XIII ENCONTRO ANUAL DE TECNOLOGIA DA INFORMAÇÃO
Anais do XIII Encontro Anual de Tecnologia da Informação - EATI
| Ano 11, n. 1 | novembro/ 2022

Relato de Experiência do Desenvolvimento de Aplicação de Jogo Digital Empregando a Linguagem Lua

Thiago D. de C. Q. Gama[1], Arthur Aleksandro A. Silva [2], Danielle S. da S. Casillo[2]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74.690-900 – Goiânia – GO – Brazil

[2]Departamento de Computação – Universidade Federal Rural do Semi-Árido (UFERSA)
Caixa Postal 137 – 59.625-900 – Mossoró – RN – Brazil

thiago.gama@discente.ufg.br, aretw0@hotmail.com,
danielle@ufersa.edu.br

Abstract. *This study contextualizes the Lua functional language in the field of Theory of Computation through works that present the elements of the functional language paradigm, the definition and origin of the mathematical function with the Lambda calculus represented in the functional language and the composition of functions, the applicability, strengths and weaknesses, examples and applications of functional languages. Finally, the Lua language is defined, its characteristics that differentiate it from other functional languages, syntax of the main functionalities, such as the data type, code examples, control structures, modules and frameworks and an application made with an object-oriented paradigm, tested in the ZeroBrane Studio IDE and using the LÖVE framework, showing its syntax and associating the theory with the practice of the theme addressed in this work.*

Resumo. *Este estudo contextualiza a linguagem funcional Lua no campo da Teoria da Computação por meio de trabalhos que apresentam os elementos do paradigma da linguagem funcional, a definição e origem da função matemática com o cálculo Lambda representado na linguagem funcional e a composição de funções, a aplicabilidade, pontos fortes e pontos fracos, exemplos e aplicações das linguagens funcionais. Por fim, define-se a linguagem Lua, suas características que diferenciam das outras linguagens funcionais, sintaxe das principais funcionalidades, como o tipo de dados, exemplos de códigos, estruturas de controle, módulos e frameworks e uma aplicação feita com paradigma orientado a objetos, testada na IDE ZeroBrane Studio e utilizando o framework LÖVE, mostrando sua sintaxe e associando a teoria com a prática da temática abordada neste trabalho.*

1. Introdução

O desenvolvimento de sistemas atualmente vem crescendo espantosamente e, segundo Guedes (2016), os ambientes produtivos geralmente utilizam linguagens imperativas ou orientadas a objetos, mas a demanda ininterrupta por escalabilidade e distribuição exigiu um novo paradigma de programação: o funcional.

A linguagem de paradigma funcional é, segundo Farias et al. (2013), um paradigma que trata a computação com a aplicação de funções matemáticas e evita a noção de estado e de dados variáveis como ocorre na programação imperativa. Segundo Sampaio (2018) o paradigma funcional é o de programas com funções que descrevem uma relação explícita e precisa entre entradas e saídas, com estilo declarativo, não havendo conceito de estados nem comandos como atribuição.

De acordo com Bove, Dybjer e Norell (2009), as linguagens funcionais são ferramentas rápidas e altamente personalizáveis para resolver atividades rotineiras. São usadas em aplicações grandes, complexas e críticas às quais seria impossível desenvolver em outra linguagem. Uma das vantagens das linguagens funcionais

consiste em tratar os programas como dados, possibilitando, dessa forma, que um programa inteiro seja dado como entrada de outro, o que não acontece em outras linguagens como C e Pascal.

Conforme Seliger (2014), as linguagens funcionais surgiram com o estudo do cálculo Lambda de Church na década de 1930 e que as linguagens funcionais surgem como um paradigma para possibilitar o desenvolvimento de programas de alto nível de abstração, com relação explícita e precisa. Essas linguagens têm soluções elegantes, poderosas e concisas com conceitos sofisticados como polimorfismo, e funções de alta ordem, possibilitando o avanço das pesquisas.

Segundo Araújo, Sellanes e Costa (1998), o cálculo Lambda é o cálculo em que a linguagem Lambda foi introduzida por Alonzo Church em 1941 para evitar ambiguidades de notação, e que são frequentes em programação, sendo representada por duas construções: Abstração Lambda, que permite abstrair a definição da função, e Aplicação Lambda, que determina o valor da função aplicada a um dado parâmetro.

Portanto, esse artigo apresenta a utilização da linguagem Lua, atendendo o paradigma funcional, na implementação de um jogo de computador. O objetivo é demonstrar o potencial e praticidade de uma linguagem funcional por meio das ferramentas e aplicação aqui apresentadas.

O presente artigo está estruturado da seguinte forma: a seção 2 apresenta uma contextualização para este artigo sobre a linguagem Lua, destacando alguns trabalhos relacionados e características da linguagem; a seção 3 mostra como foi realizada a implementação do jogo; na seção 4 é apresentado a aplicação e como utilizá-la; a seção 5 apresenta as conclusões e trabalhos futuros; e no final as referências utilizadas no artigo.

2. Lua

De acordo com Ierusalimschy (2006), Lua é uma conhecida linguagem de roteiro de multiparadigma, simples, abstraída e rápida, projetada para estender aplicações cotidianas, por ser uma linguagem extensível (que junta componentes de um software produzidos em mais de uma linguagem), para prototipagem e para ser inserida em programas complexos, por exemplo, jogos. Semelhante com Icon, Ruby e Python, entre outras.

Lua foi criada em 1993 por Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes, membros do Instituto Tecgraf (*Computer Graphics Technology Group*) na PUC-Rio, a Pontifícia Universidade Católica do Rio de Janeiro, no Brasil. Conforme Ierusalimschy, Figueiredo e Celes (2006), a linguagem de programação Lua combina programação procedural com poderosas estruturas para detalhamento de informações, fundamentadas em tabelas relacionadas e semântica extensível. É tipada dinamicamente, e tem gerenciamento automático de memória com coleta de lixo. Essas características fazem da linguagem de programação Lua, uma linguagem ideal para configuração, automação (*scripting*) e prototipagem dinâmica.

Devido à sua efetividade, simplicidade e facilidade de aprendizado, foi 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 comumente utilizada como uma linguagem de objetivo geral.

3. Metodologia

Para implementação do jogo digital proposto neste trabalho, nomeado de *Computer Invader*, na linguagem Lua foram utilizadas duas ferramentas: ZeroBrane Studio IDE e LÖVE, estas ferramentas são, respectivamente, um ambiente de desenvolvimento para programas de computador desenvolvidos em Lua e um *framework* para o desenvolvimento de jogos em duas dimensões (2D) empregando Lua.

O ZeroBrane Studio é um ADI (ambiente de desenvolvimento integrado), ou IDE (*Integrated Development Environment*) em inglês, Lua de código aberto leve com preenchimento de código, realce de sintaxe, analisador de código, *live coding* e suporte a depuração para vários interpretadores Lua como Lua 5.1, Lua 5.2, Lua 5.3, LuaJIT e o LÖVE já que o mesmo traz consigo seu próprio interpretador Lua com seu nome. O ZeroBrane Studio é um aplicativo multiplataforma produzido em Lua e interpretado nos sistemas operacionais Windows (Windows XP+), Linux e MacOS (10.7+). O ZeroBrane Studio pode ser adquirido no seguinte endereço eletrônico: https://studio.zerobrane.com/download?not-this-time.

LÖVE (ou Love2D) é um motor de jogos 2D, multiplataforma e Open Source. Criado em C++, é um espaço que possui diversos códigos pré-escritos orientados para o desenvolvimento de jogos e se conecta com a linguagem de programação Lua. Seu interpretador Lua proporciona alcance às funcionalidades de som e de vídeo de modo fácil e eficiente carregando consigo o potencial da portabilidade, ou seja, um jogo poderia funcionar em várias plataformas tendo o mesmo código fonte.

Uma versão chamada piLöve que foi produzida com ênfase especificamente no Raspberry Pi. Esse *framework* é frequentemente encontrado nas competições de desenvolvimento de videogames, como a competição internacional Ludum Dare. LÖVE pode ser baixado no seguinte link: https://love2d.org/.

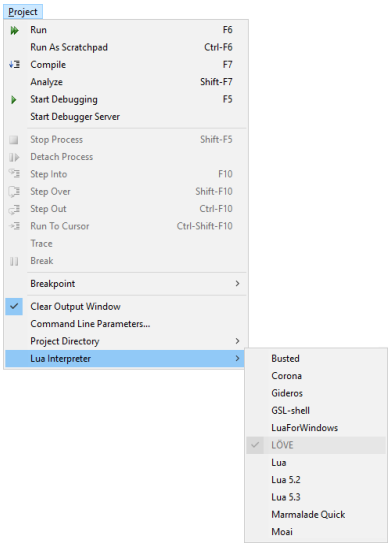

Figura 1. Menu do ZeroBrane Studio com destaque para a lista de interpretadores Lua disponíveis e com LÖVE selecionado.

4. Aplicação

Computer Invader, nome dado ao jogo implementado, é uma simples aplicação para computador desktop baseada no famoso jogo de arcade *Space Invaders*. O jogo consiste em uma espaçonave enfrentando inimigos que estão descendo a tela gradativamente. Os controles consistem em movimentar a nave para os lados e atirar, tudo descrito na tela do jogo antes de iniciar. Para iniciar espera-se que o usuário aperte a tecla *enter* do teclado.

O jogo tem três fases e cada uma delas começa com um grupo de inimigos com uma velocidade de descida, no canto da tela é informado o tempo, a pontuação e o número da fase em que o jogador se encontra. O usuário passa para a próxima fase ao derrotar todos os inimigos, ganhando mais munição e mais velocidade, no entanto, o número de inimigos aumenta como também a velocidade de descida dos mesmos. O jogo termina se todas as fases foram vencidas ou se algum inimigo tocar o solo do nível da nave. Para ambos os casos aparece uma mensagem de sucesso ou falha na tela. Se o jogo chegar ao fim, a tecla "R" deve retornar o usuário à tela inicial. Como explicado na seção anterior, seu desenvolvimento foi feito utilizando a ZeroBrane Studio IDE utilizando os recursos do framework LÖVE.

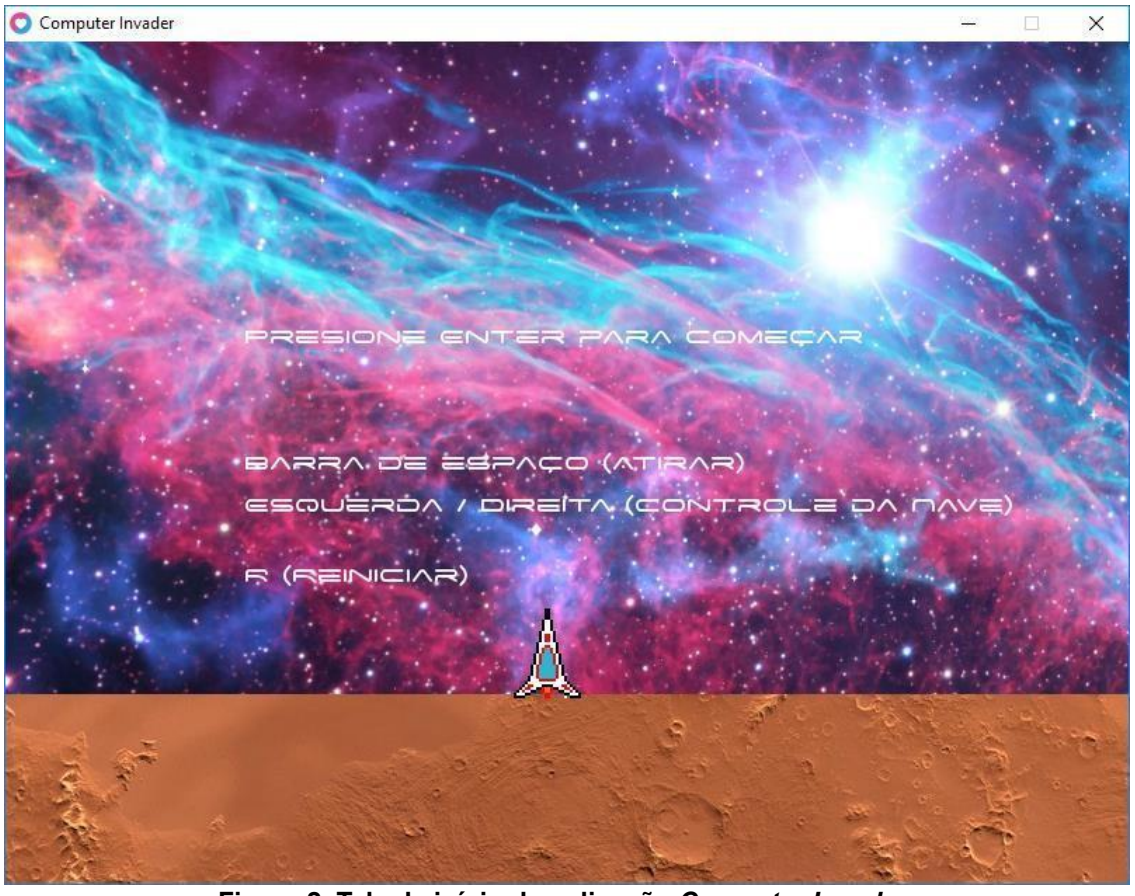

Figura 2. Tela de início da aplicação *Computer Invader*.

Quando começou-se a escrever jogos usando o LÖVE, é necessário ter conhecimento prévio sobre Lua, pois é com ela que se dirá ao LÖVE o que fazer. Em seguida deve-se conhecer alguns requisitos sobre como se estrutura o jogo. Primeiro, deve-se criar uma pasta para guardar o projeto e nomeá-la. Em seguida, deve-se criar um arquivo dentro desta pasta chamado *main.lua*. Este roteiro Lua é executado quando o jogo é iniciado; daqui pode-se carregar outros scripts, recursos e assim por diante.

Finalmente, deve-se inserir todos recursos que o jogo necessita (imagens, músicas, animações, efeitos sonoros, etc.) dentro desta pasta. Com isso, a próxima tarefa seria aprender a API (*Application Programming Interface*) do LÖVE. A API é dividida em módulos. Cada módulo (que é apenas uma tabela) vive dentro do módulo (novamente uma tabela) chamado *love*. Exemplos de módulos são *love.graphics* e *love.filesystem*. Cada um desses módulos contém funções que fazem certas coisas. Por exemplo, *love.graphics.circle* permite desenhar um círculo na tela. No entanto, para realmente criar algum jogo as funções *callback* são as partes mais importantes da API.

Uma função *callback* é uma função passada a outra função como argumento, que é então invocado dentro da função externa para completar algum tipo de rotina ou ação. A chamada de uma função regular como *love.graphics.draw* ou *math.floor*, significa que o LÖVE ou Lua farão alguma coisa. Um *callback*, por outro lado, é uma função que o desenvolvedor codifica e LÖVE chama em determinados momentos. Isso facilita manter seu código organizado e otimizado.

Os principais *callbacks* são: *love.load* para fazer uma configuração única do seu jogo; *love.update*, usado para gerenciar o estado do jogo quadro a quadro, e *love.draw* que é usado para renderizar o estado do jogo na tela. O LÖVE fornece *callbacks* padrões para esses, que pode-se substituir dentro do seu próprio código, criando sua própria função com o mesmo nome.

```
1  --Essa função é chamada sempre que uma tecla do teclado é liberada e recebe a chave liberada.
2  function love.keyreleased(key)
3    -- na v0.9.2 e mais recentes espaço é representado pelo caractere ' ', entao checo pelos dois
4    if (key == " " or key == "space") then
5      -- espaço foi apertado, chega se o jogo ta rolando, se sim, cria tiros na tabela de tiros (hero.shots)
6      if game.state == 1 then
7        shoot()
8      end
9    end
10   if key == "return" then
11     -- enter foi apertado, se o jogo não começou inicializa-se
12     if game.state == 0 then
13       game.state = 1
14     end
15   end
16   if key == "r" then
17     -- r foi apertado se o jogo terminou de alguma forma reseta o jogo chamando love.load manualmente
18     if game.over or game.state == 2 then
19       love.load()
20     end
21   end
22 end
```

Figura 3. Trecho de código: a utilização do *callback love.keyreleased* na aplicação.

As funções de *callback* na API LÖVE são chamadas por *love.run*, para executar várias tarefas e são todas opcionais. No entanto, uma experiência de jogo repleta de recursos provavelmente utilizaria quase todos eles. A aplicação em questão fez uso apenas dos principais (*load*, *update* e *draw*) adicionando mais um chamado *love.keyreleased*.

```lua
function love.run()
    if love.load then love.load(love.arg.parseGameArguments(arg), arg) end

    -- We don't want the first frame's dt to include time taken by love.load.
    if love.timer then love.timer.step() end

    local dt = 0

    -- Main loop time.
    return function()
        -- Process events.
        if love.event then
            love.event.pump()
            for name, a,b,c,d,e,f in love.event.poll() do
                if name == "quit" then
                    if not love.quit or not love.quit() then
                        return a or 0
                    end
                end
                love.handlers[name](a,b,c,d,e,f)
            end
        end

        -- Update dt, as we'll be passing it to update
        if love.timer then dt = love.timer.step() end

        -- Call update and draw
        if love.update then love.update(dt) end -- will pass 0 if love.timer is disabled

        if love.graphics and love.graphics.isActive() then
            love.graphics.origin()
            love.graphics.clear(love.graphics.getBackgroundColor())

            if love.draw then love.draw() end

            love.graphics.present()
        end

        if love.timer then love.timer.sleep(0.001) end
    end
end
```

Figura 4. Trecho de código: *love.run* padrão de acordo com a documentação de LÖVE.

Em *love.load* a aplicação carrega todo o material necessário para o jogo como fontes, imagens, variáveis e constantes referentes ao jogo. Na *love.update* podemos atualizar informações de tempo, pontuação e fase atual, determinar o deslocamento dos inimigos, da espaçonave e de seus tiros, checar se houve colisões como as dos tiros com inimigos e as dos mesmos com o nível de chão e assim determinar o estado do jogo. A *love.draw* renderiza todo o material do estado atual do jogo na tela, tendo destes três estados: tela de apresentação, com instruções de uso; o desenrolar das fases, com inimigos, espaçonave, tiros e as informações de jogo (tempo, fase e pontuação) no canto da tela; e a tela de fim de jogo, com mensagens de sucesso ou derrota.

O repositório da aplicação pode ser encontrado na plataforma GitHub, por meio da URL https://github.com/aretw0/computer-invader, para jogá-la é preciso realizar a instalação do *framework* LÖVE e executar um dos arquivos com os formatos *.love* ou *.exe* do arquivo pasta /releases.

5. Conclusão

Procurou-se ilustrar neste artigo algumas das características mais marcantes da linguagem Lua. Abordamos características e aplicações. Foi buscado também enfatizar as potencialidades desse mecanismo de programação em sistemas mais complexos.

Devido a restrições de tamanho, não foram abordados vários outros aspectos importantes da sintaxe Lua. Em particular, o tratamento da linguagem Lua foi bastante introdutório, por ser uma ferramenta não muito convencional, foi necessário introduzir vários conceitos básicos novos para podermos chegar a tópicos mais avançados.

Também não foram abordadas as bibliotecas padrão de Lua, algumas são convencionais, como a biblioteca de manipulação de arquivos. Mas outras, como a biblioteca de strings com suas funções de casamento de padrões, merecem um tratamento mais longo. Espera-se que o leitor, após este texto introdutório, sinta-se motivado a aprofundar seus conhecimentos sobre Lua, pesquisando mais a respeito das particularidades presentes nas técnicas de programação desta linguagem.

Lua tem uma merecida reputação de excelente desempenho. Foi utilizado todo o potencial da linguagem Lua no paradigma funcional para abstrair rotinas através da passagem de argumentos em cada chamada, conseguindo resultados satisfatórios com a aplicação final de acordo com a teoria. O ZeroBrane Studio facilitou bastante o desenvolvimento e depuração da aplicação, além de dispor de muitos tutoriais sobre Lua e LÖVE. Utilizar o LÖVE foi bastante simples e intuitivo, sua API (*application programming interface*) é ótima, o ambiente em geral é agradável e a comunidade é muito acolhedora.

Como trabalhos futuros, podemos personalizar ainda mais o jogo utilizando outras imagens para espaçonaves, cenários e inimigos, adicionar uma trilha sonora além de implementar fases com dificuldades específicas de cada etapa com mais níveis de complexidades. Da mesma forma, se pretende adicionar futuramente um suporte ao armazenamento da pontuação e assim gerar um ranking dos melhores jogadores que utilizaram a aplicação.

Referências

André R. (2005). Introdução à Lua, uma poderosa linguagem de programação.

Araújo, É. M. D., Sellanes, R. G., & Costa, A. C. (1998). Cálculo lambda: um formalismo para funções. *Salão de Iniciação Científica (10.: 1998: Porto Alegre). Livro de resumos. Porto Alegre: UFRGS*, 1998.

Bove, A., Dybjer, P., & Norell, U. (2009). A brief overview of Agda–a functional language with dependent types. In *International Conference on Theorem Proving in Higher Order Logics* (pp. 73-78). Springer, Berlin, Heidelberg.

Ierusalimschy, R. (2006). *Programming in Lua*. Em: Rio de Janeiro: lua.org, páginas 252. ISBN 85-903798-2-5.

Ierusalimschy, R., Figueiredo, L. e Celes, W. (2006). *Lua Reference manual*. Em: Rio de Janeiro: lua.org, páginas 103. ISBN 85-903798-3-3.

Jung, K. e Brown, A. (2007). *Beginning Lua Programming*. Em: Indianapolis: Wiley Publishing, páginas 644. ISBN 0-470-06917-2.

LÖVE. (2018) "Beginning to Write Games Using LÖVE", https://love2d.org/wiki/love, Março.

Linuxtopia. (2014) "*Package Manifest*", https://www.linuxtopia.org/online_books/rhel6/rhel_6_technical_notes/rhel_6_technotes_package_manifest.html, Março.

Lua (2018[a]) "*News*", https://www.lua.org/news.html, Março.

Lua (2018[b]) "*What is Lua?*", https://www.lua.org/about.html, Março.

MDN Web Docs - Mozilla (2017). “Função Callback”, https://developer.mozilla.org/pt-BR/docs/Glossario/Callback_function, Abril.

NovaFusion (2011). “*A Guide to Getting Started with Love2D*”, http://nova-fusion.com/2011/06/14/a-guide-to-getting-started-with-love2d/, Abril.

O Globo (2015). “Linguagem Lua, criada no Brasil, tem primeiro livro em português sobre programação”, https://oglobo.globo.com/sociedade/tecnologia/linguagem-lua-criada-no-brasil-tem-primeiro-livro-em-portugues-sobre-programacao-16685148, Março.

Open Hub (2018). *The Battle for Wesnoth Open Source Project*.

Sampaio, A. (2018) “Paradigmas de Linguagens de Programação”, https://www.cin.ufpe.br/~in1007/, Março.

Seliger, P. (2014) “*Lecture Notes on the Lambda Calculus*”, https://www.mscs.dal.ca/~selinger/papers/papers/lambdanotes.pdf, Março.

Sumana H. (2013) “*New Lua templates bring faster, more flexible pages to your wiki*”, https://blog.wikimedia.org/2013/03/11/lua-templates-faster-more-flexible-pages/, Abril.

Waldemar C., Luiz H. e Roberto I. (2004), A Linguagem Lua e suas Aplicações em Jogos.

Wall, L. (1999) “*Lua, the first postmodern computer language*”, http://www.wall.org/~larry/pm.html, Março.