Racket Programming Assignment #2: Racket Functions and Recursion

Learning Abstract: This assignments contains programs which generate images using the 2htdp/images library, mostly using but not restricted to recursion.

The first assignment required using functions to generate a randomly colored three story house that size was adjustable. It also involves making a row of houses separated by a fixed distance between one another containing all the permutations of the three random colors used.

The second assignment required us to create several functions for dice rollings to achieve certain goals such as rolling until we get 1 consecutively or for a lucky pair.

The third assignment was a number sequence in which we squared, cubed and triangulated them. After which we made a code to do it sequentially using recursion.

The fourth assignment required us to created a pattern of colorful dots known as hirst dots with it accepting only one parameter to determine the amount of columns and rows.

The fifth assignment was a code to generate images based on frank Stella.

The sixth assignment required us to finish code that had been given to us half completed.

The final assignment was a freestyle of sorts where you created your own image based on methods and functions of your choosing.

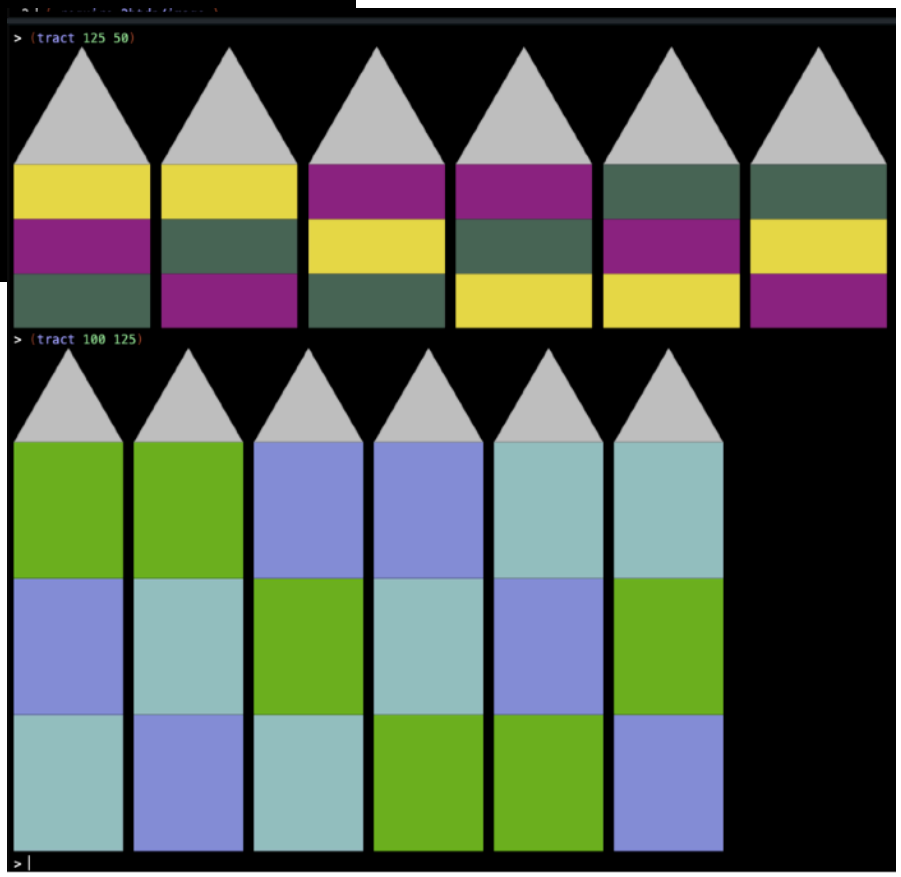
Task 1: Colorful Permutations of Tract Houses

```
3
4  ( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )
5  ( define ( rgb-value ) ( random 256 ) )
6
7  (define (house width length color1 color2 color3)
8    (above (roof width)(floor3 width length color3) (floor2 width length color2) (floor1 width length color1)))
9
10
11 (define ( floor3 width length color) ( rectangle width length "solid" color) )
12 (define ( floor2 width length color) ( rectangle width length "solid" color) )
13 (define ( floor1 width length color)( rectangle width length "solid"  color) )
14 (define ( roof width ) ( triangle width "solid" "gray" ))
15
```

House code and demo

```
(define ( floor1 width length )( rectangle width length "solid"  (random-color) ) )
(define ( floor2 width length )( rectangle width length "solid"  (random-color) ) )
(define ( floor3 width length )( rectangle width length "solid"  (random-color) ) )

(define space (square 10 "solid" "black"))
(define ( roof width ) ( triangle width "solid" "gray" ))
(define (tract width length)
  (define rooof(roof width))
  (define topfloor(floor3 width length))
  (define midfloor(floor2 width length))
  (define bottomfloor(floor1 width length))
  (define house1 (above
                  rooof
                  topfloor
                  midfloor
                  bottomfloor))
  (define house2 (above
                  rooof
                  topfloor
                  bottomfloor
                  midfloor))
  (define house3 (above
                  rooof
                  midfloor
                  topfloor
                  bottomfloor))
  (define house4 (above
                  rooof
                  midfloor
                  bottomfloor
                  topfloor))
  (define house5 (above
                  rooof
                  bottomfloor
                  midfloor
                  topfloor))
  (define house6 (above
                  rooof
                  bottomfloor
                  topfloor
                  midfloor))
  (define built-tract ( beside house1 space
                              house2 space
                              house3 space
                              house4 space
                              house5 space
                              house6))
  built-tract
```

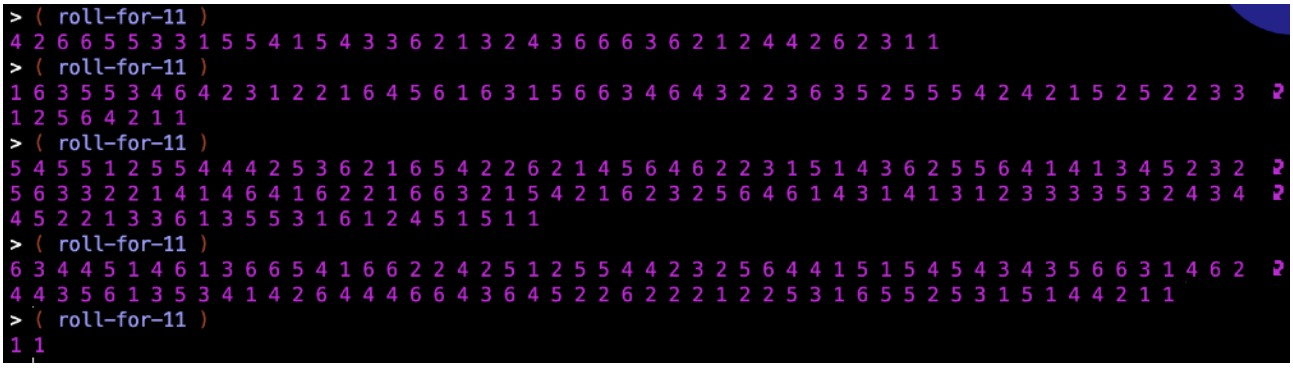
Tract Code



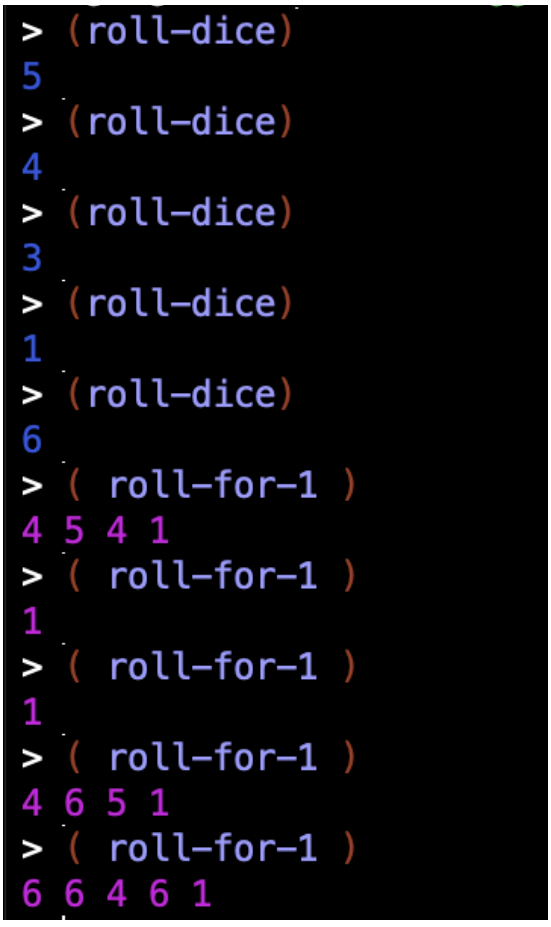
Disclaimer: I could not copy his demo for the tract as I could not fit the all of them into a screenshot together

Tract Demo

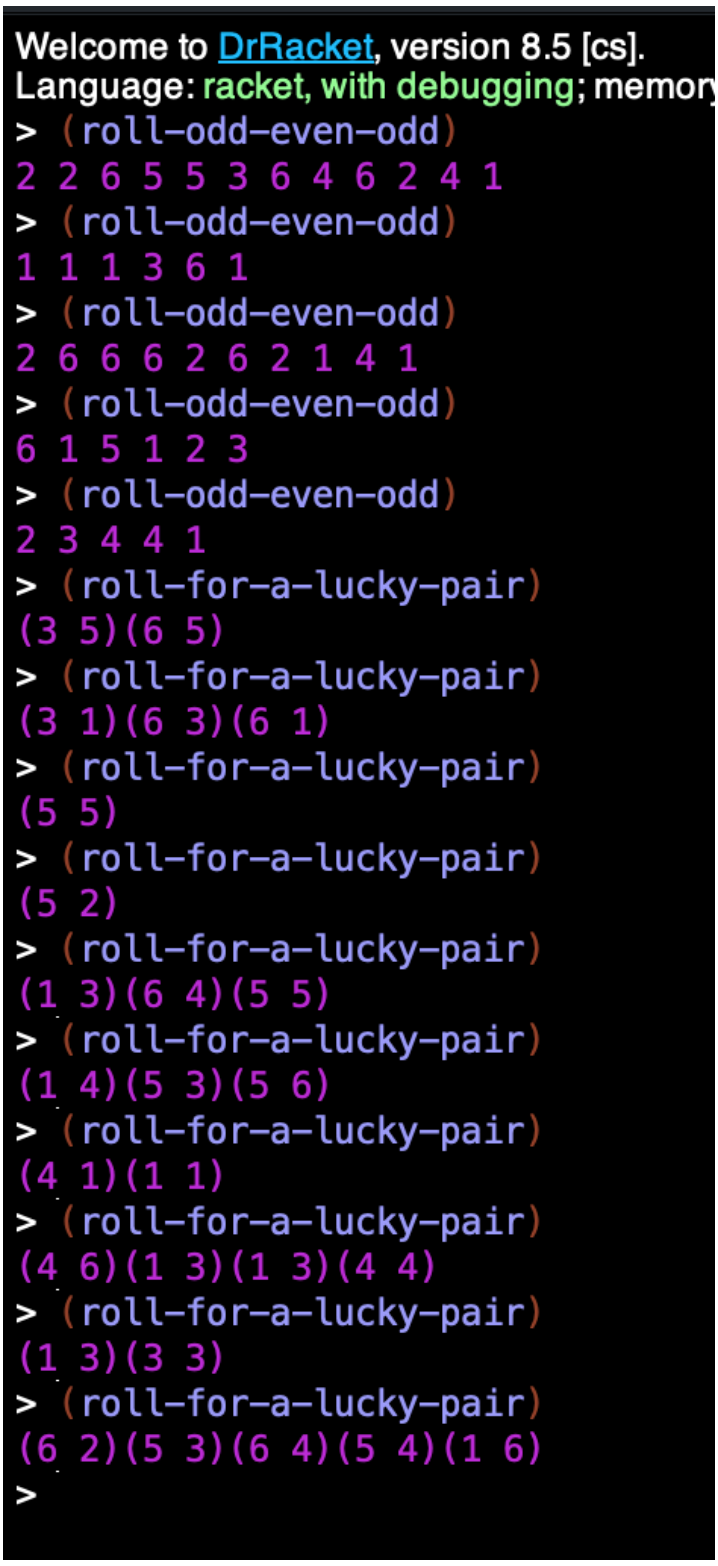
Task 2: Dice



Roll for 11 demo



Regular dice roll and roll for 1 demo



Roll for odd-even-odd and roll for lucky pair demo

```
(define (normal-dice) (random 1 7))

(define (roll-dice) (normal-dice))


( define ( roll-for-1 )
   ( define outcome ( normal-dice ) )
   ( display outcome ) ( display " " )
   ( cond
      ( ( not ( eq? outcome 1 ) )
        ( roll-for-1 )
        )
      )
   )

( define ( roll-for-11 )
   (roll-for-1)
   ( define outcome ( normal-dice ) )
   ( display outcome ) ( display " " )
   ( cond
      ( ( not ( eq? outcome 1) )
        ( roll-for-11 )
        )
      )
   )


(define(roll-for-even) (define outcome (normal-dice))
  (display outcome ) (display " ")
  (cond [(not (even? outcome))(roll-for-even)]))

(define(roll-for-odd) (define outcome (normal-dice))
  (display outcome ) (display " ")
  (cond [(not (odd? outcome))(roll-for-odd)]))



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

First half of dice code

```
(define (roll-for-a-lucky-pair)
  (display "(" )
  (define result (normal-dice))
  (define results (normal-dice))
  (display result)
  (display " ")
  (display results)
  (display ")")
 (define total
   ( + result results ))
  (define total-7
    ( = total 7))
  (define total-11
    ( = total 11) )
  (define equal
    ( = result results))
  (cond
    ( (not(or total-7 total-11 equal))
       (roll-for-a-lucky-pair))))
```

Second half of dice code

Task 3: Number Sequences

```
> (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
>
```

Square, sequence square, cube and sequence cube demo

```
Welcome to DrRacket, version 8.5 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (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
>
```

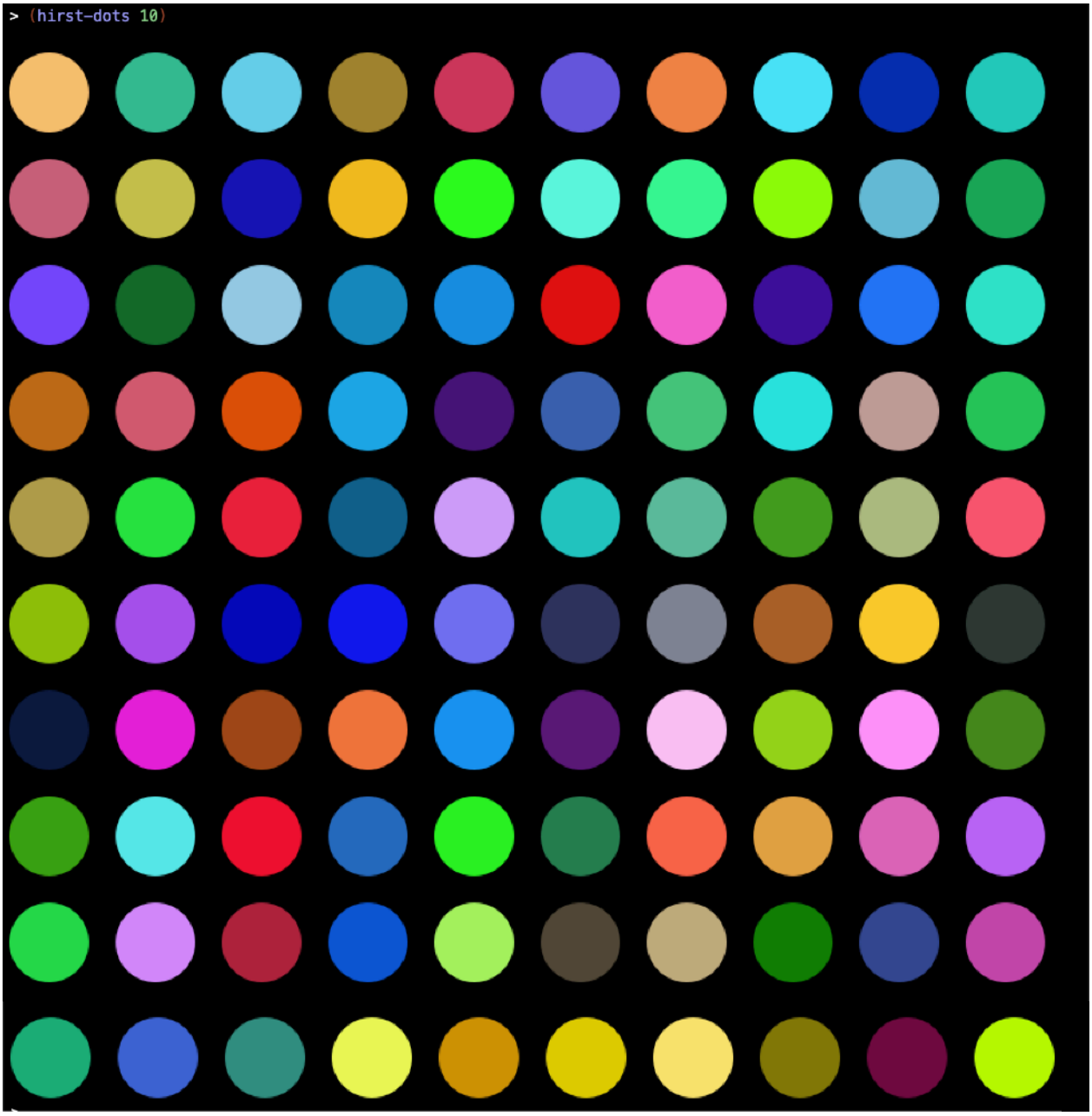
Triangular and sequence triangular demo

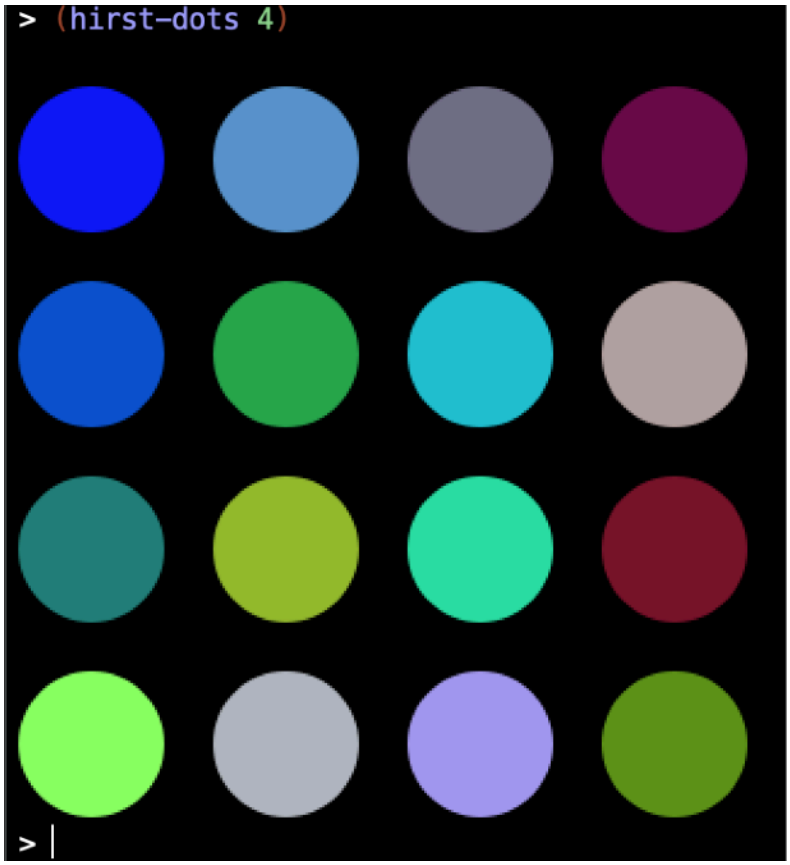
```
#lang racket

( define ( square n ) (* n n))
( define ( cube n )(* n n n))

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


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

Number Sequences code

This page is blank currently as it is where the sigma code will being going in the near future. I am just unable to complete at the time and so it will be added as soon as possible

Task 4: Hirst Dots





Hirst dots demo code

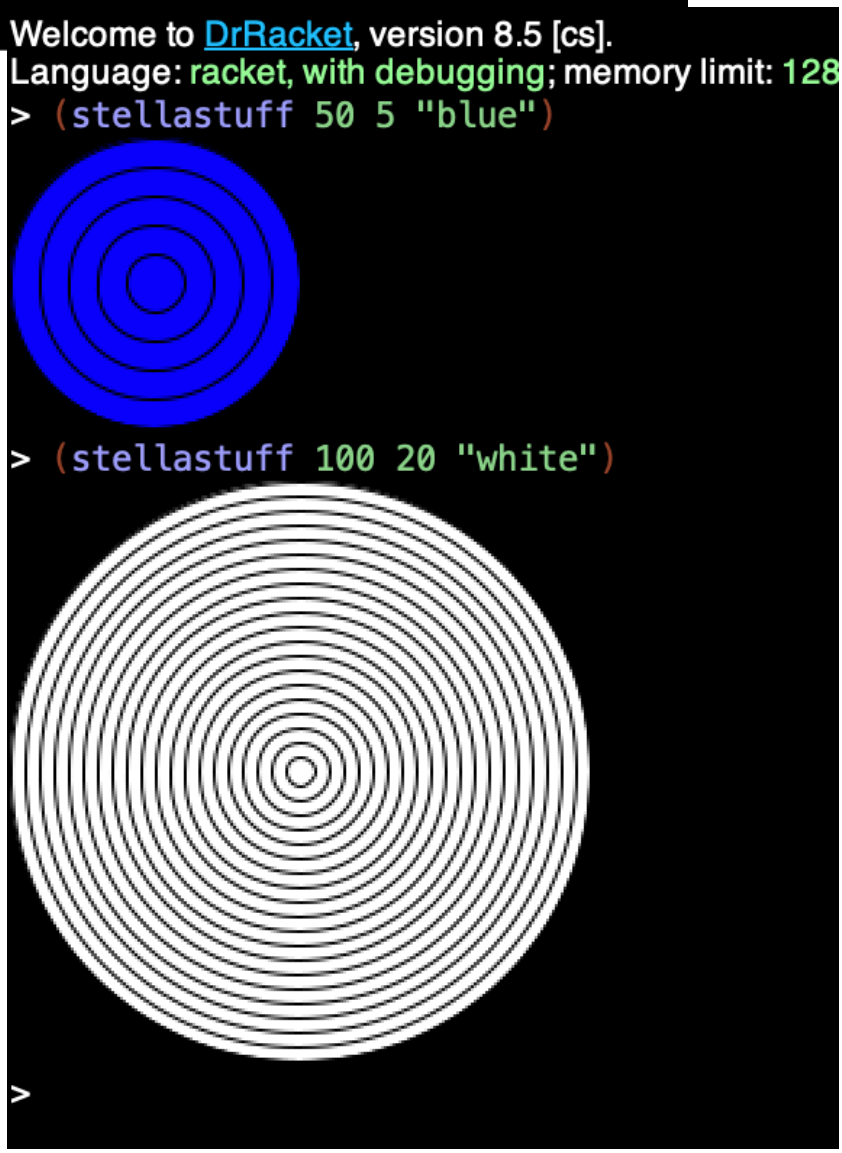
```
#lang racket
(require 2htdp/image)

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

(define space (square 20 "solid" "black"))
(define (dot) (circle 30 "solid" (random-color)))
(define (sqr) (square 50 "solid" "transparent"))

(define (row-x num) (cond
                      ((= num 0) empty-image )
                      ((> num 0)(beside (row-x (- num 1)) (dot) space))))
(define (row-y num1 num2) (cond
                            ((= num1 0) empty-image)
                            ((> num1 0) (above (row-y (- num1 1) num2) space (row-x num2)))))
(define (hirst-dots samenum) (row-y samenum samenum))
```

Hirst Dots Code

Task 5: Chanelling Frank Stella

```
#lang racket
(require 2htdp/image)

(define (stellastuff radius count color)
  (define unit (/ radius count))
  (paint-stella-stuff 1 count unit color)
  )
(define (paint-stella-stuff from to unit color)
  (define rad (* from unit))
  (cond
    ((= from to)
     (framed-circle rad color))
    ((< from to)
     (overlay (framed-circle rad color) (paint-stella-stuff (+ from 1) to unit color)))))
(define (framed-circle radius color)
  (overlay (circle radius "outline" "black")
           (circle radius "solid" color)))
```

Stella code



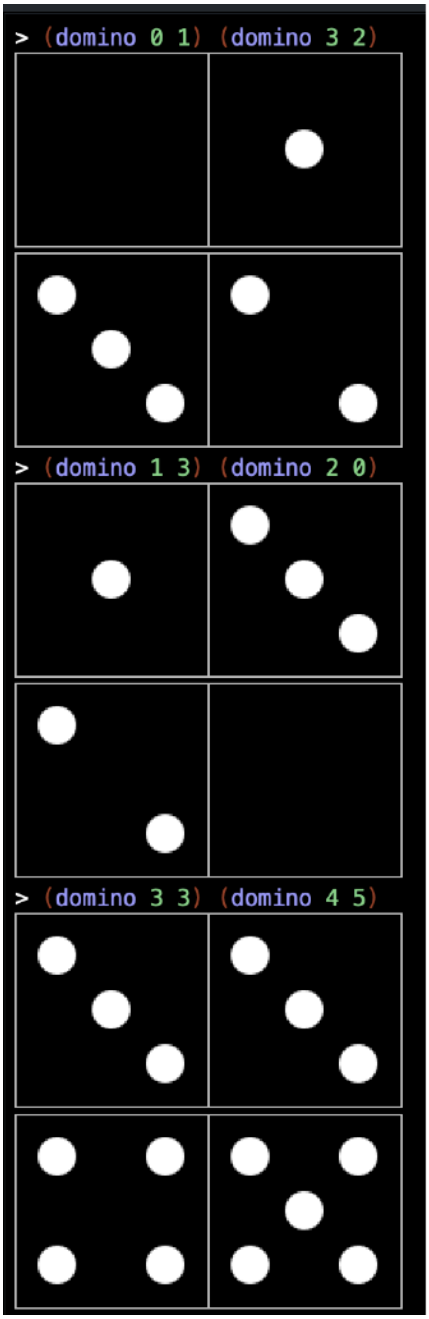
Stella demo

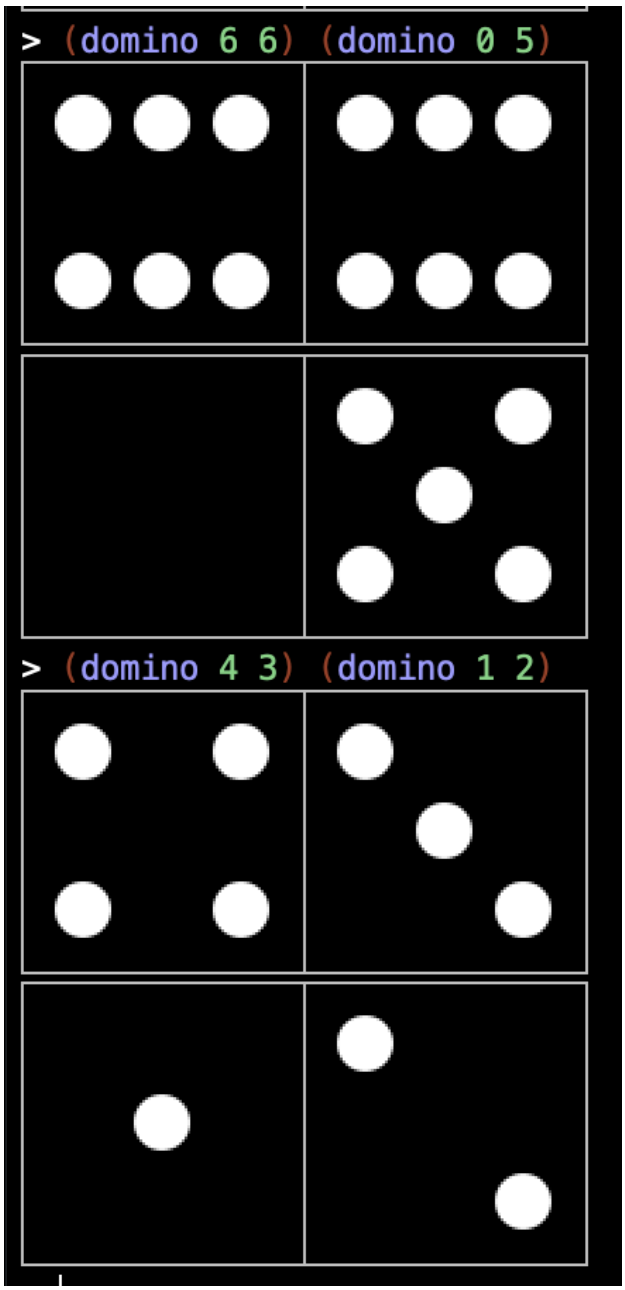
Task 6: Dominos

```
( 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 ( overlay/offset
                                     ( pip ) d nd ( overlay/offset
                                                    ( pip ) nd d ( overlay/offset
                                                                   ( pip ) nd nd blank-tile)))))
( define basic-tile5 (overlay (pip) basic-tile4))
( define basic-tile6 ( overlay/offset
                       ( pip ) 0 d
                       ( overlay/offset
                                     ( pip ) 0 nd
                                     ( overlay basic-tile4 basic-tile2))))
```

```
;-----------------------------------------------------------------------
; 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 )
)
)
```

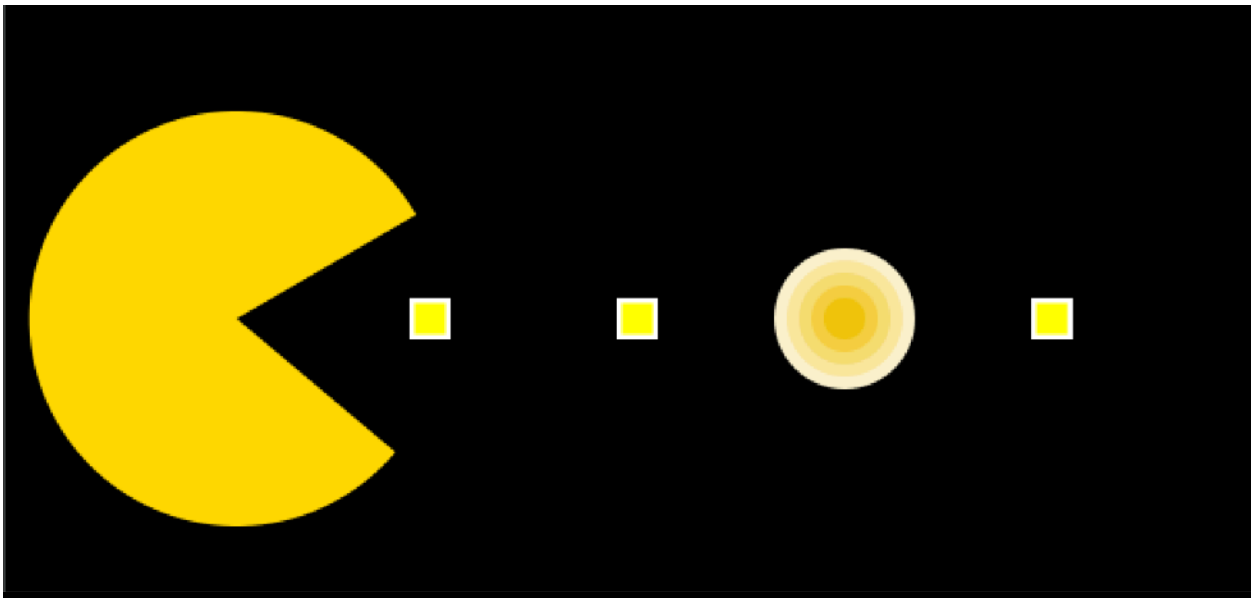
Dominos Code





Dominos Demo

Task 7: Creation



```
#lang racket
(require 2htdp/image)
```

MY CREATION DEMO

```
(define bg (empty-scene 700 500 "black"))

(define outside (square 20 "solid" "white"))
(define minipellet(overlay( square 15 "solid" "yellow") outside))

(define powerpellet (overlay (circle 10  "solid" (make-color  241  196 15))
          (circle 16  "solid" (make-color 244 208 63))
          (circle 22  "solid" (make-color 247 220 111))
          (circle 28  "solid" (make-color 249 231 159))
          (circle 34  "solid" (make-color 252 243 207))))


(define pac-man (rotate 30 (wedge 100 290 "solid" "gold")))

(define game1(place-image pac-man 100 250 bg))
(define game2(place-image minipellet 200  250 game1))
(define game3(place-image minipellet 300 250 game2))
(define game4(place-image powerpellet 400 250 game3))
(define game5(place-image minipellet 500 250 game4))
```

My creation demo and code