the agent's performance, and building confidence in it. They also underscored the importance of having the agent justify its beliefs—the evaluators frequently wanted to ask questions about assertions made by Debrief during the course of the explanation. This motivated the development of support for the Explain-Belief question type. There was immediate interest on the part of the subject matter experts in using Debrief to understand and validate the behavior of TacAir-Soar agents.

The weakest point of the current system is its natural language generation capability. However, this was found not to be a major concern for the evaluators. Their primary interest was in understanding the thinking processes of TacAir-Soar, and to the extent that Debrief made that reasoning apparent it was considered effective.

Conclusion

This paper has described a domain-independent technique for analyzing the reasoning processes of an intelligent agent in order to support explanation. This technique reduces the need for extensive knowledge acquisition and special architectures in support of explanation. Instead, the agent can construct explanations on its own. Learning plays a crucial role in this process. Next steps include extending the range to questions that can be answered, improving the natural language generation, and making greater use of multi-media presentations. There is interest in using the mental simulation framework described here to improve the agent's problem solving performance, by discovering alternative decision choices with improved outcomes.

Acknowledgements

The author wishes to thank Paul Rosenbloom, Milind Tambe, and Yolanda Gil for their helpful comments. Dr. Johnson was supported in part by the ARPA and the Naval Research Laboratory under contract number N00014-92-K-2015 (via a subcontract from the University of Michigan). Views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion or policy of the U.S. Government or any agency thereof.

References

Calder, R.; Smith, J.; Courtemanche, A.; Mar, J.; and Ceranowicz, A. 1993. ModSAF behavior simulation and control. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, 347–359. Orlando, FL: Institute for Simulation and Training, University of Central Florida.

Clancey, W. 1983a. The advantages of abstract control knowledge in expert system design. In *Proceedings of the National Conference on Artificial Intelligence*, 74–78.

Clancey, W. 1983b. The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence* 20(3):215–251.

Davis, R. 1976. *Applications of Meta-Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases.* Ph.D. Dissertation, Stanford University.

Doorenbos, R. 1993. Matching 100,000 learned rules. In *Proceedings of the National Conference on Artificial Intelligence*, 290–296. Menlo Park, CA: AAAI.

Gil, Y. 1993. Efficient domain-independent experimentation. Technical Report ISI/RR-93-337, USC / Information Sciences Institute. Appears in the Proceedings of the Tenth International Conference on Machine Learning.

Hill, R., and Johnson, W. 1994. Situated plan attribution for intelligent tutoring. In *Proceedings of the National Conference on Artificial Intelligence.*

Johnson, W. 1994. Agents that explain their own actions. In *Proc. of the Fourth Conference on Computer Generated Forces and Behavioral Representation.* Orlando, FL: Institute for Simulation and Training, University of Central Florida. World Wide Web access: http://www.isi.edu/soar/debriefable.html.

Jones, R.; Tambe, M.; Laird, J.; and Rosenbloom, P. 1993. Intelligent automated agents for flight training simulators. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, 33–42. Orlando, FL: Institute for Simulation and Training, University of Central Florida.

Jones, R. 1993. Using CGF for analysis and combat development. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, 209–219. Orlando, FL: Institute for Simulation and Training, University of Central Florida.

Neches, R.; Swartout, W.; and Moore, J. 1985. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering* SE-11(11):1337–1351.

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

Rajamoney, S. 1993. The design of discrimination experiments. *Machine Learning* 185–203.

Rosenbloom, P.; Laird, J.; and Newell, A. 1987. Knowledge level learning in Soar. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 618–623. Menlo Park, CA: American Association for Artificial Intelligence.

Scott, P., and Markovich, S. 1993. Experience selection and problem choice in an exploratory learning system. *Machine Learning* 49–68.

Swartout, W., and Moore, J. 1993. Explanation in second generation expert systems. In David, J.-M.; Krivine, J.-P.; and Simmons., R., eds., *Second Generation Expert Systems.* Springer-Verlag. 543–585.

Teach, R., and Shortliffe, E. 1984. An analysis of physicians' attitudes. In Buchanan, B., and Shortliffe, E., eds., *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley. 635–652.

Vera, A.; Lewis, R.; and Lerch, F. 1993. Situated decision-making and recognition-based learning: Applying symbolic theories to interactive tasks. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 84–95. Hillsdale, NJ: Lawrence Erlbaum Associates.

Wick, M., and Thompson, W. 1989. Reconstructive explanation: Explanation as complex problem solving. In *Proceedings of the Eleventh Intl. Joint Conf. on Artificial Intelligence*, 135–140. San Mateo, CA: Morgan Kaufmann.