Cognitive Systems Excercise 3

Maik Schünemann

June 10, 2014

Outline

How to count groups

- Similar to counting objets
- first filter for regions that are of interest
- for each object found in a interesting region scan group it belongs

filter for interesting regions

- peripheral view looks where the non-empty regions area
  - in contrast to color-based filtering from excercise 2

count groups of different types

proximity

- look at the whole cluster at once
    - cluster contains all cells reachable by going down, left,right or up

shape

- include all reachable cells with same shape as current cell

color

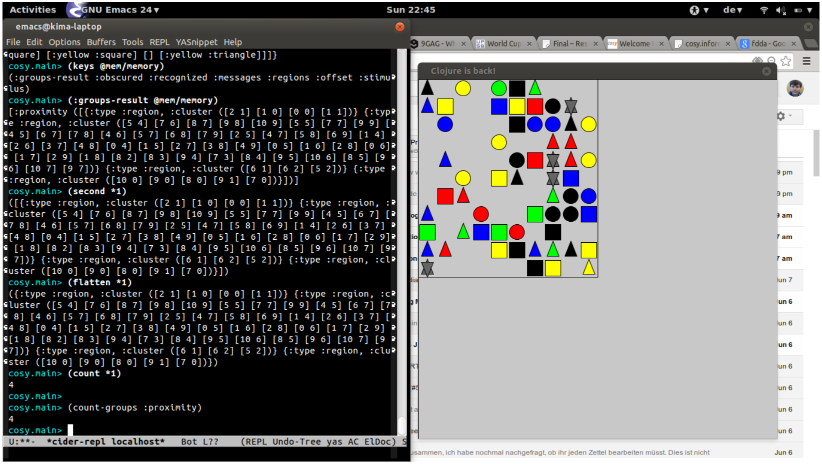
- include all reachable cells with same color as current cell

deal with obscured objects

- obscured objects means we don't know anything about the actual contents of the cell where it is
- optimistic approach:
  - if looking for objects/visual-routines include obscured cells
  - can be recognized if at least half of the objects aren't obscured
  - if included in a recognized object parts of the properties are *determined*
  - no contradictions where a obscured object is counted twice as different things

examples

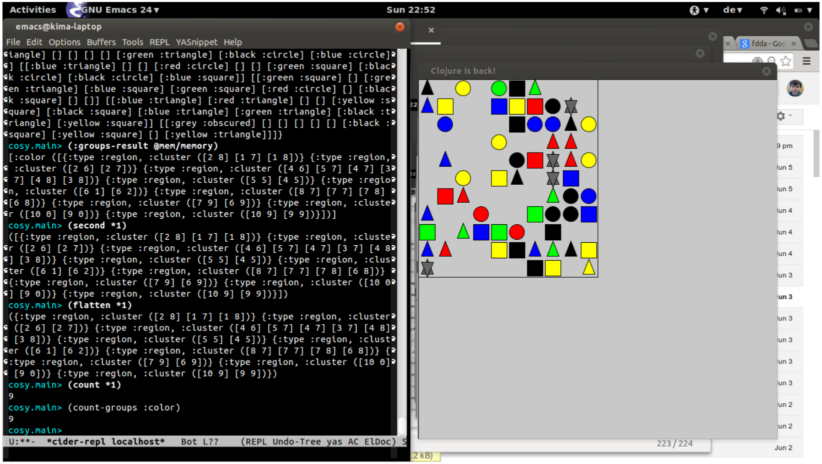
#1



```
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
```

Figure : Recognizing 4 groups of proximity

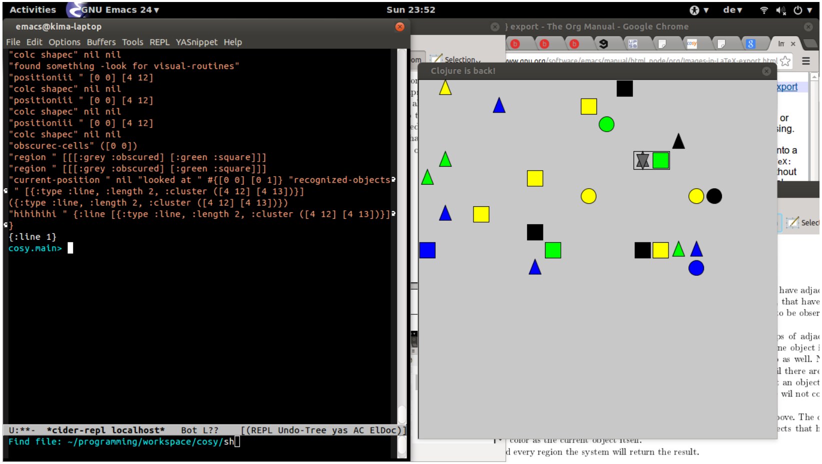
#2



```
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
```

Figure : Recognizing the groups of color

#3

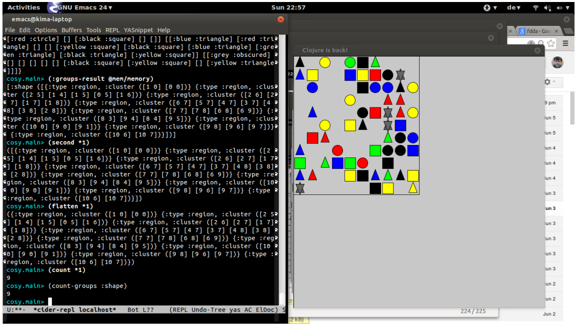


```
"colc shapec" nil nil
"found something -look for visual-routines"
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"positioniii " [0 0] [4 12]
"colc shapec" nil nil
"obscurec-cells" ([0 0])
"region " [[[:grey :obscured] [:green :square]]]
"region " [[[: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
```

Activities
GNU Emacs 24
Sun 23:52
dev
emacs@kima-laptop
File Edit Options Buffers Tools REPL YASnippet Help
} export - The Org Manual - Google Chrome
Clojure is back!

Figure : Recognizing one line-like object

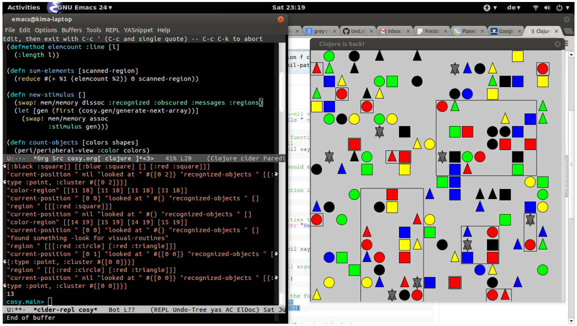
#4



```
[: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)
```

Figure : Recognizing the groups of shape

#5



```
(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)
```

```
[: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> 
```

Activities
GNU Emacs 24
Sat 23:19
emacs@kima-laptop
File Edit Options Buffers Tools REPL YASnippet Help
Edit, then exit with C-c ' (C-c and single quote) -- C-c C-k to abort
U:--- *Org Src cosy.org[clojure]*<3>   41% L20   (Clojure cider Paredi
U:**- *cider-repl cosy*   Bot L??   (REPL Undo-Tree yas AC ElDoc) Sat Ju
End of buffer
Clojure is back!

Figure : counting - including obscured objects