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.

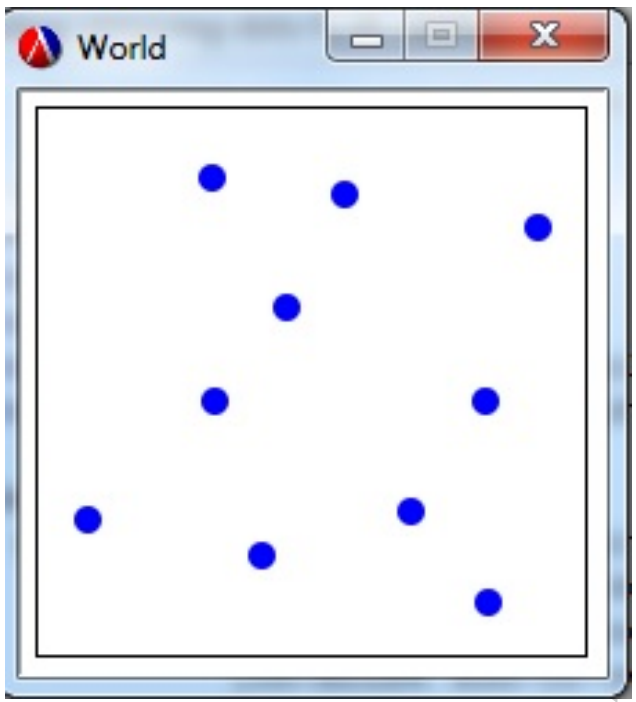
Beginner-friendly graphical library

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

Elena: Mention where the fun-mode is and where our work starts

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

```
(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!

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

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

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 beginners

Creating a Collage

`Elena: Add separate shapes for the collage`

- Functional Quil uses paintbrush approach

- Our firstclass-shapes use collage approach

Simple Shapes

- Quil shapes live in the draw function
- Quil shapes are functions to draw the shape Elena: I am not sure this true. Shapes are drawn using functions, but they don't exists in a program in the same way as ours do

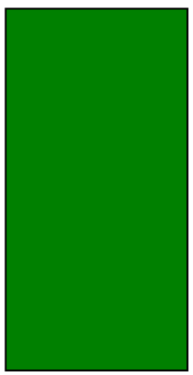
```
(defn draw-state [state]
  (q/background 100)
  (q/fill 0 255 0)
  (q/rect 300 300 100 200))
```

Our Shapes

- Our shapes are defined once in setup and reused when needed
- Our shapes are drawn through the draw-shape (or ds) function Elena: I am not sure we need to say both names: do we ever refer to it as draw-shape? Just say "ds stands for draw-shape"?

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

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



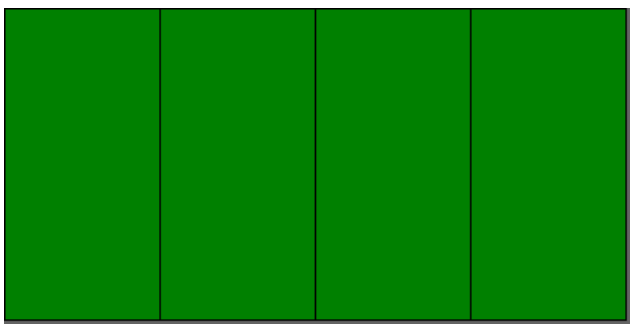
Complex Shapes

Elena: show a shape with a less obvious center, like a circle beside a rectangle?

- 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))
```

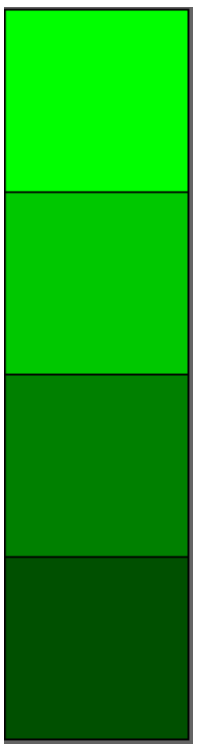


Above and Beside

- Complex shapes are constructed through calling above or beside
- Can use reduce and map `Elena: give an example of using above in reduce (in code)`

```
(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))
```

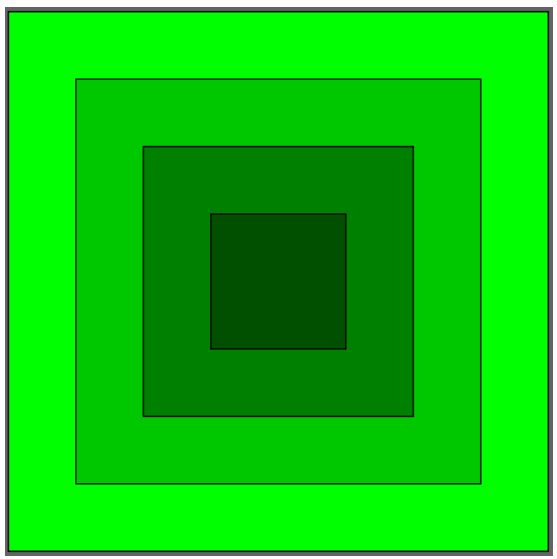


Overlay

- 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))
```



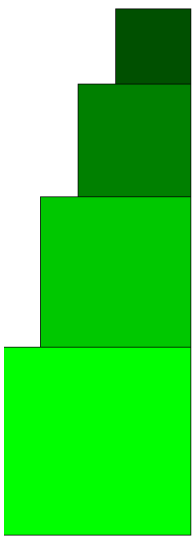
Align

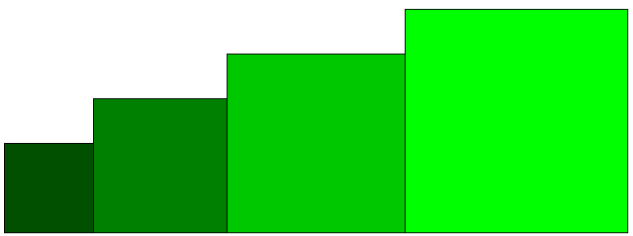
- An align version of overlay, above, and beside exist `Elena: remove ds from all of these (and prehaps from some earlier ones, too)`

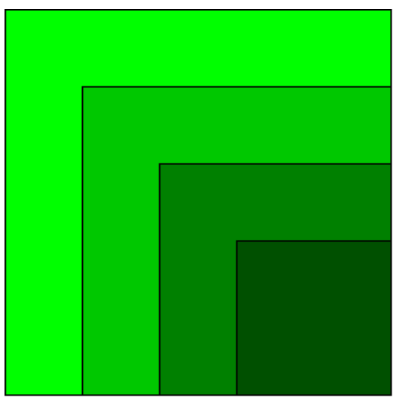
```
(def green-lean-right
  (above-align :right
               green-rectangle-top
               green-rectangle-middle
               green-rectangle-support
               green-rectangle-base))
```

```
(def green-hill
  (beside-align :right
                dark-green-rectangle
                bigger-green-rectangle
                even-bigger-green-rectangle
                biggest-green-rectangle))
```

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





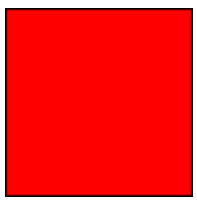


Rotation and Scaling

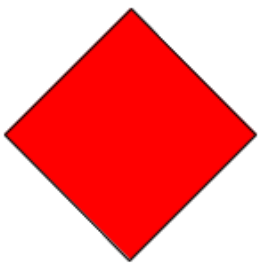
- You can modify the size and orientation of the shape

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

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



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

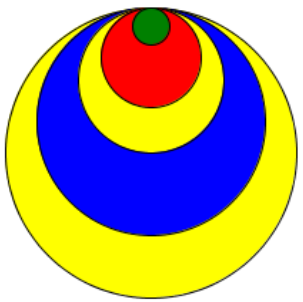


Code Comparison

Elena: Switch this slide with the next one?

- 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))))

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

Images

- images can be rotated and scaled similar to shapes `Elena: rotate the imgae, show an example of combining an image with a rectangle or some such`

```
(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
```

Simple Shape Structure

Elena: show the entire create-rectangle? Need to know what colors, pict are. Rename variables to something more reasonable, rather than wid, hei

- 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 heigh 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

Elena: Say that this implementation is different from Racket (on the slide, or at least mention)

- Complex shapes are vectors of simple shapes
- Each shape knows its position from the center of the shape
- This allows for a 'deconstructable' complex shape Elena: For the ability to access individual simple shapes? To iterate over...?

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

- 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

Elena: Nic says you should include the line for referring the package.

- 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

Clojure/conj organizers and sponsors!

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))
```