FACULDADE DE TECNOLOGIA DE SÃO PAULO

Rafael Hiroshi Ooki

Linguagem Lua: nova estrutura e aplicações

São Paulo
2013

FACULDADE DE TECNOLOGIA DE SÃO PAULO

Rafael Hiroshi Ooki

Linguagem Lua: nova estrutura e aplicações

Monografia submetida como exigência
parcial para a obtenção do Grau de
Tecnólogo em Processamento de Dados
Orientador: Prof. Valter Yogui

São Paulo
2013

Dedicatória

Aos meus pais por terem me incentivado, guiado e me ajudado a chegar até aqui, aos meus amigos por estarem me apoiando ao longo do caminho.
Ao professor orientador Valter Yogui, por ter aceitado orientar meu projeto e pela paciência comigo durante o desenvolvimento do mesmo.

Agradecimento

Aos meus pais e minha família por me ajudarem até aqui, aos meus amigos por estarem me apoiando e incentivando ao longo do caminho.
Agradeço também ao professor orientador Valter Yogui, por ter aceitado orientar meu projeto e pela paciência comigo durante o desenvolvimento do mesmo me ajudando e auxiliando sempre que necessário.

Resumo

Este documento tem como intuito descrever sobre a linguagem de programação Lua, suas características e aplicações no mercado, mostrando as áreas em que a linguagem mais atua e a partir deste estudo analisar pontos como evolução da linguagem, compatibilidade com versões anteriores e assim verificar como ela influência os softwares e aplicações que utilizamos normalmente.

Palavras-chave: Linguagem de Programação Lua, Lua, Linguagem de Extensão, scripts

Abstract

This paper has as purpose to describe on the Lua programming language, its features and applications on the market, showing the areas where the language is most active and from this study points to analyze the evolution of language, compatibility with previous versions and see how well it influence software and applications that we use normally.

Keywords: Programming Language Lua, Lua, Language Extension, scripts

Lista de Ilustrações

Figura 1 – Data Entrry Language (DEL) 12
Figura 2 – Programa Gráfico Master (PGM) 13
Figura 3 – Gráfico de evolução de código 14
Figura 4 – Gráfico de pesquisa de linguagem 18
Figura 5 – Exemplo de uma implementação NCLua 41
Figura 6 – Imagem do jogo Monkey Island 43
Figura 6 – Página inicial do Publique! 45

Lista de Tabelas

Tabela 1 – Evolução das versões Lua.. **Erro! Indicador não definido.**
Tabela 2 – Ranking de Linguagem de Programação................. **Erro! Indicador não definido.**

Sumário

1. Introdução .... Erro! Indicador não definido.
2. Conceitos básicos .... Erro! Indicador não definido.**0**
3. Histórico **11**
3. 1. Origem .... **Erro! Indicador não definido.**1
3. 2. Evolução 14
4. Linguagem Lua .... Erro! Indicador não definido.
4. 1. Características 20
4. 2. MetaTabelas 21
4. 3. Valores e Tipos 21
4. 4. Variáveis 25
4. 5. Bibliotecas 25
4. 5. i. Biblioteca Padrão 25
4. 5. ii. Biblioteca Auxiliar .... **Erro! Indicador não definido.**
4. 6. Portabilidade e Tamanho 34
4. 7. Programação Orientada a Objetos 35
4. 8. Incluindo Lua 35
5. Versão 5.2 .... **36**
5. 1. Módulos 37
5. 2. Ambientes 37
5. 3. Co-Rotinas 38
5. 3. Coletor de Lixo .... **Erro! Indicador não definido.**
5. 3. Tratamento de Erros 40
5. 3. Inconsistências Apresentadas 40
6. Aplicações **40**
6. 1. Ginga 41
6. 2. Software .... **Erro! Indicador não definido.**
6. 2. Games .... **Erro! Indicador não definido.**
6. 2. Internet 45
6. 5. Malwares 46
7. Cenário Atual **46**
8. Considerações finais **47**

1 – Introdução

A tecnologia avança cada vez mais rápido e com isso várias coisas ao nosso redor sofrem duras mudanças, desde o modo que nos comunicamos até a maneira como nós interagimos com as outras pessoas. Em consequência dependemos cada vez mais de meios eficientes de comunicação e interação, ou seja, máquinas que nos proporcionem com maior velocidade o fluxo de informações requeridas por nós naquele momento, seja um jogo, uma notícia ou até mesmo um vídeo. Com isso buscamos sempre computadores, celulares, notebooks, entre outros, com a melhor configuração, para que assim possamos desfrutar da melhor tecnologia oferecida; mas para isso não é somente necessário uma maquina boa é necessário investir por trás um software bom o suficiente que rode por detrás dessa aparelhagem toda sem comprometer seu desempenho.

Várias linguagens de programação são utilizadas para fazer os mais diversos programas para os mais diversos meios e todas elas possuem suas características próprias, como estrutura, convenções léxicas, tratamentos de erros, para que plataforma será utilizada, etc.. Dentre todas essas linguagens uma que está se destacando a um tempo no mundo é a linguagem de programação Lua. Este trabalho tem como objetivo explicar sobre a linguagem de programação Lua, uma linguagem de programação de origem brasileira, sua estrutura atual e sua participação ao redor do mundo. Este trabalho visa explicar sobre a linguagem de programação Lua sem se aprofundar tanto nos comandos da linguagem e seu respectivo funcionamento, dando uma visão mais ampla sobre aspectos da linguagem quanto a utilização, características e situação no mercado atualmente. O próximo tópico busca explicar conceitos que serão referidos ao longo deste trabalho.

2 - Conceitos Básicos

Este tópico visa explicar conceitos que irão aparecer ao longo do trabalho para que assim o leitor possa acompanhar partindo do pressuposto que este já possua conhecimentos básicos sobre a área de tecnologia da informação.

Sistemas embarcados – sistemas microprocessados no qual o computador é totalmente encapsulado ou dedicado ao dispositivo que ele controla, sendo que um sistema embarcado irá realizar tarefas pré-definidas com requisitos específicos. Em geral esse tipo de sistema não pode ter sua funcionalidade alterada durante sua utilização, caso isso seja

necessário é preciso reprogramar todo o sistema.

Linguagens de extensão - ou linguagem de script são linguagens de programação que são geralmente interpretadas em vez de serem compiladas, uma de suas principais características é que em uma linguagem de script o código é interpretado um comando de cada vez conforme ele é chamado pelo programa principal.

Sendo que este interpretador pode fazer isso de algumas maneiras como executar o código fonte diretamente, executar o código pré-compilado, entre outros. Como exemplo de linguagens de extensão temos além de Lua, linguagens como *Python, Ruby, Perl, etc.*

Linguagens dinâmicas – também conhecidas por linguagem de script são linguagens que possuem duas características fundamentais que as diferenciam das linguagens estáticas que são a tipagem dinâmica e o protocolo de meta-objeto (Meta-Object Protocol).

Reentrância – em programação refere-se à qualidade em que uma sub-rotina pode ser executuada concorrentemente de maneira segura, ou seja, a sub-rotina pode ser invocada enquanto está sendo executada. Para que isso seja atendido não a sub-rotina deve trabalhar com dados dinâmicos, trabalhando somente com os dados fornecidos pela própria sub-rotina que a chamou.

3 – Histórico

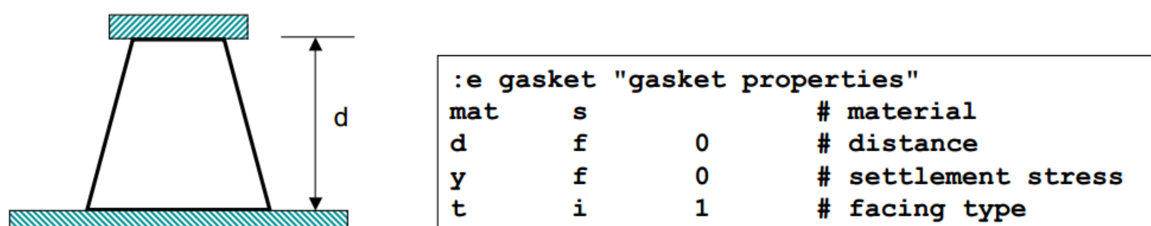
3.1 - Origem

A Petrobras em 1993 precisava analisar e gerar um mapa com os dados de poços de petróleo, para isso fora instalados na broca de perfuração várias sondas para que assim durante a perfuração pudesse ser recolhidos dados sobre o poço. Esses dados continham as mais diversas informações sobre temperatura, pressão, resistência, profundidade, condutividade dos materiais naquela profundidade, entre outros; e essas informações eram mostradas para os geólogos e especialistas que trabalhavam nas perfurações. Para isso era utilizado vários programas que pegavam os dados e convertiam de maneira gráfica as informações, mas esses programas tinhas suas limitações e sempre que era necessário verificar algo e isso não pudesse ser feito então era necessário requisitar ao desenvolvedor

do software uma atualização que atendesse aos requisitos dos especialistas na época. Essas atualizações nem sempre atendiam os requisitos necessários para a continuidade do projeto, então era necessário que fosse desenvolvida uma solução que pudesse prover maior agilidade aos geólogos na resolução desses problemas e em tempo real para que assim não houvesse tempos muito longos de espera de uma atualização para outra, então a Petrobrás que tinha a necessidade de suprir a dependência que seus geólogos tinham por atualizações realizadas pelos desenvolvedores da aplicação, então por eles usados em seus projetos e trabalhos foi até o do laboratório de pesquisa e desenvolvimento de Tecnologia e Computação Gráfica (TecGraf) da Pontifícia Universidade Católica do Rio de Janeiro (PUC -RJ), com quem tem parceria, requisitar uma solução definitiva para esse problema.

Para esse problema Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes, membros do TecGraf se uniram para desenvolver essa solução. A princípio fora necessário estudar alguns softwares e linguagens de programação já desenvolvidas antes para a Petrobras para a solução desse problema e a partir das necessidades já apresentadas pelos funcionários da Petrobras fora verificado duas linguagens de programas uma utilizada pelos geólogos para a obtenção de dados a *Data Entry Language* (DEL) e a outra a *Simple Object Language* (SOL) é uma linguagem desenvolvida que nunca fora lançada.

Figura 1 – Data Entry Language (DEL)

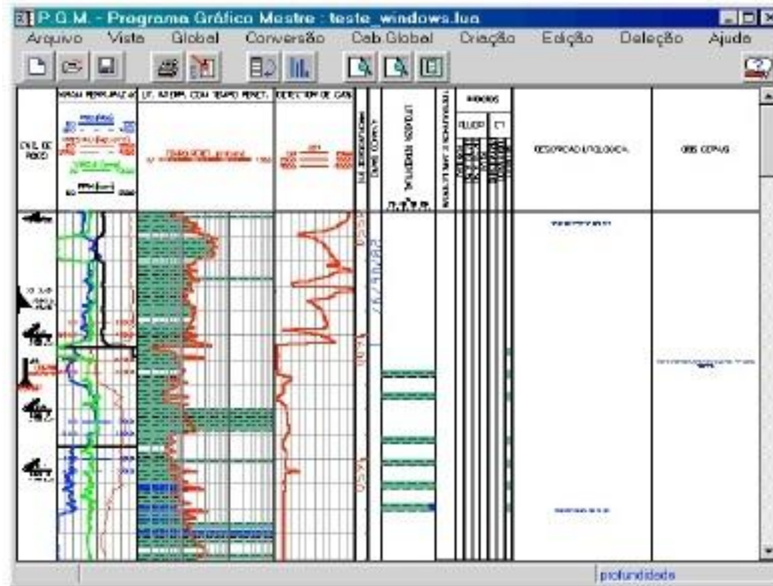


Fonte http://pt.scribd.com/doc/91344/A-Linguagem-Lua-e-suas-aplicacoes-em-jogos

A *Data Entry Language* (DEL) era utilizada para receber uma variedade de dados de entrada e assim processar esses dados e transforma-los em estruturas gráfica, DEL obteve sucesso ao ser implementado, mas ao longo do tempo foi exigido mais dessa linguagem e ficou claro que era necessária uma linguagem de programação melhor e mais precisa. Nessa mesma época Roberto Ierusalimschy e Waldemar Celes começaram a trabalhar no Programa Gráfico Mestre (PGM), um programa para gerar relatórios configuráveis para

visualização de perfis geológicos.

Figura 2 – Programa Gráfico Mestre - P.G.M.



http://pt.scribd.com/doc/91344/A-Linguagem-Lua-e-suas-aplicacoes-em-jogos

A Simple Object Language (SOL) era uma linguagem de programação para a descrição de objetos cuja a sintaxe fora inspirada no BibTeX, uma ferramenta de formatação de texto que era utilizada por LaTeX (um conjunto de macros para processador de texto TeX), era uma linguagem que seria utilizada com o Programa Gráfico Mestre, mas que não foi implementada dado que a linguagem poderia ser remodelada para que fosse possível desenvolver gráficos mais precisos e com mais dado0073, ambas as linguagens tinham vários problemas em comum e baseando-se nelas buscaram fazer uma nova linguagem de programação. Como estavam largando a Sol um amigo deles sugeriu outro nome para a nova linguagem que estava por vir.

Em 1993 nasce à linguagem de programação Lua, uma linguagem de programação que foi feita para que os engenheiros e geólogos da Petrobrás que na época precisavam de um software que conseguisse ler a pressão nas rochas, temperatura d'agua, profundidade dos poços, entre outras coisas e para atender diferentes finalidades partindo de uma linguagem de simples compreensão e poderosa para o desenvolvimento das atualizações, modificação e criação no software na hora que eles precisassem e que qualquer um que tivesse conhecimentos sobre programação pudesse desenvolver em cima dessa linguagem. Lua quando surgiu foi implementada como uma biblioteca e sua primeira versão tinha menos de 6000 linhas de código em linguagem C sendo pré-compilada com yacc/lex.

Ao criarem a linguagem Lua, Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes, tinham a expectativa de solucionar os problemas apresentados por DEL e SOL, Lua conseguiu superar as expectativas e logo seguiu para outros projetos dentro da Petrobras e dos laboratórios da Tecgraf, sendo posteriormente liberada para que qualquer um pudesse utiliza-la.

3.2 – Evolução

Lua ao longo do tempo foi uma linguagem que evoluiu lenta e gradativamente conforme as necessidades que iam aparecendo de seus usuários a cada versão era incluído novos recursos, novas funções, porque a complexidade exigida dos programas estava aumentando, então a linguagem Lua precisava acompanhar essa mudança.

O gráfico a seguir mostra a quantidade de linhas de código usadas para o desenvolvimento de cada versão da linguagem Lua, nota-se que a partir da versão 2.5 houve um aumento gradual do número de linhas que foram usadas para o desenvolvimento da linguagem.

Figura 3 – Gráfico de evolução de código

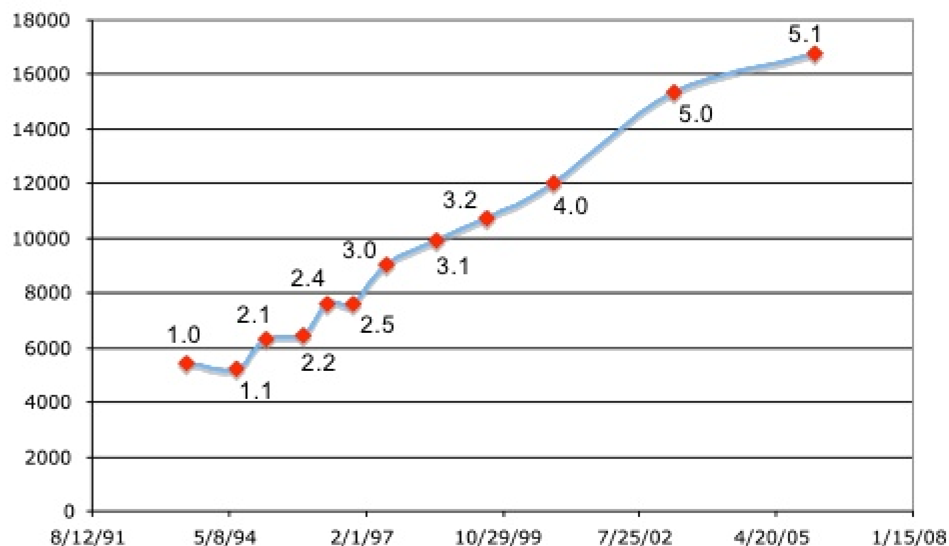


Imagem retirada de: http://www.inf.puc-rio.br/~roberto/talks/luapyconf.pdf

Desde a primeira versão de Lua, muitas coisas em sua estrutura foram sofrendo alterações, ao ser lançada em 1993. Em sua primeira versão Lua apresentava sete tipos de variáveis *numbers*, *strings*, *tables*, *nil*, *userdata* (utilizado para apontar para objetos em

C), Lua *functions* e C functions, em função de deixar a linguagem pequena a princípio não fora incluído o tipo boolean e sua versão era para fins industriais e não acadêmicos. Ao ser lançada a versão 1.1 de Lua o software passou a ser distribuído por meio de ftp, antes de ser código-aberto e podia ser usado livremente para fins acadêmicos, para fins comerciais ainda era necessário pedir autorização. Nos anos 90's com o crescimento da programação orientada a objetos e consequentemente a constante pressão dos usuários por recursos orientados a objetos dentro da linguagem Lua, fora implementado mecanismos que permitisse ao usuário a flexibilidade de construir modelos que se encaixassem em suas aplicações, sem tornar a linguagem Lua em uma linguagem orientado a objetos.

O lançamento da versão 2.1 foi marcado pela introdução dos mecanismos de extensão, esses mecanismos de extensão logo se tornaram a marca registrada de Lua. Um dos objetivos dos mecanismos de extensão é fazer com que as tabelas sejam usadas como base para objetos e classes, para isso fora necessário incluir o conceito de herança para as tabelas. Outro objetivo era tornar o tipo *userdata* em uma aplicação de proxies para a transferência de dados mais natural possível para o programador, buscando assim possibilitar chamar funções dentro tabelas a partir do índice delas, retornando o resultado para elas mesmas. Junto com esse mecanismo de extensão foi adicionado outro mecanismo que é o *fallback* que consiste em permitir ao programador escolher como Lua irá se comportar caso ocorra algum erro durante uma operação.

Tendo em vista que ao dificultar o acesso a linguagem Lua para fins comerciais os universitários ficavam desencorajados a aprender Lua em sua versão 2.1 foi decidido lançar um software de uso livre. Em sua versão 2.2 foi implementado a *debug Application Programming Interface (API)* que permitia visualizar informações sobre as funções que estavam sendo executadas em tempo real, como resposta aos vários pedidos para que fosse implementado uma função debug.

Em sua versão 2.4, Lua sofreu uma reestruturação no código tendo seu código separado em módulos, com isso um programa conseguia ser compilado com apenas 3% de código Lua pré-compilado, isso era um ganho caso fosse utilizado em dispositivos móveis, robôs e sensores. Após o lançamento da versão 2.5 foi lançado um artigo na publicação da magazine Dr. Dobb's Journal falando sobre Lua, com isso a linguagem conseguiu a atenção de várias pessoas da indústria, sendo uma delas Bret Mogilefsky, programador chefe do jogo Grim Fandango, da LucasArts. Em 1999, em uma conferência de desenvolvedores de games, Bret Mogilefsky falou sobre o sucesso que teve utilizando

Lua em seu jogo. Com essa exposição no cenário internacional e com inúmeros jogos na época começando a aderir ao uso de Lua houve um aumento considerável no número de inscritos na comunidade voltada a discutir a linguagem Lua.

Em sua versão 3.0 (1997), houve uma junção dos tipos *Lua functions* e *C functions* no tipo único *function* e o mecanismo *fallback* que fora implementado na versão 2.1 para auxiliar o mecanismo de extensão era um mecanismo global sendo possível somente uma chamada por evento, isso dificultava bastante o compartilhamento e o reuso, com isso o mecanismo *fallback* foi substituído por *tag methods,* sendo utilizado dois dados para incluir uma chamada de evento (evento e *tag*). As *tags* foram introduzidas na versão 2.1 só que na época Lua não tinha força o suficiente para trabalhar com a interpretação das *tags*, para corrigir isso eles estenderam sua área de atuação para todos os valores, conseguindo assim suportar *tag methods.*

A versão 3.1 trouxe a lua a programação funcional, nesse período também foi discutido sobre *multithreading* e *cooperative multitasking*, para mitigar esse problema foi incluído múltiplos estados independentes que podiam ser trocados em tempo de execução. Com a chegada da versão 4.0 lançada em novembro de 2000, foi apresentado a API de reentrância, motivado por aplicações que necessitavam de múltiplos estados e inserido o laço de repetição "*for*", em concorrente estava sendo desenvolvido a versão 4.1 do Lua visando implementar o *multithreading,* mas foi somente implementado um mecanismo básico de múltiplos stacks que fora chamado de threads. E na versão 4.1 Alpha lançada em julho 2001 utilizando-se do mesmo mecanismo que fora implementado nas co-rotinas foi criado o *multithreading*.

Ao ser lançada a versão 5 dois novos tipos foram introduzidos a linguagem, que são o boolean e o threads e a função de cooperative multitasking foi finalmente implantada. Com a versão 5.1 foi implementado a função "marca-e-limpa" para o Coletor de Lixo corrigindo o problema de grandes esperas entre uma varredura e outra.

A tabela abaixo mostra a evolução dos recursos e componentes que a linguagem Lua oferece para seus usuários ao longo de suas versões nota-se ao analisar o gráfico desde a primeira versão até a versão 5.1, muitas funções foram criadas, juntamente com essa evolução houve um aumento nos recursos dando a linguagem mais liberdade na criação e mais opções para o desenvolvimento de aplicações, e uma mudança no tipo da maquina

virtual que passou a ser de pilha para registradores

Tabela 1 – Evolução das versões Lua

	1.0	1.1	2.1	2.2	2.4	2.5	3.0	3.1	3.2	4.0	5.0	5.1
constructors	●	●	●	●	●	●	●	●	●	●	●	●
garbage collection	●	●	●	●	●	●	●	●	●	●	●	●
extensible semantics	○	○	●	●	●	●	●	●	●	●	●	●
support for OOP	○	○	●	●	●	●	●	●	●	●	●	●
long strings	○	○	○	●	●	●	●	●	●	●	●	●
debug API	○	○	○	●	●	●	●	●	●	●	●	●
external compiler	○	○	○	○	●	●	●	●	●	●	●	●
vararg functions	○	○	○	○	○	●	●	●	●	●	●	●
pattern matching	○	○	○	○	○	●	●	●	●	●	●	●
conditional compilation	○	○	○	○	○	○	●	●	●	○	○	○
anonymous functions, closures	○	○	○	○	○	○	○	●	●	●	●	●
debug library	○	○	○	○	○	○	○	○	●	●	●	●
multi-state API	○	○	○	○	○	○	○	○	○	●	●	●
`for` statement	○	○	○	○	○	○	○	○	○	●	●	●
long comments	○	○	○	○	○	○	○	○	○	○	●	●
full lexical scoping	○	○	○	○	○	○	○	○	○	○	●	●
booleans	○	○	○	○	○	○	○	○	○	○	●	●
coroutines	○	○	○	○	○	○	○	○	○	○	●	●
incremental garbage collection	○	○	○	○	○	○	○	○	○	○	○	●
module system	○	○	○	○	○	○	○	○	○	○	○	●

	1.0	1.1	2.1	2.2	2.4	2.5	3.0	3.1	3.2	4.0	5.0	5.1
libraries	4	4	4	4	4	4	4	4	5	6	8	9
built-in functions	5	7	11	11	13	14	25	27	35	0	0	0
API functions	30	30	30	30	32	32	33	47	41	60	76	79
vm type (stack × register)	S	S	S	S	S	S	S	S	S	S	R	R
vm instructions	64	65	69	67	67	68	69	128	64	49	35	38
keywords	16	16	16	16	16	16	16	16	16	18	21	21
other tokens	21	21	23	23	23	23	24	25	25	25	24	26

tabela retirada de http://www.lua.org/doc/hopl.pdf

Lua ao decorrer dos anos teve o seu número de usuários ativos crescer consideravelmente dado a grande repercussão que teve quando Bret Mogilefsky relatou sobre sua experiência de sucesso com a linguagem de programação Lua em seu jogo Grim Fandango e em paralelo o artigo na magazine Dr. Dobb's Journal e com isso passou a ser utilizada com mais frequência por desenvolvedores de games e a ser pesquisada por curiosos em aprender sobre a linguagem e por profissionais da área que buscam aprender e aprimorar seus conhecimentos acerca da linguagem. Atualmente Lua se encontra entre as 20 linguagens mais pesquisadas na internet.

Figura 4 – Gráfico que mostra o quão a linguagem foi pesquisada ao longo dos anos

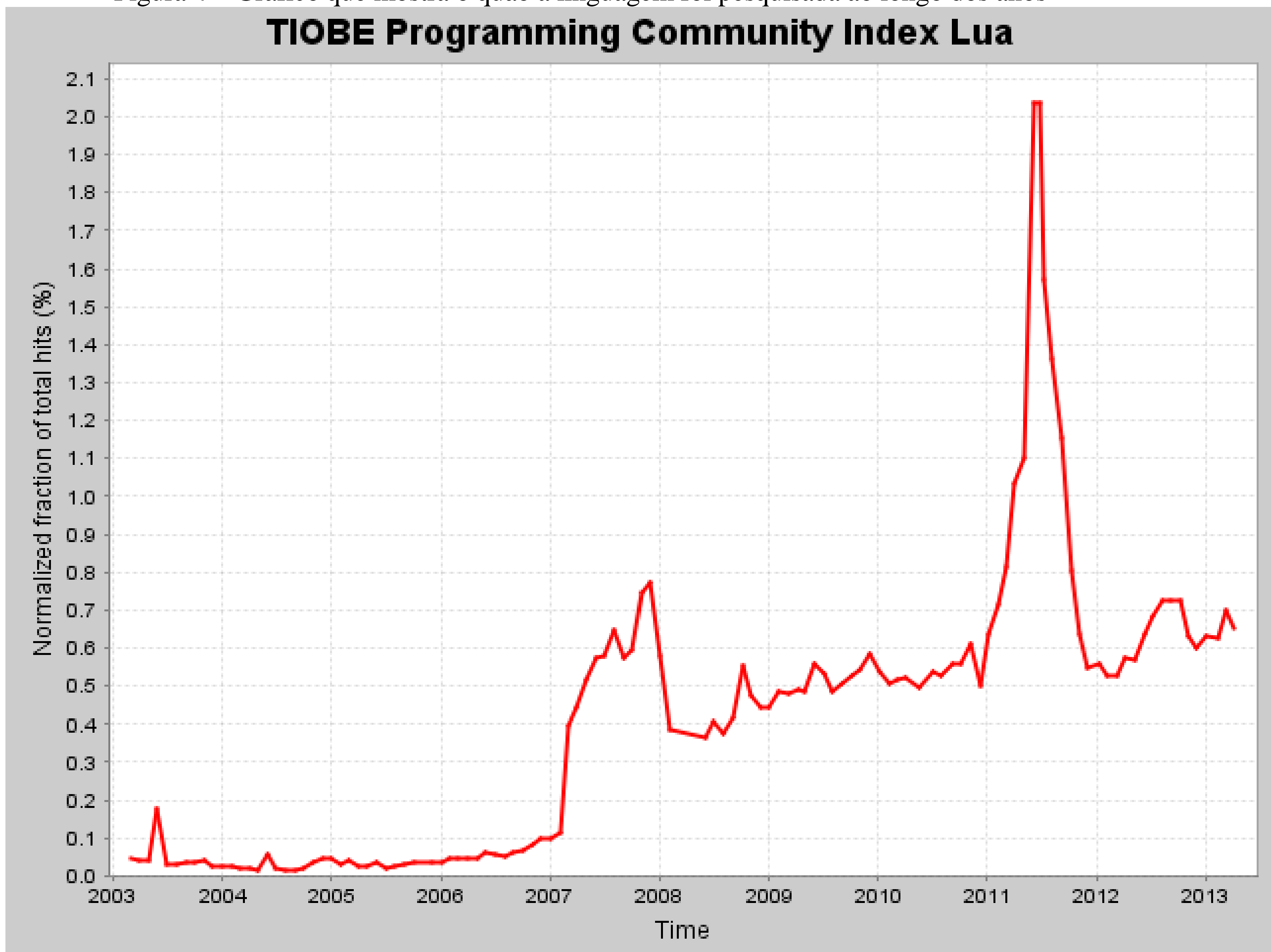


Gráfico emitido pelo site TIOBE mostra a taxa ao longo dos anos sobre o quão a linguagem foi pesquisada

Uma tabela emitida pelo site da TIOBE prepara e mostra a posição de todas as linguagens de programação existentes e conhecidas no mercado mundial e nela pode ser visto a posição da linguagem Lua em relação a outras linguagens dentro de um período de um ano, às posições dessa tabela são geradas a partir de uma medição feita com mais de 2000 linguagens de programação. Como parâmetro de entrada para a criação dessa tabela, o site utiliza as entradas de busca sobre as linguagens de programação nos mais variados sites de busca como o Google, o Bing, o Yahoo, etc. e monta a tabela mostrando a relação de posição de cada linguagem de programação, essa tabela é gerada mensalmente comparando e verificando a posição que cada linguagem de programação sofreu.

Tabela 2 – Ranking de linguagens de programação mais pesquisadas em Abril 2013

Position Apr 2013	Position Apr 2012	Delta in Position	Programming Language	Ratings Apr 2013	Delta Apr 2012	Status
1	1	=	C	17.862%	+0.31%	A
2	2	=	Java	17.681%	+0.65%	A
3	3	=	C++	9.714%	+0.82%	A
4	4	=	Objective-C	9.598%	+1.36%	A
5	5	=	C#	6.150%	-1.20%	A
6	6	=	PHP	5.428%	+0.14%	A
7	7	=	(Visual) Basic	4.699%	-0.26%	A
8	8	=	Python	4.442%	+0.78%	A
9	10	↑	Perl	2.335%	-0.05%	A
10	11	↑	Ruby	1.972%	+0.46%	A
11	9	↓↓	JavaScript	1.509%	-1.37%	A
12	14	↑↑	Visual Basic .NET	1.095%	+0.12%	A
13	15	↑↑	Lisp	0.905%	-0.05%	A
14	16	↑↑	Pascal	0.887%	+0.07%	A
15	13	↓↓	Delphi/Object Pascal	0.840%	-0.53%	A
16	32	↑↑↑↑↑↑↑↑↑↑	Bash	0.840%	+0.47%	A
17	18	↑	Transact-SQL	0.723%	-0.04%	A
18	12	↓↓↓↓↓↓	PL/SQL	0.715%	-0.66%	A
19	24	↑↑↑↑↑	Assembly	0.710%	+0.24%	A--
20	21	↑	Lua	0.650%	+0.08%	B

Tabela emitida pelo site TIOBE

Um marco para a linguagem é que o projeto e a evolução de Lua foram apresentados na terceira *History of Programming Language Conference* à HOPL III em 2007, essa conferência ocorre a cada 15 anos e somente algumas linguagens foram escolhidas até agora para serem apresentadas e discutidas nessa conferência. O fato de Lua ter sido escolhido e apresentado mostra a importância e o reconhecimento dessa linguagem para o desenvolvimento de aplicações ao redor do mundo e como uma linguagem que fora criada em um país que não seja de 1º mundo pode ter tanta influência.

4 – A Linguagem Lua

> "A idéia básica é ser uma linguagem que sacrifica um pouco eficiência de execução (não é tão eficiente como, por exemplo, C e C++, mais pesadas) em troca de flexibilidade. Lua é mais fácil de programar, você não tem que se preocupar com certas coisas. Mas seu maior diferencial é ter uma interface muito fácil com C e C++. Então, a idéia central é uma mistura: fazer partes do seu programa em C e partes em Lua, de modo que, onde você precisa de mais eficiência da máquina, usa o C; e onde precisa de mais flexibilidade para experimentação, usa Lua. Ela permite um equilíbrio melhor entre essas duas coisas, além de ser mais fácil de modificar. Pode-se, por exemplo, mexer na instalação sem ter que recompilá-lo... E também é uma linguagem com total portabilidade. Roda desde Windows a Unix e Linux, passando por sistemas como BeOS, OS/2, Amiga, e mesmo em PlayStation e supercomputadores Cray."
>
> - Roberto Ierusalimschy em entrevista para O Globo.

4.1. - Características

Como citado anteriormente Lua é uma linguagem de programação de extensão projetada para que possa ser usada em conjunto com outras linguagens e que consegue ser utilizada para vários ramos do mercado como robótica, biotecnologia, desenvolvimento web, atualmente tem uma forte atuação no desenvolvimento de jogos eletrônicos (*games*). Embora seja mais utilizado para viabilizar scripts que trabalhem como uma extensão do programa principal, Lua pode ser usada inteiramente para a construção de um sistema mais complexo com milhares de linhas de código e sendo este segundo mais difícil de ser visto. A linguagem Lua está na versão 5.2, possui a licença do MIT, ou seja, é uma linguagem de programação que é livre e de código aberta para que as pessoas utilizem como bem entenderem, sendo que estas podem utilizar para uso comercial sem ter a necessidade de arcar com custos ou burocracia sobre a utilização da Linguagem Lua em seu projeto.

> "Um conceito fundamental no projeto de Lua é fornecer *meta-mecanismos* para a implementação de construções, em vez de fornecer uma multidão de construções diretamente na linguagem. Por exemplo, embora Lua não seja uma linguagem puramente orientada a objetos, ela fornece meta-mecanismos para a implementação de classes e herança. Os meta-mecanismos de Lua trazem uma economia de conceitos e mantêm a linguagem pequena, ao mesmo tempo que permitem que a semântica seja estendida de maneiras não convencionais." - http://www.lua.org/portugues.html#sabermais

4.2 - MetaTabelas

Metatabelas são tabelas comuns do Lua que consegue manipular o valor recebido com certas operações especiais, sendo possível alterar vários aspectos comportamentais das operações sobre um valor especificando campos específicos na metatabela do valor, ou seja, a partir de um campo de uma metatabela pode-se alterar os valores armazenados nelas utilizando-se de operações especificas, configuradas ou não, das metatabelas.

Uma metatabela controla como um objeto se comporta em vários aspectos como operações aritméticas, comparações com relação à ordem, concatenação, operação de comprimento e indexação, assim uma metatabela também pode definir uma função a ser chamada quando um objeto userdata é coletado pelo coletor de lixo. Para cada uma destas operações Lua associa uma chave específica denominada de "evento". Quando é realizado uma destas operações sobre um valor, Lua verifica se este valor possui uma metatabela com o evento correspondente, caso seja encontrado o valor associado àquela chave (o metamétodo) controla como Lua irá se comportar referente a operação que a chamou.

4.3 Valores e Tipos

Lua é uma linguagem dinamicamente tipada, ou seja, não é necessário declarar o tipo da variável e suas variáveis são definidas em tempo de execução dependendo do que estiver sendo armazenado no momento. Lua possui oito tipos básicos de variáveis que são: *nil, boolean, number, string, function, userdata, thread* e *table*, cada qual com sua característica do tipo de variável, *nil* como sendo um valor único, geralmente é utilizado para indicar valor nulo.

Boolean indica valor booleano *true* ou *false,* com uma diferença que uma expressão será considerada falsa caso o valor seja igual à *nil* ou a *false* e verdadeira em todas as outras situações, mesmo quando o valor for zero. Diferente das demais linguagens, Lua possui somente um tipo numérico que é ponto flutuante por padrão, que é o *number,* o tipo

number pode representar um qualquer número inteiro de 32 bits.

O tipo *string* representa uma cadeia de caracteres, que pode ser delimitado por aspas simples ou duplas, nesse caso a regra é igual à de outras linguagens como o *Javascript*, se uma cadeia de caracteres começar com aspas simples deve terminar com aspas simples, assim como com aspas duplas. Cadeias de caracteres são únicas, então toda vez que uma cadeia de caracteres do tipo *string* é alterada, na verdade a pessoa que está programando está criando uma nova cadeia e cada cadeia de caracteres pode armazenar qualquer caractere de tamanho de 8 bits.

Function é o tipo que representa funções em Lua e também pode representar as funções escritas em C. Funções em Lua são consideradas valores de primeira classe o que significa que funções podem ser armazenadas em variáveis, passadas como parâmetros, ou retornadas como resultados. O tipo *userdata* permite que dados quaisquer em C possam ser armazenados em variáveis Lua, este tipo corresponde a um bloco de memória e não possui operações pré-definidas em Lua exceto atribuição e teste de identidade, além do que somente é possível alterar esses valores utilizando-se da API C, garantindo assim a integridade dos dados do programa principal.

O tipo thread representa fluxos de rotinas independentes usados para implementar co-rotinas, com isso Lua dá suporte a todas as co-rotinas dos sistemas.

O tipo *table* implementa arrays associativos, em outras palavras são *arrays* que podem ser indexados não somente por números, mas sim por qualquer valor diferente de *nill.* As tabelas dentro da linguagem Lua podem ser heterogêneas podendo conter todo e qualquer tipo de valor. As tabelas são os únicos mecanismos de estruturar os dados, podendo, dependendo dos valores passados representar arrays comuns, tabelas de símbolos, conjuntos, registros, grafos, árvores, etc. Para representar registros Lua utiliza o nome do campo como um índice.

Valores dos tipos *table, function*, *thread* e *userdata* são objetos, as variáveis não contêm realmente os valores, apenas referencia para eles. Atribuição, passagem de parâmetro, e retorno de funções sempre lidam com referências para tais valores; estas operações não efetuam nenhum tipo de cópia de dados do valor referenciado.

Abaixo temos um exemplo do tipo que a variável assume baseando- se no valor recebido:

a = “ola” – variável “a” recebe o valor “ola”, seu tipo será *string*;
a = 123 – ao receber o valor “123” em seguida seu tipo é reconhecido como *number*;
a = *true* – valor alterado novamente, sendo agora *boolean*;

4.4 – Variáveis

Variáveis em qualquer linguagem tem a função de armazenar um valor determinado pelo programa ou usuário para que este seja utilizado em um operação especificada pelo programa em tempo de execução. Em Lua existem três tipos de variáveis: variáveis globais, variáveis locais e campos de tabela. Um nome qualquer pode denotar uma variável global ou uma variável local (ou um parâmetro formal de uma função, que é um caso particular de variável local). Em Lua a menos que a variável seja declarada como variável local todas as variáveis são globais, variáveis locais possuem escopo léxico e podem ser acessadas por funções definidas dentro do seu escopo.

```
local x = 33 → declarando variável local e atribuindo 33 como valor a variável.
local y → declarando outra variável local.
print (x, y) → 33 nil
if x > 10 then
local x = 5 → alterando um "x" local
y = 9
print (x, y) → 5 9
else
x = 2 → alterando o "x" mais externo
print (x, y) → 2 nil
end
print (x, y) → 33 9
```

Qualquer variável antes de receber um valor tem como valor padrão o tipo *nil*; e acesso as variáveis globais e a campos de tabelas podem ser mudadas através do uso de metatabelas. Todas as variáveis globais são mantidas como campos em tabelas Lua comuns, chamadas de tabelas de ambiente ou simplesmente de ambientes e cada função tem faz sua própria referência a um ambiente, sendo que todas as variáveis globais dentro de uma função irão se referir para esta tabela de ambiente. Quando uma função é criada, ela herda o ambiente da função que a criou.

Em Lua é possível atribuir valores em diversas variáveis em um mesmo comando, seguindo uma ordem de atribuição o valor vai corresponder a variável que se encontra na mesma posição, como se fosse um "*array"*, sendo que quando o número de variáveis declaradas é diferente dos valores atribuídos a elas a linguagem ajusta automaticamente as listas, preenchendo com valores do tipo *nil* ou desprezando os valores a mais.

Exemplo:

```
a, b = 2
c, d = 2, 4, 6
```

No exemplo acima "a" recebe o valor 2, enquanto "b" recebe o valor *nil* por não ter sido atribuído nenhum valor a ele. Enquanto na segunda linha as variáveis "c" recebe 2, "d" recebe 4 e o valor "6" é desprezado por não haver nenhuma variável que possa receber esse valor.

Uma das vantagens sobre a atribuição múltipla de valores para as variáveis é que pode-se trocar os valores entre as variáveis sem a necessidade de uma variável auxiliar.

```
A= 3;
B = 7;
A, B = B, A;
```

Nesse exemplo as saídas das variáveis serão A = 7 e B = 3. Isso ocorre porque ao executar o a atribuição de valores múltipla o valor ao ser passado para variável ainda está sendo refenciado, então a variável só irá assumir o novo valor ao término da linha de comando que está executando a ação de troca de valores. Outra característica das variáveis em Lua é que ao ela podem receber resultados de formulas com operadores relacionais. Segue o exemplo abaixo

```
a = 4 < 3
b = 4 > 3
```

Nesse caso os operadores relacionais ao serem atribuídos a uma variável têm dois tipos de retorno nil quando o resultado é false e 1 quando for verdadeiro. No caso do exemplo passado a variável "a" recebe o valor nil, enquanto b recebe o valor 1.

4.5 – Bibliotecas

Neste item será explicado sobre as bibliotecas que a versão 5.2 do Lua possui, assim como alguns comandos dessas bibliotecas e exemplos mostrando a utilização de alguns dos comandos citados.

4.5.i – Biblioteca Padrão

As bibliotecas padrão de Lua oferecem várias funções úteis que são implementadas diretamente através da API C. Algumas dessas funções oferecem serviços essenciais para a linguagem, enquanto outras oferecem acesso a serviços "externos" e outras poderiam ser implementadas em Lua mesmo, mas são bastante úteis ou possuem requisitos de desempenho críticos que merecem uma implementação em C. Todas as bibliotecas são implementadas através da API C oficial e são fornecidas como módulos C separados. Lua possui as seguintes bibliotecas padrão:

- biblioteca básica;
- biblioteca de co-rotinas;
- biblioteca de pacotes;
- manipulação de cadeias de caracteres;
- manipulação de tabelas;
- funções matemáticas (sen, log, etc.);
- operações bit a bit;
- entrada e saída;
- facilidades do sistema operacional;
- facilidades de depuração.

Excetuando-se a biblioteca básica e a biblioteca de pacotes, cada biblioteca provê todas as suas funções como campos de uma tabela global ou como métodos de seus objetos.

Para ter acesso a essas bibliotecas, o programa principal C deve chamar a função *luaL_openlibs*, esta função abre todas as bibliotecas padrão. É possível chamar cada uma das bibliotecas individualmente usando os seguintes comandos para cada biblioteca. O *luaopen_base* (para a biblioteca básica), *luaopen_coroutine* (para a

biblioteca de co-rotinas), *luaopen_package* (para a biblioteca de pacotes), luaopen_string (para a biblioteca de cadeias de caracteres),*luaopen_table* (para a biblioteca de tabelas), *luaopen_math* (para a biblioteca matemática), *luaopen_bit32* (para a biblioteca de bit), *luaopen_io* (para a biblioteca de E/S), *luaopen_os* (para a biblioteca do Sistema Operacional), e *luaopen_debug* (para a biblioteca de depuração). Essas funções estão declaradas em *lualib.h* e não devem ser chamadas diretamente: você deve chamá-las como qualquer outra função C de Lua, usando *lua_call*. A seguir será passado as bibliotecas informando características e alguns comandos pertinentes a cada biblioteca.

Biblioteca Básica – oferece algumas funções essenciais a Lua.

collectgarbage (opt [, arg]) – esta função é uma interface genérica para o coletor de lixo e realiza diferentes funções de acordo com o argumento passado.

"stop": para o coletor de lixo.
"restart": reinicia o coletor de lixo.
"collect": realiza um ciclo de coleta de lixo completo.
"count": retorna a memória total que está sendo usada por Lua (em Kbytes).
"step": realiza um passo de coleta de lixo. O "tamanho" do passo é controlado por arg (valores maiores significam mais passos). Retorna true se o passo terminou um ciclo de coleta de lixo.
"setpause": estabelece arg como o novo valor para a *pausa* do coletor e retorna o valor anterior para a *pausa*.
"setstepmul": estabelece arg como o novo valor para o *multiplicador de passo* do coletor. Retorna o valor anterior para a *pausa*.
"isrunning": retorna um valor booleano que diz se o coletor está correndo.
"generational": muda o coletor para o modo geracional, mas essa opção ainda está em teste.
"incremental": muda o coletor para o modo incremental o modo padrão dele.

dofile (filename) – abre o arquivo indicado e executa o conteúdo como um trecho de código Lua. Em caso de erros no código chamado o dofile irá retornar o erro para todo o código principal. Isto é o comando dofile não executa em modo protegido.

getmetatable(objeto) – se o objeto passado como parâmetro não possuir uma metatabela associada e ele o valor que será retornado será nil, caso contrário se o objeto possui um campo "_metatable" o retorno será o valor associado a esse campo, senão o valor será a metatabela do objeto em questão.

Pcall (função, arg1,.., argn) – chama a função com os argumentos fornecidos em modo protegido, com isso a função será executada e caso ocorra algum erro durante o processo

da função esse erro não será passado ao programa principal, ao invés disso o erro será capturado e o que irá retornar ao programa principal será um valor booleano false e uma mensagem indicando qual o erro. Caso não ocorra nenhum erro o valor booleano retornado será true junto com os valores resultantes da função chamada.

Biblioteca de co-rotinas – fornece operações relacionadas a co-rotinas
coroutine.create (f) – cria uma co-rotina com uma funçãoLua

coroutine.running () - retorna a co-rotina sendo executada ou nil quando chamada pelo fluxo de execução.

coroutine.status (co-rotina) - retorna o estado da co-rotina passada, esse estado pode variar sendo os estados: "running", se a co-rotina está executando no momento da chamada; "suspended", se a co-rotina está suspensa em uma chamada ou se ela não foi iniciada; "normal" se a co-rotina está ativa mas não está executando (isto é, ela continuou outra co-rotina); e "dead" se a co-rotina se a co-rotina foi encerrada de algum modo.

Módulos – a biblioteca provê funcionalidades que permite ao programador carregar e construir módulos
require(modulo) – carrega o módulo fornecido. O comando require verifica em package.loaded se o módulo em questão já foi carregado, se o módulo já foi carregado require irá retornar o valor armazenado em package.loaded[modulo], senão a função require passa a buscar um carregador para o módulo.
package.preload – uma tabela para armazenar carregadores para módulos específicos.

Manipulação de Cadeias de Caracteres – essa biblioteca possui funções que permite a manipulação de cadeias de caracteres, como por exemplo encontrar e extrair subcadeias. A biblioteca de cadeia de caracteres também estabelece uma metatabela para cadeias onde o campo _index irá apontar para a tabela string, assim tem a possibilidade de utilizar as funções da biblioteca para a programação orientada a objetos. A biblioteca de manipulação de cadeia assume que cada caracter é codificado utilizando um byte por caracter.

string.dump(funcao) – retorna uma cadeia contendo a representação binária de uma função fornecida no “funcao”

string.reverse (s) – retorna a cadeia de caracteres invertida

string.len(s) – retorna uma cadeia de caracteres e retorna o comprimento dela, sendo que se a cadeia recebida for vazia o retorno será 0.

Manipulação de Tabelas – esta biblioteca ira disponibilizar funções genéricas para a manipulação de tabelas, sendo que boa parte das funções dessa biblioteca assumem que a tabela representa um array ou uma lista.

table.insert(tabela, pos, valor) – ira inserir um novo valor na tabela especifica após o índice determinado, caso não seja declarado nenhum índice o comando assume que o valor inserido deve ser adicionado no final da tabela.

table.remove(tabela, pos) – remove da tabela mencionada o valor especificado pelo índice que foi passado ao declarar o comando

table.maxn(tabela) – retorna o maior índice numérico positivo que a tabela possui, se a tabela não possuir índices positivos então o valor de retorno será 0.

table.sort(tabela,comprimento) – irá arrumar os valores da tabela em uma determinada ordem. Se o comprimento é fornecido então será uma função que recebe dois elementos e retorna true quando o primeiro valor é menor que o segundo.

```
local s={}

table.insert (s,9)
table.insert (s,3)
table.insert (s,6)
table.insert (s,1)
table.insert (s,5)
table.insert (s,4)

table.sort(s)
for i=0, table.maxn(s) do
	print(s[i])
end
```

No exemplo acima é declarado uma variável local “s” que é uma tabela, em seguida é incluído valores quaisquer utilizando o comando table.insert, ao término da adição dos valores na tabela é feito uma ordenação nos valores com o table.sort para que assim seja impresso os valores da tabela. O resultado é mostrado abaixo.

```
s = nil, 1, 3, 4, 5, 6, 9
```

Nota-se que o resultado impresso desse código começa com nil, isso porque o comando table.insert inclui os valores após o último valor da tabela e como não houve alteração no primeiro valor da tabela então ao ordenar a tabela o primeiro valor passa a ser nil.

Funções Matemáticas - Esta biblioteca é uma interface que compõe a biblioteca matemática padrão de C. Algumas de suas funções são:

Math.abs(x) – ira retornar o valor absoluto da variável x;

Math.min/max(r,..,z) – retornará o menor ou maior valor dentro do intervalo passado

```
a = 3      b = 8
c = 2      d = 7
e = 6      f = 4
g = 5      h = 1
```

io.write(math.min(a,b,c,d,e,f,g)) – o valor impresso aqui será 1

io.write(math.max(a,b,c,d,e,f,g)) – o valor impresso aqui será 8

Nesse exemplo os números que serão impressos do intervalo passado foram min = 1 e max = 8. Em um exemplo real essa função poderia retornar o maior valor de vendas dentro de um intervalo de tempo de um dado funcionário em uma empresa, para que assim ele pudesse receber talvez alguma bonificação pelo resultado obtido.

Math.exp(x) – retorna o valor de e^x.

Math.random (m , n) – esta função é uma interface para a função em rand da linguagem C. Quando passado sem parâmetros a função irá retornar um valor inteiro no intervalo de [0,1], se for passado com um número inteiro ou dois a função irá pegar o intervalo dos valores passados e irá retornar um número inteiro a partir desse intervalo, levando em conta que o primeiro valor passado deve ser menor que o segundo valor.

io.write(math.random(1,45));

No exemplo acima utilizamos os comandos para impressão de um valor junto com o comando math.random, no caso o intervalo dado é de 1 até 45. Então a impressão do valor será um número inteiro qualquer dentro desse intervalo

Math.pi – retorna o valor de pi

Math.log(x) – retorna o valor do logaritmo natural de x;

Operações Bit a Bit – essa biblioteca disponibiliza operações utilizando bits

bit32.bor(...) – retorna o bit provido ou um de seus operandos

bit32.bxor() – retorna um bit exclusivo ou um de seus operandos

bit32.band() – retorna um bit e seus operandos

Facilidades de Entrada e Saida – a biblioteca de entrada e saída permite duas maneiras diferentes para a manipulação de arquivos. O primeiro utiliza descritores de arquivos implícitos, ou seja, estabelece um padrão para os arquivos de entrada e saída e todas as operações são realizadas em cima desses arquivos, já o segundo utiliza descritores de arquivos explícitos que no caso será retornado um descritor de arquivos a partir da operação io.open e todas as operações são determinadas como métodos a partir do descritor de arquivos carregado. Em caso de falha na execução de alguma função dessa biblioteca será retornado nil juntamente com uma mensagem de erro.

io.open(arquivo, modo) – essa função abre um arquivo, no modo especificado pela cadeia "modo", retornando um novo manipulador ou arquivo e em caso de erro retorna um nil junto com uma mensagem de erro.

A cadeia de caracteres mode pode ser qualquer uma das seguintes:

"r": modo de leitura (o padrão);

"w": modo de escrita;

"a": modo de adição;

"r+": modo de atualização, todos os dados anteriores são preservados;

"w+": modo de atualização, todos os dados anteriores são apagados;

"a+": modo de atualização de adição, dados anteriores são preservados, a escrita somente é permitida no fim do arquivo.

A cadeia mode também pode ter um 'b' no final, que é necessário em alguns sistemas para abrir o arquivo em modo binário.

io.close(arquivo) – quando não recebe nenhum arquivo a função fecha o arquivo de saída

padrão.

io.write(arquivo) – imprime o parâmetro passado na tela.

Facilidades do Sistema operacional – esta biblioteca prove funções que tratam retornos de valores vindo do sistema operacional (S.O)

os.clock() – retorna o valor aproximado da quantidade de tempo em segundos que é usada pelo programa.

os.date(formato, tempo) – retorna uma cadeia ou uma tabela com valores de data e hora, formatada a partir do padrão definido pelo parâmetro passado em formato.

os.execute(comando) – passa um comando que sera executado por um interpretador de comandos do sistema operacional. Esta função terá como retorno uma resposta que é provida pelo sistema operacional.

os.remove(arquivo) – remove um arquivo ou diretório com o nome fornecido. Para que diretórios sejam removidos eles necessariamente precisam estar vazios, caso a função os.remove apresente algum erro, o retorno será dado por nil mais uma mensagem descrevendo o erro ocorrido

Biblioteca de Depuração – a biblioteca de depuração fornece ao programador funcionalidades da interface de depuração para programas Lua. Sendo que é necessário tomar cuidado ao usar essa biblioteca

debug.debug() - Entra em um modo interativo com o usuário, executando cada cadeia de caracteres que o usuário acessa. Usando comandos simples e outros mecanismos de depuração, o usuário pode inspecionar variáveis globais e locais, mudar o valor delas, avaliar expressões, funções, dados de tabela, etc. Uma linha contendo somente a palavra cont termina esta função, de modo que a função chamadora continua sua execução.

Note que os comandos para debug.debug não são aninhados de modo léxico dentro de nenhuma função e portanto não possuem acesso direto a variáveis locais.

debug.getregistery() - retorna a tabela de registro

debug.setmetatable (objeto, tabela) - Estabelece a variável tabela como uma metatabela do objeto fornecido (tabela pode ser nil).

debug.setfenv (objeto, tabela) - Estabelece a tabela “tabela” como o ambiente do objeto fornecido. Retorna objeto.

4.5.ii - Biblioteca Auxiliar

A biblioteca auxiliar oferece várias funções para a interface de C com Lua, enquanto a API básica fornece as funções primitivas para todas as interações entre C e Lua, a biblioteca auxiliar fornece funções de mais alto nível para algumas tarefas comuns. Todas as funções da biblioteca auxiliar são definidas no arquivo de cabeçalho lauxlib.h e possuem um prefixo luaL_.

Todas as funções na biblioteca auxiliar são construídas sobre a API básica e portanto elas não oferecem nada que não possa ser feito com a API básica. Várias funções na biblioteca auxiliar são usadas para verificar argumentos de funções C. O nome delas sempre é luaL_check* ou luaL_opt*. Todas essas funções disparam um erro se a verificação não é satisfeita. Segue uma lista de alguns comandos da biblioteca auxiliar de Lua

luaL_argerror

int luaL_argerror (lua_State *L, int narg, const char *extramsg);

Dispara um erro com a seguinte mensagem, onde func é recuperada a partir da pilha de chamada:

bad argument #<narg> to <func> (<extramsg>)

luaL_Buffer

typedef struct luaL_Buffer luaL_Buffer;
O tipo para um *buffer de cadeia de caracteres*.

Um buffer de cadeia possibilita ao código C construir cadeias Lua pouco a pouco. O seu padrão de uso é explicado logo abaixo:
Primeiro é necessário declarar uma variável do tipo luaL_Buffer.

A seguir deve inicializar com uma chamada usando o comando luaL_buffinit(L, &b). Depois o programador irá adicionar pedaços da cadeia ao buffer chamando qualquer uma das funções com o código luaL_add*.
Para obter o resultado é necessário utilizar fazer uma chamada utilizando o código

luaL_pushresult(&b). Esta chamada deixa a cadeia final no topo da pilha.

Durante essa operação qualquer, um buffer de cadeia utiliza um número variável de posições da pilha, então ao utilizar um buffer é necessário saber que não se deve assumir onde o topo da pilha está.

luaL_callmeta

int luaL_callmeta (lua_State *L, int obj, const char *e);

Chama um metamétodo. Se o objeto no índice obj possuir uma metatabela e esta metatabela por sua vez possuir um campo e, a função chama esse campo passando o objeto como seu único argumento, a função terá como retorno 1 e ira colocar na pilha o valor retornado pela chamada. Se não há metatabela ou metamétodo, esta função retorna 0 (sem empilhar qualquer valor na pilha).

luaL_checkinteger

lua_Integer luaL_checkinteger (lua_State *L, int narg);

Verifica se o argumento narg da função é um número e retorna este número convertido para um valor do tipo lua_Integer.

luaL_checknumber

lua_Number luaL_checknumber (lua_State *L, int narg);

Verifica se o argumento narg da função é um número e retorna este número.

luaL_dofile

int luaL_dofile (lua_State *L, const char *filename);

Carrega e executa o arquivo fornecido. É definida como a seguinte macro:

(luaL_loadfile(L, filename) || lua_pcall(L, 0, LUA_MULTRET, 0))

A função retorna 0 se não há erros ou 1 em caso de erros.

4.6 – Portabilidade e Tamanho

Lua é uma linguagem de programação que por ter sido escrita em ansi C e ansi C ++ consegue ser compilada por várias plataformas, dentre elas temos Unix, Windows, em dispositivos móveis que usem Android, iOS, Symbian; em microprocessadores embutidos e até em Mainframes IBM; sendo essas algumas das várias plataformas que podem compilar Lua. Além de poder ser compilada nessas plataformas, Lua dispõe de kits de desenvolvimento para boa parte das aplicações como o IOS Software Development Kit, o Lua4Delphi, NetBSD, entre outros.

Desde o início ao ser construída Lua foi uma linguagem que teve um tamanho consideravelmente pequeno, assim ao ser incluída em algum projeto ela não gera nenhum impacto ao tamanho do projeto sendo que o interpretador Lua contendo todas as bibliotecas padrões de Lua ocupa 182K e a biblioteca Lua ocupa 243K. Para isso ser possível o código Lua é dividido em três partes o núcleo, uma biblioteca auxiliar e as bibliotecas padrão. O núcleo contém toda a parte básica de Lua, como o pré-compilador, o interpretador, os algoritmos de manipulação de tabela e de coleta de lixo; entretanto o núcleo não assume nada sobre o sistema operacional. Por exemplo o núcleo faz toda a alocação de memória invocando uma função externa que fora atribuída a ele antes de sua inicialização. Do mesmo jeito que Lua não lê arquivos, mas carrega trechos de código utilizando uma função externa apropriada. A *API* do núcleo é toda definida no arquivo lua.h e todos os nomes definidos por esse arquivo tem o prefixo de lua_.

Essa estrutura é recomendada para aplicações embarcadas; para plataformas mais convencionais existe a biblioteca auxiliar que utiliza a API do núcleo e as funções normais do sistema operacional para fornecer uma *interface* de mais alto nível para o programador. Essa estrutura é definida no arquivo lauxlib.h e todos os nomes dessa API possuem o prefixo luaL_.

4.7 – Programação Orientada a Objetos

Lua não é uma linguagem de programação orientada a objetos e nem nasceu com esse objetivo, mas com o tempo foi aparecendo à necessidade de se ter uma linguagem que fosse orientada a objetos então com isso foi criado uma série de mecanismos que possibilitam que o programador consiga desenvolver aplicações utilizando-se de programação orientada a objetos.

A ideia principal da programação orientada a objetos dentro de Lua é a criação e uso de protótipos. Podemos entender uma meta-tabela como um protótipo de uma classe

```
Rectangle = {
                width = 0,
                height = 0
                }

function Rectangle.new (self, o)
        = o or {}
                setmetatable (o, self)
                self.__index = self
                return o
end
```

4.8 – Incluindo Lua

Lua sendo uma linguagem de extensão precisa ser chamada ou atribuída ao código de alguma maneira para que assim o script ou comando possa ser chamado passando as variáveis de entrada e processando o resultado desejado a partir dessas variáveis. Lua pode ser chamada para varias linguagens como C, Java, Delphi, entre outras.

Exemplo de Lua sendo chamada dentro de um programa C

```
#include <stdio.h>
#include <string.h>
#include "lua.h"
#include "lauxlib.h"
#include "lualib.h"

int main () {
char buff [256];
int error;
lua_State *L = luaL_newstate (); → cria um novo estado Lua
luaL_openlibs (L); → dá acesso a todas as bibliotecas padrões

while (fgets (buff, sizeof (buff), stdin) != NULL) {
            error = luaL_loadbuffer (L, buff, strlen (buff), "line") ||
            lua_pcall (L, 0, 0, 0);
            if (error) {
                fprintf (stderr, "%s", lua_tostring (L, -1));
                lua_pop (L, 1); → retira a mensagem de erro da pilha
            }
}

lua_close (L);
return 0;
}
```

No exemplo acima foi declarado às bibliotecas padrão e auxiliar junto com as outras bibliotecas padrão da linguagem C e no código foi chamado comandos Lua. No caso da linguagem C ao utilizar Lua é necessário declarar o início e o fim do bloco em que será usado a linguagem Lua, como visto os comandos lua lua_pcall e lua_close servem para abrir o bloco de declaração para o uso dos comandos em Lua e o lua_close obviamente para fechar o bloco de comandos. Outro ponto que pode-se notar nesse exemplo e a diferenciação dos comandos em Lua dos comandos em C, que como explicado antes todos os comandos em Lua terão a declaração de “lua_comando“.

5 – Versão 5.2

A versão mais recente do Lua trás consigo algumas alterações em comandos e códigos e reforça ideias que foram apresentadas na versão 5 de Lua.

5.1 Módulos

Módulos é outro recurso que foi criado a partir da versão 5.1, é uma biblioteca que dispõe facilidades básicas ao programador para carregar e construir módulos em Lua.

Quando foi criado ele utilizava duas funções para o ambiente global: a *require* e o *module*. Sendo que *module* serve para criar um módulo especifico e a *require* serve para carregar esse módulo, já na versão 5.2 não há mais o carregamento da função *module* para o ambiente global, somente o *require* que é carregado.

Partindo do princípio que um módulo já foi criado, a função *require* trabalha da seguinte maneira, verificará na tabela *package.loaded* se o nome do módulo já foi carregado. Caso ele encontre o módulo *require* irá retornar o valor armazenado em `package.loaded[modname]`. Caso contrário, ele tenta encontrar um carregador para o módulo, sendo que para localizar *require é* guiada pelo array package.loaders. Modificando esse este array nós podemos mudar o meio que require usa para localizar um módulo. No caso a função require irá buscar uma função na posição determinada e se não localizar nada buscará na ordem uma função em Lua, se não localizar buscará uma função em C e se não achar a função em C buscará um pacote que se encaixe na chamada. Caso ocorra algum erro durante a execução ou chamada será emitido um alerta indicando o erro.

5.2 Ambientes

Além das metatabelas, objetos do tipo thread, function e userdata possuem outra tabela associada com eles, chamada de ambiente. Assim como metatabelas, ambientes são tabelas normais e vários objetos podem compartilhar o mesmo ambiente.

A partir da versão 5.2 de Lua o conceito de ambiente some para as funções em C e passa a ser usado somente para funções em Lua. Objetos criados em Lua são criados compartilhando o ambiente que os criou. Ambientes associados com fluxos de execução (threads) são chamados de ambientes globais. Eles são usados como o ambiente padrão pelos fluxos de execução e funções não aninhadas criadas pelo fluxo de execução e podem ser diretamente acessados pelo código C.

Ambientes associados com as funções Lua são usados para resolver todos os acessos a variáveis globais dentro da função, esses ambientes são utilizados como o ambiente

padrão para outras funções em Lua criadas pela função que as chama. A partir de comandos em Lua é possível mudar o ambiente de uma função ou um fluxo de execução que está sendo executado atualmente

5.3 Co-Rotinas

Lua oferece suporte a co-rotinas, também conhecidas por fluxos de execução (threads) colaborativos, sendo que uma co-rotina em Lua representa um fluxo independente que ao contrário de processos leves em sistemas que dão suporte a múltiplos fluxos de execução, uma co-rotina só pode ter sua execução interrompida através de uma chamada explícita de uma função para encerrar a rotina ou se o processo for finalizado por ter chegado ao fim do seu ciclo de execução.

Como um exemplo, considere o seguinde código:

```
function foo (a)
print("foo", a)
return coroutine.yield(2*a)
end

co = coroutine.create(function (a,b)
        print("co-body", a, b)
        local r = foo(a+1)
        print("co-body", r)
        local r, s = coroutine.yield(a+b, a-b)
        print("co-body", r, s)
        return b, "end"
end)

print("main", coroutine.resume(co, 1, 10))
print("main", coroutine.resume(co, "r"))
print("main", coroutine.resume(co, "x", "y"))
print("main", coroutine.resume(co, "x", "y"))
```

Quando você executá-lo, ele produzirá a seguinte saída:

```
co-body 1       10
foo     2

main    true    4
co-body r
main    true    11      -9
co-body x       y
```

```
main     true     10        end
main     false    cannot resume dead coroutine
```

Uma co-rotina é parecido com um thread, mas cada um tem características próprias que permite diferenciar e assim utilizar o que melh or convém para o momento. As diferenças mais visíveis que pode ser notadas entre co-rotina e threads ambos tem linhas de execução que executam em seus próprios ambientes locais. Conceitualmente threads podem ser executados simultaneamente enquanto co-rotinas são executadas uma de cada vez e co-rotinas podem ser paradas e posteriormente retomadas a partir do ponto em que foi suspensa.

5.4 Coletor de Lixo (*Garbage Colector*)

O coletor de lixo é uma ferramenta do Lua que faz gerenciamento automático de memória, verificando e eliminando de tempos em tempos todos os objetos que não são mais usados pela aplicação, com isso a pessoa não precisa se preocupar com a alocação de memória para objetos novos e nem com a liberação de memória para os objetos que não são mais necessários para a aplicação corrente.

A partir da versão 5.1 foi implementado a função “marca-e-limpa” (*mark-and-sweep*) incremental que passa a controlar o tempo em que o coletor de lixo executa. O coletor utiliza dois números para controlar seu ciclo de execução para a coleta de lixo, a pausa do coletor de lixo e o multiplicador de passo do coletor de lixo, ambos os valores são expressos de maneira percentual.

A pausa do coletor de lixo controla quanto tempo será necessário esperar para que possa ser iniciado um novo ciclo de varredura, quanto mais alto o valor, mais o coletor irá demorar a começar, pelo passo que quanto mais baixo mais rápido ele irá executar. Já o multiplicador de passo controla a velocidade de execução em relação à alocação de memória, funcionando da mesma maneira que a pausa do coletor de lixo, valores maiores deixam ele mais rápido e valores menores mais devagar. Para poder alterar esses valores utilizam-se as funções *lua_cg* em C e *collectgarbage* em Lua.

Na versão 5.2 houve uma implementação, que ainda está em fase experimental, de um recurso para o coletor de lixo que permite que a pessoa altere o modo em que o coletor trabalha de incremental para geracional que assim o coletor passará a assumir que muitos

objetos morrem cedo, ou seja, objetos recentemente criados não terão um tempo de vida muito grande. A vantagem é que isso poderia reduzir o número de vezes que o coletor de lixo é executado, mas aumentaria o uso da memória devido ao fato que objetos mais antigos continuariam lá até o coletor ser executado.

5.5 Tratamento de Erros

Sabendo que Lua é uma linguagem de extensão, todas as ações realizadas começam a partir de um código C no programa principal que chama a função de uma biblioteca Lua. Sempre que ocorrem erros durante a execução ou compilação o controle retorna para C, para que assim sejam tomadas as medidas necessárias. E para verificar e tratar os erros que possam aparecer no programa há a biblioteca de depuração (*debug*) para verificar medição, fluxo, atribuição entre outras coisas; Mas não é recomendada sua utilização dado que elas podem ser muito lentas e várias funções do debug podem comprometer a segurança e a integridade do código.

5.6 – Inconsistências apresentadas

Da versão 5.1 e a 5.2, que é a mais atual, dado a sua evolução alguns comandos são removidos dado a sua utilização desnecessária enquanto outros somente sofrem mudanças ponderadas em sua chamada partindo do princípio que Lua é uma linguagem de script e que deve em seu desenvolvimento foi priorizado o tamanho, velocidade de execução, portabilidade e aplicabilidade houve a retirada de alguns comando e funções para que a chamada seja simplificada e assim tenha uma diminuição em partes do código para que haja a implementação de novas funcionalidades permitindo que a linguagem melhore em desempenho e funcionalidade. Então não há muita gravidade nas inconsistências encontradas visto que normalmente são mencionadas na documentação oficial da linguagem Lua.

6 – Aplicações

Neste capítulo será abordado como a linguagem Lua pode ser utilizada em algumas áreas e suas demais aplicações, será mostrado alguns exemplos em determinadas áreas

para efeito de explanação, não entrando a fundo em cada item para não fugir do objetivo deste trabalho:

6.1 Ginga – Televisão interativa brasileira

Ginga é uma camada de software que dá suporte à execução de aplicações interativas nos conversores digitais para as tvs digitais, sendo este o padrão para o Sistema Brasileiro de TV Digital [ABNT 2007], Lua trabalha com esse padrão. A arquitetura do Ginga é composta por dois ambientes, um imperativo, também conhecido por máquina de execução, que é responsável pelo suporte às aplicações desenvolvidas na linguagem Java e um ambiente declarativo, conhecido como máquina de apresentação, que interpreta aplicações desenvolvidas em *Nested Context Language* (NCL).

NCL é uma linguagem declarativa que possibilita o desenvolvimento de aplicações multimídia com sincronismo espaço-temporal, entre objetos de mídia, como áudio, vídeo, imagem e texto. Com o intuito de ampliar as aplicações que podem ser desenvolvidas o NCL suporta objetos escritos em linguagem Lua, denominados de objetos imperativos NCLua, o NCLua é uma extensão da linguagem Lua criada para o desenvolvimento de objetos, programas entre outras coisas para o Ginga. A extensão NCLua é específica para o Ginga e por isso não faz pertence a biblioteca padrão da linguagem Lua, sendo este um dos diferenciais do NCLua para a linguagem pura de Lua.

Figura 5 - Exemplo de uma implementação NCLua

```
function update ()
   logoLua.x = logoLua.x + 5
   redraw()
   if logoLua.x < 100 then
      event.timer(update, 30)
   end
end
function tratador (evt)
   if evt.action == 'start' then
      update()
   end
end
event.register(tratador, 'ncl', 'presentation')
```

Exemplo de animação em NCLua que utiliza um temporizador.

No código acima a função tratador ativa a função update, que atualiza a posição do logotipo de Lua, chama a função de redesenho e, caso o logotipo ainda não tenha alcançado a posição 100, se executa a função novamente após 30 milissegundos de espera.

6.2 Softwares

Embora não muito aparente Lua tem sido usada por muitas empresas para o desenvolvimento de softwares, como exemplo o Instituto do Coração (INCOR) que utilizou Lua na criação de um sistema que monitora as Unidade de Tratamento Intensivo (UTI's) pela internet e internet e a Nasa que desenvolveu um sistema que controla os níveis de gases perigosos na preparação para o lançamento dos ônibus espaciais. Lua também fora usado no Adobe Lightroom sendo que seu uso foi dado para desenvolver o ambiente dele.

6.3 – Games

> "A linguagem C faz toda a parte de renderização (a parte gráfica, pesada, que inclui criar todos os detalhes da animação, sombras, e assim por diante). Já o script do jogo — o que o personagem faz, como reage a determinada conjuntura — é todo comandado por Lua. O designer do roteiro de jogo não trabalha em C, e sim programa em Lua. Como o geólogo da Petrobras, ele não é um programador profissional, mas um especialista em criar games que usa Lua como ferramenta. No caso, o Bret é o programador oficial do jogo. O que ele fez? Botou Lua dentro do sistema, acoplando-a às partes mais pesadas, em C, e a ofereceu aos designers."
>
> - Roberto Ierusalimschy em entrevista para O Globo
>
> .

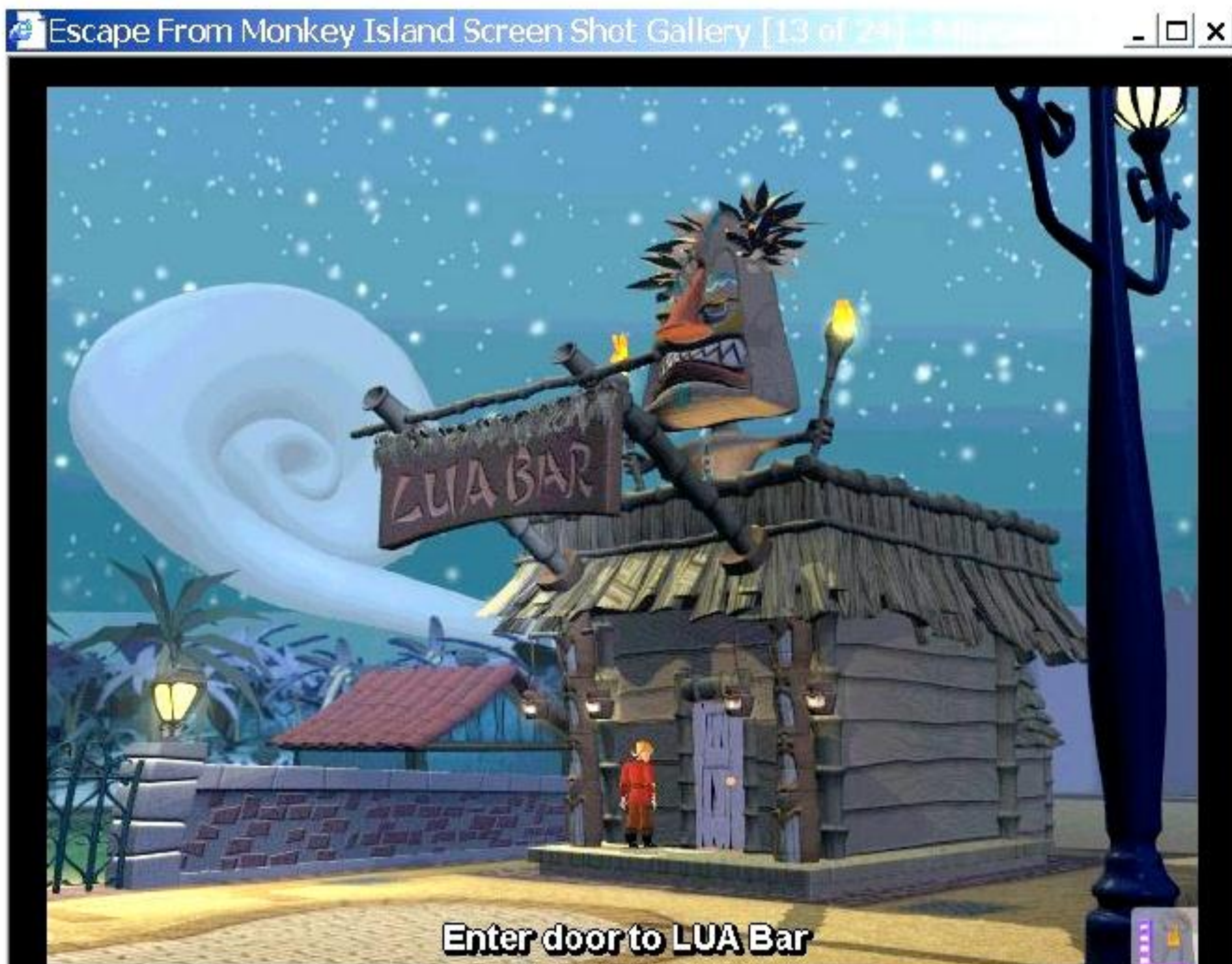
Lua é uma linguagem muito usada nos jogos de vídeo-games e muitos jogos atuais rodam Lua em seus códigos. Esses códigos podem variar suas funções que vão desde o controle da inteligência artificial dos inimigos ou personagens do jogo até o controle da física e de ações que irão ocorrer durante o jogo. Um dos jogos em que Lua foi utilizada no seu desenvolvimento é o famoso jogo Angry Birds. A primeira empresa a usar Lua em seus jogos foi a então falecida Lucas Arts com o jogo Escape From Monkey Island.

Lua já foi utilizado em vários jogos e sua utilização foi bem diversificada dentro desses jogos, em uma apresentação feita por

- Controle de IA
- Aparência de efeitos e de outros elementos gráficos

- Determinação das regras do jogo
- Edição dos atributos dos personagens
- Debug em tempo real
- Interfaces
- Edição de cenas e atributos em tempo real
- Criação de “Mod’s”
    - Criando e modificando arquivos Lua

Figura 6 - Imagem do jogo Monkey Island



http://pt.scribd.com/doc/91344/A-Linguagem-Lua-e-suas-aplicacoes-em-jogos

Lua pode ser utilizada como linguagem de configuração para viabilizar comportamentos ou ações que serão realizadas ao longo do jogo, fazendo associação de valores as variáveis abaixo há um exemplo de como isso pode ser feito utilizando-se de tabelas para configurar características de armas e possíveis diálogos que possam aparecer

para a determinada arma.

Exemplo:

```
weapons = {
faca = Weapon{
                        forca = 0.3,
                        alcance = 0.5,
                        precisao = 1.0,
                        getit = function (personagem)
                                if personagem:PossuiArma() then
                                        personagem:Speak("Não preciso dessa faca")
                                        return false
                                else
                                        personagem:Speak("Essa faca será realmente
                                        muito útil!")
                                        return true
                                end
                        end,
},
...
}
```

Neste exemplo se no jogo o personagem encontrar a faca, a arma face virá com esses atributos e irá disparar também um desses dois eventos de fala dependendo do resultado da condição.

Em um estágio mais avançado Lua pode ser utilizada como linguagem de controle, com isso a um imenso ganho de flexibilidade no desenvolvimento e como vantagem temos que os profissionais envolvidos com a programação do jogo podem programar diretamente, agilizando assim o desenvolvimento e dando mais liberdade a esses profissionais na hora do desenvolvimento. Outra vantagem em utilizar Lua como uma linguagem de controle é que não há a necessidade da compilação da aplicação com isso a velocidade de desenvolvimento do jogo aumenta consideravelmente, já que dependendo da aplicação compilar ela pode demorar muito tempo.

Um outro exemplo simples de visualizar a utilização de Lua em jogos é o uso de co-rotinas pelo fato dos jogos terem várias coisas acontecendo ao mesmo tempo em um mesmo espaço ou tela, uma co-rotina é perfeita para esse funcionamento já que permite a execução de códigos de maneira independente ao programa principal, podendo assim controlar a movimentação de Non Player Characters (NPC’s), ou personagens não jogáveis, em uma cidade. Outro exemplo do uso de Lua em games é na construção e

renderização dos cenários em tempo real como ocorre em Far Cry.

6.4 - Internet

Para a internet foram criados programas como o Wiresharck, um analisador de protocolos, o nmap que é um rastreador de redes para segurança o Publique!, um gerenciador de conteúdo para Web que foi inteiramente feito em Lua.

Página Inicial do Publique!

Fonte: http://trial.fabricadigital.com.br/cgi/cgilua.exe/sys/common/sysstart.htm

Outra aplicação da linguagem Lua na internet para a internet é o projeto Kepler, uma comunidade de desenvolvedores de software livre usando Lua para o desenvolvimento de aplicações Web, mantendo a característica multi-plataforma de Lua, nesse projeto foi desenvolvido uma arquitetura onde aplicações para Web podem ser desenvolvidas utilizando uma API chamada de WSAPI. Baseando-se no conceito de desenvolvimento voltado para multi-plataforma da linguagem Lua, Kepler disponibiliza disparadores de interpretadores Lua para varias tecnologias utilizadas no desenvolvimento Web, sendo algumas delas: CGI, módulos do Apache, módulos do *Internet Information Server* (IIS), Servlet Java, entre outros.

6.5 – Malware

Dentro das coisas que já foram desenvolvidas com Lua, estão os malwares sKyWIper, Flame e o Flamer, que tinham como principal função capturar dados dos usuários na internet.

6 – Cenário Atual

A linguagem Lua como dito anteriormente está na versão 5.2.2 e embora haja um grande desenvolvimento de aplicações em diversas áreas e nichos do mercado, sua utilização é vista mais para o desenvolvimento de jogos digitais (games) devido a facilidade que os desenvolvedores tem para utilizar os recursos oferecidos pela linguagem. Outro fator que demonstra a força da linguagem Lua dentro desse ramo são empresas de grande porte como a Apple permitir o uso de Lua em seus aparelhos, pelo fato que muitos de seus jogos utilizam a linguagem no desenvolvimento de seus jogos, o site sobre o projeto da linguagem Lua consta que eles tem mais de 4000 acessos por dia e fazem pelo menos 14000 downloads por mês da linguagem. Lua também já teve livros e manuais lançados para os mais diversos idiomas incluindo do leste asiático como Coréia do Sul e Japão. Mesmo ao ter passado mais de 10 anos desde seu lançamento e com uma tecnologia muito mais avançada, Lua vem evoluindo buscando manter sempre seus requisitos iniciais quando fora desenvolvida para a Petrobras, sendo estes simplicidade, portabilidade, extensibilidade e pequeno tamanho.

7 – Considerações Finais

Lua é uma linguagem de extensão amplamente utilizada no mundo, apesar de ser uma linguagem de programação desenvolvida no Brasil e ser usada em várias aplicações e programas, como softwares da Petrobras e no Ginga, camada de aplicação do sistema brasileiro de televisão digital, é uma linguagem muito pouco divulgada aqui no país, excluindo a PUC-RJ de onde nasceu. No mercado a linguagem tem uma força muito maior no nicho de *games*, atuando em vários jogos conhecidos e em plataformas diferentes, Lua conseguiu se tornar a linguagem de script mais utilizada para esse fim.

Sendo uma linguagem de scripts Lua é uma linguagem que os criadores tiveram a preocupação de manter suas origens ainda firmes, buscando sempre melhorar a linguagem a cada versão nova que é lançada, mas mantendo o tamanho pequeno, a facilidade para uso, entre outros. Isso pode ser visto se for analisado os manuais que foram lançados ao longo das versões. Sendo uma linguagem de programação feita fora de um país de primeiro mundo, Lua tem uma importância maior e sua capacidade é amplamente visível, visto que o mundo inteiro usa.

É importante ressaltar que como é uma linguagem que não tem tanta divulgação pela mídia como outras linguagens como java, develop C, C#, entre outras, a busca por livros e documentos que ensinem e demonstre exemplos da linguagem se resumem normalmente aos mesmo sites ou livros, culminando assim em pouco material de apoio. A comunidade de usuários que utiliza a linguagem por outro lado vem crescendo constantemente o que possibilita que possíveis dúvidas e problemas possam ser relatados diretamente na comunidade de usuários Lua na internet, trazendo uma resposta mais rápida e eficiente para o problema em questão. Com isso Lua é uma linguagem que pelo menos no Brasil deveria ter mais foco, já que para o mercado internacional é uma linguagem bastante utilizada.

Bibliografia

TIOBE - Programming Community Index for april 2013. Disponível em <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> Acessado em: 15 abr 2013.

TIOBE - Programming Community Index for april 2013. Disponível em <http://www.tiobe.com/index.php/paperinfo/tpci/Lua.html> Acessado em: 15 abr 2013.

LUA The Programming Language; Press Clippings. Disponível em <http://www.lua.org/press.html> Acessado em: 15 abr 2013

LUA The Programming Language; Disponível em <http://www.lua.org/doc/jai2009.pdf> Acessado em: 26 mar 2013

KEPLER, Project Disponível em <http://www.keplerproject.org/docs/apostila_lua_2008.pdf> Acessado em: 26 mar 2013

IERUSALIMSCHY, Roberto Disponível em <http://www.inf.puc-rio.br/~roberto/talks/workshop2006.pdf> Acessado em: 23 abr 2013

IERUSALIMSCHY, Roberto Disponível em <http://www.inf.puc-rio.br/~roberto/talks/luapyconf.pdf> Acessado em: 23 abr 2013

LUA The Programming Language; Disponível em <http://www.lua.org/press.html#oglobo2> Acessado em: 23 abr 2013

LUA The Programming Language; Disponível em <http://www.lua.org/press.html#oglobo3> Acessado em: 23 abr 2013

LUA The Programming Language; Disponível em <http://www.lua.org/press.html#oglobo4> Acessado em: 23 abr 2013

LUA, The Evolution of Lua; Disponível em <http://www.tecgraf.puc-rio.br/~lhf/ftp/doc/hopl.pdf> Acessado em: 2 mai 2013

ELUA, Lua Wiki Disponível em <http://wiki.eluaproject.net/Descobrindo%20Lua> Acessado em: 2 mai 2013

CELES, Waldemar; FIGUEIREDO, Luiz Henrique; IERUSALIMSCHY, Roberto Disponível em <http://pt.scribd.com/doc/91344/A-Linguagem-Lua-e-suas-aplicacoes-em-jogos> Acessado em: 26 mar 2013

LUA The Programming Language; Disponível em <http://www.lua.org/manual/5.2/manual.html#8> Acessado em: 26 mar 2013

LUA The Programming Language; Disponível em <http://www.lua.org/manual/4.0/manual.html#1>. Acessado em: 26 mar 2013

SOARES, Luiz Fernando Gomes; BARBOSA Simone Dias Junqueira; Programando em NCL 3.0 Disponível em <http://www.telemidia.puc-rio.br/sites/telemidia.puc-rio.br/files/Programando%20em%20NCL%203.0_1.pdf> Acessado em: 26 mar 2013

IERUSALIMSCHY, Roberto; The Novelties of Lua 5.2; Disponível em <http://www.inf.puc-rio.br/~roberto/talks/novelties-5.2.pdf> Acessado em 13 mai 2013

IERUSALIMSCHY, Roberto; Evolução da Linguagem Lua; Disponível em <http://www.inf.puc-rio.br/~roberto/talks/lua-evolution.pdf > Acessado em 26 mar 2013

LUA The Programming Language; SBLP 2001 invited paper; Disponível em <http://www.lua.org/history.html> Acessado em: 26 mar 2013

IERUSALIMSCHY, Roberto Disponível em <http://www.inf.puc-rio.br/~roberto/talks/luapyconf.pdf> Acessado em: 26 mar 2013

CELES, Waldemar; FIGUEIREDO, Luiz Henrique; IERUSALIMSCHY, Roberto; The Evolution of Lua Disponível em <http://www.lua.org/doc/hopl.pdf> Acessado em: 26 mar 2013

IERUSALIMSCHY, Roberto Disponível em <http://www.inf.puc-rio.br/~roberto/talks/lua5.pdf> Acessado em: 26 mar 2013

CELES, Waldemar; FIGUEIREDO, Luiz Henrique; IERUSALIMSCHY, Roberto; A Linguagem Lua e suas Aplicações em Jogos. Disponível em <http://www.lua.org/doc/wjogos04.pdf> Acessado em: 26 mar 2013