Super-fun with First-class Shapes in Quil

Super-fun with First-class Shapes in Quil

Elena Machkasova, Thomas Hagen, Ryan McArthur

University of Minnesota, Morris

Clojure/conj: November 16, 2015

Table of contents

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?

Specific goal: developing Clojure-based introductory CS course
(*ClojurEd project*).
General goal: making Clojure more accessible to beginners and
those with no Java background.
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*).

General goal: making Clojure more accessible to beginners and those with no Java background.

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 http://racket-lang.org/

- Separation of Model, View, Control (MVC)
- Functional implementation of MVC: world state, functions:
  old world state $\rightarrow$ new world state
  world state $\rightarrow$ image
- First-class shapes (circles, rectangles, user-added jpegs, etc)
  not attached to any position
- Functions to combine simpler shapes into complex shapes:
  `above, beside, overlay, scale...`

Beginner-friendly graphical library: MVC

```
(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
```
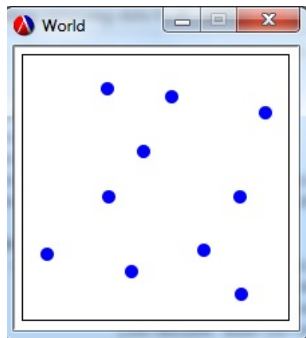
Beginner-friendly graphical library: first-class shapes

```
(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
.........
```

World States in Quil

- Using Nikita Beloglazov's Quil fun-mode (functional MVC)
- State as a HashMap

```clojure
(defn setup []
  (q/frame-rate 60)
  (q/color-mode :rgb)

  {:screen 0
   :speed 1
   :level 1
   :box-1-points 0
   :box-2-points 0
   :box-1-pos {:x 0 :y (- (q/height) 50)}
   :box-2-pos {:x (- (q/width) 50) :y (- (q/height) 50)}
   :rocks []
   :hit-player 0})
```

World States in Quil

- Elements of the state modified through functions

```
(defn update-state [state]
  "Takes in the current state and returns the updated state.
   Put functions that change your world state here"

  {:screen 1
   :speed (update-speed state)
   :level (update-level state)
   :box-1-points (update-box-1-points state)
   :box-2-points (update-box-2-points state)
   :box-1-pos (:box-1-pos state)
   :box-2-pos (:box-2-pos state)
   :rocks (update-rocks state)
   :hit-player (hit-player state)})
```

```
(defn update-rocks [state]
  (move-rocks
    (if (spawn-rocks? state)
      (assoc state :rocks (new-rock state))
      state)))


(defn update-speed [state]
  (+ 1 (* 0.1 (quot (max (:box-1-points state)
                         (:box-2-points state)) 50))))
```

Shapes as First Class Objects

- 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

Creating a Collage

- Functional Quil uses paintbrush approach

- Our firstclass-shapes use collage approach

Overview   **Ideas and examples**
Clojure first-class shapes Implementation
Future work

Simple Shapes

- 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))
```

Overview   **Ideas and examples**
Clojure first-class shapes   Implementation
Future work

Our Shapes

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

```clojure
(def green-rectangle
  (create-rect 100 200 :green))

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

Complex Shapes

- 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

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

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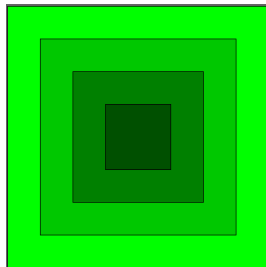
Above and Beside

- 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

- 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))
```
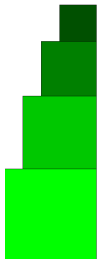
Align

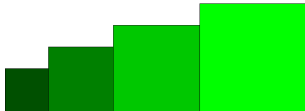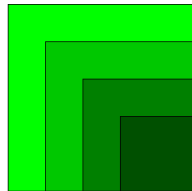- An align version of overlay, above, and beside exist

Rotation and Scaling

- You can modify the size and orientation of the shape

`(rotate-shape red-square 45)`

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

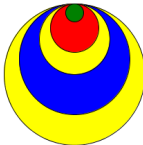`(rotate-shape (scale-shape red-square 2 2) 45)`

Code Comparison

- Strong conceptual differences in code structure


```
(defn draw-state [state]
  (q/background 255)
  (q/fill 255 255 0)
  (q/ellipse 500 500 204 204)
  (q/fill 0 0 255)
  (q/ellipse 500 482 160 160)
  (q/fill 255 255 0)
  (q/ellipse 500 453 102 102)
  (q/fill 255 0 0)
  (q/ellipse 500 437 70 70)
  (q/fill 0 128 0)
  (q/ellipse 500 415 26 26)
  (q/no-fill))
```



```
(defn setup []
  (def green-ring (create-ellipse 26 26 :green))
  (def red-ring (create-ellipse 70 70 :red))
  (def yellow-ring (create-ellipse 102 102 :yellow))
  (def blue-ring (create-ellipse 160 160 :blue))

  (def color-ring
    (overlay-align :top :center
                   green-ring
                   red-ring
                   yellow-ring
                   blue-ring
                   (scale-shape yellow-ring 2 2))))

(defn draw-state [state]
  (q/background 255)
  (ds color-ring 500 500))
```

Images

- images can be rotated and scaled similar to shapes


```
(def cool-picture
  (create-picture "src/images/kappa.png"))

(scale-shape cool-picture 2 2)

(defn draw-state [state]
  (q/background 255)
  (ds cool-picture 500 500))
```

Simple Shape Structure

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

```clojure
{: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

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

Draw-Shape Structure

- 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

- Make it easy to get the color information from shapes (currently color is hard-wired in drawing function).
- Add more functionality
  - Rotate complex shapes
  - Pixel-detail Overlay and Overlay-Align
  - Add support for text, textareas, etc.
  - More seamless integration with Quil fun-mode
- Add examples to the git repo.
- Wish-list: Integrate a Clojure sound library

Where to find it

- Clojars Page https://clojars.org/org.clojars.quil-firstclass-shapes/firstclassshapes
- Github Page https://github.com/Clojure-Intro-Course/quil-firstclass-shapes

Similar Work

Similar (completely independent) work: first-class shapes by Tom
Hall, EuroClojure 2014, based on geomlab library.
Used for educational purposes (just like ours).

Acknowledgments

Thanks to:

Morris-HHMI summer undergraduate research program at UMM

NSF North Star Stem Alliance/LSAMP grant

Science, Technology, Engineering and Mathematics

The Minnesota Louis Stokes Alliance for Minority Participation

Clojure/conj sponsors and Cognitect!

Questions?

```
(defn setup []
   (q/frame-rate 60)
   (q/color-mode :rgb)
   (def big-arc (create-arc 200 200 (- (/ q/PI -2) 0.9) (/ q/PI 2) 50))
   (def little-circle (create-ellipse 80 80 255))
   (def small-rect (create-rect 50 50 50))
   (def white-space (create-rect 50 25 255))
   (def big-rect (create-rect 50 60 50))
   (def q-mark (above (overlay-align :bottom :center
                                     big-rect
                                     (overlay
                                     little-circle
                                         big-arc))

                   big-rect
                   white-space
                   small-rect))
   {})

  (defn update-state [state] {})


 (defn draw-state [state]
  (q/background 255)
  (q/no-stroke)
  (ds q-mark 500 500))
```