{:title “Graphics using Quil in Clojure (Part 2)” :layout :post :tags [“Clojure” “Graphics”]}

If you are interested in following along please read the first part or it may not make sense.

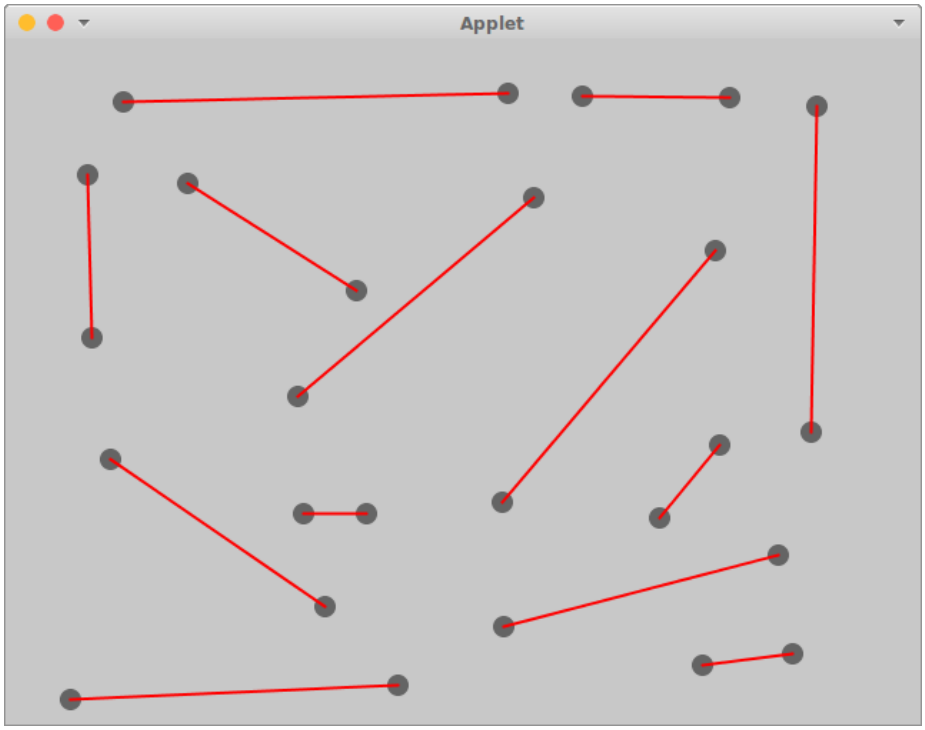

Figure 1: quil line segments

We are going to continue with the same project as we did in part 1 with newquil-project. However we are going to rewrite the draw function. First we need some defs to be added in the beginning so add the following in the top after the name space.

```
(def mouse-press-time (atom 0))     ;; The original time for when them mouse was clicked.
(def mouse-hold-x (atom 0))         ;; The x-location when the mouse was clicked.
(def mouse-hold-y (atom 0))         ;; The y-location when the mouse was clicked.
(def mouse-release-stage (atom 0))  ;; The release stage (0 - nothing, 1 - pressed)
```

Those will come in handy while setting up our draw function which we’ll completely rewrite. Change the stroke/stroke-weight/fill to the following values.

```
  (q/stroke 100)          ;; Change to 100 rather than q/random
```

```
  (q/stroke-weight 5)     ;; Change to 5
  (q/fill 100)            ;; Change to 100
```

Now you can remove everything else in the draw function. Adding the following into the draw function after the stroke/stroke-weight/fill.

```
(if (q/mouse-pressed?) ;; When the mouse is pressed it returns true.
    (do (if (= 0 @mouse-release-stage)      ;; This can only happen when mouse-release-stage
          (do (let [diam 10]
                (q/ellipse (q/mouse-x) (q/mouse-y) diam diam))  ;; Creates a circle of 10 di
              (reset! mouse-press-time (q/millis)) ;; Sets the press time to the current tim
              (reset! mouse-hold-x (q/mouse-x))    ;; Sets the mouse-hold-x time to where th
              (reset! mouse-hold-y (q/mouse-y))    ;; Same for the y coordinate.
              (reset! mouse-release-stage 1))))    ;; Sets the mouse-release-stage to 1 so t
    (do (if (= 1 @mouse-release-stage)             ;; Only occurs if the mouse-release stage
          (do (reset! mouse-release-stage 0)
              (let [delta-time (- (q/millis) @mouse-press-time) ;; The time change between t
                    mouse-release-x (q/mouse-x)     ;; Release x coordinate.
                    mouse-release-y (q/mouse-y)     ;; Release y coordinate.
                    delta-x (- mouse-release-x @mouse-hold-x)  ;; Change in x coordinate.
                    delta-y (- mouse-release-y @mouse-hold-y)  ;; Change in y coordinate.
                    distance (Math/sqrt (+ (* delta-x delta-x) (* delta-y delta-y))) ;; Find
                    vel (/ distance delta-time)] ;; Calculates the avg velocity by taking th
                (let [diam 10]
                  (q/ellipse (q/mouse-x) (q/mouse-y) diam diam)) ; creates another circle at
                (q/stroke 255 0 0)
                (q/stroke-weight 2)
                (q/line @mouse-hold-x @mouse-hold-y mouse-release-x mouse-release-y) ;; Crea
                (println "Dt: " delta-time "Dx: " delta-x "Dy: " delta-y "X: " distance "V:
```

Now you can run it by the following...

```
lein run
```

Now you can click on the applet and release in a different place and should see lines connecting two circles! As a bonus if you want to clear the applet without exiting you can add the following in the draw function...

```
(if (q/key-pressed?)
  (do (if (= (.toString (q/raw-key)) "c")
          (do (q/clear)
              (q/background 200)))
```

So this is what your entire core.clj should look like. More or less with some extra key presses.

```
(ns newquil-project.core
  (:require [quil.core :as q]
            [quil.middleware :as m]))

(def brush (atom 0)) ;; if the brush is up or down (0 - down 1 - up)
(def mouse-press-time (atom 0)) ;; The original time for the mouse press
(def mouse-hold-x (atom 0))
(def mouse-hold-y (atom 0))
(def mouse-release-stage (atom 0))

(defn setup []
  (q/frame-rate 20)
  (q/background 200))

(defn draw []
  (q/stroke 100)
  (q/stroke-weight 5)
  (q/fill 100)

  (if (= 1 @brush)
    (let [diam (q/random 100)]
      (q/ellipse (q/mouse-x) (q/mouse-y) diam diam)))

  (if (q/mouse-pressed?)
    (do (if (= 0 @mouse-release-stage)
          (do (let [diam 10]
                (q/ellipse (q/mouse-x) (q/mouse-y) diam diam))
              (reset! mouse-press-time (q/millis))
              (reset! mouse-hold-x (q/mouse-x))
              (reset! mouse-hold-y (q/mouse-y))
              (reset! mouse-release-stage 1))))
    (do (if (= 1 @mouse-release-stage)
          (do (reset! mouse-release-stage 0)
              (let [delta-time (- (q/millis) @mouse-press-time)
                    mouse-release-x (q/mouse-x)
                    mouse-release-y (q/mouse-y)
                    delta-x (- mouse-release-x @mouse-hold-x)
                    delta-y (- mouse-release-y @mouse-hold-y)
                    distance (Math/sqrt (+ (* delta-x delta-x) (* delta-y delta-y)))
                    vel (/ distance delta-time)]
                (let [diam 10]
                  (q/ellipse (q/mouse-x) (q/mouse-y) diam diam))
                (q/stroke 255 0 0)
                (q/stroke-weight 2)
                (q/line @mouse-hold-x @mouse-hold-y mouse-release-x mouse-release-y)
                (println "Dt: " delta-time "Dx: " delta-x "Dy: " delta-y "X: " distance "V:
```

```
  (if (q/key-pressed?)
    (do (if (= (.toString (q/raw-key)) "q")
          (do (q/exit)))
        (if (= (.toString (q/raw-key)) "u")
          (do (if (= 0 @brush)
                (reset! brush 1)
                (reset! brush 0))))
        (if (= (.toString (q/raw-key)) "e")
          (do (let [radius (q/random 75)]
                (q/box radius radius radius))))
        (if (= (.toString (q/raw-key)) "c")
          (do (q/clear)
              (q/background 200))))))

(q/defsketch example
  :title "Oh so many grey circles"
  :settings #(q/smooth 2)
  :setup setup
  :draw draw
  :size [640 480])
```

And there you have it you can make line segments! And do all kinds of things if you add onto them.