1. **\_loadall\_01-falling.soar**
   1. **2-1-mario-fall-state.soar**
      1. That Soar files simply contain script commands such as “load” or “sp”
      2. That you load files into Soar using the “load file …” command
      3. That elaboration rules are the most basic kind of rule. (Other kinds of rules are proposal, preference, or apply rules.)
      4. Basic Soar rule structure:
         1. sp {…}
         2. rule name
         3. opening description/comment
         4. state link “(state <s> …)
         5. working memory elements
         6. variable notation
         7. condition and action side
         8. In-line comments with the # symbol
         9. || bars to denote strings (optional if the string has no spaces or special characters)
      5. That elaboration rules fire any time in the Soar decision cycle as soon as their conditions are satisfied, and their actions happen immediately in working memory
      6. That elaboration rule actions are undone as soon as the rule conditions are no longer satisfied
      7. That Soar provides an input-link and an output-link to let agents interact with outside environments
      8. That the environment is responsible for the contents of the input-link, and the agent is responsible for the contents of the output-link
      9. That the output-link contents are sent to the environment at the end of every decision cycle, and the input-link contents are refreshed by the environment at the start of every decision cycle
      10. That the Mario world has a particular schema for input-link and output-link commands, and this is provided in the agent\_io\_spec.txt file
      11. That elaboration rules can fire in response to the actions of other elaboration rules, and they will cascade off each other within a single decision cycle until “quiescence” is reached.
2. **\_loadall\_02-right-jump.soar**
   1. **2-2-mario-right-jump-operators.soar**
      1. That operators are the basic decision element of a decision cycle
      2. That two basic kinds of rules define operators: propose and apply rules
      3. That a propose rule functions like an elaboration rule, except that it creates an “operator” structure on the working memory state
      4. Rule syntax: dot notation to chain graph edge references
      5. Rule syntax: negation conditions that test for the absence of a structure
      6. That an operator structure needs at least a single “name” attribute
      7. That propose rules can define “preferences” for the operators they propose using preference symbols such as “=”
      8. That the “=” (“indifferent”) preference tells the Soar agent to choose among proposed indifferent operators uniformly and at random
      9. That Soar will select a single operator per decision cycle from among those proposed by propose rules, based on any preferences attached to those operators
      10. That an apply rule is any rule that tests for a selected operator in its conditions
      11. Than apply rules will only fire once an operator has been selected (their conditions that test for a selected operator will not be satisfied until then)
      12. That an apply rule functions differently from an elaboration rule in that it makes permanent changes to working memory that will persist even when the apply rule conditions are no longer satisfied.
   2. **2-3-mario-clean-output-basic.soar**
      1. That it is the agent’s responsibility to clean up commands that it puts on the output-link
      2. That the environment is able to put “^status complete” WMEs on output commands to tell the agent that it has read and understood the command
      3. Rule syntax: remove WMEs using the “-“ operation in rule actions
      4. That Soar provides a number of “RHS Functions” (Right-Hand Side Functions) that a rule can invoke among its actions to do special computation
      5. That the “(write …)” RHS Function prints a string to standard out
3. **\_loadall\_03-long-jump.soar**
   1. **2-4-mario-clean-output-long-jump.soar**
      1. That you can have multiple propose rules for a single operator
      2. Rule syntax: The “<>” operation tests “not equals”
      3. That you can use variables to reference attribute names as well as values
4. **\_loadall\_04-random-move.soar**
   1. **2-5-mario-left-operator.soar**
      1. (No new concepts, just adding another operator for Mario to employ)
   2. **2-6-mario-weighted-random-move.soar**
      1. That preference rules are another type of rule
      2. That preference rules add preferences to existing propose operators
      3. Rule syntax: You test for a proposed operator (not necessarily selected) using the “+” symbol after the operator test
      4. Numeric-indifferent preferences: You can assign utility weights to operators by adding a floating point value after the “=” preference symbol
5. **\_loadall\_05-timed-movement.soar**
   1. **2-7-mario-init.soar**
      1. That it is good practice to create a “^name” WME on your state in an “init” operator as the first thing your agent does
      2. That you can use the “init” operator to create any initial working memory structures you like
      3. That the “>” (“best”) preference makes an operator preferred over other operators that lack that preference
   2. **2-8-mario-cycle-timer.soar**
      1. That it is good practice to test for your state name in any non-apply rules you write, so that you know they will only fire in that specific state (as opposed to substates, which we’ll get to later, or in the case that your .soar file gets imported into another agent project)
      2. That you can have multiple apply rules for a single operator
      3. That multiple apply rules for a single operator can fire in parallel
      4. That Soar tracks the count of decision cycles
      5. That you can get the current decision cycle number using the RHS Function “(dc)”
      6. That you can use RHS Functions to do basic math in rule actions:
         1. “(- <minuend> <subtrahend>)”, “(+ <addend> <addend>)”, etc
      7. That you can have an apply rule that matches on any operator by not specifying a specific operator name
      8. That you can give an operator “<” (“worst”) preference so that it is only selected if no other non-worst operators are proposed
   3. **2-9-mario-cycled-move-operators.soar**
      1. (No new Soar concepts. Demonstrating the use of the timer for decision making.)