JD Vargas

Racket Programming Assignment #2: Racket Functions and Recursion

— —

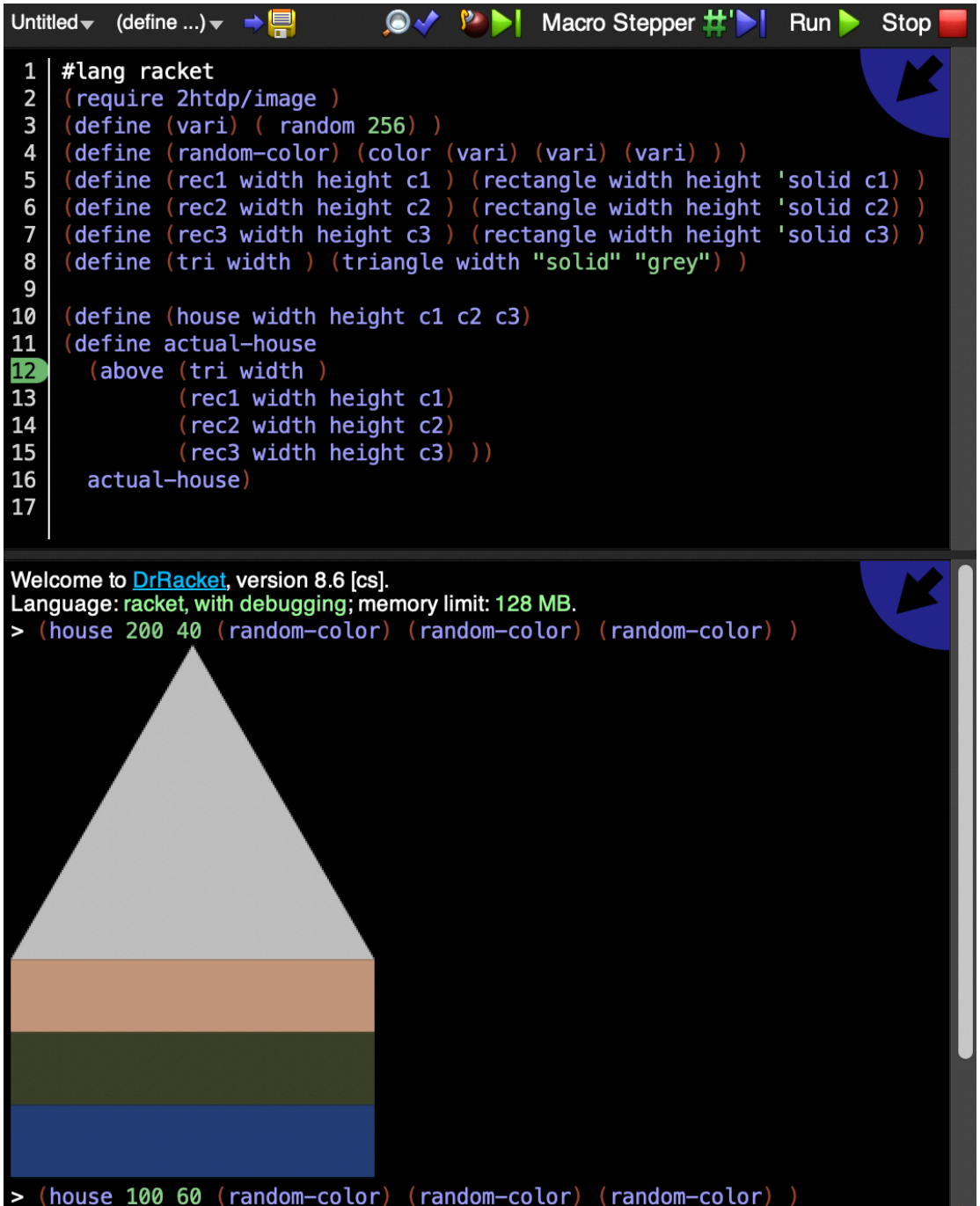
Learning Abstract:

During this assignment I learned how to use recursion to generate different types of images from the 2htdp/image library and put them together.

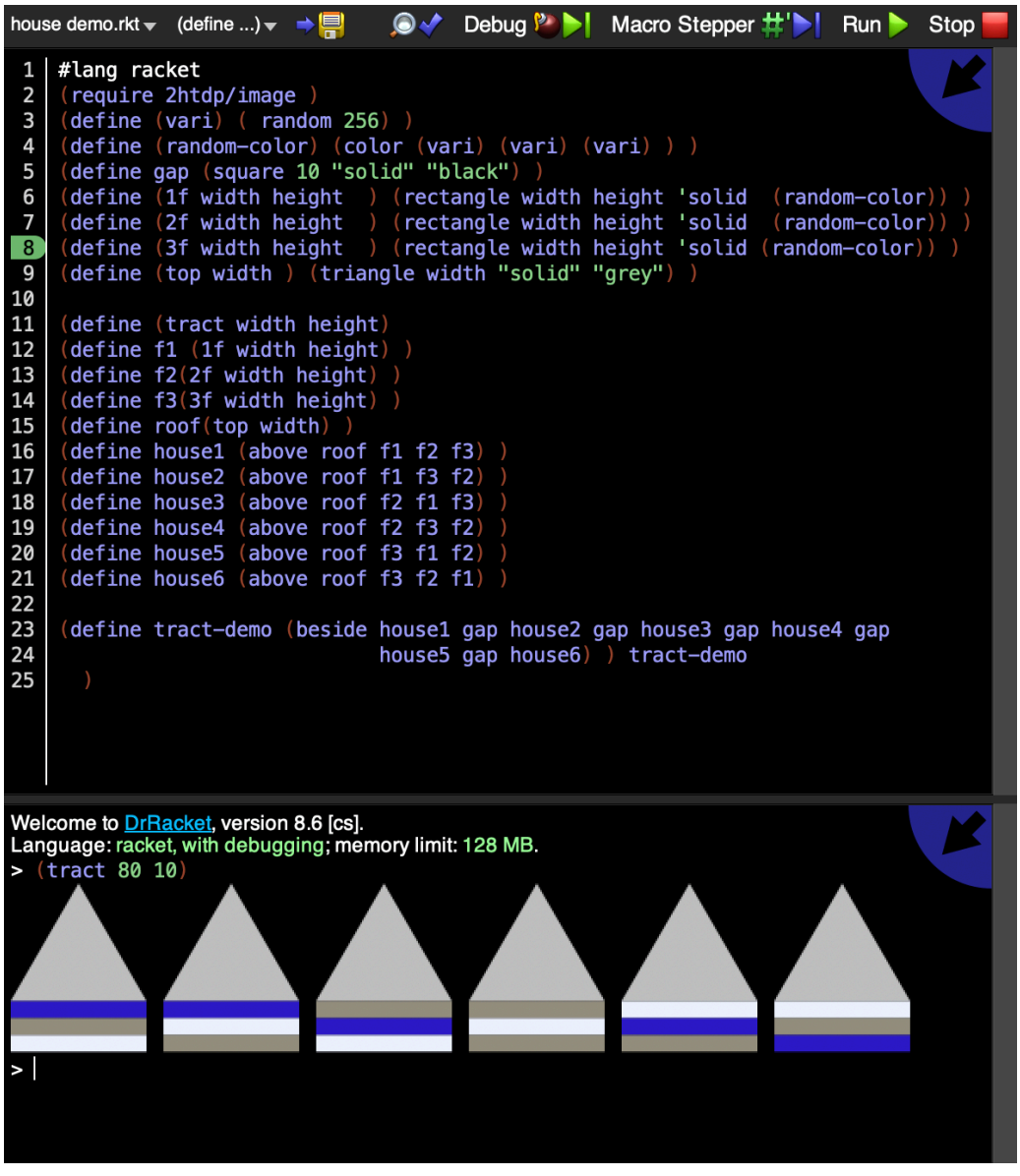
— —

Task 1: Colorful Permutations of Tract Houses

Demo/code for house:

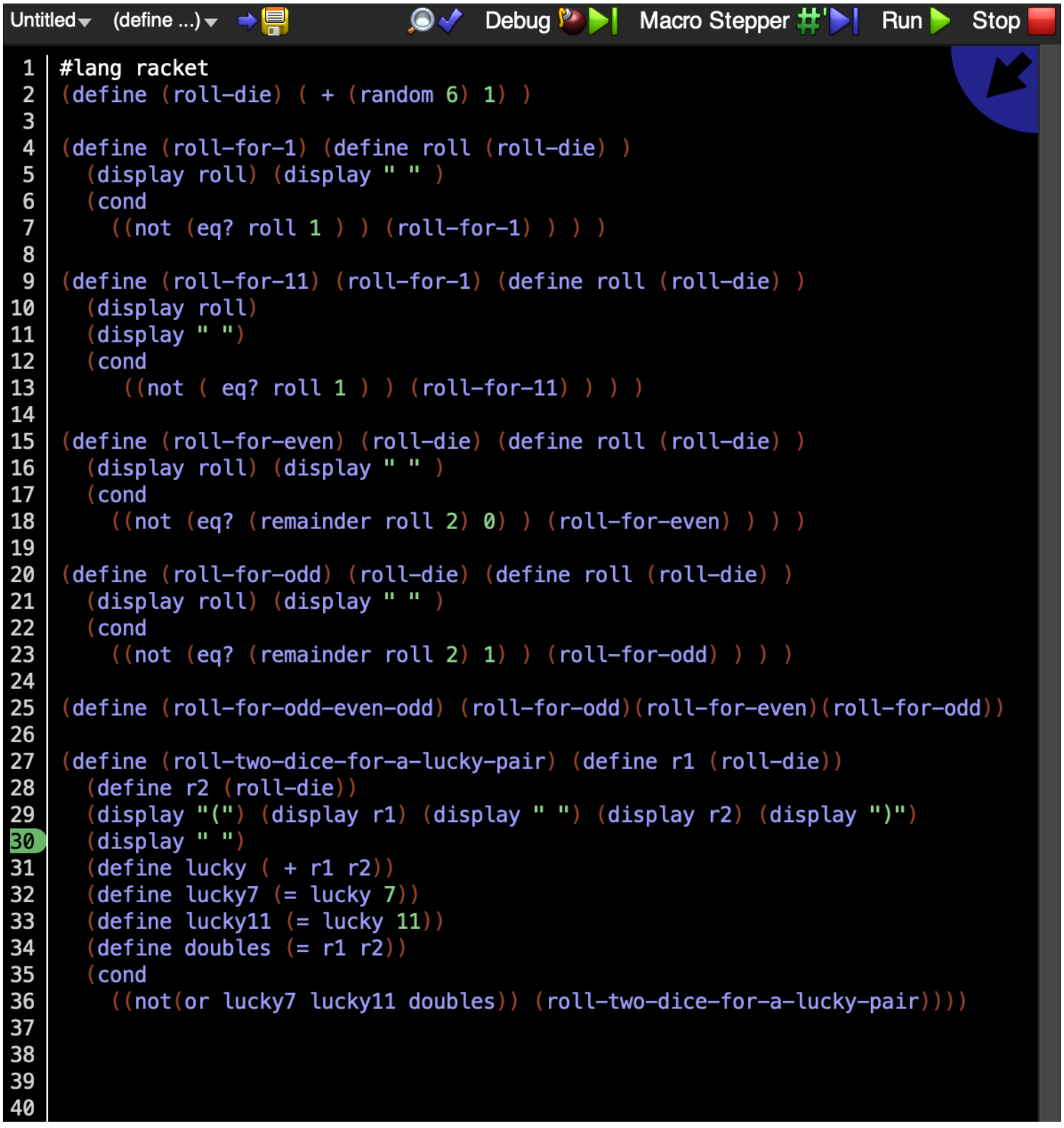


Demo/code for tract:



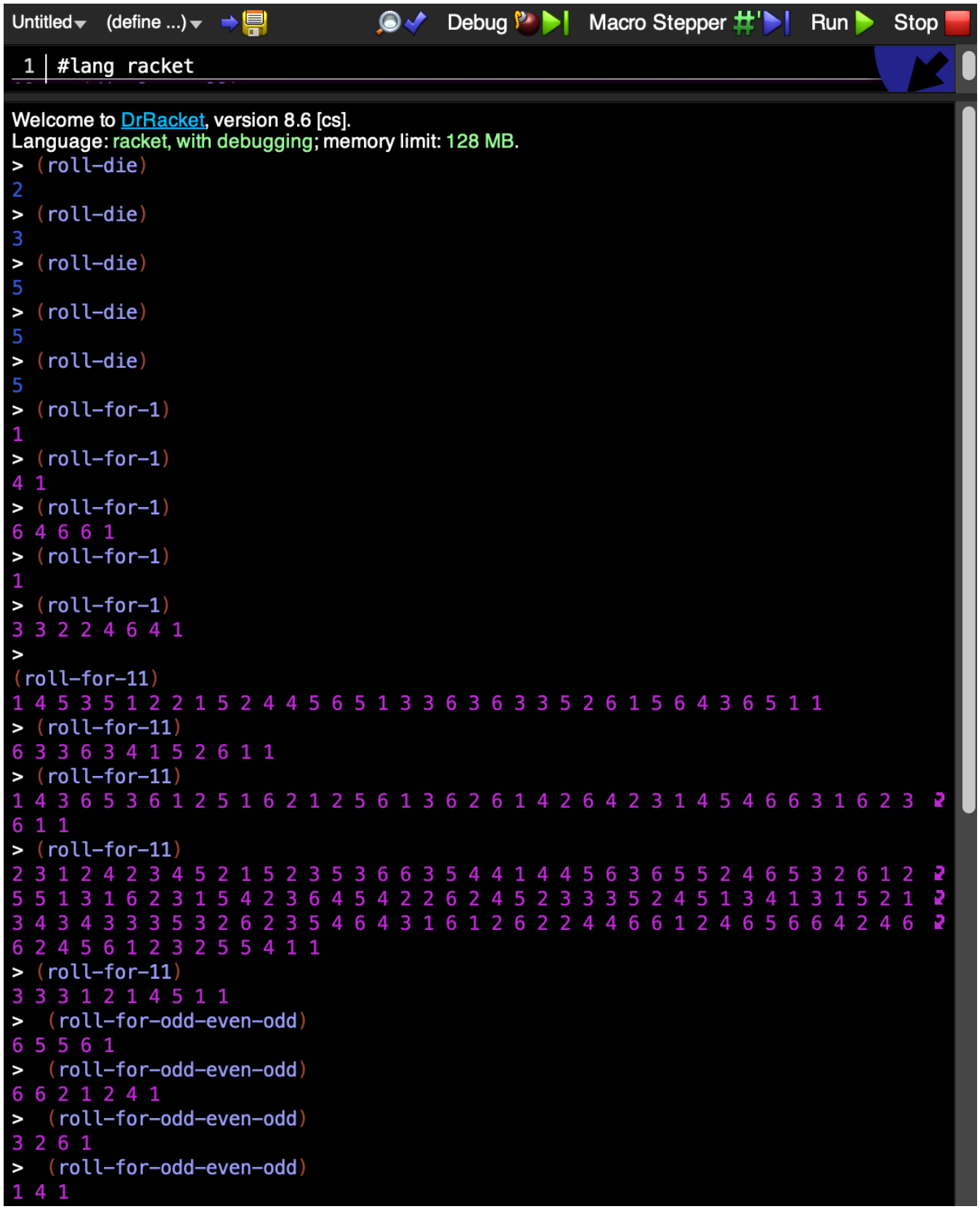
Task 2: Dice

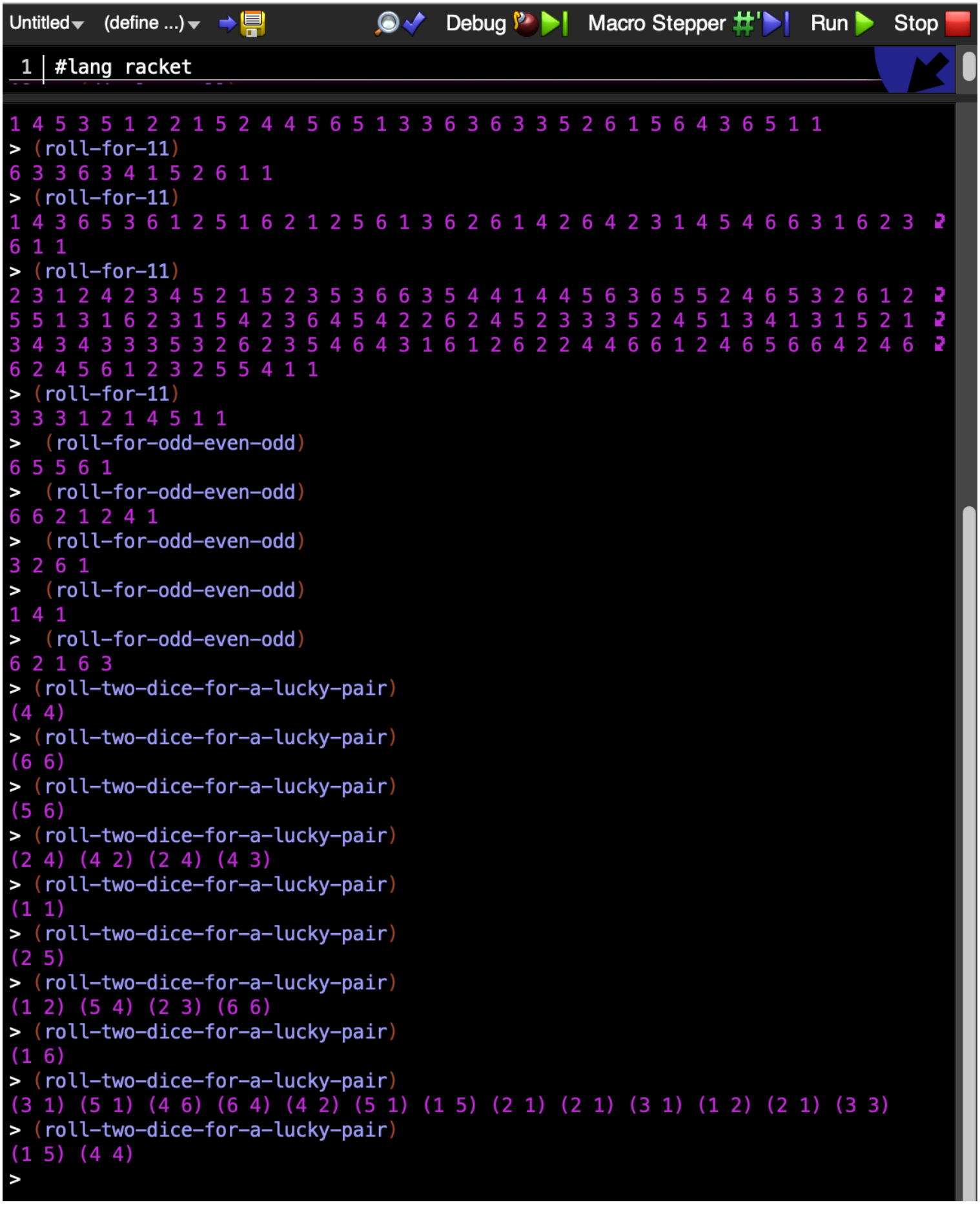
Code:




```
#lang racket
(define (roll-die) ( + (random 6) 1) )

(define (roll-for-1) (define roll (roll-die) )
  (display roll) (display " " )
  (cond
    ((not (eq? roll 1 ) ) (roll-for-1) ) ) )

(define (roll-for-11) (roll-for-1) (define roll (roll-die) )
  (display roll)
  (display " ")
  (cond
     ((not ( eq? roll 1 ) ) (roll-for-11) ) ) )

(define (roll-for-even) (roll-die) (define roll (roll-die) )
  (display roll) (display " " )
  (cond
    ((not (eq? (remainder roll 2) 0) ) (roll-for-even) ) ) )

(define (roll-for-odd) (roll-die) (define roll (roll-die) )
  (display roll) (display " " )
  (cond
    ((not (eq? (remainder roll 2) 1) ) (roll-for-odd) ) ) )

(define (roll-for-odd-even-odd) (roll-for-odd)(roll-for-even)(roll-for-odd))

(define (roll-two-dice-for-a-lucky-pair) (define r1 (roll-die))
  (define r2 (roll-die))
  (display "(") (display r1) (display " ") (display r2) (display ")")
  (display " ")
  (define lucky ( + r1 r2))
  (define lucky7 (= lucky 7))
  (define lucky11 (= lucky 11))
  (define doubles (= r1 r2))
  (cond
    ((not(or lucky7 lucky11 doubles)) (roll-two-dice-for-a-lucky-pair))))
```

Demo:


Untitled
(define ...)
Debug
Macro Stepper
Run
Stop
1 #lang racket
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (roll-die)
2
> (roll-die)
3
> (roll-die)
5
> (roll-die)
5
> (roll-die)
5
> (roll-for-1)
1
> (roll-for-1)
4 1
> (roll-for-1)
6 4 6 6 1
> (roll-for-1)
1
> (roll-for-1)
3 3 2 2 4 6 4 1
>
(roll-for-11)
1 4 5 3 5 1 2 2 1 5 2 4 4 5 6 5 1 3 3 6 3 6 3 3 5 2 6 1 5 6 4 3 6 5 1 1
> (roll-for-11)
6 3 3 6 3 4 1 5 2 6 1 1
> (roll-for-11)
1 4 3 6 5 3 6 1 2 5 1 6 2 1 2 5 6 1 3 6 2 6 1 4 2 6 4 2 3 1 4 5 4 6 6 3 1 6 2 3
6 1 1
> (roll-for-11)
2 3 1 2 4 2 3 4 5 2 1 5 2 3 5 3 6 6 3 5 4 4 1 4 4 5 6 3 6 5 5 2 4 6 5 3 2 6 1 2
5 5 1 3 1 6 2 3 1 5 4 2 3 6 4 5 4 2 2 6 2 4 5 2 3 3 3 5 2 4 5 1 3 4 1 3 1 5 2 1
3 4 3 4 3 3 3 5 3 2 6 2 3 5 4 6 4 3 1 6 1 2 6 2 2 4 4 6 6 1 2 4 6 5 6 6 4 2 4 6
6 2 4 5 6 1 2 3 2 5 5 4 1 1
> (roll-for-11)
3 3 3 1 2 1 4 5 1 1
> (roll-for-odd-even-odd)
6 5 5 6 1
> (roll-for-odd-even-odd)
6 6 2 1 2 4 1
> (roll-for-odd-even-odd)
3 2 6 1
> (roll-for-odd-even-odd)
1 4 1


Untitled
(define ...)
Debug
Macro Stepper
Run
Stop
1 #lang racket
1 4 5 3 5 1 2 2 1 5 2 4 4 5 6 5 1 3 3 6 3 6 3 3 5 2 6 1 5 6 4 3 6 5 1 1
> (roll-for-11)
6 3 3 6 3 4 1 5 2 6 1 1
> (roll-for-11)
1 4 3 6 5 3 6 1 2 5 1 6 2 1 2 5 6 1 3 6 2 6 1 4 2 6 4 2 3 1 4 5 4 6 6 3 1 6 2 3
6 1 1
> (roll-for-11)
2 3 1 2 4 2 3 4 5 2 1 5 2 3 5 3 6 6 3 5 4 4 1 4 4 5 6 3 6 5 5 2 4 6 5 3 2 6 1 2
5 5 1 3 1 6 2 3 1 5 4 2 3 6 4 5 4 2 2 6 2 4 5 2 3 3 3 5 2 4 5 1 3 4 1 3 1 5 2 1
3 4 3 4 3 3 3 5 3 2 6 2 3 5 4 6 4 3 1 6 1 2 6 2 2 4 4 6 6 1 2 4 6 5 6 6 4 2 4 6
6 2 4 5 6 1 2 3 2 5 5 4 1 1
> (roll-for-11)
3 3 3 1 2 1 4 5 1 1
> (roll-for-odd-even-odd)
6 5 5 6 1
> (roll-for-odd-even-odd)
6 6 2 1 2 4 1
> (roll-for-odd-even-odd)
3 2 6 1
> (roll-for-odd-even-odd)
1 4 1
> (roll-for-odd-even-odd)
6 2 1 6 3
> (roll-two-dice-for-a-lucky-pair)
(4 4)
> (roll-two-dice-for-a-lucky-pair)
(6 6)
> (roll-two-dice-for-a-lucky-pair)
(5 6)
> (roll-two-dice-for-a-lucky-pair)
(2 4) (4 2) (2 4) (4 3)
> (roll-two-dice-for-a-lucky-pair)
(1 1)
> (roll-two-dice-for-a-lucky-pair)
(2 5)
> (roll-two-dice-for-a-lucky-pair)
(1 2) (5 4) (2 3) (6 6)
> (roll-two-dice-for-a-lucky-pair)
(1 6)
> (roll-two-dice-for-a-lucky-pair)
(3 1) (5 1) (4 6) (6 4) (4 2) (5 1) (1 5) (2 1) (2 1) (3 1) (1 2) (2 1) (3 3)
> (roll-two-dice-for-a-lucky-pair)
(1 5) (4 4)
>

Task 3: Number Sequences

Code:

Untitled (define ...) Debug Macro Stepper Run Stop

```
#lang racket
( define ( square n )
(* n n)
)
( define ( cube n )
(* n n n)
)


(define (triangular n)
  (cond
    ((= n 1) 1)
    ((= n 2) 3)
    ((> n 2)
     (+ (triangular (- n 1)) n ))))

( define ( sequence triangular n )
( cond
((= n 1)
( display ( triangular 1 ) ) ( display " " )
)
( else
( sequence triangular ( - n 1 ) )
( display ( triangular n ) ) ( display " " )
)
)
)

(define (sigma n) (math n n))
(define (math n orig)
  (cond
    ((= n 1) 1) ((= (modulo orig n) 0) (+ n(math(- n 1) orig)))
  (else (math(- n 1) orig))))

( define ( sequencee sigma n )
( cond
((= n 1)
( display ( sigma 1 ) ) ( display " " )
)
( else
( sequence sigma ( - n 1 ) )
( display ( sigma n ) ) ( display " " ))))
```

Demo:

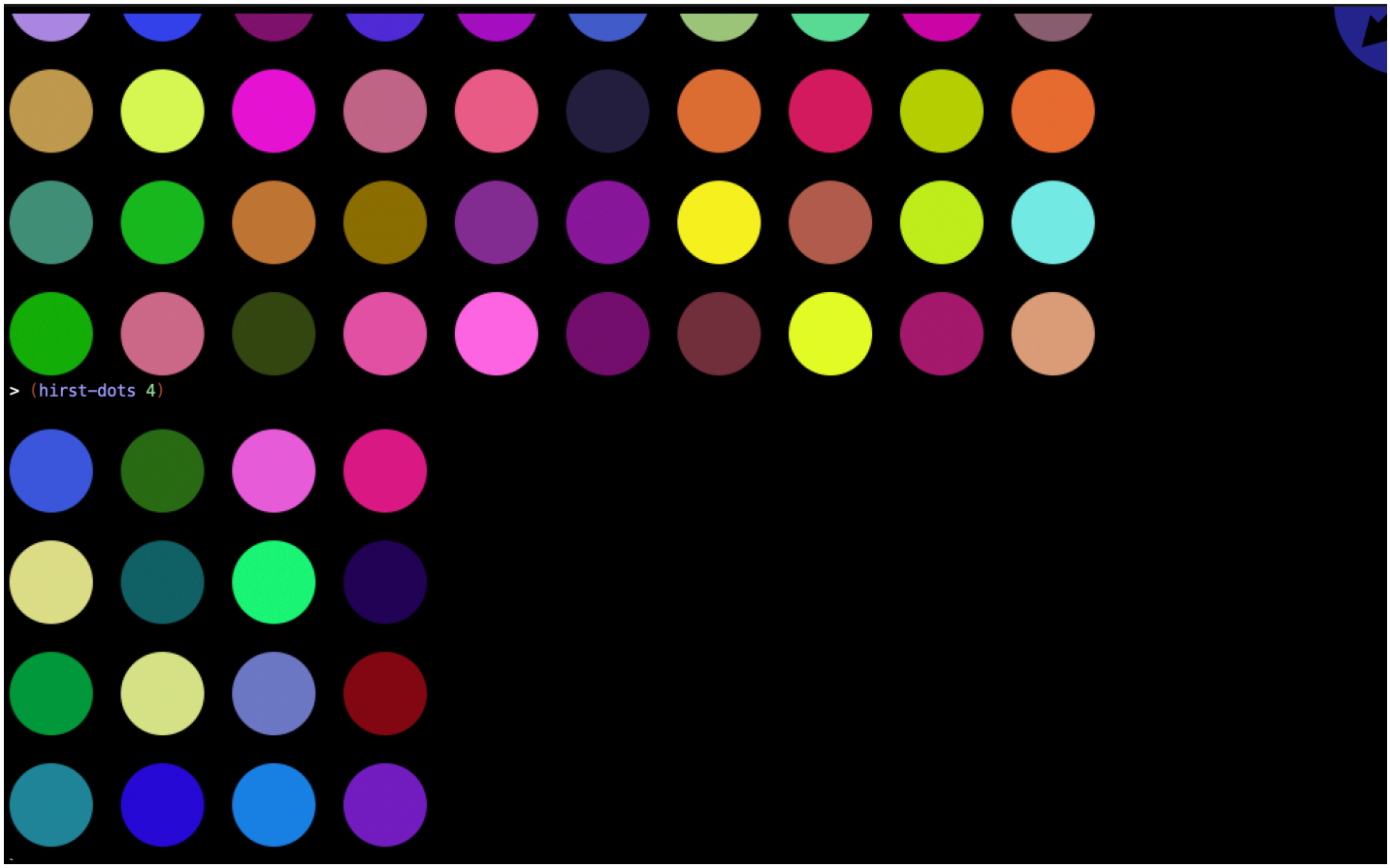
```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (square 5)
25
> (square 10)
100
> (sequence square 15)
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225
> (cube 2 )
8
> (cube 3)
27
> (sequence cube 15)
1 8 27 64 125 216 343 512 729 1000 1331 1728 2197 2744 3375
> (triangular 1)
1
> (triangular 2)
3
> (triangular 3 )
6
> (triangular 4)
10
> (triangular 5)
15
> (sequence triangular 20)
1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210
> (sigma 1 )
1
> (sigma 2)
3
> (sigma 3)
4
> (sigma 4)
7
> (sigma 5)
6
> (sequencee sigma 20)
1 3 4 7 6 12 8 15 13 18 12 28 14 24 24 31 18 39 20 42
>
```

Task 4: Hirst Dots

Code:

```
#lang racket
(require 2htdp/image)
(define (rgb-value) (random 256))
(define (random-color) (color (rgb-value) (rgb-value) (rgb-value)))
(define gap (square 20 "solid" "black"))
(define diameter 30)
(define (dot diameter color)(circle diameter 'solid (random-color)))

(define (row r)
  (cond
    ((= r 0) empty-image)((> r 0)(beside (row (- r 1))(dot diameter color)
gap))))

(define (column m n )
(cond
  ((= m 0) empty-image)((> m 0)(above (column (- m 1) n) gap (row n)))))

(define (hirst-dots r)(column r r))
```

Demo:

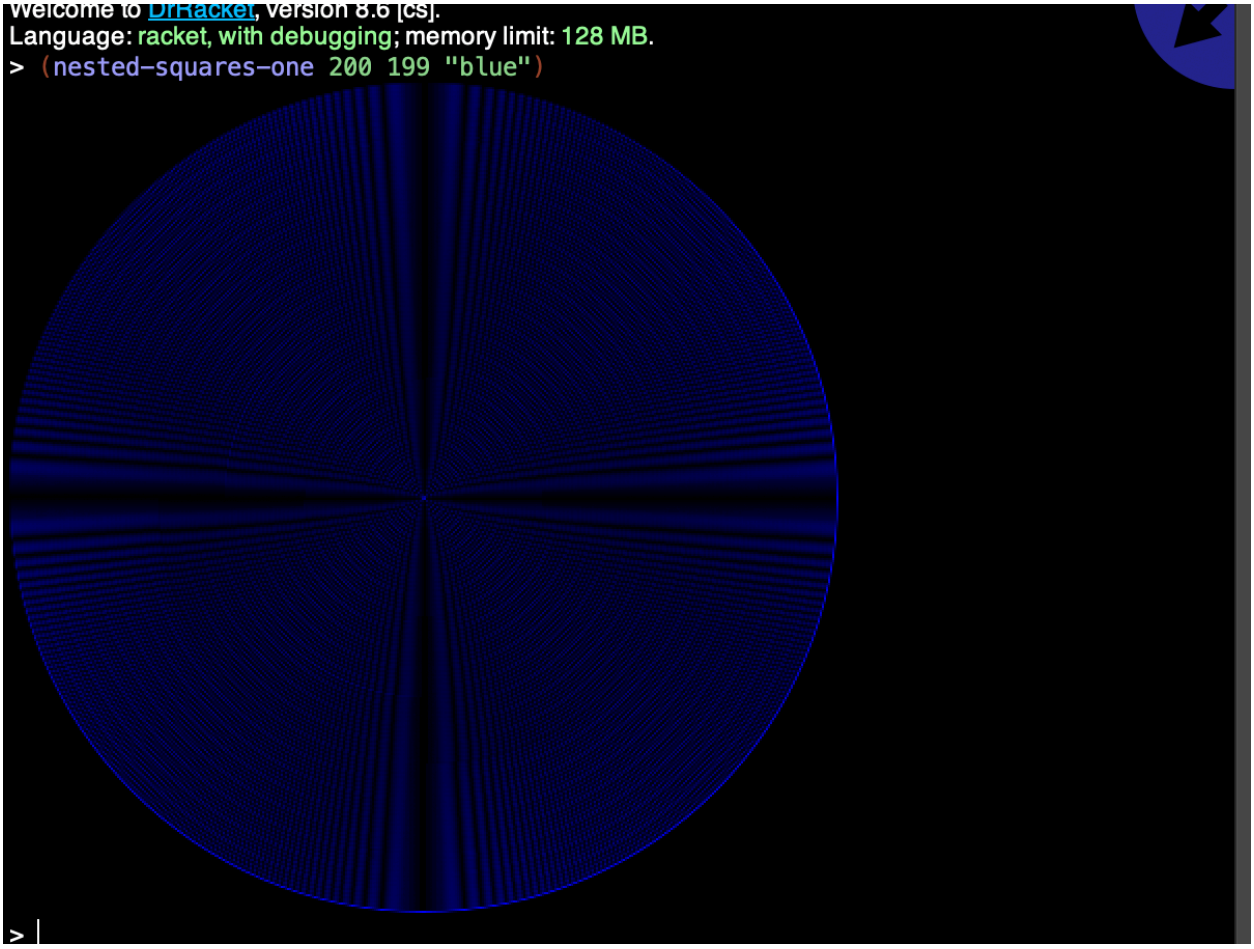


Task 5: Chanelling Frank Stella

Code:

```
#lang racket
( require 2htdp/image )
( define ( nested-squares-one side count color )
( define unit ( / side count ) )
( paint-nested-squares-one 1 count unit color )
)
( define ( paint-nested-squares-one from to unit color)
( define side-length ( * from unit ) )
( cond
( ( = from to )
( framed-square side-length color )
)
( ( < from to )
( overlay
( framed-square side-length color )
( paint-nested-squares-one ( + from 1 ) to unit color )
)
)
)
)
( define ( framed-square side-length color )
( overlay
( circle side-length "outline" "black" )
( circle side-length "solid" color )
)
)
```

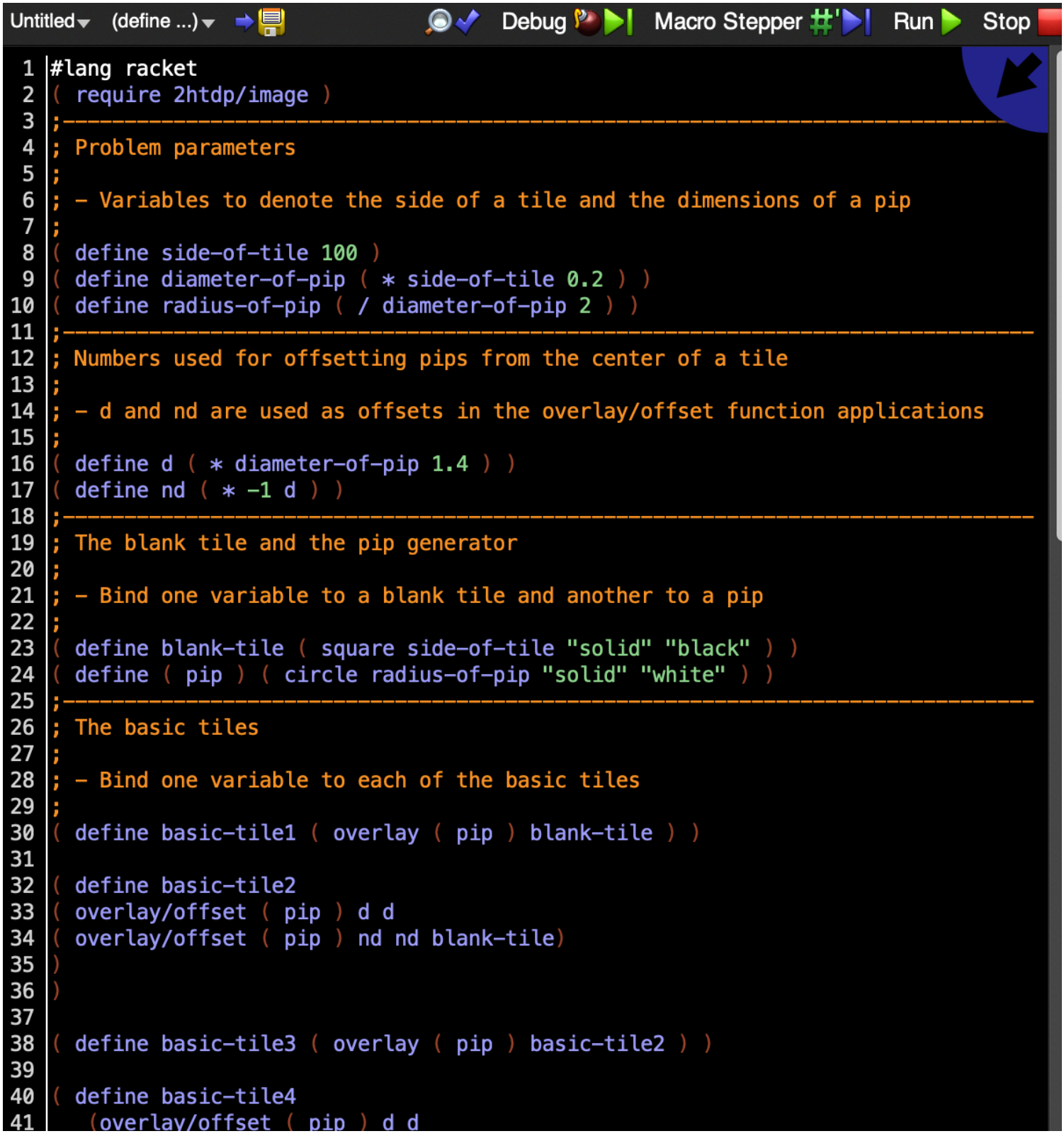
Demo:


Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (nested-squares-one 200 199 "blue")
>

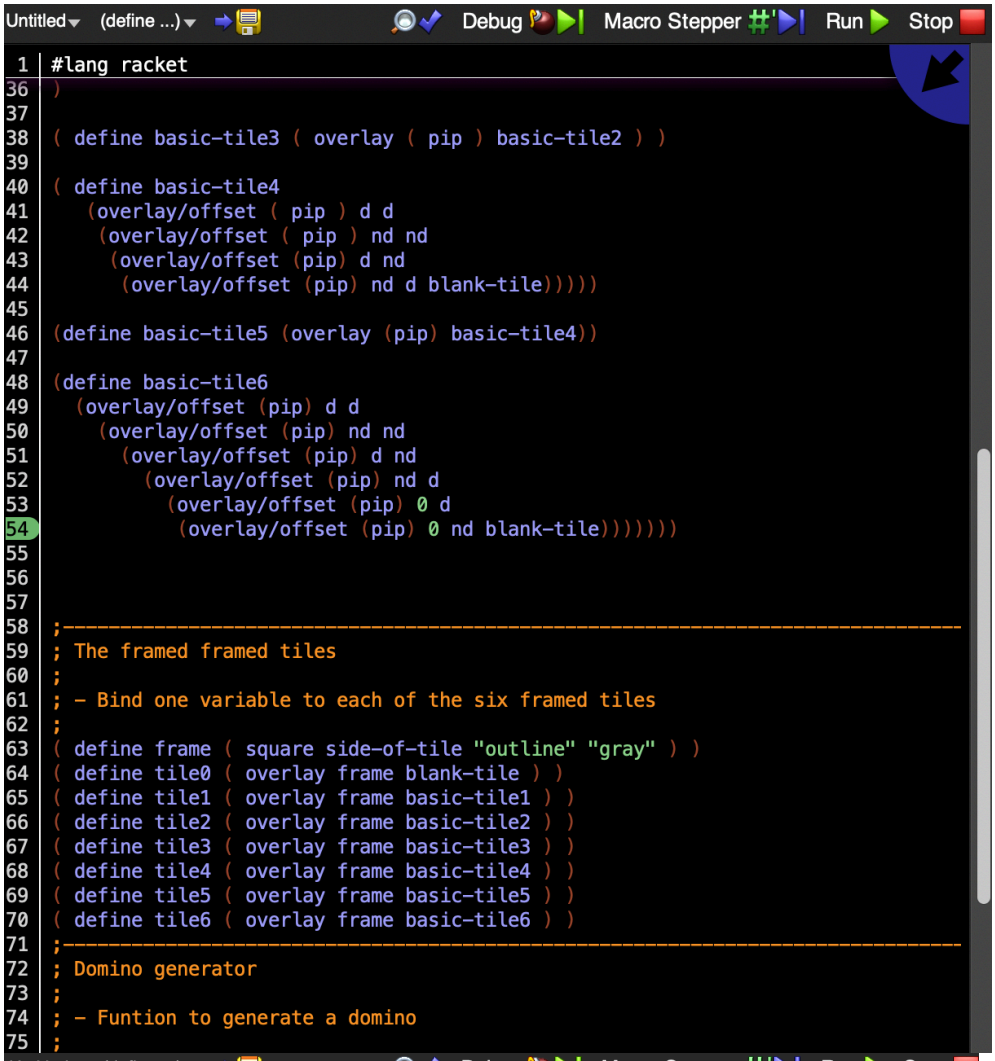

> (nested-squares-one 100 101 "green")

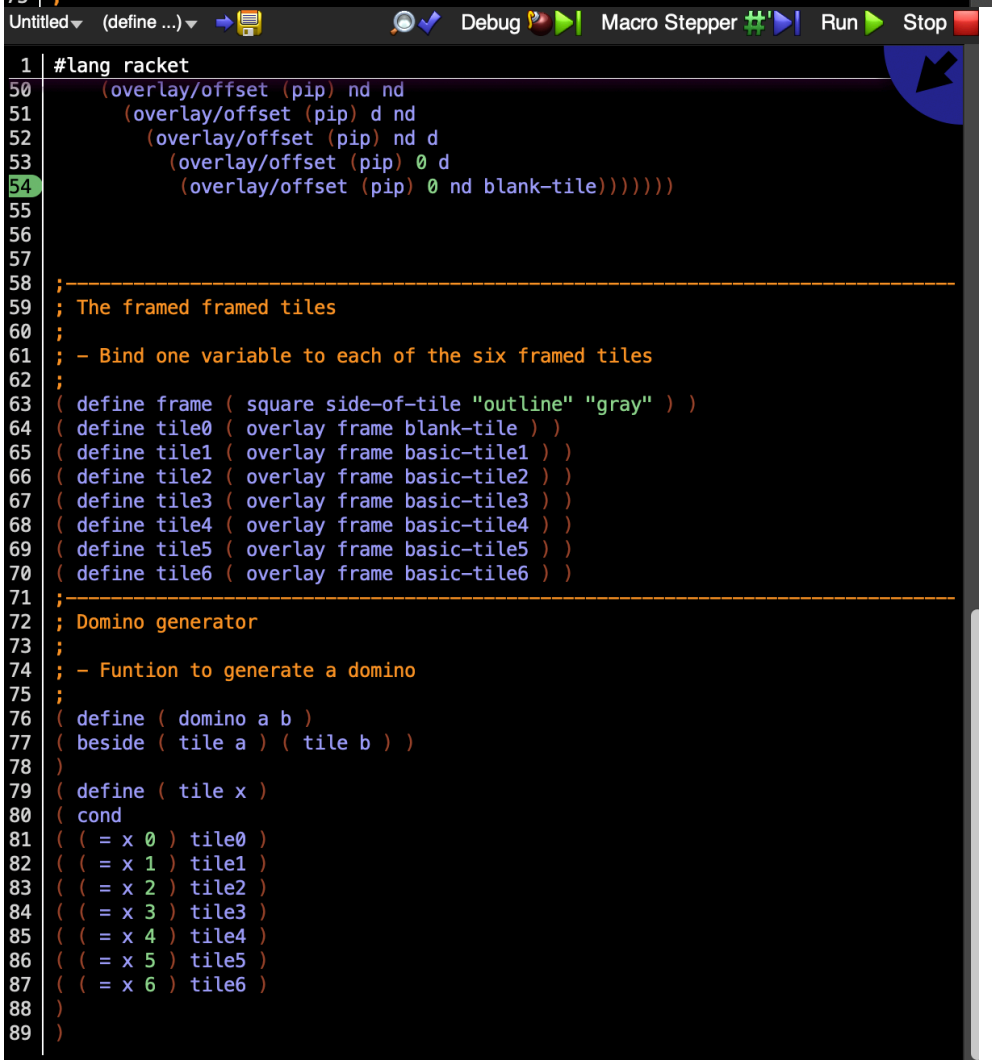
Task 6: Dominos

Code:

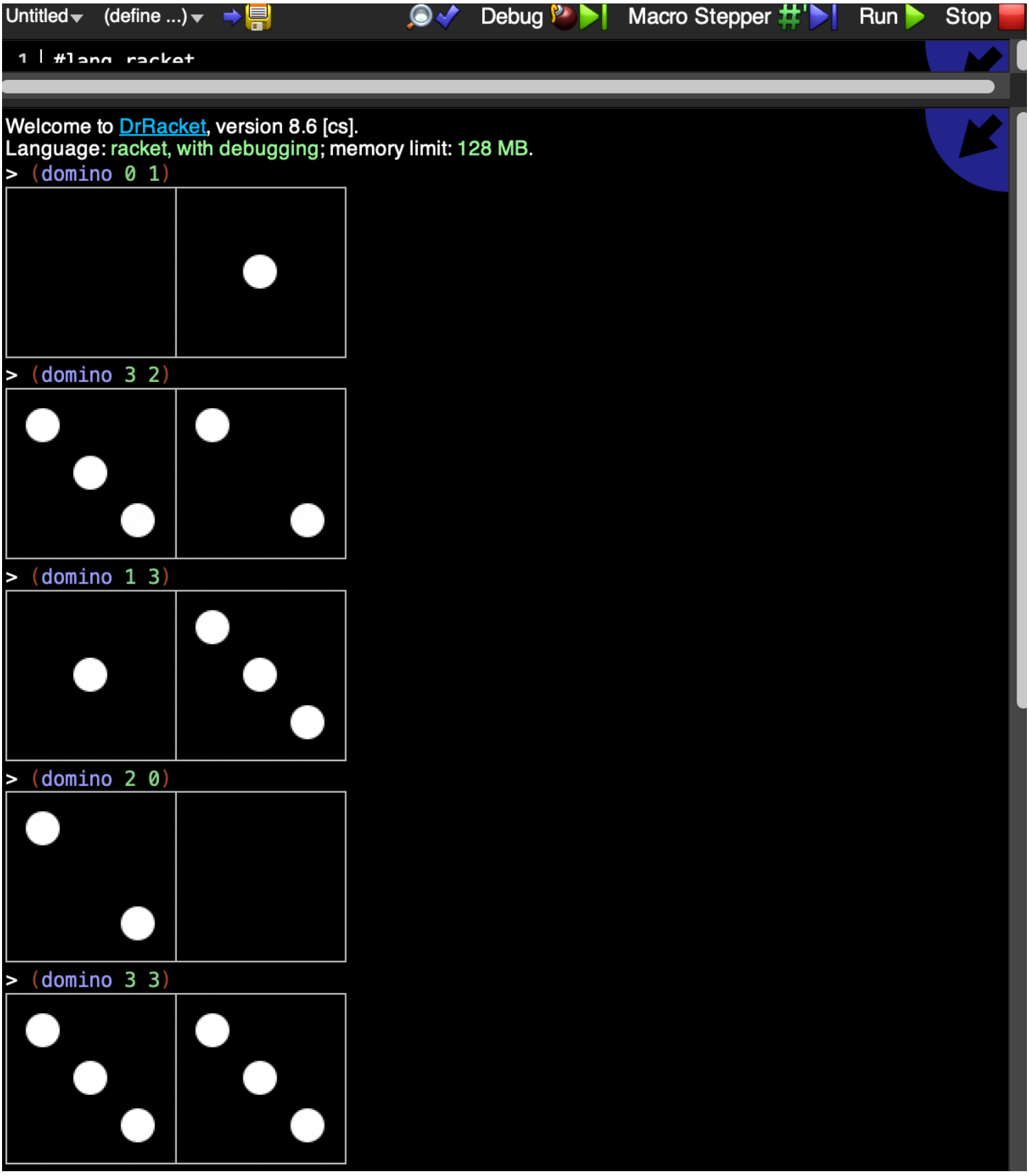



```
#lang racket
( require 2htdp/image )
;-----------------------------------------------------------------------
; Problem parameters
;
; - Variables to denote the side of a tile and the dimensions of a pip
;
( define side-of-tile 100 )
( define diameter-of-pip ( * side-of-tile 0.2 ) )
( define radius-of-pip ( / diameter-of-pip 2 ) )
;-----------------------------------------------------------------------
; Numbers used for offsetting pips from the center of a tile
;
; - d and nd are used as offsets in the overlay/offset function applications
;
( define d ( * diameter-of-pip 1.4 ) )
( define nd ( * -1 d ) )
;-----------------------------------------------------------------------
; The blank tile and the pip generator
;
; - Bind one variable to a blank tile and another to a pip
;
( define blank-tile ( square side-of-tile "solid" "black" ) )
( define ( pip ) ( circle radius-of-pip "solid" "white" ) )
;-----------------------------------------------------------------------
; The basic tiles
;
; - Bind one variable to each of the basic tiles
;
( define basic-tile1 ( overlay ( pip ) blank-tile ) )

( define basic-tile2
( overlay/offset ( pip ) d d
( overlay/offset ( pip ) nd nd blank-tile)
)
)

( define basic-tile3 ( overlay ( pip ) basic-tile2 ) )

( define basic-tile4
  (overlay/offset ( pip ) d d
```


Untitled
(define ...)
Debug
Macro Stepper
Run
Stop
#lang racket
)
(define basic-tile3 (overlay (pip) basic-tile2))
(define basic-tile4
(overlay/offset (pip) d d
(overlay/offset (pip) nd nd
(overlay/offset (pip) d nd
(overlay/offset (pip) nd d blank-tile)))))
(define basic-tile5 (overlay (pip) basic-tile4))
(define basic-tile6
(overlay/offset (pip) d d
(overlay/offset (pip) nd nd
(overlay/offset (pip) d nd
(overlay/offset (pip) nd d
(overlay/offset (pip) 0 d
(overlay/offset (pip) 0 nd blank-tile)))))))
;---
; The framed framed tiles
;
; - Bind one variable to each of the six framed tiles
;
(define frame (square side-of-tile "outline" "gray"))
(define tile0 (overlay frame blank-tile))
(define tile1 (overlay frame basic-tile1))
(define tile2 (overlay frame basic-tile2))
(define tile3 (overlay frame basic-tile3))
(define tile4 (overlay frame basic-tile4))
(define tile5 (overlay frame basic-tile5))
(define tile6 (overlay frame basic-tile6))
;---
; Domino generator
;
; - Funtion to generate a domino
;

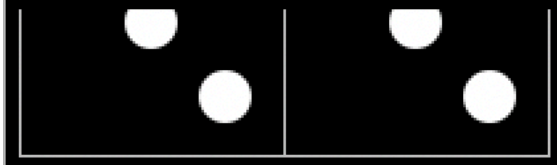

Untitled
(define ...)
Debug
Macro Stepper
Run
Stop
#lang racket
(overlay/offset (pip) nd nd
(overlay/offset (pip) d nd
(overlay/offset (pip) nd d
(overlay/offset (pip) 0 d
(overlay/offset (pip) 0 nd blank-tile)))))))
;---
; The framed framed tiles
;
; - Bind one variable to each of the six framed tiles
;
(define frame (square side-of-tile "outline" "gray"))
(define tile0 (overlay frame blank-tile))
(define tile1 (overlay frame basic-tile1))
(define tile2 (overlay frame basic-tile2))
(define tile3 (overlay frame basic-tile3))
(define tile4 (overlay frame basic-tile4))
(define tile5 (overlay frame basic-tile5))
(define tile6 (overlay frame basic-tile6))
;---
; Domino generator
;
; - Funtion to generate a domino
;
(define (domino a b)
(beside (tile a) (tile b))
)
(define (tile x)
(cond
((= x 0) tile0)
((= x 1) tile1)
((= x 2) tile2)
((= x 3) tile3)
((= x 4) tile4)
((= x 5) tile5)
((= x 6) tile6)
)
)

Demo:

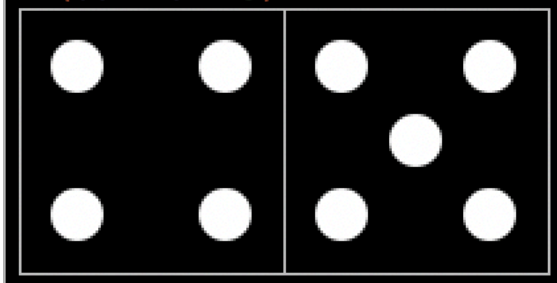

Untitled
(define ...)
Debug
Macro Stepper
Run
Stop
1 | #lang racket
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (domino 0 1)
> (domino 3 2)
> (domino 1 3)
> (domino 2 0)
> (domino 3 3)

Untitled (define ...) Debug Macro Stepper Run Stop

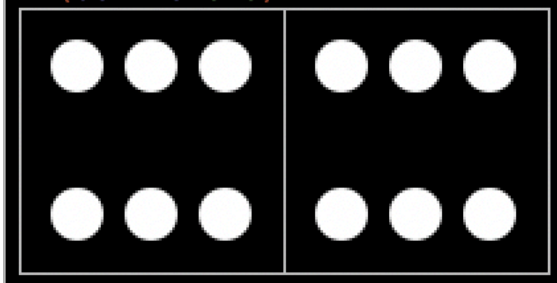
```
1 | #lang racket
```



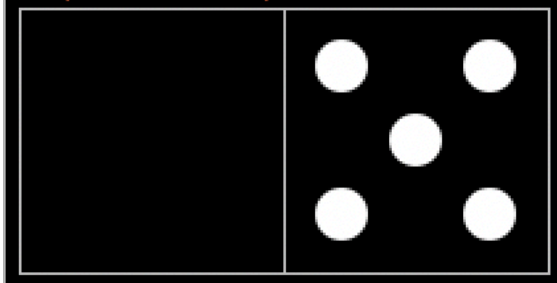
```
> (domino 4 5)
```



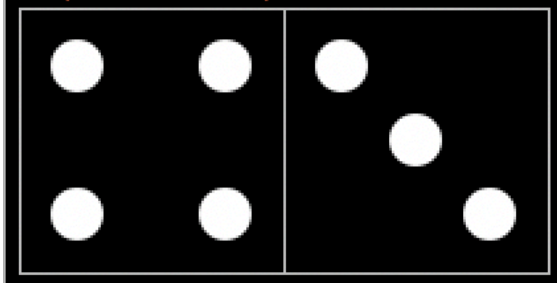
```
> (domino 6 6)
```



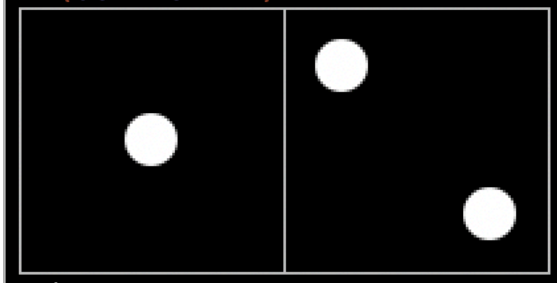
```
> (domino 0 5)
```



```
> (domino 4 3)
```



```
> (domino 1 2)
```



```
>
```

Task 7: Creation

Creation/Code:

Untitled (define ...) Check Syntax Debug Macro Stepper Run Stop

```
#lang racket
(require 2htdp/image)
(define diameter 500)
(define dot (circle diameter "solid" "yellow"))
(define o (star 200 "outline" "white"))
(define rec (wedge 100 100 "outline" "black"))
(define l (line 200 200 "yellow"))
(define tri (triangle 60 "outline" "red"))
(define li(triangle 20 "solid" "blue"))

(define (spin-alot t)
    (local [(define (spin-more i 0)
              (cond
                [(= 0 360) i]
                [else
                 (spin-more (overlay i (rotate 0 t))
                            (+ 0 1))]))]
      (spin-more t 0)))
```

```
Welcome to DrRacket, version 8.6 [cs].
Language: racket, with debugging; memory limit: 128 MB.
>  (overlay (spin-alot li) (overlay (spin-alot tri) (overlay(spin-alot rec) (overlay
(spin-alot l) (spin-alot o)))))
```