# Part VIII: Soar-EpMem – Episodic Memory

Soar-EpMem is a task-independent, architectural integration of an artificial episodic memory (EpMem) with Soar. The EpMem mechanism automatically records episodes as a Soar agent executes. These episodes can later be queried and retrieved in order to improve performance on future tasks.

In this tutorial, we will create an Eaters agent that will perform action modeling: it will use memories of past events to predict the consequences of future choices. Developing this agent will proceed in three steps. First, we will discuss the big picture goals of the agent. Second, we will develop and demonstrate the non-EpMem components of our agent. Finally, we will add rules to use Soar-EpMem.

While the final Eaters agent we will develop is stored in an *epmem* directory with all other Eater tutorial agents, the reader is encouraged to use his/her favorite Soar editor and develop the agent from scratch while reading this text.

## The Big Picture

In part 2 of the Soar tutorial, we developed an Eaters agent that moved around its world and ate food. We then learned that by adding rules to represent partial-order preferences over the types of food, the agent greedily eat more valuable food items when offered the choice.

Consider now an added challenge: we are to develop a similarly greedy Eaters agent without knowing a priori the points attributed to each type of food. Thus, instead of instilling hard-coded preferences, the agent must learn them as it performs actions in the world.

One approach to this problem would be to develop Soar-RL rules (see Soar Tutorial Part VII) to represent the value (points achieved) of eating a certain type of food. With the appropriate Soar-RL rules, reward rules, and parameter settings, the agent would learn very quickly an appropriate value function to achieve greedy behavior. While we will not cover the Soar-RL approach in this tutorial, the reader is encouraged to pursue this path as an added challenge.

In this tutorial, we will apply episodic learning to develop our agent. Abstractly, as the agent performs actions in the world, Soar-EpMem will automatically record “snapshots” of its Working Memory as episodes. When the agent is faced with a choice between multiple types of foods, we will instruct it to query past episodes for each of these foods. If the agent is able to “remember” a time in which it ate a food type of current interest, it will then roll forward time to see how the its score changed as a result of consumption. It will add this information to a knowledgebase (KB) of food information. It will then use information from its KB, with respect to its current movement choices, to move in the direction of highest expected value.

The reader should note that the procedure described above could be achieved without the use of episodic memory. Indeed, we could develop rules to deliberately store in Working Memory a score history relative to food type consumption. These rules, however, would be very task-specific and might be difficult for the agent to improve over time. In contrast, Soar-EpMem episode storage is general-purpose, and the agent can make use of Soar learning mechanisms to improve its queries and result responses.

## The Base KB Agent

This section describes developing an agent that makes use of a KB of food information. First we provide initialization/maintenance rules, then basic KB-based movement, and finally motivation for incorporating Soar-EpMem functionality.

### Agent Initialization & Maintenance

Our first set of rules follow roughly the Eaters agent template provided with tutorial code. These rules deal with state naming and top state aliasing:

sp {elaborate\*state\*name

(state <s> ^superstate.operator.name <name>)

-->

(<s> ^name <name>)

}

sp {elaborate\*state\*top-state

(state <s> ^superstate.top-state <ts>)

-->

(<s> ^top-state <ts>)

}

sp {elaborate\*top-state\*name

(state <s> ^superstate nil)

-->

(<s> ^top-state <s>)

}

Next, we will add code to maintain the output link by removing any completed actions:

sp {apply\*clean\*move

(state <s> ^io.output-link <ol>

^operator <op>)

(<ol> ^move <move>)

(<move> ^status complete)

-->

(<ol> ^move <move> -)

}

Finally, we will include an *init* operator to start agent execution and establish information about available directions for movement (and their relative opposites):

sp {eater\*propose\*init

(state <s> ^superstate nil

-^name)

-->

(<s> ^operator <op> +)

(<op> ^name init)

}

sp {apply\*init

(state <s> ^operator <op>)

(<op> ^name init)

-->

(<s> ^name eater

^direction <no> <so> <ea> <we>)

(<no> ^value north ^opposite south)

(<so> ^value south ^opposite north)

(<ea> ^value east ^opposite west)

(<we> ^value west ^opposite east)

}

### Agent Movement

Our agent’s only possible action will be to *move*. Its cognitive difficulty will be deciding the best direction in which to *go*. Thus, at each (x,y) location in the world, our agent will propose and choose movement as its course of action. Given this goal, it will then propose to go in all valid directions. Given our experience in past tutorials, we will define a valid direction as one that does not contain a wall or another eater, and is not opposite to the agent’s previous direction of movement:

sp {eater\*propose\*move

(state <s> ^name eater

^io.input-link.eater <eater>)

(<eater> ^x <x> ^y <y>)

-->

(<s> ^operator <op> + =)

(<op> ^name move)

}

sp {eater\*propose\*go

(state <s> ^name move

^top-state <ts>)

(<ts> ^direction <dir>

^io.input-link.my-location.<dir-name>.content <food-name>

{<> wall <> eater}

-^last-direction <opp-dir-name>)

(<dir> ^value <dir-name>

^opposite <opp-dir-name>)

-->

(<s> ^operator <op> +)

(<op> ^name go

^direction <dir-name>

^food <food-name>)

}

In order to help the agent decide the best direction of movement, it will rely upon a knowledgebase of food information to be maintained at the top state. Initially the KB will be empty. As the agent encounters food types it will store value information. If the value of an encountered food type is unknown, it will be indicated as such. We will discuss shortly the rules to populate this KB, but for now we will encode the rules to use it:

sp {eater\*augment\*go\*food

(state <s> ^name move

^top-state <ts>

^operator <op> +)

(<op> ^food <food-name>)

(<ts> ^food <food>)

(<food> ^name <food-name>

^value <food-value>)

-->

(<s> ^operator <op> = <food-value>)

}

sp {eater\*augment\*go\*unknown

(state <s> ^name move

^top-state <ts>

^operator <op> +)

(<op> ^food <food-name>)

(<ts> ^unknown <food-name>)

-->

(<s> ^operator <op> = 1000)

}

In the first rule, the agent represents the value of moving towards a known food type with a numeric indifferent preference valued respective to the food type in the KB.

The second rule provides a high-valued numeric indifferent preference for the direction of an unknown food type. If we assume that the constant of 1000 is much larger than any possible food type value, then our agent will be highly encouraged to learn about new foods.

At this point we will still assume that all food types are represented within our KB. Thus the preceding rules will provide enough information for Soar to make a directional decision. Once made, the agent will maintain information about the intended direction and food, as well as inform the Eaters environment of its choice on the output link:

sp {apply\*go

(state <s> ^operator <op>

^top-state <ts>)

(<op> ^name go

^direction <dir-name>

^food <food-name>)

(<ts> ^io.output-link <ol>

^operator <t-op>)

-->

(<ts> ^last-food <food-name>

^last-direction <dir-name>)

(<ol> ^move.direction <dir-name>)

(write | go | <dir-name> | towards | <food-name>)

}

sp {apply\*go\*remove\*dir

(state <s> ^operator <op>

^top-state <ts>)

(<op> ^name go

^direction <op-dir-name>)

(<ts> ^operator <t-op>

^last-direction <l-dir-name> {<> <op-dir-name>})

-->

(<ts> ^last-direction <l-dir-name> -)

}

sp {apply\*go\*remove\*food

(state <s> ^operator <op>

^top-state <ts>)

(<op> ^name go

^food <op-food-name>)

(<ts> ^operator <t-op>

^last-food <l-food-name> {<> <op-food-name>})

-->

(<ts> ^last-food <l-food-name> -)

}

The last step for our KB agent will be to configure Soar to use KB-generated numeric indifferent preferences in a very particular way. We know that food type values in Eaters do not change from moment-to-moment. Thus, it is safe to assume that our KB food type values, once known, are absolutely correct (one-step learning, as opposed to Soar-RL’s iterative process). Thus, the following commands will instruct Soar to always choose the highest valued numeric indifferent preference when making an operator decision based upon indifferent preferences:

indifferent-selection --epsilon-greedy

indifferent-selection --epsilon 0

These commands can be placed on any free line of your Soar agent file, though *\_firstload* is most logical when using *VisualSoar*.

### Agent Demonstration

In the next section we will use Soar-EpMem to populate the agent’s food information KB. Before we do so, it may be instructive to view the effects once episodic learning has taken place. Thus, we will temporarily modify the *init* application rule to begin with perfect information about the standard food types it will encounter:

sp {apply\*init

(state <s> ^operator <op>)

(<op> ^name init)

-->

(<s> ^name eater

^direction <no> <so> <ea> <we>

^food <empty> <normal> <bonus>)

(<no> ^value north ^opposite south)

(<so> ^value south ^opposite north)

(<ea> ^value east ^opposite west)

(<we> ^value west ^opposite east)

(<empty> ^name empty ^value 0)

(<normal> ^name normalfood ^value 5)

(<bonus> ^name bonusfood ^value 10)

}

Load your complete agent into the Eaters environment and run it. You should notice that the agent immediately acts greedily as in previous tutorial parts.

Now remove the KB entries in the above rule. We are ready to use Soar-EpMem to learn our KB.

## The Soar-EpMem Agent

This section first describes some basics of using Soar-EpMem and then provides the rules to complete our agent.

### Soar-EpMem Basics

As stated in the introduction to this part of the tutorial, Soar-EpMem automatically records episodes that agents can then query for later use. Agents interact with Soar-EpMem on an *epmem* link automatically created on each state.

Agents issue commands on a *command* sub-identifier of a state’s *epmem* link. A common command is a *query* identifier, whose contents include WMEs for which to look in previous episodes. Another useful command is to simply create a *next* identifier, which instructs Soar-EpMem to retrieve the episode temporally following the last successfully retrieved episode.

During the Output phase of each decision cycle, Soar-EpMem processes agent commands and places results on a *result* sub-identifier of the appropriate *epmem* link. The contents of this identifier include a *retrieved* identifier, which, if an episode is retrieved, is treated as the root of the state that was used to create the episodic memory. Agents can then access structures in this memory similarly to those in other parts of Working Memory. Query results also included a *status* WME to provide success/failure indicators. Also useful is a *memory-id* WME to indicate a unique id of the retrieved episode. Agents can use the *memory-id* in conjunction with a *prohibit* command during a query to prevent specific episodes from being retrieved.

### The Soar-EpMem Rules

Without properly populated food type KB entries, our agent will encounter a tie-impasse when it encounters at least two food types (including *empty*). At this point, Soar does not have enough information to make an operator selection. Our goal is to query Soar-EpMem’s store of episodes, such as to find past experiences that will provide more decision-relevant information to Soar.

During a tie-impasse, the agent will attempt to “remember” each of the food types contributing to the tie. These first three rules acknowledge the tie and establish the sub-goal to *remember*:

sp {elaborate\*go\*tie

(state <s> ^impasse tie

^choices multiple

^attribute operator

^superstate.name move)

-->

(<s> ^name go-tie)

}

sp {go-tie\*propose\*remember

(state <s> ^name go-tie

^top-state <ts>

^item <item>)

(<item> ^direction <dir-name>)

(<ts> ^io.input-link.my-location.<dir-name>.content <food-name>

-^food.name <food-name>

-^unknown <food-name>)

-->

(<s> ^operator <op> + =)

(<op> ^name remember

^food <food-name>)

}

sp {elaborate\*remember\*food

(state <s> ^name remember

^superstate.operator.food <food-name>)

-->

(<s> ^food <food-name>)

}

The process of remembering a food type requires a sequence of Soar-EpMem commands. First, the agent constructs a *query* command, desiring to find a past episode during which it had just eaten the food type in question:

sp {remember\*propose\*cue

(state <s> ^name remember

-^phase)

-->

(<s> ^operator <op> + =)

(<op> ^name cue)

}

sp {apply\*cue

(state <s> ^operator <op>

^epmem.command <cmd>

^food <food-name>)

(<op> ^name cue)

-->

(<cmd> ^query.last-food <food-name>)

(<s> ^phase response)

(write | trying to remember: | <food-name>)

}

During the Output phase following this rule application, Soar-EpMem will attempt to find a matching episode. If it cannot find one, the food type is unknown and this should be added to the KB:

sp {remember\*propose\*failure

(state <s> ^name remember

^phase response

^epmem.result.status failure)

-->

(<s> ^operator <op> + =)

(<op> ^name failure)

}

sp {apply\*failure

(state <s> ^operator <op>

^food <food-name>

^top-state <ts>)

(<op> ^name failure)

(<ts> ^operator <t-op>)

-->

(<ts> ^unknown <food-name>)

}

sp {apply\*go\*remove\*unknowns

(state <s> ^operator <op>

^top-state <ts>)

(<op> ^name go)

(<ts> ^operator <t-op>

^unknown <u>)

-->

(<ts> ^unknown <u> -)

}

With this KB entry addition, rules from the *go* sub-goal will fire to produce an exploratory numeric indifferent preference. Note that we also include a rule to remove unknown entries after every movement (such as to allow for new knowledge to be learned).

If Soar-EpMem successfully finds a matching episode, the agent will store away information about this episode and attempt retrieve the next episode:

sp {remember\*propose\*success

(state <s> ^name remember

^phase response

^epmem.result.status success)

-->

(<s> ^operator <op> + =)

(<op> ^name success)

}

sp {apply\*success

(state <s> ^operator <op>

^epmem <epmem>)

(<op> ^name success)

(<epmem> ^command <cmd>

^result <result>)

(<result> ^retrieved.io.input-link.eater.score <score>

^memory-id <mem-id>)

(<cmd> ^query <q>)

-->

(<s> ^old-score <score>

^old-id <mem-id>

^phase response -

^phase next)

(<cmd> ^query <q> -

^next <n>)

}

If Soar-EpMem is unable to retrieve the next episode (meaning it was the most recently recorded episode), the agent does not despair! Instead, it attempts to re-query Soar-EpMem, while prohibiting retrieval of the match it just received:

sp {remember\*propose\*next-none

(state <s> ^name remember

^phase next

^epmem.result.retrieved no-memory)

-->

(<s> ^operator <op> + =)

(<op> ^name next-none)

}

sp {apply\*next-none

(state <s> ^operator <op>

^old-id <mem-id>

^old-score <score>

^epmem.command <cmd>)

(<op> ^name next-none)

(<cmd> ^next <n>)

-->

(<s> ^phase next -

^prohibit <mem-id>

^old-id <mem-id> -

^old-score <score> -)

(<cmd> ^next <n> -)

}

sp {apply\*cue\*prohibit

(state <s> ^operator <op>

^epmem.command <cmd>

^prohibit <p>)

(<op> ^name cue)

-->

(<cmd> ^prohibit <p>)

(write | prohibit: | <p>)

}

sp {apply\*success\*prohibit

(state <s> ^operator <op>

^epmem.command <cmd>)

(<op> ^name success)

(<cmd> ^prohibit <p>)

-->

(<cmd> ^prohibit <p> -)

}

Note that we have added a rule to remove *prohibit* command(s) after *query* success, as issuing multiple types of commands to Soar-EpMem will result in an error.

If the next episode does exist, its score is compared to the previous score and the difference is stored in the food type KB:

sp {remember\*propose\*next

(state <s> ^name remember

^phase next

-^epmem.result.retrieved no-memory)

-->

(<s> ^operator <op> + =)

(<op> ^name next)

}

sp {apply\*next

(state <s> ^operator <op>

^old-score <old-score>

^epmem.result.retrieved.io.input-link.eater.score <new-score>

^top-state <ts>

^food <food-name>)

(<op> ^name next)

(<ts> ^operator <t-op>)

-->

(<ts> ^food <new-food>)

(<new-food> ^name <food-name>

^value (- <new-score> <old-score>))

}

Our Soar-EpMem agent is now complete. Load the agent into Eaters and watch it quickly develop greedy strategies seen previously in this tutorial. The reader is also encouraged to watch the Soar trace to verify creation of new KB entries after “remembering.”

## Additional Soar-EpMem Resources

The reader is encouraged to read the *Soar-EpMem Manual* for additional information on configuring Soar-EpMem. Additionally, the *Demos* directory contains a *kb* agent that acts both as a Soar-EpMem unit test system, as well as a demonstration of the full Soar-EpMem agent API.