

Introduction

This document will provide a description of the GUI tools available in ACT-R for producing experiments for models (referred to as the AGI for ACT-R GUI Interface) as well as some of the general ACT-R commands one would use for running models. These commands are used in the experiments which are used for the models included with the ACT-R tutorial. So, if you would like to see examples of their use you can look at those models and the corresponding experiment code description documents which accompany the tutorial units.

Before describing the AGI commands themselves however there will be some brief discussion of the low level interface to ACT-R's perceptual and motor systems and the tools upon which the AGI is built.

A Device

ACT-R interacts with the world through what we call a device. A device is an abstract representation of the world (typically a simulated computer) implemented as an object in Lisp. There are a handful of methods that must be defined on that object so that ACT-R will be able to “see” and “manipulate” that device. In general, defining such a device object is all that is necessary to produce something with which ACT-R can interact, and the slides titled “extending-actr” found in the docs directory of the ACT-R 6.0 distribution describes the details of creating a new device. When one needs to specify a new or different “world” for ACT-R then creating a device is the way to do so, but for simple experiments it’s often easier to just work with the simple device which is already available for use through the AGI.

Real GUI Windows as Devices

Built into ACT-R 6.0 are devices for some limited interaction with the native Lisp GUIs and interface widgets of Allegro Common Lisp, Macintosh Common Lisp (both the commercial version and also the free RMCL), and LispWorks. Thus if you build your interface with the native tools for those Lisps using the subset of widgets supported (or add the necessary support for your own) it is likely that the model will be able to interact with that interface with very little extra effort. There are some potential difficulties with that however. The first is that you have to learn how to use the GUI tools of the Lisp system you are using if you do not know them already. Another is that the GUI systems of the Lisps are not compatible or portable i.e. if you build it in MCL it is only going to work in MCL. The last is that there is no documentation for the specific Lisp devices other than the code which implements them which can be found in the devices directory of the ACT-R 6.0 distribution.

Virtual Windows

Virtual windows are an abstract representation, based roughly on the windowing system of MCL, built into ACT-R 6.0 that implements a windowing interface which is portable across Lisps. It is called virtual because it does not display anything – it is entirely abstract and only “visible” to the model (except as described below). Because it does not open real windows it is often much faster than the native windows on a given system, and can be used when speed is important. As with real windows described above, one may use the virtual windowing device directly, but as with the real windows, there is no documentation other than that found in the virtual windows’ device implementation source code.

Visible Virtual Windows

If the ACT-R Environment is connected to a running ACT-R and the AGI command for creating a window is called asking for a visible window that will cause the Environment to create and display the window instead of using the native windowing capabilities of

the Lisp (if it has any). That happens because the Environment forces the virtual device to be used for all interfaces and it has the ability to display those virtual interfaces when requested. Thus, even for command line only Lisps it is possible to create and see graphic experiments for the model using the AGI commands when the ACT-R Environment is also used.

The UWI

The first attempt to create a set of commands for ACT-R GUI use was something called the Uniform Windowing Interface (UWI). It was a set of low level windowing functions which were implemented in various Lisps and for the virtual windows to allow them to operate in a portable manner. It was not particularly user friendly and thus it is no longer supported because it has been replaced by the AGI. It is still implemented in the code and is used by the AGI functions, but it is not recommended for use directly.

The AGI

The high level interface that we provide for GUI construction for models is the AGI. It is a small set of tools designed to make creating simple experiments for ACT-R models easy. It also makes it possible for the experimenter to interact with the same experiment as well for testing and debugging with very little additional work (the AGI is not recommended for use in creating experiments for human participants however because it does not provide any support for high fidelity timing of real actions) . It is not designed for building complex experiments because those are often best suited to being implemented as a custom device.

The rest of this document will describe the functions provided in the AGI as well as some of the important ACT-R 6.0 functions for running models. Detailed examples of the AGI in action can be found in the ACT-R 6.0 tutorial. The models for each of the units (except for the first unit which does not have any experiments) create the experiment it uses through the AGI and there is an additional text with the unit which describes the experiment code for the models of that unit.

General Experiment Design

The typical procedure for creating a simple experiment for ACT-R is the following:

1. Open a window
2. Clear the display
3. Present some stimuli
4. Run the model or wait for a real user to respond
5. Collect a response
6. Repeat steps 2-5 for different conditions/stimuli
7. Repeat steps 1-6 to simulate multiple participants
8. Analyze the results

That general pattern can be found in most of the tutorial model experiments. Other than steps 6 and 7 and any averaging of the data that may be required (which are best done with the iteration constructs and functions already present in Lisp) the AGI provides the tools for carrying out those tasks. The biggest assumption in the design of the AGI is that there is only one experiment window for the task at any time, and all of the AGI functions will operate upon that window.

The following sections will describe the specific functions provided by the AGI.

Window Control (steps 1 and 2)

Open-exp-window – this function takes one required parameter which is the title for an experiment window to create. It can also take several keyword parameters that control how the window is displayed. This function opens a window for performing an

experiment and returns that window. If there is already an experiment window open with that title it clears its contents and brings it to the foreground. If there is not already an experiment window with that title it closes the previous experiment window, if one exists, and opens a new window with the requested title and brings it to the foreground. The possible keyword parameters are :height and :width which specify the size of the window in pixels and default to 300 each, :x and :y which specify the screen coordinates of the upper left corner of the window in pixels and also default to 300 each, and :visible which can be either **t** or **nil** (the default is **t**). If :visible is **t** it specifies that a real or visible virtual window be opened, and if it is **nil** it means that a purely virtual window will be opened.

Select-exp-window – this function takes no parameters and brings the currently open experiment window to the foreground.

[Note: The select-exp-window and open-exp-window commands may not always bring a visible virtual window to the foreground because of issues with which application has the focus.]

Close-exp-window – this function takes no parameters and closes the currently open experiment window. Once the window is closed it is no longer possible for a person or model to interact with it. If a model were to attempt such interaction it is likely to cause an error in Lisp.

Clear-exp-window – this function takes no parameters and removes all of the items from the currently open experiment window.

Displaying Items (step 3)

Add-text-to-exp-window – this function takes a few keyword parameters and draws a static text string on the currently open experiment window. The return value is the implementation dependent text object created. The :text parameter specifies the text string to display and defaults to “”. The :x and :y parameters specify the pixel coordinate of the upper-left corner of the box in which the text is to be displayed, and the default value for each is 0. The :height and :width parameters specify the size of the box in which to draw the text in pixels. The default value for :height is 20 and for :width is 75. One thing to note about the :height and :width parameters is that although the text shown in a real window or a visible virtual window may be clipped at the borders of the box a model will always see the entire text string. The :color parameter specifies in which color the text will be drawn and defaults to black. The color must be a symbol naming the color (not a system dependent color value thus it should be quoted i.e. 'black) and the following color names are supported across platforms black, blue, red, green, white, pink, yellow, gray, light-blue, dark-green, purple, brown, light-gray, or dark-gray.

Add-button-to-exp-window – this function takes a few keyword parameters and places a button in the currently open experiment window. The return value is the implementation dependent button object. The :text parameter specifies the text to display on the button

and defaults to “Ok”. The :x and :y parameters specify the pixel coordinate of the upper-left corner of the button and each defaults to 0. The :height and :width parameters specify the size of the button in pixels, and :height defaults to 18 and :width defaults to 60. The :action parameter specifies a function to be called when this button is pressed and defaults to **nil** (no function to call). When provided, that function will be called with the button object itself as the only parameter. The :color parameter specifies a background color for the button, which must be one of the colors as described for add-text-to-exp-window, and defaults to gray.

Add-line-to-exp-window – this function takes two required parameters and one optional parameter and it draws a line in the currently open experiment window. The return value is the implementation dependent line object. The required parameters specify the pixel coordinates of the end points of the line and each should be a two element list of the x and y coordinate respectively for one end of the line. The optional parameter can be used to specify the color in which the line is to be drawn. It must be one of the colors as described for add-text-to-exp-window, and the default is black if it is not specified.

Remove-items-from-exp-window – this function takes an arbitrary number of parameters each of which must be an object that is currently displayed on the currently open experiment window. Each of those items is removed from the current display.

Add-items-to-exp-window – this function takes an arbitrary number of parameters each of which must be an object that has been created for the current experiment window using one of the commands above. Each of those items is then added to the current display if it is not already there.

Miscellaneous (steps 3, 4 or 5)

Permute-list – this function takes one parameter which must be a list and it returns a randomly ordered copy of that list.

While – this is a looping construct. It takes an arbitrary number of parameters. The first parameter specifies the test condition, and the rest specify the body of the loop. The test is evaluated and if it returns anything other than **nil** all of the forms in the body are executed in order. This is repeated until the test returns **nil**. Thus, while the test is true (non-nil) the body is executed.

Allow-event-manager – this function takes one parameter, which must be the window of the experiment. It calls the appropriate function of the system to handle user interaction when a real user is doing the task. It is necessary to give the system a chance to handle the real user interactions otherwise the data collection functions may never be called. It is not necessary when a model is performing the task.

Sleep – sleep is actually a function defined in ANSI Common Lisp, but because it is often used for experiment generation in the tutorial it seems appropriate to discuss it here.

The Lisp specification for `sleep` says it takes one parameter, *seconds*, which is a non-negative real, and it causes execution to cease and become dormant for approximately the seconds of real time indicated by *seconds*, whereupon execution is resumed. An important thing to note is that this is only useful for a person doing a task. **The sleep function will have no effect upon the timing of actions from the model's perspective**, but it will increase the time it takes to run the model from the user's perspective.

Model interfacing (step 4)

[More details on the commands in this section can be found in the ACT-R Reference manual included in the docs directory of the ACT-R 6.0 distribution.]

Reset – this function initializes the model to time 0, initializes all of the modules, and then sets the state of the parameters, declarative memory, and productions to those specified in the model definition.

Install-device – this function takes one parameter which must be a valid device (for example an experiment window created with `open-exp-window`). This tells the model which device it is interacting with. All of the model's actions (key presses, mouse movement and mouse clicks) will be sent to this device and the contents of this device will be what the model can “see”.

Proc-display – this function can take one keyword parameter, `:clear`. Calling this function tells the model to process the currently installed device for visual information i.e. this function makes the model “look” at the current contents of the installed device. Whenever the window is changed you must call **proc-display** again to make sure the model becomes aware of those changes. The re-encoding and buffer stuffing mechanisms response described in the tutorial can only happen after this function is called. The keyword parameter, `:clear`, if specified as `t` will cause the model to treat the window as all new items – everything there will be considered unattended.

Run – this function takes one required parameter which is the time to run a model in seconds and a keyword parameter called `:real-time`. The model will run until either the requested amount of time passes, or there is nothing left for the model to do (no productions will fire and there are no pending actions that can change the state). If the keyword parameter `:real-time` is specified as `t`, then the model is advanced in step with real time instead of being allowed to run as fast as possible in its own simulated time. That can be a useful thing to do when debugging a model or if it has to interact with software that is not designed to run with the model's simulated time.

Schedule-event-relative – This function takes 2 required parameters and a number of additional parameters. It is used to schedule functions to be called during the running of the model. The first parameter specifies an offset from the current time at which the function should be called. The second parameter is the function to call. By scheduling

functions to be called during the running of the model it is possible to have the experiment change without stopping the model to do so.

Get-time – this function takes an optional parameter and it returns the current time in milliseconds. If the optional parameter is specified as true then the time returned is the model's simulated time. If the optional parameter is specified as **nil** then the time is taken from the internal Lisp timer using the command `get-internal-real-time`. If the optional parameter is not specified, then the model's simulated time is returned. The time from the internal timer is not zero referenced with respect to the task, so one needs to make sure and record the time at the start of the trial for reference when necessary. Although the time is measured in milliseconds for a person that does not mean that it is necessarily accurate to that resolution.

Set-visloc-default – This command sets the conditions that will be used to select the visual location that gets buffer stuffed. When the screen is processed by the model (`proc-display` is called) if the visual-location buffer is empty a visual-location that matches the conditions specified by this command will be placed into the visual-location buffer. Essentially, what happens is that when `proc-display` gets called, if the visual-location buffer is empty, a `+visual-location` request is automatically executed using the slot tests specified with `set-visloc-default`.

Response collection (step 5)

When a key press or mouse click occurs in an experiment window (generated by either a person or the model) the method `rpm-window-key-event-handler` or `rpm-window-click-event-handler` respectively will be called automatically. Thus, to record such responses you must define those methods on the `rpm-window` class in your model. Those definitions would look like this:

```
(defmethod rpm-window-key-event-handler ((win rpm-window) key)
  ...
)
```

```
(defmethod rpm-window-click-event-handler ((win rpm-window) pos)
  ...
)
```

with your code to record the responses inside of them.

The parameters passed to the `rpm-window-key-event-handler` will be the window in which the key press occurs and the key that is pressed. For the 'normal' keys (letters, numbers, and simple punctuation keys) the key parameter will be the character of that item, but for other things (function keys, arrow keys, etc) it may be a system dependent character or possibly a symbol representation of that key. Thus, you will have to be careful when using such keys to make sure that your method can properly handle those items.

The parameters passed to the rpm-window-click-event-handler will be the window in which the mouse click occurs and the position of the mouse when the click occurred. The position will be a two element list of the x and y pixel coordinates within the window of the mouse pointer at the time of the click.

Data analysis (step 8)

Correlation – this function takes 2 required parameters which must be equal length ‘collections’ of numbers. The numbers can be in arrays or lists and the two parameters do not need to be in the same format (one could be an array and the other a list). The only requirement is that they have the same number of numbers in them. This function then extracts those numbers and computes the correlation between the two sets of numbers. That correlation value is returned. There is a keyword parameter :output which defaults to **t**. When :output is **t** the correlation is printed to *standard-output*. If :output is **nil** then nothing is printed, and if it is a string, stream, or pathname then that is used to open a stream (if necessary) to which the results are written.

Mean-deviation – this function operates just like correlation, except that the calculation performed is the root mean square deviation between the data sets.

Final notes

The current AGI was designed to support the ACT-R tutorial. As such, it is fairly minimal in what it provides, but that was one of the goals – to keep it simple. I hope that people find the AGI useful, and if you have any questions, suggestions, or comments about the AGI feel free to contact me at db30@andrew.cmu.edu.