Intelligent Agents for Aircraft Combat Simulation

Silvia Coradeschi, Lars Karlsson, Anders Törne
Department of Computer and Information Science
Linköping University
581 83 Linköping, Sweden
{silco larka andto}@ida.liu.se

1. Abstract

In this paper we give a first account of a project at Linköping University in collaboration with Saab Military Aircraft AB, Sweden. The aim of the project is to study the design and implementation of intelligent agents for the air combat domain, especially for beyond visual range combat. In particular users should be able to construct such agents without computer science expertise. Our main interest lies in the decision process of the agent and in how the behavior of the agent can be specified.

2. Introduction

Systems for simulation of beyond visual range combat are currently used for evaluating aircraft, missiles and tactics and also for training pilots and to supply enemy aircraft in battlefield simulations. These applications require that the behavior of the automated pilots is almost indistinguishable from human behavior in the specific air-combat domain. This is especially relevant in training applications where humans interact with automated pilots. It is, in fact, important for the realism of the simulation that humans interact with automated agents as they would interact with other humans.

Human pilots are in general guided in their actions by strategies and tactics, but are also able to react to unexpected situations and to adapt general strategies to specific cases. The challenge is then to develop the capability in an intelligent agent to react in a flexible way to uncertain and dynamic environments and still follow strategies and tactics as a human pilot would.

Our interest is focused on the decision process of the automated pilot. An approach to modeling the human decision processes is proposed in TAC BRAWLER, a simulation tool for providing a detailed representation of air-to-air combat, both within and beyond visual range (Decision-Science Application 1991). In TAC BRAWLER the decisions about what actions to perform are made by the automated pilots which evaluate, for each alternative action, the situations that will result if the alternative were to be executed.

The TacAir-Soar is also an interesting approach to air-combat simulation (Tambe et. al. 1995a). TacAir-Soar has been developed within Soar, a software architecture created as a basis for general intel-ligence. In this project several features that are required for an intelligent agent have been considered, for example coordinated behavior (Laird et. al. 1994) and enemy tracking (Tambe et. al. 1995b).

In both these systems air-combat experts can specify the behavior of the agents, but it is then encoded by system experts. In our approach the experts of the air-combat domain are intended to specify the behavior of each agent directly, without the aid of a system expert. This has the advantage of letting the experts give the directives that the agents should follow, test the behavior and change the directives in case the agents do not behave as expected. It also makes it possible to tailor the behavior for testing specific features of aircraft, missiles and tactics or to train pilots for a particular situation. On the other hand this puts special requirements on the user aspects of the system, both with respect to learning it and working with it. These aspects are considered in TACSI (TACtical SImulation) (Saab Military Aircraft 1995), a system developed by Saab Military Aircraft for autonomous simulation of many vs. many beyond visual range combat. This system is currently used at Saab for evaluating and developing their products. In parallel to the continued development of TACSI, Saab and Linköping University have undertaken a collaborative project with the aim of further investigating the design and implementation of automated agents for the air combat domain.

In our approach a scenario contains the specification of the agents present in the scenario, a state for each agent describing the information that is used for making decisions, and a decision-tree for each agent.

So far we have implemented a prototype of the decision-mechanism and we have interfaced it with a simplified simulator, where the technical characteristics of aircraft and missiles are described though not yet on a high level of accuracy. The next step will be to design a user-interface for specifying the decision-trees, interface the decision system with the simulator currently used at Saab Military Aircraft and test the system with respect to both the performance and the user issues.

In the next section we see how the requirement that the user defines the decision-tree has influenced our choices in building the system. We then examine specification of the agents, state, decision-tree and actions and finally we consider an example.

3. Requirement for user-defined decision-trees

The knowledge required by an agent to perform realistically in complex and uncertain situations is in general difficult to acquire and to code precisely by the persons who implement the agents. Furthermore the user can require different behaviors of the agents for different purposes, for example a pilot's trainer may want a very simple model of the behavior of the agent in order to train an inexperienced pilot. It then seems reasonable to give the user the possibility to determine the behavior of the agents and to adapt it to their particular context.

The challenge is to design a mechanism for specifying the behavior of the agent that is powerful enough for the user to specify the desired behavior within reasonable limits, but at the same time is easy to learn and to use.

3.1 Hierarchical structure and language of conditions

Specifying the decision mechanism in the form of rules is a natural way to represent behavior. However, the number of rules and conditions necessary for specifying complex behaviors can be very large. TacAir-Soar for example contains about 1700 rules. Also several conditions can be common to more than one rule. Therefore organizing the rules in a tree can contribute to a more structured and compact representation and this can facilitate the writing of the rules and the understanding of the resulting behavior. In our solution each node in the tree is associated with a condition that has to be satisfied in order to enter the node, and each "end node", or leaf, is in addition associated with an action. A branch of the tree represents a rule that has as conditions the conjunction of all the conditions in the arcs of the branch, and as action the action in the leaf. In this way the conditions are structured from more general to more specific and can be common to several branches yielding a more compact representation.

It is also important to allow the user to write complex conditions. For example a condition for performing the action of moving toward an enemy in order to intercept him can require that the enemy in question has been explicitly selected for interception and that the agent knows his position. A language has been defined for writing conditions that allow the use of *and*, *or*, *not, there-is, for-all* and functions.

3.2 Priorities and changes of priorities for actions

Several branches of the tree can be visited in parallel. If a leaf is reached, the corresponding action is a candidate to be performed. All the candidate actions form the candidate action set. From this set, the actions that are actually performed are selected on the basis of dynamically varying priorities. The user specifies a priority value for the action in each leaf and also specifies how this value can be changed during the evaluation of the branch conditions in which the action is a leaf. We have introduced the possibility of dynamic change of priorities in order to simplify the tree and to simulate the change of priority, in the pilot's mind, of the actions that he should perform depending on the situation.

The following example illustrates how dynamic change of priorities is used. In figure 1 a part of a decision-tree is presented. The whole tree is presented and explained in section 7.

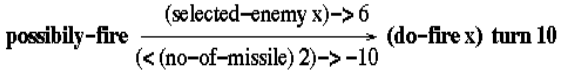

Figure 1: Example of dynamic change of priorities

The agent has the possibility to fire at an opposing agent x and should decide whether to fire or not. If x is the selected enemy, it should fire. If is not the selected enemy, it should fire only if more than 1 missile is left. In the example, if x is the selected enemy the priority of the sequential action **(do-fire x) turn** is increased by 6, and if the number of missiles is less than 2, the priority is decreased by

10. Given that the priority of the action is 10 and that the action is not performed if its priority is equal to or less than 0, the behavior is the desired one. In order to implement the same behavior without a dynamic change of priorities four action nodes would be necessary (figure 2).

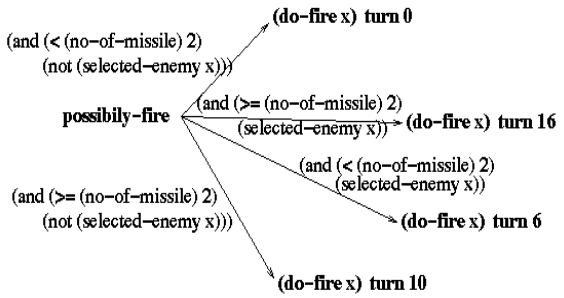

Figure 2: The same example as the previous figure without dynamic change of priorities

The actions at the leaves of the decision-tree can also be concurrent and sequential actions. Sequences of actions are useful if the user wants an action to be followed by other actions, for example firing and turning in the previous example. Concurrent actions are used when a number of actions should be started at the same time, for example turning and releasing a decoy in case a missile warning is received.

4. Specification of the agent

The specification of the agent contains the characteristics specific to each agent. These characteristics are specified by the user before starting the simulation and are maintained in the state of the agent. Examples of characteristics are:

- Physical characteristics, such as type of aircraft, number and type of missiles and amount of fuel.
- The role of the agent in the simulation, for example leader or wingman.
- A description of the mission that the agent should perform.
- General criteria the agent should follow in performing actions, for example commit criteria, i.e., criteria for deciding whether to intercept an enemy aircraft, and rules of engagement, i.e., directives about general conduct during the mission.

5. State

Each of the agents has a state where the information is maintained that is necessary for deciding what actions to perform. This information is of four types. First, the state contains the characteristics of the agent that the user has specified.

Secondly there is the information the agent receives from the simulator: position, velocity and direction of other aircraft visible on the radar and missile warnings. This is the same information that a pilot would receive from on-board sensors.

Thirdly, there is the information about the present status of the agent, the direction, velocity and position of the aircraft, the number of missiles left, friendly aircraft still present and the actions currently being performed, for example the agent is following the leader or moving toward a point.

Finally, one part of the state represents the "memory" of the agent. In the memory are recorded, among other things, important past events, for example at whom the agent has fired and how long before, and decisions previously taken, for example the decision to select an enemy as main target or the decision to intercept an enemy, and orders received via communication channels.

6. The decision-tree

The decision-tree consists of a hierarchy of decisions with one decision for each node. At every cycle in the simulation the decision-tree is visited and a list

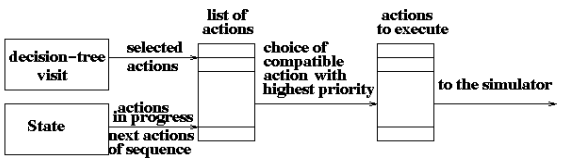

Figure 3: Phases of the decision mechanism

of possible actions is created (figure 3). The actions in the list are the actions that the agent will consider performing. To this list the actions still in progress are added. The list is sorted in order of priority, and the agent picks actions from the top and downwards, testing that each action is compatible with the higher prioritized actions already chosen.

6.1 Decision-tree structure

In a decision-tree, figure 4, a node is entered if the entry condition associated with it is satisfied. In the leaves there are actions with a basic priority value. Modifier conditions can also be associated with a node. Modifier conditions have the form (condition → number). If the condition is true, the number is added to the priority of the actions associated with the branches. If the priority of the action falls to 0 or below, the action is not performed.

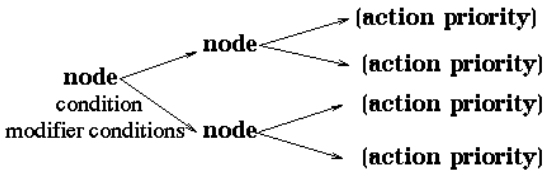

Figure 4: Decision-tree structure

6.2 Conditions

The conditions are defined as follows:

- **Primitive Conditions**

Atomic Conditions

Atomic conditions are true if the information is present in the state, for example the atomic condition *missile-warning* is true if in the state the information that the agent has received a missile warning is present. Atomic conditions can also have the form "(property agent)" and in this case a check is made in the state to establish whether the property is satisfied by the agent. For example *(selected-enemy x)* checks if x is the selected enemy of the agent. The variables can only refer to agents in the present implementation, but the system can be easily extended to allow other type of variables too.

Relational Conditions

Relational conditions consist of relational operators such as $<$, $>$, $\leq$, $\geq$, $\neq$, $=$ applied to numbers and numerical functions. An example of a relational condition is *(< (distance-to x) 100)* where *(distance-to x)* is a function that returns the distance of the agent to another agent x.

- **Composite Conditions**

Composite conditions are formed by applying the operators *and, or, not, for-all* and *there-is* to other composite or primitive conditions. If the condition associated with a node has the form "(for-all x condition)" the condition is tested for each agent x present in the simulation. The rest of the branch is visited once for each of the agents for which the condition is satisfied. All the actions selected are then taken into account as candidate actions to be performed. A "(there-is x condition)" finds the first agent that satisfies the condition and continues to evaluate the tree with x as this agent.

6.3 Actions

There are three kinds of actions: primitive actions, concurrent actions and sequential actions. Primitive actions are, for example, **(do-fire x)** i.e. fire a missile at the agent x. They can also be internal actions, i.e., actions that update the state of the agent. For example **(selection-enemy x)** records in the state the information that x is the current selected enemy of the agent. The actions sent to the simulator can be instantaneous, for example **(do-fire x),** or can have a duration such as **turn**. The agent keeps track of the actions that he is currently performing in the state and takes this information into account when deciding what actions to perform next.

Concurrent actions consist of a collection of actions that should be started at the same time. There is a concurrence of actions inherent in the system as actions in different branches can be selected and, if they are compatible, started at the same time. As for explicitly concurrent actions, however, we are sure that either all the actions are performed or none of them are. An example of a concurrent action of the latter kind is the sending of a command from the leader to the wingman to select a specific aircraft as primary target and the internal action of recording that the wingman now has a selected target.

Sequential actions are composed by actions that should be performed one after another. When a sequential action is started, the first action of the sequence is started and an automatic record is made in the state as to which are the following actions in the sequence that should be performed. When the first action of the sequence is completed, the second action of the sequence becomes a candidate action to be started and it competes for starting with the actions currently being performed and the other actions selected at the present cycle of the simulation. If the second action is started, the same thing then done with the third action and so on until the sequence is completed.

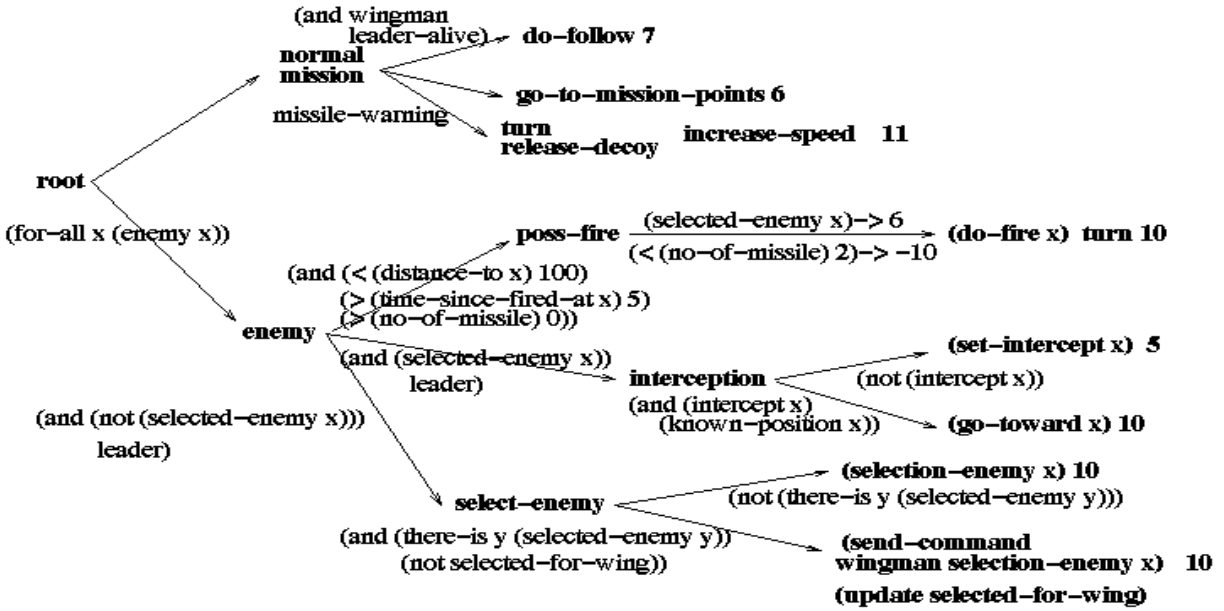

Figure 5: Example of decision-tree

If an action that is in turn is not started, the sequence is aborted. When a sequential action is started, the priority of the following actions in the sequence is increased with a value decided by the user depending on how important it is to complete the sequence once it has been started.

Finally actions can be a composition of both concurrent and sequential actions. For example, in the case of a missile warning the aircraft turns and at the same time releases a decoy and then it increases its speed.

6.4 Compatibility of actions

The test to check if two actions are compatible is made on the basis of the physical resources they use. Two primitive actions are compatible if they do not use the same resources. For example the action of firing is compatible with the action of turning left as they use different resources. A concurrent action is compatible with another action if all the actions that form the concurrent action are compatible with the other action. A sequential action is compatible with another action if all the actions that are present in the sequence are compatible with the other action. In this way we consider incompatible actions that could be performed at the same time. For example, the action of firing can be performed at the same time of the sequence of actions of turning and firing, but in order to build a compatibility criterion that would take these cases into consideration, it would be necessary to predefine the duration of the actions and this is not in general possible.

7. Example

In this section we present an example of a decision-tree (figure 5). The user will not need to write the decision-tree in this form; instead there will be a user-interface that will support the construction of the tree. The example is mostly constructed with the aim of showing the capability of the system and does not claim to be correct in terms of military tactics. Parts of this decision-tree have already been used in previous sections. Here we present a few brief comments:

- **(do-fire x) turn** is a sequential action. **turn** and **release-decoy** together constitute a concurrent action;
- if the condition *(selected-enemy x)* is true then *(selected-enemy x)* $\rightarrow$ *6* adds 6 to the priority of the action at the end of the branch;
- if an action has priority less then or equal to 0, it is not performed;
- *(> (time-since-fired-at x) 5)* means that the agent that is making the decisions has fired at the

aircraft x more than 5 time units before or has never fired at it;

- The condition *(< (distance-to x) 100)* means that the agent that is making the decisions is at a distance less than 100 spatial units from the agent x;

- The condition *selected-for-wing* is true if an enemy has already been selected for the wing-man;

- **(set-intercept x)** records the decision of intercepting x. *(intercept x)* is true if x is the enemy that the agent has previously decided to intercept;

- **(go-toward x)** makes the agent go toward the last recorded position of x;

- **(send-command** *agent command***)** means that the agent that receives the command updates the state. The value can then be tested in the context of the updated state and may influence the behavior of the agent. For example **(send-command wingman (selection-enemy x))** will make the wingman add to the state that the selected-enemy is x.

We have tested this decision-tree in a scenario with two opposing sections. Let us consider step by step some of the decisions that the leader of one of the sections makes. First, the action **go-to-mission-points** is performed and the agent starts moving toward the first of these points. When the first aircraft of the enemy section is detected, the agent selects this enemy as primary target, **(selection-enemy x),** and decides to intercept it, **(set-intercept x)**. In the next cycle the interception is started **(go-toward x)**. When the second enemy is detected a command is sent to the wingman to select this enemy as primary target and it is recorded that the wingman has now a selected enemy. When one of the enemies is close enough, a missile is fired. However in the meantime one of the enemies has also fired a missile and our agent receives a missile warning. So it turns and it releases a decoy. When the turning action is completed, the speed is increased. The scenario continues then to evolve with several interceptions, firings and avoiding of missiles until just aircraft of the same side are left.

8. Conclusions and future work

In this paper, we have presented preliminary results of a project on the design and implementation of intelligent agents for air combat simulation. In particular, we want to permit air combat experts to implement such agents without the aid of computer expertise. In order to achieve this, we are attempting to find a good balance between simplicity and expressiveness in the means given to the user to specify the behavior of the agent. We intend to continue the development of the system and to evaluate it both with respect to performance and user issues.

Our approach has the advantage of flexibility in responding to situations, as the priority of the actions is changed dynamically depending on the actual situation. At the same time describing complex situations is made easier by the fact that decisions are taken at several levels, from general decisions to specific ones, and also by the fact that the decisions do not need to be exclusive. In fact, several branchesof the decision-tree can be visited at the same time and several alternative actions can be taken into consideration. Dynamically changing priorities establish which actions are the ones that are performed. Decisions are reconsidered at each step of the simulation and this allows a quick reaction to changes in the situation. At the same time the agent can also follow strategies performing sequences of actions.

In this paper we have mainly considered the decision mechanism in itself. In the future we also intend to consider issues such as communication, coordination and tracking of enemies. Further it seems important to implement a more sophisticated handling of the interruption of sequential actions which, for example, would make it possible to continue an interrupted sequence of actions. It would also be of interest to consider a change in the priorities of the actions due to an assessment of the resulting situation after performing the action.

9. Acknowledgment

We would like to thank M. Tambe, G. Frisk, Johan Görsjö, Jenny Andersson, and Patrick Lambrix for their helpful comments. This work is partly supported by the Center for Industrial Information Technology, Linköping University (CENIIT), and the Swedish National Board for Industrial and Technical Development (NUTEK).

10. References

Decision-Science Application Inc. (1991) "The TAC BRAWLER air combat simulation management summary", *Technical Report 907,* DSA, Virginia, Usa.

Laird, J. E., Jones, R. M., and Nielsen, P. E. (1994) "Coordinated Behavior of Computer Generated Forces in TacAir-Soar", in *Proceedings of the Fourth Conference on Computer Forces and Behavioral Representation,* Orlando, FL.

Saab Military Aircraft (1995) "User Guide TACSI", *Technical Report TUCU-MI- 95:103,* Saab Military Aircraft, edition 3.1.

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

Tambe, M., and Rosenbloom, P. S. (1995b). "Resc: An Approach for Dynamic Real-time Agent Tracking", in *Proc. IJCAI'95.*

11. Authors' Biographies

Silvia Coradeschi is currently doing research at the Department of Computer and Information Science at Linköping University. Her interests are in the areas of artificial intelligence and computer generated forces.

Lars Karlsson is a graduate student at the Department of Computer and Information Science at Linköping University. His interests are in the area of artificial intelligence, in particular planning and autonomous agents.

Anders Törne is associate professor at Linköping University and director of the real-time system laboratory. His interests are in the area of specification, design, and verification of complex real-time systems.