```lisp
;;;
;;;   The Oz Interactive Fiction System
;;;     Copyright (c) 1990,1991 by Carnegie Mellon University
;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                 ;;;
;;;                        CLASS CAT-EM                   ;;;
;;;                                                                 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defvar *debug-emotions* nil)

; Emotion module for a cat agent

(class cat-em
  (em-for-actor)
  (slots

   ; passive goals that are noticed when they succeed
   ;     none in the cat agent
   (positive-passive-goals nil)

   ; passive goals that are noticed when they fail
   ;     don't ever want to be hurt
   ;     don't want to be touched when in an ignoring mood
   ;     don't want to be touched when in an aggressive mood
   (negative-passive-goals
                  '((dont-be-hurt 10
                     ,(test (or (did-see (an act
                                             verb '*hit
                                             object $$me))
                                (did-see (an act
                                             verb '*kick
                                             object $$me))
                                (did-see (an act
                                             verb '*bite
                                             object $$me))
                                (did-see (an act
                                             verb '*scratch
                                             object $$me))
                                (did-see (an act
                                             verb '*throw
                                             object $$me))
                                (did-see (an act
                                             verb '*throw
                                             iobject $$me)))))

                    (dont-be-touched-when-ignoring-or-aggressive 5
                     ,(test (and (or (match $features
                                             (list-containing 'aggressive))
                                      (match $features
                                             (list-containing 'ignoring)))
                                 (or (did-see (an act
                                                  verb '*touch
                                                  object $$me))
                                     (can-see (a player) containing
                                              (node $$me))
                                     (did-see (an act
                                                  verb '*take
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*pet
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*lick
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*push
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*nudge
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*polish
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*taste
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*rub
                                                  object $$me))
                                     (did-see (an act
                                                  verb '*pounce-on
                                                  object $$me)))))))

; be afraid of goal failures in certain situations
;     be afraid of being hurt when an enemy is around
;     be afraind of being touched when ignoring and an actor is around
;     be afrain of being topuched when aggressive and an actor is around
(fears '((dont-be-hurt .
          ,(test (and (can-see (a player ?reason)
                                nearby (node $$me))
                       (> (call relationship self ?reason 'dislike) 0))))

         (dont-be-touched-when-ignoring-or-aggressive .
          ,(test (and (can-see (a player ?reason)
                                nearby (node $$me))
                       (or (match $features
                                  (list-containing 'aggressive))
                           (match $features
                                  (list-containing 'ignoring))))))))

; be hopeful of goal achievements in certain situations
;     hopeful of being pet when friend is around
;     hopeful of door being opened when an actor is around
;     hopeful of object being gotten when an actor is around
(hopes '((wanting-to-be-pet .
          ,(test (and (can-see (a player ?reason)
                                nearby (node $$me))
                       (> (call relationship self ?reason 'like) 0))))

         (get-human-to-open-door .
          ,(test (can-see (a player ?reason)
                           nearby (node $$me))))

         (get-human-to-get-object .
          ,(test (can-see (a player ?reason)
                           nearby (node $$me))))))

; assign responsibility for goal successes
;     if pet - actor of pet act is responsible
;     if want door open - opener of door is responsible
;     if want an object - actor who gets object is responsible
(responsible-succ '((being-petted-and-like .
                     ,(test (or (did-see (an act
                                             verb '*pet
                                             actor ?responsible
                                             object $$me))
                                (did-see (an act
```

```
                                    verb '*rub
                                    actor ?responsible
                                    object $$me)))))

                        (being-held-and-like .
                         ,(test (can-see (a player ?responsible)
                                   containing (node $$me))))

                        (want-to-eat .
                         ,(test (can-see (a player ?responsible)
                                   nearby (node $$me))))

                        (get-human-to-open-door .
                         ,(test (did-see (an act
                                    verb '*open
                                    actor ?responsible
                                    object (a door)))))

                        (get-human-to-get-object .
                         ,(test (or (did-see (an act
                                    verb '*put
                                    actor ?responsible
                                    iobject (a room)))
                                 (did-see (an act
                                    verb '*give
                                    actor ?responsible
                                    iobject $$me))))))

 ; assign responsibility for goal failures
 ;     if hurt - actor of hurting act is responsible
 ;     if touched when in ignoring mood - toucher is responsiblt
 ;     if touched when in aggressive mood - toucher is responsible
 (responsible-fail `((dont-be-hurt .
                         ,(test (or (did-see (an act
                                    verb '*hit
                                    actor ?responsible
                                    object $$me))
                                 (did-see (an act
                                    verb '*kick
                                    actor ?responsible
                                    object $$me))
                                 (did-see (an act
                                    verb '*scratch
                                    actor ?responsible
                                    object $$me))
                                 (did-see (an act
                                    verb '*bite
                                    actor ?responsible
                                    object $$me))
                                 (did-see (an act
                                    verb '*throw
                                    actor ?responsible
                                    iobject $$me))
                                 (did-see (an act
                                    verb '*throw
                                    actor ?responsible
                                    object $$me)))))

                        (dont-be-touched-when-ignoring-or-aggressive .
                         ,(test (or (did-see (an act
                                    verb '*touch
                                    actor ?responsible
                                    object $$me))
                                 (can-see (a player ?responsible)
                                    containing (node $$me))
                                 (did-see (an act
                                    verb '*take
                                    actor ?responsible
                                    object $$me))
                                 (did-see (an act
                                    verb '*pet
                                    actor ?responsible
                                    object $$me))
                                  (did-see (an act
                                    verb '*lick
                                    object $$me))
                                  (did-see (an act
                                    verb '*push
                                    object $$me))
                                  (did-see (an act
                                    verb '*nudge
                                    object $$me))
                                  (did-see (an act
                                    verb '*polish
                                    object $$me))
                                  (did-see (an act
                                    verb '*taste
                                    object $$me))
                                 (did-see (an act
                                    verb '*rub
                                    actor ?responsible
                                    object $$me))
                                 (did-see (an act
                                    verb '*pounce
                                    actor ?responsible
                                    object $$me)))))

                        (get-human-to-open-door .
                         ,(test (can-see (a player ?responsible)
                                   nearby (node $$me))))

                        (become-petted-by .
                         ,(test (can-see (a player ?responsible)
                                   nearby (node $$me))))

                        (get-human-to-get-object .
                         ,(test (can-see (a player ?responsible)
                                   nearby (node $$me))))))

 ; start by slightly disliking the player
 (relationships '((player . ((dislike . 1) (like . -1)))))

 ; part of hack to get hiss when touched and ignoring/aggressive
 (has-hissed nil)
)

(methods

 ; ADJUST-FEATURES
 ; behavioral features for cats:
 ;     when hightened arousal, periodically act energetic
 ;     when angry or afraid act aggressively
 ;     when sad or feeling dislike act aloof (ignoring)
 ;     when happy, hopeful, or feeling like act friendly
 ;     otherwise alternate between curious and content
 (adjust-features ()
    (setf $features nil)
    (if (> (- (+ $hope-intensity
                 $fear-intensity
                 $anger-intensity)
```

```lisp
             (+ $distress-intensity
                $shame-intensity
                $remorse-intensity)) 0)
          (push 'energetic $features))
      (if (and (not (member 'energetic $features))
               (> (sin (call what-time-is-it $actor)) 0))
          (push 'energetic $features))
      (if (or (> $anger-intensity 4)
              (> $fear-intensity 9))
             (push 'aggressive $features))
      (if (and (> $joy-intensity 0)
               (= $anger-intensity 0)
               (= $fear-intensity 0)
               (= $hate-intensity 0)
               (= $distress-intensity 0)
               (not (member 'energetic $features))
               (not (member 'aggressive $features)))
          (push 'content $features))
      (if (or (and (< $fear-intensity 6)
                   (> $hate-intensity 0))
              (and (> $anger-intensity 0)
                   (< $fear-intensity 6)))
          (push 'ignoring $features))
      (if (and (or (> $love-intensity 2)
                   (> $hope-intensity 0)
                   (> $joy-intensity 2))
               (= $anger-intensity 0)
               (= $fear-intensity 0)
               (= $hate-intensity 0)
               (= $distress-intensity 0))
          (push 'friendly $features))
      (if (> $pride-intensity 0)
          (push 'proud $features)))

  ; COMPUTE-LOVE
  ; compute love caused by being near things you like
  (compute-love ()
    (dolist (o $objects)
            (let* ((potential (max 0 (call relationship self o 'like)))
                   (intensity (max 0 (- potential $love-threshold))))
              (setf $love-threshold (max 0 (- $love-threshold potential)))
              (setf $arousal (min $max-arousal (+ $arousal intensity)))
              (if (> potential 0)
                  (setf $love-affects
                        (call insert-affect self
                              (make love-affect
                                    attitude nil
                                    object o
                                    potential potential
                                    max-intensity intensity
                                    intensity intensity)
                              $love-affects)))))
    (call update-love-intensity self))

  ; COMPUTE-HATE
  ; compute hate caused by being near things you dislike
  (compute-hate ()
    (dolist (o $objects)
            (let* ((potential (max 0 (call relationship self o 'dislike)))
                   (intensity (max 0 (- potential $hate-threshold))))
              (setf $hate-threshold (max 0 (- $hate-threshold potential)))
              (setf $arousal (min $max-arousal (+ $arousal intensity)))
              (if (> potential 0)
                  (setf $hate-affects
                        (call insert-affect self
                              (make hate-affect
                                    attitude nil
                                    object o
                                    potential potential
                                    max-intensity intensity
                                    intensity intensity)
                              $hate-affects)))))
    (call update-hate-intensity self))

  ; COMPUTE-JOY
  ; determine joy caused by passive goals
  (compute-joy ()
   (setf $current-joy-affects nil)
   (dolist (goal $positive-passive-goals)
    (when (not (eq (matcher (caddr goal)) '*fail*))
      (let* ((joy-potential (cadr goal))
             (joy-intensity (max 0 (- joy-potential $joy-threshold))))
        (setf $joy-threshold
              (max 0 (- $joy-threshold joy-potential)))
        (setf $arousal (min $max-arousal (+ $arousal joy-intensity)))
        (when (> joy-potential 0)
              (setf $joy-affects
                    (call insert-affect self
                          (make joy-affect
                                goal (car goal)
                                event nil ; this should be an event later
                                potential joy-potential
                                max-intensity joy-intensity
                                intensity joy-intensity)
                          $joy-affects))
              (push (make joy-affect
                          goal (car goal)
                          event nil ; this should be an event later
                          potential joy-potential
                          max-intensity joy-intensity
                          intensity joy-intensity)
                    $current-joy-affects)))))
   (call update-joy-intensity self))

  ; COMPUTE-DISTRESS
  ; determine distress caused by passive goals
  (compute-distress ()
   (setf $current-distress-affects nil)
   (dolist (goal $negative-passive-goals)
     (when (not (eq (matcher (caddr goal)) '*fail*))
       (let* ((distress-potential (cadr goal))
              (distress-intensity (max 0 (- distress-potential
                                            $distress-threshold))))
         (setf $distress-threshold
               (max 0 (- $distress-threshold distress-potential)))
         (setf $arousal (min $max-arousal (+ $arousal distress-intensity)))
         (when (> distress-potential 0)

;;; crazy hack
    (cond ((member (car goal) '(dont-be-hurt))
           (distribute-sense-data $actor
             (make sound-data
                   any-object (make meow)
                   intensity 0)))
          ((and (not $has-hissed)
                (member (car goal) '(dont-be-touched-when-ignoring
                                     dont-be-touched-when-aggressive)))
           (setf $has-hissed t)
           (distribute-sense-data $actor
             (make sound-data
```

```
                   any-object (make hiss)
                   intensity 0)))
          (t nil))
;;; end crazy hack

                 (setf $distress-affects
                       (call insert-affect self
                             (make distress-affect
                                   goal goal
                                   event nil ; this should be an event later
                                   potential distress-potential
                                   max-intensity distress-intensity
                                   intensity distress-intensity)
                             $distress-affects))
                 (push (make distress-affect
                             goal goal
                             event nil ; this should be an event later
                             potential distress-potential
                             max-intensity distress-intensity
                             intensity distress-intensity)
                       $current-distress-affects)))))
     (call update-distress-intensity self))

    ; COMPUTE-ADMIRATION
    ; determine admiration caused by passive goals
    (compute-admiration ()
     (setf $current-admiration-affects nil)
     (dolist (goal $positive-passive-goals)
       (when (not (eq (matcher (caddr goal)) '*fail*))
         (let ((agent (call responsible-succ self goal)))
           (when agent
             (let* ((admiration-potential (cadr goal))
                    (admiration-intensity (max 0 (- admiration-potential
                                                    $admiration-threshold))))
               (setf $admiration-threshold
                     (max 0 (- $admiration-threshold admiration-potential)))
               (setf $arousal (min $max-arousal
                                   (+ $arousal admiration-intensity)))
               (when (> admiration-potential 0)
                     (setf $admiration-affects
                           (call insert-affect self
                                 (make admiration-affect
                                       standard 'help-my-goals-succeed
                                       agent agent
                                       action nil
                                       potential admiration-potential
                                       max-intensity admiration-intensity
                                       intensity admiration-intensity)
                                 $admiration-affects))
                   (push (make admiration-affect
                               standard 'help-my-goals-succeed
                               agent agent
                               action nil
                               potential admiration-potential
                               max-intensity admiration-intensity
                               intensity admiration-intensity)
                         $current-admiration-affects)))))))
     (call update-admiration-intensity self))

    ; COMPUTE-REPROACH
    ; determine reproach caused by passive goals
    (compute-reproach ()
     (setf $current-reproach-affects nil)
     (dolist (goal $negative-passive-goals)
       (when (not (eq (matcher (caddr goal)) '*fail*))
         (let ((agent (call responsible-fail self goal)))
           (when agent
             (let* ((reproach-potential (cadr goal))
                    (reproach-intensity (max 0 (- reproach-potential
                                                  $reproach-threshold))))
               (setf $reproach-threshold
                     (max 0 (- $reproach-threshold reproach-potential)))
               (setf $arousal (min $max-arousal
                                   (+ $arousal reproach-intensity)))
               (when (> reproach-potential 0)
                     (setf $reproach-affects
                           (call insert-affect self
                                 (make reproach-affect
                                       standard 'dont-cause-my-goals-to-fail
                                       agent agent
                                       action nil
                                       potential reproach-potential
                                       max-intensity reproach-intensity
                                       intensity reproach-intensity)
                                 $reproach-affects))
                   (push (make reproach-affect
                               standard 'dont-cause-my-goals-to-fail
                               agent agent
                               action nil
                               potential reproach-potential
                               max-intensity reproach-intensity
                               intensity reproach-intensity)
                         $current-reproach-affects)))))))
     (call update-reproach-intensity self))

    (update-state ()
      ; part of hack to hiss when touched and ignoring/aggrseeive
      (setf $has-hissed nil)

      ; specific update rules for the cat
      (let ((caller (matcher (test (and (know-of-in-ism (a player ?actor))
                                        (or (did-hear (a cat-call))
                                            (did-see (an act
                                                         verb '*call
                                                         object $$me))))))))
        (when (not (eq '*fail* caller))
              (call update-relationship self
                    (cadr (assoc 'actor caller)) 'like
                    (min 2 (+ 1 (call relationship self
                                      (cadr (assoc 'actor caller)) 'like))))
              (call update-relationship self
                    (cadr (assoc 'actor caller)) 'dislike
                    (max -2 (- (call relationship self
                                     (cadr (assoc 'actor caller))
                                     'dislike)
                               1)))))
      (let ((anger-causer
             (matcher (test (and (match (call feel-anger self)
                                        (list-containing
                                         (list-of ?actor ?int)))
                                 (> ?int 5))))))
        (when (not (eq anger-causer '*fail*))
              (call update-relationship self
                    (cadr (assoc 'actor anger-causer)) 'dislike
                    (min 2 (+ 1 (call relationship self
                                      (cadr (assoc 'actor anger-causer))
                                      'dislike))))
              (call update-relationship self
                    (cadr (assoc 'actor anger-causer)) 'like
                    (max -2 (- (call relationship self
```

```
                                     (cadr (assoc 'actor anger-causer))
                                     'like) 1)))))
      (let ((gratitude-causer
             (matcher (test (and (match (call feel-gratitude self)
                                        (list-containing
                                         (list-of ?actor ?int)))
                                 (> ?int 5))))))
        (when (not (eq gratitude-causer '*fail*))
              (call update-relationship self
                    (cadr (assoc 'actor gratitude-causer)) 'dislike
                    (min 2 (+ 1 (call relationship self
                                (cadr (assoc 'actor gratitude-causer))
                                'dislike))))
              (call update-relationship self
                    (cadr (assoc 'actor gratitude-causer)) 'like
                    (max -2 (- (call relationship self
                                     (cadr (assoc 'actor gratitude-causer))
                                     'like) 1)))))
      (call-next)
      (when *debug-emotions* (call debug-emotions self)))

    (debug-emotions ()
      (format t "~&Features:~A~&Fear:~A  Anger:~A~&Happy:~A  Sad:~A  Like:~A  Dislike:~A~&"
              (slot features self)
              (call feel-fear self)
              (call feel-anger self)
              (call feel-happy self)
              (call feel-sad self)
              (call feel-love self)
              (call feel-hate self)))
   )
)
```