

Super-fun with First-class Shapes in Quil

Super-fun with First-class Shapes in Quil

Elena Machkasova, Thomas Hagen, Ryan McArthur

November 16, 2015

Table of contents

1 Who we are and why we are here

Where are we from?



UMM is a small liberal arts campus of UMN located 3 hours driving from Minneapolis/St.Paul.

What are we working on?

Developing Clojure-based introductory CS course (*ClojurEd project*).

What does this include?

- 1 Beginner-friendly error messages.
- 2 Libraries and tools that allow beginners to explore functional approaches, recursion, and abstraction.
- 3 Integration into a beginner-friendly IDE.

What are we working on?

Developing Clojure-based introductory CS course (*ClojurEd project*).

What does this include?

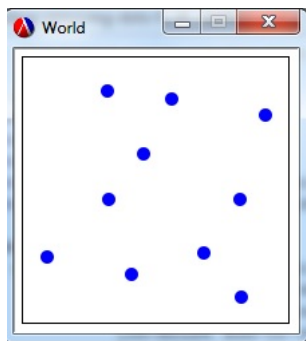
- ① Beginner-friendly error messages.
- ② **Libraries and tools that allow beginners to explore functional approaches, recursion, and abstraction: graphical library.**
- ③ Integration into a beginner-friendly IDE.

Summer project 2015.

Beginner-friendly graphical library

Inspiration: Racket "universe" package (MVC, first-class shapes):

```
(define (main duration)
  (big-bang '() ; starts with an empty list of positions.
    [to-draw display-dots] ;draw dots on canvas
    [on-tick do-nothing 1 duration] ;dots don't move w/time
    [on-mouse add-or-remove-dot])) ;click handling
```



Beginner-friendly graphical library

```
(define dot (circle 10 "solid" "blue"))

;; display-dots: list of positions -> image
(define (display-dots lop)
  (cond [(empty? lop) blank-scene]
        [else (place-image dot
                              (posn-x (first lop))
                              (posn-y (first lop))
                              (display-dots (rest lop))))]))

;; add-or-remove-dot: list of positions,
;; coordinates of click -> list of positions
.....
```

Odds and ends (not an actual slide)

Elena: Don't forget:

- ① Mention Racket influence
- ② Mention the author of Quil fun mode
- ③ Mention Tom Hall EuroClojure 2014

Shapes as First Class Objects

Thomas: like racket. Wanted to have shape object.
collage style.

- Racket-style implementation of shapes
- Shapes are treated as objects, modified through functions
- Shapes hold their specifications for drawing
- Easy to redraw wherever needed
- Easier to understand conceptually for students

Simple Shapes

Thomas: create shape template, then reuse when needed. Quil does it this way (ex)

- Quil shapes live in the draw function
- Quil shapes are functions to draw the shape

```
(defn draw-state [state]
  (q/background 100)
  (q/fill 0 255 0)
  (q/rect 300 300 100 200))
```

Our Shapes

Thomas: We do it this way (ex). Uses draw function.

- Our shapes are defined once and reused when needed
- Our shapes are drawn through the draw-shape (or ds) function

```
(def green-rectangle  
  (create-rect 100 200 :green))  
  
(defn draw-state [state]  
  (q/background 100)  
  (ds green-rectangle 300 300))
```

Creating a Collage

Thomas: We do it this way (ex). Uses draw function.

- Functional Quil uses paintbrush approach
- Our firstclass-shapes use collage approach

Thomas: Talk about mvc differences here, get Elena to word it

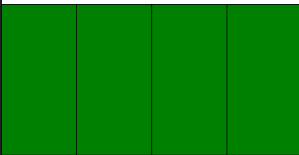
Complex Shapes

Thomas: creating complex shapes. deconstructable.

- Complex shapes are a collection of simple shapes
- Each simple shape holds their individual offsets
- Methods are used to create complex shapes from simple ones

```
(def green-rects
  (beside green-rectangle
          green-rectangle
          green-rectangle
          green-rectangle))

(defn draw-state [state]
  (q/background 100)
  (ds green-rects 300 300))
```



Above and Beside

Thomas: `show above and beside (ex)`

- Complex shapes are constructed through calling `above` or `beside`
- Can use `reduce` and `map`

```
(def green-scale-rects
  (above lime-green-rectangle
         light-green-rectangle
         green-rectangle
         dark-green-rectangle))

(defn draw-state [state]
  (q/background 100)
  (ds green-scale-rects 500 500))
```

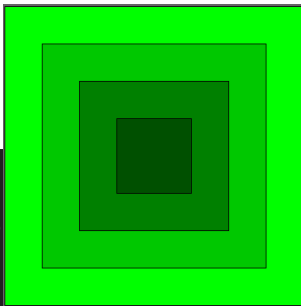


Overlay

Ryan: `show overlay`

- Complex shapes are also constructed through overlay

```
(def green-hole  
  (overlay dark-green-rectangle  
           green-rectangle  
           light-green-rectangle  
           lime-green-rectangle))  
  
(defn draw-state [state]  
  (q/background 100)  
  (ds green-hole 500 500))
```



Align

Ryan: `beside-align` `overlay-align` `align` etc. (ex)

- An `align` version of `overlay`, `above`, and `beside` exist



Rotation and Scaling

- You can modify the size and orientation of the shape

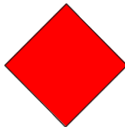
```
(rotate-shape red-square 45)
```



```
(scale-shape red-square 2 2)
```



```
(rotate-shape (scale-shape red-square 2 2) 45)
```



Images

Thomas: images treated like shapes. Rotate, applying most of the functions.

- images can be rotated and scaled similar to shapes

```
(def cool-picture  
  (create-picture "/src/images/elena.png"))  
  
(scale-shape cool-picture 2 2)  
  
(defn draw-state [state]  
  (q/background 255)  
  (ds cool-picture 500 500))
```



Thomas: Put Beach example here

Simple Shape Structure

Ryan: Explain how the shape structure is set up.

- As a data structure, simple shapes are hash's
- Shapes hold a variety of information within them

```
(:w w
:h h
:tw w
:th h
:dx 0
:dy 0
:angle 0
:ds (fn [x y pict wid hei cs angle]
      (if (> (count colors) 0)
          (apply f-fill colors)
          (no-fill))
      (with-translation [x y]
        (with-rotation [(/ (* PI angle) 180)] (f-rect 0 0 wid hei)))
      (no-fill)))
```

Complex Shape Structure

Ryan: Explain the complex shape structure

- Complex shapes are vectors of shapes
- Each shape knows its position from the core of the shape
- This allows for a 'deconstructable' complex shape

Thomas: explode example here

Draw-Shape Structure

Ryan: Explain how the draw-shape function works.

- Draw-shape calls the internal Quil draw function within the shape object
- Draw-shape also works on image objects

```
(rect-mode :center)
(image-mode :center)
(if (not (vector? shape))
  ((:ds shape) x y (:rp shape) (:w shape) (:h shape) (current-stroke) (:angle shape))
  (doall (map #((:ds %) (+ x (:dx %)) (+ y (:dy %)) (:rp %) (:w %) (:h %) (current-stroke) (:angle %)) shape)
```

Future Work

- Fill out more functionality
 - Rotate more complex shapes
 - Pixel-detail Overlay and Overlay-Align
 - More seamless integration with Quil fun-mode
- Open Source the project **Elena: Done?**
- Integrate a Clojure sound library

Acknowledgments

Elena: Need proper acknowledgments and logos; also probably thank Cognitect and other conj sponsors for providing an opportunity to talk Our research was sponsored by:

- HHMI
- LSAMP

Thank you!
Any questions?