アルゴリズムとデータ構造入門 必修課題2

1029-24-9540 山崎啓太郎

February 7, 2013

1 painterを1種類作成

```
(define perorin (lambda (part)
        (cond
                ((eq? part 'frame) (vertexes->painter
                        (list
                                (make-vect .05 .1) (make-vect .95 .1)
                                (make-vect .95 .1) (make-vect .5 .85)
                                (make-vect .5 .85) (make-vect .05 .1))
                        #t ))
                ((eq? part 'left-eye) (vertexes->painter
                        (list
                                (make-vect .35 .5) (make-vect .4 .5)
                                (make-vect .4 .5) (make-vect .4 .55)
                                (make-vect .4 .55) (make-vect .35 .55)
                                (make-vect .35 .55) (make-vect .35 .5))
                        #t ))
                ((eq? part 'right-eye) (vertexes->painter
                        (list
                                (make-vect .65 .5) (make-vect .6 .5)
                                (make-vect .6 .5) (make-vect .6 .55)
                                (make-vect .6 .55) (make-vect .65 .55)
                                (make-vect .65 .55) (make-vect .65 .5))
                        #t ))
                ((eq? part 'mouth) (vertexes->painter
                        (list
                                (make-vect .2 .25) (make-vect .8 .25)
                                (make-vect .8 .25) (make-vect .8 .28)
                                (make-vect .8 .28) (make-vect .2 .28)
                                (make-vect .2 .28) (make-vect .2 .25))
                        #t ))
```

```
30                  ((eq? part 'tongue) (vertexes->painter
31                          (list
32                                  (make-vect .6 .25) (make-vect .74 .25)
33                                  (make-vect .74 .25) (make-vect .67 .4)
34                                  (make-vect .67 .4) (make-vect .6 .25))
35                          #t )) ) ))
36
37  (define draw-perorin (lambda (frm)
38          (clear-picture)
39          (set-color #x0B6A4A)
40          ((perorin 'frame) frm)
41          (set-color #x0ffffff)
42          ((perorin 'left-eye) frm)
43          ((perorin 'right-eye) frm)
44          ((perorin 'mouth) frm)
45          (set-color #xCB361F)
46          ((perorin 'tongue) frm)
47  ))
48
49  (define draw-perorin-square-limit (lambda (n frm)
50          (clear-picture)
51          (set-color #x0B6A4A)
52          ((square-limit (perorin 'frame) n) frm)
53          (set-color #x0ffffff)
54          ((square-limit (perorin 'left-eye) n) frm)
55          ((square-limit (perorin 'right-eye) n) frm)
56          ((square-limit (perorin 'mouth) n) frm)
57          (set-color #xCB361F)
58          ((square-limit (perorin 'tongue) n) frm)
59  ))
```

2 square-limit に適用

```
1  (define draw-perorin-square-limit (lambda (n frm)
2          (clear-picture)
3          (set-color #x0B6A4A)
4          ((square-limit (perorin 'frame) n) frm)
5          (set-color #x0ffffff)
6          ((square-limit (perorin 'left-eye) n) frm)
7          ((square-limit (perorin 'right-eye) n) frm)
8          ((square-limit (perorin 'mouth) n) frm)
9          (set-color #xCB361F)
10         ((square-limit (perorin 'tongue) n) frm)
```

```
11 ))
12
13 (load "init.lsp")
14 (load "painter.scm")
15 (load "square-limit.scm")
16 (draw-perorin-square-limit 4 frm1)
```

画像は perorin.png に保存してあります。

3 空間充填曲線を 1 種類作成

```
 1 (define hilbert-a (lambda (x0 y0 x1 y1 i)
 2         (let ((xs (/ (+ (* 3.0 x0) x1) 4.0))
 3                 (ys (/ (+ (* 3.0 y0) y1) 4.0))
 4                 (xm (/ (+ x0 x1) 2.0))
 5                 (ym (/ (+ y0 y1) 2.0))
 6                 (xl (/ (+ x0 (* 3.0 x1)) 4.0))
 7                 (yl (/ (+ y0 (* 3.0 y1)) 4.0)) )
 8
 9                 (if (= i 0)
10                         (list (make-vect xl yl) (make-vect xs yl)
11                                 (make-vect xs ys) (make-vect xl ys))
12                         (append (hilbert-d xm ym x1 y1 (- i 1))
13                                 (hilbert-a x0 ym xm y1 (- i 1))
14                                 (hilbert-a x0 y0 xm ym (- i 1))
15                                 (hilbert-b xm y0 x1 ym (- i 1))) ))))
16 (define hilbert-b (lambda (x0 y0 x1 y1 i)
17         (let ((xs (/ (+ (* 3.0 x0) x1) 4.0))
18                 (ys (/ (+ (* 3.0 y0) y1) 4.0))
19                 (xm (/ (+ x0 x1) 2.0))
20                 (ym (/ (+ y0 y1) 2.0))
21                 (xl (/ (+ x0 (* 3.0 x1)) 4.0))
22                 (yl (/ (+ y0 (* 3.0 y1)) 4.0)) )
23
24                 (if (= i 0)
25                         (list (make-vect xs ys) (make-vect xs yl)
26                                 (make-vect xl yl) (make-vect xl ys) )
27                         (append (hilbert-c x0 y0 xm ym (- i 1))
28                                 (hilbert-b x0 ym xm y1 (- i 1))
29                                 (hilbert-b xm ym x1 y1 (- i 1))
30                                 (hilbert-a xm y0 x1 ym (- i 1))) ))))
31 (define hilbert-c (lambda (x0 y0 x1 y1 i)
```

```
32          (let ((xs (/ (+ (* 3.0 x0) x1) 4.0))
33                  (ys (/ (+ (* 3.0 y0) y1) 4.0))
34                  (xm (/ (+ x0 x1) 2.0))
35                  (ym (/ (+ y0 y1) 2.0))
36                  (xl (/ (+ x0 (* 3.0 x1)) 4.0))
37                  (yl (/ (+ y0 (* 3.0 y1)) 4.0)) )
38
39                  (if (= i 0)
40                          (list (make-vect xs ys)(make-vect xl ys)
41                                  (make-vect xl yl) (make-vect xs yl) )
42                          (append (hilbert-b x0 y0 xm ym (- i 1))
43                                  (hilbert-c xm y0 x1 ym (- i 1))
44                                  (hilbert-c xm ym x1 y1 (- i 1))
45                                  (hilbert-d x0 ym xm y1 (- i 1))) ))))
46  (define hilbert-d (lambda (x0 y0 x1 y1 i)
47          (let ((xs (/ (+ (* 3.0 x0) x1) 4.0))
48                  (ys (/ (+ (* 3.0 y0) y1) 4.0))
49                  (xm (/ (+ x0 x1) 2.0))
50                  (ym (/ (+ y0 y1) 2.0))
51                  (xl (/ (+ x0 (* 3.0 x1)) 4.0))
52                  (yl (/ (+ y0 (* 3.0 y1)) 4.0)) )
53
54                  (if (= i 0)
55                          (list (make-vect xl yl) (make-vect xl ys)
56                                  (make-vect xs ys) (make-vect xs yl))
57                          (append (hilbert-a xm ym x1 y1 (- i 1))
58                                  (hilbert-d xm y0 x1 ym (- i 1))
59                                  (hilbert-d x0 y0 xm ym (- i 1))
60                                  (hilbert-c x0 ym xm y1 (- i 1))) ))))
61
62  (define vectors->segments (lambda (vectors)
63          (define vector->segment (lambda (vector-list)
64                  (define a (car vector-list))
65                  (define b (cadr vector-list))
66                  (if (eq? nil b)
67                          '()
68                          (append (list (make-segment
69                                  (make-vect (car a) (cdr a))
70                                  (make-vect (car b) (cdr b))))
71                                  (vector->segment (cdr vector-list))) )))
72          (vector->segment vectors) ))
73
74  (define hilbert (lambda (n)
```

```
75          (segments->painter
76                  (vectors->segments (hilbert-a 0.0 0.0 1.0 1.0 n))) ))
77
78  ; 以下は使用例です
79  (load "init.lsp")
80  (clear-picture)
81  ((hilbert 4) frm1)
```

Hilbert circle を作成しました。
画像は hilbert.png にあります。

4 フラクタルを 1 種類作成

```
 1  (define kock-line (lambda (p0 q0 p1 q1 r i)
 2          (if (= i 0)
 3                  (list (make-vect p0 q0) (make-vect p1 q1))
 4                  (let* ((r1 (/ r 3.0))
 5                         (x3 (/ (- p1 p0) 3.0))
 6                         (y3 (/ (- q1 q0) 3.0))
 7                         (xs (/ (+ (* 2.0 p0) p1) 3.0))
 8                         (ys (/ (+ (* 2.0 q0) q1) 3.0))
 9                         (xl (/ (+ p0 (* 2.0 p1)) 3.0))
10                         (yl (/ (+ q0 (* 2.0 q1)) 3.0))
11                         (xm (+ (* 0.5 x3) (* 0.866 y3) xs))
12                         (ym (+ (* 0.5 y3) (* -0.866 x3) ys)) )
13                         (append (kock-line p0 q0 xs ys r1 (- i 1))
14                                 (kock-line xs ys xm ym r1 (- i 1))
15                                 (kock-line xm ym xl yl r1 (- i 1))
16                                 (kock-line xl yl p1 q1 r1 (- i 1)) )))))
17
18  (define vectors->segments (lambda (vectors)
19          (define vector->segment (lambda (vector-list)
20                  (define a (car vector-list))
21                  (define b (cadr vector-list))
22                  (if (eq? nil b)
23                          '()
24                          (append (list (make-segment
25                                  (make-vect (car a) (cdr a))
26                                  (make-vect (car b) (cdr b))))
27                                  (vector->segment (cdr vector-list))) )))
28          (vector->segment vectors) ))
29
30  (define kock (lambda (n)
```

```
31          (let* ((h (/ 0.75 0.86))
32                 (p0 (/ (- 1.0 h) 2))
33                 (p1 (- 1.0 p0)) )
34                 (segments->painter
35                         (vectors->segments
36                                 (append
37                                         (kock-line p0 0.25 p1 0.25 1 n)
38                                         (kock-line p1 0.25 0.5 1.0 1 n)
39                                         (kock-line 0.5 1.0 p0 0.25 1 n) )))
40
41  ; 以下は使用例
42  (load "init.lsp")
43  ((kock 3) frm1)
```

Kock Snowflake を作成しました。
画像は snowflake.png にあります。