

Unit 3: Attention

This unit is concerned with developing a better understanding of how perceptual attention works in ACT-R, particularly as it is concerned with visual attention.

3.1 Visual Attention

When a visual display such as

```

  N   Z   M   S
  F   G   J   X
  W   Y   D   R

```

is presented to ACT-R. A representation of all the visual information is immediately accessible in a visual icon. One can view the contents of this icon using the **Visicon** button in the environment or with the command `pm-print-icon`:

? (pm-print-icon)

Loc	Att	Kind	Value	Color	ID
(180 211)	NEW	TEXT	"d"	BLACK	TEXT44
(180 161)	NEW	TEXT	"j"	BLACK	TEXT45
(80 211)	NEW	TEXT	"w"	BLACK	TEXT46
(180 111)	NEW	TEXT	"m"	BLACK	TEXT47
(80 161)	NEW	TEXT	"f"	BLACK	TEXT48
(230 211)	NEW	TEXT	"r"	BLACK	TEXT49
(80 111)	NEW	TEXT	"n"	BLACK	TEXT50
(230 161)	NEW	TEXT	"x"	BLACK	TEXT51
(130 211)	NEW	TEXT	"y"	BLACK	TEXT52
(230 111)	NEW	TEXT	"s"	BLACK	TEXT53
(130 161)	NEW	TEXT	"g"	BLACK	TEXT54
(130 111)	NEW	TEXT	"z"	BLACK	TEXT55

This prints the information of all the features that are available for the model to see. It includes their screen locations, attentional status, visual object type (kind), current values, colors, and names.

3.1.1 Visual Location Requests

When requesting the visual location of an object there are many slots that can be tested: kind, attended, value, color, size, screen-x, screen-y, distance and nearest. We will describe the use of a few of those here, attended, screen-x, screen-y, and nearest.

For the x and y coordinates, screen-x and screen-y, there are several options. The first of which is to specify exact pixel coordinates:

```
+visual-location>
  isa      visual-location
  screen-x 50
  screen-y 124
```

That is useful if you know in advance exactly where something is on the screen. However, that is often not the case. If you know approximately where on the screen it is you can specify a range of acceptable values using the within specification:

```
+visual-location>
  isa      visual-location
  screen-x (within 100 150)
  screen-y (within 0 10)
```

That will request the location of an object whose x coordinate is between 100 and 150 (inclusive) and y coordinate is between 0 and 10 (inclusive).

You can also test for greater or less than a specific coordinate using the greater-than and less-than tests:

```
+visual-location>
  isa      visual-location
  screen-x (greater-than 75)
  screen-y (less-than 200)
```

If you are not concerned with the specific coordinates, but care more about relative positions then there are a couple of ways to specify that also.

You can use lowest and highest to request the location based on positions relative to all the items on the screen. Remember that x coordinates increase from left to right, so lowest corresponds to leftmost and highest rightmost, while y coordinates increase from top to bottom, so lowest means topmost and highest means bottommost. If this is used in combination with attended it can allow the model to find things on the screen in an ordered manner. For instance, to read the screen from left to right you could use:

```
+visual-location>
  isa      visual-location
  attended nil
  screen-x lowest
```

assuming that you also move attention to the items so that they become attended. There is one note about using lowest and highest when both screen-x and screen-y are specified. The screen-x test is always performed before the screen-y test. Thus, if you were to do the following:

```
+visual-location>
  isa      visual-location
  screen-y highest
  screen-x lowest
```

It will first find the set of objects whose locations have the lowest screen-x coordinate, and then choose the location of the one with the highest screen-y from that set.

It is also possible to use variables in the tests and there are also special values to test based on the currently attended location. The test `current` means the value must be the same as the location of the currently attended object. Assuming that `=y` is bound on the LHS of the production:

```
+visual-location>
  isa      visual-location
  screen-x current
  screen-y (greater-than =y)
```

will find a location that has the same x coordinate as where it is attending now, but is farther down the screen than the value of `=y`.

If you want to test relative to the currently attended object you can use `greater-than-current` and `less-than-current`. The following test will find a location up and to the right of the currently attended object:

```
+visual-location>
  isa      visual-location
  screen-x greater-than-current
  screen-y less-than-current
```

The nearest slot can be used to find objects that are in locations closest to the currently attended location, or some other location. To find the location of the object nearest to the currently attended location use the `current` test:

```
+visual-location>
  isa      visual-location
  nearest current
```

It is also possible to specify any location chunk for the nearest test, and the location of the object nearest to that location will be returned:

```
+visual-location>
  isa      visual-location
  nearest =some-location
```

If there are conditions other than `nearest` specified then they are tested first. The nearest of the locations that matches the other conditions is the one placed into the buffer.

One final note about requesting locations. If there are multiple locations that match all of the requested specifications a random one from that set will be chosen.

3.1.2 The Attended Test in Detail

The attended slot was discussed in unit 2. It encodes whether or not the model has attended the object at that location, and the possible values are **new**, **nil**, and **t**. Very often we use the fact that attention tags elements in the visual display as attended or not to enable us to draw attention to the unattended elements. Consider the following production:

```
(p find-random-letter
  =goal>
  isa      read-letters
  state    find
  tone     nil
==>
+visual-location>
  isa      visual-location
  attended nil
=goal>
  state    attending)
```

In its action, this production requests the location of an object that has not yet been attended (attended nil). Otherwise, it places no preference on the location to be selected and so will choose randomly. After a feature is attended (with a **+visual** request), it will be tagged as attended t and this production's request for a visual-location will not return the location of such an object. However, there is a limit to the number of objects which can be tagged as attended t and there is a time limit on how long an item will remain marked as attended t. These attentional markers are called finsts (INSTantiation FINgers) and are based on the work of [Zenon Pylyshyn](#). The number of finsts and the decay time can be set with the perceptual motor parameters :visual-num-finsts and :visual-finst-span respectively.

The default number of finsts is four, and the default decay time is three seconds. Thus, with these default settings, at any time there can be no more than four objects marked as attended t, and after three seconds the attended state of an item will revert from t to nil. Also, when attention is shifted to an item that would require more finsts than there are available the oldest one is reused for the new item i.e. if there are four items marked with finsts as attended t and you move attention to a fifth item the first item that had been attended will now be marked as attended nil and the fifth item will be marked as attended t. Because the default value is small, productions like the one above are not very useful for modeling tasks with a large number of items on the screen because the model will end up revisiting items very quickly. One solution is to always set the parameter to a value that works for your task, but one of the goals of ACT-R modeling is to produce parameter free models, so a different approach is generally desired.

One way to avoid such a problem is to encode an explicit visual search strategy in the model that does not require the attention marker to do all of the work. With an unlimited number of finsts the following production (in conjunction with productions that move attention to the word's location and then harvest the word after it is attended) could be used to read words on the screen from left to right

```
(p read-next-word
  =goal>
  isa      read-word
  state    find
==>
+visual-location>
  isa      visual-location
  attended nil
  screen-x lowest
=goal>
  state    attend)
```

However, if there are fewer finsts available than words to be read that production will result in a loop that reads only one more word than there are finsts. As was discussed above, other specifications can be used for finding locations and if some of those are applied here we do not need to worry about how many finsts are available. If the production is changed to this

```
(p read-next-word
  =goal>
    isa      read-word
    state    find
==>
+visual-location>
  isa      visual-location
  screen-x greater-than-current
  nearest  current
=goal>
  state    attend)
```

it will always be able to find the next word to the right of the currently attended one.

Lets look back at the icon for the sperling task displayed above. You will note that all the characters are initially tagged as attended **new**. That means that they have not yet been attended and that they have been added to the icon recently. The time that items remain marked as **new** is parameterized and defaults to .5 seconds (it can be changed with the :visual-onset-span parameter). After that time if they still have not been attended they will be tagged as attended **nil**. This allows attention to be sensitive to the onset of an item. As we saw in the previous unit, visual attention has to be shifted to the object before a representation of it is built in the **visual** buffer and it can be accessed by a production rule. This corresponds to the research in visual attention showing that preattentively we have access to features of an object but we do not have access to its identity. This preattentive access to the objects is available through the **visual-location** buffer where we can specify certain features of the object. When we move the model's attention to an object its attentional status is changed. So if we move its attention to the y and then the d we would get the following:

```
? (pm-print-icon)
```

Loc	Att	Kind	Value	Color	ID
(180 211)	T	TEXT	"d"	BLACK	TEXT44
(180 161)	NIL	TEXT	"j"	BLACK	TEXT45
(80 211)	NIL	TEXT	"w"	BLACK	TEXT46
(180 111)	NIL	TEXT	"m"	BLACK	TEXT47
(80 161)	NIL	TEXT	"f"	BLACK	TEXT48
(230 211)	NIL	TEXT	"r"	BLACK	TEXT49
(80 111)	NIL	TEXT	"n"	BLACK	TEXT50
(230 161)	NIL	TEXT	"x"	BLACK	TEXT51
(130 211)	T	TEXT	"y"	BLACK	TEXT52
(230 111)	NIL	TEXT	"s"	BLACK	TEXT53
(130 161)	NIL	TEXT	"g"	BLACK	TEXT54
(130 111)	NIL	TEXT	"z"	BLACK	TEXT55

where the T's for these elements indicate that they have now been attended.

To keep this unit simple the number of finsts and the finst span will be set to values large enough that it does not have to be considered. This unit is concerned with how the minimum time to search the display determines the behavior of the system, and the searching will be based only on the marking of the attended feature. We will go through an example that involves the Sperling task where one is encoding a briefly presented array of letters. Then the assignment will be to model a subitizing task where one is trying to count the elements on the display.

3.2 The Sperling Task

If you open the **sperling** model, you will see an example of the effects of visual attention. This model contains functions for administering the Sperling experiment where subjects are briefly presented with a set of letters and must try to report them. Subjects see displays of 12 letters such as:

```
V   N   Q   M
J   R   T   B
K   C   X   W
```

This model reproduces the partial report version of the experiment. In this condition, subjects are cued sometime after the display comes on as to which of the three rows they must report. The delay is either 0, .15, .3, or 1 second after the display appears. Then, after 1 second of total display time, the screen is cleared and the subject is to report the letters from the cued row. In the version we have implemented the responses are to be typed in and the space bar pressed to indicate completion of the reporting. The original experiment uses a tone with a different frequency for each row. You can run yourself through this experiment by changing ***actr-enabled-p*** to nil and reloading

```
(setf *actr-enabled-p* nil)
```

However, instead of different tones you will hear a different number of beeps for each row. There will be one beep for the top row, two for the second and three for the third (this will not work in ACL, you will only hear one beep no matter which row should be reported). To run yourself through the experiment with each of the cue times, you can call the function **repeat-experiment** with a number n that controls the number of times you go through the entire procedure.

Below is a comparison of my performance over 10 passes through the experiment with that of the data collected by Sperling:

```
? (repeat-experiment 10)
CORRELATION: 0.511
MEAN DEVIATION: 0.502
Condition      Current Participant      Original Experiment
0.00 sec      2.40                    3.03
0.15 sec      2.30                    2.40
0.30 sec      2.80                    2.03
1.00 sec      1.60                    1.50
```

For numerous reasons your experience in the experiment is a crude approximation to the original experiment. It is generally believed that there is an iconic memory that holds the stimuli for some time after onset. We have simulated this for ACT-R by having the display stay on for a random period of time from 0.9 to 1.1 seconds while in the original experiment the display is only presented for 50 ms. Consequently you are seeing the display for longer than the original participants did and the cue comes one while the display is still physically present. On the other hand the participants in the original experiment were extremely well practiced and unless you do this for many trials you will not be. The actual visual conditions are also very different, but this representation is sufficient for the purpose of a simple illustrative model.

The following is the trace of ACT-R's performance when the model was run through the task. In this trace the sound is presented .15 seconds after onset of the display and the target row was the middle one:

```
> (do-trial 0.15)

Time 0.000: Vision found LOC87814
Time 0.000: Attend-Medium Selected
Time 0.050: Attend-Medium Fired
Time 0.050: Module :VISION running command MOVE-ATTENTION
Time 0.135: Module :VISION running command ENCODING-COMPLETE
Time 0.135: Vision sees TEXT87812
Time 0.135: Encode-Row-And-Find Selected
Time 0.185: Encode-Row-And-Find Fired
Time 0.185: Module :VISION running command FIND-LOCATION
Time 0.185: Vision found LOC87816
Time 0.185: Attend-Low Selected
Time 0.235: Attend-Low Fired
Time 0.235: Module :VISION running command MOVE-ATTENTION
Time 0.235: Detected-Sound Selected
Time 0.285: Detected-Sound Fired
Time 0.285: Module :AUDIO running command ATTEND-SOUND
Time 0.320: Module :VISION running command ENCODING-COMPLETE
Time 0.320: Vision sees TEXT87805
Time 0.320: Encode-Row-And-Find Selected
Time 0.370: Encode-Row-And-Find Fired
Time 0.370: Module :VISION running command FIND-LOCATION
Time 0.370: Vision found LOC87820
Time 0.370: Attend-Low Selected
Time 0.420: Attend-Low Fired
Time 0.420: Module :VISION running command MOVE-ATTENTION
Time 0.505: Module :VISION running command ENCODING-COMPLETE
Time 0.505: Vision sees TEXT87803
Time 0.505: Encode-Row-And-Find Selected
Time 0.555: Encode-Row-And-Find Fired
Time 0.555: Module :VISION running command FIND-LOCATION
Time 0.555: Vision found LOC87822
Time 0.555: Attend-High Selected
Time 0.570: Module :AUDIO running command BUILD-SEVT-DMO
Time 0.605: Attend-High Fired
Time 0.605: Module :VISION running command MOVE-ATTENTION
Time 0.605: Sound-Respond-Medium Selected
Time 0.655: Sound-Respond-Medium Fired
Time 0.690: Module :VISION running command ENCODING-COMPLETE
Time 0.690: Vision sees TEXT87809
Time 0.690: Encode-Row-And-Find Selected
Time 0.740: Encode-Row-And-Find Fired
Time 0.740: Module :VISION running command FIND-LOCATION
```

```

Time 0.740: Vision found LOC87824
Time 0.740: Attend-Medium Selected
Time 0.790: Attend-Medium Fired
Time 0.790: Module :VISION running command MOVE-ATTENTION
Time 0.875: Module :VISION running command ENCODING-COMLETE
Time 0.875: Vision sees TEXT87811
Time 0.875: Encode-Row-And-Find Selected
Time 0.925: Encode-Row-And-Find Fired
Time 0.925: Module :VISION running command FIND-LOCATION
Time 0.925: Vision found LOC87814
Time 0.925: Attend-Medium Selected
Time 0.975: Attend-Medium Fired
Time 0.975: Module :VISION running command MOVE-ATTENTION
Time 1.060: Module :VISION running command ENCODING-COMLETE
Time 1.060: Start-Report Selected
Time 1.110: Start-Report Fired
Time 1.171: Text87812 Retrieved
Time 1.171: Do-Report Selected
Time 1.221: Do-Report Fired
Time 1.221: Module :MOTOR running command PRESS-KEY
Time 1.304: Text87811 Retrieved
Time 1.471: Module :MOTOR running command PREPARATION-COMLETE
Time 1.521: Module :MOTOR running command INITIATION-COMLETE
Time 1.621: Device running command OUTPUT-KEY

```

<< Window "Sperling Experiment" got key #\m at time 1621 >>

```

Time 1.771: Module :MOTOR running command FINISH-MOVEMENT
Time 1.771: Do-Report Selected
Time 1.821: Do-Report Fired
Time 1.821: Module :MOTOR running command PRESS-KEY
Time 2.021: Module :MOTOR running command PREPARATION-COMLETE
Time 2.071: Module :MOTOR running command INITIATION-COMLETE
Time 2.171: Device running command OUTPUT-KEY

```

<< Window "Sperling Experiment" got key #\w at time 2171 >>

```

Time 2.321: Module :MOTOR running command FINISH-MOVEMENT
Time 2.821: Failure Retrieved
Time 2.821: Stop-Report Selected
Time 2.871: Stop-Report Fired
Time 2.871: Module :MOTOR running command PRESS-KEY
Time 3.021: Module :MOTOR running command PREPARATION-COMLETE
Time 3.071: Module :MOTOR running command INITIATION-COMLETE
Time 3.081: Device running command OUTPUT-KEY

```

<< Window "Sperling Experiment" got key #\Space at time 3081 >>

```

Time 3.171: Module :MOTOR running command FINISH-MOVEMENT
Time 3.171: Checking for silent events.
Time 3.171: * Nothing to run: No productions, no events.

```

```

answers: ("W" "X" "Y" "M")
responses: ("w" "m")

```

While the sound is presented at .150 seconds into the run it is only encoded when **Sound-Respond-Medium** fires at .655 seconds into the run. One of the things we will discuss is what determines the delay of that response. Prior to that time the model is finding letters any where on the screen. After the sound is encoded the search is restricted to the middle row. After the display disappears, the production **Start-Report** fires which initiates the keying of the letters that have been encoded from the middle row.

3.3 Visual Attention

As in the models from the last unit there are three steps that the model must perform to encode visual objects. It must find the location of an object, shift attention to that location, and then harvest the object encoded when attention shifts to the location. In the last unit this was done with three separate productions, but in this unit because the model is trying to do this as quickly as possible the encoding and request to find the next are combined into a single production, and for the first item there is no production that does an initial find. Notice that the first production to fire is this one:

```
(p attend-medium
  =goal>
    isa      read-letters
    state    attending
  =visual-location>
    isa      visual-location
  > screen-y 154
  < screen-y 166
  =visual-state>
    isa      module-state
    modality free
==>
  =goal>
    location medium
    state    encode
  +visual>
    isa      visual-object
    screen-pos =visual-location)
```

which tests that there is a chunk in the **visual-location** buffer. It then encodes in the location slot of the goal which tone the letter corresponds to based on the position on the screen and requests a shift of visual attention to the object at that location. It matches and fires even though there has not been a request to put a chunk into the **visual-location** buffer. However, there is a line in the trace prior to that which indicates that a visual-location was found:

```
Time 0.000: Vision found LOC87814
Time 0.000: Attend-Medium Selected
Time 0.050: Attend-Medium Fired
```

This process is called buffer stuffing and it occurs for both visual and aural percepts. It is intended as a simple approximation of a bottom-up mechanism of attention. When the **visual-location** buffer is empty and the model processes the display it might automatically place the location of one of the visual objects into the **visual-location** buffer. You can specify the conditions for what gets buffer stuffed using the same conditions you would use to specify a regular +visual-location request. Thus, when the screen is processed, if there is a visual-location that matches that specification and the **visual-location** buffer is empty, then that location will be stuffed into the **visual-location** buffer.

The default specification for a visual-location to be stuffed into the buffer is attended new and screen-x lowest. If you go back and run the previous units' models you can see that before the

first production fires to request a visual-location there is in fact already one in the buffer, and it is the leftmost new item on the screen.

By using buffer stuffing the model can detect changes to the screen. The alternative method would be to continually request a location that was attended new, notice that there was a failure to find one, and request again until one was found.

One thing to keep in mind is that buffer stuffing only occurs if the buffer is empty. So if you want to take advantage of it you must make sure that the **visual-location** buffer is cleared before the update on which you want a location to be stuffed.

Something else to notice about this production is that the buffer test of the **visual-location** buffer shows two new modifiers that can be used when testing slots for values. These tests allow you to do a comparison when the slot value is a number, and the match is successful if the comparison is true. The first one (>) is a greater-than test. If the chunk in the **visual-location** buffer has a value in the screen-y slot that is greater than 104 it is a successful match. The second test (<) is a less-than test, and works in a similar fashion. If the screen-y slot value is less than 116 it is a successful match. Testing on a range of values like this is important for the visual locations because the exact location of a piece of text in the icon is determined by its “center” which is dependant on the font type and size. Thus, instead of figuring out exactly where the text is at in the icon (which can vary from letter to letter or for a given letter on different computers when not using the ACT-R environment) the model is written to accept the text in a range of positions.

After attention shifts, the production **encode-row-and-find** harvests the visual representation of the object, marks it with its row designation for future reference, and requests the next location:

```
(p encode-row-and-find
  =goal>
    isa      read-letters
    location =pos
    upper-y  =uy
    lower-y  =ly
  =visual>
    isa      text
    status   nil
==>
  =visual>
    status   =pos
  =goal>
    location nil
    state    attending
  +visual-location>
    isa      visual-location
    attended nil
    screen-y (within =uy =ly))
```

Note that this production places the row of the letter (=pos having values high, medium, and low) into the status slot of the visual object currently in the **visual** buffer (=visual). Later, when reporting, the system will restrict itself to recalling items from the designated row.

The request for a location uses a within modifier for the screen-y and the bounds used for the within are taken from slots of the goal. The initial values for the upper-y and lower-y slots are shown in the initial goal:

```
(goal isa read-letters state find upper-y 0 lower-y 300)
```

and include the whole window, thus the location of any letter that is unattended will be potentially chosen. When the tone is encoded those slots will be updated so that only the target row's letters will be found.

There is one important feature to emphasize about this model, which will be critical to the assignment to follow. The model never repeats a letter because of the attended nil test in the requests to the visual location buffer.

3.4 Auditory Attention

There are a number of productions responsible for processing the auditory message and they serve as our first introduction to the auditory buffers. As in the visual case, there is an **aural-state** buffer to hold the state of the aural system, **aural-location** to hold the location of an aural message, and an **aural** buffer to hold the sound that is attended. However, unlike the visual system we typically need only two steps to encode a sound and not three. This is because usually the auditory field of the model is not crowded with sounds and we can often rely on buffer stuffing to place the sound's location into the **aural-location** buffer. If a new sound is presented, and the **aural-location** buffer is empty, then the audio-event for that sound (the auditory equivalent of a visual-location) is placed into the buffer automatically. However, there is a delay between the initial onset of the sound and when the buffer is stuffed. The length of the delay depends on the type of sound being presented (tone, digit, or other) and represents the time necessary to encode its content. This model only hears tones, for which the default delay is .050 seconds. The **Detected-Sound** production responds to the appearance of an audio-event in the **aural-location** buffer:

```
(p detected-sound
  =aural-location>
  isa      audio-event
  attended nil
  =aural-state>
  isa      module-state
  modality free
==>
  -aural-location>
  +aural>
  isa      sound
  event    =aural-location)
```

Notice that this production does not test the goal. If there is an audio-event in the **aural-location** buffer and the **aural-state** is free this production can fire. It is not specific to this, or any task. On its RHS it clears the **aural-location** buffer (to prevent it from firing again) and requests that attention shift to the sound.

Our model for this task has three different productions to encode the sounds, one for each of high, medium, and low tones. The following is the production for the low tone:

```
(p sound-respond-low
  =goal>
    isa      read-letters
    tone     nil
  =aural>
    isa      sound
    content  500
==>
  =goal>
    tone     low
    upper-y  205
    lower-y  215)
```

The content slot of a tone sound encodes the frequency of the tone. A 500 Hertz sound is considered low, a 1000 Hertz sound medium, and a 2000 Hertz sound high. On the RHS it records the type of tone presented in the goal and updates the restrictions on the y coordinates for the search to constrain it to the appropriate row.

It takes some time for the impact of the tone to make itself felt on the information processing. Consider this portion of the trace in the case where the tone was sounded .150 seconds after the onset of the display:

```
> (do-trial 0.15)

Time 0.000: Vision found LOC87814
Time 0.000: Attend-Medium Selected
Time 0.050: Attend-Medium Fired
Time 0.050: Module :VISION running command MOVE-ATTENTION
Time 0.135: Module :VISION running command ENCODING-COMPLETE
Time 0.135: Vision sees TEXT87812
Time 0.135: Encode-Row-And-Find Selected
Time 0.185: Encode-Row-And-Find Fired
Time 0.185: Module :VISION running command FIND-LOCATION
Time 0.185: Vision found LOC87816
Time 0.185: Attend-Low Selected
Time 0.235: Attend-Low Fired
Time 0.235: Module :VISION running command MOVE-ATTENTION
Time 0.235: Detected-Sound Selected
Time 0.285: Detected-Sound Fired
Time 0.285: Module :AUDIO running command ATTEND-SOUND
Time 0.320: Module :VISION running command ENCODING-COMPLETE
Time 0.320: Vision sees TEXT87805
Time 0.320: Encode-Row-And-Find Selected
Time 0.370: Encode-Row-And-Find Fired
Time 0.370: Module :VISION running command FIND-LOCATION
Time 0.370: Vision found LOC87820
Time 0.370: Attend-Low Selected
Time 0.420: Attend-Low Fired
Time 0.420: Module :VISION running command MOVE-ATTENTION
Time 0.505: Module :VISION running command ENCODING-COMPLETE
Time 0.505: Vision sees TEXT87803
Time 0.505: Encode-Row-And-Find Selected
Time 0.555: Encode-Row-And-Find Fired
Time 0.555: Module :VISION running command FIND-LOCATION
Time 0.555: Vision found LOC87822
Time 0.555: Attend-High Selected
Time 0.570: Module :AUDIO running command BUILD-SEVT-DMO
Time 0.605: Attend-High Fired
Time 0.605: Module :VISION running command MOVE-ATTENTION
Time 0.605: Sound-Respond-Medium Selected
```

```

Time 0.655: Sound-Respond-Medium Fired
Time 0.690: Module :VISION running command ENCODING-COMplete
Time 0.690: Vision sees TEXT87809
Time 0.690: Encode-Row-And-Find Selected
Time 0.740: Encode-Row-And-Find Fired
Time 0.740: Module :VISION running command FIND-LOCATION
Time 0.740: Vision found LOC87824

```

Although the sound was initiated at .150 seconds, it takes .050 seconds to detect the nature of the sound. Thus, its event appears in the **aural-location** buffer at .200 seconds. At .235 seconds **Detected-Sound** can be selected in response to the event that happened and when this production completes at .285 seconds aural attention is shifted to the sound.

```

Time 0.235: Detected-Sound Selected
Time 0.285: Detected-Sound Fired
Time 0.285: Module :AUDIO running command ATTEND-SOUND

```

Attending to and fully encoding the sound takes .285 seconds so at .570 seconds the recognized sound becomes available in the **aural** buffer while the production **attend-high** is firing:

```

Time 0.555: Attend-High Selected
Time 0.570: Module :AUDIO running command BUILD-SEVT-DMO
Time 0.605: Attend-High Fired
Time 0.605: Module :VISION running command MOVE-ATTENTION
Time 0.605: Sound-Respond-Medium Selected
Time 0.655: Sound-Respond-Medium Fired

```

The production **Sound-Respond-Medium** is then selected and fires at .605 seconds (after the **attend-high** production completes). The next production to fire is **Encode-Row-And-Find**. It encodes the last letter that was read and issues a request to look at a letter that is in the correct row this time, instead of an arbitrary letter. Thus, even though the sound is given at .150 seconds it is not until .655 seconds that it has any impact on the processing of the visual array when the first request for a target letter is initiated.

3.5 Typing and Control

The production that initiates typing the answer is:

```

(P start-report
  =goal>
    isa      read-letters
    tone     =tone
  =visual-state>
    isa      module-state
    modality free
  ==>
  +goal>
    isa      report-row
    row      =tone
  +retrieval>
    isa      text
    status   =tone)

```

This causes a new chunk to be placed into the **goal** buffer rather than a modification to the chunk that is currently there (as indicated by the +goal rather than an =goal). The goal is no longer to

read letters but rather to report the target row. Note also that this production issues a retrieval request for a letter in the target row.

This production can match at many points in the model's run, but we do not want it to apply as long as there are letters to be perceived. We only want this rule to apply when there is nothing else to do. We can make this production less preferred by setting its cost value high. By default productions have a cost of .05 seconds. However, for this application we set the values of certain productions to be different, including **Start-Report**. The function for setting production parameters is **spp** (set production parameters). It is similar to **sgp** which is used for the global parameters as discussed in the last unit. The cost of a production is set with the `:c` parameter, so the following call sets the cost of the **Start-Report** production to 2 seconds:

```
(spp start-report :c 2)
```

By giving it a higher cost we guarantee that it will not be selected as long as there are other productions with a lower cost that will match, and in particular that will be as long as there is still something in the target row on the screen to be processed by the productions that encode the screen. Also note that the productions that process the sound are given lower values than the default 0.05 seconds:

```
(spp detected-sound :c 0)
(spp sound-respond-low :c 0)
(spp sound-respond-medium :c 0)
(spp sound-respond-high :c 0)
```

This is so that the sound will be processed as soon as possible. In later units, when we discuss conflict resolution in detail, we will learn more psychologically plausible ways to select among production rules.

For the change in cost to have an impact on the model there is a global parameter that must be set, which can be seen in the commands window:

```
(sgp :v t :esc t)
```

`esc` is the enable subsymbolic computations parameter and it must be set to `t` so that the system will compute and use the subsymbolic quantities (in this case production utility which reflects the costs).

Once the report starts, the following production is responsible for reporting all the letters in the target row:

```
(P do-report
  =goal>
  isa      report-row
  row      =tone
  =retrieval>
  isa      text
  status   =tone
  value    =val
  =manual-state>
  isa      module-state
  modality free
==>
```

```

+manual>
  isa      press-key
  key      =val
=retrieval>
  status   nil
+retrieval>
  isa      text
  status   =tone)

```

In the trace you will notice that this production waits until the keying of the last key is complete:

```

Time 1.060: Start-Report Selected
Time 1.110: Start-Report Fired
Time 1.171: Text87812 Retrieved
Time 1.171: Do-Report Selected
Time 1.221: Do-Report Fired
Time 1.221: Module :MOTOR running command PRESS-KEY
Time 1.304: Text87811 Retrieved
Time 1.471: Module :MOTOR running command PREPARATION-COMPLETE
Time 1.521: Module :MOTOR running command INITIATION-COMPLETE
Time 1.621: Device running command OUTPUT-KEY

```

```
<< Window "Sperling Experiment" got key #\m at time 1621 >>
```

```

Time 1.771: Module :MOTOR running command FINISH-MOVEMENT
Time 1.771: Do-Report Selected
Time 1.821: Do-Report Fired

```

This is because of the manual-state test on the LHS. Until the motor module is completely free this production cannot fire.

On the RHS notice that the letter that is retrieved has its status set to nil. This is done so that the model does not retrieve the same letter again.

When there are no more letters to be reported, the following production applies to terminate processing:

```

(p stop-report
  =goal>
    isa      report-row
    row      =row
  =retrieval>
    isa      error
  =manual-state>
    isa      module-state
    modality free

==>
+manual>
  isa      press-key
  key      space
=goal>
  row      nil)

```

This production will apply when the attempt to retrieve another letter ends in an error -- i.e., the system can remember no more letters in the target row. It clears the row slot of the goal (sets it to nil) which prevents this production from firing again to stop the model. The reason that it cannot

fire again is because the value of the row slot is tested on the LHS and bound to the variable =row, but a slot containing nil cannot be bound to a variable.

3.6 Data Fitting

One can see the average performance of the model run over a large number of trials by using the function **repeat-experiment** and giving it the number of trials one wants to see run. To make this go faster there are a few things that can be done.

With respect to the parameters, turning off the trace and the real-time performance by setting them to nil will improve the time required to run the model:

```
(sgp :v nil :esc t)
(pm-set-params :show-focus t :real-time nil)
```

You will also want to turn off the printing of the answers and responses:

```
(setf *show-responses* nil)
```

It can be sped up even more by making the model use a virtual window instead of a real one. The details of how to do that are in the unit 3 experiment code document. A virtual window is an abstraction of a real window (a real window is a displayed window which both a person and a model can interact with) that the model can “see” and interact with as if it were a real window without the overhead of actually displaying and updating it on the screen.

When one calls repeat-experiment with all of these changes, one sees something similar to:

```
> (repeat-experiment 100)
CORRELATION: 0.997
MEAN DEVIATION: 0.115

Condition      Current Participant      Original Experiment
0.00 sec.      3.20                    3.03
0.15 sec.      2.43                    2.40
0.30 sec.      2.17                    2.03
1.00 sec.      1.56                    1.50
```

This prints out the correlation and mean deviation between the experimental data and the average of the 100 ACT-R simulated runs. Also printed out is the original data from the Sperling experiment.

3.7 The Subitizing Task

Your task is to write a model for the subitizing task. This is an experiment where you are presented with a set of marks on the screen (in this case X's) and you have to count how many there are. If you open the **subitize** model you can run yourself in this experiment because by default ACT-R is not enabled:

```
(setf *actr-enabled-p* nil)
```

If you call the function **experiment** you will be presented with 10 trials in which you will see from 1 to 10 objects on the screen. The trials will be in a random order. You should press the number key that corresponds to the number of items on the screen unless there are 10 objects in which case you should type 0. The following is the outcome from one of my runs through the task:

```
? (experiment)

CORRELATION: 0.956
MEAN DEVIATION: 0.367
Items      Current Participant      Original Experiment
  1          0.80(T )           0.60
  2          0.93(T )           0.65
  3          0.91(T )           0.70
  4          1.16(T )           0.86
  5          1.46(T )           1.12
  6          1.84(T )           1.50
  7          1.75(T )           1.79
  8          2.85(T )           2.13
  9          2.73(T )           2.15
 10          2.58(T )           2.58
```

This provides a comparison between my data and the data from an experiment by Jensen, Reese, & Reese (1950). The value in parenthesis after the time will be either T or NIL indicating whether or not the answer the participant gave was correct (T is correct, and NIL is incorrect).

3.7.1 The Vocal System

We have already seen that the default ACT-R mechanism for pressing the keys takes a considerable amount of time, which would have a serious impact on the results of this model. One solution would be to more explicitly control the hand movements to provide faster responses, but that is beyond the scope of this unit. For this task the model will provide a vocal response i.e. it is going to say the number of items on the screen (in fact if you have a Macintosh with Apple's Speech Manager software MacInTalk you can hear the model speak). This is done by making a request of the vocal system in a manner exactly like the requests to the manual system.

Here is the production in the Sperling model that presses a key:

```
(P do-report
  =goal>
    isa      report-row
    row      =tone
  =retrieval>
    isa      text
    status   =tone
    value    =val
  =manual-state>
    isa      module-state
    modality free
==>
  +manual>
    isa      press-key
    key      =val
  =retrieval>
    status   nil
  +retrieval>
    isa      text
    status   =tone)
```

With the following changes it will now speak the response instead (note however that experiment is not written to accept a vocal response so it will not properly score those responses if you attempt to run the model with these modifications):

```
(P do-report
  =goal>
    isa      report-row
    row      =tone
  =retrieval>
    isa      text
    status   =tone
    value    =val
  =vocal-state>
    isa      module-state
    modality free
==>
  +vocal>
    isa      speak
    string   =val
  =retrieval>
    status   nil
  +retrieval>
    isa      text
    status   =tone)
```

On the LHS we test the **vocal-state** to make sure that the speech module is not currently in use:

```
=vocal-state>
  isa      module-state
  modality free
```

Then on the RHS we make a request of the **vocal** buffer to speak the response:

```
+vocal>
  isa      speak
  string   =val
```

The default timing for speech acts is .200 seconds per assumed syllable based on the length of the string to speak. That value works well for this assignment so we will not go into the details of adjusting it.

3.7.2 Exhaustively Searching the Visual Icon

When the model is doing this task it will need to exhaustively search the display. It can use the ability of the visual system to tag as attended those elements that have been attended and not go back to them -- just as in the Sperling task. To make the task easier, the number of finsts has been set to 10 in the code provided so you do not have to worry about that. However, the model also has to be able to detect when there are no more unattended visual locations. This will be signaled by an error when a request is made of the **visual-location** buffer that cannot be satisfied. This is the same as when the **retrieval** buffer reports an error when no chunk that matches the request can be retrieved. The way for a production to test for that would be to have the following test on the left-hand side:

```
(p respond
...
  =visual-location>
  isa error
...
==>
...)
```

When no location can be found to satisfy a request of the **visual-location** buffer the failure chunk, which isa error, is placed into the **visual-location** buffer.

3.7.3 The Assignment

Your task is to write a model for the subitizing task that does an approximate job of reproducing the data. The following are the results from my ACT-R model:

```
CORRELATION: 0.980
MEAN DEVIATION: 0.230
```

Items	Current	Participant	Original	Experiment
1	0.54	(T)	0.60	
2	0.77	(T)	0.65	
3	1.01	(T)	0.70	
4	1.24	(T)	0.86	
5	1.48	(T)	1.12	
6	1.71	(T)	1.50	
7	1.95	(T)	1.79	
8	2.18	(T)	2.13	
9	2.42	(T)	2.15	
10	2.65	(T)	2.58	

You can see this does a fair job of reproducing the range of the data. However, participants show little effect of set size (approx. 0.05-0.10 seconds) in the range 1-4 and a larger effect (approx. 0.3 seconds) above 4 in contrast to this model which increases about .23 seconds for each item. The small effect for little displays probably reflects the ability to perceive small numbers of objects as familiar patterns and the larger effect for large displays probably reflects the time to retrieve count

facts (which we will not accurately model until we get to subsymbolic modeling). Still this reflects a relatively good approximation that you should aspire to match.

You are provided with chunks that encode numbers and their ordering from 0 to 10:

```
(add-dm (n0 isa number-fact identity zero next one value "0")
        (n1 isa number-fact identity one next two value "1")
        (n2 isa number-fact identity two next three value "2")
        (n3 isa number-fact identity three next four value "3")
        (n4 isa number-fact identity four next five value "4")
        (n5 isa number-fact identity five next six value "5")
        (n6 isa number-fact identity six next seven value "6")
        (n7 isa number-fact identity seven next eight value "7")
        (n8 isa number-fact identity eight next nine value "8")
        (n9 isa number-fact identity nine next ten value "9")
        (n10 isa number-fact identity ten next eleven value "0")
        (goal isa count state start))
```

They also contain a slot called value that holds the string of the number to be spoken.

The **experiment** function does not take a parameter to specify how many times to run the experiment. Unlike the Sperling task there is no randomness in the experiment other than the order of the trials, so it is not necessary to run the model multiple times and average the results to assess the model's performance. There is also a function called **do-trial** that can be called to run a single trial. It takes one parameter which is the number of items to display and it returns a list of the time of the response and whether or not the answer given was correct.

```
>(do-trial 4)
(1.24 T)
```

One final note, as with the other models you have worked with so far, the model will be reset before each trial. Because of that, you do not need to take advantage of buffer stuffing to detect the screen changes between trials, but you may still use it if you want.