Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Arcabouço Baseado em Componentes para o Desenvolvimento de Interface de Usuário de Aplicações para SmartTV

Danilo Araujo de Freitas

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Computação Pervasiva

Hyggo Oliveira de Almeida e Angelo Perkusich
(Orientadores)

Campina Grande, Paraíba, Brasil

Resumo

O conceito de SmartTV permite a instalação em TVs de aplicativos pelos usuários. Atualmente, algumas marcas disponibilizam seu próprio conjunto de ferramentas para que os desenvolvedores criem seus aplicativos e publiquem na respectiva loja. O middleware Ginga foi desenvolvido no país para ser utilizado no Padrão Brasileiro de TV Digital, tornando possível o desenvolvimento de aplicativos para permitir a interatividade entre TV e usuário. O Ginga recomenda que seja utilizada a linguagem de programação Lua para desenvolver os aplicativos, em conjunto com a Nested Context Language. O middleware especifica uma biblioteca em Lua com módulos utilizados para inserir desenhos na tela e gerenciar eventos da TV e do controle remoto. Porém, a biblioteca nativa provida pelo middleware é primitiva para o desenvolvimento de interface de usuário para aplicativos de SmartTV e TV Digital. Com isso, tal atividade torna-se improdutiva devido à grande quantidade de código, repetição de código e tratamento de eventos. Além disso, o desenvolvedor deve gerenciar a memória e eficiência nos aplicativos. Neste trabalho propõe-se um arcabouço para o desenvolvimento de aplicativos com interface de usuário para melhorar a produtividade dos desenvolvedores nessa atividade. O arcabouço é baseado em componentes de interface, o que diminui o esforço necessário para a criação de um elemento e facilita o reúso. Como resultado da solução, foi criada a biblioteca LuaSmartGUI, desenvolvida em Lua. Sete componentes presentes na solução foram avaliados individualmente. Também foi realizada outra avaliação utilizando desenvolvedores voluntários para desenvolver uma tela utilizando a solução proposta e a biblioteca nativa do Ginga. A avaliação mostrou que, apesar de um aumento no uso de memória de aproximadamente 867%, o uso de LuaSmartGUI mostrou ser aproximadamente 61% mais eficiente e apresentou ganho de 31% na produtividade. Os experimentos mostraram que as meta de produtividade e eficiência foram alcançados, mas é preciso melhorar o gerenciamento de memória da biblioteca para viabilizar seu uso.

Abstract

The concept of SmarTV allows users to install applications on TVs. Nowadays, some TV brands provide their own toolkits so that developers can create their own applications and publish in the brand store. Ginga middleware was developed in Brazil to be used in Brazilian Digital TV Standard, making possible the development of applications with interaction between users and TV. Ginga recommends developers to use the programming language Lua to develop applications, along with the Nested Context Language. The middleware specifies a library in Lua with modules which allow to draw on the screen and to manage TV and remote control events. However, the native library provided by Ginga is primitive for user interface development to SmartTV and Digital TV applications. Consequently, this task becomes unproductive due to challenges of handling events, writing very large codes and avoiding code replication. Furthermore, the programmers must deal with memory management and efficiency of the applications. In this work, we propose a framework for applications development with user interface to improve the productivity of developers in this task. The framework is based on interface components, which decreases the necessary effort to create an element and facilitates reusing. As result, the library LuaSmartGUI was developed in Lua. Seven components present in our solution were evaluated. Then, another evaluation was made with volunteers developers using the both approaches: our solution and the native Ginga library. Our results suggest that, despite the increase of approximately 867% in memory usage, LuaSmartGUI proved to be approximately 61% more efficient and showed a gain of about 31% in productivity. The experiments showed that the productivity and efficiency goals have been met, but it's necessary to improve the memory management for the library use to become viable.

Agradecimentos

Agradeço primeiramente à minha família que sempre me apoiou durante toda essa jornada de estudos, me cobrindo de amor, incentivos e conselhos, sempre me motivando e erguendo minha cabeça nos momentos difíceis em que eu temia fraquejar. Em especial, agradeço-os pela compreensão e paciência.

Agradeço a Deus acima de tudo, pois sempre posso buscar forças nele nos momentos de fraqueza.

Ao professor Hyggo Almeida. Sou muito grato pela paciência, compreensão e auxílio, pela dedicação empenhada, pelas motivações, pelos construtivos feedbacks e por toda a força. Agradeço também ao professor Rohit Gheyi por ter me co-orientado durante a maior parte do trabalho.

À Envision, por ter financiado parte do trabalho desenvolvido durante esta pesquisa. Aproveito para agradecer também à equipe de desenvolvimento da Envision no Embedded.

Agradeço à Lorena Maia, por ter me guiado durante o início do trabalho, me dando o suporte e incentivo necessários.

Aos membros do CGHackspace, que acompanharam a maior parte do meu trabalho de mestrado, me dando dicas e ajudando no que fosse necessário.

A todos os meus amigos da OFF Thread que, direta ou indiretamente, me ajudaram durante a trajetória do curso dando força e conselhos tanto profissionais quanto pessoais.

Também agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

A SmartTV pode ser definida como um modelo de TV interativa que permite a instalação de aplicativos e conectividade com a Internet. Com a facilidade de acesso à Internet e barateamento do produto, a tecnologia ganha mais espaço nas casas. Tal crescimento incentiva o investimento de empresas de vários ramos a investir em aplicativos para a SmartTV.

Em paralelo, houve bastante investimento para o crescimento da TV Digital, que possibilita interatividade com os programas de TV durante sua transmissão. Para o Sistema Brasileiro de TV Digital (SBTVD) [10], foi desenvolvido o Ginga [42] para ser utilizado como *middleware* padrão, desenvolvido em uma parceria da PUC-Rio com a UFPB. O middleware também define a NCL (Nested Context Language) [44], uma linguagem descritiva baseada em XML, utilizada junto com a linguagem de programação Lua [26], para o desenvolvimento de aplicativos para a TV Digital. Apesar de também existir uma versão em Java (Ginga-J), o Ginga recomenda o uso de NCL e Lua como forma de desenvolvimento padrão.

Mesmo não tendo evoluído no mercado como a SmartTV, o Governo Federal Brasileiro instituiu que, a partir de 2013, todas as fabricantes de TV no Brasil devem possuir o Ginga instalado em pelo menos 75% dos aparelhos fabricados no país, e 90% em 2014 [11]. Dessa forma, torna-se possível o desenvolvimento de aplicativos em Lua, pelo menos de forma indireta, na maioria das SmartTVs fabricadas no Brasil, desde que possua o Ginga instalado.

Empresas investem em aplicativos para SmartTV para levar seus serviços a mais usuários. O canal de TV a cabo ESPN disponibiliza gratuitamente um aplicativo chamado ESPN Player [21] para Smart TVs da Samsung e LG para visualização de seu conteúdo. Netflix é uma empresa que fornece transmissão de conteúdo televisivo pela Internet, disponibili-

zando um aplicativo para várias plataformas, como computador, smartphones e videogames. Disponibiliza também para TVs LG, Panasonic, PHILIPS, Samsung, SHARP e Sony [31]. Outros aplicativos já conhecidos em outras plataformas também possuem versões para TVs, como Youtube [23], Deezer [16] e Skype [40].

1.1 Problema

O middleware Ginga especifica alguns módulos em Lua para o desenvolvimento de aplicações. Dentre eles, o módulo *Canvas*, utilizado para desenhar e escrever na tela, e o módulo *Event*, responsável por gerenciar eventos como o pressionamento de botões no controle remoto, mudança de canal, etc. são suficientes para desenvolver aplicativos com interface de usuário para a TV. Os desenvolvedores da equipe de desenvolvimento do projeto da Envision, no Laboratório de Sistemas Embarcados e Computação Pervasiva - Embedded [18], se queixaram da lentidão dos aplicativos desenvolvidos e algumas vezes não havia mais memória disponível durante a execução do aplicativo. Dessa forma, algumas diretrizes foram definidas, como realizar a chamada ao *Garbage Collector* em pontos específicos e a tentativa de, quando houvesse alguma alteração na tela, redesenhar apenas áreas específicas.

Porém, a especificação do Ginga limita-se apenas a funções primitivas, como desenhar retângulos, inserir imagens e escrever textos. Essa limitação torna o trabalho do desenvolvedor pouco produtivo [38], pois dificulta o reúso dos elementos criados, além de tornar complexo o gerenciamento de eventos para interação entre usuário e aplicação.

Utilizando os módulos do Ginga, o desenvolvedor também precisa se preocupar com o desempenho das aplicações criadas, pois, assim como outros sistemas embarcados, as TVs atualmente possuem configurações de *hardware* limitadas (em termos de processamento e memória). Portanto, torna-se necessário seguir diretrizes de desenvolvimento bastante rígidas para evitar problemas como a lentidão no tempo de resposta e consumo excessivo e vazamento de memória.

Existem soluções próprias de algumas fabricantes, como a Samsung Smart TV SDK, da Samsung [39], e Web SDK, da LG [28]. Porém, são soluções específicas para cada marca, limitando sua utilização em TVs de outras fabricantes.

1.2 Exemplo Motivante

O uso dos módulos Canvas e Event especificados pelo Ginga possibilitam que o desenvolvedor crie telas em seus aplicativos. Porém, o uso apenas desses módulos torna o desenvolvimento pouco produtivo, devido às limitações dos módulos.

Com o objetivo de justificar a criação de um arcabouço, temos uma tela simples, com apenas um botão, que pode ser pressionado, mas sem realizar ações. A tela pode ser vista na Figura 1.1.

Figura 1.1: Tela de exemplo desenvolvida utilizando o módulo Canvas

A implementação da tela pode ser encontrada no Código 1.1. A Linha 1 cria um novo objeto `canvas`, que representará o pedaço da tela responsável por desenhar o botão. A Linha 3 cria a função responsável por desenhar o botão, que possui 20 linhas de código. A Linha 4 limpa a área do botão para não desenhar por cima de outro desenho. A Linha 5 altera a cor de desenho e a 6 desenha um retângulo. As Linhas 7 a 19 são responsáveis por desenhar as bordas do botão, que têm suas cores dependente do estado (se está pressionado ou não). Já as Linhas 20 e 21 escrevem o texto do botão ("Button"). Finalmente, as Linhas 22 e 23 são utilizadas para posicionar o botão na tela e desenhá-lo.

Na Linha 25 é definida a função responsável por gerenciar o evento de pressionar ou soltar o botão "ENTER"do controle remoto. A Linha 28 é executada caso o tipo do evento

recebido seja do tipo "press"(tecla pressionada) e se a tecla pressionada for "ENTER". Nesse caso, irá chamar a função de desenhar o botão no estado pressionado. Já a linha 32 é chamada caso o evento seja do tipo "release"(soltar a tecla) e a tecla seja "ENTER". Assim, irá chamar a função de desenhar o botão no estado não pressionado.

A Linha 37 é utilizada para registrar a função de gerenciar os eventos da tecla "ENTER", definida na Linha 25. Já a Linha 38 finalmente desenha o botão pela primeira vez, com o estado inicial sendo não pressionado.

Portanto, a partir do exemplo mostrado, nota-se o grande esforço do desenvolvedor para criar um componente simples. A quantidade de linhas de código para desenhar um botão pode ser considerada grande. Além disso, em aplicações maiores, pode haver replicação de código para desenhar o mesmo elemento em locais diferentes. Também é necessário replicar o código quando desenvolver outro aplicativo. Além disso, o desenvolvedor precisa gerenciar os eventos do controle remoto manualmente, aumentando o esforço de desenvolvimento com a quantidade de elementos na tela. Também pode ser necessário gerenciar o uso de memória manualmente, para que não cresça ao longo do uso da aplicação.

1.3 Objetivos

Neste trabalho tem-se como objetivo a criação de um arcabouço para o desenvolvimento de aplicações com interface gráfica para SmartTV. O arcabouço abstrai o módulo Canvas para o usuário. A solução procura aumentar a produtividade dos desenvolvedores e atender os requisitos de desempenho, sendo eles a eficiência (tempo de resposta dos componentes) e consumo de memória. É necessário atender tais requisitos devido ao *hardware* limitado das TVs. O resultado da solução proposta foi chamada de LuaSmartGUI[1].

A solução proposta define 14 componentes, como botão, texto e *checkbox*, reduzindo a quantidade de linhas de código necessárias para a criação de elementos na tela e facilitando o reúso dos mesmos. O arcabouço contém componentes predefinidos, de acordo com um levantamento realizado em aplicativos desenvolvidos pela equipe de desenvolvimento do projeto da Envision, no laboratório Embedded. No levantamento, foram catalogados pos-

[1] Apesar de ter sido desenvolvida em Lua, o arcabouço criado também pode ser desenvolvido em outras linguagens.

Código 1.1: Exemplo de código para desenhar um botão

```
1  local button = canvas:new(100, 30)
2
3  function drawButton(pressed)
4      button:clear()
5      button:attrColor("#AAAAAA")
6      button:drawRect("fill", 0, 0, 100, 30)
7      if (pressed) then
8          button:attrColor("#000000")
9      else
10         button:attrColor("#FFFFFF")
11     end
12     button:drawLine(0, 0, 0, 30)
13     button:drawLine(0, 0, 99, 0)
14     if pressed then
15         button:attrColor("#FFFFFF")
16     else
17         button:attrColor("#000000") end
18     button:drawLine(0, 29, 99, 29)
19     button:drawLine(99, 0, 99, 29)
20     button:attrFont("TiresiasScreenfont", 16, "Normal")
21     button:drawText(10, 3, "Button")
22     canvas:compose(5, 5, button)
23     canvas:flush()
24 end
25 function press(evt)
26     if evt.type == "press" then
27         if evt.key == "ENTER" then
28             drawButton(true)
29         end
30     elseif evt.type == "release" then
31         if evt.key == "ENTER" then
32             drawButton(false)
33         end
34     end
35 end
36
37 event.register(press)
38 drawButton(false)
```

síveis componentes que poderiam ser utilizados em aplicativos. O desenvolvedor também pode utilizar o arcabouço para criar componentes personalizados, caso necessário.

Além de componentes, o arcabouço é baseado em layouts que definem regiões na tela e facilita o posicionamento de componentes. O uso de layouts facilita o trabalho do desenvolvedor de organizar os componentes na tela, sem precisar posicionar cada um manualmente.

Outra facilidade proposta pela solução é o gerenciamento de eventos entre os componentes e o controle remoto. É possível definir o comportamento de vários componentes de forma simples, utilizando o padrão de projeto *Observer* para responder a eventos gerados pela ação do controle remoto.

1.4 Avaliação

Foram realizados dois experimentos com o objetivo de analisar a biblioteca LuaSmartGUI com relação ao tempo de resposta, consumo de memória e produtividade, no ponto de vista do desenvolvedor de software no contexto de aplicativos para SmartTV.

O primeiro experimento foi realizado através da coleta e análise de métricas de produtividade e desempenho de sete componentes desenvolvidos utilizando LuaSmartGUI e a biblioteca nativa do Canvas. A partir dos dados coletados, foram estimados os valores esperados para o segundo experimento. Este foi realizado utilizando sete desenvolvedores voluntários para desenvolver uma mesma tela duas vezes: uma utilizando LuaSmartGUI e outra utilizando apenas o módulo Canvas. Foram coletadas métricas de produtividade e desempenho para responder três questões de pesquisa definidas:

- **Q1.** *Aplicativos desenvolvidos utilizando o arcabouço proposto são mais eficientes que os desenvolvidos com os módulos do Ginga?*
- **Q2.** *Aplicativos desenvolvidos utilizando o arcabouço proposto utilizam menos memória que os desenvolvidos com os módulos do Ginga?*
- **Q3.** *O uso do arcabouço proposto aumenta a produtividade dos desenvolvedores em relação aos módulos do Ginga?*

Os resultados do experimento apontaram ganho de produtividade de aproximadamente 31%, eficiência satisfatória (ganho de aproximadamente 61% no tempo de resposta e perda

de aproximadamente 110% no tempo de carregamento) e aumento no uso de memória em cerca de 867% quando utilizado LuaSmartGUI, respondendo positivamente a primeira e terceira questões definida.

1.5 Contribuições

Como resultado deste trabalho, as principais contribuições são:

- Um modelo de arcabouço para aumentar a produtividade de desenvolvedores de aplicações com interface de usuário para SmartTV;
- Um catálogo de 22 componentes em 9 aplicativos desenvolvidos por uma equipe de desenvolvimento da Envision;
- Uma avaliação com desenvolvedores para analisar o ganho de produtividade da ferramenta;
- Uma implementação do arcabouço proposto em Lua, que pode ser executado em qualquer TV que disponibilize os recursos do Ginga.

1.6 Organização do Trabalho

Neste trabalho, o Capítulo 2 introduz os conceitos necessários para melhor entendimento deste trabalho. Já o Capítulo 3 apresenta o arcabouço proposto neste trabalho. O Capítulo 4 mostra como foram realizados os experimentos e os resultados obtidos. O Capítulo 5 comenta sobre os trabalhos relacionados com este e o Capítulo 6 apresenta as últimas considerações sobre o trabalho e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo são introduzidos os conceitos necessários para o desenvolvimento e entendimento deste trabalho. Na Seção 2.1 são mostrados os conceitos por trás das TVs interativas. O middleware Ginga é apresentado na Seção 2.2. Na Seção 2.3 é introduzida a Nested Context Language. Já na Seção 2.4 são apresentados os módulos do Ginga utilizados no desenvolvimento. Na Seção 2.5 é apresentada a linguagem de programação Lua. Finalmente, na Seção 2.6 são apresentados os conceitos no desenvolvimento de aplicações com interface de usuário no contexto geral e no contexto de Smart TVs, também mostrando o atual estado da arte.

2.1 TV Interativa

Antes do surgimento da TV Digital, a geração do sinal som e imagem, transmissão e recepção eram realizadas de forma analógica [4]. Ou seja, todo o sinal gerado, transmitido e recebido era analógico. Atualmente, toda a geração é realizada, enviada e recebida através de sinais digitais [37]. A forma atual permite a transmissão de áudio e vídeo com maior qualidade.

O sinal digital também permite a transmissão de dados que não estão diretamente relacionados com a imagem e vídeo. Isso possibilita que sejam enviadas, por exemplo, informações sobre a programação dos canais, através de um guia EPG (*Eletronic Programming Guide*), bastante utilizado pelas operadoras de TV por assinatura. O sinal também pode ser utilizado para permitir a interatividade entre o usuário o conteúdo atual do programa ou canal de TV que está sendo assistido. Pode exemplo, durante a transmissão de uma partida de futebol, é

possível que o usuário veja placares de outras partidas, veja anúncios publicitários, etc.

Toda essa interatividade na TV Digital é provida pelos dados transmitidos e cada receptor é responsável por interpretar a informação e levá-la até o usuário. O responsável por realizar a comunicação entre os dados recebidos e as aplicações interativas é o middleware instalado. No Brasil, foi desenvolvido o middleware Ginga para ser utilizado como padrão no Sistema Brasileiro de TV Digital (SBTDV). O Ginga será detalhado na Seção 2.2. A Figura 2.1 mostra as camadas desde os recursos da TV até a camada de aplicações. O middleware provê uma API que pode ser utilizada pela aplicações para que possam utilizar os recursos da TV. No caso do SBTVD, o Middleware é o Ginga e a API são os módulos providos por ele.

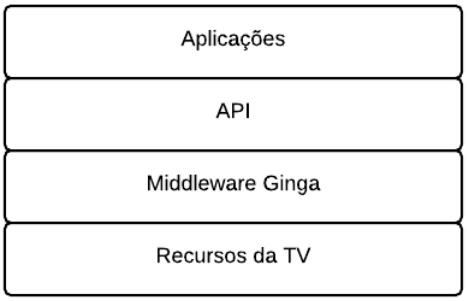

Figura 2.1: Estrutura básica das camadas de uma TV Digital

Uma SmartTV é um modelo de televisão que permite a conectividade com a Internet e a instalação de aplicativos. Ela também é um modelo de TV interativa, mas, diferente da TV digital, sua interatividade é provida pelos aplicativos instalados, independente do sinal ser digital ou analógico.

Como pôde ser visto, TV Digital e SmartTV são dois conceitos diferentes de TV Interativa. Porém, as TVs fabricadas atualmente possuem alta qualidade de som e imagem, incluindo as SmartTVs. Portanto, devido à qualidade das TVs fabricadas, é provável que qualquer SmartTV produzida atualmente possa ser considerada uma TV Digital. Dessa forma, uma Smart TV fabricada no Brasil deve atender aos requisitos de TV Interativa imposto pelo governo.

2.2 Ginga

Existem três padrões de TV Digital: o americano ATSC (*Advanced Television System Comitee*), o europeu DVB (*Digital Video Broadcasting*) e o japonês ISDB-T (*Integrated System Digital Broadcast Terrestrial*). Em 2006, o governo brasileiro anunciou a escolha do ISDB-T como padrão para o SBTVD, sendo chamado de ISDB-TB.

A Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio, em parceria com a Universidade Federal da Paraíba (UFPB), iniciou o desenvolvimento do middleware (Ginga) responsável por gerenciar o conteúdo de interatividade da TV Digital no Brasil e em outros países da América Latina. O Ginga é responsável por realizar a comunicação entre a TV e os aplicativos interativos.

O middleware possui duas versões: Ginga-NCL [3] (permite a leitura de aplicativos que utilizam *Nested Context Language*) e Ginga-J (fornece uma API em Java para o desenvolvimento de aplicativos), sendo a segunda menos utilizada. O Ginga-NCL utiliza documentos hipermídia (documentos com links para conteúdos multimídia e textuais) para realizar o sincronismo (temporal e espacial) do conteúdo multimídia, além de realizar a interação com o usuário.

O Ginga foi desenvolvido para ser executado em uma set-top box, que converte o sinal digital e executa os programas. Também existem TVs fabricadas já com um conversor de sinal embutido, junto com o Ginga instalado, eliminando a necessidade do uso das set-top boxes.

2.3 NCL

A *Nested Context Language* (NCL) é uma linguagem declarativa, baseada em XML, desenvolvida na PUC-Rio, para a autoria de documentos hipermídia. Dessa forma, é possível utilizá-la para descrever a execução de um programa de TV, além de referenciar scripts em Lua que auxiliam na interação com o usuário. Quando utilizada em conjunto com Lua, é chamada de NCLua.

Em NCL, existem alguns conceitos necessários para o desenvolvimento dos programas, como regiões, mídias, conectores e âncoras. Esses atributos são necessários para definir

regiões na tela, conteúdos multimídia e ligações entre eles e eventos.

É possível utilizar NCLua tanto para a descrição de programas para a TV Digital quanto no desenvolvimento de aplicativos para a SmartTV.

2.4 Módulos do Ginga

O middleware Ginga especifica vários módulos que devem ser estar disponíveis para as aplicações. Dentre eles, só é necessário citar os módulos *Canvas* e *Event*, pois são suficientes para desenvolver aplicativos.

O módulo Canvas define a entidade com acesso global `canvas`, que disponibiliza funções primitivas para desenhar na tela da TV, como linha, retângulo, texto e imagem. Um `canvas` também pode ser composto por outros `canvas`, permitindo a criação de camadas de desenhos.

O módulo Event define estruturas necessárias para gerenciar os eventos recebidos pelo Ginga. Tais eventos podem ser gerados por várias fontes, como TV, código NCL ou controle remoto. Os eventos gerados pelo controle remoto, como pressionar ou soltar um botão, são suficientes para o desenvolvimento de aplicações.

2.5 Lua

Lua é uma linguagem de programação criada em 1993 na PUC-Rio. Atualmente ela encontra-se na versão 5.3.0. É uma linguagem interpretada e dinamicamente tipada. Devido a essas características, é bastante utilizada na criação de scripts que complementam o funcionamento de programas. Uma das principais áreas que utilizam a linguagem é no desenvolvimento de jogos.

Por ser gratuita, sua possibilidade de uso torna-se maior, inclusive para aplicações comerciais, e de código aberto. Além disso, pode ser facilmente embutida, já que pode ser executada em qualquer ambiente que possua um compilador C padrão e todo o seu código, junto com a documentação, ocupa cerca de 960 KB em disco.

Todas as estruturas de dados em Lua são representadas utilizando tabelas, a única estrutura nativa provida pela linguagem. Tabelas podem indexar qualquer tipo de valor, incluindo

funções e outras tabelas. Também existem metatabelas, que permitem que o desenvolvedor altere o comportamento de uma tabela, personalizando as operações realizadas nela.

Apesar de não dar suporte a Orientação a Objetos, é possível utilizar metatabelas para simular o paradigma. Dessa forma, é possível fazer com que tabelas se tornem objetos e possuam atributos e funções. Como construtor, é utilizada uma função armazenada na tabela que retorne uma cópia dela mesma.

2.6 Desenvolvimento de Aplicativos com Interface Gráfica

Cerca de 60% do código de uma aplicação encontra-se na GUI [5]. Portanto, independente da plataforma destino, é estritamente necessário um conjunto de ferramentas que auxiliem o desenvolvedor na criação de elementos, posicionamento e gerenciamento de eventos.

Existem bibliotecas de componentes disponibilizadas junto com um *framework* para gerenciá-los, chamadas de *Toolkits*. Esse tipo de ferramenta têm tido sucesso como solução para desenvolvimento de aplicações com interface de usuário [30]. Em conjunto com linguagens de script, traz as vantagens de rápida prototipação e facilidade de alteração. O uso de Orientação a Objeto facilita a modelagem dos componentes como objetos visíveis, com estados e operações específicas.

Pode ser necessário reajustar os componentes quando há alterações no layout da tela [27]. No contexto de TV, as aplicações geralmente são executadas no modo "tela cheia", tornando desnecessária a aplicação de técnicas de telas dinâmicas.

2.6.1 Desenvolvimento de Interface de Usuário em SmartTVs

Assim como qualquer outro sistema embarcado, é necessário tomar alguns cuidados no desenvolvimento de aplicativos para SmartTVs. As limitações de *hardware* existentes nas TVs devem ser consideradas. Os aplicativos devem ser eficientes o suficiente para não provocar atrasos no tempo de resposta para o usuário e não utilizar mais memória que o necessário.

A quantidade de ferramentas disponíveis para desenvolvimento de aplicativos para TVs ainda é limitada. Algumas marcas disponibilizam seu próprio conjunto de ferramentas, como a Samsung [39] e LG [28]. Elas também permitem que desenvolvedores publiquem seus aplicativos em suas lojas e outros usuários instalem em suas TVs, da mesma forma como é

realizado nos SmartPhones. Porém, o uso de ferramentas comerciais tornam o desenvolvimento limitado apenas a uma marca ou plataforma específica.

2.6.2 Estado da Arte

Existem trabalhos e soluções na área de desenvolvimento de aplicativos para TVs que contribuíram para a evolução deste trabalho. São trabalhos realizados por pesquisadores buscando melhorar o desenvolvimento para TVs e soluções comerciais de fabricantes ou que possam ser utilizadas no contexto de aplicativos para TVs.

Foi realizado um experimento simples para avaliar o uso de alguns frameworks GUI em TVs [8]. Apesar de não ter utilizado ferramentas relacionadas com Lua nem com o Ginga, foram levantados pontos importantes para a criação de uma ferramenta com o objetivo proposto. Os pontos utilizados como base para a solução foram:

- **Consumo de memória**: como os recursos das TVs são limitados, é essencial se preocupar com o consumo de memória gasto pelos aplicativos que utilizem a solução;
- **Documentação**: é essencial a existência de uma boa documentação para que os desenvolvedores possam aprender a utilizar a ferramenta com facilidade;
- **Navegação**: é necessário prover uma forma de navegar pelos elementos na tela através de pelo menos quatro direções (cima, baixo, esquerda e direita).

Foi proposto um framework GUI para TVs Interativas utilizando o *Abstract Window Toolkit* (AWT) em Java [32] que define componentes UI [15]. Na solução proposta, existem componentes que podem encapsular outros, chamados de *containers*. Para focar no reúso dos componentes, nenhum deles possuíam funcionalidades especificas de alguma aplicação. O reuso de componentes também apresentou redução no uso de memória.

Existem duas soluções comerciais disponíveis desenvolvidas pelas próprias fabricantes. A Samsung possui o *framework* Caph, que utiliza Javascript, HTML e CSS para o desenvolvimento de aplicações. A solução disponibiliza uma API de componentes (CAPH WUI Widgets), que permite a instanciação e personalização de componentes. Já a LG disponibiliza a LG Web API, que é um conjunto de bibliotecas em Javascript e HTML, permitindo a instanciação de componentes pré-definidos, como `Button` e `Label`. Apesar de utilizarem

Javascript e HTML, as soluções providas pela Samsung e LG não são compatíveis uma com a outra.

Qt [34] é um *framework* destinado ao desenvolvimento de aplicações multi-plataforma em C++. Ela provê uma biblioteca de componentes UI e possui uma estrutura de layouts. A ferramenta também define a linguagem declarativa QML, permitindo a criação de telas de forma declarativa. Apesar de ser uma ferramenta inicialmente voltada para aplicações *desktop*, também pode ser utilizada em sistemas embarcados.

O Google Web Toolkit (GWT) [41] é uma ferramenta para desenvolvimento Web que utiliza JavaScript. Pode ser utilizado para desenvolver aplicativos para a Google TV [24]. GWT atualmente suporta cinco navegadores [25]. Portanto, torna-se possível a exeucção de aplicativos GWT em TVs, desde que utilize um dos navegadores suportados pela ferramenta.

Assim como Lua, Python é uma linguagem interpretada e dinamicamente tipada. Ela provê uma biblioteca padrão para desenvolvimento GUI, chamada TkInter [22], que dispõe um conjunto de *widgets* que podem ser reutilizados em várias aplicações. Assim como Qt e Lua, também pode ser embarcada em plataformas.

Capítulo 3

LuaSmartGUI: Uma biblioteca para o desenvolvimento de interface de usuário de aplicações para SmartTV

Como foi visto, o desenvolvimento de aplicativos com interface de usuário para SmartTV não é produtivo o suficiente. As ferramentas disponíveis trazem limitações de escopo, pois limitam o desenvolvimento a um conjunto específico de TVs, ou limitações de produtividade por serem primitivas.

Propõe-se um arcabouço baseado em componentes para desenvolvimento de interface de usuário com o objetivo de aumentar a produtividade no desenvolvimento de aplicativos para SmartTV, além de abranger um escopo maior que outras soluções já existentes. A solução apresentada foi chamada de LuaSmartGUI[1].

Neste capítulo, a biblioteca é apresentada na Seção 3.1. Na Seção 3.2 são apresentados os componentes presentes na biblioteca. Já na Seção 3.3, são apresentados detalhes sobre a arquitetura da biblioteca. Por último, na Seção 3.4, são apresentados pequenos exemplos da ferramenta.

[1] O código do framework e sua documentação encontra-se disponível em https://sites.google.com/site/luasmartgui

3.1 Visão Geral

A fim de maximizar a possibilidade de uso da solução, ela foi modelada para utilizar os módulos providos pelo middleware Ginga. Também foi desenvolvida em Lua, pois todas as versões do Ginga devem dar suporte a NCLua, aumenando a possibilidade de uso nas SmartTVs fabricadas no Brasil.

A modelagem do arcabouço foi baseada em outras soluções de interface de usuário já consolidadas no mercado. Sua arquitetura foi baseadas nos frameworks Qt [29] e Android [12]. A *Application Programming Interface* (API) criada e o gerenciamento de eventos foram baseados em Android.

Apesar de Lua não prover suporte nativo a orientação a objetos, é possível simular tal funcionalidade a partir de metatabelas. Dessa forma, o framework foi criado simulando orientação a objeto, unindo as vantagens do uso de linguagem interpretada, orientação a objetos e toolkit [30].

3.2 Componentes

Para a criação dos componentes, foi necessário inicialmente realizar análise em aplicativos já existentes e catalogar os possíveis componentes que poderiam ser utilizados em suas telas. A fim de obter uma maior expressividade dos componentes catalogados, os aplicativos selecionados possuíam contextos variados (dois jogos, dois aplicativos de rede social, dois de acesso a conteúdo multimídia, três de acesso a portais de notícias e um para gerenciamento de atividades pessoais).

O processo de coleta se deu através da visualização de todas as telas nos aplicativos analisados, identificando possíveis elementos que poderiam ser reutilizados. No total, foram catalogados nove aplicativos desenvolvidos pela equipe de desenvolvimento do projeto da Envision no laboratório Embedded[2], totalizando 490 possibilidades de componentes (22 distintos). A Tabela 3.1 mostra os componentes catalogados.

Foram então selecionados os componentes que foram catalogados pelo menos três vezes, resultando em 14 componentes[3]. Os componentes foram divididos em dois grupos. Um

[2]Devido às restrições de confidencialidade da Envision, não será possível informar os aplicativos utilizados.

[3]Apesar de o `VideoWidget` estar entre os componentes, sua implementação está limitada às TVs da AOC.

Componente	Quantidade	Componente	Quantidade
Label	184	ImageWidget	97
VerticalLayout	37	HorizontalLayout	32
RelativeLayout	32	TextWidget	29
ListWidget	24	Button	12
EditText	8	GridLayout	8
Checkbox	6	SeekBar	4
VideoWidget	3	Keyboard	3
Dialog	3	MenuSlider	2
MenuHorizontal	1	GridWidget	1
TextArea	1	TableLayout	1
CalendarWidget	1	MenuSpinner	1
Total	490		

Tabela 3.1: Possibilidades de componentes identificadas

`Layout` é responsável por posicionar outros componentes dentro dele, facilitando a organização da tela. Não possui interação com o usuário e normalmente são invisíveis a ele, a não ser que sejam configurados para serem mostrados na tela. Já o `Widget` é um componente visível para usuário e que possuem ações específicas. A Tabela 3.2 descreve cada componente. As Figuras 3.1, 3.2 e 3.3 mostram o diagrama de classes dos componentes da biblioteca.

Todos os componentes possuem atributos que auxiliam da personalização de seu uso na tela. Existem atributos comuns a todos os componentes, como `width` e `height`, que definem o tamanho do componente, e atributos específicos, como o `image` do `ImageWidget`, que informa o caminho para uma imagem. Uma lista com os componentes e todos os seus atributos pode ser encontrada no tutorial presente no Apêndice B.

3.3 Arquitetura e Design

A solução LuaSmartGUI, na arquitetura de camadas de uma TV, representa uma camada entre a camada de aplicações e de módulos do Ginga. A Figura 3.4 mostra a nova estrutura

Componente	Descrição
HorizontalLayout	Posiciona outros componentes de forma horizontal, da esquerda para a direita
VerticalLayout	Posiciona outros componentes de forma vertical, de cima para baixo
GridLayout	Posiciona os componentes em forma de grade
RelativeLayout	Posiciona os componentes de acordo com a posição de outros componentes já posicionados
Label	Escreve um texto na tela
ImageWidget	Insere uma imagem na tela
TextWidget	Escreve um texto na tela. Semelhante ao Label, mas pode receber ações do usuário
ListWidget	Posiciona componentes em forma de lista com páginas, podendo ela ser horizontal ou vertical
Button	Um botão que pode ser pressionado
EditText	Uma caixa para inserir texto através de um *Keyboard*
Checkbox	Uma caixa que pode ser marcada ou desmarcada
SeekBar	Uma barra de progresso horizontal que pode ser controlada
Keyboard	Teclado para inserir texto
Dialog	Mostra uma janela no estilo *pop-up*

Tabela 3.2: Componentes presentes na biblioteca LuaSmartGUI

de camadas de uma TV com a solução (a Figura 2.1 mostra as camadas originais). Com a inserção do arcabouço, as aplicações podem ser desenvolvidas sem utilizar diretamente os módulos do Ginga e utilizar diretamente os componentes providos pela solução.

O arcabouço segue uma modelagem hierárquica de componentes, onde todo componente possui um pai e alguns podem ter filhos (`Layout`, `ListWidget` e `Dialog`). Esta modelagem hierárquica foi baseada na modelagem utilizada em Android e Qt. Dessa forma, a posição de cada componente na tela é relativa ao seu pai.

Cada componente possui o seu próprio `canvas`, que é posicionado dentro de seu pai. Dessa forma, apesar de possivelmente aumentar o uso de memória da tela, facilita o gerenciamento dos componentes pela solução e pode ajudar no tempo utilizado para desenhar os

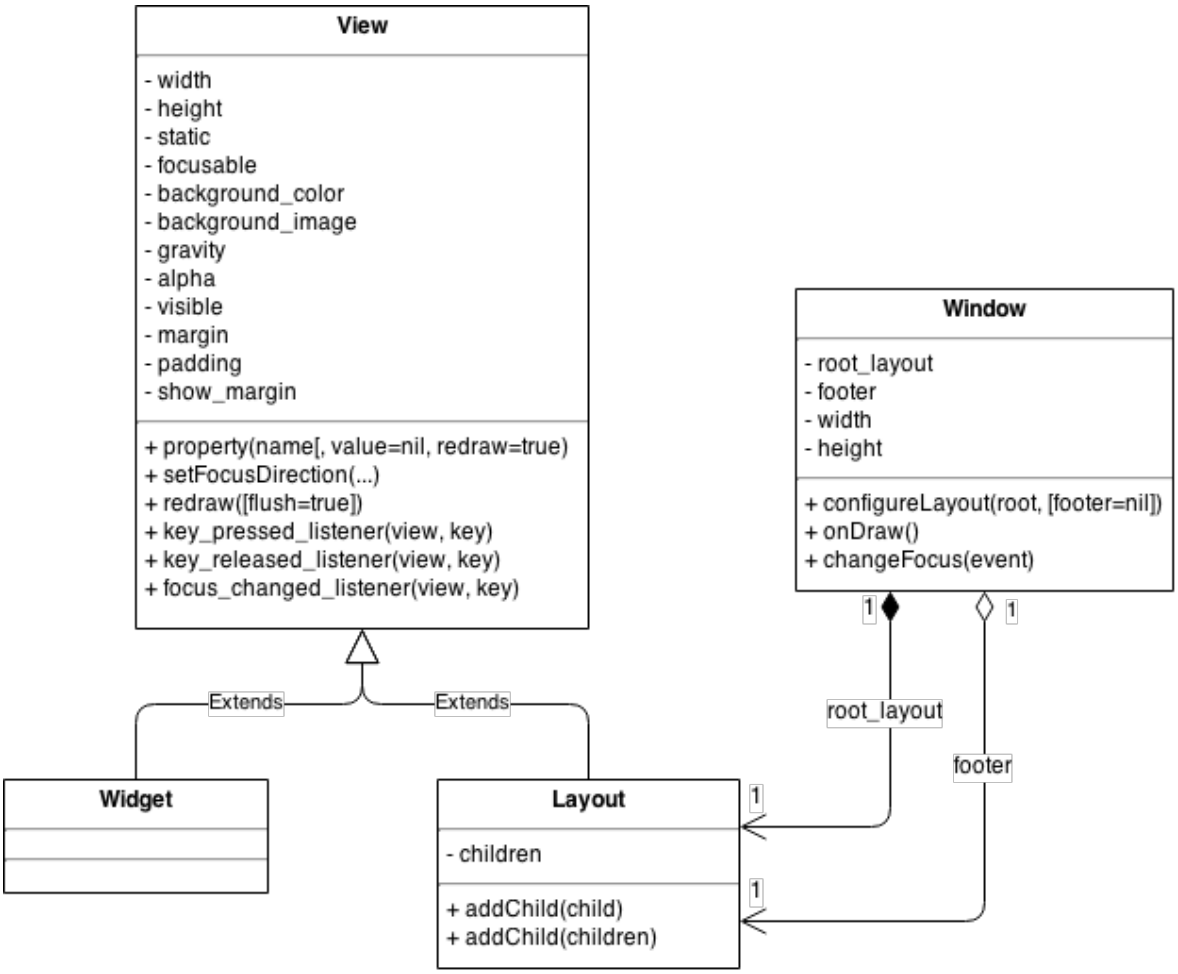

Figura 3.1: Diagrama de classes do componente `View`

componentes. Assim, quando há alteração no estado da tela, apenas os componentes afetados precisam ser desenhados novamente. Também é mais fácil liberar memória quando deseja-se remover o `canvas` de componentes específicos.

Para gerenciar os eventos de controle remoto, foi utilizado o padrão de projeto *Observer*. Desta forma, o usuário precisa definir apenas a função que executa o evento e associa ao componente desejado, podendo reutilizar funções.

3.4 Exemplos

O mesmo exemplo utilizado como exemplo motivante, mostrado na Figura 1.1 será desenvolvido utilizando a solução proposta. O Código 3.1 mostra criação da tela de exemplo com LuaSmartGUI. Comparando com a solução inicial vista no Código C.1, é possível notar a

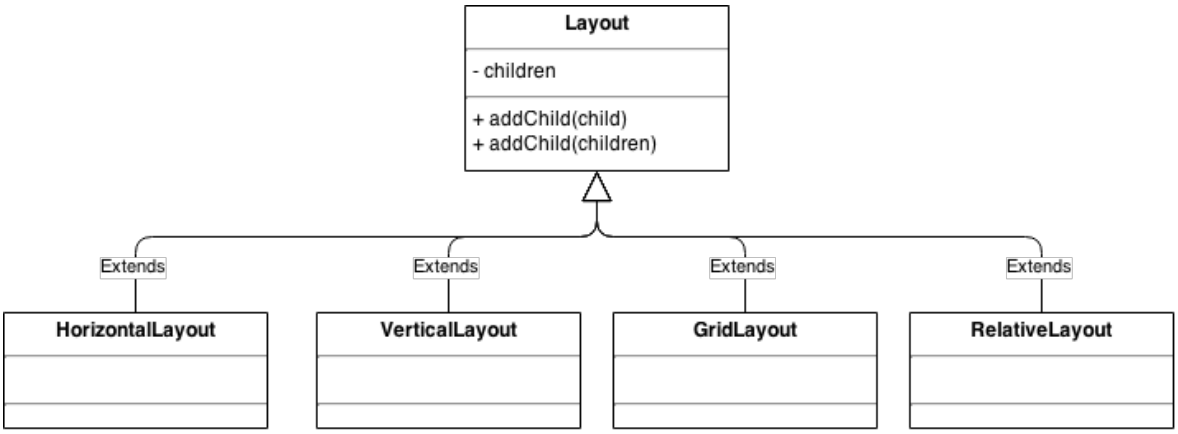

Figura 3.2: Diagrama de classes dos componentes do tipo `Layout`

diferença no tamanho do código necessário para criar o botão.

A Linha 1 do Código 3.1 importa os elementos de LuaSmartGUI para o arquivo atual, possibilitando o uso de seus componentes. A Linha 3 cria um componente do tipo `Window`, que representa a tela da TV. As Linhas 3 e 4 criam um `HorizontalLayout` define como o `Layout` raiz da tela. As Linhas 7 e 8 criam um componente `Button`, definindo seu tamanho e texto, e o adicionam ao `Layout` da tela. As Linhas 10 à 14 criam a função necessária para que o `Window` gerencie os eventos de controle remoto.[4]

O Código 3.1 possui no total 11 linhas, desconsiderando as linhas em branco. Comparando com o Código 1.1, com 36 linhas, temos um ganho de 25 linhas para desenhar apenas um componente. Outros exemplos utilizando LuaSmartGUI podem ser vistos no tutorial encontrado no Apêndice B.

[4] Atualmente o código encontrado a partir da Linha 10 é necessário em qualquer tela LuaSmartGUI. Porém, espera-se que essas linhas não sejam mais necessárias no futuro e sejam realizadas de forma automática.

Código 3.1: Criação de um botão com LuaSmartGUI

```
1  require "luasmartgui.lsgui"
2
3  local w = lsgui.Window:new{}
4  local layout = lsgui.HorizontalLayout:new{static=true}
5  w:configureLayout(layout)
6
7  local button = lsgui.Button:new{width=100, height=30, text="Button"}
8  layout:addChild(button)
9
10 function handler(evt)
11     w:changeFocus(evt)
12 end
13
14 event.register(handler)
15
16 w:onDraw()
```

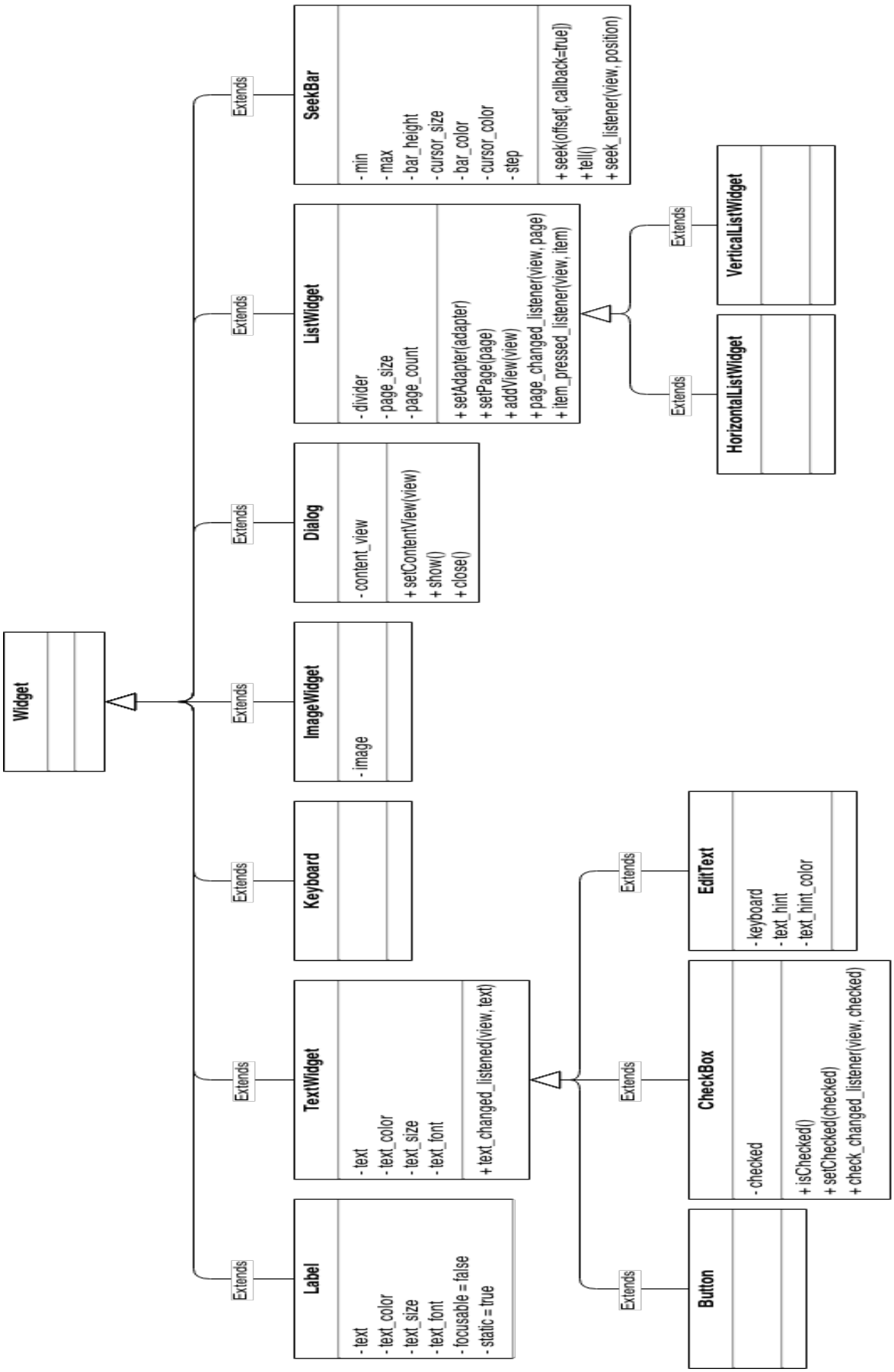

Figura 3.3: Diagrama de classes dos componentes do tipo `Widget`

Figura 3.4: Nova estrutura de camadas com a inserção de LuaSmartGUI

Capítulo 4

Avaliação

Neste capítulo, são descritos os experimentos [7] realizados com o objetivo de analisar a biblioteca LuaSmartGUI com relação ao tempo de resposta, uso de memória e produtividade, no ponto de vista do desenvolvedor de software no contexto de aplicativos para SmartTV.

Utilizando o método *Goal Question Metric* (GQM), foram definidas perguntas para auxiliar na escolha das métricas a serem coletadas pelo experimento. Para as duas ferramentas participantes dos experimentos (LuaSmartGUI e Ginga), foram definidas as as perguntas e métricas apresentadas na Tabela 4.1.

A partir das métricas definidas, foram planejados e executados dois experimentos, cada um coletando métricas específicas. O primeiro avaliou os componentes individuais da biblioteca, comparando-os com os mesmos componentes criados utilizando a biblioteca nativa do Ginga (módulo Canvas). Já o segundo foi realizado utilizando desenvolvedores voluntários para comparar principalmente a produtividade na criação de uma tela.

Com o resultado da análise dos resultados dos experimentos, devemos ser capazes de responder as seguintes questões de pesquisa:

- **Q1.** *Aplicativos desenvolvidos utilizando o arcabouço proposto são mais eficientes que os desenvolvidos com os módulos do Ginga?*
- **Q2.** *Aplicativos desenvolvidos utilizando o arcabouço proposto utilizam menos memória que os desenvolvidos com os módulos do Ginga?*
- **Q3.** *O uso do arcabouço proposto aumenta a produtividade dos desenvolvedores em relação aos módulos do Ginga?*

Perguntas	Métricas
Qual a eficiência da ferramenta?	Tempo de redesenho de um componente Tempo de carregamento na tela de um componente Tempo de carregamento de uma tela
Quanta memória a ferramenta utiliza?	Memória utilizada por um componente Memória utilizada por uma tela
Qual a produtividade no desenvolvimento utilizando a ferramenta?	Quantidade de linhas de código utilizadas em um componente Quantidade de caracteres utilizados em um componente Quantidade de linhas de código utilizadas em uma tela Quantidade de caracteres utilizados em uma tela Tempo utilizado no desenvolvimento de uma tela

Tabela 4.1: Métricas definidas utilizando o método GQM

Neste capítulo, o primeiro experimento realizado é apresentado na Seção 4.1, mostrando suas métricas, configuração, execução e resultados. Já a Seção 4.2 mostra como se deu a realização do segundo experimento. Na Seção 4.3 os resultados dos dois experimentos realizados são utilizados para responder as três questões de pesquisa definidas. Por último, na Seção 4.4, são apresentadas as limitações dos experimentos realizados e as ameaças que possam invalidar o experimento.

4.1 Avaliação Individual dos Componentes

O primeiro experimento realizado teve o objetivo de avaliar a produtividade e desempenho da criação de componentes individuais e o desempenho de cada um. Foram selecionados componentes presentes na versão atual do LuaSmartGUI para serem desenvolvidos também utilizando o módulo Canvas e comparados com os do arcabouço. A Tabela 4.2 mostra as métricas coletadas no experimento, junto com a questão de pesquisa relacionada.

Devemos descrever cada métrica utilizada e o motivo de ser coletada no experimento:

- **1A**: Para inferir o tempo de resposta de um componente, é medido o tempo gasto para redesenhá-lo;
- **1B**: Como uma tela é construída a partir de vários componentes, o tempo utilizado para

Id	Métrica	Questão
1A	Tempo de redesenho de um componente	Q1
1B	Tempo de carregamento na tela de um componente	Q1
1C	Memória utilizada por um componente	Q2
1D	Quantidade de linhas de código utilizadas em um componente	Q3
1E	Quantidade de caracteres utilizados em um componente	Q3

Tabela 4.2: Métricas definidas para o primeiro experimento

carregar um componente afeta diretamente no tempo para carregar uma tela inteira;

- **1C**: Da mesma forma da métrica 1B, a memória utilizada por um componente afeta diretamente na memória utilizada por uma tela completa;
- **1D**: A quantidade de linhas de código utilizadas mostra a quantidade de operações necessárias para criar um componente, auxiliando na análise de produtividade;
- **1E**: A quantidade de caracteres utilizados no código de um componente mostra o gasto do desenvolvedor para criá-lo, auxiliando também na análise de produtividade.

4.1.1 Unidade Experimental

Para este experimento, foram escolhidos sete componentes disponíveis na versão atual do LuaSmartGUI. Os componentes `Keyboard` e `ListWidget` não foram utilizados devido à complexidade de desenvolvê-los utilizando a biblioteca nativa. Desenvolvedores da equipe de desenvolvimento da Envision no laboratório Embedded levaram mais de um mês para desenvolver um componente desse tipo. O `EditText` também não foi incluído, pois não há sentido seu uso sem o `Keyboard`. Os componentes do tipo `Layout` também foram retirados, pois não existe o conceito de layout no módulo Canvas, não havendo sentido em utilizá-los. Sobraram então os sete componentes presentes na Tabela 4.3[1].

[1]Todos os códigos escritos para o experimento estão disponíveis no Apêndice D.

Widget
Button
CheckBox
Dialog
ImageWidget
Label
SeekBar
TextWidget

Tabela 4.3: Componentes utilizados no experimento

4.1.2 Configuração do Experimento

Foi utilizado um Notebook Dell Vostro, com processador Intel i3 com 2.40GHz, 4GB de memória RAM e Sistema Operacional Linux (distro Ubuntu 13.04). Para a execução dos componentes foi utilizado o emulador de SmartTV provido pela Envision.

Para coletar os dados referentes à eficiência, foi utilizado o módulo `socket` de Lua para instrumentar o código e coletar o tempo atual em milissegundos. Para medir o tempo de carregamento, foi necessário coletar o tempo no início do arquivo e comparar com o tempo no final do mesmo. Para coletar o valor, é necessário iniciar e encerrar o emulador manualmente com o componente analisado. Com 14 componentes a serem analisados, foi decidido coletar 10 valores para cada amostra, fazendo com que o processo manual fosse repetido 140 vezes. Para coletar os dados referentes ao tempo para redesenhar os componentes foram escritos os Códigos D.1 e D.2, que fazem com que os componentes sejam redesenhados e coleta o tempo entre o início e fim do processo. Como a coleta pôde ser automatizada, decidiu-se coletar amostras de tamanho 350, aumentando a confiança nos testes realizados e permitindo ignorar os efeitos da não-normalidade e variância das amostras. O código utilizado redesenha apenas os componentes específicos, fazendo com que os dados coletados sejam independentes da quantidade de componentes em uma tela.

O uso de memória foi coletado a partir do comando `showStatus` do emulador, que informa a memcomponetesória total disponível, a memória utilizada atualmente e a memória livre. Como o uso de memória é um dado que não varia por execução, apenas um valor foi coletado para cada unidade do experimento.

Para coletar a quantidade de linhas de código e de caracteres utilizados, foi criado um script que lê o arquivo e conta as linhas e caracteres. Apenas linhas relacionadas a instruções utilizadas pelo componente foram consideradas válidas, excluindo linhas de comentário, importação de módulos e linhas em branco. Também foram removidas as quebras de linhas. Para a contagem dos caracteres, foram considerados apenas os pertencentes às linhas computadas e desconsiderados os espaços em branco.

4.1.3 Resultados do Experimento

A primeira métrica coletada no experimento foi a Métrica 1A, que indica o tempo necessário para redesenhar um componente. Para coletar os dados, os componentes foram criados e, dentro de iterações, foram redesenhados, coletando o tempo gasto por cada componente. Como a coleta pôde ser realizada de forma automatizada e rápida, decidiu-se utilizar amostras com 350 valores coletados, podendo assim ignorar os efeitos da não-normalidade e variância dos dados, além de aumentar a confiança nos testes. A Tabela 4.4 mostra a média do tempo de redesenho para cada componente[2].

Componente	Canvas (ms)	LuaSmartGUI (ms)	Ganho da solução (%)
Button	1,81	3,20	-76,40
CheckBox	21,81	2,74	87,45
ImageWidget	1,89	2,59	-37,16
Label	21,47	2,93	86,36
SeekBar	2,14	4,71	-119,91
TextWidget	1,03	3,11	-203,01
Média	8,36	3,22	61,53

Tabela 4.4: Tempo médio de redesenho de cada componente

Pode-se visualizar que alguns componentes foram mais rápidos com LuaSmartGUI e outros com a biblioteca nativa. Porém, no geral, os componentes em LuaSmartGUI foram cerca de 61,5% mais eficientes. A Figura 4.1 mostra os gráficos bloxplot para cada componente

[2]Não foi possível coletar os dados para o componente `Dialog`, pois a forma com que o mesmo é redesenhado na tela é diferente dos outros componentes e dificultaria a automação de sua coleta.

avaliado. Nota-se que é necessário comprovar a diferença de tempo entre cada implementação a partir de testes.

Como temos amostras consideradas grandes, de mesmo tamanho e com valores numéricos, foi utilizado o teste T de Student para comparar os resultados. Todos os testes foram executados com o nível de confiança de 99% e todos resultaram em um p-valor mínimo, mostrando que as amostras são realmente diferentes. Portanto, com as diferenças comprovadas por testes, podemos dizer que, apesar de um ganho geral, apenas os componentes `CheckBox` e `Label` foram são mais rápidos quando desenvolvidos com LuaSmartGUI.

Apesar disso, ainda é necessário analisar o motivo de quatro componentes se mostrarem mais lentos. Um fator que pode justificar parte dessa perda é que, durante a limpeza do componente, a biblioteca procura pelo primeiro pai que possui um `canvas` visível e limpa o pedaço da tela com suas propriedades, para então poder desenhar novamente o componente em seu estado atual. Esse algoritmo gera um *overhead* para que o redesenho do componente seja realizado de forma correta. Porém, ainda não justifica toda a perda, sendo necessário analisar futuramente os motivos.

A segunda métrica coletada foi o tempo de carregamento de cada componente, referente à Métrica 1B. A Tabela 4.5 mostra as médias de 10 valores coletados para cada componente.

Componente	**Canvas (ms)**	**LuaSmartGUI (ms)**	**Ganho da solução (%)**
Button	3,11	43,71	-1305,47
CheckBox	29,01	47,44	-63,53
Dialog	94,31	124,00	-31,48
ImageWidget	35,81	73,88	-106,31
Label	31,26	41,84	-33,84
SeekBar	3,90	41,73	-970,00
TextWidget	3,64	42,26	-1060,99
Média	28,72	59,26	-106,35

Tabela 4.5: Tempo médio de carregamento de cada componente

A Figura 4.2 mostra os gráficos boxplot da métrica coletada para cada componente. A partir dos gráficos pode notar-se que não é necessário realizar testes para comprovar que todos os componentes LuaSmartGUI estatisticamente levam mais tempo para ser carregado

do que os componentes desenvolvidos com a biblioteca nativa.

Enquanto a biblioteca LuaSmartGUI precisa configurar o componente, posicioná-lo e desenhá-lo, o componente desenvolvido diretamente com o módulo Canvas é configurado manualmente pelo usuário e posicionado diretamente no local desejado. Dessa forma, era esperado que houvesse perda no tempo de carregamento.

Para coletar o uso de memória, referente à Métrica 1C, foi necessária apenas uma coleta, já que o uso é estático e não varia por execução. A Tabela 4.6 mostra os dados coletados. Nota-se que, apesar de alguns componentes desenvolvidos com LuaSmartGUI utilizarem mais memória, houve um ganho geral de cerca de 62,31%.

Componente	Canvas (Bytes)	LuaSmartGUI (Bytes)	Ganho da solução (%)
Button	12.000	24.000	-100,00
CheckBox	4.232	16.232	-283,55
Dialog	11.059.200	4.620.000	58,22
ImageWidget	1.440.000	0	100,00
Label	0	0	0,00
SeekBar	60.000	60.000	0,00
TextWidget	12.000	24.000	-100,00
Média	1.798.204,5710	677.747,4286	62,31

Tabela 4.6: Uso de memória de cada componente

Apesar de, no geral, os componentes LuaSmartGUI terem utilizado menos memória, alguns componentes utilizaram pelo menos o dobro de memória dos componentes com os módulos do Ginga. É necessário investigar o motivo desses componentes estarem utilizando mais memória. Caso esses componentes sejam muito utilizados em uma tela, podem ultrapassar o limite de memória disponível e inviabilizar o uso da biblioteca.

A Tabela 4.7 mostra os dados coletados referentes à quantidade de linhas de código (Métrica 1D). Os dados mostram que houve um ganho médio de aproximadamente 88% na quantidade de linhas de código por componente.

A Tabela 4.8 mostra os dados coletados para a Métrica 1D, referente à quantidade de caracteres por componente. Pode-se ver que houve um ganho médio de aproximadamente 75,5%.

Componente	Canvas	LuaSmartGUI	Ganho da solução (%)
Button	27	2	92,6
CheckBox	24	2	91,67
Dialog	35	4	88,57
ImageWidget	2	2	0,00
Label	3	2	33,33
SeekBar	32	2	93,75
TextWidget	9	2	77,78
Média	18,86	2,29	87,88

Tabela 4.7: Linhas de código utilizada por componente

Nota-se que o componente `ImageWidget` mostrou-se menos produtivo utilizando LuaSmartGUI. Isto se deve ao fato de que o componente é bastante simples de ser utilizado nas duas bibliotecas. Porém, apesar de o componente desenvolvido utilizando LuaSmartGUI possuir cerca de 34% de caracteres a mais, a diferença absoluta é de apenas 20 caracteres.

Componente	Canvas	LuaSmartGUI	Ganho da solução (%)
Button	674	100	85,16
CheckBox	561	76	86,45
Dialog	896	200	77,68
ImageWidget	59	79	-33,90
Label	104	76	26,92
SeekBar	779	204	73,81
TextWidget	253	80	68,38
Média	475,14	116,43	75,50

Tabela 4.8: Caracteres utilizados por componente

4.2 Avaliação de Telas Utilizando as Ferramentas

O segundo experimento realizado teve como objetivo avaliar a produtividade no desenvolvimento de uma tela e o desempenho (eficiência e uso de memória) da tela desenvolvida. O

experimento consistiu na especificação de uma tela a ser desenvolvida utilizando a solução proposta e a biblioteca nativa, comparando as métricas coletadas. A Tabela 4.9 mostra as métricas definidas para este experimento.

Id	Métrica	Questão
2A	Tempo de carregamento da tela	Q1
2B	Memória utilizada pela tela	Q2
2C	Quantidade de linhas de código utilizadas na tela	Q3
2D	Quantidade de caracteres utilizados na tela	Q3
2E	Tempo gasto no desenvolvimento da tela	Q3

Tabela 4.9: Métricas definidas para o segundo experimento

4.2.1 Unidade experimental

Para este experimento, foi especificada uma tela para ser desenvolvida utilizando LuaSmartGUI e a biblioteca nativa do Ginga. Foram selecionados sete voluntários[3] para participar do experimento. Todos os participantes são formados no curso Bacharelado em Ciência da Computação da Universidade Federal de Campina Grande entre 2011 e 2013, com idades entre 23 e 26 anos. Os voluntários comprovaram experiência em Lua respondendo corretamente um questionário com 4 questões[4]. O questionário pode ser encontrado no Apêndice A. Nenhum dos desenvolvedores conheciam LuaSmartGUI nem o Ginga e só tiveram acesso às suas documentações durante a execução do experimento. Todos os códigos escritos pelos desenvolvedores voluntários podem ser encontrados no Apêndice E.

A Figura 4.3 mostra a tela especificada para ser desenvolvida pelos voluntários. Como o experimento limitou-se a apenas uma tela, ela foi especificada de forma que pudesse utilizar a maioria dos componentes utilizados no experimento anterior.

A Figura 4.4 mostra o esqueleto de uma possível solução para o desenvolvimento da tela, utilizando cinco dos sete componentes analisados no experimento (1 `ImageWidget`, 2 `Label`, 5 `CheckBox`, 2 `Button` e 1 `SeekBar`). Além disso, dois tipos de `Layout` são

[3] A fim de preservar a identidade dos voluntários, eles serão enumerados de 1 a 7.

[4] As perguntas presentes no questionário foram adaptadas do livro *Programming in Lua, Second Edition*.

utilizados `VerticalLayout` e `HorizontalLayout`. Como as Métricas 2A, 2B, 2C e 2D também foram medidas por componente, podemos esperar os seguintes resultados:

- Métrica 2A: 253,5ms com a biblioteca nativa e 523,91ms com LuaSmartGUI (-106,67%);
- Métrica 2B: 1.545.160 Bytes com a biblioteca nativa e 189.160 Bytes com LuaSmartGUI (87,76%);
- Métrica 2C: 214 linhas de código com a biblioteca nativa e 22 com LuaSmartGUI (89,72%);
- Métrica 2D: 5.199 caracteres com a biblioteca nativa e 1.015 caracteres com LuaSmartGUI (80,48%).

4.2.2 Configuração do experimento

O experimento foi executado durante três dias, devido à disponibilidade dos participantes. O horário escolhido para a execução foi entre 19 horas e 23 horas, pois os voluntários estariam disponíveis e diminui o risco de interrupções comuns durante o horário de trabalho. Foram escolhidos dois ambientes para a execução do experimento. O primeiro deles foi uma sala dentro da sede do cghackspace [1]. Já o segundo foi uma das salas do laboratório Embedded. Foram disponibilizados lanches, água, suco e café para todos os participantes. A comunicação entre os voluntários foi restringida durante o desenvolvimento para evitar que a participação de um desenvolvedor seja afetada por outros. Os participantes iniciaram o experimento utilizando LuaSmartGUI e depois os módulos do Ginga.

Os participantes utilizaram uma máquina virtual Ubuntu 13.10, com 2GB de memória RAM e sem acesso à Internet. Os softwares disponíveis para desenvolvimento foram o Eclipse Juno (com o plugin LuaDev), Sublime Text 3 e gedit 3.8.3, além do emulador de TV provido pela Envision. Os documentos disponíveis para os desenvolvedores foram uma apostila com um tutorial básico sobre LuaSmartGUI, uma outra sobre os módulos do Ginga e a especificação da tela a ser desenvolvida.

Durante o experimento, três candidatos desistiram de desenvolver a tela utilizando a biblioteca nativa do Ginga. Dois deles iniciaram o desenvolvimento, mas abandonaram após

cerca de 90 minutos e o terceiro não chegou a iniciá-lo. Os dois primeiros alegaram que estava muito difícil desenvolver a tela utilizando o módulo Canvas. O terceiro alegou indisponibilidade de tempo para continuar contribuindo para o experimento.

A coleta dos dados referentes às Métricas 2A, 2B, 2C e 2D foi realizada exatamente da mesma forma utilizada para as Métricas 1A, 1B, 1D e 1E do primeiro experimento, respectivamente, também utilizando o emulador provido pela Envision. A coleta de tempo gasto por desenvolvedor foi realizada anotando o tempo de início e fim de desenvolvimento de cada voluntário. A tela desenvolvida podia ser julgada como finalizada caso fosse aprovada em um conjunto de testes visuais e manuais realizados pelo autor.

4.2.3 Resultados do Experimento

Com a desistência de três participantes, o experimento resultou em sete telas desenvolvidas com LuaSmartGUI e quatro telas com a biblioteca nativa do Ginga.

A Métrica 2A, referente ao tempo de carregamento de cada tela, foi a primeira a ser coletada. A Tabela 4.10 mostra que todas as telas desenvolvidas com LuaSmartGUI utilizaram mais tempo para serem carregadas. Em geral, elas foram aproximadamente 110,52% mais lentas, se aproximando do resultado esperado através do resultado esperado (106,67%). A partir do boxplot da Figura 4.5 pode ser visto que não é necessária a execução de testes para comprovar que as médias são estatisticamente diferentes.

Apesar desta análise mostrar perda de eficiência, o resultado já era esperado. A biblioteca LuaSmartGUI leva mais tempo para configurar os componentes, posicioná-los corretamente e desenhá-los, enquanto os componentes criados com LuaSmartGUI são configurados e posicionados diretamente pelo usuário.

Assim como a Métrica 1B, a Métrica 2B, referente ao uso de memória de cada tela, é um dado estático, que não varia por execução. Dessa forma, foi necessário coletar apenas uma amostra para cada tela desenvolvida. A Tabela 4.11 mostra os dados coletados.

Nota-se que todas as telas desenvolvidas com LuaSmartGUI utilizaram mais memória. Era esperado que as telas utilizassem memória equivalente, mas em geral, houve uma piora de aproximadamente 867,5%, distanciando-se totalmente do resultado esperado (87,76% de ganho). Um dos motivos desse resultado é que na biblioteca cada componente possui um `canvas` próprio, inclusive o *layout* raiz da tela. Dessa forma, a tela já inicia com um uso

Participante	Canvas (ms)	LuaSmartGUI (ms)	Ganho da solução (%)
1	55,54	118,00	-112,46
2	49,24	111.90	-127,25
3	47,40	116.70	-146,20
4	65,32	105.60	-61,67
5		114.10	
6		109.50	
7		125.50	
Média	54,375	114,471	-110,52

Tabela 4.10: Tempo de carregamento de cada tela, em milissegundos

Participante	Canvas	LuaSmartGUI	Ganho da solução (%)
1	100.232	4.027.560	-3.918,24
2	271.180	4.198.200	-1.448,12
3	1.439.432	4.585.160	-218,54
4	2.116	4.225.160	-199.576,75
5		4.038.360	
6		4.107.560	
7		5.513.560	
Média	453.240	4.385.080	-867,50

Tabela 4.11: Memória utilizada por cada tela, em Bytes

elevado de memória.

Também pode ser visto uma perda de 199.576,75% entre as telas desenvolvidas pelo participante 4. Apesar da diferença, o uso de memória da tela utilizando LuaSmartGUI (4.225.160 Bytes) está abaixo da média de uso das sete telas (4.385.080 Bytes). Além disso, o participante utilizou apenas um canvas para desenhar todos os componentes, como pode ser visto no Código E.11. Essa estratégia pode afetar na produtividade, como pode ser visto nas próximas métricas. O mesmo participante obteve os maiores valores em caracteres e tempo utilizados no desenvolvimento da tela, utilizando os módulos do Ginga.

A Métrica 2C se refere à quantidade de linhas de código presente em uma tela. Os dados

coletados refletem diretamente na análise da produtividade, pois mostra aproximandamente quantas instruções são necessárias criar para desenvolver uma tela. A Tabela 4.12 mostra que todas as telas desenvolvidas com LuaSmartGUI utilizaram menos linhas de código. No geral, houve uma redução de aproximadamente 70,3%, apesar de ter sido um pouco abaixo do esperado (89,72%).

Participante	Canvas	LuaSmartGUI	Ganho da solução (%)
1	125	43	65,60
2	189	43	77,25
3	109	53	51,38
4	171	43	74,85
5		49	
6		38	
7		40	
Média	148,500	44,143	70,27

Tabela 4.12: Quantidade de linhas de código em cada tela

A Métrica 2D, assim como a 1D, mostra o esforço do desenvolvedor a partir da quantidade de caracteres utilizada. a Tabela 4.13 mostra quantos caracteres foram utilizados em cada tela pelos voluntários do experimento. Apenas o participante 3 conseguiu desenvolver a tela utilizando a biblioteca nativa com menos caracteres. Era esperado um ganho de 80,48% nesta métrica. Porém, o ganho observado foi de aproximadamente 29%.

A última e mais significativa métrica coletada foi a 2E, que informa o tempo de desenvolvimento de cada tela por participante. A partir dos dados apresentados na Tabela 4.14 pode ser visto que todos os desenvolvedores levaram menos tempo desenvolvendo as telas com LuaSmartGUI do que com a biblioteca nativa do Ginga. No geral, houve um ganho de aproximadamente 31,54%.

4.3 Respostas às Questões de Pesquisa

Foram executados dois experimentos e coletadas métricas para avaliar o ganho de produtividade e a satisfatoriedade da eficiência e uso de memória no desenvolvimento de telas

Participante	Canvas	LuaSmartGUI	Ganho da solução (%)
1	2.859	2.261	20,92
2	4.125	2.523	38,84
3	2.462	2.573	-4,51
4	3.630	2.234	38,46
5		2.788	
6		1.917	
7		1.961	
Média	3.269,000	2.322,429	28,96

Tabela 4.13: Quantidade de caracteres em cada tela

Participante	Canvas (m)	LuaSmartGUI (m)	Ganho da solução (%)
1	143	77	46,15
2	162	125	22,84
3	155	95	38,71
4	171	110	35,67
5		121	
6		110	
7		97	
Média	157,75	108	31,54

Tabela 4.14: Tempo de desenvolvimento de cada tela, em minutos

utilizando a solução proposta. Os experimentos mostraram ganhos em alguns quesitos e perda em outros. Tais fatores devem ser analisados para poder responder corretamente as 3 questões de pesquisa definidas.

Aplicativos desenvolvidos utilizando o arcabouço proposto são mais eficientes que os desenvolvidos com os módulos do Ginga?

Para responder esta questão, é necessário analisar as métricas relacionadas a este quesito (1A, 1B e 2A). Apesar de as métricas 1B e 2A mostrarem que o carregamento das telas utilizando LuaSmartGUI são mais lentas, foi visto que o tempo levado para redesenhar os componentes é menor utilizando a solução apresentada. Componentes são redesenhados na

tela com maior frequência do que uma tela é carregada. Além disso, a diferença do valor absoluto no tempo de carregamento das telas foi de aproximadamente 50ms, sendo quase imperceptível ao olho humano [36].

Sobre os resultados da Métrica 1A, 4 de 6 componentes em LuaSmartGUI se mostraram mais lentos. Apesar de a maior diferença ser de quase 120% de perda, ela corresponde apenas a aproximadamente 2,5ms.

Portanto, de acordo com os experimentos realizados, a resposta para a primeira questão de pesquisa é afirmativa, pois, apesar das perdas encontradas no experimento, seus valores são suficientemente pequenos para serem ignorados pelo usuário.

Aplicativos desenvolvidos utilizando o arcabouço proposto utilizam menos memória que os desenvolvidos com os módulos do Ginga?

As Métricas 1C e 2B são necessárias para responder esta pergunta. Na análise individual, os componentes mostraram melhoria no uso de memória. Porém, quando utilizados em conjunto em uma tela mais complexa, houve claro aumento no uso de memória.

Apesar do esforço realizado pelo arcabouço de sempre que possível limpar a memória não utilizada, não foi suficiente para tornar o uso de memória baixo. Dessa forma, a resposta para a segunda pergunta é negativa, pois houve aumento considerável no uso de memória.

Este resultado torna-se preocupante, pois pode ser um empecilho para o usuário utilizar a solução proposta. Uma avaliação melhor pode ser realizada, utilizando mais componentes e analisando o crescimento do consumo de memória ao longo do tempo que uma tela é utilizada, além de seguir melhores diretrizes utilizando a biblioteca. Porém, é fato que a biblioteca necessita ser melhorada neste quesito.

O uso do arcabouço proposto aumenta a produtividade dos desenvolvedores em relação aos módulos do Ginga?

As Métricas 1D, 1E, 2C e 2D mostram um ganho na produtividade a partir da medição do esforço necessário para criar telas e componentes com LuaSmartGUI. A Métrica 2E mostra o ganho de produtividade a partir do tempo gasto no desenvolvimento de uma tela.

Todos os resultados relacionados às métricas de produtividade foram positivos. Portanto, pode-se dizer que, a partir das métricas analisadas, a resposta para a terceira questão de pesquisa é afirmativa, pois o experimento mostrou ganho de produtividade em todos os quesitos avaliados.

4.4 Limitações e Ameaças à Validade

Os experimentos realizados possuem algumas limitações que podem afetar os resultados obtidos.

Primeiramente, os componentes `Keyboard`, `ListWidget` e `EditText` não participaram dos experimentos. A participação dos 3 poderia ter afetado os resultados, pois é esperado que o desenvolvimento destes componentes utilizando o Ginga seja muito mais lento do que utilizando LuaSmartGUI.

A quantidade de desenvolvedores utilizada no experimento não foi significativa o bastante para dar total confiança nos resultados. Por se tratar de participantes voluntários, torna-se difícil o recrutamento de desenvolvedores que tenham disponibilidade e interesse em utilizar uma ou duas noites participando do experimento.

Um experimento longo com voluntários corre a ameaça de desistência de algum deles. No experimento em questão, três dos sete participantes desistiram durante o experimento. Por um lado, como dois deles desistiram por dificuldades com a biblioteca nativa, este fato mostra que a solução apresentada foi melhor aceita do que o a biblioteca nativa. Por outro lado, como a quantidade de participantes já era pequena, reduziu ainda mais o tamanho das amostras coletadas.

O planejamento inicial do experimento consistia no desenvolvimento de seis telas diferentes. Porém, devido à limitação de tempo e de participantes, o escopo foi reduzido para apenas uma tela. A realização do experimento com outras telas pode trazer resultados diferentes, pois fatores como complexidade e quantidade de componentes podem afetar as métricas de uso de memória e produtividade.

A coleta dos dados foi realizada utilizando o emulador provido pela Envision. Caso fossem coletados em uma TV, possivelmente os valores relacionados à eficiência seriam maiores, aumentando a diferença entre os resultados com LuaSmartGUI e os módulos do Ginga, podendo invalidar os resultados obtidos para a primeira questão de pesquisa.

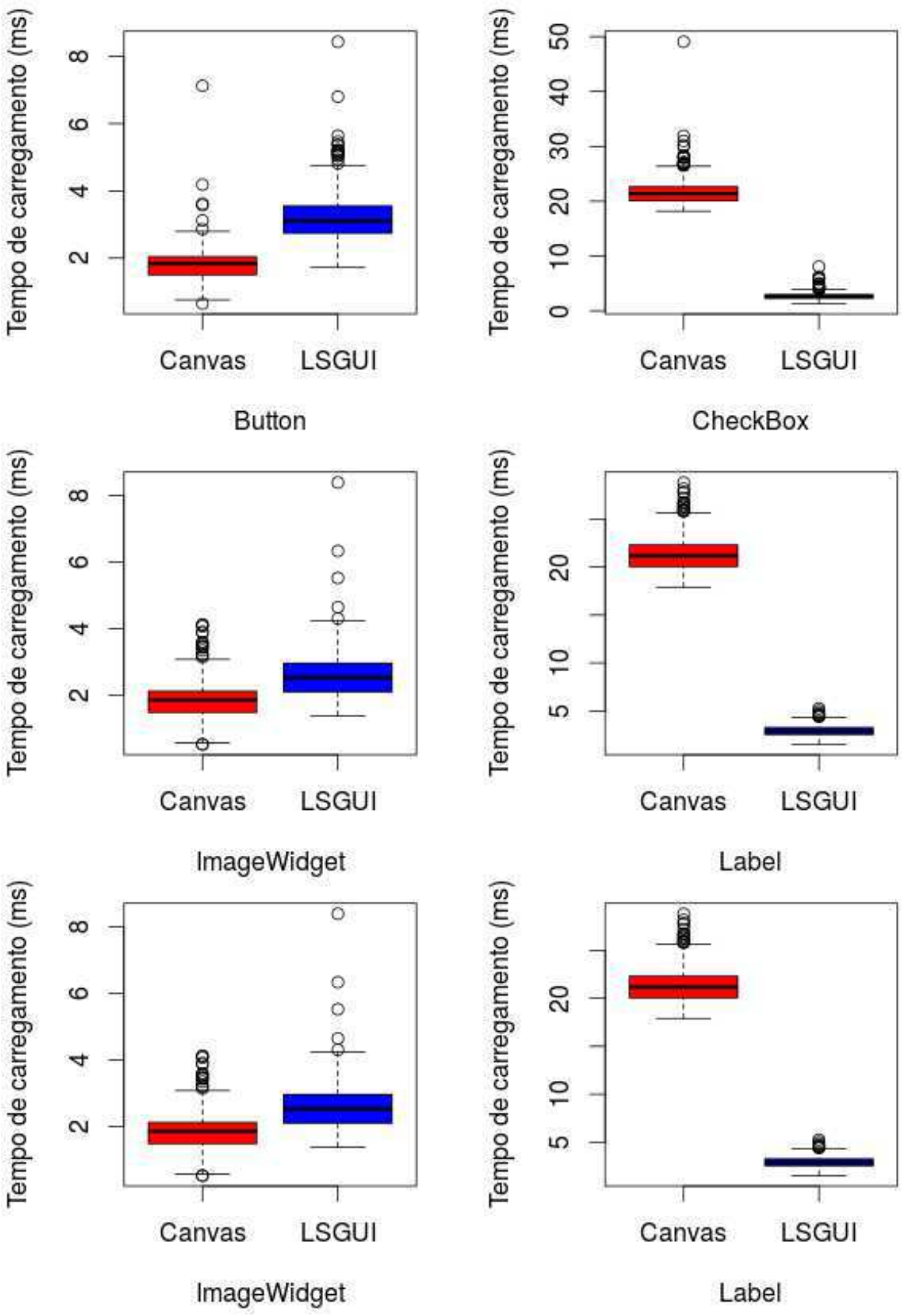

Figura 4.1: Boxplot do tempo de redesenho de cada componente

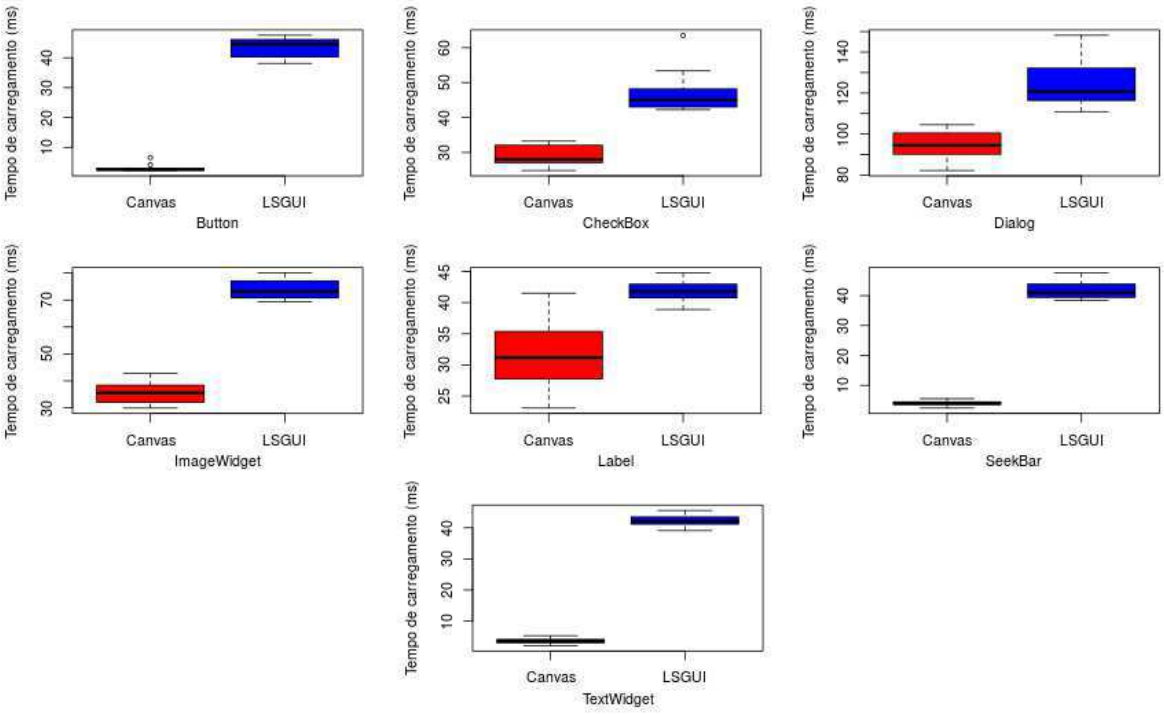

Figura 4.2: Boxplot do tempo de carregamento de cada componente

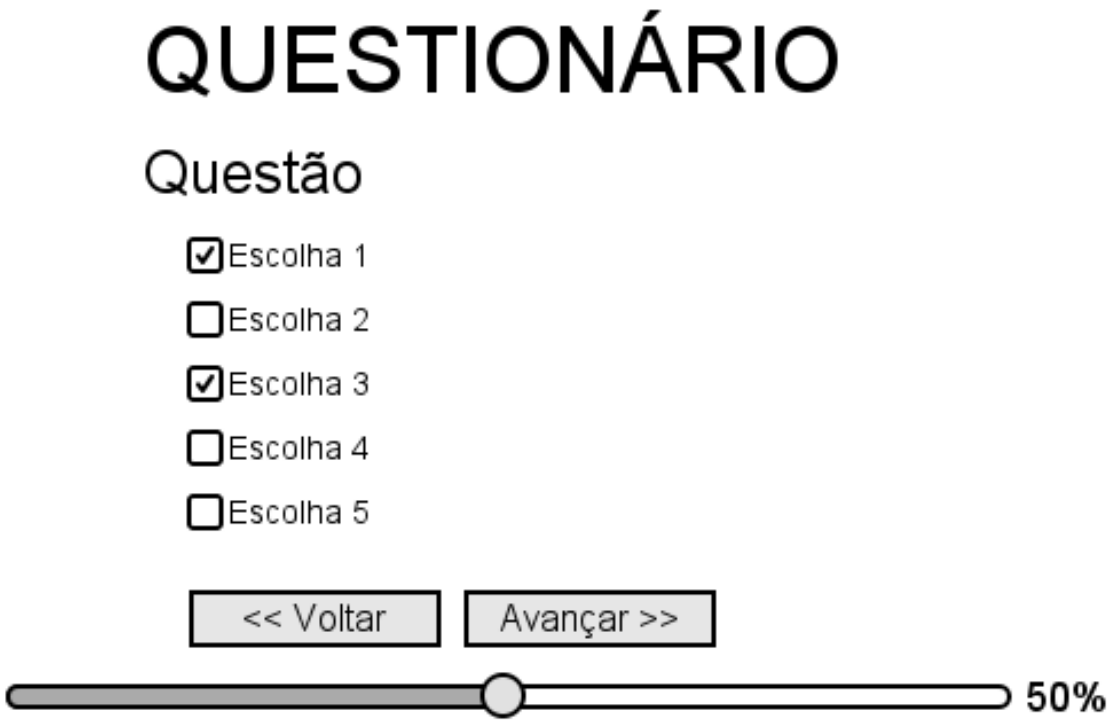

Figura 4.3: Tela expecificada para o experimento

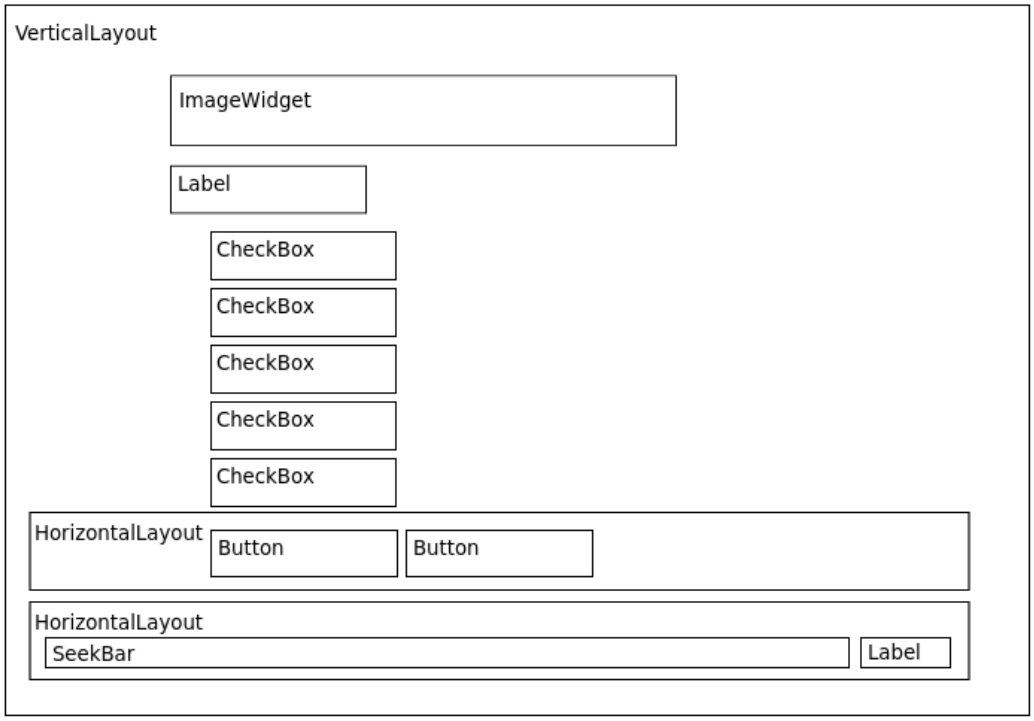

Figura 4.4: Uma possível forma de desenvolver a tela especificada para o experimento

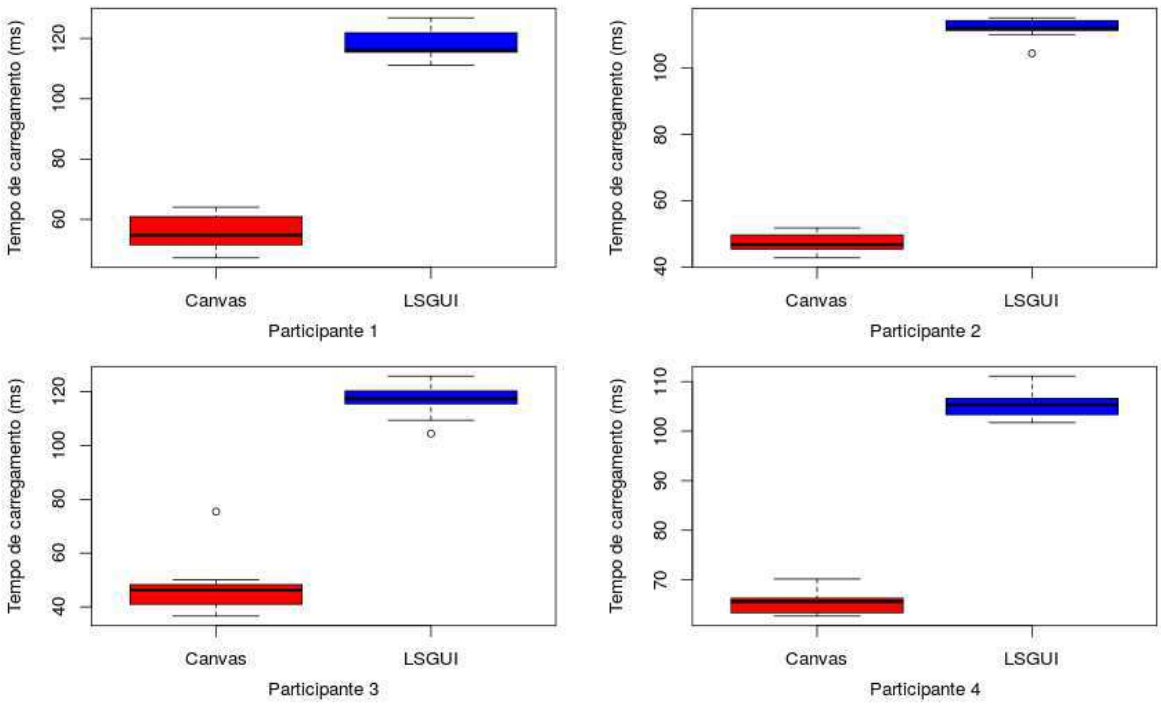

Figura 4.5: Boxplot do tempo de carregamento de cada tela

Capítulo 5

Trabalhos Relacionados

Foi realizada uma revisão sistemática para levantar trabalhos relacionados ao arcabouço proposto. A questão da pesquisa utilizada foi "Quais são as ferramentas disponíveis para o desenvolvimento de aplicativos para TVs interativas?". Inicialmente, a pesquisa retornou 386 resultados. Porém, após filtrar pelo título, resumo e conteúdo, apenas resultados a seguir foram aproveitados. Este fato mostra que ainda é necessário muito trabalho na área.

O motor de interface de usuário em Lua criado para plataformas de TV [9]. Também baseia-se em componentes e foi desenvolvida em Lua. Porém, sua arquitetura está atrelada à uma implementação em C++ e realização de bindings entre Lua e C++, através do módulo toLua [13]. A Figura 5 mostra a arquitetura definida.

A ferramenta NEXT [33] permite a montagem de telas em NCL através de uma ferramenta gráfica, sem necessidade de conhecimento específico sobre NCL. Porém, seu uso limita-se apenas à NCL e dá suporte à adição de código em Lua. A criação de uma ferramenta gráfica é um incentivo para os trabalhos futuros da solução deste trabalho.

O Ginga Game [6] é um framework baseado em componentes para o desenvolvimento de aplicativos para TV Digital com Ginga. Porém, sua modelagem é voltada para a criação de jogos, utilizando a linguagem de programação Java.

Foi proposto um *framework* aberto para TV interativa utilizando Java AWT [14]. Também provê uma biblioteca de componentes. Porém, não utiliza o conceito de layouts utilizado em LuaSmartGUI, podendo dificultar o posicionamento dos componentes. Além disso, não foi realizada uma avaliação no ganho de produtividade no uso da ferramenta.

PyGame [2] é um conjunto de bibliotecas em Python para o desenvolvimento de jo-

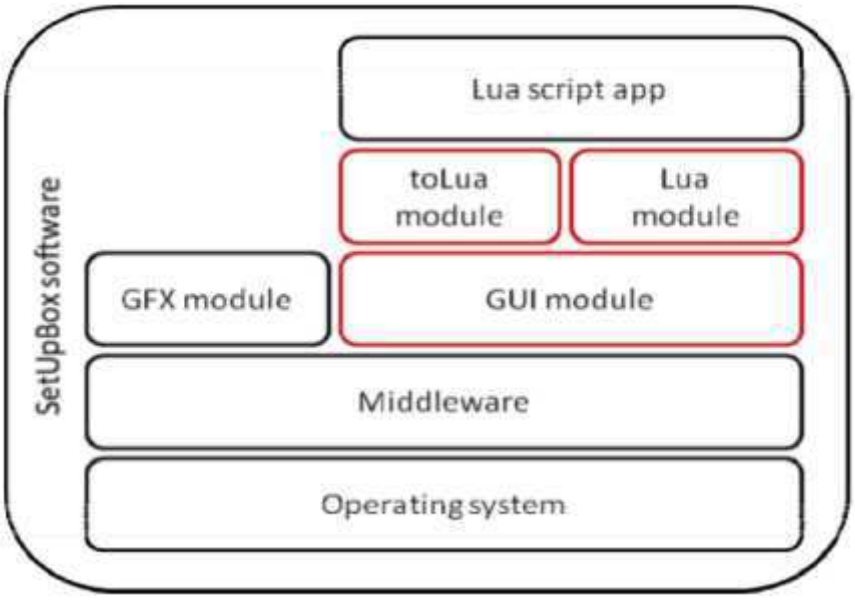

Figura 5.1: Arquitetura do motor de interface de usuário em Lua [9]

gos. Qualquer plataforma que possua Python instalado permite a execução de programas desenvolvidas com a ferramenta. Porém, por ser focada no desenvolvimento de jogos, não disponibiliza componentes como em LuaSmartGUI, limitando seu uso em aplicações para TVs.

Enlightenment [19] é um conjunto de bibliotecas para a criação de interface de usuário, bastante utilizada em dispositivos móveis. Assim como LuaSmartGUI, provê vários componentes que podem ser reutilizados e personalizados pelo desenvolvedor. Porém, utiliza a linguagem de programação C para desenvolvimento, podendo se tornar uma dificuldade para os usuários em relação à uma linguagem interpretada, como Lua ou Python.

Enyo [20] é uma biblioteca em JavaScript para a criação de aplicativos em HTML5. Atualmente é utilizada no Web SDK, da LG. Porém, ainda não são todos os navegadores que dão suporte a HTML5, além de seu uso limitar os aplicativos apenas a elementos de interface de usuário, pois não possui acesso a outros recursos do dispositivo. Em pesquisa realizada pela *Developer Economics* [17], dos desenvolvedores que abandonaram o uso de HTML5, a maioria alegaram problemas de desempenho e limitações de acesso às APIs do sistema.

Capítulo 6

Conclusões

Neste trabalho foi proposta e apresentada uma abordagem para o aumento da produtividade dos desenvolvedores de aplicativos com interface de usuário para SmartTV. A solução resume-se a um arcabouço para o desenvolvimento de interface de usuário em aplicações para SmartTVs, com uma implementação em Lua chamada de LuaSmartGUI. Foi visto que ainda é necessário um grande esforço nesta área, apesar de existir algumas soluções disponíveis que facilitam a criação de telas de aplicativos para TVs Interativas.

A solução proposta foi avaliada através da realização de dois experimentos, avaliando métricas de eficiência, consumo de memória e produtividade. O primeiro experimento avaliou sete componentes providos pelo arcabouço, comparando com uma implementação equivalente utilizando o módulo Canvas do Ginga. O segundo experimento consistiu na coleta de métricas em telas desenvolvidas por sete desenvolvedores voluntários. Apesar da desistência de três participantes, foram obtidas sete telas utilizando LuaSmartGUI e quatro telas utilizando o módulo Canvas. A indisponibilidade de tempo dos participantes fez com que a unidade experimental do experimento fosse reduzida de cinco para apenas uma tela a ser desenvolvida, reduzindo sua expressividade e confiança, tornando uma possível ameaça à validade. Porém, os dados coletados já mostraram resultados significativos, tanto a favor quanto contra a ferramenta.

Nos quesito eficiência, LuaSmartGUI mostrou um ganho de aproximadamente 61%, considerando o tempo de resposta, apesar de possuir um tempo de carregamento cerca de 110% mais lento. Como uma tela é carregada com menos frequência que um componente na tela é redesenhado, o resultado continuou sendo positivo.

Já em relação ao uso de memória, houve um aumento no uso de aproximadamente 867%, bem maior que o esperado. Este resultado foi bastante preocupante, pois pode inviabilizar o uso da ferramenta. Porém, torna-se necessário realizar outra avaliação coletando as métricas de uso de memória em várias telas, variando seus componentes e a quantidade de cada um deles, além da variação no uso de memória ao longo do tempo. Dessa forma, há a possibilidade de obter melhores resultados. Porém, não implica no fato da necessidade de melhora no gerenciamento de memória pela biblioteca.

Sobre a produtividade, principal meta da solução proposta, a avaliação mostrou ganho em todas as métricas, principalmente na medição do tempo de desenvolvimento de uma tela, apresentando um ganho de cerca de 31% no tempo gasto no desenvolvimento de uma tela. As métricas de linhas de código e quantidade de caracteres também mostraram ganho quando utilizada LuaSmartGUI (cerca de 70% e 29%, aproximadamente). Portanto, apesar de não ter atingido o objetivo secundário de manter o uso de memória satisfatório, o objetivo principal de aumentar a produtividade foi alcançado.

Porém, ainda é necessário realizar melhorias futuras neste trabalho. Apesar dos resultados obtidos no experimento, a avaliação realizada foi limitada devido à quantidade de voluntários e tempo disponível para suas participações. Torna-se necessário a realização de um novo experimento, utilizando novos desenvolvedores e aumentando a unidade experimental. Dessa forma, podemos avaliar outros aspectos que afetam diretamente a produtividade.

Para proporcionar aumentar o ganho na produtividade, é necessária a disponibilização de ferramentas auxiliares para o desenvolvimento. O uso do arcabouço em si já facilita a criação de elementos e organização da tela. Porém, uma ferramenta gráfica reduziria o tempo necessário para projetar telas, gerando códigos automaticamente e possibilitando a pré-visualização da tela. A ideia inicial é desenvolver um *plugin* para a IDE Eclipse [43].

A ferramenta Qt possui uma linguagem própria chamada QML [35], que permite a estruturação dos componentes de forma declarativa. Já Android utiliza arquivos XML, com o mesmo objetivo. Dessa forma, LuaSmartGUI também deve possuir uma forma de criar os componentes de forma declarativa, a fim de aumentar ainda mais a produtividade dos desenvolvedores.

A documentação disponível para o arcabouço proposto ainda é limitada. Ela carece de maior detalhamento sobre a arquitetura da solução e melhores exemplos sobre como utili-

zar e expandir seus componentes. Torna-se então necessário o trabalho de evoluir toda a documentação disponível para facilitar o aprendizado da ferramenta.

Por último, é necessário resolver o problema do uso de memória excessivo identificado no experimento, tornando a ferramenta viável para uso dos desenvolvedores. Espera-se que a ferramenta possa ser adotada pela Envision e utilizada no desenvolvimento de seus aplicativos.

Referências Bibliográficas

[1] cghackspace. Disponível em: http://www.cghackspace.org. Acesso em: 28 out. 2014.

[2] PyGame. Disponível em: http://www.pygame.org/wiki/about. Acesso em: 28 out. 2014.

[3] NBR ABNT. 15606-2: 2007. *Televisão digital terrestre-Codificação de dados e especificações de transmissão para radiodifusão digital–Parte*, 2.

[4] Marcelo S. Alencar. Fundamentals of digital television. In *Digital Television Systems*, pages 1–24. Cambridge University Press, 2009. Cambridge Books Online.

[5] Davor Babić. Automated testing of graphical user interfaces. 2013.

[6] Diego Cordeiro Barboza and E Clua. Ginga game: A framework for game development for the interactive digital television. In *Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on*, pages 162–167. IEEE, 2009.

[7] Victor R Basili, Richard W Selby, and David H Hutchens. Experimentation in software engineering. *Software Engineering, IEEE Transactions on*, (7):733–743, 1986.

[8] Milan Z Bjelica, N Teslic, Istvan Papp, and M Savic. A characterization to evaluate graphical user interface frameworks for television receivers. In *Telecommunication in Modern Satellite, Cable, and Broadcasting Services, 2009. TELSIKS'09. 9th International Conference on*, pages 285–288. IEEE, 2009.

[9] Milivoj Bozic, Dusan Zivkov, Istvan Pap, and Goran Miljkovic. Scriptable graphical user interface engine for embedded platforms. In *Telecommunications Forum (TELFOR), 2013 21st*, pages 971–974. IEEE, 2013.

[10] Christian Puhlmann Brackmann. Sistema brasileiro de tv digital. *UNIVERSIDADE CATÓLICA DE PELOTAS, Pelotas*, 2008.

[11] BRASIL. Ministério da Ciência, Tecnologia e Inovação. PORTARIA INTERMINISTERIAL Nº 140, DE 23 DE FEVEREIRO DE 2012. Estabelece o cronograma para introdução de capacidade de aplicações interativas radiodifundidas, de acordo com as Normas ABNT NBR 15606-1, 15606-2, 15606-3, 15606-4 e 15606-6 na fabricação de TELEVISORES COM TELA DE CRISTAL LÍQUIDO. Diário Oficial da União. Brasília, 24 fev. 2012. sec. 1. pt. 1. 2012.

[12] Ed Burnette. *Hello, Android: introducing Google's mobile development platform*. Pragmatic Bookshelf, 2009.

[13] Waldemar Celes. toLua - accessing C/C++ code from Lua. Disponível em: http://webserver2.tecgraf.puc-rio.br/~celes/tolua. Acesso em: 28 out. 2014.

[14] P. Cesar, J. Vierinen, and P. Vuorimaa. Open graphical framework for interactive tv. *Multimedia Tools and Applications*, 30(2):189–203, 2006.

[15] Tomislav Ćurin, Hrvoje Bašić, and Mario Žagar. Framework for graphical user interface personalization in interactive television. In *27th International Conference on Information Technology Interfaces*, 2005.

[16] Deezer. Aparelhos. Disponível em: http://www.deezer.com/devices/tv. Acesso em: 15 dez. 2014.

[17] Developer Economics (2013). How can HTML5 compete with Native? Disponível em: http://www.developereconomics.com/reports/can-html5-compete-native. Acesso em: 28 out. 2014.

[18] Embedded. Parceiros - Embedded Lab. Disponível em: http://www.embeddedlab.org/parceiros. Acesso em: 07 dez. 2014.

[19] Enlightenment.org. Enlightenment - Beauty at your fingertips. Disponível em: https://www.enlightenment.org. Acesso em: 28 out. 2014.

[20] Enyo. Enyo JavaScript Application Framework. Disponível em: http://enyojs.com. Acesso em: 28 out. 2014.

[21] ESPN EMEA Ltd. ESPN Player. Disponível em: http://www.espnplayer.com/devices/. Acesso em: 15 dez. 2014.

[22] Stephen Ferg. *Thinking in Tkinter*. Ferg Organization, 2005.

[23] Google. Diffierent YouTube app versions on TV. Disponível em: https://support.google.com/youtube/answer/3153576?hl=en&ref_topic=1732965. Acesso em: 15 dez. 2014.

[24] Google. Google TV. Disponível em: http://www.google.com/tv/index.html. Acesso em: 28 out. 2014.

[25] Google. [GWT] Getting Started. Disponível em: http://www.gwtproject.org/doc/latest/FAQ_GettingStarted.html#Browsers_and_Servers. Acesso em: 28 out. 2014.

[26] Roberto Ierusalimschy. *Programming in Lua, Second Edition*. Lua.Org, 2006.

[27] Stephen Kurlow. A new object-oriented technique for building a dynamic graphical user interface. In *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*, page 40. IBM Press, 1995.

[28] LG ELECTRONICS. LG Developer | Web SDK. Disponível em: http://developer.lge.com/webOSTV/sdk/web-sdk. Acesso em: 28 out. 2014.

[29] Daniel Molkentin. *The Book of Qt 4: The Art of Building Qt Applications*. No Starch Press, San Francisco, CA, USA, 2007.

[30] Brad Myers, Scott E Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):3–28, 2000.

[31] Netflix, Inc. Netflix - Assista a séries online, Assista a filmes online. Disponível em: https://www.netflix.com/Watch. Acesso em: 15 dez. 2014.

[32] O'Reilly & Associates, Inc. Java AWT Reference. Disponível em: http://docs.oracle.com/javase/6/docs/technotes/guides/awt. Acesso em: 28 out. 2014.

[33] Douglas Paulo de Mattos, Júlia Varanda da Silva, and Débora Christina Muchaluat-Saade. Next: graphical editor for authoring ncl documents supporting composite templates. In *Proceedings of the 11th european conference on Interactive TV and video*, pages 89–98. ACM, 2013.

[34] Qt Project Hosting. Qt Project. Disponível em: http://qt-project.org. Acesso em: 28 out. 2014.

[35] Qt Project Hosting. QtQML. Disponível em: https://qt-project.org/doc/qt-5-snapshot/qtqml-index.html. Acesso em: 15 dez. 2014.

[36] Keith Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological bulletin*, 124(3):373, 1998.

[37] Michael Robin and Michel Poulin. *Digital Television Fundamentals: design and installation of video and audio systems*. McGraw-Hill, 2000.

[38] David Rosenthal. A simple x11 client program-or-how hard can it really be to write"hello, world"? In *USENIX Winter*, pages 229–242, 1988.

[39] Samsung Electronics Co., Ltd. Technical Documentation - Samsung Smart TV Apps Developer Forum. Disponível em: http://www.samsungdforum.com/Guide. Acesso em: 28 out. 2014.

[40] Skype e/ou Microsoft. Baixe o Skype na TV. Disponível em: https://www.skype.com/pt-br/download-skype/skype-for-tv/. Acesso em: 15 dez. 2014.

[41] Bram Smeets, Uri Boness, and Roald Bankras. *Beginning Google Web Toolkit*. Springer, 2008.

[42] L.F.G.S. Soares and S.D.J. Barbosa. *Programando em NCL 3.0: desenvolvimento de aplicações para middleware Ginga: TV digital e Web*. Elsevier, 2009.

[43] The Eclipse Foundation. Eclipse Luna. Disponível em: https://www.eclipse.org. Acesso em: 28 out. 2014.

[44] Jianyang Zhou. Introduction to the constraint language NCL. *The Journal of Logic Programming*, 45(1–3):71 – 103, 2000.

Apêndice A

Exercício para comprovar conhecimento em Lua

Resolva as seguintes questões utilizando a linguagem de programação Lua:

1. *Write a function that reads a text and prints the "n"frequent words in that text.*

2. *We can represent a polynomial* $A_n * x^n + A_{n-1} * x^{n-1} + ... + A_1 * x^1 + A_0$ *in Lua as a list of its coefficients, such as* $\{A_0, A_1, ..., A_n\}$. *Write a function that receives a polynomial (represented as a table) and a value for* x *and returns the polynomial value.*

3. *Write a function that receives an array and prints all combinations of the elements in the array.*

4. *Implement a class Stack, with methods push, pop, top, and isempty.*

Apêndice B

Tutorial LuaSmartGUI

Tutorial LuaSmartGUI

Danilo Freitas *Rohit Gheyi*

1. Introdução

LuaSmartGUI é uma biblioteca em Lua baseada em componentes para o desenvolvimento de aplicativos com Interface Gráfica de Usuário (GUI) para SmartTV e TV Digital. Seu objetivo é aumentar a produtividade dos desenvolvedores, através de seus componentes pré-definidos e tratamento de alguns eventos. Também é responsável por gerenciar o consumo de memória e garantir um bom tempo de resposta.

2. Componentes

Atualmente a biblioteca possui 15 componentes, criados de acordo com um catálogo realizado em aplicativos já existentes. A Tabela 1 mostra os componentes disponíveis.

ID	Componente	Descrição
Layout		Componente para organizar outros dentro dele
1	GridLayout	Organiza os componentes em forma de grade
2	HorizontalLayout	Organiza os componentes da esquerda para a direita
3	VerticalLayout	Organiza os componentes de cima para baixo
4	RelativeLayout	Organiza os componentes em relação a outros dentro dele
Widget		
5	Button	Botão com texto que pode ser pressionado
6	CheckBox	Caixa de marcação
7	Dialog	Janela no estilo *pop-up*
8	EditText	Campo para inserir texto
9	ImageWidget	Mostra uma imagem na tela
10	Keyboard	Teclado com letras e números para inserção de texto
11	Label	Exibe um texto sem fundo, bordas e interação com o usuário
12	SeekBar	Barra de progresso que pode ser controlada
13	TextWidget	Exibe um texto, pondendo ter fundo, borda e interações com o usuário
ListWidget		Cria uma lista de tamanho variável com componentes similares, podendo ter paginação
14	HorizontalListWidget	Cria uma lista de itens horizontal
15	VerticalListWidget	Cria uma lista de itens vertical

Tabela 1: Componentes disponíveis em LuaSmartGUI

2.1. View

View é o componente pai de todos os Layouts e Widgets. Ele é abstrado, portanto não pode ser utilizado diretamente. Possui os seguintes atributos:

- **width**: tamanho do componente;
- **height**: altura do componente;
- **background_color**: cor de fundo;
- **background_image**: imagem de fundo;
- **visible**: se o componente está visível;
- **focusable**: se pode ganhar o foco;
- **gravity**: posicionamento de seu conteudo;
- **static**: se o componente é estático (ajuda a economizar memória);
- **alpha**: transparência do componente;

- **margin**: espaço entre os componentes. Pode ser decomposto em *margin_top*, *margin_bottom*, *margin_left* e *margin_right*;
- **padding**: espaço entre oo componente e seu conteúdo. Pode ser decomposto em *padding_top*, *padding_bottom*, *padding_left* e *padding_right*;
- **show_margin**: define se deve mostrar suas margens quando possui o foco ou não.

O componente também define algumas funções públicas para auxiliar o desenvolvedor. São eles:

- **View:property(name[, value=nil, redraw=true]**[1]**)**: consulta o valor da propriedade *name*. Caso *value* seja passado, atualiza a propriedade para seu valor. Caro *redraw* seja *true*, redesenha o componente assim que atualizar sua propriedade;
- **View:redraw([flush=true])**: redesenha o componente. Caso *flush* seja *false*, o flush não é realizado;
- **View:setFocusDirection(...)**: Define para qual componente o foco deve ir a partir desse, em alguma direção. Caso passe 2 argumentos, o primeiro será a direção e o segundo o componente. Caso passe 4, serão os 4 componentes na ordem cima, baixo, esquerda e direita.

A View ainda possui os seguintes *listeners*:

- **key_pressed_listener(view, key)**: chamado quando alguma tecla do controle remoto é pressionada;
- **key_released_listener(view, key)**: chamado quando alguma tecla do controle remoto é solta;
- **focus_changed_listener(view, focus)**: chamado quando o componente perde ou ganha o foco.

2.2. Layout

Um Layout é um componente utilizado para organizar outros componentes, chamados de filhos (*children*). Existem quatro tipos de Layout:

- **HorizontalLayout**: organiza seus componentes da esquerda para a direita;
- **VerticalLayout**: organiza seus componentes de cima para baixo;
- **GridLayout**: organiza os componentes em forma de grade;
- **RelativeLayout**: organiza os componentes de acordo com a posição de seus outros componentes.

O Layout não possui atributos extras além dos definidos pela View, mas possui algumas funções públicas:

- **Layout:addChild(child)**: adiciona um componente ao Layout;
- **Layout:addChildren(children)**: adiciona uma lista de componentes ao Layout.

2.3. Widget

Um Widget é um componente que passa alguma informação ao usuário e pode interagir com o mesmo. Pode ser de vários tipos, como Button e TextView. Assim como a View, é abstrato e não pode ser utilizado diretamente. Não possui atributos ou funções públicas além das definidas pela View.

2.4. Button

Button representa um botão na tela, que pode ser pressionado pelo usuário através do botão OK do controle remoto. Possui os seguintes atributos:

- **text**: texto apresentado pelo botão;

[1] Parâmetros entre [] são opcionais

- **text_color**: cor do texto;
- **text_size**: tamanho do text;
- **text_font**: fonte do texto.

Além dos atributos, o Button possui o seguinte *listener*:

- **text_changed_listener(view, text)**: chamado quando o texto é alterado a partir da função *property*;

2.5. CheckBox

CheckBox é uma caixa com texto que pode ser marcada ou desmarcada. Possui os seguintes atributos:

- **text**: texto a ser exibido ao lado da caixa de seleção;
- **text_color**: cor do texto exibido.

Este componente possui as seguintes funções públicas:

- **CheckBox:isChecked()**: retorna se a caixa está marcada ou não;
- **CheckBox:setChecked(checked)** define se a caixa está marcada ou não.

O desenvolvedor ainda pode utilizar o seguinte listener:

- **check_changed_listener(view_check)**: chamado quando a marcação é alterada.

2.6. Dialog

Mostra uma janela no estilo *pop-up*. Possui as seguintes funções públicas:

- **Dialog:setContentView(view)**: define o componente a ser exibido dentro do Dialog;
- **Dialog:show()**: mostra o Dialog;
- **Dialog:close()**: fecha o Dialog.

2.7. EditText

Campo para inserir texto através de um teclado. Possui os seguintes atributos:

- **text**: texto dentro do campo;
- **text_color**: cor do texto;
- **text_hint**: texto de dica exibido quando o campo está vazio;
- **text_hint_color**: cor do texto de dica;
- **keyboard**: componente do tipo Keyboard utilizado como entrada de texto.

Também possui o seguinte listener:

- **text_changed_listener(view, text)**: chamado quando o texto é alterado a partir da função *property*.

2.8. ImageWidget

Componente que mostra uma imagem na tela. Possui um único atributo:

- **image**: imagem a ser exibida.

2.9. Keyboard

Teclado para a inserir caracteres a partir do controle remoto. Deve estar associado a um EditText para ser utilizado. Não possui atributos, funções ou listeners públicos.

2.10. Label

Mostra um texto na tela. Não possui interações com o usuário nem cor de fundo. Possui os seguintes atributos:

- **text**: texto exibido;
- **text_color**: cor do texto;
- **text_size**: tamanho do texto;
- **text_font**: fonte do texto;

2.11. ListWidget

Cria uma lista de componentes semelhantes. É necessário utilizar um *adapter* para automatizar a criação dos widgets. Possui apenas um atributo além dos definidos pela View:

- **divider**: espaço entre dois componentes na lista.

  Possui as seguintes funções públicas:

- **ListWidget:setAdapter(adapter)**: define o *adapter* da lista;
- **ListWidget:setPage(page)**: define a página atual da lista;
- **ListWidget:addView(view)**: adiciona um novo componente à lista.

  O componente também define os seguintes listeners:

- **page_changed_listener(view, page)**: chamado quando a página da lista é alterada;
- **item_pressed_listener(view, item)**: chamado quando o botão OK é pressionado em algum item da lista.

2.12. SeekBar

O SeekBar representa uma barra de progresso horizontal. Possui os seguintes atributos:

- **min**: valor mínimo do progesso;
- **max**: valor máximo do progesso;
- **bar_height**: altura da barra de progresso;
- **cursor_size**: tamanho do cursor;
- **bar_red, bar_green e bar_blue**: cores RGB da barra;
- **cursor_red, cursor_green e cursor_blue**: cores RGB do cursor;
- **step**: variação no movimento do cursor.

  O componente possui as seguintes funções públicas:

- **SeekBar:seek(offset[, callback=true])**: altera o progresso atual. Caso callback seja true, o listener *seek_listeker(view, position)* será chamado após alterar a posição, caso exista.
- **SeekBar:tell()**: informa a posição atual do cursor.

  Também possui um listener:

- **seek_listener(view, position)**: chamado quando a posição do cursor é alterada.

2.13. TextWidget

Exibe um texto na tela. Diferente do Label, pode possuir fundo e interação com o usuário. Possui os seguintes atributos:

- **text**: texto exibido;
- **text_color**: cor do texto;
- **text_size**: tamanho do texto.

Não possui funções públicas específicas, mas possui um listener:

- **text_changed_listener(view, text)**: chamado quando o texto é alterado.

3. Exemplos

Para entender melhor a biblioteca, alguns exemplos são necessários para mostrar como utilizar os componentes. Para utilizá-los, é necessário importar o pacote ĭuasmartgui.lsgui̇. Assim, qualquer componente estará disponível a partir da tabela ĭsgui̇. O código 1 mostra como importar alguns componentes.

Cdigo 1: Importando componentes LuaSmartGUI

```
require "luasmartgui.lsgui"

Window = lsgui.Window
VerticalLayout = lsgui.VerticalLayout
TextWidget = lsgui.TextWidget
Button = lsgui.Button
Style = lsgui.Style
```

3.1. Exemplo 1

O Código 2 mostra um exemplo utilizando os componentes Button, TextWidget e VerticalLayout. Além disso, mostra como utilizar o Style para personalizar componentes, economizando memória e evitando repetição de código.

Cdigo 2: Exemplo utilizando Button, TextWidget, VerticalLayout e Style

```
local w = Window:new{}
local root = VerticalLayout:new{background_image="bg.jpg", static=true,
    gravity=VerticalLayout.Gravity.CENTER_HORIZONTAL}
w:configureLayout(root)

local text_style = Style:create{focused_margin="text_selected.png", background_image="text_bg.png",
    margin=1, padding_left=5}
local button_style = Style:create{focused_image="button_selected.png",
    background_image="button_bg.png", pressed_image="button_pressed.png", margin=1, padding_left=5}

local text_widget1 = TextWidget:new{text="TextWidget1", style=text_style}
local text_widget2 = TextWidget:new{text="TextWidget2", style=text_style}
local button1 = Button:new{text="Button1", style=button_style}
local button2 = Button:new{text="Button2", style=button_style}

root:addChildren({text_widget1, text_widget2, button1, button2})

function handler(evt)
    w:changeFocus(evt)
end

event.register(handler)

w:onDraw()
```

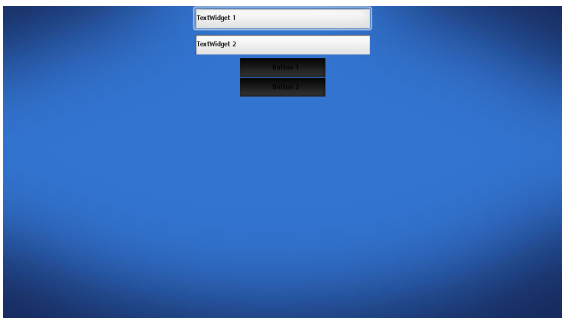

Figura 1: Tela criada a partir do Código 2

A linha 1 instancia um novo Window, que representa a tela e é necessário em todo aplicativo. As linhas 2 e 3 mostram a criação do Layout que será utilizado como raiz, ou seja, o layout pai de todos os outros componentes. Os parâmetros utilizados definem, respectivamente, a imagem de fundo, se o componente é estático e o posicionamento de seus filhos. A linha 4 define o layout criado como raiz.

Nas linhas 6, 7 e 8 são criados 2 estilos. Seu uso é importante quando se tem vários componentes com a mesma aparência. Assim, pode-se customizá-los uma única vez, evitando repetição e código e reduzindo o consumo de memória. Pode-se utilizar os seguintes parâmetros:

- **background_image**: imagem de fundo utilizada;
- **pressed_image**: imagem utilizada quando o componente está pressionado;
- **focused_image**: imagem utilizada quando o componente possui o foco;
- **focused_margin**: imagem de margem utilizada quando o componente possui o foco;
- **focus_margin**: tamanho da margem entre o *focused_margin* e o componente;
- **margin**: atributo *margin* do componente;
- **padding**: atributo *padding* do componente.

Nas linhas 10 a 13 são criados 2 TextWidgets e 2 Buttons. Os parâmetros definem, respectivamente, o texto apresentado pelo componente e o estilo a ser utilizado. A linha 16 adiciona os 4 componentes criados ao layout raiz. As linhas 18 a 22 são necessárias para que o componente Window gerencie os eventos do controle remoto. Já a linha 24 faz com que a tela seja desenhada, finalizando nosso código.

3.2. Exemplo 2

O próximo exemplo é mais complexo, então será realizado passo a passo. A Figura 2 mostra a tela final que iremos desenvolver.

Figura 2: Tela a ser desenvolvida para o Exemplo 2

Primeiro, começaremos pela base da tela. Iremos criar o Layout raíz e o rodapé. Para isso, utilizaremos um RelativeLayout (raiz) e um HorizontalLayout (rodapé). Para o layout raiz, não é necessário informar seu tamanho. Já para o rodapé, é necessário apenas a altura. Como os dois não sofrerão alteração ao longo do tempo, passamos `static=true`. O `alpha=255` informa que a opacidade do Layout é de 100%. Já o `gravity=HorizontalLayout.Gravity.CENTER_VERTICAL` define que os componentes do rodapé estarão centralizados verticalmente. O Código 3 mostra a criação dos layouts, com resultado apresentado pela Figura 3.

Cdigo 3: Criando a tela inicial

```
require "luasmartgui.lsgui"

Window = lsgui.Window
Style = lsgui.Style
RelativeLayout = lsgui.RelativeLayout
HorizontalLayout = lsgui.HorizontalLayout
Label = lsgui.Label
VerticalListWidget = lsgui.VerticalListWidget

local window = Window:new{}
local root = RelativeLayout:new{background_color="#3333FF", alpha=255, static=true}
local footer = HorizontalLayout:new{height=60, background_color="#000000", alpha=255, static=true,
    gravity=HorizontalLayout.Gravity.CENTER_VERTICAL}
window:configureLayout(root, footer)

-- Deve ficar sempre no final do arquivo
function handler(evt)
    window:changeFocus(evt)
end
event.register(handler)

window:onDraw()
```

Figura 3: Tela gerada a partir do Código 3

Agora iremos adicionar a imagem de logo, a data e o texto N̈otícias.̇ Para isso, utilizaremos 1 ImageWidget e 2 Labels. O Código 4 mostra a criação dos componentes e sua adição ao Layout raiz, e deve ser inserido na linha 14. A Figura 4 mostra o resultado.

Cdigo 4: Criando a tela inicial

```
1 local logo = ImageWidget:new{image="logo.png", parent_left=true, parent_top=true, margin=10, static=true}
2 local date = Label:new{width=120, height=30, text="10/11/2013", text_color="#FFFFFF", text_size=22,
3     top_align=date, parent_right=true, margin_right=15, margin_top=15, static=true}
4 local news = Label:new{width=200, height=50, text="Noticias", text_size=30, text_color="#FFFFFF",
5     left_align=logo, top=date, margin_top=80, static=true}
6
7 root:addChildren({logo, date, news})
```

Figura 4: Tela gerada a partir da inserção do Código 4

Os parâmetros `parent_left`, `parent_top`, `parent_right`, `top_align`, `left_align` e `top` referem-se às posições dos componentes dentro do RelativeLayout. Os possíveis parâmetros são:

- **top**: componente a qual ele ficará abaixo;
- **bottom**: componente a qual ele ficará acima;
- **left**: componente à esquerda;
- **right**: componente à direita;
- **top_align**: alinha a parte de cima do componente com a parte de cima do componente especificado;

- **bottom_align**: alinha pela parte de baixo do componente;
- **left_align**: alinha pela parte esquerda do componente;
- **right_align**: alinha pela parte direita do componente;
- **parent_top**: alinha o componente no topo do RelativeLayout;
- **parent_bottom**: alinha o componente embaixo do RelativeLayout;
- **parent_left**: alinha o componente na esquerda do RelativeLayout;
- **parent_right**: alinha o componente na direita do RelativeLayout;

Agora iremos adicionar os elementos no rodapé. São 3 ImageWidgets e 3 Labels. Como não iremos utilizá-los, não precisamos guardá-los em variáveis. O Código 5 mostra a criação e adição dos componentes, com resultado mostrado na Figura 5.

Cdigo 5: Adicionando componentes ao rodapé

```
1 footer:addChildren({
2     ImageWidget:new{image="nav.png", static=true, margin_left=20},
3     Label:new{width=70, height=30, text="NAVIGATE", text_color="#FFFFFF", margin_left=10},
4     ImageWidget:new{image="ok.png", static=true, margin_left=20},
5     Label:new{width=50, height=30, text="PRESS", text_color="#FFFFFF", margin_left=10},
6     ImageWidget:new{image="exit.png", static=true, margin_left=20},
7     Label:new{width=50, height=30, text="EXIT", text_color="#FFFFFF", margin_left=10}
8 })
```

Figura 5: Tela com os componentes adicionados ao rodapé, referente ao Código 5

O próximo passo é adicionar a lista de notícias na tela. Para isso, utilizaremos o componente VerticalListWidget. Todo ListWidget precisa de um componente extra chamado *adapter*, que deriva da classe BaseWidgetAdapter. Ela é responsável por gerar os componentes automaticamente para serem exibidos na lista. É necessário implementar 5 funções:

- **BaseWidgetAdapter:getItemsPerPage()**: informa a quantidade de elementos por página na lista;
- **BaseWidgetAdapter:getSize()**: retorna o tamanho da lista, ou seja, a quantidade de componentes;
- **BaseWidgetAdapter:getItem(position)**: retorna o componente na posição passada como parâmetro;
- **BaseWidgetAdapter:getId(position)**: retorna um identificador para o componente localizado na posição informada;
- **BaseWidgetAdapter:getView(parent, position, current_view)**: função responsável por criar o componente de acordo com a posição. O parâmetro `parent` é a lista a qual ele pertence. Já o `current_view` serve para checar se o componente já foi criado. Caso ele seja `nil`, o componente deve ser criado. Caso contrário, pode retornar o próprio `current_view`.

Primeiro, vamos criar uma lista com as notícias apresentadas, utilizando uma tabela com duplas notícida, data. Também implementaremos as funções mais simples do nosso adapter, que chamaremos de NewsAdapter, como mostra o Código 6.

Cdigo 6: Início da implementação do adapter

```
1  news = {
2      {"Noticia 1", "08/11/2013"},
3      {"Noticia 2", "08/11/2013"},
4      {"Noticia 3", "08/11/2013"},
5      {"Noticia 4", "09/11/2013"},
6      {"Noticia 5", "09/11/2013"},
7      {"Noticia 6", "09/11/2013"},
8      {"Noticia 7", "10/11/2013"},
9      {"Noticia 8", "10/11/2013"},
10     {"Noticia 9", "10/11/2013"},
11     {"Noticia 10", "10/11/2013"},
12 }
13
14 NewsAdapter = BaseWidgetAdapter:new{}
15
16 function NewsAdapter:getItemsPerPage()
17     return 6
18 end
19
20 function NewsAdapter:getSize()
21     return #self.news
22 end
23
24 function NewsAdapter:getItem(position)
25     return self.news[position]
26 end
27
28 -- Utilizaremos a propria posicao como identificador
29 function NewsAdapter:getId(position)
30     return position
31 end
32 })
```

Definimos a quantidade de itens por página sendo 6 e utilizaremos uma lista chamada de `news` dentro do adapter para armazenar as informações necessárias. Agora vamos à parte mais complexa: a função `NewsAdapter:getView(parent, position, current_view)`. Sua implementação pode ser vista no Código 7.

Cdigo 7: Implementação da função getView

```
1  function NewsAdapter:getView(parent, position, current_view)
2      if not current_view then
3          local background_color
4          if position % 2 == 0 then
5              background_color = "#555555"
6          else
7              background_color = "#AAAAAA"
8          end
9          current_view = HorizontalLayout:new{width=800, height=60, background_color=background_color,
10             alpha=255, show_margin=true, gravity=HorizontalLayout.Gravity.CENTER_VERTICAL, padding=10}
11         parent:addView(current_view, true)
12         current_view:addChildren({
13             Label:new{width=650, height=30, text=self:getItem(position)[1], static=false,
14                 text_size=20},
15             Label:new{width=120, height=30, text=self:getItem(position)[2], static=false,
16                 text_size=20, margin_left=20}
17         })
18     end
19     return current_view
20 end
```

No Código 7, a linha 2 checa se o componente já está criado, para evitar que o mesmo seja recriado, consumindo memória e tempo de execução desnecessários. As linhas 3 a 8 são utilizadas

para definir a cor de fundo de cada notícia, para que fiquem alternadas. Nas linhas 9 e 10 é criado um HorizontalLayout e armazenado na variável `current_view`, que é retornada pela função. A linha 11 é utilizada para adicionar o componente criado à lista. É necessário chamar essa função antes de realizar operações no componente. As linhas 12 a 16 são utilizadas para adicionar 2 Labels no Layout criado. O valor padrão para o atributo `static` em um Label é `true`, portanto passamos `static=false`, cada Layout poderá ganhar o foco, sendo necessário redesenhar seus componentes.

Agora basta criar um novo NewsAdapter, passando a lista de notícias, e um novo ListWidget, adicionando-o no Layout raiz, como mostra o Código 8. O resultado deve ser semelhante à Figura 6.

Cdigo 8: Finalizando a lista e adicionando-a no Layout

```
1 local news_list = VerticalListWidget:new{width=800, height=370, divider=2, left_align=news_label,
2     top=news_label, margin_top=50, alpha=255, background_color="#FFFFFF"}
3 local adapter = NewsAdapter:new{news=news}
4 news_list:setAdapter(adapter)
5 local pages = math.floor(adapter:getSize() / adapter:getItemsPerPage() + 1) .. ""
6 local page_label = Label:new{width=200, height=50, text="Pagina 1/"..pages, static=false,
7     left_align=news_list, top=news_list, margin_top=10, text_color="#FFFFFF", text_size=25}
8
9 root:addChildren({logo, date, news_label, news_list, page_label})
```

Figura 6: Tela com a lista adicionada, referente aos Códigos 6, 7 e 8

Vamos agora atualizar o texto no Label que mostra a página, quando a mesma for alterada. Para isso, criaremos uma função e a associaremos ao `page_changed_listerer`, como mostra o Código 9

Cdigo 9: Atualizando número da página quando a mesma é alterada

```
1 local function updatePageLabel(view, page)
2     page_label:property("text", "Pagina " .. page .. "/" .. pages)
3 end
4 news_list.page_changed_listener = updatePageLabel
```

Assim, finalizamos nossa tela. O Código 10 mostra o código completo da tela.

Cdigo 10: Código completo do Exemplo 2

```
1 require "luasmartgui.lsgui"
2
3 Window = lsgui.Window
4 Style = lsgui.Style
5 RelativeLayout = lsgui.RelativeLayout
6 HorizontalLayout = lsgui.HorizontalLayout
7 ImageWidget = lsgui.ImageWidget
8 Label = lsgui.Label
```

```
VerticalListWidget = lsgui.VerticalListWidget
BaseWidgetAdapter = lsgui.BaseWidgetAdapter

local window = Window:new{}
local root = RelativeLayout:new{background_color="#3333FF", alpha=255}
local footer = HorizontalLayout:new{height=60, background_color="#000000", alpha=255, static=true,
    gravity=HorizontalLayout.Gravity.CENTER_VERTICAL}
window:configureLayout(root, footer)

local logo = ImageWidget:new{image="logo.png", parent_left=true, parent_top=true, margin=10, static=true}
local date = Label:new{width=120, height=30, text="10/11/2013", text_color="#FFFFFF", text_size=22,
    top_align=date, parent_right=true, margin_right=15, margin_top=15, static=true}
local news_label = Label:new{width=200, height=50, text="Noticias", text_size=30, text_color="#FFFFFF",
    left_align=logo, top=date, margin_top=80, static=true}

news = {
    {"Noticia 1", "08/11/2013"},
    {"Noticia 2", "08/11/2013"},
    {"Noticia 3", "08/11/2013"},
    {"Noticia 4", "09/11/2013"},
    {"Noticia 5", "09/11/2013"},
    {"Noticia 6", "09/11/2013"},
    {"Noticia 7", "10/11/2013"},
    {"Noticia 8", "10/11/2013"},
    {"Noticia 9", "10/11/2013"},
    {"Noticia 10", "10/11/2013"},
}

NewsAdapter = BaseWidgetAdapter:new{}

function NewsAdapter:getItemsPerPage()
    return 6
end

function NewsAdapter:getSize()
    return #self.news
end

function NewsAdapter:getItem(position)
    return self.news[position]
end

-- Utilizaremos a propria posicao como identificador
function NewsAdapter:getId(position)
    return position
end

function NewsAdapter:getView(parent, position, current_view)
    if not current_view then
        local background_color
        if position % 2 == 0 then
            background_color = "#555555"
        else
            background_color = "#AAAAAA"
        end
        current_view = HorizontalLayout:new{width=800, height=60, background_color=background_color, alpha=255,
            show_margin=true, gravity=HorizontalLayout.Gravity.CENTER_VERTICAL, padding=10}
        parent:addView(current_view, true)
        current_view:addChildren({
            Label:new{width=650, height=30, text=self:getItem(position)[1], static=false,
                text_size=20},
            Label:new{width=120, height=30, text=self:getItem(position)[2], static=false,
                text_size=20, margin_left=20}
        })
    end
    return current_view
end

local news_list = VerticalListWidget:new{width=800, height=370, divider=2, left_align=news_label,
    top=news_label, margin_top=50, alpha=255, background_color="#FFFFFF"}
local adapter = NewsAdapter:new{news=news}
news_list:setAdapter(adapter)
local pages = math.floor(adapter:getSize() / adapter:getItemsPerPage() + 1) .. ""
```

```
local page_label = Label:new{width=200, height=50, text="Pagina 1/"..pages, static=false,
    left_align=news_list, top=news_list, margin_top=10, text_color="#FFFFFF", text_size=25}

local function updatePageLabel(view, page)
    page_label:property("text", "Pagina " .. page .. "/" .. pages)
end
news_list.page_changed_listener = updatePageLabel

root:addChildren({logo, date, news_label, news_list, page_label})

footer:addChildren({
    ImageWidget:new{image="nav.png", static=true, margin_left=20},
    Label:new{width=70, height=30, text="NAVIGATE", text_color="#FFFFFF", margin_left=10},
    ImageWidget:new{image="ok.png", static=true, margin_left=20},
    Label:new{width=50, height=30, text="PRESS", text_color="#FFFFFF", margin_left=10},
    ImageWidget:new{image="exit.png", static=true, margin_left=20},
    Label:new{width=50, height=30, text="EXIT", text_color="#FFFFFF", margin_left=10}
})

function windowKeyPressedListener(window, key)
    print(key)
    if key == "EXIT" then
        window:onDestroy()
        return false
    end
    return true
end
window.key_pressed_listener = windowKeyPressedListener

function handler(evt)
    window:changeFocus(evt)
end
event.register(handler)

window:onDraw()
```

Apêndice C

Códigos de Exemplo

Código C.1: Exemplo utilizando o módulo canvas e event

```
1  local button = canvas:new(100, 30)
2
3  function drawButton(pressed)
4      button:clear()
5      button:attrColor("#AAAAAA")
6      button:drawRect("fill", 0, 0, 100, 30)
7      if (pressed) then button:attrColor("#000000") else button:attrColor("#FFFFFF") end
8      button:drawLine(0, 0, 0, 30)
9      button:drawLine(0, 0, 99, 0)
10     if pressed then button:attrColor("#FFFFFF") else button:attrColor("#000000") end
11     button:drawLine(0, 29, 99, 29)
12     button:drawLine(99, 0, 99, 29)
13     button:attrFont("TiresiasScreenfont", 16, "Normal")
14     button:drawText(10, 3, "Button")
15     canvas:compose(5, 5, button)
16     canvas:flush()
17 end
18
19 function press(evt)
20     if evt.type == "press" then
21         if evt.key == "ENTER" then
22             drawButton(true)
23         end
24     elseif evt.type == "release" then
25         if evt.key == "ENTER" then
26             drawButton(false)
27         end
28     end
29 end
30
31 event.register(press)
32
33 drawButton(false)
```

Apêndice D

Códigos do Primeiro Experimento

Código D.1: Código para coletar tempo de redesenho dos componentes em LuaSmartGUI

```
1  require "luasmartgui.lsgui"
2
3  Window = lsgui.Window
4  HorizontalLayout = lsgui.HorizontalLayout
5  TextWidget = lsgui.TextWidget
6  Button = lsgui.Button
7  ImageWidget = lsgui.ImageWidget
8  CheckBox = lsgui.CheckBox
9  Label = lsgui.Label
10 SeekBar = lsgui.SeekBar
11
12 local w = Window:new{}
13 local root = HorizontalLayout:new{background_color="#0000FF", alpha=255, padding=5}
14 w:configureLayout(root)
15
16 local tw = TextWidget:new{name="tw1", width=100, height=30, text="TW", name="TextWidget", margin=1}
17 local bt = Button:new{name="button1", width=100, height=30, text="BT", name="Button", margin=1}
18 local cb = CheckBox:new{name="cb1", width=100, height=30, text="CB", name="CheckBox", margin=1, text_color="black", alpha
       =0}
19 local iw = ImageWidget:new{name="iw1", image="iw.png", name="ImageWidget", focusable=true, margin=1}
20 local seek_bar = SeekBar:new{name="seek1", min=0, max=100, bar_height=5, cursor_size=8, bar_red=0, bar_green=0, bar_blue
       =255, step=10, cursor_red=255, cursor_green=0, cursor_blue=0, width=500, height=30}
21 local label = Label:new{name="label1", width=100, height=30, text="Label", name="Label"}
22 root:addChildren({tw, bt, cb, iw, label, seek_bar})
23
24 function handler(evt)
25     w:changeFocus(evt)
26 end
27
28 event.register(handler)
29
30 w:onDraw()
31
32 widgets = {bt, cb, iw, label, seek_bar, tw}
33
34 for i, w in ipairs(widgets) do
35     for j=1, 350 do
36         print (w.name, j)
37         local time = socket.gettime() * 1000
38         w:redraw(true)
39         local end_time = socket.gettime() * 1000
```

```
40          local file = io.open("coletas/redraw/" .. w.name .. ".txt", "a")
41          file:write((end_time - time) .."\n")
42          file:close()
43      end
44  end
45
46  print("fim")
```

Código D.2: Código para coletar tempo de redesenho dos componentes com o Ginga

```
1   require "socket"
2
3   canvas:attrColor("#0000FF")
4   canvas:clear()
5
6   -- TextWidget
7   tw = canvas:new(100, 30)
8   tw:attrColor("#AAAAAA")
9   tw:drawRect("fill", 0, 0, 100, 30)
10  tw:attrColor("#555555")
11  tw:drawRect("frame", 0, 0, 100, 30)
12  tw:attrColor("#000000")
13  tw:attrFont("TiresiasScreenfont", 16, "Normal")
14  tw:drawText(10, 3, "TextWidget")
15
16  -- Button
17  button = canvas:new(100, 30)
18  button:attrColor("#AAAAAA")
19  button:drawRect("fill", 0, 0, 100, 30)
20  button:attrColor("#000000")
21  button:attrFont("TiresiasScreenfont", 16, "Normal")
22  button:drawText(10, 3, "Button")
23
24  -- CheckBox
25  checked = canvas:new("luasmartgui/res/widget/check.png")
26  canvas:attrColor("#000000")
27  canvas:attrFont("TiresiasScreenfont", 16, "Normal")
28  canvas:drawText(248, 10, "CheckBox")
29
30  -- ImageWidget
31  iw = canvas:new("iw.png")
32
33  canvas:compose(6, 6, tw)
34  canvas:compose(112, 6, button)
35  canvas:compose(218, 6, checked)
36  canvas:compose(324, 6, iw)
37
38  seek_bar = canvas:new(500, 30)
39
40  canvas:flush()
41
42  function redrawButton()
43      button:clear()
44      button:attrColor("#AAAAAA")
45      button:drawRect("fill", 0, 0, 100, 30)
46      if focus_index == 2 then
47          button:attrColor("#FFFFFF")
48          button:drawLine(0, 0, 0, 30)
49          button:drawLine(0, 0, 100, 0)
50          button:attrColor("#000000")
51          button:drawLine(0, 29, 100, 29)
52          button:drawLine(99, 0, 99, 29)
53      end
```

```
54      button:attrColor("#000000")
55      button:attrFont("TiresiasScreenfont", 16, "Normal")
56      button:drawText(10, 3, "Button")
57      canvas:compose(112, 6, button)
58      canvas:flush()
59  end
60
61  function redrawCheckBox()
62      canvas:attrColor("#0000FF")
63      canvas:clear(218, 6, 100, 30)
64      canvas:drawRect("fill", 218, 6, 100, 30)
65      if focus_index == 3 then
66          canvas:attrColor("#000000")
67          canvas:drawRect("frame", 218, 6, 100, 30)
68      end
69      canvas:attrColor("#000000")
70      canvas:attrFont("TiresiasScreenfont", 16, "Normal")
71      canvas:drawText(248, 10, "CheckBox")
72      canvas:compose(218, 6, checked)
73      canvas:flush()
74  end
75
76  function redrawTextWidget()
77      tw:clear()
78      tw:attrColor("#AAAAAA")
79      tw:drawRect("fill", 0, 0, 100, 30)
80      if focus_index == 1 then
81          tw:attrColor("#555555")
82          tw:drawRect("frame", 0, 0, 100, 30)
83      end
84      tw:attrColor("#000000")
85      tw:attrFont("TiresiasScreenfont", 16, "Normal")
86      tw:drawText(10, 3, "TextWidget")
87      canvas:compose(6, 6, tw)
88      canvas:flush()
89  end
90
91  function redrawImageWidget()
92      canvas:clear(324, 6, 100, 30)
93      canvas:compose(324, 6, iw)
94      canvas:flush()
95  end
96
97  function draw_seek_bar()
98      seek_bar:attrColor("black")
99      seek_bar:drawRect("fill", 0, 0, 500, 30)
100     seek_bar:attrColor("white")
101     seek_bar:drawRect("frame", 0, 0, 500, 30)
102     seek_bar:attrColor("blue")
103     seek_bar:drawRect("fill", 10, 13, 480, 5)
104 end
105
106 function draw_cursor()
107     seek_bar:attrColor("red")
108     seek_bar:drawEllipse("fill", 10 + (0 * 480 / 10), 15, 8, 8, 0, 360)
109 end
110
111 function redrawSeekBar()
112     seek_bar:clear()
113     draw_seek_bar()
114     draw_cursor()
115     canvas:compose(6, 50, seek_bar)
116     canvas:flush()
117 end
```

```
118
119 redraw_functions = {redrawTextWidget, redrawButton, redrawCheckBox, redrawImageWidget}
120
121 focus_index = 1
122
123 function press(evt)
124     if evt.type == "press" then
125         if evt.key == "CURSOR_LEFT" then
126             last_index = focus_index
127             focus_index = focus_index + 1
128             if focus_index < 1 then focus_index = 4 end
129             redraw_functions[last_index]()
130             redraw_functions[focus_index]()
131         elseif evt.key == "CURSOR_RIGHT" then
132             last_index = focus_index
133             focus_index = focus_index + 1
134             if focus_index > 4 then focus_index = 1 end
135             redraw_functions[last_index]()
136             redraw_functions[focus_index]()
137         end
138     end
139 end
140
141 function drawLabel()
142     canvas:attrColor("#000000")
143     canvas:attrFont("TiresiasScreenfont", 16, "Normal")
144     canvas:drawText(430, 6, "Label")
145     canvas:flush()
146 end
147
148 for i=1,350 do
149     local time = socket.gettime() * 1000
150     canvas:attrColor("#0000FF")
151     canvas:clear(430, 6, 100, 30)
152     drawLabel()
153     local end_time = socket.gettime() * 1000
154     local file = io.open("coletas/redraw/Label2.txt", "a")
155     file:write((end_time - time) .."\n")
156     file:close()
157
158     time = socket.gettime() * 1000
159     redrawButton()
160     end_time = socket.gettime() * 1000
161     file = io.open("coletas/redraw/Button2.txt", "a")
162     file:write((end_time - time) .."\n")
163     file:close()
164
165     time = socket.gettime() * 1000
166     redrawCheckBox()
167     end_time = socket.gettime() * 1000
168     file = io.open("coletas/redraw/CheckBox2.txt", "a")
169     file:write((end_time - time) .."\n")
170     file:close()
171
172     time = socket.gettime() * 1000
173     redrawImageWidget()
174     end_time = socket.gettime() * 1000
175     file = io.open("coletas/redraw/ImageWidget2.txt", "a")
176     file:write((end_time - time) .."\n")
177     file:close()
178
179     time = socket.gettime() * 1000
180     redrawTextWidget()
181     end_time = socket.gettime() * 1000
```

```
182     file = io.open("coletas/redraw/TextWidget2.txt", "a")
183     file:write((end_time - time) .."\n")
184     file:close()
185
186     time = socket.gettime() * 1000
187     redrawSeekBar()
188     end_time = socket.gettime() * 1000
189     file = io.open("coletas/redraw/SeekBar2.txt", "a")
190     file:write((end_time - time) .."\n")
191     file:close()
192 end
193
194 print("fim")
```

Código D.3: Botão desenvolvido com LuaSmartGUI

```
1  require "luasmartgui.lsgui"
2
3  Window = lsgui.Window
4  HorizontalLayout = lsgui.HorizontalLayout
5  Button = lsgui.Button
6  Style = lsgui.Style
7
8  local w = Window:new{}
9  local layout = HorizontalLayout:new{static=true}
10 w:configureLayout(layout)
11
12 local button = Button:new{width=100, height=30, text="Button"}
13 layout:addChild(button)
14 w:onDraw()
15
16 function handler(evt)
17     w:changeFocus(evt)
18 end
19
20 event.register(handler)
```

Código D.4: Botão desenvolvido com o módulo canvas

```
1  local button = canvas:new(100, 30)
2
3  function drawButton(pressed)
4      button:clear()
5      button:attrColor("#AAAAAA")
6      button:drawRect("fill", 0, 0, 100, 30)
7      if (pressed) then button:attrColor("#000000") else button:attrColor("#FFFFFF") end
8      button:drawLine(0, 0, 0, 30)
9      button:drawLine(0, 0, 99, 0)
10     if pressed then button:attrColor("#FFFFFF") else button:attrColor("#000000") end
11     button:drawLine(0, 29, 99, 29)
12     button:drawLine(99, 0, 99, 29)
13     button:attrFont("TiresiasScreenfont", 16, "Normal")
14     button:drawText(10, 3, "Button")
15     canvas:compose(5, 5, button)
16     canvas:flush()
17 end
18
19 function press(evt)
20     if evt.type == "press" then
21         if evt.key == "ENTER" then
22             drawButton(true)
23         end
```

```
24      elseif evt.type == "release" then
25          if evt.key == "ENTER" then
26              drawButton(false)
27          end
28      end
29  end
30
31  event.register(press)
32
33  drawButton(false)
```

Código D.5: CheckBox desenvolvido com LuaSmartGUI

```
1   require "luasmartgui.lsgui"
2
3   Window = lsgui.Window
4   HorizontalLayout = lsgui.HorizontalLayout
5   CheckBox = lsgui.CheckBox
6
7   local w = Window:new{}
8   local layout = HorizontalLayout:new{static=true, alpha=0}
9   w:configureLayout(layout)
10
11  local cb = CheckBox:new{width=100, height=30, text="CheckBox", alpha=0, text_color="black"}
12  layout:addChild(cb)
13  w:onDraw()
14
15  function handler(evt)
16      w:changeFocus(evt)
17  end
18
19  event.register(handler)
20
21  file:write((end_time - time) .."\n")
22  file:close()
```

Código D.6: CheckBox desenvolvido com o módulo canvas

```
1   local checked = canvas:new("luasmartgui/res/widget/check.png")
2   local unchecked = canvas:new("luasmartgui/res/widget/check_off.png")
3
4   function drawCheckBox()
5       canvas:compose(10, 10, checked)
6       canvas:attrColor("#000000")
7       canvas:attrFont("TiresiasScreenfont", 16, "Normal")
8       canvas:drawText(36, 10, "CheckBox")
9       canvas:flush()
10  end
11
12  function changeChecked(check)
13      canvas:clear(10, 10, 23, 23)
14      if check then
15          canvas:compose(10, 10, checked)
16      else
17          canvas:compose(10, 10, unchecked)
18      end
19      canvas:flush()
20  end
21
22  function handler(evt)
23      if evt.type == "press" and evt.key == "ENTER" then
24          check = not check
```

```
25          changeChecked(check)
26      end
27  end
28
29  local check = true
30
31  event.register(handler)
32
33  drawCheckBox()
```

Código D.7: Dialog desenvolvido com LuaSmartGUI

```
1   require "luasmartgui.lsgui"
2   Window = lsgui.Window
3   HorizontalLayout = lsgui.HorizontalLayout
4   Dialog = lsgui.Dialog
5   Label = lsgui.Label
6
7   local w = Window:new{}
8   local root = HorizontalLayout:new{background_color="#FFFFFF", static=true}
9   w:configureLayout(root)
10
11  local dialog = Dialog:new{window=w, width=w.width/2, height=w.height/2, gravity=Dialog.Gravity.CENTER}
12  local label = Label:new{width=100, height=30, text="Dialog", text_color="#FFFFFF"}
13  dialog:setContentView(label)
14  w:onDraw()
15
16  function handler(evt)
17      if evt.type == "press" and evt.key == "ENTER" and not dialog.showing then
18          dialog:show()
19      elseif evt.type == "press" and evt.key == "BACK" and dialog.showing then
20          dialog:close()
21      end
22  end
23
24  event.register(handler)
25
26  dialog:show()
```

Código D.8: Dialog desenvolvido com o módulo canvas

```
1   local background_canvas
2   local fade_canvas = canvas:new(1280, 720)
3
4   canvas:attrColor("#FFFFFF")
5   canvas:drawRect('fill', 0, 0, 1280, 720)
6
7   function showDialog()
8       background_canvas = canvas:new(1280, 720)
9       fade_canvas = canvas:new(1280, 720)
10      background_canvas:compose(0, 0, canvas)
11      fade_canvas:attrColor(0, 0, 0, 150)
12      fade_canvas:drawRect('fill', 0, 0, 1280, 720)
13      fade_canvas:attrColor("#000000")
14      fade_canvas:drawRect('fill', 320, 180, 640, 360)
15      fade_canvas:attrColor("#FFFFFF")
16      fade_canvas:drawRect('frame', 320, 180, 640, 360)
17      fade_canvas:drawText(620, 350, "Dialog")
18      canvas:compose(0, 0, fade_canvas)
19      canvas:flush()
20  end
21
```

```
22  function closeDialog()
23      fade_canvas = nil
24      canvas:compose(0, 0, background_canvas)
25      canvas:flush()
26      background_canvas = nil
27      dialog = nil
28      collectgarbage()
29  end
30
31  function handler(evt)
32      if evt.type == "press" and evt.key == "ENTER" and not showing then
33          showDialog()
34          showing = true
35      elseif evt.type == "press" and evt.key == "BACK" and showing then
36          closeDialog()
37          showing = false
38      end
39  end
40
41  event.register(handler)
42  local showing = true
43  showDialog()
```

Código D.9: ImageWidget desenvolvido com LuaSmartGUI

```
1   require "luasmartgui.lsgui"
2
3   Window = lsgui.Window
4   HorizontalLayout = lsgui.HorizontalLayout
5   ImageWidget = lsgui.ImageWidget
6
7   local w = Window:new{}
8   local layout = HorizontalLayout:new{static=true}
9   w:configureLayout(layout)
10
11  local tw = ImageWidget:new{width=100, height=30, image="lua.png"}
12  layout:addChild(tw)
13  w:onDraw()
14
15  function handler(evt)
16      w:changeFocus(evt)
17  end
18
19  event.register(handler)
```

Código D.10: ImageWidget desenvolvido com o módulo canvas

```
1   require "luasmartgui.lsgui"
2
3   Window = lsgui.Window
4   HorizontalLayout = lsgui.HorizontalLayout
5   ImageWidget = lsgui.ImageWidget
6
7   local w = Window:new{}
8   local layout = HorizontalLayout:new{static=true}
9   w:configureLayout(layout)
10
11  local tw = ImageWidget:new{width=100, height=30, image="lua.png"}
12  layout:addChild(tw)
13  w:onDraw()
14
15  function handler(evt)
```

```
16      w:changeFocus(evt)
17  end
18
19  event.register(handler)
```

Código D.11: Label desenvolvido com LuaSmartGUI

```
1   require "luasmartgui.lsgui"
2
3   Window = lsgui.Window
4   HorizontalLayout = lsgui.HorizontalLayout
5   Label = lsgui.Label
6
7   local w = Window:new{}
8   local layout = HorizontalLayout:new{static=true}
9   w:configureLayout(layout)
10
11  local label = Label:new{width=100, height=30, text="Label"}
12  layout:addChild(label)
13  w:onDraw()
14
15  function handler(evt)
16      w:changeFocus(evt)
17  end
18
19  event.register(handler)
```

Código D.12: Label desenvolvido com o módulo canvas

```
1   canvas:attrColor("#000000")
2   canvas:attrFont("TiresiasScreenfont", 16, "Normal")
3   canvas:drawText(3, 3, "Label")
4   canvas:flush()
```

Código D.13: SeekBar desenvolvido com LuaSmartGUI

```
1   require "luasmartgui.lsgui"
2
3   Window = lsgui.Window
4   VerticalLayout = lsgui.VerticalLayout
5   SeekBar = lsgui.SeekBar
6
7   local window = Window:new{}
8   local root_layout = VerticalLayout:new{static=true}
9   window:configureLayout(root_layout)
10
11  local seek_bar = SeekBar:new{min=0, max=100, bar_height=5, cursor_size=8, bar_red=0, bar_green=0, bar_blue=255, step=10,
        cursor_red=255, cursor_green=0, cursor_blue=0, width=500, height=30}
12  root_layout:addChild(seek_bar)
13
14  function handler(evt)
15      window:changeFocus(evt)
16  end
17
18  event.register(handler)
19
20  window:onDraw()
```

Código D.14: SeekBar desenvolvido com o módulo canvas

```lua
local SCREEN_WIDTH, SCREEN_HEIGHT = canvas:attrSize()
local cursor_position = 0
local seek_bar = canvas:new(500, 30)

function draw_seek_bar()
    seek_bar:attrColor("black")
    seek_bar:drawRect("fill", 0, 0, 500, 30)
    seek_bar:attrColor("white")
    seek_bar:drawRect("frame", 0, 0, 500, 30)
    seek_bar:attrColor("blue")
    seek_bar:drawRect("fill", 10, 13, 480, 5)
end

function draw_cursor()
    seek_bar:attrColor("red")
    seek_bar:drawEllipse("fill", 10 + (cursor_position * 480 / 10), 15, 8, 8, 0, 360)
end

function redraw()
    seek_bar:clear()
    draw_seek_bar()
    draw_cursor()
    canvas:compose(0, 0, seek_bar)
    canvas:flush()
end

function handler(evt)
    if evt.type == "press" then
        if evt.key == "CURSOR_LEFT" and cursor_position > 0 then
            cursor_position = cursor_position - 1
        elseif evt.key == "CURSOR_RIGHT" and cursor_position < 10 then
            cursor_position = cursor_position + 1
        end
        redraw()
    end
end

event.register(handler)

redraw()
```

Código D.15: TextWidget desenvolvido com LuaSmartGUI

```lua
require "luasmartgui.lsgui"

Window = lsgui.Window
HorizontalLayout = lsgui.HorizontalLayout
TextWidget = lsgui.TextWidget

local w = Window:new{}
local layout = HorizontalLayout:new{static=true}
w:configureLayout(layout)

local tw = TextWidget:new{width=100, height=30, text="TextWidget"}
layout:addChild(tw)

function handler(evt)
    w:changeFocus(evt)
end

event.register(handler)

w:onDraw()
```

Código D.16: TextWidget desenvolvido com o módulo canvas

```
1  local tw = canvas:new(100, 30)
2  tw:attrColor("#AAAAAA")
3  tw:drawRect("fill", 0, 0, 100, 30)
4  tw:attrColor("#555555")
5  tw:drawRect("frame", 0, 0, 100, 30)
6  tw:attrColor("#000000")
7  tw:attrFont("TiresiasScreenfont", 16, "Normal")
8  tw:drawText(10, 3, "TextWidget")
9  canvas:compose(5, 5, tw)
10 canvas:flush()
```

Apêndice E

Códigos do Segundo Experimento

Código E.1: Tela desenvolvida pelo participante 1 com LuaSmartGUI

```
1  require "luasmartgui.lsgui"
2
3  Window = lsgui.Window
4  ImageWidget = lsgui.ImageWidget
5  VerticalLayout = lsgui.VerticalLayout
6  Label = lsgui.Label
7  CheckBox = lsgui.CheckBox
8  Button = lsgui.Button
9  SeekBar = lsgui.SeekBar
10 HorizontalLayout = lsgui.HorizontalLayout
11
12 local w = Window:new{}
13 local root = VerticalLayout:new{padding_left=300, padding_top=100, background_color="white", alpha=255, gravity=
       VerticalLayout.Gravity.LEFT}
14 w:configureLayout(root)
15
16 local iw = ImageWidget:new{image="questionario.png", static=true}
17
18 local questionLabel = Label:new{text="Questão", text_color="black", width=150, height=50, text_size=36}
19
20 local cb1 = CheckBox:new{text="Escolha 1", background_color="white", text_color="black", width=100, height=50, text_size
       =12, padding=3}
21 local cb2 = CheckBox:new{text="Escolha 2", background_color="white", text_color="black", width=100, height=50, text_size
       =12, padding=3}
22 local cb3 = CheckBox:new{text="Escolha 3", background_color="white", text_color="black", width=100, height=50, text_size
       =12, padding=3}
23 local cb4 = CheckBox:new{text="Escolha 4", background_color="white", text_color="black", width=100, height=50, text_size
       =12, padding=3}
24 local cb5 = CheckBox:new{text="Escolha 5", background_color="white", text_color="black", width=100, height=50, text_size
       =12, padding=3}
25
26 local hlayout = HorizontalLayout:new{background_color="white", alpha=255, width=200, height=40, margin_top=10}
27
28 local back = Button:new{width=100, height=30, text="<< Voltar", margin=1}
29 local next_b = Button:new{width=100, height=30, text="Avançar >>", margin=1}
30
31 local barLayout = HorizontalLayout:new{margin_left=-200, padding=2, background_color="white", alpha=255, width=500, height
       =30}
32
33 local bar = SeekBar:new{width=800, height=25, min=0, max=100, margin=1, cursor_size=10, bar_red=175, bar_blue=175,
       bar_green=175, cursor_red=205, cursor_blue=205, cursor_green=205, background_color="white", focusable=false}
```

```
34
35 local percent = Label:new{text="50%", text_color="black", width=50, height=25, text_size=20}
36
37 root:addChildren({iw, questionLabel, cb1, cb2, cb3, cb4, cb5, hlayout, barLayout})
38
39 hlayout:addChildren({back, next_b})
40 back:setFocusDirection(cb5, nil, nil, next_b)
41 next_b:setFocusDirection(cb5, nil, back, nil)
42
43 barLayout:addChildren({bar, percent})
44
45 bar:seek(50)
46
47 back.key_pressed_listener = function (button, key)
48     button:onKeyPressed(key)
49     if (key == "ENTER") then
50         bar:seek(bar:tell() - 10)
51         percent:property("text", bar:tell() .. "%")
52     end
53     return true
54 end
55
56
57 next_b.key_pressed_listener = function (button, key)
58     button:onKeyPressed(key)
59     if (key == "ENTER") then
60         bar:seek(bar:tell() + 10)
61         percent:property("text", bar:tell() .. "%")
62     end
63     return true
64 end
65
66
67 function handler(evt)
68     w:changeFocus(evt)
69 end
70
71 event.register(handler)
72
73 w:onDraw()
```

Código E.2: Tela desenvolvida pelo participante 2 com LuaSmartGUI

```
1  require "luasmartgui.lsgui"
2
3  Window = lsgui.Window
4  VerticalLayout = lsgui.VerticalLayout
5  HorizontalLayout = lsgui.HorizontalLayout
6  Label = lsgui.Label
7  CheckBox = lsgui.CheckBox
8  Button = lsgui.Button
9  SeekBar = lsgui.SeekBar
10 ImageWidget = lsgui.ImageWidget
11 Style = lsgui.Style
12
13 local window = Window:new{}
14 local layoutVerticalRoot = VerticalLayout:new{background_color="white", alpha=255, padding_left=250, padding_top=50}
15 window:configureLayout(layoutVerticalRoot)
16 local layoutHorizontal = HorizontalLayout:new{background_color="white", gravity=HorizontalLayout.Gravity.CENTER_VERTICAL,
       alpha=255, width=205, height=52}
17 local layoutHorizontal2 = HorizontalLayout:new{background_color="white", gravity=HorizontalLayout.Gravity.CENTER_VERTICAL,
       alpha=255, width=800, height=50, margin_left=-150}
18
```

```
19 local img = ImageWidget:new{image="questionario.png", static=true}
20 local label = Label:new{text="Questão", text_size=22, width=150, height=50, text_size=32}
21
22 local checkBoxStyle = Style:create{margin=1,padding_left=5}
23
24 local checkBox1 = CheckBox:new{text="Opção 1", text_color="black", style=checkBoxStyle, background_color="white", width
       =100, height=50}
25 local checkBox2 = CheckBox:new{text="Opção 2", text_color="black", style=checkBoxStyle, background_color="white", width
       =100, height=50}
26 local checkBox3 = CheckBox:new{text="Opção 3", text_color="black", style=checkBoxStyle, background_color="white", width
       =100, height=50}
27 local checkBox4 = CheckBox:new{text="Opção 4", text_color="black", style=checkBoxStyle, background_color="white", width
       =100, height=50}
28 local checkBox5 = CheckBox:new{text="Opção 5", text_color="black", style=checkBoxStyle, background_color="white", width
       =100, height=50}
29
30 local button1 = Button:new{text="<< Voltar", width=100, height=30}
31 local button2 = Button:new{text="Avançar >>", width=100, height=30, margin_left=2}
32
33 local seekBar = SeekBar:new{min=0, max=100, bar_height=10, cursor_size=10, width=700, height=50, cursor_red=200,
       cursor_green=200, cursor_blue=200, bar_red=111, bar_green=111, bar_blue=111, background_color="white", focusable=
       false}
34
35 local label2 = Label:new{text="0%", text_size=20, width=100, height=50}
36
37 layoutVerticalRoot:addChildren({img, label, checkBox1, checkBox2, checkBox3, checkBox4, checkBox5, layoutHorizontal,
       layoutHorizontal2})
38 layoutHorizontal:addChildren({button1, button2})
39 layoutHorizontal2:addChildren({seekBar, label2})
40
41 window:onDraw()
42
43 function move(view, key)
44     view:onKeyPressed(key)
45     if view==button1 then
46         seekBar:seek(seekBar:tell()-10,true)
47     elseif view==button2 then
48         seekBar:seek(seekBar:tell()+10,true)
49     end
50     return true
51 end
52
53 function updatePerc(view,position)
54     label2:property("text",position.."%")
55     return true
56 end
57
58 seekBar.seek_listener=updatePerc
59
60 button1.key_pressed_listener=move
61 button2.key_pressed_listener=move
62
63 button1:setFocusDirection(checkBox5,checkBox1,nil,button2)
64 button2:setFocusDirection(checkBox5,checkBox1,button1,nil)
65
66 function handler(evt)
67     window:changeFocus(evt)
68 end
69 event.register(handler)
```

Código E.3: Tela desenvolvida pelo participante 3 com LuaSmartGUI

```
1 require "luasmartgui.lsgui"
```

```
2
3  local window = lsgui.Window:new{}
4  local rootlayout = lsgui.VerticalLayout:new{background_color="#FFFFFF",alpha=255,gravity=lsgui.VerticalLayout.Gravity.
       CENTER_HORIZONTAL}
5  window:configureLayout(rootlayout)
6
7  local questionslayout = lsgui.VerticalLayout:new{width=400, height=380, margin_top=50}
8  rootlayout:addChild(questionslayout)
9
10 local title_image = lsgui.ImageWidget:new{image="questionario.png", margin_left=-20, static=true}
11 local title_label = lsgui.Label:new{text="Questão", text_color="black",width=100, height=30, text_size=28,margin_left=-20,
       margin_top=10,margin_bottom=10}
12
13 questionslayout:addChildren({title_image, title_label})
14 answers = {}
15 for x=1,5 do
16   table.insert(answers, lsgui.CheckBox:new{text="Escolha "..x,text_color="black",width=100, height=30, text_size=20,
         background_color="#FFFFFF",background_alpha=0,margin=5,})
17 end
18 questionslayout:addChildren(answers)
19
20 local button_area = lsgui.HorizontalLayout:new{width=250, height=40}
21 local previous_button = lsgui.Button:new{text="<< Voltar",text_color="black",margin=5,width=120, height=30, text_size=20,
       background_alpha=0,background_color="#AAAAAA",show_margin=true}
22 local next_button = lsgui.Button:new{text="Avançar >>",text_color="black",margin=5,width=120, height=30, text_size=20,
       background_alpha=0,background_color="#AAAAAA",show_margin=true}
23
24 questionslayout:addChild(button_area)
25 button_area:addChildren({previous_button, next_button})
26
27 local progress_bar_area = lsgui.HorizontalLayout:new{width=600, height=30,focusable=false,gravity=lsgui.HorizontalLayout.
       Gravity.CENTER_VERTICAL}
28 local progress_bar = lsgui.SeekBar:new{min=0, max=100, bar_height=20, width=500,height=20,background_color="white",
       background_alpha=0,bar_height=10,cursor_red=50,cursor_green=50,cursor_blue=50,bar_red=220, bar_green=220, bar_blue
       =220,focusable=false}
29
30 local progress_label = lsgui.Label:new{text="0%", text_color="black",width=50, height=20, text_size=15,margin=5, focusable
       =false}
31 rootlayout:addChild(progress_bar_area)
32 progress_bar_area:addChildren({progress_bar, progress_label})
33
34 function handler(evt)
35   window:changeFocus(evt)
36 end
37 event.register(handler)
38
39 previous_button.key_pressed_listener = function(view, key)
40   view:onKeyPressed(key)
41
42   if key == "ENTER" then
43     local cur_val = progress_bar:tell()
44     if cur_val > 0 then
45       cur_val = cur_val - math.min(cur_val, 10)
46     end
47
48     progress_bar:seek(cur_val)
49     progress_label:property("text", cur_val.."%")
50   end
51
52   return true
53 end
54
55 next_button.key_pressed_listener = function(view, key)
56   view:onKeyPressed(key)
```

```
57
58    if key == "ENTER" then
59      local cur_val = progress_bar:tell()
60      if cur_val < 100 then
61        cur_val = cur_val + 10
62        cur_val = math.min (cur_val, 100)
63      end
64
65      progress_bar:seek(cur_val)
66      progress_label:property("text", cur_val.."%")
67    end
68
69    return true
70  end
71
72  window:onDraw()
```

Código E.4: Tela desenvolvida pelo participante 4 com LuaSmartGUI

```
1   require "luasmartgui.lsgui"
2
3   Window = lsgui.Window
4   VerticalLayout = lsgui.VerticalLayout
5   HorizontalLayout = lsgui.HorizontalLayout
6   ImageWidget = lsgui.ImageWidget
7   Style = lsgui.Style
8   Label = lsgui.Label
9   CheckBox = lsgui.CheckBox
10  Button = lsgui.Button
11  SeekBar = lsgui.SeekBar
12
13  local mainScreen = Window:new{}
14  local mainLayout = VerticalLayout:new{gravity = VerticalLayout.Gravity.CENTER_HORIZONTAL, alpha = 255, background_color =
        "white"}
15  mainScreen:configureLayout(mainLayout)
16
17  local navigationLayout = HorizontalLayout:new{gravity = HorizontalLayout.Gravity.CENTER_VERTICAL, width = 200, height = 50}
18  local seekLayout = HorizontalLayout:new{gravity = HorizontalLayout.Gravity.CENTER_VERTICAL, width = 600, height = 50,
        focusable = false}
19  local questionLayout = VerticalLayout:new{gravity = VerticalLayout.Gravity.CENTER_HORIZONTAL, width = 200, height = 50 * 6}
20
21  local questionImage = ImageWidget:new{image="questionario.png", static=true}
22  local questionText = Label:new{text = "Who discovered Brazil?", width = 200, height = 50, gravity = lsgui.TextWidget.
        Gravity.CENTER, text_size=20}
23  mainLayout:addChildren({questionImage, questionText})
24
25  local checkBoxList = {}
26
27  for i = 1, 5 do
28      local checkBox = CheckBox:new{name = i, text = "Option " .. i, width = 190, height = 50, background_color = "white",
            text_color = "black"}
29      mainLayout:addChild(checkBox)
30      table.insert(checkBoxList, checkBox)
31  end
32
33  local buttonStyle = Style:create{margin = 5}
34
35  mainLayout:addChild(navigationLayout)
36
37  local buttonPrevious = Button:new{text = "Previous", width = 90, height = 40, style = buttonStyle}
38  local buttonNext = Button:new{text = "Next", width = 90, height = 40, style = buttonStyle}
39  navigationLayout:addChildren({buttonPrevious, buttonNext})
40
```

```
41 local seekBar = SeekBar:new{min = 0, max = 100, width = 550, height = 50, step = 50,bar_red = 111, bar_green = 111,
       bar_blue = 111,cursor_red = 200, cursor_green = 200, cursor_blue = 200, background_color = "white",focusable = false
       }
42
43 mainLayout:addChild(seekLayout)
44 seekLayout:addChild(seekBar)
45
46 local seekLabel = Label:new{focusable = false,text = seekBar:tell() .. "%", width = 50, height = 50,gravity = lsgui.Label.
       Gravity.CENTER}
47 seekLayout:addChild(seekLabel)
48
49 function buttonHandler(button, key)
50     button:onKeyPressed(key)
51     if (button == buttonPrevious and seekBar:tell() > 0 and key == "ENTER") then
52         seekBar:seek(seekBar:tell() - 10)
53         seekLabel:property("text", seekBar:tell() .. "%")
54     elseif (button == buttonNext and seekBar:tell() < 100 and key == "ENTER") then
55         seekBar:seek(seekBar:tell() + 10)
56         seekLabel:property("text", seekBar:tell() .. "%")
57     end
58
59     return true
60 end
61
62 buttonPrevious.key_pressed_listener = buttonHandler
63 buttonNext.key_pressed_listener = buttonHandler
64
65 function handler(evt)
66     mainScreen:changeFocus(evt)
67 end
68
69 event.register(handler)
70
71 mainScreen:onDraw()
```

Código E.5: Tela desenvolvida pelo participante 5 com LuaSmartGUI

```
1  require "luasmartgui.lsgui"
2
3  Window = lsgui.Window
4  VerticalLayout = lsgui.VerticalLayout
5  HorizontalLayout = lsgui.HorizontalLayout
6  Label = lsgui.Label
7  CheckBox = lsgui.CheckBox
8  Button = lsgui.Button
9  SeekBar = lsgui.SeekBar
10 ImageWidget = lsgui.ImageWidget
11
12 local window = Window:new{}
13 local root = VerticalLayout:new{background_color='#FFFFFF', alpha=255, padding_left = 500, padding_top=100}
14 local buttons = HorizontalLayout:new{width=200, height = 100, background_color='#FFFFFF',gravity=HorizontalLayout.Gravity.
       CENTER_VERTICAL, alpha=255}
15 local seekBarLayout = HorizontalLayout:new{width=600, height = 30, background_color='#FFFFFF',gravity=HorizontalLayout.
       Gravity.CENTER_VERTICAL, alpha=255, focusable=false}
16
17 local questImage = ImageWidget:new{image="questionario.png", static=true, margin_left=-20}
18 local question = Label:new{width=120, height=60, text="Questão", text_color='#000000', text_size=30, margin_left=-20}
19 local checkbox1 = CheckBox:new{width=100, height=30, text="Opção 1", text_color='#000000', background_color='#FFFFFF',
       padding_left=5}
20 local checkbox2 = CheckBox:new{width=100, height=30, text="Opção 2", text_color='#000000', background_color='#FFFFFF',
       padding_left=5}
21 local checkbox3 = CheckBox:new{width=100, height=30, text="Opção 3", text_color='#000000', background_color='#FFFFFF',
       padding_left=5}
```

```
22 local checkbox4 = CheckBox:new{width=100, height=30, text="Opção 4", text_color='#000000', background_color='#FFFFFF',
       padding_left=5}
23 local checkbox5 = CheckBox:new{width=100, height=30, text="Opção 5", text_color='#000000', background_color='#FFFFFF',
       padding_left=5}
24 local buttonVoltar = Button:new{name='bv', width=120, height=30, text="<< Voltar", text_color='#000000', text_size=20,
       margin_right = 20}
25 local buttonAvancar = Button:new{name='ba',width=120, height=30, text="Avançar >>", text_color='#000000', text_size=20}
26 local seekBar = SeekBar:new{width=510, height=30,min=0, max=100,background_color='#FFFFFF', bar_red=111, bar_green=111,
       bar_blue=111, cursor_red=200, cursor_green=200,cursor_blue=200, margin_right = 10, margin_left=-160}
27 local seekBarLabel = Label:new{width=50, height=20, text="50%", text_color='#000000'}
28
29 window:configureLayout(root)
30 root:addChildren({questImage, question, checkbox1, checkbox2, checkbox3, checkbox4, checkbox5, buttons, seekBarLayout})
31 buttons:addChildren({buttonVoltar, buttonAvancar})
32 seekBarLayout:addChildren({seekBar, seekBarLabel})
33
34
35 seekBar:seek(50)
36 buttonVoltar:setFocusDirection(checkbox5, nil, nil, buttonAvancar)
37 buttonAvancar:setFocusDirection(checkbox5, nil, buttonVoltar, nil)
38
39 function key_pressed_listener_avancar(view, key)
40     view:onKeyPressed(key)
41     if (key == 'ENTER') then
42         if(seekBar:tell() < 100) then
43             seekBar:seek(seekBar:tell()+10)
44             seekBarLabel:property('text', seekBar:tell().."%")
45
46         end
47     end
48     return true
49 end
50
51 function key_pressed_listener_voltar(view, key)
52     view:onKeyPressed(key)
53     if (key == 'ENTER') then
54         if(seekBar:tell() > 0) then
55             seekBar:seek(seekBar:tell()-10)
56             seekBarLabel:property('text', seekBar:tell().."%")
57
58         end
59     end
60     return true
61 end
62
63 buttonVoltar.key_pressed_listener = key_pressed_listener_voltar
64 buttonAvancar.key_pressed_listener = key_pressed_listener_avancar
65
66 function handler(evt)
67     window:changeFocus(evt)
68 end
69
70 event.register(handler)
71
72 window:onDraw()
```

Código E.6: Tela desenvolvida pelo participante 6 com LuaSmartGUI

```
1 require "luasmartgui.lsgui"
2
3 Window = lsgui.Window
4 VerticalLayout = lsgui.VerticalLayout
5 ImageWidget = lsgui.ImageWidget
```

```
6  Label = lsgui.Label
7  CheckBox = lsgui.CheckBox
8  Button = lsgui.Button
9  SeekBar = lsgui.SeekBar
10 HorizontalLayout = lsgui.HorizontalLayout
11
12 local w = Window:new{}
13 local root = VerticalLayout:new{background_color = "white",alpha = 255, gravity = VerticalLayout.Gravity.LEFT,
       padding_left = 200, padding_top=100}
14 w:configureLayout(root)
15
16 local img = ImageWidget:new{image="questionario.png", margin_left=100, static=true}
17
18 local text_widget = Label:new{margin_left=100, text="Questão", text_size=30, width=150, height=100,stext_size=30}
19
20 root:addChildren({img, text_widget})
21
22 for i = 1, 5 do
23     local checkBox = CheckBox:new{margin_left=145, padding_left = 5, text = "Opção ".. i, background_color = "white",
           text_color = "black", width = 120, height = 30}
24     root:addChild(checkBox)
25 end
26
27 local botaoVoltar = Button:new{text="<< Voltar", width=100, height=30}
28 local botaoAvancar = Button:new{text="Avançar >> ", width=100, height=30, margin_left=20}
29
30 local buttons_layout = HorizontalLayout:new{margin_left=150, background_color = "white",margin_top = 20,alpha = 255,width
       = 200,height = 50,gravity = HorizontalLayout.Gravity.CENTER_VERTICAL}
31 root:addChild(buttons_layout)
32
33 buttons_layout:addChildren({botaoVoltar, botaoAvancar})
34
35 local seekBar = SeekBar:new{min=0, max=100, width=800, height=50, step=10, bar_height=5, cursor_size=10, bar_red=111,
       bar_green=111, bar_blue=111, cursor_red=200, cursor_green=200, cursor_blue=200, background_color = "white"}
36
37 local text_bar = Label:new{text="50%", width=40, height=50}
38
39 local seek_layout = HorizontalLayout:new{background_color = "white", margin_top = 20, alpha = 255, width = 400, height =
       50, focusable = false,gravity = HorizontalLayout.Gravity.CENTER_VERTICAL}
40
41 root:addChild(seek_layout)
42
43 seek_layout:addChildren({seekBar, text_bar})
44
45 function handlerButton(view, key)
46     view:onKeyPressed(key)
47
48     if key == "ENTER" and view == botaoAvancar then
49         seekBar:seek(seekBar:tell()+10)
50     elseif key == "ENTER" and view == botaoVoltar then
51         seekBar:seek(seekBar:tell()-10)
52     end
53     text_bar:clear()
54     text_bar:property("text", seekBar:tell() .. "%")
55
56     return true
57 end
58
59 botaoAvancar.key_pressed_listener = handlerButton
60 botaoVoltar.key_pressed_listener = handlerButton
61
62 function handler(evt)
63     w:changeFocus(evt)
64 end
```

```
65
66 event.register(handler)
67 w:onDraw()
```

Código E.7: Tela desenvolvida pelo participante 7 com LuaSmartGUI

```
1  require "luasmartgui.lsgui"
2
3  Window = lsgui.Window
4  ImageWidget = lsgui.ImageWidget
5  Style = lsgui.Style
6  RelativeLayout = lsgui.RelativeLayout
7  VerticalLayout = lsgui.VerticalLayout
8  HorizontalLayout = lsgui.HorizontalLayout
9  Label = lsgui.Label
10 VerticalListWidget = lsgui.VerticalListWidget
11 BaseWidgetAdapter = lsgui.BaseWidgetAdapter
12 CheckBox = lsgui.CheckBox
13 Button = lsgui.Button
14 SeekBar = lsgui.SeekBar
15
16 local alt = {"Escolha 1", "Escolha 2", "Escolha 3", "Escolha 4", "Escolha 5"}
17
18 local window = Window:new{}
19 local root = RelativeLayout:new{background_color="#FFFFFF", alpha=255}
20 window:configureLayout(root)
21 local image = ImageWidget:new{image="questionario.png", parent_left=true, parent_top=true, static=true, margin_left=400}
22 local titulo = Label:new{width=200, height=50, text="Questão", text_size=30, text_color="#000000", top=image, left_align=
       image}
23
24 local AltAdapter = BaseWidgetAdapter:new{}
25
26 function AltAdapter:getItemsPerPage()
27         return 5
28 end
29
30 function AltAdapter:getSize()
31         return #self.alt
32 end
33
34 function AltAdapter:getItem( pos )
35         return self.alt[pos]
36 end
37
38 function AltAdapter:getId( pos )
39         return pos
40 end
41
42 function AltAdapter:getView( parent, pos, view )
43         if not view then
44                 view = VerticalLayout:new{width=700, height=40, background_color="#FFFFFF", alpha=255, padding=5,
                        show_margin=false}
45                 parent:addView(view, true)
46                 view:addChildren({CheckBox:new{width=650, height=30, text=self:getItem(pos), static=false, text_color="
                        #000000", background_color="#FFFFFF"}})
47         end
48         return view
49 end
50
51 local lista = VerticalListWidget:new{width=700, height=250, divider=2, left_align=titulo, top=titulo, margin_top=20, alpha
       =255, background_color="#FFFFFF"}
52 local adapter = AltAdapter:new{alt=alt}
53 lista:setAdapter(adapter)
```

```
54
55 local voltar = Button:new{width=100, height=30, text="Voltar", text_color="#FFFFFF", text_size=15, left_align=lista, top=
       lista}
56 local proximo = Button:new{width=100, height=30, text="Próximo", text_color="#FFFFFF", text_size=15, left=voltar, top=
       lista}
57
58 local barra = SeekBar:new{width=900, height=30, background_color="#FFFFFF", bar_red=0, bar_green=0, bar_blue=0, cursor_red
       =255, cursor_green=0, cursor_blue=0, min=0, max=100, bar_height=10, cursor_size=10, parent_left=true, top=voltar,
       margin_top=10, margin_left=100}
59 local progresso = Label:new{width=100, height=30, text="0%", text_size=30, text_color="#000000", left=barra, top=voltar,
       static=true, margin_top=10}
60
61 root:addChildren({image, titulo, lista, voltar, proximo, barra, progresso})
62
63 function handler( evt )
64         window:changeFocus(evt)
65 end
66 event.register(handler)
67
68 window:onDraw()
```

Código E.8: Tela desenvolvida pelo participante 1 com a biblioteca nativa do Ginga

```
1  local WIDTH, HEIGHT = canvas:attrSize()
2  local FONT_FACE, FONT_SIZE, FONT_STYLE = canvas:attrFont()
3  local CANVAS_CHECK = canvas:new('check.png')
4  local CANVAS_CHECK_OFF = canvas:new('check_off.png')
5
6  local percentage = 0
7  local index = 1
8  local button = 1
9  local checks = {CANVAS_CHECK, CANVAS_CHECK_OFF, CANVAS_CHECK, CANVAS_CHECK_OFF, CANVAS_CHECK_OFF}
10 local questions = {'Questão 1', 'Questão 2', 'Questão 3', 'Questão 4', 'Questão 5'}
11
12 function clear()
13     canvas:attrColor('white')
14     canvas:drawRect('fill', 0, 0, WIDTH, HEIGHT)
15 end
16
17 function draw_image()
18     canvas:compose(190, 50, canvas:new("questionario.png"))
19 end
20
21 function draw_label()
22     canvas:attrFont(FONT_FACE, 40, FONT_STYLE)
23     canvas:attrColor('black')
24     canvas:drawText(190, 140, 'Questão')
25 end
26
27 function draw_checks()
28     canvas:attrFont(FONT_FACE, 19, FONT_STYLE)
29     canvas:attrColor('black')
30     for i=1,5 do
31         canvas:compose(210, 200 + (i - 1) * 40, checks[i])
32         canvas:drawText(240, 200 + (i - 1) * 40, questions[i])
33     end
34
35     if index > 0 and index < 6 then
36         canvas:drawRect('frame', 205, 200 + (index - 1) * 40 - 4, 150, 30)
37     end
38 end
39
40 function draw_buttons()
```

```
41      canvas:attrColor('gray')
42      canvas:drawRect('fill', 210, 400, 150, 30)
43      canvas:drawRect('fill', 370, 400, 150, 30)
44
45      canvas:attrColor('white')
46      canvas:drawLine(210, 400, 360, 400)
47      canvas:drawLine(210, 400, 210, 430)
48      canvas:drawLine(370, 400, 520, 400)
49      canvas:drawLine(370, 400, 370, 430)
50
51      canvas:attrColor('black')
52      canvas:drawLine(210, 430, 360, 430)
53      canvas:drawLine(360, 400, 360, 430)
54      canvas:drawLine(370, 430, 520, 430)
55      canvas:drawLine(520, 400, 520, 430)
56
57      canvas:attrFont(FONT_FACE, 20, FONT_STYLE)
58      canvas:attrColor('black')
59      canvas:drawText(230, 403, '<< Voltar')
60      canvas:drawText(390, 403, 'Avançar >>')
61
62      if index == 6 then
63          canvas:attrColor('black')
64          if button == 1 then
65              canvas:drawRect('frame', 210, 400, 150, 30)
66          elseif button == 2 then
67              canvas:drawRect('frame', 370, 400, 150, 30)
68          end
69      end
70  end
71
72  function draw_bar()
73      canvas:attrColor('gray')
74      canvas:drawRect('fill', 100, 460, 800, 10)
75      canvas:attrColor('#AAAAAA')
76      canvas:drawEllipse('fill', 100 + percentage * 800, 465, 10, 10, 0, 360)
77
78      canvas:attrFont(FONT_FACE, 19, FONT_STYLE)
79      canvas:attrColor('black')
80      canvas:drawText(910, 453, (percentage * 100) .. '%')
81  end
82
83  function draw_all()
84      clear()
85      draw_image()
86      draw_label()
87      draw_checks()
88      draw_buttons()
89      draw_bar()
90      canvas:flush()
91  end
92
93  function doEnter()
94      if index > 0 and index < 6 then
95          if checks[index] == CANVAS_CHECK then
96              checks[index] = CANVAS_CHECK_OFF
97          elseif checks[index] == CANVAS_CHECK_OFF then
98              checks[index] = CANVAS_CHECK
99          end
100     elseif index == 6 then
101         if button == 1 then
102             if percentage > 0 then
103                 percentage = percentage - 0.1
104             end
```

```
105         elseif button == 2 then
106             if percentage < 1 then
107                 percentage = percentage + 0.1
108             end
109         end
110         if percentage < 0 then
111             percentage = 0
112         elseif percentage > 1 then
113             percentage = 1
114         end
115     end
116 end
117
118 function event_handler(evt)
119     if evt.type == 'press' then
120         local key = evt.key
121         local redraw = true
122         if key == 'CURSOR_UP' and index > 1 then
123             index = index - 1
124         elseif key == 'CURSOR_DOWN' and index < 6 then
125             index = index + 1
126         elseif key == 'CURSOR_LEFT' and index == 6 and button == 2 then
127             button = 1
128         elseif key == 'CURSOR_RIGHT' and index == 6 and button == 1 then
129             button = 2
130         elseif key == 'ENTER' then
131             doEnter()
132         else
133             redraw = false
134         end
135
136         if (redraw) then
137             draw_all()
138         end
139     end
140 end
141
142 event.register(event_handler)
143
144 draw_all()
```

Código E.9: Tela desenvolvida pelo participante 2 com a biblioteca nativa do Ginga

```
1  local position = 1
2  local seek_position = 1
3  local checked = {true, false, true, false, false}
4
5  function redraw()
6      canvas:attrColor(255, 255, 255, 255)
7      canvas:clear()
8
9      local questionTitle = canvas:new("questionario.png")
10
11     local questionLabel = canvas:new(200, 50)
12     questionLabel:attrFont('Verdana', 36, nil)
13     questionLabel:drawText(0, 0, "Questão")
14
15     local box1 = canvas:new(30, 30)
16     box1:drawRect("frame", 0, 0, 30, 30)
17
18     local check1
19     if checked[1] then
20         check1 = box1:new("check.png")
```

```
21      else
22          check1 = box1:new("check_off.png")
23      end
24      check1:attrScale(30, 30)
25      box1:compose(0, 0, check1)
26
27      local cb1 = canvas:new(100, 30)
28      cb1:attrFont('Verdana', 20, nil)
29      cb1:drawText(0, 0, "Escolha 1")
30
31      local box2 = canvas:new(30, 30)
32      box2:drawRect("frame", 0, 0, 30, 30)
33
34      local check2
35      if checked[2] then
36          check2 = box1:new("check.png")
37      else
38          check2 = box1:new("check_off.png")
39      end
40      check2:attrScale(30, 30)
41      box2:compose(0, 0, check2)
42
43      local cb2 = canvas:new(100, 30)
44      cb2:attrFont('Verdana', 20, nil)
45      cb2:drawText(0, 0, "Escolha 2")
46
47      local box3 = canvas:new(30, 30)
48      box3:drawRect("frame", 0, 0, 30, 30)
49
50      local cb3 = canvas:new(100, 30)
51      cb3:attrFont('Verdana', 20, nil)
52      cb3:drawText(0, 0, "Escolha 3")
53
54      local check3
55      if checked[3] then
56          check3 = box1:new("check.png")
57      else
58          check3 = box1:new("check_off.png")
59      end
60      check3:attrScale(30, 30)
61      box3:compose(0, 0, check3)
62
63      local box4 = canvas:new(30, 30)
64      box4:drawRect("frame", 0, 0, 30, 30)
65
66      local cb4 = canvas:new(100, 30)
67      cb4:attrFont('Verdana', 20, nil)
68      cb4:drawText(0, 0, "Escolha 4")
69
70      local check4
71      if checked[4] then
72          check4 = box1:new("check.png")
73      else
74          check4 = box1:new("check_off.png")
75      end
76      check4:attrScale(30, 30)
77      box4:compose(0, 0, check4)
78
79      local box5 = canvas:new(30, 30)
80      box5:drawRect("frame", 0, 0, 30, 30)
81
82      local cb5 = canvas:new(100, 30)
83      cb5:attrFont('Verdana', 20, nil)
84      cb5:drawText(0, 0, "Escolha 5")
```

```
85
86      local check5
87      if checked[5] then
88          check5 = box1:new("check.png")
89      else
90          check5 = box1:new("check_off.png")
91      end
92      check5:attrScale(30, 30)
93      box5:compose(0, 0, check5)
94
95      local back_b = canvas:new(100, 30)
96      back_b:attrColor(175, 175, 175, 255)
97      back_b:clear()
98      back_b:attrColor(100, 100, 100, 255)
99      back_b:drawRect("frame", 0, 0, 100, 30)
100     back_b:attrColor(0, 0, 0, 255)
101     back_b:drawText(10, 5, "<< Voltar")
102
103     local next_b = canvas:new(100, 30)
104     next_b:attrColor(175, 175, 175, 255)
105     next_b:clear()
106     next_b:attrColor(100, 100, 100, 255)
107     next_b:drawRect("frame", 0, 0, 100, 30)
108     next_b:attrColor(0, 0, 0, 255)
109     next_b:drawText(10, 5, "Avançar >>")
110
111     local bar = canvas:new(400, 10)
112     bar:attrColor(110, 110, 110, 255)
113     bar:clear()
114
115     local cursor = canvas:new(20, 20)
116     cursor:attrColor(175, 175, 175, 255)
117     cursor:drawEllipse("fill", 10, 10, 10, 10, 0, 360)
118
119     local seek = canvas:new(50, 25)
120     seek:attrFont('Verdana', 20, nil)
121     seek:drawText(0, 0, (10 * (seek_position - 1)) .. "%")
122
123     if position > 0 and position < 6 then
124         canvas:attrColor(0, 0, 0, 255)
125         canvas:drawRect("frame", 297, 260 + 35 * (position - 1) - 3, 150, 36)
126     elseif position == 6 then
127         back_b:attrColor(0, 0, 0, 255)
128         back_b:drawRect("frame", 0, 0, 100, 30)
129     elseif position == 7 then
130         next_b:attrColor(0, 0, 0, 255)
131         next_b:drawRect("frame", 0, 0, 100, 30)
132     end
133
134     canvas:compose(300, 100, questionTitle)
135     canvas:compose(300, 200, questionLabel)
136     canvas:compose(300, 260, box1)
137     canvas:compose(335, 260, cb1)
138     canvas:compose(300, 295, box2)
139     canvas:compose(335, 295, cb2)
140     canvas:compose(300, 330, box3)
141     canvas:compose(335, 330, cb3)
142     canvas:compose(300, 365, box4)
143     canvas:compose(335, 365, cb4)
144     canvas:compose(300, 400, box5)
145     canvas:compose(335, 400, cb5)
146     canvas:compose(335, 400, cb5)
147     canvas:compose(300, 460, back_b)
148     canvas:compose(410, 460, next_b)
```

```
149     canvas:compose(180, 520, bar)
150     canvas:compose(175 + 40 * (seek_position - 1), 515, cursor)
151     canvas:compose(600, 512, seek)
152     canvas:flush()
153 end
154
155 function increase_pos()
156     if position < 6 then
157         position = position + 1
158     end
159 end
160
161 function decrease_pos()
162     if position > 1 then
163         position = position -1
164     end
165     if position == 6 then
166         position = position - 1
167     end
168 end
169
170 function left()
171     if position == 7 then
172         position = position - 1
173     end
174 end
175
176 function right()
177     if position == 6 then
178         position = position + 1
179     end
180 end
181
182 function press()
183     if position > 0 and position < 6 then
184         checked[position] = not checked[position]
185     elseif position == 6 then
186         seek_left()
187     elseif position == 7 then
188         seek_right()
189     end
190 end
191
192 function seek_left()
193     if seek_position > 1 then
194         seek_position = seek_position - 1
195     end
196 end
197
198 function seek_right()
199     if seek_position < 11 then
200         seek_position = seek_position + 1
201     end
202 end
203
204 function handler(evt)
205     if evt.type == 'press' then
206         local key = evt.key
207         if key == 'ENTER' then
208             press()
209         elseif key == 'CURSOR_UP' then
210             decrease_pos()
211         elseif key == 'CURSOR_DOWN' then
212             increase_pos()
```

```
213             elseif key == 'CURSOR_LEFT' then
214                 left()
215             elseif key == 'CURSOR_RIGHT' then
216                 right()
217             end
218             redraw()
219         end
220     end
221
222     event.register(handler)
223
224     redraw()
```

Código E.10: Tela desenvolvida pelo participante 3 com a biblioteca nativa do Ginga

```
1   local checkbox_img = canvas:new("check.png")
2   local checkboxoff_img = canvas:new("check_off.png")
3
4   checkbox_img:attrScale(20, 20)
5   checkboxoff_img:attrScale(20, 20)
6
7   local focus = 1
8   local perc = 0
9
10  function create_title()
11    return canvas:new("questionario.png")
12  end
13
14  function create_checkbox(text, selected, hasfocus)
15    local cbcanvas = canvas:new(200, 30)
16    if selected then
17      cbcanvas:compose(5,5, checkbox_img)
18    else
19      cbcanvas:compose(5,5, checkboxoff_img)
20    end
21    cbcanvas:drawText(30, 5, text)
22    if hasfocus then
23      cbcanvas:drawRect("frame", 0, 0, 200, 30)
24    end
25
26    return cbcanvas
27  end
28
29  function create_button(text, hasfocus)
30    local bcanvas = canvas:new(180, 30)
31
32    bcanvas:attrColor(100, 100, 100, 255)
33    bcanvas:drawRect("fill", 0, 0, 180, 30)
34    bcanvas:attrColor(0, 0, 0, 255)
35    bcanvas:drawRect("frame", 0, 0, 180, 30)
36    bcanvas:drawText(5, 5, text)
37    if hasfocus then
38      bcanvas:drawRect("frame", 1, 1, 178, 28)
39    end
40
41    return bcanvas
42  end
43
44  function create_seek_bar()
45    local sbar = canvas:new(600, 30)
46
47    sbar:attrColor(150, 150, 150, 255)
48    sbar:drawRoundRect("fill", 15, 10, 500, 10, 0, 0)
```

```
49    sbar:attrColor(0, 0, 0, 255)
50    sbar:drawRoundRect("frame", 15, 10, 500, 10, 0, 0)
51
52    local x_pos = (perc / 100.0) * 500 + 15
53    sbar:attrColor(200, 200, 200, 255)
54    sbar:drawEllipse("fill", x_pos, 15, 10, 10, 0, 360)
55    sbar:attrColor(0, 0, 0, 255)
56    sbar:drawEllipse("arc", x_pos, 15, 10, 10, 0, 360)
57
58    sbar:drawText (540, 4, perc.."%")
59
60    return sbar
61  end
62
63  function create_components()
64    checkboxes = {}
65    for i=1,5 do
66      checkboxes[i] = false
67    end
68  end
69
70  function create_question_canvas()
71    local qcanvas = canvas:new(600,500)
72
73    local tcanvas = create_title()
74    qcanvas:compose(150, 100, tcanvas)
75
76    qcanvas:attrFont("Verdana", 26, nil)
77    qcanvas:attrColor(0, 0, 0, 255)
78    qcanvas:drawText(150,200, "Questão")
79
80    local base_y = 220
81    for i=1,5 do
82      cb = create_checkbox("Resposta "..i, checkboxes[i], focus==i)
83      base_y = base_y + 30
84      qcanvas:compose(170, base_y, cb)
85    end
86    base_y = base_y + 30
87
88    local back_button = create_button ("<< Voltar", focus == 6)
89    local next_button = create_button ("Avançar >>", focus == 7)
90    qcanvas:compose(170, base_y, back_button)
91    qcanvas:compose(370, base_y, next_button)
92    base_y = base_y + 40
93
94    local seek_bar = create_seek_bar ()
95    qcanvas:compose(100, base_y, seek_bar)
96
97    return qcanvas
98  end
99
100 function draw()
101   local qcanvas = create_question_canvas()
102   canvas:attrColor(255, 255, 255, 255)
103   canvas:clear(0,0)
104   canvas:compose(150, 50, qcanvas)
105   canvas:flush()
106 end
107
108 local checkbox_x = 120
109 local checkbox_y = 70
110 create_components()
111 draw()
112
```

```
113 function handler(evt)
114   local k = evt.key
115   local t = evt.type
116
117   if t == "press" then
118     if k == "ENTER" then
119       if focus <= 5 then
120         checkboxes[focus] = not checkboxes[focus]
121       elseif focus == 6 then
122         perc = math.max(0, perc - 10)
123       else
124         perc = math.min(100, perc + 10)
125       end
126
127     elseif k == "CURSOR_DOWN" then
128       focus = math.min(7, focus + 1)
129     elseif k == "CURSOR_UP" then
130       focus = math.max(1, focus - 1)
131     end
132
133     draw()
134   end
135 end
136 event.register(handler)
```

Código E.11: Tela desenvolvida pelo participante 4 com a biblioteca nativa do Ginga

```
1  local WIDTH, HEIGHT = canvas:attrSize()
2  local FONT_FACE, FONT_SIZE, FONT_STYLE = canvas:attrFont()
3  local CHECK_IMG = "check.png"
4  local CHECK_OFF_IMG = "check_off.png"
5  local START_X, START_Y = 200, 100
6  local X_OFFSET, Y_OFFSET = 10, 40
7
8  local check_status = {true, false, true, false, false}
9  local pos = 1
10 local button_pos = 0
11 local seek_pos = 1
12
13 function drawTitleImage()
14     local title = canvas:new("questionario.png")
15     canvas:compose(START_X, START_Y, title)
16 end
17
18 function drawCheckBoxes()
19     canvas:attrColor("white")
20     canvas:drawRect("fill", START_X, START_Y + 100 + Y_OFFSET, 200, #check_status * Y_OFFSET + 20)
21     canvas:attrFont(FONT_FACE, 20, FONT_STYLE)
22     canvas:attrColor("black")
23     for i = 1, #check_status do
24         drawCheckBox(START_X + X_OFFSET, START_Y + 100 + 20 + i * Y_OFFSET, "Escolha " .. i, check_status[i])
25     end
26     if 1 <= pos and pos <= 5 then
27         canvas:attrColor("black")
28         canvas:drawRect("frame", START_X + X_OFFSET - 5, START_Y + 100 + 15 + pos * Y_OFFSET, 150, 35)
29     end
30 end
31
32 function drawCheckBox(x, y, text, checked)
33     local img
34     if checked then
35         img = canvas:new(CHECK_IMG)
36     else
```

```
37          img = canvas:new(CHECK_OFF_IMG)
38      end
39      canvas:compose(x, y, img)
40      canvas:drawText(x + 30, y, text)
41  end
42
43  function drawButton(x, y, text, focused)
44      canvas:attrColor("#555555")
45      canvas:drawRect("fill", x, y, 130, 30)
46      canvas:attrColor("white")
47      canvas:attrFont(FONT_FACE, 20, FONT_STYLE)
48      canvas:drawText(x + 10, y, text)
49      if focused then
50          canvas:attrColor("#AAAAAA")
51      else
52          canvas:attrColor("white")
53      end
54      canvas:drawLine(x, y, x + 130, y)
55      canvas:drawLine(x, y, x, y + 30)
56      if focused then
57          canvas:attrColor("#AAAAAA")
58      else
59          canvas:attrColor("black")
60      end
61      canvas:drawLine(x + 1, y + 30, x + 130, y + 30)
62      canvas:drawLine(x + 130, y, x + 130, y + 30)
63  end
64
65  function drawButtons()
66      drawButton(START_X + X_OFFSET, START_Y + 100 + #check_status * Y_OFFSET + 20 + 50, "<< Voltar", button_pos == 1)
67      drawButton(START_X + X_OFFSET + 150, START_Y + 100 + #check_status * Y_OFFSET + 20 + 50, "Avançar >>", button_pos ==
            2)
68  end
69
70  function drawSeekBar()
71      canvas:attrColor("white")
72      canvas:drawRect("fill", 20, 540, 1070, 30)
73
74      canvas:attrColor("#555555")
75      canvas:drawRect("fill", 30, 550, (seek_pos - 1) * 100, 10)
76      canvas:attrColor("#AAAAAA")
77      canvas:drawRect("fill", 30 + (seek_pos - 1) * 100, 550, (10 - seek_pos + 1) * 100, 10)
78      canvas:attrColor("black")
79      canvas:drawRect("frame", 30, 550, 1000, 10)
80
81      canvas:attrColor("#555555")
82      canvas:drawEllipse("fill", 30 + (seek_pos - 1) * 100, 555, 10, 10, 0, 360)
83      canvas:attrColor("black")
84      canvas:drawEllipse("arc", 30 + (seek_pos - 1) * 100, 555, 10, 10, 0, 360)
85
86      canvas:attrColor("black")
87      canvas:attrFont(FONT_FACE, 20, FONT_STYLE)
88      canvas:drawText(1040, 543, ((seek_pos - 1) * 10) .. '%')
89  end
90
91  function drawAll()
92
93      canvas:attrColor("white")
94      canvas:drawRect("fill", 0, 0, WIDTH, HEIGHT)
95
96      drawTitleImage()
97
98      canvas:attrColor("black")
99      canvas:attrFont(FONT_FACE, 30, FONT_STYLE)
```

```
100     canvas:drawText(START_X, START_Y + 100, "Questão")
101
102     drawCheckBoxes()
103
104     drawButtons()
105
106     drawSeekBar()
107
108     canvas:flush()
109 end
110
111 function moveUp()
112     if pos > 1 then
113         pos = pos - 1
114         drawCheckBoxes()
115         if button_pos ~= 0 then
116             button_pos = 0
117             drawButtons()
118         end
119     end
120 end
121
122 function moveDown()
123     if pos < 6 then
124         pos = pos + 1
125         drawCheckBoxes()
126         if pos == 6 then
127             button_pos = 1
128             drawButtons()
129         end
130     end
131 end
132
133 function moveLeft()
134     if button_pos > 1 and button_pos <= 2 then
135         button_pos = button_pos - 1
136         drawButtons()
137     end
138 end
139
140 function moveRight()
141     if button_pos >= 1 and button_pos < 2 then
142         button_pos = button_pos + 1
143         drawButtons()
144     end
145 end
146
147 function check()
148     check_status[pos] = not check_status[pos]
149     drawCheckBoxes()
150 end
151
152 function moveSeekBar()
153     if button_pos == 1 then
154         seekLeft()
155     elseif button_pos == 2 then
156         seekRight()
157     end
158 end
159
160 function seekLeft()
161     if seek_pos > 1 then
162         seek_pos = seek_pos - 1
163         drawSeekBar()
```

```
164     end
165 end
166 
167 function seekRight()
168     if seek_pos < 11 then
169         seek_pos = seek_pos + 1
170         drawSeekBar()
171     end
172 end
173 
174 function handler(evt)
175     if evt.type == "press" then
176         if evt.key == "CURSOR_UP" then
177             moveUp()
178         elseif evt.key == "CURSOR_DOWN" then
179             moveDown()
180         elseif evt.key == "ENTER" then
181             if 1 <= pos and pos <= 5 then
182                 check()
183             else
184                 moveSeekBar()
185             end
186         elseif evt.key == "CURSOR_LEFT" then
187             moveLeft()
188         elseif evt.key == "CURSOR_RIGHT" then
189             moveRight()
190         end
191     end
192     canvas:flush()
193 end
194 
195 function main()
196     drawAll()
197     event.register(handler)
198 end
199 
200 main()
```