ARMY RESEARCH LABORATORY

Developing a Predictive Model of Dual Task Performance

Troy D. Kelley and David R. Scribner

ARL-MR-0556 **September 2003**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturers' or trade names does not constitute an official endorsement or approval of the use thereof.

DESTRUCTION NOTICE¾Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5425

ARL-MR-0556 September 2003

Developing a Predictive Model of Dual Task Performance

Troy D. Kelley and David R. Scribner
Human Research & Engineering Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT DATE	3. DATES COVERED (From - To)
September 2003	Final	

4. TITLE AND SUBTITLE

Developing a Predictive Model of Dual Task Performance

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
61102A

6. AUTHOR(S)

Kelley, T.D.; Scribner, D.R. (both of ARL)

5d. PROJECT NUMBER
74A

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
Human Research & Engineering Directorate
Aberdeen Proving Ground, MD 21005-5425

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-MR-0556

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Predictive models of dual task performance were developed with the cognitive architecture, Atomic Components of Thought – Rational (ACT-R), as well as the Improved Research Integration Tool (IMPRINT). The tasks modeled included a shooting task and a mathematical comprehension task, with the shooting task designated as primary. The models were designed to predict one specific type of error performance during the shooting task; in this study, errors were defined as enemy targets missed (ETM). ETM was defined as an enemy target that "popped up" but was never engaged. Preliminary results have generated ETM predictions via ACT-R and IMPRINT; these results will be validated by future studies.

15. SUBJECT TERMS

cognitive modeling dual task performance workload

16. SECURITY CLASSIFICATION OF			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Troy D. Kelley
Unclassified	Unclassified	Unclassified	UL	70	19b. TELEPHONE NUMBER *(Include area code)* 410-278-5859

Contents

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK

1. Introduction

As more and more information becomes available about the battlefield, the soldier of tomorrow will be expected to integrate and synthesize various bits of information into a meaningful format which will (it is hoped) allow the soldier to achieve maximum battlefield effectiveness. The infantry soldier is critical to the United States in achieving battlefield dominance, and at the same time, the infantry soldier is becoming more responsible for organizing information into a coherent plan of attack. Studying how the soldier interprets and synthesizes information while he does other, more traditional infantry duties (i.e., shooting a rifle) will help us understand how best to present information and how to plan system design activities.

Cognitive workload and dual task performance have been studied extensively by a number of researchers, some of whom have developed influential theories about the subject matter (Atkinson, Hernstein, Lindzey, & Luce, 1988; Damos, 1991; Gopher & Donchin, 1986; Meyer & Kornblum, 1993). However, there is still no cohesive scientific theoretical viewpoint. One of the most recent theories, Multiple Resource Theory (MRT) (Wickens, 1984), has been criticized as not being predictive enough because it "lacks principled constraints" (Neumann, 1987). Indeed, many theoretical frameworks lack predictive power, which is more likely to be afforded to a computational cognitive architecture.

The goal of this project was to develop a predictive computational cognitive model that would predict human error performance in future data gathered by the Human Research and Engineering Directorate of the U.S. Army Research Laboratory (ARL). Data from a study about shooting performance under cognitive load (aurally presented math task) were used in the baseline computational model. A future study was planned to change the condition of presentation of the math problems to a visual instead of an aural modality. Errors in shooting performance (i.e., shooting at a friendly target or not shooting at an enemy target) made during the visual presentation of math problems would be the predictive condition for the cognitive model.

Models were developed in Atomic Components of Thought - Rational (ACT-R) (Anderson & Lebiere, 1998) and the Improved Performance Research and Integration Tool (IMPRINT) (Allender et al., 1995). ACT-R is freely available for Government and academic research from Carnegie Mellon University. It is a symbolic production system architecture that is capable of low-level representations of memory structures. ACT-R is implemented in the common Lisp programming language as a collection of Lisp functions and subroutines that can be accessed by the cognitive modeler. For this project, we used Macintosh common Lisp and ACT-R 4.0 running on a G4 Apple Macintosh computer with operating system 9.2. IMPRINT is a stochastic, task network modeling tool developed by ARL for analyzing system design alternatives; it simulates how task time and task accuracy data potentially affect overall mission

performance. For the IMPRINT analysis, we used a Dell OptiPlex GX 400 running Windows 98.

2. Procedure

The data used for the ACT-R model came from a dual task experiment completed by ARL (Scribner & Harper, 2001). Soldiers completed two tasks simultaneously. The primary task was shooting and the secondary task was mathematical.

The shooting task consisted of a 24-target "pop-up" scenario using friendly (gray or white circular marking on the chest of the target) and enemy (olive drab green) E-type silhouette targets. Half of the targets were friendly and half were enemy. Targets were situated at distances of 50, 100, 150, 200, 250, and 300 meters. Target exposure times were 4, 3, and 2 seconds. M16A2 rifles with iron sights were used for the original study. The mathematical tasks consisted of addition problems, presented through earphones, which the soldier processed and responded to before he heard a completion tone. A moderate level of mathematical problem solving was chosen because of its workload, sensitivity, and perceived difficulty characteristics. The problems consisted of adding double-digit and single-digit numbers.

3. ACT-R Model Development

MRT, as presented by Wickens (1984), concentrates on four aspects of resources that are allocated to every task: stages, responses, modalities, and codes. Stages here refer to the area (encoding, central processing, or responding) where the workload is occurring. Responses refer to the type of output (manual or vocal). Modalities refer to visual or auditory, and codes refer to spatial or verbal codes. Whereas these four aspects of resources might indicate an overlap in one resource or another (in terms of modality, codes, responses, or stages), the four stages give no indication of time sharing across tasks.

Eight ACT-R models were developed for this project: a simulation of the 4-, 3-, and 2-second exposure times of the targets with audio presentation of the secondary task; a simulation of 4-, 3-, and 2-second exposure times with no cognitive load (i.e., no presentation of a secondary task) and two predictive models (one for 2 seconds and one for 4 seconds) of the visual condition (i.e., math problems presented visually). The models were developed to simulate an average soldier in a typical condition (i.e. there was not one model developed for each subject). The models were developed to run for 120 seconds, which was equal to one condition for each subject. The cognitive architecture ACT-R allowed for a simulated outside world, presenting either targets or

math problems to the model at the same time intervals as indicated by the experimental data presented in Scribner and Harper (2001).

It was hypothesized that the primary factor affecting the error rates would be the amount of time the target was available for cognitive processing (i.e., target exposure). Therefore, one of the most important aspects that the models had to simulate was the amount of time that each target was available for viewing. Once this was identified, the model architecture assigned a value (similar to memory decay) to the total amount of viewing time.

In order to simulate the perceptual aspects of the outside world, the models used "mod-chunk" calls (see Appendix A). Mod-chunk calls had been used successfully in other models to simulate the outside world (Kelley, Patton, & Allender, 2001). A mod-chunk call modifies a declarative memory element at a specific time, which would be analogous to viewing new information at a specific time. The models simulated a complete 120-second condition. At appropriate times, which were consistent with each experimental condition, a mod-chunk call was used to simulate the outside world with the presentation of either targets or math problems. Since the shooting task was considered the primary task, the model defaulted to completing the shooting task over the mathematical calculations.

Certain variables were ignored for the model development because these were considered primarily perceptual variables¾something that ACT-R is just now beginning to model in a detailed, principled fashion (Byrne & Anderson, 2001). Implementation of the perceptual variables within the models created some difficulties since the task obviously has a large perceptual component. In the end, many of the perceptual aspects of the task were handled as simply as possible. For example, the models did not include distinctions between gray targets and white targets or implementation of target distances as variables. Both friendly target color (gray or white) and target distances were balanced evenly within each condition, so it was considered a constant variable and was not modeled directly. For instance, only one type of friendly target signature (gray or white) was presented during each 120-second run. The models were essentially presented as friendly and enemy targets, making no distinction between friendly gray targets and friendly white targets. In order to model the different target signatures, another separate model incorporating gray and white target signatures could have been created with an appropriate variable corresponding to target signatures; however, this is beyond the scope of the current implementations of ACT-R.

One of the most important variables was the timing of target presentation and presentation of each math problem. The models had to capture the asynchrony between the targets and the math problems being presented aurally. For example, for each target exposure time (2 and 4 seconds), the presentation of the math problems was kept at a constant 6 seconds. The time allotted for the simulation presentation was approximately 2.2 seconds, depending on the problem, and the time allotted for the response was approximately 2.8 seconds. A tone was also presented (to indicate when the response time expired) and some dead time as well, all of which totaled approximately

6 seconds. Therefore, the response time was kept at a constant 6 seconds, while the friendly and enemy target presentation times varied, depending on the condition. This led to different lengths of time between target presentation and math problem presentation for each of the exposure conditions. Also, depending on the condition, the target and the math problems were presented at the same time, while at other times, they were presented asynchronously, which created differences in stimulus onset asynchrony. Developing the models enabled the subtleties of this overlap in stimulus presentation to be captured and analyzed.

During the mathematical computation, the model performed the task as an incremental counting task instead of a memory retrieval task. For example, if the math problem was 36 + 7, an incremental counting strategy involved counting 7 digits from 36 to 43. However, a memory retrieval strategy might involve retrieving 6 + 7 = 13 and then changing the 10s digit to a 4 to be consistent with the problem (36 + 7). The model used incremental counting because this was consistent with reports from the experimenters that many subjects were counting in order to solve the problems and not using memory retrievals. While some subjects were using memory retrieval methods for solving the mathematical problems (and such a method could be implemented into each model), the predictive power of the models was not significantly changed. If a target was presented during the incremental counting task, the model responded that there was not enough time to complete the mathematical calculation.

The commented code for each model is presented in the Appendices. Appendix A presents the 4-second exposure time baseline model. Appendix B presents the commented code for the 2-second exposure time baseline model, and Appendix C presents the 2-second exposure time predicted visual condition code. Appendix D presents the code for the 2-second exposure time in the no-load condition.

4. Results and Discussion

Figure 1 shows the ACT-R model predicted results for the 2- and 4 second visual conditions. The predictions show that the number of errors committed in the visual condition should be about equal to the number of errors committed in the auditory condition.

The predictions are based on the relationship between the exposure times and the error percentage represented in the baseline data. The relationship yielded a linear correlation of $\underline{r}\,(6) = -.65$, $p = .15$. Other types of nonlinear relationships were explored, but they did not yield results that were any more significant than the linear relationship (note that the linear relationship is not significant either). Exposure times were essentially converted by the model to activation levels, depending on the amount of time the target was available for cognitive processing. The activation levels were then correlated to error rates in the baseline data; a regression equation

was used to make the predictions for the 2- and 4-second visual conditions. Table 1 shows the regression equation values for error prediction.

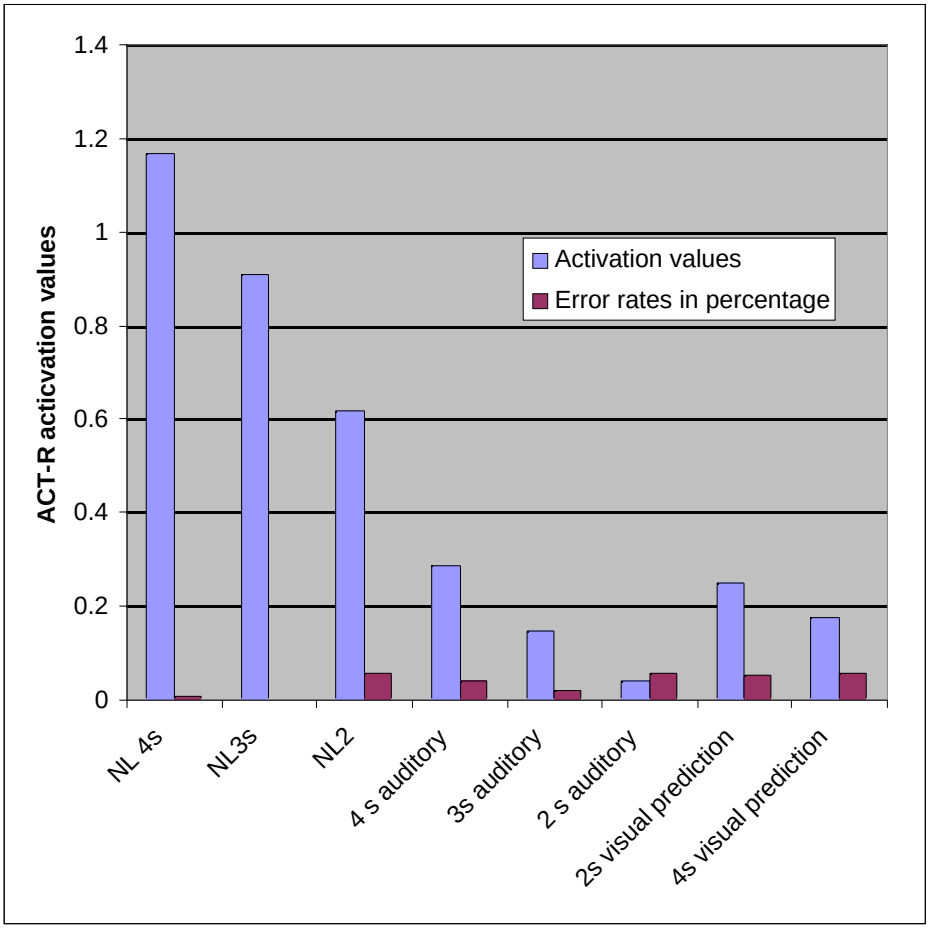

Figure 1. Activation values and percentage of errors for (a) the 4-, 3-, and 2-second no load (NL); (b) the 4-, 3-, and 2-second with math problems being presented aurally; and (c) the two prediction values for the 2- and 4-second visual conditions.

Table 1. Regression equation values

Variable	B	SE B	Beta
Error Percentage	-12.030	6.97	.889

Figure 2 shows a workload analysis that uses IMPRINT. IMPRINT employs a simplified version of the MRT of workload (Wickens & Yeh, 1986). IMPRINT also includes workload estimation scales developed by McCracken and Aldrich (1984), which allow the analyst to assign normalized workload values to each perceptual channel. There are four perceptual channels: visual, auditory, cognitive, and psychomotor. The results of the IMPRINT analysis are shown in Figure 2. The figure shows that the overall workload (combined total of all the perceptual

channels) for the auditory condition and the predicted visual condition are about equal. There is a sharp decline for the visual condition and auditory condition at about 1.25 seconds, and this could be significant. However, it is difficult to make predictions about error rates from a workload model. The previously discussed ACT-R models seem to suggest that below a certain level of activation (about .6), errors start to increase dramatically (this was one reason a nonlinear regression equation was explored). The IMPRINT model seems to suggest that the visual workload is lower over time, but the overall activation levels (produced by the ACT-R model) indicated that the lower level of workload has no impact since the activation level will still be under .6. While IMPRINT does not give a precise prediction for overall error rates in the way ACT-R does, the IMPRINT analysis does offer some insight to the predictions.

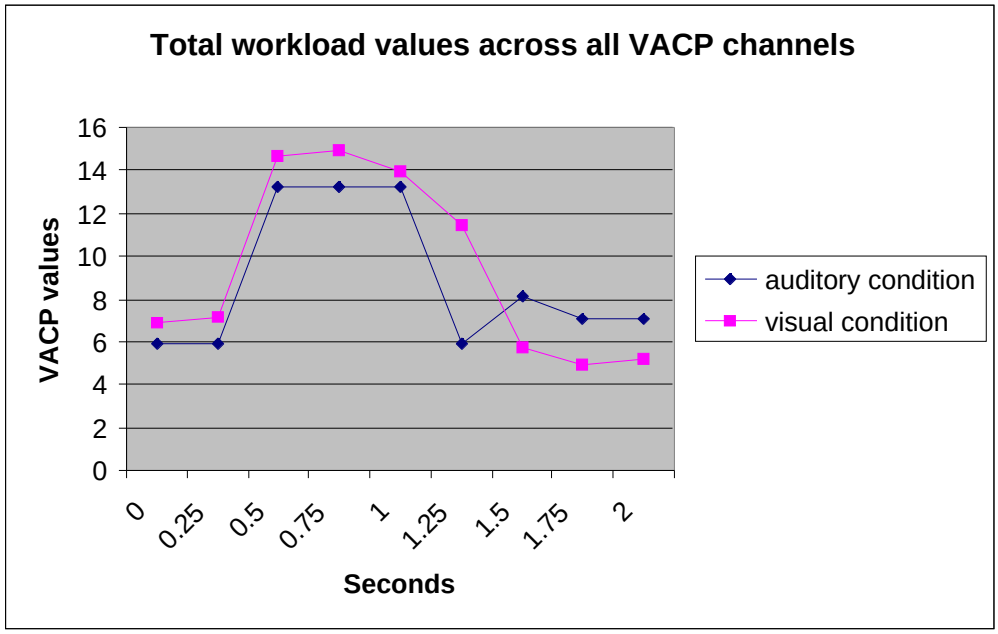

Figure 2. Total workload over time for the 2-second condition, auditory condition, and visual prediction.

In summary, results from this modeling effort indicated that for the primary task of shooting, both a visual secondary task and an auditory secondary task should yield about the same number of errors. This result was generated with a regression equation that incorporated ACT-R activation values as predictor variables for the regression equation. The results are also based on the existing auditory condition error rate data, which were taken from the Scribner and Harper (2001) study.

Prediction of cognitive workload and its effects on human error is one of the most challenging tasks that the human factors community must address in the years to come. Theoretically based computational cognitive models allow for a way to capture aspects of cognitive workload which cannot be addressed by non-computational theories. This project also allowed us to explore the problems associated with human error prediction by the use of cognitive modeling.

One problem with human error prediction is that typical error performance falls between a range of values. So the question for the cognitive modeler is how to transfer the range of human values to a range of predicted values. When the modeler is predicting human error with a cognitive model that has a stochastic output, he or she can set the degree of stochastic variability of the output. In other words, a modeler could create a large amount or a small amount of predicted error variability. Probably the best solution is to transfer the average error from the human data to the model's predicted performance. However, for this project, no agreement was reached about how to predict the error variability.

This report documents the modeling of error prediction based on target exposure time. There are a few possibilities that would create problems for these predictions. For example, the visual short-term memory store is not as long as the auditory short-term memory store (Neisser, 1967). This could create problems for soldiers in the visual condition, who will be trying to remember math problems while they are trying to identify targets. However, if they do forget the math problems, they can always refresh their memory by glancing at the display screen where the math problems are situated. Task-sharing efficiency might therefore outweigh any memory decay effects associated with visual and auditory memory stores. Secondly, a response bias could be expressed during each experimental trial. For example, if the last target was an enemy, subjects would be more likely to identify the next target as an enemy and would also be more inclined to shoot. This is something that could be examined within ACT-R and may require more research if the predictions here are incorrect. Finally, the placement of the visual images that display the math problems is critical to avoid confusion between the two different stimuli (targets and math problems). The math problem display should be relatively close to the target information without overlapping any of the target display area. In many ways, the ACT-R model represents an optimally positioned math problem display, and variations from a optimally positioned display would be likely to produce more errors than those predicted by the model.

The soldier of tomorrow will be expected to process more cognitive information about the battlefield situation. This research outlines potential bottlenecks in the cognitive workload, which future soldiers could face. Future research will validate the predictions made by the ACT-R and the IMPRINT architectures. It is hoped that the predictions made by both architectures will eventually prove to be valid and will eventually be used to aid in system design.

5. References

Allender, L., Kelley, T.D., Salvi, L., Lockett, J., Headley, D. B., Promisel, D., Mitchell, D., Richer, C., Feng, T. Verification, Validation, and Accreditation of a Soldier-System Modeling Tool. *Proceedings of the Human Factors and Ergonomics Society 29th Annual Meeting*, pp. 1219-1223, San Diego, 1995.

Anderson, J.R., Lebiere, C., *The Atomic Components of Thought*, Lawrence Erlbaum Associates: Mahwah, NJ, 1998.

Atkinson, R.C., Hernstein, R.J., Lindzey, G., Luce, R.D., Eds. *Steven's Handbook of Experimental Psychology*, 2nd ed., Wiley: New York, 1988.

Byrne, M.D., Anderson, J.R., Serial Modules in Parallel: The Psychological Refractory Period and Perfect Time Sharing, *Psychological Review*, **2001**, *108*, 847-869.

Damos, D.L., *Multiple Task Performance*. Taylor & Francis: London, England, 1991.

Gopher, D., Dochin, E., Workload: An Examination of the Concept. In *Handbook of Perception and Human Performance, Vol 2: Cognitive Processes and Performance*, Boff, K.R., Kaufman, L., Thomas, J.P., Eds., Wiley: New York, 1986, pp. 41.1-41.49.

Kelley, T.D., Patton, D.J., Allender, L., Predicting Situation Awareness Errors Using Cognitive Modeling. In *Proceedings of Human-Computer Interaction International 2001 Conference, Vol. 1*: *Usability Evaluation and Interface Design: Cognitive Engineering, Intelligent Agents and Virtual Reality.* In Smith, M.J., Salvendy, G., Harris, D., Koubek, R.J., Eds., Lawrence Erlbaum Associates: Mahwah, NJ, 2001, pp. 1455-1459.

McCracken, J.H., Aldrich, T.B., *Analyses of Selected LHX Mission Functions: Implications for Operator Workload and System Automation Goals*, Technical Note ASI479-024-84, Army Research Institute, Aviation Research and Development Activity: Fort Rucker, AL, 1984.

Meyer, D.E., Kornblum, S., Eds., *Attention and Performance XIV: Synergies in Experimental Psychology, Artificial Intelligence, and Cognitive Neuroscience*, MIT Press: Cambridge, MA, 1993.

Neisser, U., *Cognitive Psychology*. Appleton-Century-Crofts: New York, 1967.

Neumann, O., Beyond Capacity: A Functional View of Attention. In *Perspectives on Perception and Action*, Heuer, H., Sanders A.F., Eds., Lawrence Erlbaum Associates: Hillsdale, NJ, 1987, pp. 361-394.

Scribner, D.R., The Effect of Cognitive Load and Target Characteristics on Soldier Shooting Performance and Friendly Targets Engaged, ARL-TR-2838, U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2002.

Scribner, D.R., Harper, W.H., *The Effects of Mental Workload: Soldier Shooting and Secondary Cognitive Task Performance*, ARL-TR-2525, U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2001.

Wickens, C.D. Processing Resources in Attention, In *Varieties of Attention*, Parasuraman, R., Beatty, J., Davies, R., Eds., Wiley: New York, 1984, pp. 63-101.

Wickens, C.D., Yeh, Y-Y., A Multiple Resource Model of Workload Prediction and Assessment. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, Atlanta, GA, 1986.

INTENTIONALLY LEFT BLANK

Appendix A. ACT-R Code for 4-second Exposure Time, Auditory Condition

```
;; Troy Kelley
;; Army Research Laboratory, September 2002

;; This is code that simulates data taken from a study called -The Effect of Cognitive Load
;; and Target Characteristics on Soldier Shooting Performance and Friendly Targets Engaged- by
;; David R.  Scribner
;;
;;
;;
;; This is our time tracker function which will modify chunks at certain time intervals
;; to simulate the outside world


;; This will run the model multiple times

(defun myrun (X)
  (dotimes (i X)
  (run)
  (reset)))

;; we call this function after we answer the question to set the problem flag to problem complete
;; Otherwise we will try and answer the question again.

(defun zero ()
  (mod-chunk inside1 problemcomplete 1))

;; 4 second cycle


(defun timetracker (x)
    ;; at zero time a math problem starts and a target pops up - this is set in our declarative
    ;; memory.  The numbers in the conditional statement correspond to time values.  So, in the
    ;; first conditional statement, between 1.9 and 2.3 seconds, the sound chunk gets set to
zero.
    ;; The sound slot having a value of 0 means that the problem is not being read and conversly,
    ;; the sound slot value of 1 means that a problem is being read.  The same is true for the
    ;; target slot in the out1 chunk.  Target 1 means that a target is visible, while target 0
    ;; means that there is no target.


    ;; math problem completes
   (cond ((and (> x 1.9) (< x 2.3))
          (mod-chunk out1 sound 0)))

   ;; target down
   (cond ((and (> x 3.9) (< x 4.3))
          (mod-chunk out1 target 0)))

   ;; type FriendlyGray is never really used.  It is a left over from older code

   ;; target up - 2
   (cond ((and (> x 4.9) (< x 5.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))


   ;; math problem starts
   (cond ((and (> x 5.9) (< x 6.3))
          (mod-chunk out1 sound 1)))

   ;; once a problem completes we need to re-initialize some values

   (cond ((and (> x 7.9) (< x 8.3))
          (mod-chunk out1 sound 0)
```

```
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 38 secondnum 8)))

;; target down
(cond ((and (> x 8.9) (< x 9.3))
       (mod-chunk out1 target 0)))

;; target up - 3
(cond ((and (> x 9.9) (< x 10.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))


;; math problem starts
(cond ((and (> x 11.9) (< x 12.3))
       (mod-chunk out1 sound 1)))

;; target down and problem complete
(cond ((and (> x 13.9) (< x 14.3))
       (mod-chunk out1 sound 0 target 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 29 secondnum 9)))

;; target up -4
(cond ((and (> x 14.9) (< x 15.3))
       (mod-chunk out1 sound 0 target 1)
       (mod-chunk targettype1 type enemy)))

;; math problem starts
(cond ((and (> x 17.9) (< x 18.3))
       (mod-chunk out1 sound 1)))

;; target down
(cond ((and (> x 18.9) (< x 19.3))
       (mod-chunk out1 target 0)))

;; target up and end listening to problem - 5
(cond ((and (> x 19.9) (< x 20.3))
       (mod-chunk out1 target 1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 39 secondnum 5)
       (mod-chunk targettype1 type enemy)
       ))

;; target down and begin next problem
(cond ((and (> x 23.9) (< x 24.3))
       (mod-chunk out1 target 0 sound 1)))

;; target up-6
(cond ((and (> x 24.9) (< x 25.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)
      ))

;; end listening to math problem
(cond ((and (> x 25.9) (< x 26.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 34 secondnum 6)))

;; target down
(cond ((and (> x 28.9) (< x 29.3))
       (mod-chunk out1 target 0)))
```

```
;; target up and begin next problem-7
   (cond ((and (> x 29.9) (< x 30.3))
          (mod-chunk out1 target 1 sound 1)
          (mod-chunk targettype1 type friendlyGray)
          ))

    ;; math problem completes
   (cond ((and (> x 31.9) (< x 32.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 88 secondnum 3)))

   ;; target down
   (cond ((and (> x 33.9) (< x 34.3))
          (mod-chunk out1 target 0)))

   ;; target up-8
   (cond ((and (> x 34.9) (< x 35.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)
          ))

   ;; math problem starts
   (cond ((and (> x 35.9) (< x 36.3))
          (mod-chunk out1 sound 1)))

   ;; math problem completes
   (cond ((and (> x 37.9) (< x 38.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 41 secondnum 9)))

   ;; target down
   (cond ((and (> x 38.9) (< x 39.3))
          (mod-chunk out1 target 0)))

   ;; target up-9
   (cond ((and (> x 39.9) (< x 40.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)
          ))

   ;; math problem starts
   (cond ((and (> x 41.9) (< x 42.3))
          (mod-chunk out1 sound 1)))

   ;; target down and problem complete
   (cond ((and (> x 43.9) (< x 44.3))
          (mod-chunk out1 sound 0 target 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 61 secondnum 9)))

   ;; target up-10
   (cond ((and (> x 44.9) (< x 45.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type enemy)
          ))

   ;; math problem starts
   (cond ((and (> x 47.9) (< x 48.3))
          (mod-chunk out1 sound 1)))

   ;; target down
   (cond ((and (> x 48.9) (< x 49.3))
          (mod-chunk out1 target 0)))
```

```
;; target up and end listening to problem-11
   (cond ((and (> x 49.9) (< x 50.3))
          (mod-chunk out1 target 1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 85 secondnum 6)
          (mod-chunk targettype1 type enemy)
          ))

   ;; target down and begin next problem
   (cond ((and (> x 53.9) (< x 54.3))
          (mod-chunk out1 target 1 sound 1)))

   ;; target up-12
   (cond ((and (> x 54.9) (< x 55.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)
          ))

   ;; end listening to math problem
   (cond ((and (> x 55.9) (< x 56.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 32 secondnum 9)))

   ;; target down
   (cond ((and (> x 58.9) (< x 59.3))
          (mod-chunk out1 target 0)))

   ;; target up and begin next problem-13
   (cond ((and (> x 59.9) (< x 60.3))
          (mod-chunk out1 target 1 sound 1)
          (mod-chunk targettype1 type enemy)
          ))

    ;; math problem completes
   (cond ((and (> x 61.9) (< x 62.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 69 secondnum 3)))

   ;; target down
   (cond ((and (> x 63.9) (< x 64.3))
          (mod-chunk out1 target 0)))

   ;; target up-14
   (cond ((and (> x 64.9) (< x 65.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

   ;; math problem starts
   (cond ((and (> x 65.9) (< x 66.3))
          (mod-chunk out1 sound 1)))

   ;; math problem completes
   (cond ((and (> x 67.9) (< x 68.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 57 secondnum 7)))

   ;; target down
   (cond ((and (> x 68.9) (< x 69.3))
          (mod-chunk out1 target 0)))
```

```
;; target up-15
(cond ((and (> x 69.9) (< x 70.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

;; math problem starts
(cond ((and (> x 71.9) (< x 72.3))
       (mod-chunk out1 sound 1)))

;; target down and problem complete
(cond ((and (> x 73.9) (< x 74.3))
       (mod-chunk out1 sound 0 target 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 19 secondnum 9)))

;; target up-16
(cond ((and (> x 74.9) (< x 75.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; math problem starts
(cond ((and (> x 77.9) (< x 78.3))
       (mod-chunk out1 sound 1)))

;; target down
(cond ((and (> x 78.9) (< x 79.3))
       (mod-chunk out1 target 0)))

;; target up and end listening to problem-17
(cond ((and (> x 79.9) (< x 80.3))
       (mod-chunk out1 target 1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 67 secondnum 9)
       (mod-chunk targettype1 type enemy)))

;; target down and begin next problem
(cond ((and (> x 83.9) (< x 84.3))
       (mod-chunk out1 target 1 sound 1)))

;; target up-18
(cond ((and (> x 84.9) (< x 85.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; end listening to math problem
(cond ((and (> x 85.9) (< x 86.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 74 secondnum 8)))

;; target down
(cond ((and (> x 88.9) (< x 89.3))
       (mod-chunk out1 target 0)))

;; target up and begin next problem-19
(cond ((and (> x 89.9) (< x 90.3))
       (mod-chunk out1 target 1 sound 1)
       (mod-chunk targettype1 type friendlyGray)))

 ;; math problem completes
(cond ((and (> x 91.9) (< x 92.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
```

```
       (mod-chunk problem1 firstnum 48 secondnum 8)))

;; target down
(cond ((and (> x 93.9) (< x 94.3))
       (mod-chunk out1 target 0)))

;; target up-20
(cond ((and (> x 94.9) (< x 95.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

;; math problem starts
(cond ((and (> x 95.9) (< x 96.3))
       (mod-chunk out1 sound 1)))

;; problem complete
(cond ((and (> x 97.9) (< x 98.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 76 secondnum 4)))

;; target down
(cond ((and (> x 98.9) (< x 99.3))
       (mod-chunk out1 target 0)))

;; target up-21
(cond ((and (> x 99.9) (< x 100.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

;; math problem starts
(cond ((and (> x 101.9) (< x 102.3))
       (mod-chunk out1 sound 1)))

;; target down and problem complete
(cond ((and (> x 103.9) (< x 104.3))
       (mod-chunk out1 sound 0 target 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 45 secondnum 7)))

;; target up-22
(cond ((and (> x 104.9) (< x 105.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

;; math problem starts
(cond ((and (> x 107.9) (< x 108.3))
       (mod-chunk out1 sound 1)))

;; target down
(cond ((and (> x 108.9) (< x 109.3))
       (mod-chunk out1 target 0)))

;; target up and end listening to problem-23
(cond ((and (> x 109.9) (< x 110.3))
       (mod-chunk out1 target 1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 88 secondnum 3)
       (mod-chunk targettype1 type enemy)))
;; target down and begin next problem
(cond ((and (> x 113.9) (< x 114.3))
       (mod-chunk out1 target 1 sound 1)))

;; target up-24
(cond ((and (> x 114.9) (< x 115.3))
```

```
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type enemy)))

   ;; end listening to math problem
   (cond ((and (> x 115.9) (< x 116.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 38 secondnum 6)))

   ;; target down
   (cond ((and (> x 118.9) (< x 119.3))
          (mod-chunk out1 target 0)))


 ;; ending problem
   (cond ((and (> x 125.9) (< x 126.3))
          (mod-chunk out1 target nil sound nil)))

)

(clear-all)

;;  Note that we have the latency factor - lf - set very low.  This is because we really don't
care
;;  about time in the traditional sence, we only care that our conditional statements up above
will get
;; triggered at the correct times.  If our latencey factor was too long, we might go into some
;; decision process and come out 3 or 4 seconds later, which would mean our conditional
statements
;; up above would not trip at the correct times.

(sgp :lf .1 :rt 0.0 :bll .5 :blc 20 :ga 10.0 :al 2.0)

(chunk-type count-order first second)
(chunk-type count-from start end)
(chunk-type outside sound target)
(chunk-type begin)
(chunk-type problem firstnum secondnum)
(chunk-type increment num1 counter)
(chunk-type target)
(chunk-type targettype size type)
(chunk-type check)
(chunk-type inside soundcomplete targetchecked problemcomplete 0)
(chunk-type shoot)


(add-dm
(enemy isa chunk)
(friendlyGray isa chunk)
(friendlyWhite isa chunk)
(check1 isa check)
(begin1 isa begin)
(target1 isa target)
(shoot1 isa shoot)
(inside1 isa inside soundcomplete 0 targetchecked 0 problemcomplete 0)
(targettype1 isa targettype size 0 type enemy)
(increment1 isa increment num1 0 counter 0)
(problem1 isa problem firstnum 55 secondnum 9)
(out1 isa outside sound 1 target 1)
(countfrom1 isa count-from start 0)
(chunk1 isa count-order first 1 second 2)
(chunk2 isa count-order first 2 second 3)
(chunk3 isa count-order first 3 second 4)
(chunk4 isa count-order first 4 second 5)
(chunk5 isa count-order first 5 second 6)
(chunk6 isa count-order first 6 second 7)
(chunk7 isa count-order first 7 second 8)
(chunk8 isa count-order first 8 second 9)
(chunk9 isa count-order first 9 second 10)
```

```
(chunk10 isa count-order first 10 second 11)
(chunk11 isa count-order first 11 second 12)
(chunk12 isa count-order first 12 second 13)
(chunk13 isa count-order first 13 second 14)
(chunk14 isa count-order first 14 second 15)
(chunk15 isa count-order first 15 second 16)
(chunk16 isa count-order first 16 second 17)
(chunk17 isa count-order first 17 second 18)
(chunk18 isa count-order first 18 second 19)
(chunk19 isa count-order first 19 second 20)
(chunk20 isa count-order first 20 second 21)
(chunk21 isa count-order first 21 second 22)
(chunk22 isa count-order first 22 second 23)
(chunk23 isa count-order first 23 second 24)
(chunk24 isa count-order first 24 second 25)
(chunk25 isa count-order first 25 second 26)
(chunk26 isa count-order first 26 second 27)
(chunk27 isa count-order first 27 second 28)
(chunk28 isa count-order first 28 second 29)
(chunk29 isa count-order first 29 second 30)
(chunk30 isa count-order first 30 second 31)
(chunk31 isa count-order first 31 second 32)
(chunk32 isa count-order first 32 second 33)
(chunk33 isa count-order first 33 second 34)
(chunk34 isa count-order first 34 second 35)
(chunk35 isa count-order first 35 second 36)
(chunk36 isa count-order first 36 second 37)
(chunk37 isa count-order first 37 second 38)
(chunk38 isa count-order first 38 second 39)
(chunk39 isa count-order first 39 second 40)
(chunk40 isa count-order first 40 second 41)
(chunk41 isa count-order first 41 second 42)
(chunk42 isa count-order first 42 second 43)
(chunk43 isa count-order first 43 second 44)
(chunk44 isa count-order first 44 second 45)
(chunk45 isa count-order first 45 second 46)
(chunk46 isa count-order first 46 second 47)
(chunk47 isa count-order first 47 second 48)
(chunk48 isa count-order first 48 second 49)
(chunk49 isa count-order first 49 second 50)
(chunk50 isa count-order first 50 second 51)
(chunk51 isa count-order first 51 second 52)
(chunk52 isa count-order first 52 second 53)
(chunk53 isa count-order first 53 second 54)
(chunk54 isa count-order first 54 second 55)
(chunk55 isa count-order first 55 second 56)
(chunk56 isa count-order first 56 second 57)
(chunk57 isa count-order first 57 second 58)
(chunk58 isa count-order first 58 second 59)
(chunk59 isa count-order first 59 second 60)
(chunk60 isa count-order first 60 second 61)
(chunk61 isa count-order first 61 second 62)
(chunk62 isa count-order first 62 second 63)
(chunk63 isa count-order first 63 second 64)
(chunk64 isa count-order first 64 second 65)
(chunk65 isa count-order first 65 second 66)
(chunk66 isa count-order first 66 second 67)
(chunk67 isa count-order first 67 second 68)
(chunk68 isa count-order first 68 second 69)
(chunk69 isa count-order first 69 second 70)
(chunk70 isa count-order first 70 second 71)
(chunk71 isa count-order first 71 second 72)
(chunk72 isa count-order first 72 second 73)
(chunk73 isa count-order first 73 second 74)
(chunk74 isa count-order first 74 second 75)
(chunk75 isa count-order first 75 second 76)
(chunk76 isa count-order first 76 second 77)
(chunk77 isa count-order first 77 second 78)
(chunk78 isa count-order first 78 second 79)
(chunk79 isa count-order first 79 second 80)
(chunk80 isa count-order first 80 second 81)
```

```
(chunk81 isa count-order first 81 second 82)
(chunk82 isa count-order first 82 second 83)
(chunk83 isa count-order first 83 second 84)
(chunk84 isa count-order first 84 second 85)
(chunk85 isa count-order first 85 second 86)
(chunk86 isa count-order first 86 second 87)
(chunk87 isa count-order first 87 second 88)
(chunk88 isa count-order first 88 second 89)
(chunk89 isa count-order first 89 second 90)
(chunk90 isa count-order first 90 second 91)
(chunk91 isa count-order first 91 second 92)
(chunk92 isa count-order first 92 second 93)
(chunk93 isa count-order first 93 second 94)
(chunk94 isa count-order first 94 second 95)
(chunk95 isa count-order first 95 second 96)
(chunk96 isa count-order first 96 second 97)
(chunk97 isa count-order first 97 second 98)
(chunk98 isa count-order first 98 second 99)
(chunk99 isa count-order first 99 second 100)
(chunk100 isa count-order first 100 second 101))


;; we have three competing productions (begin-task, scan-for-targets and listen-for-problem)
;; which represent competing goals of the
;; multi-task environment.

(P begin-task
   =begin1>
      isa begin
    =out1>
      isa outside
      sound =sound
      target =target
==>
   !bind!     =newvar (actr-time)
   !output!   =newvar
   !eval!     (timetracker =newvar)
   !push! =out1
)

(P check-target
   =check1>
      isa check
   =outside1>
      isa outside
      sound 0
      target =target
   =target1>
      isa target
==>

     !bind!     =newvar (actr-time)
     !output!   =newvar
     !eval!     (timetracker =newvar)

      !pop!
)

;; This production will fire if we have a sound - a math problem - and a target happening at
;; the same time.  In other words, since we have both things going on at once, we really have
;; no time to double-check the target to make sure it is correct.  So we look at the target
;; once and make our shoot decision.

(P cant-check-target
   =check1>
      isa check
   =out1>
      isa outside
      sound 1
      target =target
   =shoot1>
```

```
      isa shoot

==>
  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push! =shoot1
)
(P shoot-decision
   =shoot1>
     isa shoot
   =out1>
     isa outside
     target =target
==>

    =out1>
       target 0
    !pop!
    !pop!
    !pop!
)
(P problem-complete
   =out1>
      isa outside
      sound 0
      target 0
   =inside1>
      isa inside
      problemcomplete 0
   =problem1>
       isa problem
   =target1>
      isa target
==>

  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)


    !push!       =problem1

)

(P scan-for-targets
   =out1>
      isa outside
      sound =sound
      target 1
   =inside1>
      isa inside
      problemcomplete =problemcomplete
   =problem1>
       isa problem
   =target1>
      isa target
==>

  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push!  =target1
)
(P listen-for-problem
   =out1>
      isa outside
      sound 1
      target 0
```

```
   =inside1>
       isa inside
       problemcomplete =problemcomplete
   =problem1>
       isa problem
   =target1>
       isa target
==>
     !bind!     =newvar (actr-time)
     !output!   =newvar
     !eval!     (timetracker =newvar)
)



(P identify-target
   =target1>
     isa target
   =targettype1>
     isa targettype
     size =size
     type =type
   =check1>
     isa check
==>

   !bind!     =newvar (actr-time)
   !output!   =newvar
   !push! =check1
)

(P remember-problem
   =problem1>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
   =countfrom1>
       isa count-from
       start =start
   =out1>
       isa outside
       sound =sound
       target =target
==>


    !bind!     =newvar (actr-time)
     !output!   =newvar
     !output!   =firstnum
     !output!   =secondnum
     !eval!     (timetracker =newvar)

     !push! =countfrom1


)


(P start-counting
   =countfrom1>
      ISA         count-from
      start       0
    =goal>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
    =out1>
       isa outside
       sound 0
       target =target
```

```
==>

    =countfrom1>
       start   =firstnum

   !output!       ("~S" =firstnum)
   !output!       ("~S" =secondnum)

   !bind!     =newvar (actr-time)
   !output!   =newvar

)

(P count
   =countfrom1>
      isa count-from
      start =firstnum
   =problem1>
      isa problem
      firstnum =firstnum
      secondnum =secondnum
   =increment1>
      isa increment
      num1 =num1
      counter =thecounter
   =out1>
      isa outside
      target =target
      sound =sound
==>
     !bind!     =newvar (actr-time)
     !output!   =newvar

   !bind!         =newtime (1+ =thecounter)

   =increment1>
      num1 =firstnum
      counter =newtime
   =increment1>
      !push! =increment1
      !output! =firstnum
)

;; Note that these productions are slightly different than the increment productions provided
;; in the ACT-R tutorials.  The difference is that the count-order chunk doesn't contain the
;; answer here, whereas in the ACT-R tutorials it does.  I didn't like the fact that the tutorial
;; basically included a chunk which had the answer to the increment problem as a slot.
;;  If the answer to
;; the problem was already stored, it shouldn't be a problem, you could just do a retrieval.
;;  This production instead increments
;; a certain number of times and when that number of times has been reached, stops and gets
;; the answer.  In other words, you only know how many times to increment, and where to start.
;; So you increment a certian number of times, then you stop and see what the answer is.

(P increment
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
       target 0
==>
    !bind!     =newvar (actr-time)
```

```
    !output!   =newvar


    !bind!         =newTime (1+ =thecounter)
    !eval!     (timetracker =newvar)
    =increment1>
       isa increment
       num1 =secondnum
     counter =newtime
    !output! =secondnum

)


;; This production will fire if there is also a target showing up at the same time as we are
;; trying to increment to get the answer.


(P no-time-to-increment
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
       target 1
==>
    !bind!     =newvar (actr-time)
    !output!   =newvar
    !bind!         =newTime (1+ =thecounter)
    !eval!     (timetracker =newvar)
    !pop!
    !pop!
    !pop!


)

(P answer
   =increment1>
      isa increment
      num1 =num1
      counter =counter
   =countfrom>
       isa count-order
       first =num1
       second =secondnumber
   =problem1>
       isa problem
       secondnum =counter

==>


   !output! =secondnumber
   !eval! (zero)

   !pop!
   !pop!
   !pop!


)
```

```
(spp ANSWER :Q 1.0 :R 1.0 :A 0.05 :B 0 :STRENGTH 0.0 :VALUE 1)
(spp SCAN-FOR-TARGETS :B 1)
(spp listen-for-problem :B 1)


(goal-focus begin1)
```

Appendix B. ACT-R Code for 2-second Exposure Time, Auditory Condition

```
;; Troy Kelley
;; Army Research Laboratory, September 2002

;; This is code that simulates data taken from a study called -The Effect of Cognitive Load
;; and Target Characteristics on Soldier Shooting Performance and Friendly Targets Engaged- by
;; David R.  Scribner
;;

;; This is our time tracker function which will modify chunks at certain time intervals
;; to simulate the outside world

;;(defun check (x))

(defun zero ()
  (mod-chunk inside1 problemcomplete 1))

;; 2 second cycle


(defun timetracker (x)
    ;; at zero time a math problem starts and a target pops up - this is set in our declarative
    ;; memory


    ;; target down and math problem completes
   (cond ((and (> x 1.9) (< x 2.3))
          (mod-chunk out1 target 0 sound 0)))


   ;; target up - 2
   (cond ((and (> x 4.9) (< x 5.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

   ;; math problem starts
   (cond ((and (> x 5.9) (< x 6.3))
          (mod-chunk out1 sound 1)))

    ;; target down
   (cond ((and (> x 6.9) (< x 7.3))
          (mod-chunk out1 target 0)))


   ;; math problem completes
   (cond ((and (> x 7.9) (< x 8.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 38 secondnum 8)))

   ;; target up - 3
   (cond ((and (> x 9.9) (< x 10.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

   ;; math problem starts and target down
   (cond ((and (> x 11.9) (< x 12.3))
          (mod-chunk out1 sound 1 target 0)))

   ;; problem complete
   (cond ((and (> x 13.9) (< x 14.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
```

```
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 29 secondnum 9)))

;; target up -4
(cond ((and (> x 14.9) (< x 15.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 16.9) (< x 17.3))
       (mod-chunk out1 target 0)))

;; math problem starts
(cond ((and (> x 17.9) (< x 18.3))
       (mod-chunk out1 sound 1)))

;; target up and end listening to problem - 5
(cond ((and (> x 19.9) (< x 20.3))
       (mod-chunk out1 target 1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 39 secondnum 5)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 21.9) (< x 22.3))
       (mod-chunk out1 target 0)))

 ;; math problem starts
(cond ((and (> x 23.9) (< x 24.3))
       (mod-chunk out1 sound 1)))

;; target up-6
(cond ((and (> x 24.9) (< x 25.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; end listening to math problem
(cond ((and (> x 25.9) (< x 26.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 34 secondnum 6)))

;; target down
(cond ((and (> x 26.9) (< x 27.3))
       (mod-chunk out1 target 0)))


;; target up and begin next problem-7
(cond ((and (> x 29.9) (< x 30.3))
       (mod-chunk out1 target 1 sound 1)
       (mod-chunk targettype1 type friendlyGray)))


 ;; math problem completes and target down
(cond ((and (> x 31.9) (< x 32.3))
       (mod-chunk out1 sound 0 target 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 88 secondnum 3)))

;; target up-8
(cond ((and (> x 34.9) (< x 35.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))
```

```
;; math problem starts
(cond ((and (> x 35.9) (< x 36.3))
       (mod-chunk out1 sound 1)))

;; target down
(cond ((and (> x 36.9) (< x 37.3))
       (mod-chunk out1 target 0)))

;; math problem completes
(cond ((and (> x 37.9) (< x 38.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 41 secondnum 9)))

;; target up-9
(cond ((and (> x 39.9) (< x 40.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))


;; target down and problem starts
(cond ((and (> x 41.9) (< x 42.3))
       (mod-chunk out1 sound 1 target 0)))

;; math problem completes
(cond ((and (> x 43.9) (< x 44.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 41 secondnum 9)))

;; target up-10
(cond ((and (> x 44.9) (< x 45.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 46.9) (< x 47.3))
       (mod-chunk out1 target 0)))

;; math problem starts
(cond ((and (> x 47.9) (< x 48.3))
       (mod-chunk out1 sound 1)))

;; target up and end listening to problem-11
(cond ((and (> x 49.9) (< x 50.3))
       (mod-chunk out1 target 1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 85 secondnum 6)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 51.9) (< x 52.3))
       (mod-chunk out1 target 1 sound 0)))

;; target up-12
(cond ((and (> x 54.9) (< x 55.3))
       (mod-chunk out1 sound 1)))

;; end listening to math problem
(cond ((and (> x 55.9) (< x 56.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 32 secondnum 9)))
```

```
 ;; target down
 (cond ((and (> x 56.9) (< x 57.3))
        (mod-chunk out1 target 0)))

;; target up and begin next problem-7
 (cond ((and (> x 59.9) (< x 60.3))
        (mod-chunk out1 target 1 sound 1)
        (mod-chunk targettype1 type friendlyGray)))


  ;; math problem completes and target down
 (cond ((and (> x 61.9) (< x 62.3))
        (mod-chunk out1 sound 0 target 0)
        (mod-chunk inside1 problemcomplete 0)
        (mod-chunk countfrom1 start 0)
        (mod-chunk increment1 num1 0 counter 0)
        (mod-chunk problem1 firstnum 88 secondnum 3)))

 ;; target up-8
 (cond ((and (> x 64.9) (< x 65.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type friendlyGray)))

 ;; math problem starts
 (cond ((and (> x 65.9) (< x 66.3))
        (mod-chunk out1 sound 1)))

 ;; target down
 (cond ((and (> x 66.9) (< x 67.3))
        (mod-chunk out1 target 0)))

 ;; math problem completes
 (cond ((and (> x 67.9) (< x 68.3))
        (mod-chunk out1 sound 0)
        (mod-chunk inside1 problemcomplete 0)
        (mod-chunk countfrom1 start 0)
        (mod-chunk increment1 num1 0 counter 0)
        (mod-chunk problem1 firstnum 41 secondnum 9)))

 ;; target up-9
 (cond ((and (> x 69.9) (< x 70.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type friendlyGray)))


 ;; target down and problem starts
 (cond ((and (> x 71.9) (< x 72.3))
        (mod-chunk out1 sound 1 target 0)))

 ;; math problem completes
 (cond ((and (> x 73.9) (< x 74.3))
        (mod-chunk out1 sound 0)
        (mod-chunk inside1 problemcomplete 0)
        (mod-chunk countfrom1 start 0)
        (mod-chunk increment1 num1 0 counter 0)
        (mod-chunk problem1 firstnum 41 secondnum 9)))

 ;; target up-10
 (cond ((and (> x 74.9) (< x 75.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type enemy)))

 ;; target down
 (cond ((and (> x 76.9) (< x 77.3))
        (mod-chunk out1 target 0)))

 ;; math problem starts
 (cond ((and (> x 77.9) (< x 78.3))
        (mod-chunk out1 sound 1)))
```

```
;; target up and end listening to problem-11
(cond ((and (> x 79.9) (< x 80.3))
       (mod-chunk out1 target 1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 85 secondnum 6)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 81.9) (< x 82.3))
       (mod-chunk out1 target 1 sound 0)))

;; target up-12
(cond ((and (> x 84.9) (< x 85.3))
       (mod-chunk out1 sound 1)))

;; end listening to math problem
(cond ((and (> x 85.9) (< x 86.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 32 secondnum 9)))

;; target down
(cond ((and (> x 86.9) (< x 87.3))
       (mod-chunk out1 target 0)))


 ;; target up and begin next problem-7
(cond ((and (> x 89.9) (< x 90.3))
       (mod-chunk out1 target 1 sound 1)
       (mod-chunk targettype1 type friendlyGray)))


 ;; math problem completes and target down
(cond ((and (> x 91.9) (< x 92.3))
       (mod-chunk out1 sound 0 target 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 88 secondnum 3)))

;; target up-8
(cond ((and (> x 94.9) (< x 95.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

;; math problem starts
(cond ((and (> x 95.9) (< x 96.3))
       (mod-chunk out1 sound 1)))

;; target down
(cond ((and (> x 96.9) (< x 97.3))
       (mod-chunk out1 target 0)))

;; math problem completes
(cond ((and (> x 97.9) (< x 98.3))
       (mod-chunk out1 sound 0)
       (mod-chunk inside1 problemcomplete 0)
       (mod-chunk countfrom1 start 0)
       (mod-chunk increment1 num1 0 counter 0)
       (mod-chunk problem1 firstnum 41 secondnum 9)))

;; target up-9
(cond ((and (> x 99.9) (< x 100.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))
```

```
   ;; target down and problem starts
   (cond ((and (> x 101.9) (< x 102.3))
          (mod-chunk out1 sound 1 target 0)))

   ;; math problem completes
   (cond ((and (> x 103.9) (< x 104.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 41 secondnum 9)))

   ;; target up-10
   (cond ((and (> x 104.9) (< x 105.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type enemy)))

   ;; target down
   (cond ((and (> x 106.9) (< x 107.3))
          (mod-chunk out1 target 0)))

   ;; math problem starts
   (cond ((and (> x 107.9) (< x 108.3))
          (mod-chunk out1 sound 1)))

   ;; target up and end listening to problem-11
   (cond ((and (> x 109.9) (< x 110.3))
          (mod-chunk out1 target 1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 85 secondnum 6)
          (mod-chunk targettype1 type enemy)))

   ;; target down
   (cond ((and (> x 111.9) (< x 112.3))
          (mod-chunk out1 target 1 sound 0)))

   ;; target up-12
   (cond ((and (> x 114.9) (< x 115.3))
          (mod-chunk out1 sound 1)))

   ;; end listening to math problem
   (cond ((and (> x 115.9) (< x 116.3))
          (mod-chunk out1 sound 0)
          (mod-chunk inside1 problemcomplete 0)
          (mod-chunk countfrom1 start 0)
          (mod-chunk increment1 num1 0 counter 0)
          (mod-chunk problem1 firstnum 32 secondnum 9)))

   ;; target down
   (cond ((and (> x 116.9) (< x 117.3))
          (mod-chunk out1 target 0)))


 ;; ending problem
   (cond ((and (> x 125.9) (< x 126.3))
          (mod-chunk out1 target nil sound nil)))

)

(clear-all)



(sgp :lf .1 :rt 0.0 :bll .5 :blc 20 :ga 10.0 :al 2.0)



(chunk-type count-order first second)
(chunk-type count-from start end)
```

```
(chunk-type outside sound target)
(chunk-type begin)
(chunk-type problem firstnum secondnum)
(chunk-type increment num1 counter)
(chunk-type target)
(chunk-type targettype size type)
(chunk-type check)
(chunk-type inside soundcomplete targetchecked problemcomplete 0)
(chunk-type shoot)


(add-dm
(check1 isa check)
(begin1 isa begin)
(target1 isa target)
(shoot1 isa shoot)
(inside1 isa inside soundcomplete 0 targetchecked 0 problemcomplete 0)
(targettype1 isa targettype size 0 type enemy)
(increment1 isa increment num1 0 counter 0)
(problem1 isa problem firstnum 55 secondnum 9)
;; first outside variable
(out1 isa outside sound 1 target 1)
(countfrom1 isa count-from start 0)
(chunk1 isa count-order first 1 second 2)
(chunk2 isa count-order first 2 second 3)
(chunk3 isa count-order first 3 second 4)
(chunk4 isa count-order first 4 second 5)
(chunk5 isa count-order first 5 second 6)
(chunk6 isa count-order first 6 second 7)
(chunk7 isa count-order first 7 second 8)
(chunk8 isa count-order first 8 second 9)
(chunk9 isa count-order first 9 second 10)
(chunk10 isa count-order first 10 second 11)
(chunk11 isa count-order first 11 second 12)
(chunk12 isa count-order first 12 second 13)
(chunk13 isa count-order first 13 second 14)
(chunk14 isa count-order first 14 second 15)
(chunk15 isa count-order first 15 second 16)
(chunk16 isa count-order first 16 second 17)
(chunk17 isa count-order first 17 second 18)
(chunk18 isa count-order first 18 second 19)
(chunk19 isa count-order first 19 second 20)
(chunk20 isa count-order first 20 second 21)
(chunk21 isa count-order first 21 second 22)
(chunk22 isa count-order first 22 second 23)
(chunk23 isa count-order first 23 second 24)
(chunk24 isa count-order first 24 second 25)
(chunk25 isa count-order first 25 second 26)
(chunk26 isa count-order first 26 second 27)
(chunk27 isa count-order first 27 second 28)
(chunk28 isa count-order first 28 second 29)
(chunk29 isa count-order first 29 second 30)
(chunk30 isa count-order first 30 second 31)
(chunk31 isa count-order first 31 second 32)
(chunk32 isa count-order first 32 second 33)
(chunk33 isa count-order first 33 second 34)
(chunk34 isa count-order first 34 second 35)
(chunk35 isa count-order first 35 second 36)
(chunk36 isa count-order first 36 second 37)
(chunk37 isa count-order first 37 second 38)
(chunk38 isa count-order first 38 second 39)
(chunk39 isa count-order first 39 second 40)
(chunk40 isa count-order first 40 second 41)
(chunk41 isa count-order first 41 second 42)
(chunk42 isa count-order first 42 second 43)
(chunk43 isa count-order first 43 second 44)
(chunk44 isa count-order first 44 second 45)
(chunk45 isa count-order first 45 second 46)
(chunk46 isa count-order first 46 second 47)
(chunk47 isa count-order first 47 second 48)
(chunk48 isa count-order first 48 second 49)
```

```
(chunk49 isa count-order first 49 second 50)
(chunk50 isa count-order first 50 second 51)
(chunk51 isa count-order first 51 second 52)
(chunk52 isa count-order first 52 second 53)
(chunk53 isa count-order first 53 second 54)
(chunk54 isa count-order first 54 second 55)
(chunk55 isa count-order first 55 second 56)
(chunk56 isa count-order first 56 second 57)
(chunk57 isa count-order first 57 second 58)
(chunk58 isa count-order first 58 second 59)
(chunk59 isa count-order first 59 second 60)
(chunk60 isa count-order first 60 second 61)
(chunk61 isa count-order first 61 second 62)
(chunk62 isa count-order first 62 second 63)
(chunk63 isa count-order first 63 second 64)
(chunk64 isa count-order first 64 second 65)
(chunk65 isa count-order first 65 second 66)
(chunk66 isa count-order first 66 second 67)
(chunk67 isa count-order first 67 second 68)
(chunk68 isa count-order first 68 second 69)
(chunk69 isa count-order first 69 second 70)
(chunk70 isa count-order first 70 second 71)
(chunk71 isa count-order first 71 second 72)
(chunk72 isa count-order first 72 second 73)
(chunk73 isa count-order first 73 second 74)
(chunk74 isa count-order first 74 second 75)
(chunk75 isa count-order first 75 second 76)
(chunk76 isa count-order first 76 second 77)
(chunk77 isa count-order first 77 second 78)
(chunk78 isa count-order first 78 second 79)
(chunk79 isa count-order first 79 second 80)
(chunk80 isa count-order first 80 second 81)
(chunk81 isa count-order first 81 second 82)
(chunk82 isa count-order first 82 second 83)
(chunk83 isa count-order first 83 second 84)
(chunk84 isa count-order first 84 second 85)
(chunk85 isa count-order first 85 second 86)
(chunk86 isa count-order first 86 second 87)
(chunk87 isa count-order first 87 second 88)
(chunk88 isa count-order first 88 second 89)
(chunk89 isa count-order first 89 second 90)
(chunk90 isa count-order first 90 second 91)
(chunk91 isa count-order first 91 second 92)
(chunk92 isa count-order first 92 second 93)
(chunk93 isa count-order first 93 second 94)
(chunk94 isa count-order first 94 second 95)
(chunk95 isa count-order first 95 second 96)
(chunk96 isa count-order first 96 second 97)
(chunk97 isa count-order first 97 second 98)
(chunk98 isa count-order first 98 second 99)
(chunk99 isa count-order first 99 second 100)
(chunk100 isa count-order first 100 second 101))


;; we have three competing productions (begin-task, scan-for-targets and listen-for-problem)
;; which represent competing goals of the
;; multi-task environment.  The scan-for-targets however has the highest priority
;; so we will do that one over the listen-for-sound production, which basically
;; is listening for each math problem.

(P begin-task
   =begin1>
      isa begin
    =out1>
      isa outside
      sound =sound
      target =target
==>
   !bind!     =newvar (actr-time)
   !output!   =newvar
   !eval!     (timetracker =newvar)
```

```
   !push! =out1
)

(P check-target
   =check1>
      isa check
   =outside1>
      isa outside
      sound 0
      target =target
   =target1>
      isa target
==>

     !bind!     =newvar (actr-time)
     !output!   =newvar
     !eval!     (timetracker =newvar)

      !pop!
)
(P cant-check-target
   =check1>
      isa check
   =out1>
      isa outside
      sound 1
      target =target
   =shoot1>
      isa shoot

==>
  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push! =shoot1
)
(P shoot-decision
   =shoot1>
     isa shoot
   =out1>
     isa outside
     target =target
==>

    =out1>
       target 0
    !pop!
    !pop!
    !pop!
)
(P problem-complete
   =out1>
      isa outside
      sound 0
      target 0
   =inside1>
      isa inside
      problemcomplete 0
   =problem1>
       isa problem
   =target1>
      isa target
==>

  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)


    !push!       =problem1
```

```
)

(P scan-for-targets
   =out1>
      isa outside
      sound =sound
      target 1
   =inside1>
      isa inside
      problemcomplete =problemcomplete
   =problem1>
       isa problem
   =target1>
      isa target
==>

  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push!  =target1
)
(P listen-for-problem
   =out1>
      isa outside
      sound 1
      target 0
   =inside1>
       isa inside
       problemcomplete =problemcomplete
   =problem1>
       isa problem
   =target1>
       isa target
==>
     !bind!     =newvar (actr-time)
     !output!   =newvar
     !eval!     (timetracker =newvar)
)



(P identify-target
   =target1>
     isa target
   =targettype1>
     isa targettype
     size =size
     type =type
   =check1>
     isa check
==>

   !bind!     =newvar (actr-time)
   !output!   =newvar
   !push! =check1
)

(P remember-problem
   =problem1>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
   =countfrom1>
       isa count-from
       start =start
   =out1>
       isa outside
       sound =sound
```

```
       target =target
==>


    !bind!     =newvar (actr-time)
     !output!   =newvar
     !output!   =firstnum
     !output!   =secondnum
     !eval!     (timetracker =newvar)

     !push! =countfrom1


)


(P start-counting
   =countfrom1>
      ISA         count-from
      start       0
    =goal>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
    =out1>
       isa outside
       sound 0
       target =target
==>

    =countfrom1>
       start   =firstnum

   !output!       ("~S" =firstnum)
   !output!       ("~S" =secondnum)

   !bind!     =newvar (actr-time)
   !output!   =newvar

)

(P count
   =countfrom1>
      isa count-from
      start =firstnum
   =problem1>
      isa problem
      firstnum =firstnum
      secondnum =secondnum
   =increment1>
      isa increment
      num1 =num1
      counter =thecounter
   =out1>
      isa outside
      target =target
      sound =sound
==>
     !bind!     =newvar (actr-time)
     !output!   =newvar

   !bind!         =newtime (1+ =thecounter)

   =increment1>
      num1 =firstnum
      counter =newtime
   =increment1>
      !push! =increment1
      !output! =firstnum
)
(P increment
```

```
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
       target =target
==>
    !bind!     =newvar (actr-time)
    !output!   =newvar
    !output!   =target

    !bind!         =newTime (1+ =thecounter)
    =increment1>
       isa increment
       num1 =secondnum
     counter =newtime
    !output! =secondnum

)
(P no-time-to-increment
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
       target 1
==>
    !bind!     =newvar (actr-time)
    !output!   =newvar
    !bind!         =newTime (1+ =thecounter)
    !eval!     (timetracker =newvar)
    !pop!
    !pop!
    !pop!


)

(P answer
   =increment1>
      isa increment
      num1 =num1
      counter =counter
   =countfrom>
       isa count-order
       first =num1
       second =secondnumber
   =problem1>
       isa problem
       secondnum =counter

==>


   !output! =secondnumber
```

```
   !eval! (zero)

   !pop!
   !pop!
   !pop!


)

(spp ANSWER :Q 1.0 :R 1.0 :A 0.05 :B 0 :STRENGTH 0.0 :VALUE 1)
(spp SCAN-FOR-TARGETS :B 1)
(spp listen-for-problem :B 1)



(goal-focus begin1)
```

INTENTIONALLY LEFT BLANK

Appendix C. ACT-R Code for 2-second Exposure Time, Visual Condition

```
;; Troy Kelley
;; Army Research Laboratory, September 2002

;; This is code that simulates data taken from a study called -The Effect of Cognitive Load
;; and Target Characteristics on Soldier Shooting Performance and Friendly Targets Engaged- by
;; David R.  Scribner
;;

;; This is our time tracker function which will modify chunks at certain time intervals
;; to simulate the outside world

;; Note that in this code we are using the sound slot to simulate the target being available
;; visually for a certain amount of time, then it is gone when the sound slot is set back to
;; zero  - see the 4 second code for more complete comments.

;;(defun check (x))

(defun zero ()
  (mod-chunk inside1 problemcomplete 0)
  (mod-chunk countfrom1 start 0)
  (mod-chunk increment1 num1 0 counter 0)
  (mod-chunk problem1 firstnum 38 secondnum 8))

;; 2 second cycle


(defun timetracker (x)
    ;; at zero time a math problem starts and a target pops up - this is set in our declarative
    ;; memory


    ;; target down and math problem completes
   (cond ((and (> x 1.9) (< x 2.3))
          (mod-chunk out1 target 0 sound 0)))


   ;; target up - 2
   (cond ((and (> x 4.9) (< x 5.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

   ;; math problem starts
   (cond ((and (> x 5.9) (< x 6.3))
          (mod-chunk out1 sound 1)))

    ;; target down
   (cond ((and (> x 6.9) (< x 7.3))
          (mod-chunk out1 target 0)))


   ;; math problem completes
   (cond ((and (> x 7.9) (< x 8.3))
          (mod-chunk out1 sound 0)))

   ;; target up - 3
   (cond ((and (> x 9.9) (< x 10.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

   ;; math problem starts and target down
   (cond ((and (> x 11.9) (< x 12.3))
          (mod-chunk out1 sound 1 target 0)))

   ;; problem complete
```

```
(cond ((and (> x 13.9) (< x 14.3))
       (mod-chunk out1 sound 0)))

;; target up -4
(cond ((and (> x 14.9) (< x 15.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 16.9) (< x 17.3))
       (mod-chunk out1 target 0)))

;; math problem starts
(cond ((and (> x 17.9) (< x 18.3))
       (mod-chunk out1 sound 1)))

;; target up and end listening to problem - 5
(cond ((and (> x 19.9) (< x 20.3))
       (mod-chunk out1 target 1 sound 0)))

;; target down
(cond ((and (> x 21.9) (< x 22.3))
       (mod-chunk out1 target 0)))

 ;; math problem starts
(cond ((and (> x 23.9) (< x 24.3))
       (mod-chunk out1 sound 1)))

;; target up-6
(cond ((and (> x 24.9) (< x 25.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; end listening to math problem
(cond ((and (> x 25.9) (< x 26.3))
       (mod-chunk out1 sound 0)))

;; target down
(cond ((and (> x 26.9) (< x 27.3))
       (mod-chunk out1 target 0)))


;; target up and begin next problem-7
(cond ((and (> x 29.9) (< x 30.3))
       (mod-chunk out1 target 1 sound 1)
       (mod-chunk targettype1 type friendlyGray)))


 ;; math problem completes and target down
(cond ((and (> x 31.9) (< x 32.3))
       (mod-chunk out1 sound 0 target 0)))

;; target up-8
(cond ((and (> x 34.9) (< x 35.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

;; math problem starts
(cond ((and (> x 35.9) (< x 36.3))
       (mod-chunk out1 sound 1)))

;; target down
(cond ((and (> x 36.9) (< x 37.3))
       (mod-chunk out1 target 0)))

;; math problem completes
(cond ((and (> x 37.9) (< x 38.3))
       (mod-chunk out1 sound 0)))

;; target up-9
(cond ((and (> x 39.9) (< x 40.3))
```

```
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))


 ;; target down and problem starts
 (cond ((and (> x 41.9) (< x 42.3))
       (mod-chunk out1 sound 1 target 0)))

 ;; math problem completes
 (cond ((and (> x 43.9) (< x 44.3))
       (mod-chunk out1 sound 0)))

 ;; target up-10
 (cond ((and (> x 44.9) (< x 45.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

 ;; target down
 (cond ((and (> x 46.9) (< x 47.3))
       (mod-chunk out1 target 0)))

 ;; math problem starts
 (cond ((and (> x 47.9) (< x 48.3))
       (mod-chunk out1 sound 1)))

 ;; target up and end listening to problem-11
 (cond ((and (> x 49.9) (< x 50.3))
       (mod-chunk out1 target 1 sound 0)))

 ;; target down
 (cond ((and (> x 51.9) (< x 52.3))
       (mod-chunk out1 target 1 sound 0)))

 ;; target up-12
 (cond ((and (> x 54.9) (< x 55.3))
       (mod-chunk out1 sound 1)))

 ;; end listening to math problem
 (cond ((and (> x 55.9) (< x 56.3))
       (mod-chunk out1 sound 0)))

 ;; target down
 (cond ((and (> x 56.9) (< x 57.3))
       (mod-chunk out1 target 0)))

;; target up and begin next problem-7
 (cond ((and (> x 59.9) (< x 60.3))
       (mod-chunk out1 target 1 sound 1)
       (mod-chunk targettype1 type friendlyGray)))


  ;; math problem completes and target down
 (cond ((and (> x 61.9) (< x 62.3))
       (mod-chunk out1 sound 0 target 0)))

 ;; target up-8
 (cond ((and (> x 64.9) (< x 65.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

 ;; math problem starts
 (cond ((and (> x 65.9) (< x 66.3))
       (mod-chunk out1 sound 1)))

 ;; target down
 (cond ((and (> x 66.9) (< x 67.3))
       (mod-chunk out1 target 0)))

 ;; math problem completes
 (cond ((and (> x 67.9) (< x 68.3))
       (mod-chunk out1 sound 0)))
```

```
;; target up-9
(cond ((and (> x 69.9) (< x 70.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))


;; target down and problem starts
(cond ((and (> x 71.9) (< x 72.3))
       (mod-chunk out1 sound 1 target 0)))

;; math problem completes
(cond ((and (> x 73.9) (< x 74.3))
       (mod-chunk out1 sound 0)))

;; target up-10
(cond ((and (> x 74.9) (< x 75.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 76.9) (< x 77.3))
       (mod-chunk out1 target 0)))

;; math problem starts
(cond ((and (> x 77.9) (< x 78.3))
       (mod-chunk out1 sound 1)))

;; target up and end listening to problem-11
(cond ((and (> x 79.9) (< x 80.3))
       (mod-chunk out1 target 1 sound 0)))

;; target down
(cond ((and (> x 81.9) (< x 82.3))
       (mod-chunk out1 target 1 sound 0)))

;; target up-12
(cond ((and (> x 84.9) (< x 85.3))
       (mod-chunk out1 sound 1)))

;; end listening to math problem
(cond ((and (> x 85.9) (< x 86.3))
       (mod-chunk out1 sound 0)))

;; target down
(cond ((and (> x 86.9) (< x 87.3))
       (mod-chunk out1 target 0)))


 ;; target up and begin next problem-7
(cond ((and (> x 89.9) (< x 90.3))
       (mod-chunk out1 target 1 sound 1)
       (mod-chunk targettype1 type friendlyGray)))


 ;; math problem completes and target down
(cond ((and (> x 91.9) (< x 92.3))
       (mod-chunk out1 sound 0 target 0)))

;; target up-8
(cond ((and (> x 94.9) (< x 95.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))

;; math problem starts
(cond ((and (> x 95.9) (< x 96.3))
       (mod-chunk out1 sound 1)))

;; target down
(cond ((and (> x 96.9) (< x 97.3))
       (mod-chunk out1 target 0)))
```

```
   ;; math problem completes
   (cond ((and (> x 97.9) (< x 98.3))
          (mod-chunk out1 sound 0)))

   ;; target up-9
   (cond ((and (> x 99.9) (< x 100.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))


   ;; target down and problem starts
   (cond ((and (> x 101.9) (< x 102.3))
          (mod-chunk out1 sound 1 target 0)))

   ;; math problem completes
   (cond ((and (> x 103.9) (< x 104.3))
          (mod-chunk out1 sound 0)))

   ;; target up-10
   (cond ((and (> x 104.9) (< x 105.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type enemy)))

   ;; target down
   (cond ((and (> x 106.9) (< x 107.3))
          (mod-chunk out1 target 0)))

   ;; math problem starts
   (cond ((and (> x 107.9) (< x 108.3))
          (mod-chunk out1 sound 1)))

   ;; target up and end listening to problem-11
   (cond ((and (> x 109.9) (< x 110.3))
          (mod-chunk out1 target 1 sound 0)))

   ;; target down
   (cond ((and (> x 111.9) (< x 112.3))
          (mod-chunk out1 target 1 sound 0)))

   ;; target up-12
   (cond ((and (> x 114.9) (< x 115.3))
          (mod-chunk out1 sound 1)))

   ;; end listening to math problem
   (cond ((and (> x 115.9) (< x 116.3))
          (mod-chunk out1 sound 0)))

   ;; target down
   (cond ((and (> x 116.9) (< x 117.3))
          (mod-chunk out1 target 0)))


 ;; ending problem
   (cond ((and (> x 125.9) (< x 126.3))
          (mod-chunk out1 target nil sound nil)))

)

(clear-all)



(sgp :lf .1 :rt -1 :bll .5 :blc 20 :ga 10.0 :al 2.0)



(chunk-type count-order first second)
(chunk-type count-from start end)
(chunk-type outside sound target)
(chunk-type begin)
```

```
(chunk-type problem firstnum secondnum)
(chunk-type increment num1 counter)
(chunk-type target)
(chunk-type targettype size type)
(chunk-type check)
(chunk-type inside soundcomplete targetchecked problemcomplete 0)
(chunk-type shoot)


(add-dm
(check1 isa check)
(begin1 isa begin)
(target1 isa target)
(shoot1 isa shoot)
(inside1 isa inside soundcomplete 0 targetchecked 0 problemcomplete 0)
(targettype1 isa targettype size 0 type enemy)
(increment1 isa increment num1 0 counter 0)
(problem1 isa problem firstnum 55 secondnum 9)
;; first outside variable
(out1 isa outside sound 1 target 1)
(countfrom1 isa count-from start 0)
(chunk1 isa count-order first 1 second 2)
(chunk2 isa count-order first 2 second 3)
(chunk3 isa count-order first 3 second 4)
(chunk4 isa count-order first 4 second 5)
(chunk5 isa count-order first 5 second 6)
(chunk6 isa count-order first 6 second 7)
(chunk7 isa count-order first 7 second 8)
(chunk8 isa count-order first 8 second 9)
(chunk9 isa count-order first 9 second 10)
(chunk10 isa count-order first 10 second 11)
(chunk11 isa count-order first 11 second 12)
(chunk12 isa count-order first 12 second 13)
(chunk13 isa count-order first 13 second 14)
(chunk14 isa count-order first 14 second 15)
(chunk15 isa count-order first 15 second 16)
(chunk16 isa count-order first 16 second 17)
(chunk17 isa count-order first 17 second 18)
(chunk18 isa count-order first 18 second 19)
(chunk19 isa count-order first 19 second 20)
(chunk20 isa count-order first 20 second 21)
(chunk21 isa count-order first 21 second 22)
(chunk22 isa count-order first 22 second 23)
(chunk23 isa count-order first 23 second 24)
(chunk24 isa count-order first 24 second 25)
(chunk25 isa count-order first 25 second 26)
(chunk26 isa count-order first 26 second 27)
(chunk27 isa count-order first 27 second 28)
(chunk28 isa count-order first 28 second 29)
(chunk29 isa count-order first 29 second 30)
(chunk30 isa count-order first 30 second 31)
(chunk31 isa count-order first 31 second 32)
(chunk32 isa count-order first 32 second 33)
(chunk33 isa count-order first 33 second 34)
(chunk34 isa count-order first 34 second 35)
(chunk35 isa count-order first 35 second 36)
(chunk36 isa count-order first 36 second 37)
(chunk37 isa count-order first 37 second 38)
(chunk38 isa count-order first 38 second 39)
(chunk39 isa count-order first 39 second 40)
(chunk40 isa count-order first 40 second 41)
(chunk41 isa count-order first 41 second 42)
(chunk42 isa count-order first 42 second 43)
(chunk43 isa count-order first 43 second 44)
(chunk44 isa count-order first 44 second 45)
(chunk45 isa count-order first 45 second 46)
(chunk46 isa count-order first 46 second 47)
(chunk47 isa count-order first 47 second 48)
(chunk48 isa count-order first 48 second 49)
(chunk49 isa count-order first 49 second 50)
(chunk50 isa count-order first 50 second 51)
```

```
(chunk51 isa count-order first 51 second 52)
(chunk52 isa count-order first 52 second 53)
(chunk53 isa count-order first 53 second 54)
(chunk54 isa count-order first 54 second 55)
(chunk55 isa count-order first 55 second 56)
(chunk56 isa count-order first 56 second 57)
(chunk57 isa count-order first 57 second 58)
(chunk58 isa count-order first 58 second 59)
(chunk59 isa count-order first 59 second 60)
(chunk60 isa count-order first 60 second 61)
(chunk61 isa count-order first 61 second 62)
(chunk62 isa count-order first 62 second 63)
(chunk63 isa count-order first 63 second 64)
(chunk64 isa count-order first 64 second 65)
(chunk65 isa count-order first 65 second 66)
(chunk66 isa count-order first 66 second 67)
(chunk67 isa count-order first 67 second 68)
(chunk68 isa count-order first 68 second 69)
(chunk69 isa count-order first 69 second 70)
(chunk70 isa count-order first 70 second 71)
(chunk71 isa count-order first 71 second 72)
(chunk72 isa count-order first 72 second 73)
(chunk73 isa count-order first 73 second 74)
(chunk74 isa count-order first 74 second 75)
(chunk75 isa count-order first 75 second 76)
(chunk76 isa count-order first 76 second 77)
(chunk77 isa count-order first 77 second 78)
(chunk78 isa count-order first 78 second 79)
(chunk79 isa count-order first 79 second 80)
(chunk80 isa count-order first 80 second 81)
(chunk81 isa count-order first 81 second 82)
(chunk82 isa count-order first 82 second 83)
(chunk83 isa count-order first 83 second 84)
(chunk84 isa count-order first 84 second 85)
(chunk85 isa count-order first 85 second 86)
(chunk86 isa count-order first 86 second 87)
(chunk87 isa count-order first 87 second 88)
(chunk88 isa count-order first 88 second 89)
(chunk89 isa count-order first 89 second 90)
(chunk90 isa count-order first 90 second 91)
(chunk91 isa count-order first 91 second 92)
(chunk92 isa count-order first 92 second 93)
(chunk93 isa count-order first 93 second 94)
(chunk94 isa count-order first 94 second 95)
(chunk95 isa count-order first 95 second 96)
(chunk96 isa count-order first 96 second 97)
(chunk97 isa count-order first 97 second 98)
(chunk98 isa count-order first 98 second 99)
(chunk99 isa count-order first 99 second 100)
(chunk100 isa count-order first 100 second 101))


;; we have three competing productions (begin-task, scan-for-targets and listen-for-problem)
;; which represent competing goals of the
;; multi-task environment.  The scan-for-targets however has the highest priority
;; so we will do that one over the listen-for-sound production, which basically
;; is listening for each math problem.

(P begin-task
   =begin1>
      isa begin
    =out1>
      isa outside
      sound =sound
      target =target
==>
   !bind!     =newvar (actr-time)
   !output!   =newvar
   !eval!     (timetracker =newvar)
   !push! =out1
)
```

```
(P check-target
   =check1>
      isa check
   =outside1>
      isa outside
      sound 0
      target =target
   =target1>
      isa target
==>

     !bind!     =newvar (actr-time)
     !output!   =newvar
     !eval!     (timetracker =newvar)

      !pop!
)
(P cant-check-target
   =check1>
      isa check
   =out1>
      isa outside
      sound 1
      target =target
   =shoot1>
      isa shoot

==>
  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push! =shoot1
)
(P shoot-decision
   =shoot1>
     isa shoot
   =out1>
     isa outside
     target =target
==>

    =out1>
       target 0
    !pop!
    !pop!
    !pop!
)
;;(P problem-complete
;;   =out1>
;;      isa outside
;;      sound 0
;;      target 0
;;   =inside1>
;;      isa inside
;;      problemcomplete 0
;;   =problem1>
;;       isa problem
;;   =target1>
;;      isa target
;;==>
;;
;;  !bind!     =newvar (actr-time)
;;  !output!   =newvar
;;  !eval!     (timetracker =newvar)
;;
;;
;;    !push!       =problem1
;;
;;)
```

```
(P scan-for-targets
   =out1>
      isa outside
      sound =sound
      target 1
   =inside1>
      isa inside
      problemcomplete =problemcomplete
   =problem1>
       isa problem
   =target1>
      isa target
==>

  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push!  =target1
)
(P look-at-problem
   =out1>
      isa outside
      sound 1
      target 0
   =inside1>
       isa inside
       problemcomplete =problemcomplete
   =problem1>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
   =target1>
       isa target
==>


     !bind!     =newvar (actr-time)
     !output!   =newvar
     !eval!     (timetracker =newvar)


     !push!     =problem1
)



(P identify-target
   =target1>
     isa target
   =targettype1>
     isa targettype
     size =size
     type =type
   =check1>
     isa check
==>

   !bind!     =newvar (actr-time)
   !output!   =newvar
   !push! =check1
)

(P remember-problem
   =problem1>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
   =countfrom1>
```

```
       isa count-from
       start =start
   =out1>
       isa outside
       sound =sound
       target =target
==>


    !bind!     =newvar (actr-time)
     !output!   =newvar
     !output!   =firstnum
     !output!   =secondnum
     !eval!     (timetracker =newvar)

     !push! =countfrom1


)


(P start-counting
   =countfrom1>
      ISA         count-from
      start       0
    =goal>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
    =out1>
       isa outside
       sound =sound
       target =target
==>

    =countfrom1>
       start   =firstnum

   !output!       ("~S" =firstnum)
   !output!       ("~S" =secondnum)

   !bind!     =newvar (actr-time)
   !output!   =newvar

)

(P count
   =countfrom1>
      isa count-from
      start =firstnum
   =problem1>
      isa problem
      firstnum =firstnum
      secondnum =secondnum
   =increment1>
      isa increment
      num1 =num1
      counter =thecounter
   =out1>
      isa outside
      target =target
      sound =sound
==>
     !bind!     =newvar (actr-time)
     !output!   =newvar

   !bind!         =newtime (1+ =thecounter)

   =increment1>
      num1 =firstnum
      counter =newtime
```

```
   =increment1>
      !push! =increment1
      !output! =firstnum
)
(P increment
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
       target =target
==>
    !bind!     =newvar (actr-time)
    !output!   =newvar
    !output!   =target
    !eval!     (timetracker =newvar)

    !bind!         =newTime (1+ =thecounter)
    =increment1>
       isa increment
       num1 =secondnum
     counter =newtime
    !output! =secondnum



)
(P no-time-to-increment
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
       target 1
==>
    !bind!     =newvar (actr-time)
    !output!   =newvar
    !bind!         =newTime (1+ =thecounter)
    !eval!     (timetracker =newvar)
    !pop!
    !pop!
    !pop!


)

(P answer
   =increment1>
      isa increment
      num1 =num1
      counter =counter
   =countfrom>
       isa count-order
       first =num1
```

```
        second =secondnumber
    =problem1>
        isa problem
        secondnum =counter

==>


    !output! =secondnumber
    !eval! (zero)

    !pop!
    !pop!
    !pop!
    !pop!


)

(spp ANSWER :Q 1.0 :R 1.0 :A 0.05 :B 0 :STRENGTH 0.0 :VALUE 1)
(spp SCAN-FOR-TARGETS :B 1)
(spp Look-at-problem :B 1)



(goal-focus begin1)
```

Appendix D. ACT-R Code for 2-second Exposure Time, No Load

```
;; Troy Kelley
;; Army Research Laboratory, September 2002

;; This is code that simulates data taken from a study called -The Effect of Cognitive Load
;; and Target Characteristics on Soldier Shooting Performance and Friendly Targets Engaged- by
;; David R.  Scribner
;;

;; This is our time tracker function which will modify chunks at certain time intervals
;; to simulate the outside world

;;(defun check (x))

(defun zero ()
  (mod-chunk inside1 problemcomplete 1))

;; 2 second cycle


(defun timetracker (x)
    ;; at zero time a math problem starts and a target pops up - this is set in our declarative
    ;; memory


    ;; target down and math problem completes
   (cond ((and (> x 1.9) (< x 2.2))
          (mod-chunk out1 target 0 sound 0)))


   ;; target up - 2
   (cond ((and (> x 4.9) (< x 5.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

    ;; target down
   (cond ((and (> x 6.9) (< x 7.3))
          (mod-chunk out1 target 0)))



   ;; target up - 3
   (cond ((and (> x 9.9) (< x 10.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

   ;; target down
   (cond ((and (> x 11.9) (< x 12.3))
          (mod-chunk out1 target 0)))

   ;; target up -4
   (cond ((and (> x 14.9) (< x 15.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type enemy)))

   ;; target down
   (cond ((and (> x 16.9) (< x 17.3))
          (mod-chunk out1 target 0)))

   ;; target up
   (cond ((and (> x 19.9) (< x 20.3))
          (mod-chunk out1 target 1)
          ))

   ;; target down
```

```
(cond ((and (> x 21.9) (< x 22.3))
       (mod-chunk out1 target 0)))


;; target up-6
(cond ((and (> x 24.9) (< x 25.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))


;; target down
(cond ((and (> x 26.9) (< x 27.3))
       (mod-chunk out1 target 0)))


;; target up
(cond ((and (> x 29.9) (< x 30.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))


 ;; math problem completes and target down
(cond ((and (> x 31.9) (< x 32.3))
       (mod-chunk out1 target 0)
       ))

;; target up-8
(cond ((and (> x 34.9) (< x 35.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))


;; target down
(cond ((and (> x 36.9) (< x 37.3))
       (mod-chunk out1 target 0)))


;; target up-9
(cond ((and (> x 39.9) (< x 40.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type friendlyGray)))


;; target down
(cond ((and (> x 41.9) (< x 42.3))
       (mod-chunk out1 target 0)))


;; target up-10
(cond ((and (> x 44.9) (< x 45.3))
       (mod-chunk out1 target 1)
       (mod-chunk targettype1 type enemy)))

;; target down
(cond ((and (> x 46.9) (< x 47.3))
       (mod-chunk out1 target 0)))

;; target up
(cond ((and (> x 49.9) (< x 50.3))
       (mod-chunk out1 target 1)
       ))

;; target down
(cond ((and (> x 51.9) (< x 52.3))
       (mod-chunk out1 target 1 sound 0)))

;; target up-12
(cond ((and (> x 54.9) (< x 55.3))
       (mod-chunk out1 sound 1)))

;; target down
```

```
 (cond ((and (> x 56.9) (< x 57.3))
        (mod-chunk out1 target 0)))

;; target up
 (cond ((and (> x 59.9) (< x 60.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type friendlyGray)))


  ;; target down
 (cond ((and (> x 61.9) (< x 62.3))
        (mod-chunk out1 target 0)
        ))

 ;; target up-8
 (cond ((and (> x 64.9) (< x 65.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type friendlyGray)))


 ;; target down
 (cond ((and (> x 66.9) (< x 67.3))
        (mod-chunk out1 target 0)))


 ;; target up-9
 (cond ((and (> x 69.9) (< x 70.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type friendlyGray)))


 ;; target down
 (cond ((and (> x 71.9) (< x 72.3))
        (mod-chunk out1 target 0)))

 ;; target up-10
 (cond ((and (> x 74.9) (< x 75.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type enemy)))

 ;; target down
 (cond ((and (> x 76.9) (< x 77.3))
        (mod-chunk out1 target 0)))


 ;; target up
 (cond ((and (> x 79.9) (< x 80.3))
        (mod-chunk out1 target 1)
        ))

 ;; target down
 (cond ((and (> x 81.9) (< x 82.3))
        (mod-chunk out1 target 1 sound 0)))

 ;; target up-12
 (cond ((and (> x 84.9) (< x 85.3))
        (mod-chunk out1 sound 1)))


 ;; target down
 (cond ((and (> x 86.9) (< x 87.3))
        (mod-chunk out1 target 0)))


  ;; target up
 (cond ((and (> x 89.9) (< x 90.3))
        (mod-chunk out1 target 1)
        (mod-chunk targettype1 type friendlyGray)))


  ;; target down
```

```
   (cond ((and (> x 91.9) (< x 92.3))
          (mod-chunk out1 target 0)
          ))

   ;; target up-8
   (cond ((and (> x 94.9) (< x 95.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))

   ;; target down
   (cond ((and (> x 96.9) (< x 97.3))
          (mod-chunk out1 target 0)))

   ;; target up-9
   (cond ((and (> x 99.9) (< x 100.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type friendlyGray)))


   ;; target down
   (cond ((and (> x 101.9) (< x 102.3))
          (mod-chunk out1 target 0)))

   ;; target up-10
   (cond ((and (> x 104.9) (< x 105.3))
          (mod-chunk out1 target 1)
          (mod-chunk targettype1 type enemy)))

   ;; target down
   (cond ((and (> x 106.9) (< x 107.3))
          (mod-chunk out1 target 0)))


   ;; target up
   (cond ((and (> x 109.9) (< x 110.3))
          (mod-chunk out1 target 1)
          ))

   ;; target down
   (cond ((and (> x 111.9) (< x 112.3))
          (mod-chunk out1 target 1 sound 0)))

   ;; target up-12
   (cond ((and (> x 114.9) (< x 115.3))
          (mod-chunk out1 sound 1)))

   ;; target down
   (cond ((and (> x 116.9) (< x 117.3))
          (mod-chunk out1 target 0)))


 ;; ending problem
   (cond ((and (> x 125.9) (< x 126.3))
          (mod-chunk out1 target nil)))

)

(clear-all)



(sgp :lf .1 :rt 0.0 :bll .5 :blc 20 :ga 10.0 :al 2.0)



(chunk-type count-order first second)
(chunk-type count-from start end)
(chunk-type outside target)
(chunk-type begin)
(chunk-type problem firstnum secondnum)
(chunk-type increment num1 counter)
```

```
(chunk-type target)
(chunk-type targettype size type)
(chunk-type check)
(chunk-type inside soundcomplete targetchecked problemcomplete 0)
(chunk-type shoot)


(add-dm
(check1 isa check)
(begin1 isa begin)
(target1 isa target)
(shoot1 isa shoot)
(inside1 isa inside soundcomplete 0 targetchecked 0 problemcomplete 0)
(targettype1 isa targettype size 0 type enemy)
(increment1 isa increment num1 0 counter 0)
(problem1 isa problem firstnum 55 secondnum 9)
;; first outside variable
(out1 isa outside target 1)
(countfrom1 isa count-from start 0)
(chunk1 isa count-order first 1 second 2)
(chunk2 isa count-order first 2 second 3)
(chunk3 isa count-order first 3 second 4)
(chunk4 isa count-order first 4 second 5)
(chunk5 isa count-order first 5 second 6)
(chunk6 isa count-order first 6 second 7)
(chunk7 isa count-order first 7 second 8)
(chunk8 isa count-order first 8 second 9)
(chunk9 isa count-order first 9 second 10)
(chunk10 isa count-order first 10 second 11)
(chunk11 isa count-order first 11 second 12)
(chunk12 isa count-order first 12 second 13)
(chunk13 isa count-order first 13 second 14)
(chunk14 isa count-order first 14 second 15)
(chunk15 isa count-order first 15 second 16)
(chunk16 isa count-order first 16 second 17)
(chunk17 isa count-order first 17 second 18)
(chunk18 isa count-order first 18 second 19)
(chunk19 isa count-order first 19 second 20)
(chunk20 isa count-order first 20 second 21)
(chunk21 isa count-order first 21 second 22)
(chunk22 isa count-order first 22 second 23)
(chunk23 isa count-order first 23 second 24)
(chunk24 isa count-order first 24 second 25)
(chunk25 isa count-order first 25 second 26)
(chunk26 isa count-order first 26 second 27)
(chunk27 isa count-order first 27 second 28)
(chunk28 isa count-order first 28 second 29)
(chunk29 isa count-order first 29 second 30)
(chunk30 isa count-order first 30 second 31)
(chunk31 isa count-order first 31 second 32)
(chunk32 isa count-order first 32 second 33)
(chunk33 isa count-order first 33 second 34)
(chunk34 isa count-order first 34 second 35)
(chunk35 isa count-order first 35 second 36)
(chunk36 isa count-order first 36 second 37)
(chunk37 isa count-order first 37 second 38)
(chunk38 isa count-order first 38 second 39)
(chunk39 isa count-order first 39 second 40)
(chunk40 isa count-order first 40 second 41)
(chunk41 isa count-order first 41 second 42)
(chunk42 isa count-order first 42 second 43)
(chunk43 isa count-order first 43 second 44)
(chunk44 isa count-order first 44 second 45)
(chunk45 isa count-order first 45 second 46)
(chunk46 isa count-order first 46 second 47)
(chunk47 isa count-order first 47 second 48)
(chunk48 isa count-order first 48 second 49)
(chunk49 isa count-order first 49 second 50)
(chunk50 isa count-order first 50 second 51)
(chunk51 isa count-order first 51 second 52)
(chunk52 isa count-order first 52 second 53)
```

```
(chunk53 isa count-order first 53 second 54)
(chunk54 isa count-order first 54 second 55)
(chunk55 isa count-order first 55 second 56)
(chunk56 isa count-order first 56 second 57)
(chunk57 isa count-order first 57 second 58)
(chunk58 isa count-order first 58 second 59)
(chunk59 isa count-order first 59 second 60)
(chunk60 isa count-order first 60 second 61)
(chunk61 isa count-order first 61 second 62)
(chunk62 isa count-order first 62 second 63)
(chunk63 isa count-order first 63 second 64)
(chunk64 isa count-order first 64 second 65)
(chunk65 isa count-order first 65 second 66)
(chunk66 isa count-order first 66 second 67)
(chunk67 isa count-order first 67 second 68)
(chunk68 isa count-order first 68 second 69)
(chunk69 isa count-order first 69 second 70)
(chunk70 isa count-order first 70 second 71)
(chunk71 isa count-order first 71 second 72)
(chunk72 isa count-order first 72 second 73)
(chunk73 isa count-order first 73 second 74)
(chunk74 isa count-order first 74 second 75)
(chunk75 isa count-order first 75 second 76)
(chunk76 isa count-order first 76 second 77)
(chunk77 isa count-order first 77 second 78)
(chunk78 isa count-order first 78 second 79)
(chunk79 isa count-order first 79 second 80)
(chunk80 isa count-order first 80 second 81)
(chunk81 isa count-order first 81 second 82)
(chunk82 isa count-order first 82 second 83)
(chunk83 isa count-order first 83 second 84)
(chunk84 isa count-order first 84 second 85)
(chunk85 isa count-order first 85 second 86)
(chunk86 isa count-order first 86 second 87)
(chunk87 isa count-order first 87 second 88)
(chunk88 isa count-order first 88 second 89)
(chunk89 isa count-order first 89 second 90)
(chunk90 isa count-order first 90 second 91)
(chunk91 isa count-order first 91 second 92)
(chunk92 isa count-order first 92 second 93)
(chunk93 isa count-order first 93 second 94)
(chunk94 isa count-order first 94 second 95)
(chunk95 isa count-order first 95 second 96)
(chunk96 isa count-order first 96 second 97)
(chunk97 isa count-order first 97 second 98)
(chunk98 isa count-order first 98 second 99)
(chunk99 isa count-order first 99 second 100)
(chunk100 isa count-order first 100 second 101))


;; we have three competing productions (begin-task, scan-for-targets and listen-for-problem)
;; which represent competing goals of the
;; multi-task environment.  The scan-for-targets however has the highest priority
;; so we will do that one over the listen-for-sound production, which basically
;; is listening for each math problem.

(P begin-task
   =begin1>
      isa begin
    =out1>
      isa outside
      target =target
==>
   !bind!     =newvar (actr-time)
   !output!   =newvar
   !eval!     (timetracker =newvar)
   !push! =out1
)

(P check-target
   =check1>
```

```
      isa check
   =outside1>
      isa outside
      target 1
   =target1>
      isa target
==>

     !bind!     =newvar (actr-time)
     !output!   =newvar
     !eval!     (timetracker =newvar)

      !pop!
)
(P cant-check-target
   =check1>
      isa check
   =out1>
      isa outside
      target 0
   =shoot1>
      isa shoot

==>
  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push! =shoot1
)
(P shoot-decision
   =shoot1>
     isa shoot
   =out1>
     isa outside
     target =target
==>

    =out1>
       target 0
    !pop!
    !pop!
    !pop!
)
(P problem-complete
   =out1>
      isa outside
      target 0
   =inside1>
      isa inside
      problemcomplete 0
   =problem1>
       isa problem
   =target1>
      isa target
==>

  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)


    !push!       =problem1

)

(P scan-for-targets
   =out1>
      isa outside
      target 1
   =inside1>
```

```
      isa inside
      problemcomplete =problemcomplete
   =problem1>
       isa problem
   =target1>
      isa target
==>

  !bind!     =newvar (actr-time)
  !output!   =newvar
  !eval!     (timetracker =newvar)

   !push!  =target1
)
;;(P listen-for-problem
;;   =out1>
;;      isa outside
;;      target 0
;;   =inside1>
;;       isa inside
;;       problemcomplete =problemcomplete
;;   =problem1>
;;       isa problem
;;   =target1>
;;       isa target
;;==>
;;     !bind!     =newvar (actr-time)
;;     !output!   =newvar
;;     !eval!     (timetracker =newvar)
;;)



(P identify-target
   =target1>
     isa target
   =targettype1>
     isa targettype
     size =size
     type =type
   =check1>
     isa check
==>

   !bind!     =newvar (actr-time)
   !output!   =newvar
   !push! =check1
)

(P remember-problem
   =problem1>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
   =countfrom1>
       isa count-from
       start =start
   =out1>
       isa outside
       target =target
==>


    !bind!     =newvar (actr-time)
     !output!   =newvar
     !output!   =firstnum
     !output!   =secondnum
     !eval!     (timetracker =newvar)

     !push! =countfrom1
```

```
)


(P start-counting
   =countfrom1>
      ISA         count-from
      start       0
    =goal>
       isa problem
       firstnum =firstnum
       secondnum =secondnum
    =out1>
       isa outside
       target =target
==>

    =countfrom1>
       start   =firstnum

   !output!       ("~S" =firstnum)
   !output!       ("~S" =secondnum)

   !bind!     =newvar (actr-time)
   !output!   =newvar

)

(P count
   =countfrom1>
      isa count-from
      start =firstnum
   =problem1>
      isa problem
      firstnum =firstnum
      secondnum =secondnum
   =increment1>
      isa increment
      num1 =num1
      counter =thecounter
   =out1>
      isa outside
      target =target
==>
     !bind!     =newvar (actr-time)
     !output!   =newvar

   !bind!         =newtime (1+ =thecounter)

   =increment1>
      num1 =firstnum
      counter =newtime
   =increment1>
      !push! =increment1
      !output! =firstnum
)
(P increment
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
```

```
       target =target
==>
    !bind!     =newvar (actr-time)
    !output!   =newvar
    !output!   =target

    !bind!         =newTime (1+ =thecounter)
    =increment1>
       isa increment
       num1 =secondnum
     counter =newtime
    !output! =secondnum

)
(P no-time-to-increment
   =increment1>
      isa increment
      num1 =num1
     counter =thecounter
   =countfrom>
      isa count-order
      first =num1
      second =secondnum
   =problem1>
       isa problem
       secondnum =thesecond
   =out1>
       isa outside
       target 1
==>
    !bind!     =newvar (actr-time)
    !output!   =newvar
    !bind!         =newTime (1+ =thecounter)
    !eval!     (timetracker =newvar)
    !pop!
    !pop!
    !pop!


)

(P answer
   =increment1>
      isa increment
      num1 =num1
      counter =counter
   =countfrom>
       isa count-order
       first =num1
       second =secondnumber
   =problem1>
       isa problem
       secondnum =counter

==>


   !output! =secondnumber
   !eval! (zero)

   !pop!
   !pop!
   !pop!


)

(spp ANSWER :Q 1.0 :R 1.0 :A 0.05 :B 0 :STRENGTH 0.0 :VALUE 1)
(spp SCAN-FOR-TARGETS :B 1)
;;(spp listen-for-problem :B 1)
```

```
(goal-focus begin1)
```

NO. OF
COPIES ORGANIZATION

1 ADMINISTRATOR
DEFENSE TECHNICAL INFO CTR
ATTN DTIC OCA
8725 JOHN J KINGMAN RD STE 0944
FT BELVOIR VA 22060-6218

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI IS R REC MGMT
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI OK TECH LIB
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL D D SMITH
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 US ARMY RESEARCH LABORATORY
ATTN AMSRL HR M M STRUB
6359 WALKER LANE STE 100
ALEXANDRIA VA 22310

1 ARL HRED USAFAS FLD ELMT
ATTN AMSRL HR MF L PIERCE
BLDG 3040 RM 220
FORT SILL OK 73503-5600

1 ARL HRED AMCOM FLD ELMT
ATTN AMSRL HR MD T COOK
BLDG 5400 RM C242
REDSTONE ARS AL 35898-7290

1 ARL HRED USAADASCH FLD ELMT
ATTN ATSA CD
ATTN AMSRL HR ME K REYNOLDS
5800 CARTER ROAD
FORT BLISS TX 79916-3802

1 ARL HRED AMCOM FLD ELMT
ATTN AMSRL HR MI
BLDG 5464 RM 202
REDSTONE ARSENAL AL 35898-5000

1 US ARMY RESEARCH LABORATORY
ATTN AMSRL HR MM N VAUSE
2250 STANLEY RD STE 322
FT SAM HOUSTON TX 78234

NO. OF
COPIES ORGANIZATION

1 ARL HRED ARDEC FLD ELMT
ATTN AMSRL HR MG R SPINE
BUILDING 333
PICATINNY ARSENAL NJ 07806-5000

1 ARL HRED ARMC FLD ELMT
ATTN AMSRL HR MH C BURNS
BLDG 1002 ROOM 123
1ST CAVALRY REGIMENT RD
FT KNOX KY 40121

1 ARL HRED ATEC FLD ELMT
ATTN AMSRL HR MR H DENNY
ATEC CSTE PM ARL
4501 FORD AVE RM 870
ALEXANDRIA VA 22302-1458

1 ARL HRED AVNC FLD ELMT
ATTN AMSRL HR MJ D DURBIN
BLDG 4506 (DCD) RM 107
FT RUCKER AL 36362-5000

1 ARL HRED CECOM FLD ELMT
ATTN AMSRL HR ML J MARTIN
MYER CENTER RM 2D311
FT MONMOUTH NJ 07703-5630

1 ARL HRED FT BELVOIR FLD ELMT
ATTN AMSRL HR MK J REINHART
10170 BEACH RD
FORT BELVOIR VA 22060-5800

1 ARL HRED FT HOOD FLD ELMT
ATTN AMSRL HR MV HQ USAOTC
S MIDDLEBROOKS
91012 STATION AVE RM 348
FT HOOD TX 76544-5073

1 ARL HRED FT HUACHUCA FLD ELMT
ATTN AMSRL HR MY M BARNES
2520 HEALY AVE
BLDG 51005 STE 1172
FT HUACHUCA AZ 85613

1 ARL HRED HFID FLD ELMT
ATTN AMSRL HR MP D UNGVARSKY
BATTLE CMD BATTLE LAB
415 SHERMAN AVE UNIT 3
FT LEAVENWORTH KS 66027-2326

1 ARL HRED FLW FLD ELMT
ATTN AMSRL HR MZ A DAVISON
320 MANSCEN LOOP STE 166
FT LEONARD WOOD MO 65473-8929

NO. OF
COPIES ORGANIZATION

1 ARL HRED NATICK FLD ELMT
ATTN AMSRL HR MQ M R FLETCHER
NATICK SOLDIER CTR AMSSB RSS E
BLDG 3 RM 341
NATICK MA 01760-5020

1 ARL HRED SC&FG FLD ELMT
ATTN AMSRL HR MS R ANDERS
SIGNAL TOWERS RM 303A
FORT GORDON GA 30905-5233

1 ARL HRED PEO STRI FLD ELMT
ATTN AMSRL HR MT A GALBAVY
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3276

1 ARL HRED TACOM FLD ELMT
ATTN AMSRL HR MU M SINGAPORE
6501 E 11 MILE RD MAIL STOP 284
BLDG 200A 2ND FL RM 2104
WARREN MI 48397-5000

1 ARL HRED USAIC FLD ELMT
ATTN AMSRL HR MW E REDDEN
BLDG 4 ROOM 332
FT BENNING GA 31905-5400

1 ARL HRED USASOC FLD ELMT
ATTN AMSRL HR MN R SPENCER
DCSFDI HF
HQ USASOC BLDG E2929
FORT BRAGG NC 28310-5000

1 CDR AMC - FAST
JRTC & FORT POLK
ATTN AFZX GT DR J AINSWORTH
CMD SCIENCE ADVISOR G3
FORT POLK LA 71459-5355

ABERDEEN PROVING GROUND

2 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRL CI OK (TECH LIB)
BLDG 305 APG AA

1 LIBRARY
ARL HRED
BLDG 459

2 ARL HRED
ATTN AMSRL HR MB F PARAGALLO
AMSRL HR MC J HAWLEY
BLDG 459
APG-AA