UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO E ENGENHARIA DE COMPUTAÇÃO

FELIPE FISCHER COMERLATO
LEONARDO EICH

Grupo Equipe 7
Projeto Space Invaders Utilizando Lua

Relatório apresentado como requisito parcial para a obtenção de conceito na Disciplina de Modelos de Linguagens de Programação

Prof. Dr. Lucas Mello Schnorr
Orientador

Porto Alegre
2018

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem como objetivo o estudo de uma linguagem de programação moderna com características híbridas contextualizando os conceitos vistos em aula ao longo do semestre e, por fim, analisar e avaliar diferentes linguagens de programação, seguindo os critérios vistos em aula.

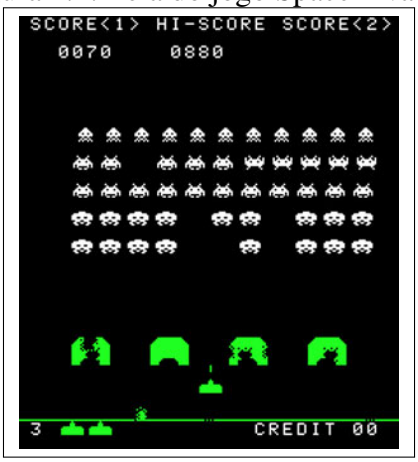
A tarefa principal do trabalho consiste em experimentar e comparar as características e funcionalidades orientadas a objeto e funcionais da linguagem de programação escolhida. De posse de uma linguagem, é necessário escolher um problema a ser solucionado com ela. O problema será, então, implementado duas vezes na mesma linguagem: uma delas usando somente Orientação a Objetos e a outra usando somente características funcionais.

O desenvolvimento do trabalho encontra-se em <https://github.com/felipefcomerlato/mlp_equipe7_2018-2>.

2 O PROBLEMA

O problema a ser resolvido neste trabalho é o desenvolvimento do jogo Space Invaders. O jogo expõe o jogador como uma espaçonave que deve destruir as espaçonaves inimigas que querem invadir o planeta do jogador. Na medida que elas avançam na tela (de cima para baixo), o jogador guia sua espaçonave horizontalmente e efetua disparos para destruir todas as ondas de inimigos que se seguem, como visto na Figura 2.1

Figura 2.1: Tela do jogo Space Invaders



Fonte: http://g1.globo.com/tecnologia/noticia/2011/10/homem-quebra-recorde-mundial-em-space-invaders-jogo-de-1978.html

3 VISÃO GERAL DA LINGUAGEM

A linguagem escolhida para o trabalho foi extbfLua. Lua é uma linguagem de script de multiparadigma, pequena, reflexiva e leve, projetada para expandir aplicações em geral, por ser uma linguagem extensível (que une partes de um programa feitas em mais de uma linguagem), para prototipagem e para ser embarcada em softwares complexos, como jogos. Assemelha-se com Python, Ruby e Icon, entre outras.

Lua foi criada por um time de desenvolvedores do Tecgraf da PUC-Rio, a princípio, para ser usada em um projeto da Petrobras. Devido à sua eficiência, clareza e facilidade de aprendizado, passou a ser usada em diversos ramos da programação, como no desenvolvimento de jogos (a Blizzard Entertainment, por exemplo, usou a linguagem no jogo World of Warcraft), controle de robôs, processamento de texto, etc. Também é frequentemente usada como uma linguagem de propósito geral.

Uma das características principais da linguagem é sua única estrutura de dados: tabelas - que podem ser usadas para representar arrays comuns, sequências, tabelas de símbolos, conjuntos, registros, grafos, árvores, etc.

Lua é uma linguagem distribuída e pode ser compilada em qualquer plataforma que tenha um compilador da linguagem C. Lua pode ser executada em ambientes Unix e Windows, em dispositivos móveis (Android, iOS, Windows Phone, etc), em microprocessadores embarcados e etc.

Incluir Lua numa aplicação não aumenta quase nada o seu tamanho. O pacote de Lua 5.3.5, contendo o código fonte e a documentação, ocupa 297K comprimido e 1.2M descompactado. O fonte contém cerca de 24000 linhas de C. No Linux de 64 bits, o interpretador Lua contendo todas as bibliotecas padrões de Lua ocupa 247K e a biblioteca Lua ocupa 421K (ABOUT, 2018).

Lua possui interpretação dinâmica, ou seja, a linguagem é capaz de executar trechos de código criados dinamicamente, no mesmo ambiente de execução do programa. Como exemplos dessa facilidade temos a função *loadstring* em Lua e a função *eval* em Scheme/Lisp e Perl. Lua possui também tipagem dinâmica forte, ou seja, a linguagem faz verificação de tipos em tempo de execução do programa. Linguagens com tipagem dinâmica em geral não possuem declarações de tipos no código e não fazem verificação de tipos em tempo de compilação. Tipagem forte significa que a linguagem jamais aplica uma operação a um tipo incorreto. Além disso, Lua conta com gerência automática de memória dinâmica ("coleta de lixo"), por tanto não precisamos gerenciar memória ex-

plicitamente no nosso programa; em especial, não há necessidade de um comando para liberar memória após seu uso (UMA. . . , 2018).

3.1 Modelo Funcional

Lua oferece suporte ao modelo de programação funcional mas nos proporciona algumas dificuldades, como a de copiar tabelas, por exemplo, já que tal estrutura de dados atribuída a uma variável é apenas uma referência. Tal dificuldade contraria uma das principais características do próprio paradigma funcional: a de usar funções puras, criando novas estruturas ao invés de alterar uma estrutura original. Por outro lado, Lua nos permite representar funções anônimas através de blocos de código chamados de trechos, característico no modelo funcional. (MANUAL. . . , 2018)

3.2 Modelo Orientado a Objetos

O suporte para a programação no paradigma orientado a objetos utilizando Lua é muito vasto. É possível encontrar inúmeros tutoriais para a solução de diversos problemas na linguagem fazendo uso deste paradigma. Como não vimos diversos pré-requisitos necessários para a implementação deste trabalho, não podemos analisar com mais propriedade o desempenho da linguagem Lua neste modelo de programação.

4 RECURSOS FUNCIONAIS

4.1 Elementos Imutáveis e Funções Puras

Funções puras são funções que não causam efeitos colaterais, ou seja, que não alteram valores de variáveis do programa. Em Lua não existe mecanismos explícitos para a criação de funções puras, ou seja, fica a cuidado do programador a implementação dessa técnica (IMMUTABLE. . . , 2018).

Um exemplo de função pura criada para o projeto é a função *newStateOfEnemies*, definida pelo código a seguir:

```
function newStateOfEnemies(enemies,death_row,death_col)
  copyEnemies = {}
  insertRows = function(r)
    if r > 0 then
      copyEnemies[r] = {}
      insertCols = function(c)
        if c > 0 then
          if r == death_row and c == death_col then
            copyEnemies[r][c] = 0
          else
            copyEnemies[r][c] = enemies[r][c]
          end
          insertCols(c-1)
        end
      end
      insertCols(#enemies[r])
      insertRows(r-1)
    end
  end
  insertRows(#enemies)
  return copyEnemies
end
```

newStateOfEnemies foi criada com a finalidade de atualizar o estado dos inimigos no jogo. A função recebe a matriz (uma tabela de tabelas, em Lua) de inimigos e a posição da matriz na qual um inimigo morreu, e devolve uma matriz totalmente nova atualizada.

4.2 Funções Anônimas

Em Lua, por definição, todas as funções são anônimas. Todavia, é possível armazenar as funções em variáveis, para que possam ser utilizadas da maneira tradicional (UMA. . . , 2018).

No projeto, um exemplo de uso dessa técnica é a implementação da função *setPlayerShot*, mostrada no código a seguir:

```
setPlayer = function()
    player_image = love.graphics.newImage("images/baseshipb.png")
    x_player = love.graphics.getWidth() / 2 - player_image:getWidth() /
    y_player = love.graphics.getHeight() - 2 * player_image:getHeight()
    max_x_player = love.graphics.getWidth() - player_image:getWidth()
    min_x_player = 0
    x_player_shift = 10

    setPlayerShot = function()
      player_shot_image = love.graphics.newImage("images/tiro.png")
      x_player_shot = 0
      y_player_shot = love.graphics.getWidth()
      player_shot_speed = 900
      y_player_shot_shift = 0
      player_shot_on_the_screen = false
    end
    setPlayerShot()
  end
```

Como pode ser visto, *setPlayerShot* foi implementada dentro da função *setPlayer*, portanto só existe dentro desse contexto. A função define a textura e posição inicial do

tiro do jogador.

4.3 Currying

Currying é a técnica de transformar uma função de N parâmetros em N funções de 1 parâmetro cada (CURRYING. . . , 2018) projeto, foi implementada uma função genérica *curry2*, mostrada abaixo:

```
function curry2(f)
  return function(a)
    return function(b)
      return f(a, b)
    end
  end
end
```

curry2 é utilizada para fazer a verificação de colisão de cada inimigo com o tiro do jogador. A chamada da função de verificação de colisão com *curry2* é como no código a seguir:

```
verifyCollision = curry2(verifyCollision)
funRow = verifyCollision(row)
funCol = funRow(enemies_on_the_row)
```

4.4 Pattern Matching

Pattern matching ou casamento de padrões é a verificação por padrões dentro de uma estrutura de dados. Lua possui este recurso, principalmente no tratamento de *strings* (PATTERNS. . . , 2018). No entanto, não foi encontrado um modo de utilizar este recurso no projeto até o momento.

4.5 Funções de Ordem Superior

Funções de ordem superior são aquelas que recebem uma função como parâmetro e/ou retornam uma função. Foi implementada genericamente a função *map*, uma vez que Lua não possui tal função em sua definição (HIGHER..., 2018). *map* é mostrada a seguir:

```
function map(fun, t, sizeT)
  if sizeT > 0 then
    fun(t[sizeT])
    map(fun, t, sizeT-1)
  end
end
```

4.6 Funções de Ordem Maior Fornecidas Pela Linguagem

Conforme estudos realizados sobre Lua, foram encontrados poucos exemplos de funções de alta ordem disponíveis na linguagem. Um exemplo (não utilizado no trabalho) é a função *table.sort*, mostrada abaixo, que espera como um dos parâmetros uma função que define o sentido de ordenação de uma tabela (FUNCTIONS..., 2018).

```
list = {{3}, {5}, {2}, {-1}}
table.sort(list, function (a, b) return a[1] < b[1] end)
```

4.7 Funções como Elementos de Primeira Ordem

Funções são utilizadas como elementos de primeira ordem quando elas podem ser passadas como parâmetro e também retornadas por outras funções. Como mostrado na seção 4.3, a função *verifyCollision* é passada como parâmetro para a função *curry2* e retornada pela mesma (FUNCTIONS..., 2018).

4.8 Recursão

Conforme especificação, foi utilizada recursão como único mecanismo de iteração sobre as estruturas. Um exemplo de recursão implementada foi com a função *drawEne-*

mies, mostrada a seguir:

```
function drawEnemies(rows)

  if rows > 0 then
    -- Desenha uma linha de inimigos
    drawEnemiesOnTheRow = function(row, enemies_on_the_row)
      x_enemie = enemies_on_the_row * x_distance_btw_enemies + x_enemie
      y_enemie = row * y_distance_btw_enemies + y_enemies_shift

      -- Testa se houve colisão de um inimigo com o tiro
      verifyCollision = function(row, enemies_on_the_row)
        -- Delimita os pontos de colisão do tiro
        limit_left = x_player_shot + player_shot_image:getWidth()/2 - e
        limit_right = x_player_shot + player_shot_image:getWidth()/2
        limit_bottom = y_player_shot + player_shot_image:getHeight()/2

        -- Se o inimigo está vivo, testa a colisão dele com o tiro
        if states_of_enemies[current_state][row][enemies_on_the_row] ==
          if x_enemie >= limit_left and x_enemie <= limit_right then
            if y_enemie + enemie1_image:getHeight() >= limit_bottom the
              -- Gera novo estado dos inimigos a partir de uma cópia do
              table.insert(states_of_enemies, newStateOfEnemies(states_
              current_state = #states_of_enemies
              resetShot()
            end
          end
        end
      end

      -- Implementação curry
      local verifyCollision = curry2(verifyCollision)
      local funRow = verifyCollision(row)
      local funCol = funRow(enemies_on_the_row)
```

```
      if enemies_on_the_row > 0 then
        -- Define inimigos diferentes conforme a linha da matriz
        if states_of_enemies[current_state][row][enemies_on_the_row] ==
            if row <= 2 then
              love.graphics.draw(enemie1_image, x_enemie, y_enemie)
            elseif row == 3 or row == 4 then
              love.graphics.draw(enemie2_image, x_enemie, y_enemie)
            else
              love.graphics.draw(enemie3_image, x_enemie, y_enemie)
            end
        end
        -- Desenha os demais inimigos da linha recursivamente
        drawEnemiesOnTheRow(row, enemies_on_the_row-1)
      end

    end
    drawEnemiesOnTheRow(rows, #states_of_enemies[current_state][rows])
    drawEnemies(rows-1)
  end
  -- Desenha o inimigo "mystery"
  x_mystery_enemie = x_mystery_enemie + x_mystery_enemie_shift
  love.graphics.draw(mystery_enemie_image, x_mystery_enemie, y_mystery_
end
```

A função *drawEnemies* desenha na tela cada inimigo através de uma tabela de tabelas, que é percorrida recursivamente para verificar se o inimigo de cada posição deve ou não ser desenhado.

5 PARALELISMO

Lua oferece recurso de paralelismo através da função *thread*, porém, tal recurso é conflitante com a biblioteca gráfica Love2d utilizada como mecanismo principal de funcionamento do jogo. Em contrapartida, a própria biblioteca Love2d possui um mecanismo de paralelismo. Apesar disso, devido à complexidade de uso de tal mecanismo, ainda não foi utilizado o recurso de paralelismo no projeto na sua atual versão, já que muitos erros foram resultantes das tentativas de uso das *threads* com a Love2d.

6 RECURSOS DE ORIENTAÇÃO À OBJETOS

6.1 Classes

6.2 Encapsulamento e Proteção dos Atributos

6.3 Construtores

6.4 Destrutores

6.5 Espaços de Nomes Diferenciados

6.6 Mecanismos de Herança

6.7 Polimorfismo por Inclusão

6.8 Polimorfismo Paramétrico

6.9 Polimorfismo por Sobrecarga

6.10 Delegates

7 CONCLUSÃO FINAL

REFERÊNCIAS

ABOUT. 2018. Available from Internet: <https://www.lua.org/about.html>.

CURRYING in Lua. 2018. Available from Internet: <https://www.reddit.com/r/lua/comments/6yy9wu/currying_in_lua/>.

FUNCTIONS Tutorial. 2018. Available from Internet: <http://lua-users.org/wiki/FunctionsTutorial>.

HIGHER Order Functions in Lua. 2018. Available from Internet: <http://inmatarian.github.io/2015/08/09/send_more_money/>.

IMMUTABLE Objects. 2018. Available from Internet: <http://lua-users.org/wiki/ImmutableObjects>.

MANUAL de Referência de Lua 5.2. 2018. Available from Internet: <https://www.lua.org/manual/5.2/pt/manual.html#2.6%20%E2%80%93%20Co-rotinas>.

PATTERNS Tutorial. 2018. Available from Internet: <http://lua-users.org/wiki/PatternsTutorial>.

UMA Introdução à Programação em Lua. 2018. Available from Internet: <https://www.lua.org/doc/jai2009.pdf>.