Cognitive Systems model

Maik Schünemann

8. Juni 2014

Inhaltsverzeichnis

---

1 Introduction

this program evolved from a series of excercises from the course Cognitive Systems. It models how human perceive and group objects in a picture.

It demonstrates the concepts only ... the tasks it has to solve are counting objects in a picture which match certain criteria in their shapes and color.

2 Internal representations

2.1 Picture

A picture is stored as a 2-dimensional array of cells. Each cell is just a a map with the keys :color and :shape and the values are just keywords. A example picture can be represented like this:

```
(ns cosy.input
  (:require [quil.core :refer :all]
    [clojure.test.check :as tc]
    [clojure.test.check.generators :as gen]
    [clojure.test.check.properties :as prop]))

(defn setup []
  (smooth)
  (frame-rate 24)
  (background 200))
```

```
(def example-array
  [[{} {} {} {} [:black :circle] {} {} {}]
   [{} {} {} {} {} {} [:blue :circle] {}]
   [[:yellow :square] [:blue :square] {} {} {} {} {} {}]
   [{} [:red :circle] [:blue :circle] {} {} {} {} {}]
   [{} [:black :circls] {} {} [:black :circle] {} {} [:blue :circle]]
   [[:blue :circle]{}{}[:red :triangle]{}{}[:red :circle][:blue :quare]]
   [{} {} [:blue :circle] {} [:yellow :square] {} {} {}]
   [[:red :circle] {} {} {} {} {} {} {}]])

(alter-var-root #'*out* (constantly *out*))

(def cellwidth 50)
(def cellheight 50)

(def rgb-color {:red [255 0 0]
:blue [0 0 255]
:green [0 255 0]
:yellow [255 255 0]
:black [0 0 0]})

(defn get-rgb-color [cell]
  (get rgb-color (first cell) [255 255 255]))

(def draw-funcs
  {:square #(rect (+ 5 %1) (+ 5 %2)
  (- cellwidth 10)
  (- cellheight 10))
   :circle #(ellipse (+ 25 %1) (+ 25 %2)
     (- cellwidth 10)
     (- cellheight 10))
   :triangle #(triangle (+ %1 25) %2
(+ %1 5) (+ %2 40)
(+ %1 45) (+ %2 40)) })

(defn draw-shape [shape x y]
  ((get draw-funcs shape (constantly nil)) x y))

(defn visualize-array [array]
  ;;each cell will be 25x25 pixel big
  (doseq[[i y] (map-indexed
vector (range 0 (* cellwidth (count array))
      cellwidth))
 [j x] (map-indexed
vector (range 0 (* cellheight (count (first array)))
      cellheight))]
    (fill (apply color (get-rgb-color (get-in array [i j]))))
    (draw-shape (second (get-in array [i j])) x y)))

(defn draw []
```

```
  (rect 0 0 25 25)
  (rect 25 0 25 25))

(defsketch example
  :title "cosy again"
  :setup setup
  :draw (fn [] (visualize-array example-array))
  :size [400 400])



(defn gen-array []
  (let [x (gen/choose 5 10)
y (gen/choose 5 10)
s (gen/elements [:circle :square :triangle])
c (gen/elements [:black :blue :red :yellow :green])]
    (as-> (gen/tuple x y) x)))
```

3 Components

The system needs the following components to model the human behaviour

- perception - from a picture to informations contained in it
- Memory - The human equivalent of storage/cache
- Processing unit - the human cpu

3.1 Perception

The perception is the most important cognitive component when the goal is to model the human behaviour. Scanning the shown picture top to botton left to right is trivial to implement and efficient for a computer, but a human does not perceive the world in this way. A Human automatically filters much information that is not needed for the task at hand and therefore does **not** look at each cell in order to count objects.

3.1.1 How does a Human perceive and process a visual input to count objects in it

Take this picture for example: A sample visual stimuli When looking at the picture we instantly see where the colors are, so when the task is to count the red circles we are not even concerned about the top right area of the picture. Shapes are a little harder to recognize. How do we do this?

- We cannot focus on the whole picture all at once but only see sharply at the center of our viewpoint. In the rest of our view we can't see details but we can recognize where what colors are on the picture.
- afterwards, we know where to focus and then move our eyes so that we can recognize the details in the areas.
- Even the focus-view does not look sequentially at each cell of the picture but can scan a whole (presumably circle but simplified to a rectangular area for this project) area at once
- We perform thus the minimal eye movement in order to focus at each interesting area one time and are so filtering a large part of the picture

3.1.2 Implementation

1. Simplifications/Assumptions We are going to assume that our system only has to process images that aren't too big so that the whole picture fits in the peripheral view field and we can quickly perceive where certain colors are. With focus view, we are able to process a small square area of the image at once (say 4x4 cells)
2. Peripheral view - recognize boundaries for regions of color