SCHOOL OF INFORMATION SCIENCES AND TECHNOLOGY
THE PENNSYLVANIA STATE UNIVERSITY

The Effects of Interface Design on Telephone Dialing Performance

Andrew R. Freed

Technical Report No. ACS 2003-1

arf132@psu.edu

6 May 2003

Phone +1 (814) 865-4455   Fax +1 (814) 865-6426

School of IST, Thomas Building, University Park, PA 16802

The effects of interface design on telephone dialing performance

Andrew R. Freed

arf132@psu.edu
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802

6 May 2003

Abstract

Interface design has an important impact on user performance. While some features of telephone interfaces have become relatively standardized (such as the layout of the keypad), there are other design features that can affect the way users interact with these interfaces. These features include the overall size of the interface and the size and labels of buttons.

We look at several prototype telephone interfaces with a variety of tools for automatic testing. Several analyses are presented. The first analysis uses Fitts' Law to determine the minimum possible usage time. Fitts' Law provided a quick analysis of the telephones but did not consider any factors other than target sizes and distances.

The second analysis uses a cognitive model to make predictions about other factors such as visual search. The model used an organized search strategy that produced similar behavior to the experiment subjects but failed for a few extreme interfaces.   Although the model dialing time predictions are too high, we propose that this model is worth paying attention to because the dialing time orderings among phones and number of fixations closely follows the observed data.

The third analysis uses a human experiment to test the predictions with reaction times and eye-tracking data. The experiment subjects were able to easily cope with these differences and retain an organized strategy, but the model was forced to revert to a random search strategy. The users often made errors while using interfaces but the cognitive model did not make the same types of errors.

Our approach should be generalizable to other types of interfaces, including VCRs, software applications, and webpages. Our analysis techniques can eventually remove the need for expensive and time-consuming user testing, which may be replaced with analytical and cognitive model analysis.

Table of Contents

Acknowledgements

To Rob St. Amant for answering many Segman questions.

To Mike Byrne for answering many RPM questions.

To Isaac Councill for helping me with many general questions along the way.

To Alex Wood for helping me learn how to use the eye-tracker.

To Anthony Hornof for providing several useful references.

To David Rosenbaum for advising me on experimental design.

To Dan Bothel for helping to fix a devastating bug in my ACT-R model.

To the experiment subjects for their willingness to help.

To two anonymous reviewers for their helpful comments.

To Dr. Ritter for his continual advice and guidance.

To Elise, Mom, and Dad for telling me I could do it.

1.0 Introduction: Towards automatic interface evaluation

Human-computer interfaces are the means users accomplish tasks within a computer system. User actions are affected by the interface programmer and sometimes in non-obvious ways. Therefore it is critical that interfaces allow users to quickly and easily accomplish their goals. In order to create better interfaces we must first be able to distinguish a “good” interface from a “bad” one. Just as SPICE (Perry, 1998) was created to automatically evaluate electronic circuits, this report was created to automatically evaluate telephone interfaces as an example of this approach.

Evaluating interfaces is both difficult and time-consuming. The most difficult and expensive part of evaluation is gathering suitable users to help test the interface. Ideally, some automated tool would be able to provide the same feedback as human users, thus freeing up time and resources from user testing. We will show that while several automated tools exist, most are incomplete and do not consider enough factors. We also present several analyses of user interfaces and compare the results.

This paper seeks to understand how telephone dialing is affected by interface design. We will show performance differences among cognitive models and people during telephone dialing and analyze these differences in detail. We will discuss telephone design considerations, perform an eye-tracking and reaction time experiment, and develop and test both cognitive and analytic models.

Testing interfaces with actual users does not require a large amount of preparation time. Instead difficulties may arise in scheduling testing times. If an interface is developed, it may require ten users to start to do a thorough test of the interface. It may take a week or more to bring in the users for testing. What happens if problems are found and the interface is changed? Any significant interface change can not be properly tested without bringing in ten more users. Because the development cycle may involve several design/build/test/fix cycles, this method can become expensive and can take a lot of time on the part of the developers and the testers.

Now consider a situation where an automatic model is created to test interfaces. The initial cost will likely be greater because the model takes time to construct. However, the second cycle of design/build/test/fix will take less time, because the model is already constructed, and is always ready to perform testing – there are no scheduling problems with cognitive models. Each of the consequent cycles will take less time with this automated model than with actual users.

Cognitive Model Interface Evaluation (CMIE) tools have been created by Ritter, Van Rooy, and St. Amant (2002). These tools work within a Cognitive Model Interface Management System (CMIMS) as proposed by Ritter et al. (2000). This combination allows cognitive models to test the same interfaces as actual human users. Unlike many previous systems, the interface may be encoded in any format and does not have to be encoded in a special format (i.e. Common Lisp). This system has already been used to evaluate the user interface to a simple driving game.

Using automated models is ideal in terms of saving time. However, these models are only useful if they make the same predictions that the users would have made. We thus present a study of testing telephone interfaces. We use two automatic models and also human users. We focus on telephones not because they are unusually interesting, but we use them as a metaphor for design in general. Others have used simple interfaces in a similar way, including Ritter et al. (2002) and Ritter and Young (2001). This work is also partly motivated by eye-tracking studies done by Byrne (2001; 1999) and Hornof (1997) on the task of menu selection.

Automatic models have been used successfully in the past to evaluate computer interfaces. Project Ernestine (Gray, John, & Atwood, 1992, 1993) was created with the purpose of comparing interfaces to be used by toll and assistance operators (TAO). A Goals, Operators, Methods, and Selection Rules (GOMS) model was created from observations on how TAOs used the old interface. Based on this knowledge, a GOMS model was also created for the new interface. The GOMS models predicted that the new interface was on average 4% slower and these predictions were backed up by an experiment performed after the models were created. Because of the validated predictions from the GOMS model, the new interface was not adopted. The new interface would have cost the company 2.4 million dollars per year due to lower operator performance, plus the cost of implementing the new interface. In this situation, an automatic model was both financially and scientifically useful.

1.1 One hundred telephones

We had one hundred telephones written in Tcl/Tk as the result of student projects for a user interface design course. These telephones varied widely in functionality, complexity, ease of use, size of buttons, and other metrics. Some telephones used colors, images, and different fonts while others maintained a more uniform look-and-feel across the entire phone.

We selected ten phones to analyze in detail. The phones were chosen so that they would have a variety of interface and button sizes and shapes, colors, numbers of extra features (in addition to the keypad), and levels of feedback. The ten phones we focused on are summarized in Table 1, and an example is displayed in Figure 1. All ten interfaces are displayed in Appendix A.

These telephones were all generated from the same initial piece of code. They can be viewed as being alternate designs from an interface design group, or as different end products from several design/build/test/fix cycles. They were also all designed around the same set of general requirements. Although the selected interfaces are telephones, we reiterate that this type of analysis can be done with any kind of interface.

1.2 Ten tasks

The purpose of any interface is to help the user accomplish a certain set of tasks. Certainly, any good interface will allow users to quickly and easily complete the task they set out to do. Before any interface test or analysis can be prepared, a task analysis must be done to determine what the user will want to accomplish while using the interface. Task analysis can be applied to a variety of situations (Schraagen, Chipman, & Shalin, 2000; Vicente, 1998), including university website design (Ritter, Freed, & Haskett, 2002). Task analyses are a useful way to summarize user behaviors and desires. Table 2 presents a list of common telephone tasks from Ritter (2000) generated by a task analysis.

Table 1. Description of telephones tested.

Phone #	Comments
1	Large numbers, simple phone
2	Like phone 1 but white text on blue background
3	Some buttons have different sizes
4	Upside down keypad
5	Keypad does not stand out
6	Small, minimal phone
7	Shaped like handheld phone
8	Small and tight keypad
9	Largest phone, includes ABC s on buttons
10	Digits have wide horizontal spacing

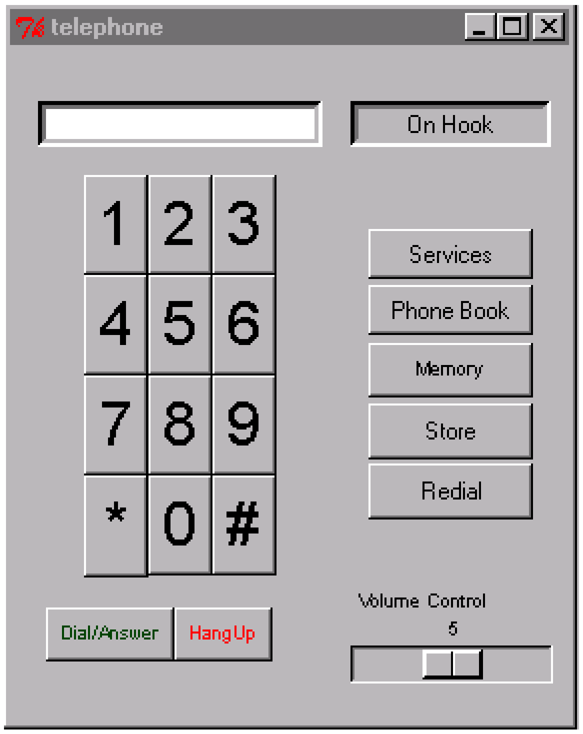


Figure 1. Example phone interface.

Table 2. The top ten telephone tasks from Ritter (2000).

1. Call home
2. Call work
3. Redial last number
4. Call directory inquiries
5. Call mother
6. Conference call work and home
7. Conference call work (flash) then home
8. Forward call to another number
9. Forward call (flash) to another number
10. Hang up

An interface should easily support the most commonly performed tasks. Just as a university department website should support common needs by listing admissions and contact information, a telephone interface should support common needs such as dialing, redial, conference call and hang up. These tasks should be considered in the interface design stage to assure that the interface will be useful to users by allowing them to complete their common tasks. If there are other common tasks, they should also be considered. We present Table 2 for illustration only but we feel that it should be nearly complete.

Some of the features from Table 2 have shortcuts. For instance, on certain telephones, calling a frequently used number can also be done via speed dialing, and redialing can be accomplished via a redial button or the sequence "##". Some features such as conference calling and forwarding may not be supported by all telephones, and even if they are supported they may be difficult to use. Nielsen argues that these features are difficult to use because of poor interface design (1997).

1.3 Problems in telephone interfaces

Nielsen (1997) argues that the simplest aspects of dialing a telephone are more difficult than expected. He claims that of the sequence "pick up the handset, punch in the number, get connected" only picking up the handset is simple. Punching in an entire telephone number is difficult due to remembering a sequence of ten to twelve digits and the fact that with most telephones one incorrect button-press invalidates the dialing sequence. Also, Salvucci has already done a comparison of dialing techniques that included speed dialing (2001). We therefore do not consider the difficulties of remembering a long telephone number or of making simple mistakes while dialing. We instead concentrate on how the interface is used when mistakes are not made while noting when errors occur.

Savage (1995) conducted a telephone usability study and noted that participants found conferencing to be "complicated" and difficult to use. With usability testing, Savage was able to transform an involved conferencing process (Table 3) into a simpler process (Table 4). The interface being designed was in the form of a tabbed notepad, where each tab on the notepad opened a menu of possible telephone features.

Table 3. Initial conferencing procedure, from Savage (1995).

Talking to Rich on Extension 46

Decide to add Linda by using Conference feature

Select Phone Features and then Conference

Now on Conference page of Phone Features

Rich is on hold, appears as First person on the line in the conference room

Use pen to tap on Add someone to an already existing call tapbox

Notebook page turns, presents options Dial from directory or dialpad

Choose option and dial

Tap on flashing Connect all parties tapbox

Conference call now includes Rich and Linda

Table 4. Revised conferencing procedure, from Savage (1995).

Talking to Rich on Extension 46

Decide to add Linda by using Conference feature

Select Phone Features and then Conference

Now on Conference page of Phone Features

Options Dial from directory or Dial from dialpad presented

Choose option and dial

Tap on flashing Connect all parties tapbox

Conference call now includes Rich and Linda

This improvement reduces the number of extra informational screens from two to one and reduces the number of active steps (minus dialing) from five to four. The new interface is simpler, faster, and more intuitive. This design can still be improved. Because phone conferencing is a popular task, an interface should not require two menu selections to get to the conference feature (Phone Features->Conference). Eliminating the “Phone Features” step reduces the number of active steps from four to three. By not presenting the user with a choice of dialing options and by not requiring an extra step to connect the conference parties the number of active steps can be reduced to one. This improvement (shown in Table 5) suggests that a GOMS-like analysis can quickly suggest design changes. Cognitive models can be used to provide quantitative predictions of improvement as well as suggest further changes. As was evident in Project Ernestine, this type of analysis can save significant time and money that would be lost due to poor interface design.

Table 5. Suggested conferencing procedure.

Talking to Rich on Extension 46

Decide to add Linda by using Conference feature

Press Conference button

Dial Linda s number

Conference call now includes Rich and Linda

2.0 Methods of evaluation

Many previous works have been written about automatic interface evaluation, including Balbo (1995), Byrne et al. (1994), Kishi (1993), Lecerof and Paterno (1998), Ritter (2000), Ritter and Young (2001), and Ritter et al. (2001; 2002). We first present a brief analysis of some existing tools from several different classes: purely analytic models, models driven by cognitive theory, and also simple user testing. We use this analysis to justify our choice of evaluation tools for looking at telephone interfaces. We then apply these different tools to telephone interfaces and present our results.

2.1 Possible tools

Several automated tools already exist to evaluate interfaces. Ivory's (2001) tools evaluate websites by searching for a pre-determined implementation of features. Apex (M. A. Freed, 1998) from NASA Ames models human performance in complex tasks. Glean from Kieras and colleagues (1995) does a GOMS analysis of tasks. iGEN (Emmerson, 2000) uses embedded cognitive models to simulate behavior.

Each of these automated tools has shortcomings. Ivory's tools can evaluate interfaces and confirm whether certain tasks are supported by an interface. However they are driven by algorithms created to run quickly and efficiently – not to simulate human behavior. Apex and iGEN are very similar to one another, and are generally used to simulate more complex tasks than telephone dialing. However, as models of expert behavior, neither model has visual search or the ability to learn. Glean could be used to evaluate interfaces, but only after they have been recreated in a special format (Lisp). We require an automated tool with a more powerful predictive theory.

Our tool must be able to use any generic interfaces and it must make the same predictions about these interfaces that would be made by actual users. Therefore, this tool must simulate several aspects of human behavior. The tool must have cognitive, perceptual, and motor capabilities, it must have visual search and learning, and it must be able to make mistakes. The tool must also be subject to the same physical and mental constraints as actual users. The tool we have just described can be realized within a cognitive architecture.

2.2 Cognitive architectures

An ideal way to analyze generic telephone interfaces would be with a cognitive architecture that uses simulated eyes and hands. Cognitive architectures have been created to model human performance. A cognitive architecture is built on a unified theory of human cognition. These architectures simulate cognitive elements such as declarative and working memory constraints, procedural memory, and internal timing mechanisms as well as additional perceptual and motor capabilities. Sample architectures include ACT-R/PM (Anderson, 1983, 1990; Anderson & Lebiere, 1998; Byrne, 2001), Soar (Newell, 1990), the Model Human Processor (Card, Moran, & Newell, 1983), CPM-GOMS (John & Kieras, 1996), and EPIC (Kieras & Meyer, 1997; Meyer & Kieras, 1997).

A cognitive model is an attempt to simulate human behavior on a given task. Cognitive models run within the cognitive architectures they were created in. A model must be supplied with information about its environment and its tasks, along with other knowledge, that the model uses to create a goal structure that it must attempt to achieve. Hundreds of cognitive models have been developed by the cognitive psychology community to simulate various tasks, i.e. the Tower of Hanoi (Andersen & Douglass, 2001), menu selection (Byrne et al., 1999), driving (Salvucci, 2001), emotions (Ritter, Belavkin, & Elliman, 1999), arithmetic (Lebiere & Anderson, 1998), and many more.

Previous works have suggested the use of cognitive models as surrogate users as an inexpensive means of evaluating user interfaces (Byrne et al., 1994; M. Freed & Remington, 2000; Ritter et al., 2000; Ritter & Young, 2001; St. Amant, 2000). Cognitive models are also much more available for testing than the typical busy human and thus these models can make interface testing quicker and more efficient.

We need to use a cognitive architecture with the ability to do visual search, learn, and interact with a generic interface through simulated eyes and hands. EPIC is a cognitive architecture with learning, visual search, and simulated eyes and hands. However, EPIC can only interact with interfaces designed in Common Lisp. Soar is also a useful cognitive architecture, but requires interfaces to be either implemented in Soar or to have a socket connection. For our analysis, we will use ACT-R/PM.

2.3 ACT-R

ACT-R/PM is a model of human cognition. It consists of two parts. First ACT-R provides the cognitive elements of the brain, and secondly PM (Perceptual-Motor) provides the use of simulated eyes, ears, and hands. Since version 5.0 of the software, ACT-R/PM goes by the abbreviated name ACT-R, which I will use for the remainder of the report. Like some other cognitive architectures, ACT-R is a production-rule based system. ACT-R uses a fifty millisecond production cycle, and unlike some architectures fires only one matching production rule per cycle (Anderson, 1983; Anderson & Lebiere, 1998; Byrne, 1997, 1999). ACT-R is also limited in that it can only act upon interfaces created with Macintosh Common Lisp including special extensions. Fortunately, a tool was created to help ACT-R overcome this problem.

2.4 Generic Eyes and Hands

Evaluating a generic interface requires an architecture that is not dependent on the interface being encoded in a special format. Ideally the architecture would be able to "look" at a screen and determine the interface pieces, and then use simulated hands to interact with it. Segman is a package that can do both of these things. Segman parses a screen bitmap and is able to locate text, buttons, windows, and other features, although with some limitations.

Segman allows ACT-R's simulated eyes and hands to interact with a generic user interface. After Segman interprets what is on the computer screen, ACT-R can use any Windows interface the same as an actual user can. When fully realized, this will allow quick evaluation of interfaces by a cognitive model (Ritter et al., 2000; Ritter et al., in press; Ritter, Van Rooy et al., 2002; Ritter & Young, 2001; St. Amant, 2000; St. Amant & Riedl, 2001). Thus, the combination of Segman and ACT-R will allow us to analyze any generic interface in its native format, without us having to recreate it in a special format. Although our model is not the first to dial telephones, the model created for this report is the first that can dial a generic telephone interface. The previous model was able to perform four of the tasks in Table 2, so in the spirit of Newell (1990), our model "raises the bar" by being able to perform six of the tasks. Modeling work done for this report used ACT-R version 5.0, with RPM version 2.17b, and Segman v3.1.

2.5 Focus of analysis

We look at three different ways to analyze the telephones we are examining: analytic models (Fitts' Law), a cognitive model, and human users. We focus on reaction times (time needed to dial a given telephone number on a given interface), while also making observations where appropriate and possible about visual search (number and location of fixations) and errors (when the wrong button is clicked or when the user gives up). Fitts' Law will show us the time needed to dial a telephone if the only factor being considered is pressing the correct buttons. The cognitive model can make additional predictions, such as the number of visual fixations required to find the correct buttons, and errors. We include an analysis of human users to evaluate the predictions made by the model.

2.5.1 Use analytic models (Fitts' Law)

The simplest theoretical analysis for an interface is to use a purely analytical model. Fitts' Law (Fitts, 1954) can be used to provide such a model. It relates target size and distance to mouse movement time that we record as user reaction time.

An application of the Law would be summing over the expected movement and execution times for each button press in phone dialing task to establish a minimum possible dialing time. While Fitts' Law analysis does consider time due to any other factors (visual search, thinking, etc.), it is still useful as a baseline measure.

This analysis can easily be done using just a spreadsheet with pixel-level information (location and sizes of the buttons) as input. A purely analytical model like Fitts' Law is the fastest type of analysis to perform, but at the cost of evaluating fewer factors and thus making less accurate predictions. The Fitts' Law analysis is presented in Section 4.

2.5.2 Use cognitive models

A cognitive model allows us to explore other possible factors of telephone interface usage time. Fitts' Law already governs the motor capabilities of the cognitive architecture. However, the model is also governed by cognitive, perceptual and motor constraints that should create predictions that are much closer to actual human performance than the predictions offered by Fitts' Law. In particular, the model should at least be able to perform and make predictions about visual search.

Several studies have done similar analyses to determine visual search strategies of computer users from eye-tracking analysis (Byrne, 2001; Ehret, 2002; Fleetwood & Byrne, 2002, 2003; Hornof & Halverson, 2002, 2003a, 2003b). It has also been used to examine tasks outside the realm of computers (Findlay, 1982; Zelinsky & Sheinberg, 1997). We develop an initial strategy of visual search within the cognitive model and compare the reaction time and number of fixations to data compiled from human users. The cognitive model analysis is presented in Section 5.

2.5.3 Human data for comparison

Human users are tested so that we may directly measure the effect of interface design on users. Note that the human data makes no theoretical predictions – it is just data! However, testing with users is probably the most common way of testing interfaces, and is the most convenient way to validate our other predictions. Because the human data makes no theoretical predictions, we include the previous analyses that do. The human data analysis is presented in Section 6.

3.0 The phones and tasks as an example

Each participant, whether ACT-R model or human user, performed ten trials with the telephones. (The analytical model worked “offline” and did not actually dial the telephones, but predicted dialing times.) Every trial consisted of dialing a randomly presented number from Table 6 on each of the ten telephones from Table 1 that were also presented in random order. Both random selections were done without replacement. Every participant dialed each of the ten numbers on each of the ten telephones. We wished to create a common environment for testing the interfaces so that both the ACT-R model and human users could interact with this environment with no modifications. In order to eliminate visual distractions, this environment should present a display devoid of stimuli other than the telephones. Because eye-tracking can be physically demanding of the user and possibly intrusive, subjects should be able to pause between trials and even between telephones.

Table 6. Telephone numbers dialed by participants.

814-866-5000	215-654-5785
123-654-7890	814-234-9657
814-863-5000	740-611-9273
412-268-3000	101-010-1010
606-193-3012	103-273-1029

The experiment environment consisted of a full-screen application with one button centered on the screen, labeled "Go". The entire screen was covered with a neutral gray, except for the button. The “Go” button provided a convenient eye-tracking fixation location because the users will fixate on the button to click it. The button also served as the way for a user to control how quickly new phones were presented. When the “Go” button was clicked, a timer started and a telephone was displayed centered on the screen. Button clicks on each telephone were timestamped and logged to a data file, and the final dialing time was also logged. See Figure 2 for the initial experimental software configuration, and Figure 3 for the in-trial configuration.

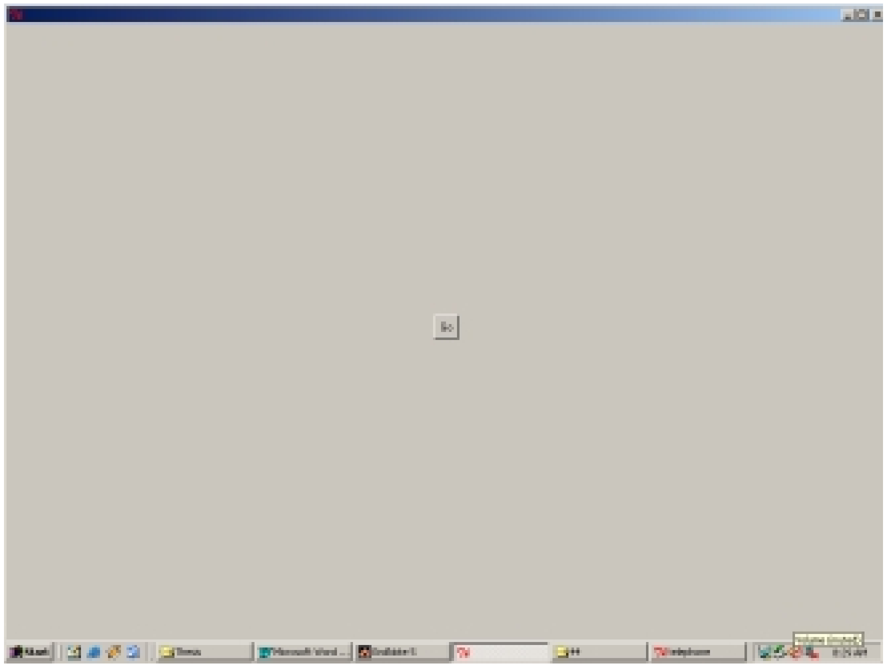


Figure 2. Experimental software (start screen).

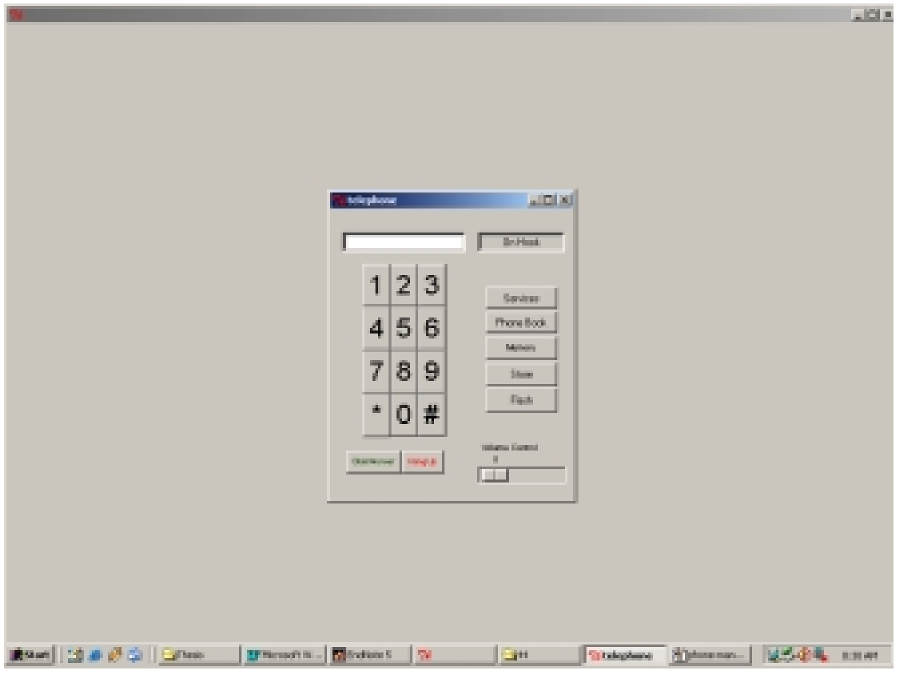


Figure 3. Experiment software with telephone.

In addition to dialing the telephones, the experiment subjects and ACT-R model performed a few additional tasks from Table 2. These tasks were call forwarding, redialing, and conference calling. Trials with these tasks ran similarly to the dialing tasks – all trials were timed and all button clicks were logged. A trial was considered complete when either the desired button was clicked (i.e. “Redial”) or when the phone was closed. In the case of the ACT-R model, the trial was considered over when the desired goal was set to a “completed” state.

While other studies have used Lisp interfaces usable by both humans and cognitive models, our experiment marks the first time that experiment participants have been able to interact with the same unmodified generic interface environment. The designers of the interfaces had no knowledge that the interfaces would eventually be tested using analytical and cognitive models. Several cognitive models have interacted with interfaces written in a format native to the cognitive architecture – such as ACT-R and EPIC models interacting with Lisp interfaces (Byrne, 2001; Byrne et al., 1999; Hornof & Kieras, 1997). Here our choice of interface format is free form. We chose to use Tcl/Tk, but this environment also could have been encoded in other formats including Java, Visual C++, Visual Basic, Lisp, HTML, and several others.

One modification was made to our experimental design. Because the screen-parsing algorithms used by Segman are time-intensive, we would parse the screen once for each phone and load this into the ACT-R model. The model ran fifty consecutive runs on each of the ten phones, for each of the ten numbers. The model was reset after each telephone number was dialed.

Two of the human participants were selected to perform the experiment while an eye-tracking machine recorded visual fixation data. The eye-tracker used was an ISCAN ETL-500 with head-tracker. Head-mounted optics recorded Point of Regard (POR) data at 60 Hz. We recorded the screen coordinates of the user eye movements and translated them into fixations using a standard algorithm available freely on the Internet (www.eyegaze.com/doc/fixfunc.c.html).

We now present the results of the analyses in increasing order of ease of evaluation.

4.0 Fitts' Law predictions

Some experts consider Fitts' Law (Fitts, 1954) to be an important part of interface design. The law states that mouse movement time is related to both the distance to the target and the width of the target along the axis of movement. The law is stated as follows:

Equation 1. One interpretation of Fitts Law (Fitts, 1954).

$MT = a + b * LOG_2(2 * D / W)$

where

a is a reaction time constant

b is a movement time multiplier

D is the distance to the target (in pixels)

W is the width of the target (in pixels) along the axis of movement

MT is the total predicted movement time in milliseconds

Experts predict that by analyzing an interface according to Fitts' Law, a general understanding of movement time can be gained (MacKenzie, 1991). If movement time was the only factor in interface design and usage, Fitts' Law could produce a total ranking of interfaces, where interfaces with smaller usage times would be deemed easier and therefore best. However, other factors are involved with interface usage. Time spent on visual search is one such factor that is also expected to be a significant contributor to usage time.

Fitts' Law is also presented as

Equation 2. Another realization of Fitts Law (Card et al., 1983).

$MT = max(t_m, k * LOG_2[0.5 + D/W])$

where k is a movement time multiplier

A variety of values can be used for k, however k = 100 ms is a value commonly used, and is the value used in ACT-R and EPIC. t_m is the minimum possible movement time, also frequently set to 100 ms. Our Fitts' Law analysis used Equation 1 with

a = 100 ms and b = 150 ms.

These equations deal only with movement times. ACT-R predicts a mouse click time of 0.3 s. Dialing the ten-digit telephone numbers used here requires ten mouse clicks for a total of three seconds. Therefore three seconds will be added to the Fitts' Law predicted movement times to generate a base dialing time.

4.1 Numerical analysis

We used standard screen-capturing software (www.snagit.com) to collect pixel-level information about the telephones. In particular we collected the sizes and locations of each of the keypad digits. Next, we created a C++ program that generated Fitts' Law movement times based on the required mouse movements in dialing a given telephone number. Each telephone number has ten transitions: the first transition is from the starting position to the first digit, and each of the nine subsequent transitions is from a previous digit to the next digit.

Using this program, we generated dialing times for each of the telephones. We then ranked the telephones according to dialing time. The results are listed in Table 7. The predictions were averaged over the results of dialing each of our ten numbers, using Fitts' Law Equation 1 with coefficients of a = 100 ms and b = 150 ms.

Table 7. Summary of Fitts Law predictions on our sample telephone numbers.

Phone	1	2	3	4	5	6	7	8	9	10
Time (s)	5.77	5.85	5.70	6.12	5.60	6.40	6.41	5.80	6.98	6.37
Rank	3	5	2	6	1	8	9	4	10	7

According to the analysis, several phones have nearly identical dialing times. Phone 9 stands out as the worst violator of Fitts' Law. In this phone the keypad buttons are moderately sized but the wide button spacing causes longer mouse movements. The analysis also suggests that half of the phones (1, 2, 3, 5, and 8) may be easier to dial than the remaining phones. In these five phones, the keypad buttons are adjacent to one another.

This analysis shows that we generally selected telephone numbers that took longer to dial than the "average" telephone number. We chose such numbers so that we would be able to test numbers with a wide range in Fitts' dialing times. Some numbers were chosen to have unusually large Fitts' movements (101-010-1010 and 103-273-1029) and some were chosen to have small Fitts' movements (814-866-5000 and 123-654-7890). Also 35% of the possible ten-digit phone numbers do not contain any "0"s, and this helps lower their Fitts' time because the average distance from the zero is greater than for any other digit. Eight of our selected numbers contained at least one zero.

Table 8 shows Fitts' Law dialing times for each phone averaged over all 10^{10} possible ten digit telephone numbers. This is included to justify our choice of sample telephone numbers. Dialing times can range from just over three seconds to slightly under seven seconds. The ordering of the telephones based on dialing times can be affected by the sample telephone numbers chosen to be dialed. Figure 4 compares the predicted dialing times for the ten sample numbers against all possible telephones. Because our sample telephone numbers have approximately the same dial time ordering as the general population, we consider these telephone numbers to constitute a good sample. The analysis took approximately thirty-five minutes per telephone on a 500MHz Pentium III processor.

This analysis shows that we generally selected telephone numbers that took longer to dial than the "average" telephone number. We chose such numbers so that we would be able to test numbers with a wide range in Fitts' dialing times. Some numbers were chosen to have unusually large Fitts' movements (101-010-1010 and 103-273-1029) and some were chosen to have small Fitts' movements (814-866-5000 and 123-654-7890). Also 35% of the possible ten-digit phone numbers do not contain any "0"s, and this helps lower their Fitts' time because the average distance from the zero is greater than for any other digit. Eight of our selected numbers contained at least one zero.

Table 8. Summary of Fitts Law predictions on all possible telephone numbers.

Phone	1	2	3	4	5	6	7	8	9	10
Time (s)	5.25	5.34	5.31	5.69	5.32	6.00	6.05	5.33	6.35	5.83
Rank	1	5	2	6	3	8	9	4	10	7

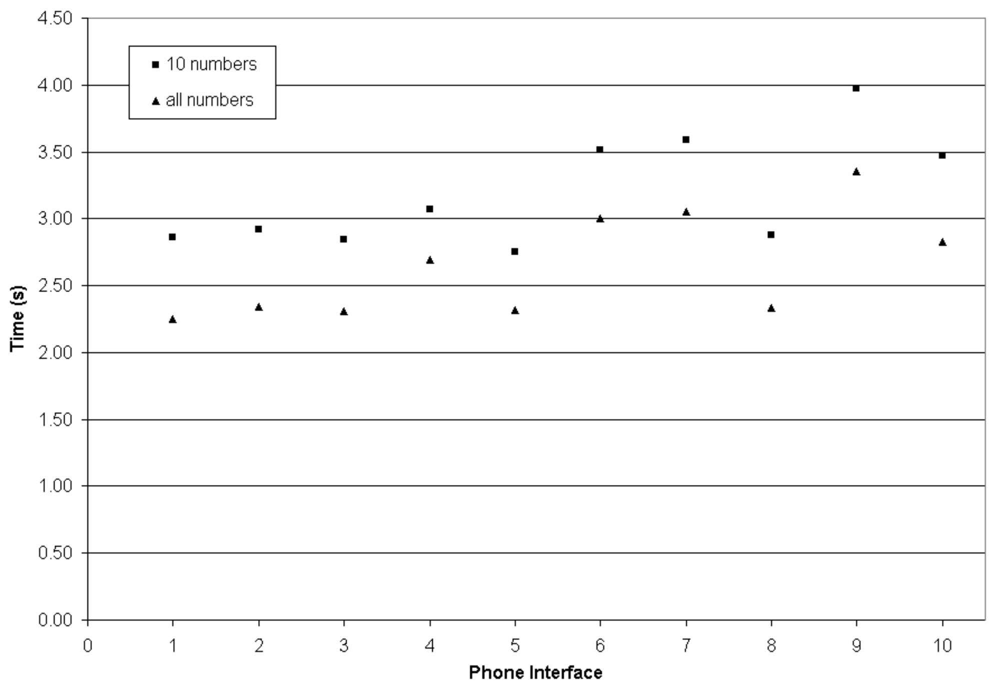


Figure 4. Fitts Law predicted dial times for sample numbers and all numbers. $R^2 = 0.96$.

4.2 Simple conclusions

According to Fitts' Law alone, the best phones are 5, 3, 1, and 8, in that decreasing order. These phones have no space between keypad buttons and thus have small target distances and high effective target widths. In these phones, the target distance to width ratio in Equations 1 and 2 achieves the minimal value of 1. Taking no other factors into account, Fitts' Law predicts that it will take the least time to click the digits of a telephone number on these telephones.

Because we have chosen a quality sample of telephone numbers we can feel confident that these results generalize to any possible telephone number. Figure 4 shows that the telephones that performed best on the sample population also performed best on the general population.

4.3 Problems with this analysis

While this analysis was the fastest of the analyses we did, it also leaves many questions unanswered. Fitts' Law is only affected by a few factors of an interface (target sizes and spacing). It does not predict the effect of interface color, number of possible visual targets, font size, layout, overall interface size or shape, nor does it make any prediction about the number or location of eye fixations, or the difficulty of visual search. While clearly useful in making predictions about dialing times, a more powerful analysis is needed, and thus we turn to cognitive models.

5.0 ACT-R model use of telephones

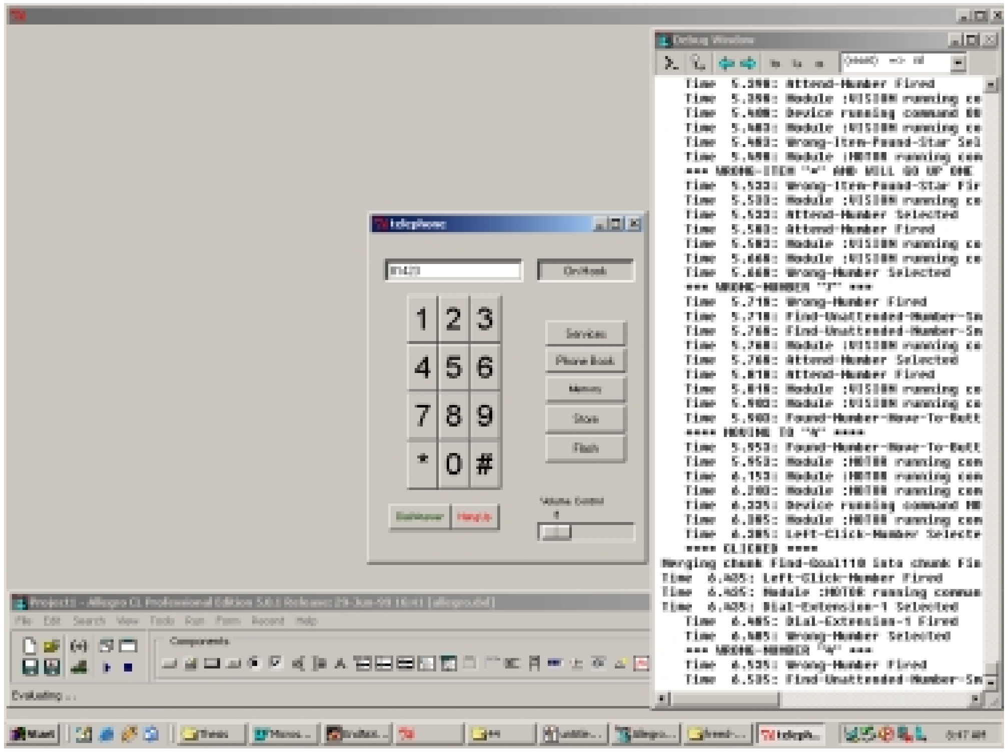
A cognitive model built within a cognitive architecture can make more predictions than a strictly analytic model. Such a model should be able to make some of the predictions that we noted Fitts' Law was unable to do. We developed a cognitive model, using the ACT-R architecture, augmented with Segman that enabled the model to use a generic Windows interface. The model contained seventy-eight productions and forty-two declarative memory elements. The productions included how to find a number, how to move visual attention from the present visual location to a new one, and how to dial a number. The other memory elements were the phone dialing goal and the ten telephone numbers.

After running a pilot eye-tracking experiment, it was discovered that once the subjects fixated on a digit within the telephone keypad, they never fixated off of the keypad until the dialing task was complete. This strict visual "barrier" was then implemented into the model. Without this enhancement the model would dial the telephones at an unacceptably slow speed (about twice as slow as the pilot users) due to frequent overshooting saccades.

5.1 Description of model

The model has a group of productions designed to implement visual search. These productions provide a clue about what direction to move the eyes from a present visual fixation to the desired target location. For instance, if the model is fixating on the "1" key and is searching for the "2" key, a production can fire to tell it to "look to the right." If the model is on the "1" key and looking for the "8", the model only knows to "look down and to the right." Without these productions, the model can only perform random search. See Appendix B for the complete model code.

The model was designed to use its visual and motor subsystems in parallel. While the motor component of the model moved the mouse and clicked on a button, the visual search component searched the interface for the next button to dial. This pipelined behavior was also seen in the pilot users. The model started by randomly searching for the keypad, and then systematically searching within the keypad for each of the buttons it needed to dial. The model never "looked ahead" more than one digit at a time. While the model searched for the last component of the area code on the phone, it retrieved the exchange from memory, and when it searched for the last component of the exchange, it retrieved the extension. See Figure 5 for a sample trace of the model.



```
  Time  5.398: Attend-Number Fired
  Time  5.398: Module :VISION running co
  Time  5.408: Device running command [illegible]
  Time  5.483: Module :VISION running co
  Time  5.483: Wrong-Item-Found-Star Sel
  Time  5.498: Module :MOTOR running com
  *** WRONG-ITEM "*" AND WILL GO UP ONE
  Time  5.533: Wrong-Item-Found-Star Fir
  Time  5.533: Module :VISION running co
  Time  5.533: Attend-Number Selected
  Time  5.583: Attend-Number Fired
  Time  5.583: Module :VISION running co
  Time  5.668: Module :VISION running co
  Time  5.668: Wrong-Number Selected
  *** WRONG-NUMBER "7" ***
  Time  5.718: Wrong-Number Fired
  Time  5.718: Find-Unattended-Number-Sm
  Time  5.768: Find-Unattended-Number-Sm
  Time  5.768: Module :VISION running co
  Time  5.768: Attend-Number Selected
  Time  5.818: Attend-Number Fired
  Time  5.818: Module :VISION running co
  Time  5.903: Module :VISION running co
  Time  5.903: Found-Number-Move-To-Butt
  **** MOVING TO "4" ****
  Time  5.953: Found-Number-Move-To-Butt
  Time  5.953: Module :MOTOR running com
  Time  6.153: Module :MOTOR running com
  Time  6.203: Module :MOTOR running com
  Time  6.335: Device running command MO
  Time  6.385: Module :MOTOR running com
  Time  6.385: Left-Click-Number Selecte
  **** CLICKED ****
Merging chunk Find-Goal110 into chunk Fin
Time  6.435: Left-Click-Number Fired
Time  6.435: Module :MOTOR running comma
Time  6.435: Dial-Extension-1 Selected
  Time  6.485: Dial-Extension-1 Fired
  Time  6.485: Wrong-Number Selected
  *** WRONG-NUMBER "6" ***
  Time  6.535: Wrong-Number Fired
  Time  6.535: Find-Unattended-Number-Sm
```

Figure 5. Model in action, dialing 814-234-9657.

The figure displays some visual search done by the model while looking for the "4" after clicking the "3". From the "3" it must move down and left, however it moves too far, to the asterisk. From the asterisk it moves to the "7" and finally the target "4". When the next digit is retrieved (in this case the "9"), the model begins searching for the "9" with knowledge of where the "9" is in relation to the current position.

5.2 Visual search predictions

An ACT-R model was used so that we could consider factors other than button sizes and spacing. In particular we were interested in visual search strategies. We created two strategies for the model to use: a random search and a systematic search. The main productions from each class are outlined below. The random search strategy had one unique production: find an unattended target within the telephone interface. The systematic search had sixty productions that instructed the model where to direct its next saccade based on the target currently fixated on.

Find-random-target

IF the goal is to find a phone target

THEN find a visual object of type *text* that has not been attended lately

Systematic-search-from-target

IF a phone target *x* is in the visual buffer

AND the goal is to find a phone target *y*

AND *y* is in direction *z* from *x*

THEN find a visual object of type *text* in direction *z* from target *x* that is within the bounds of the keypad

In both strategies the model compares a newly fixated object to the desired target. If there is a match the model begins a mouse move and click phase. If not the model must decide how to proceed in its search. The following two productions instruct the model on how to proceed in the search process.

Wrong-item-found

IF a visual target *x* is in the visual buffer

AND *x* is not a keypad digit

THEN prepare to start a random search

Wrong-number-found

IF a visual target *x* is the visual buffer

AND *x* is a keypad digit

AND *x* is not the desired target *y*

THEN prepare to start a systematic search

These productions assume that no pertinent knowledge of the interface can be learned by fixating on a non-digit. Such visual objects are intermittently placed around the telephone interfaces and provide little information about the other visual targets. If a digit is fixated on it will provide clues as to where the other digits are. The exceptions to these rules are the pound and asterisk buttons that are assumed to be "digits" that are located to the left and right of the "0".

A measure of visual search would be both the number of and location of fixations. The model makes a few predictions (summarized across all telephone numbers in Table 9). The first prediction is that it is difficult to find the correct buttons on phones 4 and 9. The phone 4 difficulties are because of the upside-down keypad. For instance when the model finds the "8" and is looking for the "1", it will try to go up a fixed distance. But there are no digits within this fixed distance because the digits are all below the 8. The model does not know what to do in this situation, so it resorts to a random search strategy. It is not able to make the adjustment to an upside-down keypad. On phone 9, the "ABC" letters on the digit buttons distract the model. Because the model has no knowledge about what to do when it sees letters, the model resorts to a random search strategy. The model is able to revert back to a systematic search when it finds another digit. The other prediction made by the model is that the remaining phones require approximately the same numbers of fixations for dialing.

Table 9. Fixations made dialing each phone across all tested telephone numbers.

Phone	1	2	3	4	5	6	7	8	9	10
Fixations	16.3	14.4	17.5	61.4	16.9	15.0	15.9	16.8	56.5	16.2
Rank	5	1	8	10	7	2	3	6	9	4

Table 10 summarizes the fixation data across eight of the telephones. The telephones with extreme fixation data (phones 4 and 9) were not included in the analysis because they skewed the averages. The telephone numbers are presented in order of increased predicted Fitts' Law time. The model sometimes predicts that "easier" telephone numbers in terms of Fitts' Law will require fewer fixations than the telephone numbers deemed difficult, as seen in the first three dialed numbers. But the model also predicts that there are several exceptions to this rule, as seen in the next three phone numbers.

The model also makes a suspicious prediction, that the number "101-010-1010" will require the most fixations despite the fact that only two buttons need to be clicked. In this case, the model predicts a visual search that might be overly difficult, but might not be in practice. Additionally, the model predicts that numbers with repeated digits usually require fewer fixations than those without repeated digits.

Table 10. Fixations made per telephone number across eight tested telephones.

Number	814 866 5000	123 654 7890	814 863 5000	412 268 3000	606 193 3012	215 654 5785	814 234 9657	740 611 9273	101 010 1010	103 273 1029
Fixations	12.8	13.2	15.0	12.8	17.4	14.9	16.8	17.0	22.3	19.1

Grant (1962) proposes that statistical tests can be used to test extreme hypotheses. These tests cannot prove that a model is correct (you cannot prove the null hypothesis). Grant argues that if you want to show your model correlates with some data you collect a few trials of noisy data – that way any data should correlate! Instead he proposes that researchers use reliable correlations to support their claims. It is not as important to match the data points as it is to match the trends. The model we provide will be better at matching trends than individual data points.

We include several T-tests to show that the model finds certain characteristics to be statistically different, and some not statistically different, and that the experiment data shows similar differences and similarities. Because we include a high number of T-tests, there is a likelihood of several Type II errors.

We performed a statistical test (two-sided T-test, alpha=0.05, 98 degrees of freedom, rejection value was T=2.39) to show statistically significant differences in the number of fixations required to dial telephone numbers. Across all of the telephone numbers, there were no statistically significant differences in number of required fixations for phones other than 4 and 9. These telephones required a statistically significant different number of fixations for all telephone numbers except 740-611-9273, where only phone 4 could be shown to be statistically different. Both telephones required an abnormally high number of fixations due to design problems: an upside-down keypad in phone 4 and letters on the keypad in phone 9. In these cases, the model was not able to adopt new strategies. For phone 4, the model should have reversed its knowledge about which buttons were above and below each other. For phone 9, the model should have simply ignored the letters on the keypad digits, or looked immediately above the letters to fixate on the digit. See Figure 6 for an example eye scan path from the ACT-R model. The square dot represents the expected first fixation location (where the “Go” button was) and the blue crosses represent fixations. Thicker crosses represent longer fixations.

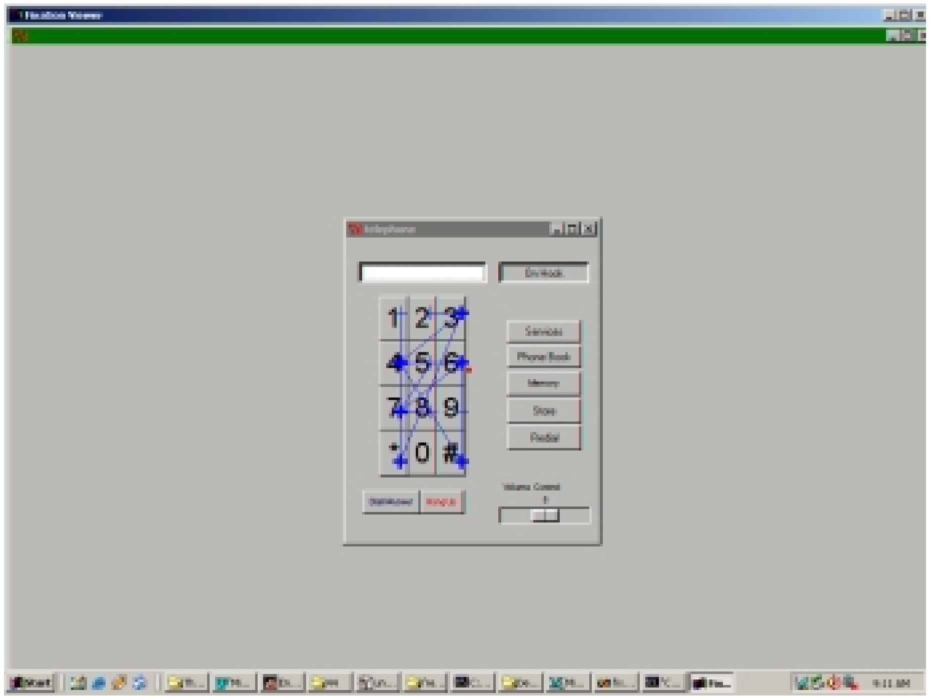


Figure 6. Model dialing 814-234-9657.

5.3 ACT-R makes different reaction time conclusions

The model was run for fifty trials on each phone, for each number. The reaction times for each phone were averaged out over the trials. The model makes a few interesting observations. First it predicts that phone 4 will be extremely difficult to use due to the extra fixations required, and these extra fixations translate into increased dialing time. This telephone has an upside-down keypad and requires a change of strategy to be dialed correctly.

The model also predicts phone 9 will be difficult to use. While it did not make any errors on this phone, it was distracted by the letters present on the telephone keypad and thus made many more fixations than was necessary. This phone also incurs extra Fitts' Law movement time due to the spacing of its buttons. Table 11 and Figure 7 show the dialing time summary of the model runs. Note the extreme dialing times predicted for phones 4 and 9.

Table 11. Summary of model reaction time data and ranks as predicted by the ACT-R model and Fitts Law.

Reaction time (s). $R^2 = 0.14$.

Phone	1	2	3	4	5	6	7	8	9	10
Model	7.84	7.44	8.25	18.43	8.05	7.83	8.17	8.05	16.80	8.05
Fitts	5.77	5.85	5.70	6.12	5.60	6.40	6.41	5.80	6.98	6.37

Ranks for ACT-R model and Fitts Law. $R^2 = 0.07$.

Model	3	1	8	10	4	2	7	5	9	6
Fitts	3	5	2	6	1	8	9	4	10	7

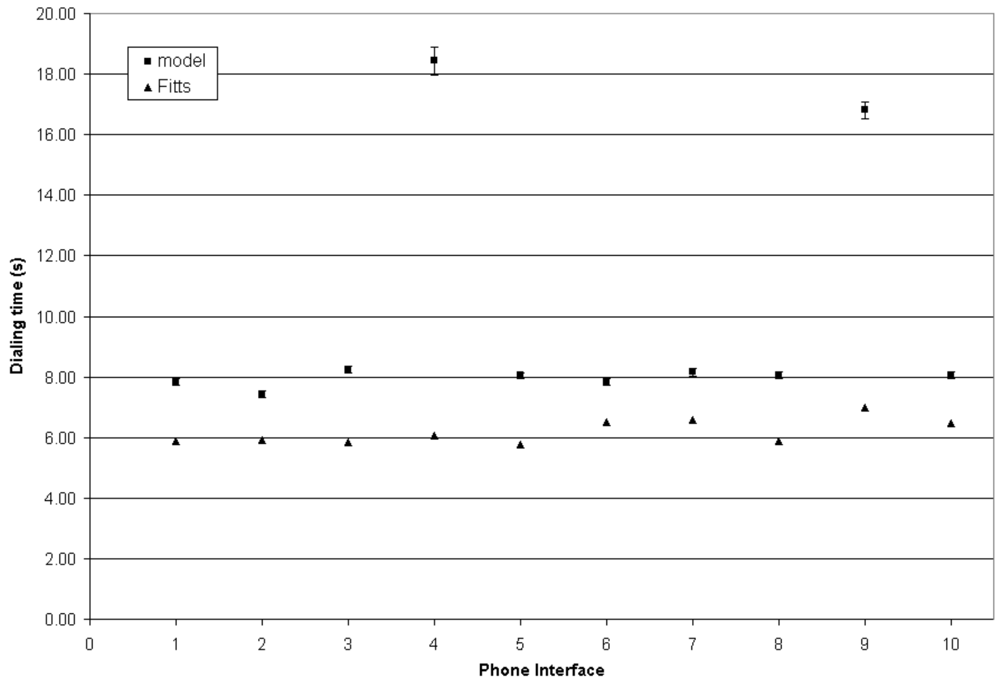


Figure 7. Model dialing time vs. Fitts Law predicted time.

The model did not make any errors dialing any of the telephones. Dialing times, as seen in Table 11 and Figure 7, remain fairly consistent across the telephones. This consistency is seen both on dialing individual telephone numbers and when averaged across all the telephone numbers. A T-test was used to determine statistically significant differences in dialing times per telephone number. The alpha value was 0.05, there were 98 degrees of freedom, and the rejection value was T=2.39. Table 12 shows the analysis of an easy number (412-268-3000), using a two-sided T-distribution with 95% confidence level (Figure 8 shows dialing times). This showed thirty-one of the forty-five possible pairs of interfaces could be said to have different mean dialing times for that telephone number. The remaining pairs differ in mean dialing times but cannot reject the null hypothesis with high confidence. On a more difficult number (101-010-1010), thirty-eight of the forty-five pairs can reject H_0, as seen in Table 13 (Figure 9 shows dialing times). The differences between telephone interfaces can change when the telephone number dialed changes. Therefore it is important to test telephones dialing several different numbers.

Table 12. T-test comparing mean dialing times for 412-268-3000 (p<.05). Black bars represent a rejection of H_0.

Phone	1	2	3	4	5	6	7	8	9	10
1	-	■	■	■			■	■	■	
2	■	-	■	■	■	■	■	■	■	■
3	■	■	-	■		■			■	
4	■	■	■	-	■	■	■	■	■	■
5		■		■	-	■			■	
6		■	■	■	■	-	■	■	■	
7	■	■		■		■	-		■	
8	■	■		■		■		-	■	
9	■	■	■	■	■	■	■	■	-	■
10		■		■					■	-

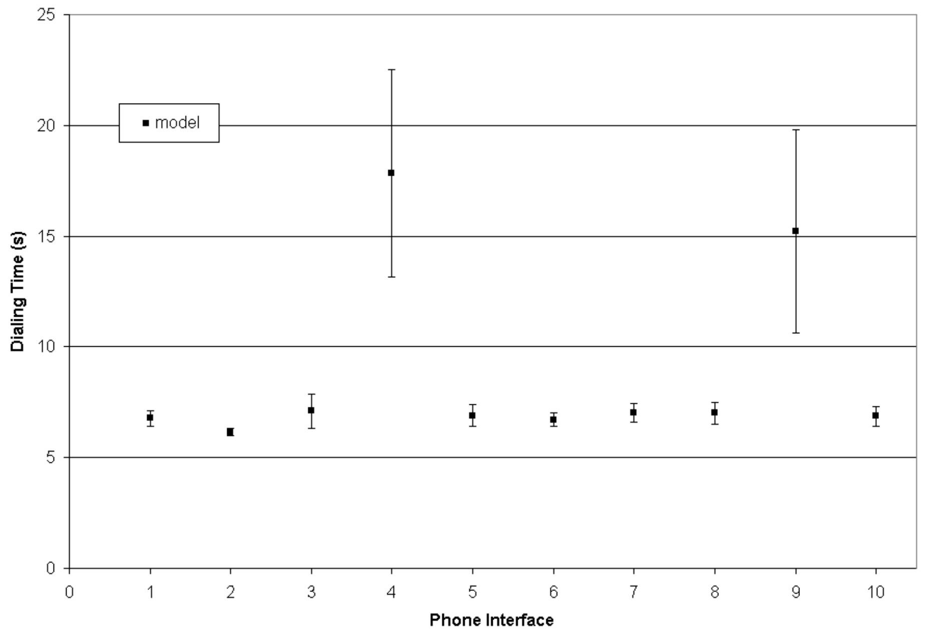


Figure 8. Model dialing time for 412-268-3000. Fifty trials per phone.

Table 13. T-test comparing mean dialing times for 101-010-1010 ($p<.05$). Black bars represent a rejection of H_0.

Phone	1	2	3	4	5	6	7	8	9	10
1	-	■	■	■			■	■	■	
2	■	-	■	■	■	■	■	■	■	■
3	■	■	-	■	■	■	■	■	■	■
4	■	■	■	-	■	■	■	■		■
5		■	■	■	-	■	■		■	
6		■	■	■	■	-	■	■	■	■
7	■	■	■	■	■	■	-	■	■	■
8	■	■	■	■		■	■	-	■	
9	■	■	■		■	■	■	■	-	■
10		■	■	■		■	■		■	-

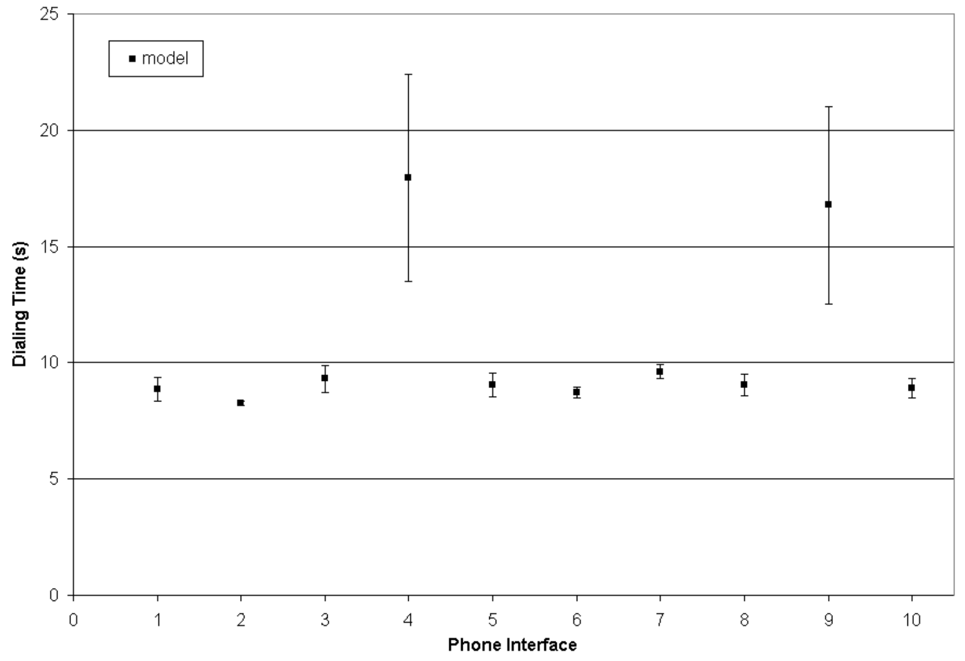


Figure 9. Model dialing time for 101-010-1010. Fifty trials per phone.

The model had more difficulty performing the tasks not strictly associated with dialing (i.e. call forwarding). While rules for the placement of numbers on the keypad are well known, the conventions for the placement of other buttons are not as well defined. Without having any other knowledge about the placement of these features within an interface the model must resort to a random-search strategy. See Table 14 for a summary of the model reaction times for call forwarding, redialing, and conference calling.

Table 14. Model reaction times for the other telephone tasks.

	Reaction time (s). - signifies an infinite search.									
Phone	1	2	3	4	5	6	7	8	9	10
Forward	-	8.13	10.70	-	6.23	-	-	6.37	-	9.50
Redial	-	7.16	10.09	5.68	-	-	-	7.38	-	11.92
Conference	-	7.04	9.36	-	6.77	-	-	-	-	-

When a button related to the current task was present on the telephone, the model would eventually find it with random search. The model did not succeed if abbreviations were used ("Conf." for conference), and did not search through menus, and did not attempt to search for buttons with similar meanings, and did not attempt clicking unrelated buttons or unlabeled buttons. This strategy might be similar to a strategy used by an untrained or computer-fearing user. Surprisingly, the model also did not succeed if the word appeared in all capital letters.

The model did not use the most efficient strategy for finding the target. For instance, the model would attend the same visual location several times in a trial. In addition, the model did not give up when the desired target was not present, and did not change from a random search to a structured search. Although the model was able to perform these telephone tasks, it was not able to perform them well.

6.0 How people use telephones

We analyzed human performance on the telephone interfaces to test and validate the cognitive model's predictions. We tested with nine experiment subjects between the ages of twenty-one and forty-five who are members and friends of our Applied Cognitive Science lab. The experiment took approximately thirty minutes without eye-tracking and approximately one hour with eye-tracking for each subject.

We analyzed human performance in terms of visual search, total dialing time, and number of errors made. Eye-tracking data was used to examine visual search components like number and locations of fixations. The button-clicking logs recorded in the experimental software formed the basis of our reaction time and error counting analysis.

From the eye-tracking videotapes, we are able to observe several instances where the eye is moving ahead of the mouse. This movement could be due to the eyes performing visual search in parallel with the hands performing motor movements, or it could be the eyes predicting where to go next. There are also instances where the mouse seems to "remember" where to go, while the eyes saccade across the telephone interface. Both of these behaviors suggest a high level of parallelism in user behavior. They also suggest that humans have a lot of knowledge about dialing telephones and are able to efficiently apply this past knowledge to the current task. The users certainly use a search strategy with a structural component because subsequent saccades almost always bring them closer to the desired phone digit.

In the case of the mouse "remembering" where to go, this does not seem to be supported by the ACT-R theory. By "remembering", we mean that the mouse starts to move in the correct direction of the target rather than moving exactly to the target. Regardless of what this premature movement means, this is different behavior than produced from the ACT-R model, which kept the mouse perfectly still between movements.

6.1 Where and how they look (eye-tracking)

People are able to quickly find the telephone keypad. This is probably due to the pop-up phenomenon of the keypad. The keypads were generally the largest structured areas on the interfaces and always contained the most relevant information for dialing. Once the keypad is discovered, the person's eyes rarely (if ever) stray from the keypad.

Our experiment used a seventeen-inch monitor with 1024x768 pixel display at a distance of twenty-four inches, and therefore the fovea should cover a circle with a thirty-four pixel diameter. Our data frequently included fixations that were more than thirty-four pixels from expected targets, suggesting that the experiment participants frequently used their para-fovea and peripheral vision.

Like much eye-tracking data, the eye-tracking data we collected is rather noisy. Hornof and Halverson (2002) suggest a method of cleaning up this noise by using “required fixation locations” (RFL). RFLs are locations that the subject must fixate on to successfully complete the experimental trial. Their experiment involved a display with widely spaced visual targets, and eight possible RFLs were chosen. However, a telephone keypad does not lend itself well to this idea. Telephone keys are generally close together. All phones tested had number keys within fifty pixels of their neighbors. This fifty-pixel distance is within the bound of the expected error from our eye-tracking unit. Also, it is expected that subjects may fixate on several items at once because more than one button can fall within the fovea. For instance, subjects were often observed clicking the “4” button while it appeared that their fixation was on the “1” button.

See Figure 10 for an eye-tracking example of human use of a telephone. The figure shows a screen-capture of the experiment environment, with fixations superimposed as crosses and connected in order of visitation. The square dot represents the expected first fixation location (where the “Go” button was) and the blue crosses represent fixations. Thicker crosses represent longer fixations. The line width of the crosses increases with the time spent at that fixation.

Next we summarize user fixation data, in terms of number of fixations. The summary is presented two ways: averaged across telephones and averaged across telephone numbers. This allows us to compare the model’s fixation predictions against users in two ways. In Table 15 and Figure 11 the telephones are presented in random order, and in Table 16 and Figure 12, the telephones are sorted in increasing Fitts’ Law time order.

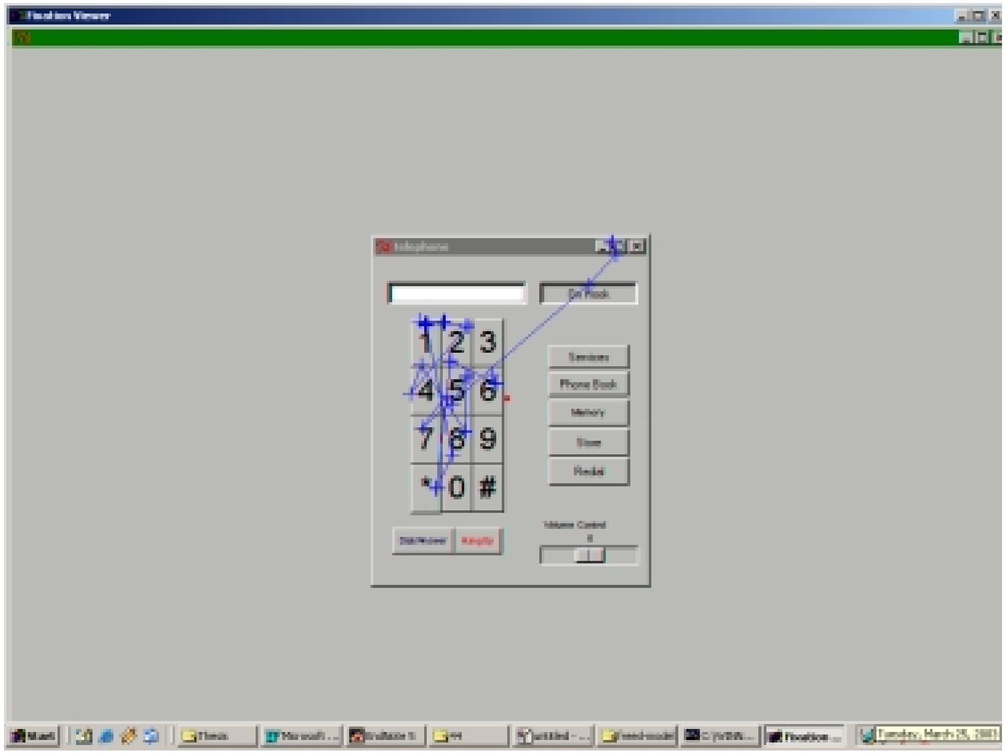


Figure 10. User dialing 814-234-9657 and closing window.

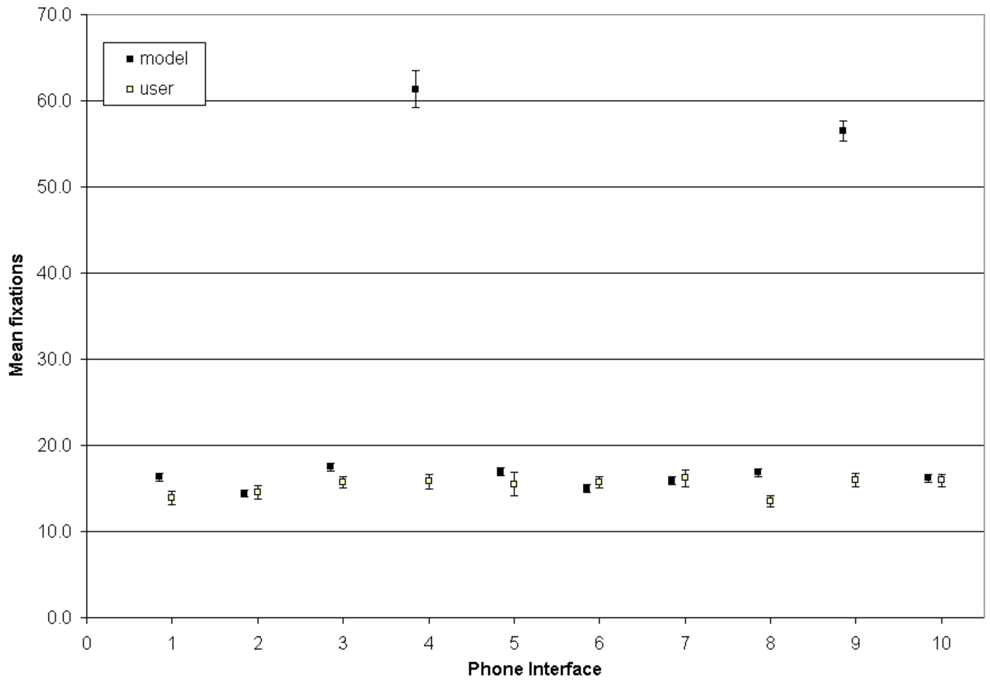


Figure 11. Fixations made dialing each phone across all tested numbers. $R^2 = 0.11$.

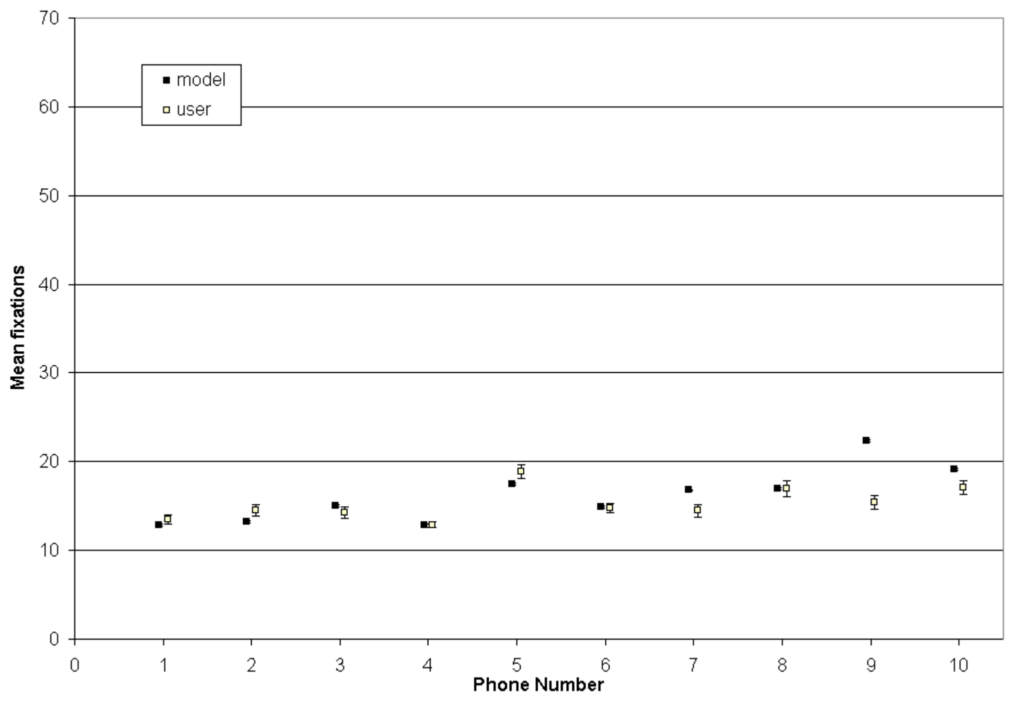


Figure 12. Fixations made on each telephone number across eight phones. $R^2 = 0.34$.

Table 15. Fixations made across all tested telephone numbers.

Number of fixations between ACT-R model and users. $R^2 = 0.11$ between model and user times.

Phone	1	2	3	4	5	6	7	8	9	10
Model	16.3	14.4	17.5	61.4	16.9	15.0	15.9	16.8	56.5	16.2
User	14.0	14.6	15.8	15.8	15.5	15.7	16.2	13.5	16.0	16.0

Rank of fixations for users and ACT-R model. $R^2 = 0.12$ between model and user ranks.

Model	5	1	8	10	7	2	3	6	9	4
User	2	3	6	7	4	5	10	1	9	8

Table 16. Fixations across eight tested telephone interfaces.

Number of fixations averaged across interfaces. $R^2 = 0.34$ between model and user times.

Number	814 866 5000	123 654 7890	814 863 5000	412 268 3000	606 193 3012	215 654 5785	814 234 9657	740 611 9273	101 010 1010	103 273 1029
Model	12.8	13.2	15.0	12.8	17.4	14.9	16.8	17.0	22.3	19.1
User	13.5	14.5	14.3	12.9	18.9	14.8	14.5	17.0	15.4	17.1

These results indicate that the ACT-R model requires approximately the same number of fixations to dial a telephone as the experiment subjects. This suggests that the visual search strategy employed by the model is similar to the one employed by the experiment subjects. However, the strategy appears to be wrong when dialing 101-010-1010. The experiment participants required approximately seven fewer fixations for this number, suggesting they were more quickly able to saccade between the “1” and “0” buttons, or that they were able to memorize the locations of the buttons, or both.

T-tests to determine whether the number of required fixations per telephone are statistically significant are not included due to the small number of subjects who participated in the eye-tracking experiment. On a per-number basis, the phones cannot be shown to require statistically significant different numbers of fixations. This correlates well with the ACT-R model that predicted that phones other than 4 and 9 did not require a significant different number of fixations per telephone number. From this we can infer that the difference in telephone dialing times is probably not due to visual search.

6.2 How people actually use phones

An interesting trend discovered from the eye-tracking data was that people would often “look ahead” to the next phone digit. For example, when subjects dialed the “2-3-4” sequence, the mouse pointer would move from the “2” to the “3”, while their eye would move towards the “4” which is nearly in the opposite direction. Part of this is due to the human ability to see things not in the fovea or para-fovea by use of peripheral vision. A mental model of the telephone keypad is likely engrained in the person’s mind that helps them anticipate the location of the next feature while still moving the mouse towards the present target.

We know that visual search has memory of approximately four items (Peterson, Kramer, Wang, Irwin, & McCarley, 2001), so perhaps this helps users remember where to look next. By this theory a user could memorize the location of a few numbers such as a row or column and use this data to help find the next desired target.

6.3 Humans make errors

One of the most important aspects of interface testing is discovering where users are likely to make errors. While Fitts’ Law can not detect likely sources of errors, and the current version of ACT-R does not make errors while dialing telephones, humans can and frequently do make errors while using a telephone. Most of the errors detected were due to failing to properly click on an intended digit and a smaller group of errors were due to clicking an extra or incorrect digit. The errors are dependent on both the telephone number dialed and the telephone interface being used.

Errors dependent on the telephone number dialed showed some interesting trends. First, the number of errors does not strictly increase with predicted Fitts' Law time. Second, errors are less likely to occur on telephone numbers with patterns of repeated digits. The numbers with the least number of errors were 814-866-5000 and 412-268-3000. The latter of those had fewer errors than the number with a very simple tracing pattern (123-654-7890) that would have consisted of small movements for every digit. The most errors were for the number 215-654-5785 that requires several small mouse movements that do not follow an easily memorized pattern. The errors were most numerous on telephone numbers that subjects informally commented on as being hard to remember. Overall there was a correlation between the Fitts' index of difficulty for dialing a telephone number and the error rate ($R^2 = 0.26$).

The error data more interesting to interface designers are the error rates among the different telephone interfaces. Unlike the telephone numbers, error trends across telephone interfaces were strongly influenced by Fitts' Law principles. The phones with the most errors were phones 7, 6, and 10, that are characterized by small keypad buttons that have space between them. Phone 9 also had many errors due partly to the large space between its large buttons and partly due to the difficulty of reading the white button labels on blue button backgrounds. The phones with the least errors were phones 3, 2, 5, and 1 that are characterized by large, adjacent keypad buttons. The phones with the median number of errors (4 and 8) had a mixture of these principles, using adjacent or nearly adjacent small keypad buttons. Overall there was a correlation between the Fitts' index of difficulty for dialing on a telephone interface and the error rate ($R^2 = 0.39$). This correlation is stronger than the correlation between errors and the telephone number being dialed. Figure 13 plots error counts against Fitts' index of difficulty.

One more factor seems to effect error rates. Subjects who dialed the telephones faster tended to make more mistakes. An exponential regression showed a slight correlation ($R^2 = 0.29$) between increase in errors and decrease of dialing time. A contributing factor to this may be the way that Tcl/Tk button clicks are recorded. The mouse must remain over the button during both the down-click and the up-click. The fastest users often did not wait until the up-click before moving the mouse towards the next target.

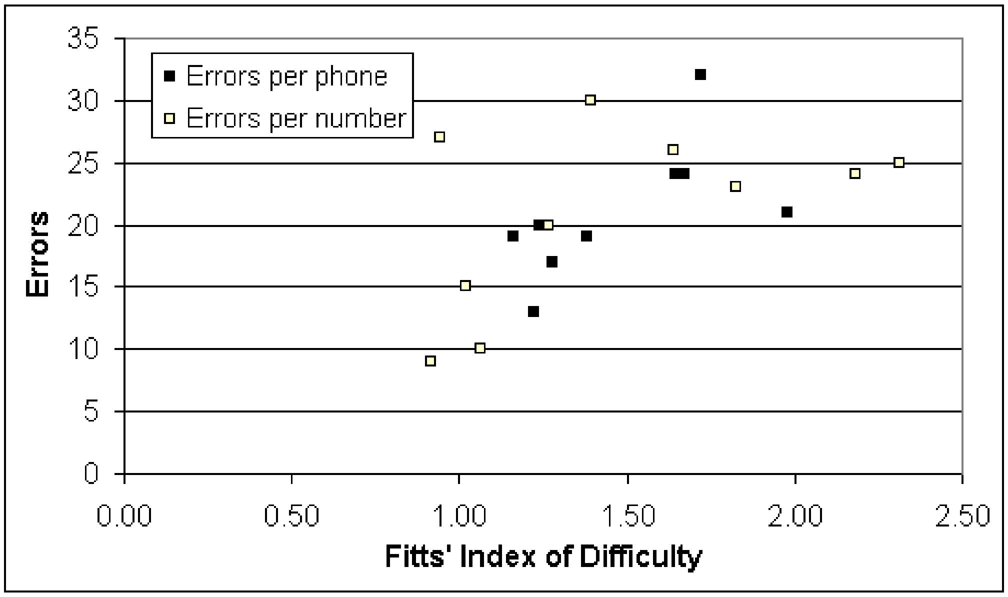


Figure 13. Errors plotted against Fitts' index of difficulty.
$R^2 = 0.39$ (per phone), $R^2 = 0.26$ (per number).

The ACT-R model did not make these kinds of mistakes because it never moves the mouse except to move to the next digit and especially not while clicking. Human users frequently move the mouse between required movements and also moved the mouse during some clicks.

6.4 Summary of user reaction times

The lowest user times are for phones 5, 3, 2, and 1 (in that decreasing order). Although participants informally claimed to dislike the telephone interfaces with buttons close together – also reported in Savage (1995) – participants are usually able to dial these more quickly due to Fitts' Law. User dialing times are summarized in Table 17 and Figure 14. The experiment subjects also made occasional mistakes and these trials were not included in the analysis.

From Table 17 and Figure 14, we can see that the model makes several correct predictions. The model is correct in predicting that phones 4 and 9 will take the longest to dial. While Fitts' Law was able to predict this poor performance for phone 9, it took a cognitive model to find the flaw in phone 4. The model is also able to predict that phones 5 and 8 will be more difficult to use than predicted by Fitts' Law, and that phones 2 and 7 will be less difficult than predicted by Fitts' Law. Further, the model predicts that Fitts' Law is correct in predicting rankings for phone 1. The model shows some room for improvement with possibly inaccurate predictions for phones 3 and 6.

Table 17. Summary of user dialing times, averaged over all telephone numbers.

$R^2 = 0.41$ (ACT-R model), $R^2 = 0.85$ (Fitts Law).

Phone	1	2	3	4	5	6	7	8	9	10
Model	7.84	7.44	8.25	18.43	8.05	7.83	8.17	8.05	16.80	8.05
Fitts	5.77	5.85	5.70	6.12	5.60	6.40	6.41	5.80	6.98	6.37
User	6.42	6.35	6.31	7.02	6.16	7.08	6.86	6.53	7.83	7.14

User rank presented in [minimum **overall** maximum] observed rankings.
$R^2 = 0.13$ (ACT-R model), $R^2 = 0.77$ (Fitts Law).

Model	3	1	8	10	4	2	7	5	9	6
Fitts	3	5	2	6	1	8	9	4	10	7
User	1 **4** 8	2 **3** 8	1 **2** 9	5 **7** 10	1 **1** 4	4 **8** 9	4 **6** 10	1 **5** 7	6 **10** 10	6 **9** 9

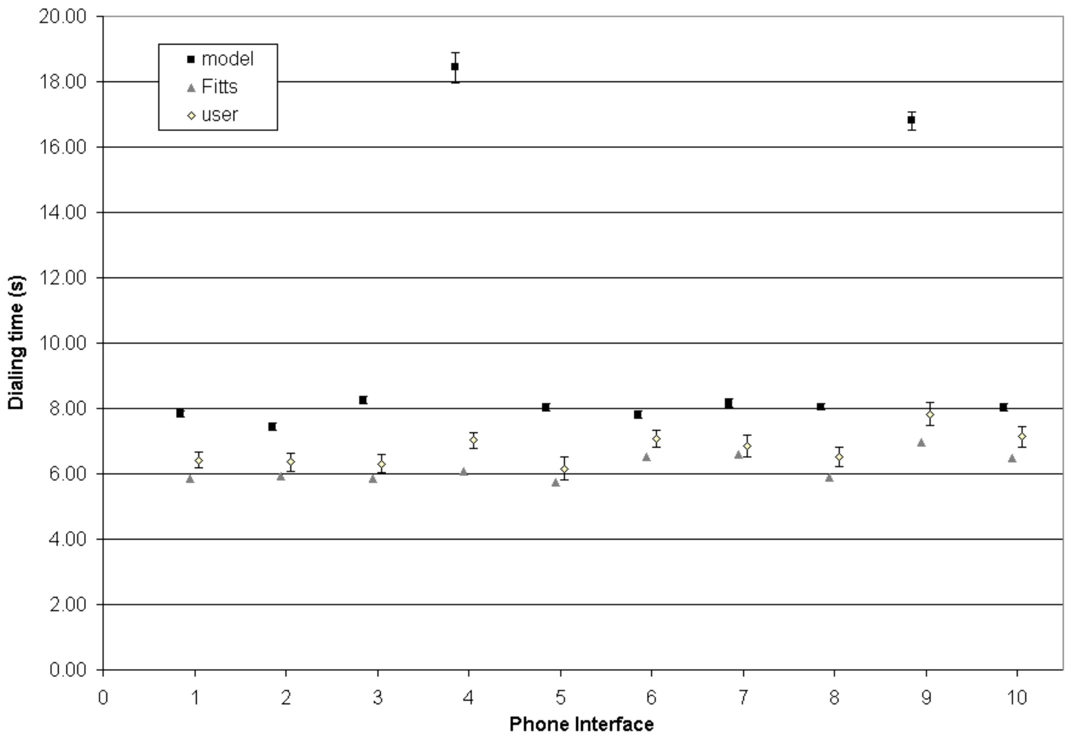


Figure 14. User dialing times compared with ACT-R model and Fitts' Law.

Certainly, the model reaction times overestimate the time needed by users to dial telephones. The model is overestimating by an average of 1.35 s (~20%) on the eight phones it easily dials, and performs much worse (overestimated more than 100%) on the remaining two telephones. This problem may be due to expert users (all experiment subjects were skilled computer users), or may be a fundamental problem in the ACT-R architecture, or may be due to an overly simplistic cognitive model.

There is a lot of variance between the users individual ranking of the phones. The predicted ranking by the ACT-R model falls within bounds predicted by users on eight of the phones, and just outside the bounds on the remaining two phones. The users generally do not uniformly improve or worsen their dialing time as the telephone interfaces change. This suggests several things. First, different users may employ different search strategies, and different phones may perform better or worse with these strategies. Secondly, users may differ in motor movement abilities. Some users may have more difficulty making movements left to right, or up to down, and if the buttons on the phone are spaced in certain ways it may adversely affect performance. Third, there may be other features of the telephone such as colors, font sizes, etc. that affect users in different ways. Finally, some of this variance is probably natural due to running a relatively small number of trials per user, as well as a relatively low number of users.

Table 18 and Table 19 summarize the statistical significance of differences in dialing times between phones for 412-268-3000 and 101-010-1010 respectively. The T-tests were carried out separately for each dialed telephone number, because the differences in fixation counts varied between telephone numbers. The user data shows fewer statistically significant differences between telephones than the model data. This data shows that the model was correct in predicting that some of the phones were statistically different from other telephones and that the level of significance differed by telephone number dialed.

Table 18. T-test comparing mean dialing times for 412-268-3000 (p<.05).

Black bars represent a rejection of H_0

Phone	1	2	3	4	5	6	7	8	9	10
1	-								■	
2		-			■				■	
3			-		■				■	
4				-	■				■	
5		■	■	■	-	■	■	■	■	
6					■	-			■	
7					■		-		■	
8					■			-	■	
9	■	■	■	■	■	■	■	■	-	■
10									■	-

Table 19. T-test comparing mean dialing times for 101-010-1010 ($p<.05$).

Black bars represent a rejection of H_0

Phone	1	2	3	4	5	6	7	8	9	10
1	-		■		■	■	■	■	■	■
2		-					■		■	
3	■		-				■		■	
4				-			■		■	
5	■				-					
6	■					-	■		■	
7	■	■	■	■		■	-			■
8	■							-		
9	■	■	■	■		■			-	■
10	■						■		■	-

Figure 15 and Figure 16 show dialing times for 412-268-3000 and 101-010-1010 respectively.

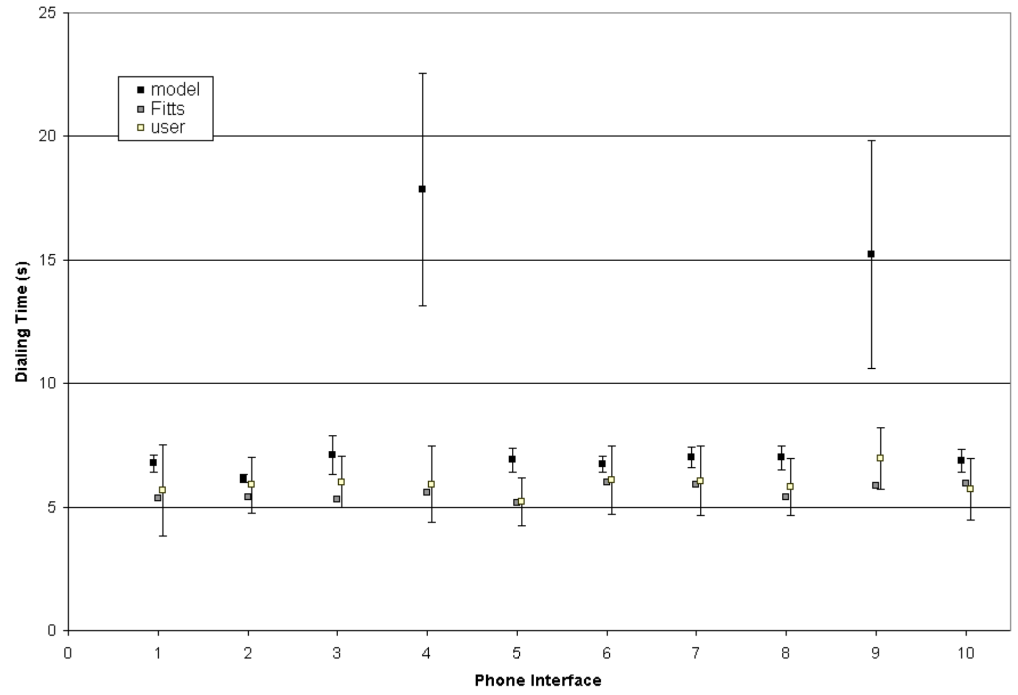


Figure 15. Dialing times for 412-268-3000.
$R^2 = 0.26$ (ACT-R model), $R^2 = 0.30$ (Fitts Law).

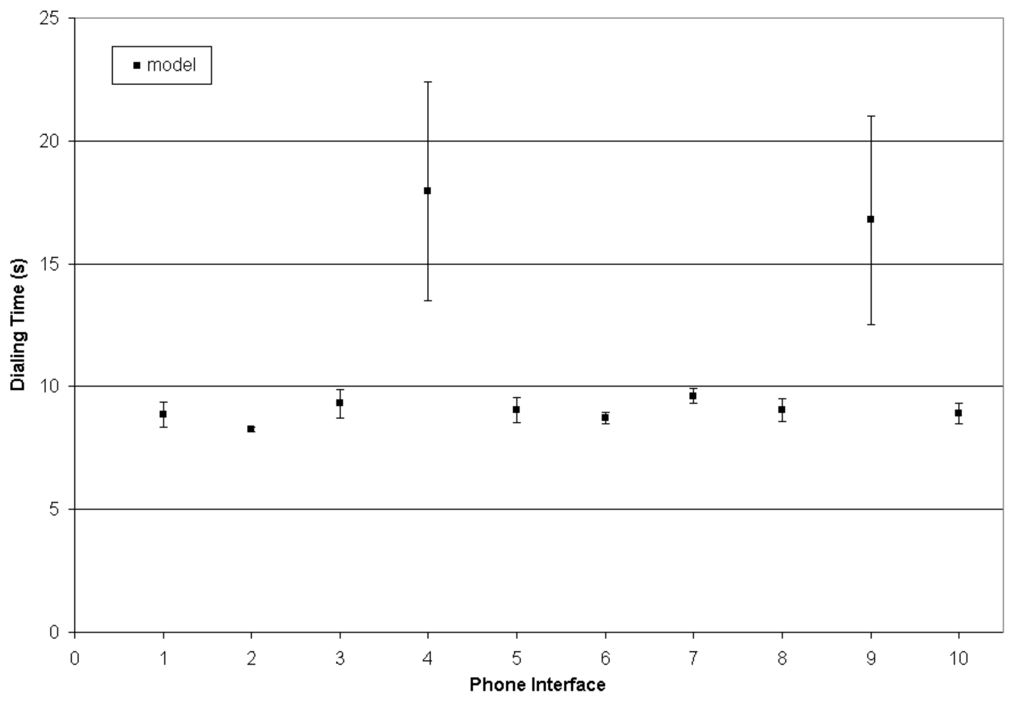


Figure 16. Dialing times for 101-010-1010.
$R^2 = 0.04$ (ACT-R model), $R^2 = 0.50$ (Fitts Law).

Table 20 and Figure 17 summarize the reaction time for the other telephone tasks. Note that the human users are able to detect the abbreviation “Conf.” for conference, although the ACT-R model could not. The experiment subjects were always able to find the target if it was present (i.e. a “forward”, “redial”, or “conference” button). The subjects frequently pursue more varied visual targets. For instance, they looked under menus and clicked buttons with missing or improper labels. Fitts’ Law analysis is not considered because this task is dominated by visual search.

Table 20. User and (ACT-R) reaction time for the other telephone tasks.

Phone	1	2	3	4	5	6	7	8	9	10
Forward	7.2	3.75	2.5	11.8	2.4	7.5	11.1	3.2	3.3	1.94
$R^2 = 0.60$	(-)	(8.1)	(10.7)	(-)	(6.2)	(-)	(-)	(6.4)	(-)	(9.5)
Redial	7.5	1.7	3.3	5.8	3.3	6.3	5.3	3.5	20.9	2.4
$R^2 = 0.26$	(-)	(7.2)	(10.1)	(5.7)	(-)	(-)	(-)	(7.4)	(-)	(11.9)
Conference	4.7	4.9	5.4	11.1	3.1	3.5	9.7	3.3	3.9	8.3
$R^2 = 0.10$	(-)	(7.0)	(9.4)	(-)	(6.8)	(-)	(-)	(-)	(-)	(-)

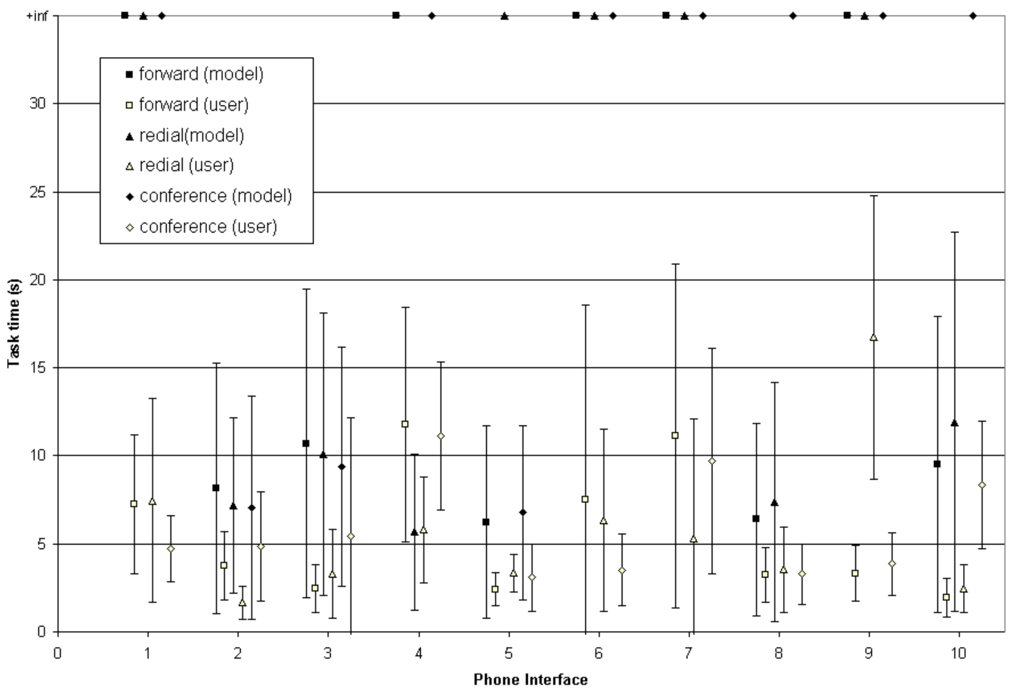


Figure 17. Dialing time for other telephone tasks.

These results show that the cognitive model can perform the tasks almost as completely as humans in call forwarding, redial, and conference calling, although much slower. If the desired target is not present the model will not give up as quickly as humans will give up because it does not give up at all. The participants might have felt that searching for a "conference" button was a waste of time if they had already scanned the entire phone and not found the desired target. On the other hand, the model does not care that it may be wasting time. The model also does not care how fast it dials the telephones (users may see it as a challenge), it does not care about making mistakes, and does not worry about remembering long phone numbers. Perhaps the introduction of behavioral moderators into ACT-R will help the model make more accurate predictions. In fact, research is already underway to model some of this into the ACT-R architecture (Ritter, Avraamides, & Councill, 2002; Ritter, Avraamides, Councill et al., 2002). At some point, if the model cannot complete the task, it should give up (Belavkin & Ritter, 2003).

7.0 Discussion and comparison of the three analyses

We considered these three types of analyses: analytic models for quick computation, cognitive models for increased predictive ability, and humans for real-world data. Each type of analysis has proven both useful and incomplete. There are several areas where the cognitive model and underlying cognitive architecture can use improvement, and the results show that there are several inadequacies of a purely analytical approach. Finally, the human data cannot make generalizable predictions, because of the large human population compared to the experiment sample population.

7.1 Analytic model is not enough

The Fitts' Law predictions were useful in determining a ranking of the phones based on dialing times, but the dialing times were underestimates of the user dialing times. While taking fewer factors into consideration, Fitts' Law enabled us to do a quick analysis of our telephones. This analysis was able to suggest that several of the telephones might outperform the others, and that there were few designs that initially stood out as being bad. However, this analysis did not catch any other usability issues, such as the telephone with the upside-down keypad.

Fitts' Law is relevant for telephone dialing, but does not provide enough details. Our Fitts' Law analysis showed that the phones would have widely varying dialing times. However, this variation did not show up as pronounced in either the model predictions or the human results. This suggests that Fitts' law may be inadequate to make predictions on interface usage because movement time might be dominated by the visual search time even though it happens in parallel. For these reasons, analytic models such as Fitts' Law are useful but cannot be considered a panacea.

7.2 Cognitive models can make more complicated predictions

Cognitive models can make stronger predictions about interfaces than Fitts' Law can, because they are able to assess more factors. For instance, our ACT-R model predicted that phone 4 would have major usability problems due to the upside-down keypad. This is a prediction that Fitts' Law did not make. The model also predicted that phones with many extra visual features would require more fixations during visual search, although this prediction was not made at a statistically significant level.

Unfortunately, several R-squared analyses showed a poor correlation between the ACT-R model performance and the users. Even more surprisingly, the analytical model generally showed higher correlation to the user data. We feel that the ACT—R/PM model might correlate better with the user data after some improvements are made to its visual search (by including EMMA) and also improvements to the motor pipeline.

A problem with cognitive models is that they are difficult to develop and use (Councill, Haynes, & Ritter, 2003; John, Vera, Matessa, Freed, & Remington, 2002; Pew & Mavor, 1998). Cognitive modeling is difficult in three ways: creating a running model, providing the correct knowledge to the model, and performing the task the same way actual users do. Council et al. (2003) and Freed et al. (2003) are looking into ways to make modeling a simpler endeavor.

7.3 Visual search differences between ACT-R and users

Humans seem to use more efficient visual search strategies than the cognitive model. They occasionally move the mouse while searching and are able to adapt to unexpected designs (the upside-down keypad). People seem to be very efficient at finding the keypad and not making too many extra fixations once they have found it.

Presently, the model must focus on every button it looks at to determine if it is the target button. Eye-tracking data from the experiment participants showed us that humans do not fixate explicitly on each visual target, because they either fixate on multiple targets or use peripheral vision. In addition, the visual search strategy is less than desired at times, as the model may make haphazard saccades. For instance, if the model is looking at the “1” but trying to find the “5”, it will use a production that tells it “the 5 is down and to the right.” The next fixation may appear somewhere much farther away, such as on the “#” (also down and to the right). This tendency to overshoot at times can lead to prolonged visual search.

The model does not take advantage of the ability to see several items within its simulated fovea nor does it effectively use its peripheral vision. The EMMA package for ACT-R created by Salvucci (2000) was designed to introduce more realistic eye movement and feature encoding patterns, including peripheral vision, into the ACT-R cognitive architecture. However, the current version of EMMA does not run on

Windows. A planned future version of EMMA will work in Windows, and may help our cognitive model better fit our visual search data.

Menu search experiments by Byrne (1999) and Hornof and Kieras (1997; 1999) yielded several insights on visual search. These insights included the following:

- people process more than one menu at a time
- people use both random and systematic search strategies
- people process items in the fovea in parallel
- people do not stop and decide on menu items individually — they process them in parallel
- people select targets from ordered menus faster than from randomly organized menus.

The eye-tracking data we have seems to support several of these claims. There are several instances of users fixating between buttons, allowing them to process multiple items at once. It seems that random search is used to find the telephone keypad, and once the keypad is discovered the eyes do not stray from it. This suggests an organizational advantage because there are no useful fixation locations off of the keypad while dialing. Once on the keypad search is mostly systematic, although occasionally random.

The ACT-R model also makes some of these same predictions. Like human users, the model employs a random search to find the keypad, and never fixates off of the keypad after it is found. The model uses a systematic approach that is generally optimal, but at times inefficient (due to overshooting the desired target). The model does not process items in parallel because only one item is seen in the fovea at a time. This is a problem we would like to address in future work, after a Windows version of EMMA is released. In addition, ACT-R implements a one-to-one mapping between eye movements and attention shifts (Fleetwood & Byrne, 2003) even though Henderson (1992) and Rayner (1995) suggest that the mapping is not always direct. This is another area where EMMA could improve the fit of our ACT-R model to the observed data. As previously mentioned, the users would often look ahead to the next digit. This behavior may be due to attention shifts without eye movements.

The model also shows poor improvisation skills or visual problem solving. On phone 4, where the keypad was upside down, the human participants were able to recognize this abnormality and adjust their strategy accordingly, although dialing performance was adversely affected. The model was not able to dial that telephone at all until a random search was introduced because the systematic rules programmed into the model would fail. This means that the knowledge needed to dial a telephone varies on a per-interface basis.

We have also discovered that different knowledge is needed to dial different telephones. In the case of the upside-down keypad, the expected visual search strategy could not work. The ACT-R model switched to a random search strategy that is clearly not optimal, as evidenced by the user dialing times. Users were also able to cope with distracting information within the keypad (letters on the buttons) while the ACT-R model was not.

In the case of bad interfaces, defined as interfaces that required prolonged visual search, we see that these interfaces have greater variance in dialing time and fixation counts. Interfaces with "normal" visual search had smaller variance in both areas.

ACT-R with Segman still lacks some common visual search primitives. For instance the model cannot follow a straight line. A user could follow the outline of the telephone, or of the keypad, but the model could not search in this way. The visual search in the model could only find certain objects, specifically buttons and text labels. With improved search primitives, perhaps the model's visual search can match up better with humans.

Finally, the ACT-R architecture needs a way to add parallelism to the visual search process. Experiments done by Hornof and Kieras suggest that humans employ a fast parallel scan when first looking at an interface (1997; 1999). It is probably by this means that users are able to find the telephone keypad more quickly than the ACT-R model.

7.4 ACT-R and Segman need better representation of interfaces

ACT-R and Segman consider a button and its label as two separate entities, although we suspect that human users do not make this distinction while using an interface. The ACT-R model would only search for a number instead of a number button. Fitts' computations in the model would therefore assume a target size of only a few pixels (the size of one digit), rather than using the button size (which was larger, generally much larger). To repair this discrepancy, we modified Segman to report the size of text objects as the size of the button they were contained in. Therefore the width of the "1" is no longer three pixels, but is instead on the order of thirty pixels. This modification was telephone-specific, and was updated between model runs. In general, it would be useful for Segman and ACT-R to consider some visual objects with higher complexity, such as a button with a label. This assumption may have to change per task.

7.5 ACT-R model is generally slower than users

The model is generally too slow, although it could be sped up by reducing some ACT-R parameters, including those related to the motor and visual subsystems. Such parameter changes proved useful in efforts such as those made by Belavkin and Ritter (2003) and Hornof (1997). For the majority of telephone numbers the ACT-R model and experiment subjects were shown to have statistically significant (two-sided T-test with 95% confidence) dialing time differences on all or nearly all of the phones.

It is for this reason that we also analyzed the ranking of telephones in terms of dialing times. In this regard, the ACT-R model predictions matched more closely with human data. The model rankings generally fell within the bounds of the user rankings.

Our cognitive model almost always ran slower than the experiment participants. This may be partly due to the experiment participants being experienced computer users skilled with both interfaces and telephones in general, and therefore having fast reaction times and low mouse movement times. The dialing time for an experiment subject, averaged over all phones, numbers, and subjects, was approximately seven seconds. Our first model, using mostly random search, required approximately thirteen seconds (averaged across all telephone numbers). An improved model with "smart searches" (i.e. the "4" is below the "1") required approximately ten and a half, and the final model with improved parallelism required approximately 8.7 seconds.

The remaining gap between user and model performance still concerns us. A comforting fact is that the model makes approximately the same number of visual fixations as users do. This implies that the performance gap may be due to motor movement. This gap is even more pronounced when considering fastest times. The fastest dial time by a user was slightly under 3.5 s, while the fastest time by the model was a little less than 6 s. It is possible that experienced users are able to do a more efficient pipelining of motor tasks. ACT-R attaches a preparation and initiation phase for each motor movement. For dialing a ten-digit telephone number, these costs total approximately three seconds. No previous cognitive model has tried to press ten buttons in rapid succession, so this problem appears to be a new one.

ACT-R predicts that clicking a button takes 300 ms after combining preparation and execution times. When dialing a sequence such as "000", the model required 300 ms for each of the three clicks. The experiment subjects frequently required less than

200 ms for the last two clicks, and occasionally less than 150 ms for the last two clicks. In these cases it is likely that the subjects did not have to pay the same preparation costs for the final two movements as for the first. It is also possible that users are able to more efficiently chunk the telephone numbers than the model. The “000” sequence was probably stored in memory as one number, rather than three digits.

We suspect that the answer to the performance gap lies within motor abilities. The experiment participants required less preparation time per movement because they knew in the beginning that they would have a certain number of movements per trial. It is also possible that the users were memorizing the patterns of movement necessary to dial the phone and that this memorization may have happened both before and during the dialing trials. Because the phones were presented in a random order for each trial, subjects would not be able to memorize the magnitudes of the movements, only the general directions.

We argue that even though the predictions made by the model were significantly different than the human data, the relative speed and quality of the phone interfaces was maintained. According to Card, Moran, and Newell (1983), this should be a goal of modelers. We hope that after improvements to the cognitive architecture, our cognitive model will be able to better fit the human data.

8.0 Further work

We now discuss how this analysis can be useful in other ways. We explain how this analysis must be modified to analyze cellular phones, how to analyze computerized menus, how to extend the present analysis, and how to apply our analysis methods to generic interfaces. We conclude with a reiteration of why interface analysis with cognitive models is useful.

8.1 Cellular phones

Computerized telephone interfaces must be tested differently than physical telephone interfaces. In general, computer interfaces with adjacent buttons will be dialed faster than other interfaces because of a lesser Fitts' Law penalty. However, it seems that this discovery would not correlate well to physical phone interfaces, because it is possible to press two buttons at once with the dialing finger. While the mouse has an effective clicking area of one square pixel, a fingertip would have an area several magnitudes larger.

While looking at a cellular phone advertisement, we noticed that these small phones generally had small buttons that were spaced out. One phone that stood out was the Nokia 7210 cellular phone (Figure 18). This phone has a designer interface that doesn't immediately seem to support usability. This interface would have performed poorly in our experiment due to Fitts' Law penalties, but when a fingertip is used to press buttons instead of a mouse we hypothesize that this phone will not perform as poorly as Fitts' Law may suggest.

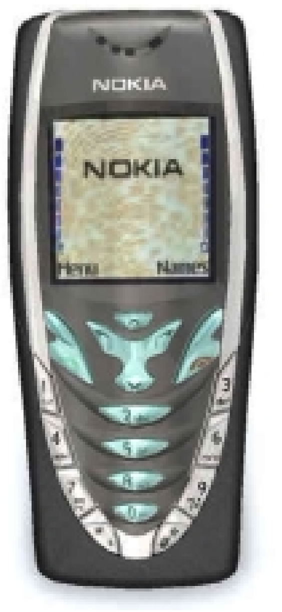


Figure 18. Nokia 7210 cellular phone. From nokia.com.

Table 21 shows the most common tasks for cellular phone users. Cellular phones are used differently than handsets, desktop phones, and computerized phones, and thus support different tasks. Even the simplest tasks on a regular phone, like dialing a number and hanging up, are supported differently on cellular phones. A study on cellular phone interfaces could start with the analysis we provided, but would need to eliminate testing of certain tasks (conference calling, call forwarding) and add testing for these new tasks.

Table 21. Top cellular phone tasks from Golightly (2003).

1. Making a call
2. Answering a call
3. Opening a newly received text message
4. Writing and sending a text message
5. Accessing a contact from your address book
6. Adding a contact from scratch
7. Adding a contact from a recently called number
8. Setting the phone to silent profile/vibration mode

8.2 Hutchinson 3G UK phone task

David Golightly is an HCI expert at Hutchinson 3G UK, a mobile phone company with over three billion dollars in capitalization, introduced us to an interface design problem for cellular phone menu selection. Several methods have been proposed for navigation through three item by three item square menus, and the methods include three different sets of allowed movements: left and right only; up, down, left, and right; and all eight directions. The methods also included two different initial starting points – top-left or center. The poorest design would combine left/right movement with starting at the top-left because it would take eight movements to get to the farthest target. The best design, eight movement directions with center starting point, takes only one movement to reach any target in the best case. This shows that usability must always be a top concern in design.

8.3 Analyzing ten additional designs

Testing interfaces with actual users is time-consuming and can be expensive. To test our ten interfaces with nine users took several weeks, mostly due to scheduling difficulties. In contrast, analyzing ten telephones for Fitts' Law took only ten minutes per phone to initialize, and then thirty-five minutes per phone to test across all possible ten-digit telephone numbers. The five hundred trials performed with the ACT-R model took less than one hour per phone. This suggests that testing interfaces with actual users might be needlessly expensive, when computational models can test the interfaces just as well, at lower cost, for particular problems.

8.4 Advice for interface designers

Interface design must take into consideration the needs of the user. Common telephone tasks should be supported easily on any interface. Flashy interfaces may be fun, but they can be more difficult to use. A task analysis is necessary to determine what your users hope to accomplish with your interface, and as a designer you must make sure you make these functions easily accessible.

An interface must be rigorously tested to ensure that it allows users to easily accomplish their tasks. Testing with actual users may be too expensive, and less costly testing can be done with analytic and cognitive models. These models are easily reusable and will ideally be available en masse on the Internet.

Testing and analyzing interface designs can have a tremendous impact on the bottom line. With interfaces that are heavily utilized to perform a certain task, a time savings of two seconds per task can translate into several minutes per day, or several hours per year, or more. Time and energy that would have otherwise been wasted using a slow or inefficient interface can now be applied to other more useful tasks.

8.5 How to do this analysis more generally

We now explain how to do an analysis of one hundred VCRs. We use VCRs as a general example of interfaces, not because they are interesting on their own. To analyze VCR interfaces, a variety of interface designs should be collected. We chose to use telephone interfaces developed in Tcl/Tk, which is platform-independent and allowed us to install some extra code within the telephones. A cognitive model should be developed that can use these VCRs. Presently, the only cognitive architecture that can use any third-party computer interface is ACT-R 5.0 with Segman v3 (Anderson & Lebiere, 1998; Byrne, 1999; St. Amant & Riedl, 2001), although future versions of these architectures would also suffice. Eye-tracking apparatus should be set up to run human participants through an experiment.

A Fitts' Law analysis could also be performed but may not be as useful. While telephones are generally well-understood interfaces, fewer people may be used to VCR interfaces. After all, many people have problems programming their VCRs. Using a VCR generally requires some additional thinking and visual search that a Fitts' Law analysis will not be able to help with.

A cognitive model may be better able to fill the predictive gaps left by Fitts' Law. The model used for this report could be reused and adapted to suit VCR usage. Our model already includes visual search capabilities and has some hand-eye coordination in terms of finding buttons and clicking on them, while searching for other buttons. Like our telephone dialing model, a VCR-using model should be able to make predictions of actual human behavior without having to use actual humans as test subjects. This approach can be used to evaluate other generic interfaces and is therefore another step in the direction of automatic interface evaluation.

9.0 References

Andersen, J. R., & Douglass, S. (2001). Tower of Hanoi: Evidence for the cost of goal retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 27*(6).

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1990). *The adaptive character of thought*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.

Balbo, S. (1995). *Automatic evaluation of user interface usability: Dream or Reality? QCHI95 Symposium*, Bond University, pp. 9-16.

Belavkin, R. V., & Ritter, F. E. (2003). *The use of entropy for analysis and control of cognitive models. Fifth International Conference on Cognitive Modeling*, Bamberg, Germany, pp. 21-26.

Byrne, M. D. (1997). ACT-R Perceptual-Motor (ACT-R/PM) version 1.0b1: A users manual. Pittsburgh, PA: Psychology Department, Carnegie-Mellon University. Available through act.psy.cmu.edu.

Byrne, M. D. (1999). ACT-R Perceptual-Motor (ACT-R/PM) version 1.0b5: A users manual. Houston, TX: Psychology Department, Rice University.

Byrne, M. D. (2001). ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human-Computer Studies, 55*, 41-84.

Byrne, M. D., Andersen, J. R., Douglass, S., & Matessa, M. (1999). Eye tracking the visual search of click-down menus. *Human Factors in Computing Systems: Proceedings of CHI 99*, 402-409.

Byrne, M. D., Wood, S. D., Sukaviriya, P., Foley, J. D., & Kieras, D. E. (1994). *Automating interface evaluation. Proceedings of the CHI 94 Conference on Human Factors in Computer Systems*, New York, NY, pp. 232-237.

Card, S., Moran, T., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Councill, I. G., Haynes, S. R., & Ritter, F. E. (2003). *Explaining Soar: Analysis of Existing Tools and User Information Requirements. Fifth International Conference on Cognitive Modeling*, Bamberg, Germany, pp. 63-68.

Ehret, B. D. (2002). *Learning where to look: Location learning in graphical user interfaces. ACM INTERCHI 2002 Conference on Human Factors in Computing Systems*, pp. 211-218.

Emmerson, P. (2000). Review of iGEN software. *Ergonomics in Design*, 29-31.

Findlay, J. M. (1982). Global visual processing for sacaddic eye movements. *Vision Research, 22*(8), 1033-1045.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology, 47*, 381-391.

Fleetwood, M. D., & Byrne, M. D. (2002). Modeling icon search in ACT-R/PM. *Cognitive Systems Research, 3*, 25-33.

Fleetwood, M. D., & Byrne, M. D. (2003). *Modeling the visual search of displays: A revised ACT-R/PM model of icon search based on eye-tracking and experimental data. Fifth International Conference on Cognitive Modeling*, Bamberg, Germany, pp. 87-92.

Freed, M., Matessa, M., Remington, R., & Vera, A. (2003). *How Apex automates CPM-GOMS. Fifth International Conference on Cognitive Modeling*, Bamberg, Germany, pp. 93-98.

Freed, M., & Remington, R. (2000). *Making human-machine system simulation a practical engineering tool: An APEX overview. Proceedings of the 3rd International Conference on Cognitive Modeling*, Veenendaal, The Netherlands, pp. 110-117.

Freed, M. A. (1998). *Simulating performance in complex, dynamic environments.* Northwestern, Evanston, IL.

Golightly, D. (2003). Personal communication. February 12, 2003.

Grant, D. A. (1962). Testing the null hypothesis and the strategy and tactics of investigating theoretical models. *Psychological Review, 69*(1), 54-61.

Gray, W. D., John, B. E., & Atwood, M. E. (1992). *The precis of Project Ernestine or An overview of a validation of GOMS. Proceedings of the CHI 92 Conference on Human Factors in Computer Systems*,

Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance. *Human-Computer Interaction, 8*(3), 237-309.

Henderson, J. M. (1992). Visual attention and eye movement control during reading and picture viewing. In K. Rayner (Ed.), *Eye movements and visual cognition: scene perception and reading*. New York: Springer-Vertag.

Hornof, A. J., & Halverson, T. (2002). Cleaning up systematic error in eye tracking data by using required fixation locations. *Research Methods, Instruments, and Computers, 34*(4), 592-604.

Hornof, A. J., & Halverson, T. (2003a). *Cognitive strategies and eye movements for searching hierarchical computer displays. ACM CHI 2003: Conference on Human Factors in Computing Systems*, New York,

Hornof, A. J., & Halverson, T. (2003b). *Predicting cognitive strategies and eye movements in hierarchical visual search. Fifth International Conference on Cognitive Modeling*, Bamberg, Germany, pp. 261-262.

Hornof, A. J., & Kieras, D. E. (1997). *Cognitive modeling reveals menu search is both random and systematic. Proceedings of the CHI 97 Conference on Human Factors in Computer Systems*, New York, NY, pp. 107-114.

Hornof, A. J., & Kieras, D. E. (1999). *Cognitive modeling demonstrates how people use anticipated location knowledge of menu items. ACM CHI 99*,

Ivory, M. Y., & Hearst, M. A. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys, 33*(4), 470-516.

John, B. E., & Kieras, D. E. (1996). The GOMS family of user interface analysis: Comparison and Contrast. *ACM Transactions on Computer-Human Interaction, 3*(4), 320-351.

John, B. E., Vera, A., Matessa, M., Freed, M., & Remington, R. (2002). *Automating CPM-GOMS. CHI 2002*,

Kieras, D. E., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction, 12*, 391-438.

Kieras, D. E., Wood, S. D., Abotel, K., & Hornof, A. (1995). *GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'95)*, New York, NY, pp. 91-100.

Kishi, N. (1993). *Tools for graphical user interface evaluation using playback. CHI '93*,

Lebiere, C., & Anderson, J. R. (1998). Cognitive arithmetic. In J. R. Anderson & C. Lebi re (Eds.), *The atomic components of thought* (pp. 297-342). Mahwah, NJ: Lawrence Erlbaum.

Lecerof, A., & Paterno, F. (1998). Automatic support for usability evaluation. *IEEE Transactions on Software Engineering, 24*(10), 863-888.

MacKenzie, I. S. (1991). *Fitts' Law as a performance model in human-computer interaction.* Unpublished Doctoral, University of Toronto, Toronto, Ontario, Canada.

Meyer, D. E., & Kieras, D. (1997). A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review, 104*(1), 3-65.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Nielsen, J. (1997). *Telephone usability: voice is just another datatype.* http://www.useit.com/papers/telephone_usability.html

Perry, T. S. (1998). The father of the SPICE program and driving force behind the development of IC simulation will receive the 1998 IEEE medal of honor. *IEEE Spectrum*, 22-27.

Peterson, M. S., Kramer, A. F., Wang, R. F., Irwin, D. E., & McCarley, J. S. (2001). Visual search has memory. *Psychological Science, 12*(4), 287-292.

Pew, R. W., & Mavor, A. S. (Eds.). (1998). *Modeling human and organizational behavior: Application to military simulations*. Washington, DC: National Academy Press. books.nap.edu/catalog/6173.html.

Rayner, K. (1995). Eye movements and cognitive processes in reading, visual search, and scene perception. In J. M. Findlay & R. Walker & R. W. Kentridge (Eds.), *Eye movement research: mechanisms, processes, and applications*. New York: Elsevier Science Publishing.

Ritter, F. E. (2000). *A role for cognitive architectures: Guiding user interface design. Seventh Annual ACT-R Workshop*, Department of Psychology, Carnegie-Mellon University, pp. 85-91.

Ritter, F. E., Avraamides, M., & Councill, I. G. (2002). *An approach for accurately modeling the effects of behavior moderators. Proceedings of the 11th Computer Generated Forces Conference*, Orlando, FL, pp. 29-40, 02-CGF-002.

Ritter, F. E., Avraamides, M. N., Councill, I., van Rooy, D., Quigley, K. S., Klein, L. C., McNeese, M. D., Stine, M. M., & Rodrigues, I. M. (2002). *Pre-task appraisal and caffeine: An architectural overlay for ACT-R. Air Force Workshop on ACT-R Models of Human-System Interaction, Mesa, AZ, January 2002*,

Ritter, F. E., Baxter, G. D., Jones, G., & Young, R. M. (2000). Supporting cognitive models as users. *ACM Transactions on Computer-Human Interaction, 7*(2), 141-173.

Ritter, F. E., Baxter, G. D., Jones, G., & Young, R. M. (2001). User interface evaluation: How cognitive models can help. In J. Carroll (Ed.), *Human-computer interaction in the new millenium* (pp. 125-147). Reading, MA: Addison-Wesley.

Ritter, F. E., Belavkin, R. V., & Elliman, D. G. (1999). *Affective computing: The role of emotion in human-computer interaction*, British HCI Group one-day meeting in conjunction with University College London (collection of abstracts),

Ritter, F. E., Freed, A. R., & Haskett, O. L. (2002). *Discovering user information needs: The case of university department websites* (2002-3): Applied Cognitive Science Lab, School of Information Sciences and Technology, Penn State.

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R., Gobet, F., & Baxter, G. D. (in press). *Techniques for modeling human performance in synthetic environments: A supplementary review*. Wright-Patterson Air Force Base, OH: Human Systems Information Analysis Center.

Ritter, F. E., Van Rooy, D., & St. Amant, R. (2002). *A user modeling design tool based on a cognitive architecture for comparing interfaces. Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002*, pp. 111-118.

Ritter, F. E., & Young, R. M. (2001). Embodied models as simulated users: Introduction to this special issue on using cognitive models to improve interface design. *International Journal of Human-Computer Studies, 55*, 1-14.

Salvucci, D. (2000). A model of eye movements and visual attention. *Proceedings of the Third International Conference on Cognitive Modeling.*, 252-259.

Salvucci, D. (2001). Predicting the effects of in-car interface use on driver performance: An integrated model approach. *International Journal of Human-Computer Studies, 55*, 85-107.

Savage, P. (1995). Designing a GUI for business telephone users. *ACM Interactions, 2*(1), 32-41.

Schraagen, J. M., Chipman, S. F., & Shalin, V. L. (Eds.). (2000). *Cognitive task analysis*. Mahwah, NJ: Lawrence Erlbaum.

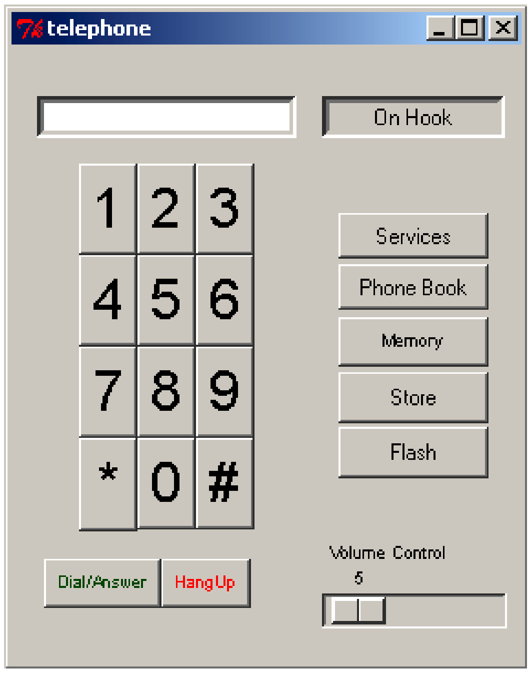
St. Amant, R. (2000). Interface agents as surrogate users. *Intelligence, 11*(Summer 2000), 29-38.

St. Amant, R., & Riedl, M. O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies, 55*, 15-39.

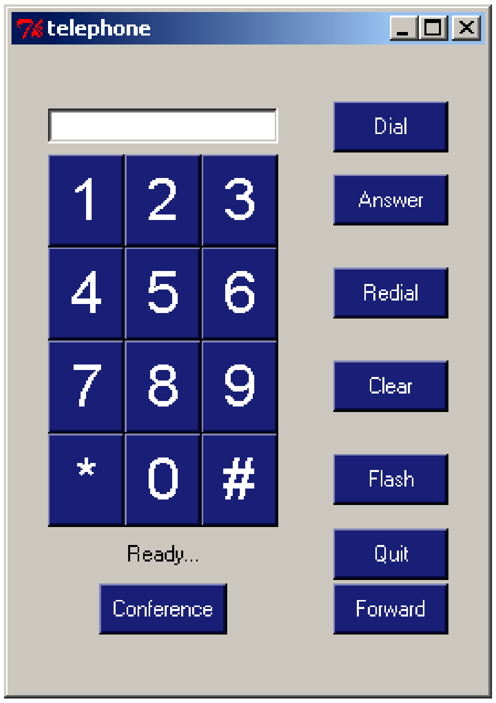
Vicente, K. (1998). *Cognitive work analysis*. Mahwah, NJ: Lawrence Erlbaum.

Zelinsky, G. J., & Sheinberg, D. L. (1997). Eye movements during parallel-serial visual search. *Journal of Experimental Psychology: Human Perception and Performance, 23*(1), 244-262.

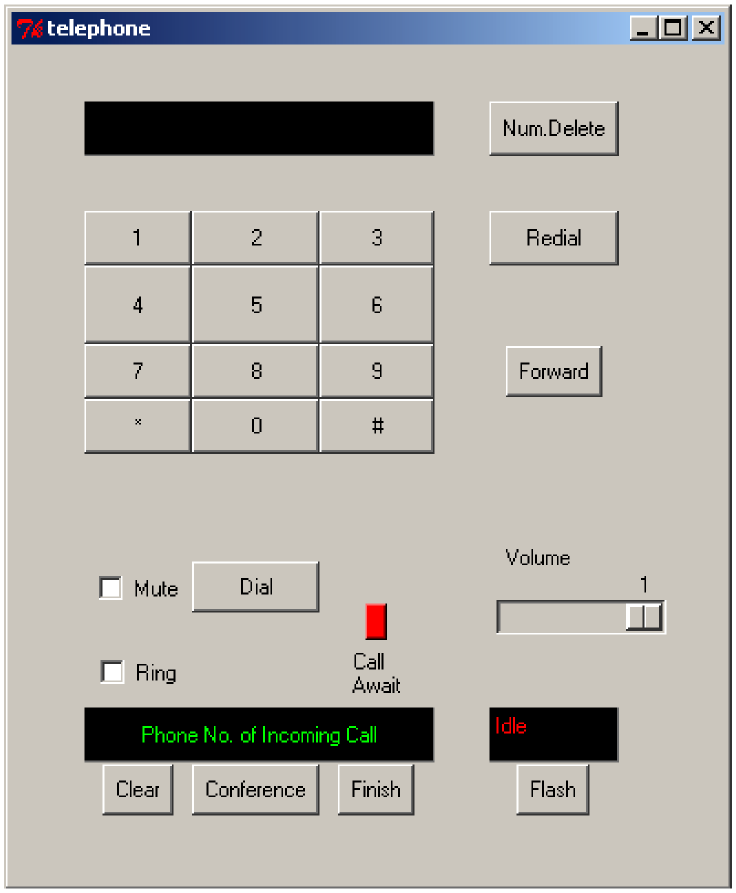
Appendix A: The Ten Telephones



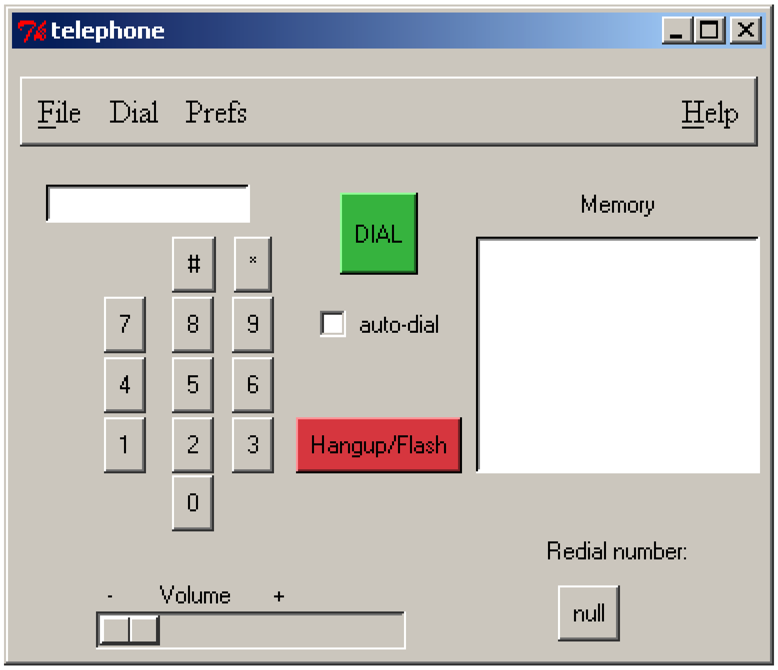
Phone 1.



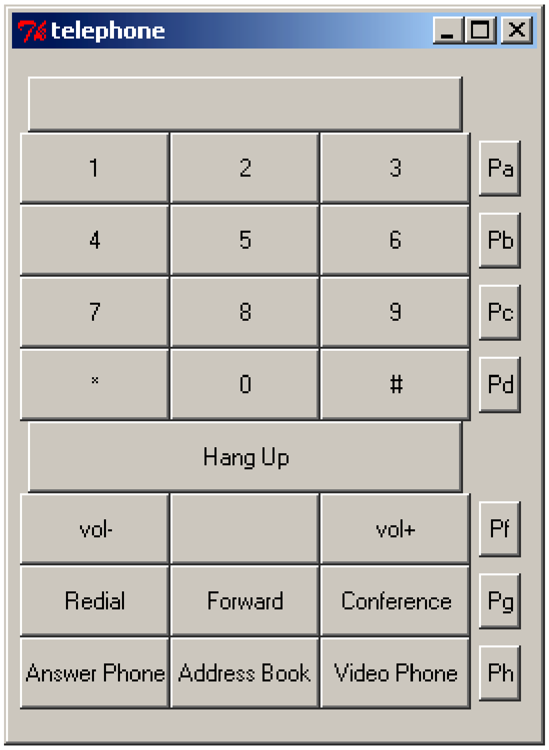
Phone 2.



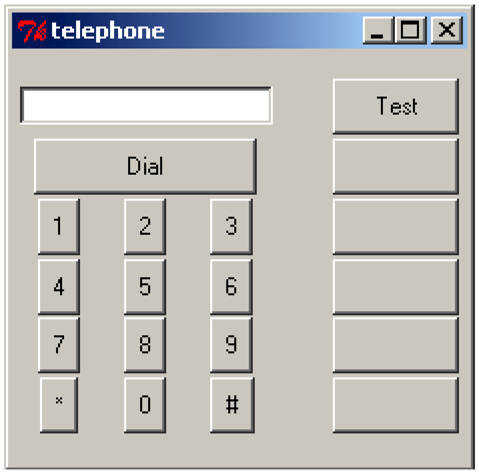
Phone 3.



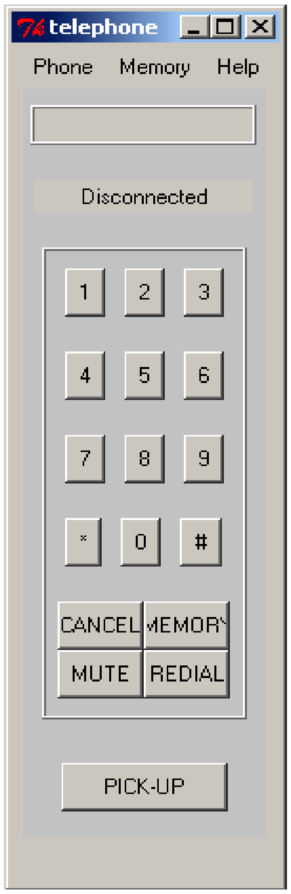
Phone 4.



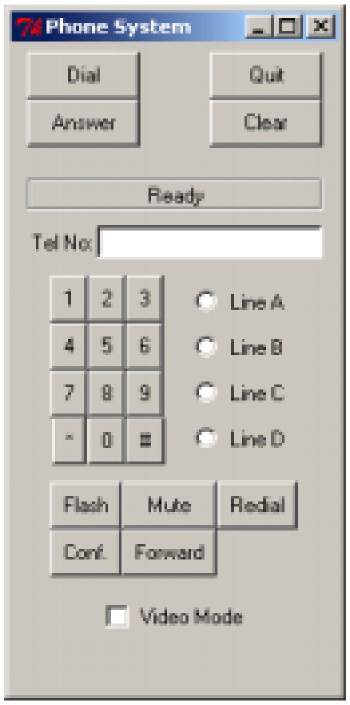
Phone 5.



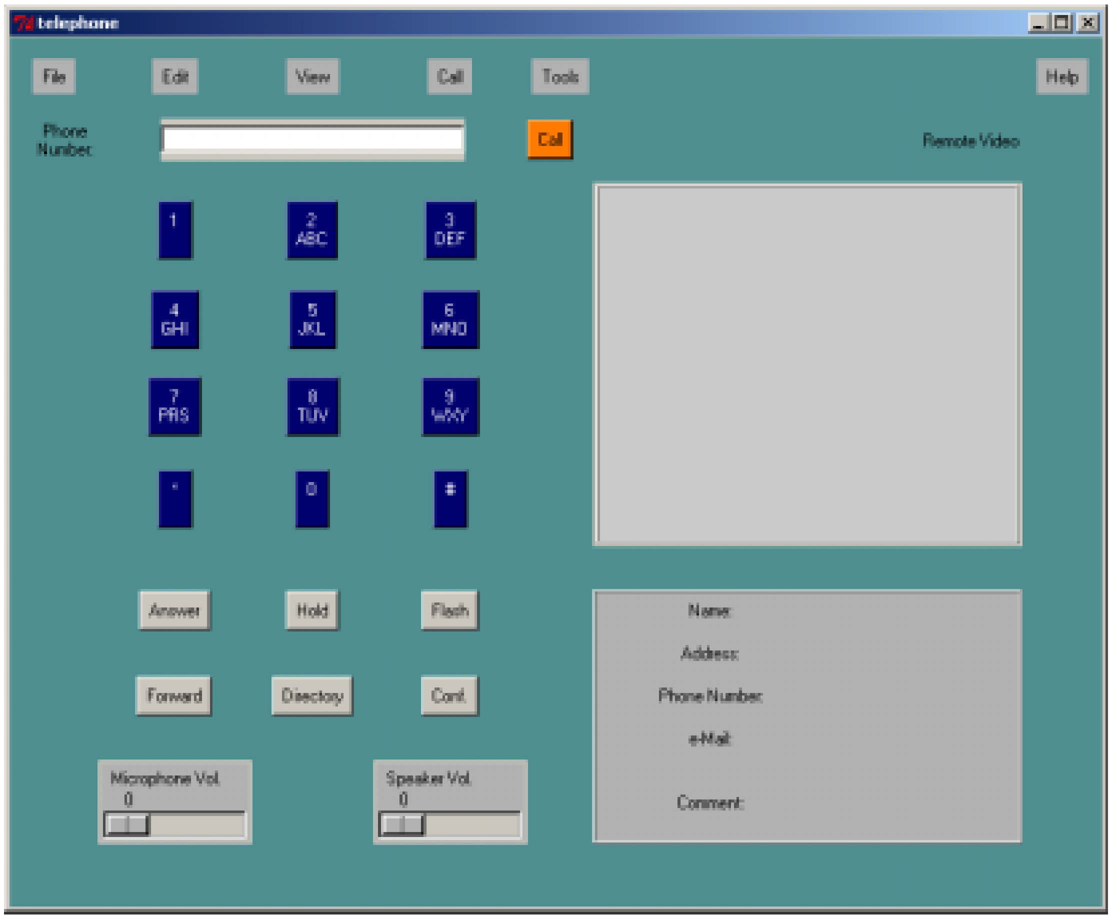
Phone 6.



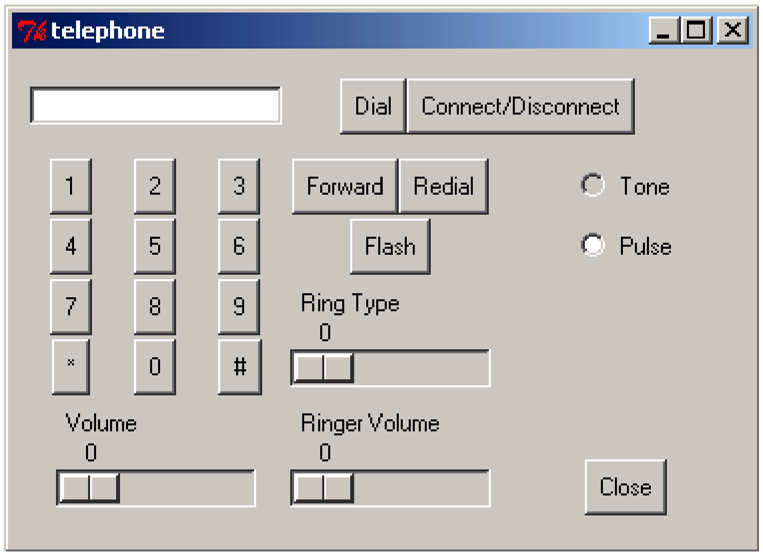
Phone 7.



Phone 8.



Phone 9.



Phone 10.

Appendix B: The ACT-R model code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Environment loading file
;;;
;;; This file contains the necessary load statements
;;; to start ACT-R and Segman.
;;;
;;;  File: env-load.lisp
;;;  Author: Andrew R. Freed
;;;  Creation date: 11/February/03
;;;  Last modified: 25/March/03
;;;
;;; This file runs under ACT-R 5.0, RPM v2.17b, Segman v3.1, and ACL 5.0.1

;; This file simply loads the ACT-R environment, with Segman.
;; Optimally, this file would be loaded on Lisp startup

(load "c:/Research/systems/testing/ACTRPM/code/+ load-rpm.lisp")
(load "c:/Research/systems/testing/segman-v3.1/load-system.lisp")
(load "c:/Research/systems/testing/segman-interface")
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Loader file
;;;
;;; This file contains the initializations necessary to
;;; start the phone dialing model.
;;;
;;;  File: loader.lisp
;;;  Author: Andrew R. Freed
;;;  Creation date: 2/July/02
;;;  Last modified: 25/March/02
;;;
;;; This file runs under ACT-R 5.0, RPM v2.17b, Segman v3.1, and ACL 5.0.1
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Table of contents
;;;
;;; i   How to run
;;; ii  Initializations
;;; iii Global variables
;;; iv  Loads
;;; I   Running a demo of the phone-dialer
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; i. How to run
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

#|
To load and run this model on eyetracker II PC in ACS Lab:

* Start up GNU emacs
* M-x IDE   will start up the Allegro + IDE system
* Minimize (but do not kill) the lisp windows you don’t need.
  typically, this is just the console window.

* from the project window, load the this file
  using the load command on the allegro menu.  do not do this by
  hand.  c:/research/systems/testing/freed_model/loader.lisp
  you may have to change the position in the file hierarchy, and you will
  have to change the file viewer file type from .fasl to *.* or *.lisp.
* to demo the phone-dialer, type to the lisp debugger window, (phone_model)
* to run the model several times, type to the debugger window,
   (collect-data n) where n is the number of times to run
* if you wish to dial a different phone, use "reset" on the control panel
    before starting the model again.
|#

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ii. Initializations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Initializes ACT-R
(clear-all)
(pm-reset)

;;; sets whether a person (nil) or the model (t) will be doing the task.
(setf *actr-enabled-p* t)

(setf mouse-fitts-coeff 0.100)  ;;standard Fitts’ coefficient

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iii. Global variables
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; sgp is the ACT-R command to Set Global Parameters
;; The ACT-R manual describes all of the available parameters.
;; er is the enable randomness flag. If it is t then
;; when multiple productions can fire a random one is
;; chosen and when it is nil a deterministic, but unspecified
;; method is used.  Note that when the subsymbolic components
;; are enabled this is much less of an issue, because then there
;; are other means of determining which production to fire.
;; v is the verbose flag and specifies whether or not the
;; trace output is displayed.  The model will run much faster
;; when the trace output is off (nil), but then you can’t see
;; what is happening along the way.

(sgp :er t :v nil)

;; pm-set-params is the command to set the Perceptual-Motor
;; parameters.  Details of all the available PM parameters are
;; available in the ACT-R manual.
;; real-time controls whether the model operates in simulated
;; time (nil) or real time (t).  If it is in simulated time it will run
;; as fast as the machine will allow, but if it is in real time
;; then it will wait (if necessary) for the appropriate amount of
;; time to pass for each action performed.

(pm-set-params :real-time nil)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; iv. Loads
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(load "c:/Research/systems/testing/freed-model/phone_model.lisp")
(load "c:/Research/systems/testing/freed-model/experiment.lisp")
(load "c:/Research/systems/testing/freed-model/segman-link.lisp")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; I. Running a demo of the phone-dialer
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun phone_model()
  (load "c:/Research/systems/testing/freed-model/experiment.lisp")
  (load "c:/Research/systems/testing/freed-model/segman-link.lisp")
  (load "c:/Research/systems/testing/freed-model/phone_model.lisp")
  (collect-data 1 "Me1")
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Experiment file
;;;
;;; This file should be generic enough that it will
;;; run any ACT-R 5.0 model, and therefore should not
;;; contain any model-specific code.
;;;
;;;  File: experiment.lisp
;;;  Author: Andrew R. Freed
;;;  Creation date: 2/July/02
;;;  Last modified: 25/March/03
;;;
;;; This file runs under ACT-R 5.0, RPM v2.17b, Segman v3.1, and ACL 5.0.1
;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Table of Contents
;;;
;;; I. Running the model
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; I. Running the model
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; collect-data(n num) is the function to be used to run the model.
;; n is a positive integer argument, the number of times the model
;; should run.  num is "Me1" up to "Me10", specifying the telephone
;; number to retrieve from memory.
;; In each run, the model will dial the telephone once.

(defun collect-data (n num)

  ;;This delay gives you enough time to bring the telephone window to
  ;;the foreground, and to note the beginning/end of model runs
  (sleep 2)

  ;;Install to ACT-R the same region Segman is parsing
  (pm-install-window ’(0 0 651 552)) ;;specific to phone 13

  ;;Initialize Segman/Phone-dialer interface
  (init-segman-for-phone)

  ;; This resets the model to its initial state.  This includes
  ;; erasing its working memory.
  (reset)

  ;;This tells ACT-R to process the visual items discovered by
  ;;Segman’s segment-screen
  (pm-proc-display)

  ;;This loop runs the model n times, prints running times, and
  ;;finally displays an average time
  (setf *score* 0)
  (setf *all-fixations* 0)
  (loop for i from 1 to n
      do
        (setf *fixations* 0)

        ;; a user trial starts with hand at mouse, and mouse at
        ;; center of screen
        (pm-start-hand-at-mouse)
        (setf (cursor-position *screen*) (make-position  512 384))

        ;;initializes the dialing goal with telephone number
        (mod-chunk phone-goal state init)
        (mod-chunk-fct ’phone-goal (list ’name num))
        (goal-focus phone-goal)

        ;;runs the model
        (do-experiment)

        ;;report time and update running counts
        (format t "~%~5,3f, ~3,0f" (- (actr-time) *score*) *fixations*)
        (setq *score* (actr-time))
        (setq *all-fixations* (+ *fixations* *all-fixations*))
  )

  (format t "~%Average time: ~5,4fs fixations: ~3,2f~%" (/ *score* n) (/ *all-
fixations* n))
)

;; do-experiment makes the actual ACT-R call to start the model
(defun do-experiment ()
  (pm-run 16)       ;; Run the model for up to n seconds
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Segman linking file
;;;
;;; This file contains the necessary Segman initializations
;;; to link Segman and the phone dialing model.
;;;
;;;  File: segman-link.lisp
;;;  Author: Andrew R. Freed
;;;  Creation date: 2/July/02
;;;  Last modified: 25/March/03
;;;
;;; This file runs under ACT-R 5.0, RPM v2.17b, Segman v3.1, and ACL 5.0.1
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Table of Contents
;;;
;;; i   Phone-specific initializations
;;; ii  Font and pattern definitions
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; i. Phone-specific initializations
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun init-segman-for-phone ()
  ;;Setup Segman to find visual features of type "text", and
  ;; the model will implicitly be able to use buttons
  (setf segman::*recognized-features* '(:text))

  ;;This sets a minimum number of characters that comprise a word
  (setf segman::*minimum-word-length* 3)

  ;;Restrict Segman to only parse a given part of the screen
  ;; this must always start at 0,0, and is manually changed for
  ;; different phones
  (segman::segment-screen (list 0 0 651 552)) ;;specific to phone file 13
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; ii. Font and pattern definitions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun group-number-p (group)
  ;; This function gets the pixel pattern (e.g., a letter of number)
  ;; associated with a group, translates it to a character, if
  ;; possible, and then makes sure that the character is a character
  ;; between #\0 and #\9
  (let ((char (segman::pattern-translation (segman::group-pattern group))))
    (and (characterp char)
         (char<= #\0 char #\9))
  )
)

;; The Segman package only comes with recognition for one standard
;; font.  Therefore I am including pattern definitions for some other
;; phone fonts.  At present I have only encoded one new font, a larger
;; version of the standard font.  The Segman help file
;; "segman-introduction.html" includes instructions on how to define
;; new patterns for Segman

;; Pattern for a large "1"
(segman::define-pattern large-one (:translation #\1)
   (:AND (:COUNT 79) (:AREA 79/176) (:SIZE 176) (:HEIGHT 22) (:WIDTH 8)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3) 7 28 30
    (62 4) 63 96 112 116 (119 2) (124 15) (127 3) 193 (199 20) 225 227 241
    243 251 252 253 (255 20))
```

```
)

;; Pattern for a large "2"
(segman::define-pattern large-two (:translation #\2)
    (:AND (:COUNT 133) (:AREA 19/44) (:SIZE 308) (:HEIGHT 22) (:WIDTH 14)
     (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 21/13) 7 (15 2)
     (30 2) (31 12) (60 4) (62 9) 63 (112 2) (119 5) (124 2) (126 3)
     (127 9) (135 2) (159 2) (193 2) (195 2) (199 4) 207 (223 4) (225 2)
    (227 6) (231 2) 239 (241 15) (243 3) (247 6) (248 2) 249 252 (253 3)
    (255 22))
)

;; Pattern for large "3"
(segman::define-pattern large-three (:translation #\3)
   (:AND (:COUNT 141) (:AREA 47/110) (:SIZE 330) (:HEIGHT 22) (:WIDTH 15)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3/2) 7 (15 3)
    (28 3) (30 2) (31 9) 60 (62 3) (63 4) (112 2) 119 (120 2) (124 4)
    (126 3) (127 7) (135 2) (143 3) (159 3) (195 3) (199 5) (207 2)
    (223 8) (225 3) (227 4) (231 2) (240 2) (241 10) (243 2) (247 8)
    (248 3) (249 3) (252 3) (253 8) (255 22))
)

;; Pattern for large "4"
(segman::define-pattern large-four (:translation #\4)
   (:AND (:COUNT 134) (:AREA 67/165) (:SIZE 330) (:HEIGHT 22) (:WIDTH 15)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3/2) (7 2) 30
    (31 5) (60 3) (62 8) 63 (112 2) (119 5) (124 11) (126 3) (127 7)
    (159 2) (193 2) 195 (199 16) 207 (223 2) (227 5) (231 2) 239 (241 9)
    243 (247 3) 249 251 (252 2) (253 2) (255 35))
)

;; Pattern for large "5"
(segman::define-pattern large-five (:translation #\5)
   (:AND (:COUNT 154) (:AREA 7/15) (:SIZE 330) (:HEIGHT 22) (:WIDTH 15)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3/2) 7 (15 2)
    (28 2) 30 (31 18) (60 2) (62 2) 63 112 115 (120 2) (124 8) (126 3)
    (127 6) 135 (143 5) 159 183 190 193 (195 2) (199 8) 203 (223 8)
    (225 3) 227 (231 2) 239 (240 2) (241 18) (243 2) (247 6) (248 2)
    (249 2) (252 2) (253 6) (255 28))
)

;; Pattern for large "6"
(segman::define-pattern large-six (:translation #\6)
   (:AND (:COUNT 170) (:AREA 17/33) (:SIZE 330) (:HEIGHT 22) (:WIDTH 15)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3/2) (15 4) 28
    (30 3) (31 10) (60 2) (62 3) (63 3) 119 (120 2) (124 11) (126 3)
    (127 8) (135 4) (143 2) (159 3) 191 193 (195 2) (199 7) (207 4)
    (223 10) 225 (227 7) (231 3) (240 3) (241 10) (243 2) (247 10) (248 5)
    (249 3) (252 2) (253 10) (255 29))
)

;; Pattern for large "7"
(segman::define-pattern large-seven (:translation #\7)
   (:AND (:COUNT 96) (:AREA 24/77) (:SIZE 308) (:HEIGHT 22) (:WIDTH 14)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 21/13) 7 28
    (31 12) (60 6) (62 2) 103 (112 2) (119 2) (124 5) (126 5) (127 5) 193
    (195 6) (199 6) (227 2) (231 5) (241 10) 246 (247 5) 249 253 (255 16))
)

;; Pattern for large "8"
(segman::define-pattern large-eight (:translation #\8)
   (:AND (:COUNT 166) (:AREA 83/165) (:SIZE 330) (:HEIGHT 22) (:WIDTH 15)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3/2) (15 3)
    (30 2) (31 10) (60 3) (62 5) (63 3) (119 3) (120 2) (124 8) (126 3)
    (127 8) (135 3) (143 4) (159 4) (195 2) (199 8) (207 3) (221 3)
    (223 8) 225 (227 8) (231 2) 239 240 (241 11) (243 2) (247 9) (248 8)
    (249 2) (252 2) (253 9) 254 (255 24))
)

;; Pattern for large "9"
(segman::define-pattern large-nine (:translation #\9)
   (:AND (:COUNT 169) (:AREA 169/330) (:SIZE 330) (:HEIGHT 22) (:WIDTH 15)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3/2) (15 3) 28
    (30 2) (31 10) (60 2) (62 5) (63 3) 119 (120 4) (124 8) (126 3)
    (127 9) (135 2) (143 5) (159 3) 193 (195 2) (199 11) (207 2) (223 10)
    (225 3) (227 3) (231 3) (240 4) (241 10) (243 3) (247 8) (248 2)
    (249 3) 251 (252 4) (253 10) (255 28))
)

;; Pattern for large "0"
(segman::define-pattern large-zero (:translation #\0)
   (:AND (:COUNT 154) (:AREA 7/15) (:SIZE 330) (:HEIGHT 22) (:WIDTH 15)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 3/2) 15 30
    (31 8) (60 3) (62 3) 63 (120 3) (124 18) (126 3) (127 7) (135 3)
    (143 3) 159 (195 3) (199 18) (207 3) (223 7) 225 (227 3) (231 3) 240
    (241 8) 243 (247 7) (248 3) 249 (252 3) (253 7) (255 30))
)

;; Pattern for large "#"
(segman::define-pattern large-pound (:translation #\#)
   (:AND (:COUNT 192) (:AREA 6/11) (:SIZE 352) (:HEIGHT 22) (:WIDTH 16)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 7/5) (7 4)
    (28 4) (31 10) (60 6) (63 4) (112 4) (124 10) (126 10) (127 10)
    (159 4) (193 4) (195 6) (199 10) (207 4) (223 4) (231 10) (241 10)
    (243 4) (247 10) (249 4) (252 4) (253 4) (255 52))
)

;; Pattern for large "*"
(segman::define-pattern large-star (:translation #\*)
   (:AND (:COUNT 48) (:AREA 12/25) (:SIZE 100) (:HEIGHT 10) (:WIDTH 10)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 1) (7 2) 15
    (28 2) 30 (31 2) 56 62 112 117 (119 2) (124 2) 126 127 131 143 (191 2)
    193 (199 2) 207 213 (221 2) 223 (224 2) (227 2) 239 (243 2) 247
    (248 2) (249 2) 253 254 (255 4))
)

;; Pattern for small "#"
(segman::define-pattern small-pound (:translation #\#)
   (:AND (:COUNT 26) (:AREA 13/27) (:SIZE 54) (:HEIGHT 9) (:WIDTH 6)
    (:RED 0) (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 8/5) (4 2)
    (56 2) (57 2) (64 2) (68 2) (78 4) (85 4) (131 2) (147 2) (228 4))
)

;; Pattern for small "*"
(segman::define-pattern small-star (:translation #\*)
   (:AND (:COUNT 5) (:AREA 5/9) (:SIZE 9) (:HEIGHT 3) (:WIDTH 3) (:RED 0)
    (:GREEN 0) (:BLUE 0) (:COLOR 0) (:PROPORTION 1) 2 8 32 128 170)
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Model file (phone dialer)
;;;
;;;  File: phone_model.lisp
;;;  Author: Andrew R. Freed
;;;  Creation date: 2/July/02
;;;  Last modified: 7/April/02
;;;
;;; This file runs under ACT-R 5.0, RPM v2.17b, Segman v3.1, and ACL 5.0.1
;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Table of Contents
;;;
;;; i   Declarative memory - chunk types
;;; ii  Declarative memory - memory chunks
;;; iii Smart search parameters
;;; iv  Productions to the phone-dialer

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;     i.      declarative memory - chunk types
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; These types define the two kinds of goals used in this model.
;; dial-phone-num is a phone dialing goal and find-feature is a visual
;; searching goal
(chunk-type dial-phone-num state name place current-num click)

;; These types represent other declarative memories the model can have,
;; from simple digits ("number") to the various components of a phone
;; number
(chunk-type number value)
(chunk-type area-code name digit1 digit2 digit3)
(chunk-type exchange name digit1 digit2 digit3)
(chunk-type extension name digit1 digit2 digit3 digit4)
(chunk-type phone-number name area-code exchange extension)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;     ii.     declarative memory - memory chunks
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; add-dm puts these chunks into declarative memory
(add-dm

   ;; Phone numbers are loaded into memory
   (my1_area_code ISA area-code name "Me1" digit1 "8" digit2 "1" digit3 "4")
   (my1_exchange ISA exchange name "Me1" digit1 "8" digit2 "6" digit3 "6")
   (my1_extension ISA extension name "Me1" digit1 "5" digit2 "0" digit3 "0" digit4
"0")
   (my1_phone_num ISA phone-number name "Me1" area-code my1_area_code exchange
                my1_exchange extension my1_extension)
   (my2_area_code ISA area-code name "Me2" digit1 "1" digit2 "2" digit3 "3")
   (my2_exchange ISA exchange name "Me2" digit1 "6" digit2 "5" digit3 "4")
   (my2_extension ISA extension name "Me2" digit1 "7" digit2 "8" digit3 "9" digit4
"0")
   (my2_phone_num ISA phone-number name "Me2" area-code my2_area_code exchange
                my2_exchange extension my2_extension)
   (my3_area_code ISA area-code name "Me3" digit1 "8" digit2 "1" digit3 "4")
   (my3_exchange ISA exchange name "Me3" digit1 "8" digit2 "6" digit3 "3")
   (my3_extension ISA extension name "Me3" digit1 "5" digit2 "0" digit3 "0" digit4
"0")
   (my3_phone_num ISA phone-number name "Me3" area-code my3_area_code exchange
                my3_exchange extension my3_extension)
   (my4_area_code ISA area-code name "Me4" digit1 "4" digit2 "1" digit3 "2")
   (my4_exchange ISA exchange name "Me4" digit1 "2" digit2 "6" digit3 "8")
   (my4_extension ISA extension name "Me4" digit1 "3" digit2 "0" digit3 "0" digit4
"0")
```

```
   (my4_phone_num ISA phone-number name "Me4" area-code my4_area_code exchange
                my4_exchange extension my4_extension)
   (my5_area_code ISA area-code name "Me5" digit1 "6" digit2 "0" digit3 "6")
   (my5_exchange ISA exchange name "Me5" digit1 "1" digit2 "9" digit3 "3")
   (my5_extension ISA extension name "Me5" digit1 "3" digit2 "0" digit3 "1" digit4
"2")
   (my5_phone_num ISA phone-number name "Me5" area-code my5_area_code exchange
                my5_exchange extension my5_extension)
   (my6_area_code ISA area-code name "Me6" digit1 "2" digit2 "1" digit3 "5")
   (my6_exchange ISA exchange name "Me6" digit1 "6" digit2 "5" digit3 "4")
   (my6_extension ISA extension name "Me6" digit1 "5" digit2 "7" digit3 "8" digit4
"5")
   (my6_phone_num ISA phone-number name "Me6" area-code my6_area_code exchange
                my6_exchange extension my6_extension)
   (my7_area_code ISA area-code name "Me7" digit1 "8" digit2 "1" digit3 "4")
   (my7_exchange ISA exchange name "Me7" digit1 "2" digit2 "3" digit3 "4")
   (my7_extension ISA extension name "Me7" digit1 "9" digit2 "6" digit3 "5" digit4
"7")
   (my7_phone_num ISA phone-number name "Me7" area-code my7_area_code exchange
                my7_exchange extension my7_extension)
   (my8_area_code ISA area-code name "Me8" digit1 "7" digit2 "4" digit3 "0")
   (my8_exchange ISA exchange name "Me8" digit1 "6" digit2 "1" digit3 "1")
   (my8_extension ISA extension name "Me8" digit1 "9" digit2 "2" digit3 "7" digit4
"3")
   (my8_phone_num ISA phone-number name "Me8" area-code my8_area_code exchange
                my8_exchange extension my8_extension)
   (my9_area_code ISA area-code name "Me9" digit1 "1" digit2 "0" digit3 "1")
   (my9_exchange ISA exchange name "Me9" digit1 "0" digit2 "1" digit3 "0")
   (my9_extension ISA extension name "Me9" digit1 "1" digit2 "0" digit3 "1" digit4
"0")
   (my9_phone_num ISA phone-number name "Me9" area-code my9_area_code exchange
                my9_exchange extension my9_extension)
   (my10_area_code ISA area-code name "Me10" digit1 "1" digit2 "0" digit3 "3")
   (my10_exchange ISA exchange name "Me10" digit1 "2" digit2 "7" digit3 "3")
   (my10_extension ISA extension name "Me10" digit1 "1" digit2 "0" digit3 "2" digit4
"9")
   (my10_phone_num ISA phone-number name "Me10" area-code my10_area_code exchange
                my10_exchange extension my10_extension)

   ;; Finally, a goal to dial a telephone number is placed in memory
   (phone-goal isa dial-phone-num state init name "Me1" click no)
  )

;;;
;;;     iii.    smart search parameters
;;;

;; This section defines the parameters that help us create systematic visual search
;; while never fixating off of the keypad after fixating within the keypad

;; these two parameters give us "tunnel vision" in one direction
;; for instance, x-column means to only consider the column that
;; has x+-15 to current x
(defparameter *x-column* 15)
(defparameter *y-column* 15)

;;to make sure we actually move up, down, left, or right, we force
;; movement of "forward-xxx" pixels in the desired direction
(defparameter *forward-width* 10)
(defparameter *forward-height* 10)

;;should be set to the width and height of buttons on the phone, respectively
;;these are phone-specific and should be over-estimated slightly
;;they help the model move approximate distance from one target to another
(defparameter *one-width* 100) ;37
(defparameter *one-height* 70) ;41
(defparameter *two-width* (* 2 *one-width*))
(defparameter *two-height* (* 2 *one-height*))
(defparameter *three-height* (* 3 *one-height*))
```

```
;;;
;;;    iv.    Productions to Phone dialer
;;;

;;; These productions work in ACT-R 5.0 and RPM v2.17b and Segman v3.1

;;; There are two types of productions here, those trying to solve the
;;; goal of dialing a phone (type: dial-phone-num) and those of finding
;;; a feature on the phone (type: find-feature).  find-feature is a
;;; subgoal of dial-phone-num, meaning that while working on a
;;; dial-phone-num, a find-feature will have to be performed.  After
;;; the find-feature completes, the dial-phone-num resumes.  Note that
;;; one find-feature will be performed for each digit in the phone
;;; number

;;; Within each goal, there is an expected pattern of production
;;; firings.
;;;   dial-phone-num:
;;;     Plan A: The first time through the model after a "reset"
;;;       1) move-hand-to-phone
;;;       2-4) dial-area-code-n (for n=1 to 3)
;;;       5-7) dial-exchange-n (for n=1 to 3)
;;;       8-11) dial-extension-n (for n=1 to 4)
;;;       12) phone-done

;; first, move hand to mouse and retrieve phone number
;; According to Chong 98 (thesis) this is taking advantage
;; of "movement pre-positioning".  See also (Wood, Kieras,
;; & Meyer 1994; Kieras, Wood & Meyer 1995a, 1995b)

;; The phone number is indexed in memory on "name"
;; "place" denotes which place digit is to be dialed next
(p start-dialing
   =goal>         ISA   dial-phone-num
                  state init
                  name  =name
 ==>
   +retrieval>    ISA   area-code
                  name  =name
   =goal> state start
          place "1"  )

;; For each dial-*-* production, the following occur:
;;  a) the type of goal is confirmed (dial-phone-num)
;;  b) the place is checked (e.g. only dial 5th digit when place=5)
;;  c) the phone number segment (area code, exchange, etc) and
;;     the correct digit within that segment are present in WM
;;  d) a sub-goal (type: find-feature) is created to find and click on
;;     the proper digit, using the !push! command
;;  e) when the sub-goal is finished, place is updated

;; For each dial-*-1 production, an additional check is performed, to
;; ensure that the segment to be dialed next is a digit of the original
;; phone number in the goal

;;This production checks the result of the phone number retrieval, and
;;sets a subgoal to dial the first number
(p dial-area-code-1
   =goal>   ISA dial-phone-num
            state start
            name =name
            place "1"
   =retrieval> ISA area-code
               digit1 =y
               name   =name
==>
   =goal> place "2"
          state find-number
          current-num =y)
```

```
(p dial-area-code-2
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "2"
   =retrieval> ISA area-code
               digit2 =y
               name   =name
==>
   =goal> place "3"
          state attend-number
          current-num =y)

(p dial-area-code-3
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "3"
   =retrieval> ISA area-code
               digit3 =y
               name   =name
==>
   +retrieval> ISA exchange
               name =name
   =goal> place "4"
          state attend-number
          current-num =y)

(p dial-exchange-1
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "4"
   =retrieval> ISA exchange
               digit1 =y
               name =name
==>
   =goal> place "5"
          state attend-number
          current-num =y)

(p dial-exchange-2
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "5"
   =retrieval> ISA exchange
              digit2 =y
==>
   =goal> place "6"
          state attend-number
          current-num =y)

(p dial-exchange-3
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "6"
   =retrieval> ISA exchange
              digit3 =y
              name =name
==>
   +retrieval> ISA extension
               name =name
   =goal> place "7"
          state attend-number
          current-num =y)

(p dial-extension-1
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "7"
   =retrieval> ISA extension
               digit1 =y
               name =name
==>
   =goal> place "8"
          state attend-number
          current-num =y)

(p dial-extension-2
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "8"
   =retrieval> ISA extension
               digit2 =y
               name =name
==>
   =goal> place "9"
          state attend-number
          current-num =y)

(p dial-extension-3
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "9"
   =retrieval> ISA extension
               digit3 =y
               name =name
==>
   =goal> place "10"
          state attend-number
          current-num =y)

(p dial-extension-4
   =goal>   ISA dial-phone-num
          - state attend-number
          - state find-number
          - state find-number-directed-search
          - state find-number-location
            place "10"
   =retrieval> ISA extension
               digit4 =y
               name =name
==>
   =goal> place "11"
          state attend-number
          current-num =y)

;;; We take advantage of buffer-stuffing, so that when a "find-goal"
;;; is created, it already has visual-location to examine
;;; Within each goal, there is an expected pattern of production
;;; firings.
;;;   find-goal:
;;;       attend-number
;;;       SUCCESS?:
;;;           S-1) found-number-move-to-button
;;;           S-2) left-click-number
;;;       FAILURE?:
;;;           (execute one of these cases and return to attend-number)
;;;           F-1a) wrong-item (unknown target)
;;;                 fires: find-unattended-number
;;;                    OR
;;;           F-1b) wrong-number
;;;                 fires the appropriate "smart" search
;;;                 smart search productions have the following form:
;;;                   smart-search-found-X-go-Y
;;;                     where
;;;                           X is the digit found (0-9) or * or #
;;;                           Y is one of the following:
;;;                             "up", "down", "left", "right",
;;;                             "up-left", "up-right",
;;;                             "down-left", "down-right"
;;;                     for adjacent digits, "-to-neighbor" is appended
;;;
;;;                 for example, if we attended "1" while looking for "5"
;;;                 smart-search-found-1-go-down-right
;;;
;;;                 smart-search-found-1-go-right-to-neighbor
;;;                 goes from the 1 to the 2
;;;
;;; the "goal" is accomplished when left-click-number fires
;;;


;; To get object in visual-location into the visual buffer, we must
;; request the vision module to encode the object (+visual> section).
(P attend-number
   =goal>  ISA   dial-phone-num
           state find-number-location
   =visual-location>  ISA  visual-location
   =visual-state>  ISA  module-state
                   modality    free
==>
   +visual>  ISA     visual-object
             screen-pos  =visual-location
   !eval!  (setf *fixations* (+ 1 *fixations*)) ;;running score
   =goal>  state  attend-number)

;;fix error
(P no-location-to-attend
   =goal>  ISA   dial-phone-num
           state find-number-location
   =visual-location>  ISA  ERROR
==>
   =goal> state find-number
   !output!    (*** wrong-item ERROR ***)
)

;; This fires during a visual search, when it is determined that the
;; object in the visual buffer is not the item that is being searched
;; for.  This is reported using the !output! statement, then the model
;; resumes the visual search, starting at the next unvisited object
;; from the left
(P wrong-item
   =goal> ISA   dial-phone-num
         state  attend-number
         current-num =y
   =visual> ISA text   ;;we are searching for =y but have instead found =z
         - value  =y   ;;The value we were hoping to find but did NOT
           value  =z   ;;The value actually found

   ;;make sure that the item found is not a digit, *, or #
```

```
       !eval! (and (string/= =z "#") (string/= =z "*") (or (string< =z "0") (string<=
"a" =z)

(not(= (c-types:strlen =z) 1)) ))
==>
   =goal> state find-number
   !output!    (*** wrong-item =z ***)
)

;;; What if after wrong-number you searched intelligently based on what
;;; you saw.  i.e. after seeing "1" you looked to the right for "2", "3".
;;; This will catch a sighting of a number and try to direct the new search
(P wrong-number
   =goal> ISA   dial-phone-num
         state  attend-number
         current-num =y
   =visual> ISA text   ;;we are searching for =y but have instead found =z
         - value  =y   ;;The value we were hoping to find but did NOT
           value  =z

   ;;make sure we've spotted a recognizable phone feature
    !eval! (or (and (string<= "0" =z) (string<= =z "9") (= (c-types:strlen =z) 1)) (or

(string= =z "#") (string= =z "*") ) )
==>
   =goal> state find-number-directed-search
   !output!    (*** wrong-number =z ***)
)

;; This function performs a random search of the phone surface area.  It
;; looks for a feature in a visual-location that is not
;; encoded in visual memory (attended nil).
;; It searches only for text objects (kind text)
(P find-unattended-number
   =goal>    ISA    dial-phone-num
             state  find-number
 ==>
   +visual-location>  ISA        visual-location
                      attended   nil
                      kind text
   =goal>   state  find-number-location)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;; BEGIN of smart finds
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Searching for 1 and decided to go right
(P smart-search-found-1-go-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "1"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "3") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *two-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 1 and decided to go right to 2
(P smart-search-found-1-go-right-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
```

```
            value "1"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "2") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 1 and decided to go down
(P smart-search-found-1-go-down
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "1"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "7") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *two-height*))
   =goal> state find-number-location)

;;; Searching for 1 and decided to go down to 4
(P smart-search-found-1-go-down-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "1"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "4") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 1 and decided to go down-right
(P smart-search-found-1-go-down-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "1"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") (string= =y "5") (string= =y "9") (string= =y "8")
(string= =y

"0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *two-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *three-height*))
   =goal> state find-number-location)

;;; Searching for 2 and decided to go right
(P smart-search-found-2-go-right-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "2"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "3") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 2 and decided to go left
(P smart-search-found-2-go-left-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "2"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "1") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 2 and decided to go down
(P smart-search-found-2-go-down
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "2"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "8") (string= =y "0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *three-height*))
   =goal> state find-number-location)

;;; Searching for 2 and decided to go down
(P smart-search-found-2-go-down-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "2"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "5") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 2 and decided to go down-right
```

```
(P smart-search-found-2-go-down-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "2"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") (string= =y "9") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *two-height*))
   =goal> state find-number-location)

;;; Searching for 2 and decided to go down-left
(P smart-search-found-2-go-down-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "2"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "4") (string= =y "7") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *two-height*))
   =goal> state find-number-location)

;;; Searching for 3 and decided to go left
(P smart-search-found-3-go-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "3"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "1") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *two-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 3 and decided to go left
(P smart-search-found-3-go-left-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "3"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "2") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)
```

```
;;; Searching for 3 and decided to go down
(P smart-search-found-3-go-down
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "3"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "9") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *two-height*))
   =goal> state find-number-location)

;;; Searching for 3 and decided to go down
(P smart-search-found-3-go-down-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "3"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)


;;; Searching for 3 and decided to go down-left
(P smart-search-found-3-go-down-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "3"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "4") (string= =y "5") (string= =y "7") (string= =y "8")
(string= =y

"0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *two-width*) (- =sx *forward-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *three-height*))
   =goal> state find-number-location)

;;; Searching for 4 and decided to go right
(P smart-search-found-4-go-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "4"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") )
==>
   +visual-location> ISA visual-location
```

```
                  kind text
                  screen-x (within (+ =sx *forward-width*) (+ =sx *two-width*))
                  screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 4 and decided to go right
(P smart-search-found-4-go-right-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "4"
   =visual-location> ISA visual-location
                  screen-x =sx
                  screen-y =sy
   !eval! (or (string= =y "5") )
==>
   +visual-location> ISA visual-location
                  kind text
                  screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                  screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)


;;; Searching for 4 and decided to go up
(P smart-search-found-4-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "4"
   =visual-location> ISA visual-location
                  screen-x =sx
                  screen-y =sy
   !eval! (or (string= =y "1") )
==>
   +visual-location> ISA visual-location
                  kind text
                  screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                  screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 4 and decided to go down
(P smart-search-found-4-go-down-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "4"
   =visual-location> ISA visual-location
                  screen-x =sx
                  screen-y =sy
   !eval! (or (string= =y "7") )
==>
   +visual-location> ISA visual-location
                  kind text
                  screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                  screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 4 and decided to go up-right
(P smart-search-found-4-go-up-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "4"
   =visual-location> ISA visual-location
                  screen-x =sx
                  screen-y =sy
   !eval! (or (string= =y "3") (string= =y "2") )
```

```
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *two-width*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 4 and decided to go down-right
(P smart-search-found-4-go-down-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "4"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "9") (string= =y "8") (string= =y "0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *two-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *two-height*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go right
(P smart-search-found-5-go-right-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go left
(P smart-search-found-5-go-left-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "4") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go up
(P smart-search-found-5-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "2") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go down
(P smart-search-found-5-go-down
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *two-height*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go down
(P smart-search-found-5-go-down-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "8") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go up-right
(P smart-search-found-5-go-up-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "3") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go up-left
(P smart-search-found-5-go-up-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                     screen-x =sx
```

```
                    screen-y =sy
   !eval! (or (string= =y "1") )
==>
   +visual-location> ISA visual-location
                    kind text
                    screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                    screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go down-right
(P smart-search-found-5-go-down-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                    screen-x =sx
                    screen-y =sy
   !eval! (or (string= =y "9") )
==>
   +visual-location> ISA visual-location
                    kind text
                    screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                    screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 5 and decided to go down-left
(P smart-search-found-5-go-down-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "5"
   =visual-location> ISA visual-location
                    screen-x =sx
                    screen-y =sy
   !eval! (or (string= =y "7") )
==>
   +visual-location> ISA visual-location
                    kind text
                    screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                    screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 6 and decided to go left
(P smart-search-found-6-go-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "6"
   =visual-location> ISA visual-location
                    screen-x =sx
                    screen-y =sy
   !eval! (or (string= =y "4") (string= =y "5") )
==>
   +visual-location> ISA visual-location
                    kind text
                    screen-x (within (- =sx *two-width*) (- =sx *forward-width*))
                    screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 6 and decided to go up
(P smart-search-found-6-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "6"
   =visual-location> ISA visual-location
```

```
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "3") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 6 and decided to go down
(P smart-search-found-6-go-down-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "6"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "9") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 6 and decided to go up-left
(P smart-search-found-6-go-up-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "6"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "1") (string= =y "2") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *two-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 6 and decided to go down-left
(P smart-search-found-6-go-down-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "6"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "7") (string= =y "8") (string= =y "0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *two-width*) (- =sx *forward-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *two-height*))
   =goal> state find-number-location)

;;; Searching for 7 and decided to go right
(P smart-search-found-7-go-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "7"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "9") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *two-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 7 and decided to go right
(P smart-search-found-7-go-right-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "7"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "8") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 7 and decided to go up
(P smart-search-found-7-go-up
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "7"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "1") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *two-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 7 and decided to go up
(P smart-search-found-7-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "7"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "4") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 7 and decided to go up-right
(P smart-search-found-7-go-up-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
```

```
            value "7"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") (string= =y "2") (string= =y "3") (string= =y "5"))
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *two-width*))
                     screen-y (within (- =sy *two-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 7 and decided to go down-right
(P smart-search-found-7-go-down-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "7"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 8 and decided to go right
(P smart-search-found-8-go-right-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "8"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "9") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 8 and decided to go left
(P smart-search-found-8-go-left-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "8"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "7") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 8 and decided to go up
(P smart-search-found-8-go-up
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "8"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "2") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *two-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 8 and decided to go up
(P smart-search-found-8-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "8"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "5"))
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 8 and decided to go up-right
(P smart-search-found-8-go-up-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "8"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") (string= =y "3") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *two-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 8 and decided to go up-left
(P smart-search-found-8-go-up-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "8"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "4") (string= =y "1") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *two-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 8 and decided to go down
(P smart-search-found-8-go-down-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "8"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 9 and decided to go left
(P smart-search-found-9-go-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "9"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "7") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *two-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 9 and decided to go left
(P smart-search-found-9-go-left-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "9"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "8") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *y-column*) (+ =sy *y-column*))
   =goal> state find-number-location)

;;; Searching for 9 and decided to go up
(P smart-search-found-9-go-up
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "9"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "3") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *two-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 9 and decided to go up
(P smart-search-found-9-go-up-to-neighbor
   =goal> ISA  dial-phone-num
```

```
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "9"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "6") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 9 and decided to go up-left
(P smart-search-found-9-go-up-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "9"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "4") (string= =y "5") (string= =y "1") (string= =y "2") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *two-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *two-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 9 and decided to go down-left
(P smart-search-found-9-go-down-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "9"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "0") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (+ =sy *forward-height*) (+ =sy *one-height*))
   =goal> state find-number-location)

;;; Searching for 0 and decided to go up
(P smart-search-found-0-go-up
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "0"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "5") (string= =y "2") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *three-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 0 and decided to go up
(P smart-search-found-0-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "0"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "8") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 0 and decided to go up-right
(P smart-search-found-0-go-up-right
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "0"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "9") (string= =y "6") (string= =y "3") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (+ =sx *forward-width*) (+ =sx *one-width*))
                     screen-y (within (- =sy *three-height*) (- =sy *forward-height*))
   =goal> state find-number-location)

;;; Searching for 0 and decided to go up-left
(P smart-search-found-0-go-up-left
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "0"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
   !eval! (or (string= =y "7") (string= =y "4") (string= =y "1") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *one-width*) (- =sx *forward-width*))
                     screen-y (within (- =sy *three-height*) (- =sy *forward-height*))
   =goal> state find-number-location)


;;; Searching for * and decided to go up
(P smart-search-found-star-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "*"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
;   !eval! (or (string= =y "8") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)
```

```
;;; Searching for # and decided to go up
(P smart-search-found-pound-go-up-to-neighbor
   =goal> ISA  dial-phone-num
         state find-number-directed-search
         current-num =y
   =visual> ISA text
            value "#"
   =visual-location> ISA visual-location
                     screen-x =sx
                     screen-y =sy
;   !eval! (or (string= =y "8") )
==>
   +visual-location> ISA visual-location
                     kind text
                     screen-x (within (- =sx *x-column*) (+ =sx *x-column*))
                     screen-y (within (- =sy *one-height*) (- =sy *forward-height*))
   =goal> state find-number-location)



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;; END of smart finds
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;; Found the number, now initiate the move and click sequence
(P found-number-move-to-button
   =goal> ISA dial-phone-num
         state attend-number
          current-num =y
          click no
   =manual-state> ISA module-state
                  modality free
   =visual> ISA text
            value  =y
            screen-pos =visual-location
==>
   +manual>  isa move-cursor
             loc =visual-location
   !output!       (**** Moving to =y ****)
   =goal> click yes
           state start)
(spp found-number-move-to-button :effort 0)



;;Once the mouse is moved to the number, we want to click on the
;;button, and make the request of the motor module.  remove the
;;find-feature subgoal from the goal stack, thereby returning the model
;;to the phone-dialing goal
(P left-click-number
   =goal> ISA dial-phone-num
          click yes
   =manual-state> ISA module-state
                  modality free
   ==>
   +manual> ISA click-mouse
   !output! (**** clicked ****)
   =goal> click no)
(spp left-click-number :P 1 :C 0 :efforts 10000 :successes 10000 :effort 0)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Model file (other telephone tasks)
;;;
;;;  File: phone_model_targets.lisp
;;;  Author: Andrew R. Freed
;;;  Creation date: 2/July/02
;;;  Last modified: 31/March/02
;;;
;;; This file runs under ACT-R 5.0, RPM v2.17b, Segman v3.1, and ACL 5.0.1
;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Table of Contents
;;;
;;; i   Declarative memory - chunk types
;;; ii  Declarative memory - memory chunks
;;; iii Productions to the target-dialer

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;     i.      declarative memory - chunk types
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; These types define the two kinds of goals used in this model.
;; dial-phone-num is a phone dialing goal and find-feature is a visual
;; searching goal
(chunk-type dial-phone-target state place name)
(chunk-type find-feature state current-num)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;     ii.     declarative memory - memory chunks
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; add-dm puts these chunks into declarative memory
(add-dm
   (phone-goal isa dial-phone-target state init)
 )

;;;
;;;     iii.    Productions to target dialer
;;;

;;; These productions work in ACT-R 5.0 and RPM v2.17b and Segman v3.1

;; The phone target is keyed in memory on "name"
;; "place" denotes which place digit is to be dialed next
(p move-hand-to-phone
   =goal>         ISA   dial-phone-target
                  state init
 ==>
   =goal> state start
          place "1"  )

(p dial-target-button
   =goal>   ISA dial-phone-target
            state start
            place "1"
            name =y
==>
   =find-goal> ISA find-feature
               state find-number
               current-num =y
   =goal> place "2"
   !push! =find-goal   )

;;The final production to dial the phone, resets the phone goal
(p phone-done
   =goal>   ISA dial-phone-target
```

```
            state start
            place "2"
==>
   =goal>   state finished_dialing
            place "1"
)

;; To get object in visual-location into the visual buffer, we must
;; request the vision module to encode the object (+visual> section).
(P attend-item
   =goal>  ISA   find-feature
           state find-number-location
   =visual-location>  ISA  visual-location
   =visual-state>  ISA  module-state
                   modality    free
==>
   +visual>  ISA     visual-object
             screen-pos  =visual-location
   !eval!  (setf *fixations* (+ 1 *fixations*)) ;;running score
   =goal>  state  attend-number)


;; This fires during a visual search, when it is determined that the
;; object in the visual buffer is not the item that is being searched
;; for.  This is reported using the !output! statement, then the model
;; resumes the visual search, starting at the next unvisited object
;; from the left
(P wrong-item
   =goal> ISA   find-feature
         state  attend-number
         current-num =y
   =visual> ISA text   ;;we are searching for =y but have instead found =z
         - value  =y   ;;The value we were hoping to find but did NOT
           value  =z   ;;The value actually found
==>
   =goal> state find-number
   !output!    (*** wrong-item =z ***)
)


;; This function performs a "sweep" of the phone surface area.  It
;; looks for a feature in a visual-location that has not been
;; visited/encoded yet (attended nil).
;; It searches only for text objects (kind text)
(P find-unattended-number
   =goal>    ISA    find-feature
             state  find-number
 ==>
   +visual-location>  ISA        visual-location
                      attended   nil
                      kind text
   =goal>   state  find-number-location)


;; Found the target, now initiate the move and click sequence
(P found-target-move-to-button
   =goal> ISA find-feature
          state attend-number
          current-num =y
   =manual-state> ISA module-state
                  modality free
   =visual> ISA text
            value  =y
            screen-pos =visual-location
==>
   +manual>  isa move-cursor
             loc =visual-location
   !output!       (**** Moving to =y ****)
   =goal>  state  click-number)

;;Once the mouse is moved to the number, we want to click on the
```

```
;;button, and make the request of the motor module.  !pop! removes the
;;find-feature subgoal from the goal stack, thereby returning the model
;;to the phone-dialing goal
(P left-click-number
   =goal> ISA find-feature
          state click-number
   =manual-state> ISA module-state
                  modality free
   ==>
   +manual> ISA click-mouse
   !output! (**** clicked ****)
   !pop!   )
```