Super-fun with First-class Shapes in Quil

Super-fun with First-class Shapes in Quil

Elena Machkasova, joint work with student Thomas Hagen, Ryan McArthur

University of Minnesota, Morris

Boston Clojure meetup January 14, 2016

The project

University of Minnesota, Morris (UMM) project on developing Clojure-based introductory CS course (*ClojurEd project*).
More general goal: making Clojure more accessible to beginners and those with no Java background.

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.

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...`

World States in Quil

- Using Nikita Beloglazov's Quil fun-mode (functional MVC)
- State as a HashMap
- Producing a new state on each frame and in event handlers

```
(defn setup []
  {: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})
```

- fun-mode + first class shapes = super-fun!

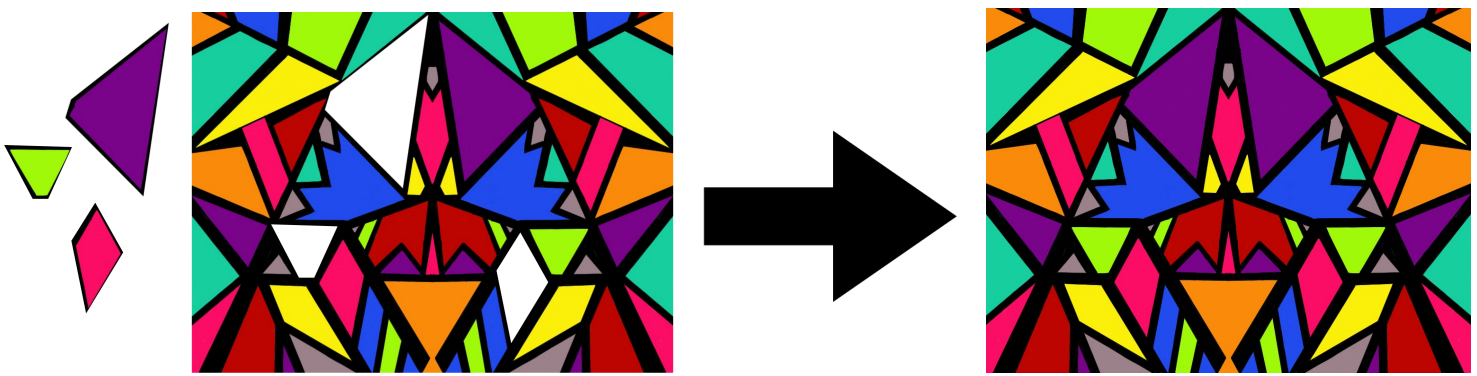
Shapes as First Class Objects

- Racket-style implementation of shapes
- Shapes are treated as objects, are combined with functions
- Shapes hold their specifications for drawing
- Easy to draw wherever needed
- Easier to understand conceptually for beginners

Creating a Collage

- Functional Quil uses paintbrush approach

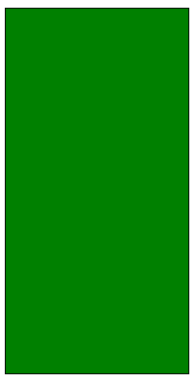
- Our firstclass-shapes use collage approach



Simple Shapes

- Quil shapes only exist in draw function, they are instructions, not objects
- Quil shapes do not exist until they are drawn
- They are not separable from their position

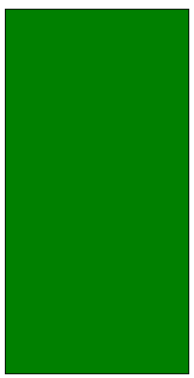
```
(defn draw-state [state]
  (q/background 100)
  (q/fill 0 255 0)
  (q/rect 300 300 100 200))
```



Our Shapes

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

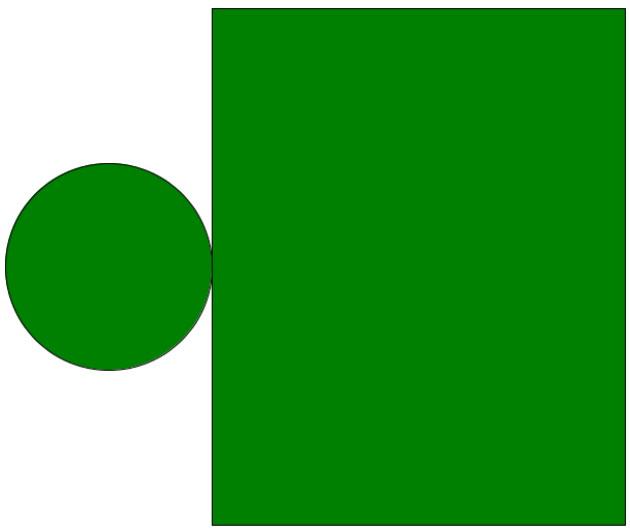
```
(def green-rectangle
  (create-rect 100 200 :green))

(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

```
(def object
  (beside green-circle
          green-rect))

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

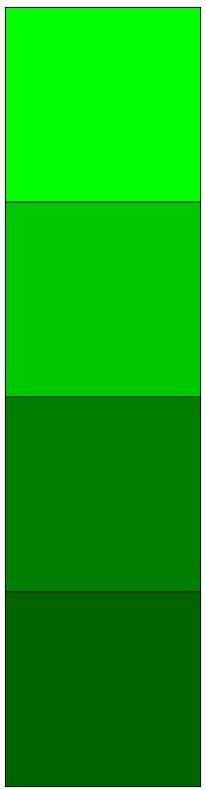


Above and Beside

- Complex shapes are constructed through calling above or beside
- Reduce can be used for similar effect

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

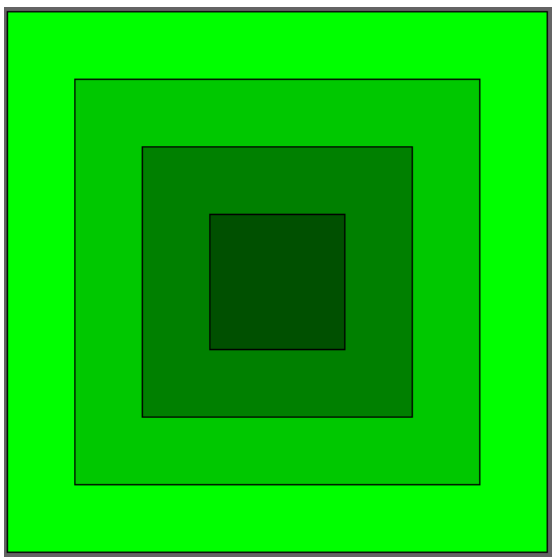
```
(def green-scale-rects-reduce
  (reduce above [lime-green-rectangle,
                 light-green-rectangle,
                 green rectangle,
                 dark-green-rectangle])
```



Overlay

- Complex shapes are also constructed through overlay

```
(def green-pyramid
  (overlay top-dark-green-rectangle
           middle-top-green-rectangle
           middle-bottom-green-rectangle
           bottom-green-rectangle))
```

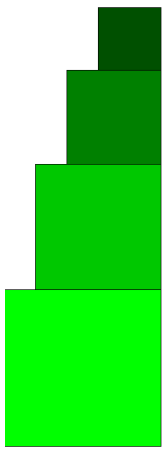


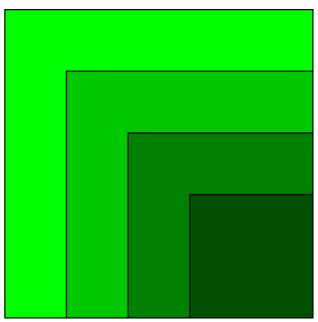
Align

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

```
(def green-lean-right
  (above-align :right
               green-rectangle-top
               green-rectangle-middle
               green-rectangle-support
               green-rectangle-base))
```

```
(def green-align-bottom-right
  (overlay-align :bottom :right
                 dark-green-rectangle-close
                 green-rectangle-step-1
                 green-rectangle-step-2
                 green-rectangle-far))
```



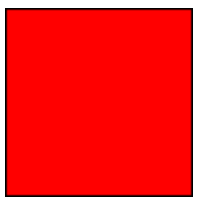


Rotation and Scaling

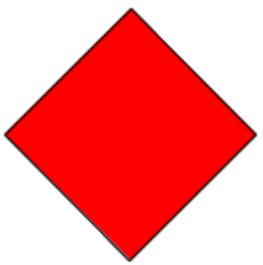
- You can modify the size and orientation of the shape

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

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



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



Images

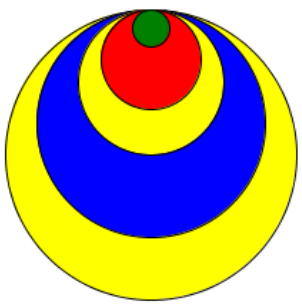
- images can be rotated and scaled similar to shapes

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

(def black-box
  (create-rect 100 50 :black))

(def on-box
  (above (scale-shape cool-picture 2 2)
         black-box))
```

Code Comparison

- Quil code vs our code

```
(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))))
```

Simple Shape Structure

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

```
(defn create-rect
  [w h & colors]
  {:width w
   :height h
   :complex-width w
   :complex-height h
   :dx 0
   :dy 0
   :angle 0
   :internal-ds-function
   (fn [x-pos y-pos picture width height current-stroke angle]
     (if (> (count colors) 0)
       (apply f-fill colors)
       (no-fill))
     (with-translation [x-pos y-pos]
       (with-rotation [(/ (* PI angle) 180)]
         (f-rect 0 0 width height)))
     (no-fill))})
```

Complex Shape Structure

- Complex shapes are vectors of simple shapes
- Each shape knows its position from the center of the shape
- This is different from Racket's implementation
- This allows for the ability to alter individual simple shapes

Draw-Shape Structure

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

```
(defn ds
  [shape x-pos y-pos]
  (if (not (vector? shape))
    ((:internal-ds-function shape) x-pos y-pos
     (:real-picture shape) (:width shape) (:height shape)
     (current-stroke) (:angle shape))

    (doall (map #((:internal-ds-function %) (+ x-pos (:dx %))
                  (+ y-pos (:dy %)) (:real-picture %) (:width %)
                  (:height %) (current-stroke) (:angle %)) shape))))
```

Future Work

- Fix bugs!
- Make it easy to get the color information from shapes (currently color is hard-wired in drawing function)
- Add more functionality
  - Add the ability to get the color of a simple shape
  - 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 overtone (music library)

Where to find it

- Clojars Page https://clojars.org/org.clojars.quil-firstclass-shapes/firstclassshapes [org.clojars.quil-firstclass-shapes/firstclassshapes "0.0.1"]
- Github Page https://github.com/Clojure-Intro-Course/quil-firstclass-shapes

Related and 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).

Prior work: Max Magnuson and Paul Schliep (UMM CS 2015 alums) developed earlier ideas for this project in 2013/14.

Acknowledgments

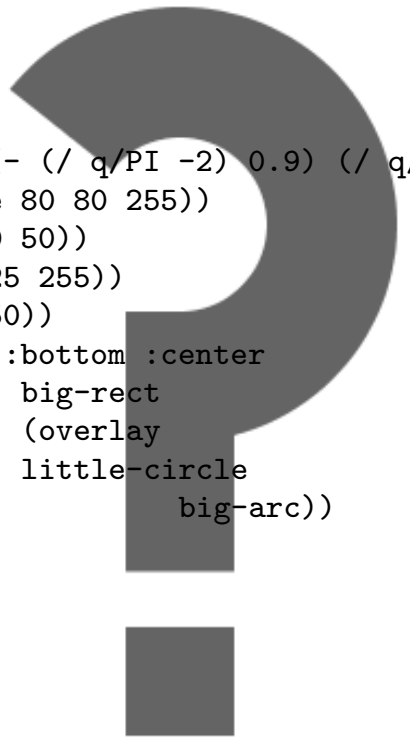
Thanks to:

Morris-HHMI summer undergraduate research program at UMM

hhmi | Howard Hughes Medical Institute

NSF North Star Stem Alliance/LSAMP grant

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))
```