(defmacro defclassic (class supers &rest slots)
  `(defclass ,class ,supers
    ,((mapcar
       ,(lambda (s) ; s = slot specification
         (flet ((build (sn) ; sn = slot name
                     (list sn ':accessor sn ':initarg
                         (read-from-string
                          (concatenate 'string "::" (symbol-name sn))))))
           (cond ((atom s) (append (build s) '(:initform nil)))
                 ((null (cddr s)) (append (build (first s)) (list ':initform (second s))))
                 ((eql t (second s))
                  (append (build (first s))
                          (nthcdr 2 s)
                          (if (not (member :initform (nthcdr 2 s))) '(:initform nil)))
                 (t s)))
          slots))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Classes;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun model-profile ()
  tracking
    class-status track-number range-strategy class-fb
  fb-tn range-help altitude-help appdist-help wm-capacity att-span
  r->l b->t screen-area initial-phase reclassify)

(defun model-stats ()
  curtime
    all
  visual-location
  screen-targ
  text
  targ-attributes
    real-number
    push-enter)

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

(defun des-target-feature (icon-feature))
(defun info-text-feature (text-feature))
(defun label-text-feature (text-feature))
(defun feedback-feature (icon-feature))
(defun help-text-feature (text-feature) low high)
(defun help-cv-feature (text-feature))
(defun help-wg-feature (text-feature))
(defun radio-button-feature (icon-feature) pushed?)
(defclassic arc-feature (icon-feature))
(defclassic gray-feature (rect-feature))
(defclassic feedback-icon (icon-feature))
(defclassic track-pd-feature (icon-feature))
(defclassic plane-feature (icon-feature))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;Variables;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defvar *initial-targets-status* nil) ; t when all initial targets have been classified
(defvar *model-stats* nil)
(defvar *arc-intervals* '[(185 199) (135 165) (85 115) (35 65)])
(defvar *arc-intervals* '[(190 199) (140 160) (90 110) (40 60)])
(defvar decision-trans '(mi::ignore mi::review mi::monitor mi::warn mi::ready mi::lock-on
mi::kill))
(defvar colors (pairlis (list *yellow-color* *blue-color* *red-color* *dark-green-color*
*light-blue-color* *light-gray-color* *orange-color* *white-color* *black-color*)
'(yellow blue red green blue gray orange white black)))
(defvar symbol-trans (pairlis '(mi::info-track-number mi::info-speed mi::info-bearing
mi::info-range
mi::info-course mi::info-altitude mi::info-bearing/course )
'(targ-tn targ-speed targ-bearing targ-range targ-course
targ-altitude targ-appdist))))
(defvar *model-profile* (make-instance 'model-profile))
(defvar *scan-index* 0)
(defvar *scan-pattern* '[(2 (512 600)) (1 (512 450)) (3 (512 625)) (0 (444 200))])
(defvar *track-duration* 5)
(defvar *performance-difference* 0)
(defvar *track-interval* *track-duration* )

(defparameter wgt-trans '(zero one two))
(defparameter *miles/segment* 50)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;Lisp functions called in model;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun new-y-coordinate (seg)
(let ((n (no-output (chunk-slot-value-fct int 'value))))
(nth seg '(200 450 600 625)))))
(defun compare-ts-p (txt int)
(let ((mn (read-from-string txt))
(mx (read-from-string (subseq txt (+ 2 (position #\- txt)))))
(n (no-output (chunk-slot-value-fct int 'value))))
(cond ((< n mn) '<)
((< n mn) '>)
(t '='))))
(defun get-x-db-min () (point-h (view-position (mi::info-window))))
(defun get-y-db-min () (point-v (view-position (mi::info-window))))
(defun get-x-db-max () (+ (point-h (view-position (mi::info-window))) (point-h (view-size
(mi::info-window))))))
(defun get-y-db-max () (+ (point-v (view-position (mi::info-window))) (point-v (view-size
(mi::info-window))))))
(defun in-rng-p (attr l v h) (if (and (eql attr 'appdist) (eql v 9999) (eql h 300)) (setq v h)) (<= l v h))

(defun tn->wme (txt)
  "First (no-output (sdm-fct `(isa integer value ,(read-from-string txt))))"

(defun get-loc-vector (loc)
  (vector (no-output (chunk-slot-value-fct loc 'screen-x)) (no-output (chunk-slot-value-fct loc 'screen-y))))

(defun cursor-over-target-p (tn-wme)
  "Let* ((tn (no-output (chunk-slot-value-fct tn-wme 'value)))
    (pos (subtract-points (view-mouse-position nil) #@'(0 20)))
    (feat (find-if #'(lambda(x) (and (typep x 'des-target-feature)
      (eql tn (mi::track-num (mi::targ-obj (screen-obj x)))))))
      (visicon (vis-m *mp*)))
    (obj (if feat (mi::targ-obj (screen-obj feat)))
      (when (and pos obj)
        (mi::point-in-object-p pos obj nil))))"

(defun shift-key-down ()
  "(mi::set-shift-key t)"
)

(defun shift-key-up ()
  "(mi::set-shift-key nil)"
)

(defun start-tracking-p ()
  "(mi::tracking-state)"
)

(defun get-help-strategy (attr)
  "Case attr
    (range (if (eql (range-help *model-profile*) 2) (get-wme 'in-goal) (get-wme 'encode)))
    (altitude (if (eql (altitude-help *model-profile*) 2) (get-wme 'in-goal) (get-wme 'encode)))
    (appdist (if (eql (appdist-help *model-profile*) 2) (get-wme 'in-goal) (get-wme 'encode))))"

(defun compare-performance (cls track)
  "(> (+ cls *performance-difference*) track)"

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;RPM related functions;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Screen objects -> Features

(defvar build-features-for ((self mi::arc-feature-item) (vis-mod vision-module))
  "Builds an icon feature for an arc"
  (let ((rb (make-instance 'arc-feature
    :x (px (view-loc self))
    :y (py (view-loc self))
    :z (pz (view-loc self)))
    (in-rng-p 'appdist)
    (tn->wme txt)
    (get-loc-vector loc)
    (cursor-over-target-p (tn-wme))
    (shift-key-down)
    (shift-key-up)
    (start-tracking-p)
    (get-help-strategy attr)
    (compare-performance cls track)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defmethod build-features-for ((self mi::track-pd) (vis-mod vision-module))
  "Builds an icon feature for an arc"
  (let ((rb (make-instance 'track-pd-feature
                      :x (px (view-loc self))
                      :y (py (view-loc self))
                      :screen-obj self
                      :width (point-h (view-size self))
                      :height (point-v (view-size self))
                      :kind 'track-pd
                      :color (cdr (assoc (mi::color self) colors)))))
    rb))

(defmethod build-features-for ((self mi::tracking-target) (vis-mod vision-module))
  "Builds an icon feature for tracking plane"
  (let ((targ (first (member 'plane-feature (visicon vis-mod) :key #'type-of))))
    (if targ
      targ
      (make-instance 'plane-feature
                      :x (px (view-loc self))
                      :y (py (view-loc self))
                      :screen-obj self
                      :width (point-h (view-size self))
                      :height (point-v (view-size self))
                      :kind 'plane))))

(defmethod build-features-for ((self radio-button-dialog-item) (vis-mod vision-module))
  "Builds an icon feature for a RADIO-BUTTON-DIALOG-ITEM"
  (let ((rb (make-instance 'radio-button-feature
                          :x (px (view-loc self))
                          :y (py (view-loc self))
                          :screen-obj self
                          :width (point-h (view-size self))
                          :height (point-v (view-size self))
                          :kind 'radio-button
                          :color (if (radio-button-pushed-p self) 'black 'white)
                          :pushed? (radio-button-pushed-p self))))
    (let* ((action->scale (position (view-nick-name self) decision-trans)
            (score-range-max (second (nth action->scale (mi::mapping (mi::get-game))))))
          (score-range-min (if (plusp action->scale) (+ 1 (second (nth (~action->scale l) (mi::mapping (mi::get-game))))))
                          0))
      (setf (val rb) (format nil "~-S -S" score-range-min score-range-max ))
      rb)))
(defun get-value-feat (feat rnge)
  (case feat
    (sector ;;which sector
      (max 0 (- 3 (floor rnge *miles/segment*))))
    (arc ;;nearest border
      (if rnge (position-if #' (lambda(x) (< (first x) rnge (second x))) *arc-intervals*)))))

(defun str->sym (str)
  (intern (string-upcase str) "CL-USER"))
(defmethod build-features-for ((self mi::fb-static-text-dialog-item) (vis-mod vision-module))
"Builds an icon feature for a feedback field"
(make-instance 'feedback-feature
:x (px (view-loc self))
:y (py (view-loc self))
:screen-obj self
:value (string-upcase (dialog-item-text self))
:color (cdr (assoc (part-color self :body) colors))
:width (point-h (view-size self))
:height (point-v (view-size self))
:kind 'fb-txt)
)

(defun write-name-to-string (n)
  (let* ((str (write-to-string n)))
    (conv-to-list1 str)))

(let ((vals (conv-to-list1 (dialog-item-text self))))
  (make-instance 'feedback-feature
:x (px (view-loc self))
:y (py (view-loc self))
:screen-obj self
:value (string-upcase (dialog-item-text self))
:width (point-h (view-size self))
:height (point-v (view-size self))
:kind 'fb-txt
:low (first vals)
:high (third vals)
)))

(let* ((str (write-to-string n)))
  (conv-to-list1 str)))
(defmethod build-features-for :around ((self button-dialog-item) (vis-mod vision-module))
  (let* ((res (call-next-method))
         (oval-feat (first res))
         (vnn (view-nick-name (screen-obj oval-feat))))
    (setf (val oval-feat) (if vnn (write-name-to-string vnn))))  ;;; why not just dialog item text???
    res))

(defun dist-to-border (r)
  (cond ((< r 75) (- r 50))
        ((< r 125) (- r 100))
        ((< r 175) (- r 150))
        (t (- r 200))))

(defun feat-to-dmo :around ((self des-target-feature))
  (let* ((tn (mi::track-num (mi::targ-obj (screen-obj self))))
          (rng (floor (mi::range (mi::targ-obj (screen-obj self)))))
          (cv (max 0 (- 3 (floor rng *miles/segment*))))
          (pos (position \\: str)))
    (if pos
        (setq str (subseq str (+ pos 2))))
    str))
(defmethod feat-to-dmo ((self gray-feature))
  (setf (attended-p self) t)
  (format t "feedback icon ~S" (color self))
  (make-dme (dmo-id self) (kind self)
    `(screen-pos ,(id (xy-to-dmo (xy-loc self) t))
      value ,(val self)
      color ,(color self))
  :obj (screen-obj self)
  :where :external))

(defmethod feat-to-dmo ((self help-text-feature))
  (setf (attended-p self) t)
  (make-dme (dmo-id self) (kind self)
    `(screen-pos ,(id (xy-to-dmo (xy-loc self) t))
      value ,(val self)
      color ,(color self)
      low ,(low self)
      high ,(high self))
  :obj (screen-obj self)
  :where :external))

(defmethod feat-to-dmo ((self help-cv-feature))
  (setf (attended-p self) t)
  (make-dme (dmo-id self) (kind self)
    `(screen-pos ,(id (xy-to-dmo (xy-loc self) t))
      value ,(val self)
      color ,(color self))
  :obj (screen-obj self)
  :where :external))

(defmethod feat-to-dmo ((self help-wg-feature))
  (setf (attended-p self) t)
  (make-dme (dmo-id self) (kind self)
    `(screen-pos ,(id (xy-to-dmo (xy-loc self) t))
      value ,(val self)
      color ,(color self))
  :obj (screen-obj self)
  :where :external))

(defmethod feat-to-dmo :around ((self radio-button-feature))
  (setf (attended-p self) t)
  (make-dme (dmo-id self) (kind self)
    `(screen-pos ,(id (xy-to-dmo (xy-loc self) t))
      value ,(val self)
      color ,(color self)
      pushed? ,(pushed? self))
  :obj (screen-obj self)
  :where :external))

(defmethod feat-to-dmo ((self track-pd-feature))
  (make-dme (dmo-id self) (kind self)
    `(screen-pos ,(id (xy-to-dmo (xy-loc self) t))
      value ,(val self)
(defmethod feat-to-dmo ((self plane-feature))
  (make-dme (dmo-id self) (kind self)
    (screen-pos ,(id (xy-to-dmo (xy-loc self) t))
      value ,(val self)
      color ,(color self))
  :obj (screen-obj self)
  :where :external))

;;; return which screen objects to process
;;;
(defun point-in-windoid-p (w point)
  (let ((x (the fixnum (point-h point)))
    (y (the fixnum (point-v point))))
    (and (>= x (the fixnum (point-h (view-position w))))
      (>= y (the fixnum (point-v (view-position w))))
      (<= x (the fixnum (+ (point-h (view-position w)) (point-h (view-size w)))))
      (<= y (the fixnum (+ (point-v (view-position w)) (point-v (view-size w)))))
      )))

(defun device-update-attended-loc ((self mcl-screen) xyloc)
  (when *attn-tracker*
    (let ((pt (make-point (px xyloc) (py xyloc)))
      (dolist (w (the-windows self))
        (if (window-shown-p w)
          (cond ((and (typep w 'windoid)
                        (point-in-windoid-p w pt))
            (let ((new-pt (subtract-points pt (view-position w))))
              (update-me *attn-tracker* w (vector (point-h new-pt) (point-v new-pt)))))))
      (update-me *attn-tracker* w (vector (point-h new-pt) (point-v new-pt))))
  )

(defmethod get-sub-objects ((win mi::radar-window))
  (mi::get-subviews win))
(defmethod get-sub-objects ((win mi::tracking-only-window))
  (mi::get-subviews win))
(defmethod get-sub-objects ((win mi::task-windoid))
  (mi::get-subviews win))
(defmethod get-sub-objects ((win mi::status-windoid))
  (remove nil (list (view-named 'mi::percent win) (view-named 'mi::track-percent win))))
(defmethod get-sub-objects ((win mi::info/decision-windoid))
  (remove-if #'(lambda(x) (eql (type-of x) 'decision-sequence-dialog-item)) (subviews win)))
(defmethod get-sub-objects ((win mi::help-windoid))
  (mi::get-subviews win))
(defmethod get-sub-objects ((win mi::fb-windoid))
  (let ((v (view-named 'mi::immed-result win))
    (v1 (view-named 'mi::targ-number win)))
    (remove nil (list v v1))))
(cond ((and (eql (type-of w) 'mi::radar-window) 
(< (point-h pt) (/ (point-h (view-size w)) 2)) 
(> (point-v pt) (point-v (view-position w))) 
(update-me *attn-tracker* w xyloc)
)))))))

(when (member "emma" *file-list :test #'equal)

(defun mcl-screen-update-me ((eye eye-spot) (self mcl-screen) xyloc)
  (let ((pt (make-point (px xyloc) (py xyloc))))
    (dolist (w (the-windows self))
      (if (window-shown-p w)
        (cond ((and (typep w 'windoid) 
                    (point-in-windoid-p w pt))
          (let ((new-pt (subtract-points pt (view-position w))))
            (update-me eye w (vector (point-h new-pt) (point-v new-pt))))))
        (cond ((and (eql (type-of w) 'mi::radar-window) 
                    (< (point-h pt) (/ (point-h (view-size w)) 2)) 
                    (> (point-v pt) (point-v (view-position w))) 
                    (update-me eye w xyloc)
))))

(defun device-update-attended-loc ((self mcl-screen) xyloc)
  (when *attn-tracker*
    (let ((pt (make-point (px xyloc) (py xyloc))))
      (dolist (w (the-windows self))
        (if (window-shown-p w)
          (cond ((and (typep w 'windoid) 
                      (point-in-windoid-p w pt))
            (let ((new-pt (subtract-points pt (view-position w))))
              (update-me *attn-tracker* w (vector (point-h new-pt) (point-v new-pt))))))
          (cond ((and (eql (type-of w) 'mi::radar-window) 
                        (< (point-h pt) (/ (point-h (view-size w)) 2)) 
                        (> (point-v pt) (point-v (view-position w))) 
                        (update-me *attn-tracker* w xyloc)
))))

(defun view-draw-contents :after ((self eye-spot))
  (push (make-instance 'mi::eye-track-data :event 'mi::eyedata 
                             :rt (get-internal-real-time) 
                             :status 0 
                             :pupil-diam 50 
                             :point-of-gaze-h (point-h (view-position self)) 
                             :point-of-gaze-v (point-v (view-position self)) 
                             :mouse (view-mouse-position nil) mi::*log-data*)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun reset-model ()
  ;; (pm-reset)
  (pm-reset :initialize-dm t) ;; makes it like init-rpm
  (add-scoring-chunks)
  ;; setup device
  (pm-install-device (make-instance 'mcl-screen))
  (cleargoalstack)
  (goal-focus next)
  ;; model stuff
  (setf *debug* t)
  ;; rpm parameters
  (set-rpm-parameters)
  (set-cs-hook-vars)
  ;; collect-statistics
  (setq *model-stats* (make-instance 'model-stats))
  ;; (pm-timed-event 30 #'collect-statistics *model-stats*)
)

(defun init-model ()
  ;; clear stuff
  (pm-reset :initialize-dm t)
  ;; setup device
  (pm-install-device (make-instance 'mcl-screen))
  ;; scoring stuff
  (add-scoring-chunks)
  (init-declarative-memory)
  ;; goal stuff
  (goal-focus read-conditions)
  ;; model stuff
  (setf *debug* t)
  ;; rpm parameters
  (set-rpm-parameters)
  (setq *conflict-set-hook-fn* #'cs-hook)
  ;; collect-statistics
  (setq *model-stats* (make-instance 'model-stats))
  ;; (pm-timed-event 30 #'collect-statistics *model-stats*)
)

(defun my-dummy-func ()

)

(defun get-real->cue ()
  (wme-type-wmes
cdr
  (nth
   (position 'real->cue (mapcar #'car *declarative-memory*))
   *declarative-memory*)))))

(defun add-scoring-chunks ()
  (let* ((g (mi::get-game))
         (r (mi::range g))
         (d (mi::bearing/course g))
         (mi::add-scoring-chunks))

)
(a (mi::altitude g))
(weights (mi::weights g))
(r-0-l (first (first r)))
(r-0-h (second (first r)))
(d-0-l (first (first d)))
(d-0-h (second (first d)))
(a-0-l (first (first a)))
(a-0-h (second (first a)))
(r-1-l (first (second r)))
(r-1-h (second (second r)))
(d-1-l (first (second d)))
(d-1-h (second (second d)))
(a-1-l (first (second a)))
(a-1-h (second (second a)))
(r-2-l (first (third r)))
(r-2-h (second (third r)))
(d-2-l (first (third d)))
(d-2-h (second (third d)))
(a-2-l (first (third a)))
(a-2-h (second (third a)))
(r-3-l (first (fourth r)))
(r-3-h (second (fourth r)))
(a-3-l (first (fourth a)))
(a-3-h (second (fourth a)))
(r-wgt (get-wme (nth (second weights) wgt-trans)))
(d-wgt (get-wme (nth (first weights) wgt-trans)))
(a-wgt (get-wme (nth (third weights) wgt-trans)))
(bvl-settings '((100000.0 -100000.0) (1 5000.0) (1 -100000.0)))
(alt-val (nth (altitude-help *model-profile*) bvl-settings))
(rng-val (nth (range-help *model-profile*) bvl-settings))
(app-val (nth (appdist-help *model-profile*) bvl-settings)))
(add-dm-fct (list
  '(range-0 isa real->cue attribute range cue-value zero low ,r-0-l high
    ,r-0-h)
  '(range-1 isa real->cue attribute range cue-value one low ,r-1-l high ,r-1-h)
  '(range-2 isa real->cue attribute range cue-value two low ,r-2-l high ,r-2-h)
  '(range-3 isa real->cue attribute range cue-value three low ,r-3-l high ,r-3-h)
  '(altitude-0 isa real->cue attribute altitude cue-value zero low ,a-0-l high ,a-0-h)
  '(altitude-1 isa real->cue attribute altitude cue-value one low ,a-1-l high ,a-1-h)
  '(altitude-2 isa real->cue attribute altitude cue-value two low ,a-2-l high ,a-2-h)
  '(altitude-3 isa real->cue attribute altitude cue-value three low ,a-3-l high ,a-3-h)
  '(appdist-00 isa real->cue attribute appdist cue-value zero low 9999 high
    9999)
  '(appdist-0 isa real->cue attribute appdist cue-value zero low ,d-0-l high ,d-0-h)
  '(appdist-1 isa real->cue attribute appdist cue-value one low ,d-1-l high ,d-1-h)
  '(appdist-2 isa real->cue attribute appdist cue-value two low ,d-2-l high ,d-2-h)
  '(range-wgt isa weight attribute range wgt ,r-wgt)
`(altitude-wgt isa weight attribute altitude wgt ,a-wgt)
`(appdist-wgt isa weight attribute appdist wgt ,d-wgt)

(set-base-level (get-wme 'altitude-0) alt-val)
(set-base-level (get-wme 'altitude-1) alt-val)
(set-base-level (get-wme 'altitude-2) alt-val)
(set-base-level (get-wme 'altitude-3) alt-val)

(set-base-level (get-wme 'range-0) rng-val)
(set-base-level (get-wme 'range-1) rng-val)
(set-base-level (get-wme 'range-2) rng-val)
(set-base-level (get-wme 'range-3) rng-val)

(set-base-level (get-wme 'appdist-0) app-val)
(set-base-level (get-wme 'appdist-1) app-val)
(set-base-level (get-wme 'appdist-2) app-val)
(set-base-level (get-wme 'appdist-00) app-val)

(defun set-strengths (&rest types)
  (for-all-wmes x
    (if (member (wme-type-name (wme-type x)) types)
      (set-base-level x '(1000000 -1000000)))))  ;;'(1000 0.0)

(defun init-declarative-memory ()
  (cond (*base-level-learning* (set-strengths 'addition-fact 'multiplication-fact 'weight 'real-number 'integer))))

(defun get-real->cue-chunks (attr all-ranges)
  (remove nil (mapcar '#(lambda (x) (if (eql (wme-name (get-slot-value x 'attribute)) attr) x))
    all-ranges)))

(defun get-number-of-chunks (typ)
  (case typ
    (all
      (let ((res 0))
        (dolist (a *declarative-memory*)
          (setq res (+ res (length (wme-type-wmes (cdr a)))))))
        res))
    (targ-attributes
      (reduce #'+ (mapcar #'(lambda (x) (length (wme-type-wmes (cdr x))))
        all-ranges)))
    (goals
      (reduce #'+ (mapcar #'(lambda (x) (length (wme-type-wmes (cdr (assoc typ *declarative-memory*)))))
        all-ranges)))
    (otherwise
      (length (wme-type-wmes (cdr (assoc typ *declarative-memory*)))))))

(defmethod collect-statistics ((obj model-stats))
  (let ((tm (pm-time)))
    (defun set-strengths (&rest types)
      (for-all-wmes x
        (if (member (wme-type-name (wme-type x)) types)
          (set-base-level x '(1000000 -1000000)))))  ;;'(1000 0.0)

(defun init-declarative-memory ()
  (cond (*base-level-learning* (set-strengths 'addition-fact 'multiplication-fact 'weight 'real-number 'integer))))

(defun get-real->cue-chunks (attr all-ranges)
  (remove nil (mapcar '#(lambda (x) (if (eql (wme-name (get-slot-value x 'attribute)) attr) x))
    all-ranges)))

(defun get-number-of-chunks (typ)
  (case typ
    (all
      (let ((res 0))
        (dolist (a *declarative-memory*)
          (setq res (+ res (length (wme-type-wmes (cdr a)))))))
        res))
    (targ-attributes
      (reduce #'+ (mapcar #'(lambda (x) (length (wme-type-wmes (cdr x))))
        all-ranges)))
    (goals
      (reduce #'+ (mapcar #'(lambda (x) (length (wme-type-wmes (cdr (assoc typ *declarative-memory*)))))
        all-ranges)))
    (otherwise
      (length (wme-type-wmes (cdr (assoc typ *declarative-memory*)))))))

(defmethod collect-statistics ((obj model-stats))
  (let ((tm (pm-time))
(slots (mapcar #'first (ccl::class-instance-slots (find-class 'model-stats)))))
(setf (slot-value obj (first slots)) tm)
(dolist (s (rest slots))
  (push (get-number-of-chunks s) (slot-value obj s)))
(pm-timed-event (+ tm 30) #'collect-statistics obj))

(defun write-actr-stats (fn)
  (let ((sn (concatenate 'string "S3" (subseq (second fn) 3))))
      (let* ((slot-labels (mapcar #'first (ccl::class-instance-slots (find-class 'model-stats))))
            (slot-values (mapcar #'(lambda (x) (reverse (slot-value *model-stats* x)))
                                (rest slot-labels))))
        (dotimes (i (length slot-labels))
          (write (read-from-string (symbol-name (nth i slot-labels))) :stream fs)
          (write-char #\tab fs)
          (write-char #\newline fs)
          (dotimes (j (length (all *model-stats*)))
            (write (* j 30) :stream fs)
            (write-char #\tab fs)
            (dotimes (i (length slot-values))
              (write (nth j (nth i slot-values)) :stream fs)
              (write-char #\tab fs)
              (write-char #\newline fs)))))

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

(defun enable-color-cond ()
)

(defvar *rehearse-wcv* 1)

(defun enable-dot-cond ()
  (with-slots (class-status) *model-profile*
    (let ((tv (* 0.0125 (- class-status 5))))
      (when (null (zerop tv))
        (if (plusp tv)
            (progn
              (spp-fct `(try-retrieval-first :p ,(- 1 v)))
              (spp-fct `(try-retrieval-first-hooked :p ,(- 1 v)))
              (spp-fct `(move-cursor-to-target :p ,(- 1 v)))
              (progn
                (spp-fct `(encode-selection :p ,(- 1 (abs v))))
                (spp-fct `(encode-selection-hooked :p ,(- 0.9875 (abs v)))))))
            (spp rehearse-then-hook :p 0.4)
            (spp class-condition-check-radio-button :p .9) ;;dot condition
            (spp look-after-no-dot :p .4)) ;;no dot condition

(defun disable-dot-cond ()
  (spp try-retrieval-first :p 0.9)
  (spp encode-selection :p 0.9)
  (spp initiate-hook :p 0.4)
  (spp move-cursor-to-target :p 0.9875))
(spp class-condition-check-radio-button :p .4) ;;dot condition
(spp look-after-no-dot :p .9)) ;;no dot condition

(defun disable-feedback ()
  (spp calc-threat-value-8-no-fb :p 0.9)
  (spp calc-threat-value-8-fb :p 0.4)
  (setq *rehearse-wcv* 10)
  (if (tracking "model-profile")
    (spp calc-threat-value-8-fb-track :p 0.4)))

(defun enable-feedback ()
  (setq *rehearse-wcv* 5)
  (with-slots (class-fb ) "model-profile"
    (let ((v (* 0.0125 (- class-fb 5))))
      (when (null (zerop v))
        (if (plusp v)
            (spp-fct `(calc-threat-value-8-no-fb :p ,(- 1 v)))
            (progn (spp-fct `(calc-threat-value-8-fb :p ,(- 1 (abs v))))
                (if (tracking "model-profile")
                   (spp-fct `(calc-threat-value-8-fb-track :p ,(- 1 (abs v))))))))))

(defun set-model ((obj model-profile))
  (with-slots (tracking track-number range-strategy fb-tn reclassify wm-capacity) "model-profile"
    (let ((v (* 0.0125 (- track-number 5))))
      (when (null (zerop v))
        (if (plusp v)
            (spp-fct `(initial-classification-start :p ,(- 1 v))) ; low
            (spp-fct `(initial-classification-verify-tn :p ,(- 1 (abs v)))) ; high
        )))
    (let ((v (* 0.0125 (- range-strategy 5))))
      (when (null (zerop v))
        (if (plusp v)
            (spp-fct `(initial-range-vis :p ,(- 1 v)))
            (spp-fct `(initial-range-txt :p ,(- 1 (abs v))))))
    (let ((v (* 0.0125 (- wm-capacity 5))))
      (when (null (zerop v))
        (if (plusp v)
            (spp-fct `(init-get-cue-values-recall :p ,(- 1 v)))
            (spp-fct `(init-get-cue-values-no-recall :p ,(- 1 (abs v))))))
    (let ((v (* 0.025 (- fb-tn 5))))
      (when (null (zerop v))
        (if (plusp v)
            (spp-fct `(feedback-done :p ,(- 1 v)))
            (spp-fct `(feedback-chk-tn :p ,(- 1 (abs v))))))
    (let ((v (* 0.025 (- reclassify 5))))
      (when (null (zerop v))
        (cond ((plusp v)
              (spp-fct `(feedback-error-1 :p ,(- 1 v)))) ;; ignore tn and don't try to reclassify
              (t
               (spp-fct `(feedback-error-2 :p ,(- 1 (abs v))))))) ;; look at track number and try to reclassify
        (spp-fct `(feedback-error-3 :p ,(- 1 (abs v))))))) ;; don't look and track number and try to reclassify
    (cond (tracking
      (spp-fct `(feedback-error-4 :p ,(- 1 (abs v)))))))
(setq *track-duration* tracking)
(setq *performance-difference* (* 5 (- tracking 5)))
(let ((v (* 0.0125 (- tracking 5))))
  (cond ((plusp v)
         (spp-fct `(select-target-next-indirect :p ,(- 1 v)))
         (spp-fct `(compare-task-% :p ,(- 1.0 (/ v 2) )))) ;;;;more switches
         (spp-fct `(switch-goals :p ,(- 1 v)))
         (spp-fct `(checking-pd-status-ok :p ,(- 1 v))))) )
(t
  (spp-fct `(check-tracking-state :p ,(- 1 (abs v)))))) ;;tracks until
blue + 3 (random *track-interval*)
(spp-fct `(compare-task-% :p ,(- 1 (abs v)))) ;;tracks when class
performance > track performance
(spp-fct `(switch-goals-tracking :p ,(- 1 (abs v))))
(spp-fct `(checking-pd-status-track-anyway :p ,(- 1 (abs v))))))
(spp check-class-% :p .6)
(spp check-track-% :p .6))
(t
  (spp select-target-next-indirect :p .9)
  (spp check-class-% :p .6)
  (spp check-tracking-state :p 0)
  (spp compare-task-% :p 0)
  (spp check-track-% :p 0)
  (spp switch-goals-tracking :p 0)
  (spp calc-threat-value-8-fb-track :p 0)
  (spp checking-pd-status-track-anyway :p 0)))
)

(defun install-model-profile (&key tracking class-status track-number range-strategy
                                class-fb
                                fb-tn range-help altitude-help appdist-help wm-capacity att-span
                                r->l b->t screen-area initial-phase reclassify)
  (setq *model-profile* (make-instance 'model-profile
                                         :tracking tracking
                                         :class-status class-status
                                         :track-number track-number
                                         :range-strategy range-strategy
                                         :class-fb class-fb
                                         :fb-tn fb-tn
                                         :range-help range-help
                                         :altitude-help altitude-help
                                         :appdist-help appdist-help
                                         :wm-capacity wm-capacity
                                         :att-span att-span
                                         :att-span att-span
                                         :r->l r->l
                                         :b->t b->t
                                         :screen-area screen-area
                                         :initial-phase initial-phase
                                         :reclassify reclassify))

  (set-model *model-profile*)
)

(defun update-targ-feature (targ &key color pos mge)
  (declare (ignore color))
(let ((targ-feat (find-if #'(lambda (x)
    (and (typep x 'des-target-feature)
        (if (numberp targ)
            (eql targ (mi::track-num (mi::targ-obj (screen-obj x))))
            (equal targ (mi::targ-obj (screen-obj x)))))))
    (visicon (vis-m *mp*)))))
  (cond (targ-feat
    ;(if color (setf (color targ-feat) (cdr (assoc color colors))))
    (if pos (setf (screen-x targ-feat) (point-h pos)))
    (if pos (setf (screen-y targ-feat) (+ (point-v pos) 20)))
    (setf (val targ-feat) (get-value-feat 'sector rnge))
    (setf (color targ-feat) (get-value-feat 'arc rnge))))))
)

(defun update-tracking (&key plane-pos pd-color)
  (let ((plane (find 'plane-feature (visicon (vis-m *mp*)) :key #'type-of))
    (pd (find 'track-pd-feature (visicon (vis-m *mp*)) :key #'type-of)))
    (when (and plane plane-pos)
      (setf (screen-x plane) (point-h plane-pos))
      (setf (screen-y plane) (point-v plane-pos)))
    (when (and pd pd-color)
      (setf (color pd) (cdr (assoc pd-color colors))))))

(defun get-tn-from-feat-lis (feat-lis)
  (mapcar #'(lambda (x)
    (if (typep x 'des-target-feature)
        (mi::track-num (mi::targ-obj (screen-obj x)))
        x)) feat-lis))

(defun get-attended-from-feat-lis (feat-lis)
  (remove nil (mapcar #'(lambda (x)
    (if (typep x 'des-target-feature)
        (list (attended-p x) (mi::track-num (mi::targ-obj (screen-obj x)))
        ))
    feat-lis)))

(defun get-model-time ()
  ;(round (/ (get-internal-real-time) 16.6666667))
  (round (/ (* (mp-time *mp*) 1000.000) 16.666666)))

(defun log-model-data (&rest r)
  (mi::log-event (make-instance 'mi::model-data :event r :model-time (get-model-time))))

(defun clear-info-attended ()
  (dolist (f (visicon (vis-m *mp*))
    (if (and (typep f 'info-text-feature) (member (kind f) '(targ-range targ-altitude
targ-appdist))) (attended-p f))
      (setf (attended-p f) nil)))

(defun print-cycles (&key (fn "cl:\muri;temp.lisp") search-pos (prnt t) (model t))
  (with-open-file (fs (if fn fn (choose-file-dialog)) :direction :input)
(let (inp (err nil) (res nil))
  (ccl::while (not (typep err 'end-of-file))
    (multiple-value-setq (inp err) (ignore-errors (read-line fs)))
    (typecase err
      (end-of-file)
      (NULL
        (setq inp (conv-to-list1 (remove-spec-chars inp))))
      (cond ((or
               (member 'fired inp)
               (member 'model inp))
             (member 'cs-hook inp)
             (if (and search (null pos))
                 (and (eql (first inp) 'time)
                      (member search inp)))
             (if (and search (numberp pos))
                 (eql search (nth pos inp)))
             (when (null (member 'eye inp))
               (if prnt
                 (format t "~%~S" inp)
                 (push inp res)))))
      (otherwise
        (format t "~%Error reading file, error = ~S" err))
      ))
  res))

(defun print-cs-hook (&key (fn "ccl:muri;temp.lisp") (prnt t))
  (with-open-file (fs (if fn fn (choose-file-dialog)) :direction :input)
    (let (inp (err nil) (res nil))
      (ccl::while (not (typep err 'end-of-file))
        (multiple-value-setq (inp err) (ignore-errors (read-line fs)))
        (typecase err
          (end-of-file)
          (NULL
            (setq inp (conv-to-list1 (remove-spec-chars inp))))
          (cond ((member 'cs-hook inp)
                   (when (null (member 'eye inp))
                     (if prnt
                       (format t "~%~S" inp)
                       (push inp res)))))
          (otherwise
            (format t "~%Error reading file, error = ~S" err))
          ))
      res))

(defun count-wmes ()
  (let ((res 0))
    (dolist (wtype *declarative-memory*)
      (if (numberp (length (wme-type-wmes (rest wtype))))
        (incf res (length (wme-type-wmes (rest wtype))))
        res))
  res))

(defun get-activations (typ)
  (mapcar #\wme-activation (wme-type-wmes (rest (assoc typ *declarative-memory*))))

(defun remove-spec-chars (x)
  (remove-if #\'(lambda(y) (member y `(#\; #\:) #\\))) x))
(defun conv-to-list1 (x)
  (cond (x
      (ignore-errors (read-from-string (concatenate 'string "(" (string-trim '(#\Space
#: #\?) x) ")"))))))

(defun count-prods ()
  (let ((fired (mapcar #'third (print-cycles :fn "ccl:muri;temp.lisp" :prnt nil :model
nil)))
    (prods (mapcar #'first *procedural-memory*))
    (total-count 0)
    (total-actions 0)
    (total-retrievals 0)
    (total-subgoals 0)
    (para-count 0))
  (dolist (p prods)
    (format t "~S ~S" p (count p fired))
    (incf total-count (count p fired)))
  (dolist (f fired)
    (let* ((rhs (mapcar #'first (production-rhs (rest (assoc f *procedural-memory*)))))
      (actions 0) (retrievals 0) (subgoals 0))
      (setq actions (count 'action-command rhs))
      (incf total-actions actions)
      (setq retrievals (count 'handle-failure rhs))
      (incf total-retrievals retrievals)
      (setq subgoals (count 'focus-fct rhs))
      (incf total-subgoals subgoals)
      (if (> (+ actions retrievals subgoals) 1) (incf para-count)))
  (list total-count para-count total-actions total-retrievals total-subgoals)))

(defun save-counts ()
  (let ((fired (mapcar #'third (print-cycles :fn "ccl:muri;temp.lisp" :prnt nil :model
nil)))
    (prods (mapcar #'first *procedural-memory*))
    (res nil))
  (dolist (p prods)
    (push (list p (count p fired)) res)
    (mi::save-prod-cnts (reverse res)))

(defun idm (typ)
  (inspect (rest (assoc typ *declarative-memory*)))))

(defun goal-types ()
  (let ((res nil))
    (dolist (p "procedural-memory")
      (setf res (append res (list (production-goal-type (rest p))))))
    (inspect res)))

(defun get-slt-idx (typ slt-lst)
  (second (first (member typ slt-lst :key #'first))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;
(defclassic search-context ()
  (num-in-1-degrees 0)
  (num-in-2-degrees 0)
  (num-near-brdr 0)
  (num-after-brdr 0)
  (num-new-in-seg 0)
  (num-new-other 0)
  dot-cnd
  fb-cnd
  cur-time
  num-icon-attended
  %near-brdr
  %in-seg
  %in-seg-3)

(defvar *context* (make-instance 'search-context))

(defclassic search-prod ()
  pb
  c
  calc-pg-c
  noise-added
  new-pg-c
  context)

(defclassic select-target-within-area (search-prod))
(defclassic select-target-scan-continue (search-prod))

(defclassic select-target-cluster-within-sector (search-prod))
(defclassic select-target-cluster (search-prod))

(defclassic select-target-brdr-by-sector (search-prod))
(defclassic select-target-brdr-nearest (search-prod))

(defclassic select-target-new-target (search-prod))
(defclassic select-target-inner-sector (search-prod))

(defclassic select-target-random (search-prod))
(defclassic select-target-random-cur-seg (select-target-random))

(defvar *prod-list* (mapcar 'make-instance '(select-target-scan-continue select-target-
  cluster-within-sector
  select-target-brdr-by-sector
  select-target-brdr-nearest
  select-target-new-target select-target-inner-sector
  select-target-random select-target-random-cur-seg))

(defvar *last-search-prod* nil)
(defvar *last-search-loc* nil)
(defvar *loc-retrieval-error* 0)
(defvar *prod-fail-list* nil)
(defvar *current-state* 'initial)
(defvar *context-trace* nil)
(defvar *prod-trace* nil)

(defun set-cs-hook-vars ()
  (setq *conflict-set-hook-fn* #'cs-hook)
  (setq *last-search-prod* nil)
  (setq *last-search-loc* nil)
  (setq *loc-retrieval-error* 0)
  (setq *prod-fail-list* nil)
  (setq *current-state* 'initial))

(defvar *cur-loc* nil)

(defun cs-hook (cs)
  (when cs
    (let* ((g (aref (first cs) 3)))
      (when (eql (wme-type g) (get-safe-type 'select-target))
        (let ((step (get-slot-value g 'step))
              (return-prod nil))
          (cond ((eql step (get-safe-wme 'find-target))
                 (case *current-state*
                 ((success prod-fail)
                  (if (eql *current-state* 'prod-fail)
                   (push *last-search-prod* *prod-fail-list*)
                   (setf *prod-fail-list* nil))
                 ;; found a target
                 (setq *cur-loc* (cond ((eql *current-state* 'success)
                                       (get-slot-value g 'loc))
                          (*last-search-loc*)
                          (incf *loc-retrieval-error*)
                          *last-search-loc*)
                (t)
                (incf *loc-retrieval-error*)
                (let* ((targs (no-output (sdm isa screen-targ)))
                        (targ (if targs (nth (random (length targs))
                                       targs))))
                (let* ((cur 0)
                       (res nil))
                  (dolist (prod *prod-list*)
                    (when (and (null (member (type-of prod) *prod-fail-list*)))
                      (when *cur-loc*
                        (make-search-context *cur-loc* ~S *cur-loc*)
                        (push (copy-instance *context*) *context-trace*)
                        (dolist (prod *prod-list*)
                          (compute-new-pg-c prod *current-state* *last-search-prod*)
                          (let ((cur 0)
                                (res nil))
                            (dolist (prod *prod-list*)
                              (when (and (null (member (type-of prod) *prod-fail-list*)))
                                (new-pg-c prod) cur))
                            (setq res prod)
                            (setq cur (new-pg-c prod)))))
                  (when res
                    (setq return-prod (type-of res))
                    (setf (context res) (copy-instance *context*)
                          (push (copy-instance res) *prod-trace*))
                    )
                )))))
    )
  )))
(setq *last-search-loc* *cur-loc*))
  (initial ;; initial search
    (setq return-prod 'select-target-within-area))
)
(if (null return-prod) (setq return-prod 'select-target-random))
(setq *last-search-prod* return-prod)
(find return-prod cs :key #'(lambda (x) (production-name (aref x 0)))))
)
(eql step (get-safe-wme 'select-next))
(setq *current-state* 'success)
nil)
(eql step (get-safe-wme 'new-strategy))
(setq *current-state* 'prod-fail)
nil)
)
))))
)

(defun make-search-context (cur-loc)
  (with-slots (num-in-1-degree num-in-2-degree num-near-brdr num-after-brdr num-new-in-
  seg num-new-other
    *context*
      )))
(setq num-in-1-degree 0 num-in-2-degree 0 num-near-brdr 0 num-after-brdr 0
    num-new-in-seg 0 num-new-other 0)
  (let* ((cur-x (get-safe-slot-value cur-loc 'screen-x)) ;;;;(aref (wme-slots cur-loc)
    (get-slt-idx 'screen-x slt-lst))) ;;get screen-x from cur-loc
    (cur-y (get-safe-slot-value cur-loc 'screen-y)) ;;get screen-y from cur-loc
    (cur-seg (get-safe-slot-value cur-loc 'value))  ;;get segment from cur-loc
    (pix1 (/ (pm-angle-to-pixels 1.0) 2))
    (pix2 (/ (pm-angle-to-pixels 2.0) 2))
    (x-min-1 (- cur-x pix1))
    (x-max-1 (+ cur-x pix1))
    (y-min-1 (- cur-y pix1))
    (y-max-1 (+ cur-y pix1))
    (x-min-2 (- cur-x pix2))
    (x-max-2 (+ cur-x pix2))
    (y-min-2 (- cur-y pix2))
    (y-max-2 (+ cur-y pix2))
    (num-attended 0)
    (targs-near-a-brdr 0)
    (num-in-seg 0)
    (num-targs 0)
    (num-in-seg-3 0))
  (setq dot-cnd (equal "on" (get-slot-value (get-wme 'read-conditions) 'class)))
  (setq fb-cnd (equal "on" (get-slot-value (get-wme 'read-conditions) 'fb)))
  (setq cur-time (/ (mp-time *mp*) 60.0)) ;; time in minutes
  (setq num-targs (count-if #'(lambda (x) (typep x 'des-target-feature)) (visicon
    (vis-m *mp*)))))
(dolist (obj (visicon (vis-m *mp*) ( novisor (vis-m *mp*)))))
  (when (typep obj 'des-target-feature)
    (if (eql cur-seg (val obj)) (incf num-in-seg))
    (when (and (<= x-min-1 (screen-x obj) x-max-1)
                (<= y-min-1 (screen-y obj) y-max-1))
      (incf num-in-1-degree))
    (when (and (<= x-min-2 (screen-x obj) x-max-2)
                (<= y-min-2 (screen-y obj) y-max-2))
      (incf num-in-2-degree)))
(when (color obj)
  (incf targs-near-a-brdr))
(let* ((rng (floor (mi::range (mi::targ-obj (screen-obj obj)))))
  (brdr-dist (dist-to-border rng))
  (if (< rng 50) (incf num-in-seg-3))
  (cond ((<= 0 brdr-dist (* pix2 mi::*miles/pixel*))
    (incf num-near-brdr))
    ((< (* -1 (* pix2 mi::*miles/pixel*)) brdr-dist 0)
    (incf num-after-brdr))))
(if (eql t (attended-p obj)) (incf num-attended) )
(when (eql 'new (attended-p obj))
  (if (eql cur-seg (val obj)) (incf num-new-in-seg) (incf num-new-other)))
))
(setf num-icon-attended num-attended)
(setq %-near-brdr (/ targs-near-a-brdr num-targs))
(setq %-in-seg (/ num-in-seg num-targs))
(setq %-in-seg-3 (/ num-in-seg-3 num-targs))))

(defun compute-new-pg-c ((self select-target-scan-continue) last-state last-prod)
  (with-slots (cur-time %near-brdr) *context*
    (with-slots (pb c new-pg-c calc-pg-c noise-added) self
      (setq pb 0 c 0)
      (when (or (and (eql last-state 'success)
          (eql last-prod (type-of self))))
        (and (eql last-state 'prod-fail)
          (not (eql last-prod (type-of self))))))
      (incf pb .4))
    (when (< cur-time (+ 0.6 (* 0.1 (random 10)))) (incf pb .4))
    (when (< cur-time (+ (initial-phase *model-profile*) (* 0.1 (random 10)))) (incf pb .2))
    (when (< %-near-brdr 0.10) (incf pb .2))
    (when (and (eql last-state 'prod-fail)
      (eql last-prod (type-of self)))
      (incf c 1.0))
    (setq calc-pg-c (- (* pb *g*) c))
    (setq noise-added (noise *exp-gain-noise*)
      (setq new-pg-c (+ calc-pg-c noise-added))
    )))

(defun compute-new-pg-c ((self select-target-cluster-within-sector) last-state last-
prod)
  (with-slots (cur-time num-in-2-degree %in-seg) *context*
    (with-slots (pb c new-pg-c calc-pg-c noise-added) self
      (setq pb 0 c 0)
      (when (or (and (eql last-state 'success)
          (eql last-prod (type-of self))))
        (and (eql last-state 'prod-fail)
          (not (eql last-prod (type-of self))))))
      (incf pb .2))
    (when (> cur-time (+ (initial-phase *model-profile*) (* 0.1 (random 10)))) (incf pb .2))
    (when (> num-in-2-degree 1) (incf pb .4))
    (when (> %in-seg .4) (incf pb .4))
    (when (and (eql last-state 'prod-fail)
      (eql last-prod (type-of self)))
      (incf c 1.0))
    (setq calc-pg-c (- (* pb *g*) c))
    (setq noise-added (noise *exp-gain-noise*)
      (setq new-pg-c (+ calc-pg-c noise-added))
    ))
(defmethod compute-new-pg-c ((self select-target-cluster) last-state last-prod)
 (with-slots (cur-time num-in-2-degree num-in-1-degree %in-seg) *context*
 (with-slots (pb c new-pg-c calc-pg-c noise-added) self
 (setq pb 0 c 0)
 (when (or (and (eql last-state 'success)
 (eql last-prod (type-of self)))
 (and (eql last-state 'prod-fail)
 (not (eql last-prod (type-of self))))))
 (incf pb .5))
 (when (> cur-time (+ (initial-phase *model-profile*) (* 0.1 (random 10))))
 (incf pb .2)
 (when (> num-in-1-degree 1) (incf pb .4))
 (when (> num-in-2-degree 3) (incf pb .4))
 (when (> %in-seg .4) (incf pb .2))
 (when (and (eql last-state 'prod-fail)
 (eql last-prod (type-of self)))
 (incf c 1.0))
 (setq calc-pg-c (- (* pb *g*) c))
 (setq noise-added (noise *exp-gain-noise*)
 (setq new-pg-c (+ calc-pg-c noise-added)))
))

(defmethod compute-new-pg-c ((self select-target-brdr-by-sector) last-state last-prod)
 (with-slots (cur-time %near-brdr num-near-brdr num-after-brdr dot-cnd) *context*
 (with-slots (pb c new-pg-c calc-pg-c noise-added) self
 (setq pb 0 c 0)
 (when (or (and (eql last-state 'success)
 (eql last-prod (type-of self)))
 (and (eql last-state 'prod-fail)
 (not (eql last-prod (type-of self))))))
 (incf pb .2))
 (when (> cur-time (+ (initial-phase *model-profile*) (* 0.1 (random 10))))
 (cond ((null dot-cnd)
 (incf pb .4)
 (setq *prod-fail-list* (remove 'select-target-brdr-by-sector *prod-fail-list*)))
 (t
 (incf pb .2))))
 (when (> %near-brdr 0.10) (incf pb .2))
 (when (> num-near-brdr 2) (incf pb .2))
 (when (> num-after-brdr 2) (incf pb .2))
 (when (and (eql last-state 'prod-fail)
 (eql last-prod (type-of self)))
 (incf c 1.0))
 (setq calc-pg-c (- (* pb *g*) c))
 (setq noise-added (noise *exp-gain-noise*)
 (setq new-pg-c (+ calc-pg-c noise-added)))
))

(defmethod compute-new-pg-c ((self select-target-brdr-nearest) last-state last-prod)
 (with-slots (cur-time %near-brdr dot-cnd) *context*
 (with-slots (pb c new-pg-c calc-pg-c noise-added) self
 (setq pb 0 c 0)
 (when (or (and (eql last-state 'success)
 (eql last-prod (type-of self)))
 (and (eql last-state 'prod-fail)))
 (and (eql last-state 'prod-fail) [(...)]))
(not (eql last-prod (type-of self))) (incf pb .2))
(when (> cur-time (+ (initial-phase *model-profile*) (* 0.1 (random 10))))
  (cond (null dot-cnd)
         (incf pb .4)
         (setq *prod-fail-list* (remove 'select-target-brdr-nearest *prod-fail-list*))
         (when (> cur-time (+ (initial-phase *model-profile*) (* 0.1 (random 10))))
          (incf pb .2))
         (t (incf pb .2)))
(when (> %-near-brdr 0.10) (incf pb .6))
(when (and (eql last-state 'prod-fail)
           (eql last-prod (type-of self)))
    (incf c 1.0))
(setq calc-pg-c (- (* 'pb *g*') c))
(setq noise-added (noise *exp-gain-noise*))
(setq new-pg-c (+ calc-pg-c noise-added)))
)
(defmethod compute-new-pg-c ((self select-target-inner-sector) last-state last-prod)
  (with-slots (cur-time num-new-in-seg num-new-other) *context*
    (with-slots (pb c new-pg-c calc-pg-c noise-added) self
      (setq pb 0 c 0)
      (when (or (and (eql last-state 'success)
                      (eql last-prod (type-of self)))
               (and (eql last-state 'prod-fail)
                    (not (eql last-prod (type-of self))))))
        (incf pb .2))
      (when (> cur-time (+ (initial-phase *model-profile*) (* 0.1 (random 10))))
        (incf pb .2))
      (when (> num-new-in-seg 2) (incf pb .2))
      (when (> num-new-other 2) (incf pb .2))
      (when (and (eql last-state 'prod-fail)
                  (eql last-prod (type-of self)))
        (incf c 1.0))
      (setq calc-pg-c (- (* 'pb *g*') c))
      (setq noise-added (noise *exp-gain-noise*))
      (setq new-pg-c (+ calc-pg-c noise-added)))
)
(defmethod compute-new-pg-c ((self select-target-random) last-state last-prod)
  (declare (ignore last-state last-prod))
  (with-slots (cur-time dot-cnd) *context*
(with-slots (pb c new-pg-c calc-pg-c noise-added) self
  (setq c 0)
  (if dot-cnd (setq pb 0.0) (setq pb 0.6))
  (when (> cur-time (+ 2.6 (* 0.1 (random 10)))) (incf pb .2))
  (setq calc-pg-c (- (* pb *g*) c))
  (setq noise-added (noise *exp-gain-noise*))
  (setq new-pg-c (+ calc-pg-c noise-added))
))
Curriculum Vitae

Michael Schoelles spent 18 years as a Software Engineer before coming to George Mason University. During this time his scientific computing experience included writing operating systems for deep-space probes and various projects for military agencies. His industrial experience includes writing transaction security software for the banking industry, as well as five years of experience in designing user interfaces. At George Mason University, his prior research includes creating and testing an intelligent tutoring system for second language learning.

Currently, he is a Research Scientist in the Applied Research in Cognition and Human Factors Laboratory at George Mason University. His research focuses on computational modeling of human computer interactions. His research interests include natural language processing, second language learning, cognitive workload, and computational cognitive modeling, and human-computer interaction.