Introduction

This document will provide an introduction to the tools provided for producing experiments for use with ACT-R 5.0. There are several interface layers available in the system, and this document will focus on the highest layer – the ACT-R GUI Interface (AGI) after briefly discussing the layers below it. It will also describe the ACT-R and ACT-R/PM interfacing functions used in the ACT-R 5.0 tutorial.

Device Interface

At the lowest level ACT-R/PM interacts with the world through the device interface. That 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 ACT-R/PM manual (available from the ACT-R/PM web site at http://chil.rice.edu/byrne/RPM/index.html) describes the details of using the device interface. When one needs complete control of the world for ACT-R this is the best way to get it, but for simple experiments it can be a bit too detailed and a higher level can be easier to work with.

Real Windows as Devices

Built into ACT-R/PM already are device interface specifications for native Lisp windows and some interface widgets for Allegro Common Lisp with the IDE (ACL) and Macintosh Common Lisp (MCL). Thus if you build your interface with the native tools for ACL and MCL 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 two potential difficulties with this 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. The second 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. For many users those issues are not a problem and this level works quite well. If this is how you wish to use the system, then you can find more details in the ACT-R/PM documentation.

Virtual Windows

There is an additional layer added that addresses the portability issue, and that is virtual windows. Virtual windows are an abstract representation, based on the windowing system in MCL, built into ACT-R/PM 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. 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. The down side of using them is that you still have to learn the MCL

windowing interface (though only a small subset of it) and it can be very difficult to debug a model if you cannot actually see what it is interacting with and how it is interacting.

The old UWI

To address the portability and visibility issues we built in another layer on top of that. It is called the Uniform Windowing Interface (UWI). It provides a set of fairly low level windowing functions that map onto the specific implementations in MCL, ACL and the virtual windows so that one can create real windows in a portable fashion and easily switch from real windows to virtual windows when speed is desired. However, there are several problems with using it and it is really not supported any more (though for the time being it still exists in ACT-R/PM to support the older tutorial models that used it and the AGI which currently sits on top of it). Thus, it is not going to be discussed any further in this document, and I would recommend against using the functions it provides directly.

The AGI

The new high level interface that we provide 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. It is not designed for building complex experiments because those are often best suited to being implemented at the device interface or real window level and portability is usually not an issue there. It provides tools for opening and closing a window, adding and removing text, buttons, and lines, collecting key presses and mouse clicks, as well as a couple of functions to do simple data analysis. It is portable, and it works with real windows in ACL and MCL as well as virtual windows, but perhaps its biggest advantage is that when used with the new ACT-R environment it provides "visible virtual windows" - a real window that can be seen and interacted with that uses the virtual window abstraction internally. The visible virtual windows are available for use in any Lisp that will run the environment (currently that is ACL with or without the IDE in Windows or Linux, OpenMCL or MCL 5 in Mac OS X, MCL in Mac OS 9, and LispWorks for Windows or Linux). One of the big advantages of the visible virtual windows is that if you transition from visible to purely virtual there is no difference in the representation that the model sees. That is not the case when switching from a real MCL or ACL window to a virtual window particularly when it comes to text displays because the fonts used can differ slightly between the real and the virtual which causes text to "move" when switching from one representation to the other.

The rest of this document will describe the functions provided in the AGI as well as some of the important ACT-R 5.0 functions for running models. Detailed examples of the AGI in action can be found in the ACT-R 5.0 tutorial – each unit from 2-8 has an additional text which describes the experiment code for the models of that unit, and those experiments are implemented using the AGI.

General Experiment Design

The basic procedure for 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, ACT-R 5.0, and ACT-R/PM provide 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 functions available in the AGI are described below.

Window Control (steps 1 and 2)

Open-exp-window – this function takes one required parameter which is the title for the window. 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 (defaulting to t). If :visible is t it specifies that a real or visible virtual window be opened (a visible virtual will be used if the environment is connected and the option is set to use an environment window for experiment display) 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: There are still some minor bugs to be worked out when a visible virtual window is being used and this function may not actually bring such a window to the front at this time.]

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 text object. 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) 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 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.

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 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 for one end of the line. The optional parameter can be used to specify the color in which the line is to be drawn, and the default is black if it is not specified. It must be a symbol (and as such it should be quoted) and can be any one of black, blue, red, green, white, pink, yellow, gray, light-blue, dark-green, purple, brown, light-gray, or dark-gray (be careful when using ACL in particular, because many of these names are global variables that evaluate to a color object and not the symbol).

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.

Audio Presentation (step 3)

New-tone-sound – this function takes 2 required parameters and a third optional parameter. The first parameter is the frequency of a tone to be presented to the model. The second is the duration of that tone in seconds. If the third parameter is specified then it indicates at what time the tone is to be presented, and if it is omitted then the tone is to be presented immediately. At the requested time a tone sound will be made available to the model's auditory module of the requested frequency and duration.

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. Giving the system a chance to handle the real user interactions is important because otherwise the data collection functions may never be called.

Sleep – sleep is actually a function defined in ANSI Common Lisp, but because it is being 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. This is only useful when a person is doing a task because it has no impact for a running model.

Model interfacing (step 4)

Reset – this function initializes the model to time 0 and sets the state of the parameters, working memory, and the productions to those specified in the model file the last time it was loaded.

Pm-install-device – this function takes one parameter which must be a window or device. This tells the model with which window (or device) it is interacting. All of the models actions (key presses, mouse movement and mouse clicks) will be sent to this window and the contents of this window will be what the model can "see".

Pm-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 **pm-proc-display** again to make sure the model becomes aware of those changes. The blink response described in the tutorial can only happen after this function is called, and the bottom-up visual attention mechanism discussed in unit 4 (buffer stuffing) will also only occur when 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.

Pm-run – this function takes one required parameter which is the time to run the model in seconds and a keyword parameter, :full-time. It runs the model until either the requested amount of simulation 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 called :full-time is specified as t, then the model will be advanced the entire amount of time requested even if there are no actions or events to process during that time.

Pm-timed-event – this function takes 2 required parameters and any number of additional parameters. It is used to schedule a function to be called during the running of the model. The first parameter specifies the simulation time (in seconds) at which the function should be called. The second parameter is the function to be called. Any additional parameters specified are passed to that function when it is called. By scheduling functions to be called during the running of the model it is possible to run the experiment without stopping the model for changes in the task to occur (for instance erasing the display or drawing something else). The alternative is to use the :full-time parameter of pm-run to run the model for the desired amount of time, and then when it stops, execute the function call, and then call pm-run again to continue running the model.

actr-enabled-p - this is a global variable defined in ACT-R/PM that should be set to indicate whether a person or a model is currently performing the task. If it is set to t, then the system assumes that a model is performing the task, and if it is set to nil then it assumes that a real user is performing the task.

Pm-get-time – this function takes no parameters. It returns the current time in milliseconds. If the model is doing the task (when *actr-enabled-p* is t) the time is taken from the simulation time and if a person is doing the task (when *actr-enabled-p* is nil) the time is taken from the internal timer. 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.

Pm-set-visloc-default – this command sets the conditions that will be used to select the visual location that gets buffer stuffed. It takes keyword parameters that correspond to the slots that would be tested in a +visual-location request i.e. :kind, :attended, :value, :color, :size, :screen-x, :screen-y, :distance, and :nearest. When the screen is processed by the model (pm-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 pm-proc-display gets called, if the visual-location buffer is empty a +visual-location request is automatically executed using the slot tests set with pm-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 so 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 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 mean deviation between the data sets.

Summary

The current AGI was designed to support the ACT-R 5.0 tutorial, and as such 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.