A Reactive Planning in WavesWorld

Introduction

This appendix details one particular configuration of agents that embody a particular approach towards perception and action in WavesWorld. Portions of this chapter are based on Chapter 3 of my SMVS thesis (**Johnson91**), although it has been updated to reflect my current research results and implementation. The original planning algorithm was developed and described first by Maes' (**Maes89**) and this particular implementation and extensions (for parallel skill execution and distributed execution) was first described in (**Zeltzer91**) and in more detail in my SMVS thesis (**Johnson91**). The extensions with regard to sampling behaviors and perceptual sampling controls have not been discussed elsewhere.

The notion of using a network of inter-connected motor skills to control the behavior of a virtual actor was first described by Henry Jappinen in 1979 (**Jappinen79**). My advisor David Zeltzer discussed this notion in the context of computer animation in1983 (**Zeltzer83**). This was later independently elaborated and first implemented by Pattie Maes in1989 (**Maes89**) for physical robots. Her algorithm was used as the starting point for the work done in my SMVS thesis, which was reused and extended for this work. In addition to implementing her algorithm, I have extended the original in several ways, with an emphasis on the issues involved in a robust, parallel, distributed implementation.

This chapter begins with an algorithm for the problem of action selection for an autonomous agent as presented by Maes, and then goes into some detail about extensions which have been made during the course of implementing it. The mathematical model presented here differs slightly from Maes' original in that it corrects two errors I found while implementing it, as first noted in my SMVS thesis. Finally, a note on naming: where Maes speaks of **competence module**, I use the term **skill agent**. Maes' set of propositions **P** map to **sensor agents** in WavesWorld, and the members of both **G(t)** and **R(t)** are referred to as **goal agents** in WavesWorld.

Maes' Mathematical Model

This section presents a mathematical description of the algorithm so as to make reproduction of the results possible. Given:

- a set of competence modules **1**…**n**
- a set of propositions **P**
- a function **S(t)** returning the propositions that are observed to be true at time *t* (the state of the environment as perceived by the agent); *S* being implemented by an independent process (or the real world)
- a function **G(t)** returning the propositions that are a goal of the agent at time *t*; *G* being implemented by an independent process
- a function **R(t)** returning the propositions that are a goal of the agent that have already been achieved at time *t*; *R* being implemented by an independent process (e.g., some internal or external goal creator)
- a function **executable(i,t)**, which returns *1* if competence module *i* is executable at time *t* (i.e., if all of the preconditions of competence module *i* are members of *S(t)*), and *0* otherwise
- a function **M(j)**, which returns the set of modules that match proposition *j*, i.e., the modules *x* for which $j \in c_x$
- a function **A(j)**, which returns the set of modules that achieve proposition *j*, i.e., the modules *x* for which $j \in a_x$
- a function **U(j)**, which returns the set of modules that undo proposition *j*, i.e., the modules *x* for which $j \in d_x$
- π, the mean level of activation
- θ, the threshold of activation, where θ is lowered 10% every time no module was selected, and is reset to its initial value whenever a module becomes active
- ϕ, the amount of activation energy injected by the state per true proposition
- γ, the amount of activation energy injected by the goals per goal
- δ, the amount of activation energy taken away by the protected goals per protected goal

Given competence module $\mathbf{x = (c_x, a_x, d_x, \alpha_x)}$, the input of activation to module *x* from the state at time *t* is:

$$inputFromState(x, t) = \sum_j \phi \frac{1}{\#M(j)} \frac{1}{\#c_x}$$

where $j \in S(t) \cap c_x$ and where # stands for the cardinality of a set.

The input of activation to competence module *x* from the goals at time *t* is:

$$inputFromGoals(x, t) = \sum_j \gamma \frac{1}{\#A(j)} \frac{1}{\#a_x}$$

where $j \in G(t) \cap a_x$.

The removal of activation from competence module *x* by the goals that are protected

at time *t* is:

$$takenAwayByProtectedGoals(x,\ t) = \sum_{j} \delta \frac{1}{\#U(j)} \frac{1}{\#d_x}$$

where $j \in R(t) \cap d_x$.

The following equation specifies what a competence module $x = (c_x, a_x, d_x, \alpha_x)$, spreads backward to a competence module $y = (c_y, a_y, d_y, \alpha_y)$:

$$spreadsBW(x, y, t) = \begin{cases} \sum_j \alpha_x(t-1) \frac{1}{\#A(j)} \frac{1}{\#a_y} & \text{if executable}(x,t) = 0 \\ 0 & \text{if executable}(x,t) = 1 \end{cases}$$

where $j \notin S(t) \wedge j \in c_x \cap a_y$.

The following equation specifies what module x spreads forward to module y:

$$spreadsFW(x,\ y,\ t) = \begin{cases} \sum_j \alpha_x(t-1) \frac{\phi}{\gamma} \frac{1}{\#M(j)} \frac{1}{\#c_y} & \text{if executable}(x, t) = 1 \\ 0 & \text{if executable}(x, t) = 0 \end{cases}$$

where $j \notin S(t) \wedge j \in a_y \cap c_y$.

The following equation specifies what module x takes away from module y:

$$takesAway(x,\ y,\ t) = \begin{cases} 0 \text{ if } (\alpha_x(t-1) < \alpha_y(t-1)) \wedge (\exists i \in S(t) \cap c_y \cap d_x) \\ \min\left(\sum_j \alpha_x(t-1) \frac{\delta}{\gamma} \frac{1}{\#U(j)} \frac{1}{\#d_y}, \alpha_{y(t-1)}\right) \text{otherwise} \end{cases}$$

where $j \in c_x \cap d_y \cap S(t)$.

The activation level of a competence module y at time 0 is and for time $t > 0$, is defined as:

$$\alpha(y, t) = decay\left(\begin{array}{l} \alpha(y, t-1)(1 - active(y, t-1)) \\ + inputFromState(y, t) \\ + inputFromGoals(y, t) \\ - takenAwayByProtectedGoals(y, t) \\ + \sum_{x,z}\left(\begin{array}{l} spreadsBW(x, y, t) \\ + spreadsFW(x, y, t) \\ + takesAway(z, y, t) \end{array}\right) \end{array}\right)$$

where x ranges over the modules of the network, z ranges over the modules of the network minus the module y, $t > 0$, and the decay function is such that the global activation remains constant:

$$\sum_y \alpha_{y(t)} = n\pi$$

The competence module that becomes active at time t is module i such that:

$$\alpha(i, t) >= \theta \quad (1)$$

active(t, i) = 1 if $executable(i, t) = 1 \quad (2)$

$$\forall_j fulfilling(1) \Lambda (2): \alpha(i, t) >= \alpha(j, t) \quad (3)$$

active(t, i) = 0 otherwise

Maes' Algorithm: Pros and Cons

The Good News

Maes' algorithm is notable on several accounts. First of all, without reference to any ethological theories, she captured many of the important concepts described in the classical studies of animal behavior. Her view of activation and inhibition, especially as a continuously varying signal, are in step with both classical and current theories of animal behavior (**Sherrington06**, **McFarland75**). Secondly, the algorithm can lend itself to a very efficient implementation, and allows for a tight interaction loop between the agent and its environment, making it suitable for real robots and virtual ones that could be interacted with in real time.

Her enumeration of how and in what amount activation flows between modules is refreshingly precise:

> *"the internal spreading of activation should have the same semantics/effects as the input/output by the state and goals. The ratios of input from the state versus input from the goals versus output by the protected goals are the same as the ratios of input from predecessors versus input from successors versus output by modules with which a module conflicts. Intuitively, we want to view preconditions that are not yet true as subgoals, effects that are about to be true as predictions, and preconditions that are true as protected subgoals."* (**Maes89**)

This correspondence gives her theory an elegance which stands head and shoulders above the tradition of hacks, heuristics, and kludges that AI is littered with.

The Bad News

As with any new and developing theory, Maes' currently suffers from several drawbacks. I'll first list what I feel to be the problems with the algorithm as stated above, and then discuss each in turn.

- the lack of variables
- the fact that loops can occur in the action selection process
- the selection of the appropriate global parameters ($\theta, \phi, \gamma, \delta$) to achieve a specific task is an open question
- the contradiction that "no 'bureaucratic' competence modules are necessary (i.e., modules whose only competence is determining which other modules should be activated or inhibited) nor do we need global forms of control" (**Maes89**) vs. efficiently implementing it as such
- the lack of a method of parallel skill execution

Lack of Variables

Maes asserts that many of the advantages of her algorithm would disappear if variables were introduced. She uses indexical-functional aspects to sidestep this problem, an approach which I originally thought would be too limiting for anything more than toy networks built by hand. I felt that any implementor would soon tire of denoting every item of interest to a virtual actor in this way. Maes argues that the use of indexical-functional notation makes realistic assumptions about what a given autonomous agent can sense in its environment. This is perhaps true in the physical world of real robots, but in the virtual worlds I am concerned with, this is much less an issue.

My original solution was to posit a sort of generic competence module, which I called a template agent. These template agents would be members of the action selection network similar to competence modules, except they do not send or receive activation. When a fully specified proposition is entered in **G(t)**, relating to the template agent, it would instance itself with all of its slots filled in. For example, a generic competence module walk-to X might have on its add-list the proposition actor-at-X, where X was some location to be specified later. If the proposition actor-at-red-chair became a member of **G(t)**, this would cause the template agent walk-to X to instance itself as a competence module walk-to-red-chair with, among other things, the proposition actor-at-red-chair on its add-list. This instanced competence module would then participate in the flow of activation just like any other competence module. When the goal was satisfied, or when it was removed from **G(t)**, the competence module could be deleted, to be reinvoked by the template agent later if needed. If the number of modules in a given network was not an issue, any instanced modules could stay around even after the proposition which invoked them disappeared.

The template idea is a reasonable start, but it helps to think about this algorithm in the context of a larger system, where given a particular situation, a piece of software might analyze the scenario and generate a action selection network, with all the requisite receptors and agents from a set of high level, general template skill and sensor agents.

My current solution is to allow the sensor agents to communicate to skill agents by manipulating shared state in the virtual actor. For example, suppose you have a character

with a sensor agent **aCupIsNearby** and a skill agent **pickUpCup**. The skill agent keys off (among other things) that sensor agent. When the sensor agent computes itself, it has access to the virtual environment via its receptors. If it computes itself to be True, it has, *at that time*, access to information concerning the particular cup that it has decided "is nearby." It stores this information by sending a message to the virtual actor and then returns True. That message is stored in the character's shared state as short term memory. Some time later, after the appropriate activation has flowed around the action selection network, the skill agent **pickUpCup** is called. The first thing the skill agent does is retrieve the necessary information about the particular cup it is supposed to be picking up from shared state of the virtual actor. This information might be stored as some static piece of data regarding some aspect of the object (i.e., where it was when it was sensed by the sensor that put it there) or it might be a piece of active data, like a pointer or handle to the actual object in the environment. Either way, this allows agents to communicate via the shared state of the virtual actor, using it as a *blackboard* (**Hayes-Roth88**). Blumberg and Galyean (**Blumberg95**) uses a similar mechanism, which they call a Pronome, after (**Minsky87**).

Loops

The second major difficulty with the original algorithm is that loops can occur. From my perspective, this isn't necessarily a bad thing, since this sort of behavior is well documented in ethology, and could be used to model such behavior in a simulated animal. From the broader perspective of trying to formulate general theories of action selection, it remains a problem to be addressed. Maes suggests a second network, built using the same algorithm, but composed of modules whose corresponding competence lies in observing the behavior of a given network and manipulating certain global parameters ($\theta, \phi, \gamma, \delta$) to effect change in that network's behavior. This is an idea much in the spirit of Minsky's B-brains (**Minsky87**), in which he outlines the notion of a B-brain that watches an A-brain, that, although it doesn't understand the internal workings of the A-brain, can effect changes to the A-brain. Minsky points out that this can be carried on indefinitely, with the addition of a C-brain, a D-brain, etc. While this idea is interesting, it does seem to suffer from the phenomenon sometimes referred to as the homunculus problem, or the meta-meta problem. The basic idea is that any such system which has some sort of "watchdog system" constructed in the same fashion as itself, can be

logically extended through infinite recursion ad infinitum.

A more interesting solution, and one that has its basis in the ethological literature, is to introduce a notion of fatigue to competence module execution. By introducing some notion of either skill-specific or general fatigue levels, the repeated selection of a given competence module (or sequence of competence modules) can be inhibited.

How to Select $\theta, \phi, \gamma, \delta$

The selection of the global parameters of the action selection network is an open issue. To generate a given task achieving behavior, it is not clear how to select the appropriate parameters. From the perspective of a user wishing to direct the actions of a virtual actor, this is a grievous flaw which must be addressed. A similar solution to the one proposed for the loops problem could be used, namely using another network to select appropriate values. Unfortunately, this doesn't really address the problem of accomplishing a specific task. One idea is to use any of several learning methods to allow the network to decide for itself appropriate parameters. Learning by example could be used to excellent effect here.

Another interesting notion which is applicable is to allow the network to have some memory of past situations it has been in before. If we allow it to somehow recognize a given situation ("I'm going to Aunt Millie's. I've done this before. Let's see: I get up, close all the windows, lock all the doors, and walk down the street to her house?"), we could allow the network to bias its actions towards what worked in that previous situation. If we allowed additional links between competence modules called *follower* links, we could modulate the activation to be sent between modules which naturally follow each others' invocation in a given behavioral context. This idea has similarities to Minsky's K-lines (**Minsky86**) and Schank's scripts and plans (**Schank77**), but is more flexible because it isn't an exact recipe, it's just one more factor in the network's action selection process. This allows continuous control over how much credence the network gives the follower links, in keeping with the continuous quality of the algorithm.

Supposedly No Global Forms of Control

Maes considers her algorithm to describe a continuous system, both parallel and distributed, with no global forms of control. One of her stated goals in the development of this algorithm was to explore solutions to the problem of action selection in which:

> *"no 'bureaucratic' competence modules are necessary (i.e., modules whose only competence is determining which other modules should be activated or inhibited) not do we need global forms of control." (* **Maes89**)

Unfortunately, by the use of coefficients on the activation flow which require global knowledge (i.e., every term which involves the cardinality of any set not completely local to a competence module), there is no way her stated goal can be achieved. Secondly, it seems that any implementation of the algorithm has to impose some form of synchronization of the activation flow through the network. These two problem are inextricably linked, as I'll discuss below. Maes asserts that the algorithm is not as computationally complex as a traditional AI search, and that it does not suffer from combinatorial explosion (**Maes89**). She also asserts that the algorithm is robust and exhibits graceful degradation of performance when any of its components fail. Unfortunately, any implementation which attempts to implement the robustness implied in the mathematical model begins to exhibit complexity of at least $\mathbf{O(n^2)}$, since each module needs to send information to the process supplying the values for **M(j)**, **A(j)**, and **U(j)**. Also, information concerning the cardinality of $\mathbf{c_x}$, $\mathbf{a_x}$, $\mathbf{d_x}$, $\mathbf{c_y}$, $\mathbf{a_y}$, and $\mathbf{d_y}$ must also be available to calculate the activation flow. This implies either a global database/shared memory in which these values are stored, or direct communication among the competence modules and the processes managing **G(t)**, **S(t)**, and **R(t)**. Either method implies some method of synchronizing the reading and writing of data. Unfortunately, Maes asserts that the process of activation flow is continuous, which implies that asynchronous behavior of the component modules of the network is acceptable, which it clearly is not.

If we are to implement this algorithm in a distributed fashion, which is desirable to take advantage of the current availability of networked workstations, we need to choose between a shared database (containing data concerning the cardinality of **M(j)**, **A(j)**, **U(j)**, $\mathbf{c_x}$, $\mathbf{a_x}$, $\mathbf{d_x}$, $\mathbf{c_y}$, $\mathbf{a_y}$, and $\mathbf{d_y}$) and direct communication among competence modules. If we are to assume a direct communication model, a given module would need to maintain a communication link to each other module that held pertinent information to it (i.e., would be returned by any of the functions **M(j)**, **A(j)**, **U(j)** or would be involved in the calculation of the cardinality of $\mathbf{a_x}$, $\mathbf{d_x}$, $\mathbf{c_y}$, $\mathbf{a_y}$, and $\mathbf{d_y}$). Additionally, a module would need some way of being notified when a new module was added to the network, and have some way of establishing a communication link to that new module. In the limit, this implies that every

module would need to maintain **(n-1)** communication links, where the network was composed of **n** modules. Although necessary values to calculate the spreading of activation could be gotten and cached by each agent, to implement the robustness implied in the mathematical model, we need to recalculate *each* assertion for *each* proposition for *each* agent *every* time-step. This implies a communication bottleneck, and semi-formidable synchronization issues.

Alternatively, if it was implemented by a shared database or global memory, each agent would need only a single connection to the shared database. Some process external to the agents could manage the connection of new agents to the database and removal of agents which become disabled. This would allow the agents not to have to worry about the integrity of the other members of the network, and would reduce the complexity of the communication involved to **O(n)**. Given that an agent's accesses to the database are known (i.e., a given agent would need to access the database the same number of times as any other agent in the network), synchronization could be handled by a simple round-robin scheme, where each agent's request was handled in turn. When an agent wished to add itself to a given action selection network, it would need to register itself with the shared database by giving it information about itself (i.e., the contents of its condition-, add-, and delete-list). This would allow the database to answer questions from other agents about **M(j)**, **A(j)**, **U(j)** and the cardinality of $\mathbf{a_x}$, $\mathbf{d_x}$, $\mathbf{c_y}$, $\mathbf{a_y}$, and $\mathbf{d_y}$. Such a registry could also have a way of marking agents which didn't respond to its requests for updated information, and perhaps even have the ability to remove agents from the network which didn't respond or whose communication channels break down.

In either method, there needs to be some agreed upon method of synchronizing messages so that activation flow proceeds according to the algorithm, and that only one action is selected at a given time step. If we postulate some agency which dispatches which action is selected, we fly in the face of Maes' assertion of no global forms of control. Unfortunately, if we are to implement the algorithm in the distributed fashion described so far, I don't see any way around having such a task fragment dispatcher.

No Parallel Skill Execution

Another problem is the assumption built into the algorithm that no competence module takes very long to execute. This seems implicit in the fact that Maes does not seem to consider the lack of a method for having parallel executing modules as a

problem. For my purposes, this is a serious problem, since without such a capability, I could never have a virtual actor that could walk and chew gum at the same time. More specifically, a network containing a **walk** competence module and a **chewGum** module could never have both of them executing in parallel. Maes' view of the granularity of time in the system is very fine, while my view is that there should be some parameter which allows control from fine to coarse.

Summary of My Algorithm Extensions

Asynchronous Action Selection

A crucial difference between my algorithm and Maes' original is that (in mine) once a skill has been selected, an execute message is dispatched to it, and the action selection process continues actively trying to select the next action. Since the process actually executing the action is distinct from the process in which action selection is taking place, there is no need to enforce synchronicity at this point (i.e. we don't have to wait for that function call to return). At some point in the future, a finished message will probably be received by the registry/dispatcher, signifying that the skill agent has finished attempting to achieve the task.

Parallel Skill Execution

An executing skill agent will tie up some of the network's resources in the world, thereby inhibiting other similar skill agents from executing, while still allowing skill agents which don't need access to the same resources the ability to be selected. For example, if the walk-to-door skill is selected, it will engage the legs resources, and other skill agents (run-to-door, sit-down, etc.) will be inhibited from executing because one of their preconditions (legs-are-available) is no longer true. Other skill agents, such as close-window, or wave-good-bye, can still be selected, because the resources they require are available, and they are (possibly) receiving enough activation energy to be selected.

Adding Fatigue

At the beginning of each step of the action selection algorithm's loop, the registry/dispatcher checks for messages from any currently executing skill agents. When a given skill agent has finished executing (either believing it accomplished its task or giving up for some reason), it sends a message back to the registry/dispatcher. At this point, and not before, the skill agent's activation level is reset. This is analogous to the last section of

Maes' mathematical model in which the selected competence module's activation level is reset to zero. In our implementation, the skill agent's activation level is reset taking into account how successful it was in affecting the world in the way it predicted it would, combined with some hysteresis to avoid continually executing a skill agent which always fails to do what it promised.

If a skill agent has been called (i.e. sent an execute message) consecutively for its maximum number of times (this can be network dependent, skill agent dependent, time dependent, or some combination of all three), its activation level is reset to zero. If however, the skill agent has not been called consecutively more than its maximum, its new activation level is calculated thus:

$$\alpha_{current}\left(1-\left(\frac{\text{consecutiveCalls}}{\text{maximumCalls}}\right)\right)\left(1-\left(\frac{\text{correctPredictionsAboutWorldState}}{\text{totalPredictionsAboutWorldState}}\right)\right)$$

The intuitive idea here is that if a skill agent has seemingly done what it said it would, its goal has been achieved and its activation level should be reset to zero (just as in Maes' original algorithm). If however, the world state as measured differs significantly from what the skill agent predicted, the obvious course of action is to allow that skill agent a high probability of being selected again (by not reducing its activation level much). The likelihood of calling a given skill agent consecutively should decrease over time, thereby building some hysteresis into the action selection. For example, if you dial a phone number and get a wrong number, you usually attempt to dial the phone number again. If the second attempt fails, you start looking around for some other way to accomplish your goal-rechecking the number in your address book, calling directory assistance, etc.

Perceptual Sampling Controls

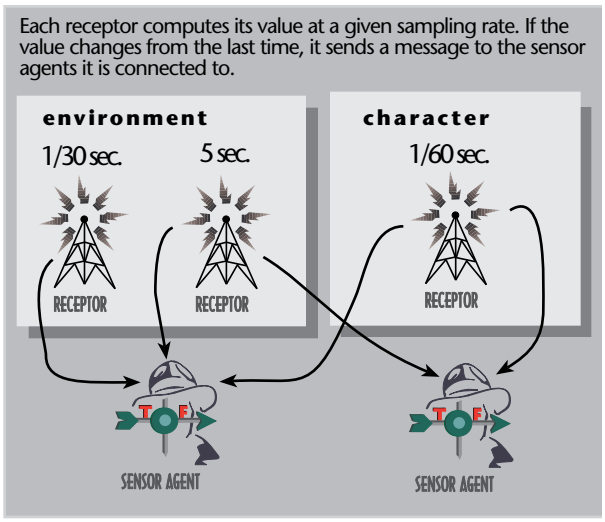
In Maes' original system, while provisions are made for the fact that a given skill may or may not be successful, there is no discussion of sensor fallibility. Two situations may lead to a sensor reporting incorrect information: incorrect assumptions by the sensor designer, or undersampling of the phenomena that the sensor is built to perceive. While there is little we can explicitly do about a badly designed sensor, we can discuss sampling.

The solution I developed was inspired by a notion from Rasmussen who talked about the signals, signs, and symbols to which a virtual actor attends:

> *"Signals represent sensor data — e.g., heat, pressure, light — that can be processed as continuous variables. Signs are facts and features of the environment or the organism."* (**Rasmussen83**)

I realized that sensor agents corresponded directly to **signs**, but I needed some sort of

representation for **signals**. In WavesWorld, these would be something that digitally sampled continuous signals in either the virtual actor or the virtual environment. I termed these perceptual samplers **receptors**. Receptors are code fragments that are "injected" by sensor agents into either the virtual actor or the virtual environment. Each receptor has a sampling frequency associated with it that can be modified by the sensor agent which injected it. These probes sample at some rate (say, every 1/30 of a second or every 1/2 hour) and if their value changes from one sample to the next they send a message to the sensor agent, which causes the sensor agent to recalculate itself.



Receptors are shared among sensor agents, and their information is also available to skill agents. Because their sampling frequency can be modulated, it is possible to trade off the cost of sampling the world at a high frequency vs. the possibility of aliased signals.