Pós-Graduação em Ciência da Computação

AUGUSTO CÉSAR RIBEIRO DA SILVA

TESTE DE GESTOS: UMA ANÁLISE DA ROTAÇÃO RETRATO E PAISAGEM

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2018

Augusto César Ribeiro da Silva

Teste de Gestos: Uma Análise da Rotação Retrato e Paisagem

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação, área de concentração em Engenharia de Software, do Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco.

ORIENTADOR: **Prof. Dr. Juliano Manabu Iyoda**

RECIFE
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586t Silva, Augusto César Ribeiro da
Teste de gestos: uma análise da rotação retrato e paisagem / Augusto César Ribeiro da Silva. – 2018.
110 f.: il., fig., tab.

Orientador: Juliano Manabu Iyoda.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências e apêndices.

1. Engenharia de software. 2. Teste de software. I. Iyoda, Juliano Manabu (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2018-063

Augusto César Ribeiro da Silva

Teste de Gestos: Uma Análise da Rotação Retrato e Paisagem

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação

Aprovado em: 02/03/2018.

BANCA EXAMINADORA

_____

Prof. Dr. Alexandre Cabral Mota
Centro de Informática / UFPE

_____

Prof. Dr, Wilkerson de Lucena Andrade
Departamento de Sistemas e Computação / UFCG

_____

Prof. Dr. Juliano Manabu Iyoda
Centro de Informática / UFPE
(**Orientador**)

À minha família e aos meus amigos da Secretária de Tecnologia da Informação – STI-Univasf

AGRADECIMENTOS

Agradeço ao meu orientador, Prof. Dr. Juliano Manabu Iyoda, por ter acreditado em mim e por todas as dicas, orientações, paciência e amizade.

Aos meus pais, Geraldo de Arruda Silva e Ivanilda Mendes Ribeiro Silva, que me deram educação suficiente para que eu pudesse chegar até aqui.

À minha irmã, Ana Paula Ribeiro Silva, que mesmo distante, sempre se preocupou comigo.

À minha esposa Gilda Clemente Xavier, por acreditar sempre em mim, até nas horas que nem eu mesmo estava acreditando e por toda sua paciência e dedicação durante todo esse tempo em que a prioridade era sempre "o mestrado".

À minha enteada, "FILHA", Giulia Xavier Rodrigues, pela paciência de me aguentar nas horas de estresse.

Ao meu sogro e minha sogra, Isaias Francisco Xavier e Maria Clemente Xavier, que me acolheram em sua casa, como um filho.

Ao meu grande amigo e concunhado, Geraldo Henrique dos Santos Medeiros (in memoriam), Jaqueline Clemente Xavier e sua filha Ísis Xavier Medeiros, por me receberem em seu lar, me fazendo se sentir em minha própria casa e também parte da família.

À Ozilma Clemente Xavier, que por vezes resolveu para mim, toda a parte burocrática.

Ao Ricardo Argenton Ramos, que tanto me ajudou no início dessa jornada e que sempre se mostrou a disposição para o que fosse preciso.

Aos amigos, Roberto Tenorio, Márcia Macedo e João Sedraz, que me ajudaram em várias discussões e estudos durante essa caminhada.

Agradeço também a todos meus amigos da Secretária de Tecnologia da Informação da Univasf que tiveram que trabalhar muito mais, para que eu pudesse passar um período de dedicação exclusiva ao mestrado. Em especial ao Gustavo Michael Camilo Sousa, que me incentivou e ajudou por demais, sem o qual não teria conseguido. Ao Valtency Remigio Souto, pelo apoio na codificação. Ao Thiago Aurélio Aleixo Oliveira pelos inúmeros testes via Skype, para que tudo ocorresse bem. E ao Jonildo Martins Cordeiro e Andrey Tavares da Silva pelo apoio e compreensão aos momentos que tive que me ausentar.

RESUMO

Durante os últimos anos, o mundo tem observado um crescimento extraordinário no campo das tecnologias móveis. E isso fica cada vez mais evidente à medida que utilizamos esses dispositivos para as mais variadas atividades de nosso cotidiano. Paralelo a esse crescimento de penetração dos dispositivos móveis, os recursos desses aparelhos vêm avançando constantemente em termos de capacidade tecnológica e proporcionando interfaces cada vez mais elaboradas. Atualmente, essas interfaces vão muito além da interface gráfica. Novas formas de interação, como interfaces que se baseiam em gestos, são um exemplo que introduz um desafio interessante para a Engenharia de *Software*. Em particular, não sabemos como as técnicas tradicionais de Teste de *Software* se aplicam a estas novas interfaces. Este trabalho se propõe a analisar o teste de interfaces baseadas em gestos em relação à rotação Retrato e Paisagem do dispositivo para mudança de orientação da apresentação de telas. Testamos interfaces que se baseiam em gestos e utilizamos a técnica de particionamento em classe de equivalência para diminuirmos os casos de teste e chegarmos a um número viável de casos de teste. Também propomos um modelo de teste baseado em cobertura de máquina de estados e definimos uma hierarquia de dominação de cobertura combinando classes de equivalência e máquina de estados. Por fim, implementamos um simulador de um testador aleatório e analisamos o nível de cobertura alcançado. Chegamos à conclusão que testes de gesto, mesmo em gestos aparentemente simples como rotação de Retrato e Paisagem, devem seguir um planejamento prévio e uso de modelos e técnicas de geração de teste para se tornarem economicamente viáveis. Nossa simulação mostra que, aleatoriamente, não é possível chegar a uma cobertura satisfatória de forma eficiente: Nossa simulação mostra que, no critério Normal Forte, a cobertura não ultrapassou 20,31%, e 75% das amostras nem chegaram a 17,97%. No critério Normal Fraco, apesar de ter cobertura de 100%, em um tempo mais que suficiente para cobrir 100%, 25% das amostras nem ultrapassam 82%. No critério Todos os Estados, a máxima cobertura alcançada também foi 100%, porém mais uma vez, em um tempo que tem 100% de cobertura, 25% das amostras nem chegaram a 84%. E, no critério Todas as Transições, a cobertura máxima foi 92%, porém, em um tempo mais que suficiente para termos 100%, a cobertura máxima foi de 78%, sendo que 75% das amostras não chegaram nem a 67% de cobertura.

Palavras-chave: Teste de *software*. Dispositivos móveis. Interface do usuário. Teste de gestos. Retrato e Paisagem.

ABSTRACT

Over the last few years, the world has seen an extraordinary growth in the field of mobile technologies. And this has become increasingly evident as we use these devices for the most varied activities in our daily lives. Parallel to this growth of mobile devices penetration, the resources of these devices are constantly advancing in terms of technological capacity and providing interfaces increasingly elaborated. Currently, these interfaces go far beyond the Graphical User Interface (GUI). New forms of interaction, such as interfaces that are based on gestures, are an example that introduces an interesting challenge for Software Engineering. In particular, we do not know how the classical techniques in Software Testing can be applied to these new interfaces. This work proposes to analyze the gesture based interfaces test about the Portrait and Landscape device rotation to change the orientation of the screen presentation. We tested gesture based interfaces and used the equivalence class partitioning technique to decrease the number of possible test cases and reach a viable quantity. We also propose a test model based on state machine coverage and define a coverage domination hierarchy that capture the subsumes relation among equivalence classes and state machines. Finally, we implemented a random tester simulator an analyzed the level of coverage reached in the simulations. We came to the conclusion that gesture testing, even in apparently simple gestures such as Portrait and Landscape rotation must follow a previous planning, and must use the models and the test generation techniques to become economically viable. Our simulation shows that, at random, it is not possible to reach satisfactory coverage in an efficient way: in the strong normal criterion, the coverage did not exceed 20.31%, and 75% of the sample did not reach 17.97%. In the weak normal criterion, despite having 100% coverage, during a time period large enough to cover 100%, 25% of samples did not exceed 82%. In the all states criterion, the maximum coverage reached was also 100%, but again, during a simulation time large enough that could have 100% coverage, 25% of the samples did not reach 84%. And, in the all transitions criterion, the maximum coverage was 92%, but during a simulation time large enough for 100% coverage, the maximum coverage was 78%, with 75% of the samples not reaching 67% coverage.

Keywords: Software testing. Mobile devices. User interface. Gesture test. Portrait and Landscape.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

TI	*Tecnologia da Informação*
GUI	*Graphical User Interface* (Interface gráfica do usuário)
IEEE	*Institute of Eletrical and Eletronics Engineers*
VV&T	Verificação, Validação e Teste
SUT	*System Under Test* (Sistema em teste)
TBM	Teste Baseado em Modelos
DS	*Distinguishing Sequence*
UIO	*Unique-Input-Output*
API	*Application Programming Interface*

SUMÁRIO

1 INTRODUÇÃO

Esta seção descreve a introdução deste trabalho, mostrando o contexto, a questão de pesquisa, a solução proposta, bem como suas contribuições. E, por fim, é apresentada a estrutura desta dissertação em seções.

Durante os últimos anos, o mundo tem observado um crescimento extraordinário no campo das tecnologias móveis. Em 2011 a venda de smartphones superou as vendas de computadores pessoais, comercializando 487,7 milhões de unidades, mostrando um aumento de mais de 62% apenas de 2010 a 2011 (CANALYS, 2012).

De acordo com uma empresa líder em pesquisas de Tecnologia da Informação (TI), as vendas globais de smartphones para usuários finais totalizaram 380 milhões de unidades somente no primeiro trimestre de 2017, um aumento de 9,1% em relação ao primeiro trimestre de 2016 (GARTNER, 2017). E isso fica cada vez mais evidente à medida que utilizamos esses dispositivos para as mais variadas atividades de nosso cotidiano. Paralelo a esse crescimento de penetração dos dispositivos móveis, os recursos desses aparelhos vêm avançando constantemente, em termos de capacidade tecnológica, e proporcionando interfaces gráficas cada vez mais elaboradas. Atualmente, essas interfaces vão muito além da interface gráfica do usuário (GUI – Graphical User Interface). Toques na tela (incluindo múltiplos toques), rotação, movimentos suaves e bruscos (shakes), comando de voz e gestos são todos exemplos de novas formas de interação.

Dentre essas novas formas de interação, as interfaces que se baseiam em gestos para mudança na orientação da apresentação de telas (retrato ou paisagem) é um exemplo que introduz um desafio interessante para a Engenharia de *Software*. Um defeito que pode ser observado nos aplicativos ao mudarem de orientação é a perda do estado da tela anterior (prejudicando a experiência do usuário) ou, pior, do estado interno do aplicativo. Em testes exploratórios informais, testemunhamos defeitos destes dois tipos em um tocador de músicas via internet e em uma calculadora. Uma investigação importante é saber como as técnicas tradicionais de teste de *software* podem ser aplicadas a este novo domínio e, caso contrário, quais dificuldades e necessidades de evolução das técnicas foram identificadas.

- Problema de pesquisa: as técnicas tradicionais de teste modelam adequadamente interações gestuais?
- Objetivos: investigar se a modelagem dos gestos de rotação retrato-paisagem podem ser feitas usando técnicas clássicas como Classes de Equivalência e cobertura de Máquina de Estados.

Não houve um critério rígido para a escolha de Classes de Equivalência e cobertura de Máquina de Estados em detrimento às demais técnicas clássicas. Entretanto, como ambas são técnicas muito populares (em comparação, por exemplo, a Tabelas de Decisão), foram as escolhidas para este estudo.

Neste trabalho, fizemos quatro análises distintas, mas complementares. Primeiro, analisamos como a técnica de classes de equivalência, uma técnica tradicional de teste de *software*, poderia ser utilizada para modelar os casos de testes para rotação retrato e paisagem. Depois, propomos a modelagem através de uma máquina de estados para gerar testes de rotação baseado na cobertura da máquina. Então definimos uma hierarquia de dominação de cobertura combinando classes de equivalência com máquina de estados. Por fim, implementamos um simulador de teste de rotação aleatório e medimos, nestas simulações, o nível de cobertura alcançado. Verificamos que dificilmente um testador manual executando de forma aleatória testes de retrato e paisagem atingiria uma cobertura eficientemente se não for guiado pelas técnicas. Nossa simulação mostra que no critério Normal Forte, a cobertura não ultrapassou 20,31%, e 75% das amostras nem chegaram a 17,97%. No critério Normal Fraco, apesar de ter cobertura de 100%, em um tempo mais que suficiente para cobrir 100%, 25% das amostras nem ultrapassam 82%. No critério Todos os Estados, a máxima cobertura alcançada também foi 100%, porém mais uma vez, em um tempo que tem 100% de cobertura, 25% das amostras nem chegaram a 84%. E, no critério Todas as Transições, a cobertura máxima foi 92%, porém, em um tempo mais que suficiente para termos 100%, a cobertura máxima foi de 78%, sendo que 75% das amostras não chegaram nem a 67% de cobertura.

Diversas pesquisas podem ser encontradas na área de teste de aplicações para dispositivos móveis, mas muitas se preocupam apenas com as interfaces gráficas. Em Amalfitano et al. (2012), podemos ver uma ferramenta que implementa uma técnica automatizada para testar aplicações Android via sua interface gráfica, a AndroidRipper. Delamaro, Vincenzi e Maldonato (2006) expõem uma estratégia de apoio ao teste de cobertura para aplicações de dispositivos móveis de uma maneira que o programa possa ser testado não só em emuladores, mas também no próprio dispositivo móvel. Já Memon, Soffa e Pollack (2001) revelam novos critérios de

cobertura, intra-componentes e inter-componentes, que auxiliam na determinação de quando uma interface gráfica foi testada adequadamente. Em Hao (2014), é apresentada a criação e implementação da PUMA, uma estrutura para análise dinâmica de aplicações móveis em larga escala. Hu e Neamtiu (2011) relatam um estudo de erros para compreender a natureza e possíveis soluções para erros em aplicações móveis e construir uma estrutura de testes automatizada para aplicações Android. Yang, Prasad e Xie (2013) propõem uma nova abordagem para engenharia reversa de modelos de interface gráficas de aplicações móveis. E, por fim, Barboza (2016) mostra o desenvolvimento e utilização de um braço robótico articulado e de uma aplicação Android de controle e testes.

No entanto, percebe-se que a academia ainda não tem dada a atenção necessária para testes de interfaces mais elaboradas, visto que poucos trabalhos levam em consideração as novas interfaces baseadas em gestos, em particular, a mudança de orientação (retrato/paisagem).

1.1 Contribuições

Dentre as principais contribuições desta dissertação, temos:

- A modelagem da interface gestual de rotação retrato e paisagem em termos de classes de equivalência, que tornam viável a execução de teste de tais interfaces de forma disciplinada;
- A modelagem da interface gestual de rotação retrato e paisagem em termos de uma máquina de estados, também produzindo testes de forma disciplinada;
- A definição de uma hierarquia de dominação de cobertura combinando os modelos de classes de equivalência com máquina de estados;
- A implementação de um simulador de execução aleatória de testes de rotação retrato e paisagem;
- A análise, através da execução de milhares de simulações, de que uma execução aleatória dificilmente consegue cobrir bem os testes produzidos pelos modelos.

1.1 Estrutura da Dissertação

As seções subsequentes deste documento estão organizados da seguinte forma:

- **Seção 2: Fundamentação teórica –** Esta seção apresenta os conceitos fundamentais de teste necessários para o entendimento do trabalho e que serão utilizados nas seções posteriores.

- **Seção 3: Modelagem -** Esta seção descreve as modelagens das classes de equivalência bem como da máquina de estados proposta para representar o *smartphone* mudando de retrato para paisagem.
- **Seção 4: Critérios de cobertura -** Esta seção mostra como os critérios de cobertura derivados dos resultados da seção 3 são organizados em uma hierarquia de dominação.
- **Seção 5: Simulações -** Esta seção descreve o desenvolvimento de um simulador de execução aleatória de rotações retrato e paisagem. O simulador foi implementado na linguagem de programação Java, para desempenhar o papel de um testador humano que gere testes aleatoriamente para um sistema em teste. Os resultados de milhares de simulações mostram a importância dos modelos desenvolvidos.
- **Seção 6: Trabalhos relacionados -** Esta seção expõe trabalhos relacionados com a dissertação.
- **Seção 7: Conclusão -** Esta seção sintetiza o trabalho realizado, apresentando as dificuldades encontradas, contribuições do trabalho, bem como suas limitações. Finaliza propondo trabalhos futuros que podem dar continuidade a esta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção tem como principal objetivo proporcionar a fundamentação mínima necessária sobre conceitos que serão abordados durante o documento. São expostos e debatidos conceitos básicos sobre teste de *software*, bem como algumas técnicas e ferramentas.

2.1 Teste de Software

O processo de desenvolvimento de *software* é composto por várias atividades nas quais, apesar da utilização de diversas técnicas, métodos e ferramenta, erros ainda podem ser encontrados (DELAMARO; MALDONATO; JINO, 2007). E com isso o *software* pode não funcionar de maneira esperada.

Em virtude disso, novas atividades, associadas com o nome de garantia de qualidade de *software* têm sido incorporadas ao processo de desenvolvimento. Entre essas novas atividades estão as atividades de VV&T (Verificação, Validação e Teste) (PRESSMAN, 2011), que suplementam os demais recursos do desenvolvimento com a finalidade de garantir que o *software* respeite as especificações e cumpra as necessidades dos clientes, produzindo assim maior qualidade no *software*.

Devido ao escopo do trabalho, entre as atividades de VV&T, iremos nos ater apenas ao teste de *software*, que segundo Myers (2004) é o processo de executar o *software* com a intenção de encontrar erros. Complementando a definição, o teste de *software* é a maneira mais comum de verificar se um *software* atende suas especificações e realiza o que o cliente deseja (SOMMERVILLE, 2003). A atividade de testes é um componente vital no processo de qualidade de *software* (BURNSTEIN, 2003), sendo um dos elementos para disponibilizar evidências da confiabilidade de *software* suplementando outras atividades.

2.1.1 Conceitos e definições importantes

Radatz, Geraci e Katki (1990) definem no padrão IEEE 610.12-1990, um glossário de terminologia da Engenharia de *Software* com intenção de evitar ambiguidades e desentendimentos de interpretação com termos de outras áreas da computação ou mesmo de outras ciências. No referido padrão os seguintes termos são distinguidos:

a) **Engano** (Mistake): Ação humana que produz um resultado incorreto (Ex.: ação incorreta realizada pelo desenvolvedor ou operador);
b) **Falha** (Fail): Refere-se a incapacidade do *software* em realizar uma função requisitada, como por exemplo, terminação anormal ou restrição temporal violada;
c) **Defeito** (Fault): Passo, processo ou definição de dado incorreto em um programa (Ex.: uma restrição incorreta no programa);
d) **Erro** (Error): É o estado intermediário entre um defeito e uma falha. Pode ter como resultado uma falha, se difundida até a saída (RADATZ; GERACI; KATKI, 1990).

Com isso podemos verificar que, segundo IEEE (Institute of Eletrical and Eletronics Engineers), um defeito gera um erro. E que esse erro, se for propagado até a saída, gera uma falha. Teste de *software* investiga o *software* com intenção de encontrar o maior número possível de falhas, motivadas por erros, em consequência de um ou mais defeitos.

Outras terminologias importantes da área de teste são:

a) **Sistema em teste ou SUT** (System Under Test): Refere-se ao sistema, subsistema ou componente de *software* que está sendo testado;
b) **Domínio de entrada**: Conjunto de todos os valores possíveis como entrada na execução do SUT;
c) **Domínio de saída**: Conjunto de todos os possíveis resultados gerados pela execução do SUT;
d) **Dado de teste**: Corresponde a um elemento que pertence ao domínio de entrada do *software*;
e) **Caso de teste** (Test Case): É um par que possui um dado de teste, como entrada e o resultado esperado como saída para a referida entrada. Um par (entrada, saída esperada) de um SUT;
f) **Conjunto de teste ou Suíte de teste** (Test suite ou Test case suite): Conjunto de todos os casos de teste utilizados em uma atividade de teste. (RADATZ; GERACI; KATKI, 1990) (DELAMARO; MALDONATO; JINO, 2007)

2.1.2 Fases de teste de software

A atividade de teste é bastante complexa e, por isso, o seu planejamento é dividido em fases ou níveis distintos (ROCHA; MALDONATO; WEBER, 2001) (DELAMARO; MALDONATO; JINO, 2007). Essas fases são:

a) **Teste de unidade**: também conhecidos como testes unitários. Tem como ênfase as menores unidades de programa, que podem ser métodos, classes, procedimentos ou funções. Nessa

fase, procura-se provocar falhas ocasionadas por defeitos de lógica e de implementação. Nesse contexto, espera-se encontrar erros relacionados a algoritmos incorretos ou mal implementados, estruturas de dados incorretas ou simples erros de programação. Como cada unidade é testada individualmente, esse teste pode ser realizado à medida que a implementação de cada uma dessas unidades vai sendo terminada. Conforme o padrão IEEE 610.12-1990 (RADATZ; GERACI; KATKI, 1990), uma unidade é o componente de *software* que não pode ser subdividido. Podemos notar então que no paradigma de programação procedural, há um consenso de que a menor unidade a ser executado é um procedimento ou sub-rotina. Porém, no paradigma orientado a objetos, verificamos que há a interpretação de que essa unidade é uma classe, defendida por ser a classe a menor unidade funcional da orientação a objetos, ou seja, que não depende de outras para executar sozinha (PERRY; KAISER, 1990) (BINDER, 1999). Ou ainda a interpretação de que já essa unidade é o método, já que a classe é composta por métodos e atributos, e que os métodos colaboram entre si para executar uma função gerando assim uma integração a ser testada (COLANZI, 1999) (VINCENZI, 2004);

b) **Teste de integração**: o foco dessa fase é o teste das interfaces entre as unidades. À medida que as diversas partes do *software* são colocadas para trabalhar juntas, é preciso verificar se a interpretação entre elas funcionam de maneira adequada e não leva a erros (DELAMARO; MALDONATO; JINO, 2007). Essa fase testa as interfaces entre os diversos módulos quando estes são integrados para compor a estrutura do *software*. É interessante destacar que a constatação de erros durante a fase de teste de integração pode levar a alterações de unidades já testadas, o que necessitaria de novos testes de unidade e integração. Logo é imprescindível que estes testes sejam realizados o quanto antes visando uma diminuição de custos dos testes e melhorando assim a qualidade do processo de desenvolvimento;

c) **Teste de sistemas**: é uma fase em que diferentes tipos de testes são realizados, com o intuito de exercitar todo o *software* (PRESSMAN, 2011). O foco dessa fase é testar o *software* totalmente integrado, verificando assim capacidades e características presentes somente no sistema inteiro. Tem como objetivo investigar se as funcionalidades especificadas nos documentos de requisitos estão todas implementadas de forma correta. Aspectos de correção, completude e coerência devem ser analisados, bem como requisitos não funcionais. Nessa fase, é averiguado o funcionamento do *software* em relação ao ambiente para qual foi projetado, confirmando o comportamento esperado (funções, desempenho, segurança, etc.) e verificando se as interações com esse ambiente (*hardware*, sistema operacional, etc.) e outros externos (banco de dados, serviços, etc.) que estão direta ou indiretamente relacionados ao sistema não provocaram o surgimento de erros. Essa fase avalia o *software* por meio da

utilização, como se fosse um usuário final. Dessa forma, os testes são realizados nos mesmos ambientes, com os mesmos dados de entrada e condições que um usuário teria na utilização do *software*;

d) **Teste de aceitação**: realizados geralmente por um restrito grupo de usuários finais do sistema. Esses usuários simulam operações de rotina do sistema verificando se o comportamento do *software* está de acordo como o solicitado;

e) **Teste de regressão**: quando uma nova versão de um *software* ou componente não desempenha mais corretamente uma funcionalidade que deveria ter sido mantida, diz-se que a nova versão regrediu em relação às versões anteriores. A não regressão (ou seja, a conservação de funcionalidades) é um requisito básico de qualidade. As atividades de teste que se concentram nos problemas de regressão são chamadas de teste de (não) regressão. Normalmente o não é omitido e diz-se teste de regressão (PEZZÈ; YOUNG, 2008). Teste de regressão é o reteste seletivo do *software* ou componente para verificar se alterações não causaram efeitos não desejados e se o *software* ou componente ainda satisfaz os requisitos especificados (RADATZ; GERACI; KATKI, 1990). O teste de regressão pode ser aplicado em qualquer fase de teste e é bastante importante para redução de "efeitos colaterais" consistindo na aplicação de todos os testes que já foram realizados nas versões ou ciclos de testes anteriores a cada nova versão do *software* ou a cada ciclo do desenvolvimento.

Cada uma dessas fases proporciona a cobertura de diferentes tipos de erros por envolver aspectos distintos do processo de desenvolvimento de *software*. Independente da fase de teste, as execuções das atividades devem ser realizadas de maneira controlada e rigorosa, conforme as necessidades inerentes ao SUT. E o processo de execução dos testes é realizado em quatro etapas distintas: planejamento, projeto de caso de teste, execução e avaliação dos resultados (BEIZER, 1990) (MYERS, 2004) (DELAMARO; MALDONATO; JINO, 2007).

2.1.3 Abordagens de teste de software

Nesta seção são expostas algumas das abordagens de teste de *software*, que darão base para um melhor entendimento do trabalho realizado.

2.1.3.1 Teste estrutural

A abordagem estrutural também é conhecida como teste de caixa-branca (White box testing) e usa a perspectiva interna, utilizando informações da estrutura e detalhes do código fonte do *software* para modelar os casos de teste (PRESSMAN, 2011). Sua funcionalidade é

avaliar particularidades comportamentais de partes básicas do *software* (MYERS, 2004). Estes testes são elaborados de uma averiguação dos caminhos lógicos possíveis de serem executados a fim de saber o comportamento interno do *software*.

Os principais critérios de cobertura da abordagem estrutural são classificados em relação ao fluxo de dados, fluxo de controle, ou complexidade (FRANKL, 1987) (WEYUKER; WEISS; HAMLET, 1991) (SUGETA, 1999) (PRESSMAN, 2001). Critérios baseados em fluxo de dados analisam associações de pontos do *software* onde são definidas variáveis e pontos onde são usadas essas variáveis. Já os critérios baseados em fluxo de controle utilizam informações do controle de execução, como repetições e desvios para determinar que testes são necessários. E, por fim, os critérios baseados em complexidade utilizam informações de complexidade do *software* para derivar os requisitos de teste.

Devido a essa abordagem levar em consideração a implementação (código fonte) do *software* é possível a identificação de erros de projeto, erro de lógica e pressuposições incorretas, bem como teste de partes do *software* que não estão nas especificações. Por outro lado, não há como testar especificações que não foram implementadas. Outra desvantagem é que os testes de caixa-branca são considerados mais complexos, por depender de lógicas não triviais, paradigmas e linguagens de programação, necessitando de um testador com maior conhecimento. E, por fim, outra desvantagem dessa abordagem é que só será possível gerar casos de teste após a implementação do sistema está finalizada.

2.1.3.2 Teste funcional

A abordagem funcional, também conhecida como teste de caixa-preta (black box testing) ou ainda teste comportamental (behavior testing) concentra-se no comportamento do *software* através da investigação de seu real funcionamento. É chamada de teste de caixa-preta, pois nessa abordgem são ignorados os mecanismos internos do *software*, considerando-o uma caixa fechada, onde a avaliação é feita pela verificação das saídas geradas em repostas às entradas selecionadas e condições de execução.

Para estabelecer os requisitos e casos de testes são utilizadas as especificações do *software* (BEIZER, 1990). O objetivo do teste funcional é encontrar possíveis divergências entre essas especificações e o comportamento real do *software* (OSTRAND; BALCER, 1988). Nota-se então a desvantagem de uma alta dependência de uma clara e correta definição e interpretação dessas especificações. Outra desvantagem é a impossibilidade de uma quantificação das

atividades de teste, não garantindo que questões específicas ou mesmo críticas tinham sido verificadas (MYERS, 2004). Como é impossível conhecer as partes do código que foram verificadas, casos de teste redundantes podem ser gerados, visto que não há como comparar, em termos de cobertura de código, casos de teste com o mesmo objetivo. E, por fim, outra desvantagem é que quando essa abordagem é utilizada em sistemas complexos, a grande quantidade de entradas resulta numa explosão de casos de teste, fazendo com que apenas um pequeno número dessas entradas seja realmente testado. Porém, com a utilização do teste de caixa-preta há a vantagem de não precisar de um testador com conhecimento específico na linguagem utilizada na construção do *software*. E ainda pode-se gerar testes mesmo antes da implementação estar completa. Além disso, caso haja modificações na implementação sem mudança de funcionalidade, os casos de teste já gerados continuam inalterados e úteis. E, outra vantagem, é que testes podem ser realizados mesmo quando não há acesso a nenhuma documentação ou implementação do *software* como é o caso de sistemas legados ou alguns *softwares* de terceiros, como é o caso de aplicativos para smartphones.

As principais categorias de erros encontrados através dessa abordagem são: (i) funções incorretas ou inexistentes; (ii) erros de interfaces; (iii) erros em estrutura de dados ou de acesso a dados externos; (iv) erros de comportamento ou desempenho; e (v) erros de inicialização e término (PRESSMAN, 2011).

Os critérios de teste da abordagem funcional, em geral, buscam limitar ou dividir o domínio de entrada do SUT, com o intuito de facilitar a extração de casos de teste efetivos para descoberta de erros. Um dos principais critérios de teste funcional é o particionamento em classes de equivalência (apesar de essa técnica também ser aplicável em teste de caixa-branca, não é isto que acontece comumente na prática).

2.1.4 Particionamento em classes de equivalência

O teste exaustivo, que testa todos os valores do domínio de entrada, é, em geral, impossível de ser realizado. Por isso, o testador deve escolher um subconjunto, entre todas as possíveis entradas, para tornar a atividade de teste viável, sendo que esse subconjunto deve ter uma alta probabilidade de encontrar falhas (MYERS, 2004).

O critério de particionamento em classe de equivalência também conhecido como particionamento de equivalência busca dividir o domínio de entrada, de acordo com a especificação, em classes de dados (classes de equivalência) válidas e inválidas, de onde os casos de testes serão gerados (PRESSMAN, 2011). Alguns autores consideram também o domínio de

saída, procurando alternativas que possam descobrir classes de equivalência no domínio de entrada (COPELAND, 2004) (ROPER, 1994). A divisão em classes visa gerar casos de teste para percorrer diversas classes de erros e simultaneamente diminuir os casos de teste necessários.

O particionamento do domínio de entrada deve ser realizado de maneira que cada elemento de uma classe faça o SUT ter o mesmo comportamento de qualquer outro elemento dessa mesma classe (MYERS, 2004). Ainda segundo Myers (2004), apesar de esse processo de definição de classes de equivalência ser um processo heurístico, algumas diretrizes podem ser seguidas:

a) Se a condição de entrada é especificada por um intervalo, identificam-se uma classe válida e duas classes inválidas. Ex.: Dado uma entrada n que é um número entre 1 e 10, identificam-se uma classe válida contendo os números entre 1 e 10 e duas classes inválidas contendo os números menores que 1 e os números maiores que 10;
b) Se uma determinada quantidade de itens de entrada é especificada, identificam-se uma classe válida e duas inválidas. Ex.: Se, na entrada, o total de itens varia de 1 a 6, identifica-se uma classe válida (quantidade de itens entre 1 e 6) e duas inválidas: (nenhum item e quantidades de itens maior que 6);
c) Se a condição de entrada é um conjunto de valores que geram comportamentos diferentes no SUT, identificam-se uma classe válida para cada elemento do conjunto e uma classe inválida. Ex.: Se uma entrada deve receber “Professor” ou “Estudante” ou “Técnico”. Identificam-se uma classe válida para cada um dos valores Professor, Estudante e Técnico, ou seja, são geradas três classes válidas. E uma classe inválida para os demais valores;
d) Se uma condição de entrada especifica uma situação do tipo “deve ser”, identificam-se uma classe válida e uma classe inválida. Ex.: O primeiro caractere de uma variável deve ser uma letra. Identificam-se uma classe válida (variáveis cujo primeiro caractere é uma letra) e uma classe inválida (variáveis cujo primeiro caractere não é uma letra);
e) Se houver algum motivo para achar que o *software* não trata elementos de uma mesma classe de equivalência de forma igual, divida a classe de equivalência em classes ainda menores.

Os tipos de particionamentos em classes de equivalência podem ser divididos em normal: onde são consideradas apenas as classes de equivalência com valores de entrada válidos, ou robusto: onde são consideradas as classes de equivalência com valores de entrada válidos e inválidos (JORGENSEN, 1995).

Para melhor entendermos os tipos de particionamentos em classe de equivalência, iremos exemplificar através de uma função F, com duas variáveis x1 e x2. Quando F é implementada

como um programa de computador, as variáveis de entrada x1 e x2 terão as seguintes especificações de limites e intervalos entre limites:

$$a \leq x1 \leq d, \text{ com intervalos } [a,b), [b,c), [c,d]$$

$$e \leq x2 \leq g, \text{ com intervalos } [e,f), [f,g]$$

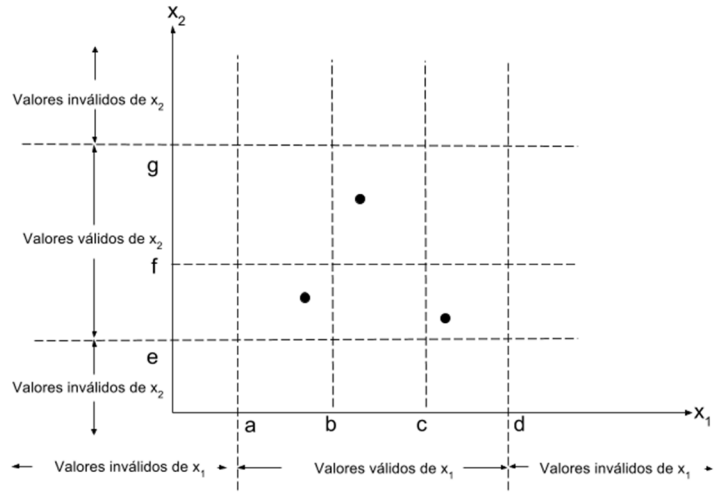
onde o colchete indica que o valor ao seu lado está incluído e o parêntese indica que o valor ao seu lado não está incluído. Cada intervalo contém valores que são equivalentes entre si. Este exemplo será utilizado nas seções que seguem

Os tipos de particionamentos em classe de equivalência podem ser classificados em Normal Fraco, Normal Forte, Robusto Fraco e Robusto Forte.

2.1.4.1 Normal Fraco

Nesse tipo de particionamento, os casos de teste são criados levando em consideração apenas um valor de cada classe de equivalência (intervalo). A Figura 1 mostra a representação dos três casos de teste. O primeiro caso de teste, representado pelo ponto no retângulo inferior esquerdo, refere-se ao valor de x1 na classe [a,b) e ao valor de x2 na classe [e,f). O segundo caso de teste, representado pelo ponto no retângulo superior do meio, refere-se ao valor de x1 na classe [b,c) e ao valor de x2 na classe [f,g]. E, por fim, o terceiro caso de teste, que possui o valor de x1 na classe [c,d], completando assim pelo menos um valor de cada classe de equivalência, e por isso, poderia ser em qualquer retângulo de valores válidos da direita (na figura está representado pelo retângulo inferior direito). O número de casos de teste de equivalência normal fraca será sempre o mesmo número de classes na partição com maior número de subconjuntos. O inconveniente desse tipo de particionamento é que há uma "independência de variáveis de entrada", pois quando um caso de teste de equivalência normal fraco falha, pode ser um problema com a variável x1, com a variável x2 ou uma interação entre as duas. Essa ambiguidade é a razão do nome "fraco". Este tipo de particionamento é utilizado quando há uma suposição de sempre haver "falha única" (assume-se que um erro é raramente causado como resultado de duas ou mais falhas que ocorrem simultaneamente). Logo, quando um erro raramente acontece, como em casos de teste de regressão essa pode ser uma escolha aceitável, mas quando é requerido um maior isolamento de erro, é mais indicado a utilização de um particionamento "forte", como será visto (JORGENSEN, 1995). Uma vantagem deste critério é o reduzido número de casos de testes.

Figura 1 - Casos de teste de equivalência Normal Fraco

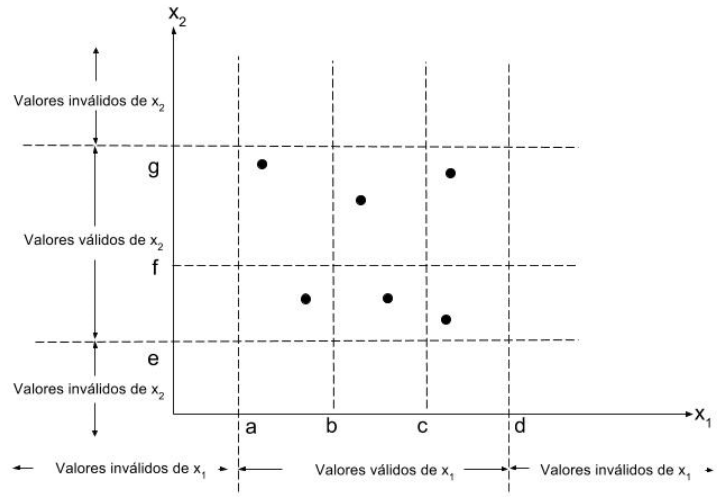


Fonte: JORGENSEN, P. C. Software testing: a craftsman's approach. CRC press, 1995.

2.1.4.2 Normal Forte

Por ser um particionamento "normal", só os valores válidos são considerados, porém o particionamento de equivalência normal forte é baseado na suposição de "múltiplas falhas", que é quando as variáveis são dependentes e que erros resultarão na combinação de falhas. Assim, são necessários casos de testes de cada elemento do produto cartesiano das classes de equivalência, como mostra a Figura 2. O produto cartesiano garante a noção de "completude" de duas formas, cobrindo todas as classes de equivalência e testando cada uma das possíveis combinações de entrada (JORGENSEN, 1995).

Figura 2 - Casos de teste de equivalência Normal Forte



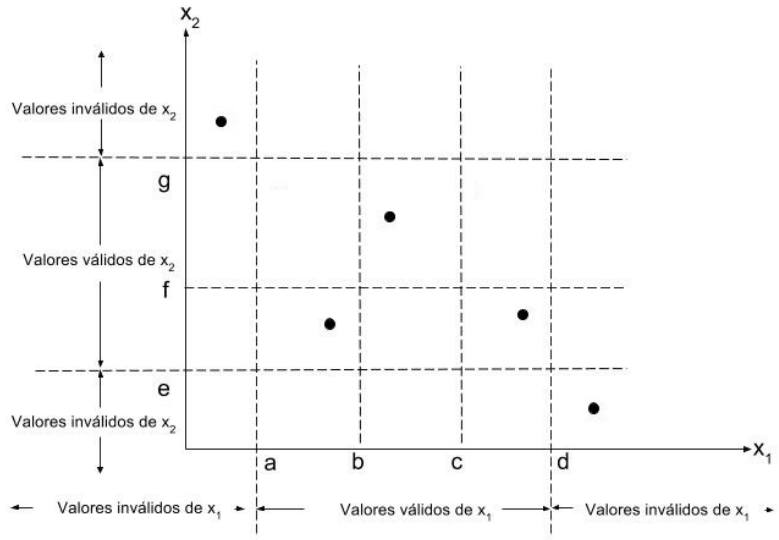
Fonte: JORGENSEN, P. C. Software testing: a craftsman's approach. CRC press, 1995.

2.1.4.3 Robusto Fraco

A palavra “robusto” na tipificação do particionamento deve-se a um aumento de abrangência dos casos de teste, pois agora é levado em consideração não somente os valores válidos, mas também os valores inválidos. E a palavra “fraco”, como já visto antes, é utilizada quando há uma suposição de “falha única”, que é quando um erro é raramente causado como resultado de duas ou mais falhas que ocorrem simultaneamente. O particionamento robusto fraco é uma extensão do particionamento normal fraco, porém levando em consideração valores inválidos, como mostra a Figura 3. Assim, para fazermos o particionamento robusto fraco devemos:

a) Para valores válidos de entradas, usarmos um valor de cada classe de equivalência (assim teremos um particionamento normal fraco);
b) Para valores inválidos de entradas, utilizar um caso de teste com um valor inválido e os demais válidos (devido a suposição de “falha única” o caso de teste deverá falhar) (JORGENSEN, 1995).

Figura 3 - Casos de teste de equivalência Robusto Fraco



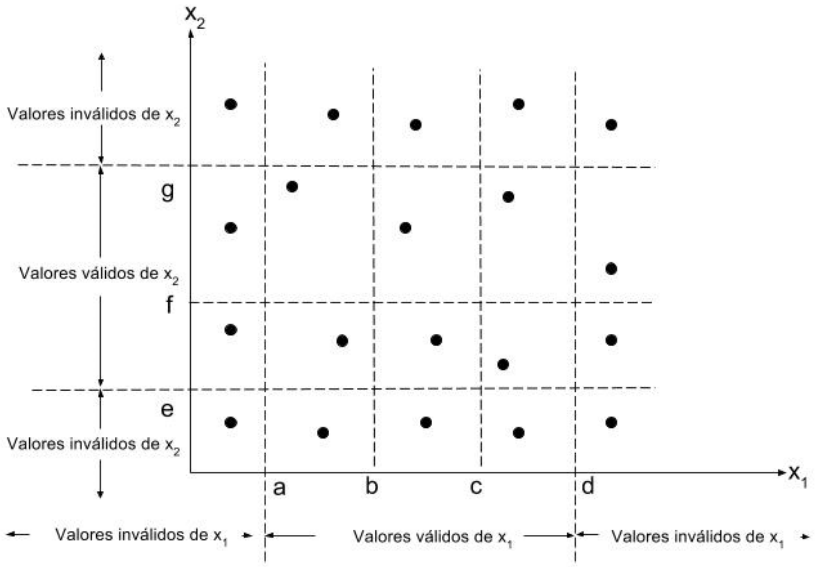
Fonte: JORGENSEN, P. C. Software testing: a craftsman’s approach. CRC press, 1995.

2.1.4.4 Robusto Forte

Assim como na tipificação do particionamento anterior, a palavra “robusto” deve-se a um aumento de abrangência dos casos de teste, que não leva em consideração apenas valores válidos, mas também os valores inválidos. Já a palavra “forte” refere-se à suposição de “múltiplas falhas” que, como mencionado anteriormente, é quando as variáveis são dependentes e que erros resultarão na combinação de falhas. Assim, são necessários casos de testes de cada

elemento do produto cartesiano das classes de equivalência, como mostra a Figura 4 (JORGENSEN, 1995).

Figura 4 - Casos de teste de equivalência Robusto Forte



Fonte: JORGENSEN, P. C. Software testing: a craftsman's approach. CRC press, 1995.

Em casos complexos, o critério de particionamento em classes de equivalência se mostra muito melhor que uma seleção aleatória de casos de teste, uma vez que reduz o domínio de entrada a um tamanho possível de ser tratado em uma atividade de teste, levando em consideração fatores importantes do sistema.

2.1.5 Máquina de Estados Finitos

Conforme Pressman (2005), uma das enumerações de categorias de produtos de *software* consiste nos sistemas reativos. O presente estudo se preocupa com um sistema reativo. Estes tipos de sistemas têm como principal aspecto a interação permanente com o ambiente externo, seja um dispositivo de entrada, um usuário, ou até mesmo outro sistema (HAREL; POLITI, 1998). Um sistema reativo é composto por um conjunto de sequências de entradas e seus eventos de saída correspondentes. Devido a esta propriedade, um tipo de modelo formal bastante interessante para a geração de testes são as Máquinas de Transições de Estados, pois são estruturas que possibilitam determinar possíveis próximas ações (ou saídas) fundamentadas no estado atual e estímulos externos (ou entradas) (ROSARIA; ROBINSON, 2000) (APFELBAUM; DOYLE, 1997). Existem alguns tipos de máquinas de transição de estados, como Statecharts, Redes de Petri e Máquina de Estados Finitos, porém vamos nos preocupar apenas com esta última que é o tipo de Máquina de Transição de Estados utilizada neste trabalho.

Uma MEF (Máquina de Estados Finitos) é classificada de acordo com o tipo de função de saída: Máquina de Moore e Máquina de Mealy. Na Máquina de Moore, as saídas são associadas a estados, os quais são ativos. Já na Máquina de Mealy, as saídas são associadas a transições e os estados são passivos. Devido ao escopo do documento, nos restringiremos apenas a Máquina de Mealy, utilizada amplamente no Teste Baseado em Modelos.

A representação chamada de Máquina de Mealy funciona da seguinte forma: o recebimento de uma entrada executa uma transição que poderá gerar uma saída. Cada transição muda o estado da MEF, de atual e para o próximo estado e', porém, é possível que o estado e' seja igual ao estado inicial e. Esse ciclo se repete até que se atinja um estado final.

De acordo com Gill (1962), uma MEF é um modelo de comportamento constituído por um número finito de estados e transições. É possível achar definições distintas para MEF que possuem algumas diferenças em termos de conceitos e funcionalidades.

Seguindo a definição formal em Gören e Ferguson (2002), uma Máquina de Estados Finitos é uma tupla $M = (\Sigma, \Delta, Q, q0, \delta, \lambda)$, onde:

$\Sigma \neq \emptyset$: um conjunto finito de símbolos de entrada,

$\Delta \neq \emptyset$: um conjunto finito de símbolos de saída,

$Q \neq \emptyset$: um conjunto finito de estados,

$q0 \in Q$: o estado inicial,

$qi \in Q$: estado de número i,

$\delta : Q\text{x}\Sigma \rightarrow Q \cup \{\emptyset\}$: a função de transição , onde $\emptyset$ denota um estado não especificado,

$\lambda : Q\text{x}\Sigma \rightarrow Q \cup \{\in\}$: a função de saída, onde $\in$ denota uma saída não especificada.

A notação $\Omega(qi)$ é empregada para simbolizar o conjunto de todas as sequências de entrada definidas a partir de qi, sendo ΩM uma maneira de simbolizar $\Omega(q0)$, correspondendo a todas as sequências de entrada válida para a MEF M.

De acordo com a utilização da MEF, existem algumas presunções que devem ser levadas em conta em relação às suas particularidades. As presunções estão relacionadas ao determinismo, a completude e aos tipos de conexões. Segundo Delamaro, Maldonato e Jino (2007), elas podem ser conforme explanadas em seguida.

Uma MEF pode ser considerada determinística se a quantidade de transições de um estado $qi \in Q$ sobre uma entrada $i \in \Sigma$ é igual a um. Ou seja, uma MEF é determinística se, a partir de um estado qualquer, qi, e para uma entrada i, a máquina segue somente uma transição

para o próximo estado. Caso contrário, a MEF é não determinística, logo, não podemos saber previamente qual transição será seguida e com isso saídas diferentes podem ser produzidas.

Se uma MEF trata todas as entradas para todos os estados, podemos dizer que ela é completamente especificada, ou completa. Caso contrário ela é parcialmente especificada ou parcial. Para máquinas parciais, quando as transições não especificadas são completadas para fins de teste, diz que ela assume uma suposição de completude.

Além da representação textual de uma MEF, que pode ser realizada através de uma simples descrição de cada um de seus elementos, como visto anteriormente em sua definição, existem mais dois outros tipos de representações utilizadas para descrever uma MEF: um formato tabular, conhecido como tabela de transições de estados e um formato gráfico, conhecido como diagrama de transições de estados.

2.1.5.1 Tabela de transições de estados

As tabelas de transições de estados são bastante oportunas para manuseio computacional e armazenamento das informações de uma MEF, pois é uma forma de esclarecer o comportamento de uma MEF em uma estrutura de dados pronta para ser empregada em aplicações de *software*. A Figura 5 mostra um exemplo de tabela de transições de estado. A primeira linha e a primeira coluna têm, respectivamente, todas as entradas possíveis e todos os estados de origem. As transições são determinadas nas intersecções entre a entrada que a dispara e seu estado de origem. Neste local da tabela consta o próximo estado, bem como a saída gerada.

Figura 5 - Representação de uma MEF em tabela de transição de estados

Estado Atual	Entradas		
	a	b	c
S_0	S_1/e	S_1/f	S_2/e
S_1	S_0/f	S_2/f	S_1/f
S_2	S_2/f	S_0/e	S_1/e

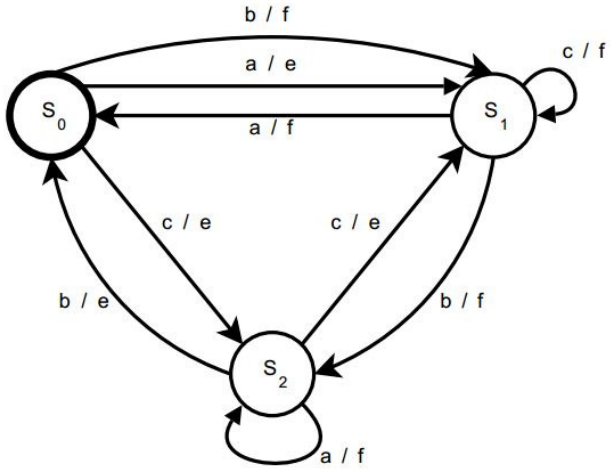
Autoria própria.

2.1.5.2 Diagrama de transições de estados

Os diagramas de transições de estados são muito úteis para facilitar a visualização das informações contidas nas MEFs. Os diagramas também são chamados de grafos de estados. Nos diagramas, os estados são retratados por círculos rotulados que são nós conectados por arcos que

representam as transições. Nesses arcos são descritas a entrada ***i*** que dispara a transição que o arco representa e a saída ***o*** produzida. Para isto, utiliza-se um rótulo **i/o**. A Figura 6 mostra um exemplo de diagrama de transições de estado, que foi utilizado para gerar a tabela de transição de estados mostrada anteriormente na Figura 5.

Figura 6 - Exemplo de diagrama de transições de estado



Fonte: Test selection based on finite state models (FUJIWARA et al., 1991).

2.1.5.3 Critérios de cobertura

Para modelos representados por meio de MEFs, os casos de teste não representam a abordagem simples em que apenas um valor de entrada (ou um conjunto de valores) deve ser passado ao programa sob teste e apenas um valor (ou um conjunto de valores) é esperado como saída. Ao invés disso, os casos de teste devem ser descritos em função dos elementos existentes no próprio modelo. Para MEFs, no caso geral, os dados de entradas de um caso de teste podem ser descritos como uma sequência de ações que levam a uma sequência de estados. E a saída esperada do caso de teste pode ser descrita como as saídas esperadas para cada transição e um estado final. Os critérios de teste baseados em MEF devem determinar conjuntos de elementos requeridos do modelo, tais como estados e transições, que devem ser exercitados durante a execução do teste.

Como as MEFs podem ser vistas essencialmente como grafos, muitos algoritmos da teoria dos grafos podem ser utilizados como meio de geração ou avaliação de elementos requeridos pelos critérios de teste. A maioria desses algoritmos está associada ao conceito de caminho. Um caminho é uma sequência de transições através dos elementos do modelo (estados).

Existem infinitamente mais casos de testes do que se pode realmente executar sobre o programa. Portanto, com o objetivo de se distinguir os casos de teste que possuem uma maior probabilidade de detecção da presença de falhas de que os outros, certos critérios de teste devem ser estabelecidos. Existem vários critérios que podem ser usados para desenvolver os casos de teste a partir de uma MEF. A escolha de quais critérios utilizar também é uma decisão muito importante, pois a eficácia da atividade de teste depende dos critérios utilizados.

A seguir iremos descrever brevemente alguns destes critérios, que são bastante utilizados em empresas produtoras de *software*.

a) **Seleção aleatória**: uma das mais populares para essa finalidade, que possibilita que os caminhos sejam selecionados aleatoriamente, escolhendo qualquer transição disponível a partir de um estado. A natureza arbitrária dessa preferência significa que elas tendem a gerar combinações não usais que testadores humanos não se preocupariam em testar. Contudo, esse critério tende a ser muito ineficiente para cobrir um grafo muito grande, por ser infinito (ROCHA; MALDONATO; WEBER, 2001) (ROBINSON, 1999) (ROBINSON, 2000);
b) **Todos os estados**: esse é um dos critérios mais simples que há, pois nele é definido que todos os estados da MEF devem ser executados. Porém, esse critério possui uma cobertura muito pequena da MEF (ROSARIA; ROBINSON, 2000);
c) **Todas as transições**: também é um dos critérios mais simples que existe, sendo um pouco mais exigente que o critério anterior "Todos os estados". Nele, todas as transições devem ser executadas. Esse critério também possui uma cobertura pequena (OFFUTT et al., 2003) (ROSARIA; ROBINSON, 2000);
d) **Caminho mais curto primeiro**: esse critério começa a escolha de caminhos por meio do estado inicial e incrementalmente escolhe todos os caminhos de tamanhos 2, 3, 4, etc. Essa é basicamente uma busca em profundidade, sem nenhuma garantia de otimização, mas os caminhos são geralmente eficientes (ROBINSON, 2000) (ROSARIA; ROBINSON, 2000);
e) **Carteiro chinês**: esse critério baseia-se no algoritmo do carteiro chinês, que produz um caminho a partir da MEF que exercita todas as transições com o menor número de passos possível. Através desse critério, alcança-se a menor sequência de testes que oferecerá cobertura completa da MEF inteira (ROBINSON, 1999) (ROBINSON, 2000) (ROSARIA; ROBINSON, 2000);
f) **Caminho mais provável primeiro**: esse critério é usado para grafos cujas transições apresentam uma estipulada probabilidade de ocorrência (Cadeias de Markov). Esse critério estabelece que os caminhos com maior probabilidade sejam executados primeiro. Esse

critério pode ser adequado para teste de funcionalidade extensiva, teste de regressão e para a comprovação de que uma versão do *software* tem as funcionalidades básicas;

g) **Todos os caminhos**: esse critério apresenta a maior cobertura, já que todas as combinações de transições dentro da MEF devem ser executadas. Todavia, do mesmo modo que o critério "Todos os Caminhos" do teste estrutural, dependendo do tamanho da MEF, esse critério pode ser encarado como inviável. No teste de *software* baseado em estados, essa abordagem também é conhecida por switch cover (BOURHFIR; DSSOULI; ABOULHAMID, 1996) (ROBINSON, 1999) (OFFUTT et al., 2003).

Além dos critérios supracitados, existem critérios que foram estabelecidos formalmente com base em trabalhos de pesquisa efetuados especificamente para Teste Baseado em MEFs. Segundo Fujiwara et al. (1991), os mais conhecidos desses critérios são: método Distinguishing Sequence – DS (GILL, 1962) (GONENC, 1970) (KOHAVI, 1978); método W (CHOW, 1978); Transition Tour (NAITO 1981); Método Unique-Input-Output – UIO (SABNANI; DAHBURA, 1988); e o método Wp (FUJIWARA et al., 1991). Embora a utilização desses critérios ofereça uma maior garantia de qualidade e confiabilidade através da realização dos testes, eles são critérios mais complexos, evidenciando consequentemente um maior custo para sua aplicação.

2.2 Considerações Finais

Nesta seção, foram apresentados conceitos e fundamentos de teste de *software* abordando técnicas tradicionais e já consolidadas, como classes de equivalências e máquina de estados, que serão utilizados no trabalho.

Após a apresentação desses conceitos e fundamentos, é possível compreender que este trabalho utiliza técnicas de testes de caixa-preta, como classes de equivalências e também faz uso de máquina de estados para gerar os casos de testes necessários para realizar testes em interfaces que se baseiam em gestos (Seção 3). Os conceitos de classes de equivalência Normal Fraco e Normal Forte são unificados ao de cobertura de Máquina de Estado para estabelecermos uma hierarquia de dominação de cobertura (Seção 4). Por fim, um simulador foi implementado para navegar na Máquina de Estados perfazendo o critério de cobertura de seleção aleatória (Seção 5).

3 MODELAGEM

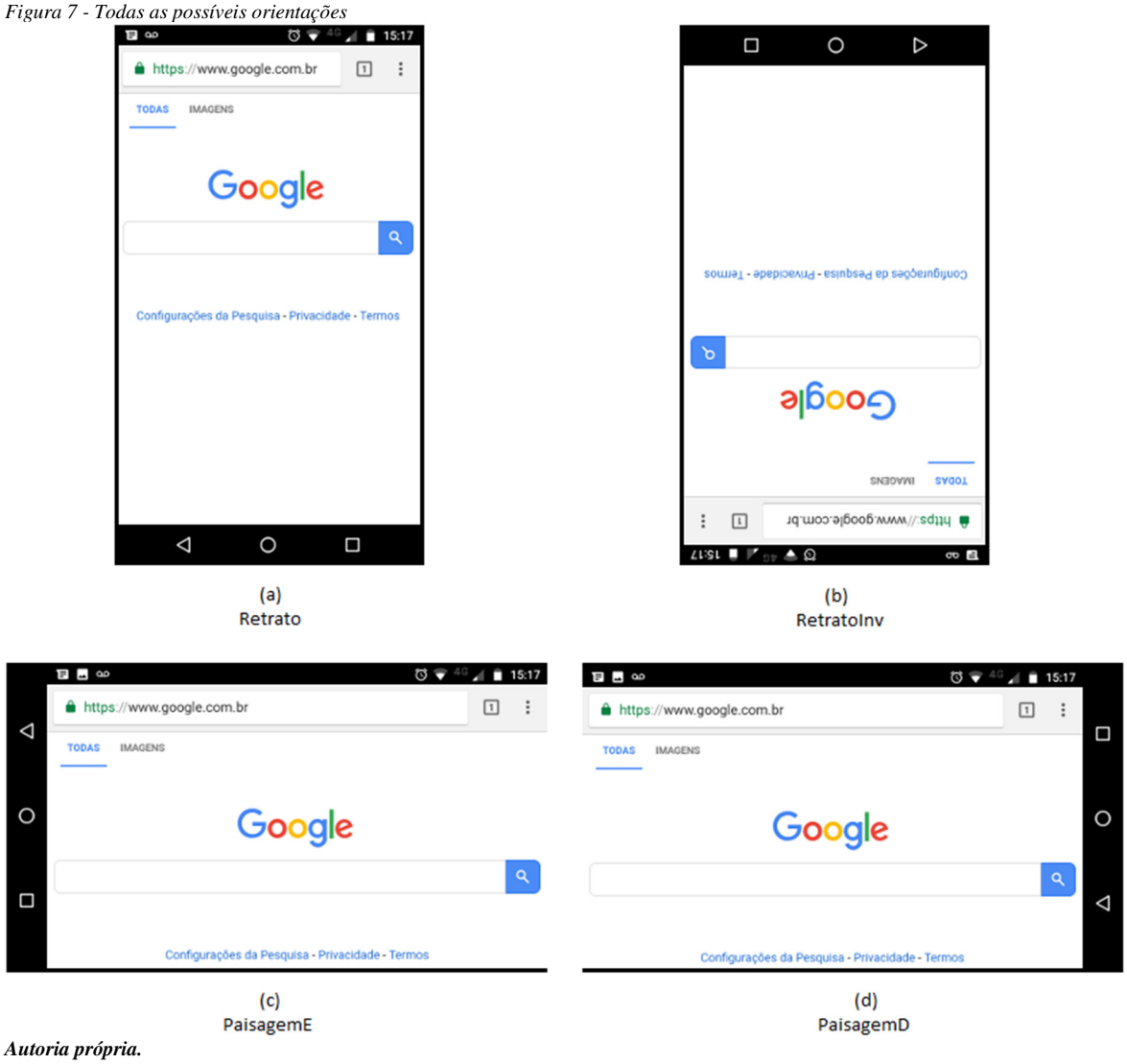
Esta seção descreve como as técnicas tradicionais de classes de equivalência e máquina de estados podem ser usadas para modelar interfaces baseadas em gesto.

3.1 Classes de Equivalência

A descoberta das classes de equivalência iniciou através de um processo exploratório de teste utilizando um smartphone executando diferentes aplicativos de calculadora. A utilização de gestos para mudança na orientação da apresentação de telas da calculadora e a procura por erros nestes aplicativos seria mais rápida e fácil, já que a realização dos testes de funcionamento e suas verificações teriam que ser feitas manualmente.

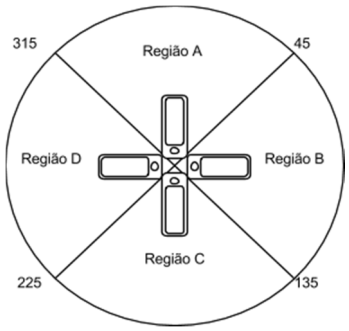
Para auxiliar na visualização das variáveis de entrada e saída, seus valores e as classes de equivalência resultantes dos testes exploratórios, considere a Figura 7 e a Figura 8.

Figura 7 - Todas as possíveis orientações



Autoria própria.

Figura 8 - Todas as possíveis regiões do dispositivo



Autoria própria.

Note que um celular pode estar na Região C, com a tela na orientação RetratoInv (se girarmos o celular com alta velocidade da Região A para a C) ou com a tela na orientação PaisagemE (se girarmos lentamente no sentido horário da Região A para a C). Ou seja, as variáveis Orientação e Região são independentes e permitem combinações diversas entre seus valores.

Modelamos o dispositivo como tendo as seguintes variáveis de entrada:

- Região inicial: Esta variável representa o ângulo de rotação, em graus, do smartphone inicialmente. Ela pode assumir um valor de 0 a 360;
- Orientação inicial: Esta variável representa a orientação da tela do aplicativo no momento inicial. Ela pode assumir um dos valores: retrato (tela do aplicativo está com a parte superior voltada para cima, ou seja, a barra de navegação fica abaixo do aplicativo, conforme Figura 7a), PaisagemE (a barra de navegação fica à esquerda do aplicativo, como na Figura 7c), RetratoInv (tela do aplicativo está com a parte superior voltada para cima, porém de cabeça para baixo, como mostra a Figura 7b) ou PaisagemD (a barra de navegação fica à direita do aplicativo, como mostra a Figura 7d);
- Sentido de rotação: Esta variável representa o sentido de rotação do smartphone. Ela pode assumir o valor horário ou anti-horário;
- Velocidade: Esta variável representa a velocidade de rotação, em graus por segundos, do smartphone. Ela pode assumir um valor de 0 a 1.000[1];
- Região final: Esta variável representa o ângulo de rotação, em graus, do smartphone no final da rotação. Ela pode assumir um valor de 0 a 360.

E com a seguinte variável de saída:

- Orientação final: Esta variável representa a orientação da tela do aplicativo no momento final. Ela pode assumir um dos valores: retrato, PaisagemE, RetratoInv ou PaisagemD.

[1] Estes valores são arbitrários e não vieram de medições reais. Porém, isto não tem nenhum impacto na modelagem ou na análise. Quando os testes precisarem ser efetivamente executados, tais valores precisam ser revistos.

Depois de diversos testes exploratórios, observações e propostas, chegamos a seguinte proposta para as classes de equivalência:

Para região inicial temos 4 classes: Em Pé, DeitadoD, Cabeça para Baixo e DeitadoE.

- Valores da classe: Em Pé (em graus) = {316, ..., 44}; representante da classe: 0
- Valores da classe: DeitadoD (em graus) = {46, …, 134}; representante da classe: 90
- Valores da classe: Cabeça para Baixo (em graus) = {136, ...,224}; representante da classe: 180
- Valores da classe: DeitadoE (em graus) = {226, ..., 314}; representante da classe: 270

Para orientação inicial da GUI temos 4 classes: Em Pé, DeitadoD, Cabeça para Baixo e DeitadoE.

- Valores da classe: Em Pé = {Retrato}; representante: Retrato
- Valores da classe: DeitadoD = {PaisagemE}; representante: PaisagemE
- Valores da classe: Cabeça para Baixo = {RetratoInv}; representante: RetratoInv
- Valores da classe: DeitadoE = {PaisagemD}; representante: PaisagemD

Para o sentido de rotação temos 2 classes: Horário e Anti-horário.

- Valores da classe: Horário = {  }; representante da classe: 
- Valores da classe: Anti-horário = {  }; representante da classe: 

Para a velocidade temos 2 classes: Devagar e Rápida.

- Valores da classe: Devagar (em graus por segundo) = {0,1, …,90}; representante: 30
- Valores da classe: Rápida (em graus por segundo) = {91, ..., 1.000}; representante: 1.000

Para a região final temos 4 classes: Em Pé, DeitadoD, Cabeça para Baixo e DeitadoE.

- Valores da classe: Em Pé (em graus) = {316, ..., 44}; representante: 0
- Valores da classe: DeitadoD (em graus) = {46, …, 134}; representante: 90
- Valores da classe: Cabeça para Baixo (em graus) = {136, ...,224}; representante: 180
- Valores da classe: DeitadoE (em graus) = {226, ..., 314}; representante: 270

Para orientação final da GUI temos 4 classes: Em Pé, DeitadoD, Cabeça para Baixo e DeitadoE.

- Valores da classe Em Pé = {Retrato}; representante: Retrato
- Valores da classe DeitadoD = {PaisagemE}; representante: PaisagemE
- Valores da classe Cabeça para Baixo = {RetratoInv}; representante: RetratoInv
- Valores da classe DeitadoE = {PaisagemD}; representante: PaisagemD

Logo, temos, na região inicial, 4 classes e, assim, 4 representantes para testar; na orientação inicial da GUI, mais 4 classes, com mais 4 representantes; na região final, mais 4 classes e seus 4 representantes; no sentido de rotação 2 classes, por isso 2 representantes; na velocidade, mais 2 classes com mais 2 representantes; e, por fim, 4 classes da região final, com mais 4 representantes. Ou seja, os possíveis testes seriam no total: 4 (região inicial) * 4

(orientação inicial da GUI) * 2 (sentido de rotação) * 2 (velocidade de rotação) * 4 (região final) = 256. Como a orientação final é uma variável de saída, não entra nos cálculos dos testes.

Devido às muitas variáveis presentes e visando um melhor entendimento, resolvemos representar os testes através de uma tabela. A Tabela 1 mostra um pequeno trecho da tabela de testes completa (que pode ser vista na íntegra no Apêndice A).

Tabela 1 - Trecho da tabela de testes com os 5 primeiros testes possíveis

Testes	**Entradas**					**Saída**
	Região inicial	Orientação inicial	Sentido de rotação	Velocidade	Região final	Orientação final
1	A	Retrato	Horário	Devagar	A	Retrato
2	A	Retrato	Horário	Devagar	B	PaisagemE
3	A	Retrato	Horário	Devagar	C	PaisagemE
4	A	Retrato	Horário	Devagar	D	PaisagemD
5	A	Retrato	Horário	Rápida	A	Retrato
			⋮			

Autoria própria.

A Tabela 2 mostra outro trecho da tabela de testes com uma amostra dos testes impossíveis de serem realizados, pois a técnica não consegue capturar a implementação adotada na indústria.

Tabela 2 - Trecho da tabela de testes com os 5 primeiros testes impossíveis

Testes	**Entradas**					**Saída**
	Região inicial	Orientação inicial	Sentido de rotação	Velocidade	Região final	Orientação final
			⋮			
17	A	RetratoInv	Horário	Devagar	A	IMPOSSÍVEL
18	A	RetratoInv	Horário	Devagar	B	
19	A	RetratoInv	Horário	Devagar	C	
20	A	RetratoInv	Horário	Devagar	D	
21	A	RetratoInv	Horário	Rápida	A	
			⋮			

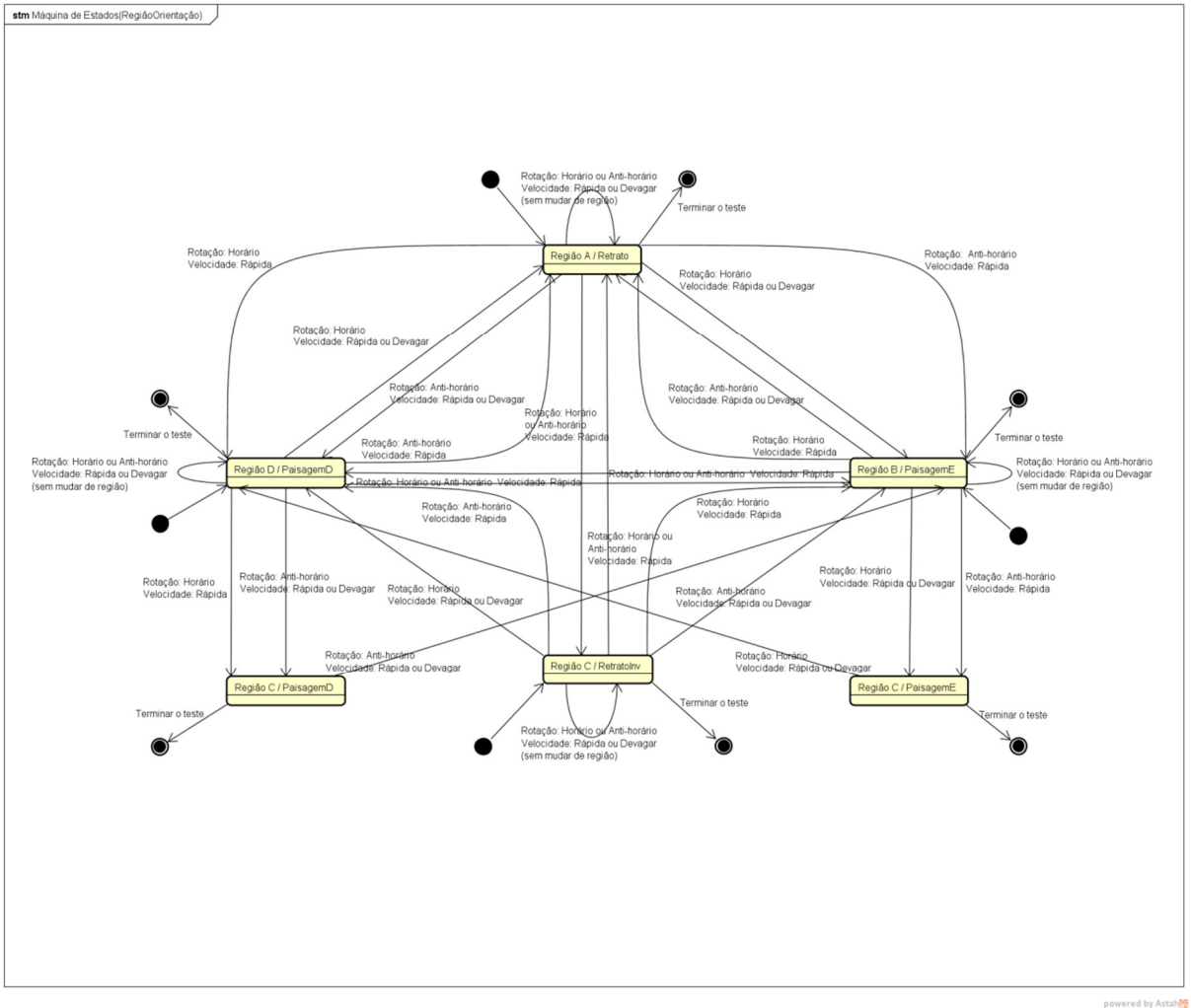
Autoria própria.

Tirando os testes impossíveis, os números de testes passam para 64. A tabela completa de testes, bem como sua versão compacta, sem os testes impossíveis de serem realizados encontram-se no Apêndice A.

3.2 Máquina de Estados Finitos

Outra técnica de geração de testes é aquela baseada na cobertura de uma máquina de estados. Nesta seção propomos uma máquina de estados que pudesse capturar a realização de todos os testes possíveis nas diversas regiões, orientações, velocidades e sentidos. A Figura 9 mostra o resultado final desta modelagem.

Figura 9 - Diagrama da máquina de estados proposta[2]



Autoria própria.

A máquina de estados exposta na Figura 9 possui 6 estados no total, sendo eles: "Região A/Retrato" que, representa o dispositivo na região A (Figura 8), com orientação Retrato; "Região

[2] Este diagrama (assim como a análise de Classes de Equivalência) modela parcialmente o teste. Estamos assumindo que o teste final é sempre composto de: (i) passos de interação com uma aplicação; (ii) rotação (onde este trabalho se enquadra); e, por fim (iii) mais alguma interação final. Os passos (i) e (iii) devem ser projetados em separado.

B/PaisagemE" que, representa o dispositivo na região B (Figura 8), com orientação PaisagemE; "Região C/PaisagemE" que, representa o dispositivo na região C (Figura 8), com orientação PaisagemE; "Região C/RetratoInv" que, representa o dispositivo na região C (Figura 8), com orientação RetratoInv; "Região C/PaisagemD" que, representa o dispositivo na região C (Figura 8), com orientação PaisagemD; e "Região D/PaisagemD" que, representa o dispositivo na região D (Figura 8), com orientação PaisagemD. Destes estados, apenas 4 podem ser estados iniciais ("Região A/Retrato", "Região B/PaisagemB", "Região C/RetratoInv" e "Região D/PaisagemD"). Isto acontece devido a impossibilidade de iniciar o dispositivo na Região C (Figura 8) com as orientações PaisagemE ou PaisagemD, o que nos dá dois estados a menos. Já como estados finais, temos a possibilidades dos 6 estados da máquina. Podemos ver na Tabela 3 alguns exemplos de testes que podem ser realizados na máquina de estados.

Tabela 3 - Exemplos de testes que podem ser realizados na máquina de estados

Testes	**Entradas**					**Saída**
	Estado inicial	Sentido de rotação	Velocidade	Estado final	Região final	Orientação final
1	Região A/Retrato	Horário	Devagar	Região D/PaisagemD	D	PaisagemD
2	Região A/Retrato	Horário	Rápido	Região A/Retrato	A	Retrato
3	Região A/Retrato	Anti-horário	Devagar	Região C/PaisagemD	C	PaisagemD
4	Região B/PaisagemE	Horário	Devagar	Região C/PaisagemE	C	PaisagemE
5	Região B/PaisagemE	Anti-horário	Devagar	Região D/PaisagemD	D	PaisagemD
6	Região C/RetratoInv	Horário	Devagar	Região C/RetratoInv	C	RetratoInv
7	Região C/RetratoInv	Horário	Devagar	Região D/PaisagemD	D	PaisagemD
8	Região D/PaisagemD	Horário	Rápido	Região A/Retrato	A	Retrato
9	Região D/PaisagemD	Anti-horário	Rápido	Região B/PaisagemE	B	PaisagemE
10	Região D/PaisagemD	Anti-horário	Devagar	Região C/PaisagemD	C	PaisagemD
	⋮					

Autoria própria.

3.3 Considerações Finais

Nesta seção mostramos como as técnicas de classes de equivalência e máquina de estados podem ser utilizadas para modelar interfaces gestuais de rotação retrato e paisagem. O modelo de classes de equivalência permite-nos executar 64 testes, enquanto o modelo de máquina de estados permite-nos gerar infinitos testes (devido aos laços do modelo). Além disto, a máquina de estados modela o sistema de forma mais detalhada: por exemplo, o teste 6 da Tabela 3 não é capturado pelas classes de equivalência.

4 CRITÉRIOS DE COBERTURA

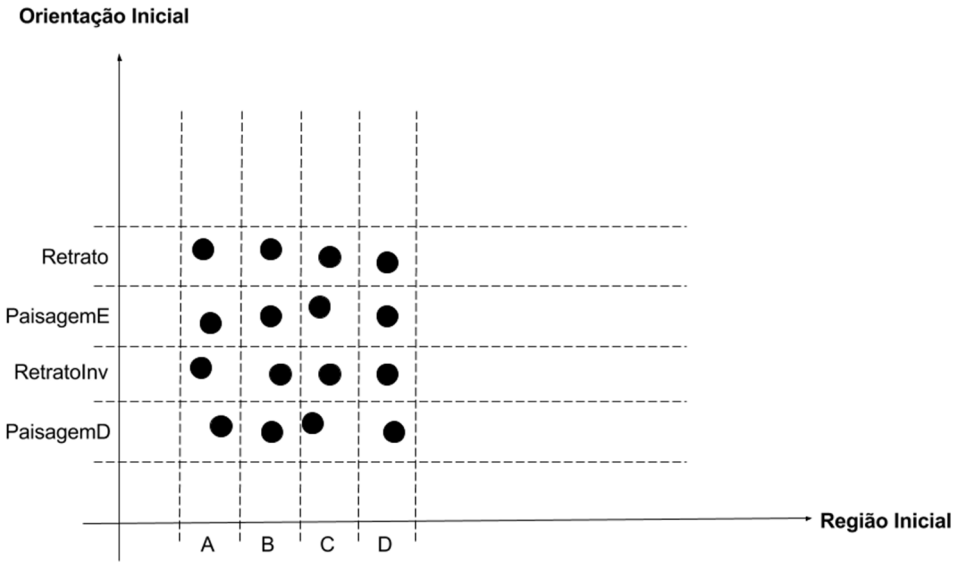
Esta seção conceitua os critérios de cobertura utilizados no estudo, bem como estabelece a hierarquia de dominação entre estes critérios.

4.1 Normal Forte

O critério de cobertura ***normal forte*** utilizado em nosso estudo é baseado no critério de cobertura normal forte de classes de equivalência e testa todas as classes de equivalência, passando por cada uma das possíveis combinações de entrada. Para chegarmos aos testes deste critério, dividimos o processo em etapas que irão analisar as variáveis dois a dois.

- Etapa 1: Como mostra a Figura 10, separam-se os testes que possuem as seguintes características:

Figura 10 - Normal Forte: Etapa 1



Autoria própria.

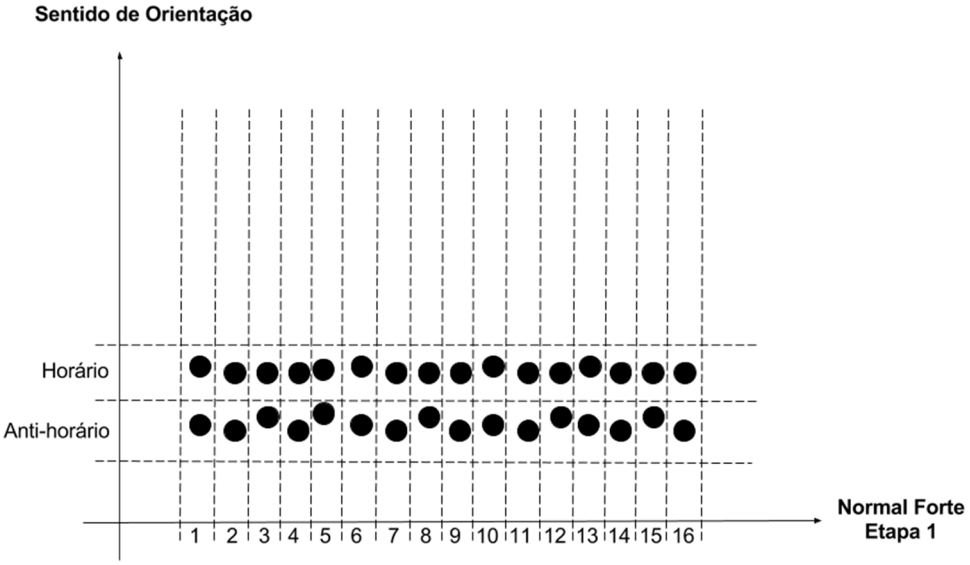
RegiãoInicial: A e OrientaçãoInicial: Retrato
RegiãoInicial: A e OrientaçãoInicial: PaisagemE
RegiãoInicial: A e OrientaçãoInicial: RetratoInv
RegiãoInicial: A e OrientaçãoInicial: PaisagemD
RegiãoInicial: B e OrientaçãoInicial: Retrato
RegiãoInicial: B e OrientaçãoInicial: PaisagemE
RegiãoInicial: B e OrientaçãoInicial: RetratoInv
RegiãoInicial: B e OrientaçãoInicial: PaisagemD
RegiãoInicial: C e OrientaçãoInicial: Retrato
RegiãoInicial: C e OrientaçãoInicial: PaisagemE
RegiãoInicial: C e OrientaçãoInicial: RetratoInv
RegiãoInicial: C e OrientaçãoInicial: PaisagemD
RegiãoInicial: D e OrientaçãoInicial: Retrato

RegiãoInicial: D e OrientaçãoInicial: PaisagemE
RegiãoInicial: D e OrientaçãoInicial: RetratoInv
RegiãoInicial: D e OrientaçãoInicial: PaisagemD

Ao final da 1ª etapa, temos como resultado 16 testes. Pegamos estes 16 testes e partimos para a Etapa 2, levando em consideração o sentido de orientação.

- Etapa 2: Podemos visualizar, na Figura 11, que os testes separados possuem:

Figura 11 - Normal Forte: Etapa 2



Autoria própria.

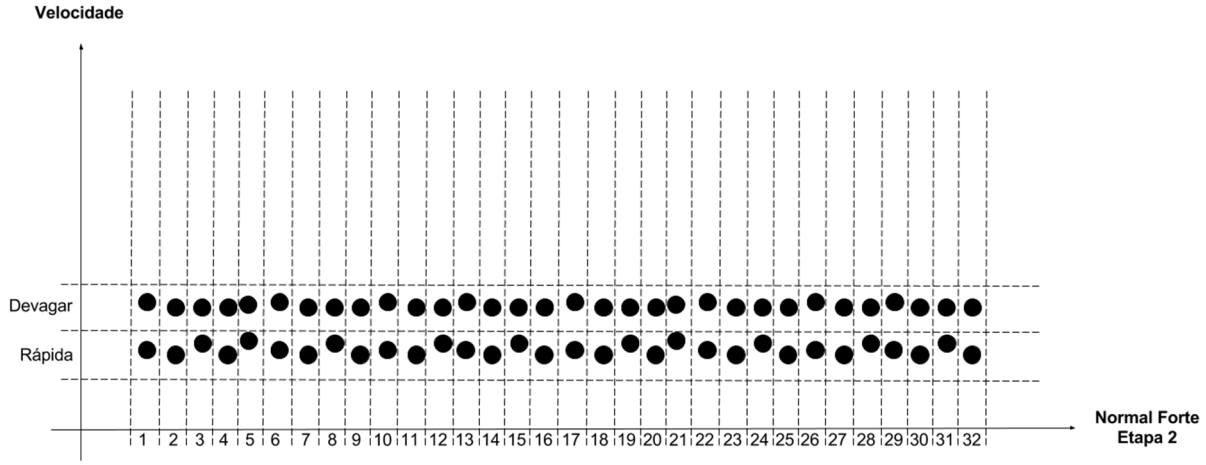
RegiãoInicial: A ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Horário
RegiãoInicial: A ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Anti-horário
RegiãoInicial: A ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Horário
RegiãoInicial: A ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Anti-horário
RegiãoInicial: A ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Horário
RegiãoInicial: A ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Anti-horário
RegiãoInicial: A ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Horário
RegiãoInicial: A ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Anti-horário
RegiãoInicial: B ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Horário
RegiãoInicial: B ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Anti-horário
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Horário
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Anti-horário
RegiãoInicial: B ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Horário
RegiãoInicial: B ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Anti-horário
RegiãoInicial: B ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Horário
RegiãoInicial: B ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Anti-horário
RegiãoInicial: C ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Horário
RegiãoInicial: C ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Anti-horário
RegiãoInicial: C ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Horário
RegiãoInicial: C ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Anti-horário
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Horário
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Anti-horário
RegiãoInicial: C ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Horário
RegiãoInicial: C ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Anti-horário

RegiãoInicial: D ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Horário
RegiãoInicial: D ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Anti-horário
RegiãoInicial: D ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Horário
RegiãoInicial: D ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Anti-horário
RegiãoInicial: D ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Horário
RegiãoInicial: D ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Anti-horário
RegiãoInicial: D ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Horário
RegiãoInicial: D ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Anti-horário

Ao final da 2ª etapa, temos 32 testes. Analogamente fazemos o mesmo o processo com todas as outras variáveis de entrada.

- Etapa 3: Apresentam-se, na Figura 12, os testes:

Figura 12 - Normal Forte: Etapa 3



Autoria própria.

RegiãoInicial: A ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: A ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: A ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: A ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: A ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Devagar

RegiãoInicial: A ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: A ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: A ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: A ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: B ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar

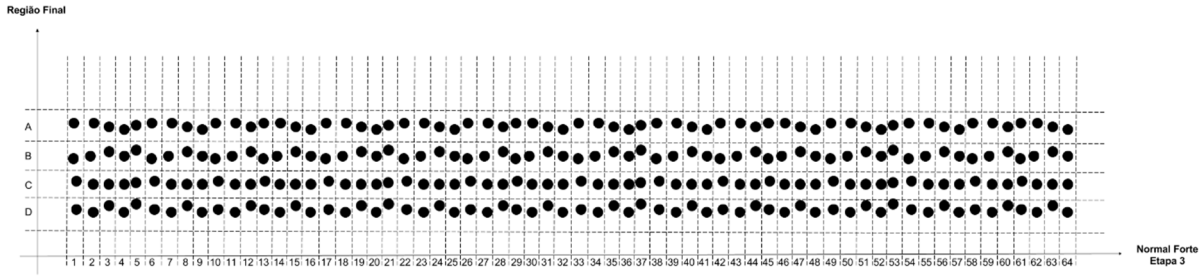
RegiãoInicial: C ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: C ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Horário e Velocidade: Devagar

RegiãoInicial: D ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: D ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Devagar

Ao final da 3ª etapa, temos 64 testes. E para finalizar vamos para 4ª e última etapa.

- Etapa 4: Ao final, exibem-se, na Figura 13, os 256 testes.

Figura 13 - Normal Forte: Etapa 4



Autoria própria.

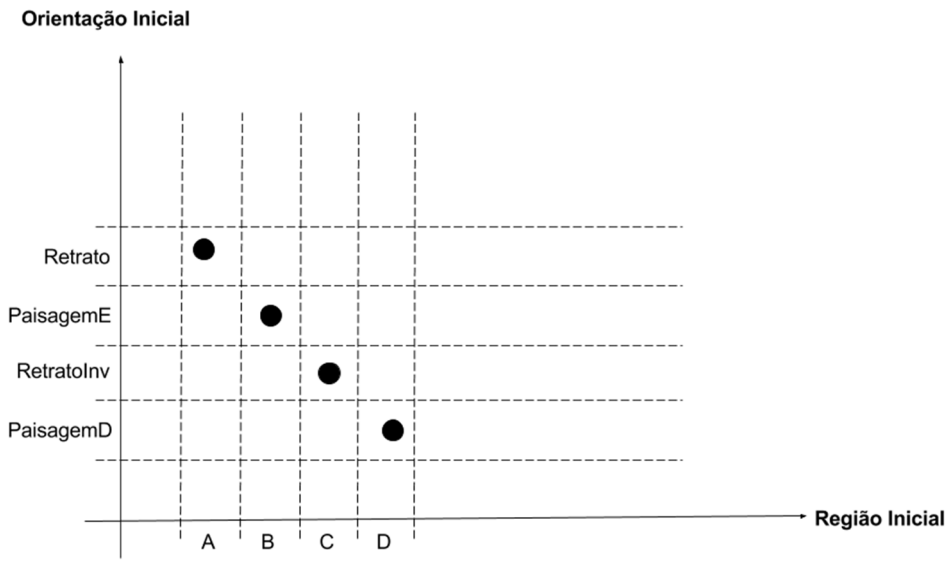
Devido às suas características, o critério normal forte necessita da realização de todas as possibilidades de testes. Logo, seria necessária a realização dos 256 testes (testes possíveis e testes impossíveis). Assim, este critério pode ser considerado o mais completo e também o mais custoso de ser realizado.

4.2 Normal Fraco

Assim como o critério anterior, o critério ***normal fraco*** utilizado por nós é baseado em classes de equivalência. Porém, os casos de teste são criados levando em consideração apenas um valor de cada classe de equivalência, como explicado na Seção 2.1.4.1. Para exemplificarmos este critério, também dividimos em etapas, visando facilitar o entendimento.

- Etapa 1: Como mostra a Figura 14, separam-se os testes que possuem as seguintes características:

Figura 14 - Normal Fraco: Etapa 1



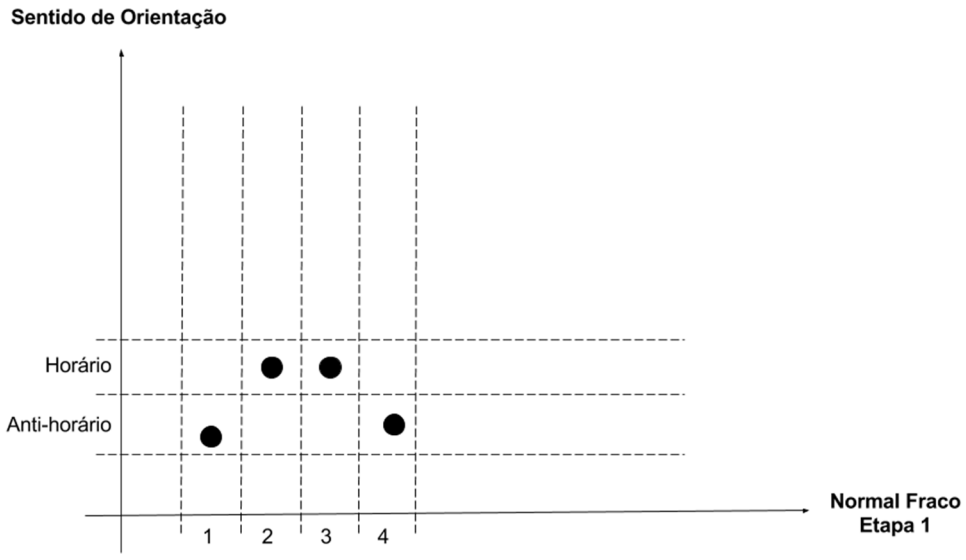
Autoria própria.

RegiãoInicial: A e OrientaçãoInicial: Retrato
RegiãoInicial: B e OrientaçãoInicial: PaisagemE
RegiãoInicial: C e OrientaçãoInicial: RetratoInv
RegiãoInicial: D e OrientaçãoInicial: PaisagemD

Ao final da 1ª etapa, temos como resultado 4 características de testes que podemos selecionar. Partimos então para a etapa 2, levando em consideração o sentido de orientação.

- Etapa 2: Não precisamos utilizar todas as possibilidades. Logo, continuamos as quatro características selecionados, levando em conta agora o sentido de orientação destes testes, conforme a Figura 15.

Figura 15 - Normal Fraco: Etapa 2



Autoria própria.

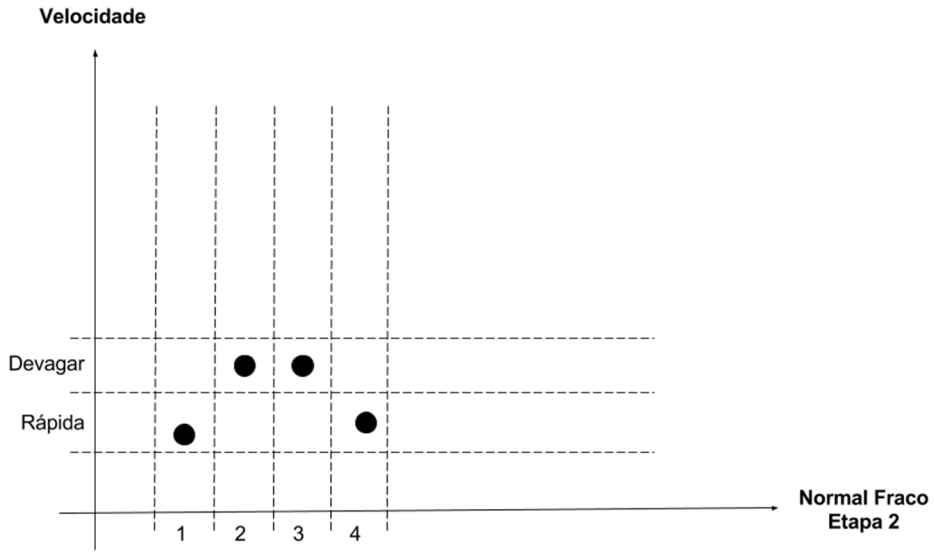
Assim podemos ter os testes com:

RegiãoInicial: A ; OrientaçãoInicial: Retrato e SentidoDeOrientação: Anti-horário
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE e SentidoDeOrientação: Horário
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv e SentidoDeOrientação: Horário
RegiãoInicial: D ; OrientaçãoInicial: PaisagemD e SentidoDeOrientação: Anti-horário

Continuamos com as demais variáveis de entrada.

- Etapa 3: Como representado na Figura 16, temos:

Figura 16 - Normal Fraco: Etapa 3



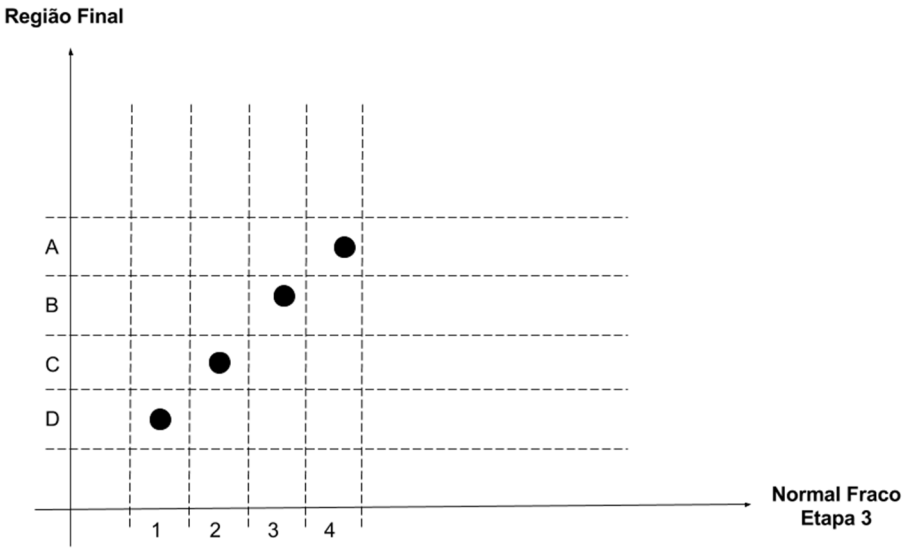
Autoria própria.

RegiãoInicial: A ; OrientaçãoInicial: Retrato ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida
RegiãoInicial: B ; OrientaçãoInicial: PaisagemE ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: C ; OrientaçãoInicial: RetratoInv ; SentidoDeOrientação: Horário e Velocidade: Devagar
RegiãoInicial: D ; OrientaçãoInicial: PaisagemD ; SentidoDeOrientação: Anti-horário e Velocidade: Rápida

E para finalizar:

- Etapa 4: Completamos as características dos testes com a última variável de entrada, como mostra a Figura 17.

Figura 17 - Normal Fraco: Etapa 4



Autoria própria.

Assim, uma das possibilidades de cobertura deste critério seria a suíte de testes 16, 115, 146 e 237 (Tabela 9).

Logo, para cobrirmos o critério de cobertura normal fraco, teríamos, no mínimo, 4 testes, pois precisamos levar em consideração as quatro regiões iniciais, as quatro orientações iniciais, os dois sentidos de orientação, as duas velocidades e as quatro regiões finais.

4.3 Todos os Estados

O critério ***todos os estados*** têm testes que devem passar por todos os estados da máquina de estados. Para isso precisa pelo menos cobrir os testes que passem pelas regiões: A com orientação Retrato (equivalente ao estado A-Retrato da máquina de estados proposta), B com orientação PaisagemE (equivalente ao estado B-PaisagemE), D com orientação PaisagemD (equivalente ao estado D-PaisagemD), C com orientação PaisagemE (equivalente ao estado C-PaisagemE), C com orientação PaisagemD (equivalente ao estado C-PaisagemD) e C com orientação RetratoInv (equivalente ao estado C-RetratoInv), que sinalizam a passagem por todos os estados da máquina de estados proposta e mostrada na Figura 9. Assim, nota-se que no caso estudado, não é necessário cobrir todas as transições.

Um exemplo de uma suíte de testes que satisfaz o critério de todos os estados seria: 1, 7, e 9 (Tabela 9).

4.4 Todas as Transições

Por fim, temos o critério de ***todas as transições***. Como o próprio nome diz, são necessários testes que passem por todas as transições da máquina de estados. Apesar de envolver muitos testes, não necessita de todos os 256, uma vez que há diversos testes que, durante sua realização, já cobrem diversas transições simultaneamente, e sua cobertura pode ser realizada dentro do contexto dos testes possíveis.

Um exemplo de uma suíte de testes que satisfaz o critério de todas as transições seria: 1, 5, 6, 7, 8, 9, 13, 14, 15, 16, 18, 21, 22, 23, 24, 26, 29, 30, 31, 32, 36, 37, 38, 39, 40, 42, 45, 46, 47, 48, 52, 53, 54, 55, 56, 60, 61, 62, 63 e 64 (Tabela 10) mais 2 testes que não são capturados pelas classes de equivalência (C-RetratoInv-Horário-Devagar-C-RetratoInv e C-RetratoInv-Anti-horário-Devagar-C-RetratoInv). Totalizando 42 testes.

4.5 Hierarquia de Dominação

Um método comum de comparação de critérios de teste é a relação de dominação (FRANKL; WEYUKER, 1988). O critério A domina o critério B se, e somente se, cada suíte de teste que satisfaça A também satisfaça B. Os quatro critérios de cobertura de teste apresentados neste trabalho possuem a hierarquia de dominação retratada na Figura 18.

Figura 18 - Hierarquia de dominação dos critérios de cobertura estudados

Autoria própria.

A relação entre Todas as Transições e Todos os Estados já é bem conhecida na literatura (MORAES; ANDRADE; MACHADO, 2016). Da mesma forma, apesar de não ter sido explicitamente introduzida em trabalhos anteriores, o critério Normal Fraco é, por definição, dominado pelo Normal Forte. Como este trabalho trata de um modelo instanciado da interface gestual de rotação Retrato e Paisagem, é possível unificar a hierarquia destes critérios como

mostrado na Figura 18. A dominação de Normal Forte sobre Normal Fraco e de Normal Forte sobre Todos os Estados só ocorrem para os modelos da Seção 3. Estes relacionamentos não são genéricos para qualquer Classe de Equivalência e qualquer Máquina de Estados, diferentemente da dominação Todas as Transições sobre Todos os Estados, que é universal.

O critério Normal Forte, como visto na Seção 2.1.4.2, possui casos de testes de cada elemento do produto cartesiano das classes de equivalência, passando obrigatoriamente por todos os estados da máquina de estados proposta, satisfazendo também o critério Todos os Estados. Logo, o critério Normal Forte domina o critério Todos os Estados. Entretanto, o critério Normal Forte não domina o critério Todas as Transições, como podemos ver no teste 6 da Tabela 3 que não é capturado pelo critério Normal Forte. E o critério Todas as Transições também não domina o critério Normal Forte, afinal uma das suítes de teste que satifaz o critério Todas as Transições possui 42 testes e o critério Normal Forte possui 256 testes.

4.6 Considerações Finais

Nesta seção foram conceituados e exemplificados os critérios de cobertura propostos, bem como foi estabelecida a hierarquia de dominação destes critérios para medições posteriores, nas simulações (Seção 5).

Como este trabalho instancia os modelos de classe de equivalência e diagrama de estados para interface gestual de rotação Retrato e Paisagem, foi possível criar um diagrama de dominação unificado para ambos os modelos.

5 SIMULAÇÕES

Esta seção descreve o desenvolvimento de um software codificado para gerar as simulações de um testador humano realizando testes aleatórios de rotação Retrato e Paisagem. Nele também são exibidos os resultados das simulações realizadas.

A seguir, descreveremos o desenvolvimento do *software* codificado para gerar as simulações de um testador humano realizando testes aleatórios de rotação Retrato e Paisagem. A simulação deste testador humano se dá através do caminhar na máquina de estados introduzida na Seção 3, onde cada transição a ser disparada é decidida por sorteio. O caminhar na máquina (a simulação) dura o tempo determinado pelo usuário. Também mostramos como este simulador foi usado para gerar os dados de porcentagem de cobertura de cada critério descrito na Seção 4. Como resultado, construímos gráficos *box-plot* com as porcentagens de cobertura de cada critério descrito na Seção 4. Como resultado para 1000 repetições, com tempos diferenciados, assim como suas análises.

5.1 Software Para Simular Um Testador Humano

Em Arcuri e Briand (2011), mostra-se que é uma boa prática, ao se envolver algoritmos aleatórios, a utilização de pelo menos 1000 execuções e sugere-se também o gráfico box-plot para representação dos resultados. Por isso, levamos em consideração essas boas práticas e optamos por fazer 1000 repetições e gerar o gráfico box-plot de cada cobertura, mostrando os diferentes tempos das execuções.

O *software* utilizado para gerar as simulações e os dados necessários para o estudo foi de autoria própria. A codificação foi realizada em linguagem de programação Java (DEITEL e DEITEL, 2011), porém, para não nos atermos a uma linguagem específica, iremos demonstrar os trechos mais importantes para o entendimento do estudo em pseudocódigo.

Para funcionamento do *software* entra-se com um argumento, que é um valor do tempo (milissegundos) de uma simulação, um inteiro, e o *software* executa as 1000 repetições de geração de testes aleatórios[3] para a configuração de tempo desejada. Primeiramente são inicializadas as variáveis necessárias. Optamos por utilizar vetores com valores lógicos, que são inicializados com **falso** para todos os valores, e à medida que constatamos a cobertura de um determinado item de um critério, alteramos o específico valor para **verdadeiro**. O Algoritmo 1 é

[3] A semente aleatória é gerada a cada seleção de variável de entrada de um teste, para tornar cada teste mais aleatória possível

um exemplo de trecho do código que mostra uma função ***passouPorEstado*** que, dado um estado, altera o valor de um elemento do vetor ***percorreuTodosEstados***, referente ao estado dado, para **verdadeiro**. Analogamente temos as funções: ***passouPorRegiaoInicial***, ***passouPorOrientacaoInicial***, ***passouPorSentido***, ***passouPorVelocidade***, ***passouPorRegiaoFinal***, ***passouPorTransicao*** e ***passouNormalForte***, onde a entrada da última é um teste. O vetor ***todosEstados*** foi inicializado no início do *software* com o valor de todos os estados possíveis. Da mesma forma, temos: ***regioesIniciais***, ***orientacoesIniciais***, ***sentidosDeRotacao***, ***velocidades***, ***regioesFinais***, ***todasTransicoes*** e ***todosNormalForte***, onde a última é um vetor contendo todos os testes do critério normal forte.

Algoritmo 1 - Função passouPorEstado[4]

```
1.  funcao passouPorEstado(estado:caracter):logico[]
2.  Inicio
3.    Para inteiro i ← 0 ate i < tamanho(percorreuTodosEstados) passo +1 faca
4.      Para inteiro j ← 0 ate j < tamanho(todosEstados) passo +1 faca
5.        Se (todosEstados[j]=estado) entao
6.          percorreuTodosEstados[j] ← verdadeiro
7.        fimSe
8.      fimPara
9.    fimPara
10.   retorne percorreuTodosEstados
11. fimFuncao
```

Autoria própria.

Depois de inicializadas as variáveis necessárias, o *software* entra no laço de 1000 repetições. Logo após, recebe o horário atual do sistema, guarda a informação em uma variável ***tempoInicio*** e calcula a variável ***tempoTotal*** que recebe (horário atual - ***tempoInicio***). O valor do ***tempoTotal*** é testado e então entra-se no laço de realização dos testes aleatórios, onde o *software* deve executar a máquina de estados. O Algoritmo 2 mostra a estrutura do código referente à execução da máquina de estados, onde foram retirados os códigos dentro de cada caso para facilitar a visualização.

Algoritmo 2 - Estrutura do trecho do código, referente à execução da máquina de estados

```
1.  Enquanto (tempoTotal<=tempoSolicitado) faca
2.      Se (ehEstadoInicial=verdadeiro) entao
3.        estadoAtual ← selecionaEstado(estadosInicias)
4.      senao
5.        estadoAtual ← proximoEstado
6.      fimSe
7.      escolha (estadoAtual)
8.        caso "A-Retrato"
9.          ...
10.       caso "B-PaisagemE"
11.         ...
12.       caso "C-RetratoInv"
13.         ...
14.       caso "D-PaisagemD"
15.         ...
16.     fimEscolha
17.     tempoTotal ← horarioAtualdoSistema()-tempoInicio //Atualiza o tempoTotal
18. fimEnquanto
```

Autoria própria.

[4] No Algoritmo 1 (linha 3) o código referente a "passo +1" expressa que no comando Para há um incremento de 1 à variável *i* a cada iteração.

Na linha 3 do Algoritmo 2, podemos ver o início das escolhas aleatórias através da função ***selecionaEstado***, que, dado um vetor de estados, escolhe um estado aleatoriamente. O Algoritmo 3 mostra o código da função ***selecionaEstado***. Da mesma maneira, existem as funções ***selecionaSentidoDeRotacao***, ***selecionaVelocidade*** e ***selecionaSeEstadoFinal***, que selecionam aleatoriamente um sentido de rotação, uma velocidade e se o estado é um estado final ou não.

Algoritmo 3 - Função selecionaEstado

```
1.  funcao selecionaEstado (var estados:caracter[]):caracter
2.  Inicio
3.    retorne estados[AleatorioProximoInt(tamanho(estados))]
4.  fimFuncao
```

Autoria própria.

O Algoritmo 4 mostra o funcionamento do caso ***"A-Retrato"***, onde nota-se que, como já foi selecionado o estado, é chamada a função ***passouPorEstado*** com o parâmetro do ***estadoAtual***. Caso seja um estado inicial, são chamadas as funções ***passouPorRegiaoInicial*** e ***passouPorOrientacaoInicial***, com os parâmetros necessários. É selecionado aleatoriamente um sentido e registrado sua cobertura pela função ***passouPorSentido***. Logo após, seleciona-se também aleatoriamente uma velocidade e registra-se a cobertura pela chamada da função ***passouPorVelocidade***. Depois disso, testa-se o sentido e a velocidade selecionados e, dependendo desses resultados, tomam-se caminhos diferentes da máquina de estados. Nesses caminhos, é selecionado, aleatoriamente, o próximo estado dentro dos valores possíveis para a máquina de estados e, a depender do valor, são registrados os estados e as transições que devem ser cobertos, repectivamente, pelas funções ***passouPorEstado*** e ***passouPorTransicao***. Feito isso, registra-se através da função ***passouPorEstado*** o valor da variável ***proximoEstado***, transforma-se os valores ***estadoAtual***, ***velocidade***, ***sentidoDeRotacao*** e ***proximoEstado*** em uma transição para então registrar a sua cobertura através da função ***passouPorTransicao*** e também na função ***passouNormalForte***. E já finalizando o caso, é chamada a função ***passouPorRegiaoFinal*** para registrar a passagem da região final; ***ehEstadoInicial*** recebe **verdadeiro** sinalizando que o próximo estado será um estado inicial, já que será um outro teste a ser realizado e as porcentagens são calculadas. Essas porcentagens são cumulativas, até o momento da próxima repetição que zera os valores. O funcionamento dos demais casos também são bem parecidos, modificando apenas os caminhos percorridos na máquina de estados, o que interfere na cobertura dos testes gerados.

Algoritmo 4 - Caso "A-Retrato"

```
1.  caso "A-Retrato"
2.    passouPorEstado(estadoAtual)
3.    Se (ehEstadoInicial=verdadeiro) entao
4.      //recebeSubstring(str, inicio, n) - Recebe como resultado uma Substring de str, começando da posição inicio com quantidade de
5.  caracteres igual n, no caso abaixo: a substring "A"
6.      passouPorRegiaoInicial(recebeSubstring(estadoAtual, 0,1))
7.      //retiraSubstring(str, n) - Recebe como resultado uma Substring de str, retirando os n caracteres, no caso abaixo: a substring
8.  "Retrato"
9.      passouPorOrientacaoInicial(substring(estadoAtual, 2))
10.     ehEstadoInicial ← falso
```

```
11.     fimSe
12.     sentidoDeRotacao ← selecionaSentidoDeRotacao(sentidosDeRotacao)
13.     passouPorSentido(sentidoDeRotacao)
14.     velocidade ← selecionaVelocidade(velocidades)
15.     passouPorVelocidade(velocidade)
16.     Se (sentidoDeRotacao="Horário") entao //Horário
17.       Se (velocidade="Devagar") entao //Horário-Devagar
18.         proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE","C-PaisagemE", "D-PaisagemD"])
19.         escolha (proximoEstado)
20.           caso "A-Retrato"
21.             passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
22.             passouPorEstado("B-PaisagemE")
23.             passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
24.             passouPorEstado("C-PaisagemE")
25.             passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
26.             passouPorEstado("D-PaisagemD")
27.             passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
28.           caso "C-PaisagemE"
29.             passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
30.             passouPorEstado("B-PaisagemE")
31.             passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
32.           caso "D-PaisagemD"
33.             passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
34.             passouPorEstado("B-PaisagemE")
35.             passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
36.             passouPorEstado("C-PaisagemE")
37.             passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
38.         fimEscolha
39.       senao //Horário-Rápida
40.           proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-RetratoInv", "D-PaisagemD"])
41.       fimSe
42.     senao //Anti-horário
43.       Se(velocidade="Devagar") //Anti-horário-Devagar
44.         proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemD", "D-PaisagemD"])
45.         escolha (proximoEstado)          caso "A-Retrato"
46.             passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
47.             passouPorEstado("D-PaisagemD")
48.             passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
49.             passouPorEstado("C-PaisagemD")
50.             passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
51.             passouPorEstado("B-PaisagemE")
52.             passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
53.           caso "B-PaisagemE"
54.             passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
55.             passouPorEstado("D-PaisagemD")
56.             passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
57.             passouPorEstado("C-PaisagemD")
58.             passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
59.           caso "C-PaisagemD"
60.             passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
61.             passouPorEstado("D-PaisagemD")
62.             passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
63.         fimEscolha
64.       senao //Anti-horário-Rápida
65.         proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-RetratoInv", "D-PaisagemD"])
66.       fimSe
67.     fimSe
68.     passouPorEstado(proximoEstado)
69.     transicao ← transformaEmTransicao(estadoAtual,velocidade,sentidoDeRotacao,proximoEstado)
60.     passouPorTransicao(transicao)
71.     passouNormalForte(transicao)
72.     ehEstadoFinal←verdadeiro
73.     Se (ehEstadoFinal=verdadeiro) entao
74.       passouPorRegiaoFinal(recebeSubstring(estadoAtual, 0,1))
75.       ehEstadoInicial ← verdadeiro
76.       porcentagemNormalForte ← coberturaNormalForte()
77.       porcentagemNormalFraco ← coberturaNormalFraco(percorreuTodasRegioesIniciais, percorreuTodasOrientacoesIniciais,
78.   percorreuTodosSentidos, percorreuTodasVelocidades, percorreuTodasRegioesFinais)
79.       porcentagemTodosOsEstados ← coberturaTodosOsEstados()
80.       porcentagemTodasAsTransicoes ← coberturaTodaAsTransicoes()
81.     fimSe
```

Autoria própria.

As funções ***coberturaNormalForte***, ***coberturaTodosOsEstados*** e ***coberturaTodaAsTransicoes*** têm funcionamento bastante parecidos. Elas percorrem

respectivamente os vetores ***percorreuTodosNormalForte***, ***percorreuTodosEstados*** e ***percorreuTodasTransicoes***, verificando qual valor foi coberto, ou seja, quais elementos do vetor tem valor igual a **verdadeiro**, contando esses valores para depois calcular a porcentagem de cobertura do critério em questão e retorná-la. Os Algoritmo 5, 6 e 7 são das respectivas funções: ***coberturaNormalForte***, ***coberturaTodosOsEstados*** e ***coberturaTodaAsTransicoes***.

Algoritmo 5 - Trecho do código da função coberturaNormalForte

```
1.  funcao coberturaNormalForte(): real
2.  Var cont: inteiro
3.  Inicio
4.    cont ←0
5.    Para inteiro i ← 0 ate i < tamanho(percorreuTodosNormalForte) passo +1 faca
6.      Se (percorreuTodosNormalForte[i] = verdadeiro) entao
7.        cont ← cont+1
8.      fimSe
9.    fimPara
10.   escolha (cont)
11.     caso 1
12.       porcentagemNormalForte ← (1/256)*100
13.     caso 2
14.       porcentagemNormalForte ← (2/256)*100
15.     caso 3
16.       porcentagemNormalForte ← (3/256)*100
17.     ...
18.     caso 254
19.       porcentagemNormalForte ← (254/256)*100
20.     caso 255
21.       porcentagemNormalForte ← (255/256)*100
22.     caso 256
23.       porcentagemNormalForte ← (256/256)*100
24.   fimEscolha
25.   retorne porcentagemNormalForte
26. fimFuncao
```

Autoria própria.

Algoritmo 6 - Função coberturaTodosOsEstados

```
1.  funcao coberturaTodosOsEstados(): real
2.  Var cont: inteiro
3.  Inicio
4.    cont ←0
5.    Para inteiro i ← 0 ate i < tamanho(percorreuTodosEstados) passo +1 faca
6.      Se (percorreuTodosEstados[i] = verdadeiro) entao
7.        cont ← cont+1
8.      fimSe
9.    fimPara
10.   escolha (cont)
11.     caso 1
12.       porcentagemTodosOsEstados ← (1/6)*100
13.     caso 2
14.       porcentagemTodosOsEstados ← (2/6)*100
15.     caso 3
16.       porcentagemTodosOsEstados ← (3/6)*100
17.     caso 4
18.       porcentagemTodosOsEstados ← (4/6)*100
19.     caso 5
20.       porcentagemTodosOsEstados ← (5/6)*100
21.     caso 6
22.       porcentagemTodosOsEstados ← (6/6)*100
23.   fimEscolha
24.   retorne porcentagemTodosOsEstados
25. fimFuncao
```

Autoria própria.

Algoritmo 7 - Trecho do código da função coberturaTodasAsTransicoes

```
1.  funcao coberturaTodaAsTransicoes(): real
2.  Var cont: inteiro
3.  Inicio
4.    cont ←0
5.    Para inteiro i ← 0 ate i < tamanho(percorreuTodasTransicoes) passo +1 faca
6.      Se (percorreuTodasTransicoes[i] = verdadeiro) entao
7.        cont ← cont+1
8.      fimSe
9.    fimPara
```

```
10.     escolha (cont)
11.       caso 1
12.         porcentagemTodasAsTransicoes ← (1/42)*100
13.       caso 2
14.         porcentagemTodasAsTransicoes ← (2/42)*100
15.       caso 3
16.         porcentagemTodasAsTransicoes ← (3/42)*100
17.       ...
18.       caso 40
19.         porcentagemTodasAsTransicoes ← (40/42)*100
20.       caso 41
21.         porcentagemTodasAsTransicoes ← (41/42)*100
22.       caso 42
23.         porcentagemTodasAsTransicoes ← (42/42)*100
24.     fimEscolha
25.     retorne porcentagemTodasAsTransicoes
26.   fimFuncao
```

Autoria própria.

Já a função ***coberturaNormalFraco*** difere um pouco das demais, uma vez que o critério de cobertura Normal Fraco, como visto na Seção 2.1.4.1, deve levar em consideração as quatro regiões iniciais, as quatro orientações iniciais, os dois sentidos de orientação, as duas velocidades e as quatro regiões finais. Sendo assim, é necessário passar uma vez por cada um dos valores da classe. Logo, percorrem-se os vetores ***percorreuTodasRegioesIniciais***, ***percorreuTodasOrientacoesIniciais***, ***percorreuTodosSentidos***, ***percorreuTodasVelocidades*** e ***percorreuTodasRegioesFinais*** verificando qual valor foi coberto, ou seja, quais elementos do vetor tem valor igual a **verdadeiro**, contando esses valores para depois calcular a porcentagem de cobertura do critério normal fraco e retorná-la.

Algoritmo 8 - Trecho do código da função coberturaNormalFraco

```
1.    funcao coberturaNormalFraco(ri1, oi2, s3, v4, rf5: logico[]): real
2.    Var cont: inteiro
3.    Inicio
4.      cont ←0
5.      Para inteiro i ← 0 ate i < tamanho(ri1) passo +1 faca
6.        Se (ri1[i] = verdadeiro) entao
7.          cont ← cont+1
8.        fimSe
9.      fimPara
10.     Para inteiro i ← 0 ate i < tamanho(oi2) passo +1 faca
11.       Se (oi2[i] = verdadeiro) entao
12.         cont ← cont+1
13.       fimSe
14.     fimPara
15.     Para inteiro i ← 0 ate i < tamanho(s3) passo +1 faca
16.       Se (s3[i] = verdadeiro) entao
17.         cont ← cont+1
18.       fimSe
19.     fimPara
20.     Para inteiro i ← 0 ate i < tamanho(v4) passo +1 faca
21.       Se (v4[i] = verdadeiro) entao
22.         cont ← cont+1
23.       fimSe
24.     fimPara
25.     Para inteiro i ← 0 ate i < tamanho(rf5) passo +1 faca
26.       Se (rf5[i] = verdadeiro) entao
27.         cont ← cont+1
28.       fimSe
29.     fimPara
30.     escolha (cont)
31.       caso 1
32.         porcent'agemNormalFraco ← (1/16)*100
33.       caso 2
34.         porcentagemNormalFraco ← (2/16)*100
35.       ...
36.       caso 15
37.         porcentagemNormalFraco ← (15/16)*100
```

```
38.       caso 16
39.         porcentagemNormalFraco ← (16/16)*100
30.     fimEscolha
21.     retorne porcentagemNormalFraco
22.   fimFuncao
```

Autoria própria.

Ao final do laço de geração dos testes aleatórios e ainda no laço de repetições, o *software* gera as porcentagens, como vistos nas linhas de 2 a 7 do Algoritmo 9. Então, ele guarda seus valores em vetores que serão posteriormente ordenados, como mostra as linhas 9 até 12 do Algoritmo 9, aumenta o contador de repetições (na linha 15) e zera as variáveis, porcentagens e vetores lógicos, para a nova repetição (linhas 18-54).

Algoritmo 9 - Trecho final do laço de repetições

```
1.
2.      ...
3.      porcentagemNormalForte ← coberturaNormalForte()
4.      porcentagemNormalFraco ← coberturaNormalFraco(percorreuTodasRegioesIniciais,
5.    percorreuTodasOrientacoesIniciais, percorreuTodosSentidos, percorreuTodasVelocidades, percorreuTodasRegioesFinais)
6.      porcentagemTodosOsEstados ← coberturaTodosOsEstados()
7.      porcentagemTodasAsTransicoes ← coberturaTodaAsTransicoes()
8.
9.      porcentagensNormalForteOrdenadas[contador_de_repeticoes] ← porcentagemNormalForte
10.     porcentagensNormalFracoOrdenadas[contador_de_repeticoes] ← porcentagemNormalFraco
11.     porcentagensTodosOsEstadosOrdenadas[contador_de_repeticoes] ← porcentagemTodosOsEstados
12.     porcentagensTodasAsTransicoesOrdenadas[contador_de_repeticoes] ← porcentagemTodasAsTransicoes
13.
14.     //Soma +1 ao contador_de_repeticoes para ir para próxima repetição
15.     contador_de_repeticoes ← contador_de_repeticoes + 1
16.
17.     //Zera todas as porcentagens para recomeçar na próxima repetição
18.     porcentagemNormalForte ← 0
19.     porcentagemNormalFraco ← 0
20.     porcentagemTodosOsEstados ← 0
21.     porcentagemTodasAsTransicoes ← 0
22.
23.     //Zera todas os vetores lógicos que indicam por onde passou, para recomeçar na próxima repetição
24.     Para inteiro i ← 0 ate i < tamanho(percorreuTodosEstados) passo +1 faca
25.       percorreuTodosEstados[i] ← falso
26.     fimPara
27.
28.     Para inteiro i ← 0 ate i < tamanho(percorreuTodasRegioesIniciais) passo +1 faca
29.       percorreuTodasRegioesIniciais[i] ← falso
30.     fimPara
31.
32.     Para inteiro i ← 0 ate i < tamanho(percorreuTodasOrientacoesIniciais) passo +1 faca
33.       percorreuTodasOrientacoesIniciais[i] ← falso
34.     fimPara
35.
36.     Para inteiro i ← 0 ate i < tamanho(percorreuTodosSentidos) passo +1 faca
37.       percorreuTodosSentidos[i] ← falso
38.     fimPara
39.
40.     Para inteiro i ← 0 ate i < tamanho(percorreuTodasVelocidades) passo +1 faca
41.       percorreuTodasVelocidades[i] ← falso
42.     fimPara
43.
44.     Para inteiro i ← 0 ate i < tamanho(percorreuTodasRegioesFinais) passo +1 faca
45.       percorreuTodasRegioesFinais[i] ← falso
46.     fimPara
47.
48.     Para inteiro i ← 0 ate i < tamanho(percorreuTodasTransicoes) passo +1 faca
49.       percorreuTodasTransicoes[i] ← falso
50.     fimPara
51.
52.     Para inteiro i ← 0 ate i < tamanho(percorreuTodosNormalForte) passo +1 faca
53.       percorreuTodosNormalForte[i] ← falso
54.     fimPara
55.   fimEnquanto
```

Autoria própria.

E, por fim, depois do laço de repetições temos, para cada critério de cobertura, a ordenação do vetor de porcentagens de cobertura recebidos e o calcular dos pontos necessários para geração dos gráficos box-plot: **média**, **q3** (terceiro quartil), **mediana** (ou segundo quartil), **q1** (primeiro quartil), **máximo** (máximo valor do vetor) e **mínimo** (mínimo valor do vetor), como podemos ver no Algoritmo 10. O código completo do *software* encontra-se no Apêndice B.

Algoritmo 10 - Trecho final do software

```
1.    ...
2.    //Ordena o vetor, para calcular os dados necessários para plotagem do
3.    //gráfico boxplot do critério Normal Forte
4.    ordena(porcentagensNormalForteOrdenadas)
5.    real media←0
6.    real mediana←0
7.    real q3←0
8.    real q1←0
9.    real maximo←0
10.   real minimo←0
11.   Para inteiro i ← 0 ate i < tamanho(porcentagensNormalForteOrdenadas) passo +1 faca
12.     media ← media + porcentagensNormalForteOrdenadas[i]
13.   fimPara
14.   media ← media/1000
15.   q3 ← porcentagensNormalForteOrdenadas[((1000/4)*3)]
16.   mediana ← (porcentagensNormalForteOrdenadas[((1000/2)-1)] + porcentagensNormalForteOrdenadas[(1000/2)])/2
17.   q1 ← porcentagensNormalForteOrdenadas[((1000/4)-1)]
18.   maximo←porcentagensNormalForteOrdenadas[999]
19.   minimo←porcentagensNormalForteOrdenadas[0]
20.   escreval("Normal Forte: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)
21.
22.   ordena(porcentagensNormalFracoOrdenadas)
23.   media←0
24.   mediana←0
25.   q3←0
26.   q1←0
27.   maximo←0
28.   minimo←0
29.   Para inteiro i ← 0 ate i < tamanho(porcentagensNormalFracoOrdenadas) passo +1 faca
30.     media ← media + porcentagensNormalFracoOrdenadas[i]
31.   fimPara
32.   media ← media/1000
33.   q3 ← porcentagensNormalFracoOrdenadas[((1000/4)*3)]
34.   mediana ← (porcentagensNormalFracoOrdenadas[((1000/2)-1)] + porcentagensNormalFracoOrdenadas[(1000/2)])/2
35.   q1 ← porcentagensNormalFracoOrdenadas[((1000/4)-1)]
36.   maximo←porcentagensNormalFracoOrdenadas[999]
37.   minimo←porcentagensNormalFracoOrdenadas[0]
38.   escreval("Normal Fraco: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)
39.
40.   ordena(porcentagensTodosOsEstadosOrdenadas)
41.   media←0
42.   mediana←0
43.   q3←0
44.   q1←0
45.   maximo←0
46.   minimo←0
47.   Para inteiro i ← 0 ate i < tamanho(porcentagensTodosOsEstadosOrdenadas) passo +1 faca
48.     media ← media + porcentagensTodosOsEstadosOrdenadas[i]
49.   fimPara
50.   media ← media/1000
51.   q3 ← porcentagensTodosOsEstadosOrdenadas[((1000/4)*3)]
52.   mediana ← (porcentagensTodosOsEstadosOrdenadas[((1000/2)-1)] + porcentagensTodosOsEstadosOrdenadas[(1000/2)])/2
53.   q1 ← porcentagensTodosOsEstadosOrdenadas[((1000/4)-1)]
54.   maximo←porcentagensTodosOsEstadosOrdenadas[999]
55.   minimo←porcentagensTodosOsEstadosOrdenadas[0]
56.   escreval("Todos os Estados: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)
57.
58.   ordena(porcentagensTodasAsTransicoesOrdenadas)
59.   media←0
60.   mediana←0
61.   q3←0
62.   q1←0
63.   maximo←0
```

```
64.  minimo←0
65.  Para inteiro i ← 0 ate i < tamanho(porcentagensTodasAsTransicoesOrdenadas) passo +1 faca
66.    media ← media + porcentagensTodasAsTransicoesOrdenadas[i]
67.  fimPara
68.  media ← media/1000
69.  q3 ← porcentagensTodasAsTransicoesOrdenadas[((1000/4)*3)]
70.  mediana ← (porcentagensTodasAsTransicoesOrdenadas[((1000/2)-1)] + porcentagensTodasAsTransicoesOrdenadas[(1000/2)])/2
71.  q1 ← porcentagensTodasAsTransicoesOrdenadas[((1000/4)-1)]
72.  maximo←porcentagensTodasAsTransicoesOrdenadas[999]
73.  minimo←porcentagensTodasAsTransicoesOrdenadas[0]
74.  escreval("Toda as Transições: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)
75.
76.
77.
78.
79.
```

Autoria própria.

Depois de explicado o funcionamento do *software*, vamos agora para os resultados das simulações. Como nossos modelos são parciais, o oráculo deve ser definido pelo testador nos passos finais de interação que sucedem a rotação.

5.2 Execução dos Testes Aleatórios

Como o *software* produzido executa 1000 repetições de geração de casos de teste, para gerarmos os gráficos que serão mostrados abaixo, executamos o *software* 9 vezes, com configuração de tempos diferentes, respectivamente 1, 5, 10, 15, 20, 25, 30, 35 e 40 milissegundos[5]. Ao final das execuções, tivemos, para cada configuração de tempo, uma lista de porcentagens de cobertura para cada critério. E, de posse dessas listas, ordenamos e pegamos os dados necessários para a geração dos gráficos box-plot de porcentagens de cobertura de cada um dos critérios por tempo configurado.

A Tabela 4 mostra alguns dados utilizados nas análises posteriores sobre os testes gerados, aleatoriamente, em relação ao tempo configurado. A coluna Tempo está em milissegundos; a coluna Média é a quantidade média de testes executados; e a coluna Máximo mostra a quantidade máxima de testes executados dentre as 1000 execuções.

[5] A escolha destes valores se deve pelo tempo necessário para gerar a quantidade de testes que fosse possível atingir a cobertura total dos critérios estudados

Tabela 4 - Dados dos testes aleatórios por tempo de configuração

Tempo	Média[6]	Máximo
1	2,31	4
5	9,05	12
10	18,76	22
15	27,39	32
20	37,52	43
25	46,73	53
30	55,61	64
35	65,99	73
40	75,76	84

Autoria própria.

5.2.1 Cobertura do Critério Normal Forte

Devido às características do critério Normal Forte, já mencionadas na Seção 2.1.4.2, há a necessidade de realização de todas as possibilidades de testes, que seriam 256 testes (testes possíveis e testes impossíveis), como retratado na Tabela 9. Já que nosso estudo mostrou não ser possível a realização dos testes destacados (testes impossíveis) da Tabela 9, resta apenas 64 testes (Tabela 10). Podemos calcular que a porcentagem máxima teórica de cobertura desse critério seria de 25% (eliminando-se os testes impossíveis). Na Tabela 5 podemos ver os percentuais de cobertura gerados pelo *software* para o critério Normal Forte que deram origem ao gráfico da Figura 19.

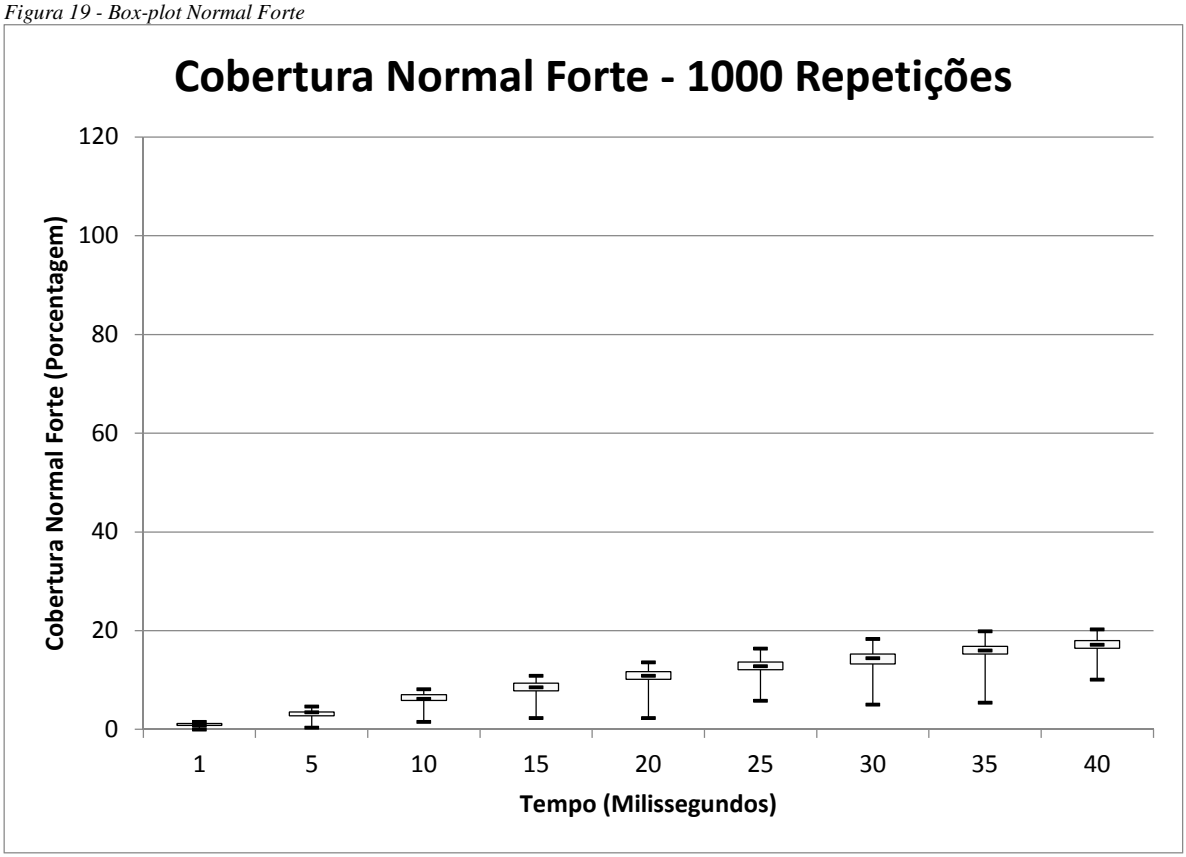
Tabela 5 - Dados gerados pelo software para gerar o gráfico box-plot Normal Forte

	Máximo	Q3	Mediana	Q1	Mínimo
1	1,56	1,17	0,78	0,78	0
5	4,69	3,52	3,52	2,73	0,39
10	8,20	7,03	6,25	5,86	1,56
15	10,94	9,38	8,59	7,81	2,34
20	13,67	11,72	10,94	10,16	2,34
25	16,41	13,67	12,89	12,11	5,86
30	18,36	15,23	14,45	13,28	5,08
35	19,92	16,80	16,02	15,23	5,47
40	20,31	17,97	17,19	16,41	10,16

Autoria própria.

[6] Em 1000 execuções de, digamos, 5ms, a quantidade de testes que conseguem rodar variam. Por isso coletamos estas quantidades e tiramos a média por 1.000

Figura 19 - Box-plot Normal Forte



Autoria própria.

Analisando o gráfico gerado, podemos ver que a cobertura aumenta à medida que o tempo de execução de testes também aumenta, como já era de se esperar uma vez que, quanto mais testes melhor a chance de cobertura. Era esperada uma cobertura baixa, pois conforme características supracitadas, o critério Normal Forte é o critério mais difícil de cobrir e, já que a porcentagem máxima teórica de cobertura desse critério seria de 25%, isso só seria alcançado se fossem realizados, no mínimo, os 64 testes da Tabela 10, gerada pelo nosso estudo. Analisando os dados da Tabela 4 é possível inferir que com o tempo de 35 milissegundos, já tem-se uma média de 65,99 testes, e já seria possível alcançar a cobertura máxima teórica. Nota-se, através dos dados da Tabela 5, que o máximo de cobertura alcançado foi de 20,31%, quando o tempo estava configurado em 40 milissegundos. Nota-se ainda, através da análise do 3º quartil, que mesmo na melhor configuração de tempo (40 milissegundos), 75% das execuções alcançam 17,97% de cobertura. Assim podemos dizer que aleatoriamente, a cobertura gerada não é eficiente. Esses dados só corroboram com a importância de um planejamento prévio.

5.2.2 Cobertura do Critério Normal Fraco

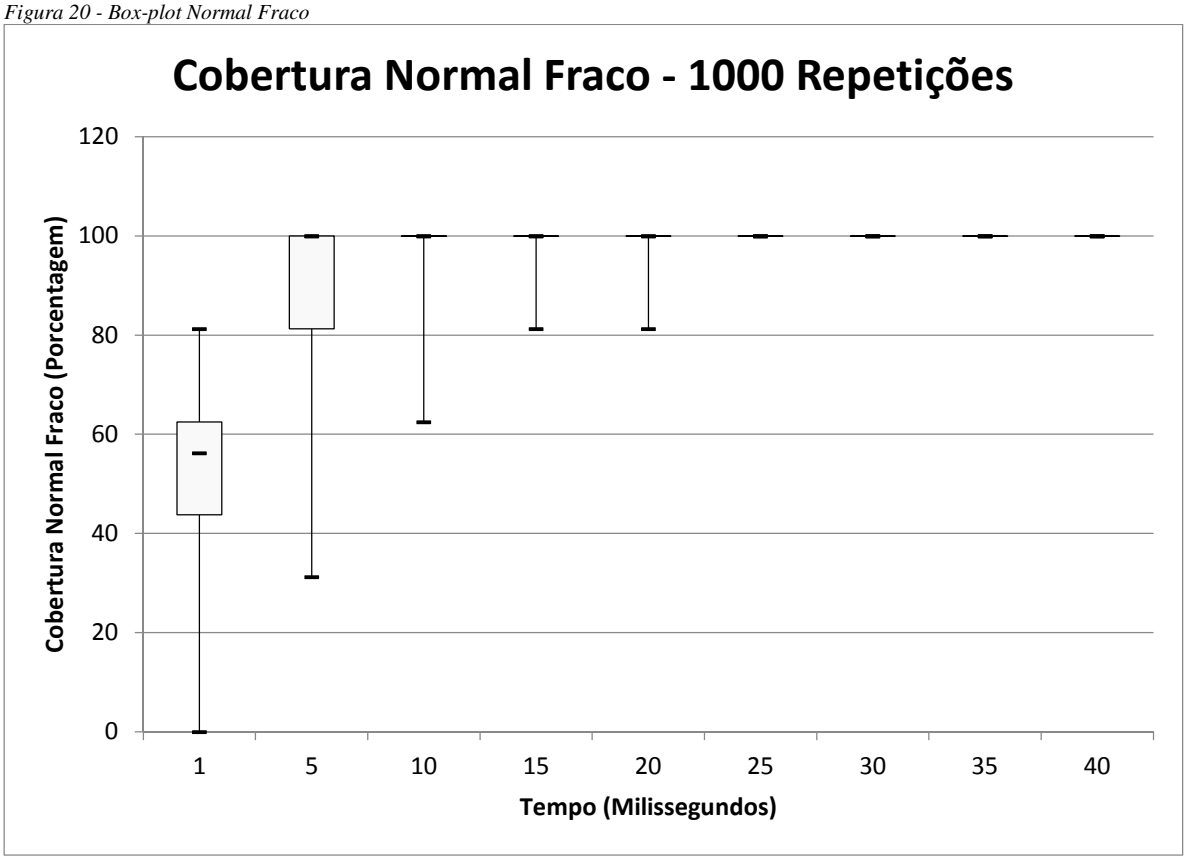
O critério Normal Fraco pode chegar à porcentagem máxima de 100%. Na Tabela 6, podemos ver os dados gerados pelo *software* em relação ao critério Normal Fraco que deram origem ao gráfico da Figura 20.

Tabela 6 - Dados gerados pelo software para gerar o gráfico box-plot Normal Fraco

	Máximo	Q3	Mediana	Q1	Mínimo
1	81,25	62,50	56,25	43,75	0
5	100,00	100,00	100,00	81,25	31,25
10	100,00	100,00	100,00	100,00	62,50
15	100,00	100,00	100,00	100,00	81,25
20	100,00	100,00	100,00	100,00	81,25
25	100,00	100,00	100,00	100,00	100,00
30	100,00	100,00	100,00	100,00	100,00
35	100,00	100,00	100,00	100,00	100,00
40	100,00	100,00	100,00	100,00	100,00

Autoria própria.

Figura 20 - Box-plot Normal Fraco



Autoria própria.

Analisando o gráfico da Figura 20 junto com os dados da Tabela 6 verifica-se que a cobertura máxima alcançada a partir de 5 milissegundos já é de 100%. Porém, podemos notar, analisando também a Tabela 4, que, com o tempo de 5 milissegundos, uma vez que nesse tempo tem-se um média de 9,05 testes, já há testes mais que suficiente (o máximo de testes numa

repetição chegou a 12) para alcançar a cobertura máxima de 100%. Mas podemos perceber (através da análise do 1º quartil) que 25% das execuções atingiram 81,25%. Para as demais configurações de tempo, a cobertura é ótima. Apesar de coberturas ótimas, podemos dizer que, aleatoriamente, a cobertura gerada não é a mais eficiente, pois nosso estudo mostra na Seção 4.2 que, com apenas 4 testes bem elaborados, pode-se chegar a 100% de cobertura do critério Normal Fraco.

5.2.3 Cobertura do Critério Todos os Estados

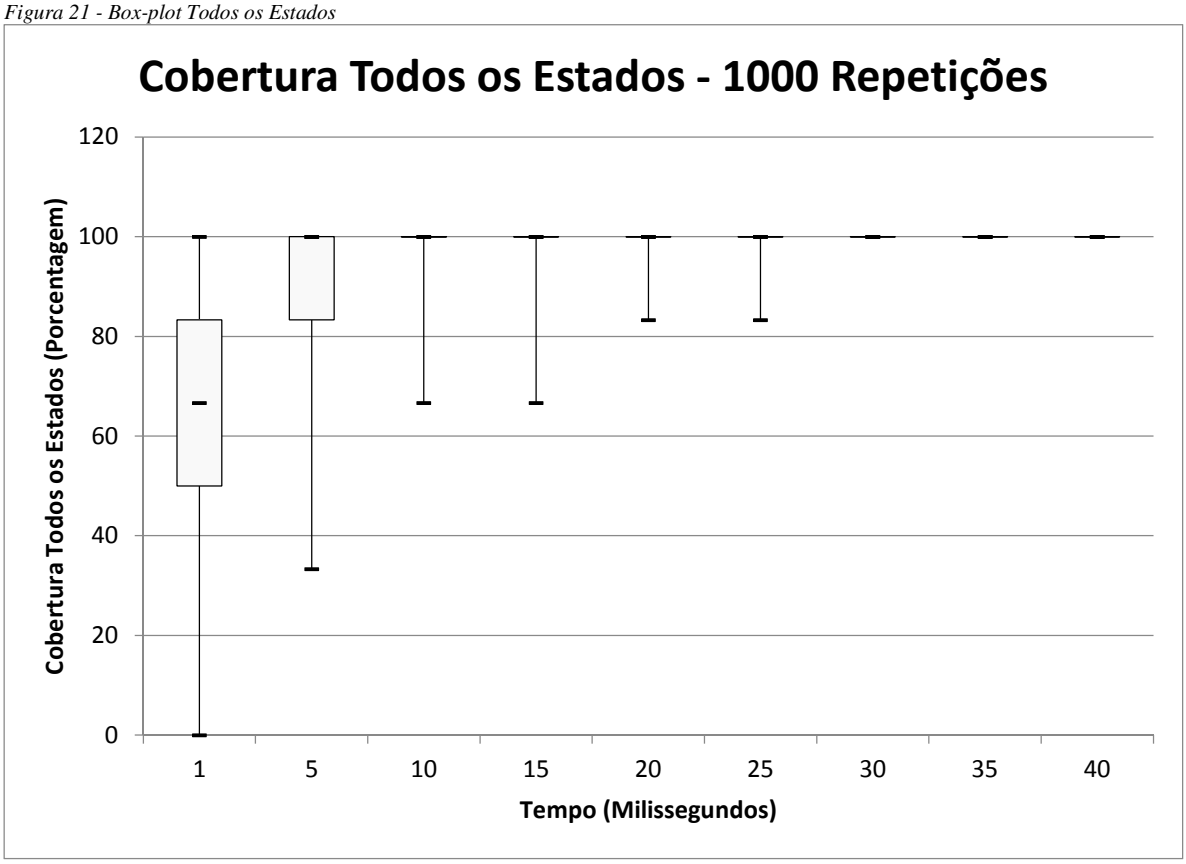
O critério Todos os Estados é bastante fácil de ser coberto em sua totalidade, conforme já vimos na Seção 4.3. A porcentagem máxima teórica de cobertura desse critério é de 100%. Na Tabela 7, podemos ver os dados gerados pelo *software* para o critério Todos os Estados que deram origem ao gráfico da Figura 21.

Tabela 7 - Dados gerados pelo software para gerar o gráfico box-plot Todos os Estados

	Máximo	**Q3**	**Mediana**	**Q1**	**Mínimo**
1	100,00	83,33	66,67	50,00	0
5	100,00	100,00	100,00	83,33	33,36
10	100,00	100,00	100,00	100,00	66,67
15	100,00	100,00	100,00	100,00	66,67
20	100,00	100,00	100,00	100,00	83,33
25	100,00	100,00	100,00	100,00	83,33
30	100,00	100,00	100,00	100,00	100,00
35	100,00	100,00	100,00	100,00	100,00
40	100,00	100,00	100,00	100,00	100,00

Autoria própria.

Figura 21 - Box-plot Todos os Estados



Autoria própria.

Os dados da Tabela 7 junto com o gráfico box-plot da Figura 21 expõem que houve o máximo de 100% de cobertura a partir do tempo de 5 milissegundos. Porém, com o tempo de 5 milissegundos (analisando o 1º quartil), vemos que 25% das execuções não chegaram a 84% de cobertura. E podemos notar, analisando a Tabela 4, que, com esse tempo, já temos uma média 9,05 testes, que seriam capaz de cobrir o critério em 100%. Para as demais configurações de tempo, a cobertura é ótima. Dessa forma, chegamos à conclusão que, aleatoriamente, a cobertura gerada também não é a mais eficiente, visto que nosso estudo mostra, na Seção 4.3, que, com apenas 3 testes bem elaborados, pode-se chegar a 100% de cobertura do critério Todos os Estados.

5.2.4 Cobertura do Critério Todas as Transições

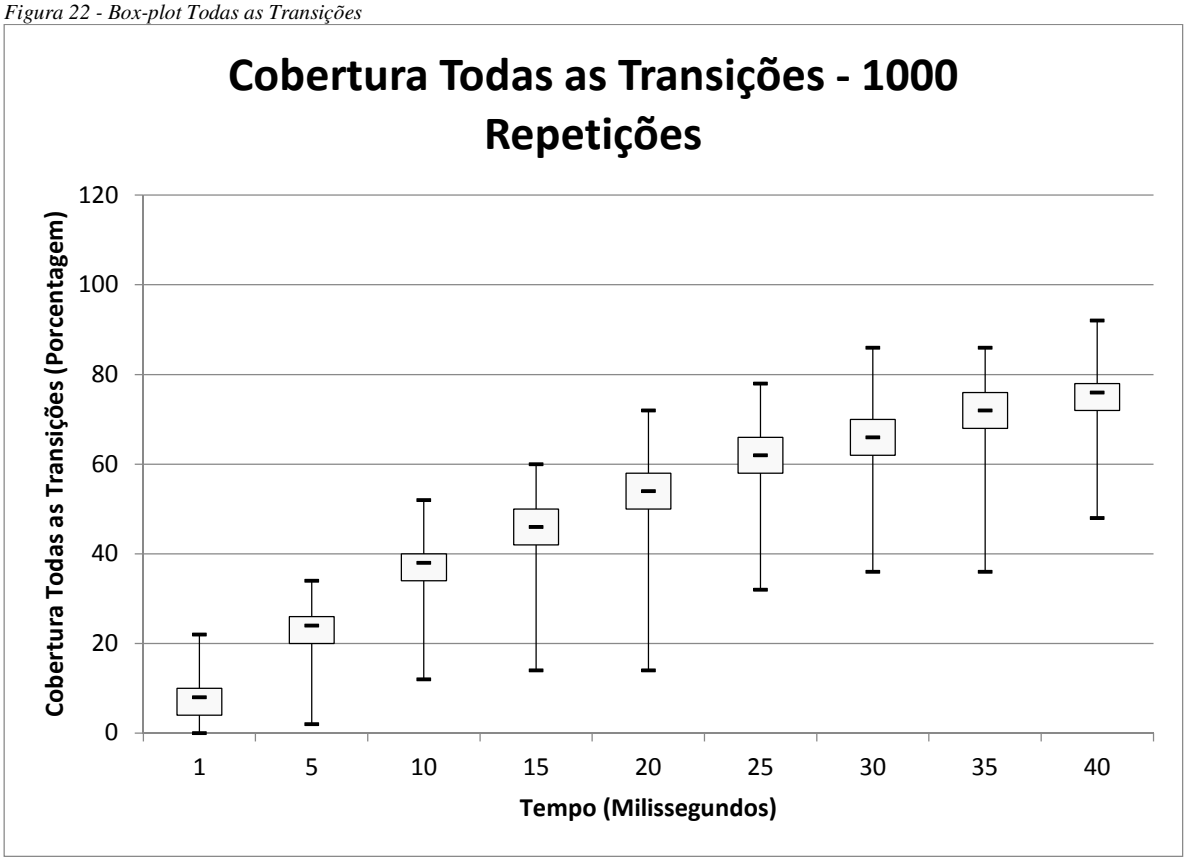
O critério Todas as Transições já é mais complexo para realizar cobertura total, apesar de sua porcentagem máxima teórico de cobertura ser de 100%. Na Tabela 8 podemos ver os dados gerados pelo *software* para o critério Todas as Transições que deram origem ao gráfico da Figura 22.

Tabela 8 - Dados gerados pelo software para gerar o gráfico box-plot Todas as Transições

	Máximo	Q3	Mediana	Q1	Mínimo
1	22,00	10,00	8,00	4,00	0
5	34,00	26,00	24,00	20,00	2,00
10	52,00	40,00	38,00	34,00	12,00
15	60,00	50,00	46,00	42,00	14,00
20	72,00	58,00	54,00	50,00	14,00
25	78,00	66,00	62,00	58,00	32,00
30	86,00	70,00	66,00	62,00	36,00
35	86,00	76,00	72,00	68,00	36,00
40	92,00	78,00	76,00	72,00	48,00

Autoria própria.

Figura 22 - Box-plot Todas as Transições



Autoria própria.

O box-plot da Figura 22, e os dados da Tabela 8 indicam que o máximo de cobertura alcançado foi de 92% com o tempo de 40 milissegundos, que seria uma boa cobertura. Entretanto, ao observarmos os dados da Tabela 4, percebemos que com o tempo de 25 milissegundos já teríamos testes suficientes (média de 46,73) para alcançarmos os 100% de cobertura, visto que nosso estudo mostra, na Seção 4.4, que, com pelo menos 42 testes, podemos chegar à cobertura total do critério Todas as Transições. Ainda podemos constatar que, nessa configuração de tempo (25 milissegundos), o máximo de cobertura alcançado foi de 78% e que 75% das execuções nem atingiram 67% de cobertura. E mesmo com a melhor configuração (40 milissegundos) o máximo de cobertura alcançado foi de 92%. Assim, podemos dizer mais uma vez que, aleatoriamente, a cobertura gerada não é a mais eficiente.

5.3 Considerações Finais

Nesta seção descrevemos como um simulador de um testador aleatório, baseado na máquina de estado introduzida na Seção 3, foi desenvolvido. Também foram exibidos os resultados das simulações realizadas no estudo.

Chegamos à conclusão, através dos dados e gráficos analisados, que, aleatoriamente, a cobertura gerada não é a mais eficiente para nenhum dos critérios e que nosso estudo demonstra maneiras mais eficientes de chegar a uma maior cobertura. Ou seja, fica claro que há necessidade de maior cuidado e uso das técnicas de geração de testes para se conseguir um mínimo aceitável de cenários variados pois, aleatoriamente, nem mesmo gestos aparentemente simples como rotação Retrato e Paisagem são fáceis de testar.

6 TRABALHOS RELACIONADOS

Nesta seção são apresentados trabalhos relacionados com essa dissertação.

6.1 Using GUI ripping for automated testing of Android applications

Amalfitano et al. (2012) mostra a AndroidRipper, ferramenta que testa aplicações Android. A ferramenta implementa uma técnica automatizada, que testa aplicações através de sua interface gráfica. O objetivo da AndroidRipper não é desenvolver um modelo do aplicativo. Em vez disso, ela utiliza o ripping para automaticamente e, sistematicamente, percorrer a interface gráfica da aplicação, gerando e executando casos de teste à medida que novos eventos são encontrados. Os casos de teste são compostos por sequências de eventos que podem ser "disparados". E a geração dos casos de teste é baseada na análise dinâmica automática da interface gráfica que é executada para encontrar e disparar eventos da própria interface gráfica.

É exposto que os aspectos cruciais da técnica de análise dinâmica de interface gráfica incluem: o caminho e a ordem em que os eventos da interface são encontrados e disparados; pré-condições da aplicação e de seu ambiente no momento em que os eventos são disparados; os critérios utilizados para parar a exploração da interface analisada; o estado inicial da aplicação; e assim por diante. Logo, dependendo das opções de análise de interfaces gráficas específicas, são obtidos casos de teste com diferentes recursos de detecção de falhas. Consequentemente, a AndroidRipper baseia-se numa técnica de análise de interface gráfica configurável, realizada por um ripper, cujo comportamento pode ser ajustado, através de alguns parâmetros, de acordo com a aplicação sob teste e os objetivos dos testes. A AndroidRipper, ao explorar a interface gráfica da aplicação, detecta falhas de tempo de execução.

O autor avalia a eficácia de diferentes suítes de testes com resultados de várias configurações de parâmetros geradas pela AndroidRipper comparando as suítes de teste na aplicação Android de código aberto WordPress. Os resultados mostram a viabilidade e a relação custo-eficácia da abordagem. Além disso, é comparada a técnica com a abordagem de testes aleatórios implementada pelo Monkey, a partir das ferramentas de desenvolvimento do Android. Os resultados do trabalho mostraram que a AndroidRipper é mais eficaz do que o Monkey em detecção de falhas e cobertura de código.

O trabalho exposto, assim como o nosso, se preocupa com a interface de uma aplicação e também faz uso de testes aleatórios para comparações com a abordagem proposta. Porém, os

autores não analisam as características baseadas em gestos, nem desenvolvem um modelo da aplicação. Em vez disso, a ferramenta proposta percorre a interface gráfica, gerando e executando casos de teste.

6.2 A strategy to perform coverage testing of mobile applications

Delamaro, Vincenzi e Maldonato (2006) apresentam uma estratégia de apoio ao teste de cobertura de aplicações Java desenvolvidas para dispositivos móveis através da avaliação de cobertura estrutural. Tal processo leva em consideração que a aplicação possa ser testada não só em emuladores, mas também em seu ambiente real de execução, que é o próprio dispositivo móvel. É ressaltado que a estratégia apontada pode ser utilizada para detectar erros em aplicações móveis durante o processo de desenvolvimento (teste de unidade).

Os autores defendem que, com as limitações dos dispositivos móveis (desempenho, memória e armazenamento), não se torna prático a execução de uma ferramenta de teste diretamente no dispositivo. Por isso, propõem uma estratégia que permita que o teste seja realizado em plataforma cruzada, de forma que as informações sobre a execução de uma aplicação sejam coletadas e enviadas para a ferramenta de teste, que irá analisar e gerar relatórios sobre a cobertura dos casos de teste.

Os autores ainda expõem um ambiente que dá apoio à estratégia demonstrada. Esse ambiente é implementado em uma ferramenta, chamada JaBUTi/ME. E é exemplificada, através de um caso simples, como a ferramenta pode ser utilizada, apresentando o funcionamento da JaBUTi/ME na cobertura de teste em aplicações móveis.

Delamaro, Vincenzi e Maldonato (2006) retratam, tal qual nosso trabalho, uma estratégia de cobertura de aplicações móveis. Mas a estratégia utilizada pelos autores difere da nossa por ser uma estratégia voltada para cobertura estrutural com utilização de gráfico de fluxo de controle. Já a nossa utiliza classes de equivalência e máquina de estados para análise da cobertura de interfaces baseadas em gestos.

6.3 Coverage Criteria for GUI Testing

Memon, Soffa e Pollack (2001) apresentam novos critérios de cobertura para ajudar a determinar quando uma interface gráfica foi testada adequadamente. Os autores começam conceituando interface gráfica, componentes de uma interface gráfica e as classificações de

eventos. Depois da introdução destes conceitos, mostram que um componente de interface gráfica pode ser representado pelo gráfico de fluxo de eventos, que representa todas as possibilidades de interações entre eventos em um componente. Logo após, introduzem uma estrutura, desenvolvida por eles, para identificar a interação entre componentes, chamada de árvore de integração. Feito isso, partem para a definição dos critérios de cobertura, que são divididos em critérios de cobertura para eventos dentro de um componente (intra-componentes) e critérios de cobertura para eventos entre componentes (inter-componentes).

Ao expor os critérios de cobertura intra-componentes, são definidos vários critérios de cobertura, como: cobertura de evento, que requer que cada componente de um evento seja realizado pelo menos uma vez; cobertura interação-evento, que verifica as interações entre todos os possíveis pares de eventos do componente; e cobertura sequência-evento tamanho-n, que captura o impacto do contexto, que nada mais é do que a sequência de n eventos realizados antes do evento em questão. Da mesma forma, ao expor os critérios de cobertura inter-componentes, são definidos: cobertura de invocação, que requer que cada evento de foco restrito na interface gráfica seja realizado pelo menos uma vez; cobertura invocação-terminação, que checa quando um componente pode ser invocado e terminado; e cobertura sequência-evento tamanho-n inter-componente, que requer o teste de todas as sequências de eventos que começam com um evento em um componente e terminam com um evento em outro componente. Os autores também expõem a hierarquia de dominação dos critérios apresentados.

E, por fim, é apresentado um estudo de caso que ilustra a utilidade do relatório de cobertura para orientar testes adicionais e uma importante correlação entre a cobertura baseada em eventos de uma interface gráfica e a cobertura de comandos do código da interface.

No trabalho acima, bem como no nosso trabalho, são apresentados critérios de cobertura para ajudar na adequação de testes de interface, e também é exposta a hierarquia de dominação entre esses critérios. Porém, diferentemente do nosso trabalho, os autores não levam em consideração as mudanças de orientações causadas por gestos.

6.4 PUMA: Programmable UI-Automation for Large-Scale Dynamic Analysis of Mobile Apps

Hao (2014) descreve a criação e implementação da PUMA, uma estrutura para análise dinâmica de aplicações móveis em larga escala.

Os autores descrevem que as aplicações móveis iniciam em uma página inicial, com um ou mais elementos de interface gráfica (botões, caixas de texto e outros elementos) e que a interação dos usuários com estes elementos os conduzem para outras páginas, que por sua vez possuem outros elementos de interface gráfica. E que, a análise dinâmica tem como objetivo navegar por todas as páginas e analisar os estados interno da aplicação, bem como as saídas de cada página.

PUMA é composto por uma ferramenta monkey genérica e expõe uma abstração de programação dirigida a evento. As análises escritas para a PUMA são realizadas através de uma linguagem de script, PUMAScript, que é uma extensão da linguagem Java. O usuário pode personalizar a exploração de aplicações, já que é possível escrever manipuladores de eventos compactos, que separam a lógica de análise da lógica de exploração. A estrutura foi criada para executar a análise dinâmica tanto em emuladores, quanto em dispositivos móveis. Ao final da execução, a PUMA pode produzir logs que contêm saídas especificadas, além das saídas da ferramenta monkey.

É apresentado o projeto da PUMA, sua implementação para Android e uma avaliação executada com a PUMA em 3.600 aplicações da Google Play Store. A avaliação realizada levou em consideração propriedades de desempenho, segurança e correção das aplicações móveis e revelou bastante informações sobre o ecossistema Android.

Em Hao (2014), tal como no nosso trabalho, há um interesse pelas aplicações móveis e pelo teste de suas interfaces gráficas. No entanto, os autores não investigam as peculiaridades referentes às mudanças de orientações baseadas em gestos.

6.5 Automating GUI Testing for Android Applications

Hu e Neamtiu (2011) exibem um estudo de erros para compreender a natureza e possíveis soluções para erros em aplicações móveis. Para identificar as categorias de erros mais frequentes, realizaram um estudo empírico com as 10 aplicações mais populares do repositório oficial para aplicações Android. Para serem escolhidas, as aplicações teriam que: ser popular, ter um longo tempo de vida, ter um histórico detalhado de erros e ter o código disponível.

Os autores fizeram uso de uma abordagem de análise dinâmica que combina algumas técnicas de ferramentas de automação de casos de teste com monitoramento de tempo de execução, geração automática de eventos e análise de arquivo de log para encontrar erros. E defendem que a técnica apresentada tem o potencial de ajudar os desenvolvedores a re-descobrir

erros existentes, assim como achar novos erros, aumentando com isso a qualidade das aplicações Android.

É exibido como construir uma estrutura de testes automatizada para aplicações Android, com foco em erros de interface gráfica, visando garantir a confiabilidade das aplicações Android.

No trabalho acima, bem como no nosso trabalho, é observada a interface de aplicações móveis, e são realizados estudos empíricos. Entretando, Hu e Neamtiu (2011) apresentam um estudo de erros, através de analise dinâmica, realizada em 10 aplicações. Além disso, as observações restringem-se às interfaces gráficas sem levar em consideração a mudança de orientação baseada em gestos.

6.6 A Grey-box Approach for Automated GUI-Model Generation of Mobile Applications

Os autores Yang, Prasad e Xie (2013) defendem que as aplicações móveis geralmente são desenvolvidas em projetos de pequena escala, que podem não suportar testes manuais extensos e caros, por isso é particularmente importante desenvolver ferramentas de teste automatizadas para aplicações móveis.

No trabalho, é introduzida uma nova abordagem para extração automática de modelo de uma determinada aplicação móvel. Nessa abordagem, a análise estática extrai o conjunto de eventos suportados pela interface gráfica da aplicação. E, em seguida, um rastreamento dinâmico faz a engenharia reversa do modelo da aplicação através da execução sistemática dos eventos na aplicação em execução.

Também é apresentada a Orbit, uma ferramenta implementando a nova abordagem na plataforma Android. E exibida uma avaliação empírica desta ferramenta em várias aplicações Android, demonstrando seu funcionamento.

Os autores defendem que a Orbit pode extrair eficientemente modelos compactos, ainda que razoavelmente abrangentes e de alta qualidade para as aplicações móveis. E que, já que a Orbit foi projetada especificamente para aplicações móveis e usa um modelo baseado em estados, também se integraria bem com outras técnicas de teste baseadas em estados.

Em Yang, Prasad e Xie (2013), tal qual em nosso trabalho, é utilizado um modelo da aplicação móvel. Porém, ao contrátrio do nosso, os autores fazem a extração automática do

modelo da aplicação. E, não levam em consideração as mudanças de orientação baseadas em gestos.

6.7 AR(m)obo Test: um braço robótico para suporte à testes automáticos de retrato e paisagem para smartphones

Em Barboza (2016), o autor defende que as execuções de testes de rotação e translação eventualmente têm que ser feitas no aparelho e não em emuladores. E que o trabalho manual é mais caro e passível de perdas, como erros de execução e danos à saúde do indivíduo. Por isso, propõe-se o desenvolvimento e utilização de um braço robótico articulado e de uma aplicação Android capaz de controlar e apurar as informações de angulação do braço robótico e do dispositivo móvel em teste, com o objetivo de testar o comportamento do *software* e do *hardware* durante as mudanças de orientação para os modos retrato e paisagem.

O projeto do braço robótico é modelado através de diagramas SysML e implementado integrando tecnologias como Android, servomotores, Bluetooth e Arduino. E propõe a apoiar a realização de testes automáticos aplicados em smartphones e tabletes embarcados com sensores como acelerômetro e giroscópio, garantindo que as aplicações desenvolvidas respondam de forma adequada às mais variadas orientações do dispositivo.

No trabalho, foram propostas métricas para o projeto. Estas métricas também são aplicáveis a qualquer projeto de automação de teste de caixa-preta robotizada. O autor estimou as medições destas métricas para o AR(m)obo Test e concluiu que é possível introduzir manipuladores robóticos para validar o comportamento do *software* através das mais variadas mudanças de orientação do smartphone de uma maneira lógica, segura, precisa e com retorno ao investimento em poucos meses.

Este trabalho de Barboza (2016) é complementar ao nosso, pois provê uma forma mecanizada de executar os testes de rotação propostos no nosso projeto.

6.8 Considerações Finais

Podemos ver, com a apresentação destes trabalhos, que testar aplicações a partir de suas interfaces não é uma solução trivial e que a abordagem proposta pela maioria dos autores, diferentemente da nossa abordagem, não leva em consideração a mudança de orientação da aplicação. A única exceção é do trabalho de Barboza (2016), que, apesar de levar em

consideração a mudança de orientação, o faz em uma perpectiva diferente da nossa, utilizando um braço robótico para testar o comportamento do *software* e do *hardware* durante as mudanças.

Podemos perceber ainda que, testar aplicações beseadas em suas interfaces, envolve múltiplas variáveis e condições. E que, apesar de se ter muitos trabalhos sobre o tema, poucos abordam interfaces mais elaboradas baseadas em gestos, mesmo considerando gestos aparentemente simples como rotação Retrato e Paisagem.

7 CONCLUSÕES E TRABALHOS FUTUROS

Esta seção mostra as conclusões deste trabalho, pontuando dificuldades e limitações encontradas, bem como recomendações para trabalhos posteriores.

As interfaces com o usuário têm evoluído significativamente. Os dispositivos móveis, por exemplo, não se limitam mais às interfaces gráficas. Gestos, como rotações, translações e shakes fazem parte das novas interfaces. Neste trabalho, investigamos como a técnica de classes de equivalência, uma técnica tradicional de teste de *software*, poderia ser utilizada para modelar os casos de testes para rotação retrato e paisagem. Depois, propomos a modelagem através de uma máquina de estados para gerar testes de rotação baseado na cobertura da máquina. Então definimos uma hierarquia de dominação de cobertura combinando classes de equivalência com máquina de estados. E, por fim, implementamos um simulador de teste de rotação aleatório e medimos, nestas simulações, o nível de cobertura alcançado.

Os resultados das simulações mostram que, no critério Normal Forte, a cobertura não ultrapassou 20,31%, e 75% das amostras nem chegaram a 17,97%. No critério Normal Fraco, apesar de ter cobertura de 100%, em um tempo mais que suficiente para cobrir 100%, 25% das amostras nem ultrapassam 82%. No critério Todos os Estados, a máxima cobertura alcançada também foi 100%, porém mais uma vez, em um tempo que tem 100% de cobertura, 25% das amostras nem chegaram a 84%. E, no critério Todas as Transições, a cobertura máxima foi 92%, porém, em um tempo mais que suficiente para termos 100%, a cobertura máxima foi de 78%, sendo que 75% das amostras não chegaram nem a 67% de cobertura. Logo, aleatoriamente, a cobertura gerada não é a mais eficiente para nenhum dos critérios e nosso estudo demonstra maneiras mais eficientes de chegar a uma maior cobertura, apesar de não tratar da eficácia dessas coberturas na descoberta de defeitos. Ou seja, fica claro que há necessidade de maior cuidado e uso das técnicas de geração de testes para se conseguir um mínimo aceitável de cenários variados pois, aleatoriamente, nem mesmo gestos aparentemente simples como rotação Retrato e Paisagem são fáceis de testar. Por isso concluímos que, mesmo interações aparentemente simples, não devem ser subestimados em relação aos testes. Nossa modelagem formal e geração sistemática revelou que um teste manual ad-hoc sempre deixa escapar alguns cenários.

Nas próximas seções serão apresentadas algumas dificuldades e limitações identificadas na realização deste trabalho, assim como recomendações para trabalhos futuros.

7.1 Dificuldades e Limitações

Algumas dificuldades foram detectadas na realização deste trabalho. Foram elas:

- O processo para identificar cada classe de equivalência proposta que, por se tratar de um processo heurístico e exploratório, demandou bastante tempo;
- O grande volume (inesperado, a priori) de possibilidades de execução antes de identificarmos as classes de equivalência propostas tornou o processo de definição destas classes de difícil execução e bastante repetitivo;
- E a falta de experiência em programação de simuladores dificultou e aumentou o tempo gasto na codificação do *software* que realizou as simulações aleatórias, que também geraram os dados para a plotagem dos gráficos necessários para a pesquisa.

Dentre as limitações, podemos citar:

- Faltou uma análise da eficácia das coberturas estudadas na descoberta de defeitos;
- O nível de detalhamento das classes de equivalência capturam regiões iniciais e finais, ficando a critério do testador como executar essa transição;
- A velocidade de rotação tanto nas classes de equivalência quanto na máquina de estados não captura variação de velocidade;
- O simulador foi baseado na máquina de estados, ao invés de um modelo independente e com granularidade de eventos mais fina que os modelos propostos.

7.2 Recomendações para trabalhos Futuros

Apesar de termos alcançado o objetivo da pesquisa, vários outros trabalhos podem ser realizados para dar continuidade ao estudo apresentado. Dentre os quais podemos citar:

- Avaliar a eficácia das coberturas analisadas na detecção de defeitos em softwares que utilizam gestos;
- Executar estudos práticos para verificar quais ferramentas descobrem mais defeitos em softwares que utilizem gestos;
- Propor outros critérios de cobertura;
- Propor novos modelos de rotação Retrato e Paisagem, inclusive com transições cujas velocidades são variáveis;
- Realizar os testes e gerar modelos para outros dispositivos (tabletes, vídeo games portáteis, etc.) que possuem as mesmas características de interface, visando uma generalização do comportamento das interfaces testadas;

- Propor novos modelos para outros tipos de gestos.

REFERÊNCIAS

AMALFITANO, D.; FASOLINO , A. R.; TRAMONTANA, P.; CARMINE, S.; MEMON, A. Using GUI ripping for automated testing of Android applications. In: 27th IEEE/ACM International Conference on Automated Software Engineering. **Anais**. 2012. p. 258-261.

APFELBAUM, L.; DOYLE, J. Model based testing. In: Software Quality Week Conference. **Anais**. 1997. p. 293-300.

ARCURI, A.; BRIAND, L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: 33rd IEEE/ACM International Conference on Software Engineering. **Anais**. 2011. p. 1-10.

BARBOZA, L. **AR(m)obo Test**: um braço robótico para suporte à testes automáticos de retrato e paisagem para smartphones. Dissertação de Mestrado. Recife: Universidade Federal de Pernambuco, 2016.

BEIZER, B. **Software Testing Techniques**. 2. ed. New York: Van Nostrand Reinhold Company, 1990.

BINDER, R. V. **Testing object-oriented systems**: models, patterns, and tools. Addison-Wesley, 1999.

BOURHFIR, C.; DSSOULI, R.; ABOULHAMID, E. M. Automatic test generation for EFSM-based systems. **Relatório técnico IRO**, v. 1043, University of Montreal, 1996.

BURNSTEIN, I. **Practical Software Testing**. New York: Springer-Verlag, 2003.

CANALYS. **Smart phones overtake client PCs in 2011**. Canalys, 2012. Disponível em: <https://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011>. Acesso em: 18 Junho 2017.

CHOW, T. S. Testing software design modeled by finite-state machines. **IEEE Transactions on Software Engineering**, v. 4, n. 3, 1978, p. 178-187.

COLANZI, T. E. **Uma abordagem integrada de desenvolvimento e teste de software baseada na UML**. Dissertação de Mestrado. São Carlos: Universidade de São Paulo, Instituto de Ciências Matemáticas e Computação, 1999.

COPELAND, L. **A practitioner's guide to software test design**. Artech House, 2004.

DEITEL, P.; DEITEL, H. **Java How to program**. Prentice Hall Press, 2011.

DELAMARO, M. E.; MALDONATO, J. C.; JINO, M. **Introdução ao teste de software**. Rio de Janeiro: Campus Elsevier, 2007.

DELAMARO, M. E.; VINCENZI, A. M. R.; MALDONADO, J. C. A strategy to perform coverage testing of mobile applications. In: International Workshop on Automation of Software Test. **Anais**. 2006. p. 118-124.

FRANKL, P. G. **Use of data flow information for the selection and evaluation of software test data**. PhD Thesis. University of New York, New York, 1987.

FRANKL, P. G..; WEYUKER, E. J. An applicable family of data flow testing criteria. **IEEE Transactions on Software Engineering**, v. 14, n. 10, Outubro 1988. p. 1483-1498.

FUJIWARA, S.; BOCHMANN, G. V.; KHENDEK, F.; AMALOU, M.; GHEDAMSI, A. Test selection based on finite state models. **IEEE Transactions on Software Engineering**, v. 17, n. 6, Junho, 1991. p. 591-603

GARTNER. **Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017**. Gartner, 2017. Disponível em: <http://www.gartner.com/newsroom/id/3725117>. Acesso em: 18 Junho 2017.

GILL, A. **Introduction to the theory of finite-state machines**. New York: McGraw-Hill, 1962.

GONENC, G. A method for the design of fault detection experiments. **IEEE Transactions on Computers**, v. 19, n. 6, 1970. p. 551-558.

GÖREN, S.; FERGUSON, F. J. CHESMIN: a heuristic for state reduction in imcompletely specified finite state machines. In: Design, Automation and Test in Europe Conference and Exhibition. **Anais**. 2002. p. 248-254.

HAO, S.; LIU, B.;NATH S.; HALFOND, W. G.;GOVINDAN, R. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps. In: 12th Annual International Conference on Mobile Systems, Applications, and Services. **Anais**. 2014. p. 204-217.

HAREL, D.; POLITI, M. **Modeling reactive system with statecharts: the STATEMATE approach**. New York: McGraw-Hill, 1998.

HU, C.; NEAMTIU, I. Automating GUI testing for Android applications. In: 6th International Workshop on Automation of Software Test. **Anais**. 2011. p. 77-83.

JORGENSEN, P. C. **Software testing: a craftsman's approach**. Boca Raton, Florida, USA: CRC press, 1995.

KOHAVI, Z. **Switching and finite automata theory**. 2. ed. New York: McGraw-Hill, 1978.

MEMON, A. M.; SOFFA, M. L.; POLLACK, M. E. Coverage criteria for GUI testing. In: 8th European Software Engineering Conference e 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering. **Anais**. v. 26, n. 5, 2001. p. 256-267.

MORAES, A.; ANDRADE, W. L.; MACHADO, P. DL. A family of test selection criteria for timed input-output symbolic transition system models. **Science of Computer Programming**, v. 126, p. 52-72, 2016.

MYERS, G. J. **The art of software testing**. Atualização e revisão Tom Badgett e Todd M. Thomas com Corey Sandler. 2. ed. New Jersey: John Wiley & Sons Inc., 2004.

NAITO, S.; TSUNOYAMA, M. Fault detection for sequential machines by transition-tours. In: 11th IEEE Fault Tolerant Computer Systems. **Anais**. 1981. p. 238-243.

OFFUTT, J.; LIU S.; ABDURAZIK A.; AMMANN P.Generating test data from state-based specifications. In: The Journal of Software Testing, Verification, and Reliability. **Anais**. v. 13, n. 1, p. 25-53, 2003.

OSTRAND, T. J..; BALCER, M. J.. The category-partition method for specifying and generating fuctional tests. **Communications of the ACM**, v. 31, n. 6, 1988. p. 676-686.

PERRY, D. E.; KAISER, G. E. Adequate testing and object-oriented programming. **Journal on Object-Oriented Programming**, v. 2, n. 5, 1990. p. 13-19.

PEZZÈ, M.; YOUNG, M. **Teste e análise de software: processos, princípios e técnicas**. Porto Alegre: Bookman, 2008.

PRESSMAN, R. S. **Software Engineering**: a Practitioner's Approach. 5. ed. McGraw-Hill, 2001.

PRESSMAN, R. S. **Software Engineering**: A Practitioner's Approach. 6. ed. McGraw-Hill, 2005.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. 7. ed. McGraw Hill, 2011.

RADATZ, J.; GERACI, A.; KATKI, F. IEEE standard glossary of software engineering terminology. **IEEE Std 610.12-1990**, v. 121990, 1990.

ROBINSON, H. Finite state model-based testing on a shoestring. 1999 International Conference on Software Testing Analysis and Review Conference. **Anais**. San Jose, 1999.

ROBINSON, H. Intelligent test automation. **Software Testing and Quality Engineering**, v. 2, set. 2000. p. 24-33.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de software**: teoria e prática. São Paulo: Prentice Hall, 2001.

ROPER, M. **Software testing**. McGraw-Hill, 1994.

ROSARIA, S.; ROBINSON, H. Applying models in your testing process. **Information and software technology**, v. 42, n. 12, 2000. p. 815-824.

SABNANI, K.; DAHBURA, A. A protocol test generation procedure. **Computer Networks and ISDN systems**, v. 15, n. 4, 1988. p. 285-297.

SOMMERVILLE, I. **Engenharia de software**. Tradução de André Maurício de Andrade Ribeiro. Revisão técnica de Kechi Hirama. 6 ed. São Paulo: Addison Wesley, 2003.

SUGETA, T. **PROTEUM-RS/ST**: uma ferramenta para apoiar a validação de especificações statecharts baseada na análise de mutantes. Dissertação de Mestrado. São Carlos: Universidade de São Paulo, Instituto de Ciências Matemáticas e Computação, 1999.

VINCENZI, A. M. R. **Orientaçao a Objeto**: Definiçao, Implementaçao e Análise de Recursos de Teste e Validaçao. Tese de Doutorado. São Carlos: Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computaçao, 2004.

WEYUKER, E. J.; WEISS, S. N.; HAMLET, D. Comparison of program testing strategies. In: 4th Symposium on Software Testing, Analysis and Verification. **Anais**. 1991. p. 1-10.

YANG, W.; PRASAD, M. R.; XIE, T. A grey-box approach for automated GUI-model generation of mobile applications. In: International Conference on Fundamental Approaches to Software Engineering. **Anais**. 2013. p. 250-265.

APÊNDICE A – TABELA DE TESTES

Nas tabelas abaixo, são usadas as seguintes legendas:

A = 0 grau
B = 90 graus
C = 180 graus
D = 270 graus
Devagar = 30 graus/segundo
Rápida = 1.000 graus/segundo

Tabela 9 - Tabela completa de testes

Testes	**Entradas**					**Saída**
	Região inicial	Orientação inicial	Sentido de rotação	Velocidade	Região final	Orientação final
1	A	Retrato	Horário	Devagar	A	Retrato
2	A	Retrato	Horário	Devagar	B	PaisagemE
3	A	Retrato	Horário	Devagar	C	PaisagemE
4	A	Retrato	Horário	Devagar	D	PaisagemD
5	A	Retrato	Horário	Rápida	A	Retrato
6	A	Retrato	Horário	Rápida	B	PaisagemE
7	A	Retrato	Horário	Rápida	C	RetratoInv
8	A	Retrato	Horário	Rápida	D	PaisagemD
9	A	Retrato	Anti-horário	Devagar	A	Retrato
10	A	Retrato	Anti-horário	Devagar	B	PaisagemE
11	A	Retrato	Anti-horário	Devagar	C	PaisagemD
12	A	Retrato	Anti-horário	Devagar	D	PaisagemD
13	A	Retrato	Anti-horário	Rápida	A	Retrato
14	A	Retrato	Anti-horário	Rápida	B	PaisagemE
15	A	Retrato	Anti-horário	Rápida	C	RetratoInv
16	A	Retrato	Anti-horário	Rápida	D	PaisagemD
17	A	RetratoInv	Horário	Devagar	A	IMPOSSÍVEL
18	A	RetratoInv	Horário	Devagar	B	
19	A	RetratoInv	Horário	Devagar	C	
20	A	RetratoInv	Horário	Devagar	D	
21	A	RetratoInv	Horário	Rápida	A	
22	A	RetratoInv	Horário	Rápida	B	
23	A	RetratoInv	Horário	Rápida	C	
24	A	RetratoInv	Horário	Rápida	D	
25	A	RetratoInv	Anti-horário	Devagar	A	
26	A	RetratoInv	Anti-horário	Devagar	B	

27	A	RetratoInv	Anti-horário	Devagar	C	
28	A	RetratoInv	Anti-horário	Devagar	D	
29	A	RetratoInv	Anti-horário	Rápida	A	
30	A	RetratoInv	Anti-horário	Rápida	B	
31	A	RetratoInv	Anti-horário	Rápida	C	
32	A	RetratoInv	Anti-horário	Rápida	D	
33	A	PaisagemD	Horário	Devagar	A	IMPOSSÍVEL
34	A	PaisagemD	Horário	Devagar	B	
35	A	PaisagemD	Horário	Devagar	C	
36	A	PaisagemD	Horário	Devagar	D	
37	A	PaisagemD	Horário	Rápida	A	
38	A	PaisagemD	Horário	Rápida	B	
39	A	PaisagemD	Horário	Rápida	C	
40	A	PaisagemD	Horário	Rápida	D	
41	A	PaisagemD	Anti-horário	Devagar	A	
42	A	PaisagemD	Anti-horário	Devagar	B	
43	A	PaisagemD	Anti-horário	Devagar	C	
44	A	PaisagemD	Anti-horário	Devagar	D	
45	A	PaisagemD	Anti-horário	Rápida	A	
46	A	PaisagemD	Anti-horário	Rápida	B	
47	A	PaisagemD	Anti-horário	Rápida	C	
48	A	PaisagemD	Anti-horário	Rápida	D	
49	A	PaisagemE	Horário	Devagar	A	IMPOSSÍVEL
50	A	PaisagemE	Horário	Devagar	B	
51	A	PaisagemE	Horário	Devagar	C	
52	A	PaisagemE	Horário	Devagar	D	
53	A	PaisagemE	Horário	Rápida	A	
54	A	PaisagemE	Horário	Rápida	B	
55	A	PaisagemE	Horário	Rápida	C	
56	A	PaisagemE	Horário	Rápida	D	
57	A	PaisagemE	Anti-horário	Devagar	A	
58	A	PaisagemE	Anti-horário	Devagar	B	
59	A	PaisagemE	Anti-horário	Devagar	C	
60	A	PaisagemE	Anti-horário	Devagar	D	
61	A	PaisagemE	Anti-horário	Rápida	A	
62	A	PaisagemE	Anti-horário	Rápida	B	
63	A	PaisagemE	Anti-horário	Rápida	C	

64	A	PaisagemE	Anti-horário	Rápida	D	
65	B	Retrato	Horário	Devagar	A	IMPOSSÍVEL
66	B	Retrato	Horário	Devagar	B	
67	B	Retrato	Horário	Devagar	C	
68	B	Retrato	Horário	Devagar	D	
69	B	Retrato	Horário	Rápida	A	
70	B	Retrato	Horário	Rápida	B	
71	B	Retrato	Horário	Rápida	C	
72	B	Retrato	Horário	Rápida	D	
73	B	Retrato	Anti-horário	Devagar	A	
74	B	Retrato	Anti-horário	Devagar	B	
75	B	Retrato	Anti-horário	Devagar	C	
76	B	Retrato	Anti-horário	Devagar	D	
77	B	Retrato	Anti-horário	Rápida	A	
78	B	Retrato	Anti-horário	Rápida	B	
79	B	Retrato	Anti-horário	Rápida	C	
80	B	Retrato	Anti-horário	Rápida	D	
81	B	RetratoInv	Horário	Devagar	A	IMPOSSÍVEL
82	B	RetratoInv	Horário	Devagar	B	
83	B	RetratoInv	Horário	Devagar	C	
84	B	RetratoInv	Horário	Devagar	D	
85	B	RetratoInv	Horário	Rápida	A	
86	B	RetratoInv	Horário	Rápida	B	
87	B	RetratoInv	Horário	Rápida	C	
88	B	RetratoInv	Horário	Rápida	D	
89	B	RetratoInv	Anti-horário	Devagar	A	
90	B	RetratoInv	Anti-horário	Devagar	B	
91	B	RetratoInv	Anti-horário	Devagar	C	
92	B	RetratoInv	Anti-horário	Devagar	D	
93	B	RetratoInv	Anti-horário	Rápida	A	
94	B	RetratoInv	Anti-horário	Rápida	B	
95	B	RetratoInv	Anti-horário	Rápida	C	
96	B	RetratoInv	Anti-horário	Rápida	D	
97	B	PaisagemD	Horário	Devagar	A	IMPOSSÍVEL
98	B	PaisagemD	Horário	Devagar	B	
99	B	PaisagemD	Horário	Devagar	C	
100	B	PaisagemD	Horário	Devagar	D	

101	B	PaisagemD	Horário	Rápida	A	
102	B	PaisagemD	Horário	Rápida	B	
103	B	PaisagemD	Horário	Rápida	C	
104	B	PaisagemD	Horário	Rápida	D	
105	B	PaisagemD	Anti-horário	Devagar	A	
106	B	PaisagemD	Anti-horário	Devagar	B	
107	B	PaisagemD	Anti-horário	Devagar	C	
108	B	PaisagemD	Anti-horário	Devagar	D	
109	B	PaisagemD	Anti-horário	Rápida	A	
110	B	PaisagemD	Anti-horário	Rápida	B	
111	B	PaisagemD	Anti-horário	Rápida	C	
112	B	PaisagemD	Anti-horário	Rápida	D	
113	B	PaisagemE	Horário	Devagar	A	Retrato
114	B	PaisagemE	Horário	Devagar	B	PaisagemE
115	B	PaisagemE	Horário	Devagar	C	PaisagemE
116	B	PaisagemE	Horário	Devagar	D	PaisagemD
117	B	PaisagemE	Horário	Rápida	A	Retrato
118	B	PaisagemE	Horário	Rápida	B	PaisagemE
119	B	PaisagemE	Horário	Rápida	C	PaisagemE
120	B	PaisagemE	Horário	Rápida	D	PaisagemD
121	B	PaisagemE	Anti-horário	Devagar	A	Retrato
122	B	PaisagemE	Anti-horário	Devagar	B	PaisagemE
123	B	PaisagemE	Anti-horário	Devagar	C	PaisagemD
124	B	PaisagemE	Anti-horário	Devagar	D	PaisagemD
125	B	PaisagemE	Anti-horário	Rápida	A	Retrato
126	B	PaisagemE	Anti-horário	Rápida	B	PaisagemE
127	B	PaisagemE	Anti-horário	Rápida	C	PaisagemE
128	B	PaisagemE	Anti-horário	Rápida	D	PaisagemD
129	C	Retrato	Horário	Devagar	A	
130	C	Retrato	Horário	Devagar	B	
131	C	Retrato	Horário	Devagar	C	
132	C	Retrato	Horário	Devagar	D	
133	C	Retrato	Horário	Rápida	A	IMPOSSÍVEL
134	C	Retrato	Horário	Rápida	B	
135	C	Retrato	Horário	Rápida	C	
136	C	Retrato	Horário	Rápida	D	
137	C	Retrato	Anti-horário	Devagar	A	

138	C	Retrato	Anti-horário	Devagar	B	
139	C	Retrato	Anti-horário	Devagar	C	
140	C	Retrato	Anti-horário	Devagar	D	
141	C	Retrato	Anti-horário	Rápida	A	
142	C	Retrato	Anti-horário	Rápida	B	
143	C	Retrato	Anti-horário	Rápida	C	
144	C	Retrato	Anti-horário	Rápida	D	
145	C	RetratoInv	Horário	Devagar	A	Retrato
146	C	RetratoInv	Horário	Devagar	B	PaisagemE
147	C	RetratoInv	Horário	Devagar	C	PaisagemE
148	C	RetratoInv	Horário	Devagar	D	PaisagemD
149	C	RetratoInv	Horário	Rápida	A	Retrato
150	C	RetratoInv	Horário	Rápida	B	PaisagemE
151	C	RetratoInv	Horário	Rápida	C	RetratoInv
152	C	RetratoInv	Horário	Rápida	D	PaisagemE
153	C	RetratoInv	Anti-horário	Devagar	A	Retrato
154	C	RetratoInv	Anti-horário	Devagar	B	PaisagemE
155	C	RetratoInv	Anti-horário	Devagar	C	PaisagemD
156	C	RetratoInv	Anti-horário	Devagar	D	PaisagemD
157	C	RetratoInv	Anti-horário	Rápida	A	Retrato
158	C	RetratoInv	Anti-horário	Rápida	B	PaisagemE
159	C	RetratoInv	Anti-horário	Rápida	C	RetratoInv
160	C	RetratoInv	Anti-horário	Rápida	D	PaisagemD
161	C	PaisagemD	Horário	Devagar	A	
162	C	PaisagemD	Horário	Devagar	B	
163	C	PaisagemD	Horário	Devagar	C	
164	C	PaisagemD	Horário	Devagar	D	
165	C	PaisagemD	Horário	Rápida	A	
166	C	PaisagemD	Horário	Rápida	B	
167	C	PaisagemD	Horário	Rápida	C	IMPOSSÍVEL
168	C	PaisagemD	Horário	Rápida	D	
169	C	PaisagemD	Anti-horário	Devagar	A	
170	C	PaisagemD	Anti-horário	Devagar	B	
171	C	PaisagemD	Anti-horário	Devagar	C	
172	C	PaisagemD	Anti-horário	Devagar	D	
173	C	PaisagemD	Anti-horário	Rápida	A	
174	C	PaisagemD	Anti-horário	Rápida	B	

175	C	PaisagemD	Anti-horário	Rápida	C	
176	C	PaisagemD	Anti-horário	Rápida	D	
177	C	PaisagemE	Horário	Devagar	A	IMPOSSÍVEL
178	C	PaisagemE	Horário	Devagar	B	
179	C	PaisagemE	Horário	Devagar	C	
180	C	PaisagemE	Horário	Devagar	D	
181	C	PaisagemE	Horário	Rápida	A	
182	C	PaisagemE	Horário	Rápida	B	
183	C	PaisagemE	Horário	Rápida	C	
184	C	PaisagemE	Horário	Rápida	D	
185	C	PaisagemE	Anti-horário	Devagar	A	
186	C	PaisagemE	Anti-horário	Devagar	B	
187	C	PaisagemE	Anti-horário	Devagar	C	
188	C	PaisagemE	Anti-horário	Devagar	D	
189	C	PaisagemE	Anti-horário	Rápida	A	
190	C	PaisagemE	Anti-horário	Rápida	B	
191	C	PaisagemE	Anti-horário	Rápida	C	
192	C	PaisagemE	Anti-horário	Rápida	D	
193	D	Retrato	Horário	Devagar	A	IMPOSSÍVEL
194	D	Retrato	Horário	Devagar	B	
195	D	Retrato	Horário	Devagar	C	
196	D	Retrato	Horário	Devagar	D	
197	D	Retrato	Horário	Rápida	A	
198	D	Retrato	Horário	Rápida	B	
199	D	Retrato	Horário	Rápida	C	
200	D	Retrato	Horário	Rápida	D	
201	D	Retrato	Anti-horário	Devagar	A	
202	D	Retrato	Anti-horário	Devagar	B	
203	D	Retrato	Anti-horário	Devagar	C	
204	D	Retrato	Anti-horário	Devagar	D	
205	D	Retrato	Anti-horário	Rápida	A	
206	D	Retrato	Anti-horário	Rápida	B	
207	D	Retrato	Anti-horário	Rápida	C	
208	D	Retrato	Anti-horário	Rápida	D	
209	D	RetratoInv	Horário	Devagar	A	IMPOSSÍVEL
210	D	RetratoInv	Horário	Devagar	B	
211	D	RetratoInv	Horário	Devagar	C	

212	D	RetratoInv	Horário	Devagar	D	
213	D	RetratoInv	Horário	Rápida	A	
214	D	RetratoInv	Horário	Rápida	B	
215	D	RetratoInv	Horário	Rápida	C	
216	D	RetratoInv	Horário	Rápida	D	
217	D	RetratoInv	Anti-horário	Devagar	A	
218	D	RetratoInv	Anti-horário	Devagar	B	
219	D	RetratoInv	Anti-horário	Devagar	C	
220	D	RetratoInv	Anti-horário	Devagar	D	
221	D	RetratoInv	Anti-horário	Rápida	A	
222	D	RetratoInv	Anti-horário	Rápida	B	
223	D	RetratoInv	Anti-horário	Rápida	C	
224	D	RetratoInv	Anti-horário	Rápida	D	
225	D	PaisagemD	Horário	Devagar	A	Retrato
226	D	PaisagemD	Horário	Devagar	B	PaisagemE
227	D	PaisagemD	Horário	Devagar	C	PaisagemE
228	D	PaisagemD	Horário	Devagar	D	PaisagemD
229	D	PaisagemD	Horário	Rápida	A	Retrato
230	D	PaisagemD	Horário	Rápida	B	PaisagemE
231	D	PaisagemD	Horário	Rápida	C	PaisagemD
232	D	PaisagemD	Horário	Rápida	D	PaisagemD
233	D	PaisagemD	Anti-horário	Devagar	A	Retrato
234	D	PaisagemD	Anti-horário	Devagar	B	PaisagemE
235	D	PaisagemD	Anti-horário	Devagar	C	PaisagemD
236	D	PaisagemD	Anti-horário	Devagar	D	PaisagemD
237	D	PaisagemD	Anti-horário	Rápida	A	Retrato
238	D	PaisagemD	Anti-horário	Rápida	B	PaisagemE
239	D	PaisagemD	Anti-horário	Rápida	C	PaisagemD
240	D	PaisagemD	Anti-horário	Rápida	D	PaisagemD
241	D	PaisagemE	Horário	Devagar	A	
242	D	PaisagemE	Horário	Devagar	B	
243	D	PaisagemE	Horário	Devagar	C	
244	D	PaisagemE	Horário	Devagar	D	IMPOSSÍVEL
245	D	PaisagemE	Horário	Rápida	A	
246	D	PaisagemE	Horário	Rápida	B	
247	D	PaisagemE	Horário	Rápida	C	
248	D	PaisagemE	Horário	Rápida	D	

249	D	PaisagemE	Anti-horário	Devagar	A	
250	D	PaisagemE	Anti-horário	Devagar	B	
251	D	PaisagemE	Anti-horário	Devagar	C	
252	D	PaisagemE	Anti-horário	Devagar	D	
253	D	PaisagemE	Anti-horário	Rápida	A	
254	D	PaisagemE	Anti-horário	Rápida	B	
255	D	PaisagemE	Anti-horário	Rápida	C	
256	D	PaisagemE	Anti-horário	Rápida	D	

Destes 256 testes apenas 64 são possíveis, pois nos outros 192 que estão em destaque, ocorre alguma incompatibilidade, a exemplo de: "O celular começar na região A, porém com a orientação inicial como PaisagemE, que seria impossível", logo podemos reduzir a tabela para:

Tabela 10 - Tabela compacta de testes, sem os testes impossíveis

Testes	**Entradas**					**Saída**
	Região inicial	Orientação inicial	Sentido de rotação	Velocidade	Região final	Orientação final
1	A	Retrato	Horário	Devagar	A	Retrato
2	A	Retrato	Horário	Devagar	B	PaisagemE
3	A	Retrato	Horário	Devagar	C	PaisagemE
4	A	Retrato	Horário	Devagar	D	PaisagemD
5	A	Retrato	Horário	Rápida	A	Retrato
6	A	Retrato	Horário	Rápida	B	PaisagemE
7	A	Retrato	Horário	Rápida	C	RetratoInv
8	A	Retrato	Horário	Rápida	D	PaisagemD
9	A	Retrato	Anti-horário	Devagar	A	Retrato
10	A	Retrato	Anti-horário	Devagar	B	PaisagemE
11	A	Retrato	Anti-horário	Devagar	C	PaisagemD
12	A	Retrato	Anti-horário	Devagar	D	PaisagemD
13	A	Retrato	Anti-horário	Rápida	A	Retrato
14	A	Retrato	Anti-horário	Rápida	B	PaisagemE
15	A	Retrato	Anti-horário	Rápida	C	RetratoInv
16	A	Retrato	Anti-horário	Rápida	D	PaisagemD
17	B	PaisagemE	Horário	Devagar	A	Retrato
18	B	PaisagemE	Horário	Devagar	B	PaisagemE
19	B	PaisagemE	Horário	Devagar	C	PaisagemE
20	B	PaisagemE	Horário	Devagar	D	PaisagemD
21	B	PaisagemE	Horário	Rápida	A	Retrato

22	B	PaisagemE	Horário	Rápida	B	PaisagemE
23	B	PaisagemE	Horário	Rápida	C	PaisagemE
24	B	PaisagemE	Horário	Rápida	D	PaisagemD
25	B	PaisagemE	Anti-horário	Devagar	A	Retrato
26	B	PaisagemE	Anti-horário	Devagar	B	PaisagemE
27	B	PaisagemE	Anti-horário	Devagar	C	PaisagemD
28	B	PaisagemE	Anti-horário	Devagar	D	PaisagemD
29	B	PaisagemE	Anti-horário	Rápida	A	Retrato
30	B	PaisagemE	Anti-horário	Rápida	B	PaisagemE
31	B	PaisagemE	Anti-horário	Rápida	C	PaisagemE
32	B	PaisagemE	Anti-horário	Rápida	D	PaisagemD
33	C	RetratoInv	Horário	Devagar	A	Retrato
34	C	RetratoInv	Horário	Devagar	B	PaisagemE
35	C	RetratoInv	Horário	Devagar	C	PaisagemE
36	C	RetratoInv	Horário	Devagar	D	PaisagemD
37	C	RetratoInv	Horário	Rápida	A	Retrato
38	C	RetratoInv	Horário	Rápida	B	PaisagemE
39	C	RetratoInv	Horário	Rápida	C	RetratoInv
40	C	RetratoInv	Horário	Rápida	D	PaisagemE
41	C	RetratoInv	Anti-horário	Devagar	A	Retrato
42	C	RetratoInv	Anti-horário	Devagar	B	PaisagemE
43	C	RetratoInv	Anti-horário	Devagar	C	PaisagemD
44	C	RetratoInv	Anti-horário	Devagar	D	PaisagemD
45	C	RetratoInv	Anti-horário	Rápida	A	Retrato
46	C	RetratoInv	Anti-horário	Rápida	B	PaisagemE
47	C	RetratoInv	Anti-horário	Rápida	C	RetratoInv
48	C	RetratoInv	Anti-horário	Rápida	D	PaisagemD
49	D	PaisagemD	Horário	Devagar	A	Retrato
50	D	PaisagemD	Horário	Devagar	B	PaisagemE
51	D	PaisagemD	Horário	Devagar	C	PaisagemE
52	D	PaisagemD	Horário	Devagar	D	PaisagemD
53	D	PaisagemD	Horário	Rápida	A	Retrato
54	D	PaisagemD	Horário	Rápida	B	PaisagemE
55	D	PaisagemD	Horário	Rápida	C	PaisagemD
56	D	PaisagemD	Horário	Rápida	D	PaisagemD
57	D	PaisagemD	Anti-horário	Devagar	A	Retrato
58	D	PaisagemD	Anti-horário	Devagar	B	PaisagemE

59	D	PaisagemD	Anti-horário	Devagar	C	PaisagemD
60	D	PaisagemD	Anti-horário	Devagar	D	PaisagemD
61	D	PaisagemD	Anti-horário	Rápida	A	Retrato
62	D	PaisagemD	Anti-horário	Rápida	B	PaisagemE
63	D	PaisagemD	Anti-horário	Rápida	C	PaisagemD
64	D	PaisagemD	Anti-horário	Rápida	D	PaisagemD

APÊNDICE B – PSEUDOCÓDIGO DO SOFTWARE

Algoritmo 11 – Código completo

```
Var
// Seção de Declarações das variáveis
real porcentagemNormalForte ← 0
real porcentagemNormalFraco ← 0
real porcentagemTodosOsEstados ← 0
real porcentagemTodasAsTransicoes ← 0
inteiro contador_de_repeticoes ← 0

percorreuTodosNormalForte: vetor [0..255] de logico
percorreuTodasRegioesIniciais: vetor [0..3] de logico
percorreuTodasOrientacoesIniciais: vetor [0..3] de logico
percorreuTodosSentidos: vetor [0..1] de logico
percorreuTodasVelocidades: vetor [0..1] de logico
percorreuTodasRegioesFinais: vetor [0..3] de logico
percorreuTodosEstados: vetor [0..5] de logico
percorreuTodasTransicoes: vetor [0..49] de logico

percorreuTodosNormalForte ← [falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso]
percorreuTodasRegioesIniciais ← [falso, falso, falso, falso]
percorreuTodasOrientacoesIniciais ← [falso, falso, falso, falso]
percorreuTodosSentidos ← [falso, falso]
percorreuTodasVelocidades ← [falso, falso]
percorreuTodasRegioesFinais ← [falso, falso, falso, falso]
percorreuTodosEstados ← [falso, falso, falso, falso, falso, falso]
percorreuTodasTransicoes ← [falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso, falso,
falso, falso, falso, falso, falso, falso, falso, falso, falso]

porcentagensNormalForteOrdenadas: vetor [0..999] de real
porcentagensNormalFracoOrdenadas: vetor [0..999] de real
porcentagensTodosOsEstadosOrdenadas: vetor [0..999] de real
porcentagensTodasAsTransicoesOrdenadas: vetor [0..999] de real

caracter todosNormalForte ← ["A-Retrato-Devagar-Horário-A-Retrato", "A-Retrato-Devagar-Horário-B-PaisagemE", "A-Retrato-Devagar-
Horário-C-PaisagemE", "A-Retrato-Devagar-Horário-D-PaisagemD", "A-Retrato-Rápida-Horário-A-Retrato", "A-Retrato-Rápida-Horário-
B-PaisagemE", "A-Retrato-Rápida-Horário-C-RetratoInv", "A-Retrato-Rápida-Horário-D-PaisagemD", "A-Retrato-Devagar-Anti-horário-
A-Retrato", "A-Retrato-Devagar-Anti-horário-B-PaisagemE", "A-Retrato-Devagar-Anti-horário-C-PaisagemD", "A-Retrato-Devagar-Anti-
horário-D-PaisagemD", "A-Retrato-Rápida-Anti-horário-A-Retrato", "A-Retrato-Rápida-Anti-horário-B-PaisagemE", "A-Retrato-Rápida-
Anti-horário-C-RetratoInv", "A-Retrato-Rápida-Anti-horário-D-PaisagemD", "A-RetratoInv-Devagar-Horário-A--", "A-RetratoInv-
Devagar-Horário-B--", "A-RetratoInv-Devagar-Horário-C--", "A-RetratoInv-Devagar-Horário-D--", "A-RetratoInv-Rápida-Horário-A--",
"A-RetratoInv-Rápida-Horário-B--", "A-RetratoInv-Rápida-Horário-C--", "A-RetratoInv-Rápida-Horário-D--",              "A-RetratoInv-
Devagar-Anti-horário-A--", "A-RetratoInv-Devagar-Anti-horário-B--", "A-RetratoInv-Devagar-Anti-horário-C--", "A-RetratoInv-Devagar-
Anti-horário-D--", "A-RetratoInv-Rápida-Anti-horário-A--", "A-RetratoInv-Rápida-Anti-horário-B--", "A-RetratoInv-Rápida-Anti-horário-
C--", "A-RetratoInv-Rápida-Anti-horário-D--", "A-PaisagemE-Devagar-Horário-A--", "A-PaisagemE-Devagar-Horário-B--", "A-
PaisagemE-Devagar-Horário-C--", "A-PaisagemE-Devagar-Horário-D--", "A-PaisagemE-Rápida-Horário-A--", "A-PaisagemE-Rápida-
Horário-B--", "A-PaisagemE-Rápida-Horário-C--", "A-PaisagemE-Rápida-Horário-D--", "A-PaisagemE-Devagar-Anti-horário-A--", "A-
PaisagemE-Devagar-Anti-horário-B--", "A-PaisagemE-Devagar-Anti-horário-C--", "A-PaisagemE-Devagar-Anti-horário-D--", "A-
PaisagemE-Rápida-Anti-horário-A--", "A-PaisagemE-Rápida-Anti-horário-B--", "A-PaisagemE-Rápida-Anti-horário-C--", "A-PaisagemE-
Rápida-Anti-horário-D--", "A-PaisagemD-Devagar-Horário-A--", "A-PaisagemD-Devagar-Horário-B--", "A-PaisagemD-Devagar-Horário-
C--", "A-PaisagemD-Devagar-Horário-D--", "A-PaisagemD-Rápida-Horário-A--", "A-PaisagemD-Rápida-Horário-B--", "A-PaisagemD-
Rápida-Horário-C--", "A-PaisagemD-Rápida-Horário-D--", "A-PaisagemD-Devagar-Anti-horário-A--", "A-PaisagemD-Devagar-Anti-
horário-B--", "A-PaisagemD-Devagar-Anti-horário-C--", "A-PaisagemD-Devagar-Anti-horário-D--", "A-PaisagemD-Rápida-Anti-horário-
A--", "A-PaisagemD-Rápida-Anti-horário-B--", "A-PaisagemD-Rápida-Anti-horário-C--", "A-PaisagemD-Rápida-Anti-horário-D--", "B-
Retrato-Devagar-Horário-A--", "B-Retrato-Devagar-Horário-B--", "B-Retrato-Devagar-Horário-C--", "B-Retrato-Devagar-Horário-D--",
"B-Retrato-Rápida-Horário-A--", "B-Retrato-Rápida-Horário-B--", "B-Retrato-Rápida-Horário-C--", "B-Retrato-Rápida-Horário-D--", "B-
Retrato-Devagar-Anti-horário-A--", "B-Retrato-Devagar-Anti-horário-B--", "B-Retrato-Devagar-Anti-horário-C--", "B-Retrato-Devagar-
Anti-horário-D--", "B-Retrato-Rápida-Anti-horário-A--", "B-Retrato-Rápida-Anti-horário-B--", "B-Retrato-Rápida-Anti-horário-C--", "B-
Retrato-Rápida-Anti-horário-D--", "B-RetratoInv-Devagar-Horário-A--", "B-RetratoInv-Devagar-Horário-B--", "B-RetratoInv-Devagar-
Horário-C--", "B-RetratoInv-Devagar-Horário-D--", "B-RetratoInv-Rápida-Horário-A--", "B-RetratoInv-Rápida-Horário-B--", "B-
RetratoInv-Rápida-Horário-C--", "B-RetratoInv-Rápida-Horário-D--", "B-RetratoInv-Devagar-Anti-horário-A--", "B-RetratoInv-Devagar-
Anti-horário-B--", "B-RetratoInv-Devagar-Anti-horário-C--", "B-RetratoInv-Devagar-Anti-horário-D--", "B-RetratoInv-Rápida-Anti-
horário-A--", "B-RetratoInv-Rápida-Anti-horário-B--", "B-RetratoInv-Rápida-Anti-horário-C--", "B-RetratoInv-Rápida-Anti-horário-D--",
```

"B-PaisagemE-Devagar-Horário-A-Retrato", "B-PaisagemE-Devagar-Horário-B-PaisagemE", "B-PaisagemE-Devagar-Horário-C-PaisagemE", "B-PaisagemE-Devagar-Horário-D-PaisagemD", "B-PaisagemE-Rápida-Horário-A-Retrato", "B-PaisagemE-Rápida-Horário-B-PaisagemE", "B-PaisagemE-Rápida-Horário-C-PaisagemE", "B-PaisagemE-Rápida-Horário-D-PaisagemD", "B-PaisagemE-Devagar-Anti-horário-A-Retrato", "B-PaisagemE-Devagar-Anti-horário-B-PaisagemE", "B-PaisagemE-Devagar-Anti-horário-C-PaisagemD", "B-PaisagemE-Devagar-Anti-horário-D-PaisagemD", "B-PaisagemE-Rápida-Anti-horário-A-Retrato", "B-PaisagemE-Rápida-Anti-horário-B-PaisagemE", "B-PaisagemE-Rápida-Anti-horário-C-PaisagemE", "B-PaisagemE-Rápida-Anti-horário-D-PaisagemD", "B-PaisagemD-Devagar-Horário-A--", "B-PaisagemD-Devagar-Horário-B--", "B-PaisagemD-Devagar-Horário-C--", "B-PaisagemD-Devagar-Horário-D--", "B-PaisagemD-Rápida-Horário-A--", "B-PaisagemD-Rápida-Horário-B--", "B-PaisagemD-Rápida-Horário-C--", "B-PaisagemD-Rápida-Horário-D--", "B-PaisagemD-Devagar-Anti-horário-A--", "B-PaisagemD-Devagar-Anti-horário-B--", "B-PaisagemD-Devagar-Anti-horário-C--", "B-PaisagemD-Devagar-Anti-horário-D--", "B-PaisagemD-Rápida-Anti-horário-A--", "B-PaisagemD-Rápida-Anti-horário-B--", "B-PaisagemD-Rápida-Anti-horário-C--", "B-PaisagemD-Rápida-Anti-horário-D--", "C-Retrato-Devagar-Horário-A--", "C-Retrato-Devagar-Horário-B--", "C-Retrato-Devagar-Horário-C--", "C-Retrato-Devagar-Horário-D--", "C-Retrato-Rápida-Horário-A--", "C-Retrato-Rápida-Horário-B--", "C-Retrato-Rápida-Horário-C--", "C-Retrato-Rápida-Horário-D--", "C-Retrato-Devagar-Anti-horário-A--", "C-Retrato-Devagar-Anti-horário-B--", "C-Retrato-Devagar-Anti-horário-C--", "C-Retrato-Devagar-Anti-horário-D--", "C-Retrato-Rápida-Anti-horário-A--", "C-Retrato-Rápida-Anti-horário-B--", "C-Retrato-Rápida-Anti-horário-C--", "C-Retrato-Rápida-Anti-horário-D--", "C-RetratoInv-Devagar-Horário-A-Retrato", "C-RetratoInv-Devagar-Horário-B-PaisagemE", "C-RetratoInv-Devagar-Horário-C-PaisagemE", "C-RetratoInv-Devagar-Horário-D-PaisagemD", "C-RetratoInv-Rápida-Horário-A-Retrato", "C-RetratoInv-Rápida-Horário-B-PaisagemE", "C-RetratoInv-Rápida-Horário-C-RetratoInv", "C-RetratoInv-Rápida-Horário-D-PaisagemD", "C-RetratoInv-Devagar-Anti-horário-A-Retrato", "C-RetratoInv-Devagar-Anti-horário-B-PaisagemE", "C-RetratoInv-Devagar-Anti-horário-C-PaisagemD", "C-RetratoInv-Devagar-Anti-horário-D-PaisagemD", "C-RetratoInv-Rápida-Anti-horário-A-Retrato", "C-RetratoInv-Rápida-Anti-horário-B-PaisagemE", "C-RetratoInv-Rápida-Anti-horário-C-RetratoInv", "C-RetratoInv-Rápida-Anti-horário-D-PaisagemD", "C-PaisagemE-Devagar-Horário-A--", "C-PaisagemE-Devagar-Horário-B--", "C-PaisagemE-Devagar-Horário-C--", "C-PaisagemE-Devagar-Horário-D--", "C-PaisagemE-Rápida-Horário-A--", "C-PaisagemE-Rápida-Horário-B--", "C-PaisagemE-Rápida-Horário-C--", "C-PaisagemE-Rápida-Horário-D--", "C-PaisagemE-Devagar-Anti-horário-A--", "C-PaisagemE-Devagar-Anti-horário-B--", "C-PaisagemE-Devagar-Anti-horário-C--", "C-PaisagemE-Devagar-Anti-horário-D--", "C-PaisagemE-Rápida-Anti-horário-A--", "C-PaisagemE-Rápida-Anti-horário-B--", "C-PaisagemE-Rápida-Anti-horário-C--", "C-PaisagemE-Rápida-Anti-horário-D--", "C-PaisagemD-Devagar-Horário-A--", "C-PaisagemD-Devagar-Horário-B--", "C-PaisagemD-Devagar-Horário-C--", "C-PaisagemD-Devagar-Horário-D--", "C-PaisagemD-Rápida-Horário-A--", "C-PaisagemD-Rápida-Horário-B--", "C-PaisagemD-Rápida-Horário-C--", "C-PaisagemD-Rápida-Horário-D--", "C-PaisagemD-Devagar-Anti-horário-A--", "C-PaisagemD-Devagar-Anti-horário-B--", "C-PaisagemD-Devagar-Anti-horário-C--", "C-PaisagemD-Devagar-Anti-horário-D--", "C-PaisagemD-Rápida-Anti-horário-A--", "C-PaisagemD-Rápida-Anti-horário-B--", "C-PaisagemD-Rápida-Anti-horário-C--", "C-PaisagemD-Rápida-Anti-horário-D--", "D-Retrato-Devagar-Horário-A--", "D-Retrato-Devagar-Horário-B--", "D-Retrato-Devagar-Horário-C--", "D-Retrato-Devagar-Horário-D--", "D-Retrato-Rápida-Horário-A--", "D-Retrato-Rápida-Horário-B--", "D-Retrato-Rápida-Horário-C--", "D-Retrato-Rápida-Horário-D--", "D-Retrato-Devagar-Anti-horário-A--", "D-Retrato-Devagar-Anti-horário-B--", "D-Retrato-Devagar-Anti-horário-C--", "D-Retrato-Devagar-Anti-horário-D--", "D-Retrato-Rápida-Anti-horário-A--", "D-Retrato-Rápida-Anti-horário-B--", "D-Retrato-Rápida-Anti-horário-C--", "D-Retrato-Rápida-Anti-horário-D--", "D-RetratoInv-Devagar-Horário-A--", "D-RetratoInv-Devagar-Horário-B--", "D-RetratoInv-Devagar-Horário-C--", "D-RetratoInv-Devagar-Horário-D--", "D-RetratoInv-Rápida-Horário-A--", "D-RetratoInv-Rápida-Horário-B--", "D-RetratoInv-Rápida-Horário-C--", "D-RetratoInv-Rápida-Horário-D--", "D-RetratoInv-Devagar-Anti-horário-A--", "D-RetratoInv-Devagar-Anti-horário-B--", "D-RetratoInv-Devagar-Anti-horário-C--", "D-RetratoInv-Devagar-Anti-horário-D--", "D-RetratoInv-Rápida-Anti-horário-A--", "D-RetratoInv-Rápida-Anti-horário-B--", "D-RetratoInv-Rápida-Anti-horário-C--", "D-RetratoInv-Rápida-Anti-horário-D--", "D-PaisagemE-Devagar-Horário-A--", "D-PaisagemE-Devagar-Horário-B--", "D-PaisagemE-Devagar-Horário-C--", "D-PaisagemE-Devagar-Horário-D--", "D-PaisagemE-Rápida-Horário-A--", "D-PaisagemE-Rápida-Horário-B--", "D-PaisagemE-Rápida-Horário-C--", "D-PaisagemE-Rápida-Horário-D--", "D-PaisagemE-Devagar-Anti-horário-A--", "D-PaisagemE-Devagar-Anti-horário-B--", "D-PaisagemE-Devagar-Anti-horário-C--", "D-PaisagemE-Devagar-Anti-horário-D--", "D-PaisagemE-Rápida-Anti-horário-A--", "D-PaisagemE-Rápida-Anti-horário-B--", "D-PaisagemE-Rápida-Anti-horário-C--", "D-PaisagemE-Rápida-Anti-horário-D--", "D-PaisagemD-Devagar-Horário-A-Retrato", "D-PaisagemD-Devagar-Horário-B-PaisagemE", "D-PaisagemD-Devagar-Horário-C-PaisagemE", "D-PaisagemD-Devagar-Horário-D-PaisagemD", "D-PaisagemD-Rápida-Horário-A-Retrato", "D-PaisagemD-Rápida-Horário-B-PaisagemE", "D-PaisagemD-Rápida-Horário-C-PaisagemD", "D-PaisagemD-Rápida-Horário-D-PaisagemD", "D-PaisagemD-Devagar-Anti-horário-A-Retrato", "D-PaisagemD-Devagar-Anti-horário-B-PaisagemE", "D-PaisagemD-Devagar-Anti-horário-C-PaisagemD", "D-PaisagemD-Devagar-Anti-horário-D-PaisagemD", "D-PaisagemD-Rápida-Anti-horário-A-Retrato", "D-PaisagemD-Rápida-Anti-horário-B-PaisagemE", "D-PaisagemD-Rápida-Anti-horário-C-PaisagemD", "D-PaisagemD-Rápida-Anti-horário-D-PaisagemD"]
caracter estadosInicias ← ["A-Retrato", "B-PaisagemE", "C-RetratoInv", "D-PaisagemD"]
caracter estadosFinais ← ["A-Retrato", "B-PaisagemE", "C-PaisagemE", "C-RetratoInv", "C-PaisagemD", "D-PaisagemD"]
caracter todosEstados ← ["A-Retrato", "B-PaisagemE", "C-PaisagemE", "C-RetratoInv", "C-PaisagemD", "D-PaisagemD"]
caracter todasTransicoes ← ["A-Retrato-Devagar-Horário-A-Retrato", "A-Retrato-Devagar-Anti-horário-A-Retrato", "A-Retrato-Rápida-Horário-A-Retrato", "A-Retrato-Rápida-Anti-horário-A-Retrato", "A-Retrato-Devagar-Horário-B-PaisagemE", "A-Retrato-Rápida-Horário-B-PaisagemE", "A-Retrato-Rápida-Anti-horário-B-PaisagemE", "A-Retrato-Rápida-Horário-C-RetratoInv", "A-Retrato-Rápida-Anti-horário-C-RetratoInv", "A-Retrato-Devagar-Anti-horário-D-PaisagemD", "A-Retrato-Rápida-Anti-horário-D-PaisagemD", "A-Retrato-Rápida-Horário-D-PaisagemD", "B-PaisagemE-Devagar-Horário-B-PaisagemE", "B-PaisagemE-Devagar-Anti-horário-B-PaisagemE", "B-PaisagemE-Rápida-Horário-B-PaisagemE", "B-PaisagemE-Rápida-Anti-horário-B-PaisagemE", "B-PaisagemE-Devagar-Anti-horário-A-Retrato", "B-PaisagemE-Rápida-Anti-horário-A-Retrato", "B-PaisagemE-Rápida-Horário-A-Retrato", "B-PaisagemE-Devagar-Horário-C-PaisagemE", "B-PaisagemE-Rápida-Horário-C-PaisagemE", "B-PaisagemE-Rápida-Anti-horário-C-PaisagemE", "B-PaisagemE-Rápida-Horário-D-PaisagemD", "B-PaisagemE-Rápida-Anti-horário-D-PaisagemD", "C-PaisagemE-Devagar-Horário-D-PaisagemD", "C-RetratoInv-Devagar-Horário-C-RetratoInv", "C-RetratoInv-Devagar-Anti-horário-C-RetratoInv", "C-RetratoInv-Rápida-Horário-C-RetratoInv", "C-RetratoInv-Rápida-Anti-horário-C-RetratoInv", "C-RetratoInv-Rápida-Horário-A-Retrato", "C-RetratoInv-Rápida-Anti-horário-A-Retrato", "C-RetratoInv-Devagar-Anti-horário-B-PaisagemE", "C-RetratoInv-Rápida-Anti-horário-B-PaisagemE", "C-RetratoInv-Rápida-Horário-B-PaisagemE", "C-RetratoInv-Devagar-Horário-D-PaisagemD", "C-RetratoInv-Rápida-Horário-D-PaisagemD", "C-RetratoInv-Rápida-Anti-horário-D-PaisagemD", "C-PaisagemD-Devagar-Anti-horário-B-PaisagemE", "D-PaisagemD-Devagar-Horário-D-PaisagemD", "D-PaisagemD-Devagar-Anti-horário-D-PaisagemD", "D-PaisagemD-Rápida-Horário-D-PaisagemD", "D-PaisagemD-Rápida-Anti-horário-D-PaisagemD", "D-PaisagemD-Devagar-Horário-A-Retrato", "D-PaisagemD-Rápida-Horário-A-Retrato", "D-PaisagemD-Rápida-Anti-horário-A-Retrato", "D-PaisagemD-Rápida-Horário-B-PaisagemE", "D-PaisagemD-Devagar-Anti-horário-C-PaisagemD", "D-PaisagemD-Rápida-Anti-horário-C-PaisagemD", "D-PaisagemD-Rápida-Horário-C-PaisagemD"]

caracter regioesIniciais ← ["A", "B", "C", "D"]
caracter orientacoesIniciais ← ["Retrato", "PaisagemE", "RetratoInv", "PaisagemD"]
caracter sentidosDeRotacao ← ["Horário", "Anti-horário"]
caracter velocidades ← ["Devagar", "Rápida"]
caracter regioesFinais ← ["A", "B", "C", "D"]

```
caracter estadoFinal ← [verdadeiro, falso]

logico ehEstadoFinal ← falso
logico ehEstadoInicial ← verdadeiro

real tempoInicio ← 0
real tempoTotal ← 0

caracter estadoAtual ← ""
caracter proximoEstado ← ""
caracter sentidoDeRotacao ← ""
caracter velocidade ← ""
caracter transicao ← ""
real tempoSolicitado ← args[0]*1 //milissegundos

Inicio

Enquanto (contador_de_repeticoes < 1000) faca
  tempoInicio ← horarioAtualdoSistema()
  tempoTotal ← horarioAtualdoSistema()-tempoInicio
  Enquanto (tempoTotal<=tempoSolicitado) faca
    Se (ehEstadoInicial=verdadeiro) entao
      estadoAtual ← selecionaEstado(estadosInicias)
    senao
      estadoAtual ← proximoEstado
    fimSe
    escolha (estadoAtual)
      caso "A-Retrato"
        passouPorEstado(estadoAtual)
        Se (ehEstadoInicial=verdadeiro) entao
          //recebeSubstring(str, inicio, n) - Recebe como resultado uma Substring de str, começando da posição inicio com quantidade de
caracteres igual n, no caso abaixo: a substring "A"
          passouPorRegiaoInicial(recebeSubstring(estadoAtual, 0,1))
          //retiraSubstring(str, n) - Recebe como resultado uma Substring de str, retirando os n caracteres, no caso abaixo: a substring
"Retrato"
          passouPorOrientacaoInicial(substring(estadoAtual, 2))
          ehEstadoInicial ← falso
        fimSe
        sentidoDeRotacao ← selecionaSentidoDeRotacao(sentidosDeRotacao)
        passouPorSentido(sentidoDeRotacao)
        velocidade ← selecionaVelocidade(velocidades)
        passouPorVelocidade(velocidade)
        Se (sentidoDeRotacao="Horário") entao //Horário
          Se (velocidade="Devagar") entao //Horário-Devagar
            proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE","C-PaisagemE", "D-PaisagemD"])
            escolha (proximoEstado)
              caso "A-Retrato"
                passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                passouPorEstado("B-PaisagemE")
                passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
                passouPorEstado("C-PaisagemE")
                passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
              caso "C-PaisagemE"
                passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                passouPorEstado("B-PaisagemE")
                passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
              caso "D-PaisagemD"
                passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                passouPorEstado("B-PaisagemE")
                passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
                passouPorEstado("C-PaisagemE")
                passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
            fimEscolha
          senao //Horário-Rápida
            proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-RetratoInv", "D-PaisagemD"])
          fimSe
        senao //Anti-horário
          Se(velocidade="Devagar") //Anti-horário-Devagar
            proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemD", "D-PaisagemD"])
            escolha (proximoEstado)
              caso "A-Retrato"
                passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
                passouPorEstado("C-PaisagemD")
                passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
                passouPorEstado("B-PaisagemE")
```

```
                passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
              caso "B-PaisagemE"
                passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
                passouPorEstado("C-PaisagemD")
                passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
              caso "C-PaisagemD"
                passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
            fimEscolha
          senao //Anti-horário-Rápida
            proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-RetratoInv", "D-PaisagemD"])
          fimSe
        fimSe
        passouPorEstado(proximoEstado)
        transicao ← transformaEmTransicao(estadoAtual,velocidade,sentidoDeRotacao,proximoEstado)
        passouPorTransicao(transicao)
        passouNormalForte(transicao)
        ehEstadoFinal←verdadeiro
        Se (ehEstadoFinal=verdadeiro) entao
          passouPorRegiaoFinal(recebeSubstring(estadoAtual, 0,1))
          ehEstadoInicial ← verdadeiro
          porcentagemNormalForte ← coberturaNormalForte()
          porcentagemNormalFraco ← coberturaNormalFraco(percorreuTodasRegioesIniciais, percorreuTodasOrientacoesIniciais,
percorreuTodosSentidos, percorreuTodasVelocidades, percorreuTodasRegioesFinais)
          porcentagemTodosOsEstados ← coberturaTodosOsEstados()
          porcentagemTodasAsTransicoes ← coberturaTodaAsTransicoes()
        fimSe
      caso "B-PaisagemE"
        passouPorEstado(estadoAtual)
        Se (ehEstadoInicial=verdadeiro) entao
          passouPorRegiaoInicial(recebeSubstring(estadoAtual, 0,1))
          passouPorOrientacaoInicial(substring(estadoAtual, 2))
          ehEstadoInicial ← falso
        fimSe
          sentidoDeRotacao ← selecionaSentidoDeRotacao(sentidosDeRotacao)
          passouPorSentido(sentidoDeRotacao)
          velocidade ← selecionaVelocidade(velocidades)
          passouPorVelocidade(velocidade)
          Se (sentidoDeRotacao="Horário") entao //Horário
            Se (velocidade="Devagar") entao //Horário-Devagar
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemE", "D-PaisagemD"])
              escolha (proximoEstado)
                caso "A-Retrato"
                  passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
                  passouPorEstado("C-PaisagemE")
                  passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
                  passouPorEstado("D-PaisagemD")
                  passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
                caso "B-PaisagemE"
                  passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
                  passouPorEstado("C-PaisagemE")
                  passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
                  passouPorEstado("D-PaisagemD")
                  passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
                  passouPorEstado("A-Retrato")
                  passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                caso "D-PaisagemD"
                  passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
                  passouPorEstado("C-PaisagemE")
                  passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
              fimEscolha
            senao //Horário-Rápida
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemE", "D-PaisagemD"])
            fimSe
          senao //Anti-horário
            Se(velocidade="Devagar") //Anti-horário-Devagar
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemD", "D-PaisagemD"])
              escolha (proximoEstado)
                caso "B-PaisagemE"
                  passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                  passouPorEstado("A-Retrato")
                  passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
                  passouPorEstado("D-PaisagemD")
                  passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
                  passouPorEstado("C-PaisagemD")
                  passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
```

```
                caso "C-PaisagemD"
                  passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                  passouPorEstado("A-Retrato")
                  passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
                  passouPorEstado("D-PaisagemD")
                  passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
                caso "D-PaisagemD"
                  passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                  passouPorEstado("A-Retrato")
                  passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
              fimEscolha
            senao //Anti-horário-Rápida
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemE", "D-PaisagemD"])
            fimSe
          fimSe
          passouPorEstado(proximoEstado)
          transicao ← transformaEmTransicao(estadoAtual,velocidade,sentidoDeRotacao,proximoEstado)
          passouPorTransicao(transicao)
          passouNormalForte(transicao)
          ehEstadoFinal←verdadeiro
          Se (ehEstadoFinal=verdadeiro) entao
            passouPorRegiaoFinal(recebeSubstring(estadoAtual, 0,1))
            ehEstadoInicial ← verdadeiro
            porcentagemNormalForte ← coberturaNormalForte()
            porcentagemNormalFraco ← coberturaNormalFraco(percorreuTodasRegioesIniciais, percorreuTodasOrientacoesIniciais,
percorreuTodosSentidos, percorreuTodasVelocidades, percorreuTodasRegioesFinais)
            porcentagemTodosOsEstados ← coberturaTodosOsEstados()
            porcentagemTodasAsTransicoes ← coberturaTodaAsTransicoes()
          fimSe
        caso "C-RetratoInv"
          passouPorEstado(estadoAtual)
          Se (ehEstadoInicial=verdadeiro) entao
            passouPorRegiaoInicial(recebeSubstring(estadoAtual, 0,1))
      passouPorOrientacaoInicial(substring(estadoAtual, 2))
            ehEstadoInicial ← falso
          fimSe
          sentidoDeRotacao ← selecionaSentidoDeRotacao(sentidosDeRotacao)
          passouPorSentido(sentidoDeRotacao)
          velocidade ← selecionaVelocidade(velocidades)
          passouPorVelocidade(velocidade)
          Se (sentidoDeRotacao="Horário") entao //Horário
            Se (velocidade="Devagar") entao //Horário-Devagar
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemE", "C-RetratoInv", "D-PaisagemD"])
              escolha (proximoEstado)
              caso "A-Retrato"
                passouPorTransicao("C-RetratoInv-Devagar-Horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
              caso "B-PaisagemE"
                passouPorTransicao("C-RetratoInv-Devagar-Horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
                passouPorEstado("A-Retrato")
                passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
              caso "C-PaisagemE"
                passouPorTransicao("C-RetratoInv-Devagar-Horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
                passouPorEstado("A-Retrato")
                passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                passouPorEstado("B-PaisagemE")
                passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
              fimEscolha
            senao //Horário-Rápida
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-RetratoInv", "D-PaisagemD"])
            fimSe
          senao //Anti-horário
            Se(velocidade="Devagar") //Anti-horário-Devagar
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-RetratoInv", "C-PaisagemD", "D-PaisagemD"])
              escolha (proximoEstado)
                caso "A-Retrato"
                  passouPorTransicao("C-RetratoInv-Devagar-Anti-horário-B-PaisagemE")
                  passouPorEstado("B-PaisagemE")
                  passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                caso "D-PaisagemD"
                  passouPorTransicao("C-RetratoInv-Devagar-Anti-horário-B-PaisagemE")
                  passouPorEstado("B-PaisagemE")
                  passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                  passouPorEstado("A-Retrato")
```

```
                passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
              caso "C-PaisagemD"
                passouPorTransicao("C-RetratoInv-Devagar-Anti-horário-B-PaisagemE")
                passouPorEstado("B-PaisagemE")
                passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                passouPorEstado("A-Retrato")
                passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
                passouPorEstado("D-PaisagemD")
                passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
              fimEscolha
            senao //Anti-horário-Rápida
              proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-RetratoInv", "D-PaisagemD"])
            fimSe
          fimSe
          passouPorEstado(proximoEstado)
          transicao ← transformaEmTransicao(estadoAtual,velocidade,sentidoDeRotacao,proximoEstado)
          passouPorTransicao(transicao)
          passouNormalForte(transicao)
          ehEstadoFinal←verdadeiro
          Se (ehEstadoFinal=verdadeiro) entao
            passouPorRegiaoFinal(recebeSubstring(estadoAtual, 0,1))
            ehEstadoInicial ← verdadeiro
            porcentagemNormalForte ← coberturaNormalForte()
            porcentagemNormalFraco ← coberturaNormalFraco(percorreuTodasRegioesIniciais, percorreuTodasOrientacoesIniciais,
percorreuTodosSentidos, percorreuTodasVelocidades, percorreuTodasRegioesFinais)
            porcentagemTodosOsEstados ← coberturaTodosOsEstados()
            porcentagemTodasAsTransicoes ← coberturaTodaAsTransicoes()
          fimSe
        caso "D-PaisagemD"
          passouPorEstado(estadoAtual)
          Se (ehEstadoInicial=verdadeiro) entao
            passouPorRegiaoInicial(recebeSubstring(estadoAtual, 0,1))
            passouPorOrientacaoInicial(substring(estadoAtual, 2))
            ehEstadoInicial ← falso
          fimSe
          sentidoDeRotacao ← selecionaSentidoDeRotacao(sentidosDeRotacao)
          passouPorSentido(sentidoDeRotacao)
          velocidade ← selecionaVelocidade(velocidades)
          passouPorVelocidade(velocidade)
            Se (sentidoDeRotacao="Horário") entao //Horário
              Se (velocidade="Devagar") entao //Horário-Devagar
                proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemE", "D-PaisagemD"])
                escolha (proximoEstado)
                  caso "B-PaisagemE"
                    passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
                    passouPorEstado("A-Retrato")
                    passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                  caso "C-PaisagemE"
                    passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
                    passouPorEstado("A-Retrato")
                    passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                    passouPorEstado("B-PaisagemE")
                    passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
                  caso "D-PaisagemD"
                    passouPorTransicao("D-PaisagemD-Devagar-Horário-A-Retrato")
                    passouPorEstado("A-Retrato")
                    passouPorTransicao("A-Retrato-Devagar-Horário-B-PaisagemE")
                    passouPorEstado("B-PaisagemE")
                    passouPorTransicao("B-PaisagemE-Devagar-Horário-C-PaisagemE")
                    passouPorEstado("C-PaisagemE")
                    passouPorTransicao("C-PaisagemE-Devagar-Horário-D-PaisagemD")
                  fimEscolha
               senao //Horário-Rápida
                  proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemD", "D-PaisagemD"])
                fimSe
              senao //Anti-horário
                Se(velocidade="Devagar") //Anti-horário-Devagar
                  proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemD", "D-PaisagemD"])
                  escolha (proximoEstado)
                    caso "A-Retrato"
                      passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
                      passouPorEstado("C-PaisagemD")
                      passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
                      passouPorEstado("B-PaisagemE")
                      passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                    caso "B-PaisagemE"
                      passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
                      passouPorEstado("C-PaisagemD")
                      passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
```

```
                    caso "D-PaisagemD"
                      passouPorTransicao("D-PaisagemD-Devagar-Anti-horário-C-PaisagemD")
                      passouPorEstado("C-PaisagemD")
                      passouPorTransicao("C-PaisagemD-Devagar-Anti-horário-B-PaisagemE")
                      passouPorEstado("B-PaisagemE")
                      passouPorTransicao("B-PaisagemE-Devagar-Anti-horário-A-Retrato")
                      passouPorEstado("A-Retrato")
                      passouPorTransicao("A-Retrato-Devagar-Anti-horário-D-PaisagemD")
                  fimEscolha
                senao //Anti-horário-Rápida
                  proximoEstado ← selecionaEstado(["A-Retrato", "B-PaisagemE", "C-PaisagemD", "D-PaisagemD"])
                fimSe
              fimSe
              passouPorEstado(proximoEstado)
              transicao ← transformaEmTransicao(estadoAtual,velocidade,sentidoDeRotacao,proximoEstado)
              passouPorTransicao(transicao)
              passouNormalForte(transicao)
              ehEstadoFinal←verdadeiro
              Se (ehEstadoFinal=verdadeiro) entao
                passouPorRegiaoFinal(recebeSubstring(estadoAtual, 0,1))
                ehEstadoInicial ← verdadeiro
                porcentagemNormalForte ← coberturaNormalForte()
                porcentagemNormalFraco ← coberturaNormalFraco(percorreuTodasRegioesIniciais, percorreuTodasOrientacoesIniciais,
percorreuTodosSentidos, percorreuTodasVelocidades, percorreuTodasRegioesFinais)
                porcentagemTodosOsEstados ← coberturaTodosOsEstados()
                porcentagemTodasAsTransicoes ← coberturaTodaAsTransicoes()
              fimSe
      fimEscolha
      tempoTotal ← horarioAtualdoSistema()-tempoInicio //Atualiza o tempoTotal
    fimEnquanto
    porcentagemNormalForte ← coberturaNormalForte()
    porcentagemNormalFraco ← coberturaNormalFraco(percorreuTodasRegioesIniciais,
percorreuTodasOrientacoesIniciais, percorreuTodosSentidos, percorreuTodasVelocidades, percorreuTodasRegioesFinais)
    porcentagemTodosOsEstados ← coberturaTodosOsEstados()
    porcentagemTodasAsTransicoes ← coberturaTodaAsTransicoes()

    porcentagensNormalForteOrdenadas[contador_de_repeticoes] ← porcentagemNormalForte
    porcentagensNormalFracoOrdenadas[contador_de_repeticoes] ← porcentagemNormalFraco
    porcentagensTodosOsEstadosOrdenadas[contador_de_repeticoes] ← porcentagemTodosOsEstados
    porcentagensTodasAsTransicoesOrdenadas[contador_de_repeticoes] ← porcentagemTodasAsTransicoes

    //Soma +1 ao contador_de_repeticoes para ir para próxima repetição
    contador_de_repeticoes ← contador_de_repeticoes + 1

    //Zera todas as porcentagens para recomeçar na próxima repetição
    porcentagemNormalForte ← 0
    porcentagemNormalFraco ← 0
    porcentagemTodosOsEstados ← 0
    porcentagemTodasAsTransicoes ← 0

    //Zera todas os vetores lógicos que indicam por onde passou, para recomeçar na próxima repetição
    Para inteiro i ← 0 ate i < tamanho(percorreuTodosEstados) passo +1 faca
      percorreuTodosEstados[i] ← falso
    fimPara

    Para inteiro i ← 0 ate i < tamanho(percorreuTodasRegioesIniciais) passo +1 faca
      percorreuTodasRegioesIniciais[i] ← falso
    fimPara

    Para inteiro i ← 0 ate i < tamanho(percorreuTodasOrientacoesIniciais) passo +1 faca
      percorreuTodasOrientacoesIniciais[i] ← falso
    fimPara

    Para inteiro i ← 0 ate i < tamanho(percorreuTodosSentidos) passo +1 faca
      percorreuTodosSentidos[i] ← falso
    fimPara

    Para inteiro i ← 0 ate i < tamanho(percorreuTodasVelocidades) passo +1 faca
      percorreuTodasVelocidades[i] ← falso
    fimPara

    Para inteiro i ← 0 ate i < tamanho(percorreuTodasRegioesFinais) passo +1 faca
      percorreuTodasRegioesFinais[i] ← falso
    fimPara

    Para inteiro i ← 0 ate i < tamanho(percorreuTodasTransicoes) passo +1 faca
      percorreuTodasTransicoes[i] ← falso
    fimPara
```

```
  Para inteiro i ← 0 ate i < tamanho(percorreuTodosNormalForte) passo +1 faca
    percorreuTodosNormalForte[i] ← falso
  fimPara
fimEnquanto

//Ordena o vetor, para calcular os dados necessários para plotagem do
//gráfico boxplot do critério Normal Forte
ordena(porcentagensNormalForteOrdenadas)
real media←0
real mediana←0
real q3←0
real q1←0
real maximo←0
real minimo←0
Para inteiro i ← 0 ate i < tamanho(porcentagensNormalForteOrdenadas) passo +1 faca
  media ← media + porcentagensNormalForteOrdenadas[i]
fimPara
media ← media/1000
q3 ← porcentagensNormalForteOrdenadas[((1000/4)*3)]
mediana ← (porcentagensNormalForteOrdenadas[((1000/2)-1)] + porcentagensNormalForteOrdenadas[(1000/2)])/2
q1 ← porcentagensNormalForteOrdenadas[((1000/4)-1)]
maximo←porcentagensNormalForteOrdenadas[999]
minimo←porcentagensNormalForteOrdenadas[0]
escreval("Normal Forte: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)

ordena(porcentagensNormalFracoOrdenadas)
media←0
mediana←0
q3←0
q1←0
maximo←0
minimo←0
Para inteiro i ← 0 ate i < tamanho(porcentagensNormalFracoOrdenadas) passo +1 faca
  media ← media + porcentagensNormalFracoOrdenadas[i]
fimPara
media ← media/1000
q3 ← porcentagensNormalFracoOrdenadas[((1000/4)*3)]
mediana ← (porcentagensNormalFracoOrdenadas[((1000/2)-1)] + porcentagensNormalFracoOrdenadas[(1000/2)])/2
q1 ← porcentagensNormalFracoOrdenadas[((1000/4)-1)]
maximo←porcentagensNormalFracoOrdenadas[999]
minimo←porcentagensNormalFracoOrdenadas[0]
escreval("Normal Fraco: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)

ordena(porcentagensTodosOsEstadosOrdenadas)
media←0
mediana←0
q3←0
q1←0
maximo←0
minimo←0
Para inteiro i ← 0 ate i < tamanho(porcentagensTodosOsEstadosOrdenadas) passo +1 faca
  media ← media + porcentagensTodosOsEstadosOrdenadas[i]
fimPara
media ← media/1000
q3 ← porcentagensTodosOsEstadosOrdenadas[((1000/4)*3)]
mediana ← (porcentagensTodosOsEstadosOrdenadas[((1000/2)-1)] + porcentagensTodosOsEstadosOrdenadas[(1000/2)])/2
q1 ← porcentagensTodosOsEstadosOrdenadas[((1000/4)-1)]
maximo←porcentagensTodosOsEstadosOrdenadas[999]
minimo←porcentagensTodosOsEstadosOrdenadas[0]
escreval("Todos os Estados: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)

ordena(porcentagensTodasAsTransicoesOrdenadas)
media←0
mediana←0
q3←0
q1←0
maximo←0
minimo←0
Para inteiro i ← 0 ate i < tamanho(porcentagensTodasAsTransicoesOrdenadas) passo +1 faca
  media ← media + porcentagensTodasAsTransicoesOrdenadas[i]
fimPara
media ← media/1000
q3 ← porcentagensTodasAsTransicoesOrdenadas[((1000/4)*3)]
mediana ← (porcentagensTodasAsTransicoesOrdenadas[((1000/2)-1)] + porcentagensTodasAsTransicoesOrdenadas[(1000/2)])/2
q1 ← porcentagensTodasAsTransicoesOrdenadas[((1000/4)-1)]
maximo←porcentagensTodasAsTransicoesOrdenadas[999]
minimo←porcentagensTodasAsTransicoesOrdenadas[0]
escreval("Toda as Transições: "tempoSolicitado + ";" + maximo + ";" + q3 + ";" + media + ";" + mediana + ";" + q1 + ";" + minimo)
```

```
funcao selecionaEstado (var estados:caracter[]):caracter
Inicio
  retorne estados[AleatorioProximoInt(tamanho(estados))]
fimFuncao

funcao selecionaSentidoDeRotacao (var sentidos:caracter[]):caracter
Inicio
  retorne sentidos[AleatorioProximoInt(tamanho(sentidos))]
fimFuncao

funcao selecionaVelocidade (var velocidades:caracter[]):caracter
Inicio
  retorne velocidades[AleatorioProximoInt(tamanho(velocidades))]
fimFuncao

funcao selecionaSeEstadoFinal(var estadoFinal:logico[]):logico
Inicio
  retorne estadoFinal[AleatorioProximoInt(tamanho(estadoFinal))]
fimFuncao

funcao passouNormalForte(teste:caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodosNormalForte) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(todosNormalForte) passo +1 faca
      Se (todosNormalForte[j]=teste) entao
        percorreuTodosNormalForte[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodosNormalForte
fimFuncao

funcao passouPorEstado(estado:caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodosEstados) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(todosEstados) passo +1 faca
      Se (todosEstados[j]=estado) entao
        percorreuTodosEstados[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodosEstados
fimFuncao

funcao passouPorRegiaoInicial(regiaoInicial: caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodasRegioesIniciais) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(regioesIniciais) passo +1 faca
      Se (regioesIniciais[j]=regiaoInicial) entao
        percorreuTodasRegioesIniciais[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodasRegioesIniciais
fimFuncao

funcao passouPorOrientacaoInicial(orientacaoInicial: caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodasOrientacoesIniciais) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(orientacoesIniciais) passo +1 faca
      Se (orientacoesIniciais[j]=orientacaoInicial) entao
        percorreuTodasOrientacoesIniciais[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodasOrientacoesIniciais
fimFuncao

funcao passouPorSentido(sentido: caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodosSentidos) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(sentidosDeRotacao) passo +1 faca
      Se (sentidosDeRotacao[j]=sentido) entao
        percorreuTodosSentidos[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodosSentidos
fimFuncao
```

```
funcao passouPorVelocidade(velocidade: caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodasVelocidades) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(velocidades) passo +1 faca
      Se (velocidades[j]=velocidade) entao
        percorreuTodasVelocidades[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodasVelocidades
fimFuncao

funcao passouPorRegiaoFinal(regiaoFinal: caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodasRegioesFinais) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(regioesFinais) passo +1 faca
      Se (regioesFinais[j]=regiaoFinal) entao
        percorreuTodasRegioesFinais[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodasRegioesFinais
fimFuncao

funcao transformaEmTransicao(estadoOrigem, veloc, sentido, estadoDestino: caracter): caracter
Var v,s,t: caracter
Inicio
  Se (sentido="Horário") entao
    s ←"Horário"
  senao
    s ← "Anti-horário"
  fimSe
  Se (veloc="Devagar") entao
    v ← "Devagar"
  senao
    v ← "Rápida"
  fimSe
  t ← estadoOrigem + "-" + v + "-" + s + "-" + estadoDestino
  retorne t
fimFuncao

funcao passouPorTransicao(transicao: caracter):logico[]
Inicio
  Para inteiro i ← 0 ate i < tamanho(percorreuTodasTransicoes) passo +1 faca
    Para inteiro j ← 0 ate j < tamanho(todasTransicoes) passo +1 faca
      Se (todasTransicoes[j]=transicao) entao
        percorreuTodasTransicoes[j] ← verdadeiro
      fimSe
    fimPara
  fimPara
  retorne percorreuTodasTransicoes
fimFuncao

funcao coberturaNormalForte(): real
Var cont: inteiro
Inicio
  cont ←0
  Para inteiro i ← 0 ate i < tamanho(percorreuTodosNormalForte) passo +1 faca
    Se (percorreuTodosNormalForte[i] = verdadeiro) entao
      cont ← cont+1
    fimSe
  fimPara
  escolha (cont)
    caso 1
      porcentagemNormalForte ← (1/256)*100
    caso 2
      porcentagemNormalForte ← (2/256)*100
    caso 3
      porcentagemNormalForte ← (3/256)*100
    caso 4
      porcentagemNormalForte ← (4/256)*100
    caso 5
      porcentagemNormalForte ← (5/256)*100
    caso 6
      porcentagemNormalForte ← (6/256)*100
    caso 7
      porcentagemNormalForte ← (7/256)*100
    caso 8
```

```
    porcentagemNormalForte ← (8/256)*100
  caso 9
    porcentagemNormalForte ← (9/256)*100
  caso 10
    porcentagemNormalForte ← (10/256)*100
  caso 11
    porcentagemNormalForte ← (11/256)*100
  caso 12
    porcentagemNormalForte ← (12/256)*100
  caso 13
    porcentagemNormalForte ← (13/256)*100
  caso 14
    porcentagemNormalForte ← (14/256)*100
  caso 15
    porcentagemNormalForte ← (15/256)*100
  caso 16
    porcentagemNormalForte ← (16/256)*100
  caso 17
    porcentagemNormalForte ← (17/256)*100
  caso 18
    porcentagemNormalForte ← (18/256)*100
  caso 19
    porcentagemNormalForte ← (19/256)*100
  caso 20
    porcentagemNormalForte ← (20/256)*100
  caso 21
    porcentagemNormalForte ← (21/256)*100
  caso 22
    porcentagemNormalForte ← (22/256)*100
  caso 23
    porcentagemNormalForte ← (23/256)*100
  caso 24
    porcentagemNormalForte ← (24/256)*100
  caso 25
    porcentagemNormalForte ← (25/256)*100
  caso 26
    porcentagemNormalForte ← (26/256)*100
  caso 27
    porcentagemNormalForte ← (27/256)*100
  caso 28
    porcentagemNormalForte ← (28/256)*100
  caso 29
    porcentagemNormalForte ← (29/256)*100
  caso 30
    porcentagemNormalForte ← (30/256)*100
  caso 31
    porcentagemNormalForte ← (31/256)*100
  caso 32
    porcentagemNormalForte ← (32/256)*100
  caso 33
    porcentagemNormalForte ← (33/256)*100
  caso 34
    porcentagemNormalForte ← (34/256)*100
  caso 35
    porcentagemNormalForte ← (35/256)*100
  caso 36
    porcentagemNormalForte ← (36/256)*100
  caso 37
    porcentagemNormalForte ← (37/256)*100
  caso 38
    porcentagemNormalForte ← (38/256)*100
  caso 39
    porcentagemNormalForte ← (39/256)*100
  caso 40
    porcentagemNormalForte ← (40/256)*100
  caso 41
    porcentagemNormalForte ← (41/256)*100
  caso 42
    porcentagemNormalForte ← (42/256)*100
  caso 43
    porcentagemNormalForte ← (43/256)*100
  caso 44
    porcentagemNormalForte ← (44/256)*100
  caso 45
    porcentagemNormalForte ← (45/256)*100
  caso 46
    porcentagemNormalForte ← (46/256)*100
  caso 47
    porcentagemNormalForte ← (47/256)*100
```

```
caso 48
  porcentagemNormalForte ← (48/256)*100
caso 49
  porcentagemNormalForte ← (49/256)*100
caso 50
  porcentagemNormalForte ← (50/256)*100
caso 51
  porcentagemNormalForte ← (51/256)*100
caso 52
  porcentagemNormalForte ← (52/256)*100
caso 53
  porcentagemNormalForte ← (53/256)*100
caso 54
  porcentagemNormalForte ← (54/256)*100
caso 55
  porcentagemNormalForte ← (55/256)*100
caso 56
  porcentagemNormalForte ← (56/256)*100
caso 57
  porcentagemNormalForte ← (57/256)*100
caso 58
  porcentagemNormalForte ← (58/256)*100
caso 59
  porcentagemNormalForte ← (59/256)*100
caso 60
  porcentagemNormalForte ← (60/256)*100
caso 61
  porcentagemNormalForte ← (61/256)*100
caso 62
  porcentagemNormalForte ← (62/256)*100
caso 63
  porcentagemNormalForte ← (63/256)*100
caso 64
  porcentagemNormalForte ← (64/256)*100
caso 65
  porcentagemNormalForte ← (65/256)*100
caso 66
  porcentagemNormalForte ← (66/256)*100
caso 67
  porcentagemNormalForte ← (67/256)*100
caso 68
  porcentagemNormalForte ← (68/256)*100
caso 69
  porcentagemNormalForte ← (69/256)*100
caso 70
  porcentagemNormalForte ← (70/256)*100
caso 71
  porcentagemNormalForte ← (71/256)*100
caso 72
  porcentagemNormalForte ← (72/256)*100
caso 73
  porcentagemNormalForte ← (73/256)*100
caso 74
  porcentagemNormalForte ← (74/256)*100
caso 75
  porcentagemNormalForte ← (75/256)*100
caso 76
  porcentagemNormalForte ← (76/256)*100
caso 77
  porcentagemNormalForte ← (77/256)*100
caso 78
  porcentagemNormalForte ← (78/256)*100
caso 79
  porcentagemNormalForte ← (79/256)*100
caso 80
  porcentagemNormalForte ← (80/256)*100
caso 81
  porcentagemNormalForte ← (81/256)*100
caso 82
  porcentagemNormalForte ← (82/256)*100
caso 83
  porcentagemNormalForte ← (83/256)*100
caso 84
  porcentagemNormalForte ← (84/256)*100
caso 85
  porcentagemNormalForte ← (85/256)*100
caso 86
  porcentagemNormalForte ← (86/256)*100
caso 87
```

```
    porcentagemNormalForte ← (87/256)*100
  caso 88
    porcentagemNormalForte ← (88/256)*100
  caso 89
    porcentagemNormalForte ← (89/256)*100
  caso 90
    porcentagemNormalForte ← (90/256)*100
  caso 91
    porcentagemNormalForte ← (91/256)*100
  caso 92
    porcentagemNormalForte ← (92/256)*100
  caso 93
    porcentagemNormalForte ← (93/256)*100
  caso 94
    porcentagemNormalForte ← (94/256)*100
  caso 95
    porcentagemNormalForte ← (95/256)*100
  caso 96
    porcentagemNormalForte ← (96/256)*100
  caso 97
    porcentagemNormalForte ← (97/256)*100
  caso 98
    porcentagemNormalForte ← (98/256)*100
  caso 99
    porcentagemNormalForte ← (99/256)*100
  caso 100
    porcentagemNormalForte ← (100/256)*100
  caso 101
    porcentagemNormalForte ← (101/256)*100
  caso 102
    porcentagemNormalForte ← (102/256)*100
  caso 103
    porcentagemNormalForte ← (103/256)*100
  caso 104
    porcentagemNormalForte ← (104/256)*100
  caso 105
    porcentagemNormalForte ← (105/256)*100
  caso 106
    porcentagemNormalForte ← (106/256)*100
  caso 107
    porcentagemNormalForte ← (107/256)*100
  caso 108
    porcentagemNormalForte ← (108/256)*100
  caso 109
    porcentagemNormalForte ← (109/256)*100
  caso 110
    porcentagemNormalForte ← (110/256)*100
  caso 111
    porcentagemNormalForte ← (111/256)*100
  caso 112
    porcentagemNormalForte ← (112/256)*100
  caso 113
    porcentagemNormalForte ← (113/256)*100
  caso 114
    porcentagemNormalForte ← (114/256)*100
  caso 115
    porcentagemNormalForte ← (115/256)*100
  caso 116
    porcentagemNormalForte ← (116/256)*100
  caso 117
    porcentagemNormalForte ← (117/256)*100
  caso 118
    porcentagemNormalForte ← (118/256)*100
  caso 119
    porcentagemNormalForte ← (119/256)*100
  caso 120
    porcentagemNormalForte ← (120/256)*100
  caso 121
    porcentagemNormalForte ← (121/256)*100
  caso 122
    porcentagemNormalForte ← (122/256)*100
  caso 123
    porcentagemNormalForte ← (123/256)*100
  caso 124
    porcentagemNormalForte ← (124/256)*100
  caso 125
    porcentagemNormalForte ← (125/256)*100
  caso 126
    porcentagemNormalForte ← (126/256)*100
```

```
caso 127
  porcentagemNormalForte ← (127/256)*100
caso 128
  porcentagemNormalForte ← (128/256)*100
caso 129
  porcentagemNormalForte ← (129/256)*100
caso 130
  porcentagemNormalForte ← (130/256)*100
caso 131
  porcentagemNormalForte ← (131/256)*100
caso 132
  porcentagemNormalForte ← (132/256)*100
caso 133
  porcentagemNormalForte ← (133/256)*100
caso 134
  porcentagemNormalForte ← (134/256)*100
caso 135
  porcentagemNormalForte ← (135/256)*100
caso 136
  porcentagemNormalForte ← (136/256)*100
caso 137
  porcentagemNormalForte ← (137/256)*100
caso 138
  porcentagemNormalForte ← (138/256)*100
caso 139
  porcentagemNormalForte ← (139/256)*100
caso 140
  porcentagemNormalForte ← (140/256)*100
caso 141
  porcentagemNormalForte ← (141/256)*100
caso 142
  porcentagemNormalForte ← (142/256)*100
caso 143
  porcentagemNormalForte ← (143/256)*100
caso 144
  porcentagemNormalForte ← (144/256)*100
caso 145
  porcentagemNormalForte ← (145/256)*100
caso 146
  porcentagemNormalForte ← (146/256)*100
caso 147
  porcentagemNormalForte ← (147/256)*100
caso 148
  porcentagemNormalForte ← (148/256)*100
caso 149
  porcentagemNormalForte ← (149/256)*100
caso 150
  porcentagemNormalForte ← (150/256)*100
caso 151
  porcentagemNormalForte ← (151/256)*100
caso 152
  porcentagemNormalForte ← (152/256)*100
caso 153
  porcentagemNormalForte ← (153/256)*100
caso 154
  porcentagemNormalForte ← (154/256)*100
caso 155
  porcentagemNormalForte ← (155/256)*100
caso 156
  porcentagemNormalForte ← (156/256)*100
caso 157
  porcentagemNormalForte ← (157/256)*100
caso 158
  porcentagemNormalForte ← (158/256)*100
caso 159
  porcentagemNormalForte ← (159/256)*100
caso 160
  porcentagemNormalForte ← (160/256)*100
caso 161
  porcentagemNormalForte ← (161/256)*100
caso 162
  porcentagemNormalForte ← (162/256)*100
caso 163
  porcentagemNormalForte ← (163/256)*100
caso 164
  porcentagemNormalForte ← (164/256)*100
caso 165
  porcentagemNormalForte ← (165/256)*100
caso 166
```

```
    porcentagemNormalForte ← (166/256)*100
  caso 167
    porcentagemNormalForte ← (167/256)*100
  caso 168
    porcentagemNormalForte ← (168/256)*100
  caso 169
    porcentagemNormalForte ← (169/256)*100
  caso 170
     porcentagemNormalForte ← (170/256)*100
  caso 171
     porcentagemNormalForte ← (171/256)*100
  caso 172
     porcentagemNormalForte ← (172/256)*100
  caso 173
    porcentagemNormalForte ← (173/256)*100
  caso 174
    porcentagemNormalForte ← (174/256)*100
  caso 175
    porcentagemNormalForte ← (175/256)*100
  caso 176
    porcentagemNormalForte ← (176/256)*100
  caso 177
    porcentagemNormalForte ← (177/256)*100
  caso 178
    porcentagemNormalForte ← (178/256)*100
  caso 179
    porcentagemNormalForte ← (179/256)*100
  caso 180
    porcentagemNormalForte ← (180/256)*100
  caso 181
    porcentagemNormalForte ← (181/256)*100
  caso 182
    porcentagemNormalForte ← (182/256)*100
  caso 183
    porcentagemNormalForte ← (183/256)*100
  caso 184
    porcentagemNormalForte ← (184/256)*100
  caso 185
    porcentagemNormalForte ← (185/256)*100
  caso 186
    porcentagemNormalForte ← (186/256)*100
  caso 187
    porcentagemNormalForte ← (187/256)*100
  caso 188
    porcentagemNormalForte ← (188/256)*100
  caso 189
    porcentagemNormalForte ← (189/256)*100
  caso 190
    porcentagemNormalForte ← (190/256)*100
  caso 191
    porcentagemNormalForte ← (191/256)*100
  caso 192
    porcentagemNormalForte ← (192/256)*100
  caso 193
    porcentagemNormalForte ← (193/256)*100
  caso 194
    porcentagemNormalForte ← (194/256)*100
  caso 195
    porcentagemNormalForte ← (195/256)*100
  caso 196
    porcentagemNormalForte ← (196/256)*100
  caso 197
    porcentagemNormalForte ← (197/256)*100
  caso 198
    porcentagemNormalForte ← (198/256)*100
  caso 199
    porcentagemNormalForte ← (199/256)*100
  caso 200
    porcentagemNormalForte ← (200/256)*100
  caso 201
    porcentagemNormalForte ← (201/256)*100
  caso 202
    porcentagemNormalForte ← (202/256)*100
  caso 203
    porcentagemNormalForte ← (203/256)*100
  caso 204
    porcentagemNormalForte ← (204/256)*100
  caso 205
    porcentagemNormalForte ← (205/256)*100
```

```
caso 206
  porcentagemNormalForte ← (206/256)*100
caso 207
  porcentagemNormalForte ← (207/256)*100
caso 208
  porcentagemNormalForte ← (208/256)*100
caso 209
  porcentagemNormalForte ← (209/256)*100
caso 210
  porcentagemNormalForte ← (210/256)*100
caso 211
  porcentagemNormalForte ← (211/256)*100
caso 212
  porcentagemNormalForte ← (212/256)*100
caso 213
  porcentagemNormalForte ← (213/256)*100
caso 214
  porcentagemNormalForte ← (214/256)*100
caso 215
  porcentagemNormalForte ← (215/256)*100
caso 216
  porcentagemNormalForte ← (216/256)*100
caso 217
  porcentagemNormalForte ← (217/256)*100
caso 218
  porcentagemNormalForte ← (218/256)*100
caso 219
  porcentagemNormalForte ← (219/256)*100
caso 220
  porcentagemNormalForte ← (220/256)*100
caso 221
  porcentagemNormalForte ← (221/256)*100
caso 222
  porcentagemNormalForte ← (222/256)*100
caso 223
  porcentagemNormalForte ← (223/256)*100
caso 224
  porcentagemNormalForte ← (224/256)*100
caso 225
  porcentagemNormalForte ← (225/256)*100
caso 226
  porcentagemNormalForte ← (226/256)*100
caso 227
  porcentagemNormalForte ← (227/256)*100
caso 228
  porcentagemNormalForte ← (228/256)*100
caso 229
  porcentagemNormalForte ← (229/256)*100
caso 230
  porcentagemNormalForte ← (230/256)*100
caso 231
  porcentagemNormalForte ← (231/256)*100
caso 232
  porcentagemNormalForte ← (232/256)*100
caso 233
  porcentagemNormalForte ← (233/256)*100
caso 234
  porcentagemNormalForte ← (234/256)*100
caso 235
  porcentagemNormalForte ← (235/256)*100
caso 236
  porcentagemNormalForte ← (236/256)*100
caso 237
  porcentagemNormalForte ← (237/256)*100
caso 238
  porcentagemNormalForte ← (238/256)*100
caso 239
  porcentagemNormalForte ← (239/256)*100
caso 240
  porcentagemNormalForte ← (240/256)*100
caso 241
  porcentagemNormalForte ← (241/256)*100
caso 242
  porcentagemNormalForte ← (242/256)*100
caso 243
  porcentagemNormalForte ← (243/256)*100
caso 244
  porcentagemNormalForte ← (244/256)*100
caso 245
```

```
      porcentagemNormalForte ← (245/256)*100
    caso 246
      porcentagemNormalForte ← (246/256)*100
    caso 247
      porcentagemNormalForte ← (247/256)*100
    caso 248
      porcentagemNormalForte ← (248/256)*100
    caso 249
      porcentagemNormalForte ← (249/256)*100
    caso 250
      porcentagemNormalForte ← (250/256)*100
    caso 251
      porcentagemNormalForte ← (251/256)*100
    caso 252
      porcentagemNormalForte ← (252/256)*100
    caso 253
      porcentagemNormalForte ← (253/256)*100
    caso 254
      porcentagemNormalForte ← (254/256)*100
    caso 255
      porcentagemNormalForte ← (255/256)*100
    caso 256
      porcentagemNormalForte ← (256/256)*100
  fimEscolha
  retorne porcentagemNormalForte
fimFuncao

funcao coberturaNormalFraco(ri1, oi2, s3, v4, rf5: logico[]): real
Var cont: inteiro
Inicio
  cont ←0
  Para inteiro i ← 0 ate i < tamanho(ri1) passo +1 faca
    Se (ri1[i] = verdadeiro) entao
      cont ← cont+1
    fimSe
  fimPara
  Para inteiro i ← 0 ate i < tamanho(oi2) passo +1 faca
    Se (oi2[i] = verdadeiro) entao
      cont ← cont+1
    fimSe
  fimPara
  Para inteiro i ← 0 ate i < tamanho(s3) passo +1 faca
    Se (s3[i] = verdadeiro) entao
      cont ← cont+1
    fimSe
  fimPara
  Para inteiro i ← 0 ate i < tamanho(v4) passo +1 faca
    Se (v4[i] = verdadeiro) entao
      cont ← cont+1
    fimSe
  fimPara
  Para inteiro i ← 0 ate i < tamanho(rf5) passo +1 faca
    Se (rf5[i] = verdadeiro) entao
      cont ← cont+1
    fimSe
  fimPara
  escolha (cont)
    caso 1
      porcent'agemNormalFraco ← (1/16)*100
    caso 2
      porcentagemNormalFraco ← (2/16)*100
    caso 3
      porcentagemNormalFraco ← (3/16)*100
    caso 4
      porcentagemNormalFraco ← (4/16)*100
    caso 5
      porcentagemNormalFraco ← (5/16)*100
    caso 6
      porcentagemNormalFraco ← (6/16)*100
    caso 7
      porcentagemNormalFraco ← (7/16)*100
    caso 8
      porcentagemNormalFraco ← (8/16)*100
    caso 9
      porcentagemNormalFraco ← (9/16)*100
    caso 10
      porcentagemNormalFraco ← (10/16)*100
    caso 11
      porcentagemNormalFraco ← (11/16)*100
```

```
      caso 12
        porcentagemNormalFraco ← (12/16)*100
      caso 13
        porcentagemNormalFraco ← (13/16)*100
      caso 14
        porcentagemNormalFraco ← (14/16)*100
      caso 15
        porcentagemNormalFraco ← (15/16)*100
      caso 16
        porcentagemNormalFraco ← (16/16)*100
    fimEscolha
    retorne porcentagemNormalFraco
fimFuncao

funcao coberturaTodosOsEstados(): real
Var cont: inteiro
Inicio
    cont ←0
    Para inteiro i ← 0 ate i < tamanho(percorreuTodosEstados) passo +1 faca
      Se (percorreuTodosEstados[i] = verdadeiro) entao
        cont ← cont+1
      fimSe
    fimPara
    escolha (cont)
      caso 1
        porcentagemTodosOsEstados ← (1/6)*100
      caso 2
        porcentagemTodosOsEstados ← (2/6)*100
      caso 3
        porcentagemTodosOsEstados ← (3/6)*100
      caso 4
        porcentagemTodosOsEstados ← (4/6)*100
      caso 5
        porcentagemTodosOsEstados ← (5/6)*100
      caso 6
        porcentagemTodosOsEstados ← (6/6)*100
    fimEscolha
    retorne porcentagemTodosOsEstados
fimFuncao

funcao coberturaTodaAsTransicoes(): real
Var cont: inteiro
Inicio
    cont ←0
    Para inteiro i ← 0 ate i < tamanho(percorreuTodasTransicoes) passo +1 faca
      Se (percorreuTodasTransicoes[i] = verdadeiro) entao
        cont ← cont+1
      fimSe
    fimPara
    escolha (cont)
      caso 1
        porcentagemTodasAsTransicoes ← (1/42)*100
      caso 2
        porcentagemTodasAsTransicoes ← (2/42)*100
      caso 3
        porcentagemTodasAsTransicoes ← (3/42)*100
      caso 4
        porcentagemTodasAsTransicoes ← (4/42)*100
      caso 5
        porcentagemTodasAsTransicoes ← (5/42)*100
      caso 6
        porcentagemTodasAsTransicoes ← (6/42)*100
      caso 7
        porcentagemTodasAsTransicoes ← (7/42)*100
      caso 8
        porcentagemTodasAsTransicoes ← (8/42)*100
      caso 9
        porcentagemTodasAsTransicoes ← (9/42)*100
      caso 10
        porcentagemTodasAsTransicoes ← (10/42)*100
      caso 11
        porcentagemTodasAsTransicoes ← (11/42)*100
      caso 12
        porcentagemTodasAsTransicoes ← (12/42)*100
      caso 13
        porcentagemTodasAsTransicoes ← (13/42)*100
      caso 14
        porcentagemTodasAsTransicoes ← (14/42)*100
      caso 15
```

```
      porcentagemTodasAsTransicoes ← (15/42)*100
    caso 16
      porcentagemTodasAsTransicoes ← (16/42)*100
    caso 17
      porcentagemTodasAsTransicoes ← (17/42)*100
    caso 18
      porcentagemTodasAsTransicoes ← (18/42)*100
    caso 19
      porcentagemTodasAsTransicoes ← (19/42)*100
    caso 20
      porcentagemTodasAsTransicoes ← (20/42)*100
    caso 21
      porcentagemTodasAsTransicoes ← (21/42)*100
    caso 22
      porcentagemTodasAsTransicoes ← (22/42)*100
    caso 23
      porcentagemTodasAsTransicoes ← (23/42)*100
    caso 24
      porcentagemTodasAsTransicoes ← (24/42)*100
    caso 25
      porcentagemTodasAsTransicoes ← (25/42)*100
    caso 26
      porcentagemTodasAsTransicoes ← (26/42)*100
    caso 27
      porcentagemTodasAsTransicoes ← (27/42)*100
    caso 28
      porcentagemTodasAsTransicoes ← (28/42)*100
    caso 29
      porcentagemTodasAsTransicoes ← (29/42)*100
    caso 30
      porcentagemTodasAsTransicoes ← (30/42)*100
    caso 31
      porcentagemTodasAsTransicoes ← (31/42)*100
    caso 32
      porcentagemTodasAsTransicoes ← (32/42)*100
    caso 33
      porcentagemTodasAsTransicoes ← (33/42)*100
    caso 34
      porcentagemTodasAsTransicoes ← (34/42)*100
    caso 35
      porcentagemTodasAsTransicoes ← (35/42)*100
    caso 36
      porcentagemTodasAsTransicoes ← (36/42)*100
    caso 37
      porcentagemTodasAsTransicoes ← (37/42)*100
    caso 38
      porcentagemTodasAsTransicoes ← (38/42)*100
    caso 39
      porcentagemTodasAsTransicoes ← (39/42)*100
    caso 40
      porcentagemTodasAsTransicoes ← (40/42)*100
    caso 41
      porcentagemTodasAsTransicoes ← (41/42)*100
    caso 42
      porcentagemTodasAsTransicoes ← (42/42)*100
  fimEscolha
  retorne porcentagemTodasAsTransicoes
fimFuncao
fimAlgoritmo
```

Autoria própria.