Trabalho de Paradigmas de Linguagens de Programação

ALUNOS: ALAN WILLY LEISER

LUAN VICTOR

MATHEUS MATIAZZO

PROFESSOR:EVANDRO BACCARIN

Objetivo

Implementar um script em Racket que gere casos de testes para a fase III do trabalho de Estrutura de dados.

Implementação GEO

Implentação

Cores

```
(define cores (list "darkmagenta" "darkblue" "brown" "mediumvioletred"
                    "hotpink" "darkgreen" "ivory" "darkolivegreen"
                    "darkseagreen" "darkturquoise" "dimgray" "firebrick"
                    "goldenrod" "lightseagreen" "orange" "royalblue""orchid"
                  "limegreen" "indianred" "lightgoldenrodyellow" "lawngreen" "darkcyan"))

(define cor1 (list-ref cores (random 0 (length cores))))
```

Implentação

Definição da quantidade de figuras, quadras e prédios

```
;definindo quantidades
(printf "Digite a Quantidade de figuras: ")
(define totalFig (read-line (current-input-port) 'any))
(printf "Digite a o Tamanho do Bairro: ")
(define quantTotalQuad (read-line (current-input-port) 'any))
(printf "Digite a Quantidade de Predios: ")
(define  qPredios(read-line (current-input-port) 'any))
```

Implentação

Gerando quadra e equipamentos urbanos

```
;desenha equipamentos urbanos
(define (funcWhile terminonf)

  (set! yInicio (Cond1 qDir yInicio intervalo))
  (set! xQuadra (Cond2 qDir xInicio xQuadra))
  (set! yQuadra (Cond3 qDir yInicio yQuadra))
  (set! qBaixo (Cond4 qDir qBaixo))
  (set! qDir (Cond5 qDir quantTotalQuad))
  (define insereQuadra (quadra ale qBaixo stringindice xQuadra yQuadra w h out))
```

```
(indiceFunc)
(cond
  [(> terminonf 0) (funcWhile (- terminonf 1))]));volta para o inicio da funcao caso ainda existam quadras a serem printadas
```

Ajuste para impressão

```
(define (Cond1 qDir yInicio intervalo);1
  (if (< qDir 0)
       (+  yInicio intervalo)
         yInicio))

(define (Cond2 qDir xInicio xQuadra);2
  (if (< qDir 0)
      xInicio
      xQuadra))
```

```
(define (Cond3 qDir  yInicio yQuadra);3
  (if (< qDir 0)
      yInicio
      yQuadra))

(define (Cond4  qDir  qBaixo);4
  (if (< qDir 0)
      (+ qBaixo 1)
      qBaixo))

(define (Cond5 qDir quantTotalQuad);5
  (if (< qDir 0)
      (- quantTotalQuad 1)
      qDir))
```

Implementação

Gera prédios

```
(define (funcWhile4 qPredios nq)
  (set! aleatorioQ1 (random 1 nq))
  (set! aleatorioQ2 (random 1 nq))
  (set! fPredio (random 30 45))
  (set! pPredio (random 35 70))
  (set! pMrg (random 10 20))
  (set! facePredio (random-nslo))
  (display (string-append "prd" " " "b0"(number->string aleatorioQ1)"."(number->string aleatorioQ2) " "
                          facePredio " " (number->string fPredio) " " (number->string pPredio) " "
                          (number->string pMrg) "\n") out)
  (cond
    [(> qPredios 1) (funcWhile4 (- qPredios 1) nq)]))

(define resultado4 (funcWhile4 qPredios nq))
```

Implementação e Impressão

Gera circulo e retângulo

```
(define (funcWhile2)
  (set! aleFig (random-crt))
  (set! stringale (random-string))
  (set! xFig (random 20 250));numero aleatorio para ponto no eixo x
  (set! yFig (random xFiguras (+ xFiguras 100)));ponto aleatorio para ponto no eixo y
  (set! rFig (random 50));numero aleatorio para o tamanho do raio
  (set! wFig (random 200));largura do retangulo
  (set! hFig (random 200));altura do retangulo
  (cond
    [(string=? "t" aleFig) (set! textCont(+ textCont 1))])
  (cond
    [ (> textCont 3) (set! aleFig (random-cr))])
  (cond
    [(string=? aleFig "c") (set! indiceFig(+ indiceFig 1))]
    [(string=? aleFig "r") (set! indiceFig(+ indiceFig 1))])
  (cond
    [(string=? aleFig "c") (set! totalFig(- totalFig 1))]
    [(string=? aleFig "r") (set! totalFig(- totalFig 1))])
  (cond
    [;primeira condição caso for um circulo printa (tipo (id)->ainda nao foi implementado r x y cor1 cor2), os outros seguem respectivamente a mesma ideia
     (string=? "c" aleFig) (display (string-append aleFig " " (number->string indiceFig) " "
                                                   (number->string rFig) " " (number->string xFig) " " (number->string yFig) " " corFig2 " " corFig1"\n") out)]

    [(string=? "r" aleFig) (display (string-append aleFig " " (number->string indiceFig) " "
                                                   (number->string wFig) " " (number->string hFig) " " (number->string xFig) " " (number->string yFig) " "
                                                   corFig1 " " corFig2"\n") out)]

    [(string=? "t" aleFig) (display (string-append aleFig " " (number->string xFig) " " (number->string yFig) " " stringale "\n") out)])
```

Implementação QRY

Leitura do GEO

```
;func de leitura geo
(define (leitura arq)
  (define in (open-input-file arq));abre o arquivo
  (define nx(values (read-line in)));le geo
  (close-input-port in)
  (define semNX(substring nx 3))
  (define aux(string-split semNX))
;aplica a funcao num em todos os elementos da lista de strings(b)
  (map num aux)
  )
```

Chamada dos comandos

```
(define trns(comand9 max))

(define trnsdir(comand10 max raizQ))

(define fh(comand11 nh raizQ))

(define fs(comand12 ns raizQ))

(define fi(comand13 ns))

(define brl(comand14))

(define trnsesq(comand15 max raizQ))

(define trnnbaixo(comand16 max raizQ))
```

```
(define o(comand1 i max))

(define i?(comand2 i max))

(define d(comand3 i max))

(define bb(comand4))

(define dq(comand5))

(define del(comand6 raizQ raizR raizS))

(define cbq(comand7 raizQ))

(define crd(comand8 raizQ raizR raizS))
```

O?

```
;comando 1
(define (comand1 i max)
(define out1 (open-output-file "qry/q-o.qry"))
(define (funcWhileo n)
  (define j(random 1 i))
  (define k(random 1 i))
  (cond
  [(= j k) (set! j(random 1 i))])
  (display (string-append "o?" " " (number->string j) " " (number->string k)"\n") out1)
  (cond
    [(> n 0) (funcWhileo (- n 1))]
    ))
(define resultadoo (funcWhileo max))
(close-output-port out1))
```

TRNS

```
;trnsbaixo
(define (comand16 max raizQ)
(define out16 (open-output-file "qry/q-trnsbaixo.qry"))
(define (funcWhiletrnsdir n)
  (define w 500)
  (define h 500)
  (define x (* raizQ 4))
  (define y (* raizQ 4))
  (define dx 0)
  (define dy 500)
  (display (string-append "trns" " "(number->string x) " "(number->string y) " " (number->string w) " "
                          (number->string h) " "(number->string dx) " "(number->string dy) "\n") out16)
  (cond
    [(> n 0) (funcWhiletrnsdir (- n 1))]
    ))
  (define resultadotrnsdir (funcWhiletrnsdir 0))
(close-output-port out16))
```

FH

```
;comando fh
(define (comand11 nh raizQ)
  (define out11 (open-output-file "qry/q-fh.qry"))
  (define prox(list "-" "+"))
  (define k(random 0 nh))
  (define face(list "n" "s" "l" "o"))
  (define cep "b0")
  (define cepnum(random 0 raizQ))
  (define cepnum2(random 0 raizQ))
  (display(string-append (list-ref prox(random 0 (length prox)))"" (number->string k) " "
                         cep (number->string cepnum)"."(number->string cepnum2) " "
                         (list-ref face(random 0 (length face)))" " "\n")out11)

  (close-output-port out11))
```

Demonstração