The Ego Machine: a Computational Phenomenal Model

Eric Schulte

May 9, 2011

1 Introduction

The *self* is fundamental to human experience. Every conscious moment necessarily includes both the perceived external world and an internal embodied perceiver. Despite this pervasiveness, philosophical and neuro-scientific models of the self are wide ranging and grossly underdeveloped showing more influence from folk wisdom than scientific experimentation.

Recent neuro-scientific advances especially in brain imaging technologies are making possible for the first time a truly scientific and objective investigation of the phenomenon of the self. These changes are leading to a "disenchantment of the self" [2] analogous to Weber's "disenchantment of the world" which followed the enlightenment [1]. The functional models of consciousness resulting from these advances, especially Thomas Metzinger's phenomenal self model (PSM) promise to provide a coherent scientifically verifiable description of consciousness and self-hood.

Surprisingly, many of the functional roles of the PSM are analogous to practical concerns in the construction of modern computational systems. These include the coordination of heterogeneous parallel components, managing system-wide global state and the offloading of computationally expensive functions into specialized sub-components.

This work proposes to further investigate Metzinger's PSM through computational experimentation. A computational implementation of a PSM which aligns PSM functionality with real computational problems such as those mentioned above can be used to test the logical consistency and completeness of Metzinger's PSM, to gain experiential knowledge of the workings of a PSM, and to perform experimentation on a PSM which would be impossible, impractical or immoral to perform on a living system.

In this proposal Metzinger's PSM is described with a special focus on those aspects relevant to the proposed computational implementation (Section 2). The computational implementation is described and a detailed work plan for its construction is presented (Section 3). Metzinger addresses the morality of implementing artificial consciousness [2], expressing deep concern that such an act would be unethical. This concern is addressed in section 4 in which we present a moral justification for this work. The proposal concludes with a discussion of the implications of such a computational model on our understanding of both consciousness and computational systems (Section 5).

2 Background

Thomas Metzinger is a materialist and an analytical philosopher of the mind who is working to build a functional model of conscious experience. Metzinger works closely with neuro-scientists and takes care to ensure his work incorporates data from basic scientific investigation as well as often ignored "edge-cases" of consciousness such as out-of-body experiences, deep meditation and drug induced hallucination.

The PSM model of consciousness proposed by Metzinger is a biological tool which is a product of natural selection and is generated by the brain to perform a number of functions which increase our ability to act in the world. Under this model the perception of *self-hood* is a trick we play on ourselves in order to better communicate, synthesize phenomenal experience and coordinate action. This model makes claims about the logical structure of the PSM as well as the physical brain states through which it arises.

The remainder of this section borrows heavily from Metzingers book *The Ego Tunnel: The Science of the Mind and the Myth of the Self* [2].

2.1 Historical roots of consciousness

In different historical eras the term *consciousness* has taken on varied meanings, and has been used to refer to both a moral role such as a guiding conscious, and a phenomenal role such the *experience* of consciousness. The following four aspects of consciousness cover many of the essential properties of consciousness.

group knowledge From the latin roots *cum* meaning "together" and *scire* meaning "to know" consciousness has been seen as a form of group knowledge which is only possible as part of a community.

moral Such communal knowledge is most often used in the context of moral knowledge, in this sense *conscious* is seen as an internal perfect moral observer which has access to our PSM and can apply normative judgment to our actions. Under such a conception the fundamental role or our conscious lives may have been moral.

certainty With Descartes the term takes on a sense of immediacy which separates first-person conscious experience from all other forms of knowledge. Conscious experience becomes the only knowledge of which we can be certain, and to which we have direct access.

unity An essential aspect of all conceptions of consciousness seems to be that of unity, that is a single indivisible conscious experience of a single self existing in a single world.

2.2 Humanity and self-consciousness

It seems plausible that many animals including all mammals and most birds implement some form of conscious experience. Metzinger suggests that part of what may make humans exceptional among animals is the inclusion of ourselves as reasoning agents into our PSM.

> by representing the process of representation itself, we can catch ourselves – as Antonio Damasio would call it – in the act of knowing.

This in turn enables the communication and cooperation which is the foundation of human culture and "allow[ed] biological evolution to explode into cultural evolution".

The computational implementation of consciousness proposed below will be maximally simple and will not rise to the level of "self-consciousness". The representation of the self in the PSM not contain any state, thus, rather than the complex cultural consciousness of humans the computational consciousness described below will be more akin to the simplest animal consciousness – a direct consciousness of an environment.

2.3 The PSM

Metzinger introduces the concept of a *virtual organ* which is a functional unit implemented by the body. Examples given of virtual organs include the immune response (which is only present during an infection) and emotional states such as fear or courage which can be implemented by the body to aid in dealing with specific situations.

The PSM is an instance of such a *virtual organ* which is produced by the brain whenever a subject reports being conscious, and is absent during unconscious states such as deep sleep or during sedation.

The PSM serves to synthesize the many outputs of disparate functional modules of the brain into a single phenomenological model which is of much lower dimension than either the actual physical world, or the raw sensory input into the brain. The contents of this phenomenological model are then made (through the PSM) globally available across the entire brain. Phenomenological experience can be seen as a biological data type through which information is stored and shared in the brain.

This conception is compatible with the *dynamic core* concept of consciousness [4] in which the *dynamic core* is the neural correlate of consciousness, and is defined as a group of neurons which are both:

integrated meaning that the group of neurons achieves sustained high levels of correlated firing over hundreds of milliseconds (the time scale of conscious experience).

differentiated as indicated both by high values of complexity (where “complexity” is a measure of the mutual information between all bi-partitions of the group of neurons) and by spatial distribution through the brain.

2.3.1 Properties of the PSM

The remainder of this section will attend to specific properties of the PSM which are relevant to the proposed computational implementation of consciousness. In the next section each of these properties will be revisited in the context of the proposed computational PSM.

coherence and re-entry In the human visual system, the result of high-level processing (e.g., I am looking at a chair) can *re-enter* lower levels of visual processing, informing the formation of lower level features such as simple edges.

In the same way that the higher level representations of perceived objects may re-enter the lower levels of perception, so too consciousness can be seen as the continuous large-scale re-entry of our prior knowledge to our current situation. Through the global availability of the contents of the PSM across all aspects of the brain, the PSM can influence the lower level processes which provide the raw material of consciousness and can directly influence the subsequent contents of consciousness, leading to continuity through time.

temporal concurrency Many studies have shown that the synchronous firing of groups of neurons may be the relevant to what neuronal activity gains access to consciousness. When disparate groups of neurons fire synchronously it is more likely that the contents of their output will become incorporated into the PSM and become conscious. Such synchrony seems to also be relevant to defined atomic *elements* of conscious experience. To adapt an example from Metzinger [2] – when you read this paper you are aware of both the visual appearance of the written text, as well as the tactile feel of the paper (or possibly your computer keyboard). As these visual and tactile neurons fire in synchrony, the contents of their output (namely this paper) are incorporated into the active PSM as a single entity.

serial global control The brain is largely a heterogeneous collection of processing centers specializing in different tasks (e.g., visual processing, memory). These centers run concurrently and in parallel, however conscious experience appears to us as a serial linear series of experiences. The PSM is responsible for organizing the information from these many disparate centers in the brain into a single series of experienced moments.

global state The contents of consciousness are globally available throughout the brain. When a sensation is consciously attended, its contents become globally available across the brain. This is thought to enable the flexible creative responses which are only possible through conscious thought.

general processing As just mentioned novel situations often require conscious attention to form appropriate reactions. This is because of the greater flexibility of conscious states which may draw upon and coordinate disparate elements of the brain. Once a task becomes route (e.g., riding a bike) its performance may no longer require conscious attention. In experiments, such novels tasks were characterized by global activation of the brain, however when subjects report learning a task to the point where its performance does not require conscious attention, the pattern of activation is limited to a small portion of the brain.

3 Preliminary Implementation & Work Plan

This computational implementation of a PSM will seek to align real computational problems with the functional aspects of the PSM model of consciousness. Through applying the PSM in such a practical manner, new insight may be gained into the PSM model. Additionally the functionality of the PSM may provide effective solutions to existing computational problems such as global control and coordination between disparate parts, and generalized processing in the face of novel stimulus.

Our computational environment will consists of a *world* composed of a frictionless 2D plane containing both an *agent* which can cause itself to move, and a number of balls which move with momentum. The goal of the agent will be to avoid proximity to the balls – contact not being possible as both the agent and the balls are points with no extension.

Both the world and the content of the PSM will be visualizable, specifically the PSM will consist of a circle of scalar values (indicating the nearness of balls to the agent in each angular direction), thus the visualization of the PSM through time could in fact be viewed as an “ego tunnel”.

In addition to the PSM, the agent will consist of a number of heterogeneous concurrent functional units intended to correspond to functional subsets of the brain. These units will both determine the content of the PSM and receive information from the PSM.

3.1 Artificial World

The agent is embedded in a simple frictionless finite circular 2D physical world. In order to keep the agent and all objects within the borders of the world, at a certain distance from the center of the world object begin to accelerate back towards the world center. A snapshot of a world with six balls and no agent is show in Figure 1, and the corresponding video is available at http://cs.unm.edu/∼eschulte/notes/ego-machine/empty-world.mp4.

3.2 Agent

The external *body* of the agent in the world consists of a single point which, like the balls, has position and velocity. In addition the body of the agent includes a single *eye* which can perceive balls in a cone expanding from the agent in the direction of the eye and a set of short *whiskers* which can detect balls within a short range of the agent. Additionally the body of the agent has two actuators, the first of which can affect the velocity of the agent, and the second of which can rotate the agent effectively changing the direction of the eye.

The computational *mind* of the agent is composed of a single PSM which coordinates many concurrently executing heterogeneous modules, each of which implements a specific function (Figure 2).

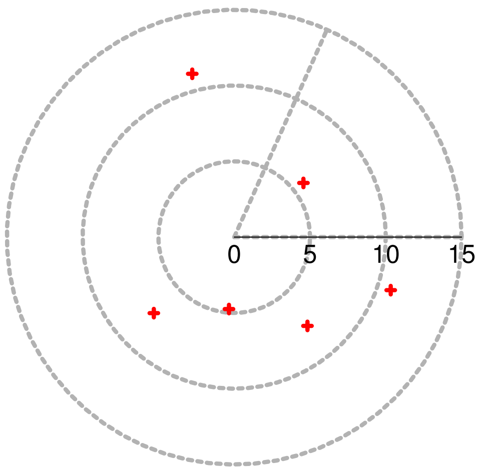

Figure 1: A world containing 6 balls and no agent.

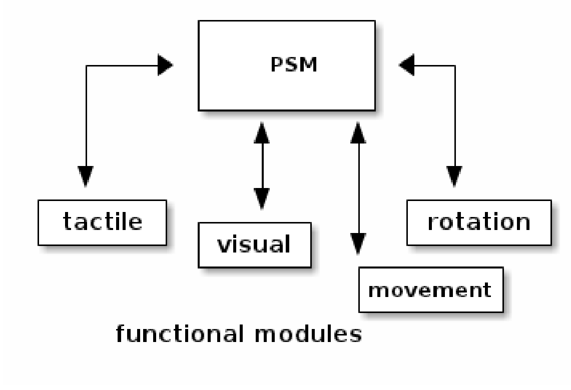

Figure 2: Architectural overview of the Agent

3.3 Modules

Modules are analogous to functional sections of the brain. Modules accept input from the PSM and possibly from external sensors and send output to the PSM and possibly to external actuators. All inter-module coordination occurs exclusively through the contents of the PSM. A sensor module with connections is shown in Figure 3.

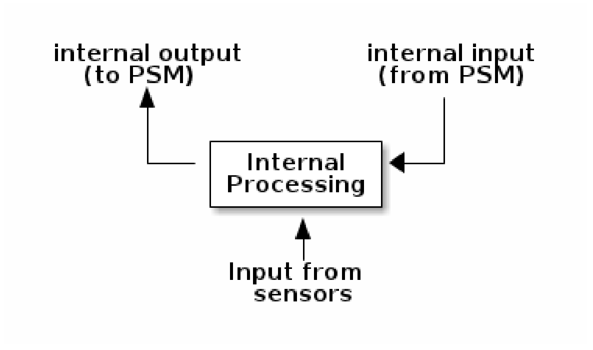

Figure 3: Module with sensor and PSM connections.

A minimal set of modules for successful avoidance behavior are presented below. Given the extremely loose coupling the addition of new modules in either an engineered or an automated evolutionary manner would be trivial.

visual Receives sensory input from a single eye in the form of a set of polar coordinates. Each coordinate corresponds to a ray extending from the agent inside the cone of vision, the angle indicates the position inside of the code and the length indicates the length to the nearest ball in that direction (or 0 if no balls are intersected by that ray).

This information along with the contents of the PSM is interpreted by the visual module and the results are sent to the PSM in a manner which may be easily incorporated into the phenomenological data of the agent (namely the circle of scalar values).

tactile Can detect balls in close proximity to the agent. Like the visual system, the tactile system also has access to the contents of the PSM and it reduces its combined input into a form which can be incorporated into the phenomenological model of the agent.

rotation Takes the phenomenological content of the PSM as input and rotates the agent in response.

movement Takes the contents of the PSM as input, and as output can change the velocity of the agent.

3.4 PSM

The PSM (Figure 4) shares information across computational modules, and provides a single global serial ordering to the phenomenological states of the agent analogous to a *stream of consciousness*.

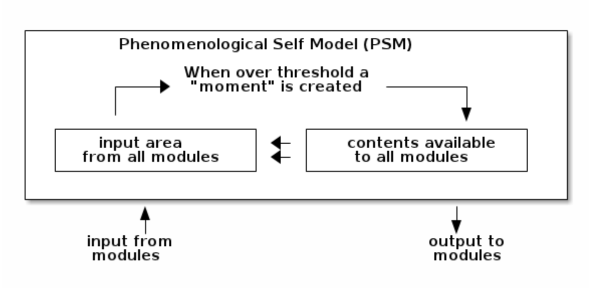
Phenomenological Self Model (PSM)
When over threshold a "moment" is created
input area from all modules
contents available to all modules
input from modules
output to modules

Figure 4: Phenomenological Self Model

The PSM is composed of an *input* area into which the output of the various modules is collected and an area which holds the current contents of the PSM. Both of these contain the same data structure, namely a circle of polar coordinates which roughly correspond to the *ball-proximity* in each direction away from the agent. In the absence of input the contents of both areas degrade with time.

As the functional modules run they may add their output to the input of the PSM. When sufficient information has accumulated in the input area, the PSM copies the input area into the *contents* area. This generates a conscious *moment* for the agent. When the contents area is not empty the agent can be said to be conscious, when the contents area is empty the agent can be considered to be in a state analogous to sleep.

The phenomenological *contents* of the PSM are available to all functional modules, and it is in response to these contents that the agent moves.

An agent in a world with 6 balls is shown in Figure 5. The green lines extending from the agent display the contents of the agents PSM indicating that the agent is conscious of the presence of balls in three directions. Video of the run form which this snapshot was taken is available at http://cs.unm.edu/∼eschulte/notes/ego-machine/full-world.mp4.

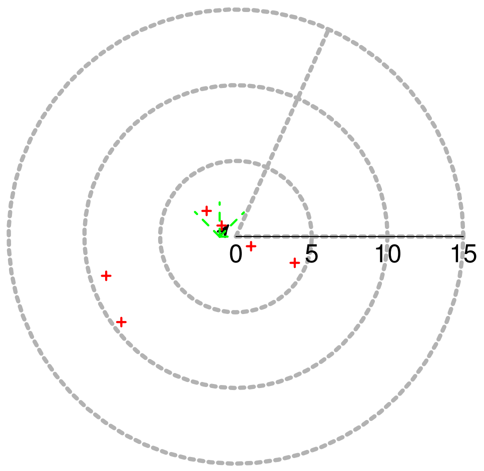

Figure 5: A world containing 6 balls and an agent with the PSM shown in green.

3.5 Preliminary Implementation & Shortcomings

The preliminary implementation [3] used to generate the results shown in Figures 1 and 5 and whose source code is available in the appendix is merely a proof of concept. It remains insufficient to investigate the PSM model in the following important ways.

neural networks All of the modules in the current implementation are hand-coded to exhibit simple *reasonable* behavior. A more biologically plausible neural network (NN) module implementation would allow evolution and adaptation of the modules, and would introduce generality, enabling many of the following points of refinement.

coherence and re-entry The initial model does not implement any re-entry of the contents of the PSM into the sensing modules. The use of hierarchical NN sensory modules would allow the contents of the PSM to be naturally incorporated back into sensory input.

temporal concurrency In the current implementation all modules execute in lock-step once per world-second. This does not allow for sympathetic firing of modules sensing related aspects of the external world. A NN implementation of sensing models which explicitly includes adjustable firing rates would allow modules to *synchronize* when sensing related external phenomena.

serial global control In the current implementation all modules execute in a single thread. A trivial extension separating each module into its own thread would allow testing of the organizational facilities of the PSM.

global state The PSM does provide global state in the current implementation, however due to the small number of modules and the serial module evaluation interesting race conditions do not arise. An extended model would be required to test the coordination facilities of the PSM.

general processing Every module in the current implementation is assigned an explicit functional role, and there is no room for the introduction of new modules or recruitment of modules to new tasks. Because of this the process of learned tasks dropping out of consciousness is not testable. A much more general system of flexible modules would be required for such experimentation.

3.6 Work Plan

The following work plan addresses the shortcoming of the current implementation, expanding it into a genuine computational platform for experimentation.

1. Parallelization of the module execution.

2. Replacement of modules with biologically plausible neural network architectures.

3. Uniform module interface allowing modules to change roles.

4. Embedding of agent system into a Genetic Algorithm framework allowing the process of natural selection to be used to select new functionality.

4 Ethics Review

Metzinger explicitly addresses the morality of implementing an artificial *consciousness* with an integrated and dynamical world model which organizing its information flow in such a way as to result in an *experiential now*. To Metzinger such machines only become "objects of moral concern" when they include a "transparent internal image of itself" into their phenomenal reality [2]. This is explicitly not the case in the proposed implementation performed as part of this work, as the content of the PSM is restricted to a circle of external scalars and includes no internal state.

It is interesting to note however that by Metzinger's description many modern operating systems may be considered both conscious and self conscious and thus "potentially able to *suffer*".

5 Discussion

Recent developments in both applied computer science and in experimental neuro-science and philosophy of the mind have decreased the distance between the subject mater of both fields. The topics of concern to computer scientists may already overlap the areas of study of consciousness researchers. If this is not the case already it will certainly be the case in the near future.

Through increasing the awareness of overlap between these fields, an explicit computational investigation into functional models of consciousness will help each fields to benefit from the other's work, and will highlight potential areas of moral concern before harm is done.

References

[1] B. B. Koshul. *The postmodern significance of Max Weber's legacy*. Palgrave Macmillan, 2005.

[2] T. Metzinger. *The Ego Tunnel: The Science of the Mind and the Myth of the Self*. Basic Books, 1 edition, March 2009.

[3] E. Schulte. Ego machine, May 2011. http://gitweb.adaptive.cs.unm.edu/ego-machine.git.

[4] G. Tononi and G. M. Edelman. Consciousness and complexity. *Science*, 282(5395):1846–1851, dec 1998.

```lisp
;;; ego-machine.lisp --- A Computational Phenomenal Self Model

;; Adapted from the description in Thomas Metzinger's
;; _The Ego Tunnel_ Basic Books, 1 edition, March 2009

;; Copyright (C) 2011  Eric Schulte

;;; License:

;; This program is free software; you can redistribute it and/or modify
;; it under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 3, or (at your option)
;; any later version.
;;
;; This program is distributed in the hope that it will be useful,
;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;; GNU General Public License for more details.
;;
;; You should have received a copy of the GNU General Public License
;; along with GNU Emacs; see the file COPYING.  If not, write to the
;; Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
;; Boston, MA 02110-1301, USA.

;;; Code:
(defpackage #:ego-machine)
(in-package :ego-machine)

;;; world
(defvar *field-radius* 10)
(defvar *max-init-vel* 0.1)
(defvar *field-acel* 0.001)
(defconstant tau (* 2 Pi) "tau=2Pi is a more natural constant to use than Pi.")

(defclass polar ()
  ((r :initarg :r :accessor r)
   (theta :initarg :theta :accessor theta)))

(defmethod print-object ((polar polar) stream)
  (format stream "#polar(~a,~a)" (r polar) (theta polar)))

(defun rand-polar (&key (r-max *field-radius*))
  (make-instance 'polar
    :r (random (float r-max))
    :theta (random tau)))

(defmethod to-rect ((polar polar))
  (with-slots (r theta) polar
    (make-instance 'rect
      :x (* r (cos theta))
      :y (* r (sin theta)))))

(defmethod add ((a polar) (b polar))
  (to-polar (add (to-rect a) (to-rect b))))

(defmethod sub ((a polar) (b polar))
  (to-polar (sub (to-rect a) (to-rect b))))

(defclass rect ()
  ((x :initarg :x :accessor x)
   (y :initarg :y :accessor y)))

(defmethod print-object ((rect rect) stream)
  (format stream "#rect(~a,~a)" (x rect) (y rect)))

(defmethod to-polar ((rect rect))
  (with-slots (x y) rect
    (make-instance 'polar
      :r (sqrt (+ (* x x) (* y y)))
      :theta (handler-case (atan y x) (error nil 0)))))

(defmethod add ((a rect) (b rect))
  (make-instance 'rect
    :x (+ (x a) (x b))
    :y (+ (y a) (y b))))

(defmethod sub ((a rect) (b rect))
  (make-instance 'rect
    :x (- (x a) (x b))
    :y (- (y a) (y b))))

(defclass ball ()
  ((place :initarg :place :accessor place)
   (vel   :initarg :vel   :accessor vel)))

(defmethod print-object ((ball ball) stream)
  (format stream "#ball(~a->~a)" (place ball) (vel ball)))

(defun rand-ball ()
  (make-instance 'ball
    :place (rand-polar)
    :vel (rand-polar :r-max *max-init-vel*)))

(defmethod move ((ball ball))
  ;; update the ball's velocity
  (when (> (r (place ball)) *field-radius*)
    (setf (vel ball)
          (add (vel ball)
               (with-slots (r theta) (place ball)
                 (make-instance 'polar
                  :r (* *field-acel* (- (r (place ball)) *field-radius*))
                  :theta (+ (theta (place ball)) Pi))))))
  ;; update the ball's position
  (setf (place ball) (add (place ball) (vel ball))))

;;; agent
(defvar *cone-width* (/ tau 8))
(defvar *whisker-length* 2)
(defvar *agent-acel* 0.001)
(defvar *agent-friction* 0.9)
(defvar *agent-rotate-speed* 0.025)
(defvar *psm-degrees* 8)
(defvar *agent-arrow-length* 1)
(defvar *psm-threshold* 2)
(defvar *psm-fade* 0.75)
(defvar *num-balls* 6)
(defvar *balls* nil)

(defclass agent (ball)
  ((angle   :initarg :angle   :accessor angle)
   (psm     :initarg :psm     :accessor psm)
   (modules :initarg :modules :accessor modules)))

(defclass psm ()
  ((input  :initarg :input  :accessor input)
   (phenom :initarg :phenom :accessor phenom)))

(defmethod add ((psm psm) (polar polar))
  (with-slots (input) psm
    (let ((theta (mod (theta polar) tau)))
      (incf (r (nth (mod (round (/ theta (/ tau 8))) 8) input))))))

(defmethod add ((psm psm) nothing)
  (declare (ignorable psm nothing)))

(defun in-cone (tip from to)
  "Return the offsets of all balls in the cone."
  (remove nil
          (mapcar (lambda (offset)
                    (when (and (< from (theta offset)) (< (theta offset) to))
                      offset))
                  (mapcar (lambda (ball) (sub tip (place ball))) *balls*))))

(defun visual (agent)
  (with-slots (place angle psm) agent
    (let ((width (/ *cone-width* 2)))
      (add psm (first (sort (in-cone place (- angle width) (+ angle width))
                            #'< :key #'r))))))

(defun tactile (agent)
  (dolist (offset (mapcar (lambda (ball) (sub (place agent) (place ball))) *balls*))
    (when (< (r offset) *whisker-length*)
      (add (psm agent) (make-instance 'polar
                         :r 1
                         :theta (+ (theta offset) Pi))))))

(defun movement (agent)
  (when (phenom (psm agent))
    (let (max)
      ;; set max to greatest value in phenom
      (dolist (polar (phenom (psm agent)))
        (when (or (not max) (> (r polar) (r max)))
          (setf max polar)))
      (setf (vel agent)
            (add (vel agent)
                 (make-instance 'polar
                   :r (* *agent-acel* (r max))
                   :theta (+ (theta max) Pi)))))))

(defun rotation (agent)
  (setf (angle agent) (mod (+ *agent-rotate-speed* (angle agent)) tau)))

(defmethod run ((agent agent))
  ;; slow down
  (when (> (r (vel agent)) 0)
    (setf (r (vel agent)) (* *agent-friction* (r (vel agent)))))
  ;; update the psm
  (flet ((fade (obj)
           (setf obj (mapcar (lambda (el) (setf (r el) (* (r el) *psm-fade*)) el)
                             obj))))
    (fade (phenom (psm agent)))
    (when (> (apply #'+ (mapcar #'r (input (psm agent)))) *psm-threshold*)
```

```lisp
      (setf (phenom (psm agent)) (copy-seq (input (psm agent)))))
    (fade (input (psm agent))))
  ;; run all modules
  (dolist (module (modules agent))
    (funcall module agent)))

;;; Run with gnuplot
(defvar *steps* 250)

(defvar *agent*
  (make-instance ’agent
    ;; start in the middle
    :place (make-instance ’polar :r 0 :theta 0)
    :vel (make-instance ’polar :r 0 :theta 0)
    :angle 0
    :psm (let ((input (loop for theta to tau by (/ tau *psm-degrees*)
                         collect (make-instance ’polar :r 0 :theta theta))))
           (make-instance ’psm
             :input #+ccl (butlast input) #+clisp input
             :phenom nil))
    :modules (list #’visual #’tactile #’movement #’rotation)))

(setf *balls* (loop for n from 1 to *num-balls* collect (rand-ball)))

(defun gnuplot (step &key (stream t) (to-file nil))
  (format stream "set title \"~d:~6f\"~%unset arrow~%"
          step
          (apply #’+ (mapcar #’r (input (psm *agent*)))))
  ;; set output to a file
  (when to-file
    (format stream (concatenate ’string
                     "~&set output \"~a/~d.png\"~%"
                     "set term png transparent large~%") to-file step))
  ;; agent
  (when *agent*
    (let ((rect-tip (to-rect (add (place *agent*) (make-instance ’polar
                                                    :r *agent-arrow-length*
                                                    :theta (angle *agent*)))))
          (rect-agent (to-rect (place *agent*))))
      (format stream "~&set arrow from ~f,~f to ~f,~f~%"
              (x rect-agent) (y rect-agent)
              (x rect-tip)   (y rect-tip))))
  ;; psm
  (when (and *agent* (psm *agent*))
    (dolist (dir (phenom (psm *agent*)))
      (let ((beg (to-rect
                  (add (place *agent*)
                       (make-instance ’polar :r 0 :theta (theta dir)))))
            (end (to-rect
                  (add (place *agent*)
                       (make-instance ’polar :r (r dir) :theta (theta dir))))))
        (format stream "~&set arrow from ~f,~f to ~f,~f nohead ls 2~%"
                (x beg) (y beg) (x end) (y end)))))
  ;; balls
  (format stream "~&plot ’-’ notitle~%~T~{~a~%~T~}e~%"
          (mapcar (lambda (place) (format nil "~f~T~f" (theta place) (r place)))
                  (mapcar #’place *balls*))))

(defun run-w/o-gnuplot ()
  (do ((step 0 (1+ step)))
      ((= step *steps*))
    (run *agent*)
    (mapcar #’move (cons *agent* *balls*))))

(defun run-w/gnuplot (&key (to-file nil) (stream t))
  ;; setup gnuplot
  (format stream (concatenate ’string
                   "~&set polar~%"
                   "set grid polar 20~%"
                   "unset border~%"
                   "unset param~%"
                   "unset xtics~%"
                   "unset ytics~%"
                   "set size square~%"
                   "set xrange [~f:~f]~%"
                   "set yrange [~f:~f]~%"
                   "set trange [-p1:p1]~%"
                   "set rrange [~f:~f]~%")
          (- 0 (* 1.5 *field-radius*)) (* 1.5 *field-radius*)
          (- 0 (* 1.5 *field-radius*)) (* 1.5 *field-radius*)
          0                             (* 1.5 *field-radius*))
  (do ((step 0 (1+ step)))
      ((= step *steps*))
    (run *agent*)
    (mapcar #’move (cons *agent* *balls*))
    (gnuplot step :to-file to-file :stream stream))) ; plot with gnuplot

;;; ego-machine.lisp ends here
```