Cognitive Systems Excercise 3

Maik Schünemann

10. Juni 2014

Inhaltsverzeichnis

1 Excercise 1

1.1 Concept of the extension

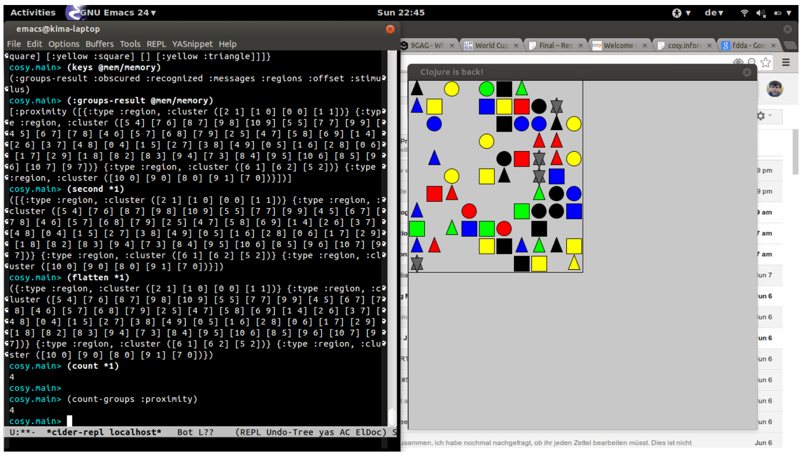
1.1.1 Additional Components

We added group recognizer components to our cognitive architecture, one for each group of proximity. They extend the counting/recognizing visual routines - functionality of excercise 2.

1.1.2 Interplay between the components

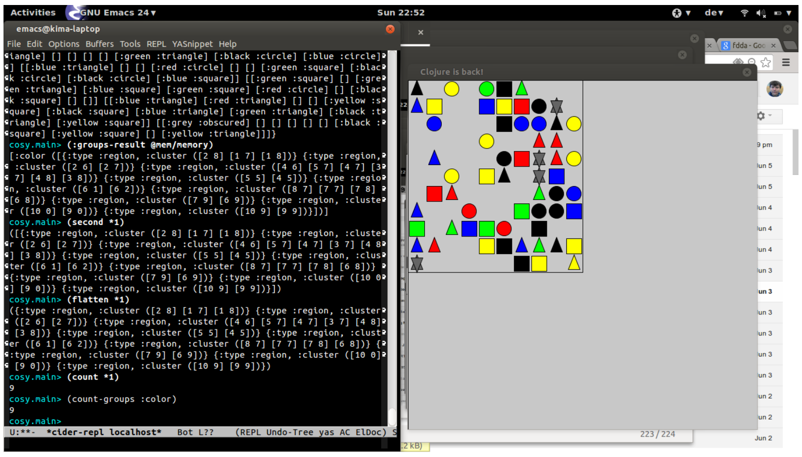
To model human behaviour, we only concentrate on the filled regions of the visual stimulus. We extended the peripheral view component to quickly scan the picture to find the regions of the stimuli that are filled. On this regions we invoke the actual algorithm to count the groups (implemented in the components above). If we have found a non-empty cell our system looks at all adjacent cells if they also belong to the group. It does this until it has found the whole group the cell belongs to. It then continues looking for other groups but knows what cells it has looked at before and doesn't revisit them. The components for recognizing groups of proximity, color, shape only behave differently when they decide whether the adjacent cell belongs to the same group or not

1.1.3 Examples:



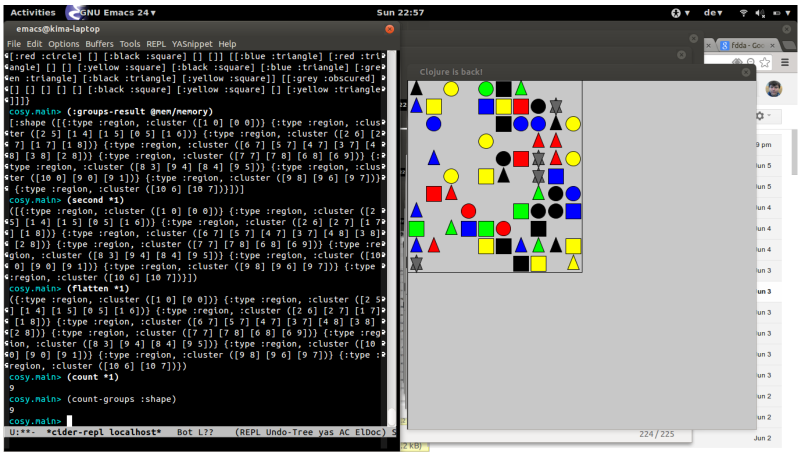
```
quare] [:yellow :square] [] [:yellow :triangle]]]}
cosy.main> (keys @mem/memory)
(:groups-result :obscured :recognized :messages :regions :offset :stimu
lus)
cosy.main> (:groups-result @mem/memory)
[:proximity ([{:type :region, :cluster ([2 1] [1 0] [0 0] [1 1])} {:typ
e :region, :cluster ([5 4] [7 6] [8 7] [9 8] [10 9] [5 5] [7 7] [9 9] [
4 5] [6 7] [7 8] [4 6] [5 7] [6 8] [7 9] [2 5] [4 7] [5 8] [6 9] [1 4]
[2 6] [3 7] [4 8] [0 4] [1 5] [2 7] [3 8] [4 9] [0 5] [1 6] [2 8] [0 6]
 [1 7] [2 9] [1 8] [8 2] [8 3] [9 4] [7 3] [8 4] [9 5] [10 6] [8 5] [9
6] [10 7] [9 7])} {:type :region, :cluster ([6 1] [6 2] [5 2])} {:type
:region, :cluster ([10 0] [9 0] [8 0] [9 1] [7 0])}])]
cosy.main> (second *1)
([{:type :region, :cluster ([2 1] [1 0] [0 0] [1 1])} {:type :region, :
cluster ([5 4] [7 6] [8 7] [9 8] [10 9] [5 5] [7 7] [9 9] [4 5] [6 7] [
7 8] [4 6] [5 7] [6 8] [7 9] [2 5] [4 7] [5 8] [6 9] [1 4] [2 6] [3 7]
[4 8] [0 4] [1 5] [2 7] [3 8] [4 9] [0 5] [1 6] [2 8] [0 6] [1 7] [2 9]
 [1 8] [8 2] [8 3] [9 4] [7 3] [8 4] [9 5] [10 6] [8 5] [9 6] [10 7] [9
 7])} {:type :region, :cluster ([6 1] [6 2] [5 2])} {:type :region, :cl
uster ([10 0] [9 0] [8 0] [9 1] [7 0])}])
cosy.main> (flatten *1)
({:type :region, :cluster ([2 1] [1 0] [0 0] [1 1])} {:type :region, :c
luster ([5 4] [7 6] [8 7] [9 8] [10 9] [5 5] [7 7] [9 9] [4 5] [6 7] [7
 8] [4 6] [5 7] [6 8] [7 9] [2 5] [4 7] [5 8] [6 9] [1 4] [2 6] [3 7] [
4 8] [0 4] [1 5] [2 7] [3 8] [4 9] [0 5] [1 6] [2 8] [0 6] [1 7] [2 9]
[1 8] [8 2] [8 3] [9 4] [7 3] [8 4] [9 5] [10 6] [8 5] [9 6] [10 7] [9
7])} {:type :region, :cluster ([6 1] [6 2] [5 2])} {:type :region, :clu
ster ([10 0] [9 0] [8 0] [9 1] [7 0])})
cosy.main> (count *1)
4
cosy.main>
cosy.main> (count-groups :proximity)
4
cosy.main>
U:**-  *cider-repl localhost*   Bot L??    (REPL Undo-Tree yas AC ElDoc) S
```

Abbildung 1: Recognizing 4 groups of proximity



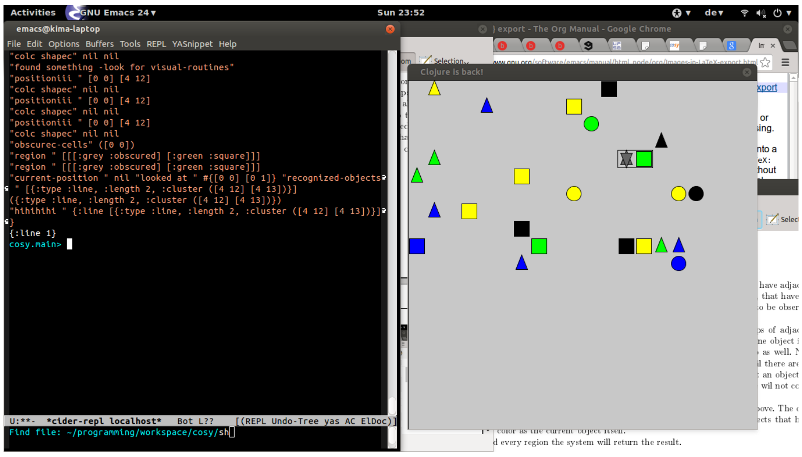
```
iangle] [] [] [] [] [:green :triangle] [:black :circle] [:blue :circle]
] [[:blue :triangle] [] [] [:red :circle] [] [] [:green :square] [:blac
k :circle] [:black :circle] [:blue :square]] [[:green :square] [] [:gre
en :triangle] [:blue :square] [:green :square] [:red :circle] [] [:blac
k :square] [] []] [[:blue :triangle] [:red :triangle] [] [] [:yellow :s
quare] [:black :square] [:blue :triangle] [:green :triangle] [:black :t
riangle] [:yellow :square]] [[:grey :obscured] [] [] [] [] [] [:black :
square] [:yellow :square] [] [:yellow :triangle]]]}
cosy.main> (:groups-result @mem/memory)
[:color ([{:type :region, :cluster ([2 8] [1 7] [1 8])} {:type :region,
 :cluster ([2 6] [2 7])} {:type :region, :cluster ([4 6] [5 7] [4 7] [3
 7] [4 8] [3 8])} {:type :region, :cluster ([5 5] [4 5])} {:type :regio
n, :cluster ([6 1] [6 2])} {:type :region, :cluster ([8 7] [7 7] [7 8]
[6 8])} {:type :region, :cluster ([7 9] [6 9])} {:type :region, :cluste
r ([10 0] [9 0])} {:type :region, :cluster ([10 9] [9 9])}])]
cosy.main> (second *1)
([{:type :region, :cluster ([2 8] [1 7] [1 8])} {:type :region, :cluste
r ([2 6] [2 7])} {:type :region, :cluster ([4 6] [5 7] [4 7] [3 7] [4 8
] [3 8])} {:type :region, :cluster ([5 5] [4 5])} {:type :region, :clus
ter ([6 1] [6 2])} {:type :region, :cluster ([8 7] [7 7] [7 8] [6 8])}
{:type :region, :cluster ([7 9] [6 9])} {:type :region, :cluster ([10 0
] [9 0])} {:type :region, :cluster ([10 9] [9 9])}])
cosy.main> (flatten *1)
({:type :region, :cluster ([2 8] [1 7] [1 8])} {:type :region, :cluster
 ([2 6] [2 7])} {:type :region, :cluster ([4 6] [5 7] [4 7] [3 7] [4 8]
 [3 8])} {:type :region, :cluster ([5 5] [4 5])} {:type :region, :clust
er ([6 1] [6 2])} {:type :region, :cluster ([8 7] [7 7] [7 8] [6 8])} {
:type :region, :cluster ([7 9] [6 9])} {:type :region, :cluster ([10 0]
 [9 0])} {:type :region, :cluster ([10 9] [9 9])})
cosy.main> (count *1)
9
cosy.main> (count-groups :color)
9
cosy.main>
U:**-  *cider-repl localhost*   Bot L??    (REPL Undo-Tree yas AC ElDoc) S
```

Abbildung 2: Recognizing the groups of color



```
[:red :circle] [] [:black :square] [] []] [[:blue :triangle] [:red :tri
angle] [] [] [:yellow :square] [:black :square] [:blue :triangle] [:gre
en :triangle] [:black :triangle] [:yellow :square]] [[:grey :obscured]
[] [] [] [] [] [:black :square] [:yellow :square] [] [:yellow :triangle
]]]}
cosy.main> (:groups-result @mem/memory)
[:shape ([{:type :region, :cluster ([1 0] [0 0])} {:type :region, :clus
ter ([2 5] [1 4] [1 5] [0 5] [1 6])} {:type :region, :cluster ([2 6] [2
 7] [1 7] [1 8])} {:type :region, :cluster ([6 7] [5 7] [4 7] [3 7] [4
8] [3 8] [2 8])} {:type :region, :cluster ([7 7] [7 8] [6 8] [6 9])} {:
type :region, :cluster ([8 3] [9 4] [8 4] [9 5])} {:type :region, :clus
ter ([10 0] [9 0] [9 1])} {:type :region, :cluster ([9 8] [9 6] [9 7])}
 {:type :region, :cluster ([10 6] [10 7])}])]
cosy.main> (second *1)
([{:type :region, :cluster ([1 0] [0 0])} {:type :region, :cluster ([2
5] [1 4] [1 5] [0 5] [1 6])} {:type :region, :cluster ([2 6] [2 7] [1 7
] [1 8])} {:type :region, :cluster ([6 7] [5 7] [4 7] [3 7] [4 8] [3 8]
 [2 8])} {:type :region, :cluster ([7 7] [7 8] [6 8] [6 9])} {:type :re
gion, :cluster ([8 3] [9 4] [8 4] [9 5])} {:type :region, :cluster ([10
 0] [9 0] [9 1])} {:type :region, :cluster ([9 8] [9 6] [9 7])} {:type
:region, :cluster ([10 6] [10 7])}])
cosy.main> (flatten *1)
({:type :region, :cluster ([1 0] [0 0])} {:type :region, :cluster ([2 5
] [1 4] [1 5] [0 5] [1 6])} {:type :region, :cluster ([2 6] [2 7] [1 7]
 [1 8])} {:type :region, :cluster ([6 7] [5 7] [4 7] [3 7] [4 8] [3 8]
[2 8])} {:type :region, :cluster ([7 7] [7 8] [6 8] [6 9])} {:type :reg
ion, :cluster ([8 3] [9 4] [8 4] [9 5])} {:type :region, :cluster ([10
0] [9 0] [9 1])} {:type :region, :cluster ([9 8] [9 6] [9 7])} {:type :
region, :cluster ([10 6] [10 7])})
cosy.main> (count *1)
9
cosy.main> (count-groups :shape)
9
cosy.main>
U:**-  *cider-repl localhost*   Bot L??    (REPL Undo-Tree yas AC ElDoc) S
```

btp] Recognizing the groups of shape



```
"colc shapec" nil nil
"found something -look for visual-routines"
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"obscurec-cells" ([0 0])
"region " [[[:grey :obscured] [:green :square]]]
"region " [[[:grey :obscured] [:green :square]]]
"current-position " nil "looked at " #{[0 0] [0 1]} "recognized-objects
 " [{:type :line, :length 2, :cluster ([4 12] [4 13])}]
({:type :line, :length 2, :cluster ([4 12] [4 13])})
"hihihihi " {:line [{:type :line, :length 2, :cluster ([4 12] [4 13])}]
}
{:line 1}
cosy.main> 
U:**-  *cider-repl localhost*   Bot L??    [(REPL Undo-Tree yas AC ElDoc)]
Find file: ~/programming/workspace/cosy/sh
```

Abbildung 3: Recognizing one line-like object

2 Excercise2

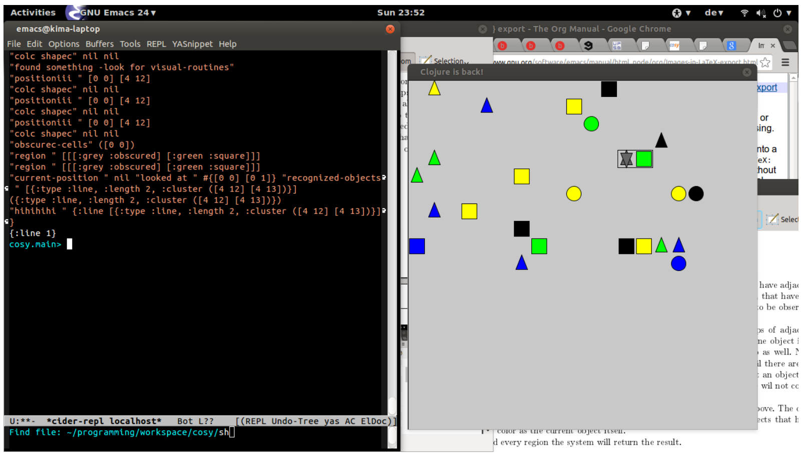
We introduced the special object 'obscured' to our cognitive architecture. A obscured object behaves like a undetermined cell in the array. This lets us handle obscured objects uniformely in the whole cognitive architecture:

- The regions detected from the peripheral view are increased to include the obscured objects because they could obscure a cell of interest
- When looking at cells for visual routines we allow obscured cells as long as we have more known cells than obscured cells. When we allow obscured cells in that way we have determined some aspects of them, for example when we recognized a 2x2 square of red circles with one obscured object in them we have determined for our system that this obscured object is a red circle. This means that our system never counts a obscured object as two different things
- because of the uniform handling of obscured objects they are also supported when counting groups.
- to count the number of obscured objects we let the peripheral view filter for regions that contain obscured objects and include all directly reachable cells from the obscured objects in them. On this region we count all objects/recognize the visual routines and count how much had obscured objects in them

3 Examples

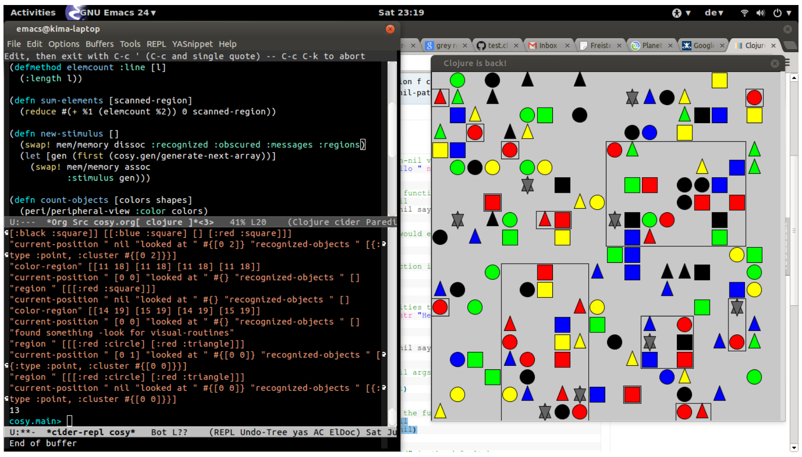
1.Here is one example where our System recognizes that there is one line-like obscured object

2.Another Example of how the system can deal with obscured objects in all circumstances. Here it has the task to count all red circles and finds 13. This is the right answer because if recognizes a line of two red circles at the bottom of the image with one being obscured



```
"colc shapec" nil nil
"found something -look for visual-routines"
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"obscurec-cells" ([0 0])
"region " [[[:grey :obscured] [:green :square]]]
"region " [[[:grey :obscured] [:green :square]]]
"current-position " nil "looked at " #{[0 0] [0 1]} "recognized-objects
 " [{:type :line, :length 2, :cluster ([4 12] [4 13])}]
({:type :line, :length 2, :cluster ([4 12] [4 13])})
"hihihihi " {:line [{:type :line, :length 2, :cluster ([4 12] [4 13])}]
}
{:line 1}
cosy.main>
U:**-  *cider-repl localhost*   Bot L??    [(REPL Undo-Tree yas AC ElDoc)]
Find file: ~/programming/workspace/cosy/sh
```

Abbildung 4: Recognizing one line-like object



```
Edit, then exit with C-c ' (C-c and single quote) -- C-c C-k to abort
(defmethod elemcount :line [l]
  (:length l))

(defn sum-elements [scanned-region]
  (reduce #(+ %1 (elemcount %2)) 0 scanned-region))

(defn new-stimulus []
  (swap! mem/memory dissoc :recognized :obscured :messages :regions)
  (let [gen (first (cosy.gen/generate-next-array))]
    (swap! mem/memory assoc
           :stimulus gen)))

(defn count-objects [colors shapes]
  (peri/peripheral-view :color colors)
U:---  *Org Src cosy.org[ clojure ]*<3>   41% L20    (Clojure cider Paredi
[:black :square]] [[:blue :square] [] [:red :square]]]
"current-position " nil "looked at " #{[0 2]} "recognized-objects " [{:
type :point, :cluster #{[0 2]}}]
"color-region" [[11 18] [11 18] [11 18] [11 18]]
"current-position " [0 0] "looked at " #{} "recognized-objects " []
"region " [[[:red :square]]]
"current-position " nil "looked at " #{} "recognized-objects " []
"color-region" [[14 19] [15 19] [14 19] [15 19]]
"current-position " [0 0] "looked at " #{} "recognized-objects " []
"found something -look for visual-routines"
"region " [[[:red :circle] [:red :triangle]]]
"current-position " [0 1] "looked at " #{[0 0]} "recognized-objects " [
{:type :point, :cluster #{[0 0]}}]
"region " [[[:red :circle] [:red :triangle]]]
"current-position " nil "looked at " #{[0 0]} "recognized-objects " [{:
type :point, :cluster #{[0 0]}}]
13
cosy.main>
U:**-  *cider-repl cosy*   Bot L??    (REPL Undo-Tree yas AC ElDoc) Sat Ju
End of buffer
```

Abbildung 5: counting - including obscured objects