The basic idea is to make the pedagogical module into a Markov process. That is, it has no state of its own (hidden state; latent variables). Its response is determined solely by chance and the observable state. In this case, the observable state consists of

* a user interface event that Dragoon needs to react to,
* few state variables maintained by the editor, such as whether the student has made any errors while defining a node, and
* the student profile.

For now, the student profile only indicates the student mode (i.e., coached, immediate feedback, delayed feedback and power user) and some counters associated with the action SayInSequence” (explained below).

Author mode is so different from the student modes (e.g., authors type in descriptions whereas students select descriptions from a menu) that the author vs. student distinction should be hard-coded. For instance, the description widget should have two subclasses, one for author mode and one for the student modes. When the node editor is built, the right object is associated with the widget.

The initial implementation of the PM can be based on a large table. Whenever the user does a non-trivial user interface action, such as trying to close the node editor, the editor sends a message to the PM that has three components:

* The UI action done by the student, e.g., “description menu selection”
* The argument to that action, which describes it completely. For instance, if the action is a menu selection, then the argument is the menu item selected.
* The editor’s interpretation of the UI action e.g., correct, incorrect, etc.

The PM has one two dimensional table per UI action. It is indexed by action’s interpretation and by student mode. The cell in the table indicates one or more possible response that the tutor could make, where a response consists of one or more UI actions. Usually there will just be one response per cell. If there are multiple responses in the table cell, then the PM chooses one randomly. The response is then sent back to the editor, which executes the actions inside it. The argument of the UI action (2nd item in the list above) is often ignored, but could be used inside things that the tutor says to the student.

Each UI event message and the tutor’s response comprise a turn-pair (sometimes called an exchange), which is inserted into the activity log along with the times and the session id.

The main advantage of this tabular approach is that it simplifies the learning of improved policies for selecting reactions to events. That is, the four student modes are just policies about how to react to the events. Given enough data on student performance, we may be able to induce improved ones.

The rest of this memo illustrates the idea by walking through the construction of a rabbit growt & death model. The heading are for readers, and play no role in the PM design.

# Creating the initial node

When the rabbit problem opens, the student sees a blank canvas and clicks on the Create Node button on the menu bar. The editor generates this event:

* Action: click on Create Node button
* State: No nodes created yet

For all student modes, there is just one response:

* Open the node editor with the Description menu active and the Close button (a red X in Windows) active; everything else inactive.

The node editor enacts this response, and now the student is looking at the node editor (This and the other node editor images needs to have its input buttons revised; Megha is working on that).

# The first description menu selection event



Suppose the user selects something from the Description menu. The editor does all the interpretation of that selection with respect to the target model and the TNS. It sends the PM a message that has “Description menu selection” as the action, the menu item selected as the argument of the action, and one of the following as the action’s interpretation (this list includes some states that cannot occur when the canvas is blank; they are included here for completeness):

* *Optimal*: The student picked a quantity that the TNS would pick. In the case of the rabbit problem, the student would pick “the number of rabbits in the population.”
* *Not top level*: The model is empty, so the TNS wants the student pick a top level quantity, but the student has instead picked a different quantity that is in the target model.
* *Premature*: The model is not empty, and the student picked a quantity that is in the target model, but it is not one that the TNS would pick. In the case of the rabbit problem, one premature quantity is “the number of rabbits born per year.”
* *Initial value*: The student picked a quantity corresponding to the initial value of some accumulator in the target model. In the rabbit problem, this would be “100, the number of rabbits in the population initially.”
* *Extra value*: The student picked a quantity corresponding to value stated in the problem that is irrelevant to the target model. In the rabbit problem, this would be “130, the number of rabbits at the beginning of year 2.”
* *Irrelevant*: The student picked a description that is irrelevant to the target model and not mentioned in the problem statement.
* *Redundant*: The student picked a description for a node that already exists.
* *Last failure*: The student this is the third failed selection attempt for this menu, where “failure” counts selections classified as Initial value, Extra value, Irrelevant or Redundant but does not count selections classified as Not top level, Premature orOptimal. The counter belongs to the widget, so different widgets have different counters. All counters start at zero when the node editor opens.

Below are all the actions that Dragoon could feasibly make in reaction to an action of “description menu selection”. The action SayInSequence means that the first time this function is called with this student, the first string is said to the student; on the second time, the second string is said. This continues until all the strings have been said. Thereafter the function does nothing when called. Each call to SayInSequence requires its own counter to be stored in the student profile. That is, the SayInSequence in B below has a different counter than the SayInSequence in C.

1. Color the student’s selection red.
2. SayInSequence:
   1. “The quantity is irrelevant to this problem. Choose a different one.”
   2. “This quantity is irrelevant for modeling the system. Try again”
   3. “Irrelevant. Try again.”
3. SayInSequence:
   1. “You tried to define a parameter for the initial value of an accumulator. This is unnecessary, because you can put the initial value for the accumulator right into the definition of the accumulator itself.”
   2. “That should be the initial value of an accumulator, not a parameter node.”
   3. “That should be the initial value of an accumulator.”
4. SayInSequence:
   1. “You tried to define a parameter for a number you read in the problem. Not all numbers in the problem statement are necessary for the model. You will save effort if you follow the Target Node Strategy, which says you should start by defining a node for a quantity that the problem asks you to graph, then define nodes for its inputs, and then define nodes for their inputs, etc. That way, every node you create is an input to some node.”
   2. “Not every number in the problem statement is necessary for the model. You should define a node for a quantity only when either (1) it is required as input to a previously defined node, or (2) the problem statement asks you to graph it.”
   3. “Please be sure you need a node before defining it. Even if a number appears in the problem statement, it may not be needed in the model.”
5. Say “A node already exists for that quantity. If you want to edit it, click on it.” Notice that “say” is not the same as SayInSequence. It means that the phrase is always said to the student.
6. Color the student’s selection blue.
7. SayInSequence:
   1. “Blue means that the quantity is relevant for modeling the system, but it is not yet time to define it. You should follow the Target Node Strategy, which says you should edit an existing node that is not yet defined. Such nodes have dotted outlines. Click on one to edit it. ”
   2. “Blue means that according to the Target Node Strategy, it is too early to define a node for this quantity. Edit a node that has a dotted outline.”
   3. “Blue means premature. Edit a node with a dotted outline instead.”
8. SayInSequence:
   1. “Blue means that quantity isn’t one that the problem statement asks you to graph. Although this quantity will eventually be in your model, you should follow the Target Node Strategy, which says you should first define a node for a top level goal quantity.”
   2. “Please start with a quantity mentioned in the problem statement as one that needs to be graphed.”
9. Color the student’s selection green.
10. SayInSequence:
    1. “Green means correct. Good job!”
    2. “Green means correct.”
11. Replace the student’s selection with an optimal selection, and color it yellow.
12. SayInSequence:
    1. “Sorry, but that quantity isn’t relevant to the model. Moreover, this is the third failure, so a correct selection is being done for you. Please study it and figure out why it is correct. Your goal should be to make a correct selection on the first attempt.”
    2. “Here’s a correct solution. Please figure out why it is correct so that next time, your first selection will be correct.”
    3. “Please study this correct selection.”
13. Leave the description menu active.
14. Freeze the description menu so that the selection becomes permanent.
15. Activate the Type widget.

The policy table for the “Description menu action” maps states to tutor actions. There is one column per student mode (= policy, in Markov language). We may define other policies later. Each table cell contains a set (usually a singleton set) of messages, where each message consists of one or more tutor actions from the list above.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Interpretation** | **Coached** | **Feedback** | **Test** | **Power user** |
| Optimal | {IJNO} | {IJNO} | {INO} | {INO} |
| Not top level | {FHM} | {IJNO} | {INO} | {INO} |
| Premature | {FGM} | {IJNO} | {INO} | {INO} |
| Initial value | {ACM} | {ACM} | {AM} | {AM} |
| Extra value | {ADM} | {ABM} note: B, not D | {AM} | {AM} |
| Irrelevant | {ABM} | {ABM} | {AM} | {AM} |
| Redundant | {AEM} | {AEM} | {AEM} | {AEM} |
| Last failure | {KLNO} | {KLNO} | {KNO} | {KNO, AM} |

## Entering the node’s type

Suppose that the student clicks on an optimal menu item. Dragoon colors it green and pops up a modal dialog box that says “Green mean correct. Good job!”. The student closes the dialog box, and now the node editor looks like this, with the Type menu and the Close button being the only active widgets.



When the user clicks on the Type menu widget, the following menu appears: Parameter, Accumulator, Sum, and Product. If the UI were in Power-user mode, then the menu would include Hidden Function. When the user selects a menu item, the editor sends an UI event message whose action is “Type menu selection,” whose argument is the menu item selected, and whose interpretation is one of the following:

* *Correct*.
* *First failure*: The selection is incorrect, and this is the first attempt.
* *Second failure*: The section is incorrect, and this is the second attempt. The counter for attempts is zero when the node editor opens and never gets zeroed after that. Thus, if the student is in test mode, selects an incorrect type, fills out the form, then changes the type making it incorrect again, then the interpretation is “Second failure” and not “First failure.”
* *Yet another failure*: The selection is incorrect, and there have been three or more attempts.

The legal actions for the “type menu selection” action are:

1. Color the selection red.
2. Color the selection green.
3. Leave the selection white
4. Post the correct selection in yellow
5. Leave the widget open.
6. Freeze the widget showing the current menu selection.
7. Enable the next widget, where “next” depends on the type selected by the user. For instance, if the type is Accumulator node, the next widget is the type-in box labeled “Initial value.” If the type is Parameter, the next widget is the type-in box labelled “Value.”
8. Enable all the widgets appropriate for the selected type. If some widgets are already open (because the user has changed their mind about what type this node should be), then their values are lost.
9. SayInSequence:
   1. “Yellow means that you made an incorrect choice too many times, so you are being shown the correct choice. You should figure out why it is correct so that next time your first choice will be correct.”
   2. “Can you figure out why this is the right type for the node?”
10. Hint.
11. SayInSequence:
    1. Green means correct. Well done!

Eventually, we need to provide a way for authors to provide hints. A hint is just a rich-text-format string that is indexed by (1) the target model (2) the node in the target model (3) a widget e.g., type menu. The PM doesn’t need to know what the hint strings are, so the hint function can be implemented by the editor instead of the PM. Some samples of hints for the type menu are:

The PM may also have the controller popup a modal window with a text that was entered by the author and explains the node’s type. Some examples are:

* The problem only tells you that Jared’s weight is decreasing by 2.5 pounds a week. That is, it doesn’t tell you what his weight is each week, it only tells you about the *change* in his weight. Whenever you only know about the *change* in a quantity’s value, you need to use an accumulator node.
* This problem tells you that Jared’s weight is the sum of Oprah’s weight and a constant (50 pounds). Thus, you need to represent Jared’s weight with a Sum node.
* Whenever a problem says that a quantity is increasing or decreasing by a certain percentage per unit of time, then it is telling how much the quantity *changes* per unit of time. In this case, the problem says the rabbit population is growing by 30% each month, so it is telling how to calculate the change in the population per month. As you know, being told about the changes in a quantity’s value means that you need to represent it with an accumulator node.

If there is no hint for a particular index, then Hint calls SayInSequence with “Red means incorrect.” That is, if this particular combination of problem, node and UI action has no hint, and this is the first time that the user has gotten this UI action incorrect, the tutor says, “Red means incorrect.” On the other hand, if there is a hint for this combination of problem, node and UI action, then Hint displays it every time it is called.

The policy table for the Type menu selection action is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Interpretation** | **Coached** | **Feedback** | **Test** | **Power** |
| Correct | {BFGK} | {BEHK} | {CEH} | {CEH} |
| First failure | {AJE} | {AJE} | {CEH} | {CEH} |
| Second failure | {DIFG} | {DIFH} | {CEH} | {CEH} |
| Yet another failure | error: this case shouldn’t occur | error: this case shouldn’t occur | {CEH} | {CEH} |

## Entering the initial value of an accumulator node

Let’s suppose the student makes two mistakes on the node type, so after closing the popup modal window that explained what yellow means, the node editor looks like this:



There are two cases for the Initial value widget depending on whether the target model has a blank for the initial value of an accumulator or a number.

If the target model has a blank value, then the widget initially displays gray text saying “Enter a number.” When the student clicks in the box, the gray text disappears and a type-in cursor appears. When the student clicks outside the widget or presses the Enter or Tab keys, then the contents is validated inside the editor – students must enter a well-formed positive number. When they have finally done so, the number is interpreted and a message is sent to the PM. The action is “Typed initial value”, the argument is the value, and interpretation is always *Correct*. We may later want to have authors specify the maximum and minimum values for such values. When this feature is added, there will be other interpretations besides *Correct*. The tutor’s actions are:

1. Color the widget green.
2. Leave the widget enabled.
3. Open the next widget. If the target model has units for this value, then the next widget is the Units widget. If the target model does not have units for this value, then the next widget is the Input widget. If the next widget is already active, then this action does nothing.

The policy table for “Typed initial value” is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Interpretation** | **Coached** | **Feedback** | **Test** | **Power** |
| Correct | {ABC} | {ABC} | {ABC} | {ABC} |

Now for the second case: the target model does have a numeric initial value. Clicking in the widget displays a menu composed of all possible values. These are gleaned from all the parameter and accumulator nodes in the model. The author may also have entered some extra values, e.g., ones that appear in the problem statement. The menu presents the values in numerical order. When the student selects one, the editor sends the PM a message with the action “Selected initial value”, the menu item and one of the following interpretations:

* *Correct.*
* *First failure*
* *Second failure*
* *Yet another failure*.

The legal actions for this UI action are:

1. Color the widget red.
2. Color the widget green.
3. Leave the widget white.
4. Fill in the correct value and color the widget yellow.
5. Freeze the widget on its current value
6. Leave the widget active.
7. Enable the next widget (see above “C” above).
8. Enable all the widgets appropriate for this node’s type. With the current 4 policies, this is redundant. It only gets called with the widgets are already open. However, the redundancy doesn’t hurt and might make other policies easier to implement later.
9. Hint
10. SayInSequence:
    1. “Yellow means that you made an incorrect choice too many times, so you are being shown the correct choice. You should figure out why it is correct so that next time your first choice will be correct.”
    2. “Can you figure out why this is the right initial value for the quantity?”

The policy table for this UI action is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Interpretation** | **Coached** | **Feedback** | **Test** | **Power** |
| Correct | {BEG} | {BFH} | {BFH} | {BFH} |
| First failure | {AIF} | {AIF} | {C} | {C} |
| Second failure | {DEGJ} | {DJEH} | {C} | {C} |
| Yet another failure | error | error | {C} | {C} |

## Entering inputs

Let’s suppose that the author has left the units field blank on the target model for Rabbit population. Thus, the next widget, as determined by the controller, is the inputs field. The Units field remains disabled. If it were enabled, it would have nearly the same behavior as the Initial value field. Anyway, the node editor now looks like this.



Clicking in the white box automatically displays the same menu of possible node descriptions that is used by the Description widget. Because an accumulator must have at least one input, the “No more inputs” button is disabled.

The interpretations for the Input menu selection are:

* *Optimal*: The student picked a quantity that the TNS would pick. In the case of the rabbit problem, the student would pick “the number of rabbits born each year” or “the number of rabbit deaths each year.”
* *Premature*: The student picked a quantity that is in the target model, but it is not one that the TNS would pick. In the case of the rabbit problem, one premature quantity is “the rabbit birth rate.”
* *Initial value*: The student picked a quantity corresponding to the initial value of some accumulator in the target model. In the rabbit problem, this would be “100, the number of rabbits in the population initially.”
* *Extra value*: The student picked a quantity corresponding to value stated in the problem that is irrelevant to the target model. In the rabbit problem, this would be “130, the number of rabbits at the beginning of year 2.”
* *Irrelevant*: The student picked a description that is irrelevant to the target model and not mentioned in the problem statement.
* *Redundant*: The student picked a description for a node that already exists.
* *Last failure*: The student this is the third failed selection attempt for this menu, where “failure” counts selections classified as Initial value, Extra value, Irrelevant or Redundant but does not count selections classified as Premature or Optimal. The counter belongs to the widget, so different widgets have different counters. All counters start at zero when the node editor opens.

The possible tutor actions in response to this student UI action are:

1. Color the student’s selection red.
2. SayInSequence:
   1. “The quantity is irrelevant to this problem. Choose a different one.”
   2. “This quantity is irrelevant for modeling the system. Try again”
   3. “Irrelevant. Try again.”
3. SayInSequence:
   1. “You tried to define a node for the initial value of an accumulator. This is unnecessary, because you can put the initial value for the accumulator right into the definition of the accumulator itself.”
   2. “That should be the initial value of an accumulator, not a parameter node.”
   3. “That should be the initial value of an accumulator.”
4. SayInSequence:
   1. “You tried to define a node for a number you read in the problem. Not all numbers in the problem statement are necessary for the model. You will save effort if you follow the Target Node Strategy, which says you should start by defining a node for a quantity that the problem asks you to graph, then define nodes for its inputs, and then define nodes for their inputs, etc. That way, every node you create is an input to some node.”
   2. “Not every number in the problem statement is necessary for the model. You should define a node for a quantity only when either (1) it is required as input to a previously defined node, or (2) the problem statement asks you to graph it.”
   3. “Please be sure you need a node before defining it. Even if a number appears in the problem statement, it may not be needed in the model.”
5. Say “A node already exists for that quantity. If you want to edit it, click on it.”
6. Color the student’s selection blue.
7. SayInSequence:
   1. “Blue means that the quantity is relevant for modeling the system, but it is not yet time to define it. You should follow the Target Node Strategy, which says you should edit an existing node that is not yet defined. Such nodes have dotted outlines. Click on one to edit it. ”
   2. “Blue means that according to the Target Node Strategy, it is too early to define a node for this quantity. Edit a node that has a dotted outline.”
   3. “Blue means premature. Edit a node with a dotted outline instead.”
8. SayInSequence:
   1. “Blue means that quantity isn’t one that the problem statement asks you to graph. Although this quantity will eventually be in your model, you should follow the Target Node Strategy, which says you should first define a node for a top level goal quantity.”
   2. “Please start with a quantity mentioned in the problem statement as one that needs to be graphed.”
9. Color the student’s selection green.
10. SayInSequence:
    1. “Green means correct. Good job!”
    2. “Green means correct.”
11. Replace the student’s selection with an optimal selection, and color it yellow.
12. SayInSequence:
    1. “Sorry, but that quantity isn’t relevant to the model. Moreover, this is the third failure, so a correct selection is being done for you. Please study it and figure out why it is correct. Your goal should be to make a correct selection on the first attempt.”
    2. “Here’s a correct solution. Please figure out why it is correct so that next time, your first selection will be correct.”
    3. “Please study this correct selection.”
13. Leave the input menu active.
14. Freeze the input menu so that the selection becomes permanent.
15. Activate the “Add as + input” and “Add as – input” buttons.
16. Hint
17. Leave the input menu white.

Hints address all the inputs at once. Some sample hints are:

* What is the minimal set of quantities that you need to know in order to calculate the next value for Rabbit Population? You need to know the change in rabbits, right? In general, populations change by births, deaths, emigration or immigration. Rabbits in this population neither arrive (immigration) nor leave (emigration), so that leaves just births and deaths as sources of population change. Thus, the inputs required for calculating the change in Rabbit population each month are just the rabbit births that month and the rabbit deaths that month.
* To calculate the number of moose deaths, you will need to know the current size of the moose population—the larger the moose population, the more moose die. Now because the wolves eat moose, you’ll need to know the size of the wolf population too—if there are many wolves, then more moose will be eaten. Lastly, you’ll need a parameter.

The policy table for the Inputs menu selection action is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Interpretation** | **Coached** | **Feedback** | **Test** | **Power user** |
| Optimal | {IJNO} | {IJNO} | {QO} | {QO} |
| Premature | {FGM} | {IO} | {QO} | {QO} |
| Initial value | {ACM} | {ACM} | {QO} | {QO} |
| Extra value | {ADM} | {ABM} note: B, not D | { QO } | { QO } |
| Irrelevant | {AHM} | {AHM} | { QO } | { QO } |
| Redundant | {AEM} | {AEM} | {AEM} | {AEM} |
| Last failure | {KLNO} | {KLNO} | { QO } | { QO } |