FUNDAÇÃO DE "ENSINO EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" - UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

ELISANGELA CITRO TURCI

Utilização de Teste Estrutural em Código Móvel Java

MARÍLIA/SP
2005

ELISANGELA CITRO TURCI

Utilização de Teste Estrutural em Código Móvel Java

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação.

Orientador:
Prof Dr. Márcio Eduardo Delamaro.

MARÍLIA/SP

2005

TURCI, Elisangela Citro.
Utilização de Teste Estrutural em Código Móvel Java / Elisangela Citro Turci; Orientador: Márcio Eduardo Delamaro. Marília/SP:[s.n.], 2005

157 f.

Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília - Fundação de Ensino Eurípides Soares da Rocha.

1.Código Móvel. 2. Teste Estrutural. 3. Ferramenta de Teste.

CDD: 005.2

ELISANGELA CITRO TURCI

Utilização de Teste Estrutural em Código Móvel Java

Banca Examinadora da dissertação apresentada ao Programa de Mestrado em Ciência da Computação do Centro Universitário Eurípides de Marília - UNIVEM, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação.

Resultado:_____

ORIENTADOR: Prof. Dr. Márcio Eduardo Delamaro

1º. EXAMINADOR: _____

2º. EXAMINADOR: _____

Marília/SP, ____ de _____ de 2005.

Ao meu querido esposo, Marcos Rogério Okuyama Turci, com muito amor, por sempre me incentivar nos estudos e, acima de tudo, pelo companheirismo, carinho, amor e compreensão durante o desenvolvimento deste trabalho.

AGRADECIMENTOS

Primeiramente agradeço à Deus pelo amparo, proteção e ânimo para a conclusão deste trabalho. Com profundo amor agradeço aos meus pais que sempre me incentivaram aos estudos, de modo a me permitirem galgar mais um degrau. E, fraternalmente agradeço aos meus sogros pelo carinho, paciência e compreenssão durante o desenvolvimento deste.

Em especial, agradeço a meu querido esposo por estar sempre ao meu lado, me protegendo e me aconselhando em momentos tão decisivos, sustentando a minha fé.

Agradeço com muito carinho a todos os meus familiares, em particular a minha irmã Edimari e a minha cunhada Ellen pelo incentivo e ajuda. Também agradeço a minhas queridas amigas Elzi e Nildeth que me apoiaram neste momento tão especial e importante para mim.

Com ressalvas quero externar meu profundo desvelo pelo meu querido professor e orientador, Márcio Eduardo Delamaro, por me compreender em momentos tão difíceis da minha vida e por me amparar no término deste trabalho, muito obrigada!

Também deixo aqui registrado minha estima e meu agradecimento pelo aluno de iniciação científica, Gustavo Rondina, que não mediu esforços para me ajudar na conclusão deste.

Agradeço com profundo respeito e afeto aos meus queridos professores de mestrado, em especial Dino, Fátima, Mucheroni e Jorge pelos ensinamentos, e sobre tudo pela amizade. Obrigada!

E, por fim, agradeço com todo amor e amizade aos meus queridos amigos de mestrado os "Fora da Curva" (Léo, Piva, Luiz, Claudia, Adriane, Lucilena, Gislene e Rosiane) pelos sorrisos, abraços e risos. Por serem presentes mesmo longe. Por serem amigos de modo infinito, de modo a não poder esquecer, a não poder escrever em palavras exatas a definição de nossa amizade. De modo a perpetuarem em um cantinho do meu coração. Obrigada pelo carinho, só posso retribuir com muitos risos e bom humor. Que Deus abençoe a todos, e um sincero muito obrigada!

“Tente, e não diga que a vitória está perdida, se é de batalhas que se vive a vida.
Tente outra vez.”

Raul Seixas

TURCI, Elisangela Citro. **Utilização de Teste Estrutural em Código Móvel Java**. 2004. x f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília/SP, 2004.

RESUMO

Este trabalho utiliza alguns critérios do teste estrutural com o intuito de testar código móvel através de uma ferramenta de auxílio à atividade de teste. Para tanto, especifica os mecanismos que dão suporte à mobilidade de código, os ambientes e linguagens de desenvolvimento. Utiliza uma API desenvolvida em Java, denominada μCODE, para criar os exemplos de mobilidade. Descreve um estudo sobre o teste estrutural e seus critérios, enfatizando os de fluxo de dados e de controle, e algumas ferramentas de teste. Como auxílio à atividade de teste emprega a ferramenta JaBUTi, (VINCENZI et al., 2003), que testa programas ou componentes Java, utiliza critérios de teste estrutural e suporta o teste em código móvel.

Palavras-chave: Código Móvel. Teste Estrutural. Ferramenta de Teste.

TURCI, Elisangela Citro. **Utilização de Teste Estrutural en Código Móvel Java** . 2004. x f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília/SP, 2004.

ABSTRACT

This work aims to present mobile code testing by using some structural test criteria and a testing support tool. To such purpose, the mechanisms which support code mobility, the environments and the development languages will be specified. An API developed in Java, named Code, is used to create the examples of mobility. Also, it will be described a study on structural test and its criteria, emphasizing the data flow and control criteria, and some testing tools. The employed testing support tool is JaBUTi (VICENZI, 2003), which tests Java programs or components, uses structural testing criteria and supports mobile code testing.

Keywords: Mobile code. Structural testing. Testing tools.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Em meio a tanta tecnologia, é fácil notar a proliferação de sistemas baseados em computador. Pois basta observar pequenas empresas que se utilizam desta tecnologia. Em adição, tem-se o paradigma distribuído que só veio a disseminar amplamente os sistemas computacionais, agora, distribuídos, facilitando o acesso as informações e popularizando o meio computacional de forma a não haver retrocesso.

Com tanta tecnologia a disposição, é imprescindível a boa distribuição de tais tecnologias. Assim, sistemas computacionais distribuídos em larga escala requerem uma atenção maior quanto à sua disponibilização, devido as questões de perfomance.

Dentro deste contexto, surge o termo "código móvel" em pesquisas de sistemas operacionais distribuídos (TANENBAUM; STEEN, 2002). Assim, sistemas de código móvel vêm oferecer uma nova forma de construção de sistemas distribuídos com o propósito de auxiliar e minimizar os problemas que circundam as aplicações distribuídas.

Não obstante, tais sistemas como qualquer outro precisam ser executados precisamente sem a ocorrência de erros ou falhas. Em determinadas situações, o risco é tão alto que não há como desfazer de uma fase tão importante, dentro do processo de desenvolvimento de *software*, como a fase de teste.

Segundo Pressman, (PRESSMAN, 1995), a atividade de teste é um elemento crítico da garantia de qualidade de *software* e representa a última revisão de especificação, projeto e codificação. Assim, o teste de *software* vem como um dos meios principais para mostrar a corretitude e promover a distribuição segura de um *software*.

Para contribuir com este cenário da engenharia de software foram desenvolvidas várias técnicas de teste, dentre elas temos a técnica estrutural, que se baseia no código fonte do programa de *software*. Tal técnica apresenta vários critérios de teste que serão

melhor abordados no Capítulo 3.

Vale ressaltar que, muitas ferramentas de teste foram desenvolvidas com o intuito de auxiliar a atividade de teste de *software*, como exemplo tem-se Chaim (CHAIM, 1991), Frankl e Weyuker (FRANKL P. G.; WEISS; WEYUKER, 1985). Neste trabalho serão apresentadas várias ferramentas que apoiam o teste estrutural, porém será melhor apresentada e utilizada, como *software* de apoio ao teste estrutural, a Ferramenta JaBUTi que permite o teste em programas escritos em Java e apresenta uma extensão adaptada para suportar a mobilidade de código oferecida pelo ambiente μCODE, procurando atender o teste em código móvel (DELAMARO; VINCENZI; MALDONADO, 2004).

1.1 Objetivo

O intuito deste trabalho é expor, como meio para validar a corretitude de sistemas de código móvel, a utilização do teste estrutural em aplicações distribuídas que se utilizam do paradigma de agentes móveis.

A dificuldade deste projeto estava em aplicar e avaliar o teste estrutural em código móvel, devido à própria característica do código. O código móvel pode trafegar em um ambiente distribuído tornando difícil saber quais partes do código foram executadas bem como a ordem de execução das mesmas. Portanto, este projeto possibilitou a verificação do comportamento do código enquanto se movimentava pelos nós de uma rede, capturando todas as informações de seu estado em cada nó durante o trajeto percorrido.

Com base neste cenário de mobilidade de código, o teste de código móvel requer o suporte de uma ferramenta que seja robusta o suficiente para conseguir capturar todo o processo de execução do código móvel, ao longo do percurso por onde ele se locomove. Assim, este trabalho além de averiguar a possibilidade da utilização do teste estrutural em códigos móveis, contribuiu com a adaptação da Ferramenta JaBUTi, (DELAMARO; VINCENZI; MALDONADO, 2004), auxiliando na extensão de suas funcionalidades objetivando suportar o teste de código móvel.

Estudos de casos foram escritos e apresentados com o intuito de estudar a aplicação de teste estrutural em código móvel, e conseqüentemente permitir a avaliação da robustez da extensão da Ferramenta JaBUTi contribuindo com a inclusão de melhorias na mesma.

1.2 Organização do Texto

A introdução apresentou um resumo sobre o cenário atual para sistemas distribuídos e, conseqüentemente, as motivações para o desenvolvimento deste projeto. No Capítulo 2 é apresentado o conceito de mobilidade de código, mostrando os mecanimos que dão suporte a mobilidade e os ambientes de implementação de códigos móveis. O Capítulo 3 apresenta a importância do teste de software, e se concentra na apresentação da técnica de teste estrutural e os critérios que a compõe. A Ferramenta JaBUTi é escrita no Capítulo 4 apresentando suas funcionalidades, sua forma de análise de cobertura do código em teste e sua adaptação para suportar o teste de código móvel. O Capítulo 5 apresenta os estudos de casos implementados e utilizados para a aplicação das atividades de teste e avaliação dos resultados. E, por fim, o trabalho é concluído no Capítulo 6 sendo apresentadas as considerações finais e os trabalhos futuros.

2 Mobilidade

A chegada das redes de computadores proporcionou o aumento da utilização dos computadores e popularizou seu uso. Tal tecnologia tem crescido a passos largos devido não só à facilidade de uso, mas também ao custo baixo da infra-estrutura de *hardware* aplicada para sua implantação. Ao mesmo tempo em que tal crescimento colabora com as aplicações computacionais, em contra partida, aumenta o tráfego da rede. Este aumento do tráfego é um dos fatores preponderantes para a pesquisa na área de performance na comunicação em rede (FUGGETTA; PICCO; VIGNA, 1998).

O aumento e a melhoria de performance da rede, além de facilitar a computação, torna possível a exploração da conectividade de rede independente da localização física do usuário, ou seja, possibilita o advento da computação móvel. Assim, tem-se que, usuários podem estar se conectando na rede hora em um local, hora em outro, independente da sua localização geográfica utilizando-se para tanto recursos da tecnologia sem fio.

A Internet é outro fator importante que ajudou a mudar o papel das redes de computadores, gerando novos domínios de aplicações, sendo que a *World Wide Web*[2.1] é uma das tecnologias disponibilizadas pela Internet que mais contribui para o crescimento da mesma (FUGGETTA; PICCO; VIGNA, 1998).

Com a evolução das redes de computadores surgiram vários problemas, como a queda do desempenho em larga escala, a topologia da rede deixa de ser estática, dificuldades na difusão de serviços, busca de informações e execução de tarefas distribuídas. Como se pode ver, a infra-estrutura da comunicação básica da rede passa a requerer maior flexibilidade e extensibilidade.

A mobilidade de código é uma nova técnica para auxiliar na solução desses proble-

[2.1] A *World Wide Web* (também chamada *Web* ou WWW) é um sistema de informações organizado de maneira a englobar todos os outros sistemas de informação disponíveis na Internet, permitindo o intercâmbio destas informações (CASTRO, 1995).

mas. Apesar de não ser um novo conceito, ela está em fase de amadurecimento, faltando-lhe ainda uma boa estrutura de conceitos e métodos (PICCO, 2000).

Conforme Fuggeta (FUGGETTA; PICCO; VIGNA, 1998) e Picco (PICCO, 2000), a mobilidade de código pode ser definida, informalmente, como a capacidade de mudar dinamicamente as ligações entre fragmentos de código e o local onde eles são executados. Detalhes sobre o suporte à mobilidade de código estão descritos na Seção 2.1.

O desenvolvimento de Sistemas de Código Móvel se deu com pesquisas na área de sistemas operacionais distribuídos que utilizam técnicas de migração de processos (TANENBAUM; STEEN, 2002). A técnica de migração de processos, em (FUGGETTA; PICCO; VIGNA, 1998), é descrita como a transferência de um processo do sistema operacional da máquina onde ele está rodando para uma máquina diferente. Acrescentando, tal migração se torna insuficiente quando aplicada em larga escala.

Sistemas de Código Móvel são concebidos para operar em configurações de larga escala, conhecem o local de execução e podem agir com base neste conhecimento, e a mobilidade fica sob controle do programador. Estes sistemas também se preocupam com a adaptação de serviços, extensão dinâmica da funcionalidade da aplicação, autonomia, tolerância à falha e suporte às operações desconectadas.

Também é importante apresentar o termo "agente móvel". Segundo Fuggeta (FUGGETTA; PICCO; VIGNA, 1998), na área de sistemas distribuídos, este termo é utilizado para designar um componente de *software* que é capaz de se movimentar entre diferentes ambientes de execução.

Esta capacidade, que permite que programas possam trafegar livremente pela rede, depende da tecnologia de código móvel empregada. Tal tecnologia engloba linguagens de programação e seus suportes em tempo de execução.

Fuggeta *et al.* (FUGGETTA; PICCO; VIGNA, 1998) apresenta três dimensões para o processo de desenvolvimento de Sistemas de Código Móvel: tecnologia, paradigmas de projeto e domínio de aplicações.

Tecnologia diz respeito às linguagens e aos mecanismos que permitem e dão suporte à mobilidade de código. **Paradigmas de projeto** referem-se à arquitetura de aplicação utilizada, podendo ser: cliente-servidor, código por demanda, execução remota e agente móvel. E, **domínios de aplicação** são classes de aplicações que compartilham o

mesmo objetivo geral, por exemplo: recuperação de informações distribuídas, documentos ativos, serviços avançados de telecomunicação, controle e configuração de dispositivos remotos, gerenciamento e cooperação de *wokflows*, redes ativas e comércio eletrônico (PICCO, 2001a).

Neste capítulo serão descritos na Seção 2.1 os mecanimos que dão suporte à mobilidade de código, ou seja, que permitem a movimentação de um código. Na Seção 2.2 serão apresentados alguns exemplos de linguagens e ambientes de programação de código móvel existentes. E a Seção 2.3 enfoca a API μCODE que é uma API utilizada para programar código móvel em Java, oferecendo pacotes que dão suporte à mobilidade de código.

2.1 Mecanismos de Suporte ao Código Móvel

Segundo Fuggeta *et al.* (FUGGETTA; PICCO; VIGNA, 1998),

> "Mobilidade de código pode ser definida como a capacidade de modificar dinamicamente as ligações entre fragmentos de código e o local onde eles são executados".

Esta capacidade, que permite que programas possam trafegar livremente pela rede, depende da tecnologia de código móvel empregada. Tal tecnologia engloba linguagens de programação e seus suportes em tempo de execução.

Sistemas distribuídos tradicionais são apresentados, conforme Fuggeta (FUGGETTA; PICCO; VIGNA, 1998), na Figura 2.1. A camada mais baixa acima da camada de *hardware* é constituída pelo Núcleo do Sistema Operacional (COS[2.2]), que fornece as funcionalidades básicas do sistema operacional. O suporte para serviços de comunicação não-transparentes é fornecido pela camada do Sistema Operacional de Rede (NOS[2.3]). A camada NOS utiliza os serviços fornecidos pela camada COS. A transparência da rede é fornecida pela camada do Sistema Distribuído Verdadeiro (TDS[2.4]).

Um TDS implementa uma plataforma onde componentes localizados em diferentes lugares de uma rede são percebidos como locais. Em sistemas tradicionais, com a utilização de serviços do TDS não se tem noção sobre a estrutura da rede, ou seja, não é

[2.2] *Core Operating System*
[2.3] *Network Operating System*
[2.4] *True Distributed System*

possível saber qual nó está disponibilizando um determinado serviço ou recurso.

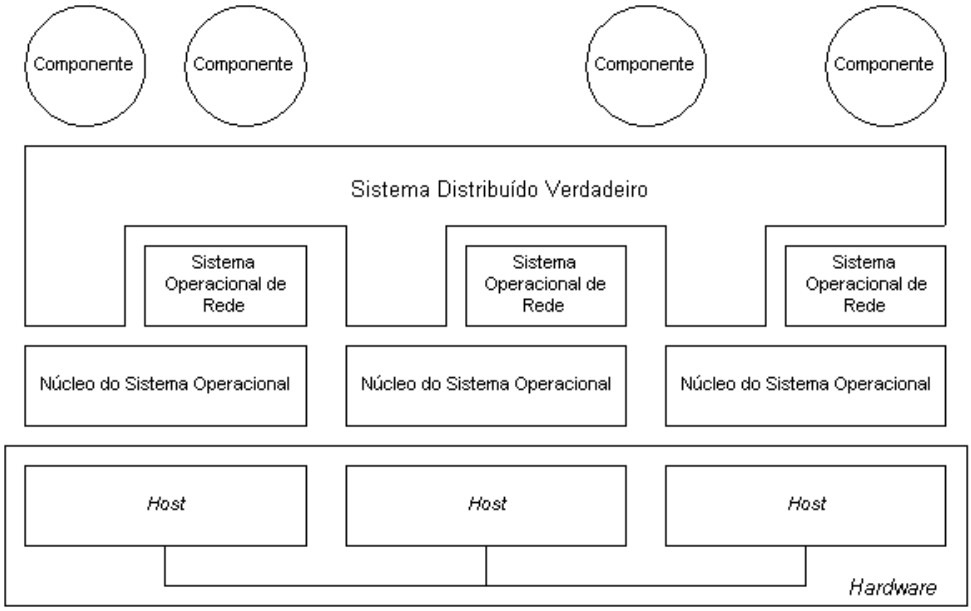

Figura 2.1: Sistemas Distribuídos Tradicionais (FUGGETTA; PICCO; VIGNA, 1998)

As tecnologias de suporte à mobilidade de código, conforme a Figura 2.2, apresentam um outro ponto de vista. A estrutura da rede não está escondida do programador. Na Figura 2.2, o TDS é substituído por Ambientes Computacionais (CE[2.5]) colocados sobre o NOS de cada *host* da rede. Em contraste ao TDS o CE guarda a "identidade" do *host* onde ele está localizado. A função do CE é prover aplicações com a capacidade de relocar dinamicamente seus componentes para diferentes *host*.

Os componentes hospedados pelo CE são denominados Unidades de Execução (EU[2.6]) e Recursos. Unidades de execução, conforme a Figura 2.3, representam fluxos seqüenciais de computação. Recursos representam entidades que podem ser compartilhadas entre múltiplas EUs, tais como um arquivo em um sistema de arquivos. Uma unidade de execução é composta por:

- **Segmento de Código** - fornece a descrição estática para o comportamento de uma computação.
- **Espaço de dados** - é o conjunto de referências para recursos que podem ser acessadas pela EU.

[2.5] *Computational Environments*
[2.6] *Executing Units*

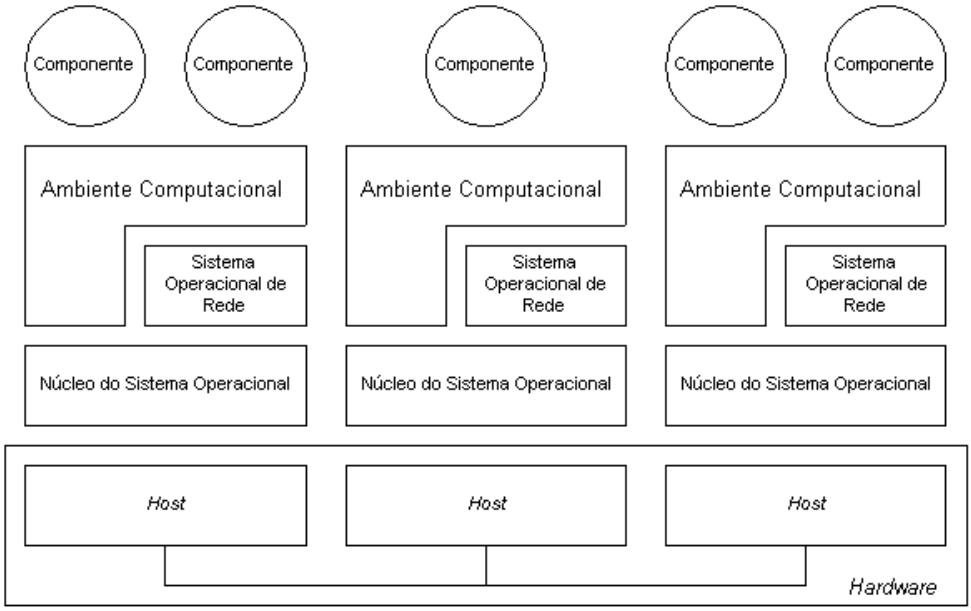

Figura 2.2: Sistemas de Código Móvel (FUGGETTA; PICCO; VIGNA, 1998)

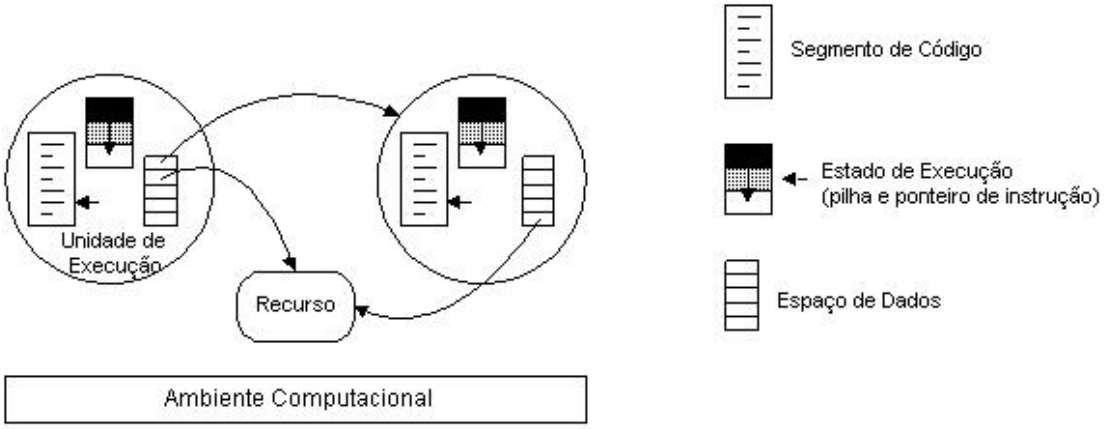

Figura 2.3: Estrutura Interna de uma Unidade de Execução (FUGGETTA; PICCO; VIGNA, 1998)

- **Estado de execução** - contém dados privados que não podem ser compartilhados, assim como as informações de controle relacionadas ao estado da EU.

Em Sistemas de Código Móvel, o segmento de código, o estado de execução e o espaço de dados de uma EU podem ser transferidos em diferentes CEs. Acrescentando, cada um desses constituintes da EU podem se mover independentemente pelos *hosts*. Fuggeta (FUGGETTA; PICCO; VIGNA, 1998) apresenta vários mecanismos de mobilidade para gerenciar código e estado e para gerenciar espaço de dados, conforme ilustrado na Figura 2.4.

Os sistemas de código móvel existentes oferecem duas formas de mobilidade, caracterizadas pelos componentes de unidade de execução que podem ser migrados:

- **Mobilidade Forte** - é a capacidade de um sistema de código móvel permitir a migração de ambos o código e o estado de execução de uma unidade de execução para um ambiente computacional diferente.
- **Mobilidade Fraca** - é a capacidade de um sistema de código móvel permitir a transferência somente do código através de diferentes ambientes computacionais. O código pode ser acompanhado por algumas informações de inicialização, mas não envolve a migração do estado de execução.

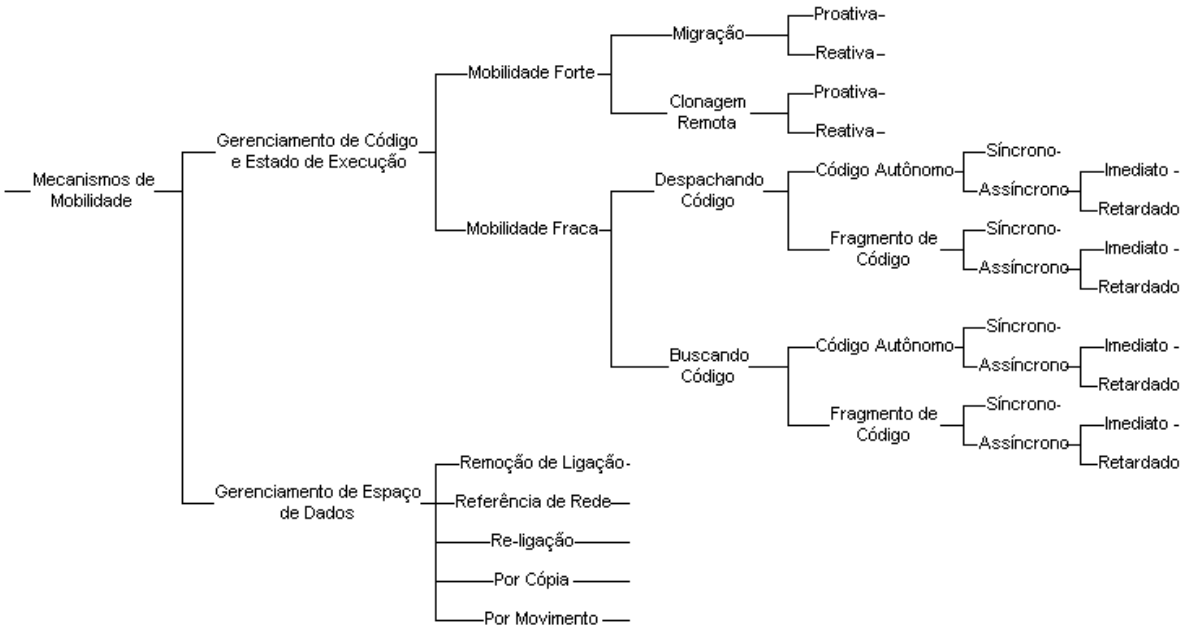

Figura 2.4: Classificação dos Mecanismos de Mobilidade (FUGGETTA; PICCO; VIGNA, 1998)

Detalhando, a Figura 2.4 apresenta que, a mobilidade forte é apoiada por dois mecanismos: migração e clonagem remota. O mecanismo de migração suspende uma unidade de execução, transmite-a para o ambiente computacional de destino, e então prossegue. A migração pode ser proativa ou reativa. Na migração proativa, o tempo e o destino para migração são determinados de forma autônoma pela unidade de execução migradora. Na migração reativa, o movimento é favorecido por uma unidade de execução diferente que se relaciona com a unidade de execução a ser migrada.

O mecanismo de clonagem remota cria uma cópia de uma unidade de execução em um ambiente computacional remoto. A clonagem remota difere-se do mecanismo de migração por que a unidade de execução original não é separada de seu ambiente computacional corrente. Como na migração, a clonagem remota pode ser tanto proativa como reativa, conforme mostra a ilustração da Figura 2.4.

Mecanismos que dão suporte à mobilidade fraca podem ser classificados de acordo com a direção de transferência e a natureza do código, com a sincronização e o momento em que o código será executado no local de destino.

Uma EU pode tanto buscar o código para ser dinamicamente ligado e/ou executado, ou despachar o código para um outro CE. A Figura 2.4 mostra que o código pode ser migrado tanto como código autônomo, quanto como fragmento de código. Código autônomo é completo e é usado para instanciar uma nova EU no lugar de destino. Reciprocamente, um fragmento de código deve ser ligado no contexto do código já em andamento e finalmente executado. Estes mecanismos podem ser síncronos ou assíncronos. Em mecanismos assíncronos, a execução do código transferido pode acontecer de modo imediato (o código é executado assim que é recebido) ou retardado (o código é executado somente quando uma determinada condição é satisfeita).

Para conseguir realizar a migração de uma EU para um novo CE, deve haver um gerenciamento do espaço de dados, para tanto devem ser avaliados os recursos envolvidos na migração.

Fuggeta, em (FUGGETTA; PICCO; VIGNA, 1998), modela os recursos da seguinte forma: $Recurso = \{I, V, T\}$, onde I é um identificador único, V é o valor do recurso, e T é seu tipo, que determina a estrutura da informação contida no recurso, bem como sua interface. O tipo do recurso determina se é transferível ou não-transferível, ou seja, se ele pode ser migrado além da rede ou não. Recurso transferível pode ser livre ou fixo. Recursos livres podem ser ligadas a outro CE, enquanto recursos fixos são associados permanentemente ao CE.

Conforme apresentado na Tabela 2.1, os recursos podem ser ligados a uma unidade de execução por meio de três formas:

- **Ligação por identificador** - é a forma mais forte de ligação, é quando uma unidade de execução deseja ser ligada a um recurso que não pode ser substituído por outro

equivalente.

- **Ligação por valor** - é quando uma unidade de execução está interessada nos conteúdos de um recurso e quer ser capaz de acessá-los localmente, o recurso migrado deve ter o mesmo tipo e valor daquele presente no ambiente computacional de origem.

- **Ligação por tipo** - a forma mais fraca de ligação é por tipo, neste caso, a EU requer que o recurso seja de um determinado tipo, nao importando qual o seu valor ou identidade. Esta ligação é usada para ligar recursos que estejam disponíveis sobre o ambiente computacional, como variaveis de sistemas, bibliotecas ou dispositivos de rede.

Tabela 2.1: Mecanismos de Gerenciamento de Espaço de Dados (FUGGETTA; PICCO; VIGNA, 1998)

	Transferência Livre	**Transferência Fixa**	**Transferência Não Fixa**
Por Identificador	Por Movimento (Referência de Rede)	Referência de Rede	Referência de Rede
Por Valor	Por Cópia (Por Movimento, Referência de Rede)	Por Cópia (Referência de Rede)	Referência de Rede
Por Tipo	Re-ligação (Referência de Rede, Por Cópia, Por Movimento)	Re-ligação (Referência de Rede, Por Cópia)	Re-ligação (Referência de Rede)

A migração de uma unidade de execução gera duas classes de problemas: a relocação de recursos e a reconfiguração de ligação. Fuggeta em (FUGGETTA; PICCO; VIGNA, 1998), conforme Tabela 2.1, apresenta mecanismos de gerenciamento de espaço de dados para lidar com esses problemas.

Os mecanismos de gerenciamento de espaço de dados, ilustrados na Figura 2.5, são descritos a seguir[2.7]:

[2.7]Vale esclarecer que, na Figura 2.5, o círculo sombreado é a unidade de execução que se movimenta do *host* de origem para o de destino.

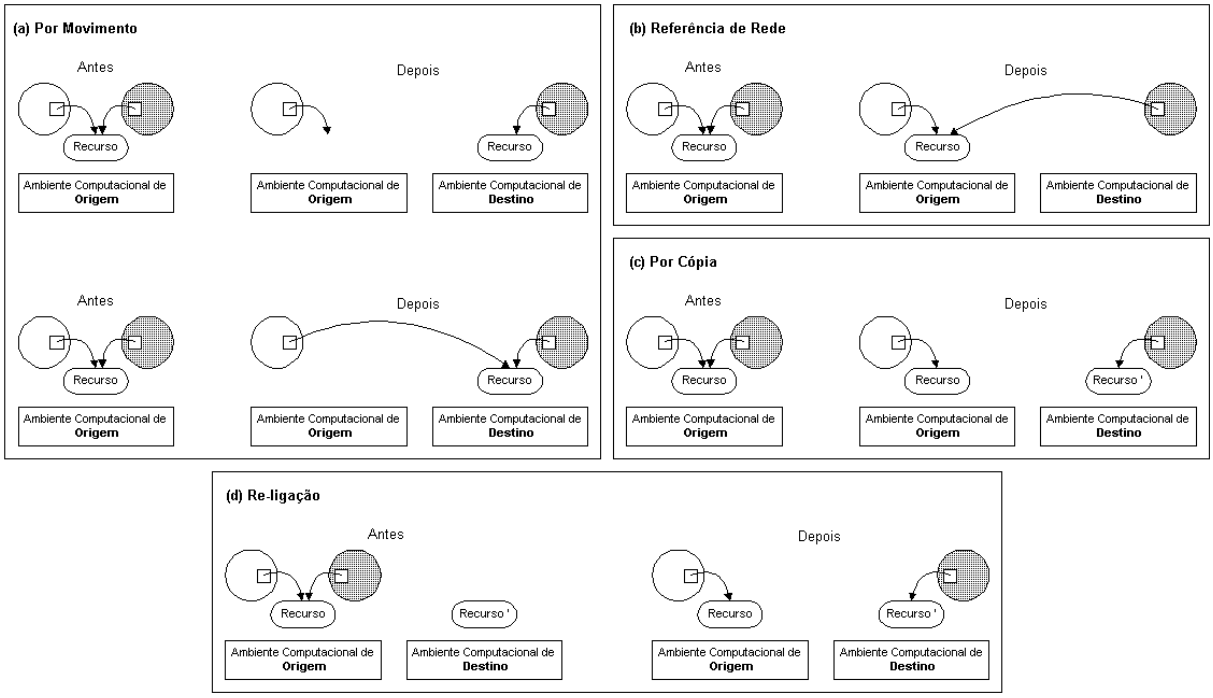

Figura 2.5: Mecanismos de Gerencimanto de Espaço de Dados (FUGGETTA; PICCO; VIGNA, 1998)

- **Remoção de ligação** - quando uma unidade de execução migra para outro ambiente computacional, então, a ligação é descartada (Figura 5a superior).

- **Relocação por deslocamento** - quando uma unidade de execução está ligada a um recurso livre transferível por identificador, então, o recurso é transferido juntamente com a unidade de execução para outro ambiente computacional, a ligação não muda (Figura 2.5a inferior).

- **Referência de Rede** - quando uma unidade de execução está ligada a um recurso não-transferível por identificador, então, o recurso não é transferido, e após a migração da unidade de execução para outro ambiente computacional, a ligação passa a referenciar o recurso no ambiente computacional de origem (Figura 2.5b).

- **Por Cópia** - quando a ligação é por valor e o recurso é transferível, então, uma cópia do recurso original é criada e a ligação passa a referenciar a cópia do recurso, depois, esta cópia é transferida juntamente com a unidade de execução para o ambiente computacional de destino (Figura 2.5c).

- **Re-ligação** - quando uma unidade de execução está ligada a um recurso por tipo, então, depois da migração da unidade de execução a ligação passa a referenciar outro recurso no ambiente computacional de destino. Para tanto, o recurso do

ambiente computacional de destino deve ser do mesmo tipo do recurso do ambiente computacional de origem (Figura 2.5d).

Na seção seguinte serão apresentados alguns exemplos de linguagens e ambientes existentes para a implementação de um código móvel.

2.2 Linguagens e Ambientes para Mobilidade

Na geração de Sistemas de Código Móvel é importante contar com uma linguagem de programação que dê suporte para o desenvolvimento do código. A seguir serão apresentadas algumas linguagens e ambientes de programação que dão suporte a mobilidade de código.

***Agent*-Tcl:** desenvolvida pela Universidade de Darthmouth, provê um interpretador Tcl[2.8] estendido com suporte à mobilidade forte. Unidades de execução executam em espaços de endereços separados. Elas podem compartilhar somente recursos fornecidos pelo sistema operacional básico. A abstração do ambiente computacional é implementada pelo sistema operacional e pela linguagem de suporte em tempo de execução (FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

Ara: desenvolvida pela Universidade de Kaiserslautern, é uma multi-linguagem de sistemas de código móvel que suporta mobilidade forte. As unidades de execução, chamada de agentes, são gerenciadas por uma linguagem de interpretadores extras de núcleo de sistemas independentes em vez de linguagens de suporte - em tempo de escrita C, C++ e Tcl. O núcleo e os interpretadores constituem o ambiente computacional, cujos serviços são acessíveis para agentes sem interrupção à abstração local (FUGGETTA; PICCO; VIGNA, 1998).

***Facile*:** desenvolvida pela ECRC[2.9] em Münich, é uma linguagem funcional que estende a linguagem *Standard ML* com primitivas para distribuição, concorrência e comunicação. Possui suporte à mobilidade fraca. As unidades de execução são implementadas como *threads* que executam em ambientes computacionais *Facile*, chamados de nós (FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

[2.8] *Tool Command Language.*
[2.9] *The European Computer-Industry Research Center*

Java: desenvolvida pela *Sun Microsystems*, tem despertado mais atenção e expectativas em mobilidade de código, devido à facilidade de programação e à portabilidade. Ao compilar um arquivo denominado "Movel.java" é gerado um arquivo "Movel.class" que pode ser transferido, através de protocolos, de um cliente para um servidor. A classe compilada está em formato de *bytecode*, que é interpretado pela Máquina Virtual Java (JVM[2.10]), portanto, basta a plataforma possuir a JVM para portar as classes Java sem a necessidade de alteração do código. Java suporta mobilidade fraca usando mecanismos para buscar fragmentos de código (FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

Java Aglets: a API *Java Aglets*, desenvolvida pelo Laboratório de Pesquisa de Tóquio da IBM[2.11] no Japão, estende Java com suporte à mobilidade fraca. *Aglets* (unidades de execução) são *threads* em um interpretador Java que constituem o ambiente computacional (FUGGETTA; PICCO; VIGNA, 1998).

M0: implementada pela Universidade de Geneva, é uma linguagem interpretada baseada em pilha que implementa o conceito de mensageiros. Mensageiros (unidades de execução) são seqüências de instruções que são transmitidas entre plataformas (ambientes computacionais) e executadas incondicionalmente conforme recebimento (FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

Mole: desenvolvida pela Universidade de Stuttgart, é uma API Java que suporta mobilidade fraca. Agentes Mole são objetos Java que executam como *threads* da JVM, que são abstraídos dentro de um lugar, o ambiente computacional Mole (FUGGETTA; PICCO; VIGNA, 1998).

Obliq: desenvolvida pela DEC[2.12], é uma linguagem interpretada sem tipo, baseada em objeto, estendida lexicamente. Uma *thread* (unidade de execução), pode requisitar a execução de um procedimento em uma máquina de execução remota. O código para tal procedimento é enviado para a máquina de destino e executado lá por uma nova unidade de execução (FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

***Safe*-Tcl:** desenvolvida pelos autores do padrão Internet MIME[2.13], é uma extensão da Tcl concebida para suporte de *e-mail* ativo. Em um *e-mail* ativo, as mensagens podem incluir código para ser executado quando o receptor recebe ou lê as mensagens

[2.10] *Java Virtual Machine*
[2.11] International Business Machines Corporation
[2.12] *Digital Equipament Corporation*
[2.13] *Multipurpose Internet Mail Extensions*

(FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

Sumatra: desenvolvida pela Universidade de Maryland, é uma extensão Java projetada expressamente para suportar a implementação de programas móveis *resource-aware*. Provê suporte à mobilidade forte (FUGGETTA; PICCO; VIGNA, 1998).

TACOMA (*Tromsø And Cornell Mobile Agents*): a linguagem Tcl é estendida para incluir primitivas que suportam mobilidade fraca. Unidades de execução, chamadas de agentes, são implementadas como processos Unix executando o interpretador Tcl (FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

Telescript: desenvolvida pela *General Magic*, é uma linguagem orientada a objetos concebida para o desenvolvimento de grandes aplicações distribuídas. A segurança tem sido um dos principais fatores de condução do projeto da linguagem e simultaneamente, enfoca a mobilidade forte (FUGGETTA; PICCO; VIGNA, 1998), (PICCO et al., 1997).

Voyager: é uma plataforma de suporte à distribuição e à mobilidade de código implementada pela *ObjectSpace Inc.* Desenvolvida em Java, suporta comunicação distribuída, resolução de referências, ativação remota, transparência de localização e mobilidade de componentes (DUARTE; DOTTI, 2001).

μCode: é uma pequena API, desenvolvida em linguagem Java que tem por objetivo oferecer um conjunto mínimo de primitivas para suportar a mobilidade de código e estado.(PICCO, 1998), (PICCO, 2001b).

Na próxima seção será melhor descrita a API μCode, que foi utilizada para implementar os estudos de casos apresentados neste trabalho, procurando expor as características da API e a função dos pacotes, das classes e das interfaces existentes que a compõem.

2.3 A API μCode

A API μCode (PICCO, 1998) oferece primitivas para suportar a mobilidade de código e estado, não é apenas um Sistema de Agentes Móveis, pois permite que o usuário defina suas próprias primitivas de mobilidade, incluindo sua própria noção de agente móvel. Ou, o usuário poderá continuar a usar as abstrações de agentes móveis oferecidas por um dos pacotes que compõe a distribuição μCode.

Esta API enfatiza a mobilidade de código (fina granularidade)[2.14] ao invés dos agentes móveis, trabalha com Java 1.1 (ou superior), requer pouco espaço, execução extremamente "leve", possui projeto aberto (modularidade, flexibilidade, composabilidade) e suporte a qualquer estratégia de realocação de classes(PICCO, 2001b).

A distribuição da μCODE é composta por três pacotes, conforme (PICCO, 2001b):

- ***mucode*** **-** contém o núcleo do sistema, provê um conjunto mínimo de primitivas para construir qualquer operação de mobilidade de alto-nível.
- ***mucode.util*** **-** contém utilitários para iniciar um servidor μCODE e mecanismos que complementam a capacidade de reflexão da linguagem Java.
- ***mucode.abstractions*** **-** contém abstrações de alto nível, incluindo as primitivas para relocação de código e estado de várias maneiras, bem como uma implementação de um agente móvel.

O pacote *mucode* é o núcleo do μCODE, sendo que abstrações de alto nível podem ser construídas com base nas seguintes classes:

- ***Group*** **-** é a única unidade de mobilidade. Oferece um *container* ao programador que pode ser preenchido arbitrariamente com classes e objetos (incluindo objetos *Thread*). Classes e objetos não necessitam pertencer à mesma unidade de execução.
- ***MuServer*** **-** é o destino de um *Group*, é a abstração do suporte de *run-time* do μCODE. Provê o mecanismo básico para relocar código e estado.
- ***GroupHandler*** **-** é o responsável pela extração do *mix* de classes e objetos em um grupo e pelo uso coerente destes, possibilitando gerar novas unidades de execução. Objeto criado pelo programador, que é instanciado, e cujas operações são automaticamente invocadas no destino.
- ***ClassSpace*** **-** durante a reconstrução do *Group*, o sistema necessita realocar as classes e torná-las disponíveis para a subseqüente execução do *Group Handler*. Classes extraídas de um *Group* precisam ser alocadas em algum *name space*, de modo a evitar conflitos com classes reconstruídas de outros *Groups*.

[2.14] Mobilidade de uma única classe ou objeto.

Outras abstrações são oferecidas pelo pacote *mucode.abstractions*, que é composto pelas seguintes classes: *Relocator*, *RelocatorHandler* e *MuAgent*.

A classe *Relocator* provê os seguintes métodos que permitem a relocação de classes e *threads*:

- ***copyThread*** - permite copiar uma *Thread* de um servidor μServer para outro.
- ***spawnThread*** - permite criar uma outra *Thread* em outro servidor μServer, usando um conjunto de classes especificado pelo usuário.
- ***shipClasses*** - permite despachar um conjunto de classes para um *class space* compartilhado de outro μServer.
- ***fetchClasses*** - permite buscar um conjunto de classes de um *class space* compartilhado de outro μServer.

As operações da classe *Relocator* permitem implementar arquiteturas que exploram mobilidade, mas não necessariamente um agente móvel[2.15]. A classe *MuAgent*, provida pelo pacote *abstractions*, permite a implementação do paradigma de agentes móveis.

O pacote *mucode.util* (PICCO, 1998) é composto pela interface *ClosureConstants*, pelas classes *ClassInspector* e *Launcher*, detalhadas a seguir:

- ***ClosureConstants*** - são contantes que identificam diferentes tipos de classes *closure*.
- ***ClassInspector*** - é utilizada para recuperar informações sobre as classes que são declaradas e referenciadas por uma determinada classe, permitindo, assim, determinar sua classe *closure*.
- ***Launcher*** - esta classe simplifica a tarefa de passagem de parâmetros de linha de comando de uma aplicação para um μServer embutida neste.

Para finalizar esta sessão são apresentadas as hierarquias de pacote, classes e interfaces que compõem a API μCode (PICCO, 2001b), ilustradas conforme as Figuras 2.6, 2.7 e 2.8 e modeladas através da UML[2.16] (BOOCH; RUMBAUGH; JACOBSON, 2000).

[2.15] Um agente móvel representa uma unidade de mobilidade que corresponde a unidade de execução, e que é capaz de determinar autonomamente sua própria migração (PICCO, 2001b).
[2.16] *Unified Modeling Language*

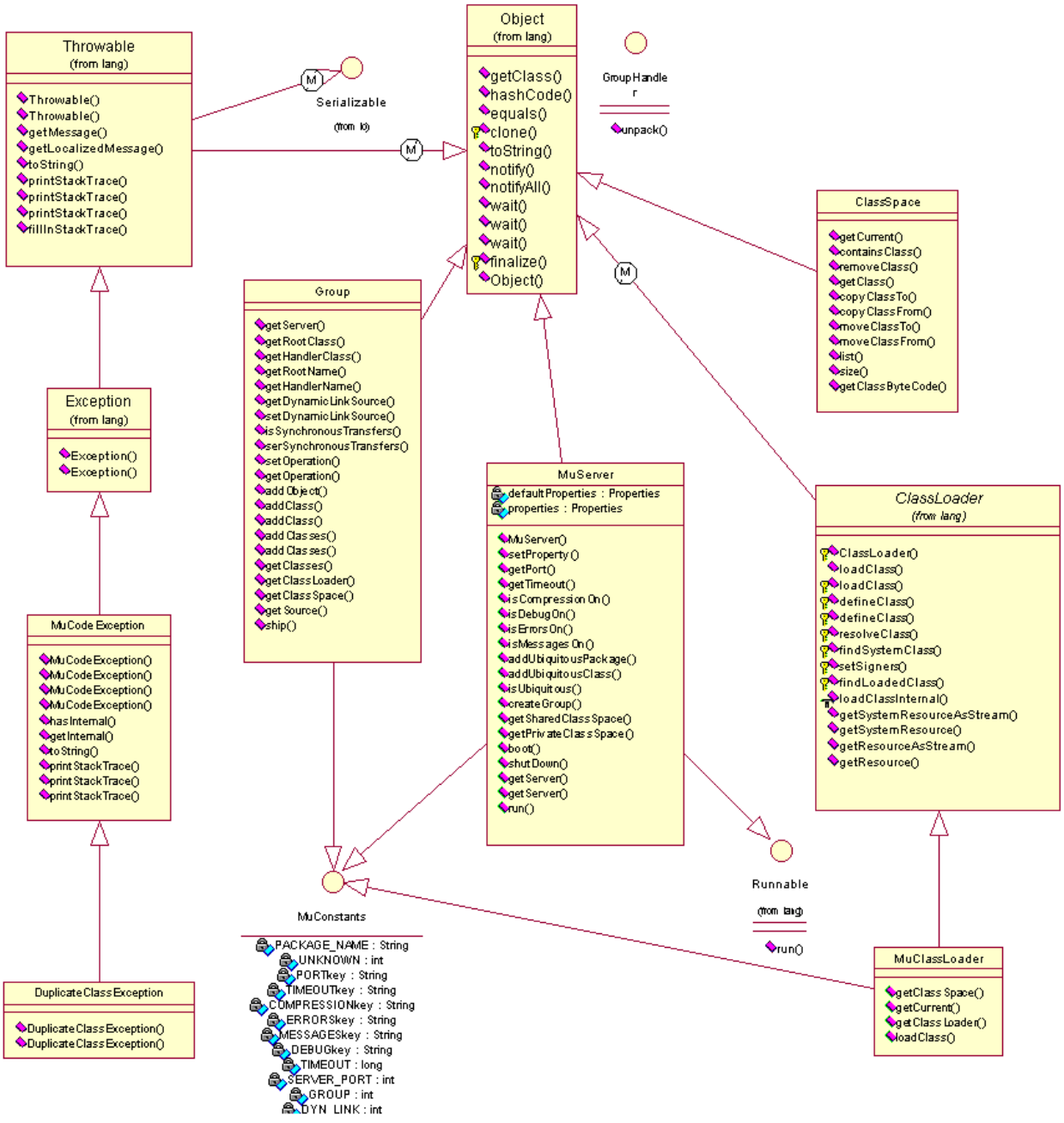

Figura 2.6: Hierarquia do Pacote *mucode*

No próximo capítulo serão apresentados os conceitos básicos sobre o teste estrutural, já que a intenção deste trabalho é apresentar a utilização do teste estrutural em código móvel, desenvolvido pela linguagem Java, suportando a mobilidade de código oferecida pela API μCode.

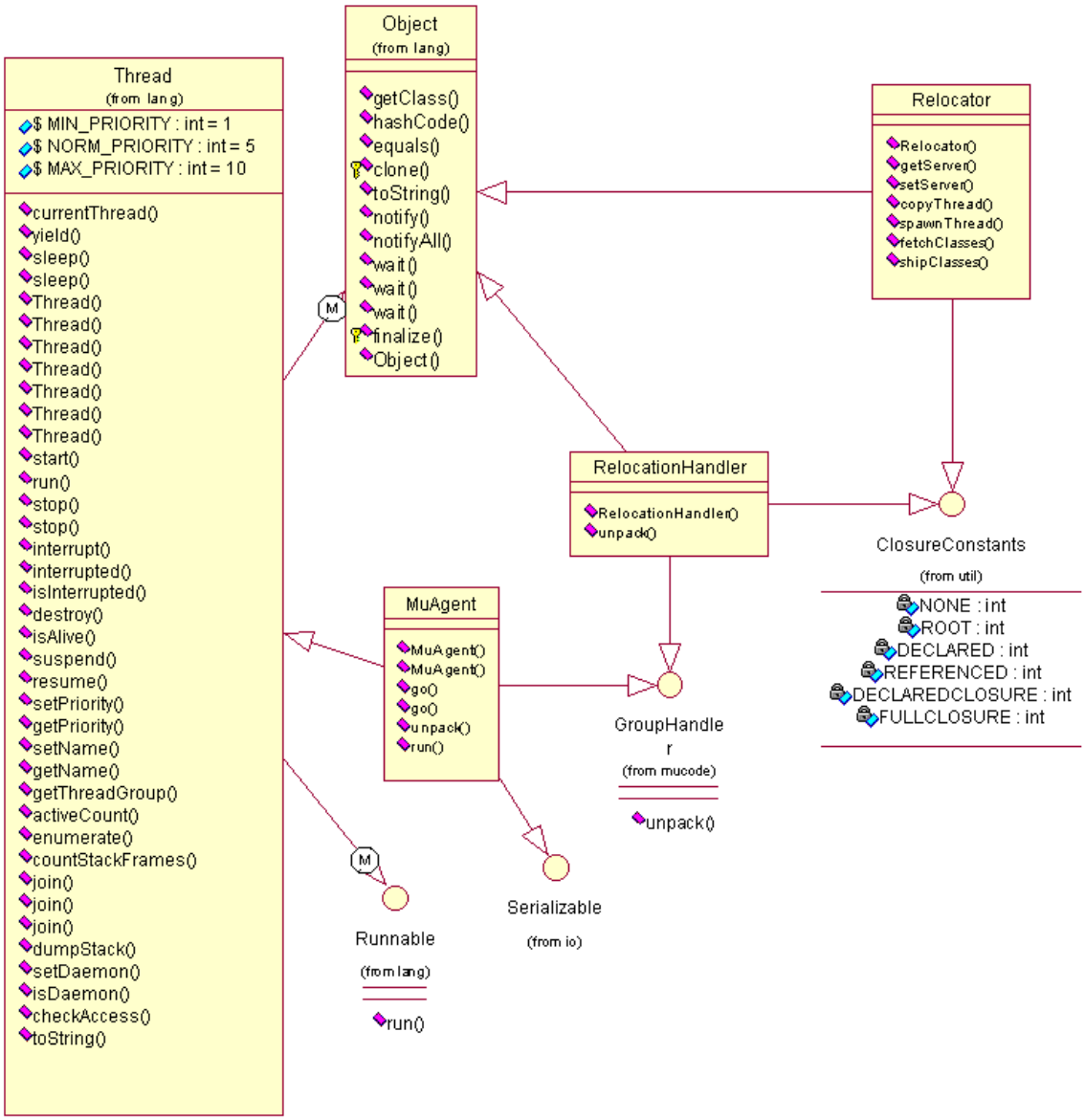

Figura 2.7: Hierarquia do Pacote *mucode.abstractions*

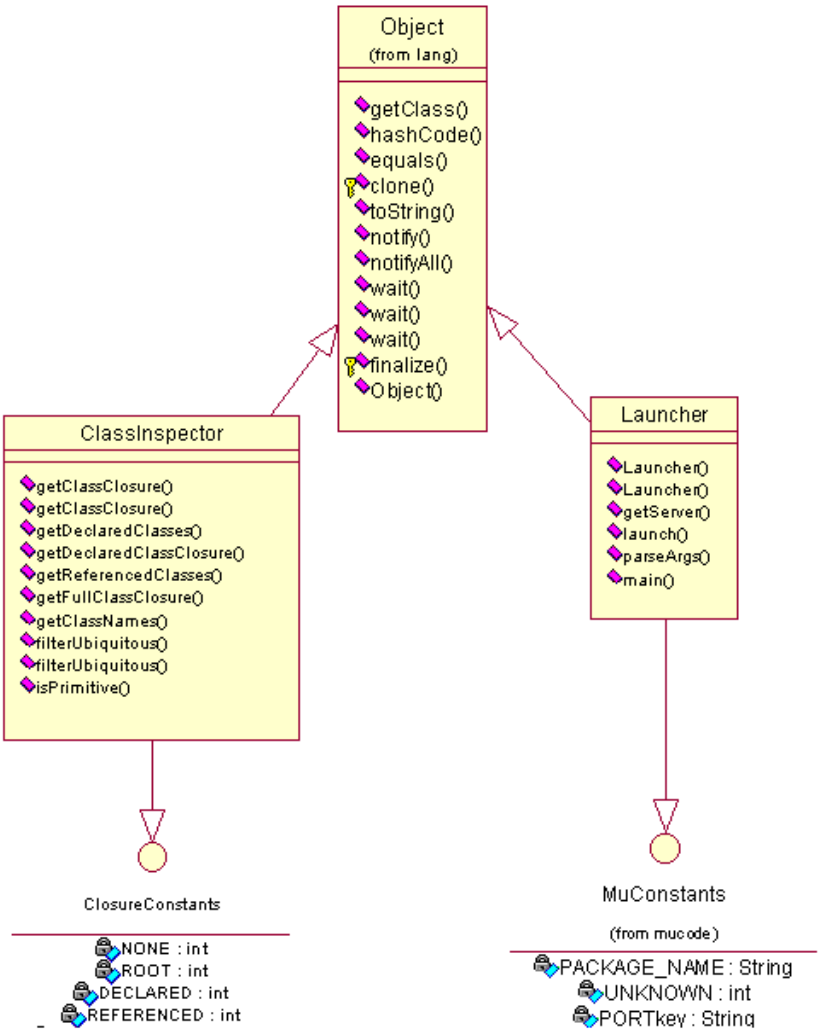

Figura 2.8: Hierarquia do Pacote *mucode.util*

3 Teste Estrutural

Sistemas de *software* têm desempenhado um papel cada vez mais importante na sociedade e, em muitas situações, o seu funcionamento correto é imprescindível. Várias aplicações críticas necessitam de uma execução extremamente segura, correta e confiável, ou seja, devem cumprir totalmente seus requisitos. Entretanto, Carvalho (CARVALHO; CHIOSI, 2001) diz que o desenvolvimento de tais sistemas é complexo, pois deve lidar com requisitos rígidos, restrições de integridade e um amplo conhecimento sobre a aplicação para que o produto final seja satisfatório.

Segundo Carvalho (CARVALHO; CHIOSI, 2001) a engenharia de *software* vem criando, durante anos, formas de melhorar o processo de desenvolvimento de *software*, visto que, é crescente o surgimento de novos métodos e técnicas para ajudar tal processo. Todavia, apesar dos esforços referentes ao processo de desenvolvimento de *software*, este não deixa de ter sua complexidade dificultando sua criação. As atividades de teste de *software* vêm dar apoio ao desenvolvimento deste, objetivando garantir a sua qualidade. Os custos envolvidos às falhas de software justificam a execução de uma atividade de teste bem planejada.

Igualmente aos sistemas de *software* comuns, os sistemas de código móvel também necessitam de um funcionamento correto, seguro e confiável. E isto pode ser verificado através da aplicação de técnicas de teste de *software*. Neste trabalho, em particular, foi utilizada a técnica estrutural para validar a corretitude de um código móvel.

O teste de *software*, segundo Pressman (PRESSMAN, 1995), envolve o planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados dos testes. O projeto de casos de teste possui um conjunto de técnicas, critérios e métodos para a elaboração dos casos de testes, ajudando, assim, a garantir o teste por completo e o aumento da probabilidade de detecção de defeitos no *software*. Apesar de ser uma atividade cara é uma das melhores formas de remover os defeitos do *software*.

A maior parte dos defeitos contidos em um *software* é de origem humana, são gerados durante a comunicação e transformação de informações. Tais defeitos podem ser encontrados em *softwares* disponíveis no mercado, sendo que sua maioria encontra-se em partes do código que são executadas raramente. Quanto mais cedo for detectado os defeitos de um *software*, menor será o custo da sua correção e maior será a probabilidade de corrigí-los corretamente (PRESSMAN, 1995).

Conforme Myers (MYERS, 1979), o objetivo da atividade de teste é executar um programa com a intenção de descobrir um erro. Assim, um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto, e um teste bem-sucedido é aquele que revela um erro ainda não descoberto.

Portanto, o objetivo da atividade de teste é projetar testes que descubram erros e façam-na com tempo e esforço mínimos. É importante salientar que a atividade de teste não mostra a ausência de defeitos, ela só pode mostrar que defeitos de *software* estão presentes (PRESSMAN, 1995).

Para auxiliar a atividade de teste foram propostos vários métodos que oferecem mecanismos que ajudam a garantir a integridade do teste e proporcionam uma maior probabilidade de revelar erros. Para tanto, existem as seguintes técnicas que auxiliam a atividade de teste do *software*:

- **Técnica Funcional** - também conhecida como teste de caixa preta, concentra-se nos requisitos funcionais do *software*. É usada para demonstrar que as funções são operacionais, a entrada é aceita e a saída é corretamente produzida, e que a integridade das informações externas é mantida (MYERS, 1979).
- **Técnica Estrutural** - também conhecida como teste de caixa branca, baseia-se no exame dos detalhes procedimentais. Todas as linhas de código são testadas, através de casos de teste que põem à prova condições e/ou laços de repetição (MYERS, 1979), (MALDONADO, 1991).
- **Técnica Baseada em Defeitos** - se baseia nos erros mais freqüentes, cometidos durante o processo de desenvolvimento de *software* (DEMILLO; LIPTON; SAYWARD, 1978).

Neste trabalho será apresentado um estudo referente a técnica estrutural ou caixa branca que se baseia na estrutura interna do *software*. O teste de caixa branca é um

método de projeto de casos de teste que usa a estrutura de controle e de dados do projeto procedimental para derivar casos de teste.

Utilizando o teste estrutural podem ser gerados casos de teste que garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez; executem todas as decisões lógicas para valores falsos ou verdadeiros; executem todos os laços em suas fronteiras e dentro de seus limites operacionais; e que exercitem os fluxos de dados internos para garantir a sua validade (RAPPS; WEYUKER, 1985).

A seguir são apresentados alguns exemplos de critérios do teste estrutural:

- **Critérios Baseados em Fluxo de Controle** - utilizam informações de fluxo de controle do programa como base para a seleção de dados de teste, de tal forma que determinados tipos de estrutura do grafo do programa sejam exercitadas (RAPPS; WEYUKER, 1985).
- **Critérios Baseados em Fluxo de Dados** - utilizam informações do fluxo dos dados existentes no programa, visando identificar atribuições e utilizações das variáveis através do programa, gerando componentes elementares a serem exercitados pelo testador (RAPPS; WEYUKER, 1985), (MALDONADO, 1991).
- **Critérios Baseados em Complexidade** - utilizam informações da representação do fluxo de controle lógico de um programa, fornece uma medida quantitativa da dificuldade do teste e uma indicação da confiabilidade final, este critério foi proposto por McCabe (MCCABE; BUTLER, 1989).

Os critérios estruturais geralmente utilizam o Grafo de Fluxo de Controle (GFC) para representar um programa. O grafo de fluxo de controle descreve o fluxo de controle lógico de um programa. Para cada construção de uma estrutura de dados tem-se um símbolo de grafo correspondente, conforme Pressman (PRESSMAN, 1995), ilustrado na Figura 3.1. Isto permite que qualquer representação de um projeto procedimental possa ser traduzida num grafo de fluxo.

Segundo Weyuker (RAPPS; WEYUKER, 1985), um grafo de programa G representando um programa Q consiste de um nó i correspondente para cada bloco b_i de Q e uma aresta do nó j para o nó k, denotada (j, k), se o último comando de b_j não é um desvio

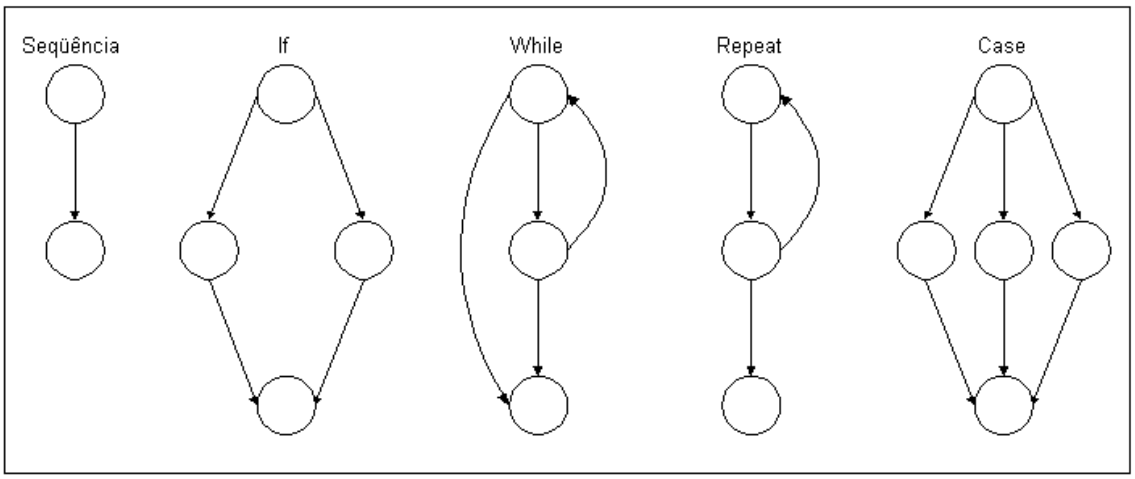

Figura 3.1: Notação de Grafo de Fluxo de Controle (PRESSMAN, 1995)

incondicional e ele precede fisicamente o primeiro comando de b_k, ou se o último comando de b_j é um desvio cujo alvo é o primeiro comando de b_k.

Cada círculo do GFC, denominado **nó**, representa uma ou mais instruções de procedimentos, ou seja, um conjunto de comandos ordenados, também denominado de **bloco**. Conforme Weyuker (RAPPS; WEYUKER, 1985), um programa pode ser decomposto em uma série de blocos disjuntos. Quando o primeiro comando de um bloco é executado, os comandos seguintes são executados em ordem. Complementando, o primeiro comando de um bloco é o único que pode ser executado após outro bloco.

Conforme Weyuker (RAPPS; WEYUKER, 1985), o **arco**, denominado ramo ou aresta, representa um fluxo de controle, ou seja, é a transferência de controle entre blocos. Um ramo deve terminar em um nó. Cada nó que contém uma condição é denominado de **nó-predicativo** e possui dois ou mais ramos que saem dele. Portanto, o último comando do nó-predicativo é um desvio condicional.

Para representar um programa o grafo possuirá um único nó de entrada ou inicial e poderá apresentar atualmente, devido às novas linguagens existentes, mais de um nó de saída ou final (CHAIM, 1991).

Rapps e Weyuker, em (RAPPS; WEYUKER, 1985), definem que um grafo é composto por vários caminhos, um **caminho** é uma seqüência finita de nós. Um **caminho simples** é aquele em que todos os nós são distintos, exceto o primeiro e o último. Um **caminho livre-de-laço** é aquele em que todos os nós são distintos. Um **caminho** é considerado **completo** quando o primeiro nó é o nó inicial e o último é um nó final. Vale

ressaltar que nem todo caminho completo é executável e é indecidível em geral verificá-lo.

Um **caminho completo** é considerado **executável** se existir alguma associação de valores às variáveis de entrada que cause a execução do caminho.

Um **caminho** é definido como **executável** se ele é um subcaminho de um determinado caminho completo executável. A executabilidade de um caminho depende da semântica e não somente da sintaxe do programa.

Neste capítulo serão descritos na Seção 3.1 os critérios baseados em fluxo de controle e um código de exemplo para mostrar como satisfazer tais critérios de teste. Na Seção 3.2 são descritos os critérios baseados em fluxo de dados e também apresenta um exemplo mostrando como satisfazer tais critérios. A Seção 3.3 apresenta alguns exemplos de ferramentas que apoiam o teste estrutural.

3.1 Critérios Baseados em Fluxo de Controle

Os critérios baseados em fluxo de controle foram os primeiros a surgirem, como exemplos temos os critérios, conforme Weyuker (RAPPS; WEYUKER, 1985):

- **Todos-Nós** - requer que todos os nós (blocos de comandos seqüenciais de um programa) sejam executados pelo menos uma vez.
- **Todos-Arcos** - requer que todos os arcos (comandos de transferência entre blocos) sejam executados.
- **Todos-Caminhos** - requer que todos os caminhos de um programa sejam executados.

O critério de Todos-Caminhos é, em geral, impossível de ser aplicado devido poder gerar um número infinito de requisitos de teste. E apesar dos critérios Todos-Nós e Todos-Arcos serem viáveis, sua utilização torna-se pouco eficaz na medida em que a maioria dos defeitos pode não ser descoberta. Para complementar tais critérios, foram propostos os critérios de fluxo de dados que têm por objetivo verificar a atribuição de valores às variáveis e o uso destes valores.

Para exemplificar os critérios baseados em fluxo de controle é apresentado um código, Figura 3.2, baseado em (RAPPS; WEYUKER, 1985). Com base neste código será gerado um grafo de fluxo de controle, conforme a ilustração da Figura 3.3. Com base na Figura 3.3, os seguintes caminhos satisfariam os critérios descritos anteriormente:

Todos-Nós:

```
(1, 2, 4, 5, 6, 5, 7, 8, 9)        (1, 3, 4, 5, 7, 8, 9)
```

Todos-Caminhos - com restrição de seleção em relação ao laço:

```
(1, 2, 4, 5, 6, 5, 7, 8, 9)        (1, 2, 4, 5, 7, 9)              (1, 3, 4, 5, 7, 8, 9)
(1, 2, 4, 5, 6, 5, 7, 9)           (1, 3, 4, 5, 6, 5, 7, 8, 9)     (1, 3, 4, 5, 7, 9)
(1, 2, 4, 5, 7, 8, 9)              (1, 3, 4, 5, 6, 5, 7, 9)
```

Todos-Arcos:

```
(1, 2, 4, 5, 6, 5, 7, 8, 9)        (1, 3, 4, 5, 7, 9)
```

```
1      START
1      READ X,Y
1      IF Y<0 THEN GOTO A
2      Pow <- Y
       GOTO B
3  [A] Pow <- -Y
4  [B] Z <- 1
5  [C] IF Pow = 0 THEN GOTO D
6      Z <- Z*X
6      Pow <- Pow-1
       GOTO C
7  [D] IF Y >= 0 THEN GOTO E
8      Z <- 1/Z
9  [E] ANSWER <- Z+1
9      PRINT ANSWER
9      STOP
```

Figura 3.2: Exemplo de Código

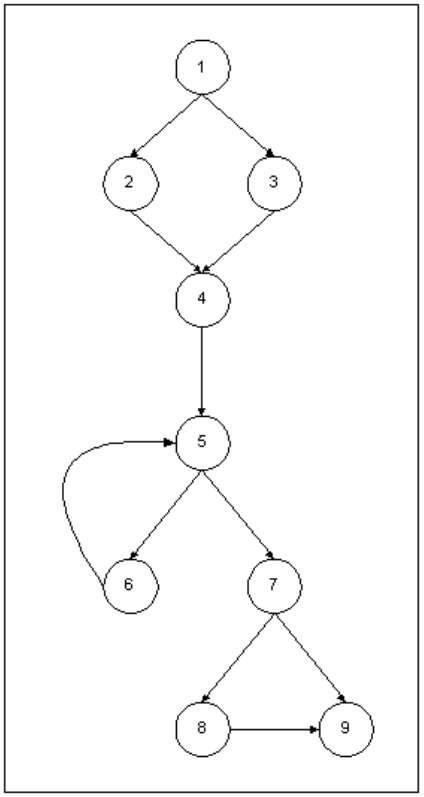

Figura 3.3: Grafo de Fluxo de Controle para o Exemplo

3.2 Critérios Baseados em Fluxo de Dados

Os critérios baseados em fluxo de dados fornecem uma hierarquia de critérios que são mais exigentes do que o critério Todos-Arcos e menos custosos que o critério Todos-Caminhos.

Conforme Weyuker (RAPPS; WEYUKER, 1985), os critérios baseados em fluxo de dados se apóiam na idéia de que não se pode acreditar na corretude de uma computação se o resultado desta computação nunca foi utilizado.

Para tanto, na análise de fluxo de dados existem três tipos de ocorrências de variáveis:

- **Definição** - ocorre quando um valor de uma variável é armazenado em uma posição de memória associada a essa variável. Isto ocorre quando a variável está no lado esquerdo de um comando de atribuição; em um comando de entrada; ou em chamadas de procedimentos como parâmetro de saída, por exemplo.

  def-global - uma definição de uma variável x do nó i é global se ela é a última definição de x em i e se existe um **caminho livre de definição**[3.1] de i até um c-uso global de x ou até um p-uso de x (RAPPS; WEYUKER, 1985).

  def-local - uma definição de uma variável x do nó i é local se existe um c-uso local de x em i que segue esta definição, e outra definição de x não aparece entre a definição e o c-uso local (RAPPS; WEYUKER, 1985).

- **Uso** - é a recuperação de um valor de uma variável em uma posição de memória associada a essa variável. Existem dois tipos de usos:

  c-uso - aquele que afeta a computação que está sendo realizada ou permite a saída de variáveis definidas anteriormente. Vale ressaltar que quando não existir nenhuma definição da variável precedente a um c-uso no mesmo bloco tem-se um **c-uso global**, caso contrário tem-se um **c-uso local** (RAPPS; WEYUKER, 1985).

  p-uso - que afeta o fluxo de controle do programa, ou seja, a variável é usada como predicado em um comando de decisão ou de repetição (RAPPS; WEYUKER,

[3.1] Um caminho $(i, n_1, ..., n_m, j)$, $m >= 0$, que não contém nenhuma definição de x nos nós n_1, ... n_m é chamado livre de definição em relação a x do nó i ao nó j. Um caminho $(i, n_1, ..., n_m, j, k)$, $m >= 0$, que não contém nenhuma definição de x nos nós n_1, ... n_m, j é chamado livre de definição em relação a x do nó i à aresta (j, k). Uma aresta (i, j) é um caminho livre de definição com respeito a x do nó i à aresta (i, j) (RAPPS; WEYUKER, 1985).

1985).

- **Indefinição** - acontece quando a localização de uma variável não estiver definida na memória ou se seu valor não for acessível.

Os critérios de fluxo de dados, segundo Weyuker (RAPPS; WEYUKER, 1985), são utilizados para estender o grafo de fluxo de controle associando os tipos de ocorrências de variáveis aos elementos deste grafo. Desta forma, cria-se um **grafo def/uso** a partir de um grafo de fluxo de controle associando-se aos nós e arcos as definições e usos de variáveis ocorridas nesses nós ou arcos. O grafo def-uso é utilizado para determinar quais caminhos e associações serão requeridas.

Utilizando o exemplo do código da Figura 3.2 e o exemplo do grafo de fluxo de controle ilustrado pela Figura 3.3, gera-se o grafo def-uso conforme a Figura 3.4.

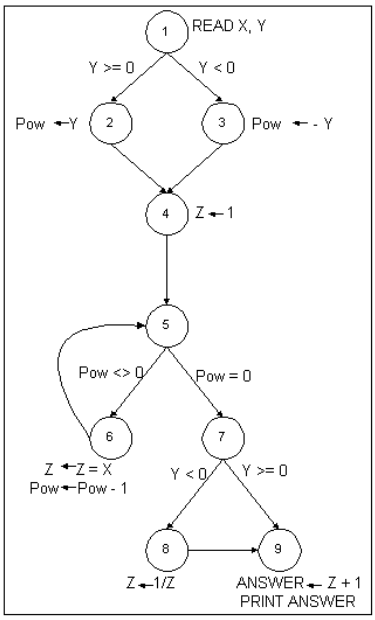

Figura 3.4: Exemplo do Grafo def-uso

Rapps e Weyuker, em (RAPPS; WEYUKER, 1985), apresentam uma família de critérios que requerem que dados de teste executem caminhos livres de definição de cada nó contendo uma definição global de uma variável para nós contendo usos computacionais globais e arcos contendo usos predicativos desta variável.

A seguir é definida uma série de conceitos estabelecidos a partir do grafo-def/uso do programa, conforme (RAPPS; WEYUKER, 1985). Para tanto, considera-se i, j, k como sendo nós distintos do grafo do programa:

- **def(i)** - é o conjunto de variáveis que possuem uma definição global no nó i.
- **c-uso(i)** - é o conjunto de variáveis que possuem um c-uso global no nó i.
- **p-uso(i, j)** - é o conjunto de variáveis para as quais a aresta (i, j) contém um p-uso.
- **dcu(x, i)** - seja $x \in$ def(i), é o conjunto de todos os nós j tal que $x \in$ c-uso(j) e para o qual existe um caminho livre de definição com respeito a x de i até j.
- **dpu (x, i)** - seja $x \in$ def(i), é o conjunto de todas as arestas (j, k) tal que $x \in$ p-uso(j, k) e para o qual existe um caminho livre de definição com respeito a x de i até j.
- **du-caminho** - um caminho $(n_1, ..., n_j, n_k)$ é du-caminho com respeito a x se n_1 tiver uma definição global de x e: **1)** n_k tiver um c-uso de x e $(n_1, ..., n_j, n_k)$ é um caminho simples livre de definição com respeito a x; ou **2)** (n_j, n_k) tem um p-uso de x e $(n_1, ..., n_j)$ é um caminho livre de laço e livre de definição com respeito a x. Um du-caminho com respeito a x é executável se existe algum conjunto de valores de entrada capaz de executar um caminho completo que o inclua.

Conforme (RAPPS; WEYUKER, 1985), obtido em (SPOTO; PERES; BUENO, 1995), dado G como um grafo def/uso, e P como um conjunto de caminhos completos de G. Então:

- P satisfaz o critério **Todos-Nós** se todo nó de G está incluído em P.
- P satisfaz o critério **Todos-Arcos** se toda aresta de G está incluída em P.
- P satisfaz o critério **Todos-Caminhos** se todos os caminhos completos de G são incluídos em P .
- P satisfaz o critério **Todas-Definições** se para todo nó i de G e todo $x \in$ def(i), P inclui um caminho livre de definição com respeito a x de i para todos elementos de dcu(x, i) ou dpu(x, i).
- P satisfaz o critério **Todos-p-usos** se para todo nó i e todo $x \in$ def(i), P inclui um caminho livre de definição com respeito a x de i para todos elementos de dpu(x, i).

- P satisfaz o critério **Todos-c-usos/Alguns-p-usos** se para todo nó i e todo $x \in$ def(i), P inclui algum caminho livre de definição com respeito a x de i para todo nó em dcu(x, i); se dcu(x, y) é vazio, então P deve incluir um caminho livre de definição com respeito a x de i para alguma aresta contida em dpu(x, i). Este critério requer que todo c-uso de uma variável x definida em um nó i deve ser incluído em algum caminho de P.

- P satisfaz o critério **Todos-p-usos/Alguns-c-usos** se para todo nó i e todo $x \in$ def(i), P inclui um caminho livre de definição com respeito a x de i para todos elementos de dpu(x, i); se dpu(x, y) é vazio, então P deve incluir um caminho livre de definição com respeito a x de i para algum nó contido em dcu(x, i). Este critério requer toda definição que é constantemente usada para ser usada em algum caminho de P.

- P satisfaz o critério **Todos-Usos** se para todo nó i e para todo $x \in$ def(i), P inclui um caminho livre de definição com respeito a x do nó i até cada um dos elementos de dcu(x, i) e até cada elemento de dpu(x, i).

- P satisfaz o critério **Todos-du-Caminhos** se para todo nó i e para toda variável $x \in$ def(i), P inclui todos os du-caminhos com respeito a x.

Exemplificando, baseado na Figura 3.4, o grafo de def-uso apresenta os conjuntos de definições e usos conforme mostra a Tabela 3.1 (SPOTO; PERES; BUENO, 1995).

Tabela 3.1: Conjuntos de Definição e Uso

Nó	**c-uso**	**def**	**aresta**	**p-uso**
1	ϕ	{x, y}	(1, 2)	{y}
2	{y}	{pow}	(1, 3)	{y}
3	{y}	{pow}	(5, 6)	{pow}
4	ϕ	{z}	(5, 7)	{pow}
5	ϕ	ϕ	(7, 8)	{y}
6	{x, z, pow}	{z, pow}	(7, 9)	{y}
7	ϕ	ϕ		
8	{z}	{z}		
9	{z}	ϕ		

Vale mencionar que Maldonado em (MALDONADO, 1991) apresenta o conceito de **potencial-uso**, onde a relação definição-uso é caracterizada sem a necessidade de

ocorrência de um uso. Para tanto, caminhos livres de definição para uma determinada variável devem ser executados, independentemente da ocorrência de uso da variável em questão em tais caminhos.

Para contribuir com o teste estrutural, Maldonado, Jino e Chaim (CHAIM, 1991) introduziram os Critérios Potenciais Usos fundamentados em variações da família de critérios apresentada por Rapps e Weyuker (RAPPS; WEYUKER, 1985), são eles: critérios todos-potenciais-usos, todos-potenciais-du-caminhos e todos-potenciais-usos/du.

Nesta próxima seção serão apresentados alguns exemplos de ferramentas que apoiam os critérios de teste estrutural.

3.3 Ferramentas de Teste Estrutural

Apesar dos esforços da engenharia de *software* para melhorar o processo de desenvolvimento de *software*, o produto final ainda fica comprometido, pois o volume e a complexidade do *software* aumentaram e o custo da atividade de teste chega a ser, conforme Pressman (PRESSMAN, 1995), 40% do custo do desenvolvimento do *software*. Mesmo realizando a atividade teste, o *software* pode ser distribuído ainda com defeitos não descobertos. Diante destes fatos, uma ferramenta de teste surge como auxílio para a atividade de teste, proporcionando um teste mais efetivo e facilitando a análise dos resultados do teste.

Segundo Chaim (CHAIM, 1991), uma ferramenta de teste estrutural deve prover as seguintes atividades: análise estática do código fonte, instrumentação do código, medição da cobertura de um conjunto de casos de teste e produção de relatórios. As ferramentas de teste podem utilizar critérios de teste estruturais para gerar casos de teste que satisfaçam tais critérios, ou analisar a cobertura de um conjunto de casos de teste.

É interessante que uma ferramenta também possa auxiliar o tratamento da não executabilidade, documentar o processo de teste e gerar a visualização de grafos (CHAIM, 1991).

A maioria das ferramentas de teste estrutural faz análise de cobertura de um conjunto de casos de teste segundo algum critério de teste selecionado. Elas apresentam ao usuário quais requisitos de teste são exigidos para que os critérios sejam satisfeitos,

objetivando orientar e auxiliar os usuários na elaboração dos casos de teste.

A seguir são apresentadas algumas ferramentas de teste estrutural que suportam a aplicação de critérios baseados em análise de fluxo de dados.

RXVP80: é uma ferramenta de teste comercial distribuída pela *General Research Corporation*, Santa Barbara, Califórnia, EUA. Realiza análise de cobertura do teste de ramos em programas escritos em Fortran. Provê suporte ao teste dinâmico, via instrumentação, análise estática do código fonte, geração do grafo de chamada dos módulos; geração do grafo de fluxo de controle dos módulos; verificação de anomalias no código fonte; verificação de tipos nas chamadas de procedimentos; e geração de relatórios (CHAIM, 1991).

TCAT (*Test-Coverage Analysis Tool*): é uma ferramenta comercial fornecida por *Software Research Corporation*, San Francisco, Califórnia, EUA. Assegura qualidade de teste de caixa branca, foi desenvolvida para guiar a geração de conjuntos de teste, avaliar o progresso do teste e dos níveis convencionais de detecção de erro. TCAT é um analisador de cobertura de ramo que provê capacidade para instrumentação automática de programas para análise e relatório de nível de cobertura de teste. TCAT é disponível para Ada, C, COBOL[3.2], Fortran e Pascal (INC., 2002a), (CHAIM, 1991).

TCAT-*Path* (*Path Test Coverage Analysis System*): é uma ferramenta distribuída pela mesma empresa que distribui TCAT. Faz análise de cobertura de todos os caminhos de uma unidade. Esta ferramenta faz uso de um utilitário chamado APG[3.3] que lista todos os possíveis caminhos de cada unidade e o número que tem sido exercitado. Possui um analisador para a complexidade ciclomática de McCabe (MCCABE; BUTLER, 1989), instrumenta o código fonte e analisa a cobertura de casos de teste. É uma abordagem mais rigorosa para análise de cobertura do que para cobertura de ramo lógico. TCAT-PATH é recomendada para projetos de *software* de missão-crítica. Freqüentemente, um conjunto de teste com 90% a 100% de cobertura de ramo lógico tem somente 20% a 30% de cobertura de caminho. TCAT-PATH é disponível para Ada, C, Fortran, COBOL e Pascal (INC., 2002b).

Ferramenta de Herman: suporta a aplicação do primeiro critério baseado em análise de fluxo de dados proposto por Herman, este critério é semelhante ao critério

[3.2] COmmon Business Oriented Language
[3.3] All Paths Generator

todos-c-uso proposto por Rapps e Weyuker (RAPPS; WEYUKER, 1985). Sendo assim, a ferramenta faz análise de cobertura para o critério todos-c-usos de programas escritos em Fortran, trata aspectos inter-procedurais e associações definição-uso de elementos de vetores e matrizes, instrumenta o número do elemento do vetor sempre que este é referenciado no código fonte (CHAIM, 1991).

ASSET[3.4]**:** foi desenvolvida por Frankl e Weyuker, suporta basicamente a aplicação dos critérios de Rapps e Weyuker em programas escritos em Pascal. Não faz nenhum tratamento inter-procedural, fornece geração automática do programa executável, possui uma interface gráfica que apresenta o grafo de fluxo de controle e provê uma interação amigável com o usuário. ASSET, determina se um determinado conjunto de teste é adequado com respeito ao critério, e produz uma lista de alguns pares de nós requeridos pelo critério mas não exercitado pelo dado de teste. Esta lista pode então ser usada para fortalecer o conjunto de dados de teste (FRANKL P. G.; WEISS; WEYUKER, 1985), (CHAIM, 1991).

PROTESTE: é um protótipo desenvolvido na Universidade Federal do Rio Grande do Sul. Tem por objetivo proporcionar um ambiente completo para dar suporte ao teste estrutural, incluindo critérios baseados em fluxo de dados e controle. Gera a visualização do grafo de fluxo de controle, auxilia na derivação dos dados de entrada para um caso de teste, instrumenta a unidade em teste e suporta o teste de programas escritos em Pascal. Porém, pode ser ajustado para outras linguagens (SILVA, 1995), (CHAIM, 1991).

POKE-TOOL (*Potencial Uses Criteria Tool for Program Testing*): Em (CHAIM, 1991), Chaim define uma ferramenta de apoio ao teste estrutural de programas baseado na análise de fluxo de dados. Suporta o teste de programas escritos em C, faz análise do código fonte e criação de uma base de dados, gera relatórios com informações da análise estática e identificação da estrutura de dados e de controle, instrumenta o código fonte, executa casos de teste e analisa os resultados do teste.

xSuds[3.5]**:** Foi desenvolvida no laboratório *Telcordia Applied Research* para analisar o comportamento dinâmico do *software* e para permitir ao usuário visualizar todos os dados do programa. É composta de sete ferramentas (*xAtac*, *xRegress*, *xVue*, *xSlice*, *xProf*, *xFind*, *xDiff*) que possibilitam compreender, depurar, testar, manter e analisar programas escritos em C ou C++. O testador pode visualizar o código fonte do programa através de cores (INC., 1998), (LI et al., 1999), (HORGAN; LONDON; LYU, 1994).

[3.4] A System to Select and Evaluate Tests
[3.5] *Software understanding system*

O próximo capítulo descreve uma ferramenta de teste chamada JaBUTi, apresentada em (VINCENZI et al., 2003), que foi adaptada sendo denominada de JaBUTi/MA, conforme (DELAMARO; VINCENZI; MALDONADO, 2004), e utilizada para testar os códigos móveis implementados, como estudos de casos, para este trabalho.

4 JaBUTi

Para auxiliar a aplicação de técnicas de teste é necessária a utilização de ferramentas de apoio. Sendo o teste de *software*, como dito anteriormente, responsável por 40% do esforço utilizado num projeto de desenvolvimento de *software*, a utilização de ferramentas de apoio pode reduzir o tempo despendido para a atividade de teste sem reduzir a eficácia (PRESSMAN, 1995).

O uso de ferramentas automatizadas que auxiliam a atividade de teste está crescendo e melhorando a confiabilidade dos sistemas baseados em computador.

Vincenzi *et al.*, em (VINCENZI et al., 2003), apresenta uma ferramenta de teste baseada em fluxo de dados e fluxo de controle para programas e componentes Java. Esta ferramenta é chamada de JaBUTi[4.1]. A principal característica desta ferramenta é que ela não requer o código fonte para realizar o teste, podendo se basear no código de *bytes* do programa Java, visando proporcionar a execução do teste estrutural sobre componentes e programas Java.

A Ferramenta JaBUTi implementa dois critérios baseados em fluxo de controle: todos-nós e todos-arcos; e um critério baseado em fluxo de dados: todos-usos. Tais critérios são usados para avaliar a qualidade de um determinado componente ou programa Java. A JaBUTi implementa um conjunto de funcionalidades fornecidas por *xSuds*, como já mencionado no Capítulo 3, que é um conjunto de ferramentas para testar programas C e C++.

A interface gráfica da JaBUTi permite que um testador principiante possa explorar e aprender conceitos referentes ao teste de fluxo de dados e fluxo de controle. Além disso, ela fornece uma boa visualização da cobertura do programa em teste, mostrando quais partes do código foram cobertas e quais não obtiveram cobertura. A JaBUTi

[4.1] *Java Bytecode Understanding and Testing*

através de sua análise de cobertura facilita a geração de bons casos de teste e através da ferramenta *slice*, apresentada na seção 4.5.2, auxilia o testador na detecção de falhas.

Neste capítulo serão descritas na Seção 4.1 as características referentes a linguagem Java, as fases de programação em Java, a independência de plataforma provida pela Máquina Virtual Java e as instruções de *bytecode* utilizadas pela Ferramenta JaBUTi. Na Seção 4.2 será descrito como são gerados os grafos através da Ferramenta JaBUTi. A Seção 4.3 descreve a forma que a Ferramenta JaBUTi analisa o peso de cobertura do teste aplicado através do GFC do programa. Na Seção 4.4 é descrito de forma sucinta o processo de instrumentação que a Ferramenta JaBUTi realiza. A Seção 4.5 apresenta as principais funções que a Ferramenta JaBUTi disponibiliza, bem como as ferramentas de análise de cobertura e *slice*. Na Seção 4.6 é apresentada a Ferramenta JaBUTi/MA que é uma adaptação da Ferramenta JaBUTi com extensão para suportar o teste de agentes móveis.

4.1 Java, JVM e Instruções *Bytecode*

A Ferramenta JaBUTi suporta programas desenvolvidos através da linguagem Java. Tal foco é interessante devido ao grande número de programadores utilizar a linguagem Java como principal meio de desenvolvimento de *software*. Dentro deste contexto, verifica-se a grande necessidade de empregar a atividade de teste em programas e componentes desenvolvidos em Java, visando oferecer suporte à qualidade de tais *softwares*.

A linguagem Java é amplamente utilizada não só pelas suas características clássicas, pois também, como em (CHAN; GRIFFITH; IASI, 1999), é uma linguagem de programação orientada a objetos; é uma linguagem segura, pois possui defesas de segurança que reduz a possibilidade de ser utilizada para criar programas maliciosos; é robusta, pois é confiável gerenciando memória e tratando exceções; possibilita desenvolver *applets*, que são programas embutidos em HTML[4.2] próprios para executarem na Internet; e a principal de todas as características é que a linguagem Java é independente de plataforma, ou seja, um programa desenvolvido em Java pode ser executado em qualquer *hardware* ou *software*.

Esta independência de plataforma foi o principal fator para que a linguagem Java pudesse ser mundialmente utilizada de forma crescente, sendo, conforme Deitel (DEITEL;

[4.2] *HyperText Markup Language*

DEITEL, 2001), uma das linguagens de desenvolvimento de *software* mais populares. Para melhor compreenssão, quando uma aplicação Java é desenvolvida, realizam-se as seguintes fases ilustradas na Figura 4.1, segundo Deitel (DEITEL; DEITEL, 2001):

- **Edição:** consiste em editar um arquivo de programa e armazená-lo em disco com extensão `.java`.
- **Compilação:** o compilador java traduz o programa para *bytecodes*, gerando o arquivo `.class`.
- **Carga:** o carregador de classe transfere o arquivo de *bytecode*, ou seja, o `.class` para a memória.
- **Verificação:** o verificador de *bytecodes* verifica se os *bytecodes* contidos no `.class` são válidos e não violam restrições de segurança[4.3] da linguagem Java.
- **Execução:** o interpretador interpreta cada *bytecode* realizando a ação especificada pelo programa.

A independência de plataforma provida pela linguagem Java deve-se à Máquina Virtual Java (JVM[4.4]). A Máquina Virtual Java é um programa que interpreta os *bytecodes* contidos no arquivo `.class`, ou seja, os *bytecodes* gerados após a compilação são instruções de máquina para a JVM.

Conforme Chan (CHAN; GRIFFITH; IASI, 1999), cada computador pode ter seu próprio interpretador, ou seja, cada plataforma possui sua própria implementação da Máquina Virtual Java, mas todas elas são capazes de ler *bytecodes* ou um arquivo `.class` comum.

Tal ambiente é ideal para a Internet, já que um programa pode ser executado em diferentes máquinas pela *Web*. Portanto, o que deve ser comprendido é que a JVM é uma camada adicional entre o processador e um arquivo de *bytecodes*, conforme ilustrado na Figura 4.2.

A JVM especificamente, conforme Chan (CHAN; GRIFFITH; IASI, 1999), inclui um conjunto de instruções de *bytecode*, um conjunto de registradores, uma pilha, um coletor

[4.3] As *applets* Java não podem ler nem escrever arquivos do disco local, não tem ponteiros para a memória principal e não podem danificar a memória fora de seu próprio espaço de memória (CHAN; GRIFFITH; IASI, 1999).

[4.4] *Java Virtual Machine*

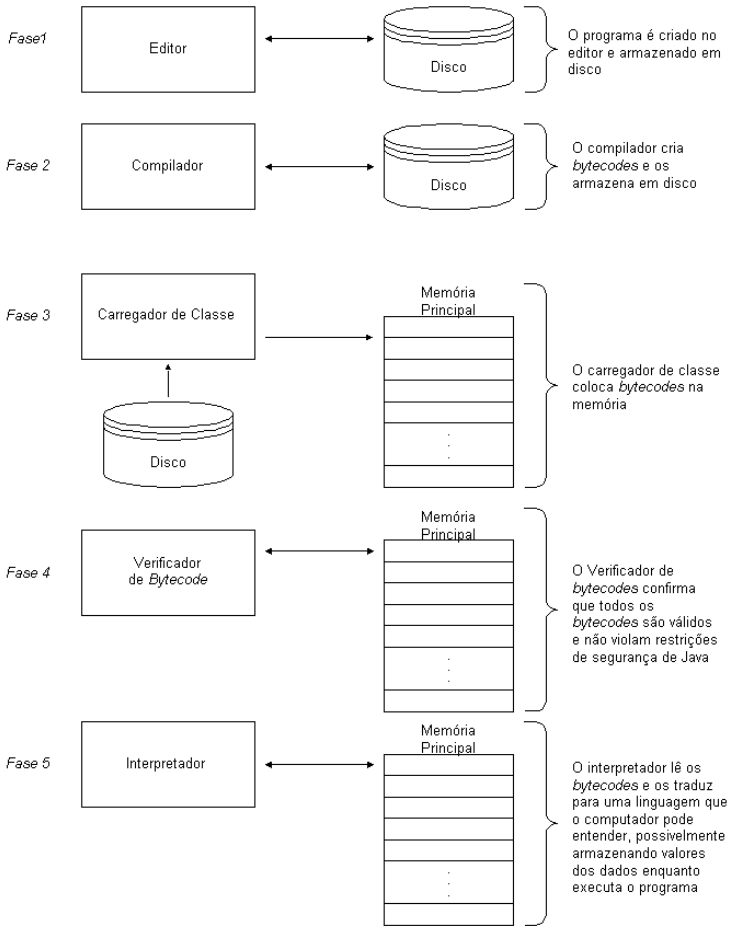

Figura 4.1: Fases de Desenvolvimento de *Software* em Java

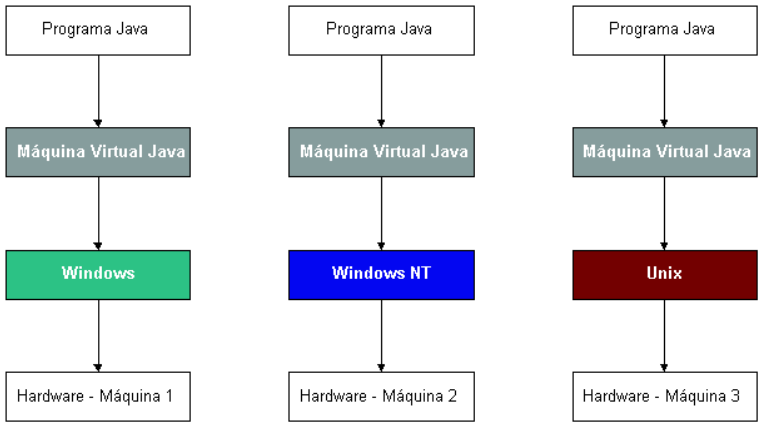

Figura 4.2: Independência de Plataforma provida pela JVM

de lixo e uma área para armazernar métodos. Colocado em *software*, o processador Java (JVM) possibilita a execução de um mesmo programa em máquinas diferentes.

A Máquina Virtual Java, conforme Lindholm e Yellin (LINDHOLM; YELLIN, 1999), tem uma estrutura orientada à pilha e é usada para carregar e executar arquivos de classe.

O arquivo de classe é uma representação binária que contém dados da classe, tal como: nome da classe, nome da superclasse, informações sobre as variáveis e constantes, e as instruções *bytecode* para cada método.

Segundo Lindholm e Yellin (LINDHOLM; YELLIN, 1999):

> Uma instrução JVM consiste de um *opcode* especificando a operação a ser executada, seguida por zero ou mais operandos agregando valores a serem operados.

Ainda em (LINDHOLM; YELLIN, 1999), um formato de *opcode* deve apresentar a seguinte forma:

```
<index> <opcode> [<operand1> [<operand2> ...]] [<comment>]
```

O `<index>` é o índice de *opcode*[4.5] da instrução no *array* que contém os *bytes* do código JVM para o método. O `<opcode>` é o mnemônico para os *opcodes* de instrução, e os `<operandN>` são os operandos de instrução. O `<comment>` é opcional e é dado na sintaxe de comentário de fim de linha.

A JVM possui três *opcodes* reservados para uso interno de sua implementação. Dois destes com mnemônicos `impdep1` e `impdep2` foram planejados para prover "*back doors*"[4.6], e o terceiro com mnemônico `breakpoint` foi planejado para ser usado por depuradores implementando *breakpoints*[4.7].

As instruções de *bytecode* definidas em (LINDHOLM; YELLIN, 1999), são classificadas em 10 grupos diferentes, conforme a Tabela 4.1 obtida em (VINCENZI et al., 2003), sendo que, tais grupos possuem subgrupos apresentando um conjunto completo composto por 204 instruções. Desse conjunto, há 3 instruções reservadas (`impdep1`, `impdep2` e `breakpoint`) e 43 que podem levantar uma exceção (na Tabela 4.1 tais instruções constam em negrito), sendo que dessas 43 só a `athrow` pode explicitamente atirar uma exceção, todas as outras 42 instruções podem atirar exceções implicitamente.

Em suma, uma instrução *bytecode* é representada por um *one-byte opcode* seguido por valores de operando. Cada *one-byte opcode* tem um mnemônico correspondente.

[4.5] Código de operação.

[4.6] A porta de fundo permite que sempre que uma máquina estiver ligada em rede os outros *hosts*, pertencentes a esta rede, terão acesso remoto a tal máquina.

[4.7] Ponto onde o programa para quando ocorre um erro de programação, auxilia a depuração de programas.

O tipo de instrução é representado por uma letra, indicando o tipo de operando, para tanto deve-se usar: `i` para *integer*, `l` para *long*, `s` para *short*, `b` para *byte*, `c` para *char*, `f` para *float*, `d` para *double* e `a` para *referência*.

Tabela 4.1: Conjunto de Instruções *Bytecode*

Grupo	**SubGrupo**	**Conjunto de Instrução**
Do nothing	-	nop
Carregamento e Armazenamento	Carrega uma variável local para a pilha de operando	aload, aload_<n>†, dload, dload_<n>, fload, fload_<n>, iload, iload_<n>, lload, lload_<n>
	Armazena um valor da pilha de operando dentro de uma variável local	astore, astore_<n>, dstore, dstore_<n>, fstore, fstore_<n>, istore, istore_<n>, lstore, lstore_<n>
	Carrega uma constante para a pilha de operando	bipush, sipush, ldc, ldc_w, ldc2_w, aconst_null, dconst_<d>, fconst_<f>, iconst_m1, iconst_<i>, lconst_<l>
	Oferece acesso para mais variáveis locais usarem um indíce extenso ou para um operando imediato amplo	wide
Aritmética	Adição	dadd, fadd, iadd, ladd
	Subtração	dsub, fsub, isub, lsub
	Multiplicação	dmul, fmul, imul, lmul
	Divisão	ddiv, fdiv, **idiv**, **ldiv**
	Resto	drem, frem, **irem**, **lrem**
	Negação	dneg, fneg, ineg, lneg
	Substituição	ishl, ishr, iushr, lshl, lshr, lushr
	Bitwise	iand, ior, ixor, land, lor, lxor
	Incremento de Variável Local	iinc
	Comparação	dcmpg, dcmpl, fcmpg, fcmpl, lcmp
Conversão de Tipo	-	d2f, d2i, d2l, f2d, f2i, f2l, i2b, i2c, i2d, i2f, i2l, i2s, l2d, l2f, l2i
Criação e Manipulação de Objeto	Cria instância de classe	**new**
	Cria matriz	**newarray**, **anewarray**, **multianewarray**
	Accesso de Campos de Classes	**getfield**, **getstatic**, **putfield**, **putstatic**
	Carrega um componente de matriz para a pilha de operando	**aaload**, **baload**, **caload**, **daload**, **faload**, **iaload**, **laload**, **saload**
	Armazena um valor da pilha de operando como um componente de matriz	**aastore**, **bastore**, **castore**, **dastore**, **fastore**, **iastore**, **lastore**, **sastore**
	Obtém o comprimento da matriz	**arraylength**
	Checa propriedades de instâncias de classes ou matriz	**checkcast**, **instanceof**
Transferência de Controle	Ramo Incondicional	goto,goto_w, jsr, jsr_w, ret, **athrow**
	Ramo Condicional	if_acmpeq, if_acmpne, if_icmpeq, if_icmpge, if_icmpgt, if_icmple, if_icmplt, if_icmpne, ifeq, ifge, ifgt, ifle, iflt, ifne, ifnonnull, ifnull
	Ramo Condicional Composto	lookupswitch, tableswitch
Acesso de Método	Invocação	**invokeinterface**, **invokespecial**, **invokestatic**, **invokevirtual**
	Retorno	**areturn**, **dreturn**, **freturn**, **ireturn**, **lreturn**, **return**
Gerenciamento de Pilha	-	dup, dup_x1, dup_x2, dup2, dup2_x1, dup2_x2, pop, pop2, swap
Sincronização	-	**monitorenter**, **monitorexit**
Reservado	-	breakpoint, impdep1, impdep2

† <n> representa um índice válido a respeito da variável local da matriz da estrutura corrente (VINCENZI et al., 2003).

Através das instruções de *bytecode* apresentadas, vale mostrar um exemplo de código em Java, obtido em (LINDHOLM; YELLIN, 1999), ilustrado pela Figura 4.3, onde tem-se um método denominado `spin()` composto por uma estrutura de repetição.

```
void spin()
{
  int i;
  for(i = 0; i < 100; i++)
  {
     ;    //o corpo do laço de repetição está vazio
  }
}
```

Figura 4.3: Exemplo de Código em Java - Método `spin()`

Após a compilação do método serão gerados *bytecodes*, apresentados pela Figura 4.4, obtidos em (LINDHOLM; YELLIN, 1999), que representam toda a estrutura de dados do método `spin()`.

```
Method void spin()
0   iconst_0      //empilha inteiro constante 0
1   istore_1      //armazena dentro da variavel local 1 (i=0)
2   goto 8        //primeira vez no laço não faz incremento
5   iinc 1 1      //incrementa variável local 1 por 1 (i++)
8   iload_1       //carrega variável local 1 (i)
9   bipush 100    //empilha inteiro constante 100
11  if_icmplt 5   //compara e repete se menor do que (i<100)
14  return        //retorna void quando concluído
```

Figura 4.4: Bytecode do Método `spin()`

Observando os *bytecodes* do arquivo de classe gerado verifica-se que um arquivo de classe contém muitas informações de alto nível sobre um programa. Deste modo, a JaBUTi (VINCENZI et al., 2003) tem por finalidade utilizar tal recurso fornecendo o máximo de informações possíveis durante a atividade de teste, mesmo que o código fonte original não esteja disponível. Coleta, para tanto, informações de fluxo de controle e de dados a partir do *bytecode* e usa tais informações para fornecer critérios de cobertura para programas Java.

A seção seguinte descreve como a JaBUTi gera o grafo Def-Uso de um programa escrito em Java, com base em cada método do programa, e apresenta exemplos para elucidar a geração de grafos.

4.2 Gerando Grafos com a JaBUTi

Considerando o conjunto de instruções de *bytecode*, Vincenzi *et al.* (VINCENZI et al., 2003) descreve uma abordagem para construir o grafo de programa através da leitura da instrução de *bytecode* de um determinado método, tendo como parâmetro a análise de fluxo de controle de *bytecodes* de Zhao (ZHAO, 1999).

Para tanto, Vincenzi *et al.* em (VINCENZI et al., 2003) diz que $G(m) = (N, E, s, T)$ é um grafo de programa de um método m se para cada bloco de sentença de m existir um nó $n \in N$ e para cada possível transferência de controle entre um bloco representado por n_1 e um bloco representado por n_2 existir uma aresta $(n_1, n_2) \in E$. Sendo que, para melhor esclarecimento, N é o conjunto de nós, E é o conjunto de arestas (*edge*), s (*start*) corresponde ao nó inicial, ou seja, o bloco cuja sentença é a primeira do programa. E, por fim, T é o conjunto de nós terminais, ou seja, os blocos cujas últimas sentenças são as terminais do programa.

A JaBUTi (VINCENZI et al., 2003), devido ao mecanismo de manipulação de exceção em Java, distingue dois conjuntos de arestas: primárias e secundárias. As primárias representam o fluxo regular de controle, ou seja, sem exceção, e as secundárias representam o fluxo de controle que manipulam a exceção. Deste modo, na JaBUTi, o GFC para um determinado método m é um grafo estendido $G_b(m) = (N, E_p, E_s, s, T)$, onde E_p é o conjunto de arestas primárias e E_s é o conjunto de arestas secundárias, sendo, portanto, subconjuntos separados de E.

Para melhor entendimento, Vicenzi *et al.* mostra em (VINCENZI et al., 2003) como um grafo de bloco é criado. Inicialmente, cria-se um Grafo de Instrução ($G_i(m)$) contendo uma única instrução por nó. Neste GFC, se uma instrução j puder ser executada após a instrução i, então existe uma aresta do nó i ao nó j. Se a instrução i está no alcance de um manipulador de exceção que começa na instrução j, então existe uma segunda aresta do nó i ao nó j. Depois de criado, $G_i(m)$ é reduzido gerando o Grafo de Bloco $G_b(m)$. Em (VINCENZI et al., 2003) consta o algoritmo detalhado para realizar a redução do grafo de instrução dando origem ao grafo de bloco.

Tal redução ocorre tendo em mãos o grafo de instrução $G_i(m)$ e verificando cada bloco de instrução, ou seja, verifica se é uma instrução inicial de um bloco, então, caso seja, esta instrução passa a ser a instrução inicial de um bloco em $G_b(m)$, isto também

vale quando encontra-se uma instrução final, então, esta instrução passa a ser a instrução final de um bloco em $G_b(m)$. Também verifica se uma determinada instrução de $G_i(m)$ faz parte do conjunto de arestas primárias ou secundárias do grafo $G_b(m)$. Maiores detalhes sobre o algoritmo constam em (VINCENZI et al., 2003).

Conforme Vincenzi *et al.* (VINCENZI et al., 2003), ao construir o GFC, caso exista um manipulador de exceção, um novo nó também é criado para qualquer instrução que puder gerar uma exceção (conforme instruções em negrito na Tabela 4.1), o que pode aumentar significativamente o número de nós no GFC.

Vincenzi *et al.* (VINCENZI et al., 2003) continua dizendo que, após o Grafo de Bloco $G_b(m)$ ser construído, é realizado um pré-processamento eliminando nós desnecessários, por exemplo, nós que contenham uma única instrução `goto`. Eles são eliminados conectando diretamente suas arestas que chegam com as arestas que apontam para seus sucessores. Em seguida, após a redução de nós, estes são rotulados conforme o número de seu primeiro *bytecode* equivalente. E os nós expandidos são rotulados com o rótulo do nó que causou sua invocação seguido pelo rótulo do nó de destino para o qual irá.

Para exemplificar a criação do GFC é apresentado um exemplo de (VINCENZI et al., 2003), no qual tem-se um simples programa Java com um método que tem por função calcular a média de um conjunto de números inteiros. Tal exemplo é ilustrado pela Figura 4.5 que consta o código fonte Java, o seu *bytecode* e a tabela de exceção que mostra o manipulador de exceção válido de acordo com o correspondente *bytecode* equivalente.

Na Figura 4.5 as chaves à esquerda das instruções de *bytecode* indicam o conjunto de instruções que compõem cada nó. A Figura 4.6a apresenta o GFC para o método `average` da classe `Vet`, conforme o *bytecode* apresentado na Figura 4.5.

Analisando o exemplo de (VINCENZI et al., 2003), ilustrado pelas Figuras 4.5 e 4.6, verifica-se que o terceiro nó possui rótulo 15, devido sua primeira instrução de *bytecode* ser "`15: aload_0`". Os nós 12, 57 e 71 foram eliminados do GFC devido possuírem uma única instrução `goto`.

A JaBUTi (VINCENZI et al., 2003) apresenta três tipos de ilustrações de nós, conforme Figura 4.6:

- **Linha em negrito:** representa nós terminais, no exemplo temos os nós 79 e 97.

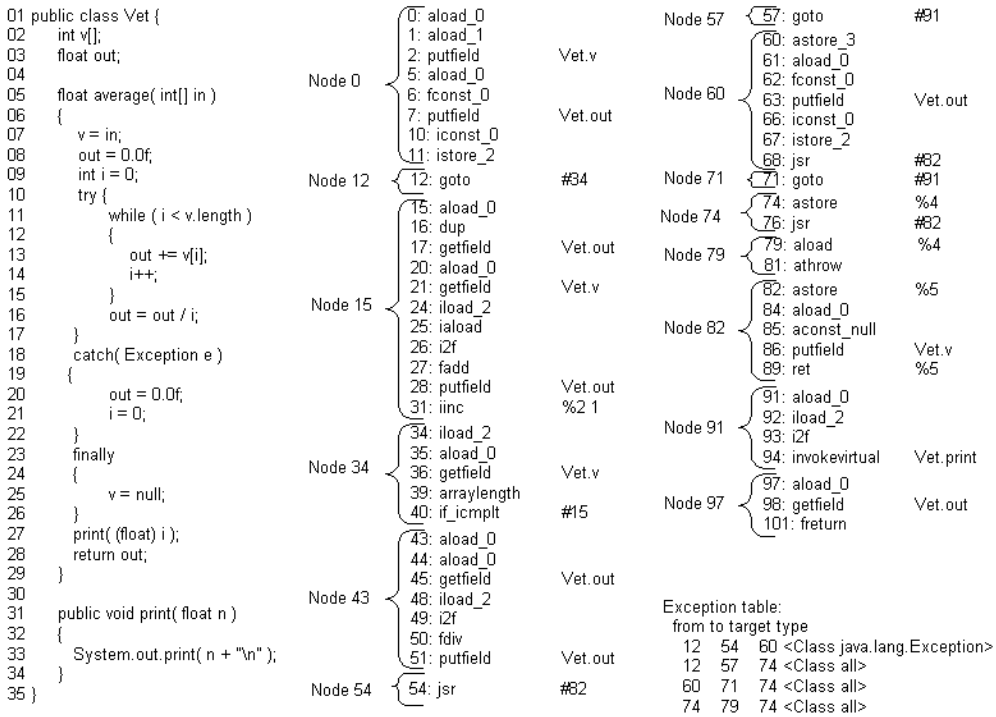

Figura 4.5: Um programa Java, suas instruções de *bytecode* e os blocos de comandos básicos.

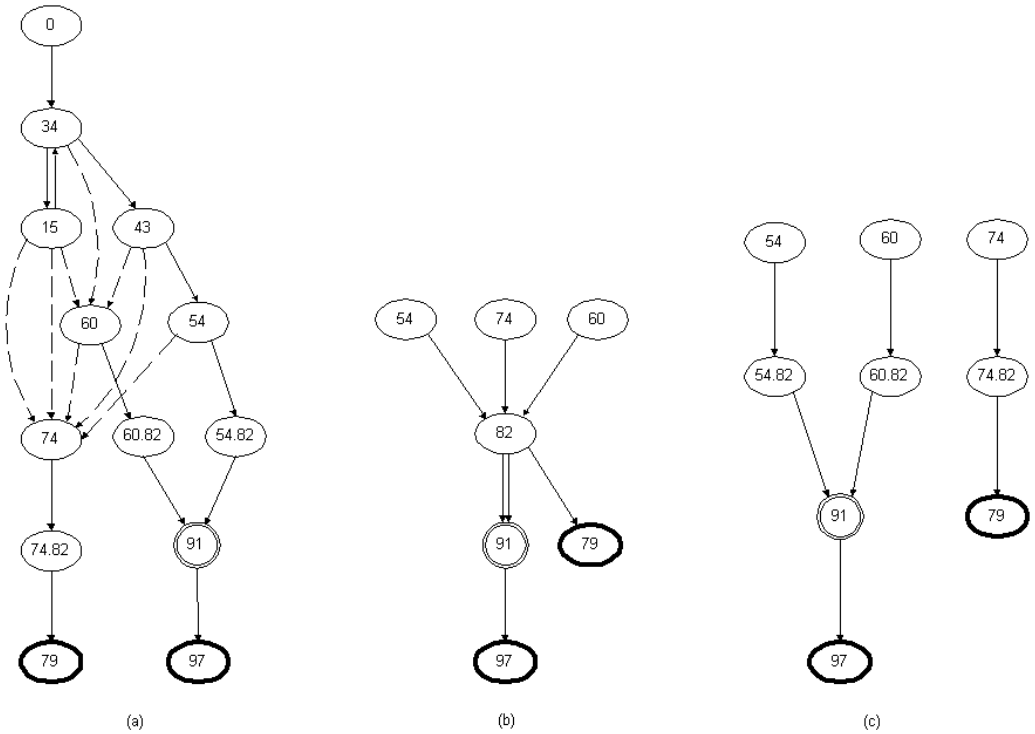

Figura 4.6: Grafo do Método `Vet.average` (a), um único nó *finally* (b), nó *finally* extendido (c).

- **Linha dupla:** representa nós de chamada, no exemplo temos o nó 91.
- **Linha única:** representa todos os outros nós, no exemplo temos vários como 0 e 43.

Também, em (VINCENZI et al., 2003) existe dois tipos de ilustrações de arestas, conforme ilustração da Figura 4.6:

- **Linha contínua:** representa arestas primárias, no exemplo temos várias como (0, 34) e (34, 43).
- **Linha tracejada:** representa arestas secundárias que tratam de exceção, no exemplo temos várias arestas secundárias como (34, 74) e (15, 74).

Continuando a análise do exemplo, em (VINCENZI et al., 2003), ilustrado na Figura 4.6a, o nó 82 é um nó de destino de uma instrução de salto presente no fim dos nós 54, 60 e 74, ele deve ser expandido três vezes, conforme os nós rotulados 54.82, 60.82 e 74.82. Assim, como ilustrado na Figura 4.6c, tal expansão é importante devido às chamadas à partir dos nós 54 e 60, onde a execução é retomada no nó 91, ao contrário da chamada do nó 74, onde a execução é retomada ao nó 79. Portanto, se não fosse realizada a expansão, conforme ilustra a Figura 4.6b, só um nó 82 existiria dando a impressão de que qualquer chamada dos nós 54, 60 e 74 podem retomar a execução indistintamente dos nós 91 e 79, o que na realidade não ocorre. Conforme dito anteriormente, tal situação é ilustrada na Figura 4.6c.

Com o intuito de prover análise de fluxo de dados, a JaBUTi gera grafos def-uso, tendo como base a análise de dependência de *bytecode* proposta por Zhao em (ZHAO, 2000).

Para melhor entendimento, Vincenzi *et al.* (VINCENZI et al., 2003) gera um grafo def-uso para o exemplo da Figura 4.5, onde apresenta a classe `Vet` e o método `average`. Tem-se, portanto, na Figura 4.7, o grafo def-uso para o método `Vet.average`, no qual observa-se que os nomes das variáveis são diferentes dos seus nomes no código fonte original. Para tanto, para representar uma variável local usa-se um `L@` precedendo o índice da mesma, e para representar variável estática usa-se `S@`.

Analisando o exemplo do método `Vet.average`, a variável `i` na linha 14 do código fonte (Figura 4.5) é referida pelo *bytecode* com `%2` da instrução de *bytecode* (`15: iinc`

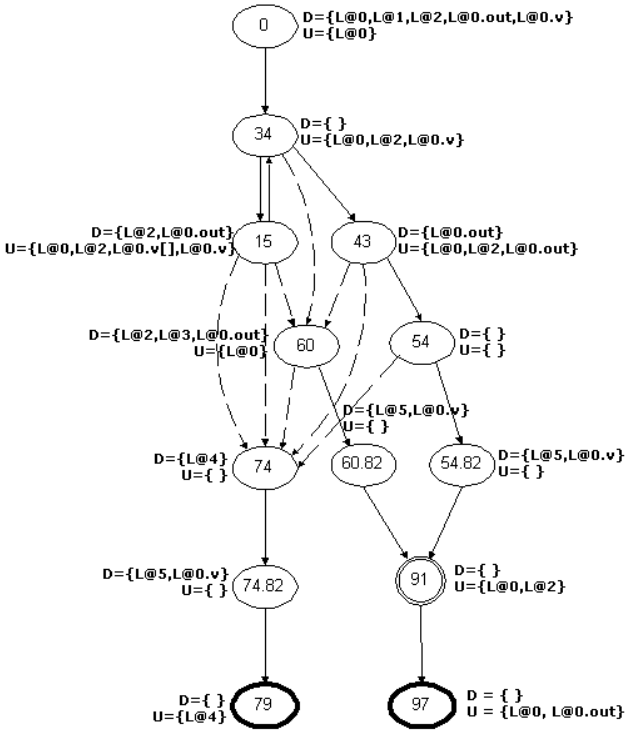

Figura 4.7: Grafo Def-Uso para o método `Vet.average`.

`%2 1`) na Figura 4.5, ou como 2 conforme a instrução de *bytecode* (`11: istore_2`) e (`34: iload_2`) na mesma Figura 4.5, desde que esteja localizada no conector 2 na tabela de variável local. Assim, a Ferramenta JaBUTi (VINCENZI et al., 2003) chama essa variável de `L@2`. Em (VINCENZI et al., 2003), quando tem-se `L@0` sempre se refere a `this`[4.8], o objeto corrente. Dessa forma, considerando as instruções de *bytecode* da Figura 4.5, usa-se `L@0.out` para indicar que `out` é uma variável de instância. Considerando a referência de *arrays* e seus componentes, tem-se `L@0.v` para representar a própria referência do *array* `v` da Figura 4.5 e `L@0.v[]` para representar um determinado componente do *array*.

Vincenzi *et al.* (VINCENZI et al., 2003) estende a análise de dependência de Zhao (ZHAO, 2000) para gerar grafos def-uso incluindo variáveis de instância, variáveis estáticas, referência de *array* e componente de *array* ao determinar o conjunto de variáveis usadas. Por fim, cada nó do grafo def-uso possui dois conjuntos:

1. Um conjunto de definição D, incluindo todas as variáveis definidas no nó.

2. Um conjunto de uso U, incluindo todas as variáveis usadas no nó.

[4.8]Cada objeto tem acesso a uma referência a ele próprio, chamada de referência `this`. A referência `this` é implicitamente utilizada para referenciar variáveis de instância e métodos de um objeto (DEITEL; DEITEL, 2001).

A Figura 4.7 apresenta o conjunto completo de variáveis definidas e usadas do método `Vet.average`, tendo como referência o código de *bytecode* ilustrado pela Figura 4.5. Segundo Weyuker (RAPPS; WEYUKER, 1985), um grafo def-uso é muito útil quando se trata de critérios de teste baseados em fluxo de dados. Vincenzi *et al.* (VINCENZI et al., 2003) tendo o intuito de utilizar a aplicação de tal teste, em conjunto com o teste de fluxo de controle, retira informações de fluxo de dados e de controle dos *bytecodes* contidos no arquivo de classe a ser testado. Assim, coletadas as informações para cada método, pode-se definir e aplicar os critérios de teste intra-método.

A Ferramenta JaBUTi (VINCENZI et al., 2003) propõe conduzir teste de cobertura em programas e componentes Java através da utilização de três critérios diferentes de teste, como já dito anteriormente: dois são baseados em fluxo de controle e um é baseado em fluxo de dados. Sendo que os critérios baseados em fluxo de controle utilizados pela JaBUTi são todos-nós e todos-arcos, e o critério baseado em fluxo de dados é todos-usos, composto pelos critérios c-uso e p-uso.

Vale ressaltar que, quanto ao critério todos-arcos, conforme dito no Capítulo 3, este requer que cada aresta do GFC seja coberta pelo menos uma vez. Na JaBUTi (VINCENZI et al., 2003), isso quer dizer que deve-se cobrir além do desvio verdadeiro e falso de cada sentença condicional no método e de cada sentença condicional composta, também o conjunto de exceções manipuláveis que possam ser atiradas em cada bloco.

Já no caso do critério todos-usos, segundo Vincenzi (VINCENZI et al., 2003), a partir do *bytecode* não há possibilidade de distinguir p-uso e c-uso. Portanto, na implementação do critério todos-usos, a JaBUTi assume que um uso, em um nó com mais de uma aresta que sai, é um p-uso e o associa com cada uma das arestas.

A próxima seção apresenta como a Ferramenta JaBUTi analisa a cobertura de um teste aplicado, estabelecendo pesos de cobertura para cada nó existente no grafo de um determinado método.

4.3 A Análise de Cobertura da JaBUTi

Segundo Delamaro, em (DELAMARO, 2005), a **análise de cobertura** é uma das funcionalidades mais importantes que uma ferramenta de teste deve apresentar, pois, além de permitir a visualização da cobertura da execução dos casos de teste também permite

identificar quais nós devem ser cobertos para se obter maior cobertura.

Vincenzi *et al.* (VINCENZI et al., 2003) atribui diferentes pesos a cada nó do GFC baseado em análise de dominador e "super-bloco". O intuito é gerar um teste para cobrir a área com o peso mais alto antes de outras áreas para conseguir a máxima cobertura. Assim, a JaBUTi, fornece informações referente à cobertura do código para que possa ser realizada uma atividade de teste que possua poucos casos de teste, porém realiza uma ampla cobertura baseada em fluxo de controle (bloco e decisão) e em fluxo de dados (todos-usos).

Segundo Vincenzi *et al.* (VINCENZI et al., 2003), um **super-bloco** é um subconjunto de nós com a propriedade de que se qualquer nó em um super bloco for coberto, então, todos os nós nesse super bloco serão cobertos.

Para melhorar a compreenssão sobre a dependência entre os blocos de comando em um GFC, Vincenzi continua o exemplo do método `Vet.average` e apresenta, conforme a Figura 4.8a, os nós 0 e 34 que pré-dominam o nó 74, e os nós 74.82 e 79 que o pós-dominam. As Figuras 4.8a e 4.8b mostram as árvores de pré e pós-dominador do GFC ilustrado pela Figura 4.6a.

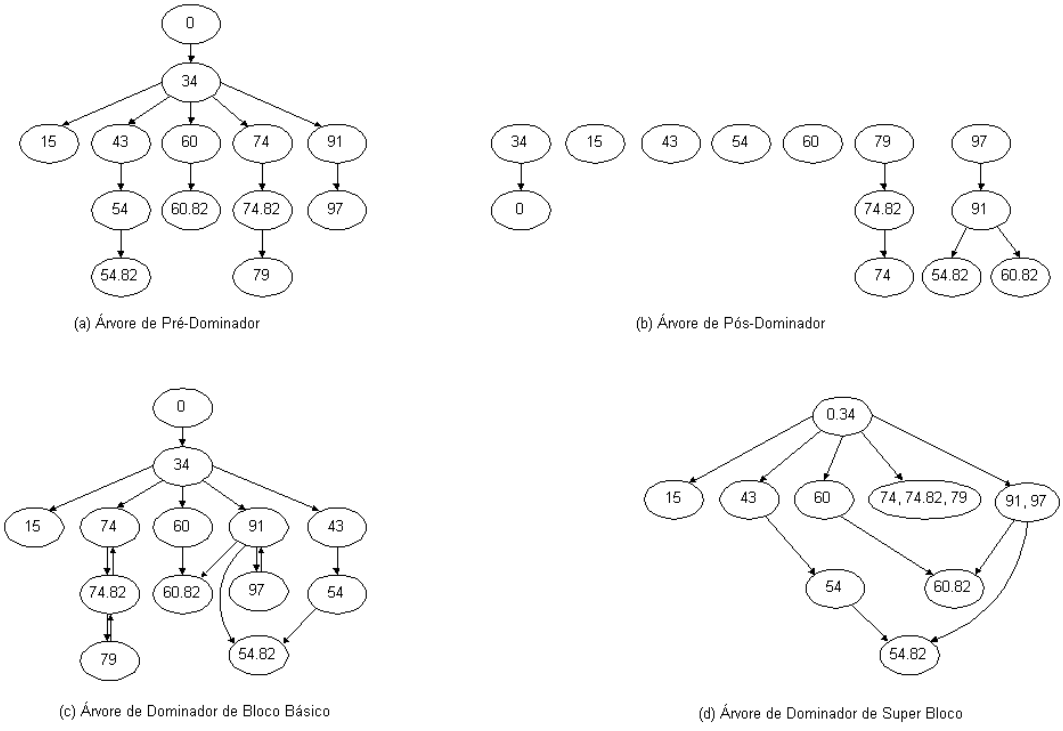

Figura 4.8: Análise de Dependência de Fluxo de Controle

A relação de dominador entre nós do GFC é representada por um "grafo de

dominador de bloco básico" que corresponde à união das árvores de pré e pós-dominador, conforme a Figura 4.8c. Através de tal grafo os super blocos são identificados, gerando, assim, o "grafo de dominador de super bloco", como por exemplo, os nós 74, 74.82 e 79 que compõem um super bloco, como ilustrado na Figura 4.8d. Com tal grafo pode-se calcular o peso de cada nó do GFC.

Em (VINCENZI et al., 2003), o peso de um nó é o número de nós que não foram cobertos, mas serão, se aquele nó for coberto. A Figura 4.9a mostra em negrito o peso inicial de cada super bloco.

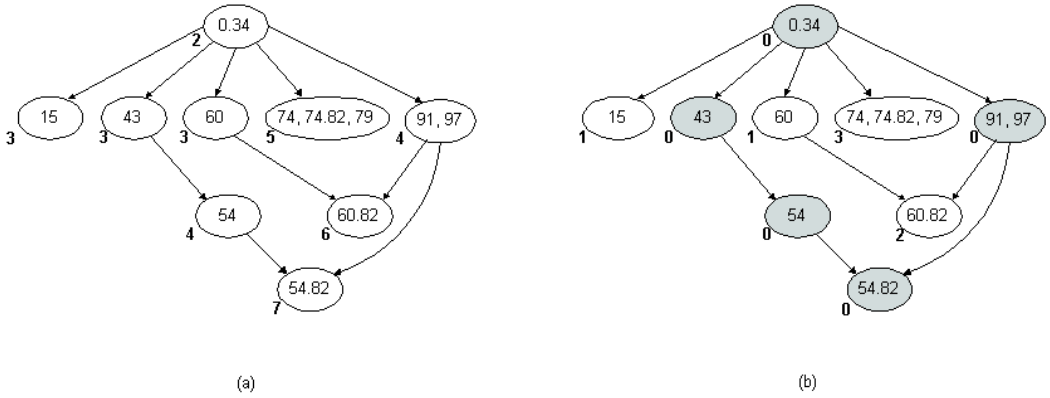

Figura 4.9: Grafo de Dominador de Super Bloco com pesos: (a) antes da execução de qualquer caso de teste, e (b) após a execução de um caso de teste

Exemplificando, a execução do nó 54.82 da Figura 4.9a requer a execução dos nós 0, 34, 43, e 54, e, em seguida os nós 91 e 97 também serão executados. Isso significa que os nós 0, 34, 43, 54, 91 e 97 serão cobertos pela execução de teste que executa o nó 54.82. Portanto, o nó 54.82 tem peso 7, pois cobri-lo implica na cobertura de no mínimo 7 nós ou mais e, isto dependerá do fluxo de controle no qual o teste irá executar.

Portanto, a Ferramenta JaBUTi consegue mostrar ao testador qual área do programa tem maior proporção de cobertura. Desta forma, contribui para a diminuição da geração de casos de teste e amplificação da cobertura do programa.

A seção seguinte apresenta de forma sucinta o processo de instrumentação que a JaBUTi realiza no código escrito em Java.

4.4 O Processo de Instrumentação da JaBUTi

A JaBUTi instrumenta o conjunto de instrução de *bytecode* de cada método das classes que foram escolhidas para serem instrumentadas. O método `Prober.probe()`, contido em cada nó do GFC, é o principal responsável por coletar informações dinâmicas, de fluxo de dados e de controle de cada nó, as quais são armazenadas em um arquivo de *trace*[4.9] para serem usadas na análise de cobertura e outras operações. Toda vez que o método `Prober.prober()`[4.10] é invocado, é salvo no arquivo de *trace* as seguintes informações, conforme (VINCENZI et al., 2003):

- **⟨*class_id*⟩:** é um número inteiro que identifica unicamente uma dada classe C;
- **⟨*method_id*⟩:** é um número inteiro que identifica unicamente um só método m de C;
- **⟨*block_id*⟩:** é um número inteiro que corresponde ao deslocamento da primeira instrução de *bytecode* de um bloco de GFC b de $C.m$. Os nós expandidos são identificados depois de pós-processar o arquivo de *trace*;
- **⟨*object_id*⟩:** é uma *string* que identifica o objeto sendo usado para executar $C.m$. Um valor nulo é utilizado para representar execução de um método estático;
- **⟨*thread_id*⟩:** é um inteiro que identifica a *thread* corrente em uso para executar $C.m$.

Vincenzi *et al.*, em (VINCENZI et al., 2003), implementa um carregador de classe instrumentador que estende o carregador de classe padrão, assim, durante a execução, os arquivos de classe a serem testados são identificados, instrumentados e executados. Os arquivos que não requerem instrumentação são entregues ao carregador de classe padrão para serem executados. Depois de cada execução, as informações de *trace* são acrescentadas no fim do arquivo de *trace*, caracterizando um novo caso de teste. O arquivo de *trace* é lido pela JaBUTi e as informações de cobertura são computadas para cada método.

[4.9] O arquivo de *trace* (rastreio) é gerado pela Ferramenta JaBUTi com a extensão `trc` e armazena os dados da execução do programa instrumentado (DELAMARO; VINCENZI; MALDONADO, 2004).

[4.10] O método `Prober.prober()` é invocado toda vez que a execução alcança a primeira instrução de um determinado nó do GFC (VINCENZI et al., 2003).

Na próxima seção são apresentadas as principais funcionalidades pertinentes a Ferramenta JaBUTi, o processo de criação de uma sessão de teste, as ferramentas de análise de cobertura e *slice*.

4.5 Funcionalidades e Sessão de Projetos de Teste

Segundo Vincenzi *et al.* (VINCENZI et al., 2003), um projeto de teste na JaBUTi é caracterizado por um banco de dados que armazena o arquivo da classe principal, o conjunto completo de classes necessárias para executar tal arquivo, o conjunto de classes que será instrumentado, e o conjunto de classes que não será. Esta informação é armazenada em um arquivo com extensão `.jbt` . Durante a execução de qualquer arquivo `.class` que pertence ao conjunto de classes a ser instrumentado de um determinado projeto, informações de fluxo de controle dinâmicas são coletadas e salvas em um arquivo separado que tem o mesmo nome do arquivo de projeto, mas com uma extensão diferente `.trc` (*trace*).

A JaBUTi (VINCENZI et al., 2003) é composta pela ferramenta de análise de cobertura e pela ferramenta *slice* descrita na subseção 4.5.2. A JaBUTi usa o *bytecode* Java, em vez do código fonte, para coletar as informação necessárias e executar suas atividades. Esta característica permite testar qualquer tipo de programa Java, incluindo componentes Java. Ela também permite o teste estrutural de componentes Java e a identificação de qual parte destes componentes precisam realizar mais testes, ou ainda, quais partes não foram cobertas.

Ao executarmos a Ferramenta JaBUTi é apresentada a tela do *Menu* Principal, conforme ilustrado na Figura 4.10.

A JaBUTi apresenta os seguintes *menus*:

- ***File*** **-** fornece opções para criar, abrir, fechar, salvar um projeto e para sair da ferramenta.
- ***Tools*** **-** fornece opções para escolher a ferramenta de cobertura ou ferramenta *slice*.
- ***Visualization*** **-** fornece opções para alterar a visualização em três modos diferentes: *bytecode*, GFC ou o código fonte.

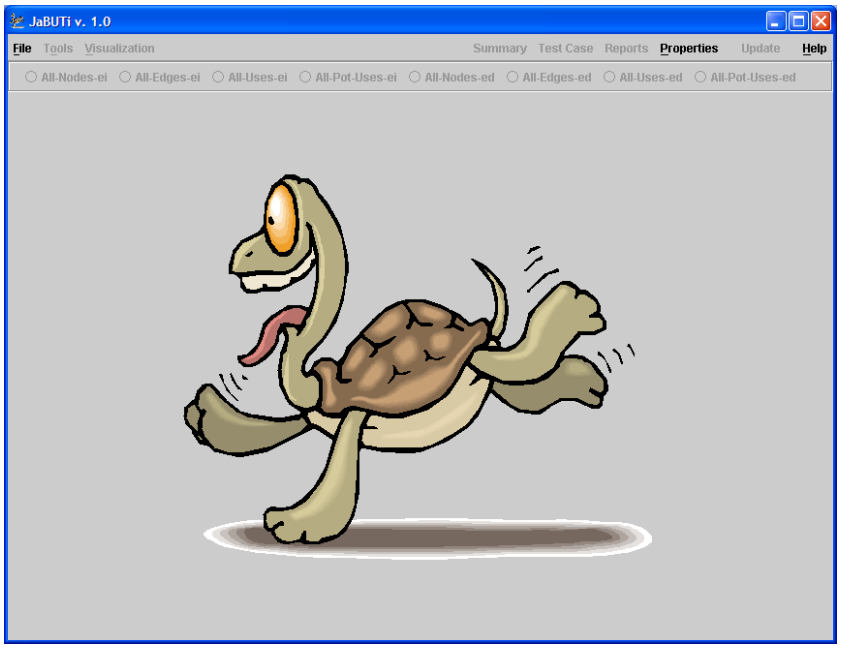

Figura 4.10: Tela Principal da Ferramenta JaBUTi

- ***Summary*** **-** fornece informações de cobertura em diferentes níveis: cobertura do projeto inteiro com respeito a cada critério de teste, cobertura por critério de teste com respeito a cada classe, cobertura por critério de teste com respeito a cada método.

- ***Test Case*** **-** mostra a cobertura por caso de teste, com respeito ao projeto inteiro, e permite habilitar ou desabilitar casos de teste.

- ***Properties*** **-** permite mudar o estilo da interface gráfica e checar as classes que compõem o projeto corrente.

- ***Update*** **-** indica que há casos de teste adicionais que podem ser importados no projeto de teste corrente.

- ***Help*** **-** mostra a ajuda *on-line* da ferramenta e o nome dos criadores da ferramenta.

Na JaBUTi, (VINCENZI et al., 2003), a atividade de teste requer a criação de um projeto de teste. Para criar um projeto é necessário acessar o *menu* ***File*** $\Rightarrow$ ***Open Class***, uma caixa de diálogo aparecerá, como na Figura 4.11. O testador seleciona o diretório e o arquivo da classe desejada, conforme ilustração da Figura 4.12, e a ferramenta automaticamente identifica o pacote que a classe pertence. O campo "*Classpath*" contém o conteúdo corrente da variável de ambiente `CLASSPATH` e possibilita a inclusão de caminhos adicionais. Esta tela é finalizada com um clique no botão OK.

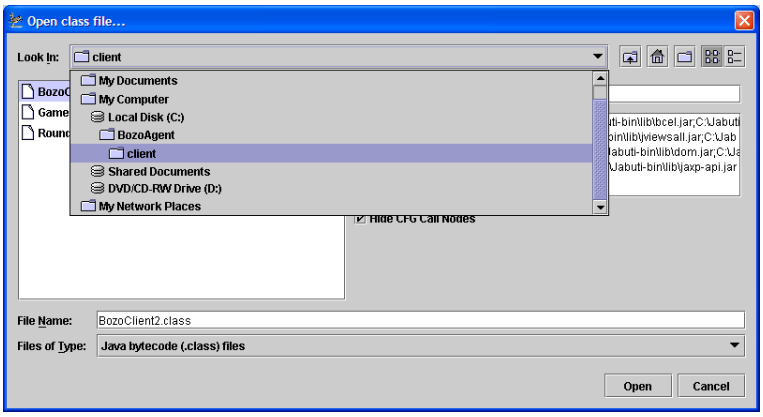

Figura 4.11: Caixa de Diálogo *Open Class*

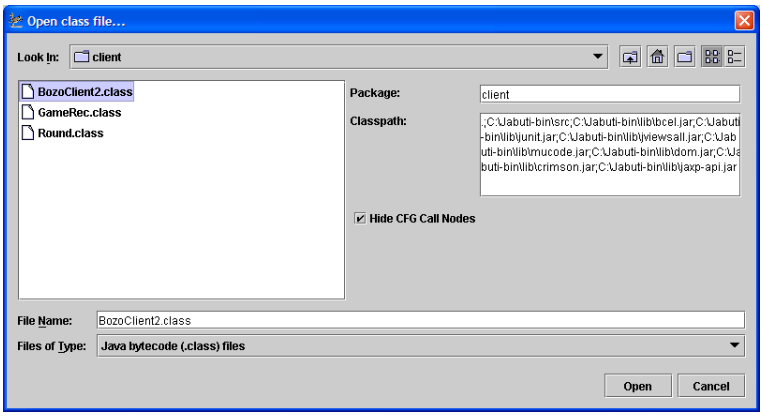

Figura 4.12: Selecionando o Arquivo de Classe

Em seguida, a tela do Gerenciador de Projeto é aberta conforme a Figura 4.13, através da seleção da classe a ferramenta identifica o conjunto completo de arquivos de classe do sistema e não-sistema necessários para executar a classe selecionada. O usuário, então, poderá selecionar qual arquivo de classe será testado e nomear o projeto criado, concluindo, o usuário clica no botão OK e o projeto é salvo.

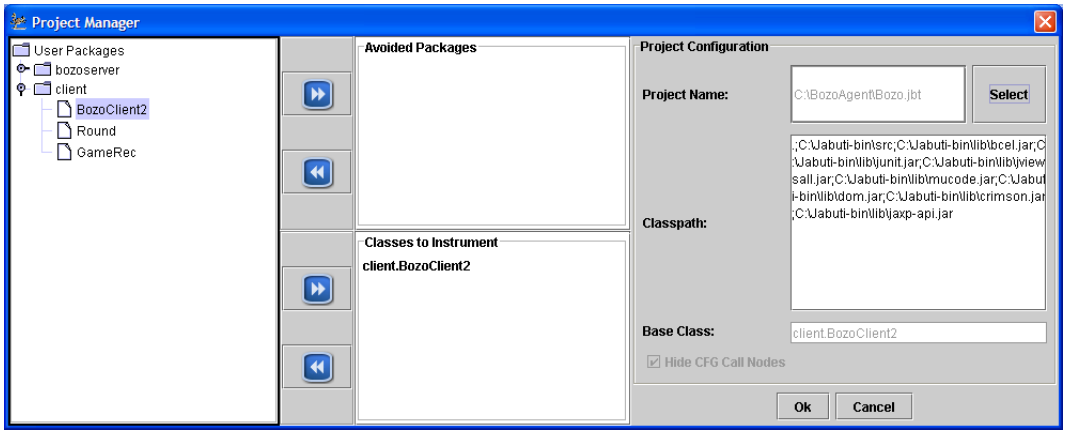

Figura 4.13: Selecionando os arquivos de classes a serem testados

Continuando, a tela da ferramenta de teste de cobertura é disponibilizada com a apresentação da classe selecionada, como ilustrado na Figura 4.14. Depois de ter criado um projeto, a JaBUTi fornece uma ferramenta de análise de cobertura e uma ferramenta *slice*, que serão descritas nas seções seguintes.

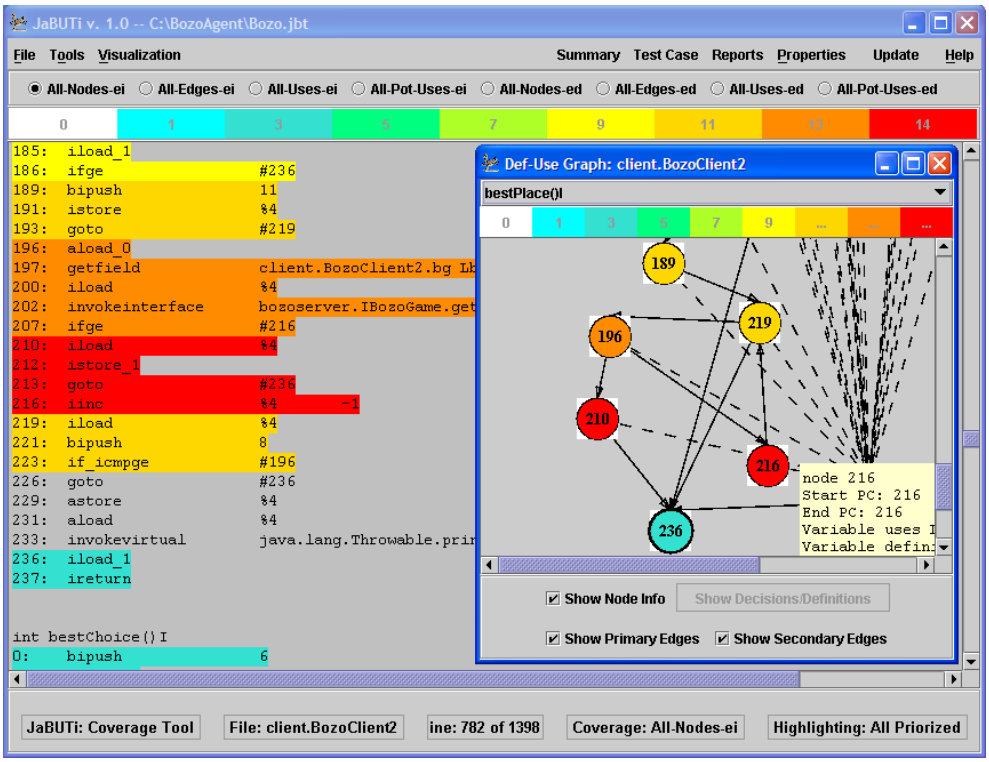

Figura 4.14: Tela da Ferramenta de Teste de Cobertura

As subseções a seguir apresentam a ferramenta de análise de cobertura e a ferramenta *slice*, ambas são oferecidas pela JaBUTi.

4.5.1 Ferramenta de Análise de Cobertura

Ao ser carregada a ferramenta (VINCENZI et al., 2003) mostra o *bytecode*, em vez do código fonte Java, pois este nem sempre pode estar disponível. A ferramenta usa diferentes cores para fornecer sugestões sobre qual parte do *bytecode*/código fonte/GFC deverá ser coberta primeiro para aumentar a cobertura. Tais cores são mostradas considerando o critério de blocos[4.11].

Apesar do código fonte não ser requisitado pela JaBUTi (VINCENZI et al., 2003),

[4.11] O Critério de blocos tem por objetivo gerar um caso de teste para cobrir uma das áreas com alta relevância, se possível, na frente de outras áreas com baixa relevância, deste modo, as áreas de baixa relevância são cobertas assim que a cobertura máxima possa ser executada.

quando este estiver disponível, ele também poderá ser exibido em cores, facilitando a identificação de qual parte do código deverá ser coberta primeiro.

Em (VINCENZI et al., 2003), a notação da ferramenta possui diferentes tipos de círculos para representar diferentes tipos de nós:

- **Círculos duplos** - representam nós de chamada.
- **Círculos em negrito** - representam nós de saída.
- **Círculos simples** - representam todos os outros nós restantes.

A ferramenta também apresenta notação para dois tipos diferentes de arestas para representar o fluxo de controle, conforme (VINCENZI et al., 2003):

- **Linha contínua** - representa fluxo de controle "normal".
- **Linhas tracejadas** - representa fluxo de controle de exceção.

A JaBUTi (VINCENZI et al., 2003) suporta a aplicação de três critérios de teste estruturais: todos-nós, todos-arcos e todos-usos. Dependendo de qual critério esteja ativado, o *bytecode*, o código fonte ou o GFC é colorido de um modo diferente. As cores relacionadas aos pesos fornecidos pela ferramenta referem à alta ou baixa cobertura, podendo serem vistos como sugestões para facilitar a geração de casos de teste, proporcionando uma cobertura maior com menos casos de teste.

Para avaliar a cobertura obtida, a ferramenta fornece relatórios de teste personalizados que podem ser acessados a partir do *menu* "*Sumary*" e do *menu* "*Test Case*", conforme dito anteriormente. Através destes relatórios, pode-se avaliar a cobertura de cada classe sob teste com diferentes granularidades e decidir se uma determinada classe requer testes adicionais.

A JaBUTi (VINCENZI et al., 2003) oferece suporte à avaliação de adequação de conjunto de caso de teste e fornece orientação na seleção dos mesmos. Se existe um conjunto de teste pronto, tal conjunto pode ser avaliado, isso permite verificar se todas as sentenças no componente foram executadas ou não por este conjunto de teste. Se não, casos de teste adicionais podem ser elaborados para melhorar a qualidade deste conjunto.

Por outro lado, se nenhum conjunto de teste estiver disponível, podem ser utilizadas as sugestões fornecidas pela própria ferramenta para elaborar o conjunto referente aos critérios de teste implementados pela JaBUTi.

Os critérios de teste, segundo Vincenzi *et al.* (VINCENZI et al., 2003), podem ser aplicados com incrementação, ou seja, se necessário, podem ser desenvolvidos casos de teste adicionais, e toda vez que critérios sejam adicionados, o botão "*Update*" na interface gráfica da JaBUTi torna-se vermelho, indicando que a informação de cobertura pode ser atualizada considerando os novos casos.

Durante a adição de um determinado caso de teste, deve ser verificado se a saída obtida está correta relacionada com a especificação. Se for detectada uma discrepância, a falha deve ser localizada e corrigida. Uma alternativa para localização de falhas é a ferramenta *Slice*.

4.5.2 Ferramenta *Slice*

A ferramenta *Slice* implementa uma simples heurística de busca de falhas, baseada em informação de fluxo de controle. A idéia é reduzir o prazo de pesquisa para a localização de falhas.

Utilizando a ferramenta *slice*, deve-se escolher, entre os casos de teste, aquele que cause uma falha e dois que não a revelem. Baseada nestas informações, a ferramenta destaca a parte do código mais provável de conter a falha. Após ter observado o caminho de execução do caso de teste da falha, pode-se habilitar um ou dois casos de teste adicionais bem-sucedidos, para que através da execução dos caminhos bem-sucedidos a ferramenta possa identificar os pedaços mais prováveis de conterem a falha, bem como os menos prováveis.

Segundo Vincenzi *et al.* em (VINCENZI et al., 2003), *slice* é uma técnica de análise de programa amplamente aplicável que pode ser usada para realizar diversas atividades de engenharia de *software*, tais como: compreensão de programa, depuração, teste, paralelização, re-engenharia, manutenção, e outras. No contexto da JaBUTi, as atividades mais importantes são compreensão de programa e teste.

A JaBUTi (VINCENZI et al., 2003) utiliza um critério simples de *slice* dinâmico, baseado nas informações de fluxo de controle, para identificar um subconjunto de sentenças

mais prováveis de conter o erro. Portanto, tem o intuito de:

1. Computar o conjunto de falha F_s de nós do GFC que inclui todas as sentenças executadas pelo caso de teste de falha;
2. Computar o conjunto de sucesso S_s de nós do GFC considerando um ou dois casos de teste de sucesso;
3. Calcular a diferença e a intersecção desses conjuntos para priorizar as sentenças executadas pelo caso de teste de falha.

Vincenzi *et al.* (VINCENZI et al., 2003) exemplifica considerando o GFC apresentado na Figura 4.6a, onde N é o conjunto de nós do GFC (N = 0, 15, 34, 43, 54, 54.82, 60, 60.82, 74, 74.82, 79, 91, 97). Assim, suponha que um caso de teste de falha atravesse os nós do GFC F_s = 0, 34, 15, 34, 43, 54, 54.82, 91, 97 e um caso de teste de sucesso atravesse os nós do GFC S_s= 0, 34, 43, 60, 60.82, 91, 97. As localizações mais prováveis da falha estão nos nós 15, 54 ou 54.82, uma vez que são executados pelo caso de teste de falha ($F_s \setminus S_s$). Se a falha não estiver localizada em tais nós, ela será encontrada nos nós do GFC 0, 34, 43, 91 e 97 ($Fs \cap Ss$). Tal procedimento, baseado em informações de fluxo de controle, facilita a atividade de teste já que todos os outros nós do GFC não precisam ser analisados. Assim, a ferramenta *Slice* facilita a detecção de falhas.

A próxima seção apresenta a adaptação realizada na Ferramenta JaBUTi para suportar o teste de agentes móveis.

4.6 Testando Agentes Móveis com a JaBUTi/MA

Segundo Delamaro, (DELAMARO, 2005), sob a perspectiva de teste estrutural, é mais trabalhoso testar uma aplicação de agente móvel do que um programa convencional. A dificuldade está no fato de que a coleta de informações de um programa convenvional é simples, ao contrário do teste de agentes móveis que requer uma forma para coletar dados de *trace* (rastreio) de vários pontos diferentes da rede, isto é, de todos lugares na rede onde o agente executa, e depois passá-los para a ferramenta de análise.

A JaBUTi/MA[4.12] é a extensão da JaBUTi para tratar de agentes móveis, usa a

[4.12] *Java Bytecode Understanding and Testing Tool for Mobile Agents*

estrutura de mobilidade fornecida pela μCODE para suportar a transferência de dados de *trace* do agente móvel sob teste (MAUT[4.13]) para a ferramenta de análise. Para garantir a comunicação entre o MAUT e a JaBUTi é requisitada a instalação do μCODE em cada *host* que consta no caminho do agente.

Não há mudanças entre a ferramenta de análise da JaBUTi e da JaBUTi/MA, pois ambas verificam constantemente se existem novos dados no arquivo de trace. Todavia, existem mudanças no processo de instrumentação da ferramenta (DELAMARO, 2005).

A Figura 4.15(a), obtida em (DELAMARO, 2005), mostra como funciona a JaBUTi/MA. Vale esclarecer que, em conjunto com a ferramenta de análise, existe um servidor de teste TμS[4.14] que é uma extensão do μServer[4.15], porém com alterações para realizar a instrumentação nas classes a serem testadas, recebe dados de *trace* de um agente móvel e os escreve no arquivo de *trace*.

Na Figura 4.15(b) é ilustrado um MAUT sendo instrumentado e executado em um determinado *host*. Conforme Delamaro, (DELAMARO, 2005), a instrumentação na JaBUTi/MA acopla uma classe `probe`[4.16] para o MAUT que sustenta trilhas de todos os pontos alcançados pelo agente, como na intrumentação convencional. Em adição, a classe `probe` armazena informações sobre para onde transferir informações de *trace*, isto é, para o endereço do TμS.

Durante a migração ou terminação do MAUT, a classe `probe` cria um novo agente de teste (TA[4.17]) para transferir informações de *trace* para o servidor de teste (TμS), conforme ilustração da Figura 4.15(c). A classe `probe` deve migrar com o MAUT juntamente com as informações sobre o TμS (DELAMARO, 2005).

Segundo Delamaro, (DELAMARO, 2005), a JaBUTi/MA provê dois diferentes modos de testar um agente. O primeiro é instrumentando o agente e, então, o agente já inicializa instrumentado. O segundo modo é instrumentar o agente somente quando ele chega em um determinado *host*, isto é, a intrumentação e a coleta de dados de *trace* são inicializados por um μServer.

Para realizar a atividade de teste na JaBUTi/MA, testando o agente no cliente,

[4.13] *Mobile Agent under Test*
[4.14] μServer de Teste
[4.15] μServer é o servidor que recebe e executa um agente móvel na plataforma μCODE (DELAMARO, 2005).
[4.16] na Figura 4.15 a classe probe é apresentada como classe-sonda
[4.17] *Test Agent*

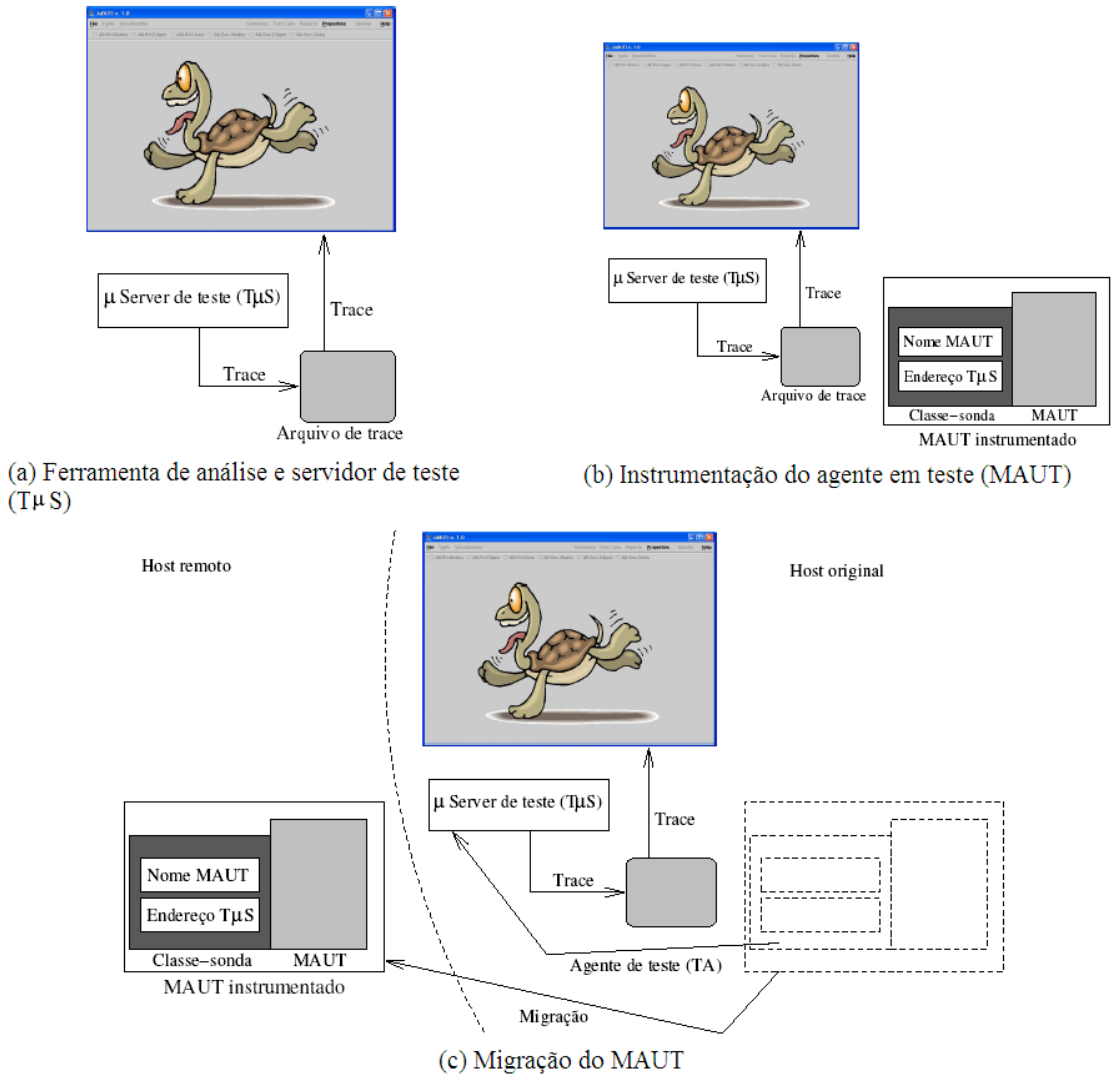

Figura 4.15: Funcionamento da JaBUTi/MA (DELAMARO, 2005)

Delamaro (DELAMARO; VINCENZI; MALDONADO, 2004) sugere a seqüência dos seguintes passos:

1. Crie um projeto de teste para o agente, normalmente como se faria para qualquer programa.

2. De dentro da própria JaBUTi deve-se instalar o servidor de teste que vai receber os dados da execução do agente nos servidores por onde ele passar instrumentado.

3. Executar o agente e colher os dados da sua execução intrumentando-o no cliente. Para tanto, todos os servidores necessários para executar a aplicação devem ser inicializados normamente. Então, deve-se executar o agente utilizando o programa `mobility.HostProberLoader`.

4. Então, o agente começa a executar normamente e os dados de *trace* são enviados ao servidor de teste. Caso o projeto do agente esteja aberto na ferramenta, tais dados são imediatamente processados. Nesse caso o botão "*Update*" fica vermelho, indicando que novos casos de teste chegaram a ferramenta. A cada vez que o agente migra para um novo *host*, um novo caso de teste é enviado ao servidor de teste.

No caso de realizar o teste do agente no servidor, Delamaro (DELAMARO; VINCENZI; MALDONADO, 2004) sugere:

1. Crie um projeto de teste para o agente, normalmente como se faria para qualquer programa.
2. De dentro da própria JaBUTi deve-se instalar o servidor de teste que vai receber os dados da execução do agente nos servidores por onde ele passar instrumentado.
3. Executar o agente instrumentando-o no servidor. Neste caso o agente será instrumentado somente quando chegar em algum μServer configurado para realizar a instrumentação.

Essa configuração é feita passando-se três parâmetros: o endereço do servidor de teste; o nome do agente (o mesmo nome que identifica o agente no servidor de teste) e as classes a serem instrumentadas.

Caso a aplicação móvel utilizar um servidor padrão, este pode ser instalado e parametrizado utilizando-se o programa `mucode.util.TestLauncher` (em vez de utilizar o `mucode.util.Launcher`) que aceita parâmetros na linha de comando para setar cada um destes argumentos.

O capítulo seguinte apresenta os estudos de casos implementados com o intuito de serem testados para validar o tema proposto por este trabalho. Para tanto, foram criados dois estudos de casos o primeiro é um jogo denominado Bozó, e o segundo é um aplicativo de conversação denominado *Mobile Chat*. Além dos estudos de casos criados, também são apresentadas as atividades de teste realizadas nos mesmos.

5 Estudos de Casos

Desenvolver aplicações para ambientes distribuídos é um trabalho complexo e pesquisas tem sido realizadas para melhorar o desenvolvimento e a execução de tais aplicações. A mobilidade de código é uma nova técnica para contribuir com o desenvolvimento e execução de sistemas distribuídos complexos. Isto se deve ao fato de que a mobilidade de código permite desenvolver sistemas mais configuráveis, customizáveis, flexíveis e escaláveis, tais características são essenciais para sistemas distribuídos (PICCO, 2000).

Várias linguagens têm sido propostas para implementar sistemas de código móvel, conforme citado no Capítulo 2. A API μCODE foi desenvolvida em linguagem Java. Diferente de outras linguagens propostas para suportar código móvel a μCODE, além de oferecer primitivas de suporte à mobilidade de código, permite que o desenvolvedor defina suas próprias primitivas de mobilidade, incluindo sua própria noção de agente móvel.

Este capítulo tem o objetivo de apresentar aplicações desenvolvidas em Java utilizando a API μCODE para alcançar a mobilidade do código, implementando o paradigma de agente móvel. Estas aplicações foram desenvolvidas com o intuito de serem testadas para evidenciar que a utilização do teste estrutural em código móvel é uma boa forma de comprovar a corretude do mesmo.

Para tanto, foram desenvolvidos e apresentados dois programas, o primeiro é um jogo denominado Bozó que implementa um cliente que é um agente móvel e um servidor com o qual o agente estabelece comunicação. O servidor gerencia o jogo fornecendo métodos que realizam as ações do jogo e aguarda a conexão de um cliente. As jogadas foram implementadas de forma automática, ou seja, é o próprio programa que assume a escolha de uma determinada jogada através de estratégias previamentes implementadas pelo programador.

O segundo exemplo é um aplicativo de conversação, semelhante ao ICQ[5.1], porém com menos funcionalidades, pois o intuito não estava na elaboração de um bom programa de *chat* e sim na construção de um aplicativo móvel. O *Mobile Chat* foi desenvolvido com o intuito de mostrar o exemplo de uma aplicação móvel mais complexa e de uso real. Este exemplo, diferente do primeiro, é mais robusto, pois foram implementados dois servidores, um fixo e outro móvel, e treze agentes móveis que estabelecem comunicação e se movimentam entre tais servidores. Como se não bastasse, o servidor móvel se movimenta pelos *hosts* clientes, pois este foi implementado de forma a realizar a migração autônoma baseada na verificação do melhor *host*, participante da sessão de *chat*, para alocar o servidor itinerante com o intuito de melhorar o tráfego da rede.

Os exemplos citados foram testados com o intuito de comprovar o tema proposto por este trabalho. Vale ressaltar que a depuração de aplicações móveis é especialmente difícil, pois, além de tratarem-se de aplicações distribuídas, são móveis, dificultando ainda mais o processo de teste. Em adição, também não se pode afirmar se uma eventual falha faz parte da aplicação ou do ambiente onde ela executa.

Para a realização das atividades de teste, apresentadas nas subseções 5.1.3 e 5.2.3, foi estabelecido que seria utilizada a seguinte metodologia: aplicação do teste funcional, aplicação do teste estrutural e comparação dos testes.

A comparação dos testes tem o intuito de verificar qual técnica de teste apresenta os melhores resultados. A intenção é mostrar que, o resultado desta comparação evidencie que, a cobertura da técnica estrutural é maior do que a funcional. Denotando que o teste estrutural é uma boa técnica para testar código móvel.

Para auxiliar a aplicação do teste estrutural nos exemplos citados foi utilizada a Ferramenta JaBUTi/MA. Os exemplos foram testados utilizando a simulação de um ambiente real, ou seja, os programas foram executados e testados em uma única máquina simulando *hosts* diferentes através da alocação de clientes e servidores em diferentes portas.

Este capítulo, portanto, apresenta dois estudos de casos. A Seção 5.1 apresenta o Jogo Bozó, suas características, forma de execução, os aspectos de implementação e a atividade de teste realizada no mesmo. Na Seção 5.2 é apresentado o aplicativo de comunicação *Mobile Chat*, suas funcionalidades, os aspectos de implementação e a atividade

[5.1] "*I Seek You*"

de teste realizada em tal exemplo.

5.1 Jogo Bozó

O objetivo desta seção é testar um estudo de caso que implementa um jogo de dados que tem por objetivo acumular o maior número de pontos, este é denominado de Bozó.

O Bozó utiliza cinco dados os quais poderão ser rolados no mínimo uma e no máximo três vezes para cada jogada, no final da jogada é marcado o valor da pontuação obtida. A realização desta pontuação se dá através de um placar onde são anotados os valores obtidos para cada item contido no mesmo. Os itens que compõem o placar são determinados através dos resultados obtidos com a rolagem dos dados, sendo estes:

- **ONES:** valor da soma dos números dos dados que tiverem o valor igual a 1.
- **TWOS:** valor da soma dos números dos dados que tiverem o valor igual a 2.
- **THREES:** valor da soma dos números dos dados que tiverem o valor igual a 3.
- **FOURS:** valor da soma dos números dos dados que tiverem o valor igual a 4.
- **FIVES:** valor da soma dos números dos dados que tiverem o valor igual a 5.
- **SIXES:** valor da soma dos números dos dados que tiverem o valor igual a 6.
- **3 OF KIND:** valor da soma dos números de 3 dados que tiverem o mesmo valor.
- **4 OF KIND:** valor da soma dos números de 4 dados que tiverem o mesmo valor.
- **FULL HAND:** vale 25 pontos, ocorre quando existem 3 dados com valor igual a x e 2 dados com valor igual a y.
- **SMALL SEQUENCE:** vale 30 pontos, ocorre quando existem 4 dados com valores seguidos a, b, c e d. Exemplo: 2, 3, 4 e 5.
- **LARGE SEQUENCE:** vale 40 pontos, ocorre quando existem 5 dados com valores seguidos a, b, c, d e e. Exemplo: 2, 3, 4, 5 e 6.
- **5 OF KIND:** vale 50 pontos, ocorre quando existem 5 dados com o mesmo valor.

- **CHANCE:** valor da soma dos números dos 5 dados, independente dos valores obtidos nos mesmos. Este item é uma chance que o jogador possui quando o resultado da jogada não é favorável para marcar qualquer pontuação no placar com exceção do item "CHANCE".

É importante salientar que quando não existe item para marcar o resultado obtido, isso caso a chance já tiver sido marcada, aí, então, ocorre um descarte. Para tanto, o jogador deve anular algum item que não esteja marcado. O jogo termina quando todos os itens forem marcados com valores e/ou descartes, ou seja, após treze jogadas. Aqui, o ganhador é aquele jogador que conseguir obter a maior pontuação.

No término do jogo, caso o valor do resultado do cálculo da soma dos itens de ONES até SIXES for igual ou maior que 63, é incrementada a pontuação somando-se mais 35 pontos, pois é um caso especial na qual ocorreu no mínimo a marcação de três dados iguais em cada item de ONES à SIXES.

Esta seção é composta por três subseções. Na Subseção 5.1.1 é apresentado como executar o jogo Bozó, na Subseção 5.1.2 são descritos os aspectos referente a implementação do Bozó e, por fim, na Subseção 5.1.3 é detalhada a atividade de teste realizada em tal jogo.

5.1.1 Execução do Jogo Bozó

Para executar o jogo Bozó é necessário primeiramente a execução de um servidor `BozoServer`, em seguida, de um cliente `BozoClient2`. Para tanto, deve-se inicializar o servidor que será ativado na porta padrão 1968, conforme ilustrado na Figura 5.1, da seguinte forma:

```
java bozoserver.BozoServerImpl
```

Em seguida, deve-se executar o cliente, que irá receber como parâmetro o *host* do servidor (endereço IP e porta 1968) e será inicializado no *host* local na porta padrão 2000, Figura 5.2, da seguinte forma:

```
java client.BozoClient2 localhost:1968 localhost
```

Ainda na execução do cliente, deve-se informar o nome do usuário, como na Figura 5.2. Então, o jogo é iniciado realizando as treze jogadas, após o término do jogo é apresentada na tela a pontuação obtida e cada jogada realizada, como ilustrado na Figura 5.3.

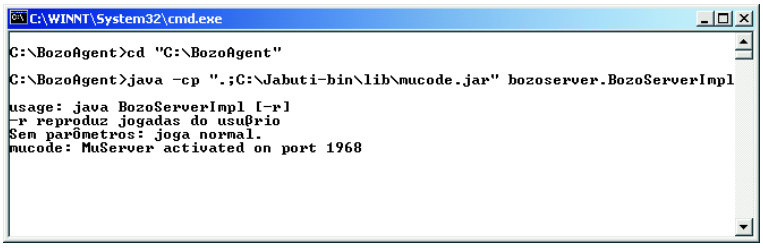

Figura 5.1: `BozoServer` iniciado.

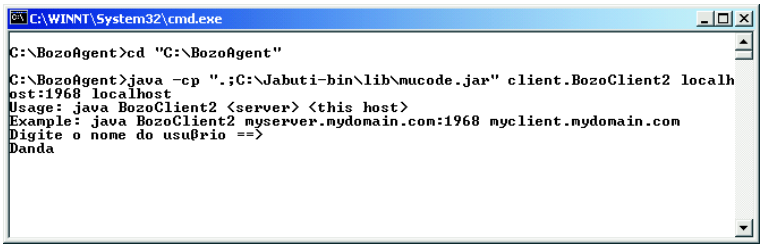

Figura 5.2: `BozoClient2` iniciado

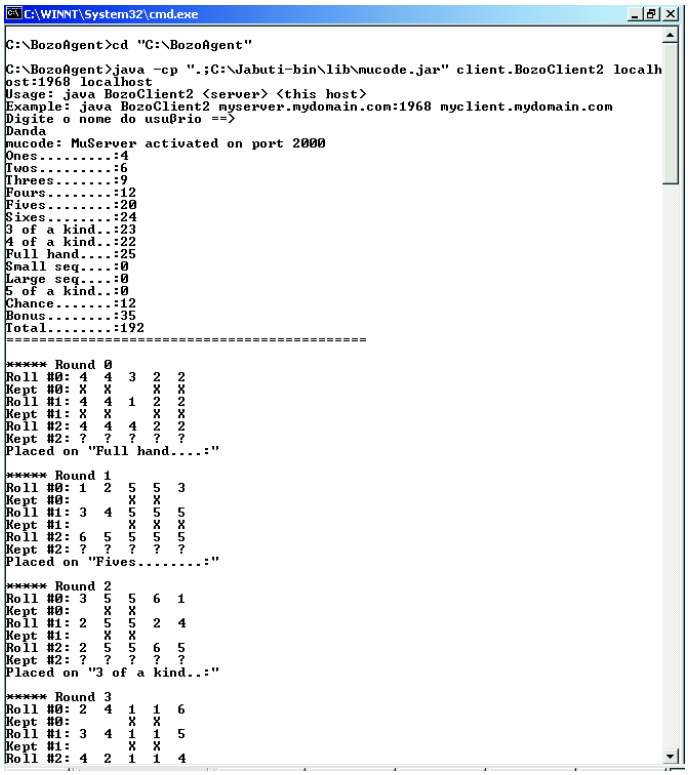

Figura 5.3: Resultado da execução do `BozoClient2`

Após a execução do cliente é criado um diretório com o mesmo nome do jogador, e neste é armazenado dois arquivos, um referente a todos os dados obtidos durante o jogo e outro referente à pontuação obtida em cada item do placar. Tal diretório e arquivos estão ilustrados na Figura 5.4.

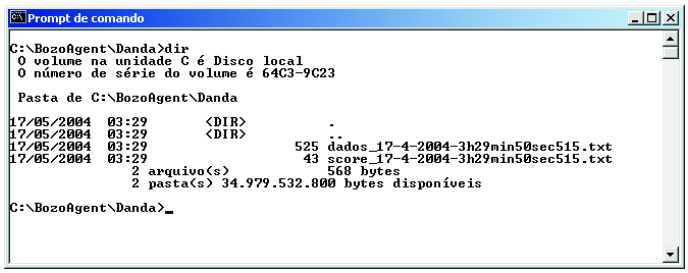

Figura 5.4: Diretório e arquivos gerados pelo `BozoClient2`

5.1.2 Aspectos de Implementação - Jogo Bozó

O jogo Bozó gera o resultado da rolagem dos dados de forma aleatória, também possui uma estratégia implementada para realizar automaticamente a escolha dos dados e a marcação do placar.

No jogo Bozó tem-se a implementação de um cliente que utiliza um servidor para gerenciar o jogo. As Figuras 5.5, 5.6 e 5.7 ilustram os pacotes, as classes e as interfaces pertinentes ao jogo. Estas ilustrações foram criadas utilizando-se a modelagem orientada a objetos UML[5.2], conforme (BOOCH; RUMBAUGH; JACOBSON, 2000).

Figura 5.5: Pacotes que compõem o Bozó

O servidor `BozoServerImpl`, conforme ilustrado na Figura 5.6, fornece os métodos que efetuam as ações do jogo, tais como rolar dados, permitir que o cliente selecione quais dados serão mantidos para a próxima rolada e manter os itens do placar atualizado após cada jogada. O servidor, também, aguarda a conexão de um cliente e suas respectivas requisições para, então, realizar as operações desejadas. A Figura 5.6 também apresenta a

[5.2] *Unified Modeling Language*

interface `IBozoGame` que possui constantes que referenciam o placar e métodos que obtem os valores para cada item do placar.

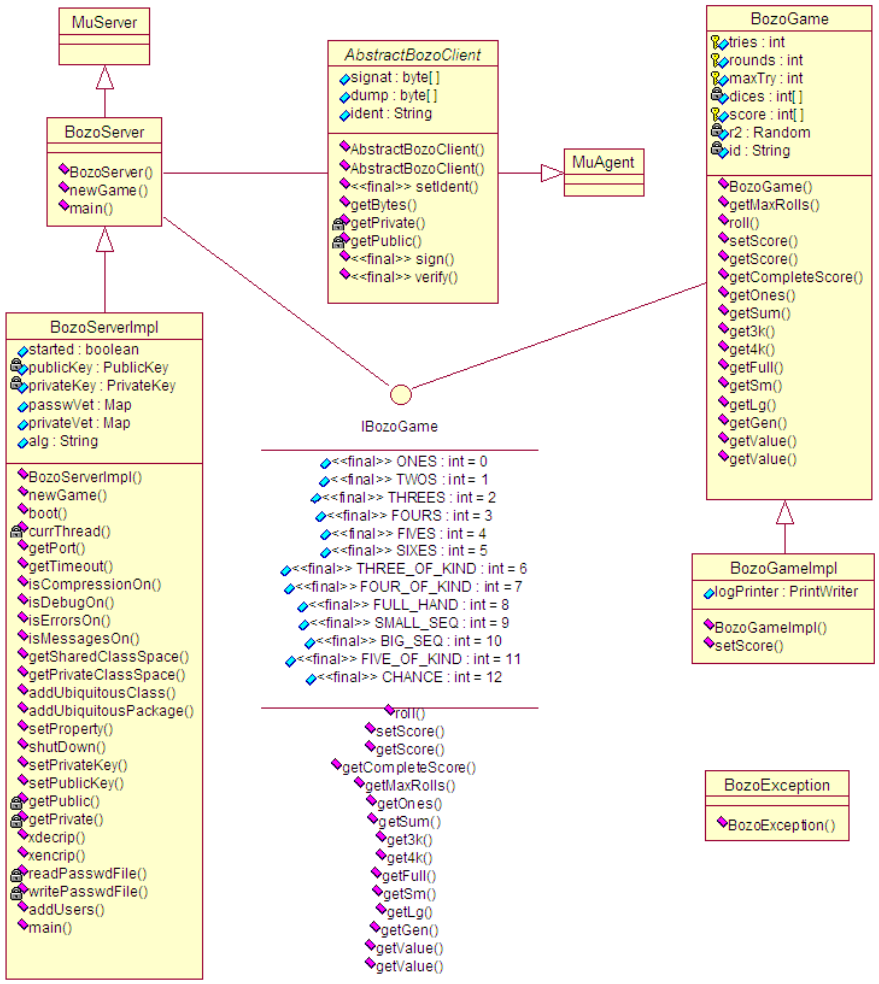

Figura 5.6: Classes que compõem o pacote `bozoserver`

O cliente `BozoClient2`, ilustrado na Figura 5.7, é um agente móvel que, quando executado, migra seu código para o servidor e lá executa o corpo do método `run()`, contendo as chamadas e definições de métodos do cliente e do servidor, tais chamadas são utilizadas para migrar o cliente e para realizar o funcionamento do jogo.

O método `playGame()`, contido na classe `BozoClient2`, conforme Figura 5.7, é responsável pela realização do jogo e tem por função: controlar quais dados serão mantidos para as próximas roladas de dados e determinar em qual placar a pontuação será marcada.

A marcação dos itens que compõem o placar é realizada através de um algoritmo que seleciona a primeira posição livre encontrada no vetor booleano `gone[]`. Este vetor contém treze posições inicializadas como `false`, indicando que nenhum item do placar foi

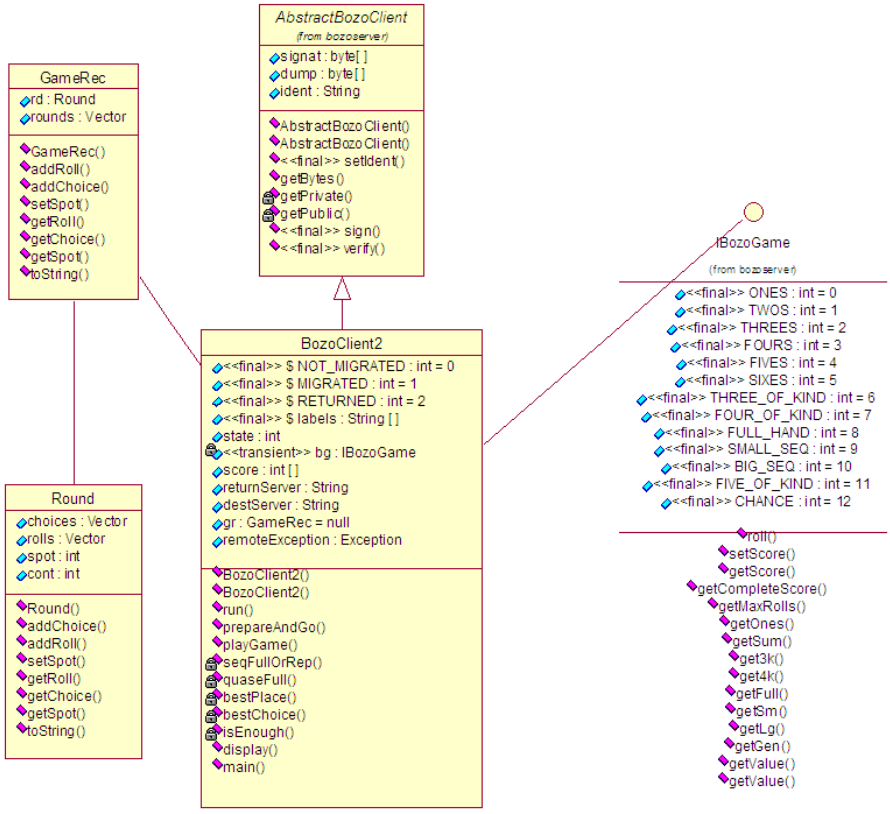

Figura 5.7: Classes que compõem o pacote `bozoclient`

marcado. Quando um item é marcado, a respectiva posição é configurada para `true`. A posição de cada item do placar é identificada no vetor `gone[]` conforme a Tabela 5.1.

Quando os dados são rolados, o algoritmo verifica a freqüência dos valores de cada face dos dados para avaliar em qual item do placar será atribuída a pontuação. Esta atribuição se dá através das estratégias implementadas pelo programador.

5.1.3 Atividade de Teste - Jogo Bozó

Após compreender a execução e a implementação do Bozó, o interesse está em realizar a atividade de teste para obter resultados satisfatórios com relação ao teste de código móvel. Para tanto, foi desenvolvido o seguinte plano de teste:

- **1ª Fase -** Criação do conjunto de teste T_{func}, adequado a requisitos funcionais;

Tabela 5.1: Valor das posições dos itens do placar no vetor `gone[]`

Posição	Placar
0	ONES
1	TWOS
2	THREES
3	FOURS
4	FIVES
5	SIXES
6	THREE_OF_KIND
7	FOUR_OF_KIND
8	FULL_HAND
9	SMALL_SEQ
10	BIG_SEQ
11	FIVE_OF_KIND
12	CHANCE

- **2ª Fase -** Criação do conjunto de teste T_{estr}, adequado a critérios estruturais;
- **3ª Fase -** Medição da cobertura dos requisitos funcionais, pelo conjunto de casos de teste T_{estr};
- **4ª Fase -** Medição da cobertura dos critérios estruturais, pelo conjunto de casos de teste T_{func};
- **5ª Fase -** Avaliação dos resultados obtidos.

Vale ressaltar que na realização do teste estrutural será utilizada, como apoio, a Ferramenta JaBUTi. O intuito, aqui, está em mostrar a capacidade do teste estrutural como meio para realizar a análise de corretude de um programa móvel escrito em java e, em contrapartida, mostrar que a adaptação da Ferramenta JaBUTi, para testar agentes móveis, foi implementada com êxito.

Esta Subseção é dividida em três partes onde constam as três fases da atividade de teste, o item 5.1.3.1 consta a aplicação do teste funcional, apresentando a especificação dos requisitos funcionais, os casos de testes gerados e a cobertura dos requisitos funcionais especificados. O item 5.1.3.2 consta a aplicação do teste estrutural, apresentando a especificação dos critérios a serem analisados, os casos de testes gerados e a cobertura dos critérios estruturais. Por fim, no item 5.1.3.5 é apresentado o resultado da comparação dos testes aplicados (funcional *versus* estrutural).

5.1.3.1 Bozó - 1ª Fase - Cobertura dos Requisitos Funcionais por T_{func}

Aqui o intuito está em averiguar se as características do agente móvel são realizadas, ou seja, se as funções que o `BozoClient2` deve desempenhar estão funcionando corretamente. Para tanto, será aplicado o teste funcional, analisando quais requisitos funcionais do agente móvel foram cobertos.

Dando início ao teste foram especificados 24 requisitos funcionais (Apêndice A) para o jogo Bozó. Os requisitos funcionais foram especificados com base nas estratégias implementadas pelo próprio programador, por exemplo, o requisito de número 1 implementa uma estratégia que "se saírem randomicamente cinco dados iguais e o placar FIVE_OF_KIND estiver livre, o programa marca-o e finaliza a jogada".

Segundo Pressman (PRESSMAN, 1995), o teste funcional se concentra no domínio de informação, procurando responder as seguintes perguntas:

- Como a validade funcional é testada?
- Quais classes de entrada constituirão bons casos de teste?
- O sistema é sensível a certos valores de entrada?
- Como as fronteiras de uma classe são isoladas?
- Quais índices de dados e volume o sistema pode tolerar?
- Que efeito terão combinações específicas de dados sobre a operação do sistema?

Respondendo as perguntas de Pressman, a validade funcional será testada através da avaliação dos resultados de cada jogo, identificando, através da cobertura dos requisitos funcionais especificados, quais estratégias foram adotadas durante a realização do jogo.

Quanto a questão das classes de entrada, estas são constituídas pelo valor da face de cinco dados e pela escolha da marcação do item do placar. Porém, devido ao jogo Bozó ser randômico não há como estabelecer os valores a serem apresentados por cada face de dado, nem como realizar a escolha de uma determinada estratégia e a marcação de um determinado item do placar, ou seja, é o programa que rola os dados randomicamente, escolhe a estratégia e marca o placar automaticamente, então, para tal caso fica impossível gerar de forma intencional os dados de entrada.

A natureza randômica do exemplo impossibilita manter o controle total sobre a atividade de teste, impedindo verificar a sensibilidade do programa a certos valores de entrada, isolar fronteiras de uma classe, verificar a tolerância à índice de dados e averiguar os efeitos obtidos através de combinações de dados, conforme sugestão de Pressman.

Assim, para avaliar o teste funcional foi estabelecido que através das estratégias fossem gerados os requisitos funcionais e que, então, fossem gerados os casos de teste para conseguir a cobertura dos requisitos.

Para a realização do teste funcional, foram gerados 150 casos de teste[5.3] e destes apenas 5 obtiveram sucesso na cobertura dos requisitos funcionais. Tais casos de teste foram os de número 1, 2, 3, 54 e 61 (Apêndice B) em ordem de execução, os demais casos de testes foram excluídos pois não obtiveram êxito na cobertura. Portanto, deixa-se claro aqui que existe um conjunto $T_{func} = \{1, 2, 3, 54, 61\}$ que contém os casos de teste selecionados para testar os requisitos funcionais do jogo Bozó.

Em seguida, é realizada a análise de cobertura dos requisitos funcionais através da aplicação dos casos de teste no jogo Bozó. Analisando o **caso de teste de número 01**, detalhes no Apêndice B, verificou-se que foram cobertos 14 dos 24 requisitos funcionais, sendo os de número: 01, 08, 09, 11, 12, 13, 15, 16, 17, 19, 20, 21, 22 e 24. Tal cobertura se deu conforme segue:

- No `round 0` foram satisfeitos os requisitos 13 e 01, no qual foram selecionados na primeira rolada três dados com o valor 4 (requisito 13), na segunda rolada conseguiu-se, então, quatro dados com o mesmo valor anterior e, na terceira rolada o *round* é finalizado com cinco dados com o valor 4 marcando a opção FIVE_OF_KIND (requisito 01).

- O `round 1` cobriu os requisitos 12, 11 e 9, onde foram selecionados dois dados com valores iguais a 6 (requisito 12), na segunda rolada encontrou-se dois pares de dados sendo um com o valor 6 e a outro com o valor 5 (requisito 11). Em seguida, na terceira rolada obteve-se mais um dado com o valor 6, fechando o *round* com três dados iguais a 6 e dois iguais a 5 marcando-se a opção FULL_HAND do placar (requisito 9).

- No `round 2` foram cobertos os requisitos 12 e 16, no qual a primeira e a segunda

[5.3] Tais casos de testes foram gerados através da execução aleatória de vários jogos, portanto, não foram previamente elaborados. Vale esclarecer que cada jogo é um caso de teste.

rolada cobriram novamente o requisito 12 e a terceira rolada apresentou quatro dados iguais com valor igual à 5, marcando, então, o item FIVES do placar (requisito 16).

- No `round 3`, foram satisfeitos os requisitos 12 e 19, a primeira e segunda rolada cobriram o requisito 12, onde dois dados apresentam repetição do valor 6, por fim, na terceira rolada obteve-se um dado com valor 1 e dois dados com valor 4, sendo interessante marcar apenas o item CHANCE do placar (requisito 19).

- No `round 4` foram cobertos os requisitos 12 e 17, no qual na primeira e na segunda rolada saíram três dados com valor igual à 5 cobrindo novamente o requisito 12 e, a terceira rolada continuou apresentando somente três dados com valor 5, marcando-se, então, o item THREE_OF_KIND do placar (requisito 17).

- O `round 5` cobriu novamente os requisitos 13 e 16, no qual foram selecionados na primeira rolada dois dados com valor igual a 2 cobrindo o requisito 13, na segunda rolada conseguiu-se três dados com o valor 2 e, na terceira rolada fechou-se o *round* com três dados com o valor 2 marcando a opção TWOS do placar, cobrindo o requisito 16.

- Nos `rounds 6`, `7` e `8` como no `round 5` foram satisfeitos os requisitos 13 e 16, marcando-se, respectivamente, os itens ONES, FOURS e SIXES do placar.

- No `round 9` foram cobertos os requisitos 12 e 22, no qual na primeira e na segunda rolada saíram três dados com valor igual à 5 cobrindo o requisito 12 e, na terceira rolada os valores não diferenciaram, porém, marcou-se o item FOUR_OF_KIND do placar (requisito 22). Aqui o item FOUR_OF_KIND foi descartado, ou seja, a pontuação foi igual à zero.

- O `round 10` cobriu os requisitos 21, 13 e 16, pois na primeira rolada nenhum dado foi selecionado (requisito 21), na segunda rolada selecionou-se um dado com o valor 3 cobrindo o requisito 13 e na terceira rolada continuou com apenas um dado com o valor 3 cobrindo novamente o requisito 16, marcando, portanto, o item THREES.

- O `round 11` cobriu os requisitos 21, 15 e 20, pois, como no *round* anterior, na primeira rolada nenhum dado foi selecionado (requisito 21), na segunda rolada selecionou-se o dado com o maior valor (requisito 15), na terceira rolada os valores apresentados não satisfizeram os placares disponíveis, assim, marcou-se o descarte do item LARGE_SEQ (requisito 20).

- O `round 12` satisfez os requisitos 24 e 08, no qual na primeira e segunda rolada foram selecionados dois dados (valores 3 e 4) do início de uma pequena seqüência de quatro valores com a ausência do valor 2 (requisito 24), e na terceira rolada verificou-se que não houve aumento da seqüência dos dados, então, foi marcado o item SMALL_SEQ (requisito 08).

Continuando a avaliação dos casos de teste funcionais, serão analisados somente os requisitos que ainda não foram cobertos. Portanto, foi verificado que o **caso de teste de número 02**, cobriu três novos requisitos: 06, 10 e 23. Esta cobertura foi verificada através da seguinte análise:

- O `round 1` cobriu na primeira rolada o requisito 10 que ainda não havia sido coberta. Obteve-se uma seqüência de dados, porém nesta sequência saiu o valor 2, então, manteve-se três dados do início da seqüência e rolaram os restantes (requisito 10). Na segunda e terceira rolada do mesmo round, os requisitos 13 e 19 foram satisfeitos novamente.
- No `round 9` foi satisfeito o requisito 6, no qual na primeira rolada dos dados foi satisfeito o requisito 13 e na segunda rolada já obteve-se a marcação do item BIG_SEQ (requisito 6).
- O item BÔNUS do placar foi marcado (requisito 23).

O **caso de teste 03** contribuiu para cobrir apenas um único requisito, o de número 14. No `round 7` foi coberto já na primeira rolada tal requisito, pois não havia repetição nos valores dos dados e foi selecionado o dado de maior valor (requisito 14).

O próximo caso executado com êxito foi o **caso de teste 54**, tal caso cobriu o requisito 7 no `round 10`, no qual na terceira rolada conseguiu-se obter uma seqüência grande, mas o placar para *Big Sequence* já havia sido marcado no *round* 1, então, foi marcado o placar SMALL_SEQ que estava livre (requisito 7).

No **caso de teste 61** foi coberto unicamente o requisito 4 no `round 4`, no qual na terceira rolada obteve-se cinco dados com o mesmo valor, porém o item *Five of kind* já havia sido marcado no *round* 2, então, foi marcado o item FULL_HAND (requisito 4).

Após várias tentativas, ainda faltou cobrir os requisitos 2, 3, 5, e 18. Com o intuito de cobrir tais requisitos deu-se continuidade a geração dos casos de testes, porém mesmo

com mais de 150 casos de teste gerados a cobertura de tais requisitos não foi alcançada. Vale lembrar que os casos de testes que não obtiveram êxito foram descartados, pois não houve razão de apresentá-los.

Analisando os requisitos 2, 3, 5 e 18, especificados no Apêndice A, verifica-se que eles são muito particulares e a probabilidade de saída de dados aleatória para cobrir tais requisitos é mínima. Isto se deve ao fato do jogo Bozó rolar os dados de forma randômica, portanto, dificulta-se a geração de casos de teste específicos para cobrir tais requisitos. Diante disto, os requisitos 2, 3 , 5 e 18 foram marcados como não executáveis.

A Tabela 5.2 apresenta a cobertura dos requisitos funcionais do jogo Bozó, com a exclusão dos quatro requisitos não executáveis. Vale esclarecer que a Tabela 5.2 é composta por quatro colunas: a primeira mostra o número dos casos de teste de sucesso, a segunda apresenta o número dos requisitos funcionais que foram cobertos com a execução de um determinado caso de teste, a terceira coluna mostra o "total de requisitos cobertos" do "total de requisitos existentes" e a quarta coluna apresenta a porcentagem de requisitos cobertos por requisitos existentes.

Tabela 5.2: Bozó - Cobertura dos Requisitos Funcionais por T_{func}

Caso de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
1	01, 08, 09, 11, 12, 13, 15, 16, 17, 19, 20, 21, 22 e 24	14 de 20	70%
2	06, 10 e 23	17 de 20	85%
3	14	18 de 20	90%
54	07	19 de 20	95%
61	04	**20 de 20**	**100%**

Analisando a Tabela 5.2 pode-se verificar que os casos de teste do conjunto T_{func} (Apêndice B) obtiveram 100% de cobertura dos requisitos funcionais, conforme desejado.

A primeira fase desta atividade de teste, além de ter sido cansativa, foi custosa devido à verificação de cada rolagem de dados em conjunto com a avaliação dos requisitos funcionais especificados. O tempo de **duração da primeira fase do teste** para o jogo Bozó **foi de 21 horas e 30 minutos**, sendo que 2h30min foram utilizadas para gerar os 150 casos de teste, 17h40min para avaliar tais casos e 1h20min para gerar uma documentação simples. Isto com exceção do tempo utilizado para o entendimento do estudo de caso e especificação dos requisitos funcionais que levou cerca de 16 horas.

5.1.3.2 Bozó - 2ª Fase - Cobertura dos Critérios Estruturais por T_{estr}

Prosseguindo com a atividade de teste do jogo Bozó, o objetivo é avaliar a cobertura dos critérios estruturais. Desta forma, foi especificado o seguinte plano para o teste estrutural:

1. Especificar os critérios estruturais a serem testados.
2. Gerar casos de teste para testar os critérios estruturais.
3. Aplicar os casos de teste para cobrir 100% dos critérios estruturais especificados.

Para realizar tal teste foi criado um projeto de teste na JaBUTi, em seguida, foi configurado e inicializado o servidor de teste da mesma, depois o servidor do Bozó foi executado através do seguinte comando:

```
java -cp ".;C:\Jabuti-bin\lib\mucode.jar" bozoserver.BozoServerImpl
```

Após a execução do servidor, a aplicação cliente do Bozó foi inicializada pelo comando:

```
java -cp ".;c:\jabuti-bin;c:\jabuti-bin\lib\bcel.jar;
c:\jabuti-bin\lib\junit.jar;c:\jabuti-bin\lib\jviewsall.jar;
c:\jabuti-bin\lib\mucode.jar;c:\jabuti-bin\lib\dom.jar;
c:\jabuti-bin\lib\crimson.jar;c:\jabuti-bin\lib\jaxp-api.jar"
mobility.HostProberLoader -P testBozo2.jbt -o xxx.jar -name bozoAgent
-h localhost:1988  client.BozoClient2 localhost:1968 localhost
```

O comando para inicializar a aplicação cliente, é composto pelas variáveis de ambiente que a Ferramenta JaBUTi precisa para estabelecer conexão com o código a ser testado, através da classe `HostProberLoader` do pacote `mobility`. O comando `-P` é usado para informar o nome do arquivo do projeto de teste, que neste caso é `testBozo2.jbt`; `-o` para informar o arquivo que contém as classes instrumentadas do exemplo a ser tratado; `-name` para informar o nome do agente móvel; `-h` para informar o *host* do servidor de teste da JaBUTi e, por fim, é informada a chamada de execução da classe e seus argumentos.

Ao iniciar o teste estrutural foi determinado que seriam avaliados os seguintes critérios estruturais, que a JaBUTi contempla: Todos-Nós, Todas-Arestas, Todos-Usos e Todos-Pontenciais-Usos.

Em seguida, foram gerados mais de 150 casos de teste e destes apenas 19 cobriram um novo critério estrutural. Tais casos de teste foram os de número 01, 02, 03, 04, 05, 06, 07, 08, 11, 13, 16, 21, 29, 31, 37, 38, 66, 68 e 87 em ordem de execução, apresentados no Apêndice C, os demais casos de testes que não obtiveram êxito na cobertura foram excluídos, pois não houve o porquê de apresentá-los.

Vale esclarecer que, existe um conjunto $T_{estr} = \{01, 02, 03, 04, 05, 06, 07, 08, 11, 13, 16, 21, 29, 31, 37, 38, 66, 68, 87\}$ que contém os casos de teste selecionados para testar os critérios estruturais pertinentes ao agente móvel `BozoClient2`.

A Tabela 5.3 apresenta a cobertura dos critérios estruturais com relação aos 19 casos de teste que obtiveram êxito. Tais informações foram capturadas através dos relatórios emitidos pela própria Ferramenta JaBUTi.

Vale elucidar que, na Tabela 5.3 a primeira coluna refere-se aos números dos casos de testes selecionados; a segunda coluna (Todos-Nós) é uma mescla de duas colunas, primeiro apresenta a "quantidade de item coberto" do "total de itens existentes" e segundo apresenta a "porcentagem da cobertura obtida"; e as demais colunas destinam-se, respectivamente, aos critérios Todas-Arestas, Todos-Usos e Todos-Potenciais-Usos.

Com base na Tabela 5.3 pode-se verificar que através destes 19 casos de teste gerados com sucesso houve a cobertura de 100% dos requisitos estruturais executáveis.

A **duração da segunda fase** desta atividade de teste **foi de 9 horas e 40 minutos**, sendo que 6 horas foram utilizadas para gerar os 150 casos de teste, 2h30min para avaliar tais casos e 1h10min para gerar uma documentação simples. Para auxiliar esta atividade de teste foi utilizada a ferramenta de análise de cobertura da JaBUTi/MA.

5.1.3.3 Bozó - 3ª Fase - Cobertura dos Requisitos Funcionais por T_{estr}

Com base nos conjuntos de casos de teste T_{func} e T_{estr} foi estipulado uma troca na cobertura, ou seja, agora o interesse está em avaliar a aplicação do conjunto T_{estr} para cobrir os requisitos funcionais. Vale lembrar que o conjunto T_{func} cobriu 100% dos

Tabela 5.3: Bozó - Cobertura dos Critérios Estruturais por T_{estr}

Caso de Teste	Todos-Nós		Todas-Arestas		Todos-Usos		Todos-Potenciais-Usos	
1	161 de 176	91%	195 de 233	84%	410 de 527	78%	1264 de 1673	75%
2	162 de 176	92%	200 de 233	86%	430 de 527	81%	1338 de 1673	80%
3	164 de 176	93%	206 de 233	88%	452 de 527	86%	1397 de 1673	83%
4	-	-	-	-	-	-	1399 de 1673	84%
5	-	-	-	-	453 de 527	86%	1406 de 1673	84%
6	168 de 176	95%	215 de 233	92%	477 de 527	90%	1446 de 1673	86%
7	172 de 176	98%	221 de 233	95%	485 de 527	92%	1476 de 1673	88%
8	173 de 176	98%	224 de 233	96%	493 de 527	93%	1496 de 1673	89%
11	174 de 176	99%	227 de 233	97%	505 de 527	96%	1508 de 1673	90%
13	-	-	-	-	-	-	1510 de 1673	90%
16	-	-	-	-	506 de 527	96%	1514 de 1673	90%
21	175 de 176	99%	229 de 233	98%	510 de 527	97%	1522 de 1673	91%
29	-	-	-	-	-	-	1531 de 1673	91%
31	-	-	230 de 233	99%	511 de 527	97%	1543 de 1673	92%
37	**176 de 176**	**100%**	232 de 233	99%	515 de 527	98%	1561 de 1673	93%
38			**233 de 233**	**100%**	519 de 527	98%	1595 de 1673	95%
66					523 de 527	99%	1633 de 1673	98%
68					**527 de 527**	**100%**	1659 de 1673	99%
87							**1673 de 1673**	**100%**

(-) não houve alteração na cobertura

requisitos funcionais.

Assim, foram aplicados os casos de testes do conjunto T_{estr} (Apêndice C), destes os casos de número: 01, 02, 03, 04, 06 e 31 foram suficientes para cobrir 100% dos requisitos funcionais do Bozó. Tal cobertura é apresentada na Tabela 5.4.

Vale mostrar as tabelas referentes à cobertura dos requisitos funcionais com relação a cada critério estrutural.

Com base na Tabela 5.3 foi selecionado os casos de teste que cobriram apenas o critério Todos-Nós, sendo estes os de número 01, 02, 03, 06, 07, 08, 11, 21 e 37. Assim, tem-se um conjunto de casos de teste, derivado do conjunto T_{estr}, representado por $T_{estr\_nós} = \{01, 02, 03, 06, 07, 08, 11, 21, 37\}$.

A Tabela 5.5 apresenta a cobertura dos requisitos funcionais com relação ao

Tabela 5.4: Bozó - Cobertura dos Requisitos Funcionais por T_{estr}

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
1	01, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20 e 21	14 de 20	70%
2	22 e 23	16 de 20	80%
3	06	17 de 20	85%
4	07	18 de 20	90%
6	24	19 de 20	95%
31	04	**20 de 20**	**100%**

Tabela 5.5: Bozó - Cobertura dos Requisitos Funcionais por $T_{estr\_nós}$

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
1	01, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20 e 21	14 de 20	70%
2	22 e 23	16 de 20	80%
3	06	17 de 20	85%
6	07 e 24	19 de 20	95%
7	-	19 de 20	95%
8	-	19 de 20	95%
11	-	19 de 20	95%
21	-	19 de 20	95%
37	-	19 de 20	95%

(-) não houve alteração na cobertura

conjunto $T_{estr\_nós}$ na qual obteve-se uma cobertura final de 95% dos requisitos funcionais.

Dos casos de teste do conjunto T_{estr} que cobriram o critério Todas-Arestas (conforme Tabela 5.3) tem-se $T_{estr\_arestas} = \{01, 02, 03, 06, 07, 08, 11, 21, 31, 37, 38\}$, destes os casos de teste de número: 01, 02, 03, 06 e 31 foram o suficiente para cobrir 100% dos requisitos funcionais, conforme apresenta a Tabela 5.6.

Os casos de teste do conjunto T_{estr} que cobriram o critério Todos-Usos (conforme Tabela 5.3) gera o conjunto $T_{estr\_usos} = \{01, 02, 03, 05, 06, 07, 08, 11, 16, 21, 31, 37, 38\}$ que conseguiu cobrir 100% dos requisitos funcionais, sendo que os casos de teste: 01, 02, 03, 06 e 31 foram suficientes para obter tal cobertura, conforme mostra a Tabela 5.7.

Para os casos de teste do conjunto T_{estr} que cobriram o critério Todos-Potenciais-Usos (conforme Tabela 5.3) tem-se $T_{estr\_potUsos} = \{1, 2, 3, 4, 5, 6, 7, 8, 11, 13, 16, 21, 29, 31, 37, 38, 66, 68, 87\}$.

Tabela 5.6: Bozó - Cobertura dos Requisitos Funcionais por $T_{estr\_arestas}$

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
1	01, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20 e 21	14 de 20	70%
2	22 e 23	16 de 20	80%
3	06	17 de 20	85%
6	07 e 24	19 de 20	95%
31	04	**20 de 20**	**100%**

Tabela 5.7: Bozó - Cobertura dos Requisitos Funcionais por T_{estr_usos}

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
1	01, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20 e 21	14 de 20	70%
2	22 e 23	16 de 20	80%
3	06	17 de 20	85%
6	07 e 24	19 de 20	95%
31	04	**20 de 20**	100%

Na tabela 5.8 pode-se verificar que também houve 100% de cobertura dos requisitos funcionais para o conjunto $B_{potUsos}$, sendo que os casos de teste de número: 01, 02, 03, 04, 06 e 31 foram suficientes para realizar tal cobertura.

Tabela 5.8: Bozó - Cobertura dos Requisitos Funcionais por $T_{estr\_potUsos}$

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
1	01, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20 e 21	14 de 20	70%
2	22 e 23	16 de 20	80%
3	06	17 de 20	85%
4	07	18 de 20	90%
6	24	19 de 20	95%
31	04	**20 de 20**	**100%**

Esta **terceira fase durou 5 horas**, sendo que 3h20min foram utilizadas para avaliar os casos de teste do conjunto T_{estr} e conjuntos derivados ($T_{estr\_nós}$, $T_{estr\_arestas}$, T_{estr_usos} e $T_{estr\_potUsos}$) e 1h40min para gerar uma documentação simples.

5.1.3.4 Bozó - 4ª Fase - Cobertura dos Critérios Estruturais por T_{func}

Agora o interesse está em avaliar a cobertura do conjunto T_{func} para cobrir os critérios estruturais, vale lembrar que o conjunto T_{estr} cobriu 100% dos critérios estrutu-

rais.

Nesta fase da atividade de teste, o conjunto de casos de teste T_{func} (Apêndice B) foi aplicado na Ferramenta JaBUTi/MA com o intuito de verificar a porcentagem de cobertura obtida para os critérios estruturais. Vale esclarecer que, os itens não executáveis referentes aos critérios estruturais foram excluídos.

A Tabela 5.9 apresenta a cobertura dos critérios estruturais com relação ao conjunto de casos de teste T_{func}.

Tabela 5.9: Bozó - Cobertura dos Critérios Estruturais por T_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos		Todos-Potenciais-Usos	
1	161 de 176	91%	197 de 233	84%	416 de 527	79%	1299 de 1673	77%
2	164 de 176	93%	204 de 233	87%	439 de 527	83%	1364 de 1673	81%
3	167 de 176	95%	211 de 233	90%	460 de 527	87%	1414 de 1673	84%
54	170 de 176	97%	218 de 233	93%	480 de 527	91%	1442 de 1673	86%
61	171 de 176	97%	220 de 233	94%	491 de 527	93%	1463 de 1673	87%

Esta quarta fase da aplicação do teste funcional no jogo Bozó, não foi cansativa, pois a Ferramenta JaBUTi/MA tornou fácil e atraente a gerência e a avaliação da cobertura dos critérios estruturais. O tempo de **duração desta quarta fase foi de 1 hora**, sendo que 20min foram utilizados para criar um projeto de teste na JaBUTi/MA, e executar os casos de teste, e 40min para avaliar tais casos e gerar uma documentação simples sobre a cobertura.

5.1.3.5 Bozó - 5ª Fase - Comparação dos Resultados

Aqui o intuito é de comparar os conjuntos de casos de teste gerados. Assim, temos que no item 5.1.3.1 foram gerados casos de teste para cobrir 100% dos requisitos funcionais criando-se, para tanto, o conjunto de casos de teste T_{func} (Apêndice B). E o item 5.1.3.2 apresentou o conjunto de casos de teste T_{estr} (Apêndice C), gerado pela JaBUTi/MA, que obteve 100% de cobertura dos critérios estruturais.

Com base nos conjuntos de casos de teste T_{func} e T_{estr} foi estipulada uma troca na cobertura, ou seja, o item 5.1.3.3 apresentou a cobertura dos requisitos funcionais pelo conjunto T_{estr} e o item 5.1.3.4 apresentou a cobertura dos critérios estruturais por T_{func}.

Avaliando a troca percebe-se que o conjunto T_{func}, inicialmente gerado para cobrir os requisitos funcionais, apesar de ter coberto 100% destes, quando aplicado para cobrir os critérios estruturais apresentou uma cobertura, conforme Tabela 5.9, de 97% do critério Todos-Nós, 94% para Todas-Arestas, 93% para Todos-Usos e 87% para Todos-Potenciais-Usos. Assim, pode-se verificar que não houve critério estrutural 100% coberto.

Portanto, o conjunto T_{func} não apresentou uma cobertura satisfatória com relação aos critérios estruturais.

Com base no conjunto T_{estr}, gerado pela ferramenta JaBUTi/MA para cobrir os critérios estruturais, verifica-se que além de ter obtido 100% de cobertura dos critérios estruturais, também conseguiu obter 100% de cobertura dos requisitos funcionais. Detalhando a cobertura, com base em cada critério estrutural, verificou-se que os conjuntos $T_{estr\_arestas}$, T_{estr_usos} e $T_{estr\_potUsos}$ (derivados de T_{estr}) obtiveram 100% de cobertura dos requisitos funcionais e somente o conjunto $T_{estr\_nós}$ apresentou uma cobertura menor, porém considerável, de 95%.

5.1.3.6 Conclusão da Atividade de Teste - Jogo Bozó

A análise de cobertura da Ferramenta JaBUTi é altamente interessante de ser utilizada na atividade de teste estrutural, facilitando a verificação da cobertura do fluxo de dados e de controle do programa.

Comparando o teste estrutural com auxílio da JaBUTi e o teste funcional realizado manualmente, verificou-se que com o auxílio da Ferramenta JaBUTi a atividade de teste fica muito mais fácil de ser realizada e a probabilidade de ocorrência de erros, por descuido de observação e anotação ocorrido no teste manual, é remota. O teste funcional realizado manualmente exige que o testador esteja sempre atento às suas anotações e avaliações. Além disso, o tempo gasto na atividade de teste funcional foi muito superior quando comparado com o teste estrutural auxiliado pela JaBUTi.

Analisando as atividades de teste (funcional e estrutural) aqui realizadas, pode-se observar que a maior cobertura (Tabela 5.10) obtida pelo conjunto de casos de teste T_{func}, gerado durante o teste funcional, foi de 97% para o critério Todos-Nós. Quando comparado com a cobertura (Tabela 5.11) do conjunto T_{estr}, gerado durante o teste estrutural com o auxílio da JaBUTi/MA, verificou-se que a menor cobertura foi de 95% dos requisitos com

relação ao conjunto $T_{estr\_nós}$, os demais conjuntos derivados de T_{estr} apresentaram 100% de cobertura dos requisitos funcionais.

Tabela 5.10: Bozó - Cobertura Geral dos Critérios Estruturais por T_{func}

	Cobertura dos Critérios Estruturais por T_{func}
Todos-Nós	97%
Todas-Arestas	94%
Todos-Usos	93%
Todos-Potenciais-Usos	87%

Tabela 5.11: Bozó - Cobertura Geral dos Requisitos Funcionais pelos Derivados de T_{estr}

	Cobertura dos Requisitos Funcionais
$T_{estr\_nós}$	95%
$T_{estr\_arestas}$	100%
T_{estr_usos}	100%
$T_{estr\_potUsos}$	100%

5.2 *Mobile Chat*

O *Mobile Chat* é um estudo de caso que foi especialmente desenvolvido para este trabalho no intuito de ser testado pela Ferramenta JaBUTi/MA e fortalecer o tema proposto pelo mesmo. Como o próprio nome exprime, é um programa de conversação em tempo real que tem por função enviar e receber mensagens entre usuários (clientes conectados em rede) através de um servidor.

Apesar de parecer um programa comum de comunicação, o *Mobile Chat* se diferencia dos demais pela sua tecnologia. É um exemplo, desenvolvido em Java utilizando a API μCODE, onde tem-se na máquina cliente a troca de mensagens e informações sendo realizada através de agentes móveis comunicando-se com o servidor e vice-versa.

Também apresenta um servidor móvel que migra entre os *hosts* clientes conforme as variações ocorridas no ambiente da rede, relacionadas à latência (custo) entre os nós da rede, ou através de uma solicitação do cliente possibilitando ao mesmo utilizar localmente o servidor. Isto se torna interessante quando algum cliente utiliza excessivamente o servidor, livrando o tráfego na rede.

Esta característica migratória do servidor é um dos fatores principais para que este trabalho possa utilizar tal exemplo, já que o intuito é testar os códigos móveis em situações diversas reforçando o teste estrutural.

Nas subseções a seguir serão apresentadas a forma de execução e a descrição da implementação do *Mobile Chat*. Em seguida, será descrita a atividade de teste realizada no mesmo. Vale ressaltar que, devido ao servidor ser móvel, esta atividade de teste passa a ser muito mais interessante do que no exemplo anterior.

5.2.1 Funcionalidades do *Mobile Chat*

O servidor principal do *Mobile Chat* deve ser o primeiro elemento a ser iniciado. Só existe uma única instância deste servidor na rede, executando continuamente no *host* em que foi iniciado, tal servidor é denominado de `MainServer`.

Quando é solicitada a migração do servidor, o `MainServer` fica desativado, uma instância de um servidor itinerante é criada, sendo denominada de `ChatServer`, esta instância é migrada e passa a ser conhecida como o servidor ativo. Porém, o servidor principal continua rodando para que um novo usuário se conecte, para evitar falha na conexão do cliente, devido ao desconhecimento do cliente sobre o endereço IP do servidor itenerante. O servidor principal se tornará ativo novamente somente quando o servidor itenerante retornar para o *host* de origem.

Para conseguir realizar a migração do servidor, cada cliente e o próprio servidor precisam ter uma tabela de roteamento contendo o custo associado entre todos os nós que estarão aptos a hospedar um cliente ou um servidor. A tabela deverá ter o seguinte formato:

```
endereco_ip_1:porta1 custo1
endereco_ip_2:porta2 custo2
```

Cada cliente apresenta uma tabela distinta de forma que o último cliente a se conectar sempre levará ao servidor as informações mais atualizadas sobre o estado da rede. Assim, é possível simular variações no comportamento da rede informando valores de custo propositais para ocorrer uma migração planejada.

A migração baseada na tabela de custo ocorre quando um novo cliente se conecta. A tabela de custo que acompanha o cliente é analisada pelo agente móvel `SpawnMigrationAgent` que identifica o endereço que apresenta o custo médio mínimo em relação a todos os outros nós da rede, desta forma, o servidor migra para tal endereço.

Para a execução do *Mobile Chat*, é necessário que seja executado um servidor e mais de um cliente para que haja a sessão de *chat*. Assim, o primeiro passo é executar o servidor que é invocado através da classe `Server`, que se encontra no pacote `server`, através do seguinte comando[5.4]:

```
java server.Server
```

Então, o servidor é inicializado na porta padrão 1968 do `MuServer`, conforme ilustrado pela Figura 5.8.

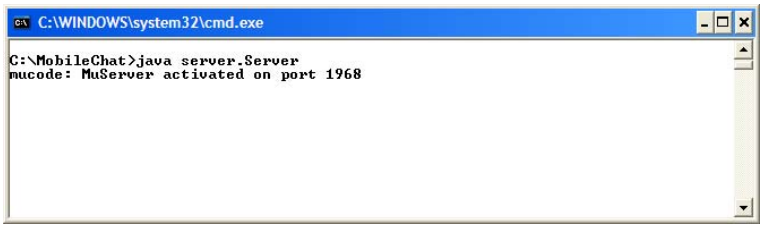

Figura 5.8: Servidor inicializado.

Outra forma de inicializar o servidor é informando o parâmetro -port seguido da porta desejada, conforme o exemplo a seguir (neste caso o servidor é iniciado na porta 2005):

```
java server.Server -port 2005
```

Não importando a forma utilizada para executar o servidor, após ser iniciado, ele fica aguardando conexões de novos clientes na porta em que está sendo executado.

Para que o usuário possa se conectar ao servidor para participar de uma sessão de *chat*, ele deve utilizar a classe `Client` do pacote `client`. Diferente do servidor, o cliente possui uma interface gráfica que permite ao usuário digitar mensagens e comandos para

[5.4] Os comandos devem ser digitados dentro do diretório onde se encontram os pacotes do *Mobile Chat* e o *classpath* deve estar configurado de maneira correta, ou seja, informando tanto o diretório corrente quanto o arquivo `mucode.jar`.

interagir com outros clientes e com o servidor. Assim, para invocar o cliente, a seguinte linha de comando deve ser utilizada:

```
java client.Client <nick> <ipCliente:porta> <ipServidor:porta> <arqRoteamento>
```

Onde, *nick* é o parâmetro que trata do apelido utilizado pelo usuário, sendo que cada usuário possui um apelido distinto. O segundo parâmetro especifica o endereço IP do cliente e a porta[5.5] em que o servidor cliente (`ClientServer`) irá utilizar para execução, possibilitando que o servidor principal (`MainServer`) estabeleça comunicação com o cliente. O terceiro parâmetro caracteriza o endereço IP da máquina servidora em conjunto com a porta onde o servidor principal (`MainServer`) está em execução, isto possibilita que a máquina cliente participe da sessão de *chat*. E, por fim, o último parâmetro é o nome do arquivo que contém a tabela de roteamento[5.6] com os custos envolvidos na comunicação entre o cliente que está sendo iniciado e os demais nós da rede.

A abstração de agentes móveis provida pela API μCODE exige que cada nó cliente possua um servidor através do qual serão enviados e recebidos os agentes móveis. Desse modo, mesmo os clientes devem possuir servidores, que no caso são instâncias da classe `ClientServer`.

Como exemplo, pode ser utilizado o seguinte comando para a inicialização de um cliente:

```
java client.Client Jose 192.168.0.5:2000 192.168.0.1:1968 tabela.txt
```

Neste exemplo tem-se um usuário com o apelido de "Jose", que se encontra na máquina com o endereço IP 192.168.0.5 informando que a porta para a execução do `ClientServer` é a porta 2000, que o endereço do servidor principal (`MainServer`) é 192.168.0.1, e que este está sendo executado na porta 1968. Também está utilizando o arquivo tabela.txt onde estão armazenadas as informações sobre a tabela de custo de roteamento.

[5.5] A porta pode ser qualquer valor inteiro superior a 1024.
[5.6] Determinação da rota (direção) de informações enviadas na rede.

Dessa forma, o cliente irá iniciar o `ClientServer` que ficará executando na porta 2000 e estabelecerá conexão com o `MainServer` informando o apelido e as informações extraídas da tabela de custo de roteamento.

Como ilustração é apresentada a inicialização de um cliente, conforme as Figuras 5.9 e 5.10 que mostram, respectivamente, o *shell* de execução referente ao `ClientServer` e a interface gráfica do *Mobile Chat*.

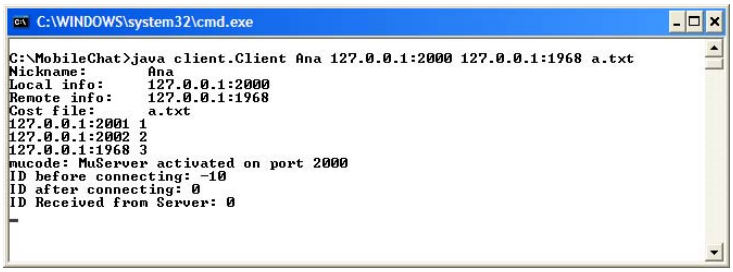

Figura 5.9: *Shell* de execução do `ClientServer`

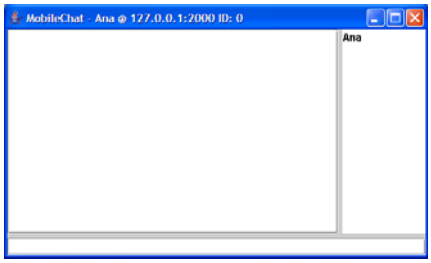

Figura 5.10: Tela do *Mobile Chat*

A interface gráfica do cliente foi confeccionada de forma bem simples, utilizando comandos de texto de pouca complexidade para simplificar a interação entre o cliente e o servidor. O comando diferencia-se de uma mensagem comum através de uma barra "/" inserida no começo da linha. A sintaxe é definida da seguinte forma:

```
/<comando>
```

A interpretação do texto digitado é realizada através da verificação da existência de uma barra no início do texto. Desta forma, o texto é considerado um comando e a execução de sua funcionalidade é realizada. Caso contrário, ou seja, se o texto digitado for diferente de "/", o programa considera que o texto é uma mensagem comum destinada a outros usuários.

Para se obterem os comandos disponíveis pelo *Mobile Chat* basta digitar o comando `/help` para que o mesmo possa apresentar uma mensagem de ajuda na tela com todos os comandos oferecidos juntamente com a descrição de suas respectivas funcionalidades, conforme a ilustração da Figura 5.11.

O *Mobile Chat* apresenta vários comandos para facilitar seu uso:

- `/quit`: faz com que o cliente seja desconectado do servidor e finalizado. Para tanto, primeiramente é verificado se o cliente está conectado e, caso esteja, o mesmo é

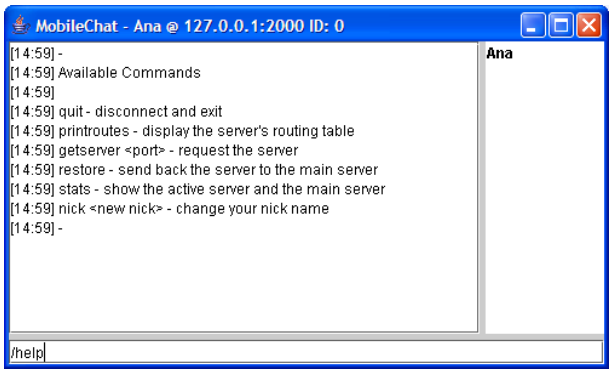

Figura 5.11: Apresentação do comando (`/help`)

desconectado do servidor. Em seguida, ou caso o cliente não esteja conectado, ele é finalizado.

- `/printroutes`: exibe a tabela de custo presente no servidor.
- `/getserver <porta>`: faz com que o servidor migre para o cliente que o requisitou, instalando o servidor itinerante (`ChatServer`) na porta especificada.
- `/restore`: envia o servidor de volta à sua posição inicial caso esteja presente na máquina do cliente, ou seja, estando na máquina cliente o servidor retorna para o nó de origem. Para isto, é verificado se o servidor está realmente hospedado na máquina local, caso não esteja uma mensagem de erro é exibida ao usuário.
- `/stats`: informa ao usuário o endereço do servidor principal (`MainServer`) e o endereço do servidor ativo, que pode ser diferente do principal como é o caso do servidor itinerante (`ChatServer`) que migra para outros nós da rede. Além disso, se o servidor estiver em execução na máquina do cliente, uma mensagem é exibida na tela informando que o cliente está hospedando o servidor.
- `/nick <novo_apelido>`: troca o apelido atual do usuário pelo apelido especificado. Portanto, após executar o comando é verificado se um apelido foi de fato digitado, caso contrário um erro é acusado.

No caso do usuário enviar uma mensagem para outros usuários, é preciso primeiro digitar a mensagem na caixa de entrada e, em seguida, pressionar a tecla *ENTER*. Como foi apresentado, o *Mobile Chat* é um *software* de comunicação simples em relação a sua utilização, porém autêntico em sua tecnologia e forma de tratamento do envio das mensagens e execução de tarefas realizadas através de seus comandos.

A seguir é apresentado um exemplo simples de uma sessão de *chat* para ilustrar melhor como ocorre a comunicação entre os usuários. Vale esclarecer que, o servidor principal é inicializado e três máquinas clientes são conectadas ao mesmo com o intuito de trocarem mensagens e executarem alguns comandos.

A Figura 5.12 ilustra todo o processo ocorrido no servidor principal durante a sessão de *chat*, como: a inicialização do servidor principal na porta padrão 1968, a inclusão de novos clientes na sessão de *chat*, a apresentação do melhor *host* para alocar o servidor ativo, a migração do servidor itenerante para um outro *host* e o endereço atualizado do servidor itenerante.

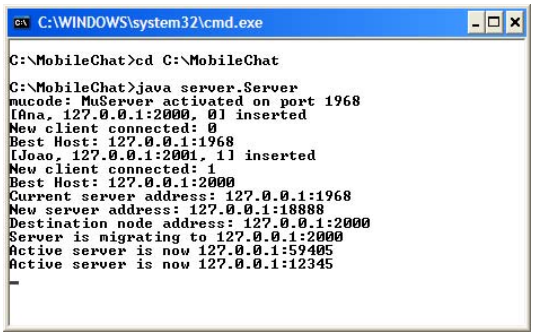

Figura 5.12: Simulação de *chat* - Execução do servidor principal.

A Figura 5.13 apresenta o *shell* de execução do cliente para o usuário "Ana", onde são apresentadas todas as informações ocorridas durante a sessão de *chat*, como: as informações referentes a conexão do cliente (*nick*, endereço dos *hosts* do cliente e do servidor principal); o identificador do cliente gerado pelo servidor principal quando o cliente é conectado (que neste caso é "0"); a informação sobre a migração do servidor para a sua própria máquina e a porta atual que está usando; a conexão de novos clientes (já que o servidor se encontra em tal *host*); a informação sobre o melhor servidor; a migração do servidor para outro *host* e o retorno do servidor para o *host* da Ana através da utilização do comando `/getserver` informando a porta "12345".

A interface gráfica da Ana, conforme Figura 5.14, apresenta, do lado direito, a lista de todos os usuários conectados, do lado esquerdo são apresentadas todas as informações ocorridas durante a sessão de *chat* sobre a conexão de novos usuários, já que a Ana foi a primeira a se conectar, e as conversas entre os usuários. Na caixa de texto, que recebe todos as entradas de mensagens e comandos, o usuário Ana insere o comando `/getserver <porta>` para chamar o servidor para ser alocado em sua máquina.

A Figura 5.15 apresenta o *shell* de execução do cliente para o usuário "João".

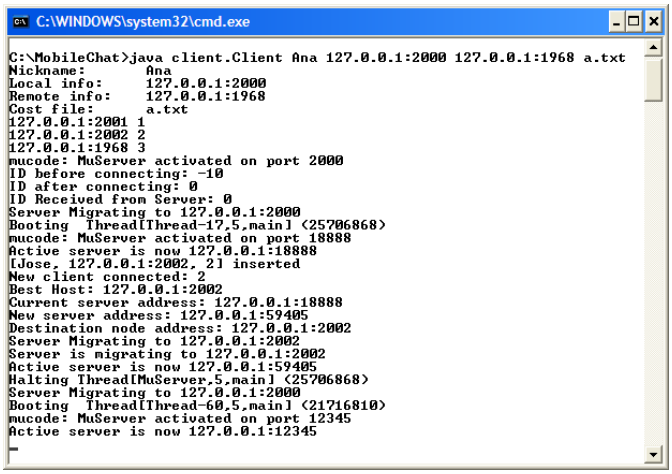

Figura 5.13: Simulação de *chat* - *shell* de execução do usuário Ana.

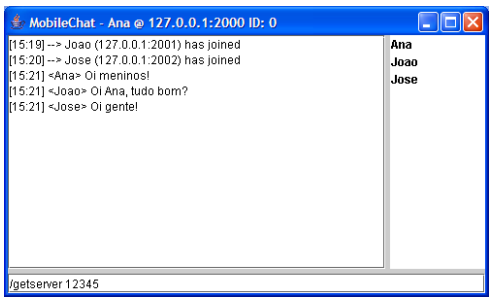

Figura 5.14: Simulação de *chat* - tela do cliente para o usuário Ana.

Neste caso, como no anterior, o servidor cliente apresenta: as informações da conexão do usuário João; o *id* "1" que o usuário recebeu do servidor; as informações de que o servidor principal irá migrar; as informações de que está migrando; o endereco e porta do *host* onde o servidor está ativo; e a impressão da tabela de roteamento através da invocação do comando `/printroutes`.

A interface gráfica do usuário João, conforme ilustrado pela Figura 5.16, apresenta igualmente como no usuário anterior, as informações sobre quais usuários estão conectados e a conversa entre eles, bem como a entrada do comando `/printroutes` solicitando a impressão da tabela de roteamento no *shell* do cliente.

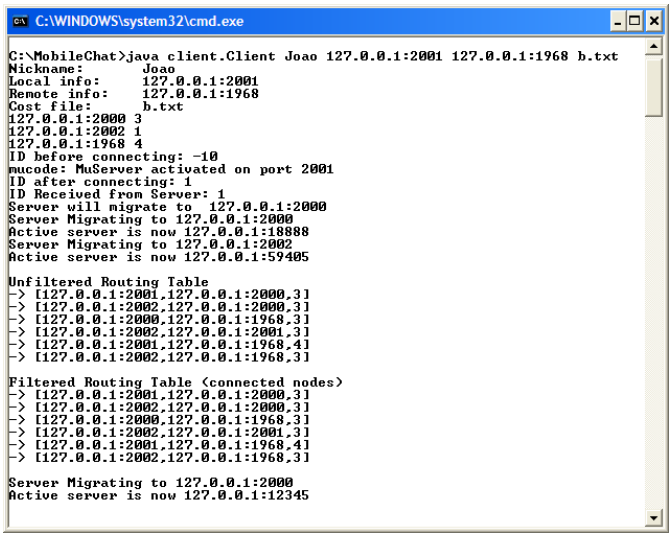

Figura 5.15: Simulação de *chat* - *shell* de execução do usuário João.

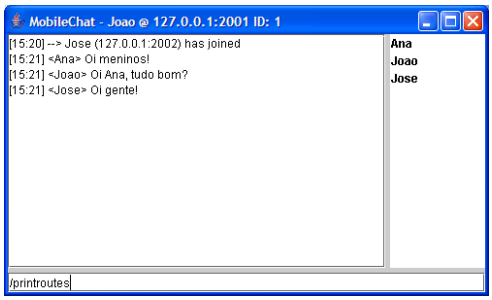

Figura 5.16: Simulação de *chat* - tela do cliente para o usuário João.

A Figura 5.17 apresenta o *shell* de execução do cliente para o usuário "José", onde são exibidas informações referentes a conexão, o *id* de número 2 gerado pelo servidor principal, a localização do servidor ativo, também mostra que este *host* aloca o servidor ativo e informa que o servidor será migrado para o *host* 127.0.0.1:2000 e que o mesmo está ativado na porta "12345".

Na interface gráfica do cliente, Figura 5.18, como nas outras, apresenta a conversação entre os usuários e a invocação do comando `/stats`. Neste caso, quando o comando foi invocado, o servidor estava sendo executado no mesmo *host* do cliente, então, uma mensagem informando que o cliente já estava hospedando o servidor foi exibida.

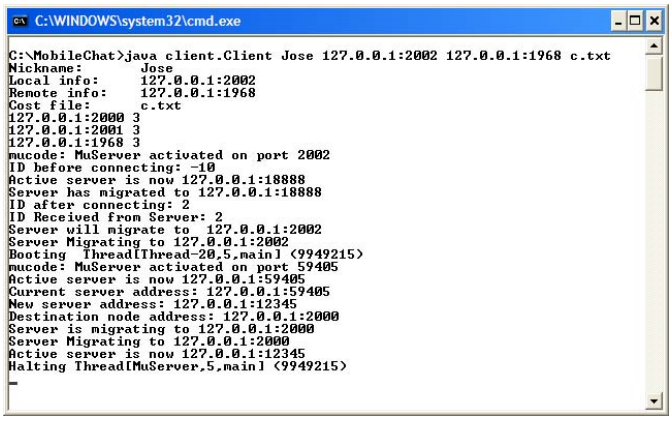

Figura 5.17: Simulação de *chat* - *shell* de execução do usuário José.

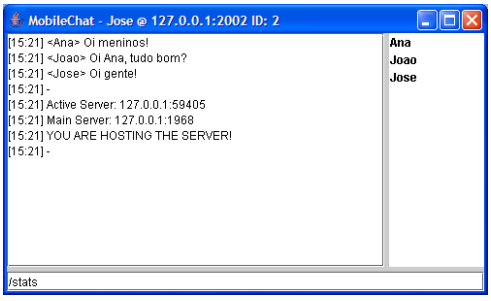

Figura 5.18: Simulação de *chat* - tela do cliente para o usuário José.

Como foi apresentado, através da exemplificação de uma sessão de *chat*, pode-se verificar que o estudo de caso *Mobile Chat* é bastante simples e interessante de ser utilizado. Sua tela é clara e objetiva ao apresentar as informações necessárias para uma conversação simples, seus comandos tornam o aplicativo mais interessante, possibilitando a observação da migração e o controle do servidor mais claro para o usuário, além de mostrar uma boa forma de utilizar a mobilidade de código dentro do contexto de um sistema de comunicação.

Na próxima sessão serão descritos detalhadamente todos os processos de conexão, troca de mensagens e migração realizados pelo aplicativo.

5.2.2 Aspectos de Implementação - *Mobile Chat*

Existem dois elementos principais que compõem o *Mobile Chat*: o cliente e o servidor. Com a aplicação cliente o usuário poderá interagir com os outros usuários através de mensagens, além de obter informações sobre os servidores e sobre os usuários através de comandos. O *Mobile Chat* apresenta três tipos de servidores:

- `ChatServer:` fornece as funcionalidades básicas do servidor.
- `MainServer:` é uma extensão do `ChatServer`, trata-se do servidor principal.
- `ClientServer:` também é uma extensão do `ChatServer`, porém trata-se do servidor responsável pelo cliente. A abstração de agentes móveis provida pela API μCODE exige que cada nó da rede possua um servidor através do qual serão enviados e recebidos os agentes móveis. Deste modo, os clientes também devem possuir servidores que no caso são instâncias da classe `ClientServer`.

Vale ressaltar que os três tipos de servidores apresentados são extensões do servidor `MuServer`, oferecido pela μCODE, que possibilita a relocação de código através dos nós da rede fornecendo o ambiente que dá suporte à mobilidade de código e tornando assim possível a implementação deste aplicativo.

O *Mobile Chat* é composto por quatro pacotes (Figura 5.19): `agents`, `chat`, `client` e `server`.

O pacote `agents` possui todos os agentes móveis necessários para a realização de troca de mensagens, conexão e desconexão de um cliente, impressão da tabela de roteamento, migração do servidor através de uma requisiçao(comando `/getserver`) ou de uma restauração(comando `/restore`), informação atualizada da localização do servidor e a troca de apelidos (Figura 5.20). Vale ressaltar que o pacote `agents` é composto por treze agentes móveis responsáveis pelos serviços citados.

O pacote `chat`, Figura 5.21, contém todas as classes que mantém a sessão de *chat* com relação as informações do usuário, da tabela de roteamento, e das constantes utilizadas no *chat*(conectado, não conectado, adicionar ou excluir apelido).

O pacote `client`, ilustrado pela Figura 5.22, possui classes que tratam das informações do cliente; e o pacote `server`, Figura 5.23, é composto pelo servidor principal

`MainServer` e pelo servidor itinerante `ChatServer`.

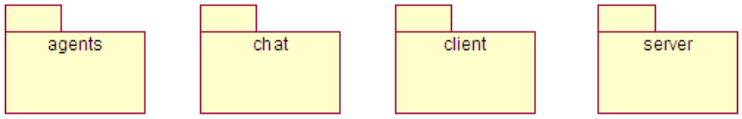

Figura 5.19: Pacotes que compõem o *Mobile Chat*.

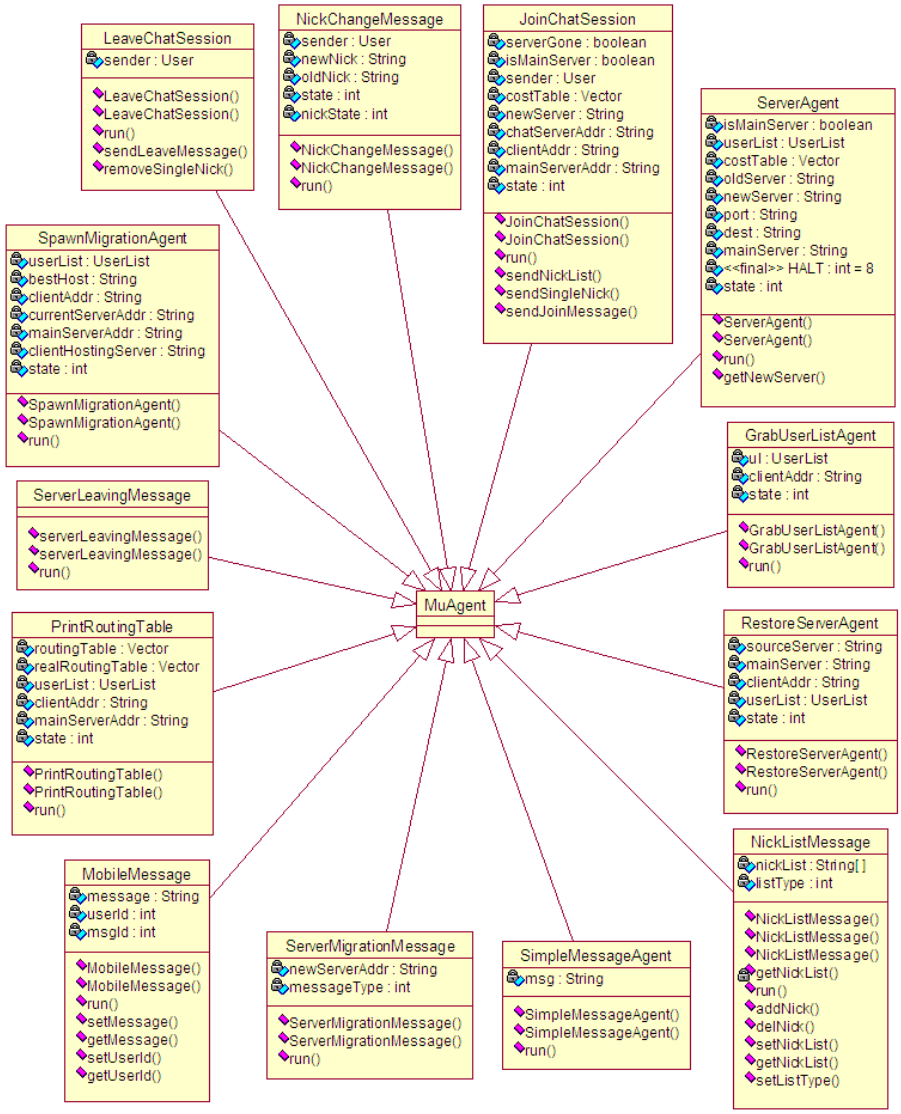

Figura 5.20: Classes que compõem o pacote *agents*.

5.2.2.1 Conexão de Clientes

Primeiramente, o servidor principal (`MainServer`) é iniciado em algum nó da rede. Só existe uma instância deste servidor na rede que fica rodando continuamente no *host* onde foi iniciado. O `MainServer` aguarda conexões de novos clientes na porta onde

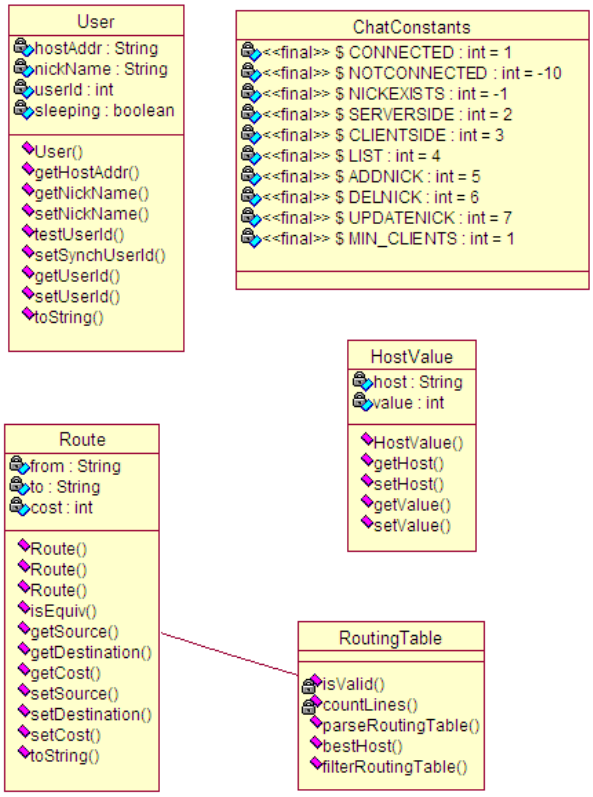

Figura 5.21: Classes que compõem o pacote *chat*.

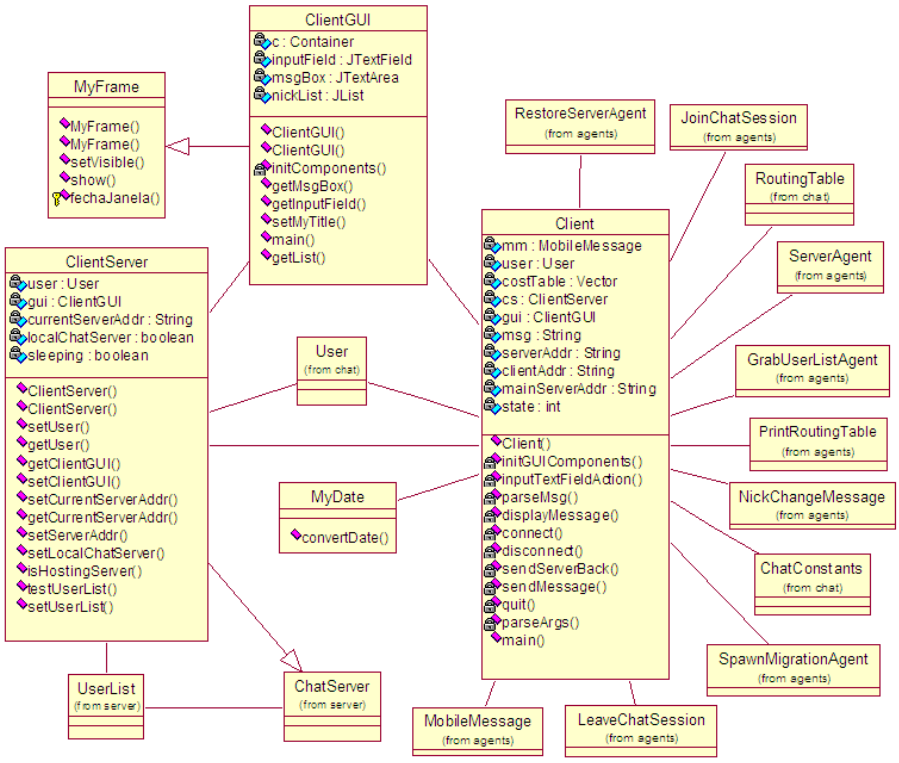

Figura 5.22: Classes que compõem o pacote *client*.

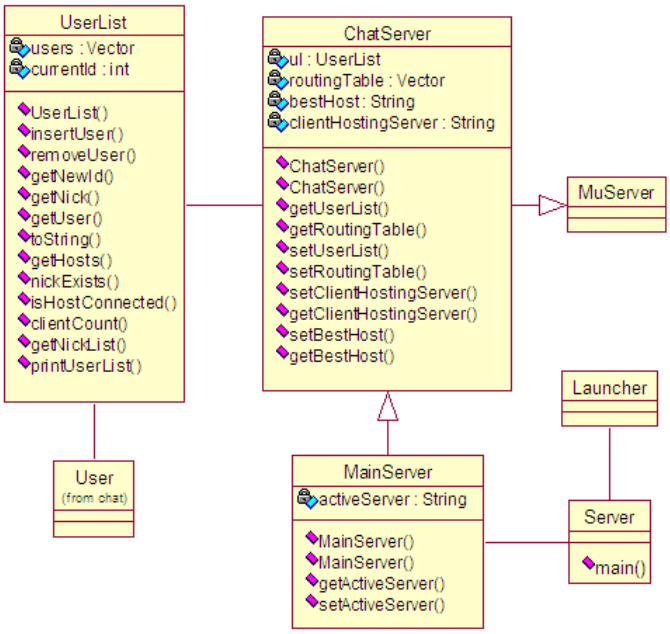

Figura 5.23: Classes que compõem o pacote *server*.

está sendo executado. Ao ser inicializado, o `MainServer`, possui apenas a lista de usuários e a tabela de custos vazias, pois não há nenhum cliente conectado.

Em seguida deve-se iniciar o cliente, que interpreta os parâmetros do usuário através da linha de comando, conforme apresentado na Seção 5.2.1. Se ocorrer algum problema durante a interpretação dos parâmetros o cliente é finalizado, caso contrário, as informações colhidas são exibidas na saída padrão para o cliente certificando-o de que tal interpretação ocorreu com sucesso.

O próximo passo para a conexão do cliente é a leitura do arquivo com as informações de roteamento através da classe `RoutingTable`. Durante tal processo, todas as linhas que compõem o arquivo são lidas. Uma linha precedida do símbolo # é interpretada como um comentário e as linhas em branco são ignoradas. Para cada linha válida é gerada uma tripla contendo: o endereço IP do cliente, o endereço IP de um outro nó e o custo envolvido na comunicação entre estes dois nós. Se essa tripla ainda não estiver presente na tabela que está sendo gerada, ela é inserida.

Após o processamento do arquivo, o cliente possuirá uma estrutura de dados interna contendo o resultado da interpretação. Essa tabela de custo será enviada posteriormente para o servidor para que este tenha a relação atualizada dos endereços dos clientes e seus respectivos custos.

O cliente então inicia um servidor cliente (`ClientServer`) na porta especificada pelo segundo parâmetro e se conecta ao `MainServer` cujo endereço foi especificado no terceiro parâmetro de execução do cliente, enviando uma mensagem de requisição através do agente móvel `JoinChatSession` contendo o apelido, o endereço do usuário e a tabela de custo gerada pela interpretação do arquivo lido.

Se durante a conexão do cliente com o servidor, percebe-se que este último migrou para um *host* diferente, então, o `MainServer` encaminha o `JoinChatSession` para o *host* onde o servidor itinerante (`ChatServer`) está localizado e notifica o cliente a respeito da nova localização do servidor.

Em seguida, estando no servidor (itinerante ou principal), o agente `JoinChat Session` verifica se o apelido escolhido pelo usuário já está sendo utilizado. Caso esteja, o agente retorna ao cliente, avisando-o do uso do apelido e requisita a escolha de outro. Tal processo se repete até que o usuário escolha um apelido válido.

Após a verificação do apelido, o servidor gera um número de identificação para o usuário e o insere na lista de participantes do *chat*. Esta lista é mantida no servidor e contém diversas entradas, cada uma representando um participante, constistuída por três campos: o apelido, o endereço IP e o número de identificação do usuário. Então, cada cliente recebe do servidor o agente `MobileMessage`, responsável por transportar as mensagens, que leva consigo uma mensagem com o apelido e o endereço do novo usuário[5.7] informando a sua entrada no *chat*.

Em seguida, o agente `NickListMessage`, responsável por inserir o apelido na lista de participantes do *chat*[5.8], é enviado para todos os participantes, levando somente o apelido do novo usuário para atualizar a lista de participantes do cliente.

O agente `JoinChatSession`, ainda no servidor, atualiza a tabela de custo do servidor através dos dados existentes na tabela de custo que ele trouxe consigo, proveniente do cliente. Essa atualização se dá através da inserção das entradas recebidas do cliente na tabela de custo do servidor, caso elas ainda não existam, evitando entradas duplicadas.

A introdução de uma nova tabela de custo atualiza a tabela presente no servidor, alterando a impressão do servidor sobre a rede como se ocorresse uma variação no ambiente de rede. Então, um algoritmo de seleção é aplicado para identificar qual nó da tabela

[5.7] O apelido e o endereço IP serão exibidos na caixa de texto dos participantes
[5.8] A lista de participantes é encontrada no lado direito da interface gráfica do cliente

apresenta o menor custo médio. O endereço desse nó é armazenado no servidor para posterior utilização.

Em seguida, o cliente que acabou de se conectar recebe uma instância do agente `NickListMessage` que atualiza a lista de apelidos na interface do cliente, com os apelidos de todos os clientes conectados.

Então, o agente `JoinChatSession` volta para o cliente com o número de identificação do usuário, alterando a barra de título da interface do cliente com as informações do mesmo e habilitando o campo de entrada de texto. Caso o agente `JoinChatSession` não retorne, isso significa que o cliente não conseguiu estabelecer a conexão, este é removido da lista de participantes e uma mensagem de notificação é enviada para os demais participantes informando que o cliente que estava tentando se conectar não vai mais participar da seção de *chat*.

Vale ressaltar que, devido aos agentes serem autônomos, não existe uma conexão constante entre os clientes e o servidor, ou seja, há possibilidade de troca de mensagens em qualquer situação, uma mensagem sendo independente das demais. O estabelecimento de conexão entre o cliente e o servidor apenas significa que o servidor está ciente da existência do cliente e armazena o mesmo em sua lista de clientes conectados.

Após conectado, o usuário está apto à trocar mensagens de texto com os outros usuários.

5.2.2.2 Comunicação entre Usuários

A comunicação entre usuários no *Mobile Chat* é efetuada através da troca de mensagens. Para que um usuário possa enviar uma mensagem para os demais é preciso, primeiro, digitar o texto da mensagem na caixa de entrada e, em seguida, pressionar a tecla *ENTER*. O texto digitado será encapsulado no agente móvel `MobileMessage` juntamente com o número de identificação do cliente. Este agente terá como destino o servidor ativo.

Ao chegar no servidor, o agente `MobileMessage` obtém a lista de clientes conectados e identifica qual o apelido do usuário que o enviou, através do número de identificação que trouxe consigo, criando então um novo texto, concatenando o apelido e o texto da mensagem para a identificação do remetente. O formato desse novo texto é representado da seguinte forma:

```
<apelido> texto
```

Então, o novo texto é encapsulado em um outro agente, o `MobileMessage`, que é enviado para todos os usuários presentes na lista de conectados, inclusive para o cliente que a enviou. Quando o agente chega em um dos clientes, ele cria um outro texto novo concatenando o formato do texto anterior com a hora em que o agente chegou, de modo a exibir para o usuário a hora de chegada da mensagem. O formato do texto concatenado que é exibido para o usuário é o seguinte:

```
[hh:mm] <apelido> texto
```

Esclarecendo, `hh` representa as horas, `mm` os minutos e `apelido` o remetente da mensagem. Desta forma, o destinatário tem a noção sobre qual o horário que a mensagem chegou, quem enviou (remetente) e qual o texto enviado.

Além da troca de mensagens o *Mobile Chat* oferece comandos que fortalecem o controle da sessão de *chat* e a comunicação entre o usuário e o servidor e vice-versa, bem como entre os próprios usuários, possibilitando um maior aproveitamento da sessão já que disponibiliza as informações de custo da rede, permite a requisição de migração e restauração do servidor, informa o estado atual da rede e permite a troca de apelido do usuário.

Um comando diferencia-se de uma mensagem através de uma barra "/" inserida no começo da linha como já apresentado na Seção 5.2.1. Durante a interpretação do texto digitado, se for detectada uma "/" o texto é considerado um comando e tratado de maneira específica pelo aplicativo cliente, caso contrário o texto é caracterizado como uma mensagem qualquer.

Detalhando melhor os comandos, já apresentados na Seção 5.2.1, tem-se para solicitar ajuda o comando `/help` que exibe uma mensagem de ajuda na tela contendo todos os comandos disponíveis e uma breve explicação sobre a funcionalidade de cada um deles.

Para finalizar o aplicativo basta invocar o comando `/quit` para que o cliente seja desconectado do servidor e finalizado. O processo de desconexão é parecido com o processo de conexão, porém ao em vez de enviar o agente `JoinChatSession` para entrar na

seção de *chat*, o cliente envia o agente `LeaveChatSession` requisitando a saída. Quando o `LeaveChatSession` chega no servidor, ele remove as informações do cliente que o enviou da tabela de conectados.

Em seguida, o agente `MobileMessage` envia uma mensagem de texto para os outros clientes informando que tal cliente foi desconectado. Após o envio dessa notificação, o agente `LeaveChatSession`, ainda no servidor, envia o agente `NickListMessage` para os clientes com a função de remover o apelido do cliente que se desconectou da lista de apelidos presente na interface gráfica dos clientes. Por fim, o agente `LeaveChatSession` remove as entradas relativas do cliente desconectado da tabela de custo do servidor, já que tal cliente não será mais um nó alvo para migração do servidor.

As informações de custo poderão ser obtidas através da invocação do comando `/printroutes` que exibe as tabelas de custo presentes no servidor. Quando este comando é acionado, o agente `PrintRoutingTable` é enviado para o servidor. Lá, o agente obtém a tabela de custo e retorna para o cliente. Estando no cliente, o agente exibe duas tabelas de custo, a primeira sem formatação, ou seja, da forma que foi obtida no servidor, a segunda exibe o resultado de uma filtragem, na qual apenas as entradas relativas aos clientes conectados são exibidas. Esta tabela filtrada é a que realmente interessa para o servidor determinar qual é o melhor nó para ele migrar.

Quando um cliente achar necessário poderá requisitar o servidor através do comando `/getserver <porta>`. Tal comando faz com que o servidor migre para o cliente que o requisitou, instalando o `ChatServer` na porta especificada pelo cliente. A execução deste comando faz com que o agente `ServerAgent` seja enviado para o servidor ativado no momento, seja ele o próprio `MainServer` localizado no *host* inicial ou algum `ChatServer` localizado na máquina de um dos clientes. Chegando no servidor ativo o `ServerAgent` efetua a migração[5.9] do servidor para o cliente da requisição.

Para restaurar o servidor principal, ou seja, para que o servidor ativo possa ser novamente o servidor principal `MainServer`, basta invocar o comando `/restore`. Este comando envia o servidor de volta à sua posição inicial, ou seja, se o servidor está presente na máquina de um cliente, ele retorna para o nó onde foi executado a princípio. Antes de tudo, o aplicativo cliente certifica-se de que o servidor está hospedado na máquina local. Caso esteja, o agente `RestoreAgent` é utilizado para enviar o servidor de volta para a posição inicial, ativando assim o `MainServer` novamente, caso contrário uma mensagem

[5.9] Detalhes sobre a migração serão tratados na Seção 5.2.2.3

de erro é exibida para o usuário.

O comando `/stats` informa ao usuário o estado dos servidores exibindo o endereço do servidor principal e o endereço do servidor ativo. Este último pode ser diferente do primeiro devido à ocorrência de migração do servidor. Vale esclarecer que o servidor ativo pode ser o principal, caso não houver a migração do servidor, ou pode ser o itinerante, caso houver migração do servidor. Além disso, se o servidor estiver em execução na máquina do cliente, uma mensagem é exibida na tela informando que o mesmo está hospedando o servidor.

Quando acontece a migração do servidor, do local de origem para uma máquina cliente, o `MainServer` fica desativado na máquina de origem e uma instância do `ChatServer` (servidor itinerante) é inicializada como servidor ativo na máquina de um cliente.

A troca de apelido é realizada pelo comando `/nick <novo_apelido>`. Tal comando troca o apelido atual do usuário pelo novo apelido especificado. Ao executar este comando é verificado se algum apelido foi realmente digitado, caso contrário um erro é acusado.

Se o apelido foi digitado, verifica-se se é diferente do apelido atual e se não contém nenhum espaço em branco. Após essa verificação, o agente `NickChangeMessage` é enviado para o servidor carregando o código que será executado no servidor e o novo apelido.

Ao chegar no servidor, o agente verifica se o novo apelido já está em uso por um outro usuário, caso esteja, o agente retorna ao cliente informando que o apelido não está disponível para uso. Caso o apelido não esteja em uso, a entrada do usuário na tabela de clientes conectados é atualizada.

Em seguida, o agente `NickListMessage` é enviado para todos os clientes conectados com o intuito de atualizar a lista de apelidos na interface gráfica dos clientes. Após a atualização da lista, o agente `MobileMessage` é enviado para todos os clientes conectados, com o objetivo de informá-los sobre a troca do apelido através de uma mensagem apresentada na interface gráfica dos clientes.

Finalmente, o agente `NickChangeMessage`, ainda no servidor, migra de volta para o cliente e atualiza o conteúdo da barra de título da interface gráfica com a intenção de exibir o novo apelido do usuário.

5.2.2.3 Migração do Servidor

Inicialmente, o servidor principal (`MainServer`) é executado em um dos nós da rede permanecendo neste nó até a finalização do sistema. Durante a execução do servidor, pode ocorrer a migração do mesmo para um nó de um cliente, ou a migração entre clientes. Neste segundo caso, o servidor já está localizado em um cliente e migrará para a máquina de outro cliente. Além disso, estando o servidor em um cliente, este poderá migrar para o *host* inicial, onde foi executado primeiramente, no servidor principal.

As migrações podem ocorrer de forma autônoma que é a migração baseada nas informações da tabela de custo, ou deliberada que é a migração acionada pelo comando `/getserver <porta>` de acordo com a vontade de algum usuário.

O *Mobile Chat* apresenta as seguintes formas de migração: do servidor principal para o cliente, de cliente para cliente, do cliente para o servidor principal e a migração baseada na tabela de custo

A **migração do servidor principal para o cliente** ocorre de duas formas: através da requisição do cliente ou autonomamente com base na tabela de custo.

Quando o servidor principal migra para um nó cliente e a migração foi requisitada pelo cliente, então, tal requisição faz com que o agente `ServerAgent` seja enviado ao `MainServer`.

Se a migração ocorrer de forma autônoma, a partir de uma análise realizada na tabela de custo, o agente `ServerAgent` também parte de um cliente tendo como destino o `MainServer`. Neste caso, o nó que apresenta maior vantagem é determinado quando o cliente se conecta através do agente `SpawnMigrationAgent` que dispara o `ServerAgent` que é o responsável pela migração do servidor.

Esclarecendo, independente do tipo de migração o agente `ServerAgent` sempre é enviado para o `MainServer`.

Assim que chega no `MainServer`, o `ServerAgent` informa para o servidor principal qual é o endereço do cliente que o enviou, desativa o `MainServer` e envia o agente `SimpleMessageAgent` para cada um dos clientes conectados. Esse agente, ao chegar no seu destino, exibe uma mensagem no *shell* de execução do `ServerClient` notificando o usuário que o servidor vai migrar. Em seguida, o `ServerAgent` migra para o cliente que irá

receber o servidor carregando consigo a lista de participantes que obteve no `MainServer` e a tabela de custo.

Estando na máquina cliente, uma instância da classe `ChatServer` é criada e o servidor fica escutando na porta em que se instalou. No caso de uma migração deliberada a porta é especificada pelo usuário, já no caso de uma migração autônoma a porta é aleatória. Essa instância do `ChatServer` será o servidor ativo, responsável pela troca de mensagens entre os usuários da rede.

Estando o `ChatServer` ativado, o agente `ServerMigrationMessage` informa à todos os clientes sobre a nova localização do servidor, de modo que eles saibam para onde enviar as mensagens quando forem se comunicar. Assim, o `ServerMigrationMessage` acessa a variável que armazena o endereço do servidor ativo e substitui pelo novo endereço.

Vale ressaltar que, mesmo estando desativado, o `MainServer` continua em execução. Isto ocorre devido ao fato de que um novo cliente, ao se conectar, não conhece a localização do servidor ativo no momento, pois o servidor ativo pode ser o itinerante e este pode estar em qualquer nó da rede. O que acontece é que o cliente se conecta ao `MainServer` e no caso do servidor ter sido migrado, o cliente é informado da migração e a requisição de entrada na sessão de *chat* é encaminhada para o servidor ativo. Isto é possível devido ao `MainServer` armazenar o destino da migração do servidor. Vale esclarecer que se o `MainServer` for finalizado, nenhum outro cliente conseguirá participar da sessão de *chat*.

A **migração de cliente para cliente** ocorre quando o servidor itinerante está na máquina de um cliente e vai migrar para a máquina de outro cliente. Essa forma de migração também pode ser efetuada de forma deliberada ou autônoma. O processo é similar ao processo de migração entre o servidor principal e um cliente, descrito anteriormente. A diferença é que o agente móvel `ServerAgent`, antes de migrar para o cliente que vai hospedar o servidor, envia o agente `ServerMigrationMessage` para o `MainServer`, avisando-o de que a localização do servidor vai mudar. Além disso, após o `ServerAgent` disparar o novo `ChatServer`, ele precisa retornar ao antigo `ChatServer` do *host* onde partiu e finalizá-lo.

A **migração do cliente para o servidor principal** é caracterizada pelo servidor saindo de um cliente e retornando para o servidor principal (`MainServer`). Como as anteriores, esta migração pode ocorrer de forma deliberada ou autônoma. Em qualquer um dos casos o agente `RestoreServerAgent` é utilizado partindo sempre do cliente que

está hospedando o servidor. Para tanto, o cliente obtém primeiro a lista de participantes através do agente `GrabUserListAgent` que é enviado através do `ClientServer` do cliente que está hospedando o servidor para o `ChatServer`. Quando o `GrabUserListAgent` retorna ao `ClientServer` ele traz consigo a lista de participantes. Com tal lista, o agente `RestoreServerAgent` é enviado para o `MainServer` e o reativa. Após a reativação, o `RestoreServerAgent` migra para o `ChatServer` em execução no cliente e finaliza-o, já que ele não é mais o servidor ativo.

A **migração baseada na tabela de custo** não é mais um tipo de migração, e sim uma forma de acionar a migração do servidor. Aqui a migração ocorre de forma autônoma. Esta autonomia de migração é proporcionada através de variações ocorridas na rede apresentadas a seguir.

O cliente, estando conectado, envia para o servidor o agente `SpawnMigrationAgent` que tem a função de disparar a migração autônoma do servidor caso seja vantajosa. Tal vantagem é verificada através de uma análise na tabela de custo. Dessa forma, sempre que um cliente se conectar, esse agente será enviado para o servidor que poderá ou não migrar. A conexão de um novo usuário pode ser vista como uma alteração no ambiente da rede e o agente `SpawnMigrationAgent` faz com que o servidor responda a esta alteração migrando caso houver necessidade.

Chegando no servidor, o agente `SpawnMigrationAgent` obtém o endereço que apresenta custo médio mínimo em relação a todos os outros nós da rede e, em seguida, retorna ao cliente. Estando no cliente, o agente avalia o endereço obtido e caso houver necessidade a migração é realizada, conforme já descrito nesta mesma seção.

Se o endereço obtido for o endereço do `MainServer` e o servidor itinerante estiver ativo em um nó cliente, então, o agente `SpawnMigrationAgent` faz com que o servidor migre de volta para o `MainServer` através do agente `RestoreAgent`. No caso do endereço obtido no servidor ser de um dos clientes conectados, o agente `SpawnMigrationAgent` dispara o agente `ServerAgent` que faz com que o servidor migre para a máquina do cliente que apresenta o menor custo médio de comunicação na rede. Quando o endereço obtido representa um nó que já esteja hospedando o servidor, nenhuma migração ocorre.

A Seção 5.2.3 apresenta todo o processo de elaboração e aplicação da atividade de teste realizada no *Mobile Chat*, bem como a avaliação dos resultados obtidos.

5.2.3 Atividade de Teste - *Mobile Chat*

Agora o intuito está em apresentar a atividade de teste realizada no *Mobile Chat*. Como já dito, este estudo de caso é muito interessante de ser testado devido a natureza migratória do servidor. Assim, como tal estudo de caso reforça a utilização da mobilidade de código e da API μCODE, esta atividade de teste também visa reforçar o emprego do teste estrutural para validar a corretude de códigos móveis.

Para tanto, foi estabelecido o seguinte plano de teste:

- **1ª Fase -** Criação do conjunto de teste T_{func}, adequado a requisitos funcionais;
- **2ª Fase -** Criação do conjunto de teste T_{estr}, adequado a critérios estruturais;
- **3ª Fase -** Medição da cobertura dos requisitos funcionais, pelo conjunto de casos de teste T_{estr};
- **4ª Fase -** Medição da cobertura dos critérios estruturais, pelo conjunto de casos de teste T_{func};
- **5ª Fase -** Avaliação dos resultados obtidos.

5.2.3.1 *Chat* - 1ª Fase - Cobertura dos Requisitos Funcionais por T_{func}

Ao iniciar a 1ª Fase desta atividade de teste foi proposto o seguinte plano de teste:

1. Especificar os requisitos funcionais.
2. Gerar casos de teste para testar os requisitos funcionais.
3. Aplicar os casos de teste para cobrir 100% os requisitos funcionais especificados.

Com base no plano apresentado, deu-se início a atividade de teste especificando os requisitos funcionais (Apêndice D) do *Mobile Chat*. Em seguida, foi gerado o conjunto de casos de teste T_{func} (Apêndice E) com o intuito de cobrir os requisitos funcionais especificados. Vale mostrar que $T_{func} = \{01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22\}$.

Os casos de teste foram aplicados e obteve-se os seguintes resultados:

- O **caso de teste 01** teve o intuito de realizar uma sessão de *chat*, entre dois clientes, de forma a ocorrer tudo com sucesso. Neste caso de teste foram cobertos os seguintes requisitos funcionais: 05, 14, 16, 20, 23, 24, 25, 26, 29, 36, 41, 42, 43 e nenhum erro foi encontrado.

- O **caso de teste 02**, ao contrário do anterior, visou testar funcionalidades diferentes do habitual. Aqui o servidor foi iniciado em uma porta diferente da porta padrão (1968), em seguida, dois clientes foram conectados e tentaram trocar mensagens vazia ou em branco. Aqui foram cobertos os requisitos funcionais 07, 18 e 19 e nenhum erro foi encontrado.

- O **caso de teste 03** teve o objetivo de verificar se havia o tratamento de erro no caso de informar uma porta inválida na chamada de execução do servidor. Este caso de teste foi executado com sucesso, pois **encontrou um erro**, ou seja, não foi feito o tratamento para a leitura de uma porta inválida.

- O **caso de teste 04** se preocupou em verificar se havia o tratamento de erro no caso de informar mais de quatro parâmetros na chamada de execução do cliente. Vale esclarecer que a chamada de execução do cliente espera receber quatro parâmetros. Este caso de teste obteve êxito, pois **encontrou um erro**, ou seja, não foi feito tratamento para a leitura de mais de quatro parâmetros.

- O **caso de teste 05** visou verificar se existia o tratamento de erro no caso de informar menos de quatro parâmetros na chamada de execução do cliente, já que a chamada espera por quatro. Este caso de teste também obteve êxito, pois **encontrou um erro**, ou seja, não foi feito tratamento para a leitura de menos de quatro parâmetros.

- O **caso de teste 06** teve o intuito de verificar se havia um tratamento de erro ao executar um cliente em uma porta que já estava sendo utilizada. Aqui foi coberto o requisito funcional 03 e nenhum erro foi encontrado.

- O **caso de teste 07** verificou se havia o tratamento de erro no caso de informar um endereço IP, para o cliente na chamada de execução do mesmo, que não exista. Este caso de teste foi executado com sucesso, pois **encontrou um erro**, ou seja, não foi feito tratamento de erro para a leitura de um endereço IP inexistente.

- O **caso de teste 08** visou verificar se havia o tratamento de erro no caso de informar um arquivo de roteamento que não exista. O caso de teste obteve êxito, pois **encontrou um erro**, ou seja, não houve tratamento de erro para a leitura de um arquivo de roteamento inexistente.

- O **caso de teste 09** preocupou-se em verificar se existia o tratamento de erro no caso de informar um arquivo com o formato da especificação de roteamento inadequado. Neste caso de teste também foi **encontrado um erro**, ou seja, não foi feito tratamento para a leitura de um arquivo de roteamento com formato inadequado.

- O **caso de teste 10** verificou se existia o tratamento de erro no caso de informar o endereço IP errado para o servidor na chamada do cliente. Este caso de teste não encontrou nenhum erro e conseguiu cobrir os requisitos funcionais 10, 12 e 15.

- O intuito do **caso de teste 11** estava em verificar se havia o tratamento de erro no caso de informar a porta do servidor, na chamada do cliente, errada. Aqui nenhum erro foi encontrado e os requisitos funcionais 11 e 13 foram cobertos.

- O **caso de teste 12** teve o objetivo de verificar se havia o tratamento de erro em relação ao comando `/nick`, no caso de informar: um apelido com espaço (Jo ão); apenas espaço ao invés do apelido; um apelido igual ao atual; um apelido que já estava sendo utilizado por outro usuário. Nenhum erro foi encontrado e os requisitos funcionais 37, 38, 39 e 40 foram cobertos.

- O **caso de teste 13** teve o objetivo de verificar se havia o tratamento de erro em relação ao comando `/getserver`, no caso de informar: um espaço vazio e uma cadeia de caracter para o parâmetro porta. Apesar de cobrir o requisito funcional 30, **um erro foi encontrado**, pois não houve tratamento de erro ao informar uma cadeia de caracter para o parâmetro porta.

- O **caso de teste 14** verificou se havia o tratamento de erro em relação ao comando `/getserver` no caso de informar a mesma porta, na qual estava sendo executado o `ClientServer`, para o servidor itinerante (`ChatServer`). Aqui **foi encontrado um erro**, pois não houve tratamento ao informar para a porta do `ChatServer` o valor da mesma porta do `ClientServer`, então, a execução do cliente foi interrompida imediatamente.

- O **caso de teste 15** verificou se existia o tratamento de erro, durante a chamada de execução do servidor, no caso da porta padrão estar indisponível. Nenhum erro

foi encontrado e o requisito funcional 06 foi coberto.

- O **caso de teste 16** verificou se havia o tratamento de erro, durante a chamada de execução do servidor, no caso da porta especificada estar indisponível. Nenhum erro foi encontrado e o requisito funcional 08 foi coberto.

- O **caso de teste 17** verificou se existia o tratamento de erro no caso de informar, na caixa de entrada de texto, apenas uma barra "/" ou um comando inexistente. Neste caso de teste nenhum erro foi encontrado e foram cobertos os requisitos funcionais 21 e 22.

- O **caso de teste 18** verificou se existia o tratamento em relação ao comando `/stats`, no caso de invocar tal comando e o servidor estar sendo executado localmente na máquina. Este caso de teste cobriu o requisito 27 e nenhum erro foi encontrado.

- O **caso de teste 19** teve o intuito de verificar se havia o tratamento em relação ao comando `/restore`, no caso de invocar tal comando e o servidor não estar sendo executado localmente na máquina. O requisito funcional 28 foi coberto e nenhum erro foi encontrado.

- O **caso de teste 20** verificou se havia o tratamento em relação ao comando `/getserver` no caso de informar: uma porta com valor inferior à 1024; mais de um valor para o parâmetro porta; e uma porta correta sendo que o cliente já alocava o servidor. Apesar dos requisitos funcionais 33 e 34 serem cobertos, este caso de teste **encontrou um erro**, pois não houve tratamento no caso da porta ser inferior à 1024.

- O **caso de teste 21** verificou o comportamento do cliente ao ser inicializado sem a existência de um servidor em execução. O requisito 44 foi coberto e nenhum erro foi encontrado.

- O **caso de teste 22** verificou qual o comportamento do aplicativo no caso de haver uma queda do servidor ativo e depois uma queda do servidor principal. Em seguida foi realizada a tentiva de troca de mensangens e de invocação de um comando de controle. Os requisitos funcionais 45, 46 e 48 foram cobertos e nenhum erro foi encontrado.

Todos os 9 erros encontrados nesta atividade de teste foram corrigidos. Após a correção de tais erros foram replicados os 22 casos de teste, do conjunto T_{func}, obtendo-se

a seguinte cobertura: no terceiro caso de teste foi coberto o requisito funcional 47, no quarto caso de teste o requisito funcional 02, no quinto o requisito funcional 01, no sétimo o requisito funcional 09, no oitavo o requisito funcional 04, no nono o requisito funcional 17, no décimo terceiro o requisito funcional 32, no décimo quarto o requisito funcional 35, e por fim, o vigésimo caso de teste cobriu o requisito funcional 31.

Assim, com a replicação dos casos de teste do conjunto T_{func}, houve a cobertura de 100% dos requisitos funcionais especificadas para o *Mobile Chat*, conforme mostra a Tabela 5.12.

Tabela 5.12: *Chat* - Cobertura dos Requisitos Funcionais por T_{func}

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
01	05, 14, 16, 20, 23, 24, 25, 26, 29, 36, 41, 42 e 43	13 de 48	27%
02	07, 18 e 19	16 de 48	33%
03	47	17 de 48	35%
04	02	18 de 48	37%
05	01	19 de 48	39%
06	03	20 de 48	41%
07	09	21 de 48	43%
08	04	22 de 48	45%
09	17	23 de 48	47%
10	10, 12 e 15	26 de 48	54%
11	11 e 13	28 de 48	58%
12	37, 38, 39 e 40	32 de 48	66%
13	30, 32	34 de 48	70%
14	35	35 de 48	72%
15	06	36 de 48	75%
16	08	37 de 48	77%
17	21 e 22	39 de 48	81%
18	27	40 de 48	83%
19	28	41 de 48	85%
20	31, 33 e 34	44 de 48	91%
21	44	45 de 48	93%
22	45, 46 e 48	**48 de 48**	**100%**

Esta **primeira fase do teste durou 23 horas**, no qual 6 horas foram utilizadas para especificar os requisitos funcionais, 2 horas para criar os casos de teste, 3 horas para aplicar os casos de teste, 10 horas para correção, 2 horas para replicar os casos de teste e

1 hora para gerar um relatório simples sobre a atividade de teste.

5.2.3.2 *Chat* - 2ª Fase - Cobertura dos Critérios Estruturais por T_{estr}

Nesta segunda fase de teste o seguinte plano foi elaborado:

1. Especificar os critérios estruturais a serem testados.
2. Gerar casos de teste baseados na análise de cobertura provida pela JaBUTi/MA.
3. Aplicar os casos de teste na JaBUTi/MA até conseguir 100% de cobertura dos critérios estruturais, com a exclusão dos itens não executáveis.

Iniciando esta atividade de teste foi estabelecido que seria testado os seguintes critérios estruturais: Todos-Nós, Todas-Arestas e Todos-Usos.

Em seguida, foi gerado o conjunto de casos de teste T_{estr} (Apêndice F) com o auxílio da ferramenta de análise de cobertura da JaBUTi. Assim, $T_{estr} = \{01, 02, 03, 04, 05, 06, 07, 08, 09, 10\}$.

Vale ressaltar que a ferramenta de análise de cobertura da JaBUTi ajuda o testador a gerar casos de teste, pois apresenta em cor vermelha a parte do código mais interessante de ser coberta, sendo que a cobertura da área vermelha implica na cobertura de boa parte do código.

A Tabela 5.13 apresenta 100% de cobertura de todos os critérios estruturais com relação à todo o projeto de teste, ou seja, com relação a todas as classes que compõem o pacote `agents`.

Detalhes sobre a cobertura dos critérios estruturais referentes a cada agente móvel, ou seja, a cada classe do pacote `agents`, estão contidos no Apêndice G.

Esta **segunda fase durou 2 horas**, no qual 1h30min foram utilizadas para criar e aplicar os casos de teste na JaBUTi, e 30 minutos para gerar um relatório simples. Isto com exceção do tempo usado para excluir os itens não executáveis.

Tabela 5.13: *Chat* - Cobertura dos Critério Estruturais do Pacote `agents` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	138 de 199	69%	137 de 208	65%	347 de 559	62%
02	154 de 199	77%	155 de 208	74%	400 de 559	71%
03	155 de 199	77%	157 de 208	75%	410 de 559	73%
04	173 de 199	86%	177 de 208	85%	457 de 559	81%
05	178 de 199	89%	184 de 208	88%	476 de 559	85%
06	181 de 199	90%	189 de 208	90%	506 de 559	90%
07	184 de 199	92%	192 de 208	92%	509 de 559	91%
08	195 de 199	97%	204 de 208	98%	541 de 559	96%
09	**199 de 199**	**100%**	**208 de 208**	**100%**	558 de 559	99%
10					**559 de 559**	**100%**

5.2.3.3 *Chat* - 3ª Fase - Cobertura dos Requisitos Funcionais por T_{estr}

Nesta terceira fase, com base nos conjuntos de casos de teste T_{func} e T_{estr} gerados, foi estipulada uma troca na cobertura, ou seja, agora o interesse está em avaliar a cobertura do conjunto T_{estr} para cobrir os requisitos funcionais. Vale lembrar que o conjunto T_{func} cobriu 100% dos requisitos funcionais.

Assim, foram aplicados os casos de testes do conjunto T_{estr} (Apêndice F) e obteve-se a cobertura apresentada pela Tabela 5.14.

Tabela 5.14: *Chat* - Cobertura dos Requisitos Funcionais por T_{estr}

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
01	05, 07, 14, 16, 24, 42 e 43	07 de 48	15%
02	36 e 48	09 de 48	19%
04	41	10 de 48	21%
05	20	11 de 48	23%
06	40	12 de 48	25%
07	26 e 29	14 de 48	29%
08	25	15 de 48	31%
09	-	15 de 48	31%
10	-	15 de 48	31%

(-) não houve alteração na cobertura

Em uma primeira análise da Tabela 5.14 pode-se deduzir que o conjunto de casos

de teste T_{estr} não foi interessante para cobrir os requisitos funcionais do *Mobile Chat*. Todavia, como os requisitos funcionais especificados se referem à todas as funções do *Mobile Chat*, não há a possibilidade de se conseguir uma cobertura desejável, de todos os requisitos funcionais, já que o conjunto T_{estr} foi gerado na JaBUTi através do teste aplicado somente nos agentes móveis, ou seja, no pacote `agents`.

Como o intuito deste trabalho é testar código móvel, foi selecionado apenas os requisitos que referenciam um agente móvel. Assim, foi constatado que os requisitos de número: 05, 07, 14, 16, 20, 24, 25, 26, 29, 36, 40, 41, 42, 43 e 48 ativam alguma função de mobilidade em um agente móvel.

A Tabela 5.15 apresenta a cobertura dos requisitos funcionais móveis citados. Assim, pode-se verificar que o conjunto de casos de teste T_{estr} obteve 100% de cobertura dos requisitos funcionais móveis, sendo que os oito primeiros casos de teste foram suficientes para obter tal cobertura.

Tabela 5.15: *Chat* - Cobertura dos Requisitos Funcionais Móveis por T_{estr}

Casos de Teste	Requisitos Funcionais Cobertos	Itens Cobertos	Porcentagem de Cobertura
01	05, 07, 14, 16, 24, 42 e 43	07 de 15	47%
02	36 e 48	09 de 15	60%
04	41	10 de 15	67%
05	20	11 de 15	73%
06	40	12 de 15	80%
07	26 e 29	14 de 15	93%
08	25	**15 de 15**	**100%**

Detalhando a cobertura dos requisitos funcionais pelo conjunto de casos de teste T_{estr}, com base na Tabela 5.13, tem-se que $T_{estr\_nós}$ e $T_{estr\_arestas} = \{01, 02, 03, 04, 05, 06, 07, 08, 09\}$; e $T_{estr\_usos} = \{01, 02, 03, 04, 05, 06, 07, 08, 09, 10\}$.

Ao analisar que os oito primeiros casos de teste de T_{estr} foram suficientes para cobrir 100% dos requisitos funcionais, e se os conjuntos derivados de T_{estr} ($T_{estr\_nós}$, $T_{estr\_arestas}$ e T_{estr_usos}) apresentam casos de teste além dos oito primeiros, então, pode-se confirmar que para todos os conjuntos derivados de T_{estr} obtem-se 100% de cobertura dos requisitos funcionais.

A **terceira fase durou 1 hora e 30 minutos**, sendo que 1 hora foi utilizada para a aplicação dos casos de teste estruturais e avaliação da cobertura dos requisitos

funcionais, e 30 minutos para a elaboração de um relatório simples.

5.2.3.4 *Chat* - 4ª Fase - Cobertura dos Critérios Estruturais por T_{func}

A quarta fase desta atividade de teste contempla a aplicação do conjunto de casos de teste T_{func} na JaBUTi/MA, com o intuito de cobrir os critérios estruturais: Todos-Nós, Todas-Arestas e Todos-Usos.

A Tabela 5.16 apresenta a cobertura dos critérios estruturais obtida pelo conjunto T_{func}, e mostra que a maior cobertura obtida para o pacote `agents` foi de 86% no critério Todos-Nós.

Tabela 5.16: *Chat* - Cobertura dos Criterios Estruturais por T_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	170 de 199	85%	171 de 208	82%	431 de 559	77%
02	-	-	-	-	-	
03	-	-	-	-	-	
04	-	-	-	-	-	
05	-	-	-	-	-	
06	-	-	-	-	-	
07	-	-	-	-	-	
08	-	-	-	-	-	
09	-	-	-	-	-	
10	-	-	-	-	-	
11	-	-	-	-	-	
12	172 de 199	86%	175 de 208	84%	446 de 559	79%
13	-	-	-	-	-	
14	-	-	-	-	447 de 559	79%
15	-	-	-	-	-	
16	-	-	-	-	-	
17	-	-	-	-	-	
18	-	-	-	-	-	
19	-	-	-	-	-	
20	-	-	-	-	-	
21	-	-	-	-	-	
22	-	-	-	-	-	

(-) não houve alteração na cobertura

Para maiores detalhes, o Apêndice I apresenta a cobertura dos critérios estruturais referente a cada classe contida no pacote `agents`, ou seja, cada agente móvel pertencente ao *Mobile Chat*.

Através da avaliação da Tabela 5.17, verifica-se que o conjunto de casos de teste T_{func} conseguiu cobrir 100% dos critérios estruturais para os agentes: `SimpleMessageAgent`, `NickChangeMessage`, `GrabUserListAgent` e `PrintRountingTable`. Em contrapartida, o agente `ServerLeavingMessage` não obteve cobertura em nenhum critério. Ainda na Tabela 5.17, pode-se verificar claramente que a maioria dos agentes não obteve cobertura total dos critérios estruturais.

Tabela 5.17: *Chat* - Cobertura dos Criterios Estruturais dos Agentes Móveis por T_{func}

Agentes Analisados	Todos-Nós		Todas-Arestas		Todos-Usos	
agents.LeaveChatSession	**21 de 21**	**100%**	**24 de 24**	**100%**	59 de 66	89%
agents.SimpleMessageAgent	**03 de 03**	**100%**	**01 de 01**	**100%**	**02 de 02**	**100%**
agents.ServerLeavinfMessage	00 de 03	0%	00 de 01	0%	00 de 00	0%
agents.ServerAgent	20 de 26	76%	20 de 28	71%	66 de 85	77%
agents.NickChangeMessage	**17 de 17**	**100%**	**19 de 19**	**100%**	**52 de 52**	**100%**
agents.GrabUserListAgent	**05 de 05**	**100%**	**04 de 04**	**100%**	**06 de 06**	**100%**
agents.PrintRoutingTable	**11 de 11**	**100%**	**12 de 12**	**100%**	**32 de 32**	**100%**
agents.MobileMessage	07 de 12	58%	04 de 11	36%	05 de 24	20%
agents.RestoreServerAgent	11 de 13	84%	12 de 13	92%	36 de 42	85%
agents.SpawnMigrationAgent	10 de 15	66%	10 de 18	55%	22 de 49	44%
agents.ServerMigrationMessage	06 de 07	85%	04 de 06	66%	08 de 11	72%
agents.NickListMessage	**24 de 24**	**100%**	23 de 24	95%	**49 de 49**	**100%**
agents.JoinChatSession	37 de 42	88%	42 de 47	89%	110 de 141	78%

Esta **quarta fase durou cerca de 3 horas e 30 minutos**, sendo que 1 hora foi utilizada para a aplicação dos casos de testes na JaBUTi e 2h30min para a avaliação dos resultados e criação de um relatório simples.

Com o intuito de cobrir totalmente os critérios estruturais de todos os agentes móveis, foram gerados mais casos de teste criando-se o conjunto T'_{func} (Apêndice H). Os novos casos de teste foram gerados com base na análise de cobertura da Ferramenta JaBUTi. Vale mostrar que $T'_{func} = \{a, b, c, d, e, f, g, h\}$.

A apresentação detalhada da cobertura dos critérios estruturais pelo conjunto T'_{func} consta no Apêndice I, no qual os casos de teste identificados por letras são os casos

de testes adicionais, ou seja, os casos de teste pertencentes ao conjuntos T'_{func}.

A cobertura dos critérios estruturais com adição do conjunto T'_{func} é apresentada pela Tabela 5.18.

Tabela 5.18: *Chat* - Cobertura dos Criterios Estruturais do pacote `agents` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	170 de 199	85%	171 de 208	82%	431 de 559	77%
12	172 de 199	86%	175 de 208	84%	446 de 559	79%
14	-	-	-	-	447 de 559	79%
a	177 de 199	88%	182 de 208	87%	465 de 559	83%
b	184 de 199	92%	189 de 208	90%	496 de 559	88%
c	185 de 199	92%	190 de 208	91%	501 de 559	89%
d	189 de 199	94%	194 de 208	93%	517 de 559	92%
e	198 de 199	99%	205 de 208	98%	539 de 559	96%
f	**199 de 199**	**100%**	207 de 208	99%	545 de 559	97%
g			**208 de 208**	**100%**	-	-
h					**559 de 559**	**100%**

(-) não houve alteração na cobertura

Para a geração dos casos de teste adicionais (T'_{func}), com o intuito de concluir a cobertura dos critérios estruturais, **foram despendidas 2 horas**, sendo que 1h30min foram utilizados para analisar o código, gerar os casos de teste adicionais e aplicá-los na Ferramenta JaBUTi, e 30 minutos para gerar uma documentação simples.

5.2.3.5 *Chat* - 5ª Fase - Comparação dos Resultados

Agora a intenção é de comparar os conjuntos de casos de teste gerados. Assim, temos que no item 5.2.3.1 foram gerados casos de teste para cobrir 100% dos requisitos funcionais criando-se, para tanto, o conjunto de casos de teste T_{func} (Apêndice E). E o item 5.2.3.2 apresentou o conjunto de casos de teste T_{estr} (Apêndice F), gerado pela JaBUTi/MA, que obteve 100% de cobertura dos critérios estruturais.

Com base nos conjuntos de casos de teste T_{func} e T_{estr} foi estipulada uma troca na cobertura, ou seja, o item 5.2.3.3 apresentou a cobertura dos requisitos funcionais pelo conjunto T_{estr} e o item 5.2.3.4 apresentou a cobertura dos critérios estruturais por T_{func}

e T'_{func}.

Avaliando a troca percebeu-se que o conjunto T_{func}, inicialmente gerado para cobrir os requisitos funcionais, apesar de ter coberto 100% destes, quando aplicado para cobrir os critérios estruturais apresentou uma cobertura, conforme Tabela 5.16, de 86% do critério Todos-Nós, 84% para Todas-Arestas e 79% para Todos-Usos. Assim, pode-se verificar que não houve critério estrutural com 100% de cobertura.

Portanto, o conjunto T_{func} não apresentou uma cobertura satisfatória com relação aos critérios estruturais. Continuando, teve que ser gerado mais oito casos de teste (T'_{func}) para concluir a cobertura dos critérios estruturais iniciada pelo conjunto T_{func}.

Com base no conjunto T_{estr}, gerado pela ferramenta JaBUTi/MA para cobrir os critérios estruturais, verificou-se que além de ter obtido 100% de cobertura dos critérios estruturais, também conseguiu obter 100% de cobertura dos requisitos funcionais. Ao detalhar a cobertura, com base em cada critério estrutural, verificou-se que os conjuntos $T_{estr\_nós}$, $T_{estr\_arestas}$ e T_{estr_usos} (derivados de T_{estr}) obtiveram 100% de cobertura dos requisitos funcionais.

5.2.3.6 Conclusão da Atividade de Teste - *Mobile Chat*

Mais uma vez foi comprovado que o uso da JaBUTi como ferramenta de apoio ao teste foi muito importante, devido a mesma oferecer, através da análise de cobertura, uma forma simples de gerar os casos de teste, de forma a cobrir a maior parte do código sem a morosidade verificada na aplicação do teste funcional.

Com esta atividade de teste, verificou-se que a JaBUTi é muito rica em informações referente ao estado atual da atividade de teste, dando suporte ao testador para verificar o grau de dificuldade para se cobrir uma determinada linha de código e facilitando a identificação de itens não executáveis.

Os grafos da JaBUTi são bem montados e apresentam informações sobre cada nó, aresta e *bytecode* que o compõem e, ainda apresenta a situação atualizada da cobertura de cada método das classes instrumentadas.

A atividade de teste ficou fácil de ser realizada com o apoio da JaBUTi, não foi tedioso e nem estressante como o teste funcional realizado. O teste funcional manual é

custoso e exige muita atenção por parte do testador com relação a cobertura de cada requisito funcional.

Analisando as atividades de teste realizadas no *Mobile Chat*, observou-se que o tempo gasto na atividade de teste funcional foi superior ao teste estrutural auxiliado pela JaBUTi.

Na questão de cobertura pode-se verificar, na Tabela 5.19, que a maior cobertura dos critérios estruturais obtida pelo conjunto de casos de teste T_{func} foi de 86% para o critério Todos-Nós. Quando comparado com a cobertura (Tabela 5.20) do conjunto T_{estr}, gerado durante o teste estrutural com o auxílio da JaBUTi/MA, verificou-se que todos conjuntos derivados de T_{estr} apresentaram 100% de cobertura dos requisitos funcionais.

Tabela 5.19: *Chat* - Cobertura Geral dos Criterios Estruturais por T_{func}

	Cobertura dos Critérios Estruturais
Todos-Nós	86%
Todas-Arestas	84%
Todos-Usos	79%

Tabela 5.20: *Chat* - Cobertura Geral dos Requisitos Funcionais Móveis pelos Derivados de T_{estr}

	Cobertura dos Requisitos Funcionais
$T_{estr\_nós}$	100%
$T_{estr\_arestas}$	100%
T_{estr_usos}	100%

Portanto, acredita-se que esta atividade de teste mostrou claramente que o teste estrutural em agentes móveis é uma excelente técnica de teste para mostrar a corretude de um código móvel, e, ainda, reforçou a utilização da Ferramenta JaBUTi/MA como base de auxílio para testar agentes móveis desenvolvidos em Java com a API μCODE.

6 Conclusão

Esta dissertação teve por objetivo estudar e avaliar formas para apoiar o teste em código móvel utilizando-se para tanto os critérios de teste estrutural. Em particular, foram focados os critérios de fluxo de dados e fluxo de controle, que a Ferramenta JaBUTi suporta.

Também possibilitou verificar o comportamento do código móvel enquanto ele se movimentava pelos nós de uma rede, capturando informações de seu estado em cada nó, do trajeto que percorreu, e quais partes do código foram executadas.

A utilização de uma ferramenta de teste neste trabalho tornou-se imprescindível, pois permitiu a avaliação dos testes aplicados. Permitindo, também, avaliar a própria Ferramenta JaBUTi, como sendo uma excelente ferramenta de teste estrutural não só de programas e componentes Java, mas, também, de programas que suportam mobilidade de código desenvolvidos pela μCODE .

Pôde-se verificar, com a aplicação das atividades de teste funcional e estrutural nos estudos de casos aqui apresentados, que os casos de teste gerados com o auxílio da Ferramenta JaBUTi/MA com base no teste estrutural foi superior aos gerados com base no teste funcional. Pois, apresentaram maior cobertura, durante a trajetória de movimentação do agente móvel, tanto para cobrir os critérios estruturais como para cobrir os requisitos funcionais.

A Ferramenta JaBUTi além de auxiliar a geração de casos de teste, também se mostrou uma ferramenta robusta, caracterizando-se pela facilidade de uso de suas funcionalidades e pela gama de informações capturadas e gerenciadas pela mesma.

Com as atividades de teste realizadas neste trabalho pode-se concluir que a adaptação da Ferramenta JaBUTi para testar agente móvel (JaBUTi/MA) é totalmente aplicável para a realização de testes em código móvel Java, pois conseguiu capturar in-

formações dinâmicas de *trace* durante a trajetória executada pelo agente móvel em todos os *hosts* percorridos pelo mesmo.

Assim, através desta dissertação, espera-se contribuir com pesquisas referentes a mobilidade de código, principalmente em relação aos estudos empíricos referente a aplicação de critérios de testes estruturais para revelar erros em código móvel utilizando-se de outras plataformas e/ou outras abordagens de mobilidade de código diferente das de agentes móveis aqui apresentadas.

Espera-se, também, desmistificar a tecnologia de código móvel, auxiliando na divulgação de seus mecanimos e linguagens, principalmente da API μCODE. Proporcionando, através da verificação da confiabilidade e da funcionalidade do código móvel, a construção de aplicações distribuídas baseadas em mobilidade de código com qualidade.

Este trabalho permitirá que sejam estudados a cobertura de código móvel para outras linguagens e ambientes que suportam mobilidade, através da aplicação de agentes móveis de teste que percorrem o mesmo caminho que um agente móvel comum realiza, possibilitando a captura de informações de *trace*.

Por fim, confirmou-se que a utilização de teste estrutural em código móvel é uma forma eficaz de mostrar a corretitude de um sistema baseado em mobilidade de código. E, em acréscimo, contribuiu para adaptar a Ferramenta JaBUTi/MA e validar tal adaptação.

Referências

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML Guia do Usuário*. Rio de Janeiro - RJ: Campus, 2000.

CARVALHO, A. M. B. R.; CHIOSI, T. C. S. *Introdução a Engenharia de Software*. Campinas - SP: Unicamp, 2001.

CASTRO, M. A. S. *Tutorial HTML do ICMC-USP*. 1995. Website (URL: http://www.icmc.usp.br/ensino/material/html/).

CHAIM, M. L. *POKE-TOOL - Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados*. Campinas, SP: Universidade Estadual de Campinas - UNICAMP, 1991.

CHAN, M. C.; GRIFFITH, S. W.; IASI, A. F. *Java - 1001 Dicas de Programação*. São Paulo - SP: Makron Books, 1999.

DEITEL, H. M.; DEITEL, P. J. *Java - Como Programar*. 3a. ed. Porto Alegre, RS: Bookman, 2001.

DELAMARO, M. E. *Automatização do Teste Estrutural de Agentes Móveis*. São Carlos, SP: ICMC-USP, 2005.

DELAMARO, M. E.; VINCENZI, A. M. R.; MALDONADO, J. C. Jabuti/ma: An environment to test mobile agents. Preprint submited to Elsevier Science, Marília - SP, August 2004.

DEMILLO, R. A.; LIPTON, R. J.; SAYWARD, F. G. *Hints on Test Data Selection: Help for the Practicing Programmer*. [S.l.]: Computer, 1978.

DUARTE, L. M.; DOTTI, F. L. *Desenvolvimento de Aplicações Móveis Corretas*. Porto Alegre, RS: Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS, 2001.

FRANKL P. G.; WEISS, S. N.; WEYUKER, E. J. Asset: A system to select and evaluate tests. *IEEE Conference on Software Tools*, IEEE, p. 72–79, April 1985.

FUGGETTA, A.; PICCO, G. P.; VIGNA, G. Understanding code mobility. *IEEE Transactions on Software Engineering*, IEEE, v. 24, n. 5, May 1998.

HORGAN, J. R.; LONDON, S.; LYU, M. R. Achieving software quality with testing coverage measures. *IEEE Transactions on Software Engineering*, IEEE, v. 27, n. 9, p. 60–69, September 1994.

INC., S. R. *TCAT - Test Coverage Analysis Tool*. May 2002. Website(URL: http://www.soft.com/TestWorks/index.html).

INC., S. R. *TCAT-PATH (Path Test Coverage Analysis Tool)*. May 2002. Website(URL: http://www.sgi.com/products/appsdirectory.dir/apps/app_number1461.html).

INC., T. T. *Telcordia Software Visualization and Analysis Toolsuite - xSuds*. 1998. Website(URL: http://xsuds.argreenhouse.com).

LI, J. J. et al. χsuds-sdl: A tool for diagnosis and understanding software specifications. *FastAbstract ISSRE*, Telcordia Technologies, 1999.

LINDHOLM, T.; YELLIN, F. *The Java Virtual Machine Specification*. 2ª. ed. [S.l.]: Addison-Wesley, 1999.

MALDONADO, J. C. *Critérios Potenciais Usos: Uma contribuição ao Teste Estrutural de Software*. Campinas, SP: DCA/FEE/UNICAMP, 1991.

MCCABE, T. J.; BUTLER, C. W. Design complexity measurement and testing. *Communications of the ACM*, ACM Press, New York, NY, USA, v. 32, n. 12, p. 1415–1425, December 1989.

MYERS, G. J. *The Art of Software Testing*. New York, NY, USA: John Wiley Sons, Inc., 1979. (ISBN 0471043281).

PICCO, G. P. μCODE: A lightweight and flexible mobile code toolkit. In: *Mobile Agents - Proceedings of the 2nd International Workshop on Mobile Agents*. Stuttgart (Germany): K. Rothermel and F. Holh, 1998. (ISBN 3-540-64959-X, v. 1477), p. 160–171.

PICCO, G. P. Understanding code mobility. *Proceedings of the 22th International Conference on Software Engineering*, International Conference on Software Engineering, 2000.

PICCO, G. P. Mobile agents: An introduction. *Journal of Microprocessors and MicroSystems*, Elsevier Science, 2001.

PICCO, G. P. *A Mobile Code Toolkit*. April 2001. Website (URL: http://mucode.sourceforge.net).

PICCO, G. P. et al. Analyzing mobile code languages. *Mobile Object Systems: Towards the Programmable Internet*, Mobile Object Systems: Towards the Programmable Internet, 1997.

PRESSMAN, R. S. *Engenharia de Software*. 3ª. ed. São Paulo, SP: MAKRON Books, 1995.

RAPPS, S.; WEYUKER, E. J. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, IEEE Press, v. 11, n. 4, p. 367–375, April 1985.

SILVA, J. *Proteste+: Ambiente de Validação Automática de Qualidade de Software através de Técnicas de Teste e de Métricas de Complexidade*. Porto Alegre, RS: Master Thesis, CPGCC-UFRS, 1995.

SPOTO, E. S.; PERES, L. M.; BUENO, P. M. S. *Um Estudo de Critérios de Teste de Software Baseados em Fluxo de Dados*. Campinas, SP: UNICAMP, 1995.

TANENBAUM, A. S.; STEEN, M. *Distributed Systems - Principles and Paradigms*. Upper Saddle River, New Jersey: Prentice Hall, 2002.

VINCENZI, A. M. R. et al. *JaBUTi - Java Bytecode Understanding and Testing*. São Carlos, SP: Universidade de São Paulo - USP, 2003.

ZHAO, J. Analyzing control flow in java bytecode. *In 16th Conference of Japan Society for Software Science and Technology*, p. 313–316, September 1999.

ZHAO, J. Dependence analysis of java bytecode. *In 24th IEEE annual International Computer Software and Applications Conference (COMPSAC'2000)*, IEEE Computer Society Press, Taipei, Taiwan, p. 486–491, October 2000.

APÊNDICE A – Bozó - Requisitos Funcionais

Este apêndice apresenta a especificação dos requisitos funcionais (estratégias) do jogo Bozó, conforme segue:

1. Se saírem cinco dados iguais e o placar FIVE_OF_KIND estiver livre, marca-o e finaliza a jogada.
2. Se saírem cinco dados iguais, o placar FIVE_OF_KIND estiver ocupado, e o placar BIG_SEQ estiver livre, marca-o e finaliza a jogada.
3. Se saírem cinco dados iguais, os placares FIVE_OF_KIND e BIG_SEQ estiverem ocupados, o placar SMALL_SEQ estiver livre e for a última rolada, marca o placar SMALL_SEQ e finaliza a jogada.
4. Se saírem cinco dados iguais, os placares FIVE_OF_KIND, BIG_SEQ e SMALL_SEQ estiverem ocupados e for a última jogada, e o placar FULL_HAND estiver livre, marca-o e finaliza a jogada.
5. Se saírem cinco dados iguais, os placares FIVE_OF_KIND e BIG_SEQ estiverem ocupados, não for a última jogada e o placar FULL_HAND estiver livre, marca-o e finaliza a jogada.
6. Se sair uma seqüência grande e o placar BIG_SEQ estiver livre, marca-o e finaliza a jogada.
7. Se for a terceira rolada e sair uma seqüência grande, se o placar BIG_SEQ estiver ocupado e o placar SMALL_SEQ estiver livre, marca o SMALL_SEQ e finaliza a jogada.
8. Se for a última rolada, sair uma seqüência pequena e o placar SMALL_SEQ estiver livre, marca-o e finaliza a jogada.
9. Se saírem dois dados de um valor e três dados de outro valor e o placar FULL_HAND estiver livre, marca-o e finaliza a jogada.
10. Se for a primeira ou a segunda rolada e sair qualquer seqüencia, de qualquer tamanho, constando o valor 2, e o placar SMALL_SEQ estiver livre, mantém no máximo três dados em seqüencia e continua jogando.
11. Se for a primeira ou a segunda rolada e saírem duas duplas e o placar FULL_HAND estiver livre, mantém as duplas e continua jogando.

12. Se for a primeira ou a segunda rolada, os dados apresentarem alguma repetição, e os valores que mais se repetirem forem 5 ou 6, e os placares correspondentes (FIVES ou SIXES), ou o CHANCE, ou FOUR_OF_KIND, ou THREE_OF_KIND estiverem livres, mantém os dados de maior repetição, ou os dados de maior valor (caso o placar correspondente aos dados de maior repetição já tenha sido marcado), e continua jogando.

13. Se for a primeira ou a segunda rolada, os dados apresentarem alguma repetição, e os valores que mais se repetirem forem 1, 2, 3 ou 4 e os placares correspondentes (ONES, TWOS, THREES ou FOURS), ou FOUR_OF_KIND, ou THREE_OF_KIND estiverem livres, mantém os dados de maior repetição e continua jogando (se houver duas duplas, mantém a dupla maior).

14. Se for a primeira ou a segunda rolada, não houver repetições, e o placar CHANCE, ou FOUR_OF_KIND, ou THREE_OF_KIND, ou o placar correspondente (entre ONES e SIXES) estiver livre, guarda o dado de maior valor numérico e continua jogando.

15. Se for a primeira ou a segunda rolada, houver repetições, mas os placares entre ONES e SIXES correspondentes aos valores das repetições estiverem ocupados, e os placares THREE_OF_KIND e FOUR_OF_KIND também estiverem ocupados, mantém apenas os dados de maior valor entre todos, mesmo que seja uma repetição, e continua jogando.

16. Se for a terceira rolada, sair repetições de um determinado valor, e o placar correspondente àquele valor entre ONES e SIXES estiver livre, marca-o.

17. Se for a terceira rolada de dados e existir pelo menos uma repetição, verifica se o placar THREE_OF_KIND ou o FOUR_OF_KIND está livre. Se estiver, e se a maior pontuação possível com os OF_KINDs livres for maior do que a pontuação possível no placar numérico correspondente ao valor da repetição, e o CHANCE estiver ocupado; ou a pontuação do OF_KIND for maior do que a do CHANCE, marca o OF_KIND.

18. Se for a terceira rolada, existir pelo menos uma repetição, e não tiver nenhum placar correspondente aos valores das repetições, entre ONES e SIXES livre, nem o THREE_OF_KIND ou FOUR_OF_KIND, nem o CHANCE, então descarta o primeiro placar entre ONE e SIXES que achar livre.

19. Se for a terceira rolada e o CHANCE ainda estiver livre, e a soma dos valores for a maior pontuação possível em relação aos placares que estão livres, marca o CHANCE.

20. Se for a terceira rolada e não existir nenhum placar entre ONES e SIXES livre, e o THREE_OF_KIND e FOUR_OF_KIND estiverem ocupados e o CHANCE também estiver ocupado, então descarta o placar que esteja livre.

21. Se for a primeira ou a segunda rolada e nenhuma das regras se aplicar, então rola novamente os dados e continua jogando.

22. Se for a terceira rolada e nenhuma opção for satisfeita, descarta a opção que esteja livre.

23. Se o valor da soma dos 6 primeiros elementos (de ONES até SIXES) no placar final for maior ou igual a 63, então é acrescentado um bônus[1.1] extra de 35 pontos ao total.

[1.1] O valor 63 corresponde a obter três repetições em cada uma das entradas de ONES até SIXES.

24. Se for a primeira ou a segunda rolada e sair qualquer seqüência pequena, porém não consta o valor 2, e o placar SMALL_SEQ estiver livre, mantém no máximo dois dados do início da seqüência e continua jogando.

APÊNDICE B – Bozó - Conjunto de Casos de Teste T_{func}

Este apêndice apresenta, dos 150 casos de teste gerados, somente os que obtiveram êxito na cobertura dos requistos funcionais do jogo Bozó. Tais casos de teste foram os de número: 01, 02, 03, 54 e 61. Sendo assim, tem-se que $T_{func} = \{01, 02, 03, 54, 61\}$.

Vale esclarecer que o resultado da execução de cada caso de teste é apresentado da seguinte forma. Primeiro é apresentada a pontuação obtida em cada item do placar (de *Ones* a *Bonus*), bem como o total das pontuações obtidas (*Total*). Em seguida, é apresentado o resultado de cada *Round* do jogo, perfazendo um total de 13 *rounds*.

Em cada *round* é apresentado o resultado da rolagem (*Roll*) dos dados, os dados mantidos (*Kept*) e, por fim, a marcação (*Placed on*) de um item do placar.

CASO DE TESTE 01

```
Ones.........:4
Twos.........:6
Threes.......:3
Fours........:16
Fives........:20
Sixes........:12
3 of a kind..:23
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:0
5 of a kind..:50
Chance.......:21
Bonus........:0
Total........:210
=========================

***** Round 0
Roll #0: 6  4  4  4  5
Kept #0:    X  X  X
Roll #1: 4  4  4  4  6
Kept #1: X  X  X  X
Roll #2: 4  4  4  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 1
Roll #0: 1  2  5  6  6
Kept #0:          X  X
Roll #1: 2  5  5  6  6
Kept #1:    X  X  X  X
Roll #2: 6  5  5  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 2  3  5  5  5
Kept #0:       X  X  X
Roll #1: 3  3  5  5  5
Kept #1:       X  X  X
Roll #2: 5  6  5  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 3
Roll #0: 6  1  4  5  6
Kept #0: X          X
Roll #1: 6  2  3  5  6
Kept #1: X          X
Roll #2: 6  1  4  4  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 4
Roll #0: 6  5  5  4  5
Kept #0:    X  X     X
Roll #1: 4  5  5  3  5
Kept #1:    X  X     X
Roll #2: 4  5  5  4  5
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 5
Roll #0: 6  3  2  5  2
Kept #0:       X     X
Roll #1: 1  2  2  3  2
Kept #1:    X  X     X
Roll #2: 6  2  2  1  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 6
Roll #0: 6  2  1  1  5
Kept #0:       X  X
Roll #1: 1  1  1  1  3
Kept #1: X  X  X  X
Roll #2: 1  1  1  1  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 7
Roll #0: 2  4  5  4  1
Kept #0:    X     X
Roll #1: 3  4  3  4  5
Kept #1:    X     X
Roll #2: 4  4  1  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 8
Roll #0: 1  3  1  5  2
Kept #0: X     X
Roll #1: 1  3  1  1  5
Kept #1: X     X  X
Roll #2: 1  6  1  1  6
Kept #2: ?  ?  ?  ?  ?
```

```
Placed on "Sixes........:"

***** Round 9
Roll #0: 5  5  1  5  3
Kept #0: X  X     X
Roll #1: 5  5  1  5  6
Kept #1: X  X     X
Roll #2: 5  5  4  5  4
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 10
Roll #0: 6  5  4  5  5
Kept #0:
Roll #1: 3  1  4  5  1
Kept #1: X
Roll #2: 3  6  2  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 11
Roll #0: 4  3  2  2  4
Kept #0:
Roll #1: 6  5  4  1  1
Kept #1: X
Roll #2: 6  2  1  6  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 12
Roll #0: 3  6  4  6  5
Kept #0: X     X
Roll #1: 3  5  4  6  3
Kept #1: X     X
Roll #2: 3  6  4  4  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"
```

CASO DE TESTE 02

```
Ones.........:4
Twos.........:4
Threes........:9
Fours.........:8
Fives.........:15
Sixes.........:24
3 of a kind..:0
4 of a kind..:8
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance........:17
Bonus.........:35
Total.........:219
==========================

***** Round 0
Roll #0: 2  4  1  1  6
Kept #0:       X  X
Roll #1: 1  5  1  1  1
Kept #1: X     X  X  X
Roll #2: 1  6  1  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 1
Roll #0: 2  3  5  4  3
Kept #0: X  X     X
Roll #1: 2  3  6  4  3
Kept #1:    X        X
Roll #2: 4  3  6  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 2
Roll #0: 5  1  6  2  1
Kept #0:    X        X
Roll #1: 6  1  6  1  1
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 3
Roll #0: 2  3  3  4  6
Kept #0:    X  X
Roll #1: 4  3  3  2  4
Kept #1: X           X
Roll #2: 4  6  3  2  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 4
Roll #0: 3  1  5  5  5
Kept #0:       X  X  X
Roll #1: 4  6  5  5  5
Kept #1:       X  X  X
Roll #2: 4  4  5  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 5
Roll #0: 1  1  1  1  3
Kept #0: X  X  X  X
Roll #1: 1  1  1  1  3
Kept #1: X  X  X  X
Roll #2: 1  1  1  1  4
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 6
Roll #0: 6  5  5  3  1
Kept #0: X
Roll #1: 6  6  6  6  5
Kept #1: X  X  X  X
Roll #2: 6  6  6  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 7
Roll #0: 1  4  1  3  1
Kept #0:          X
Roll #1: 1  1  2  3  1
Kept #1:          X
Roll #2: 3  6  2  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 8
Roll #0: 5  4  5  1  2
Kept #0:
Roll #1: 4  5  2  2  6
Kept #1:
Roll #2: 5  6  2  2  6
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 9
Roll #0: 5  3  5  5  4
Kept #0:    X
Roll #1: 4  3  2  5  6
Kept #1: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 10
Roll #0: 2  5  5  6  6
Kept #0:
Roll #1: 2  1  5  3  2
Kept #1:          X
Roll #2: 1  3  2  3  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 11
Roll #0: 2  3  1  6  4
Kept #0:    X
Roll #1: 4  3  1  3  6
Kept #1:    X     X
Roll #2: 2  3  4  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 12
Roll #0: 5  4  5  4  5
Kept #0:
Roll #1: 3  1  3  2  4
Kept #1:
Roll #2: 2  5  1  3  5
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"
```

CASO DE TESTE 03

```
Ones.........:3
Twos.........:4
Threes........:6
Fours.........:12
Fives.........:5
Sixes.........:18
3 of a kind..:0
4 of a kind..:9
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance........:21
Bonus.........:0
Total.........:173
==========================

***** Round 0
Roll #0: 6  5  2  6  2
Kept #0: X     X  X  X
Roll #1: 6  1  2  6  2
Kept #1: X     X  X  X
Roll #2: 6  5  2  6  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 5  1  4  6  1
Kept #0:    X        X
Roll #1: 5  1  1  4  1
Kept #1:    X  X     X
Roll #2: 6  1  1  3  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 2
Roll #0: 2  6  4  5  2
Kept #0: X           X
Roll #1: 2  4  1  1  2
Kept #1: X     X  X  X
Roll #2: 2  1  1  1  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 3
Roll #0: 3  6  1  2  2
Kept #0:          X  X
Roll #1: 3  5  6  2  2
Kept #1:          X  X
Roll #2: 4  4  5  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 4
Roll #0: 3  5  6  3  1
Kept #0: X        X
Roll #1: 3  4  5  3  6
Kept #1: X  X
Roll #2: 3  4  1  6  2
Kept #2: ?  ?  ?  ?  ?
```

```
Placed on "Small seq....:"

***** Round 5
Roll #0: 1  3  1  4  3
Kept #0:    X        X
Roll #1: 4  3  1  5  3
Kept #1:    X        X
Roll #2: 2  3  6  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 6
Roll #0: 2  1  2  4  5
Kept #0: X     X
Roll #1: 2  1  2  3  1
Kept #1: X     X
Roll #2: 2  2  2  2  1
Kept #2: ?  ?  ?  ?  ?
```

```
Placed on "4 of a kind..:"

***** Round 7
Roll #0: 4  6  2  3  1
Kept #0:    X
Roll #1: 3  6  5  4  4
Kept #1:          X  X
Roll #2: 6  4  3  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 8
Roll #0: 6  2  1  6  1
Kept #0: X        X
Roll #1: 6  6  4  6  4
Kept #1: X  X     X
Roll #2: 6  6  2  6  2
Kept #2: ?  ?  ?  ?  ?
```

```
Placed on "Sixes........:"

***** Round 9
Roll #0: 5  4  2  2  1
Kept #0: X
Roll #1: 5  2  6  6  3
Kept #1: X
Roll #2: 5  2  1  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 10
Roll #0: 5  1  6  5  2
Kept #0:       X
Roll #1: 2  4  6  4  3
Kept #1:       X
Roll #2: 2  2  6  4  6
Kept #2: ?  ?  ?  ?  ?
```

```
Placed on "3 of a kind..:"

***** Round 11
Roll #0: 5  3  5  6  3
Kept #0:          X
Roll #1: 1  3  2  6  4
Kept #1:          X
Roll #2: 1  5  3  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 12
Roll #0: 5  4  5  2  2
Kept #0:
Roll #1: 3  4  1  2  5
Kept #1: ?  ?  ?  ?  ?
Placed on "Large seq....:"
```

CASO DE TESTE 54

```
Ones.........:4
Twos.........:8
Threes.......:12
Fours........:4
Fives........:15
Sixes........:18
3 of a kind..:0
4 of a kind..:9
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance.......:10
Bonus........:0
Total........:175
==========================

***** Round 0
Roll #0: 1  3  1  3  2
Kept #0: X  X  X  X
Roll #1: 1  3  1  3  2
Kept #1: X  X  X  X
Roll #2: 1  3  1  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 4  2  6  1  3
Kept #0: X  X        X
Roll #1: 4  2  1  5  3
Kept #1: ?  ?  ?  ?  ?
Placed on "Large seq....:"
```

```
***** Round 2
Roll #0: 1  1  3  1  5
Kept #0: X  X     X
Roll #1: 1  1  6  1  5
Kept #1: X  X     X
Roll #2: 1  1  5  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 3
Roll #0: 5  6  4  6  5
Kept #0: X  X     X  X
Roll #1: 5  6  6  6  5
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 4
Roll #0: 5  2  2  4  1
Kept #0:    X  X
Roll #1: 2  2  2  6  3
Kept #1: X  X  X
Roll #2: 2  2  2  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 5
Roll #0: 4  5  6  1  3
Kept #0: X           X
Roll #1: 4  1  6  6  3
Kept #1:       X  X
Roll #2: 1  3  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"
```

```
***** Round 6
Roll #0: 4  5  6  6  2
Kept #0:       X  X
Roll #1: 3  4  6  6  3
Kept #1:       X  X
Roll #2: 5  2  6  6  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 7
Roll #0: 5  2  4  2  2
Kept #0:    X     X  X
Roll #1: 6  2  3  2  2
Kept #1:    X     X  X
Roll #2: 2  2  1  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 5  6  3  5  3
Kept #0: X        X
Roll #1: 5  1  2  5  1
Kept #1: X        X
Roll #2: 5  5  2  5  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 9
Roll #0: 3  4  5  6  3
Kept #0: X  X
Roll #1: 3  4  2  3  2
Kept #1: X        X
```

```
Roll #2: 3  3  3  3  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 10
Roll #0: 3  5  4  4  3
Kept #0:
Roll #1: 3  6  1  4  5
Kept #1: X        X
Roll #2: 3  1  2  4  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 11
Roll #0: 1  5  5  6  5
Kept #0:          X
Roll #1: 1  3  6  6  3
Kept #1:       X  X
Roll #2: 4  5  6  6  5
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 12
Roll #0: 1  2  1  5  5
Kept #0:
Roll #1: 3  3  1  5  2
Kept #1:
Roll #2: 4  2  2  3  5
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"
```

CASO DE TESTE 61

```
Ones.........:2
Twos.........:2
Threes.......:12
Fours........:12
Fives........:10
Sixes........:18
3 of a kind..:21
```

```
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:50
Chance.......:18
Bonus........:0
```

```
Total........:240
==========================

***** Round 0
Roll #0: 3  2  5  1  3
Kept #0: X           X
Roll #1: 3  3  2  3  3
```

```
Kept #1: X  X     X  X
Roll #2: 3  3  1  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 1
Roll #0: 6  6  4  2  3
```

```
Kept #0: X  X
Roll #1: 6  6  6  5  3
Kept #1: X  X  X
Roll #2: 6  6  6  4  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 2
Roll #0: 3  5  1  1  2
Kept #0:       X  X
Roll #1: 1  6  1  1  5
Kept #1: X     X  X
Roll #2: 1  1  1  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 3
Roll #0: 6  3  1  4  2
Kept #0:    X     X  X
Roll #1: 4  3  2  4  2
Kept #1: X     X  X  X
Roll #2: 4  6  2  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 4
Roll #0: 6  2  6  4  1
Kept #0: X     X
Roll #1: 6  6  6  6  5
Kept #1: X  X  X  X
Roll #2: 6  6  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 5
Roll #0: 4  2  6  5  3
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 6
Roll #0: 3  4  1  6  3
Kept #0: X           X
Roll #1: 3  4  6  3  3
Kept #1: X        X  X
Roll #2: 3  6  6  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 7
Roll #0: 5  3  5  6  2
Kept #0: X     X
Roll #1: 5  3  5  6  3
Kept #1: X     X
Roll #2: 5  3  5  2  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 8
Roll #0: 4  2  1  6  4
Kept #0: X           X
Roll #1: 4  2  3  2  4
Kept #1: X           X
Roll #2: 4  1  2  3  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 9
Roll #0: 4  2  3  3  5
Kept #0:       X  X
Roll #1: 5  1  3  3  3
Kept #1:       X  X  X
Roll #2: 5  5  3  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 10
Roll #0: 6  4  3  4  3
Kept #0:    X     X
Roll #1: 1  4  3  4  4
Kept #1:    X     X  X
Roll #2: 5  4  2  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 11
Roll #0: 4  6  1  3  6
Kept #0:       X
Roll #1: 1  4  1  4  6
Kept #1: X     X
Roll #2: 1  3  1  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 12
Roll #0: 6  3  1  4  2
Kept #0:             X
Roll #1: 3  5  1  1  2
Kept #1:             X
Roll #2: 4  4  1  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"
```

APÊNDICE C – Bozó - Conjunto de Casos de Teste T_{estr}

Neste apêndice constam, dos 150 casos de teste gerados para o Teste Estrutural, somente aqueles que obtiveram êxito na cobertura dos critérios estruturais (Todos-Nós, Todas-Arestas, Todos-Usos e Todos-Potenciais-Usos) referentes ao jogo Bozó. Sendo assim, tem-se que $T_{estr} = \{01, 02, 03, 04, 05, 06, 07, 08, 11, 13, 16, 21, 29, 31, 37, 38, 66, 68, 87\}$.

CASO DE TESTE 01

```
Ones.........:2
Twos.........:4
Threes........:6
Fours.........:8
Fives.........:15
Sixes.........:24
3 of a kind..:23
4 of a kind..:6
Full hand....:25
Small seq....:30
Large seq....:0
5 of a kind..:50
Chance........:20
Bonus.........:0
Total.........:213
============================

***** Round 0
Roll #0: 6  1  2  4  3
Kept #0:       X  X  X
Roll #1: 1  4  2  4  3
Kept #1:    X  X     X
Roll #2: 2  4  2  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 1
Roll #0: 1  6  2  4  6
Kept #0:    X        X
Roll #1: 3  6  5  3  6
Kept #1: X  X     X  X
Roll #2: 3  6  3  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 6  3  6  6  3
Kept #0: X     X  X
Roll #1: 6  2  6  6  6
Kept #1: X     X  X  X
Roll #2: 6  2  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes.........:"

***** Round 3
Roll #0: 1  2  3  4  6
Kept #0:             X
Roll #1: 6  2  4  5  6
Kept #1: X           X
Roll #2: 6  1  5  2  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 4
Roll #0: 3  6  4  6  3
Kept #0:    X     X
Roll #1: 6  6  4  6  6
Kept #1: X  X     X  X
Roll #2: 6  6  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 5
Roll #0: 1  4  6  4  6
Kept #0:       X     X
Roll #1: 3  6  6  5  6
Kept #1:    X  X     X
Roll #2: 2  6  6  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 6
Roll #0: 1  3  5  1  6
Kept #0: X        X
Roll #1: 1  2  3  1  4
Kept #1: X        X
Roll #2: 1  5  5  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 7
Roll #0: 2  4  1  3  1
Kept #0:       X     X
Roll #1: 1  5  1  3  1
Kept #1: X     X     X
Roll #2: 1  1  1  2  1
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 2  4  6  3  1
Kept #0:    X
Roll #1: 5  4  6  5  5
Kept #1: X        X  X
Roll #2: 5  2  6  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 9
Roll #0: 3  4  2  3  2
Kept #0: X        X
Roll #1: 3  5  5  3  4
Kept #1: X        X
Roll #2: 3  4  6  3  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 10
Roll #0: 6  3  1  4  1
Kept #0:          X
Roll #1: 1  1  1  4  5
Kept #1:          X
Roll #2: 5  5  5  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 11
Roll #0: 4  4  2  1  6
Kept #0:       X
Roll #1: 6  5  2  2  4
Kept #1:       X  X
Roll #2: 3  6  2  2  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 12
Roll #0: 4  3  5  3  1
Kept #0:
Roll #1: 1  6  5  2  4
Kept #1:    X
Roll #2: 4  6  2  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"
```

CASO DE TESTE 02

```
Ones.........:1
Twos.........:8
Threes.......:6
Fours........:12
Fives........:20
Sixes........:18
3 of a kind..:23
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:0
5 of a kind..:50
Chance.......:18
Bonus........:35
Total........:246
============================

***** Round 0
Roll #0: 4  2  6  4  4
Kept #0: X        X  X
Roll #1: 4  2  1  4  4
Kept #1: X        X  X
Roll #2: 4  2  5  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 1
Roll #0: 2  2  4  6  5
Kept #0: X  X
Roll #1: 2  2  6  6  6
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 3  4  6  4  1
Kept #0:    X     X
Roll #1: 2  4  1  4  3
Kept #1: X  X        X
Roll #2: 2  4  6  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 3
Roll #0: 1  2  4  3  4
Kept #0:    X  X  X
Roll #1: 3  2  4  3  4
Kept #1:       X     X
Roll #2: 3  1  4  3  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 4
Roll #0: 3  6  1  2  3
Kept #0: X           X
Roll #1: 3  5  4  2  3
Kept #1: X     X  X
Roll #2: 3  3  4  2  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 5
Roll #0: 2  5  6  2  3
Kept #0: X        X
Roll #1: 2  3  3  2  2
Kept #1: X        X  X
Roll #2: 2  2  3  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 6
Roll #0: 4  3  5  6  6
Kept #0:          X  X
Roll #1: 3  4  2  6  6
Kept #1:          X  X
Roll #2: 2  5  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 7
Roll #0: 3  5  5  4  4
Kept #0:    X  X
Roll #1: 4  5  5  2  2
Kept #1:    X  X
Roll #2: 5  5  5  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 8
Roll #0: 5  3  2  5  2
Kept #0: X        X
Roll #1: 5  5  1  5  2
Kept #1: X  X     X
Roll #2: 5  5  2  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 9
Roll #0: 3  6  6  1  4
Kept #0:    X  X
Roll #1: 6  6  6  3  1
Kept #1: X  X  X
Roll #2: 6  6  6  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 10
Roll #0: 6  3  3  3  5
Kept #0:    X  X  X
Roll #1: 3  3  3  3  5
Kept #1: X  X  X  X
Roll #2: 3  3  3  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 11
Roll #0: 3  5  2  4  5
Kept #0:    X        X
Roll #1: 1  5  3  6  5
Kept #1:    X        X
Roll #2: 5  5  1  2  5
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 12
Roll #0: 1  2  6  3  4
Kept #0:       X
Roll #1: 1  6  6  1  6
Kept #1:    X  X     X
Roll #2: 4  6  6  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"
```

CASO DE TESTE 03

```
Ones.........:3
Twos.........:6
Threes.......:9
Fours........:12
Fives........:15
Sixes........:12
3 of a kind..:20
4 of a kind..:19
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance.......:22
Bonus........:0
Total........:213
============================

***** Round 0
Roll #0: 3  4  3  4  3
Kept #0: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 1
Roll #0: 1  3  2  6  4
Kept #0:    X  X     X
Roll #1: 1  3  2  5  4
Kept #1: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 2
Roll #0: 5  6  5  3  1
Kept #0: X     X
Roll #1: 5  4  5  4  4
Kept #1:    X     X  X
Roll #2: 3  4  2  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 3
Roll #0: 4  5  4  4  5
Kept #0: X     X  X
Roll #1: 4  4  4  4  1
Kept #1: X  X  X  X
Roll #2: 4  4  4  4  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 4
Roll #0: 5  3  6  1  3
Kept #0:    X        X
Roll #1: 2  3  5  6  3
Kept #1:    X        X
Roll #2: 4  3  2  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 5
Roll #0: 2  2  3  6  4
Kept #0: X  X
Roll #1: 2  2  2  1  6
Kept #1: X  X  X
Roll #2: 2  2  2  3  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 6
Roll #0: 2  4  5  1  3
Kept #0:       X
Roll #1: 1  4  5  5  4
Kept #1:       X  X
Roll #2: 2  5  5  5  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 7
Roll #0: 6  4  6  2  5
Kept #0: X     X
Roll #1: 6  1  6  2  5
Kept #1: X     X
Roll #2: 6  5  6  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 8
Roll #0: 4  6  3  2  3
Kept #0:       X     X
Roll #1: 4  2  3  3  3
Kept #1:       X  X  X
Roll #2: 5  4  3  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 9
Roll #0: 4  3  2  4  4
Kept #0:
Roll #1: 3  3  5  6  3
Kept #1:          X
Roll #2: 4  5  3  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 10
Roll #0: 6  6  3  5  4
Kept #0:
Roll #1: 6  3  6  2  1
Kept #1:             X
Roll #2: 1  6  4  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 11
Roll #0: 3  3  6  3  3
Kept #0:       X
Roll #1: 6  3  6  6  3
Kept #1: X     X  X
Roll #2: 6  1  6  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 12
Roll #0: 5  1  4  3  2
Kept #0:
Roll #1: 4  6  3  3  6
Kept #1:    X        X
Roll #2: 3  6  4  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"
```

CASO DE TESTE 04

```
Ones.........:4
Twos.........:4
Threes.......:6
Fours........:8
Fives........:15
Sixes........:18
3 of a kind..:20
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:50
Chance.......:16
Bonus........:0
Total........:236
============================

***** Round 0
Roll #0: 3  2  6  4  5
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 1
Roll #0: 3  3  3  4  4
Kept #0: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 4  4  1  3  3
Kept #0: X  X
Roll #1: 4  4  2  3  6
Kept #1: X  X
Roll #2: 4  4  1  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 3
Roll #0: 1  2  1  4  5
Kept #0: X     X
Roll #1: 1  5  1  6  5
Kept #1:    X        X
Roll #2: 4  5  5  3  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 4
Roll #0: 1  4  4  4  4
Kept #0:    X  X  X  X
Roll #1: 4  4  4  4  4
Kept #1: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 5
Roll #0: 3  5  2  4  6
Kept #0: X     X  X
Roll #1: 3  1  2  4  2
Kept #1: X     X  X
Roll #2: 3  4  2  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 6
Roll #0: 1  5  1  5  1
Kept #0: X     X     X
Roll #1: 1  2  1  1  1
Kept #1: X     X  X  X
Roll #2: 1  2  1  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 7
Roll #0: 3  4  5  3  4
Kept #0:    X        X
Roll #1: 1  4  3  2  4
Kept #1:    X  X  X
Roll #2: 6  4  3  2  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 8
Roll #0: 4  3  3  5  2
Kept #0:    X  X
Roll #1: 5  3  3  4  6
Kept #1:    X  X
Roll #2: 2  3  3  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 9
Roll #0: 5  3  4  6  4
Kept #0:       X     X
Roll #1: 5  1  4  3  4
Kept #1:       X     X
Roll #2: 5  3  4  6  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 10
Roll #0: 5  6  4  1  3
Kept #0:    X
Roll #1: 5  6  6  1  4
Kept #1:    X  X
Roll #2: 2  6  6  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 11
Roll #0: 6  6  4  2  5
Kept #0: X  X
Roll #1: 6  6  3  1  2
Kept #1: X  X
Roll #2: 6  6  1  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 12
Roll #0: 5  1  6  6  2
Kept #0:       X  X
Roll #1: 4  5  6  6  4
Kept #1:       X  X
Roll #2: 1  2  6  6  2
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"
```

CASO DE TESTE 05

```
Ones.........:3
Twos.........:8
Threes.......:3
Fours........:12
Fives........:5
Sixes........:18
3 of a kind..:12
4 of a kind..:7
Full hand....:25
Small seq....:0
Large seq....:40
5 of a kind..:0
Chance.......:19
Bonus........:0
Total........:152
============================

***** Round 0
Roll #0: 2  6  5  5  1
Kept #0:       X  X
Roll #1: 4  5  5  5  4
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 1
Roll #0: 6  1  5  4  1
Kept #0:    X        X
Roll #1: 5  1  2  6  1
Kept #1:    X        X
Roll #2: 6  1  1  5  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 2
Roll #0: 3  4  5  4  1
Kept #0:    X     X
Roll #1: 3  4  4  4  2
Kept #1:    X  X  X
Roll #2: 5  4  4  4  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 3
Roll #0: 2  1  1  1  2
Kept #0:    X  X  X
Roll #1: 6  1  1  1  3
Kept #1:    X  X  X
Roll #2: 6  1  1  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 4
Roll #0: 4  6  4  4  2
Kept #0: X     X  X
Roll #1: 4  2  4  4  6
Kept #1: X     X  X
Roll #2: 4  5  4  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 5
Roll #0: 1  5  3  2  4
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 6
Roll #0: 2  5  6  4  4
Kept #0:          X  X
Roll #1: 6  5  6  4  4
Kept #1: X     X
Roll #2: 6  3  6  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 7
Roll #0: 4  6  6  4  1
Kept #0:    X  X
Roll #1: 4  6  6  2  3
Kept #1:    X  X
Roll #2: 3  6  6  5  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 8
Roll #0: 2  5  4  6  2
Kept #0: X           X
Roll #1: 2  2  3  2  2
Kept #1: X  X     X  X
Roll #2: 2  2  4  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 9
Roll #0: 4  1  2  4  3
Kept #0: X     X     X
Roll #1: 4  3  2  4  3
Kept #1: X        X
Roll #2: 4  4  1  4  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 10
Roll #0: 1  1  4  2  1
Kept #0: X  X        X
Roll #1: 1  1  5  1  1
Kept #1: X  X     X  X
Roll #2: 1  1  3  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 11
Roll #0: 6  3  2  5  3
Kept #0: X
Roll #1: 6  4  4  2  2
Kept #1: X
Roll #2: 6  3  5  2  5
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 12
Roll #0: 1  6  5  3  5
Kept #0:    X
Roll #1: 6  6  5  1  3
Kept #1: X  X
Roll #2: 6  6  5  6  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"
```

CASO DE TESTE 06

```
Ones.........:1
Twos.........:6
Threes.......:9
Fours........:12
Fives........:10
Sixes........:24
3 of a kind..:21
4 of a kind..:18
Full hand....:25
Small seq....:30
Large seq....:0
5 of a kind..:0
Chance.......:16
Bonus........:0
Total........:172
============================

***** Round 0
Roll #0: 4  2  1  2  6
Kept #0:    X     X
Roll #1: 1  2  5  2  4
Kept #1:    X     X
Roll #2: 4  2  3  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 1
Roll #0: 3  1  6  1  5
Kept #0:    X     X
Roll #1: 3  1  3  1  4
Kept #1: X  X  X  X
Roll #2: 3  1  3  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 4  4  2  2  6
Kept #0: X  X
Roll #1: 4  4  2  1  6
Kept #1: X  X
Roll #2: 4  4  2  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 3
Roll #0: 3  5  3  3  6
Kept #0: X     X  X
Roll #1: 3  5  3  3  1
Kept #1: X     X  X
Roll #2: 3  5  3  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 4
Roll #0: 3  6  4  1  5
Kept #0: X     X
Roll #1: 3  5  4  3  1
Kept #1: X        X
Roll #2: 3  2  1  3  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 5
Roll #0: 5  3  3  4  1
Kept #0:    X  X
Roll #1: 6  3  3  1  4
Kept #1:    X  X
Roll #2: 2  3  3  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 6
Roll #0: 5  6  2  1  5
Kept #0: X           X
Roll #1: 5  3  2  2  5
Kept #1: X           X
Roll #2: 5  4  2  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 7
Roll #0: 5  2  3  2  3
Kept #0:       X     X
Roll #1: 1  4  3  3  3
Kept #1:       X  X  X
Roll #2: 3  6  3  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 5  5  6  2  6
Kept #0:       X     X
Roll #1: 6  2  6  3  6
Kept #1: X     X     X
Roll #2: 6  6  6  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 9
Roll #0: 2  3  4  2  1
Kept #0: X  X  X
Roll #1: 2  3  4  4  1
Kept #1: X  X  X
Roll #2: 2  3  4  1  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 10
Roll #0: 6  1  3  3  3
Kept #0:
Roll #1: 1  6  6  1  4
Kept #1:             X
Roll #2: 3  4  4  1  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 11
Roll #0: 4  3  5  3  2
Kept #0:
Roll #1: 2  1  2  2  3
Kept #1:
Roll #2: 6  4  1  3  5
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 12
Roll #0: 3  5  1  5  6
Kept #0:             X
Roll #1: 3  4  6  5  6
Kept #1:       X     X
Roll #2: 4  6  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"
```

CASO DE TESTE 07

```
Ones.........:1
Twos.........:8
Threes.......:12
Fours........:8
Fives........:10
Sixes........:18
3 of a kind..:13
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance.......:18
Bonus........:0
Total........:183
============================

***** Round 0
Roll #0: 6  6  6  3  3
Kept #0: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 1
Roll #0: 4  2  2  1  2
Kept #0:    X  X     X
Roll #1: 2  2  2  4  2
Kept #1: X  X  X     X
Roll #2: 2  2  2  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 2
Roll #0: 1  4  6  4  6
Kept #0:       X     X
Roll #1: 2  1  6  6  6
Kept #1:       X  X  X
Roll #2: 4  1  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 3
Roll #0: 5  2  2  1  6
Kept #0:    X  X
Roll #1: 5  2  2  3  6
Kept #1:    X  X
Roll #2: 4  2  2  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 4
Roll #0: 3  3  4  3  3
Kept #0: X  X     X  X
Roll #1: 3  3  1  3  3
Kept #1: X  X     X  X
Roll #2: 3  3  2  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 5
Roll #0: 2  6  4  3  6
Kept #0:    X        X
Roll #1: 4  6  2  2  6
Kept #1:    X        X
Roll #2: 1  6  3  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 6
Roll #0: 1  4  6  6  3
Kept #0:       X  X
Roll #1: 3  3  6  6  2
Kept #1:       X  X
Roll #2: 1  5  6  6  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 7
Roll #0: 1  5  3  2  5
Kept #0:    X        X
Roll #1: 6  5  6  1  5
Kept #1: X     X
Roll #2: 6  3  6  3  5
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 1  3  2  1  1
Kept #0:
Roll #1: 2  1  1  4  3
Kept #1: X  X     X  X
Roll #2: 2  1  5  4  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 9
Roll #0: 1  3  3  3  4
Kept #0:             X
Roll #1: 5  1  3  5  4
Kept #1:             X
Roll #2: 6  4  2  5  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 10
Roll #0: 4  4  3  3  1
Kept #0:
Roll #1: 1  2  1  1  6
Kept #1:             X
Roll #2: 2  1  2  2  6
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"
```

```
***** Round 11
Roll #0: 4  5  3  4  2
Kept #0: X     X     X
Roll #1: 4  1  3  6  2
Kept #1: X     X     X
Roll #2: 4  6  3  5  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 12
Roll #0: 4  4  2  5  2
Kept #0:
Roll #1: 4  3  3  1  4
Kept #1:
Roll #2: 4  2  2  4  6
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"
```

CASO DE TESTE 08

```
Ones.........:3
Twos.........:8
Threes........:9
Fours.........:4
Fives.........:5
Sixes.........:12
3 of a kind..:12
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:0
5 of a kind..:0
Chance........:18
Bonus.........:0
Total.........:126
============================

***** Round 0
Roll #0: 4  6  5  2  2
Kept #0:          X  X
Roll #1: 4  5  4  2  2
Kept #1: X     X  X  X
Roll #2: 4  6  4  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 1  1  1  6  6
Kept #0: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 6  1  5  3  3
Kept #0:          X  X
Roll #1: 4  5  3  3  3
Kept #1:       X  X  X
Roll #2: 6  1  3  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 3
Roll #0: 1  1  1  6  3
Kept #0: X  X  X
Roll #1: 1  1  1  4  6
Kept #1: X  X  X
Roll #2: 1  1  1  2  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 4
Roll #0: 3  4  1  3  3
Kept #0: X        X  X
Roll #1: 3  1  2  3  3
Kept #1: X        X  X
Roll #2: 3  2  1  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 5
Roll #0: 3  3  3  4  2
Kept #0: X  X  X
Roll #1: 3  3  3  1  6
Kept #1: X  X  X
Roll #2: 3  3  3  1  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 6
Roll #0: 6  3  4  5  5
Kept #0:    X  X
Roll #1: 1  3  4  1  5
Kept #1: X        X
Roll #2: 1  3  4  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 7
Roll #0: 1  2  4  1  6
Kept #0: X        X
Roll #1: 1  3  1  1  5
Kept #1: X     X  X
Roll #2: 1  3  1  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 3  4  6  3  5
Kept #0: X  X
Roll #1: 3  4  1  4  5
Kept #1:
Roll #2: 1  4  3  5  3
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 9
Roll #0: 1  3  6  5  2
Kept #0:       X
Roll #1: 5  2  6  1  3
Kept #1:       X
Roll #2: 4  6  6  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 10
Roll #0: 2  2  3  2  2
Kept #0: X  X     X  X
Roll #1: 2  2  3  2  2
Kept #1: X  X     X  X
Roll #2: 2  2  6  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 11
Roll #0: 5  4  4  3  5
Kept #0:
Roll #1: 2  3  2  3  6
Kept #1:             X
Roll #2: 6  2  1  4  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 12
Roll #0: 4  2  2  5  1
Kept #0:
Roll #1: 6  2  5  3  4
Kept #1:    X     X  X
Roll #2: 5  2  6  3  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"
```

CASO DE TESTE 11

```
Ones.........:3
Twos.........:8
Threes........:6
Fours.........:8
Fives.........:10
Sixes.........:24
3 of a kind..:0
4 of a kind..:24
Full hand....:25
Small seq....:0
Large seq....:0
5 of a kind..:0
Chance........:18
Bonus.........:0
Total.........:126
============================

***** Round 0
Roll #0: 2  5  2  5  6
Kept #0: X  X  X  X
Roll #1: 2  5  2  5  4
Kept #1: X  X  X  X
Roll #2: 2  5  2  5  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 5  1  3  5  4
Kept #0: X        X
Roll #1: 5  6  1  5  2
Kept #1: X        X
Roll #2: 5  4  2  5  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 2
Roll #0: 5  5  2  6  5
Kept #0: X  X        X
Roll #1: 5  5  6  1  5
Kept #1: X  X        X
Roll #2: 5  5  5  4  5
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 3
Roll #0: 2  2  1  2  5
Kept #0: X  X     X
Roll #1: 2  2  5  2  1
Kept #1: X  X     X
Roll #2: 2  2  5  2  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 4
Roll #0: 1  6  6  4  5
Kept #0:    X  X
Roll #1: 6  6  6  6  3
Kept #1: X  X  X  X
Roll #2: 6  6  6  6  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 5
Roll #0: 2  2  4  1  5
Kept #0: X  X
Roll #1: 2  2  1  2  6
Kept #1: X  X     X
Roll #2: 2  2  2  2  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 6
Roll #0: 1  4  1  2  3
Kept #0: X  X     X  X
Roll #1: 1  4  4  2  3
Kept #1:    X     X  X
Roll #2: 3  4  2  2  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"
```

```
***** Round 7
Roll #0: 3  6  6  4  2
Kept #0:          X
Roll #1: 6  1  3  4  4
Kept #1:          X  X
Roll #2: 6  2  2  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 8
Roll #0: 1  5  1  6  4
Kept #0: X     X
Roll #1: 1  3  1  5  1
Kept #1: X     X     X
Roll #2: 1  3  1  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 9
Roll #0: 6  5  5  2  4
Kept #0: X
Roll #1: 6  2  5  2  5
Kept #1: X
Roll #2: 6  6  2  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 10
Roll #0: 5  3  6  1  4
Kept #0:    X        X
Roll #1: 3  3  6  4  4
Kept #1:       X
Roll #2: 2  3  6  6  2
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 11
Roll #0: 2  3  2  2  2
Kept #0:
Roll #1: 2  2  6  3  1
Kept #1:       X
Roll #2: 1  5  6  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 12
Roll #0: 5  3  2  4  4
Kept #0:    X  X  X
Roll #1: 5  3  2  4  1
Kept #1:    X  X  X
Roll #2: 6  3  2  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"
```

CASO DE TESTE 13

```
Ones.........:2
Twos.........:8
Threes.......:12
Fours........:12
Fives........:10
Sixes........:18
3 of a kind..:0
4 of a kind..:0
Full hand....:25
Small seq....:0
Large seq....:40
5 of a kind..:0
Chance.......:20
Bonus........:0
Total........:147
============================

***** Round 0
Roll #0: 4  6  6  3  5
Kept #0: X        X
Roll #1: 4  6  5  3  5
Kept #1: X        X
Roll #2: 4  4  3  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 3  3  6  2  5
Kept #0: X  X
Roll #1: 3  3  3  2  1
Kept #1: X  X  X
Roll #2: 3  3  3  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 2
Roll #0: 3  5  4  2  1
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 3
Roll #0: 6  2  3  2  4
Kept #0:    X     X
Roll #1: 1  2  1  2  2
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 4
Roll #0: 5  6  2  2  2
Kept #0:       X  X  X
Roll #1: 3  2  2  2  2
Kept #1:    X  X  X  X
Roll #2: 3  2  2  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 5
Roll #0: 3  6  6  6  5
Kept #0:    X  X  X
Roll #1: 4  6  6  6  1
Kept #1:    X  X  X
Roll #2: 1  6  6  6  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 6
Roll #0: 3  2  3  4  2
Kept #0: X     X
Roll #1: 3  4  3  5  4
Kept #1:    X        X
Roll #2: 2  4  3  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 7
Roll #0: 5  6  3  6  3
Kept #0:    X     X
Roll #1: 2  6  4  6  3
Kept #1:    X     X
Roll #2: 2  6  4  6  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 3  2  4  2  5
Kept #0: X  X  X
Roll #1: 3  2  4  2  6
Kept #1:
Roll #2: 4  1  2  2  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 9
Roll #0: 5  3  6  2  2
Kept #0: X
Roll #1: 5  5  6  1  1
Kept #1: X  X
Roll #2: 5  5  3  6  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 10
Roll #0: 4  6  2  1  4
Kept #0:    X
Roll #1: 2  6  2  5  1
Kept #1:    X
Roll #2: 5  6  5  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 11
Roll #0: 1  1  1  4  3
Kept #0:
Roll #1: 1  1  4  2  1
Kept #1:
Roll #2: 3  3  6  1  5
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 12
Roll #0: 4  4  5  3  6
Kept #0: X        X
Roll #1: 4  3  3  3  1
Kept #1:
Roll #2: 5  1  2  6  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"
```

CASO DE TESTE 16

```
Ones.........:4
Twos.........:8
Threes.......:6
Fours........:12
Fives........:20
Sixes........:18
3 of a kind..:20
4 of a kind..:28
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:50
Chance.......:13
Bonus........:35
Total........:309
============================

***** Round 0
Roll #0: 2  6  1  2  2
Kept #0: X        X  X
Roll #1: 2  5  2  2  2
Kept #1: X     X  X  X
Roll #2: 2  6  2  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 1
Roll #0: 3  5  3  4  4
Kept #0: X     X  X  X
Roll #1: 3  4  3  4  4
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 1  4  6  3  2
Kept #0:    X     X  X
Roll #1: 1  4  4  3  2
Kept #1:    X     X  X
Roll #2: 3  4  5  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 3
Roll #0: 5  2  5  5  2
Kept #0: X     X  X
Roll #1: 5  1  5  5  5
Kept #1: X     X  X  X
Roll #2: 5  1  5  5  5
```

```
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 4
Roll #0: 5  3  3  2  5
Kept #0: X           X
Roll #1: 5  4  5  4  5
Kept #1: X     X     X
Roll #2: 5  4  5  1  5
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 5
Roll #0: 1  2  4  4  4
Kept #0:       X  X  X
Roll #1: 6  1  4  4  4
Kept #1:       X  X  X
Roll #2: 2  2  4  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 6
Roll #0: 2  2  1  3  4
Kept #0: X  X
Roll #1: 2  2  2  2  1
Kept #1: X  X  X  X
Roll #2: 2  2  2  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 7
Roll #0: 6  6  6  5  4
Kept #0: X  X  X
Roll #1: 6  6  6  4  5
Kept #1: X  X  X
Roll #2: 6  6  6  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 8
Roll #0: 3  1  1  1  6
Kept #0:    X  X  X
Roll #1: 1  1  1  1  4
Kept #1: X  X  X  X
Roll #2: 1  1  1  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 9
Roll #0: 1  2  4  3  1
Kept #0: X           X
Roll #1: 1  1  5  4  1
Kept #1: X  X        X
Roll #2: 1  1  5  5  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 10
Roll #0: 5  3  1  2  4
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 11
Roll #0: 4  3  6  6  4
Kept #0:       X  X
Roll #1: 5  6  6  6  3
Kept #1:    X  X  X
Roll #2: 4  6  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 12
Roll #0: 4  1  2  1  5
Kept #0:
Roll #1: 4  2  6  4  2
Kept #1:
Roll #2: 2  5  3  1  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"
```

CASO DE TESTE 21

```
Ones.........:2
Twos.........:8
Threes.......:12
Fours........:12
Fives........:15
Sixes........:24
3 of a kind..:0
4 of a kind..:29
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance.......:13
Bonus........:35
Total........:245
============================

***** Round 0
Roll #0: 5  4  1  2  6
Kept #0:            X
Roll #1: 1  1  4  3  6
Kept #1: X  X
Roll #2: 1  1  4  5  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 2  4  3  1  5
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 2
Roll #0: 3  3  4  3  5
Kept #0: X  X     X
Roll #1: 3  3  1  3  4
Kept #1: X  X     X
Roll #2: 3  3  6  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 3
Roll #0: 6  4  2  6  3
Kept #0: X        X
Roll #1: 6  6  4  6  2
Kept #1: X  X     X
Roll #2: 6  6  6  6  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 4
Roll #0: 1  4  6  2  4
Kept #0:    X        X
Roll #1: 6  4  3  3  4
Kept #1:    X  X  X  X
Roll #2: 4  4  3  3  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 5
Roll #0: 2  5  4  6  4
Kept #0:       X     X
Roll #1: 6  2  4  2  4
Kept #1:       X     X
Roll #2: 2  3  4  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 6
Roll #0: 1  4  2  1  3
Kept #0: X  X  X     X
Roll #1: 1  4  2  2  3
Kept #1:    X  X     X
Roll #2: 2  4  2  5  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 7
Roll #0: 4  2  5  2  2
Kept #0:    X     X  X
Roll #1: 3  2  4  2  2
Kept #1:    X     X  X
Roll #2: 4  2  2  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 8
Roll #0: 2  5  6  5  6
Kept #0:       X     X
Roll #1: 6  1  6  2  6
Kept #1: X     X     X
Roll #2: 6  6  6  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 9
Roll #0: 4  1  4  4  1
Kept #0:    X        X
Roll #1: 3  1  6  5  1
Kept #1:    X        X
Roll #2: 2  1  2  3  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 10
Roll #0: 5  1  3  6  6
Kept #0: X
Roll #1: 5  5  6  2  2
Kept #1: X  X
Roll #2: 5  5  1  5  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 11
Roll #0: 5  1  1  1  3
Kept #0:
Roll #1: 5  5  5  6  2
Kept #1:          X
Roll #2: 2  3  2  6  3
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 12
Roll #0: 2  3  4  2  1
Kept #0:
Roll #1: 2  5  3  2  4
Kept #1:
Roll #2: 1  4  3  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"
```

CASO DE TESTE 29

```
Ones.........:3
Twos.........:6
Threes.......:12
Fours........:12
Fives........:15
Sixes........:18
3 of a kind..:25
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance.......:19
Bonus........:35
Total........:240
============================

***** Round 0
Roll #0: 3  2  2  5  6
Kept #0:    X  X
```

```
Roll #1: 3  2  2  5  6
Kept #1:    X  X
Roll #2: 4  2  2  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 1
Roll #0: 6  5  6  5  2
Kept #0: X     X
Roll #1: 6  6  6  4  4
Kept #1: X  X  X
Roll #2: 6  6  6  2  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 2
Roll #0: 4  1  5  5  2
Kept #0:       X  X
Roll #1: 5  6  5  5  1
Kept #1: X     X  X
Roll #2: 5  6  5  5  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 3
Roll #0: 4  4  2  4  3
Kept #0: X  X     X
Roll #1: 4  4  6  4  5
Kept #1: X  X     X
Roll #2: 4  4  1  4  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 4
Roll #0: 4  5  1  4  3
Kept #0: X        X
Roll #1: 4  2  3  4  2
Kept #1: X        X
Roll #2: 4  3  5  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 5
Roll #0: 1  6  6  4  5
Kept #0:    X  X
Roll #1: 5  6  6  3  4
Kept #1:    X  X
Roll #2: 3  6  6  4  6
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 6
Roll #0: 4  2  4  5  1
Kept #0: X     X
Roll #1: 4  4  4  6  3
Kept #1: X  X  X
Roll #2: 4  4  4  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 7
Roll #0: 2  1  4  6  3
Kept #0:          X
Roll #1: 3  5  2  6  5
Kept #1:    X        X
Roll #2: 6  5  6  6  5
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 1  3  1  3  3
Kept #0:    X     X  X
Roll #1: 4  3  3  3  3
Kept #1:    X  X  X  X
Roll #2: 1  3  3  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 9
Roll #0: 4  4  3  2  5
Kept #0:          X
Roll #1: 3  3  1  2  4
Kept #1:          X
Roll #2: 3  2  2  2  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 10
Roll #0: 3  5  6  1  4
Kept #0:          X
Roll #1: 1  3  1  1  5
Kept #1: X     X  X
Roll #2: 1  2  1  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 11
Roll #0: 6  6  6  1  3
Kept #0: X  X  X
Roll #1: 6  6  6  5  3
Kept #1: X  X  X
Roll #2: 6  6  6  3  1
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 12
Roll #0: 1  4  5  1  2
Kept #0:
Roll #1: 3  6  4  2  5
Kept #1: ?  ?  ?  ?  ?
Placed on "Large seq....:"
```

CASO DE TESTE 31

```
Ones.........:4
Twos.........:6
Threes........:9
Fours........:8
Fives........:15
Sixes........:24
3 of a kind..:16
4 of a kind..:17
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:50
Chance.......:20
Bonus........:35
Total........:299
============================

***** Round 0
Roll #0: 4  3  2  6  2
Kept #0:       X     X
Roll #1: 5  2  2  4  2
Kept #1:    X  X     X
Roll #2: 1  2  2  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 1
Roll #0: 3  4  2  4  1
Kept #0: X  X  X
Roll #1: 3  4  2  1  5
Kept #1: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 2
Roll #0: 3  3  3  1  3
Kept #0: X  X  X     X
Roll #1: 3  3  3  3  3
Kept #1: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 3
Roll #0: 1  2  2  6  2
Kept #0:    X  X     X
Roll #1: 4  2  2  1  2
Kept #1:    X  X     X
Roll #2: 6  2  2  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 4
Roll #0: 3  6  1  6  4
Kept #0:    X     X
Roll #1: 3  6  2  6  6
Kept #1:    X     X  X
Roll #2: 6  6  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 5
Roll #0: 5  2  3  6  2
Kept #0:    X        X
Roll #1: 5  2  5  1  2
Kept #1: X     X
Roll #2: 5  5  5  1  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 6
Roll #0: 2  1  3  4  5
Kept #0: X     X  X
Roll #1: 2  2  3  4  5
Kept #1: X     X  X
Roll #2: 2  3  3  4  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 7
Roll #0: 6  4  6  2  5
Kept #0: X     X
Roll #1: 6  3  6  3  3
Kept #1:    X     X  X
Roll #2: 5  3  3  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 6  2  4  2  2
Kept #0: X
Roll #1: 6  6  6  4  6
Kept #1: X  X  X     X
Roll #2: 6  6  6  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 9
Roll #0: 4  2  2  6  5
Kept #0:          X
Roll #1: 1  5  6  6  5
Kept #1:       X  X
Roll #2: 6  1  6  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 10
Roll #0: 1  2  5  1  4
Kept #0: X        X
Roll #1: 1  4  1  1  1
Kept #1: X     X  X  X
Roll #2: 1  4  1  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 11
Roll #0: 5  6  4  4  3
Kept #0:       X     X
Roll #1: 5  6  4  6  3
Kept #1:       X     X
Roll #2: 1  1  4  2  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 12
Roll #0: 5  5  5  4  3
Kept #0:          X
Roll #1: 2  4  1  4  1
Kept #1:    X     X
Roll #2: 2  4  5  4  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"
```

CASO DE TESTE 37

```
Ones.........:3
Twos.........:4
Threes.......:9
Fours........:16
Fives........:15
Sixes........:12
3 of a kind..:10
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance.......:20
Bonus........:0
Total........:184
============================

***** Round 0
Roll #0: 4  5  4  1  5
Kept #0: X  X  X     X
Roll #1: 4  5  4  1  5
Kept #1: X  X  X     X
Roll #2: 4  5  4  2  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 3  2  4  5  1
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 2
Roll #0: 4  4  4  4  1
Kept #0: X  X  X  X
Roll #1: 4  4  4  4  2
Kept #1: X  X  X  X
Roll #2: 4  4  4  4  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 3
Roll #0: 2  3  2  4  4
Kept #0: X     X  X  X
Roll #1: 2  3  2  4  4
Kept #1: X     X  X  X
Roll #2: 2  5  2  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 4
Roll #0: 2  3  3  5  3
Kept #0:    X  X     X
Roll #1: 4  3  3  4  3
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 5
Roll #0: 5  3  2  4  5
Kept #0:    X  X  X
Roll #1: 6  3  2  4  4
Kept #1:          X  X
Roll #2: 2  3  5  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 6
Roll #0: 6  1  4  5  2
Kept #0: X
Roll #1: 6  2  5  2  1
Kept #1:    X     X
Roll #2: 1  2  3  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 7
Roll #0: 5  3  1  5  4
Kept #0: X        X
Roll #1: 5  5  1  5  1
Kept #1: X  X     X
Roll #2: 5  5  2  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 8
Roll #0: 2  1  1  6  3
Kept #0:    X  X
Roll #1: 5  1  1  5  6
Kept #1: X        X
Roll #2: 5  2  5  5  4
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 9
Roll #0: 3  6  3  6  1
Kept #0:    X     X
Roll #1: 4  6  2  6  3
Kept #1:    X     X
Roll #2: 3  6  1  6  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 10
Roll #0: 4  1  5  1  5
Kept #0:    X     X
Roll #1: 2  1  4  1  6
Kept #1:    X     X
Roll #2: 1  1  4  1  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 11
Roll #0: 3  5  5  4  3
Kept #0: X           X
Roll #1: 3  4  1  1  3
Kept #1: X           X
Roll #2: 3  3  1  6  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 12
Roll #0: 5  4  2  4  5
Kept #0:
Roll #1: 2  4  2  2  3
Kept #1:
Roll #2: 4  1  6  5  4
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"
```

CASO DE TESTE 38

```
Ones.........:2
Twos.........:6
Threes.......:9
Fours........:12
Fives........:15
Sixes........:18
3 of a kind..:28
4 of a kind..:14
Full hand....:25
Small seq....:0
Large seq....:40
5 of a kind..:0
Chance.......:13
Bonus........:0
Total........:182
============================

***** Round 0
Roll #0: 1  2  5  1  6
Kept #0: X        X
Roll #1: 1  5  1  1  5
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 1
Roll #0: 3  6  2  1  3
Kept #0: X           X
Roll #1: 3  6  3  6  3
Kept #1: X     X     X
Roll #2: 3  2  3  6  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 2
Roll #0: 3  1  1  3  6
Kept #0: X        X
Roll #1: 3  3  2  3  3
Kept #1: X  X     X  X
Roll #2: 3  3  2  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 3
Roll #0: 3  1  2  2  1
Kept #0:       X  X
Roll #1: 6  1  2  2  2
Kept #1:       X  X  X
Roll #2: 1  4  2  2  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 4
Roll #0: 6  3  1  3  6
Kept #0: X           X
Roll #1: 6  1  6  2  6
Kept #1: X     X     X
Roll #2: 6  2  6  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 5
Roll #0: 6  2  2  6  3
Kept #0: X        X
Roll #1: 6  6  6  6  3
Kept #1: X  X  X  X
Roll #2: 6  6  6  6  4
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 6
Roll #0: 1  4  1  4  3
Kept #0:    X     X
Roll #1: 5  4  1  4  2
Kept #1:    X     X
Roll #2: 1  4  4  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 7
Roll #0: 3  1  2  6  4
Kept #0: X     X     X
Roll #1: 3  1  2  4  4
Kept #1: X     X  X
Roll #2: 3  2  2  4  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 8
Roll #0: 1  3  2  5  4
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 9
Roll #0: 1  2  5  3  5
Kept #0:       X     X
Roll #1: 5  1  5  1  5
Kept #1: X     X     X
Roll #2: 5  1  5  6  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 10
Roll #0: 3  2  2  3  6
Kept #0:
Roll #1: 2  4  1  6  4
Kept #1:       X
Roll #2: 4  3  1  4  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 11
Roll #0: 1  5  4  2  6
Kept #0:             X
Roll #1: 1  1  5  2  6
Kept #1:             X
Roll #2: 5  1  1  2  6
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 12
Roll #0: 2  3  6  4  1
Kept #0: X  X     X
Roll #1: 2  3  5  4  1
Kept #1: X  X     X
Roll #2: 2  3  3  4  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"
```

CASO DE TESTE 66

```
Ones.........:2
Twos.........:4
Threes.......:9
Fours........:12
Fives........:15
Sixes........:18
3 of a kind..:15
4 of a kind..:18
Full hand....:25
Small seq....:30
Large seq....:40
5 of a kind..:0
Chance.......:20
Bonus........:0
Total........:208
============================

***** Round 0
Roll #0: 3  6  2  5  2
Kept #0:       X     X
Roll #1: 5  6  2  5  2
Kept #1: X     X  X  X
Roll #2: 5  6  2  5  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 2  6  4  6  3
Kept #0:    X     X
Roll #1: 1  6  6  6  5
Kept #1:    X  X  X
Roll #2: 4  6  6  6  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 2
Roll #0: 3  6  5  2  3
Kept #0: X           X
Roll #1: 3  5  1  4  3
Kept #1: X           X
Roll #2: 3  5  6  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 3
Roll #0: 3  4  6  1  6
Kept #0:       X     X
Roll #1: 5  3  6  4  6
Kept #1:    X     X
Roll #2: 6  3  4  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 4
Roll #0: 3  2  4  3  6
Kept #0: X        X
Roll #1: 3  3  6  3  4
Kept #1: X  X     X
Roll #2: 3  3  2  3  4
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 5
Roll #0: 3  4  2  6  3
Kept #0: X           X
Roll #1: 3  2  4  4  3
Kept #1: X     X  X  X
Roll #2: 3  4  4  4  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 6
Roll #0: 2  2  3  6  4
Kept #0: X  X
Roll #1: 2  2  5  1  3
Kept #1: X  X
Roll #2: 2  2  3  1  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 7
Roll #0: 3  6  3  3  6
Kept #0: X     X  X
Roll #1: 3  3  3  3  5
Kept #1: X  X  X  X
Roll #2: 3  3  3  3  6
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 8
Roll #0: 3  5  3  6  5
Kept #0:    X        X
Roll #1: 4  5  2  5  5
Kept #1:    X     X  X
Roll #2: 6  5  1  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 9
Roll #0: 6  6  3  3  5
Kept #0:
Roll #1: 1  4  2  3  2
Kept #1:    X  X  X
Roll #2: 4  4  2  3  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 10
Roll #0: 6  4  6  6  6
Kept #0:
Roll #1: 2  4  4  5  1
Kept #1:             X
Roll #2: 1  5  4  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 11
Roll #0: 1  5  3  4  5
Kept #0:
Roll #1: 5  6  5  5  5
Kept #1:    X
Roll #2: 4  6  5  2  4
Kept #2: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 12
Roll #0: 4  6  2  5  3
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"
```

CASO DE TESTE 68

```
Ones.........:4
Twos.........:4
Threes.......:6
Fours........:8
Fives........:10
Sixes........:18
3 of a kind..:27
4 of a kind..:27
Full hand....:25
Small seq....:0
Large seq....:40
5 of a kind..:50
Chance.......:21
Bonus........:0
Total........:240
============================

***** Round 0
Roll #0: 1  3  1  1  2
Kept #0: X     X  X
Roll #1: 1  1  1  1  3
Kept #1: X  X  X  X
Roll #2: 1  1  1  1  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 1
Roll #0: 6  2  2  1  6
Kept #0: X  X  X     X
Roll #1: 6  2  2  1  6
Kept #1: X  X  X     X
Roll #2: 6  2  2  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 2
Roll #0: 4  4  6  4  5
Kept #0: X  X     X
Roll #1: 4  4  1  4  1
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 3
Roll #0: 4  3  2  1  5
Kept #0: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 4
Roll #0: 5  2  3  6  3
Kept #0:       X     X
Roll #1: 5  6  3  5  3
Kept #1: X           X
Roll #2: 5  4  1  5  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 5
Roll #0: 3  4  3  5  5
Kept #0:          X  X
Roll #1: 2  3  6  5  5
Kept #1:          X  X
Roll #2: 2  6  6  5  5
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 6
Roll #0: 6  2  1  5  6
Kept #0: X           X
Roll #1: 6  2  5  5  6
Kept #1: X           X
Roll #2: 6  5  5  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 7
Roll #0: 1  5  6  3  6
Kept #0:       X     X
Roll #1: 3  2  6  6  6
Kept #1:       X  X  X
Roll #2: 2  4  6  6  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"

***** Round 8
Roll #0: 1  1  1  4  2
Kept #0: X  X  X
Roll #1: 1  1  1  1  1
Kept #1: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 9
Roll #0: 1  1  6  6  5
Kept #0:       X  X
Roll #1: 4  6  6  6  1
Kept #1:    X  X  X
Roll #2: 6  6  6  6  3
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 10
Roll #0: 6  2  3  5  5
Kept #0:       X
Roll #1: 3  4  3  1  1
Kept #1: X     X
Roll #2: 3  5  3  4  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 11
Roll #0: 1  5  4  1  5
Kept #0:
Roll #1: 3  2  4  6  6
Kept #1:    X
Roll #2: 4  2  6  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 12
Roll #0: 6  1  2  6  4
Kept #0: X        X
Roll #1: 6  2  6  6  3
Kept #1: X     X  X
Roll #2: 6  4  6  6  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"
```

CASO DE TESTE 87

```
Ones.........:2
Twos.........:6
Threes.......:12
Fours........:12
Fives........:10
Sixes........:6
3 of a kind..:18
4 of a kind..:0
Full hand....:25
Small seq....:30
Large seq....:0
5 of a kind..:50
Chance.......:10
Bonus........:0
Total........:181
============================

***** Round 0
Roll #0: 2  1  3  3  5
Kept #0:       X  X
Roll #1: 1  1  3  3  2
Kept #1: X  X  X  X
Roll #2: 1  1  3  3  2
Kept #2: ?  ?  ?  ?  ?
Placed on "Chance.......:"

***** Round 1
Roll #0: 4  3  1  1  4
Kept #0: X     X  X  X
Roll #1: 4  4  1  1  4
Kept #1: ?  ?  ?  ?  ?
Placed on "Full hand....:"

***** Round 2
Roll #0: 2  3  5  5  6
Kept #0:       X  X
Roll #1: 6  1  5  5  2
Kept #1:       X  X
Roll #2: 6  2  5  5  6
Kept #2: ?  ?  ?  ?  ?
Placed on "Fives........:"

***** Round 3
Roll #0: 2  2  3  5  3
Kept #0:       X     X
Roll #1: 6  1  3  6  3
Kept #1: X        X
Roll #2: 6  4  1  6  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Ones.........:"

***** Round 4
Roll #0: 3  2  6  2  3
Kept #0: X           X
Roll #1: 3  3  5  3  3
Kept #1: X  X     X  X
Roll #2: 3  3  2  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Threes.......:"

***** Round 5
Roll #0: 3  2  3  5  2
Kept #0: X     X
Roll #1: 3  5  3  1  1
Kept #1: X     X
Roll #2: 3  3  3  4  5
Kept #2: ?  ?  ?  ?  ?
Placed on "3 of a kind..:"

***** Round 6
Roll #0: 5  6  3  3  3
Kept #0:       X  X  X
Roll #1: 3  3  3  3  3
Kept #1: ?  ?  ?  ?  ?
Placed on "5 of a kind..:"

***** Round 7
Roll #0: 6  4  3  2  1
Kept #0:    X  X  X
Roll #1: 5  4  3  2  5
Kept #1:    X  X  X
Roll #2: 2  4  3  2  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Small seq....:"

***** Round 8
Roll #0: 4  3  4  6  2
Kept #0: X     X
Roll #1: 4  4  4  2  6
Kept #1: X  X  X
Roll #2: 4  4  4  3  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Fours........:"

***** Round 9
Roll #0: 4  1  2  1  6
Kept #0:    X     X
Roll #1: 6  1  2  1  5
Kept #1:    X     X
Roll #2: 5  1  3  1  1
Kept #2: ?  ?  ?  ?  ?
Placed on "4 of a kind..:"

***** Round 10
Roll #0: 3  2  4  2  6
Kept #0:    X     X
Roll #1: 2  2  6  2  3
Kept #1: X  X     X
Roll #2: 2  2  5  2  1
Kept #2: ?  ?  ?  ?  ?
Placed on "Twos.........:"

***** Round 11
Roll #0: 4  4  3  3  2
Kept #0:
Roll #1: 4  5  5  4  2
Kept #1:
Roll #2: 3  2  2  2  3
Kept #2: ?  ?  ?  ?  ?
Placed on "Large seq....:"

***** Round 12
Roll #0: 6  1  2  5  5
Kept #0: X
Roll #1: 6  4  1  4  1
Kept #1: X
Roll #2: 6  3  2  4  4
Kept #2: ?  ?  ?  ?  ?
Placed on "Sixes........:"
```

APÊNDICE D – Mobile Chat - Requisitos Funcionais

Este apêndice mostra os requisitos funcionais especificados para o aplicativo *Mobile Chat*, conforme segue:

1. Quando a linha de comando que invoca o cliente possuir menos do que quatro parâmetros, então, uma mensagem deverá ser exibida na saída padrão indicando o modo correto de executar a linha de comando, e o cliente não será iniciado.
2. Quando a linha de comando que invoca o cliente ter mais do que quatro parâmetros, então, uma mensagem deverá ser exibida na saída padrão indicando o modo correto de executar a linha de comando, e o cliente não será iniciado.
3. Ao executar o cliente e a porta do `ClientServer`, informada no segundo parâmetro da linha de comando, já estiver em uso, então, uma mensagem deverá ser exibida na saída padrão indicando que tal porta já está em uso, e o cliente não será iniciado.
4. Ao executar o cliente e o arquivo contendo as informações de custo, informado no último parâmetro da linha de comando, não for encontrado, então, uma mensagem deverá ser exibida na saída padrão indicando que o arquivo não foi encontrado, por fim, o cliente não será iniciado.
5. Quando o servidor for executado sem nenhum parâmetro, ele deverá ser iniciado na porta 1968.
6. Quando o servidor for executado sem nenhum parâmetro e a porta 1968 não estiver disponível, então, o servidor não deverá ser iniciado.
7. Quando o servidor for executado com a opção `-port`, seguido por um número que corresponda à porta em que se deseja executar o servidor, e tal porta não estiver em uso, então, o servidor deverá ser iniciado na porta especificada.
8. Quando o servidor for executado com a opção `-port`, seguido de um número que corresponda à porta em que se deseja executar o servidor, e tal porta já estiver em uso, então, o servidor não será iniciado.
9. Ao executar um cliente e o endereço do `ClientServer`, informado no segundo parâmetro da linha de comando, não corresponder ao endereço associado à máquina onde o cliente está sendo executado, então, uma mensagem de aviso deverá ser exibida informando que o endereço está incorreto. O cliente não será iniciado.

10. Ao executar um cliente e o endereço do `MainServer`, informado no terceiro parâmetro da linha de comando, não corresponder ao endereço associado à máquina onde o servidor principal está sendo executado, então, o cliente será iniciado, porém não conseguirá estabelecer conexão com o servidor e uma caixa de diálogo deverá aparecer perguntando sobre o desejo de uma nova conexão.

11. Ao executar um cliente e a porta do `MainServer`, informado no terceiro parâmetro da linha de comando, não corresponder à porta na qual o servidor principal está em execução, o cliente será iniciado, porém não conseguirá estabelecer conexão com o servidor e uma caixa de diálogo deverá ser exibida perguntando sobre o desejo de uma nova conexão.

12. Quando uma caixa de diálogo for exibida informando que não foi possível estabelecer conexão com o servidor e, se há o desejo de tentar conexão novamente, e o botão "*No*" for pressionado, então, o cliente deverá ser finalizado.

13. Quando uma caixa de diálogo for exibida informando que não foi possível estabelecer conexão com o servidor e, se há o desejo de tentar conexão novamente, e o botão "*Yes*" for pressionado, então, deverá ocorrer uma nova tentativa de estabelecimento da conexão.

14. Ao executar um cliente e a linha de comando possuir quatro parâmetros informados corretamente, então, o cliente deverá ser executado com sucesso e deverá ocorrer uma tentativa de conexão com o servidor.

15. Após a tentativa do cliente de se conectar ao servidor e a interface gráfica não apresentar a lista de apelidos e o campo de entrada de texto estiver desabilitado, isto indica que não foi possível se conectar com o servidor principal, então, uma caixa de diálogo deverá ser exibida perguntando se o usuário deseja tentar se conectar novamente.

16. Quando a conexão for estabelecida, então, o usuário deverá receber um número de identificação do servidor que será exibido na saída padrão do cliente. Em seguida, a lista de apelidos da interface gráfica deverá conter os apelidos dos usuários conectados. O campo de entrada de texto, presente na parte inferior da interface, deverá ser habilitado.

17. Quando o arquivo contendo as informações de custo não estiver escrito no formato adequado, o cliente será iniciado e a interpretação do arquivo deverá ocorrer sem sucesso. Uma mensagem deverá ser exibida para o usuário informando que o formato do arquivo não está correto, em seguida, o cliente deverá ser finalizado.

18. Quando nenhum texto for digitado na caixa de entrada de texto do aplicativo e a tecla *ENTER* for pressionada, então, não deverá acontecer nada.

19. Quando um texto contendo apenas caracteres em branco for digitado na caixa de entrada de texto e a tecla *ENTER* for pressionada, então, não deverá acontecer nada.

20. Quando o usuário digitar um texto iniciado por um caractere diferente de "/" e pressionar a tecla *ENTER*, então, o texto será considerado uma mensagem que deverá ser enviada para todos os usuários conectados e exibida na caixa de mensagens das interfaces dos clientes, inclusive do cliente que a enviou.

21. Quando o usuário digitar apenas uma barra na caixa de entrada e logo em seguida pressionar a tecla *ENTER*, então, uma mensagem informando que houve um erro de sintaxe deve ser exibida na caixa de mensagens.

22. Quando o usuário digitar um comando inválido, ou seja, um comando inexistente, então, uma mensagem informando que o comando digitado não foi encontrado deverá ser exibida para o usuário através da caixa de mensagens.

23. Ao invocar o comando "`help`", uma mensagem contendo um sumário dos comandos disponíveis, bem como uma breve descrição de cada um deverá ser exibida na caixa de mensagens.

24. Ao invocar o comando "`quit`", o cliente deverá ser desconectado do servidor e a aplicação será finalizada.

25. Ao invocar o comando "`printroutes`", a tabela de custo do servidor deverá ser exibida na saída padrão do cliente.

26. Ao invocar o comando "`stats`" e o cliente não estiver hospedando o servidor, então, deverão ser exibidas mensagens na caixa de texto do cliente informando a localização do servidor principal e do servidor ativo.

27. Ao invocar o comando "`stats`" e o cliente estiver hospedando o servidor, então, uma mensagem, notificando que o servidor está em execução localmente no cliente, deverá ser exibida.

28. Ao invocar o comando "`restore` e o cliente não estiver hospedando o servidor, então, uma mensagem de erro deverá ser exibida na caixa de texto do cliente informando que o servidor não está em execução localmente.

29. Ao invocar o comando "`restore`" e o cliente estiver hospedando o servidor, então, o servidor deverá ser enviado de volta para a posição inicial, ou seja, o servidor principal voltará a ficar ativo.

30. Ao invocar o comando "`getserver`" sem nenhum parâmetro, uma mensagem de erro deverá ser exibida na caixa de mensagens do usuário acusando erro de sintaxe.

31. Ao invocar o comando "`getserver`" com um parâmetro que consiste em um número inferior à 1024, então, uma mensagem de erro deverá ser exibida na caixa de mensagens do usuário acusando que o valor da porta deve ser acima de 1024.

32. Ao invocar o comando "`getserver`" com a porta desejada, que consiste em uma cadeia de caracteres alfanuméricos, então, uma mensagem de erro deverá ser exibida na caixa de mensagens do usuário acusando um valor inválido.

33. Ao invocar o comando "`getserver`" com mais de um parâmetro, uma mensagem deverá ser exibida na caixa de mensagens acusando erro de sintaxe.

34. Ao invocar o comando "`getserver`" tendo como parâmetro para porta um número superior à 1024 e o cliente já estiver hospedando o servidor, então, uma mensagem de erro deverá ser exibida na caixa de mensagens acusando que o servidor já está em execução naquela máquina.

35. Ao invocar o comando "`getserver`", tendo como parâmetro para porta um número igual ao número da porta onde está sendo executado o `ClientServer`, então, uma mensagem deverá ser exibida na caixa de mensagens do usuário acusando que a porta já está em uso.

36. Ao invocar o comando "`getserver`" tendo como valor para a porta um número superior à 1024 e o cliente ainda não estiver hospedando o servidor, então, o servidor deverá migrar para a máquina do cliente e aguardar por conexões na porta especificada.

37. Ao invocar o comando "`nick`" sem nenhum parâmetro, uma mensagem deverá ser exibida na caixa de mensagens do usuário acusando um erro de sintaxe.

38. Ao invocar o comando "`nick`" tendo como parâmetro um novo apelido, sendo uma cadeia de caracteres numéricos ou alfanuméricos livre de espaços em branco, idêntico ao apelido atual do usuário, então, uma mensagem deverá ser exibida na caixa de mensagens indicando que o novo apelido deve ser diferente do atual.

39. Ao invocar o comando "`nick`" tendo como parâmetro para o novo apelido uma cadeia de caracteres numéricos ou alfanuméricos contendo espaços em branco, então, uma mensagem deverá ser exibida na caixa de mensagens indicando que houve um erro de sintaxe.

40. Ao invocar o comando "`nick`" tendo como parâmetro para o novo apelido uma cadeia de caracteres numéricos ou alfanuméricos livre de espaços em branco e algum outro usuário conectado já estiver utilizando-o, então, uma mensagem deverá ser exibida na caixa de textos do usuário informando que o apelido já está em uso.

41. Ao invocar o comando "`nick`" tendo como parâmetro para o novo apelido uma cadeia de caracteres numéricos ou alfanuméricos livre de espaços em branco e nenhum outro usuário estiver utilizando-o, então, o apelido do usuário deverá ser trocado. Em seguida, todos os usuários participantes da sessão de *chat* deverão ser informados da troca através de uma notificação que deverá ser exibida na caixa de mensagens. A lista de apelidos presente na interface gráfica dos clientes conectados deverá ser atualizada (o apelido antigo será removido e o novo será adicionado).

42. Quando um novo usuário se conectar, todos os outros usuários conectados deverão receber uma notificação, exibida na caixa de mensagens, indicando que tal usuário acaba de entrar na sessão de *chat*. Em seguida, a lista de apelidos presente na interface gráfica dos clientes conectados deverá ser atualizada através da inserção do apelido do novo usuário.

43. Quando um usuário se desconectar, então, todos os outros usuários deverão receber uma notificação na caixa de mensagens indicando que tal usuário acaba de deixar a sessão de *chat*. Em seguida, a lista de apelidos presente na interface gráfica dos clientes conectados deverá ser atualizada através da remoção do apelido do usuário que deixou a sessão de *chat*.

44. Quando o servidor principal não estiver em execução em nenhum dos nós da rede no momento em que um cliente tenta se conectar, então, uma caixa de diálogo deverá ser exibida perguntando se o usuário deseja tentar se conectar novamente.

45. Quando algum dos agentes que carregam as mensagens de texto não conseguir atingir o destino, devido a algum problema na rede (como a desconexão inesperada do servidor ativo, por exemplo). Então, uma mensagem será perdida, mas o funcionamento do sistema como um todo (servidores e clientes) não será comprometido. Uma mensagem deverá ser exibida na caixa de mensagens do cliente que tentou enviá-la informando da impossibilidade do envio e a sessão de *chat* prossegue normalmente.

46. Quando algum dos agentes de controle (troca de apelido, entrada e saída de usuário, migração do servidor, requisição de entrada e saída, e os demais agentes que não têm a função de carregar mensagens de texto) forem perdidos ou não puderem ser enviados devido a algum problema na rede (como a desconexão inesperada do servidor ativo), o sistema deverá atingir um estado de inconsistência, podendo futuramente não apresentar um funcionamento correto, então, o cliente que tentou enviar o agente será finalizado.

47. Quando o servidor for executado com a opção `-port` seguida de uma cadeia de caracteres alfanuméricos, uma mensagem indicando um erro deverá ser exibida na tela e o servidor não será iniciado.

48. Quando o cliente que estiver hospedando o servidor ativo (`ChatServer`) tiver sua conexão interrompida, então, o servidor principal deverá ser ativado por intermédio de algum dos outros clientes conectados. Em seguida, o usuário deverá ser removido da lista de usuários conectados e todos os clientes serão notificados da localização do novo servidor e da partida daquele usuário. A sessão de *chat* prossegue.

APÊNDICE E – Mobile Chat - Conjunto de Casos de Teste T_{func}

Neste apêndice constam os casos de teste gerados com o intuito de cobrir os requisitos funcionais do *Mobile Chat*. Sendo assim, tem-se que $T_{func} = \{01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22\}$.

```
CASO DE TESTE 01
- Executa o servidor corretamente sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Realiza uma troca de mensagem corretamente entre os clientes.
- Invoca corretamente todos os comandos disponíveis (/help, /printroutes, /stats, /nick Jose,
  /getserver 2020, /restore, /quit).


CASO DE TESTE 02
- Executa o servidor com parâmetros informando uma porta válida (2005).
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Realiza uma troca de mensagem vazia, ou seja, só informa a tecla ENTER.
- Realiza uma troca de mensagem com espaços em branco, ou seja, informa espaço e depois a
  tecla ENTER.


CASO DE TESTE 03
- Executa o servidor com parâmetros informando uma porta inválida (abobrinha):
       java server.Server -port abobrinha


CASO DE TESTE 04
- Executa o servidor sem parâmetros.
- Executa um cliente com cinco parâmetros (os quatro primeiros corretos e o último um
  qualquer):
       java client.Client Ana 127.0.0.1:2000 127.0.0.1:1968 a.txt abobrinha
```

```
CASO DE TESTE 05
- Executa o servidor sem parâmetros.
- Executa um cliente com apenas três parâmetros, sendo todos corretos:
       java client.Client Ana 127.0.0.1:2000 127.0.0.1:1968


CASO DE TESTE 06
- Executa o servidor sem parâmetros.
- Executa um cliente em uma porta ocupada.


CASO DE TESTE 07
- Executa o servidor sem parâmetros.
- Executa um cliente informando um endereço IP para a máquina local que não exista.


CASO DE TESTE 08
- Executa o servidor sem parâmetros.
- Executa um cliente informando um arquivo de roteamento inexistente.


CASO DE TESTE 09
- Executa o servidor sem parâmetros.
- Executa um cliente informando um arquivo de roteamento existente, porém não está no formato adequado.


CASO DE TESTE 10
- Executa o servidor sem parâmetros.
- Executa um cliente informando o endereço IP do servidor errado.


CASO DE TESTE 11
- Executa o servidor sem parâmetros (porta 1968).
- Executa um cliente informando a porta do servidor errada (porta 2020).


CASO DE TESTE 12
- Executa o servidor sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Invoca o comando /nick <apelido> colocando espaço no apelido (/nick Jo ão).
- Invoca o comando /nick e espaço vazio.
- Invoca o comando /nick e o mesmo apelido atual.
- Invoca o comando /nick e o apelido de outro usuário conectado.


CASO DE TESTE 13
- Executa o servidor sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Invoca o comando /getserver <porta> e na porta coloca um  espaço vazio.
- Invoca o comando /getserver e informa um caracter para a porta (abobrinha, por exemplo).
```

```
CASO DE TESTE 14
- Executa o servidor sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Invoca o comando /getserver e informa a mesma porta do ClientServer.


CASO DE TESTE 15
- Executa o servidor sem parâmetros, mas a porta 1968 não está disponível.


CASO DE TESTE 16
- Executa o servidor com parâmetro, mas a porta especificada não está disponível.


CASO DE TESTE 17
- Executa o servidor sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Informa apenas uma barra na caixa de entrada do cliente e em seguida tecla ENTER}.
- Informa um comando inválido na caixa de entrada do cliente.


CASO DE TESTE 18
- Executa o servidor sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- O cliente que aloca o servidor invoca o comando /stats}.


CASO DE TESTE 19
- Executa o servidor sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- O cliente que não aloca o servidor invoca o comando /restore.


CASO DE TESTE 20
- Executa o servidor sem parâmetros.
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Invoca o comando /getserver informando uma porta inferior à 1024.
- Invoca o comando /getserver informando mais de um parâmetro.
- Invoca o comando /getserver corretamente, porém o cliente já está alocando o servidor.


CASO DE TESTE 21
- Executa um cliente com parâmetros corretos.


CASO DE TESTE 22
- Executa o servidor sem parâmetros.
```

```
- Executa um cliente com parâmetros corretos.
- Executa outro cliente com parâmetros corretos.
- Desconecta o servidor ativo.
- Desconecta o servidor principal.
- Envia uma mensagem.
- Invoca o comando /getserver corretamente.
```

APÊNDICE F – Mobile Chat - Conjunto de Casos de Teste T_{estr}

Neste apêndice são apresentados os casos de teste gerados com a finalidade de cobrir os critérios estruturais (Todos-Nós, Todas-Arestas e Todos-Usos) do aplicativo *Mobile Chat*. Vale mostrar que $T_{estr} = \{01, 02, 03, 04, 05, 06, 07, 08, 09, 10\}$.

```
CASO DE TESTE 01
Conectar dois clientes, cada um utilizando arquivos de custo contendo
entradas equivalentes, ou seja, apresentando o mesmo custo entre dois nós.


CASO DE TESTE 02
Conectar um cliente, que requisita o servidor. Conectar outro cliente,
que irá se conectar ao servidor que migrou para o primeiro cliente.


CASO DE TESTE 03
Conectar dois clientes, sendo que o apelido escolhido pelo segundo
cliente é o mesmo que o do primeiro cliente.


CASO DE TESTE 04
O usuário realiza uma troca de apelido.


CASO DE TESTE 05
Um usuário envia uma mensagem comum.


CASO DE TESTE 06
Um cliente tenta trocar de apelido indicando um apelido já em uso.


CASO DE TESTE 07
Um cliente que está hospedando o servidor executa o comando /restore.
```

```
CASO DE TESTE 08
O cliente executa o comando /printroutes.


CASO DE TESTE 09
O servidor ativo está hospedado em um dos clientes. Um novo cliente se
conecta e é feita a avaliacão sobre qual o melhor host para o servidor migrar.
É determinado que o host mais vantajoso é o servidor principal, então o
servidor migra de volta para o servidor principal.


CASO DE TESTE 10
Um novo cliente se conecta e a avaliacão para selecionar o nó mais
vantajoso elege esse cliente. Se o servidor estiver rodando em um outro
cliente, então, migra para o nó mais vantajoso.
```

APÊNDICE G – Mobile Chat - Cobertura dos Critérios Estruturais por T_{estr}

Este apêndice apresenta as tabelas, com a cobertura dos critérios estruturais (Todos-Nós, Todas-Arestas e Todos-Usos) pelo conjunto de casos de teste T_{estr}, de cada agente móvel do *Mobile Chat*. Vale esclarecer que os casos de teste que não apresentaram cobertura não foram listados.

Tabela 7.1: *Chat* - Cobertura da Classe: `agents.LeaveChatSession` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**21 de 21**	**100%**	**24 de 24**	**100%**	59 de 66	89%
07					**66 de 66**	**100%**

Tabela 7.2: *Chat* - Cobertura da Classe: `agents.ServerAgent` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	20 de 26	76%	20 de 28	71%	66 de 85	77%
02	**26 de 26**	**100%**	**28 de 28**	**100%**	**85 de 85**	**100%**

Tabela 7.3: *Chat* - Cobertura da Classe: `agents.SimpleMessageAgent` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**03 de 03**	**100%**	**01 de 01**	**100%**	**02 de 02**	**100%**

Tabela 7.4: *Chat* - Cobertura da Classe: `agents.ServerLeavingMessage` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	00 de 03	0%	00 de 01	0%	**00 de 00**	**100%**
02	**03 de 03**	**100%**	**01 de 01**	**100%**		

Tabela 7.5: *Chat* - Cobertura da Classe: `agents.NickChangeMessage` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	00 de 17	0%	00 de 19	0%	00 de 52	0%
04	15 de 17	88%	16 de 19	84%	41 de 52	78%
07	**17 de 17**	**100%**	**19 de 19**	**100%**	**52 de 52**	**100%**

Tabela 7.6: *Chat* - Cobertura da Classe: `agents.GrabUserListAgent` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**05 de 05**	**100%**	**04 de 04**	**100%**	**06 de 06**	**100%**

Tabela 7.7: *Chat* - Cobertura da Classe: `agents.PrintRoutingTable` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	00 de 11	0%	00 de 12	0%	00 de 32	0%
09	**11 de 11**	**100%**	**12 de 12**	**100%**	**32 de 32**	**100%**

Tabela 7.8: *Chat* - Cobertura da Classe: `agents.MobileMessage` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	07 de 12	58%	04 de 11	36%	06 de 24	25%
05	**12 de 12**	**100%**	**11 de 11**	**100%**	**24 de 24**	**100%**

Tabela 7.9: *Chat* - Cobertura da Classe: `agents.RestoreServerAgent` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	08 de 13	61%	08 de 13	61%	25 de 42	59%
03	-	-	09 de 13	69%	30 de 42	71%
05	-	-	-	-	31 de 42	73%
08	11 de 13	84%	12 de 13	92%	34 de 42	80%
10	**13 de 13**	**100%**	**13 de 13**	**100%**	41 de 42	97%
11					**42 de 42**	**100%**

(-) não houve alteração na cobertura

Tabela 7.10: *Chat* - Cobertura da Classe: `agents.SpawnMigrationAgent` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	10 de 15	66%	10 de 18	55%	22 de 49	44%
02	12 de 15	80%	13 de 18	72%	33 de 49	67%
07	13 de 15	86%	15 de 18	83%	39 de 49	79%
10	**15 de 15**	**100%**	**18 de 18**	**100%**	**49 de 49**	**100%**

Tabela 7.11: *Chat* - Cobertura da Classe: `agents.ServerMigrationMessage` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	06 de 07	85%	04 de 06	66%	08 de 11	72%
02	**07 de 07**	**100%**	**06 de 06**	**100%**	**11 de 11**	**100%**

Tabela 7.12: *Chat* - Cobertura da Classe: `agents.NickListMessage` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	21 de 24	87%	20 de 24	83%	43 de 49	87%
04	**24 de 24**	**100%**	**24 de 24**	**100%**	**49 de 49**	**100%**

Tabela 7.13: *Chat* - Cobertura da Classe: `agents.JoinChatSession` por T_{estr}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	37 de 42	88%	42 de 47	89%	110 de 141	78%
02	41 de 42	97%	46 de 47	97%	130 de 141	92%
03	**42 de 42**	**100%**	**47 de 47**	**100%**	135 de 141	95%
07					**135 de 135**	**100%**

APÊNDICE H – Mobile Chat - Conjunto de Casos de Teste T'_{func}

Este apêndice apresenta os 08 casos de teste adicionais, gerados com a finalidade de cobrir os critérios estruturais, dando continuidade a cobertura dos 22 casos de teste pertinentes ao conjunto T_{func} (Apêndice E) gerados anteriormente. Para tanto, tem-se que $T'_{func} = \{a, b, c, d, e, f, g, h\}$.

```
CASO DE TESTE (a)
Troca de mensagens entre dois usuários.


CASO DE TESTE (b)
Um usuário se conecta e requisita o servidor. Um segundo usuário se conecta.


CASO DE TESTE (c)
Um usuário se conecta utilizando o apelido de um usuário já conectado.
Quando requisitado quanto a troca de apelido, escolhe um novo apelido e aperta OK.


CASO DE TESTE (d)
O usuário A se conecta e requisita o servidor. O usuário B, então se conecta.
O nó mais vantajoso é eleito o servidor principal, então o servidor migra de
volta para a posicao inicial.

As tabelas de custo do usuário A e B são, respectivamente:

127.0.0.1:2001 9
127.0.0.1:2002 9
127.0.0.1:1968 1

127.0.0.1:2000 9
127.0.0.1:2002 9
127.0.0.1:1968 1

O nó mais vantajoso é o servidor principal (127.0.0.1:1968).
```

```
CASO DE TESTE (e)
O usuário A se conecta e requisita o servidor. O usuário B se conecta e requisita
o servidor, que migra de A para B.


CASO DE TESTE (f)
O usuário B se conecta e requisita o servidor. O usuário A se conecta e o servidor
migra de B para A, pois A é o nó mais vantajoso.
As tabelas de custo dos usuários A e B são, respectivamente:

127.0.0.1:2001 1
127.0.0.1:2002 2
127.0.0.1:1968 3

127.0.0.1:2000 3
127.0.0.1:2001 3
127.0.0.1:1968 3


CASO DE TESTE (g)
O usuário A se conecta, troca de nick, e sai.


CASO DE TESTE (h)
Três clientes (A, B e C) se conectam e se desconectam nessa mesma ordem.
```

APÊNDICE I – Mobile Chat - Cobertura dos Critérios Estruturais por T_{func} e T'_{func}

Aqui são apresentadas as tabelas, com a cobertura dos critérios estruturais (Todos-Nós, Todas-Arestas e Todos-Usos) pelo conjunto de casos de teste T_{func}, de cada agente móvel contido no pacote `agents`. Tais tabelas foram geradas com base nos relatórios de cobertura gerados pela Ferramenta JaBUTi. Vale esclarecer que os casos de teste que não apresentaram cobertura não foram listados.

Vale elucidar que os casos de teste identificados por número são os pertencentes ao conjunto T_{func} e os casos de teste identificados por letras são os casos de teste adicionais pertencentes ao conjunto T'_{func}.

Tabela 9.1: *Chat* - Cobertura da Classe: `agents.LeaveChatSession` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**21 de 21**	**100%**	23 de 23	95%	55 de 66	83%
12			**24 de 24**	**100%**	59 de 66	89%
h					**66 de 66**	**100%**

Tabela 9.2: *Chat* - Cobertura da Classe: `agents.SimpleMessageAgent` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**03 de 03**	**100%**	**01 de 01**	**100%**	**02 de 02**	**100%**

Tabela 9.3: *Chat* - Cobertura da Classe: `agents.ServerLeavingMessage` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	00 de 03	0%	00 de 01	0%	**00 de 00**	**100%**
e	**03 de 03**	**100%**	**01 de 01**	**100%**		

Tabela 9.4: *Chat* - Cobertura da Classe: `agents.ServerAgent` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	20 de 26	76%	20 de 28	71%	66 de 85	77%
b	21 de 26	80%	-	-	-	-
e	**26 de 26**	**100%**	**28 de 28**	**100%**	**85 de 85**	**100%**

(-) não houve alteração na cobertura

Tabela 9.5: *Chat* - Cobertura da Classe: `agents.NickChangeMessage` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	15 de 17	88%	16 de 19	84%	41 de 52	78%
12	**17 de 17**	**100%**	**19 de 19**	**100%**	**52 de 52**	**100%**

Tabela 9.6: *Chat* - Cobertura da Classe: `agents.GrabUserListAgent` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**05 de 05**	**100%**	**04 de 04**	**100%**	**06 de 06**	**100%**

Tabela 9.7: *Chat* - Cobertura da Classe: `agents.PrintRoutingTable` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**11 de 11**	**100%**	**12 de 12**	**100%**	**32 de 32**	**100%**

Tabela 9.8: *Chat* - Cobertura da Classe: `agents.MobileMessage` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	07 de 12	58%	04 de 11	36%	05 de 24	20%
a	**12 de 12**	**100%**	**11 de 11**	**100%**	23 de 24	95%
h					**24 de 24**	**100%**

Tabela 9.9: *Chat* - Cobertura da Classe: `agents.RestoreServerAgent` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	11 de 13	84%	12 de 13	92%	35 de 42	83%
14	-	-	-	-	36 de 42	85%
d	**13 de 13**	**100%**	**13 de 13**	**100%**	**42 de 42**	**100%**

(-) não houve alteração na cobertura

Tabela 9.10: *Chat* - Cobertura da Classe: `agents.SpawnMigrationAgent` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	10 de 15	66%	10 de 18	55%	22 de 49	44%
b	12 de 15	80%	13 de 18	72%	33 de 49	67%
d	14 de 15	93%	16 de 18	88%	43 de 49	87%
e	**15 de 15**	**100%**	**18 de 18**	**100%**	**49 de 49**	**100%**

Tabela 9.11: *Chat* - Cobertura da Classe: `agents.ServerMigrationMessage` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	06 de 07	85%	04 de 06	66%	08 de 11	72%
e	**07 de 07**	**100%**	**06 de 06**	**100%**	**11 de 11**	**100%**

Tabela 9.12: *Chat* - Cobertura da Classe: `agents.NickListMessage` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	**24 de 24**	**100%**	23 de 24	95%	**49 de 49**	**100%**
g			**24 de 24**	**100%**		

Tabela 9.13: *Chat* - Cobertura da Classe: `agents.JoinChatSession` por T_{func} e T'_{func}

Casos de Teste	Todos-Nós		Todas-Arestas		Todos-Usos	
01	37 de 42	88%	42 de 47	89%	110 de 141	78%
b	41 de 42	97%	46 de 47	97%	130 de 141	92%
c	**42 de 42**	**100%**	**47 de 47**	**100%**	135 de 141	95%
h					**141 de 141**	**100%**