This file provides the task environment for the Nilsen menu selection experiment. The experiment is Experiment #2 detailed in Erik Nilsen's dissertation entitled 'Perceptual Motor Control In Human-Computer Interaction', which is identical to CSMIL Tech. Report #37.

The task environment was built on Dave's mini-model, which did the following: the following is a little model for a simple choice RT task. put up a ready signal (a cross) for one second and provide a tone signal as well have a delay of half a second then put up a digit that is black in color to the right and below the ready signal then wait for a keystroke check that the keystroke matches the digit, and update an average RT change the color of the digit to white for half a second then remove the digit wait for another second, and then repeat

Anthony Hornof modified the following:
6/23/95 split STIMULUS-DIGIT-VALUE and STIMULUS-DIGIT-OBJECT into 1 and 2, changed procedures to deal with both objects, such as DO-DISPLAY-STIMULUS. 7/10/95 modified DO-TRIAL-STATS so it prints as it goes.
7/23/95 converted *1-STIMULUS-DIGIT-VALUE and *1-STIMULUS-DIGIT-OBJECT by removing the 'DIGIT-' and adding a 3rd, 4th, and 5th stimulus. Added TARGET-STIMULUS, which corresponds to which object the subject is primed for outside of the simulation.
11/13/95 - Note that I have changed Dave's methodology for triggering stimuli. How:
2/23/96 - Add function that takes in distance between menu items and distance from screen and returns visual angle.
- Make target item appear in the device.
- renamed DO-DISPLAY-STIMULUS to DISPLAY-MENU
5/31/96 - Have the device verify correct mouse locations.
- Have the device look for mouse up and down, not punches.

Menu Selection Models Built Using
The EPIC Simulation Software

Copyright (c) 1998
The Regents of the University of Michigan
ALL RIGHTS RESERVED

Permission is granted to use this software for noncommercial education and research purposes and to modify this software for such purposes only. Permission to copy or distribute this software or any modification is not granted.

This software is provided as is, without representation as to its fitness for any purpose, and without warranty of any kind, either express or implied, including without limitation the implied warranties of merchantability and fitness for a particular purpose. The Regents of the University of Michigan shall not be liable for any damages, including special, indirect, incidental, or consequential damages, with respect to any claim arising out of or in connection with the use of the software, even if it has been or is hereafter advised of the possibility of such damages.

The authors of this software expect that the proper attribution will be made in any public reports of work conducted with this
software. The report should distinguish between results due to the
architecture represented in this software and the task-specific encodings,
parameter estimates, and production-rules used to construct a model
using this software. It should also include the date or version of this
software that was used, and identify any substantive modifications.

This menu selection simulation is under development and may change prior
to publication of this research.

Anthony J. Hornof, Artificial Intelligence Laboratory
Electrical Engineering and Computer Science Department
University of Michigan, Ann Arbor, MI 48109-2110
hornof@eecs.umich.edu

*********************************************************
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; What do these do?  Do I need them?  -ajh 3/21/96
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Other global variable definitions.
;; Control variables
(DEFVAR *TRIAL-CLEAR-WINDOWS T)
(DEFVAR *STOP-IF-ERROR T)
;; Static variables
;; (DEFVAR *STIMULUS NIL) ;holds target object (no longer used? -ajh 7/5/96)
(DEFVAR *GO-BOX-OBJECT NIL)
(DEFVAR *PRECUE-OBJECT NIL)
;; Stimuli variables
(DEFVAR *MENU-TEXT-LIST NIL)
(DEFVAR *MENU-VISOBJ-LIST NIL)
(DEFVAR *MENU-ITEM-COUNTER 0)
(DEFVAR *CURSOR NIL) ; holds CURSOR object
(DEFVAR *TRACKING-ORIGIN-X 0)
(DEFVAR *TRACKING-ORIGIN-Y 0)
;; Screen position of go-box and precue. Set below after determining start
position of menu and visual angle of menu items.
(DEFVAR *GO-BOX-X NIL)
(DEFVAR *GO-BOX-Y NIL)
(DEFVAR *PRECUE-X NIL)
(DEFVAR *PRECUE-Y NIL)
(DEFVAR *MENU-START-X 0)
(DEFVAR *MENU-START-Y 0)
(DEFVAR *TARGET-TEXT NIL) ;; SET BELOW!
(DEFVAR *TARGET-OBJECT NIL)
;; used for setting the next sweep items in epicfns
(DEFVAR *CURRENT-NEXT-OBJ-DOWN NIL)
(DEFVAR *CURRENT-NEXT-OBJ-UP NIL)
(DEFVAR *NEXT-OBJ-DOWN-CREATED-AT-TIME 0)
(DEFVAR *NEXT-OBJ-UP-CREATED-AT-TIME 0)
(DEFVAR *NEXT-OBJ-DOWN-CREATED-FOR-MENU-ITEM NIL)
(DEFVAR *NEXT-OBJ-UP-CREATED-FOR-MENU-ITEM NIL)
;; (DEFVAR *NEXT-SWEEP-LOCATION NIL) ;; no longer used 4/15/01
(DEFVAR *CURRENT-GAZE-MENU-ITEM NIL)
;; Statistics variables

(eval-when (:compile-toplevel)
  (proclaim '(optimize (speed 2)(safety 1)(space 1)(debug 3)))
  (proclaim '(optimize (speed 3)(safety 0)(space 0)(debug 0))))

(eval-when (:compile-toplevel :load-toplevel :execute)
  (use-package '(
    "UTILITY-FUNCS"
    "WINDOW-FUNCS"
    "PPS"
    "CL-USER")
)
(DEFVAR *MENU-APPEARS-TIME 0)
(DEFVAR *SERIAL-POSITION 0)
(DEFVAR *CURRENT-RT 0)
(DEFVAR *GRAND-TOT-N 0)
(DEFVAR *GRAND-AVG-RT 0)
(DEFVAR *GRAND-TOTAL-RT 0)
(DEFVAR *TRIAL 0)
(DEFVAR *TIMING-IN-PROGRESS NIL) ;; Nil before timing. T during timed trial.
(DEFVAR *RESPONSE-TIME 0)
(DEFVAR *FIXATION-POSITIONS NIL) ;; Eye movement locations for this trial

;; cumulative totals
(DEFVAR *MAX-NUM-OF-MENU-ITEMS 12)
(DEFVAR *CUMULATIVE-RT-FOR-POS (MAKE-ARRAY *MAX-NUM-OF-MENU-ITEMS))
(DEFVAR *NUM-TRIALS-FOR-POS (MAKE-ARRAY *MAX-NUM-OF-MENU-ITEMS))

;; Eye movement data
(defvar *output-eyes-trace) ;; temporary start to adding another output file. 9/20/02

;; From -11 to 11, offset by 11
;; Observed: -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10 11 12
;; Put in slot: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
(DEFVAR *ARRAY-SIZE 24) ;; For arrays holding fixation stats
(DEFVAR *FIRST-SACCADE-DISTANCE (MAKE-ARRAY *ARRAY-SIZE))
(DEFVAR *MIDDLE-SACCADE-DISTANCE (MAKE-ARRAY *ARRAY-SIZE))
(DEFVAR *FINAL-SACCADE-DISTANCE (MAKE-ARRAY *ARRAY-SIZE))

;;; The dimensions of the following two arrays are:
;;; Dimension 1 = target location, dimension 2 = menu length
(DEFVAR *NUM-FIXATIONS-AS-FUNCTION-OF-TARGET-LOCATION (MAKE-ARRAY (LIST *ARRAY-SIZE *ARRAY-SIZE)))
(DEFVAR *FIRST-FIXATION-AS-FUNCTION-OF-TARGET-LOCATION (MAKE-ARRAY (LIST *ARRAY-SIZE *ARRAY-SIZE)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Menu experiment global variables.
;; Change here or in code to simulate different experimental conditions.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun INITIALIZE-EXPERIMENT ()
  ;; Run once per experiment, not for each trial.
  ;; These need to be reset in case previous block was interrupted.
  (setq *TRIAL 0)
  (setq *TIMING-IN-PROGRESS NIL)
  (setq *TARGET-TEXT 1) ;; Initial target is this number + 1
  (setq *MENU-TEXT-LIST NIL)
  (setq *MENU-VISOBJ-LIST NIL)
  (setq *MENU-ITEM-COUNTER 0)
  (setq *GO-BOX-OBJECT NIL)
  (setq *PRECUE-OBJECT NIL)
  (setq *TARGET-OBJECT NIL) ;; Would rather reset every trial. -ajh 10/27/96
  (setq *STIMULUS NIL)
  (setq *CURSOR NIL)
  (setq *MENU-START-X 0)
  (setq *MENU-START-Y 3)

  (INITIALIZE-STATS)
  ;; (INITIALIZE-TRIAL)  ;; *** A bunch of stuff is set here. -ajh-5/18/01

  ;; Determine screen position of go-box and precue. Should only have to
  ;; change if our assumptions of Nilsen’s positions are incorrect.
  (setq *GO-BOX-Y (+ *MENU-START-Y *MENU-ITEM-ANGLE)) ;; until 6/6/96
  (setq *GO-BOX-X *MENU-START-X) ;; until 6/6/96
  (setq *GO-BOX-Y (+ *MENU-START-Y (* .1 *MENU-ITEM-ANGLE))) ;; so cursor on target.
  (setq *GO-BOX-X *MENU-START-X) ;; This was just fine, and simpler. 8/16/98 -ajh

  ;; after Nilsen 6/6/96 email, to put cursor just inside first item...
  (setq *GO-BOX-Y (+ *MENU-START-Y (* .45 *MENU-ITEM-ANGLE)))
  (setq *PRECUE-X *MENU-START-X)

  ;; the 1.5 offset is a guess, but it shouldn’t affect outcome provided it is above
  ;; the go box.
(DEFUN INITIALIZE-STATS ()
  "Called by INITIALIZE-EXPERIMENT."

  (SETQ *CURRENT-RT 0)
  (SETQ *GRAND-TOT-N 0)
  (SETQ *GRAND-AVG-RT 0)
  (SETQ *GRAND-TOTAL-RT 0)

  (DO TIMES (X *MAX-NUm-OF-MENU-ITEMS)
    (SETF (AREF *CUMULATIVE-RT-FOR-POS X) 0)
    (SETF (AREF *NUM-TRIALS-FOR-POS X) 0))
)

(DEFUN RESET-EYE-MOVEMENT-VARIABLES ()
  "Called by RUN."

  (DO TIMES (X *ARRAY-SIZE)
    (SETF (AREF *FIRST-SACCADE-DISTANCE X) 0)
    (SETF (AREF *MIDDLE-SACCADE-DISTANCE X) 0)
    (SETF (AREF *FINAL-SACCADE-DISTANCE X) 0))

  (DO TIMES (X *ARRAY-SIZE)
    (DO TIMES (Y *ARRAY-SIZE)
      (SETF (AREF *NUM-FIXATIONS-AS-FUNCTION-OF-TARGET-LOCATION X Y) 0)))

  (SETQ *NUM-MENU-STYLES 0) ;; 1 if one style is run, 2 if both are run.
  (SETQ *FIXATION-POSITIONS NIL)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Device high level control, including some input and output
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN DEV1P (PROCESSOR-START-TIME RESET)
  (PROG (TIME INPUT-EVENT)

    (COND (RESET
      (DEV1P-SIGNAL-SELF PROCESSOR-START-TIME 'START-EXPERIMENT)
      (RETURN NIL)))

    ;; TIME keeps track of the time for output events
    LOOP
      (SETQ TIME PROCESSOR-START-TIME)
      (SETQ INPUT-EVENT (RECEIVE-TTD))
      (COND ((NOT INPUT-EVENT) (RETURN T)))

    ;; DO the appropriate thing
    (CASE (TYPE-OF INPUT-EVENT)
      (MOVEMENT-EVENT (HANDLE-DEVICE-INPUT TIME INPUT-EVENT))
      (SIGNAL-EVENT (HANDLE-SIGNAL-EVENT TIME INPUT-EVENT))
      (T (PRINT (LIST "Unknown input in Dev1:" INPUT-EVENT))
      (RETURN NIL)))

    )
  (GO LOOP)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN HANDLE-SIGNAL-EVENT (TIME INPUT-EVENT)
  "Weird high-level control mechanism that Dave uses and I'm trying to remove.
  Only used for DO-START-EXPERIMENT and DO-START-TRIAL
  Question: Where did these signals come from anyway?? ajh 9/8/95
  Answer: In statements like (DEV1P-SIGNAL-SELF TIME 'START-TRIAL)
  that are at the end of the previous functions."
(DISPLAY-INPUT TIME "Dev1:"
  (FORMAT NIL "Signal ~A" (SIGNAL-EVENT-SIGNAL INPUT-EVENT)))
(CASE (SIGNAL-EVENT-SIGNAL INPUT-EVENT)
  (START-EXPERIMENT (DO-START-EXPERIMENT TIME))
  (START-TRIAL (DO-START-TRIAL TIME))
  ;; (REMOVE-READY (print (list 'DO-REMOVE-READY 'was 'here)))
  ;; (DISPLAY-STIMULUS (print (list 'DISPLAY-MENU 'was 'here)))
  ;; (REMOVE-STIMULUS (print (list 'REMOVE-MENU 'was 'here)))
  (T (PRINT (LIST "Unknown Signal in Dev1:" INPUT-EVENT))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(schedule a message at the specified time to be sent to DEV1P
(DEFUN DEV1P-SIGNAL-SELF (TIME SIGNAL)
  (DISPLAY-OUTPUT TIME "Dev:" SIGNAL)
  (SEND-TTD TIME (MAKE-SIGNAL-EVENT :SIGNAL SIGNAL) DEV1P)
  (SCHEDULE-PROCESSOR TIME DEV1P)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN DO-START-EXPERIMENT (TIME)
  "Start whole experiment"
  ;create and define the cursor object
  (SETQ *CURSOR (MAKE-VISUAL-OBJECT-APPEAR TIME
    *TRACKING-ORIGIN-X *TRACKING-ORIGIN-Y
    (LIST 'SHAPE 'CROSS 'SIZE 0.5)))
  ;; come back after a delay ;; not needed ??? ajh 9/8/95
  (SETQ TIME (+ TIME 100))
  (DEV1P-SIGNAL-SELF TIME 'START-TRIAL)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN DO-START-TRIAL (TIME)
  "Start each trial.
  Stop the experiment when all serial positions are used."
  (COND
    ((NOT *SERIAL-POSITION-LIST)
     (STOP-EPIC))
    (T ;; Else if the experiment is not over...
     (SETQ *GO-BOX-OBJECT
       (MAKE-VISUAL-OBJECT-APPEAR TIME
        *GO-BOX-X    *GO-BOX-Y
        (LIST 'TEXT 'GO))
     (SET-VISUAL-PHYSICAL-PROPERTY *GO-BOX-OBJECT 'IN-MENU 'NO)
     ;; Build menu and determine target item, so precue can be displayed.
     (BUILD-MENU-CONTENTS)
     (MAKE-PRECUE-APPEAR-IF-APPROPRIATE TIME)
     (UPDATE-OVERVIEW) ;update our overview display only when needed
    ;; Finesse!!! Internal cognition triggered directly in the device file!
    ;; Represents user's decision about where to make the first
    ;; saccade when making an efficient sweep down the list. We want to
    ;; determine it afresh for every trial to build in variation.
    ;; (Currently only used for randomly-ordered menus. -ajh 4/7/98)
    (DETERMINE-FIRST-FIXATION-LOCATION *FIRST-FIXATION)
    ;; Finesse! Again, the device driver reaches into the simulated human.
    ;; Rather than creating a new movement location (with variable error)
    ;; to all possible locations (for ordered menus), just create one
    ;; movement to the known target location. Just to save programming cycles.
    (ADD-ERROR-TO-NAMED-LOCATION-OF-TARGET )
  ))
(DEFUN DO-REMOVE-READY (TIME)
  ; Remove GO-BOX and precue
  (MAKE-VISUAL-OBJECT-DISAPPEAR TIME *GO-BOX-OBJECT)
  (MAKE-VISUAL-OBJECT-DISAPPEAR TIME *PRECUE-OBJECT) ;; should be when cursor moves
  (SETQ *GO-BOX-OBJECT NIL)
  (SETQ *PRECUE-OBJECT NIL)
  (UPDATE-OVERVIEW))

(DEFUN HANDLE-DEVICE-INPUT (TIME INPUT-EVENT)
  (LET ()
    (DISPLAY-INPUT TIME "Dev1:" (STRING-OF-MOVEMENT-EVENT INPUT-EVENT))
    (CASE (MOVEMENT-EVENT-DEVICE-INPUT-TYPE INPUT-EVENT)
      (MOUSE
        (DO-MOUSE-MOVEMENT TIME INPUT-EVENT)
      )
      (MOUSE-BUTTON
        (DO-MOUSE-CLICK TIME)
        (if *output-eyes-trace
          (print (list 'mouse 'clicked 'at 'time TIME)))
        )
    ))

(DEFUN DO-MOUSE-CLICK (TIME)
  ;; If the cursor is in the "go" box location and the timing has not yet
  ;; started, then put up the menu and start the timing.
  ;; If the timing is already underway, then stop the timing. Report
  ;; to the subject and to the statistics record whether the response
  ;; was correct or not.  -ajh 9/8/95
  (LET ()
    (COND
      ;; Subject initiates
      ((NOT *TIMING-IN-PROGRESS)
        (SETQ *TIMING-IN-PROGRESS T)
        (DO-REMOVE-READY TIME)
        (DISPLAY-MENU TIME)
        ;; save the current time when the menu appeared
        (SETQ *MENU-APPEARS-TIME TIME)
        ;; trash this: (print (list 'timing 'starts 'at *MENU-APPEARS-TIME))
        ;; the next event should be the response, a movement event
        )
      ;; Subject responds (*TIMING-IN-PROGRESS is true)
      (T
        (VERIFY-CLICK-ON-TARGET)
        (SETQ *TIMING-IN-PROGRESS NIL)
        (DISPLAY-INPUT TIME "Dev:")
        (FORMAT T "~%Response is recorded.")
        (DO-TRIAL-STATS *MENU-APPEARS-TIME TIME)
        )
    ))

;; (SEND-TTD TIME NIL VSENP) ;; removed 7/2/96 -ajh
(SCHEDULE-PROCESSOR TIME VSENP)
(SETQ *TRIAL (1+ *TRIAL))
(REMOVE-MENU (+ TIME 100))
(DO-START-TRIAL (+ TIME 2000))

;; Perhaps we only want to do these when we are done with all trials.
;; Perhaps they need separate function.  ajh 9/8/95
;; (TRIAL-JITTER) ;randomize model parameters if option set
;; (IF *TRIAL-CLEAR-WINDOWS (CLEAREM))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN VERIFY-CLICK-ON-TARGET ()
"Determine whether or not the subject clicks on the target item. -ajh -4/7/98"

(COND

(*OUTPUT-POINT-ERRORS

;; perhaps should be disabled because does not work when there is more
;; than one POINT in the strategy

(SETQ Y-START *GO-BOX-Y)

;; Determine the target for the POINT or SACCADE.
(SETQ Y-TARGET (LOCATION-Y (GET-VISUAL-OBJECT-PHYSICAL-LOCATION *TARGET-OBJECT)))

;; Where the POINT or SACCADE actually landed.
(SETQ Y-LAND (LOCATION-Y (GET-VISUAL-OBJECT-PHYSICAL-LOCATION *CURSOR)))

(SETQ Y-CORRECT-DISTANCE (ABS (- Y-START Y-TARGET)))
(SETQ Y-ACTUAL-DISTANCE (ABS (- Y-START Y-LAND)))

;; Determine vertical error between target and where POINT or SACCADE landed.
(SETQ Y-ERROR (- Y-ACTUAL-DISTANCE Y-CORRECT-DISTANCE))

(DOLIST (OUTPUT-STREAM '(*STANDARD-OUTPUT* *RESULTS-FILE))
;; Output Target digit, target position, start position, and error.
(PRIN1C (FORMAT NIL "~A~A~,6F~A~,6F~A~,6F" *TARGET-TEXT #	ab Y-TARGET #	ab *GO-BOX-Y #	ab Y-ERROR) (EVAL OUTPUT-STREAM))
;;(print OUTPUT-STREAM)
)

;; ...else if you are not displaying POINT errors...
(T

;; If cursor is not on target item, break with an error.
(IF (NOT (CURSOR-ON-OBJECT-P *CURSOR *TARGET-OBJECT *MENU-ITEM-ANGLE))
(IF *STOP-IF-ERROR
 (BREAK (FORMAT NIL "SUBJECT MISSED the target ~A with the cursor ~A"*TARGET-OBJECT *CURSOR)))))
)

;; Perhaps discard......
;; If cursor is not on target item, issue an error and continue.
;;(IF (NOT (CURSOR-ON-OBJECT-P *CURSOR *TARGET-OBJECT *MENU-ITEM-ANGLE))
;; (PRINTC (FORMAT NIL
;; "SUBJECT MISSED the target ~A with the cursor ~A"
;; *TARGET-OBJECT *CURSOR) *STANDARD-OUTPUT*))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN DO-MOUSE-MOVEMENT (TIME INPUT-EVENT)
;; Move the crosshairs as specified by the input event of the mouse movement.
This function gets executed once for each mouse/cursor movement.

TIME is the time that the mouse movement is completed, and stays the same
throughout the execution of this function.

```lisp
(let (change-x change-y new-x new-y output points-to-tmp)

;; Get the movement and convert it to change in X and Y.
;; (I believe the X and Y values themselves are also available.  -ajh 5/19/98)
(multiple-value-setq (change-x change-y)
  (polar-to-cartesian
   (polar-vector-r (movement-event-vector input-event))
   (polar-vector-theta (movement-event-vector input-event))))

(setq new-x
  (+ change-x (location-x (get-visual-object-physical-location *cursor))))
(setq new-y
  (+ change-y (location-y (get-visual-object-physical-location *cursor))))

(set-visual-object-physical-location *cursor (make-location :x new-x :y new-y))

;; Set CURSOR POINTS-TO to the menu item pointed to, or to nothing if it points
;; to no item.  Set it to NIL to remove it if the trial is not in progress.
;; -ajh 5/19/98
(setq points-to-tmp 'nothing)
(cond
  (*timing-in-progress
    (dolist (obj *menu-visoj-list)
      (cond ((cursor-on-object-p *cursor obj *menu-item-angle)
        (setq points-to-tmp obj)))))
  (t
    (setq points-to-tmp nil)))

;; Queue the event to update the POINTS-TO feature  -ajh 5/22/98
(set-visual-physical-property *cursor 'points-to points-to-tmp)
(display-output time "Dev1" (format nil "~a Points to ~a" *cursor points-to-tmp))
(send-tdt time output vsenp)
(schedule-processor time vsenp)
(update-overview)

;; Queue the event to update the location.
;; Copied from Wickens.  Do something similar.
;; note - any queued events for this signal will be pointing
;; to the CURRENT version of *CURSOR's properties
(setq output (make-visual-event :object *cursor :change 'location))
(display-output time "Dev1" (format nil "~a Moving to ~a" *cursor output))
(send-tdt time output vsenp)
(schedule-processor time vsenp)
(update-overview)

(make-precue-appear-if-appropriate (+ time 100))  ;; delay 11/7/97 ajh
))

;;;;;;
(defun make-precue-appear-if-appropriate (time)
  "If during precue stage, the go box has appeared, the mouse is in go box,
  and the precue has not yet appeared, then make precue (to-be-selected item)
  appear in the visual space.  ajh."

(cond
  ((and
    (not *timing-in-progress)
    *go-box-object
    (cursor-on-object-p *cursor *go-box-object *menu-item-angle)
    (not *precue-object))
   (setq *precue-object
     (make-visual-object-appear
      (get-visual-object-physical-location *cursor))))
))
(COND
  ((AND
    ;; If cursor is within half a height of the center of the object...
    (< (ABS (-
        (LOCATION-X (GET-VISUAL-OBJECT-PHYSICAL-LOCATION CURSOR))
        (LOCATION-X (GET-VISUAL-OBJECT-PHYSICAL-LOCATION TARGET-OBJ)))))
      (/ ITEM-HEIGHT 2))
    (< (ABS (-
        (LOCATION-Y (GET-VISUAL-OBJECT-PHYSICAL-LOCATION CURSOR))
        (LOCATION-Y (GET-VISUAL-OBJECT-PHYSICAL-LOCATION TARGET-OBJ)))))
      (/ ITEM-HEIGHT 2))) ;; Use height for width.
    T) ;; then return T
  (T NIL)) ;; else return NIL
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Output from device, lower level functions

(DEFUN BUILD-MENU-CONTENTS ()
  "Build the text contents of the menu. Store the text in *MENU-TEXT-LIST, which will later be transformed into *MENU-VISOBJ-LIST."

  ; ; Create the list of menu items.
  (SETQ *MENU-TEXT-LIST (DIGIT-LIST *MENU-ORDERED *MENU-NUMBER-ITEMS))

  ; ; Get the next desired serial position from the list, set it as the target.
  (SETQ *SERIAL-POSITION (CAR *SERIAL-POSITION-LIST))
  (SETQ *SERIAL-POSITION-LIST (CDR *SERIAL-POSITION-LIST))
  (SETQ *TARGET-TEXT (NTH (- *SERIAL-POSITION 1) *MENU-TEXT-LIST)))

(DEFUN DISPLAY-MENU (TIME)
  ;; Build and display the EPIC physical objects for the textual menu list.
  ;; Determine the correct serial position.
  (SETQ *SERIAL-POSITION (+ 1 (position *TARGET-TEXT *MENU-TEXT-LIST)))

  ;; Reset the menu item list here for lack of an obvious place higher up.
  (SETQ *MENU-VISOBJ-LIST NIL)
  (SETQ *MENU-ITEM-COUNTER 0)

  (BUILD-MENU-OBJECTS *MENU-TEXT-LIST 'vert *MENU-ITEM-ANGLE
   (LIST *MENU-START-X *MENU-START-Y) TIME)
  (UPDATE-OVERVIEW))

(DEFUN BUILD-MENU-OBJECTS (ITEM-LIST HORIZ-VERT ANGLE-SEP START-POINT TIME)
  "Given a list of text objects, a horizontal versus vertical orientation, an angle of separation between each consecutive pair of text objects, and a starting point in the visual space, create a physical menu list of these objects in the visual space and link them with the appropriate features. Note: The initial version of the function will not wrap lists."

  (LET (NEW-ITEM ITEM-NUMBER X-POS Y-POS)
    (SETQ ITEM-NUMBER 0)
    (SETQ X-POS (CAR START-POINT))
    (SETQ Y-POS (CADR START-POINT))

    (IF (< (LENGTH ITEM-LIST) 1)
      (IF *STOP-IF-ERROR
        (BREAK "BUILD-MENU-OBJECTS requires at least one item.")))

    ;; Create each menu item and its required properties.
    (DOLIST (ITEM ITEM-LIST)
      (SETQ NEW-ITEM
        (MAKE-VISUAL-OBJECT-APPEAR ; ; Create the object.
         TIME X-POS Y-POS
         (LIST 'IN-MENU 'YES 'COLOR 'BLK 'TEXT ITEM 'SIZE ANGLE-SEP ; ; in this case menu height is Fitts width )))

      ;; If this is the target item...
      (COND ((EQUAL ITEM *TARGET-TEXT)
         ; ; Make a special pointer to the object. -ajh 10/27/96
         (SETQ *TARGET-OBJECT NEW-ITEM))
           ))

    ;; Pass through the physical boundaries (vertical only) of the object.
    (SET-VISUAL-PHYSICAL-PROPERTY NEW-ITEM 'TOP-BOUNDARY (+ Y-POS (/ ANGLE-SEP 2)))
    (SET-VISUAL-PHYSICAL-PROPERTY NEW-ITEM 'BOTTOM-BOUNDARY (- Y-POS (/ ANGLE-SEP 2))))
;; Increment loop variables.
(IF (EQUAL HORIZ-VERT 'VERT) (setq y-pos (- y-pos angle-sep)))
(IF (EQUAL HORIZ-VERT 'HORIZ) (setq x-pos (+ x-pos angle-sep)))
(INCF ITEM-NUMBER)
(SETQ *MENU-VISOBJ-LIST (CONS NEW-ITEM *MENU-VISOBJ-LIST))
(INCF *MENU-ITEM-COUNTER)

;;; Assign the properties that are specific to vertical or horiz. menu.
(COND

;;; Vertical menu.
((EQUAL HORIZ-VERT 'VERT)
 ;; Step through the objects and assign IS-ABOVE's and IS-BELOW's.
 (DOITIMES (ITEM-NUM (- *MENU-ITEM-COUNTER 1))

   (SET-VISUAL-PHYSICAL-PROPERTY (NTH ITEM-NUM *MENU-VISOBJ-LIST)
     'IS-BELOW (NTH (+ 1 ITEM-NUM) *MENU-VISOBJ-LIST))
   (SET-VISUAL-PHYSICAL-PROPERTY (NTH ITEM-NUM *MENU-VISOBJ-LIST)
     'IS-ABOVE (NTH (+ 1 ITEM-NUM) *MENU-VISOBJ-LIST)))

   ;; End of DOITIMES assigning above and below.

   (SET-VISUAL-PHYSICAL-PROPERTY (CAR *MENU-VISOBJ-LIST)
     'IS-ABOVE 'NOTHING)
   (SET-VISUAL-PHYSICAL-PROPERTY (CAR (LAST *MENU-VISOBJ-LIST))
     'IS-BELOW 'NOTHING)

   (SET-VISUAL-PHYSICAL-PROPERTY (CAR *MENU-VISOBJ-LIST)
     'POSITION-IN-MENU 'LAST)
   (SET-VISUAL-PHYSICAL-PROPERTY (CAR (LAST *MENU-VISOBJ-LIST))
     'POSITION-IN-MENU 'FIRST))

   (IF *STOP-IF-ERROR (BREAK "HORIZ menu not implemented yet.")))

   (T (IF *STOP-IF-ERROR (BREAK "BUILD-MENU-OBJECTS requires HORIZ or VERT."))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN DIGIT-LIST (ORDERED-P NUM-DIGITS)
 "Make a list of the digits from 1 to NUM-DIGITS. If ORDERED-P is NIL, randomize the list. Return the list."
 (LET (INIT-LIST THIS-ITEM NEW-LIST)

   ;; Build an ordered list of digits from 1 to NUM-DIGITS.
   (DOITIMES (X NUM-DIGITS)
     (SETQ INIT-LIST (APPEND INIT-LIST (LIST (+ 1 X)))))

   (COND
     ((NOT ORDERED-P) ;; If random
     (DOITIMES (X (LENGTH INIT-LIST))
       ;; Pick a random number from 1 to the number of items left in the set.
       (SETQ THIS-ITEM (NTH (RANDOM (LENGTH INIT-LIST)) INIT-LIST))
       (SETQ NEW-LIST (CONS THIS-ITEM NEW-LIST))
       ;; Pull out that item. (Note: DELETE is destructive; REMOVE is not.)
       (SETQ INIT-LIST (REMOVE THIS-ITEM INIT-LIST)))

     (NEW-LIST)
     ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(DEFUN MAKE-SERIAL-POSITION-LIST (NUM-ITEMS TRIALS-PER-POS)
 "Generate a random list of numbers ranging from 1 to NUM-ITEMS, such that each number occurs exactly TRIALS-PER-POS times. Create an ordered list,
then randomly select and delete one item at a time, putting each into a new list pointed to by *SERIAL-POSITION-LIST.

Dave's idea: Create an ordered list within an array. Swapping each array item with another randomly chosen array item.

(LET (ORDERED-LIST NEXT-ITEM)
  (SETQ *SERIAL-POSITION-LIST NIL)
  ;; Start the list (so we can use NCONC)
  (SETQ ORDERED-LIST (MAKE-LIST TRIALS-PER-POS :INITIAL-ELEMENT '1))
  ;; Create an ordered list.
  (DOTIMES (X (- NUM-ITEMS 1))
    (NCONC ORDERED-LIST (MAKE-LIST TRIALS-PER-POS :INITIAL-ELEMENT (+ 2 X)))
  )
  ;; Until the ordered list is empty...
  (DOTIMES (X (LENGTH ORDERED-LIST))
    ;; Select a random item from the ordered list.
    (SETQ NEXT-ITEM (NTH (RANDOM (LENGTH ORDERED-LIST)) ORDERED-LIST))
    ;; Put it in the random list
    (PUSH NEXT-ITEM *SERIAL-POSITION-LIST)
    ;; Remove it (or one just like it) from the ordered list.
    ;; (DELETE didn't work here.)
    (SETQ ORDERED-LIST (REMOVE NEXT-ITEM ORDERED-LIST :COUNT 1))
  )
)

;;;;; Keep track of experimental results.

;;;;; Do a trial stats

(DEFUN DO-TRIAL-STATS (MENU-APPEARED-TIME RESPONSE-TIME)
  (LET (RESULT)
    (SETQ *CURRENT-RT (- RESPONSE-TIME MENU-APPEARED-TIME))
    ;; update grand averages data for display purposes
    (SETQ *GRAND-TOT-N (+ *GRAND-TOT-N))
    (SETQ *GRAND-TOTAL-RT (+ *GRAND-TOTAL-RT *CURRENT-RT))
    (SETQ *GRAND-AVG-RT (/ *GRAND-TOTAL-RT *GRAND-TOT-N))
    ;; update totals based on serial position
    (INCF (AREF *CUMULATIVE-RT-FOR-POS (- *SERIAL-POSITION 1)) *CURRENT-RT)
    (INCF (AREF *NUM-TRIALS-FOR-POS (- *SERIAL-POSITION 1)))
    (DO-EYE-MOVEMENT-TRIAL-STATS)
    (COND (*OUTPUT-RT-DATA-EVERY-TRIAL
      (COND (*OUTPUT-RT-DATA-EVERY-TRIAL
        (DISPLAY-BAD-INPUT (TIME RESPONSE))
        (DISPLAY-INPUT TIME "Dev:" (FORMAT NIL "Response ~A received when not expected" RESPONSE))
      )))
  )
)

;;;;; Remove menu

(DEFUN REMOVE-MENU (TIME)
  (LET ()
    ;; clear the menu from the visual space
    (DO-LIST (ITEM *MENU-VISOBJ-LIST)
      (MAKE-VISUAL-OBJECT-DISAPPEAR TIME ITEM))
    (UPDATE-OVERVIEW)
  ))
(DEFUN DO-EYE-MOVEMENT-TRIAL-STATS ()
  "Collect the eye movement data necessary to compare against Byrne's IJHCS article
  and Nilsen's CHI 98 PowerPoint slides.
  Note that *FIXATION-POSITIONS are actually stored in reverse-order.
  In retrospect, I should have just used a REVERSE and put them in the right order.
  ajh-6/10/01"

  ;; Collect the number of fixations as a function of the target position.
  (INCF (AREF *NUM-FIXATIONS-AS-FUNCTION-OF-TARGET-LOCATION
               *SERIAL-POSITION *MENU-NUMBER-ITEMS) (LENGTH *FIXATION-POSITIONS))

  ;; Collect the first fixation as a function of the target position.
  ;; Add up the distance to the first fixation for each target position and menu length
  (INCF (AREF *FIRST-FIXATION-AS-FUNCTION-OF-TARGET-LOCATION
               *SERIAL-POSITION *MENU-NUMBER-ITEMS) (CAR (LAST *FIXATION-POSITIONS)))

  ;; Collect the first, middle, and last saccade frequencies. Note that if there
  ;; is only one saccade, it will get counted as both the first AND the last saccade.
  ;; Also note the offset by 11 in the middle and final saccade arrays.

  (IF (> *MENU-NUMBER-ITEMS 3)

    ;; If there are more than 3 items in the menu....
    ;; then pop and classify each fixation.
    (COND
     ((EQ 0 (LENGTH *FIXATION-POSITIONS))
      (BREAK (FORMAT NIL "No *FIXATION-POSITIONS recorded.")))
     ((EQ 1 (LENGTH *FIXATION-POSITIONS))
      (INCF (AREF *FINAL-SACCADE-DISTANCE (+ 11 (CAR *FIXATION-POSITIONS))))
      (INCF (AREF *FIRST-SACCADE-DISTANCE (+ 11 (POP *FIXATION-POSITIONS))))
     ((EQ 2 (LENGTH *FIXATION-POSITIONS))
      (INCF (AREF *FIRST-SACCADE-DISTANCE (+ 11 (POP-AND-DETERMINE-DISTANCE))))
      (INCF (AREF *FINAL-SACCADE-DISTANCE (+ 11 (POP-AND-DETERMINE-DISTANCE))))
      (T ;; There are 3 or more fixation positions.
       (INCF (AREF *FIRST-SACCADE-DISTANCE (+ 11 (POP *FIXATION-POSITIONS))))
       (INCF (AREF *FINAL-SACCADE-DISTANCE (+ 11 (POP *FIXATION-POSITIONS))))
       (LOOP WHILE (> (LENGTH *FIXATION-POSITIONS) 1) DO
        (INCF (AREF *MIDDLE-SACCADE-DISTANCE (+ 11 (POP-AND-DETERMINE-DISTANCE))))
        (INCF (AREF *FIRST-SACCADE-DISTANCE (+ 11 (POP *FIXATION-POSITIONS))))))
    )

    ;; Else there are 3 or fewer items in the menu....
    (SETQ *FIXATION-POSITIONS NIL)

    (BREAK (FORMAT NIL "*FIXATION-POSITIONS -A was not fully emptied."
                 *(FIXATION-POSITIONS)))
  )

 造FUNCTION POP-AND-DETERMINE-DISTANCE ()
  "Given a *FIXATION-POSITIONS list with 2 or more elements, pop the first element
  on the list (which is actually in reverse order) and return the distance between
  the first two elements on the list. (5 3) => 2 (3 5) => -2"
(LET (NEXT-FIXATION)
  (SETQ NEXT-FIXATION (POP *FIXATION-POSITIONS))
  (- NEXT-FIXATION (CAR *FIXATION-POSITIONS)))

(defun output-eye-movement-data (&optional (output-stream nil))
  "Cumulative results across all menu styles. Note that the distance frequencies are only computed for menu lengths greater than 3, assuming that is what Byrne used."
  (let (result fsd-total msd-total flsd-total)
    ;; Calculate the total number of saccades, so frequencies can be computed.
    (setq fsd-total (sum-array *first-saccade-distance))
    (setq msd-total (sum-array *middle-saccade-distance))
    (setq flsd-total (sum-array *final-saccade-distance))
    ;; across all subjects
    (terpri output-stream)
    (printf (format nil ""))
    ;; Loop through all the frequency distributions.
    (dotimes (x (- *array-size 1))
      ;; (if (not (eql 11 x)) ;; Don't print the results for a distance of 0
        (progn
          (setq result
            (format nil ""))
        ;; Fix3 Fix6 Fix9 Fix12 | Ffix3 Ffix6 Ffix9 Ffix12 | Dist. First Middle
        ;; Avg number of fixations in a trial as a func of the target frequencies position, for each menu size
        (cond ((< x 11) (- x 11))
          (t (- x 10)))))
    ;; First-fixation as a function of the target location
    (calc-frequency-result *first-saccade-distance x fsd-total)
    ;; #tab
    ;; Num-fixations as a function of the target location
    (calc-frequency-result *num-fixations-as-function-of-target-location x 3)
    ;; #tab
    (calc-frequency-result *num-fixations-as-function-of-target-location x 6)
    ;; #tab
    (calc-frequency-result *num-fixations-as-function-of-target-location x 9)
    ;; #tab
    (calc-frequency-result *num-fixations-as-function-of-target-location x 12)
    ;; #tab
    ;; First-fixation as a function of the target location
    (calc-frequency-result *first-fixation-as-function-of-target-location x 3)
    ;; #tab
    (calc-frequency-result *first-fixation-as-function-of-target-location x 6)
    ;; #tab
    (calc-frequency-result *first-fixation-as-function-of-target-location x 9)
    ;; #tab
    (calc-frequency-result *first-fixation-as-function-of-target-location x 12)
    ;; #tab
    ;; Distance, first, middle
    (terpri output-stream)
(CALC-FREQUENCY-RESULT *MIDDLE-SACCADE-DISTANCE X MSD-TOTAL)
#\tab
(CALC-FREQUENCY-RESULT *FINAL-SACCADE-DISTANCE X FLSD-TOTAL))

(PRINTC RESULT OUTPUT-STREAM))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN SUM-ARRAY (THE-ARRAY)
  "Return the sum of all the elements in the array."
  (LET ((RETURN-VALUE 0))
    (DOTIMES (X *ARRAY-SIZE)
      (INCF RETURN-VALUE (AREF THE-ARRAY X)))
    RETURN-VALUE)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN CALC-FIXATION-RESULT (THE-ARRAY POS MENU-LENGTH)
  "Given the array holding the number of fixations to each location across all trials, a position number, and the menu length; return the average number of fixations to that location in each trial, or a period if there is no result for that position. (Returning a space confused CricketGraph.)"
  (IF (< POS MENU-LENGTH)
    (float (/ (AREF THE-ARRAY (+ 1 POS) MENU-LENGTH)
      (* *TOTAL-TRIALS-PER-POSITION *NUM-MENU-STYLES)))
    #\.)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN CALC-FREQUENCY-RESULT (THE-ARRAY POS TOTAL)
  "Given the array holding the total number of occurances of a saccade distance, indexed by distance; an index into the array; and the total number of distances recorded; return the frequency with which a saccade of that distance occurred ajh-6/14/01"
  (COND ((EQL TOTAL 0) ;; Just return 0 if TOTAL is zero
    ((< POS 11) (FLOAT (/ (AREF THE-ARRAY POS) TOTAL)))
    (T (FLOAT (/ (AREF THE-ARRAY (+ 1 POS)) TOTAL))))
  ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(DEFUN DO-FINAL-STATS (&OPTIONAL (OUTPUT-STREAM NIL))
  (LET (RESULT)
    ;eventually use the OUTPUT-STREAM in the PRINTC's
    ;output cumulative results
    (TERPRI OUTPUT-STREAM)
    (PRINTC (FORMAT NIL "-----------------------------------------------------") OUTPUT-STREAM)
    (PRINTC (FORMAT NIL "Serial Position Number Trials Avg Select Time") OUTPUT-STREAM)
    (PRINTC (FORMAT NIL "-----------------------------------------------------") OUTPUT-STREAM)
    ; Print cumulative results.
(DO TIMES (X *MENU-NUMBER-ITEMS)
  (SETQ RESULT (FORMAT NIL "~8D~A~11D~A~15D"
    (+ 1 X) #\tab (AREF *NUM-TRIALS-FOR-POS X) #\tab
    (ROUND (/ (AREF *CUMULATIVE-RT-FOR-POS X)
      (AREF *NUM-TRIALS-FOR-POS X))))
  (PRINTC RESULT OUTPUT-STREAM))

(SETQ RESULT (FORMAT NIL "--------------------------------")
  OUTPUT-STREAM)

(SETQ RESULT (FORMAT NIL "  Averages: ~5D~A~15D"
  (ROUND (*GRAND-TOT-N *MENU-NUMBER-ITEMS)) #\tab
  (ROUND *GRAND-AVG-RT)))

(DEFUN UPDATE-OVERVIEW ()
  "shadows the one in overview-display.lisp
  clear and rewrite the overview window to show the current external
  situation.  Shadow this with an application specific function as needed
  NOTE CALL TO GRAPHIC OVERVIEW UPDATE FUNCTION"

  (UPDATE-GRAPHIC-OVERVIEW)

  (IF (NOT *OVERVIEW-WINDOW)
      (RETURN-FROM UPDATE-OVERVIEW NIL))
  (WINDOW-CLEAR *OVERVIEW-WINDOW)
  (WINDOW-PRINTC *MASTER-CLOCK *OVERVIEW-WINDOW)
  (WINDOW-PRINTC (LIST "Eye at: " (STRING-OF-LOCATION (GET-EYE-LOCATION)))
    *OVERVIEW-WINDOW)
  (DOLIST (OBJECT *VISUAL-PHYSICAL-SPACE)
    (WINDOW-PRINTC (LIST (GET-VISUAL-PHYSICAL-PROPERTY OBJECT 'SHAPE) (GET-VISUAL-PHYSICAL-PROPERTY
  _ OBJECT 'TEXT)
    "at"
    (STRING-OF-LOCATION (GET-VISUAL-OBJECT-PHYSICAL-LOCATION OBJECT)))
    *OVERVIEW-WINDOW)
  )
  (WINDOW-PRINTC (FORMAT NIL "Trial ~D: RT: ~6,0F" *TRIAL *CURRENT-RT) *OVERVIEW-WINDOW)
  ;force last line to show up
  (WINDOW-PRINTC "---" *OVERVIEW-WINDOW)
  (WINDOW-TERPRI *OVERVIEW-WINDOW)
)

(DEFUN UPDATE-GRAPHIC-OVERVIEW ()
  "shadows the one in overview-display.lisp
  clear and rewrite the overview window to show the current external
  situation.  Shadow this with an application specific function as needed"

  (LET (TEXT SHAPE X Y SIZE COLOR MANUAL-STYLE MANUAL-HAND MANUAL-FINGER
          MANUAL-FEATURES MANUAL-ACTION OCULOMOTOR-ACTION)
    (IF (NOT *GRAPHIC-OVERVIEW-WINDOW)
        (RETURN-FROM UPDATE-GRAPHIC-OVERVIEW NIL))
    (ERASE-GRAPHIC-WINDOW *GRAPHIC-OVERVIEW-WINDOW)
    (SETQ MANUAL-ACTION (GET-PROCESSOR STATE MOVEMENT-IN-PROGRESS MMOTP))
    (COND (MANUAL-ACTION
            (SETQ MANUAL-FEATURES (GET-PROCESSOR STATE EXECUTION-MOVEMENT-FEATURES MMOTP))
            (SETQ MANUAL-STYLE (GETF MANUAL-FEATURES 'STYLE))
            (SETQ MANUAL-HAND (GETF MANUAL-FEATURES 'HAND)))
    )
(SETQ MANUAL-FINGER (GETF MANUAL-FEATURES 'FINGER)))
(SETQ OCULOMOTOR-ACTION (GET-PROCESSOR STATE MOVEMENT-IN-PROGRESS OMOTP))
(DRAW-VISUAL-OBJECT 'ABSOLUTE-TEXT
  0 10
  (FORMAT NIL "Time:-D" *MASTER-CLOCK ))
(DRAW-VISUAL-OBJECT 'ABSOLUTE-TEXT
  0 20
  (FORMAT NIL "Manual:-A ~A -A, Ocular:-A"
           MANUAL-STYLE MANUAL-HAND MANUAL-FINGER OCULOMOTOR-ACTION))
(DRAW-VISUAL-OBJECT 'ABSOLUTE-TEXT
  0 30
  (FORMAT NIL "Target:-A" *TARGET-TEXT))
(DRAW-VISUAL-OBJECT 'ABSOLUTE-TEXT
  0 (- *GRAPHIC-OVERVIEW-WINDOW-VSIZE 10)
  (FORMAT NIL "Trial:-D RT:-5,0F Avg. RT:-5,0F" *TRIAL *CURRENT-RT
           *GRAND-AVG-RT))

;draw the visual state
(DRAW-EYE)

;draw each object in physical visual space
(DOLIST (OBJECT *VISUAL-PHYSICAL-SPACE)
  (SETQ TEXT (GET-VISUAL-PHYSICAL-PROPERTY OBJECT 'TEXT))
  (SETQ SHAPE (GET-VISUAL-PHYSICAL-PROPERTY OBJECT 'SHAPE))
  (SETQ SIZE (GET-VISUAL-PHYSICAL-PROPERTY OBJECT 'SIZE))
  (SETQ COLOR (GET-VISUAL-PHYSICAL-PROPERTY OBJECT 'COLOR))
  (SETQ X (LOCATION-X (GET-VISUAL-OBJECT-PHYSICAL-LOCATION OBJECT)))
  (SETQ Y (LOCATION-Y (GET-VISUAL-OBJECT-PHYSICAL-LOCATION OBJECT)))
  (CASE SHAPE
    (SQUARE (DRAW-VISUAL-OBJECT 'OPEN-SQUARE X Y SIZE))
    (CIRCLE (DRAW-VISUAL-OBJECT 'OPEN-CIRCLE X Y SIZE))
    (CROSS (DRAW-VISUAL-OBJECT 'CROSSHAIRS X Y SIZE))
    (OTHERWISE
     (COND
      ((AND SHAPE COLOR)
       (DRAW-VISUAL-OBJECT 'TEXT X Y SHAPE)
       (DRAW-VISUAL-OBJECT 'TEXT (- X 1.5) (+ Y 1.) COLOR)
       (T (DRAW-VISUAL-OBJECT 'TEXT X Y TEXT))))
     )
    ))