Racket Assignment #3: Recursion in Racket

Abstract

This paper showcases my experience with some relatively simple recursive functions in Racket programming.

Task 1: Counting Down, Counting Up

Code

```
( define ( count-down n )
      ( cond
         ( ( > n 0 )
           ( display n ) ( display "\n")
           ( count-down ( - n 1 ) )
         )
      )
)

(define (count-up n)
    ( define ( count-pro i )
       ( cond
          (  ( < n 0 )
             ( display "\n" ) )
          (  ( <= i n )
            ( display i ) ( display "\n")
             ( count-pro ( + i 1) ) ) ) )
   ( count-pro 1 )
  )
```

Demo

```
Welcome to DrRacket, version 8.7 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (count-down 5 )
5
4
3
2
1
> (count-down 10 )
10
9
8
7
6
5
4
3
2
1
> (count-down 20 )
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
>
```

```
> ( count-up 5 )
1
2
3
4
5
> ( count-up 10 )
1
2
3
4
5
6
7
8
9
10
> ( count-up 20 )
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
>
```

Task 2: Triangle of Stars

Code

```
(define (triangle-of-stars n)
 (define (count-up i)
   (cond
     ((= i 0) (display "\n"))
     (else
      (display "* ")
      (count-up (- i 1)))))
 (let loop ((i 1))
   (when (<= i n)
     (count-up i)
     (loop (+ i 1)))))
```

Demo

```
> (triangle-of-stars 5 )
*
* *
* * *
* * * *
* * * * *
> (triangle-of-stars 0 )
> (triangle-of-stars 15 )
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
>
```

Task 3: Flipping a Coin

Code

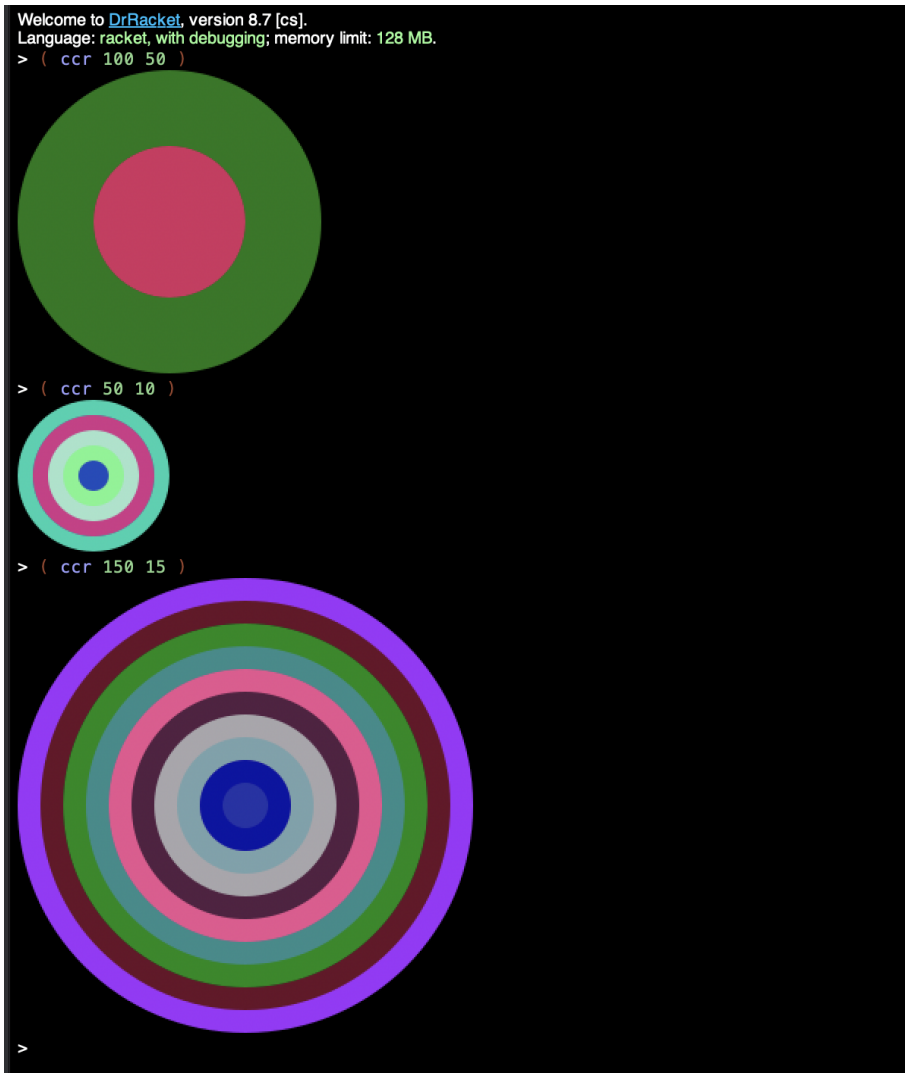
```
(define (flip-for-difference n)
  ( define ( flip-coin )
     ( define outcome ( random 2 ) )
     ( cond
        ( ( = outcome 0 ) 't )
        ( ( = outcome 1 ) 'h ) ) )
  (define (flip-for-difference-helper heads tails)
      (let ([result (flip-coin)])
        (display (if (eq? result 'h) "h " "t "))
        (if (= (abs (- (+ (if (eq? result 'h) 1 0) heads)
                       (+ (if (eq? result 't) 1 0) tails)))
               n)
            (displayln "")
  (flip-for-difference-helper (+ (if (eq? result 'h) 1 0) heads)
                          (+ (if (eq? result 't) 1 0) tails)))))
  (flip-for-difference-helper 0 0))
```

Demo

```
> (flip-for-difference 1)
t
> (flip-for-difference 1)
h
> (flip-for-difference 1)
t
> (flip-for-difference 1)
t
> (flip-for-difference 2)
h h
> (flip-for-difference 2)
h t h t h t h t t h h t t t
> (flip-for-difference 2)
t h t t
> (flip-for-difference 2)
h t t h h t t h h t t t
> (flip-for-difference 2)
h t t h h h
> (flip-for-difference 2)
t h t t
> (flip-for-difference 3)
t h h t t h t t t
> (flip-for-difference 3)
t h t h h t t h t t t
> (flip-for-difference 3)
t t t
> (flip-for-difference 3)
h h h
> (flip-for-difference 3)
t h t h h t h h h
> (flip-for-difference 3)
t h t t h t h h t h h t h h t h t h h
> (flip-for-difference 4)
h h t h t h h t h t t h t h h t t h t t h t t h t t h t h t h h h h h t t t t t t h t h t t
> (flip-for-difference 4)
t t t t
> (flip-for-difference 4)
t h t t h t h t h t h t h t t h t t
> (flip-for-difference 4)
h h t h t t h h h h
> (flip-for-difference 4)
h t h t h t h t t t h t t t
> (flip-for-difference 4)
t h t h h t h t h t h h t t t h h t h t t h t t t t
> (flip-for-difference 4)
t h t t h h t t h t h h t h t t t h h t h h h h h h
> (flip-for-difference 4)
h h h h
>
```

Task 4: Laying Down Colorful Concentric Disks

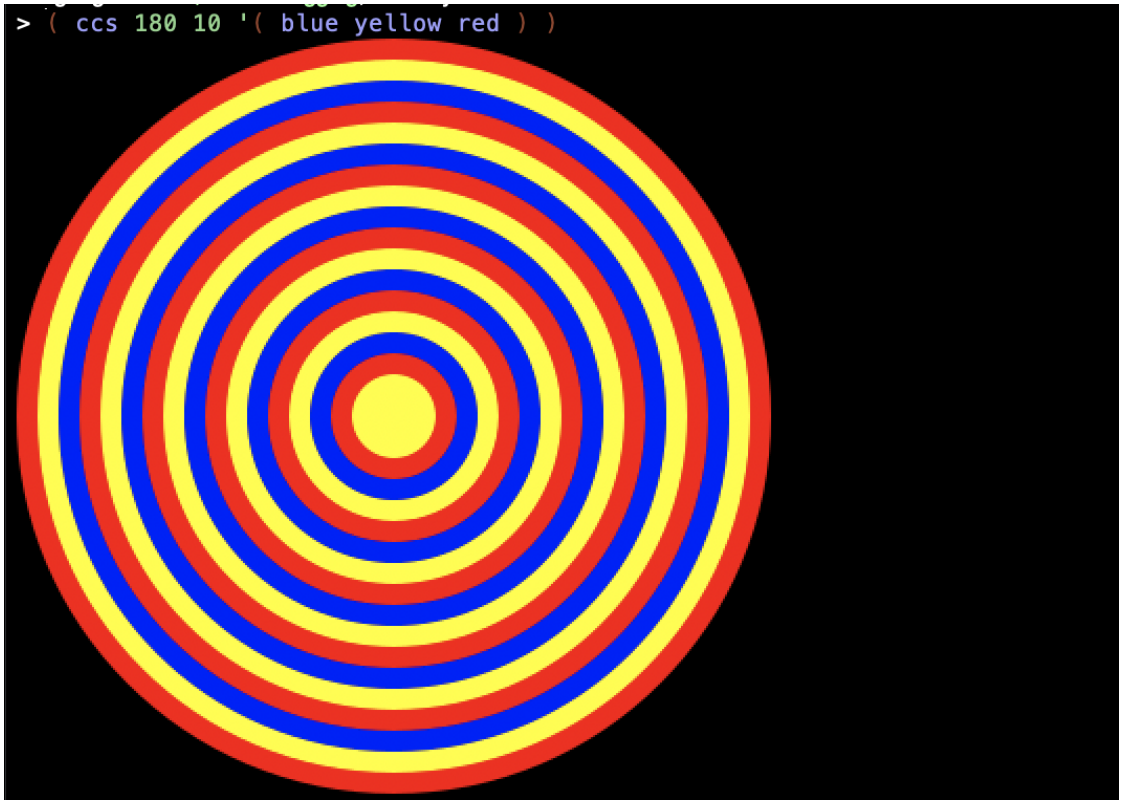
CCR Demo



CCA Demo


Language: racket, with debugging; memory limit: 128 MB.
> (cca 160 10 'black 'white)
> (cca 150 25 'red 'orange)
>

CCS Demo 1



CCS Demo 2

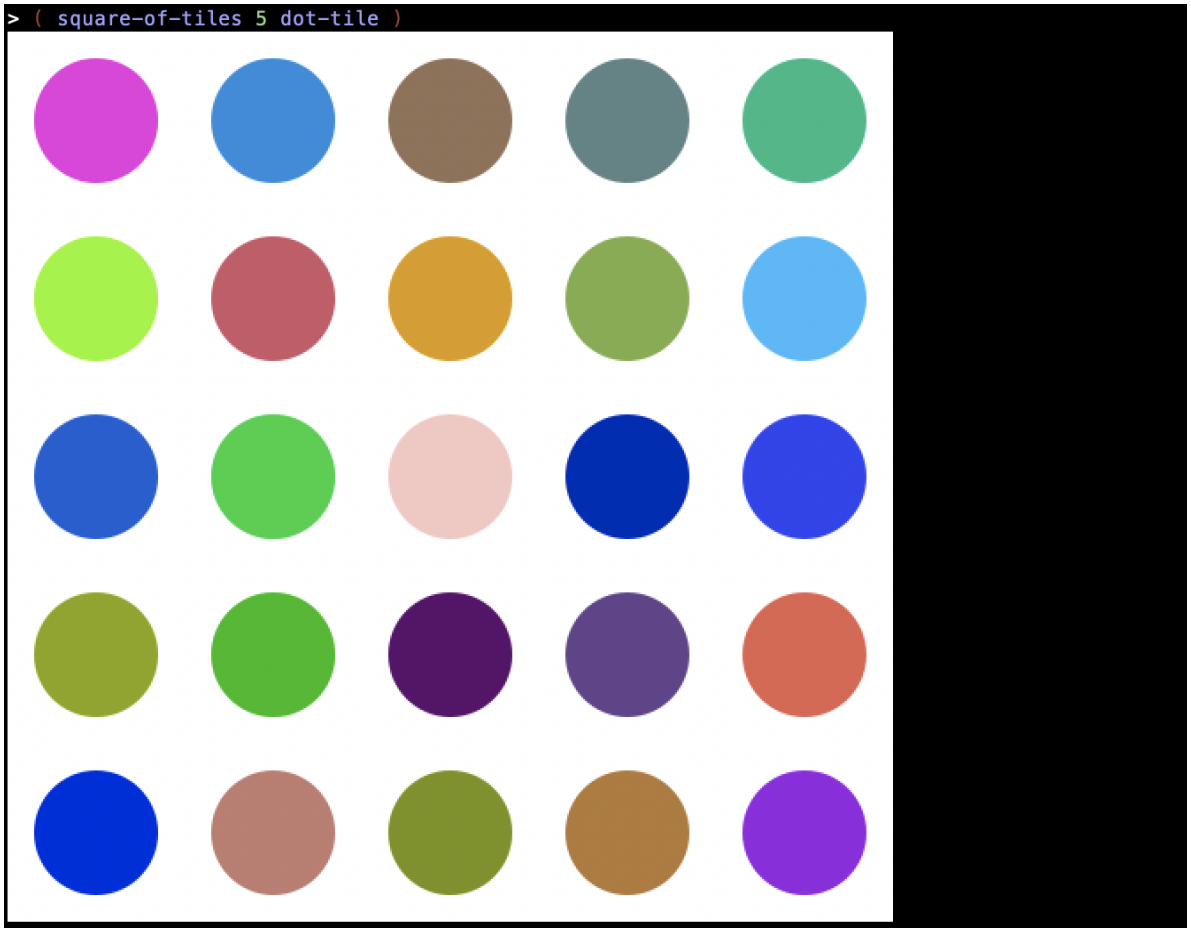
Code

```
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value) ) )
( define ( rgb-value ) ( random 256 ) )
( define ( ccr  n i )
    ( cond
       ( ( <= n 0 ) empty-image )
       ( ( > n 0 )
       ( overlay ( ccr ( - n i ) i )
                        ( circle n "solid" ( random-color ) ) ) ) ) )


(define (cca n i color1 color2)
  (cond
    ((<= n 0) empty-image)
    (else (overlay
            (cca (- n i) i color1 color2)
            (circle n "solid" (if (even? (round (/ n i))) color1 color2))
           ))) )

(define (ccs n i color-list)
  (cond
    ((<= n 0) empty-image)
    ((> n i)
     (overlay (ccs (- n i) i color-list)
         (circle n "solid" (list-ref color-list (modulo (- n i) (length color-list))))))
    (else empty-image)))
```

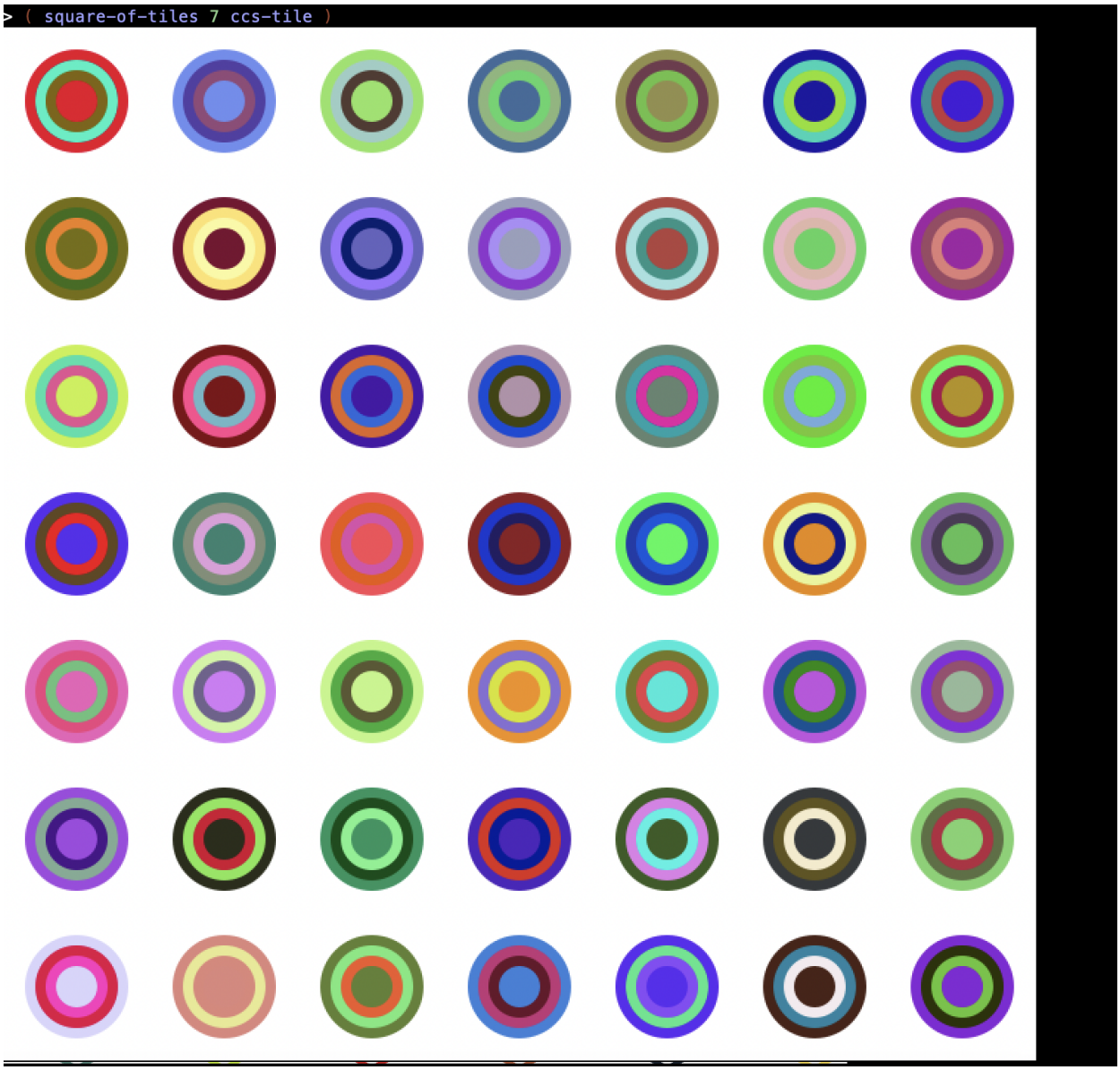
Task 5: Variations on Hirst Dots

Random Colored Tile Demo

Hirst Dots Demo

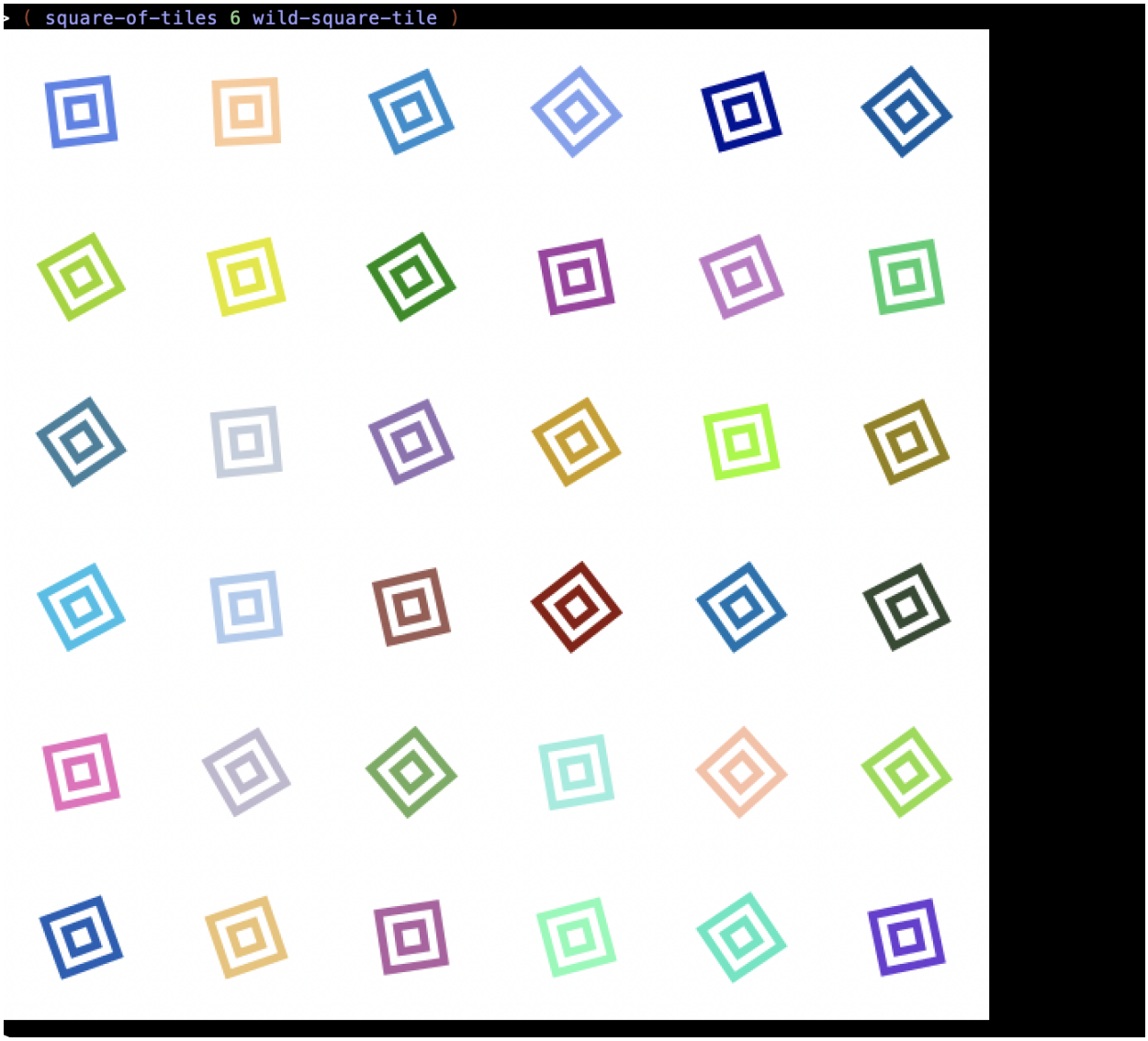


CCS Dots Demo



Nested Diamonds Demo

Unruly Squares Demo



Code

```
(define (ccs n i color-list)
  (cond
    ((<= n 0) empty-image)
    ((> n i)
     (overlay (ccs (- n i) i color-list)
              (circle n "solid" (list-ref color-list (modulo (-
n i) (length color-list))))))
    (else empty-image)))

( define ( rc n )
  ( cond
     ( ( > n 0 )
       ( cons ( random-color ) ( rc ( - n 1 ) ) )
      )
      ( ( = n 0)
         empty
      ) ) )


( define ( row-of-tiles n tile )
   ( cond
      ( ( = n 0 )
         empty-image )
      ( ( > n 0 )
        ( beside ( row-of-tiles ( - n 1 ) tile ) ( tile ) )
      )
     )
    )

   ( define ( rectangle-of-tiles r c tile )
      ( cond
         ( ( = r 0 )
           empty-image
```

```
        )
        ( ( > r 0 )
          ( above
            ( rectangle-of-tiles ( - r 1 ) c tile ) (
row-of-tiles c tile ) )
          )))



   ( define ( square-of-tiles n tile )
      ( rectangle-of-tiles n n tile )
      )


   ( define ( random-color-tile )
      ( overlay
        ( square 40 "outline" "black" )
        ( square 40 "solid" ( random-color) ) ) )


   ( define ( random-color )
      ( define (rgb-value) (random 256 ) )
        ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )


( define ( dot-tile )
   ( overlay
     ( circle 35 "solid" (random-color ) )
     ( square 100 "solid" "white" ) ) )




( define ( ccs-tile )
   ( define color ( rc 3 ) )
   ( overlay
     ( ccs 35 7 color )
```

```
     ( square 100 "solid" "white" ) ) )

 ( define ( diamond-tile )
    ( define diaColor ( random-color ) )
     ( overlay
       ( rotate 45 ( square 10 "solid" "white" ) )
       ( rotate 45 ( square 20 "solid" diaColor ) )
       ( rotate 45 ( square 30 "solid" "white" ) )
       ( rotate 45 ( square 40 "solid" diaColor ) )
       ( square 100 "solid" "white" ) ) )


( define (wild-square-tile )
   ( define sqColor ( random-color ) )
   ( define angle ( random 0 45 ) )
   ( overlay
       ( rotate angle ( square 10 "solid" "white" ) )
       ( rotate angle ( square 20 "solid" sqColor ) )
       ( rotate angle ( square 30 "solid" "white" ) )
       ( rotate angle ( square 40 "solid" sqColor ) )
       ( square 100 "solid" "white" ) ) )
```