

CS-257L

Nonimperative Programming: Scheme!

Instructor:

Joel Castellanos

e-mail: joel@unm.edu

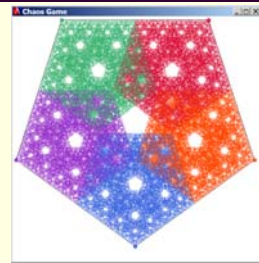
Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering
Center (FEC) room 321

Scheme
Vectors

Vectors applied
in the
Chaos Game

Vectors to
define Matrix



5/7/2008

Quiz Friday

Bring to class a short (2 to 5 minute) demo of a cool / novel / artistic I-system drawn by your I-system scheme code.

Content:

- May show off whatever extra credit you did.
- May be a new I-system that you make up.
- May be an I-system you find on the Internet or in a book and draw with your Scheme software.

Content format:

- May be a JPEG or PNG image.
- May be a few PowerPoint slides.
- May be running scheme code.

Media:

Your laptop connected to the projector
USB drive.

5/7/2008

2

Scheme Vectors

```
(vector 5 6 7 8)
(vector 5 6 7 8 9)
(define x (vector 5 6 7 8 9))
(vector-ref x 2)
(vector-set! x 2 33)
x
```

Output

```
#4(5 6 7 8)
#5(5 6 7 8 9)
7
#5(5 6 33 8 9)
```

5/7/2008

3

Vectors of Objects: Pens

```
(define penVector
  (vector
    (instantiate pen% ("RoyalBlue" 4 'solid))
    (instantiate pen% ("DarkOrchid" 4 'solid))
    (instantiate pen% ("MediumSeaGreen" 4 'solid))
    (instantiate pen% ("Crimson" 4 'solid))
    (instantiate pen% ("OrangeRed" 4 'solid))
  ))

(define getPen
  (lambda (idx)
    (vector-ref penVector idx)
  )
)
```

5/7/2008

4

Vectors of Objects: Vertices

```
;Vertices of equilateral triangle
(ChaosGame
  (vector (vector 0.0 1.0)
          (vector 1.0 1.0)
          (vector 0.5 (- 1.0 (sin (/ pi 3.0))))))
)

(define getX (lambda (idx)
  (vector-ref
   (vector-ref vertexVector idx) 0))
)

(define getY (lambda (idx)
  (vector-ref
   (vector-ref vertexVector idx) 1))
)
```

5/7/2008

5

Sierpiński Triangle via Chaos Game

1. Define the vertices of an equilateral triangle inscribed in a unit square.
2. Choose a random point, p_0 , within the unit square.
3. Choose a random vertex v_i of the equilateral triangle.
4. Iteratively define p_i to be the midpoint of the line segment with endpoints p_{i-1} and v_i .
5. Excepting the first few points, the set $\{p_i, p_{i+1}, \dots, p_n\}$ form the Sierpiński Triangle.

5/7/2008

6

Sierpiński Triangle: World to Screen

```
;Since the triangle is inscribed in a unit square,  
;The only offset needed is the border.  
(set! scale (- (- (min drawWidth drawHeight)  
                border ) border)  
)  
  
(define getScreenX  
  (lambda (x)  
    (+ (floor (* x scale)) border)  
  )  
)  
  
(define getScreenY  
  (lambda (y)  
    (+ (floor (* y scale)) border)  
  )  
)
```

5/7/2008

7

Sierpiński Triangle: plot-ball

```
(define plot-ball  
  (lambda (dc x y idx)  
    (send dc set-pen (getPen idx))  
  
    (send dc draw-ellipse  
      (- (getScreenX x) 3)  
      (- (getScreenY y) 3) 7 7)  
    )  
  )
```

5/7/2008

8

Sierpiński Triangle: DrawVertexies

```
(define DrawVertexies (lambda (dc)
  (do ((i 0 (+ i 1))) ((>= i vertexCount))
    (plot-ball dc (getX i) (getY i) i)
    (send dc set-pen pen-black)
    (let ((ii (modulo (+ i 1) vertexCount)))
      (plot-line dc (getX i) (getY i)
                   (getX ii) (getY ii))
      )
    )
  )
))
```

5/7/2008

9

Sierpiński Triangle: Main Loop

```
(do ((i 0 (+ i 1))) ((> i 10000000))
  (sleep .001)
  (set! idx
    (inexact->exact (floor
                      (* (random) vertexCount))))
  (set! x (/ (+ x (getX idx)) 2))
  (set! y (/ (+ y (getY idx)) 2))
  (plot dc x y)
  )
```

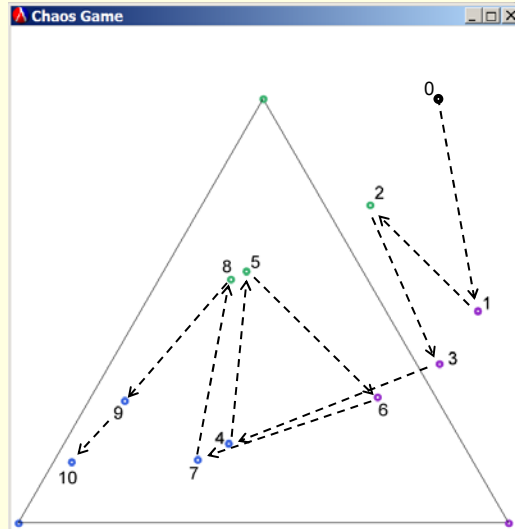
5/7/2008

10

Chaos Game – First 10 Points

```
(ChaosGame
;Vertices of
;Equilateral Triangle
(vector
(vector 0.0 1.0)
(vector 1.0 1.0)
(vector
0.5
(- 1.0 (sin (/ pi 3.0))))
) 10 ;points
)
```

Points are colored green blue or violet to match the color of the next randomly chosen vertex.

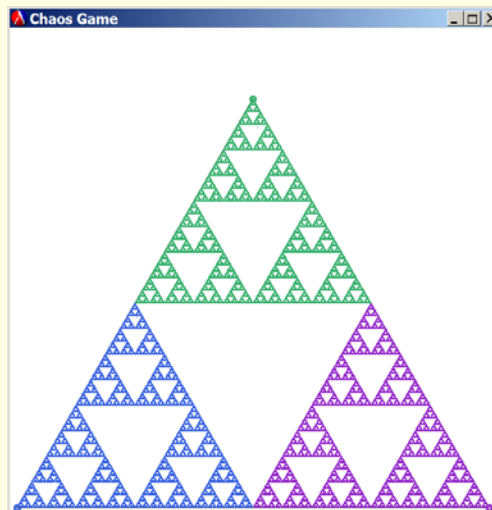


5/7/2008

11

Chaos Game – Equilateral Triangle

```
(ChaosGame
;Vertices of
;Equilateral Triangle
(vector
(vector 0.0 1.0)
(vector 1.0 1.0)
(vector
0.5
(- 1.0 (sin (/ pi 3.0))))
) 500000 ;points
)
```

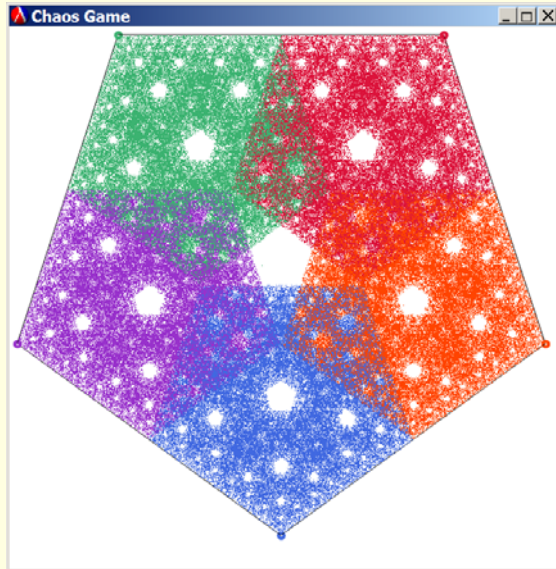


5/7/2008

12

Chaos Game – Regular Pentagon

```
(ChaosGame
;Vertices of
;regular pentagon
(vector
  (vector
    0.5 0.951056516)
  (vector
    0.0 0.587785252)
  (vector
    0.190983006 0.0)
  (vector
    0.809016994 0.0)
  (vector
    1.0 0.587785252)
)
500000 ;points
)
```



5/7/2008

Lists as Function Arguments

```
(define faa
  (lambda ()
    (define mylist '("Abby" "Bill" "Carl"))
    (baa mylist)
    (display mylist) (newline)
  ))

(define baa
  (lambda (localList)
    (set! localList '("Abby" "ZZZZZ" "Carl"))
    (display localList) (newline)
  ))

(faa)
```

(Abby ZZZZZ Carl)
(Abby Bill Carl)

Output

5/7/2008

14

Vectors as Function Arguments

```
(define foo
  (lambda ()
    (define myVector (vector "Abby" "Bill" "Carl"))
    (boo myVector)
    (display myVector) (newline)
  ))

(define boo
  (lambda (localVector)
    (vector-set! localVector 1 "XXXXX")
    (display localVector) (newline)
  ))

(foo)
```

Output

```
 #(Abby XXXXX Carl)
 #(Abby XXXXX Carl)
```

5/7/2008

15

Vector - Copy by Reference

```
(define fun
  (lambda (myVector1)
    (define myVector2 myVector1)
    (vector-set! myVector2 1 "YYYYY")
    (display myVector1) (newline)
    (display myVector2) (newline)
  )
)

(fun (vector "Abby" "Bill" "Carl"))
```

Output

```
 #(Abby YYYYY Carl)
 #(Abby YYYYY Carl)
```

5/7/2008

16

Vector - Copy by Value

```
(define fun
  (lambda (myVector1)
    (define myVector2
      (apply vector (vector->list myVector1)))
    )
    (vector-set! myVector2 1 "YYYYY")
    (display myVector1) (newline)
    (display myVector2) (newline)
  )
)
(fun (vector "Abby" "Bill" "Carl"))
```

Output

```
 #(Abby Bill Carl)
 #(Abby YYYYY Carl)
```

5/7/2008

17

Nested Vectors

```
(define f1 (lambda (myVector1)
  (define myVector2
    (apply vector (vector->list myVector1)))
  (define myVector3
    (apply vector (vector->list myVector1)))
  (vector-set! myVector2 0 (vector 77 88))
  (vector-set! (vector-ref myVector3 1) 0 333)
  (display myVector1) (newline)
  (display myVector2) (newline)
  (display myVector3) (newline)
))
(f1 (vector (vector 1 2) (vector 7 8) (vector 11 12)))
```

Output

```
##(1 2) ##(333 8) ##(11 12))
##(77 88) ##(333 8) ##(11 12))
##(1 2) ##(333 8) ##(11 12))
```

(apply) uses
Flat Recursion not
Deep Recursion

5/7/2008

18

copyVectorOfOrderedPairsByValue

```
(define copyVectorOfOrderedPairsByValue
  (lambda (source sink)
    (define x 0) (define y 0)
    (do ((i 0 (+ i 1))) ((>= i (vector-length source)))
      (set! x (vector-ref (vector-ref source i) 0))
      (set! y (vector-ref (vector-ref source i) 1))
      (vector-set! (vector-ref sink i) 0 x)
      (vector-set! (vector-ref sink i) 1 y)
    ))
  (define v1 (vector (vector 1 2) (vector 7 8)))
  (define v2 (vector (vector 0 0) (vector 0 0)))
  (copyVectorOfOrderedPairsByValue v1 v2)
  (vector-set! (vector-ref v2 0) 0 333)
  (display v1) (newline)
  (display v2) (newline)
```

What are possible errors?

Output

```
##(1 2) ##(7 8)
##(333 2) ##(7 8)
```

5/7/2008

19

Matrix

;Defines a 2D Matrix class.

```
(define matrix
  (lambda (columns rows)
    (define myVector
      (make-vector
        (+ 1 (* rows columns)) 0)
    )
    ;Save the number of rows in myVector[0]
    ;This is needed for referencing elements.
    (vector-set! myVector 0 rows)
    myVector
  )
)
```

5/7/2008

20

Matrix – Setting an Element

```
;Use analogous syntax to (vector-set! ... )
(define matrix-set!
  (lambda (myVector x y value)
    (define rows (vector-ref myVector 0))
    (vector-set! myVector
      (+ (+ (* rows x) y) 1)
      value)
  )
)
```

5/7/2008

21

Matrix – Reading an Element

```
;Use analogous syntax to (vector-ref ... )
(define matrix-ref
  (lambda (myVector x y)
    (define rows (vector-ref myVector 0))
    (vector-ref myVector
      (+ (+ (* rows x) y) 1)
    )
  )
)
```

5/7/2008

22