FACULDADES DE CIÊNCIAS EXATAS E TECNOLOGIA
Bacharelado em Ciência da Computação
Trabalho de Conclusão de Curso

Desenvolvimento de Jogos para dispositivos móveis

Luiz Fernando Pires dos Santos Forgas
Rogério de Paula Aguilar

São Paulo
2003

LUIZ FERNANDO P. S. FORGAS
ROGÉRIO DE PAULA AGUILAR

Desenvolvimento de Jogos para dispositivos móveis

Trabalho de conclusão de curso apresentado as Faculdades Senac de Ciências Exatas e Tecnologia como exigência parcial para obtenção de grau de Bacharel em Ciências da Computação.

Orientadores:
Prof. Luciano Silva
Prof. Fábio Miranda

São Paulo
2003

Forgas, Luiz Fernando Pires dos Santos
Aguilar, Rogério de Paula
Desenvolvimento de Jogos para Dispositivos Móveis / Luiz Fernando P. S. Forgas e Rogério de Paula Aguilar. – São Paulo, 2003.
(383)f.

Trabalho de Conclusão de Curso – Faculdades Senac de Ciências Exatas e Tecnologia.

Orientadores: Profº Luciano Silva / Profº Fabio Miranda

Alunos: Luiz Fernando P. S. Forgas
Rogério de Paula Aguilar

Título: Desenvolvimento de Jogos para Dispositivos Móveis

A banca examinadora dos Trabalhos de Conclusão em sessão pública realizada em 09/12/2003, considerou os candidatos:

(X) Aprovados ( ) Reprovados

1) Examinador(a) Alice Shimada Bacic

2) Examinador(a) Fábio Miranda

3) Presidente Elias Roma Neto

Dedicamos este trabalho aos nossos pais, amigos e professores e em especial a Júlio de Paula, que infelizmente nos deixou este ano.

AGRADECIMENTO

A Deus, a nossos pais, amigos, professores e orientadores, que sempre nos apoiaram na conclusão deste trabalho.

“O raciocínio lógico leva você de A a B. A imaginação leva você a qualquer lugar”

Albert Einstein (1879 - 1955)

RESUMO

O desenvolvimento de jogos para dispositivos móveis vem crescendo durante os últimos anos. Além disso, o mercado de jogos no Brasil ainda é muito pequeno, o que abre uma boa perspectiva em investimentos e pesquisas nesta área. O desenvolvimento de jogos para dispositivos móveis apresenta alguns desafios, principalmente relacionados com a limitação de processamento e memória que estes dispositivos apresentam e problemas relacionados com o desenvolvimento de sistemas distribuídos, no caso de jogos on-line. Neste trabalho mostramos algumas limitações no desenvolvimento de jogos para dispositivos móveis e técnicas para contornar estas limitações. Também mostramos algumas dificuldades encontradas no desenvolvimento de jogos on-line e os mecanismos que foram utilizados para contornar tais limitações.

Palavras – Chave: jogos, programação de jogos, dispositivos móveis.

ABSTRACT

The development of games for mobile devices has grown during the last years. Moreover, being the market of games in Brazil still very small allows for a good perspective in terms of investments and research in this area. The development of games for mobile devices presents some challenges mainly related to their inherent limitation of processing power and memory, and problems related to the development of distributed systems as in the case of on-line games. In this work it is intended to show some limitations in the development of games for mobile devices and techniques employed to skirt them. Furthermore, some difficulties found in the development of on-line games together with the mechanisms that were used to skirt such limitations are presented.

LISTA DE FIGURAS

SUMÁRIO

Introdução

Os jogos são uma forma de entretenimento bastante antiga na humanidade. O jogo mais antigo de que se tem notícia chama-se Mancala, e trata-se de um jogo de 5000 anos, encontrado desenhado numa pedra no deserto do Sahara [6]. Outro exemplo é o jogo Go, popular no oriente e que possui 2000 anos de idade.
Com o surgimento dos computadores, os *games* entraram em um novo patamar.
Jogos simples, mas ao mesmo tempo interessantes , como por exemplo o *Pong* e o *Zork* [21], surgiram e entraram nos lares de milhões de pessoas.
Com o surgimento dos aparelhos de *videogame*, a indústria de jogos cresceu de forma espantosa, e hoje é uma das indústrias que mais crescem no mundo.
Nos últimos anos ocorreu um crescimento muito grande nos chamados jogos *on-line*. O conceito de jogo *on-line* permite que várias pessoas joguem simultaneamente o mesmo jogo, porém cada pessoa pode estar em um lugar diferente no mundo. O que permite a interação destas pessoas no jogo são as redes de computadores. Também ocorreu um crescimento muito grande na quantidade de telefones celulares e outros dispositivos móveis que as pessoas possuem. Estes dispositivos começam a possuir um poder de processamento razoável, o que permite o desenvolvimento de jogos para eles. Além disto, estes dispositivos possuem mecanismos de troca de informações, o que permite que sejam desenvolvidos jogos *multi-player*, onde vários jogadores podem jogar simultaneamente o mesmo jogo.
O desenvolvimento de jogos é uma tarefa bastante interessante, já que envolve diversas áreas da Ciência da Computação, como algoritmos, estrutura de dados, redes de computadores, etc.
Os jogos são um produto de *software* e, para desenvolvê-los, existem metodologias que são um pouco diferentes das metodologias tradicionais de desenvolvimento de *software*. Esta diferença ocorre porque no desenvolvimento de jogos existem não apenas tarefas técnicas, mas também tarefas artísticas, como por exemplo o desenvolvimento dos desenhos que serão utilizados no jogo. Desta forma, a metodologia utilizada deve considerar estas tarefas e criar uma forma de interação entre as tarefas artísticas e técnicas.
Durante os últimos anos, um novo campo de estudo foi aberto na área de desenvolvimento de jogos, que é o de desenvolvimento de jogos para dispositivos móveis, tais como celulares e PDAs. A grande dificuldade no desenvolvimento de jogos para estas plataformas é a pouca capacidade de processamento dos dispositivos. No

entanto, hoje em dia tais dispositivos já possuem um poder de processamento razoável, o que permite o desenvolvimento de jogos mais elaborados. Além deste aumento de capacidade dos dispositivos, surgiram novas tecnologias que permitem o desenvolvimento de jogos portáveis, ou seja, jogos que podem ser executados em vários modelos diferentes de dispositivos móveis. A portabilidade associada à interatividade que estes dispositivos móveis proporcionam, normalmente através das redes *wireless*, faz com que jogos possam ser desenvolvidos de forma que várias pessoas possam jogar pela rede, mesmo que estas pessoas possuam modelos diferentes de dispositivos móveis.
Desta forma, estes dispositivos são uma plataforma interessante para o desenvolvimento de jogos.
Os objetivos deste trabalho estão listados abaixo:

- Apresentar técnicas e ferramentas de desenvolvimento de jogos utilizadas atualmente;
- apresentar técnicas e ferramentas de desenvolvimento de jogos para dispositivos móveis utilizadas atualmente;
- apresentar uma metodologia de desenvolvimento de jogos;
- estudar limitações dos dispositivos móveis;
- aplicar a melhor técnica para cada limitação encontrada, a fim de obter o melhor resultado;
- desenvolver um protótipo de um jogo para dispositivos móveis, com possibilidade de dois ou mais jogadores jogarem simultaneamente, de forma cooperativa.

No primeiro capítulo o conceito de jogos é apresentado. Uma metodologia utilizada no desenvolvimento de jogos é apresentada neste capítulo, assim como algumas ferramentas disponíveis atualmente para o desenvolvimento.
O segundo capítulo aborda o desenvolvimento de jogos para dispositivos móveis. Estes tipos de dispositivos são apresentados neste capítulo, assim como algumas ferramentas utilizadas atualmente no desenvolvimento de jogos. Também são apresentados alguns exemplos de jogos desenvolvidos para estes dispositivos e as dificuldades encontradas no desenvolvimento de jogos para dispositivos móveis.
O capítulo três apresenta a aplicação desenvolvida como protótipo. Este capítulo contém uma descrição da aplicação, o seu funcionamento, as dificuldades e soluções encontradas durante o desenvolvimento e a documentação de engenharia de software.

Capítulo 1
Jogos

1.1 Introdução

Jogos são programas de computador direcionados para o entretenimento das pessoas. Há alguns anos, os jogos eram muito simples, porém hoje nota-se um desenvolvimento muito grande nesta área, com jogos tridimensionais e com opções de jogo *on-line*. Os jogos tem uma influência cada vez maior na vida das pessoas e, com o advento das opções *on-line*, surgiram vários grupos de jogos na internet, onde as pessoas criam, mais do que um grupo de jogo, laços de amizade, o que reforça a tese de que os games possuem uma grande influência na vida das pessoas. Como todo produto de *software*, os jogos são desenvolvidos seguindo alguma metodologia, porém esta é um pouco diferente das metodologias tradicionais de desenvolvimento [10].

1.2 Metodologias para desenvolvimento de jogos

O processo de desenvolvimento de jogos é bastante complicado, principalmente devido ao fato de que este desenvolvimento possui atividades técnicas, como programação, por exemplo, e atividades artísticas, como o desenvolvimento dos desenhos que o jogo irá utilizar.
Um jogo é um produto de *software* e, como tal, segue uma metodologia de desenvolvimento, que busca organizar o processo de desenvolvimento de forma a garantir que o produto final seja um produto de qualidade.
Um processo típico de engenharia de *software*, que possui as fases de análise, projeto, implementação e teste, pode ser aplicado ao desenvolvimento de jogos, porém, devido à natureza artística de alguns processos envolvidos no desenvolvimento, metodologias rígidas não são muito empregadas [10].
A equipe de desenvolvimento é uma mistura de pessoas com conhecimentos puramente técnicos, como por exemplo especialistas em estruturas de dados e algoritmos, com pessoas que possuem profundo conhecimento artístico, como desenhistas, por exemplo.

Antes de começar o processo de criação, é preciso definir algumas características do jogo que será desenvolvido. Entre elas, pode-se destacar:

- **Concepção inicial:** é a idéia básica por trás do jogo, que pode servir como premissa para iniciar o processo de desenvolvimento;

- **Publicação e distribuição:** quando o jogo está pronto, é preciso que o mesmo seja colocado no mercado. A escolha de um bom distribuidor pode ser crucial para o sucesso comercial do jogo. O distribuidor é o responsável pela produção em larga escala do jogo, assim como pela produção de embalagens e manuais. Normalmente o distribuidor é escolhido no início do processo de desenvolvimento;

- **Investidores:** é comum que exista a necessidade de se buscar um investidor, que colocará recursos financeiros no desenvolvimento do projeto de um jogo, porém no Brasil conseguir um investidor ainda é uma tarefa bastante complicada;

- **Público-alvo:** embora muitas pessoas entendam que os jogos são direcionados para as crianças, isto não é mais verdade nos dias de hoje. Muitos jogos possuem cenas de violência que não são apropriadas para uma criança. No desenvolvimento de um jogo, é preciso considerar o público-alvo do mesmo, ou seja, a que tipo de pessoas o jogo é destinado. Pode-se definir, por exemplo, que o jogo é apropriado para pessoas com mais de 18 anos, caso existam cenas de violência;

- **Plataforma-alvo:** a plataforma é o *hardware* sobre o qual o *game* (jogo) será executado. A decisão da plataforma é importante, pois pode direcionar outros processos de desenvolvimento. Por exemplo, se o jogo será executado em consoles de *videogame*, a fase de testes do *software* é muito importante, e deve levar em conta todos os aspectos que podem levar a erros no código do jogo, já que não é possível instalar *patches* (correções) em um jogo de *videogame* que já foi publicado. Se o jogo é direcionado para PCs, é possível distribuir correções após a publicação do *game*, e a fase de testes deve ser conduzida de outra forma, e deve considerar os diferentes tipos de *hardware* (como por exemplo diferente modelos de placas de vídeo) que poderão ser utilizados pelo usuário final;

1.2.1 Fases do processo de desenvolvimento

O processo de desenvolvimento de um jogo envolve as seguintes fases:

- ***Design:*** esta fase é parecida com a etapa de análise de requisitos de um processo tradicional de desenvolvimento de *software*. Nesta fase são levantadas as seguintes informações:

    - quais as tecnologias atuais de desenvolvimento;

- quais as ferramentas acessíveis;
- qual é o estilo musical da atualidade;
- qual é o estilo visual da atualidade;
- como será a interface com o jogador;
- qual será o estilo do jogo;
- qual será a dificuldade do jogo;
- qual será a estória do jogo;
- quantas fases ou estágios o jogo irá possuir.

Com as informações obtidas durante a concepção inicial e durante a fase de *design*, é possível obter as principais características que o jogo deve possuir.
Os seguintes documentos são produtos da fase de *design*:

- **Proposta:** documento inicial, que é apresentado aos investidores, contendo uma introdução, a estória do jogo, a motivação para o desenvolvimento, descrição do jogo, características chave do jogo e plataforma(s) alvo;

- **Documentação Funcional:** documento detalhado contendo a descrição do jogo e informações como funcionamento do jogo, interface com o usuário, estória, fases e estágios, características de som, arte e vídeo. Esta documentação ainda não possui detalhamento técnico;

- ***Storyboard*:** em muitos jogos é desenvolvido o *storyboard*, que é uma espécie de roteiro, que serve para ilustrar como será a seqüência do jogo pelos estágios e fases.

- **Projeto:** durante a fase de projeto, não é utilizada nenhuma metodologia específica. Isto não significa que não é possível utilizar alguma metodologia nesta fase, porém esta varia de equipe para equipe. Na fase de projeto, existe uma documentação parecida com a documentação funcional desenvolvida durante a fase de *design*, porém considerando agora aspectos técnicos, como por exemplo a descrição dos algoritmos e estruturas de dados que serão utilizados. Tendo os requisitos do jogo e o projeto sobre como estes requisitos serão implementados, a equipe deve decidir qual tecnologia será utilizada na implementação. É interessante notar que muitas vezes a equipe de desenvolvimento implementa internamente ferramentas que auxiliam na criação do jogo. Isto pode ser feito quando a equipe quer adicionar um novo efeito, que não é implementado pelas ferramentas disponíveis.
  Na fase de projeto também é criado um cronograma, que contém uma previsão geral sobre o processo de desenvolvimento. As tarefas são divididas entre a equipe de desenvolvimento, e esta divisão deve levar em conta, além do tamanho da equipe e do prazo estipulado para término do desenvolvimento, os orçamento disponível.

Nesta fase os trabalhos artísticos também são iniciados, como por exemplo o desenvolvimento de modelos tridimensionais de cenários e personagens. O processo de desenvolvimento de músicas e efeitos sonoros do jogo também começa nesta fase;

- **Implementação:** nesta fase o código do jogo é desenvolvido. Os programadores envolvidos nesta fase devem possuir um profundo conhecimento técnico, e, além disso, devem ser criativos e devem manter uma boa relação com os artistas e designers;

- **Testes:** os testes são realizados durante todo o desenvolvimento do jogo. Estes testes não apenas identificam problemas com o código do jogo, como também procuram verificar a aceitação do jogo pelos usuários finais, através de testes conhecidos como *play-test*. Existem algumas abordagens diferentes nesta fase, das quais pode-se destacar:

    - **Usuários jogam e o programador observa:** neste tipo de teste, o programador verifica as reações do usuário enquanto este joga, e anota as dificuldades encontradas pelo jogador durante as fases. Neste tipo de teste, o programador não tem contato com o jogador. Num jogo de luta, por exemplo, os programadores podem observar a reação dos jogadores às cenas violentas exibidas, e ajustar estas cenas para produzir maior ou menor impacto, dependendo do resultado do teste e do público-alvo do jogo;

    - **Usuários jogam e o programador faz perguntas:** o programador permanece no mesmo ambiente que o jogador e, conforme este avança pelas fases, o programador faz perguntas sobre dificuldades e aspectos do jogo para o jogador. Num jogo de ação, por exemplo, o programador pode ter dúvidas na hora de escolher que tipo de iluminação aplicar a um cenário. O programador implementa o cenário com algumas variações do sistema de iluminação, e pode, através deste tipo de teste, identificar qual tipo de iluminação agrada mais os jogadores;

    - **Usuários fazem relatórios de teste:** nesta abordagem, os usuários jogam e fazem relatórios sobre o funcionamento do jogo. Existem pessoas especialistas em testar jogos. É preciso tomar cuidado com este tipo de teste, pois os jogadores tem que testar todos os aspectos do jogo, inclusive fases escondidas e “surpresas” que o jogo oferece e, portanto, é possível que estas “surpresas” sejam reveladas antes do lançamento do jogo. Um jogo que está sendo finalizado normalmente passa por este tipo de teste, onde alguns jogadores são “recrutados” e devem testar todos os aspectos do jogo.

- ***Feedback:*** após o desenvolvimento e a distribuição de um jogo, muitas empresas utilizam algum mecanismo que permita obter um *feedback* dos jogadores, para realizar correções em problemas encontrados pelos jogadores e/ou para recolher sugestões, que serão utilizadas numa próxima versão do jogo.

1.2.2 Ferramentas para o desenvolvimento de jogos

No desenvolvimento de jogos, muitas ferramentas são utilizadas. Algumas destas ferramentas estão relacionadas com os aspectos artísticos de desenvolvimento, e nesta categoria pode-se destacar programas como Photoshop, 3D Studio Max, Softimage XSI e Maya [15], que são utilizados para o tratamento de imagens e para a criação de modelos tridimensionais.
Para a programação, existem algumas bibliotecas disponíveis para os programadores, entre elas pode-se destacar:

- **OpenGL:** a biblioteca OpenGL [11] é uma interface de *software* para dispositivos de *hardware*. Esta interface consiste em cerca de 150 comandos distintos usados para especificar os objetos e operações necessárias para produzir aplicativos tridimensionais interativos.
  Diante das funcionalidades providas pelo OpenGL, tal biblioteca tem se tornado um padrão amplamente adotado na indústria de desenvolvimento de aplicações e jogos. Este fato tem sido encorajado também pela facilidade de aprendizado, pela estabilidade das rotinas, pela boa documentação disponível e pelos resultados visuais consistentes para qualquer sistema de exibição concordante com este padrão.
  A biblioteca OpenGL não se preocupa com detalhes como por exemplo o tratamento de eventos e a manipulação de janelas. Estas tarefas são realizadas por alguma biblioteca dependente do sistema operacional. Desta forma, esta biblioteca possui alto grau de portabilidade entre os diferentes sistemas operacionais.
  Desde sua introdução em 1992, a biblioteca OpenGL transformou-se num padrão extensamente utilizado pelas indústrias. Ela promove a inovação e acelera o desenvolvimento de aplicações incorporando um grande conjunto de funções de renderização, mapeamento de texturas, efeitos especiais e outras poderosas funções de visualização. O jogo *Half-Life* [16], apresentado na Figura 1, é um exemplo de jogo que utiliza a biblioteca OpenGL.

Figura 1: *Half-Life* - Jogo desenvolvido com OpenGL.

- **DirectX:** a biblioteca DirectX [17] é uma biblioteca para criação de aplicações gráficas interativas desenvolvida pela Microsoft. Esta biblioteca permite que os programas se comuniquem diretamente com o *hardware*, através de uma interface de programação. A biblioteca DirectX é subdividida em outras bibliotecas:

    - **Direct Draw:** contém funções de desenho de figuras 2D;
    - **Direct Sound:** contém funções para manipulação de áudio;
    - **Direct Input :** contém funções para comunicação com os dispositivos de entrada e saída;
    - **Direct3D:** contém funções de manipulação de gráficos tridimensionais.

  A biblioteca DirectX é muito utilizada no desenvolvimento de jogos, porém ela não possui como característica a portabilidade e, desta forma, jogos desenvolvidos com DirectX devem ser obrigatoriamente executados em computadores com o sistema operacional Windows. O jogo *Flight Simulator 2004* [18], apresentado na Figura 2, é um exemplo de jogo desenvolvido com a biblioteca DirectX.

Figura 2: *Flight Simulator 2004* - Jogo desenvolvido com DirectX.

Muitas vezes trabalhar diretamente com estas bibliotecas de programação pode ser muito complicado, por isso existem alguns ambientes de desenvolvimento que podem ser utilizados para a criação de um jogo, que trabalham internamente com as bibliotecas gráficas, mas escondem toda a complexidade deste trabalho do programador de jogos. Estes ambientes procuram facilitar o desenvolvimento, reunindo em um só produto vários utilitários que facilitam o processo de desenvolvimento. Abaixo estão listados alguns destes ambientes:

- ***3D GameStudio:*** o *3D Game Studio* [12] é um ambiente de desenvolvimento de aplicações 2D e 3D em tempo-real. Ele combina funções avançadas de manipulação de imagens bidimensionais e tridimensionais, funções de simulação de física, editores de mapas e modelos, um compilador que utiliza uma linguagem de programação similar à linguagem C e uma biblioteca de objetos 3D. Com o *3d Game Studio* é possível criar jogos 2D, jogos 3D, jogos de esportes, apresentações 3D interativas, etc;

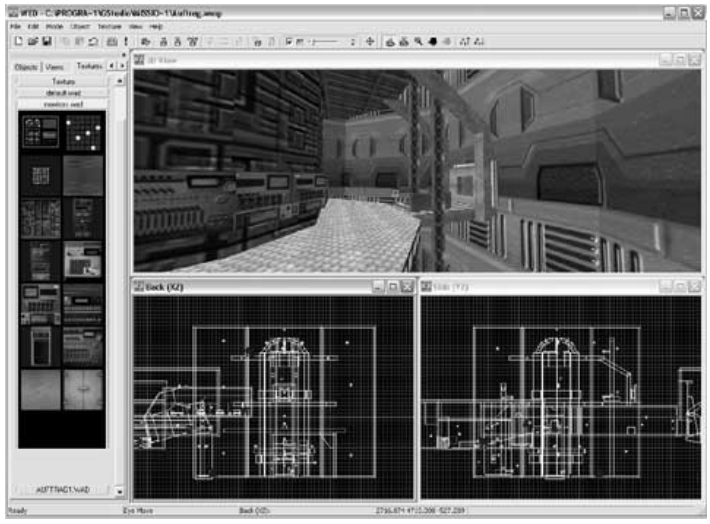


Figura 3: Interface do *3D Game Studio*.

- ***DarkBasic:*** o *DarkBasic* [13] é um ambiente de desenvolvimento que trabalha com uma linguagem de programação parecida com o *basic*, que é uma linguagem de fácil aprendizado.
  Características do *DarkBasic*:

  - Código compilado e otimizado, o que torna os executáveis gerados muito mais rápidos;

  - suporte a BSP e compilador de BSPs embutido. Suporta também modelos MD2 (Quake 2), MD3 (Quake 3), MDL (Half-Life), X (DirectX) e 3DS (3D Studio) e animação por bone com *mesh deformation*;

  - técnicas avançadas de renderização, como por exemplo *bump mapping;*

  - suporte a modelos animados.

Figura 4: Exemplo de jogo criado com o *DarkBasic*.

- **Kits de desenvolvimento proprietários:** as empresas que fabricam consoles de *videogame* possuem ambientes de desenvolvimento proprietários. Estes ambientes procuram explorar as características de cada console e fornecer um conjunto de ferramentas que facilite o desenvolvimento de jogos para este console. Por exemplo, o console *Playstation 2* [22], fabricado pela Sony, possui um ambiente de desenvolvimento chamado *PS2 Development Kit*, que deve ser adquirido pelas empresas interessadas no desenvolvimento de jogos para este console.

Capítulo 2

Jogos para dispositivos móveis

2.1 Dispositivos móveis

Utilizamos o termo dispositivos móveis para indicar dispositivos com pequeno poder de processamento, que possuam algum mecanismo de troca de informações pela rede e que possuam como característica a mobilidade, ou seja, as pessoas podem carregar estes aparelhos consigo. Exemplos típicos destes aparelhos são os telefones celulares, os handhelds e os palms.

Figura 5: Dispositivos Móveis.

Uma característica interessante que foi levada em consideração na escolha destes dispositivos como plataforma-alvo para o desenvolvimento de jogos é o fato de que as

pessoas carregam este tipo de dispositivo para qualquer lugar, e podem, portanto, jogar a qualquer hora e em qualquer lugar.

2.2 Implementação de jogos para dispositivos móveis

Existem algumas tecnologias que podem ser utilizadas para o desenvolvimento de jogos para dispositivos móveis, das quais pode-se destacar:

2.2.1 Jogos nativos

Alguns jogos são programados na linguagem nativa do dispositivos, e são gravadas pelo próprio fabricante no *chipset* do aparelho. Estes jogos vem de fábrica, e não é possível instalar mais jogos deste tipo após o dispositivo sair da fábrica.
O jogo *snake* [6], presente em praticamente qualquer telefone celular, é um exemplo de jogo nativo.

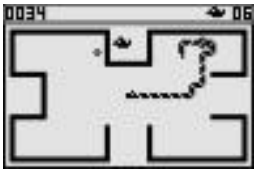


Figura 6: *Snake* - Exemplo de jogo nativo.

2.2.2 Jogos criados com SMS

SMS é um serviço de transferência de pequenas mensagens de texto entre telefones celulares [6].
Um jogo que utiliza este serviço normalmente trabalha da seguinte forma: um jogador envia uma mensagem de texto que está relacionada com as regras do jogo. Um servidor de jogo recebe esta mensagem, realiza o processamento necessário e retorna uma mensagem com os resultados. Além de ser muito simples para o desenvolvimento de jogos mais elaborados, o SMS é um serviço caro. Um usuário paga em média 10 centavos por mensagem transmitida. A Figura 7 mostra um exemplo de um jogo criado com SMS. O nome do jogo é *Pet*. Nele o jogador escolhe um animal pelo qual ele será representado, e, através de mensagens de texto que o usuário envia ao servidor, as características deste animal são modificadas. É interessante notar que a interface com o usuário é bastante simples.

Figura 7: *Pet* - Jogo criado com SMS.

2.2.3 Jogos criados com WAP

Desde 1999, todos os celulares possuem um *browser*(navegador) para o protocolo WAP (*Wireless Application Protocol – Protocolo para aplicações wireless*). Um browser WAP é parecido com um browser comum para web. A diferença é que um *browser* WAP é muito mais simples que um *browser* para web, pois ele considera as limitações de um dispositivo móvel. Um jogo WAP funciona da seguinte maneira: o usuário digita o endereço (URL) do jogo, e recebe uma espécie de formulário que é chamado de *card*. O usuário então deve interagir com este *card* , que pode conter texto, imagens e pequenos campos de formulário. Toda a interação entre o jogador e o servidor de jogo é feita através destes *cards*. A Figura 8 mostra um jogo criado com a tecnologia WAP, conhecido como *Gladiator* [6], onde o jogador controla um lutador, e luta contra o lutador de outro jogador. O usuário, na sua vez de jogar, seleciona qual região do corpo do lutador adversário ele quer acertar e que tipo de golpe ele vai acionar para atacar o adversário. Cada vitória acrescenta mais créditos ao lutador do jogador, que fica mais forte para a próxima luta.

Figura 8: *Gladiator* - exemplo de jogo criado com WAP.

2.2.4 Jogos criados com J2ME (*Java 2 Micro Edition*)

J2ME (*Java 2 Micro Edition*) [2] é uma versão simplificada da plataforma Java de desenvolvimento. O J2ME foi criado para ser executado em ambientes com pouco poder de processamento e pouca memória. O J2ME é interpretado, de forma que é preciso existir uma máquina virtual, que é um programa que irá traduzir dinamicamente as instruções escritas em Java para instruções nativas do sistema. Isto garante à tecnologia J2ME a portabilidade e, desta forma, um programa escrito em J2ME pode ser executado em várias plataformas diferentes, sem alteração no código.
Apesar disto, alguns fabricantes adicionam bibliotecas de programação que são específicas para algum aparelho, o que pode comprometer a portabilidade [6], por isso é importante utilizar apenas as bibliotecas padrão, para que a portabilidade seja mantida. A desvantagem da utilização desta tecnologia é que ocorre uma pequena perda de performance, devido à sobrecarga de processamento causada pela execução da máquina virtual.

A Figura 9 apresenta um exemplo de um jogo desenvolvido com a tecnologia J2ME, conhecido como *Metal Slug Mobile* [19].

Figura 9: *Metal Slug Mobile* - exemplo de jogo criado com J2ME.

2.2.5 Jogos criados com C++

Muitos fabricantes, como por exemplo a Nokia e a Sony possuem compiladores para a linguagem C++. Desta forma, é possível desenvolver jogos utilizando esta linguagem. As vantagens de utilização desta linguagem incluem acesso direto ao *hardware* e maior velocidade de execução do código. A desvantagem é que é preciso recompilar o código para cada modelo de aparelho no qual o jogo será executado. A Figura 10 mostra um jogo criado para o dispositivo portátil N-Gage [20], da Nokia, que é um misto de telefone celular e *videogame*.

Figura 10: *Tomb Raider* - Exemplo de um jogo criado com C++.

2.2.6 Limitações no desenvolvimento de jogos para dispositivos móveis

Os dispositivos móveis apresentam algumas limitações para o desenvolvimento de jogos, das quais pode-se destacar:

2.2.6.1 Velocidade do display

Um dos primeiros problemas encontrados está relacionado com a velocidade dos *displays*. Enquanto os consoles e PCs atualmente possuem uma taxa de atualização que varia entre 30 fps (frames por segundo) e 60 fps, a geração atual de dispositivos móveis (celulares e PDAs) trabalha a uma taxa máxima de 10 fps [3]. Desenhar diretamente numa tela com esta taxa de atualização muitas vezes causa um efeito conhecido como *flicker* (cintilação), que faz com que a imagem pareça estas piscando na tela . Para resolver este problema, utilizamos uma técnica conhecida como *back-buffer*, onde a imagem do jogo é desenhada primeiro num *buffer*, que o usuário da aplicação não está vendo. Quando todo o processo de desenho no *buffer* terminar, então este é copiado para a tela principal do jogo. Esta operação faz com o que o efeito de *flicker* desapareça.

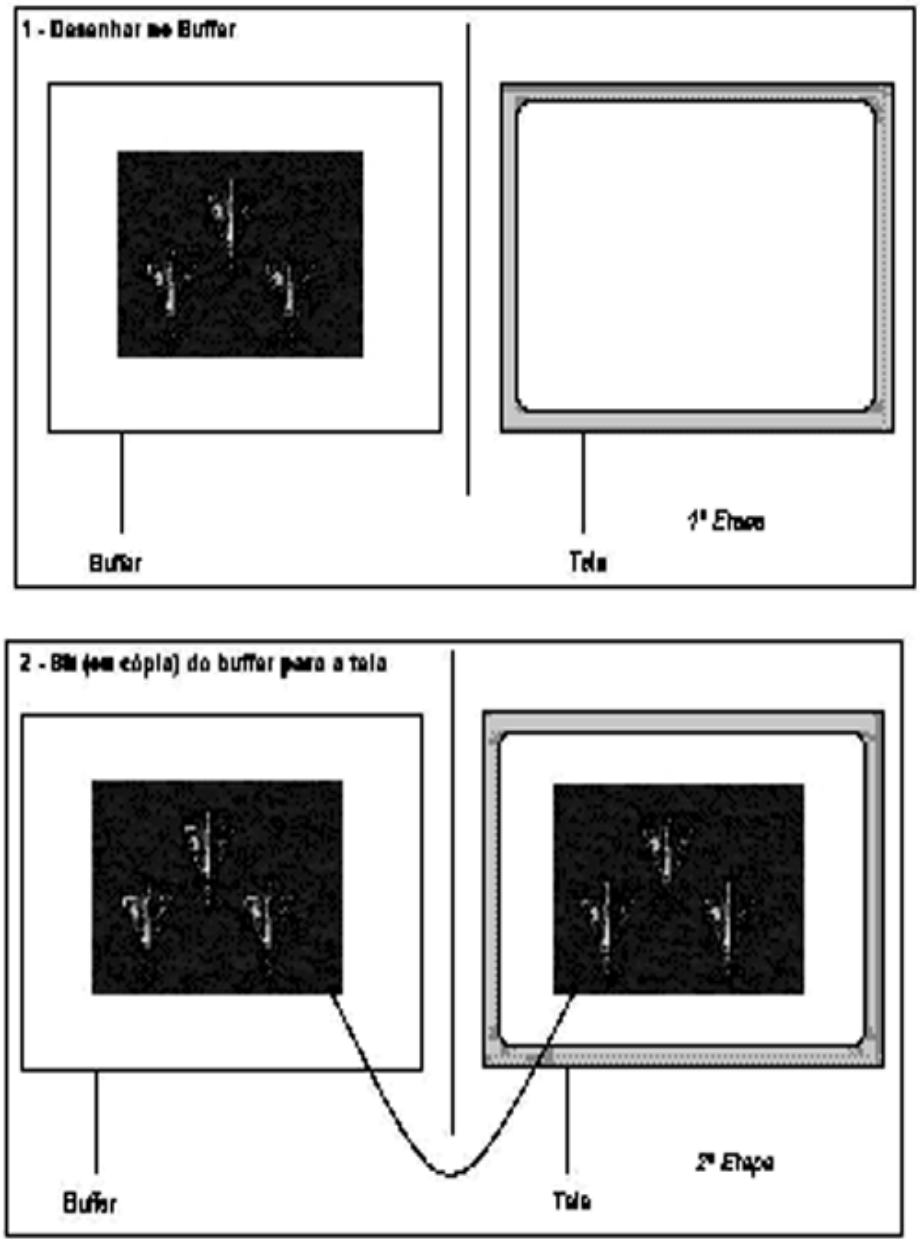


Figura 11: *Buffer* utilizado para eliminar cintilação.

2.2.6.2 - Limitação de memória e tamanho das imagens

Os dispositivos móveis possuem limitação de memória, o que faz com que o número de imagens que podem ser utilizadas para desenvolver o jogo seja pequeno. Os arquivos de imagens guardam, além da própria imagem, informações de cabeçalho, como por exemplo o tamanho do arquivo, cores, etc. Se um personagem do jogo possuir dez imagens, cada uma

representando um movimento do personagem, e se cada imagem for salva em um arquivo diferente, ocorrerá um desperdício de memória, pois para cada imagem o arquivo teria que guardar as informações de cabeçalho. Embora este desperdício de memória não seja relevante para o desenvolvimento de jogos para outras plataformas, nos dispositivos móveis este é um fator relevante. Para resolver este problema, várias imagens são salvas num único arquivo e, desta forma, a informação referente ao cabeçalho é salva apenas uma vez.

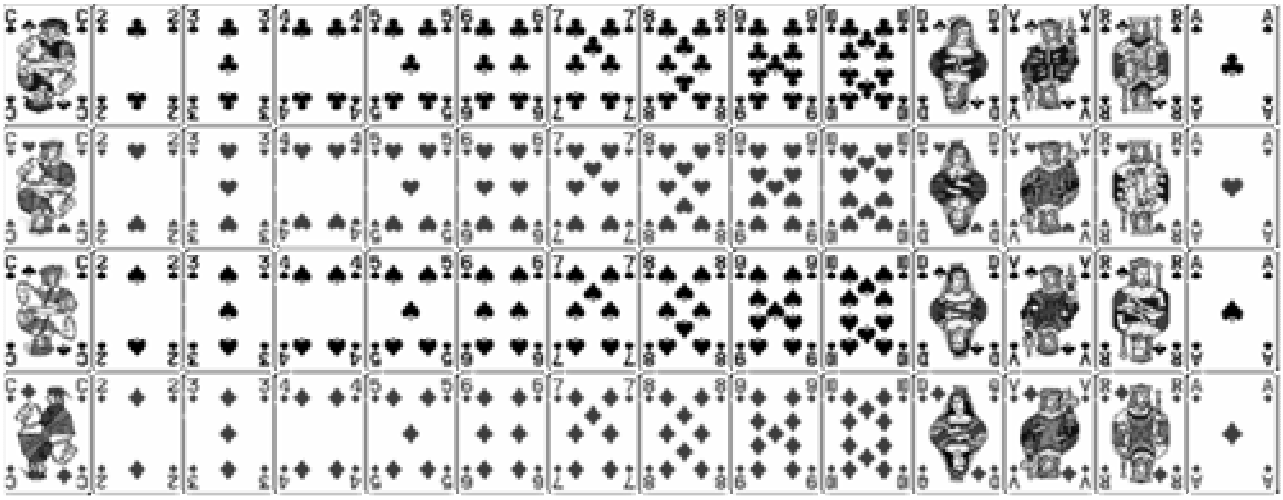


Figura 12: Exemplo de várias imagens num único arquivo.

2.2.6.3 Problemas com o teclado e com o tamanho da tela

Os dispositivos móveis apresentam diferentes configurações de teclado. Se mapearmos as teclas que o jogo irá utilizar de forma estática, problemas podem ocorrer, já que o posicionamento das mesmas em diferentes dispositivos pode variar muito. A solução, neste caso, é utilizar uma abordagem em que o mapeamento de teclas seja dinâmico. Desta forma, o usuário poderá modificar as teclas de ação do jogo de acordo com o seu dispositivo. Além do *layout* do teclado, outro problema ocorre com os dispositivos móveis. Estes dispositivos normalmente não reagem instantaneamente quando pressionamos uma tecla, e muitos deles não suportam o pressionamento simultâneo de duas ou mais teclas [6]. A solução destes problemas está na escolha do estilo do jogo, que não deve exigir tais características.

Com relação ao tamanho da tela, nota-se que ele varia de dispositivo para dispositivo. Algumas bibliotecas de programação oferecem mecanismos que permitem adaptar o jogo de acordo com o tamanho da tela do jogador.

É preciso verificar, na biblioteca escolhida para implementação, se existe uma forma de contornar este problema. Se não existir, é preciso implementar manualmente este mecanismo ou escolher apenas um modelo de dispositivo como plataforma-alvo.

Figura 13: Diferentes configurações de teclado.

2.2.6.4 Transparência de Imagens

Algumas bibliotecas de desenvolvimento não suportam diretamente a transparência nas imagens. Caso não exista este suporte por parte da biblioteca utilizada no jogo, é preciso desenvolver algum mecanismo que trate deste problema e, para isto, existem três abordagens [3]:

- Desenhar as figuras utilizando apenas as primitivas gráficas disponibilizadas pela biblioteca. Esta solução não é a ideal pois as figuras teriam que ser muito simples e a velocidade ficaria comprometida ;

- Implementar a transparência utilizando algoritmos que cortam a imagem nos pontos onde ela deveria ser transparente. Este algoritmo também não é o mais indicado, pois os cortes muitas vezes não conseguem cortar as partes que deveriam ser transparentes .

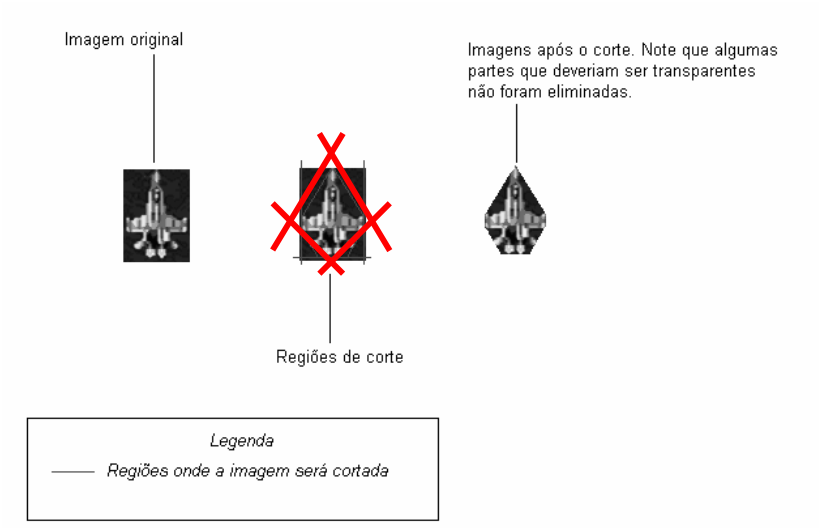


Figura 14: Exemplo de técnica para implementar transparência.

- Escolher uma cor para o fundo da imagem e utilizar esta cor apenas no fundo. Quando for desenhar a imagem, desenhá-la ponto a ponto e não desenhar os pontos que possuem a cor escolhida (que são os pontos que deveriam ser transparentes). Este algoritmo também não é rápido, porém é o que apresenta os melhores resultados.

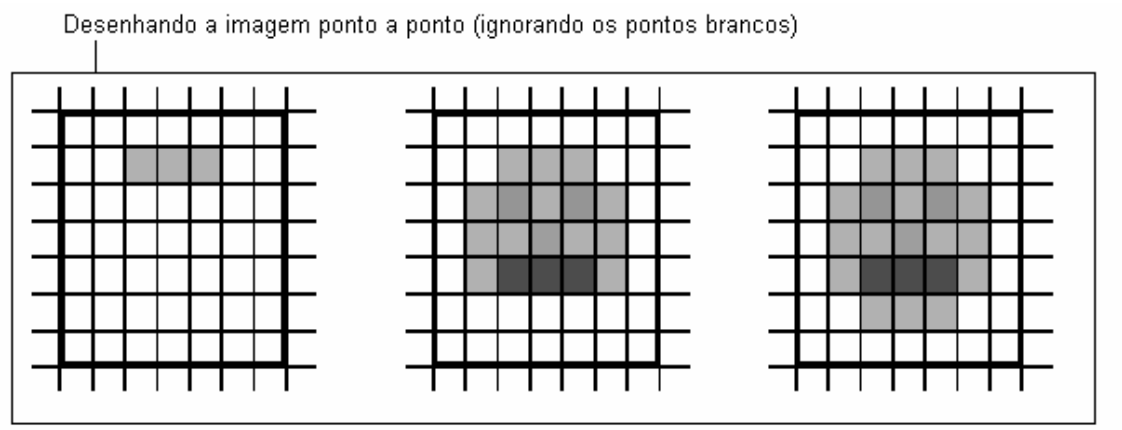


Figura 15: Técnica de implementação de transparência.

Capítulo 3
Estrutura da aplicação

3.1 Jogo escolhido para implementação

O jogo escolhido para implementação chama-se mau-mau. Mau-mau é um jogo de cartas onde o jogador pode jogar sozinho, contra o computador, ou on-line, de 2 a 4 jogadores. As regras para este jogo são as seguintes:

- Cada jogador recebe 7 cartas no início do jogo. Uma carta é virada do baralho e as outras permanecem no monte;
- O jogo começa. O primeiro jogador deve jogar uma carta do mesmo valor ou naipe da carta que está virada no baralho ou comprar uma carta e pular a sua vez;
- Ganha o jogo quem conseguir ficar sem nenhuma carta;
- Se o baralho ficar vazio durante o jogo, o jogador que possuir o menor número de cartas será eleito vencedor.

O jogador pode jogar algumas cartas especiais, que são citadas abaixo.
Quando o valor da carta vier seguido por um *, isto quer dizer que o efeito é válido para cartas de qualquer naipe, senão o naipe para o qual a jogada especial é válida será especificado:

- **A***: o jogador que jogar um as pode jogar novamente;
- **2***: Pode ser usado para se defender de uma carta hostil defensável: um 9 ou 10 (o dois do mesmo naipe), ou uma seqüência de 7 (naipe da última carta);

- **4 de espadas**: Todo mundo na mesa (menos quem jogou) compra uma carta. Indefensável;
- **5***: Inverte o sentido do jogo;
- **7***: O próximo deve jogar um 7 ou comprar 3 cartas. O jogador seguinte deve jogar um sete, ou comprar 6 cartas, e assim sucessivamente (o próximo jogador deve jogar um sete ou comprar 9 cartas, etc. Sempre somando 3 ao número de cartas que o próximo jogador deve comprar). Uma seqüência de setes é defensável por um dois do mesmo naipe do último sete, um coringa ou um valete;
- **8***: O jogador atual recebe uma carta do próximo jogador e pode jogar novamente;
- **9***: O próximo jogador compra uma carta; pode ser defendido por um dois do mesmo naipe, por um coringa ou por um valete;
- **10 de paus**: O próximo compra cinco cartas. Pode ser defendido pelo dois de paus , pelo coringa ou por um valete;
- **J(Valete)***: Pode ser colocado sobre qualquer carta. Ao jogar, você escolhe um naipe; o próximo jogador só pode jogar uma carta desse naipe ou um coringa ou outro valete;
- **Q(Dama)***: Pula o próximo jogador;
- **K(Rei)***: Faz o anterior comprar uma carta. Indefensável;
- **Coringa***: poder ser colocado sobre qualquer carta e anula o efeito de qualquer carta especial.

Quando o jogador possuir apenas duas cartas, ao jogar a sua penúltima carta ele deve “dizer mau-mau”. Se ele não o fizer, receberá três cartas como penalidade. A ação dizer mau-mau só pode ser acionada nesta situação. Se o jogador acionar esta ação em outra situação, a penalidade também é aplicada e o jogador recebe três cartas.

3.1.1 Funcionamento do protótipo

Neste item será descrito o funcionamento da aplicação desenvolvida como protótipo.
Requisitos para a execução do protótipo:

1. Computador com Windows 98/ME/2000/NT/XP → requisito necessário para as aplicações cliente e servidor);

2. Java Runtime Envioronment ou J2SDK 1.4.0 ou superior (diponível em http://java.sun.com ou no cd-rom fornecido junto com o trabalho) → requisito necessário para as aplicações cliente e servidor;
3. J2ME Wireless Toolkit 2.0 ou superior (diponível em http://java.sun.com/j2me ou no cd-rom fornecido com o trabalho) → requisito necessário para a aplicação cliente.

Estrutura de diretórios do cd-rom que acompanha este trabalho:

Figura 16: Estrutura de diretórios do cd-rom.

- A pasta doc possui este documento e a pasta api, que, por sua vez, possui a documentação da API desenvolvida;
- A pasta MauMau possui os arquivos da aplicação cliente;
- A pasta ServidorMauMau possui os arquivos do servidor;
- A pasta util possui os softwares necessários para executar o aplicativo (requisitos 2 e 3 listados anteriormente).

Funcionamento do protótipo:

- **Funcionamento do servidor:**

A máquina que irá executar o servidor deve possuir os requisitos 1 e 2. Se for a mesma máquina que será utilizada para testar a aplicação cliente, a mesma deve possuir também o requisito 3.
Para executar o servidor, escolha alguma máquina da rede (pode ser a mesma onde a aplicação cliente será executada). Verifique se existe algum processo sendo executado na porta 5656, que é a porta que o servidor utiliza. Se estiver, encerre este processo.
Após isto, siga os seguintes passos:

1. Copie a pasta ServidorMauMau para algum diretório da máquina;
2. Abra a pasta ServidoMauMau e edite o arquivo executarServidor.bat;
3. Na linha *set PATH=C:\Program Files\j2sdk_nb\j2sdk1.4.2\bin;* , substitua o caminho *C:\Program Files\j2sdk_nb\j2sdk1.4.2\bin* pelo caminho onde está localizado o java na sua máquina. Por exemplo, se o java está instalado em c:\j2sdk1.4.0, você deve substituir o caminho por *C:\j2sdk1.4.0\bin* , e a linha ficaria assim:

set PATH=C:\j2sdk1.4.0\bin;

4. Salve o arquivo;
5. Clique duas vezes sobre o arquivo e aparecerá a seguinte tela:

   É importante executar o arquivo executarServidor.bat à partir do diretório ServidorMauMau, pois algumas imagens utilizadas na interface gráfica são encontradas pelo aplicativo à partir deste caminho.

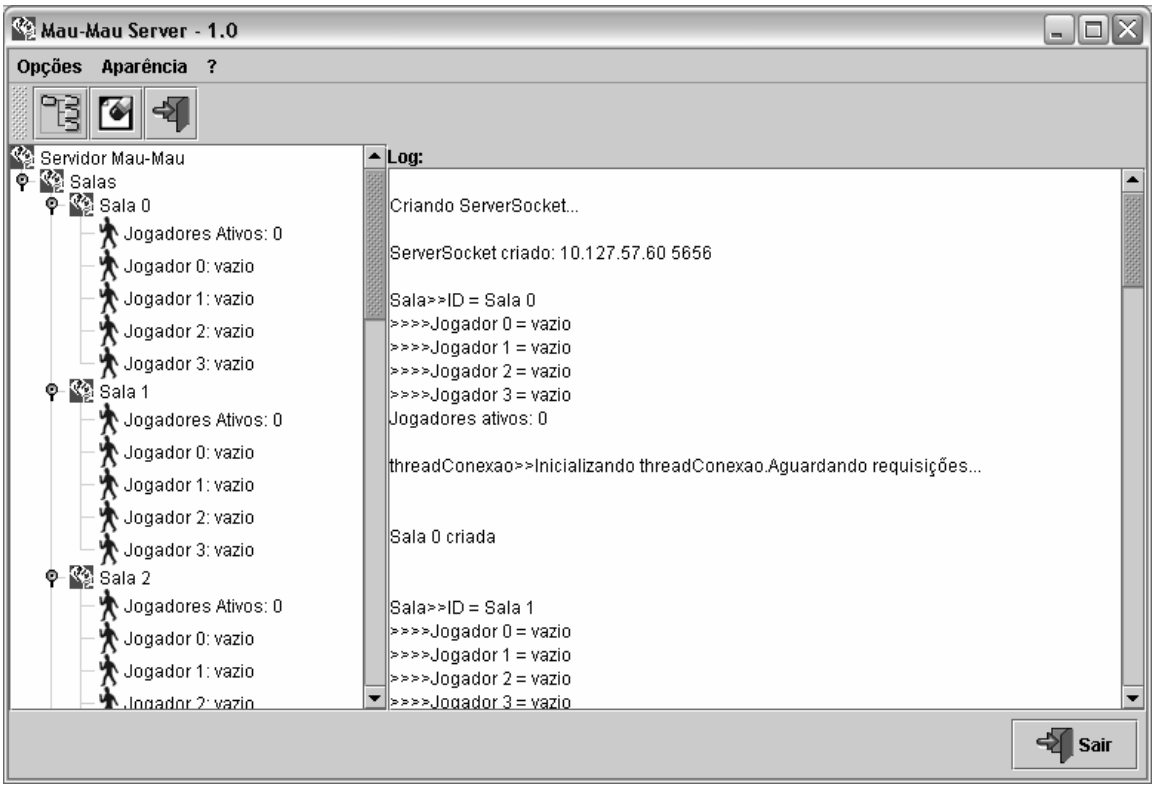


Figura 17: Interface gráfica do servidor.

A tela acima é a interface gráfica do servidor. Abaixo a função de cada parte desta interface será explicada:

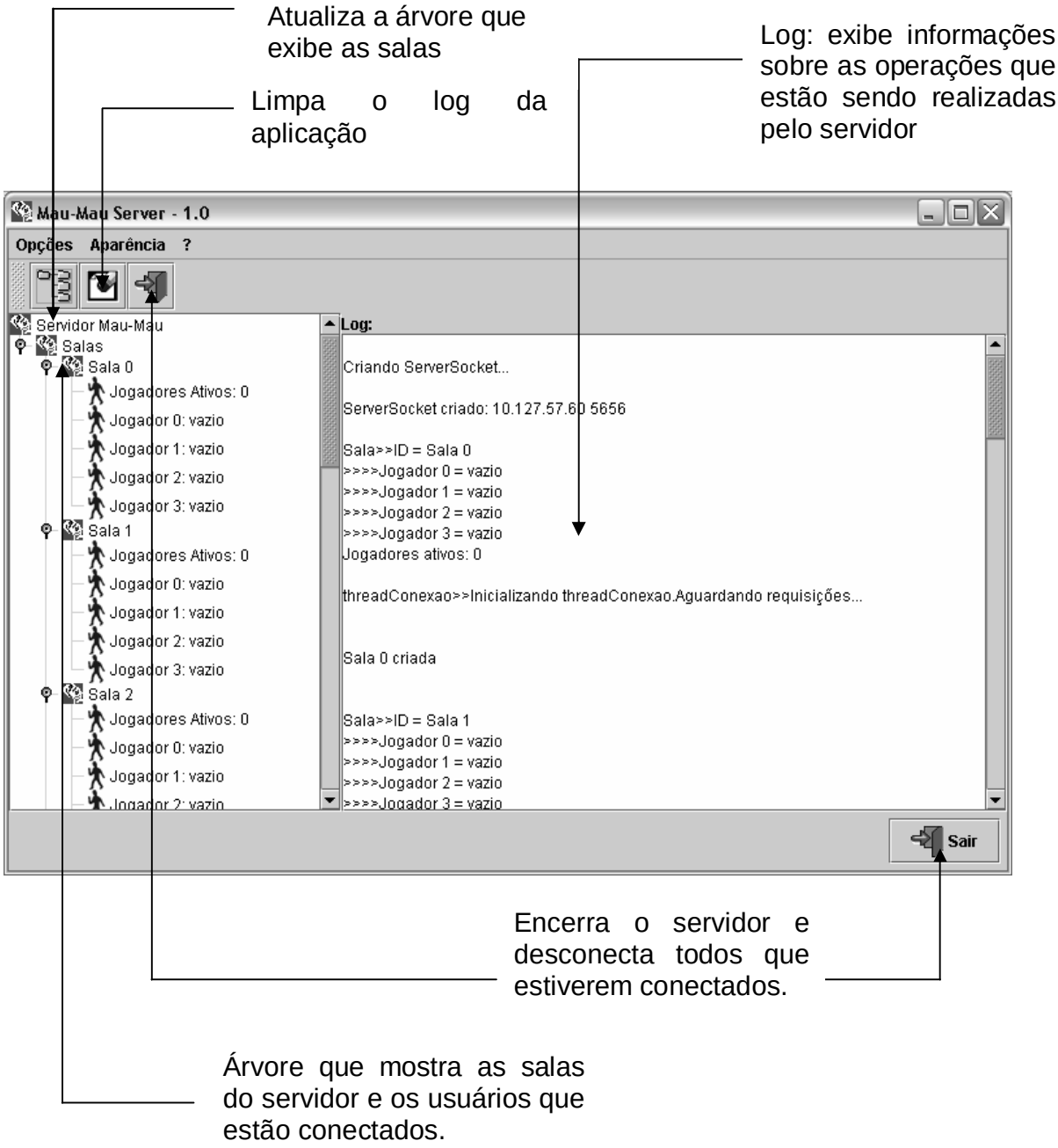


Figura 18: Interface do servidor.

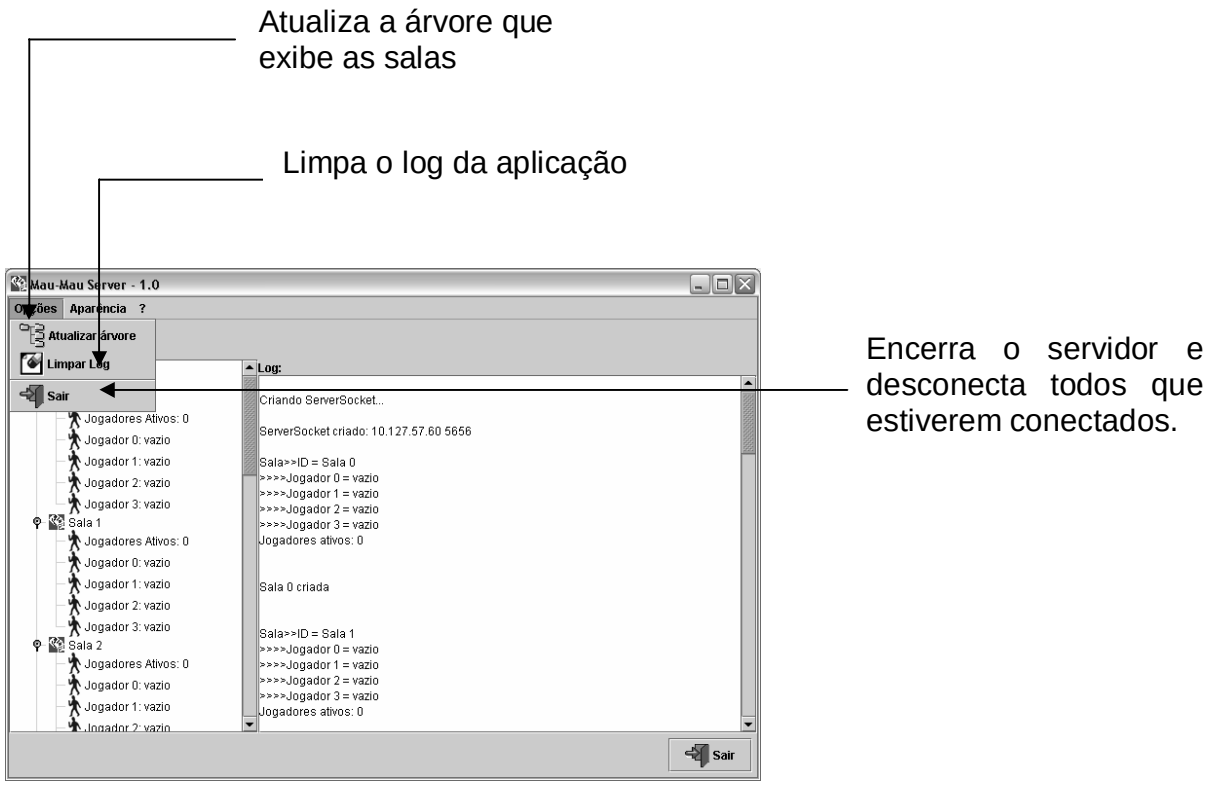


Figura 19: Descrição do menu Opções.

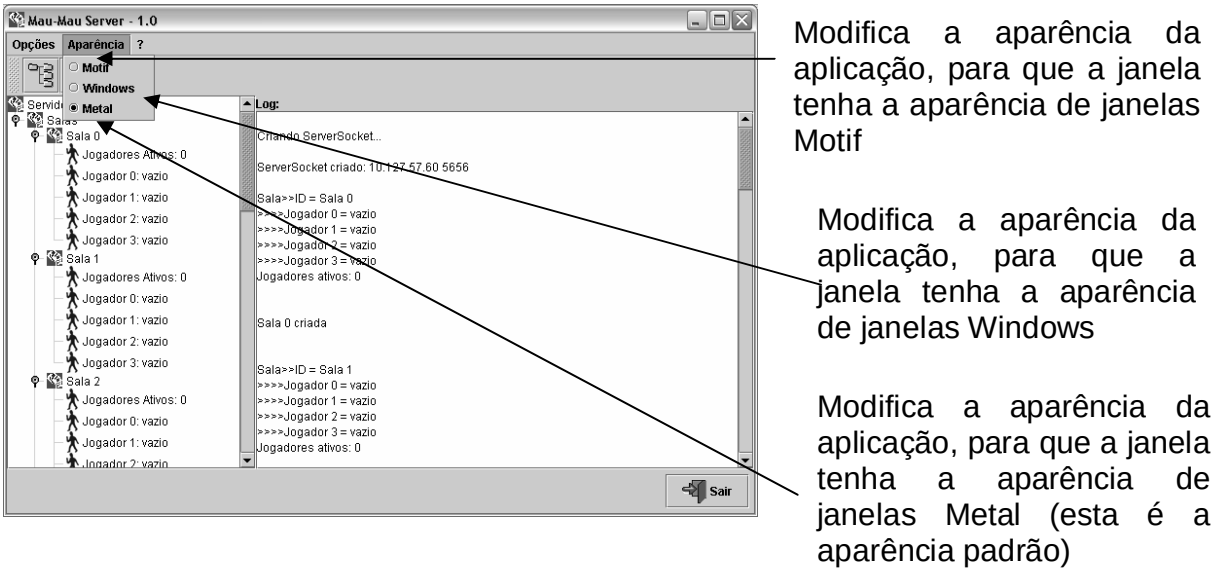


Figura 20: Descrição do menu Aparência.

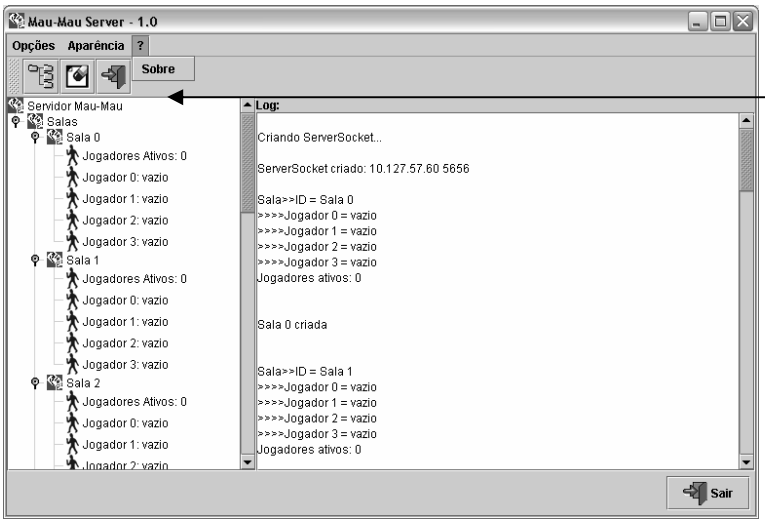


Figura 21: Descrição do menu sobre.

- **Funcionamento da aplicação cliente:**

Para executar a aplicação cliente, escolha uma das máquinas da rede e copie os arquivos MauMau.jad e MauMau.jar (que estão no diretório MauMau do cd-rom) para algum diretório da máquina que será utilizada no teste. Esta máquina deve possuir os requisitos 1, 2 e 3. Após certificar-se de que estes requisitos estejam corretamente instalados, abra o console do sistema operacional, posicione no diretório de instalação do requisito 3 (normalmente c:\WTK20); posicione no diretório bin (normalmente c:\WTK20\bin); digite a seguinte linha de comando:

emulatorw.exe -gui -Xdescriptor:

Aguarde alguns instantes e será exibida a seguinte tela:

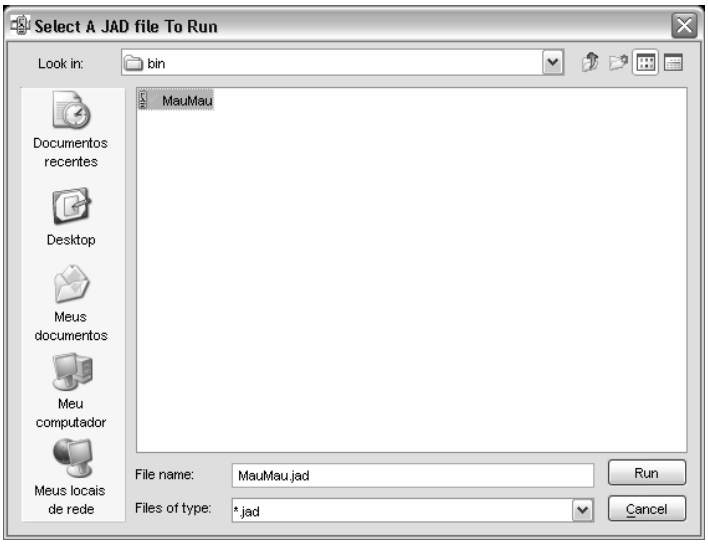


Figura 22: Tela de seleção da aplicação para execução.

Também é possível acessar esta tela através da opção *Run MIDP Application* do menu de instalação do Wireless Toolkit. Se você instalou o J2ME Wireless Toolkit 2.0 com as opções padrão, você pode selecionar esta opção através do seguinte caminho:

Menu Iniciar → Programas → J2ME Wireless Toolkit → Run MIDP Application...

Na tela que aparece, selecione o arquivo MauMau.jad no diretório onde você copiou os arquivos e clique no botão Run. Ao clicar no botão Run, a seguinte tela aparece:

Figura 23: Tela para entrada no jogo Mau-Mau.

Abaixo será descrito o funcionamento da aplicação cliente:

Figura 24: Iniciando o jogo.

Após iniciar a aplicação, a seguinte tela será exibida:

Figura 25: Tela de apresentação do jogo.

Clique aqui para sair da aplicação

Selecionando a opção menu, as seguintes opções aparecem:

- Jogar: inicia um novo jogo contra o computador;
- Jogo on-line: inicia um novo jogo pela rede, onde é possível jogar contra um, dois ou três adversários (OBS: para utilizar este modo, o servidor deve estar sendo executado, verifique como no item Funcionamento do servidor);
- Visualizador de cartas: exibe um formulário onde é possível visualizar as cartas do baralho;
- Opções: exibe opções do jogo;
- Ajuda: exibe um formulário contendo informações que ajudam o usuário na execução da aplicação.

Figura 26: Descrição do menu principal.

- **Iniciando um novo jogo contra o computador:**

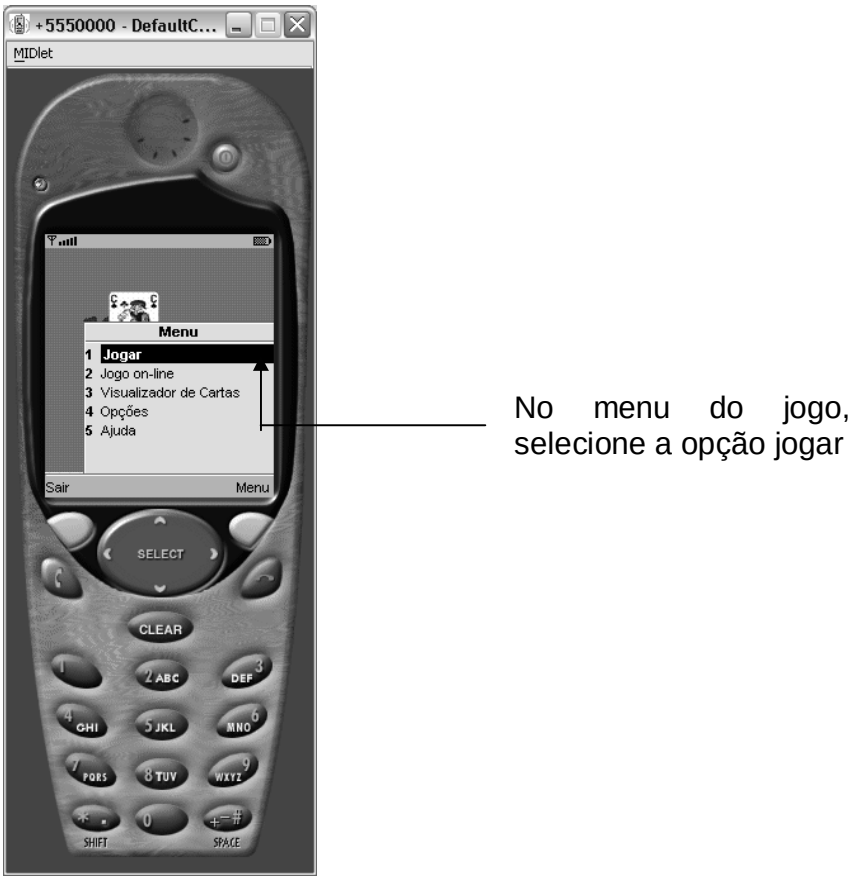


Figura 27: Iniciando um jogo contra o computador.

Após iniciar um novo jogo, a tela contendo os elementos do jogo aparece. Abaixo há uma descrição dos elementos da interface de jogo. Esta descrição é válida tanto para o jogo *on-line* como para o jogo *off-line*:

Figura 28: Interface do jogo.

As teclas de navegação e a tecla de ação são mapeadas nas teclas correspondentes no aparelho onde o jogo está sendo executado, e estas teclas variam de aparelho para aparelho.

Para jogar uma carta, selecione a mesma utilizando as teclas de navegação e pressione a tecla de ação do seu aparelho. O software irá verificar a jogada e exibirá

uma mensagem. Se a jogada for legal, o jogo passa para o próximo jogador e exibe a tela de jogo deste, senão uma mensagem de erro aparece e o jogo exibe novamente a tela de jogo, para que o jogador selecione outra carta.
Selecionando o menu da tela de jogo, as seguintes opções aparecem:

Menu da tela de jogo:

- Sair: Sai do jogo;
- Comprar carta: compra uma carta do baralho e passa para o próximo jogador
- Dizer “Mau-Mau”: diz “mau-mau” (ver regras do jogo para maiores detalhes).

A opção de comprar carta pode variar de acordo com a carta que estiver no baralho.

Figura 29: Menu da tela de jogo.

- **Tela do baralho:**

Apertando a tecla correspondente à seta de navegação para cima na interface de jogo, a tela do baralho aparece, como pode-se ver abaixo:

Figura 30: Tela do baralho.

- **Iniciando o jogo on-line:**

Para iniciar o jogo on-line, certifique-se que o servidor esteja sendo executado em alguma máquina da rede. Para saber como executar o servidor, verifique o item Funcionamento do servidor. Depois de certificar que o servidor esteja sendo executado corretamente, siga os passos (estes passos devem ser seguidos por cada jogador da rede):

1. Entre no menu Opções e digite o endereço IP e a porta do servidor:

Figura 31: Iniciando um jogo *on-line*.

2. No menu principal do jogo, selecione a opção Jogo on-line. Se uma mensagem de erro aparecer, verifique novamente no menu opções se o endereço ip e a porta do servidor estão corretos. Se estiverem corretos, verifique se o servidor está funcionando corretamente e tente novamente. Se nenhuma mensagem de erro ocorrer, a tela abaixo aparecerá. Nesta tela você deve digitar um apelido, selecionar uma das salas de jogo e selecionar a opção Entrar na Sala.

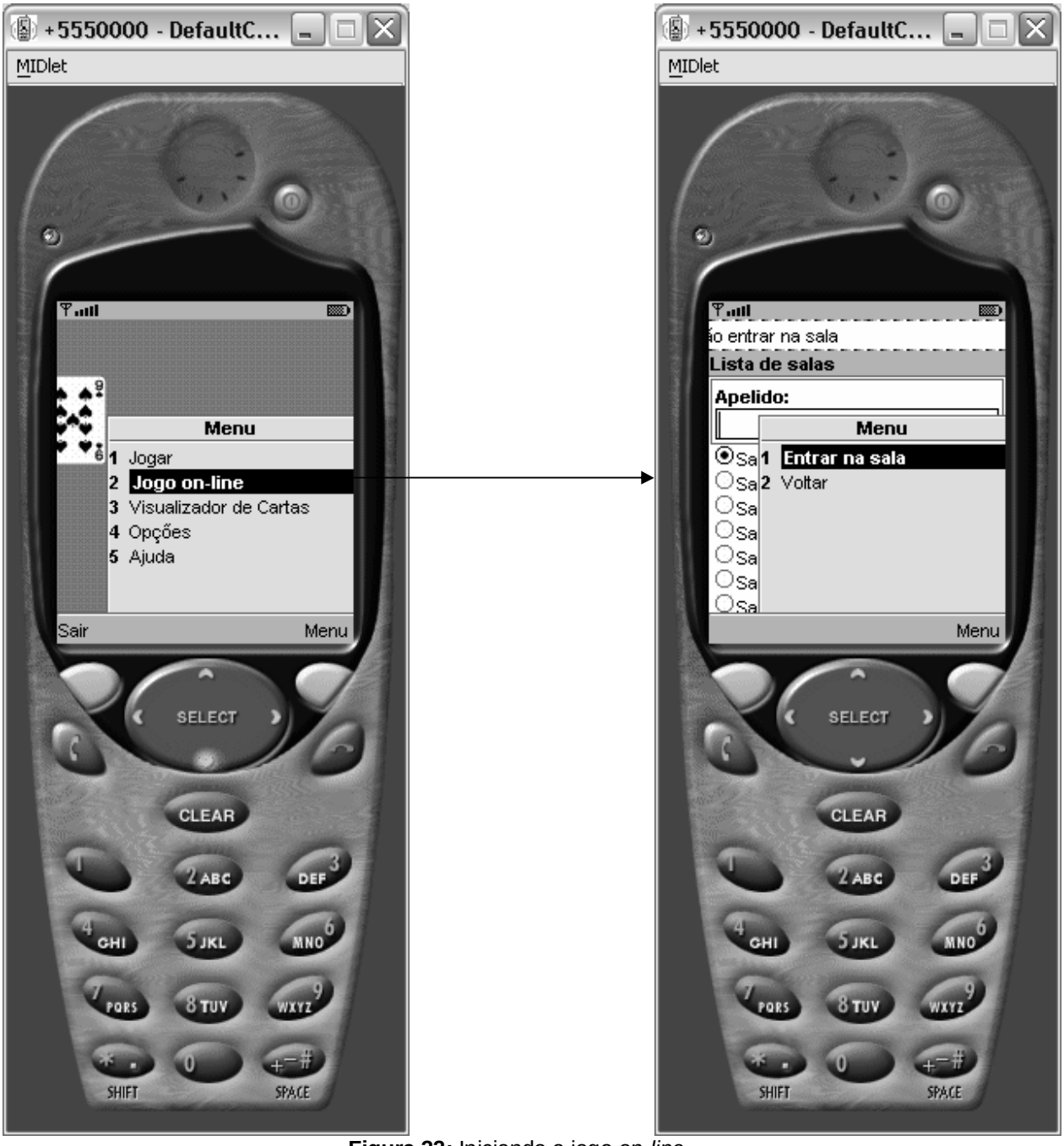


Figura 32: Iniciando o jogo *on-line*.

3. Ao entrar na sala, a seguinte tela aparece:

Figura 33: Iniciando o jogo *on-line*.

Conforme os jogadores forem entrando na sala, esta tela é atualizada nos clientes, mostrando os novos jogadores e o primeiro jogador da sala recebe a opção começar

jogo. Se você observar a tela do servidor, poderá ver que o mesmo lista os jogadores que estão na sala:

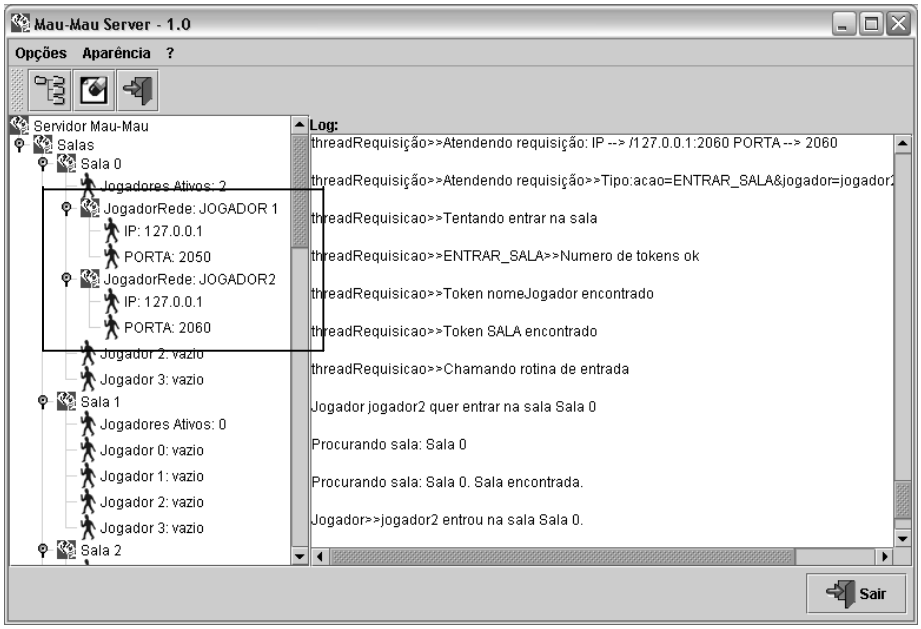


Figura 34: Tela do servidor.

Para começar o jogo, o primeiro jogador que entrou na sala deve selecionar a opção Começar Jogo, conforme mostrado abaixo:

Após selecionar esta opção, o jogo on-line começa. Os elementos da interface para o jogo on-line são os mesmos elementos do jogo off-line. Você pode obter mais detalhes sobre os elementos da interface no item Iniciando um novo jogo contra o computador.

Figura 35: Início do jogo *on-line*.

- **Visualizador de cartas:**

Para entrar no visualizador de cartas, selecione a opção Visualizador de Cartas no menu principal do jogo

Figura 36: Visualizador de cartas.

Após selecionar a opção Visualizador de Cartas, a seguinte tela aparece, onde é possível visualizar as cartas do baralho:

Clicando na opção Menu, as seguintes opções aparecem:

- Próxima carta: exibe a próxima carta;
- Carta anterior: exibe a carta anterior;
- Voltar: volta para a tela principal.

Figura 37: Visualizador de cartas.

- **O Menu Opções:**

Figura 38: Menu Opções.

Selecionando o menu opções, a seguinte tela aparece:

Selecionando a opção Opções, as seguintes opções aparecem:

- Open Mau Mau: ativa ou desativa a opção Open Mau Mau. Se estiver ativada, esta opção faz com que os jogadores consigam ver as cartas dos outros jogadores. Se estiver desativada, os jogadores podem ver apenas as suas cartas (comportamento padrão do jogo);
- Exibir informações sobre a carta selecionada: ativa ou desativa a exibição de informações sobre a carta selecionada durante o jogo;
- Endereço do servidor: neste campo deve ser especificado o endereço IP do servidor do jogo;
- Porta do servidor: neste campo deve ser especificada a porta do servidor do jogo (a porta padrão é a porta 5656).

OBS: Para o jogo on-line funcionar corretamente, as duas últimas opções devem estar configuradas corretamente.

Após configurar as opções, selecione a opção voltar para retornar à tela inicial do jogo.

Figura 39: Tela do menu Opções.

- **O menu Ajuda:**

Figura 40: Menu Ajuda.

Após selecionar a opção Ajuda, a seguinte tela aparece:

Na tela de ajuda, são apresentadas as regras do jogo, uma descrição da interface do jogo, uma breve descrição dos menus do jogo e um pequeno tutorial sobre como começar um novo jogo on-line. Para voltar para a tela inicial do jogo, selecione a opção Voltar.

Figura 41: Tela do menu Ajuda.

3.2 Escolha da tecnologia

O primeiro desafio do projeto foi procurar uma tecnologia que desse suporte às idéias iniciais. Dentre as tecnologias disponíveis atualmente, duas se destacaram por oferecerem características que são importantes para o jogo escolhido como protótipo. Estas duas tecnologias são o BREW [1] e o J2ME – *Java 2 Micro Edition* [2]. O BREW é uma plataforma que suporta algumas linguagens de programação, tais como o C e o C++. Já o J2ME é uma linguagem interpretada e, portanto, independente de sistema operacional, podendo ser executada até no próprio BREW, desde que exista uma máquina virtual Java desenvolvida para esta plataforma[5]. O BREW apresenta algumas vantagens com relação ao J2ME, como um código que é executado numa velocidade maior, já que o código desenvolvido para o BREW não é interpretado, como no caso do J2ME. Esta característica faz com que o BREW seja uma plataforma mais indicada para o desenvolvimento de jogos, porém, existe um fator muito importante, que nos levou à escolha do J2ME, que é a diferença no número de plataformas que suportam cada tecnologia. Várias empresas de telefonia já incluíram em seus dispositivos o suporte ao J2ME.

Além disso, uma API específica para jogos está disponível para a plataforma J2ME.

No início do projeto, foi utilizada a versão 1.0 do MIDP, que é a API que vem com o J2ME. Esta API era bastante limitada, de forma que nas primeiras análises foram detectados vários problemas, como por exemplo o problema da transparência de imagens, que não era suportada, ou o mapeamento de teclas para a execução do jogo. No decorrer do projeto surgiu a segunda versão do MIDP, e com ela vários destes problemas foram sanados pela API. A transparência já é suportada nesta versão da API, assim como o mapeamento de teclas, que ocorre automaticamente. A API fornece mecanismos para contornar o problema de diferença no tamanho da tela nos diferentes dispositivos.

Para resolver o problema do tamanho das imagens , as imagens das cartas do baralho foram agrupadas numa única imagem utilizando a técnica descrita no item 2.2.6.2, e uma única imagem com tamanho de 25 kb foi obtida. Cada imagem separada teria o tamanho de 4 kb, devido ao cabeçalho associado. Esta técnica foi muito importante, pois permitiu que, ao invés de se utilizar 56 imagens, totalizando 224 kb de informação, apenas uma imagem foi utilizada, e esta possui 25 kb de informação.

No desenvolvimento de sistemas que trocam informações através de uma rede de computadores, são desenvolvidos três modelos fundamentais [9]:

- **Modelo de interação:** descreve como a aplicações está distribuída, o modelo utilizado nesta distribuição e a semântica da troca de mensagens;
- **Modelo de falhas:** modelo que identifica as principais falhas que podem ocorrer no sistema e propõe soluções para estas falhas;
- **Modelo de segurança:** modelo que descreve os mecanismos de segurança implementados pela aplicação.

Abaixo são apresentados os três modelos desenvolvidos para o protótipo do jogo mau-mau.

3.3 Modelo de interação

Para a comunicação entre programas em uma rede de computadores, existem alguns modelos que podem ser utilizados. No protótipo do jogo, o modelo cliente-servidor foi utilizado, onde um computador atua como servidor, e reponde requisições que vem dos clientes.
Existem algumas formas de transmissão de mensagens disponíveis na tecnologia utilizada. Embora algumas delas sejam interessantes de se utilizar, como por exemplo a técnica de comunicação por raios infra-vermelhos, nem todos os dispositivos possuem esta interface de comunicação [7]. O único protocolo de rede que a especificação MIDP 1.0 exige em suas implementações é o http [6]. A especificação MIDP 2.0 passou a exigir, além do http, o https, que adiciona alguns recursos de segurança na transmissão das informações. Os outros mecanismos de comunicação (sms, infra-vermelho, etc) são opcionais, de forma que apenas alguns dispositivos possuem implementações para estes mecanismos. Os protocolos http e htttps são conhecidos como protocolos sem estado. Isto significa que eles não mantém nenhuma informação sobre o cliente que faz a requisição. Existem algumas mensagens utilizadas no protótipo que não guardam informações sobre o cliente, porém quando o jogo está ocorrendo de forma *on-line*, a conexão entre os clientes e o servidor precisa ficar ativa, e os protocolos http e https não permitem este comportamento de forma transparente. Por esta razão, utilizamos um mecanismo de comunicação conhecido como *socket*, que é um mecanismo que constrói um canal de comunicação bidirecional entre dois programas [9]. Este canal utiliza o protocolo TCP/IP, que garante uma comunicação confiável e, por fim, este mecanismo permite que a conexão permaneça aberta pelo tempo necessário [9].
Para que o modelo de interação cliente-servidor funcione corretamente, os clientes e o servidor devem concordar com a semântica das mensagens que são trocadas. O padrão destas mensagens é conhecido como protocolo de comunicação.
Um protocolo de comunicação define um padrão de comunicação e, seguindo este padrão, vários programas diferentes conseguem se comunicar pela rede de forma correta. Um exemplo típico de um protocolo de comunicação que utiliza a abordagem cliente-servidor é o protocolo *request-reply* [9].

Para o protótipo, foi definido um protocolo de comunicação que permite que informações sejam transmitidas entre os clientes e o servidor.

Neste protocolo, existem dois tipo de mensagens. As mensagens do primeiro tipo são representadas por um simples quadro, e, neste quadro, são colocadas constantes que identificam a semântica da mensagem, da seguinte forma:

Figura 42: Quadro contendo a constante que identifica a mensagem.

Um exemplo deste tipo de mensagem é a que utiliza a constante LISTA_SALAS, que retorna a lista de salas disponíveis para o cliente.

Adicionalmente, algumas mensagens podem conter parâmetros, como por exemplo o nome do usuário ou a identificação da sala de jogo. Neste caso, os parâmetros vão logo após as constantes, da seguinte forma:

CONSTANTE	Parâmetro 1	...	Parâmetro N

Figura 43: Quadro que contém constante de mensagem e parâmetros.

Estes parâmetros serão apresentados de acordo com o tipo da mensagem.

A Tabela 1 mostra o que cada constante representa.

Os detalhes sobre como executar as aplicações cliente e servidor estão no item 3.1.1 Funcionamento do Protótipo.

CONSTANTE	PARÂMETROS	SIGNIFICADO
LISTA_SALA	-	retorna a lista de salas disponíveis
ENTRAR_SALA	Apelido do jogador e id da sala	Coloca um jogador numa sala de jogo
SAIR_CLIENTE	-	Mensagem enviada pelo cliente informando sua saída da sala
VENCEDOR_JOGO	-	Identifica o vencedor do jogo
SAIR_SERVIDOR	-	Mensagem enviada para o cliente quando o servidor quer desconecta-lo
ACKATUALIZACAO	-	Mensagem enviada pelo cliente para informar ao servidor uma confirmação de atualização de status de jogo.
PODE_JOGAR	-	Mensagem enviada para o próximo jogador indicando que ele pode fazer sua jogada.
OK	-	Mensagem enviada para o cliente confirmando sua entrada na sala
OKPRIM	-	Mensagem enviada para o

		primeiro jogador da sala confirmando sua entrada na mesma
ERRO	Mensagem de erro	Mensagem enviada ao cliente quando ocorre algum erro no processamento de sua requisição no servidor
SAIR_SALA	Apelido do jogador que saiu da sala	Mensagem enviada para os clientes de um jogo *on-line* quando algum jogador sai da sala.

Tabela 1: Lista de constantes do protocolo de comunicação.

Quando o jogo *on-line* começa, o jogador atual deve realizar a sua jogada. Quando este faz a jogada, a aplicação cliente atualiza o jogo localmente, da mesma forma que é feito quando o jogo é *off-line* (contra o computador). Quando o jogo está atualizado localmente, a aplicação cliente monta um quadro que contém o status do jogo e envia este quadro para o servidor. O servidor, ao receber o quadro, envia o mesmo para todos os outros jogadores da sala. Cada aplicação cliente recebe o quadro que vem do servidor, atualiza localmente o jogo baseando-se nas informações que estão no quadro e enviam uma mensagem especial contendo uma confirmação de atualização (a explicação detalhada sobre esta mensagem está no modelo de falhas). Quando o servidor recebe esta mensagem de atualização de todos os jogadores da sala, ele envia uma mensagem para o próximo jogador indicando que este pode realizar a sua jogada. Abaixo está uma descrição do quadro que contém as informações de status de jogo e o significado de cada campo.

Formato do quadro:

ST	B	N	V	...		N	V
J	N	V	N	V	...	N	V
J	N	V	N	V	...	N	V
...							
J	N	V	N	V	...	N	V
OMM	SJ	MSG					
QC	JA	PA	NE	FJ	FST		

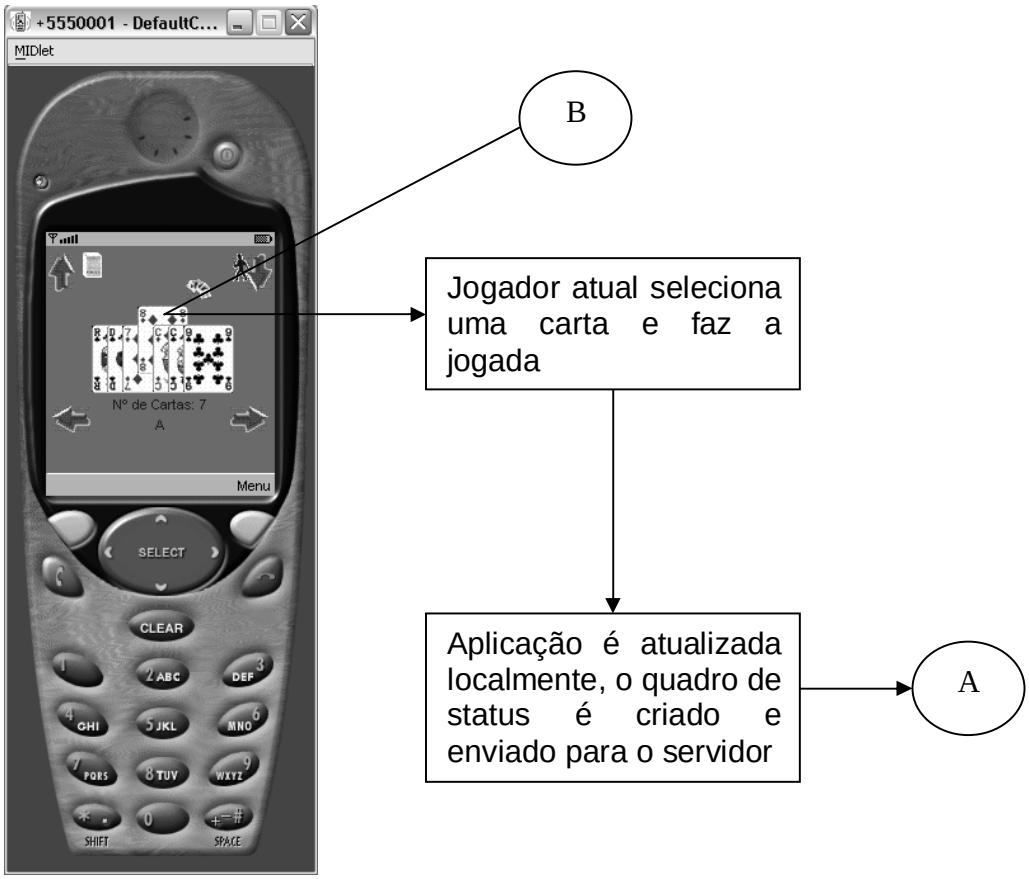
Figura 44: Quadro utilizado na transferência do status do jogo.

A Tabela 2 apresenta o significado de cada campo do quadro de status do jogo:

CAMPO	SIGNIFICADO
ST	Início do quadro de status de jogo
B	Início do conjunto de cartas do baralho
N	Naipe de uma carta
V	Valor de uma carta
J	Início do conjunto de cartas de um jogador
OMM	Indica se o modo de jogo Open Mau-Mau está ativado ou não
SJ	Indica o sentido do jogo
MSG	Mensagem que deve ser exibida para o jogador
QC	Indica quantas cartas devem ser compradas para a jogada atual (depende da carta que estiver no baralho)
JÁ	Índice do jogador atual
PA	Indica se a penalidade relacionada com a carta atual do baralho já foi aplicada ao jogador anterior ou não
NE	Indica o naipe escolhido (utilizado apenas quando a carta jogada é um valete)
FJ	Indica se o jogo deve ser finalizado após a exibição da mensagem
FST	Indica o fim do quadro de status de jogo

Tabela 2: Campos do quadro de status de jogo.

A figura abaixo mostra como ocorre a transferência deste quadro entre as aplicações cliente e servidor, quando um jogador realiza uma jogada num jogo *on-line*:



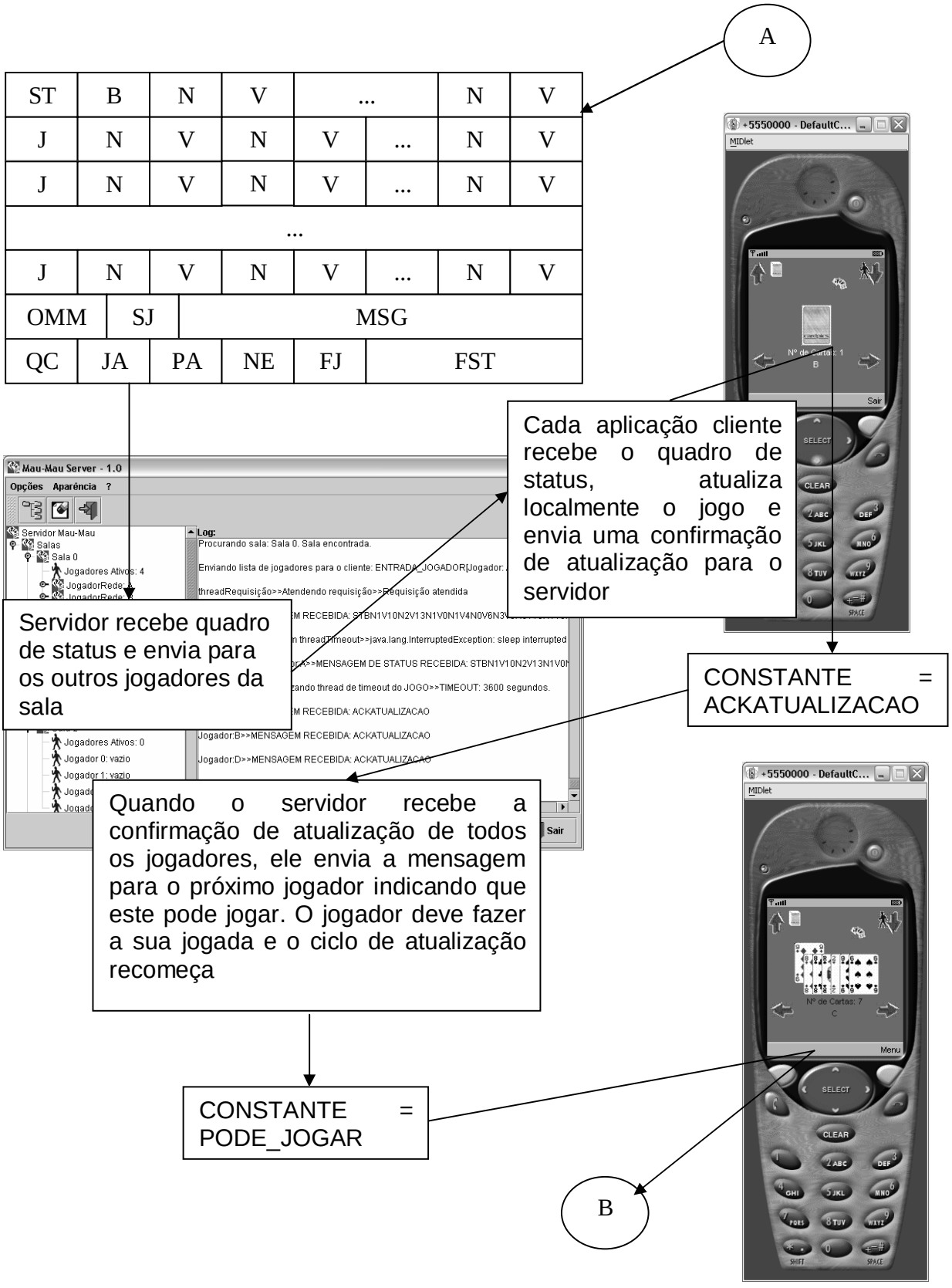


Figura 45: Troca de quadro de atualização de status de jogo.

3.4 Modelo de Falhas

Quando se desenvolve um sistema distribuído, a preocupação com a ocorrência de falhas é maior, já que existem fatores adicionais que contribuem de forma significativa na ocorrência destas falhas, tais como os meios de comunicação envolvidos, a capacidade do servidor em atender grandes volumes de requisições, etc.
Uma das primeiras preocupações que aparecem quando se desenvolve o servidor está relacionada ao tempo de resposta para as requisições do cliente. Se o processamento do servidor fosse sequencial, então um cliente teria que aguardar o processamento da requisição corrente antes de poder fazer a sua requisição, já que existe normalmente um único processador no computador, e este é capaz de realizar apenas uma instrução por vez. Para atenuar este problema, o servidor foi desenvolvido de forma a trabalhar com o conceito de *threads* (linhas de execução). Utilizando *threads*, o mesmo processador é compartilhado por diferentes conjuntos de código. Existem algumas abordagens neste sistema, e estas variam de acordo com o sistema operacional utilizado. O sistema mais comunmente utilizado é conhecido como *time-slice*, onde o sistema operacional fornece um pequeno tempo para cada processo e, após o tempo expirar, o processo corrente é colocado numa fila de espera e outro entra no seu lugar. Variações deste sistema incluem filas com diferentes prioridades, onde o sistema operacional escolhe o processo que será executado baseando-se nestas filas.

A figura abaixo ilustra o mecanismo de *threads* (o exemplo mostra o mecanismo de *time-slice* com apenas uma fila de execução).

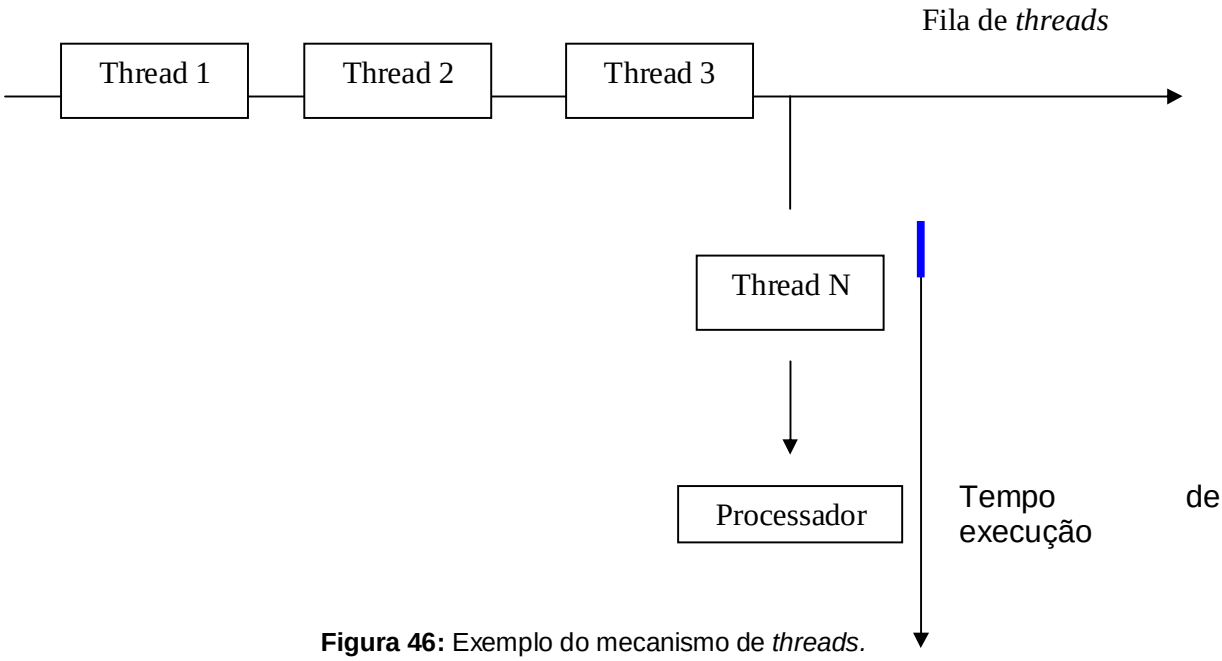


Figura 46: Exemplo do mecanismo de *threads.*

Na figura acima, a *thread* N está sendo executada e quando o tempo de execução para esta *thread* terminar (o tempo passando é representado pela linha azul), esta será colocada de volta na fila e a próxima *thread* será executada, conforme pode-se observar na figura abaixo.

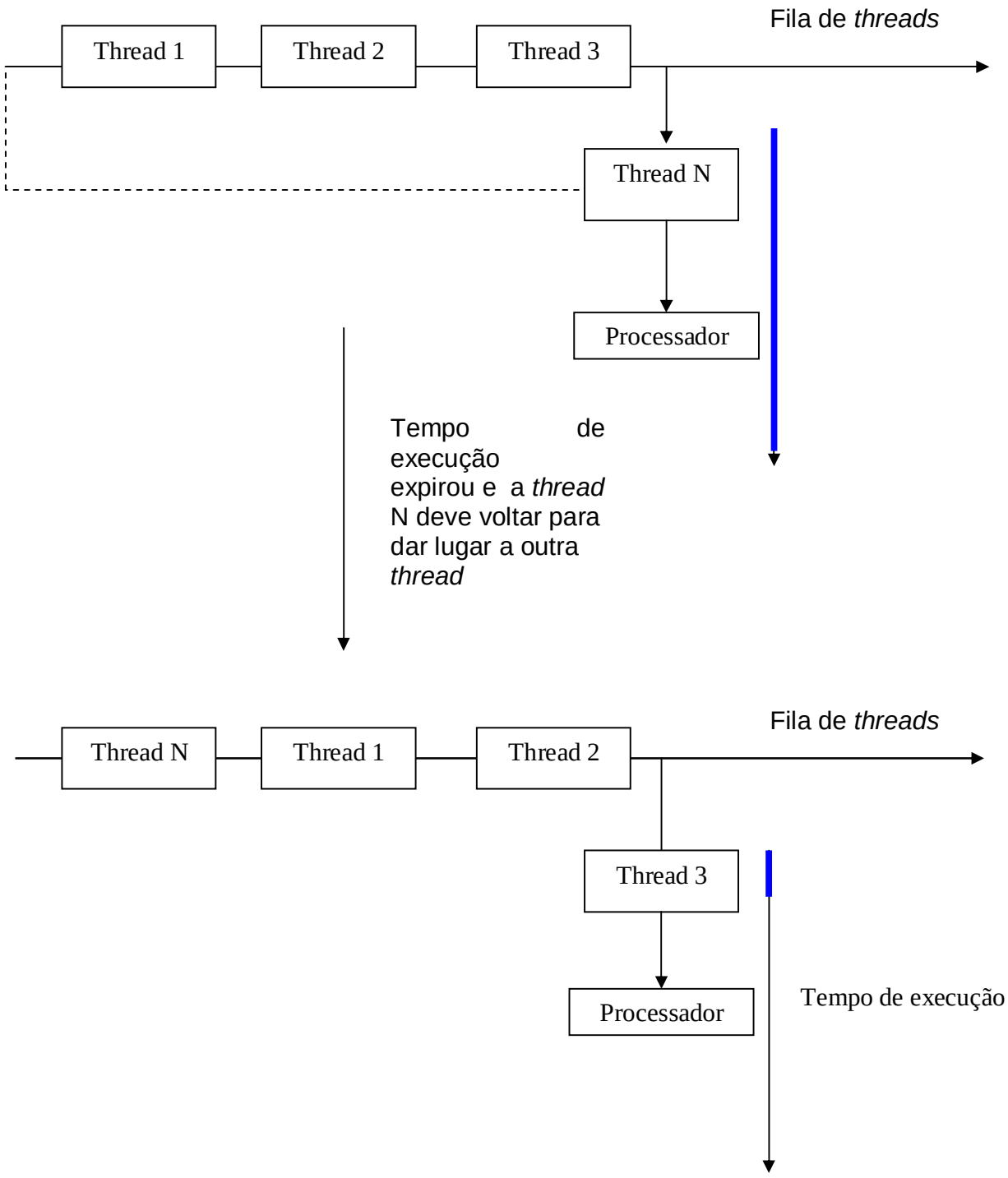


Figura 47: Exemplo de uma fila de *threads.*

Com a utilização de *threads*, o servidor pode atender diversas requisições com um desempenho muito mais satisfatório [9].
Para cada requisição recebida, o servidor cria uma nova *thread*, que vai processar esta requisição.
Com a utilização de *threads*, deve-se ter um cuidado especial quanto a algumas partes do código, já que duas ou mais *threads* executando o mesmo trecho de código podem produzir resultados inesperados.
Considere como exemplo a ação de entrar numa sala de jogo. Existe, para cada sala, um contador que contém o número de usuários da sala. Agora considere dois jogadores tentando entrar na mesma sala. Nesta situação, o contador de usuários pode ficar com um valor errado quando as duas *threads* terminarem. A Figura 48 apresenta um exemplo desta situação:

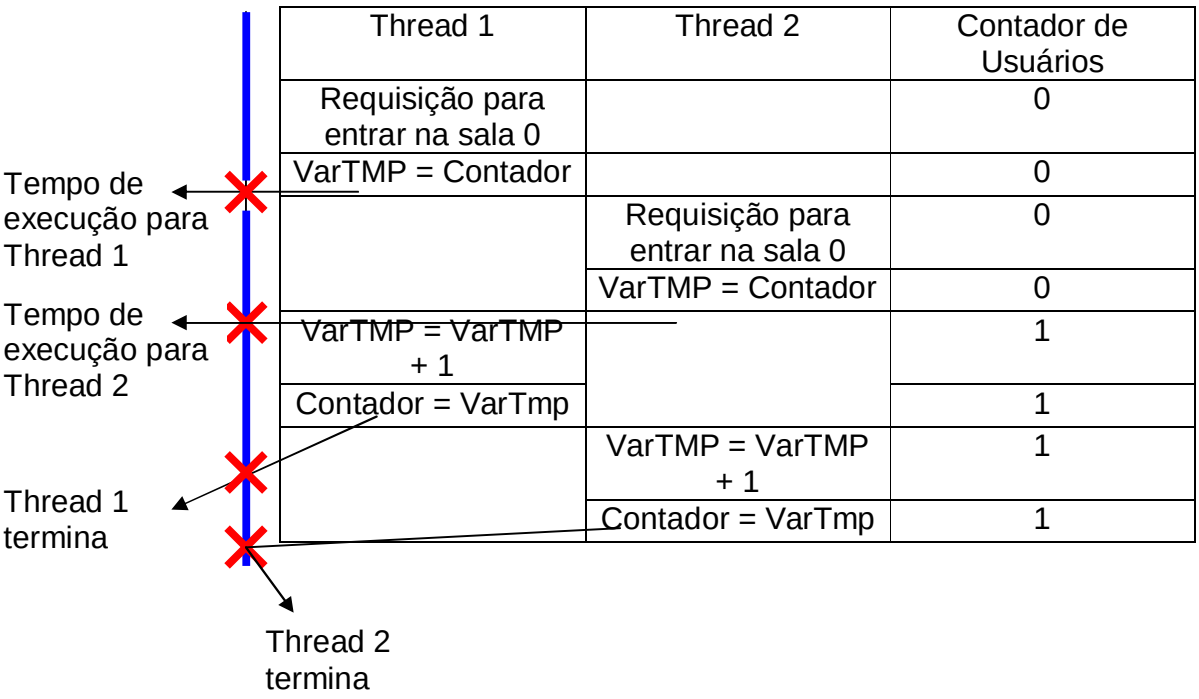


Figura 48: Exemplo de um problema que ocorre na utilização de *threads*.

Observando a figura acima, percebe-se que o valor do contador de usuários deveria ser dois no final da execução das duas *threads*, porém, devido à ordem de execução das *threads* e da falta de sincronização do código, o valor do contador ao final da execução das duas *threads* está errado.
Para resolver isto, algumas operações que podem gerar problemas quando acessadas por mais de uma *thread* devem ser sincronizadas. Uma operação sincronizada pode ser acessada por apenas uma *thread* por vez. Desta forma, uma *thread* que deseja acessar uma operação sincronizada que já esteja sendo utilizada por outra *thread* deve esperar

que esta termine a operação antes de poder utilizá-la. Com a sincronização conseguimos resolver alguns destes problemas de acesso concorrente a algumas operações no servidor.

Quando um jogador entra numa sala de jogo, ele deve esperar que outros jogadores entrem na sala para poder começar o jogo. Quando o segundo jogador entra na sala, o primeiro jogador que entrou recebe uma opção de começar o jogo. Agora imagine que este jogador nunca selecione esta opção e nunca saia da sala de jogo. Então poderia acontecer de o jogo nesta sala nunca começar. Para resolver este tipo de falha, foi criado um mecanismo de timeout de início de jogo. Quando o primeiro jogador entra numa sala, este mecanismo é acionado e, após 15 minutos, verifica se o jogo na sala já começou. Se o jogo ainda não começou após 15 minutos da entrada do primeiro jogador, este e os outros jogadores que estiverem na sala são desconectados e a sala fica novamente disponível para que outros jogadores possam utilizá-la.

Um problema semelhante pode ocorrer após o jogo começar. Se ninguém na sala jogar, o jogo “poderia durar para sempre” e a sala nunca ficaria disponível para novos jogadores. Para resolver este problema, foi criado um mecanismo que de timeout para o fim de um jogo. Após iniciado, o jogo pode durar no máximo 1 hora. Após este tempo, se o jogo não tiver acabado, o servidor cancela o mesmo, desconecta os usuários da sala e deixa esta novamente disponível para utilização.

Quando o servidor é encerrado, os jogadores são avisados e automaticamente desconectados.

Quando algum cliente sai da sala de jogo, este envia uma mensagem para o servidor, que, por sua vez, repassa a mensagem para os outros usuários da sala. Neste caso, o servidor cancela o jogo nesta sala, desconecta os usuários da sala e deixa esta novamente disponível para utilização.

O protótipo sempre foi testado com as aplicações cliente e servidor sendo executadas na mesma máquina. Quando o teste foi realizado utilizando máquinas diferentes e na internet, foi percebido que as diferentes velocidades de conexão com a rede e as diferentes capacidades de processamento dos computadores utilizados no teste interferiam na atualização da aplicação nos clientes. Quando um usuário que possuia uma máquina com maior poder de processamento ou uma conexão de maior velocidade recebia o quadro de status do jogo, o jogo era atualizado localmente de forma muito rápida e o quadro de status representando esta jogada já era enviado ao servidor. Outros clientes que possuiam conexões de menor velocidade ou aparelhos com menor poder de processamento demoravam mais para processar o quadro de status da jogada anterior. Sendo assim, alguns destes clientes recebiam a mensagem de uma jogada sem ter tido tempo de ver o status do jogo da jogada anterior, um comportamento muito ruim para o tipo do jogo do protótipo. Para resolver este problema, foi criado um mecanismo onde existe um contador de atualizações. Quando um quadro de status de jogo chega ao servidor, este contador é resetado. O servidor então envia o quadro para todos os jogadores da sala de jogo. A aplicação cliente recebe o quadro e o processa, exibindo a mensagem apropriada para o jogador. A aplicação cliente correspondente ao próximo jogador fica travada nesta tela de mensagem, enquanto as outras aplicações cliente estão atualizando o jogo. Quando uma aplicação cliente termina de atualizar o jogo localmente, esta envia uma mensagem para o servidor indicando que a atualização local ocorreu com sucesso. O servidor, ao receber esta confirmação, incrementa o contador de atualizações. Quando

o servidor recebe esta mensagem de atualização de todos os jogadores da sala, significa que estes jogadores já atualizaram localmente o jogo e já estão exibindo a tela do próximo jogador. A única exceção é o próximo jogador, que continua travado na tela de mensagem da jogada anterior. O servidor então envia uma mensagem para este jogador indicando que o mesmo pode realizar a jogada, e só então a aplicação cliente mostra a tela de jogo para este jogador. Desta forma, o jogo flui normalmente para todos os jogadores, independente das diferenças de velocidade de comunicação e do poder de processamento dos dispositivos.
Outro problema identificado ocorreu quando um jogador tentava entrar na sala ao mesmo tempo em que a primeira mensagem de status de jogo era transmitida. Se a mensagem de entrada na sala chegasse antes da mensagem de status, ocorria um problema, pois o jogo na sala não estava considerando este novo jogador, já que o mesmo entrou na sala após o início do jogo, representado pela primeira mensagem de status. Para resolver este problema, o servidor, ao receber a mensagem de status, verifica o número de jogadores desta mensagem e compara com o número de jogadores atual. Caso estes números sejam diferentes, o que indica que uma mensagem de entrada chegou antes da mensagem de status, a aplicação cliente do primeiro jogador é notificada, e o processo de início de jogo reinicializado. Este processamento adicional é transparente para os jogadores da sala.

3.5 Modelo de segurança

O foco deste trabalho não é a segurança da troca de informações na rede. Sendo assim, não foi implementado nenhum mecanismo de segurança de troca de mensagens entre os clientes e o servidor.

3.6 Modelagem da aplicação

Como mencionado anteriormente, a aplicação segue o modelo de interação cliente-servidor. A figura abaixo mostra a organização da aplicação no cliente e no servidor.

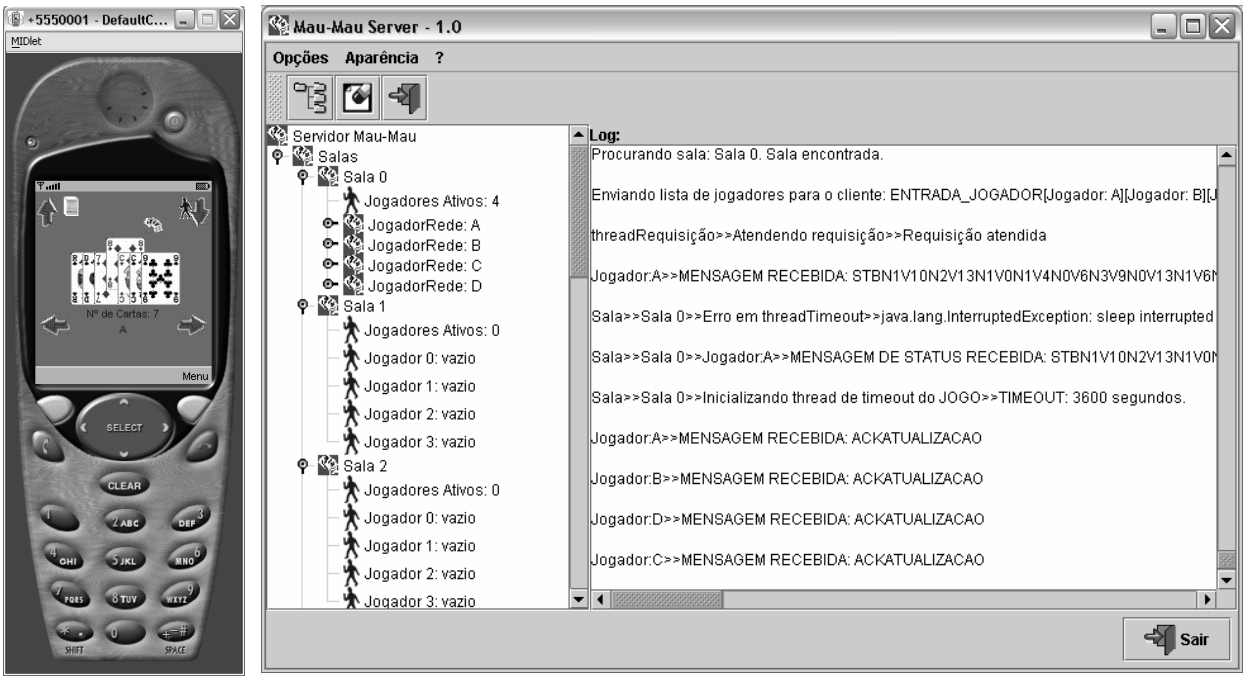


Figura 49: Aplicações Cliente e Servidor.

Como pode-se observar na Figura 49, foram desenvolvidas duas aplicações, que se comunicam através da rede de computadores. A primeira aplicação, representada na figura acima pela aplicação que está sendo executada no telefone celular, é executada no dispositivo móvel. Esta aplicação gerencia o jogo *off-line*, ou seja, o jogo no qual o jogador joga contra o computador e gerencia a atualização do status de jogo, no caso do jogo *on-line*. A segunda aplicação, representada pela tela mostrada na parte direita da figura acima, é o servidor de jogo *on-line*. A função deste programa é gerenciar as salas de jogo e controlar a execução do jogo *on-line* nestas salas.

Abaixo estão os diagramas desenvolvidos durante a fase de modelagem da aplicação:

3.6.1 Modelo de negócio (IDEF0)

Durante a fase inicial de modelagem, foram identificadas três etapas principais no processo de execução de um jogo. A etapa de configuração do jogo representa a fase em que o mesmo é adaptado de acordo com o dispositivo móvel utilizado. Dados como o tamanho da tela, por exemplo, devem ser obtidos nesta fase para que, durante a execução do jogo, os desenhos possam ser criados de acordo com o tamanho da tela do dispositivo.
A próxima fase chama-se execução do jogo. Nesta fase, os comandos do jogador são analisados e o status do jogo é atualizado.
A fase de registro/atualização de dados é opcional. Ela pode ser utilizada, por exemplo, em jogos que possuam algum mecanismo de *ranking.*
O diagrama IDEF0 abaixo mostra as três fases descritas acima:

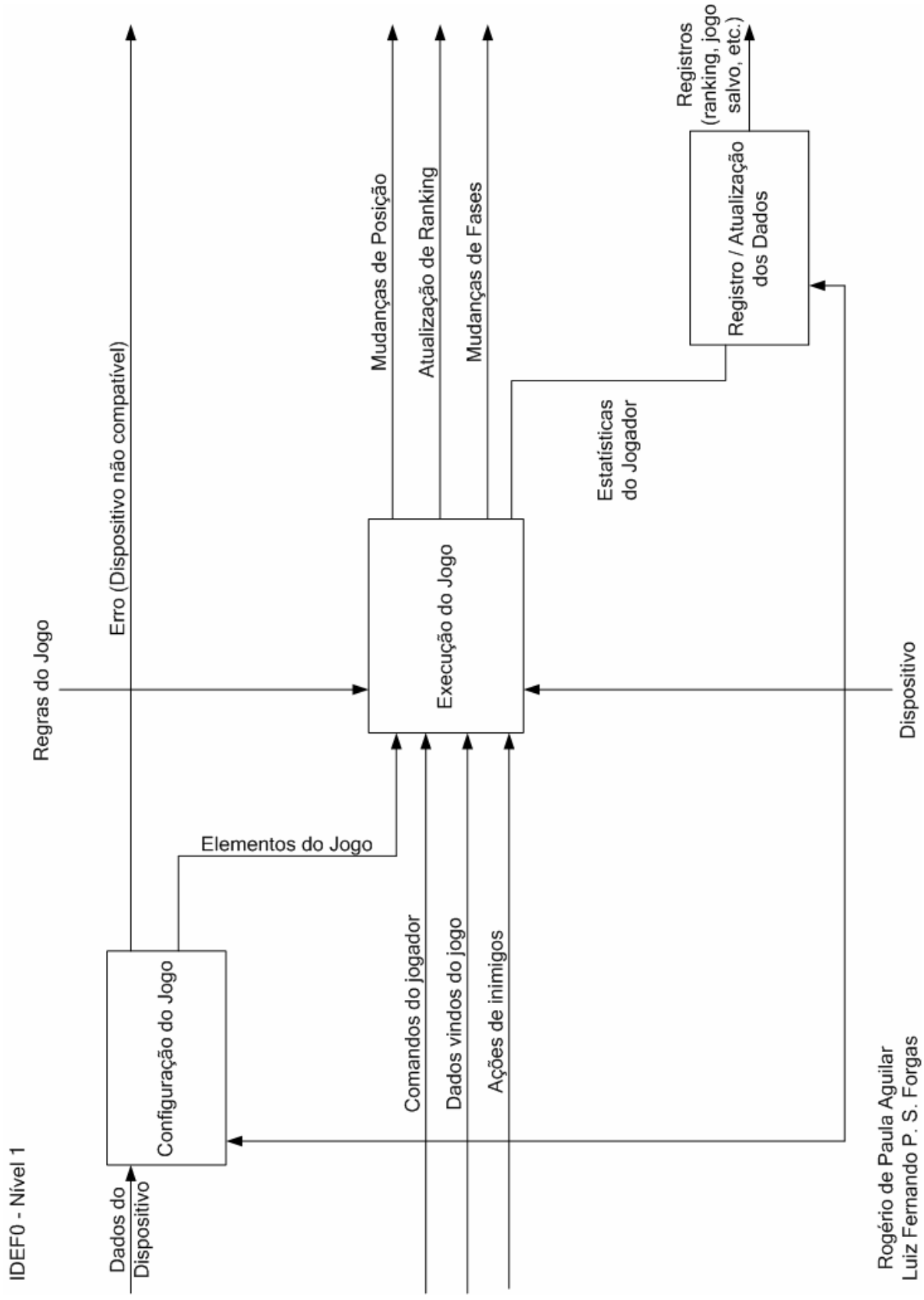


Figura 50: Modelo de Negócio.

3.6.2 Diagrama de classes da aplicação cliente

O diagrama abaixo contém uma visão geral das classes desenvolvidas para a aplicação cliente.

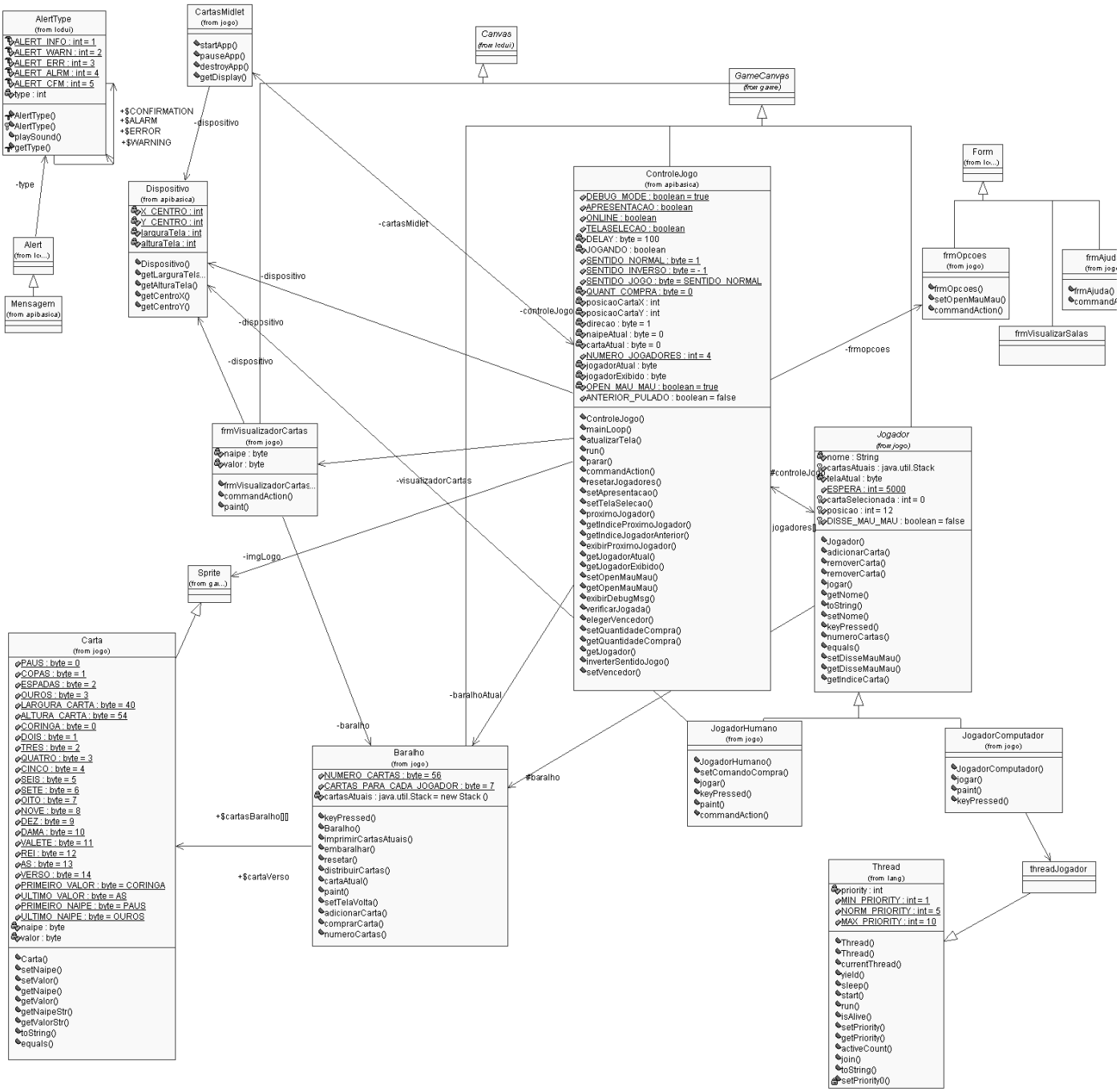


Figura 51: Diagrama de classes da aplicação cliente.

A classes ControleJogo controla o fluxo de execução de um jogo. Quando um jogador seleciona uma carta e joga, esta classe verifica se a jogada é válida. Se for, o status do jogo é atualizado e o controle do jogo é passado ao próximo jogador. No caso do jogo *on-line*, o status do jogo é atualizado localmente, o quadro de status de jogo é criado e enviado ao servidor. O gerenciamento de conexões com o servidor também é feito pela classe ControleJogo. A classe Baralho guarda as informações do baralho no jogo. Esta classe possui operações para embaralhar as cartas e para distribuí-las entre os jogadores. A classe Jogador guarda as informações referentes a um jogador, que pode ser uma pessoa ou o computador. A especialização de acordo com o tipo de jogador é feita nas classes JogadorHumano e JogadorComputador. A classe threadJogador escolhe a carta que deve ser jogada, no caso do jogador ser o computador. A classe Dispositivo guarda algumas informações importantes sobre o dispositivo móvel, como por exemplo a largura e a altura da tela. As outras classes que possuem o prefixo frm, como por exemplo a classes frmListaSalas, implementam a interface gráfica com o usuário.

3.6.3 Diagrama de classes do servidor

O diagrama abaixo contém uma visão geral das classes desenvolvidas para o servidor de jogos.

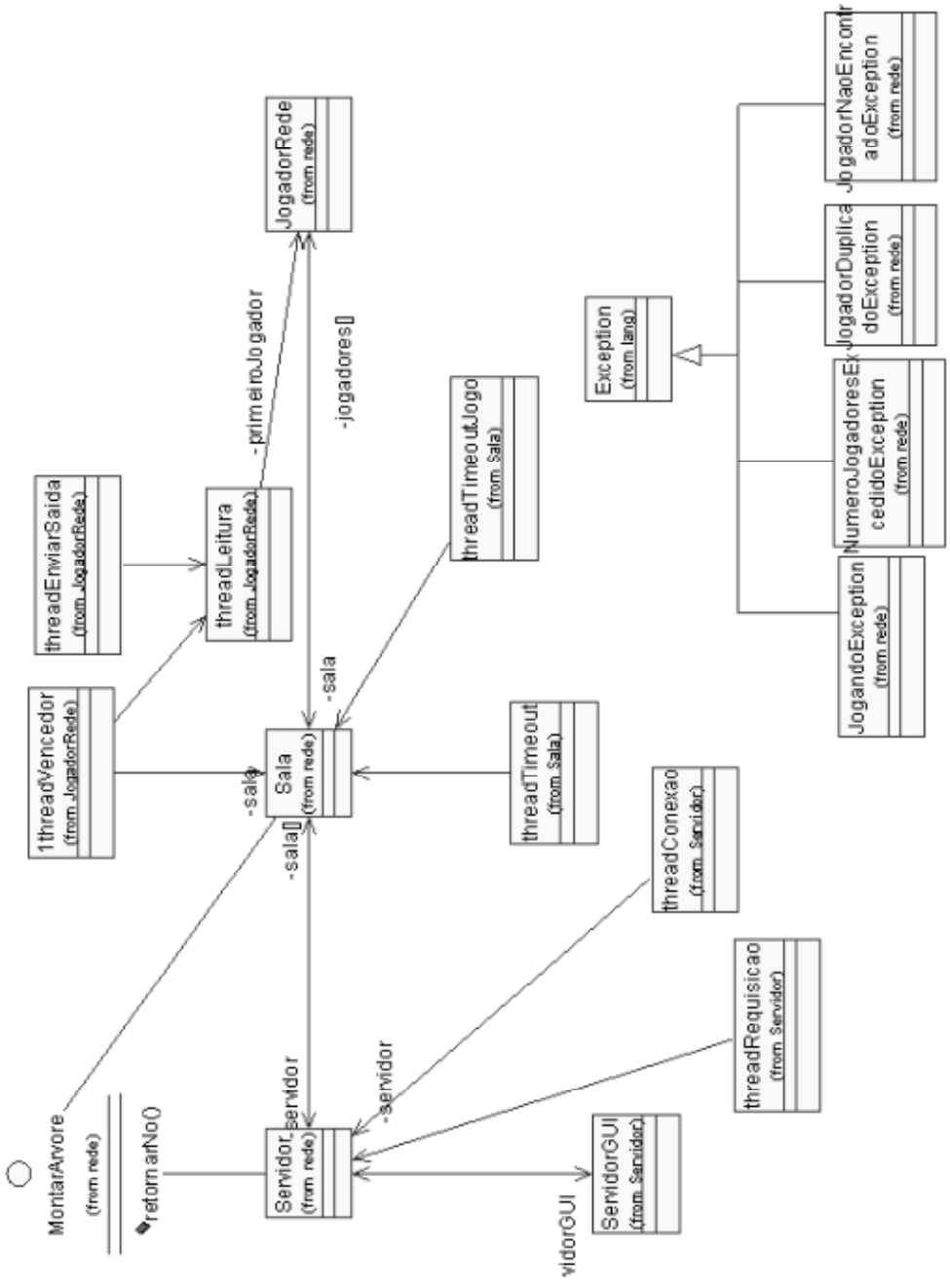


Figura 52: Diagrama de classes do servidor.

A classe Servidor controla o fluxo de trabalho do servidor. Esta classe tem como principal função gerenciar as salas de jogo. A classe ServidorGUI implementa a interface gráfica do servidor. A classe threadConexao é a classe que fica “aguardando” por uma nova requisição de um cliente. Quando esta requisição chega, a classe threadConexao cria uma instância da classe threadRequisicao, que é responsável por processar esta requisição. A classe Sala tem a responsabilidade de gerenciar os jogadores que estão atualmente nela. As classes threadTimeout e threadTimeoutJogo são responsáveis pelos mecanismos de *timeout* descritos no item 3.4. As classes threadLeitura, threadVencedor e threadEnviarSaida são responsáveis pela leitura de comandos que vem das aplicações cliente e pelo envio de comandos para os clientes. A classe JogadorRede guarda as informações referentes aos jogadores de um jogo *on-line.* As classes JogandoException, NumeroJogadoresExcedidoException, JogadorDuplicadoException e JogadorNaoEncontradoException são utilizadas em situações de erro, como por exemplo quando um jogador tenta entrar numa sala utilizando um apelido que já esteja sendo utilizado por outro jogador da sala.

3.6.4 Diagramas de Seqüência

Os diagramas de seqüência mostram a troca de mensagens entre as classes para realização de alguma operação no sistema. Abaixo serão mostrados alguns diagramas:

- **Jogador começa um novo jogo:** representa o início de um novo jogo contra o computador.

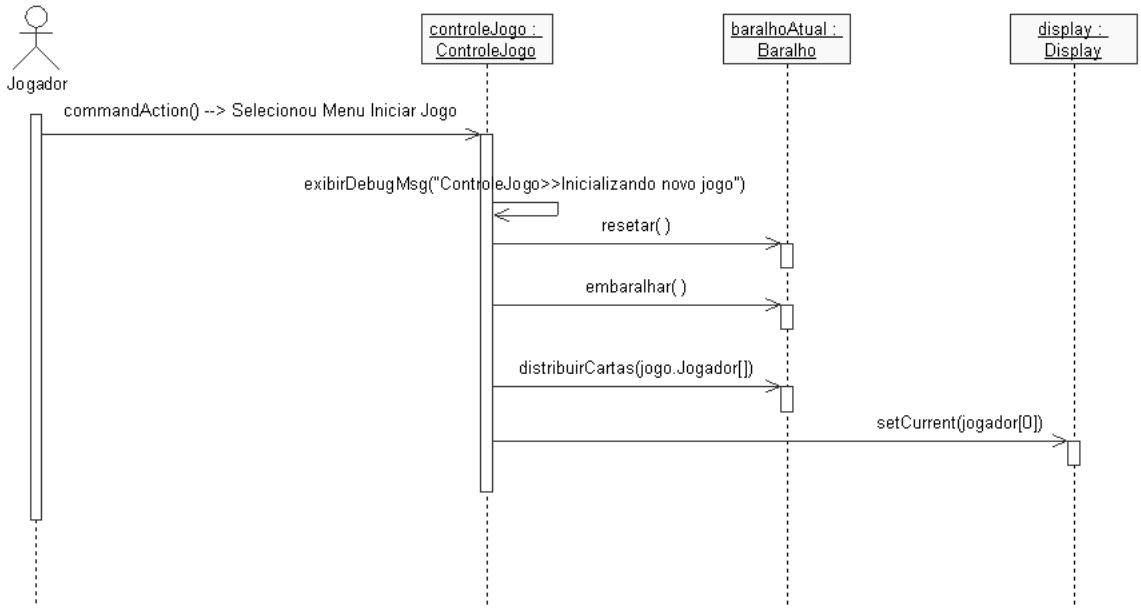


Figura 53: Jogador começa um novo jogo.

- **Jogador humano faz uma jogada válida:** ocorre quando um jogador seleciona uma carta válida e faz uma jogada utilizando esta carta.

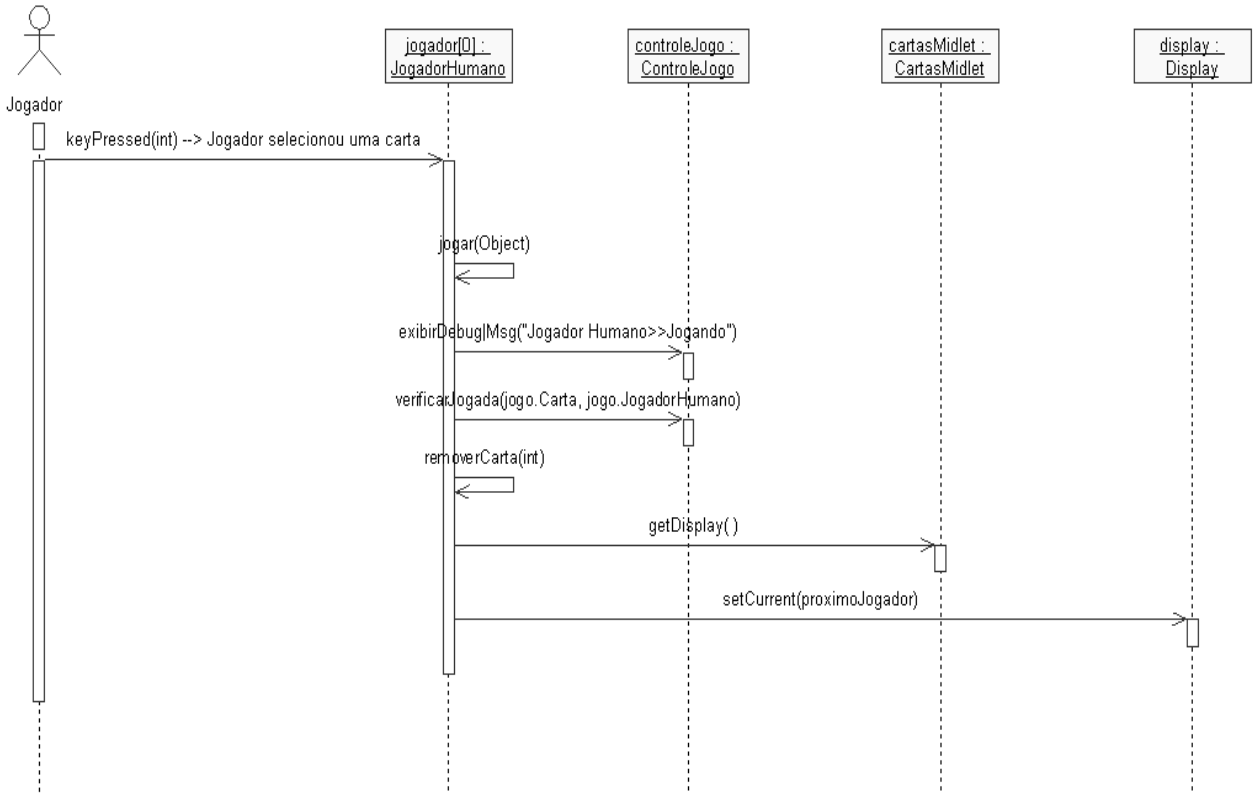


Figura 54: Jogador humano faz uma jogada válida.

- **Jogador humano faz uma jogada inválida:** ocorre quando um jogador seleciona uma carta inválida e tenta utilizar esta carta na sua próxima jogada.

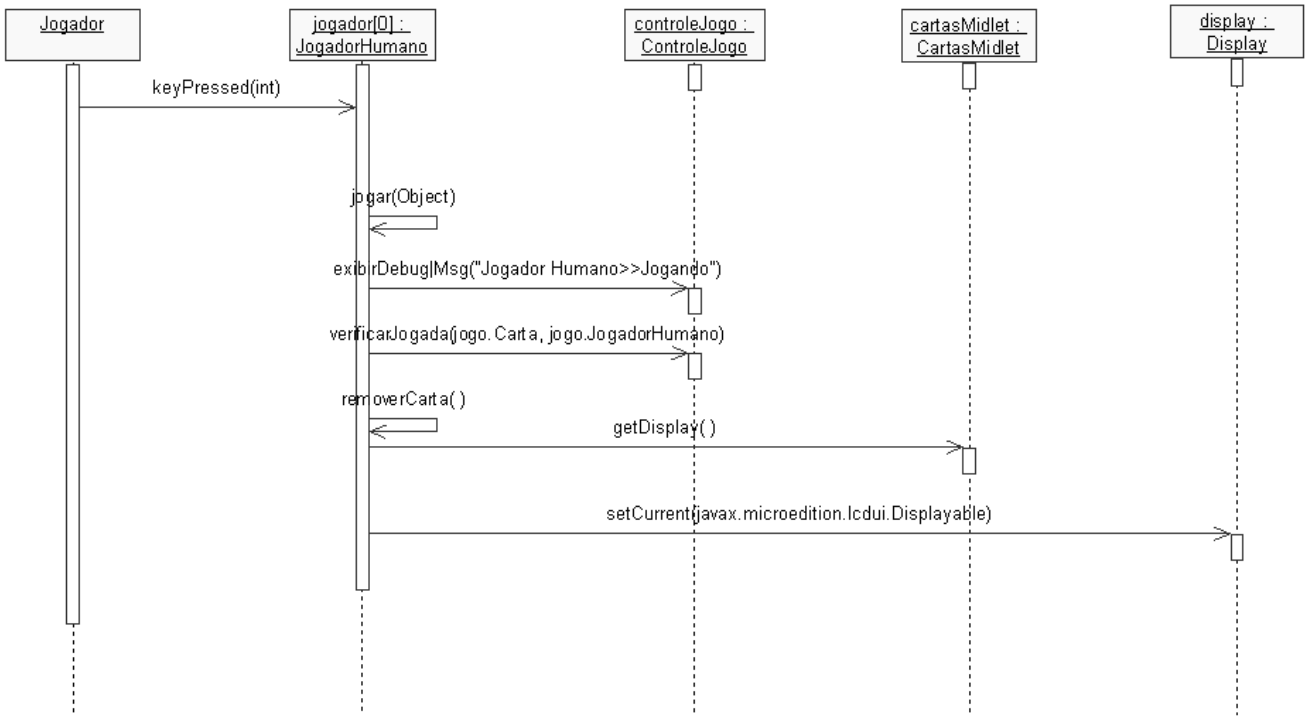


Figura 55: Jogador humano faz uma jogada inválida.

- **Jogador computador faz uma jogada válida:** ocorre quando um jogador controlado pelo computador seleciona uma carta válida e faz uma jogada utilizando esta carta.

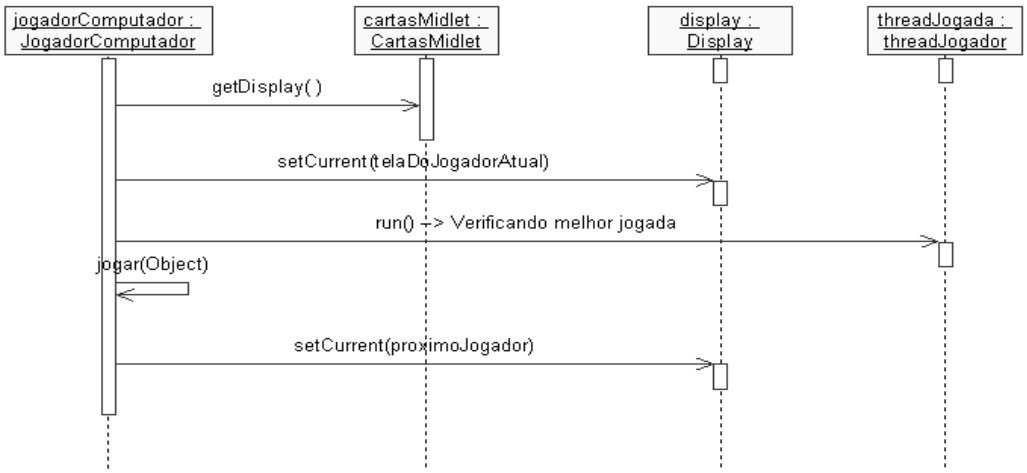


Figura 56: Computador faz uma jogada.

- **Jogador seleciona formulário de opções:** ocorre quando um jogador seleciona a o item Opções no menu principal do jogo.

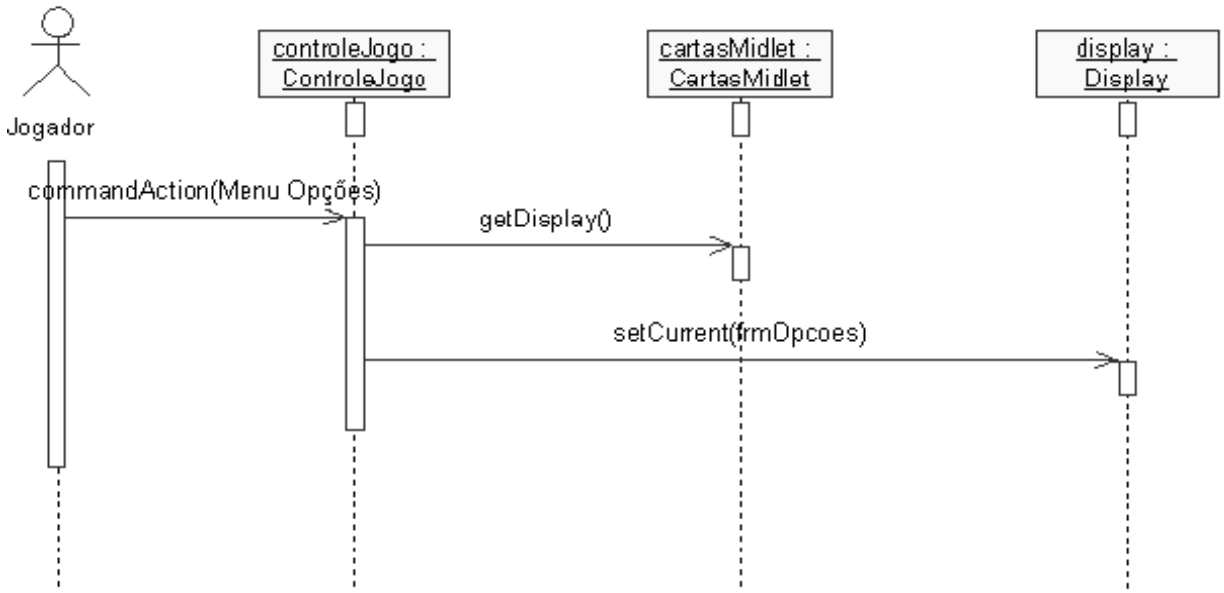


Figura 57: Jogador seleciona formulário de opções.

- **Jogador entra numa sala de jogo *on-line*:** ocorre quando um jogador entra numa das salas de jogo disponibilizadas pelo servidor de jogos.

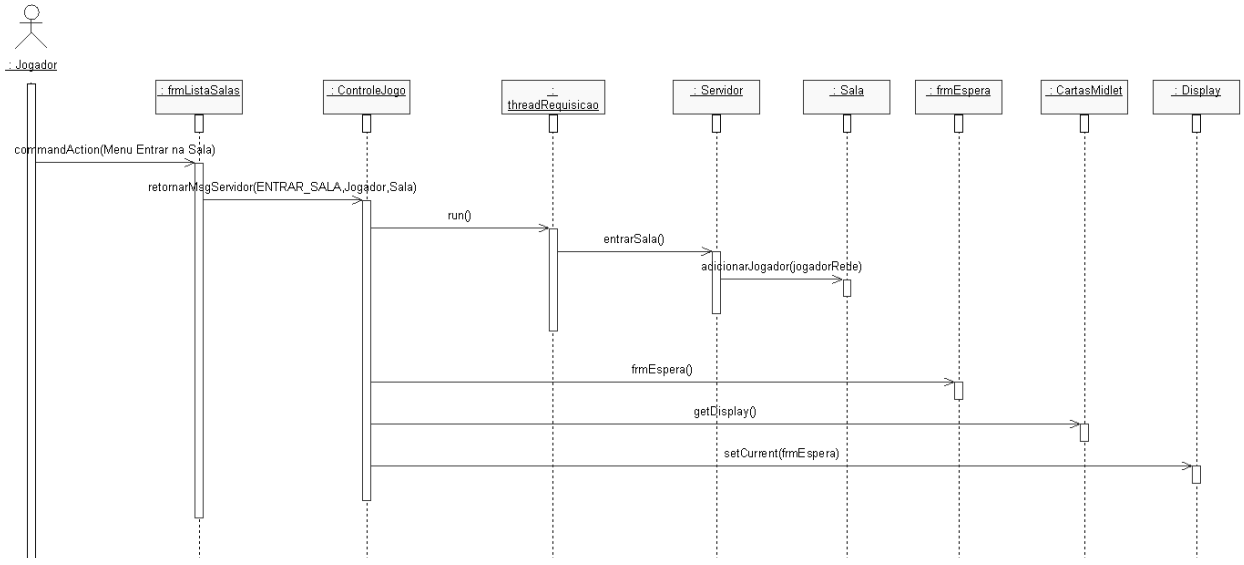


Figura 58: Jogador entra numa sala de jogo *on-line*.

- **Jogador faz uma requisição das salas disponíveis:** ocorre quando um jogador seleciona a opção jogo *on-line* no menu principal do jogo.

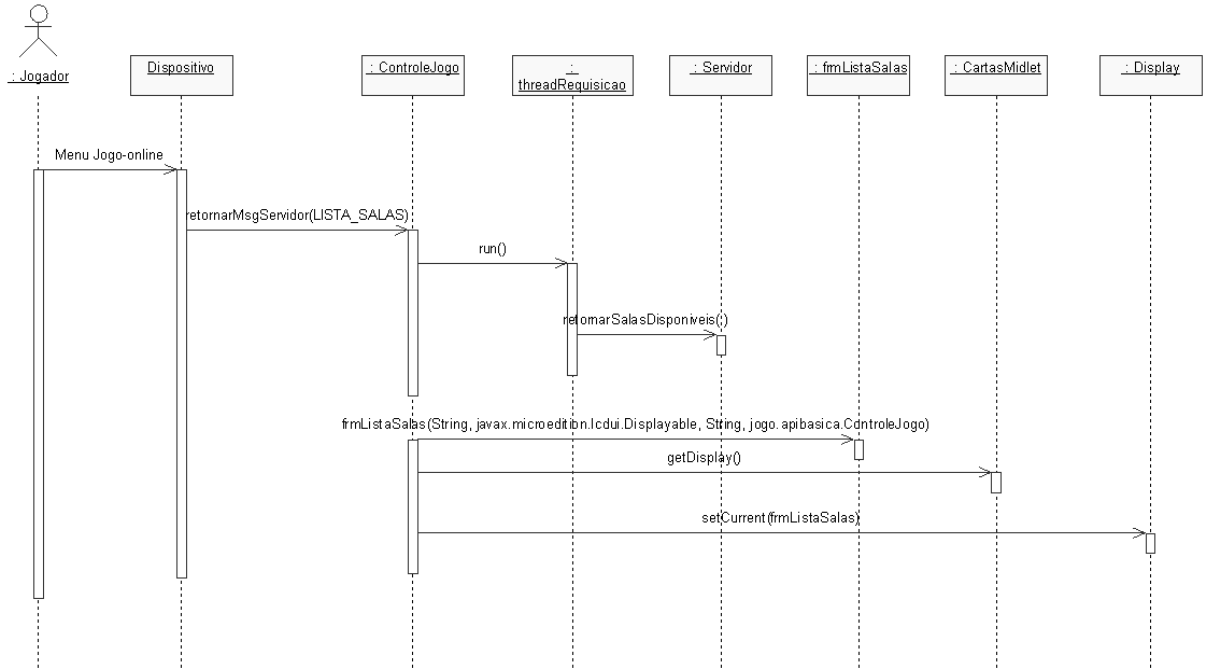


Figura 59: Jogador faz uma requisição das salas de jogo disponíveis no servidor.

- **Aplicação cliente atualiza o status do jogo atual:** ocorre quando a aplicação cliente recebe um quadro de status do servidor e atualiza o jogo localmente utilizando as informações deste quadro.

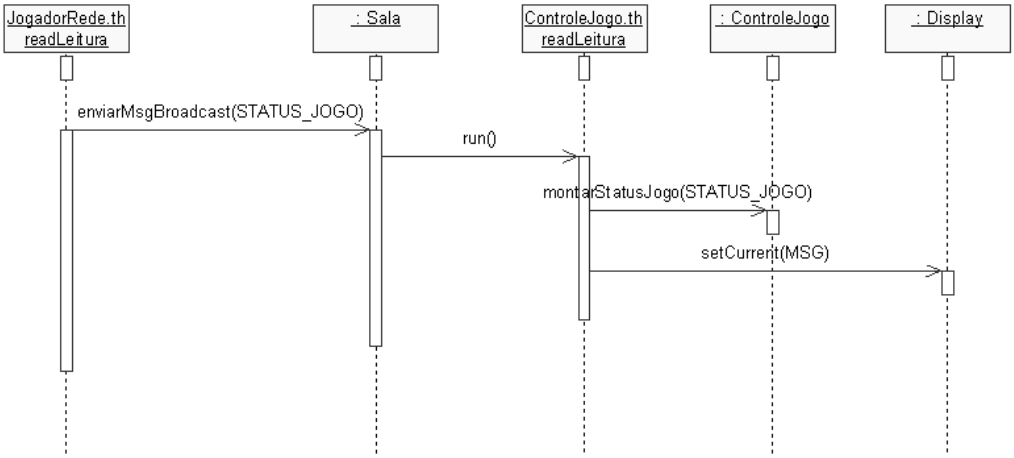


Figura 60: Aplicação cliente atualiza status do jogo atual.

- **Jogador faz uma jogada num jogo *on-line*:** ocorre quando um jogador que está participando de um jogo *on-line* seleciona uma carta e faz a sua jogada utilizando esta carta.

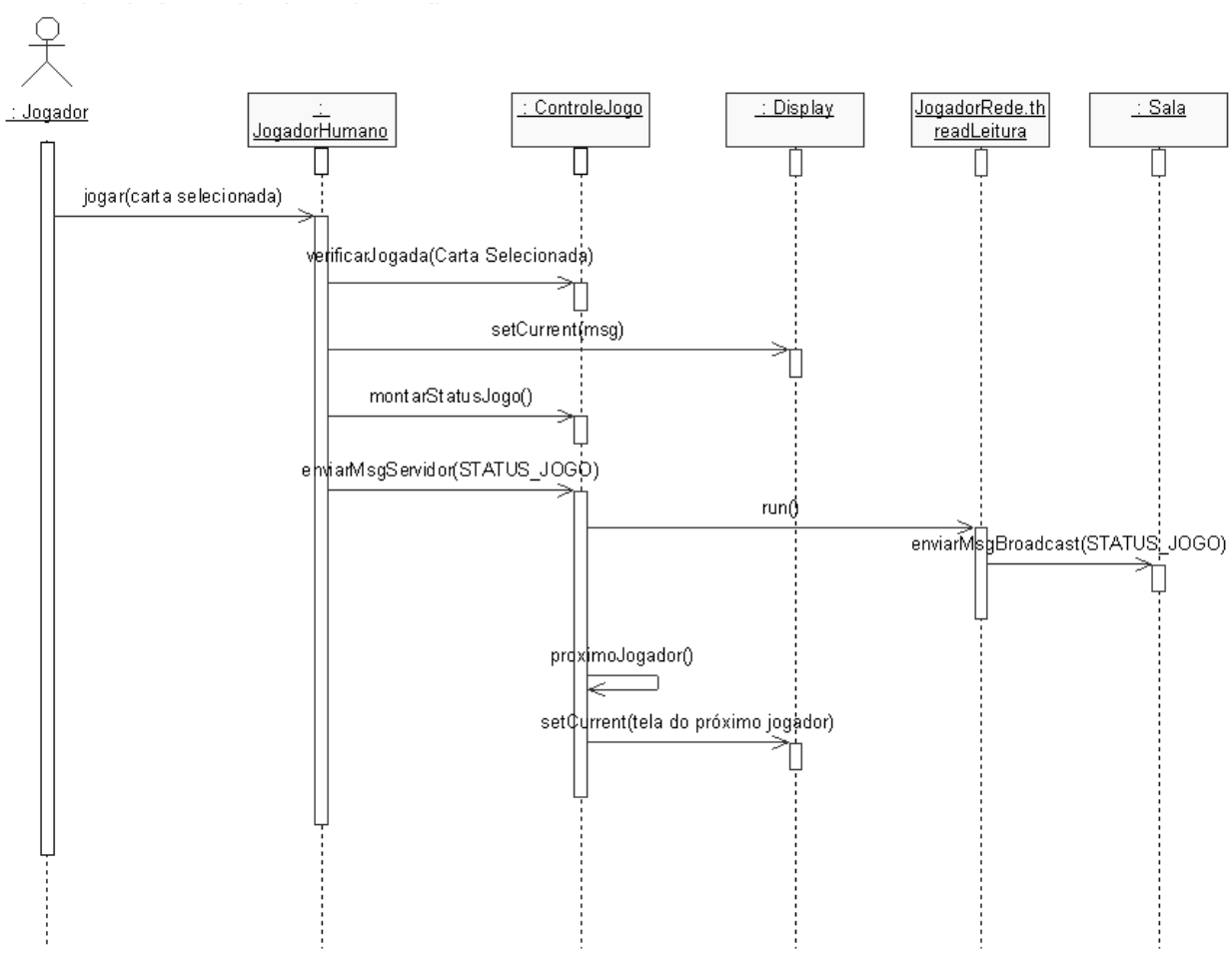


Figura 61: Jogador faz uma jogada num jogo on-line.

Conclusões e trabalhos futuros

Jogos são programas de computador direcionados ao entretenimento das pessoas. Existem algumas bibliotecas de programação e ferramentas que auxiliam no desenvolvimento de jogos. Como um jogo é um produto de *software*, existem metodologias de desenvolvimento que podem ser seguidas, porém estas são um pouco diferentes das metodologias tradicionais de desenvolvimento, já que envolvem, além das tarefas técnicas, tarefas artísticas, como por exemplo o desenvolvimento de modelos de personagens e cenários.

No desenvolvimento de jogos para dispositivos móveis, são utilizadas algumas técnicas comuns ao desenvolvimento tradicional de jogos, como por exemplo a técnica de *back-buffer*. Existem algumas preocupações adicionais que precisam ser tomadas no desenvolvimento do projeto de um jogo para dispositivos móveis, principalmente levando-se em conta as limitações de processamento e memória que os dispositivos apresentam. Entre as dificuldades encontradas, destacam-se a limitação de memória e processamento, as diferentes configurações de tela e teclado, o suporte à transparência de imagens e a pequena velocidade de atualização da tela. O desenvolvimento de um jogo *on-line* adiciona algumas dificuldades, já que o mesmo é um sistema distribuído e, como tal, está sujeito a todas as falhas que podem ocorrer neste tipo de sistema, tais como falhas relacionadas aos meios de comunicação e falhas de coordenação e sincronização durante a execução do jogo. Neste trabalho foram identificadas as principais dificuldades dos dois mundos (o de desenvolvimento de jogos para dispositivos móveis e o de desenvolvimento de jogos *on-line*) e foi aplicada, para cada dificuldade encontrada, uma solução que atendesse aos requisitos do jogo escolhido.

O jogo implementado neste trabalho chama-se mau-mau. Este é um jogo de cartas onde o jogador pode jogar sozinho, contra o computador, ou on-line, de 2 a 4 jogadores.

No desenvolvimento deste jogo, foram enfrentadas as dificuldades relacionadas ao desenvolvimento de jogos para dispositivos móveis mencionadas anteriormente e pode-se destacar alguns problemas de sincronização enfrentados, como por exemplo quando dois usuários do jogo tentam entrar numa sala de jogo ao mesmo tempo. Também foram adicionados alguns mecanismos de *timeout* para controlar o fluxo de execução do jogo e prevenir algumas situações ruins, como por exemplo um jogo iniciado que pode ficar executando por um tempo indefinido.

Foi definido um pequeno protocolo de comunicação, utilizado pelas aplicações cliente e servidor. Neste protocolo foi definido o formato e a semântica das mensagens trocadas

entre as aplicações. Foi desenvolvido um modelo de interação, que define como a aplicação está distribuída e um modelo de falhas, que identifica as principais falhas que podem ocorrer durante a execução do jogo *on-line* e propõe soluções para estas falhas. Melhorias podem ser feitas no protótipo do jogo. Estas melhorias estão relacionadas com a otimização de algumas tarefas realizadas pelo jogo. Uma otimização que pode ser feita é no mecanismo de conexão entre as aplicações clientes e o servidor de jogos *on-line*. Atualmente a conexão permanece aberta durante todo o tempo do jogo. Pode-se implementar algum mecanismo de gerenciamento de sessões, que permita que a conexão seja estabelecida apenas quando a troca de mensagens é feita, e fechada logo após a troca de informações. Isto reduz a utilização dos meios de comunicação e permite que mais clientes compartilhem estes meios. Outra otimização pode ser feita no mecanismo utilizado para troca de informações. Os quadros desenvolvidos para a troca de dados podem ser otimizados, o que reduziria o tempo de transmissão de mensagens entre os clientes e o servidor de jogos e, por fim, pode-se implementar um mecanismo que permita que o computador participe como um jogador de um jogo on-line, já que no protótipo um jogo contra o computador é possível apenas no modo *off-line*.

Referências

[1] **Qualcomm Brew**. Disponível em:
<http://www.qualcomm.com/brew/>.
Acesso em: 03/04/2003.

[2] **Java 2 Platform, Micro Edition (J2ME)**. Disponível em:
<http://java.sun.com/j2me>.
Acesso em: 06/04/2003.

[3]KREITMAIR, Markus; Lozar, Albin; Verhovsek, Roman.
The Java™ 2 Platform, Mi.cro Edition (J2ME™): Game Development for MID Profile.
Disponível em:
<http://servlet.java.sun.com/javaone/conf/sessions/779-4-0/2001-sf2001.jsp>.
Acesso em: 25/05/2003.

[4] MATTHEWS, Marcus. **Developing Action-based Mobile Games**. Disponível em:
<http://www.gamasutra.com/resource_guide/20021125/matthews_01.htm>.
Acesso em: 03/04/2003.

[5] FOX, David. **Creating Games using J2ME**. Disponível em:
<http://www.gamasutra.com/resource_guide/20010917/fox_pfv.htm>.
Acesso em: 04/05/2003.

[6] FOX, David; Verhovsek, Roman. **Micro Java Game Development**, USA,
Pearson Education, 2002.

[7] PRICE, David; Salomaa, Jyri. **Multi-User MIDP Game Design** , Helsinki,
Nokia Research Center, 2002.

[8] Forum Nokia, **MIDP 2.0: Working with Pixels and drawRGB() - Version 1.0**.
Disponível em:
<http://www.forum.nokia.com/main/1,,040,00.html?fsrParam=33/main.html&fileID=3233>
Acesso em : 10/05/2003

[9] COULOURIS, George; Dollimore, Jean; Kindberg, Tim.
Distributed Systems – Concepts and Design - Third Edition, USA,
Pearson Education, 2001

[10] REIS, Ademar de Souza; Nassu, Bogdan T; Jonack, Marco Antonio.
Um Estudo Sobre os Processos de Desenvolvimento de Jogos Eletrônicos.
Disponível em:
<http://planeta.terra.com.br/informatica/ademar.org/textos/proc_desenv_games/proc_desenv_games.pdf>
Acesso em: 15/11/2003.

[11] SOBRINHO, Marcionílio Barbosa. **Tutorial de OpenGL**. Disponível em:
<http://www.ingleza.com.br/opengl/1.html>
Acesso em: 10/11/2003.

[12] **3D Game Studio/A6.** Disponível em:
<http://www.conitec.net/a4info.htm>
Acesso em: 10/11/2003.

[13] **DarkBASIC.** Disponível em:
<http://darkbasic.thegamecreators.com/>
Acesso em: 12/11/2003.

[14] Fórum Nokia, **Introduction to Mobile Game Development Version 1.0.**
Disponível em:
<http://nds1.forum.nokia.com/nnds/ForumDownloadServlet?id=2768&name=Mobile%5Game%5FIntro%5Fv1%5F1%2Epdf>
Acesso em: 12/11/2003.

[15] **Maya Design, Inc**. Disponível em:
<http://www.maya.com>
Acesso em: 15/11/2003.

[16] **Half-Life**. Disponível em:
<http://www.half-life.com>
Acesso em: 16/11/2003.

[17] **DirectX Home Page**. Disponível em:
<http://www.microsoft.com/directx>
Acesso em: 16/11/2003.

[18] **Flight Simulator 2004: A Century of Flight**. Disponível em:
<http://www.microsoft.com/games/PC/flightsimulator.aspx>
Acesso em: 16/11/2003.

[19] WGR Game Directory:**Metal Slug Mobile.** Disponível em:
<http://wgamer.com/gamedir/game-2665>
Acesso em: 16/11/2003.

[20] **Nokia N-Gage**. Disponível em:
<http://www.n-gage.com>
Acesso em: 16/11/2003.

[21] JULL, Jesper. **A Class Between Game and Narrative – A thesis on computer games and interactive fiction**. Tese – Institute of Nordic Language and Literature, University of Copenhagen, 1999

[22] **Playstation.com – Global Splash**.
Disponível em:
<http://www.playstation2.com>
Acesso em: 17/11/2003

Apêndice A
Documentação da API

Este apêndice apresenta a documentação das classes desenvolvidas para o protótipo do jogo Mau-Mau (O conteúdo deste apêndice também pode ser consultado no diretório doc\api do cd-rom).

Hierarchy For All Packages

Package Hierarchies:

jogo, jogo.apibasica, jogo.rede

Class Hierarchy

- class java.lang.Object
  - class javax.microedition.lcdui.Displayable
    - class javax.microedition.lcdui.Canvas
      - class jogo.**frmVisualizadorCartas** (implements javax.microedition.lcdui.CommandListener)
      - class javax.microedition.lcdui.game.GameCanvas
        - class jogo.**Baralho**
        - class jogo.apibasica.**ControleJogo** (implements javax.microedition.lcdui.CommandListener, java.lang.Runnable)
        - class jogo.**Jogador** (implements javax.microedition.lcdui.CommandListener)
          - class jogo.**JogadorComputador**
          - class jogo.**JogadorHumano** (implements javax.microedition.lcdui.CommandListener)
    - class javax.microedition.lcdui.Screen
      - class javax.microedition.lcdui.Alert
        - class jogo.apibasica.**ControleJogo.Mensagem**
      - class javax.microedition.lcdui.Form
        - class jogo.**frmAjuda** (implements javax.microedition.lcdui.CommandListener)
        - class jogo.**frmEscolhaNaipe** (implements javax.microedition.lcdui.CommandListener)
        - class jogo.**frmEspera**

- class jogo.**frmListaSalas** (implements javax.microedition.lcdui.CommandListener)
- class jogo.**frmOpcoes** (implements javax.microedition.lcdui.CommandListener)

- class jogo.apibasica.**Dispositivo**
- class jogo.rede.**JogadorRede** (implements java.io.Serializable)
- class javax.microedition.lcdui.game.Layer
  - class javax.microedition.lcdui.game.Sprite
    - class jogo.**Carta**
- class javax.microedition.midlet.MIDlet
  - class jogo.**CartasMidlet**
- class jogo.rede.**Sala** (implements jogo.rede.MontarArvore)
- class jogo.rede.**Servidor** (implements jogo.rede.MontarArvore)

Interface Hierarchy

- interface jogo.rede.**MontarArvore**

Packages

jogo	
jogo.apibasica	
jogo.rede	

Package jogo

Class Summary	
Baralho	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.Baralho Responsabilidades: Representa o baralho do jogo
Carta	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.Carta Responsabilidades: Representa uma carta do baralho
CartasMidlet	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.CartasMidlet Responsabilidades: Classe principal, que inicializa os objetos de controle do jogo
frmAjuda	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para

	dispositivos móveis Classe: jogo.frmAjuda Responsabilidades: Exibir a ajuda para o usuário
frmEscolhaNaipe	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.frmfrmEscolhaNaipe Responsabilidades: permite que o usuário escolha o naipe da próxima carta (Valete(*))
frmEspera	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.frmEspera Responsabilidades: Exibir uma tela para o usuário enquanto este aguarda o término de algum processo do jogo
frmListaSalas	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.frmListaSalas Responsabilidades: Exibe as salas para o usuário e um campo para digitar o apelido, quando o jogador estiver começando um jogo on-line
frmOpcoes	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.frmOpcoes Responsabilidades: Exibir opções para o usuário, entre elas opção de habilitar/desabilitar o modo open mau mau, opção de habilitar/desabilitar a visualização de informações sobre a carta selecionada.
frmVisualizadorCartas	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.frmAjuda Responsabilidades: Exibir as cartas do baralho
Jogador	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.Jogador Responsabilidades: Guarda as informações e ações que são comuns aos jogadores, independente se estes são jogadores humanos ou o computador
JogadorComputador	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.JogadorComputador Responsabilidades: Guarda as informações sobre um jogador controlado pelo computador e realiza a a jogada do computador
JogadorHumano	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.JogadorHumano Responsabilidades:

	Guarda as informações sobre o jogador que está utilizando o dispositivo móvel

jogo

Class Baralho

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Canvas
              |
              +--javax.microedition.lcdui.game.GameCanvas
                    |
                    +--jogo.Baralho
```

public class **Baralho**
extends javax.microedition.lcdui.game.GameCanvas

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.Baralho

Responsabilidades: Representa o baralho do jogo

Version:
1.0
Author:
Rogério de Paula Aguilar

Field Summary

static byte	**CARTAS_PARA_CADA_JOGADOR** Número de cartas que é distribuído para cada jogador
static jogo.Carta[][]	**cartasBaralho** Cartas do baralho
static jogo.Carta	**cartaVerso** Carta com o verso de uma carta
static jogo.Carta	**cartaVersoMiniatura** Carta contendo o verso em miniatura
protected byte	**deslocamentoSetaBaixo** Variável auxiliar para o desenho das setas de navegação
static byte	**NUMERO_CARTAS** Número de cartas do baralho

Fields inherited from class javax.microedition.lcdui.game.GameCanvas
DOWN_PRESSED, FIRE_PRESSED, GAME_A_PRESSED, GAME_B_PRESSED, GAME_C_PRESSED, GAME_D_PRESSED, LEFT_PRESSED, RIGHT_PRESSED, UP_PRESSED

Fields inherited from class javax.microedition.lcdui.Canvas
DOWN, FIRE, GAME_A, GAME_B, GAME_C, GAME_D, KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND, KEY_STAR, LEFT, RIGHT, UP

Constructor Summary

Baralho() Construtor padrão

Method Summary

void	**adicionarCarta**(jogo.Carta carta) Adiciona uma carta ao baralho
jogo.Carta	**carta**(int indice) Retorna uma carta do baralho pelo seu índice

jogo.Carta	**cartaAtual**() Retorna a carta no topo do baralho
jogo.Carta	**comprarCarta**() Retorna uma carta para compra
void	**distribuirCartas**(jogo.Jogador[] jogadores) Distribui as cartas para os jogadores
void	**embaralhar**() Embaralha as cartas atuais
void	**imprimirCartasAtuais**() Imprime as cartas atuais
void	**keyPressed**(int keyCode) Método invocado quando o jogador pressiona uma tecla na tela do baralho
void	**keyReleased**(int keyCode) Método invocado quando o jogador solta uma tecla na tela do baralho
int	**numeroCartas**() Retorna a quantidade de cartas que o baralho possui atualmente
void	**paint**(javax.microedition.lcdui.Graphics g) Desenha a interface do baralho
void	**resetar**() Reseta o baralho
void	**setTelaVolta**(javax.microedition.lcdui.Displayable d) Muda a tela de volta
void	**zerar**() Deixa o baralho sem nenhuma carta

Methods inherited from class javax.microedition.lcdui.game.GameCanvas
flushGraphics, flushGraphics, getGraphics, getKeyStates

Methods inherited from class javax.microedition.lcdui.Canvas
getGameAction, getKeyCode, getKeyName, hasPointerEvents, hasPointerMotionEvents, hasRepeatEvents, hideNotify, isDoubleBuffered, keyRepeated, pointerDragged, pointerPressed, pointerReleased, repaint, repaint, serviceRepaints, setFullScreenMode, showNotify, sizeChanged

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

cartasBaralho

```
public static final jogo.Carta[][] cartasBaralho
```

Cartas do baralho

---

cartaVerso

```
public static jogo.Carta cartaVerso
```

Carta com o verso de uma carta

---

cartaVersoMiniatura

```
public static jogo.Carta cartaVersoMiniatura
```

Carta contendo o verso em miniatura

NUMERO_CARTAS

```
public static final byte NUMERO_CARTAS
```

Número de cartas do baralho

See Also:

Constant Field Values

CARTAS_PARA_CADA_JOGADOR

```
public static final byte CARTAS_PARA_CADA_JOGADOR
```

Número de cartas que é distribuído para cada jogador

See Also:

Constant Field Values

---

deslocamentoSetaBaixo

```
protected byte deslocamentoSetaBaixo
```

Variável auxiliar para o desenho das setas de navegação

Constructor Detail

Baralho

```
public Baralho()
```

Construtor padrão

Method Detail

keyPressed

```
public void keyPressed(int keyCode)
```

Método invocado quando o jogador pressiona uma tecla na tela do baralho

Overrides:

keyPressed in class javax.microedition.lcdui.Canvas

---

keyReleased

```
public void keyReleased(int keyCode)
```

Método invocado quando o jogador solta uma tecla na tela do baralho

Overrides:

keyReleased in class javax.microedition.lcdui.Canvas

---

imprimirCartasAtuais

```
public void imprimirCartasAtuais()
```

Imprime as cartas atuais

---

embaralhar

```
public void embaralhar()
```

Embaralha as cartas atuais

---

resetar

```
public void resetar()
```

Reseta o baralho

---

distribuirCartas

```
public void distribuirCartas(jogo.Jogador[] jogadores)
```

Distribui as cartas para os jogadores

---

cartaAtual

```
public jogo.Carta cartaAtual()
```

Retorna a carta no topo do baralho

---

paint

```
public void paint(javax.microedition.lcdui.Graphics g)
```

Desenha a interface do baralho

Overrides:

paint in class javax.microedition.lcdui.game.GameCanvas

---

setTelaVolta

```
public void setTelaVolta(javax.microedition.lcdui.Displayable d)
```

Muda a tela de volta

---

adicionarCarta

```
public void adicionarCarta(jogo.Carta carta)
```

Adiciona uma carta ao baralho

---

comprarCarta

```
public jogo.Carta comprarCarta()
                      throws java.util.EmptyStackException
```

Retorna uma carta para compra
`java.util.EmptyStackException`

---

numeroCartas

`public int **numeroCartas**()`

Retorna a quantidade de cartas que o baralho possui atualmente

---

carta

`public jogo.Carta **carta**(int indice)`

Retorna uma carta do baralho pelo seu índice

---

zerar

`public void **zerar**()`

Deixa o baralho sem nenhuma carta

---

jogo

Class Carta

```
java.lang.Object
  |
  +--javax.microedition.lcdui.game.Layer
        |
        +--javax.microedition.lcdui.game.Sprite
              |
              +--jogo.Carta
```

public class **Carta**
extends javax.microedition.lcdui.game.Sprite

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.Carta

Responsabilidades: Representa uma carta do baralho

Version:
1.0

Author:
Rogério de Paula Aguilar

Field Summary

static byte	**ALTURA_CARTA** Altura da imagem da carta
static byte	**AS**

	Valores das cartas
`static byte`	**`CINCO`** Valores das cartas
`static byte`	**`COPAS`** Naipes
`static byte`	**`CORINGA`** Valores das cartas
`static byte`	**`DAMA`** Valores das cartas
`static byte`	**`DEZ`** Valores das cartas
`static byte`	**`DOIS`** Valores das cartas
`static byte`	**`ESPADAS`** Naipes
`static byte`	**`LARGURA_CARTA`** Largura da imagem da carta
`static byte`	**`NOVE`** Valores das cartas
`static byte`	**`OITO`** Valores das cartas
`static byte`	**`OUROS`** Naipes
`static byte`	**`PAUS`** Naipes
`static byte`	**`PRIMEIRO_NAIPE`**
`static byte`	**`PRIMEIRO_VALOR`** Variáveis utilizadas em loops
`static byte`	**`QUATRO`** Valores das cartas
`static byte`	**`REI`** Valores das cartas
`static byte`	**`SEIS`** Valores das cartas
`static byte`	**`SETE`** Valores das cartas
`static byte`	**`TRES`** Valores das cartas
`static byte`	**`ULTIMO_NAIPE`**

static byte	**ULTIMO_VALOR** Variáveis utilizadas em loops
static byte	**VALETE** Valores das cartas
static byte	**VERSO** Valores das cartas

Fields inherited from class javax.microedition.lcdui.game.Sprite
TRANS_MIRROR, TRANS_MIRROR_ROT180, TRANS_MIRROR_ROT270, TRANS_MIRROR_ROT90, TRANS_NONE, TRANS_ROT180, TRANS_ROT270, TRANS_ROT90

Constructor Summary

Carta(javax.microedition.lcdui.Image imagem, byte naipe, byte valor) Construtor

Method Summary

boolean	**equals**(java.lang.Object obj) Verifica a igualdade entre duas instâncias da classe
byte	**getNaipe**() Retorna o naipe da carta
java.lang.String	**getNaipeStr**() Retorna uma representação do naipe da carta num objeto String
byte	**getValor**() Retorna o valor da carta
java.lang.String	**getValorStr**() Retorna uma representação do valor da carta num objeto String
void	**setNaipe**(byte naipe) Modifica o naipe da carta
void	**setValor**(byte valor) Modifica o valor da carta
java.lang.String	**toString**() Retorna uma representação da carta num objeto String

Methods inherited from class javax.microedition.lcdui.game.Sprite
collidesWith, collidesWith, collidesWith, defineCollisionRectangle, defineReferencePixel, getFrame, getFrameSequenceLength, getRawFrameCount, getRefPixelX, getRefPixelY, nextFrame, paint, prevFrame, setFrame, setFrameSequence, setImage, setRefPixelPosition, setTransform

Methods inherited from class javax.microedition.lcdui.game.Layer
getHeight, getWidth, getX, getY, isVisible, move, setPosition, setVisible

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

PAUS

```
public static final byte PAUS
```

Naipes

See Also:

Constant Field Values

COPAS

```
public static final byte COPAS
```

Naipes

See Also:

Constant Field Values

ESPADAS

```
public static final byte ESPADAS
```

Naipes

See Also:

Constant Field Values

OUROS

```
public static final byte OUROS
```

Naipes

See Also:

Constant Field Values

LARGURA_CARTA

```
public static final byte LARGURA_CARTA
```

Largura da imagem da carta

See Also:

Constant Field Values

ALTURA_CARTA

```
public static final byte ALTURA_CARTA
```

Altura da imagem da carta

See Also:
Constant Field Values

CORINGA

`public static final byte` **`CORINGA`**

Valores das cartas

See Also:
Constant Field Values

DOIS

`public static final byte` **`DOIS`**

Valores das cartas

See Also:
Constant Field Values

TRES

`public static final byte` **`TRES`**

Valores das cartas

See Also:
Constant Field Values

QUATRO

`public static final byte` **`QUATRO`**

Valores das cartas

See Also:
Constant Field Values

CINCO

`public static final byte` **`CINCO`**

Valores das cartas

See Also:
Constant Field Values

SEIS

`public static final byte` **`SEIS`**

Valores das cartas

See Also:
Constant Field Values

SETE

`public static final byte` **`SETE`**

Valores das cartas

See Also:
Constant Field Values

OITO

`public static final byte` **`OITO`**

Valores das cartas

See Also:

Constant Field Values

NOVE

`public static final byte` **`NOVE`**

Valores das cartas

See Also:

Constant Field Values

DEZ

`public static final byte` **`DEZ`**

Valores das cartas

See Also:

Constant Field Values

DAMA

`public static final byte` **`DAMA`**

Valores das cartas

See Also:

Constant Field Values

VALETE

`public static final byte` **`VALETE`**

Valores das cartas

See Also:

Constant Field Values

REI

`public static final byte` **`REI`**

Valores das cartas

See Also:

Constant Field Values

AS

`public static final byte` **`AS`**

Valores das cartas

See Also:

Constant Field Values

VERSO

`public static final byte` **`VERSO`**

Valores das cartas

See Also:
Constant Field Values

PRIMEIRO_VALOR

```
public static final byte PRIMEIRO_VALOR
```

Variáveis utilizadas em loops

See Also:
Constant Field Values

ULTIMO_VALOR

```
public static final byte ULTIMO_VALOR
```

Variáveis utilizadas em loops

See Also:
Constant Field Values

PRIMEIRO_NAIPE

```
public static final byte PRIMEIRO_NAIPE
```

See Also:
Constant Field Values

ULTIMO_NAIPE

```
public static final byte ULTIMO_NAIPE
```

See Also:
Constant Field Values

Constructor Detail

Carta

```
public Carta(javax.microedition.lcdui.Image imagem,
             byte naipe,
             byte valor)
```

Construtor

Method Detail

setNaipe

```
public void setNaipe(byte naipe)
```

Modifica o naipe da carta

---

setValor

```
public void setValor(byte valor)
```

Modifica o valor da carta

---

getNaipe

```
public byte getNaipe()
```

Retorna o naipe da carta

---

getValor

`public byte **getValor**()`

Retorna o valor da carta

---

getNaipeStr

`public java.lang.String **getNaipeStr**()`

Retorna uma representação do naipe da carta num objeto String

---

getValorStr

`public java.lang.String **getValorStr**()`

Retorna uma representação do valor da carta num objeto String

---

toString

`public java.lang.String **toString**()`

Retorna uma representação da carta num objeto String

Overrides:

`toString` in class `java.lang.Object`

---

equals

`public boolean **equals**(java.lang.Object obj)`

Verifica a igualdade entre duas instâncias da classe

Overrides:

`equals` in class `java.lang.Object`

---

jogo

Class CartasMidlet

```
java.lang.Object
  |
  +--javax.microedition.midlet.MIDlet
        |
        +--jogo.CartasMidlet
```

public class **CartasMidlet**
extends javax.microedition.midlet.MIDlet

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.CartasMidlet

Responsabilidades: Classe principal, que inicializa os objetos de controle do jogo

Version:
1.0

Author:
Rogério de Paula Aguilar

Constructor Summary

CartasMidlet()

Method Summary

void	**destroyApp**(boolean b) Sai da aplicação
static javax.microedition.lcdui.Display	**getDisplay**() Retorna o display do jogo
void	**pauseApp**()
void	**startApp**() Inicializa o MIDlet

Methods inherited from class javax.microedition.midlet.MIDlet

checkPermission, getAppProperty, notifyDestroyed, notifyPaused, platformRequest, resumeRequest

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CartasMidlet

public **CartasMidlet**()

Method Detail

startApp

public void **startApp**()

Inicializa o MIDlet

Specified by:

startApp in class javax.microedition.midlet.MIDlet

---

pauseApp

public void **pauseApp**()

Specified by:

pauseApp in class javax.microedition.midlet.MIDlet

---

destroyApp

public void **destroyApp**(boolean b)

Sai da aplicação

Specified by:

destroyApp in class javax.microedition.midlet.MIDlet

getDisplay

```
public static javax.microedition.lcdui.Display getDisplay()
```

Retorna o display do jogo

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS **FRAMES NO FRAMES** **All Classes All Classes**
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

jogo

Class frmAjuda

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Screen
              |
              +--javax.microedition.lcdui.Form
                    |
                    +--jogo.frmAjuda
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

public class **frmAjuda**
extends javax.microedition.lcdui.Form
implements javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.frmAjuda

Responsabilidades: Exibir a ajuda para o usuário

Version:

1.0

Author:

Rogério de Paula Aguilar

Constructor Summary

`frmAjuda(java.lang.String titulo, javax.microedition.lcdui.Displayable d, javax.microedition.lcdui.Display display)`

Construtor

Method Summary

void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Método invocado quando o jogador seleciona algum comando

Methods inherited from class javax.microedition.lcdui.Form
append, append, append, delete, deleteAll, get, getHeight, getWidth, insert, set, setItemStateListener, size

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getTicker, getTitle, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

frmAjuda

```
public frmAjuda(java.lang.String titulo,
                javax.microedition.lcdui.Displayable d,
                javax.microedition.lcdui.Display display)
```

Construtor

Method Detail

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Método invocado quando o jogador seleciona algum comando

Specified by:

commandAction in interface javax.microedition.lcdui.CommandListener

jogo

Class frmEscolhaNaipe

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Screen
              |
              +--javax.microedition.lcdui.Form
                    |
                    +--jogo.frmEscolhaNaipe
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

public class **frmEscolhaNaipe**
extends javax.microedition.lcdui.Form
implements javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.frmfrmEscolhaNaipe

Responsabilidades: permite que o usuário escolha o naipe da próxima carta (Valete(*))

Version:

1.0

Author:

Rogério de Paula Aguilar

Constructor Summary

frmEscolhaNaipe(jogo.JogadorHumano d, jogo.apibasica.ControleJogo c, ControleJogo.Mensagem m) Construtor

Method Summary

void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Método invocado quando o jogador seleciona algum comando

Methods inherited from class javax.microedition.lcdui.Form
append, append, append, delete, deleteAll, get, getHeight, getWidth, insert,

set, setItemStateListener, size

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getTicker, getTitle, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

frmEscolhaNaipe

```
public frmEscolhaNaipe(jogo.JogadorHumano d,
                       jogo.apibasica.ControleJogo c,
                       ControleJogo.Mensagem m)
```

Construtor

Method Detail

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Método invocado quando o jogador seleciona algum comando

Specified by:

commandAction in interface javax.microedition.lcdui.CommandListener

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

jogo

Class frmEspera

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Screen
              |
```

```
javax.microedition.lcdui.Form
      |
      +--jogo.frmEspera
```

---

public class **frmEspera**
extends javax.microedition.lcdui.Form

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.frmEspera

Responsabilidades: Exibir uma tela para o usuário enquanto este aguarda o término de algum processo do jogo

Version:
1.0

Author:
Rogério de Paula Aguilar

Constructor Summary

`frmEspera``()` Construtor
`frmEspera``(java.lang.String msg)` COnstrutor

Method Summary

`javax.microedition.lcdui.ImageItem`	**`getImage`**`()` Retorna a imagem associada a esta tela de espera
`void`	**`reset`**`(java.lang.String msg)` Retira todos os itens do formulário

Methods inherited from class javax.microedition.lcdui.Form
`append, append, append, delete, deleteAll, get, getHeight, getWidth, insert, set, setItemStateListener, size`

Methods inherited from class javax.microedition.lcdui.Displayable
`addCommand, getTicker, getTitle, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged`

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

frmEspera

`public frmEspera()`

Construtor

---

frmEspera

`public frmEspera(java.lang.String msg)`

COnstrutor

Method Detail

getImage

`public javax.microedition.lcdui.ImageItem getImage()`

Retorna a imagem associada a esta tela de espera

---

reset

`public void reset(java.lang.String msg)`

Retira todos os itens do formulário

Overview **Package** **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Overview **Package** **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

---

jogo

Class frmListaSalas

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Screen
              |
              +--javax.microedition.lcdui.Form
                    |
                    +--jogo.frmListaSalas
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

---

public class **frmListaSalas**
extends javax.microedition.lcdui.Form
implements javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.frmListaSalas

Responsabilidades: Exibe as salas para o usuário e um campo para digitar o apelido, quando o jogador estiver começando um jogo on-line

Version:
1.0
Author:
Rogério de Paula Aguilar

Constructor Summary

frmListaSalas(java.lang.String titulo, javax.microedition.lcdui.Displayable d, java.lang.String salas, jogo.apibasica.ControleJogo c) Construtor

Method Summary

void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Método que é invocado quando o jogador seleciona algum comando
void	**reset**(java.lang.String salas) Apaga o formulário coloca a nova lista de salas

Methods inherited from class javax.microedition.lcdui.Form

append, append, append, delete, deleteAll, get, getHeight, getWidth, insert, set, setItemStateListener, size

Methods inherited from class javax.microedition.lcdui.Displayable

addCommand, getTicker, getTitle, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

frmListaSalas

```
public frmListaSalas(java.lang.String titulo,
                     javax.microedition.lcdui.Displayable d,
                     java.lang.String salas,
                     jogo.apibasica.ControleJogo c)
```

Construtor

Method Detail

reset

```
public void reset(java.lang.String salas)
```

Apaga o formulário coloca a nova lista de salas

---

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Método que é invocado quando o jogador seleciona algum comando

Specified by:

`commandAction` in interface `javax.microedition.lcdui.CommandListener`

---

jogo

Class frmOpcoes

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Screen
              |
              +--javax.microedition.lcdui.Form
                    |
                    +--jogo.frmOpcoes
```

All Implemented Interfaces:
javax.microedition.lcdui.CommandListener

public class **frmOpcoes**
extends javax.microedition.lcdui.Form
implements javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.frmOpcoes

Responsabilidades: Exibir opções para o usuário, entre elas opção de habilitar/desabilitar o modo open mau mau,

opção de habilitar/desabilitar a visualização de informações sobre a carta selecionada.
Também exibe campos onde o jogador deve digitar o endereço e a porta do servidor.

Constructor Summary

frmOpcoes(java.lang.String titulo, javax.microedition.lcdui.Displayable d) Construtor

Method Summary

void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Método que é invocado quando o jogador seleciona algum comando
void	**setExibirInformacoesCarta**(boolean b) Seta a opção Exibir informações
void	**setIPServidor**(java.lang.String ip) Modifica o ip do servidor
void	**setOpenMauMau**(boolean b) Seta a opção open Mau Mau
void	**setPortaServidor**(java.lang.String porta) Modifica a porta do servidor

Methods inherited from class javax.microedition.lcdui.Form
append, append, append, delete, deleteAll, get, getHeight, getWidth, insert, set, setItemStateListener, size

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getTicker, getTitle, isShown, removeCommand, setCommandListener, setTicker, setTitle, sizeChanged

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

frmOpcoes

```
public frmOpcoes(java.lang.String titulo,
                 javax.microedition.lcdui.Displayable d)
```

Construtor

Method Detail

setOpenMauMau

```
public void setOpenMauMau(boolean b)
```

Seta a opção open Mau Mau

---

setExibirInformacoesCarta

```
public void setExibirInformacoesCarta(boolean b)
```

Seta a opção Exibir informações

---

setIPServidor

```
public void setIPServidor(java.lang.String ip)
```

Modifica o ip do servidor

---

setPortaServidor

```
public void setPortaServidor(java.lang.String porta)
```

Modifica a porta do servidor

---

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Método que é invocado quando o jogador seleciona algum comando

Specified by:

`commandAction` in interface `javax.microedition.lcdui.CommandListener`

---

jogo

Class frmVisualizadorCartas

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Canvas
              |
              +--jogo.frmVisualizadorCartas
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

public class **frmVisualizadorCartas**
extends javax.microedition.lcdui.Canvas
implements javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.frmAjuda

Responsabilidades: Exibir as cartas do baralho

Version:

1.0

Author:

Rogério de Paula Aguilar

Field Summary

Fields inherited from class javax.microedition.lcdui.Canvas

DOWN, FIRE, GAME_A, GAME_B, GAME_C, GAME_D, KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND, KEY_STAR, LEFT, RIGHT, UP

Constructor Summary

frmVisualizadorCartas(java.lang.String titulo, javax.microedition.lcdui.Displayable telaVolta, javax.microedition.lcdui.Display display, jogo.Baralho baralho, jogo.apibasica.Dispositivo d) Construtor

Method Summary

void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Método que é invocado quando o usuário seleciona algum comando
void	**paint**(javax.microedition.lcdui.Graphics g) Desenha a carta atual e a s informações sobre a mesma

Methods inherited from class javax.microedition.lcdui.Canvas
getGameAction, getKeyCode, getKeyName, hasPointerEvents, hasPointerMotionEvents, hasRepeatEvents, hideNotify, isDoubleBuffered, keyPressed, keyReleased, keyRepeated, pointerDragged, pointerPressed, pointerReleased, repaint, repaint, serviceRepaints, setFullScreenMode, showNotify, sizeChanged

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

frmVisualizadorCartas

```
public frmVisualizadorCartas(java.lang.String titulo,
                             javax.microedition.lcdui.Displayable telaVolta,
                             javax.microedition.lcdui.Display display,
```

```
                        jogo.Baralho baralho,
                        jogo.apibasica.Dispositivo d)
```

Construtor

Method Detail

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Método que é invocado quando o usuário seleciona algum comando

Specified by:

`commandAction` in interface `javax.microedition.lcdui.CommandListener`

---

paint

```
public void paint(javax.microedition.lcdui.Graphics g)
```

Desenha a carta atual e a s informações sobre a mesma

Specified by:

`paint` in class `javax.microedition.lcdui.Canvas`

---

jogo

Class Jogador

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Canvas
              |
              +--javax.microedition.lcdui.game.GameCanvas
                    |
                    +--jogo.Jogador
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

Direct Known Subclasses:

JogadorComputador, JogadorHumano

public abstract class **Jogador**
extends javax.microedition.lcdui.game.GameCanvas
implements javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.Jogador

Responsabilidades: Guarda as informações e ações que são comuns aos jogadores, independente se estes
são jogadores humanos ou o computador

Version:

1.0

Author:

Rogério de Paula Aguilar

Field Summary

protected jogo.Baralho	**baralho** Baralho
protected java.util.Stack	**cartasAtuais** Cartas do jogador
protected int	**cartaSelecionada** Índice da carta selecionada
protected javax.microedition.lcdui.Command	**cmdSair** Comando utilizado para sair

protected jogo.apibasica.ControleJogo	**controleJogo**
protected byte	**deslocamentoSetaCima**
protected byte	**deslocamentoSetaCimaBaixo**
protected byte	**deslocamentoSetaDireita**
protected byte	**deslocamentoSetaEsquerda**
protected boolean	**DISSE_MAU_MAU** Indica se o jogador disse mau-mau ou não
static int	**ESPERA** Tempo de espera
static javax.microedition.lcdui.game.Sprite	**imgJogador** Imagem do jogador
static javax.microedition.lcdui.game.Sprite	**imgPensar** Imagem que aparece na tela do jogador que está jogando
static javax.microedition.lcdui.game.Sprite	**imgSeta** Imagem da seta de navegação
protected int	**posicao** Posição inicial para o desenho das cartas
protected int	**transformacaoAtual** Transformação atual (utilizada para o desenho das setas de navegação)

Fields inherited from class javax.microedition.lcdui.game.GameCanvas
DOWN_PRESSED, FIRE_PRESSED, GAME_A_PRESSED, GAME_B_PRESSED, GAME_C_PRESSED, GAME_D_PRESSED, LEFT_PRESSED, RIGHT_PRESSED, UP_PRESSED

Fields inherited from class javax.microedition.lcdui.Canvas
DOWN, FIRE, GAME_A, GAME_B, GAME_C, GAME_D, KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND, KEY_STAR, LEFT, RIGHT, UP

Constructor Summary

Jogador(java.lang.String nome, jogo.Baralho baralho, jogo.apibasica.ControleJogo controleJogo) Construtor

Method Summary	
void	**adicionarCarta**(jogo.Carta carta) Adiciona uma carta para este jogador
jogo.Carta	**carta**(int indice) Retorna uma carta do baralho
void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Método invocado quando o jogador seleciona algum comando do jogo
boolean	**equals**(java.lang.Object obj) Indica se duas instâncias de um jogador são iguais
boolean	**getDisseMauMau**() Retorna o valor de DISSE_MAU_MAU
int	**getIndiceCarta**(jogo.Carta carta) Retorna o índice de determinada carta
java.lang.String	**getNome**() Retorna o nome do jogador
abstract void	**jogar**(java.lang.Object[] args) Método de jogo
void	**keyPressed**(int keyCode) Método invocado quando o ujogador pressiona alguma tecla
void	**keyReleased**(int keyCode) Método invocado quando o usuário solta uma tecla que estava pressionada
int	**numeroCartas**() Retorna o número de cartas do jogador
void	**paint**(javax.microedition.lcdui.Graphics g) Desenha os elementos da interface gráfica que são comuns a todos os tipos de jogadores
jogo.Carta	**removerCarta**() Remove uma carta do jogador
jogo.Carta	**removerCarta**(int i) Remove uma carta do jogador pelo índice da carta
void	**setDisseMauMau**(boolean d) Seta a opção disse mau mau
void	**setNome**(java.lang.String nome) Modifica o nome do jogador

java.lang.String	**toString**() Retorna uma representação do jogador no formato de um objeto String
void	**zerar**() Deixa o jogador sem nenhuma carta

Methods inherited from class javax.microedition.lcdui.game.GameCanvas
flushGraphics, flushGraphics, getGraphics, getKeyStates

Methods inherited from class javax.microedition.lcdui.Canvas
getGameAction, getKeyCode, getKeyName, hasPointerEvents, hasPointerMotionEvents, hasRepeatEvents, hideNotify, isDoubleBuffered, keyRepeated, pointerDragged, pointerPressed, pointerReleased, repaint, repaint, serviceRepaints, setFullScreenMode, showNotify, sizeChanged

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

cartasAtuais

protected java.util.Stack **cartasAtuais**

Cartas do jogador

---

baralho

protected jogo.Baralho **baralho**

Baralho

ESPERA

public static final int **ESPERA**

Tempo de espera

See Also:

Constant Field Values

---

controleJogo
`protected jogo.apibasica.ControleJogo` **`controleJogo`**

---

cartaSelecionada
`protected int` **`cartaSelecionada`**

Índice da carta selecionada

---

posicao
`protected int` **`posicao`**

Posição inicial para o desenho das cartas

DISSE_MAU_MAU
`protected boolean` **`DISSE_MAU_MAU`**

Indica se o jogador disse mau-mau ou não

---

cmdSair
`protected javax.microedition.lcdui.Command` **`cmdSair`**

Comando utilizado para sair

---

imgPensar
`public static javax.microedition.lcdui.game.Sprite` **`imgPensar`**

Imagem que aparece na tela do jogador que está jogando

---

imgJogador
`public static javax.microedition.lcdui.game.Sprite` **`imgJogador`**

Imagem do jogador

---

transformacaoAtual
`protected int` **`transformacaoAtual`**

Transformação atual (utilizada para o desenho das setas de navegação)

---

imgSeta
`public static javax.microedition.lcdui.game.Sprite` **`imgSeta`**

Imagem da seta de navegação

---

deslocamentoSetaEsquerda
`protected byte` **`deslocamentoSetaEsquerda`**

---

deslocamentoSetaDireita
`protected byte` **`deslocamentoSetaDireita`**

---

deslocamentoSetaCima
`protected byte` **`deslocamentoSetaCima`**

---

deslocamentoSetaCimaBaixo

```
protected byte deslocamentoSetaCimaBaixo
```

Constructor Detail

Jogador

```
public Jogador(java.lang.String nome,
               jogo.Baralho baralho,
               jogo.apibasica.ControleJogo controleJogo)
```

Construtor

Method Detail

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Método invocado quando o jogador seleciona algum comando do jogo

Specified by:

`commandAction` in interface `javax.microedition.lcdui.CommandListener`

---

adicionarCarta

```
public void adicionarCarta(jogo.Carta carta)
```

Adiciona uma carta para este jogador

---

removerCarta

```
public jogo.Carta removerCarta()
```

Remove uma carta do jogador

---

removerCarta

```
public jogo.Carta removerCarta(int i)
```

Remove uma carta do jogador pelo índice da carta

---

jogar

```
public abstract void jogar(java.lang.Object[] args)
```

Método de jogo

---

getNome

```
public java.lang.String getNome()
```

Retorna o nome do jogador

---

toString

```
public java.lang.String toString()
```

Retorna uma representação do jogador no formato de um objeto String

Overrides:

`toString` in class `java.lang.Object`

---

setNome

`public void **setNome**(java.lang.String nome)`

Modifica o nome do jogador

---

keyPressed

`public void **keyPressed**(int keyCode)`

Método invocado quando o ujogador pressiona alguma tecla

Overrides:

`keyPressed` in class `javax.microedition.lcdui.Canvas`

---

keyReleased

`public void **keyReleased**(int keyCode)`

Método invocado quando o usuário solta uma tecla que estava pressionada

Overrides:

`keyReleased` in class `javax.microedition.lcdui.Canvas`

---

numeroCartas

`public int **numeroCartas**()`

Retorna o número de cartas do jogador

---

equals

`public boolean **equals**(java.lang.Object obj)`

Indica se duas instâncias de um jogador são iguais

Overrides:

`equals` in class `java.lang.Object`

---

setDisseMauMau

`public void **setDisseMauMau**(boolean d)`

Seta a opção disse mau mau

---

getDisseMauMau

`public boolean **getDisseMauMau**()`

Retorna o valor de DISSE_MAU_MAU

---

getIndiceCarta

`public int **getIndiceCarta**(jogo.Carta carta)`

Retorna o índice de determinada carta

---

paint

`public void **paint**(javax.microedition.lcdui.Graphics g)`

Desenha os elementos da interface gráfica que são comuns a todos os tipos de jogadores

Overrides:

`paint` in class `javax.microedition.lcdui.game.GameCanvas`

---

carta

`public jogo.Carta **carta**(int indice)`

Retorna uma carta do baralho

---

zerar

`public void **zerar**()`

Deixa o jogador sem nenhuma carta

---

jogo

Class JogadorComputador

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Canvas
              |
              +--javax.microedition.lcdui.game.GameCanvas
                    |
                    +--jogo.Jogador
                          |
                          +--jogo.JogadorComputador
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

public class **JogadorComputador**
extends Jogador

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.JogadorComputador

Responsabilidades: Guarda as informações sobre um jogador controlado pelo computador e realiza a a jogada do computador

Version:

1.0

Author:

Rogério de Paula Aguilar

Field Summary

jogo.JogadorComputador.threadJogada	**jogada**

Fields inherited from class jogo.Jogador

baralho, cartasAtuais, cartaSelecionada, cmdSair, controleJogo, deslocamentoSetaCima, deslocamentoSetaCimaBaixo, deslocamentoSetaDireita, deslocamentoSetaEsquerda, DISSE_MAU_MAU, ESPERA, imgJogador, imgPensar, imgSeta, posicao, transformacaoAtual

Fields inherited from class javax.microedition.lcdui.game.GameCanvas
DOWN_PRESSED, FIRE_PRESSED, GAME_A_PRESSED, GAME_B_PRESSED, GAME_C_PRESSED, GAME_D_PRESSED, LEFT_PRESSED, RIGHT_PRESSED, UP_PRESSED

Fields inherited from class javax.microedition.lcdui.Canvas
DOWN, FIRE, GAME_A, GAME_B, GAME_C, GAME_D, KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND, KEY_STAR, LEFT, RIGHT, UP

Constructor Summary

JogadorComputador(java.lang.String nome, jogo.Baralho baralho, jogo.apibasica.ControleJogo controleJogo) Construtor

Method Summary

void	**jogar**(java.lang.Object[] arg) Método invocado para realizar a jogada do computador
void	**keyPressed**(int keyCode) Método invocado quando o jogador pressiona alguma tecla
void	**paint**(javax.microedition.lcdui.Graphics g) Desenha a interface gráfica do jogador

Methods inherited from class jogo.Jogador
adicionarCarta, carta, commandAction, equals, getDisseMauMau, getIndiceCarta, getNome, keyReleased, numeroCartas, removerCarta, removerCarta, setDisseMauMau, setNome, toString, zerar

Methods inherited from class javax.microedition.lcdui.game.GameCanvas
flushGraphics, flushGraphics, getGraphics, getKeyStates

Methods inherited from class javax.microedition.lcdui.Canvas
getGameAction, getKeyCode, getKeyName, hasPointerEvents, hasPointerMotionEvents, hasRepeatEvents, hideNotify, isDoubleBuffered, keyRepeated, pointerDragged, pointerPressed, pointerReleased, repaint, repaint, serviceRepaints, setFullScreenMode, showNotify, sizeChanged

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

jogada

```
public jogo.JogadorComputador.threadJogada jogada
```

Constructor Detail

JogadorComputador

```
public JogadorComputador(java.lang.String nome,
                         jogo.Baralho baralho,
                         jogo.apibasica.ControleJogo controleJogo)
```

Construtor

Method Detail

jogar

```
public void jogar(java.lang.Object[] arg)
```

Método invocado para realizar a jogada do computador

Specified by:

jogar in class Jogador

---

paint

```
public void paint(javax.microedition.lcdui.Graphics g)
```

Desenha a interface gráfica do jogador

Overrides:

paint in class Jogador

---

keyPressed

```
public void keyPressed(int keyCode)
```

Método invocado quando o jogador pressiona alguma tecla

Overrides:

keyPressed in class Jogador

---

jogo

Class JogadorHumano

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Canvas
              |
              +--javax.microedition.lcdui.game.GameCanvas
                    |
                    +--jogo.Jogador
                          |
                          +--jogo.JogadorHumano
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener

---

public class **JogadorHumano**
extends Jogador
implements javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.JogadorHumano

Responsabilidades: Guarda as informações sobre o jogador que está utilizando o dispositivo móvel

Version:

1.0

Author:

Rogério de Paula Aguilar

Field Summary

Fields inherited from class jogo.Jogador
baralho, cartasAtuais, cartaSelecionada, cmdSair, controleJogo, deslocamentoSetaCima, deslocamentoSetaCimaBaixo, deslocamentoSetaDireita, deslocamentoSetaEsquerda, DISSE_MAU_MAU, ESPERA, imgJogador, imgPensar, imgSeta, posicao, transformacaoAtual

Fields inherited from class javax.microedition.lcdui.game.GameCanvas
DOWN_PRESSED, FIRE_PRESSED, GAME_A_PRESSED, GAME_B_PRESSED, GAME_C_PRESSED, GAME_D_PRESSED, LEFT_PRESSED, RIGHT_PRESSED, UP_PRESSED

Fields inherited from class javax.microedition.lcdui.Canvas
DOWN, FIRE, GAME_A, GAME_B, GAME_C, GAME_D, KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND, KEY_STAR, LEFT, RIGHT, UP

Constructor Summary

JogadorHumano(java.lang.String nome, jogo.apibasica.Dispositivo dispositivo, jogo.Baralho baralho, jogo.apibasica.ControleJogo controleJogo) Construtor	

Method Summary

void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Procedimento chamado quando o jogador aciona algum comando
void	**jogar**(java.lang.Object[] args) Chama o método verificarJogada de ControleJogo para verificar a jogada, exibe uma mensagem para o usuário e envia a atualização para o servidor, caso o jogo seja on-line
void	**keyPressed**(int keyCode) Método invocado quando o usuário pressiona alguma tecla
void	**paint**(javax.microedition.lcdui.Graphics g) Desenha a interface gráfica do jogador
void	**setComandoCompra**(byte tipo) Adiciona o comando de compra de cartas na tela do usuário
void	**simularFimDeJogo**() Simula o fim de jogo

Methods inherited from class jogo.Jogador
adicionarCarta, carta, equals, getDisseMauMau, getIndiceCarta, getNome, keyReleased, numeroCartas, removerCarta, removerCarta, setDisseMauMau, setNome, toString, zerar

Methods inherited from class javax.microedition.lcdui.game.GameCanvas

flushGraphics, flushGraphics, getGraphics, getKeyStates

Methods inherited from class javax.microedition.lcdui.Canvas
getGameAction, getKeyCode, getKeyName, hasPointerEvents, hasPointerMotionEvents, hasRepeatEvents, hideNotify, isDoubleBuffered, keyRepeated, pointerDragged, pointerPressed, pointerReleased, repaint, repaint, serviceRepaints, setFullScreenMode, showNotify, sizeChanged

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle

Methods inherited from class java.lang.Object
clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

JogadorHumano

```
public JogadorHumano(java.lang.String nome,
                     jogo.apibasica.Dispositivo dispositivo,
                     jogo.Baralho baralho,
                     jogo.apibasica.ControleJogo controleJogo)
```

Construtor

Method Detail

simularFimDeJogo

```
public void simularFimDeJogo()
```

Simula o fim de jogo

---

setComandoCompra

```
public void setComandoCompra(byte tipo)
```

Adiciona o comando de compra de cartas na tela do usuário

---

jogar

```
public void jogar(java.lang.Object[] args)
```

Chama o método verificarJogada de ControleJogo para verificar a jogada, exibe uma mensagem para o usuário
e envia a atualização para o servidor, caso o jogo seja on-line

Specified by:

jogar in class Jogador

---

keyPressed

```
public void keyPressed(int keyCode)
```

Método invocado quando o usuário pressiona alguma tecla

Overrides:

keyPressed in class Jogador

---

paint

```
public void paint(javax.microedition.lcdui.Graphics g)
```

Desenha a interface gráfica do jogador

Overrides:

paint in class Jogador

---

commandAction

```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```

Procedimento chamado quando o jogador aciona algum comando

Specified by:

commandAction in interface javax.microedition.lcdui.CommandListener

Overrides:

commandAction in class Jogador

Package jogo.apibasica

Class Summary

ControleJogo	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.apibasica.ControleJogo Responsabilidades: Controlar a execução do jogo e a atualização das telas de apresentação e principal do mesmo.
Dispositivo	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis Classe: jogo.apibasica.Dispositivo Objetivo: Guarda informações sobre o dispositivo (largura da tela, altura da tela, coordenadas X e Y do centro da tela)

jogo.apibasica

Class ControleJogo

```
java.lang.Object
  |
  +--javax.microedition.lcdui.Displayable
        |
        +--javax.microedition.lcdui.Canvas
              |
              +--javax.microedition.lcdui.game.GameCanvas
                    |
                    +--jogo.apibasica.ControleJogo
```

All Implemented Interfaces:

javax.microedition.lcdui.CommandListener, java.lang.Runnable

---

public class **ControleJogo**
extends javax.microedition.lcdui.game.GameCanvas
implements java.lang.Runnable, javax.microedition.lcdui.CommandListener

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.apibasica.ControleJogo

Responsabilidades: Controlar a execução do jogo e a atualização das telas de apresentação e principal do mesmo. Controla o loop principal do jogo e a navegabilidade da aplicação.

Version:

1.0

Author:

Rogério de Paula Aguilar

Nested Class Summary

class	**ControleJogo.Mensagem** Classe que exibe mensagens na tela

Field Summary

boolean	**ANTERIOR_PULADO** Indica se o jogador anterior foi pulado
static boolean	**APRESENTACAO**

static boolean	**CONECTADO** Indica se o jogador está conectado ao servidor
static javax.microedition.io.SocketConnection	**conexao** Objeto que guarda a conexão com o servidor
static boolean	**DEBUG_MODE** Indica se o modo debug está ativo ou não
static boolean	**EXIBIR_INFORMACAO_CARTA** Estado do jogo
static jogo.frmEspera	**frmespera** Formulário de espera
static java.lang.String	**idSala** Objeto que guarda o id da sala que o usuário está conectado
static int	**indiceJogadorRede** Guarda o índice do jogador na rede
static java.io.InputStream	**is** Objeto utilizado para ler comandos que vem do servidor
static boolean	**JOGO_EM_ANDAMENTO** Indica se o jogador está no meio de uma partida
static byte	**NAIPE_ESCOLHIDO** Guarda o naipe escolhido para a próxima jogada (Jogada: Valete(*))
static java.lang.String	**nomeJogador** Objeto que guarda o nome do jogador atual
static java.lang.String[]	**nomeJogadores** Objeto utilizado para guardar os nomes dos jogadores
static int	**NUMERO_JOGADORES** Número de jogadores
static boolean	**ONLINE**
static java.io.OutputStream	**os** Objeto utilizado para enviar mensagens para o servidor

static boolean	**PENALIDADE_APLICADA** Indica se a penalidade de uma carta hostil já foi aplicada ou não
static boolean	**primeiroJogador** Indica se o jogador é o primeiro
static byte	**SENTIDO_INVERSO** Variáveis que controlam o sentido do jogo
static byte	**SENTIDO_JOGO**
static byte	**SENTIDO_NORMAL** Variáveis que controlam o sentido do jogo
static java.io.PrintStream	**socketWriter** Objeto utilizado para enviar mensagens para o servidor
static boolean	**TELASELECAO**

Fields inherited from class javax.microedition.lcdui.game.GameCanvas
DOWN_PRESSED, FIRE_PRESSED, GAME_A_PRESSED, GAME_B_PRESSED, GAME_C_PRESSED, GAME_D_PRESSED, LEFT_PRESSED, RIGHT_PRESSED, UP_PRESSED

Fields inherited from class javax.microedition.lcdui.Canvas
DOWN, FIRE, GAME_A, GAME_B, GAME_C, GAME_D, KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_POUND, KEY_STAR, LEFT, RIGHT, UP

Constructor Summary

ControleJogo(boolean s, jogo.apibasica.Dispositivo d, javax.microedition.lcdui.Display display, jogo.CartasMidlet cartasMidlet, java.lang.String ip, int porta) Construtor

Method Summary

void	**atualizarStatusJogo**(java.lang.String msg) Atualiza o status do jogo de acordo com o quadro recebido do servidor

void	**atualizarTela**() Desenha a tela de apresentação
void	**comecarJogoOnLine**(boolean enviar) Inicializa o estado do jogo para começar um novo jogo on-line
void	**commandAction**(javax.microedition.lcdui.Command command, javax.microedition.lcdui.Displayable displayable) Listener de eventos
ControleJogo.Mensagem	**elegerVencedor**() Elege o vencedor do jogo
static void	**enviarMsgServidor**(java.lang.String msg) Envia uma mensagem de texto para o servidor
static void	**exibirDebugMsg**(java.lang.String msg) Exibe informação de debug (apenas se o modo debug estiver acionado)
void	**exibirMsg**(java.lang.String msg) Mostra uma mensagem na tela do usuário
void	**exibirProximoJogador**() Muda o jogador que está sendo exibido
static void	**fecharConexao**() Fecha a conexão com o servidor
static void	**fecharConexaoSemConfirmacao**() Fecha a conexão com o servidor, mas não envia mensagem informando a saída do cliente (Utilizar somente quando processar msgs do servidor)
static boolean	**getConectado**()
static java.lang.String	**getHostServidor**() Retorna a string de conexão com o servidor
int	**getIndiceJogadorAnterior**() Rotina que retorna o índice do jogador anterior
int	**getIndiceJogadorAtual**() Retorna o índice do jogador atual
int	**getIndiceProximoJogador**() Rotina que retorna o índice do próximo jogador, mas não modifica o índice do jogador atual
static java.lang.String	**getIPServidor**() Retorna o endereço ip do servidor
jogo.Jogador	**getJogador**(int indiceJogador) Retorna um jogador
byte	**getJogadorAtual**() Retorna o índice do jogador atual

byte	**getJogadorExibido**() Retorna o jogador que está sendo exibido
int	**getJogoAtual**() Retorna o índice do jogo atual
static int	**getNumeroAleatorio**() Retorna o gerador de números aleatórios
static boolean	**getOpenMauMau**() Retorna o valor de OpenMauMau
boolean	**getPenalidadeAplicada**() Retorna o valor da variável penalidadeAplicada
static java.lang.String	**getPortaServidor**() Retorna a porta do servidor
static byte	**getQuantidadeCompra**() Retorna a quantidade de cartas que devem ser compradas
boolean	**getRecebeuPermissaoJogada**()
void	**incrementaJogoAtual**() Incrementa o índice do jogo atual
void	**inverterSentidoJogo**() Inverte o sentido do jogo
void	**loopback**() Verifica a integridade entre os procedimentos atualizarStatusJogo e montarStatusJogo
void	**mainLoop**() Loop do jogo (descontinuado)
java.lang.String	**montarStatusJogo**(ControleJogo.Mensagem mensagem, boolean fecharJogo) Monta o quadro contendo o status do jogo
void	**parar**() Para a execução do jogo
int	**proximoJogador**() Rotina que verifica qual é o próximo jogador e modfica o índice do jogador atual para o próximo
void	**resetarJogadores**() Reseta o estado dos jogadores
static java.lang.String	**retornarMsgServidor**(java.lang.String URL, java.lang.String str, boolean manterConexao, jogo.apibasica.ControleJogo controleJogo) Conecta com o servidor e envia uma requisição.
void	**run**() Loop de atualização da tela de apresentação do jogo
void	**setApresentacao**(boolean b)

static void	**setConectado**(boolean b)
static void	**setIPServidor**(java.lang.String IP) Modifica o endereço ip do servidor
static void	**setOpenMauMau**(boolean b) Seta Open MauMau
void	**setPenalidadeAplicada**(boolean b) Modifica o valor da variável PenalidadeAplicada
static void	**setPortaServidor**(java.lang.String porta) Modifica a porta do servidor
static void	**setQuantidadeCompra**(byte quantidade) Seta a quantidade de cartas que devem ser compradas
void	**setRecebeuPermissaoJogada**(boolean b)
void	**setTelaSelecao**(boolean b)
void	**setVencedor**(jogo.Jogador jogador) Mostra a tela informando o vencedor do jogo
ControleJogo.Mensagem	**verificarJogada**(jogo.Carta carta, jogo.JogadorHumano jogador) Verifica se a jogada que está sendo feita é válida ou não

Methods inherited from class javax.microedition.lcdui.game.GameCanvas
flushGraphics, flushGraphics, getGraphics, getKeyStates, paint

Methods inherited from class javax.microedition.lcdui.Canvas
getGameAction, getKeyCode, getKeyName, hasPointerEvents, hasPointerMotionEvents, hasRepeatEvents, hideNotify, isDoubleBuffered, keyPressed, keyReleased, keyRepeated, pointerDragged, pointerPressed, pointerReleased, repaint, repaint, serviceRepaints, setFullScreenMode, showNotify, sizeChanged

Methods inherited from class javax.microedition.lcdui.Displayable
addCommand, getHeight, getTicker, getTitle, getWidth, isShown, removeCommand, setCommandListener, setTicker, setTitle

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,

```
wait, wait, wait
```

Field Detail

DEBUG_MODE

```
public static final boolean DEBUG_MODE
```

Indica se o modo debug está ativo ou não

See Also:

Constant Field Values

JOGO_EM_ANDAMENTO

```
public static boolean JOGO_EM_ANDAMENTO
```

Indica se o jogador está no meio de uma partida

CONECTADO

```
public static boolean CONECTADO
```

Indica se o jogador está conectado ao servidor

EXIBIR_INFORMACAO_CARTA

```
public static boolean EXIBIR_INFORMACAO_CARTA
```

Estado do jogo

APRESENTACAO

```
public static boolean APRESENTACAO
```

ONLINE

```
public static boolean ONLINE
```

TELASELECAO

```
public static boolean TELASELECAO
```

SENTIDO_NORMAL

```
public static final byte SENTIDO_NORMAL
```

Variáveis que controlam o sentido do jogo

See Also:

Constant Field Values

SENTIDO_INVERSO

```
public static final byte SENTIDO_INVERSO
```

Variáveis que controlam o sentido do jogo

See Also:

Constant Field Values

SENTIDO_JOGO
`public static byte` **`SENTIDO_JOGO`**

PENALIDADE_APLICADA
`public static boolean` **`PENALIDADE_APLICADA`**

Indica se a penalidade de uma carta hostil já foi aplicada ou não

---

frmespera
`public static jogo.frmEspera` **`frmespera`**

Formulário de espera

NUMERO_JOGADORES
`public static int` **`NUMERO_JOGADORES`**

Número de jogadores

ANTERIOR_PULADO
`public boolean` **`ANTERIOR_PULADO`**

Indica se o jogador anterior foi pulado

---

conexao
`public static javax.microedition.io.SocketConnection` **`conexao`**

Objeto que guarda a conexão com o servidor

---

is
`public static java.io.InputStream` **`is`**

Objeto utilizado para ler comandos que vem do servidor

---

os
`public static java.io.OutputStream` **`os`**

Objeto utilizado para enviar mensagens para o servidor

---

socketWriter
`public static java.io.PrintStream` **`socketWriter`**

Objeto utilizado para enviar mensagens para o servidor

---

nomeJogadores
`public static java.lang.String[]` **`nomeJogadores`**

Objeto utilizado para guardar os nomes dos jogadores

---

primeiroJogador
`public static boolean` **`primeiroJogador`**

Indica se o jogador é o primeiro

---

idSala
`public static java.lang.String` **`idSala`**

Objeto que guarda o id da sala que o usuário está conectado

---

nomeJogador
```
public static java.lang.String nomeJogador
```
Objeto que guarda o nome do jogador atual

---

indiceJogadorRede
```
public static int indiceJogadorRede
```
Guarda o índice do jogador na rede

NAIPE_ESCOLHIDO
```
public static byte NAIPE_ESCOLHIDO
```
Guarda o naipe escolhido para a próxima jogada (Jogada: Valete(*))

Constructor Detail

ControleJogo
```
public ControleJogo(boolean s,
                    jogo.apibasica.Dispositivo d,
                    javax.microedition.lcdui.Display display,
                    jogo.CartasMidlet cartasMidlet,
                    java.lang.String ip,
                    int porta)
```
Construtor

Method Detail

getNumeroAleatorio
```
public static int getNumeroAleatorio()
```
Retorna o gerador de números aleatórios

---

setRecebeuPermissaoJogada
```
public void setRecebeuPermissaoJogada(boolean b)
```

---

getRecebeuPermissaoJogada
```
public boolean getRecebeuPermissaoJogada()
```

---

mainLoop
```
public void mainLoop()
```
Loop do jogo (descontinuado)

---

atualizarTela
```
public void atualizarTela()
```
Desenha a tela de apresentação

---

run
```
public void run()
```
Loop de atualização da tela de apresentação do jogo

Specified by:
`run` in interface `java.lang.Runnable`

---

parar
```
public void parar()
```
Para a execução do jogo

---

commandAction
```
public void commandAction(javax.microedition.lcdui.Command command,
                          javax.microedition.lcdui.Displayable displayable)
```
Listener de eventos
Specified by:
`commandAction` in interface `javax.microedition.lcdui.CommandListener`

---

comecarJogoOnLine
```
public void comecarJogoOnLine(boolean enviar)
```
Inicializa o estado do jogo para começar um novo jogo on-line

---

resetarJogadores
```
public void resetarJogadores()
```
Reseta o estado dos jogadores

---

setApresentacao
```
public void setApresentacao(boolean b)
```

---

setTelaSelecao
```
public void setTelaSelecao(boolean b)
```

---

proximoJogador
```
public int proximoJogador()
```
Rotina que verifica qual é o próximo jogador e modfica o índice do jogador atual para o próximo

---

getIndiceProximoJogador
```
public int getIndiceProximoJogador()
```
Rotina que retorna o índice do próximo jogador, mas não modifica o índice do jogador atual

---

getIndiceJogadorAnterior
```
public int getIndiceJogadorAnterior()
```
Rotina que retorna o índice do jogador anterior

---

exibirProximoJogador
```
public void exibirProximoJogador()
```
Muda o jogador que está sendo exibido

getJogadorAtual
`public byte **getJogadorAtual**()`
Retorna o índice do jogador atual

getJogadorExibido
`public byte **getJogadorExibido**()`
Retorna o jogador que está sendo exibido

setOpenMauMau
`public static void **setOpenMauMau**(boolean b)`
Seta Open MauMau

getOpenMauMau
`public static boolean **getOpenMauMau**()`
Retorna o valor de OpenMauMau

exibirDebugMsg
`public static void **exibirDebugMsg**(java.lang.String msg)`
Exibe informação de debug (apenas se o modo debug estiver acionado)

verificarJogada
`public ControleJogo.Mensagem **verificarJogada**(jogo.Carta carta, jogo.JogadorHumano jogador)`
Verifica se a jogada que está sendo feita é válida ou não

elegerVencedor
`public ControleJogo.Mensagem **elegerVencedor**()`
Elege o vencedor do jogo

setQuantidadeCompra
`public static void **setQuantidadeCompra**(byte quantidade)`
Seta a quantidade de cartas que devem ser compradas

getQuantidadeCompra
`public static byte **getQuantidadeCompra**()`
Retorna a quantidade de cartas que devem ser compradas

getJogador
`public jogo.Jogador **getJogador**(int indiceJogador)`
Retorna um jogador

inverterSentidoJogo
`public void **inverterSentidoJogo**()`
Inverte o sentido do jogo

setVencedor

```
public void setVencedor(jogo.Jogador jogador)
```

Mostra a tela informando o vencedor do jogo

retornarMsgServidor

```
public static java.lang.String retornarMsgServidor(java.lang.String URL,
                                                    java.lang.String str,
                                                    boolean manterConexao,

jogo.apibasica.ControleJogo controleJogo)
```

Conecta com o servidor e envia uma requisição. Se manterConexao for igual a true,

mantém a conexão com o servidor aberta e inicializa a thread de leitura de comandos,

senão fecha a conexão após receber a resposta ao comando enviado.

getHostServidor

```
public static java.lang.String getHostServidor()
```

Retorna a string de conexão com o servidor

exibirMsg

```
public void exibirMsg(java.lang.String msg)
```

Mostra uma mensagem na tela do usuário

fecharConexao

```
public static void fecharConexao()
```

Fecha a conexão com o servidor

fecharConexaoSemConfirmacao

```
public static void fecharConexaoSemConfirmacao()
```

Fecha a conexão com o servidor, mas não envia mensagem informando a saída do cliente (Utilizar somente quando processar msgs do servidor)

setConectado

```
public static void setConectado(boolean b)
```

getConectado

```
public static boolean getConectado()
```

enviarMsgServidor

```
public static void enviarMsgServidor(java.lang.String msg)
```

Envia uma mensagem de texto para o servidor

montarStatusJogo

```
public java.lang.String montarStatusJogo(ControleJogo.Mensagem mensagem,
                                         boolean fecharJogo)
```

Monta o quadro contendo o status do jogo

---

atualizarStatusJogo

```
public void atualizarStatusJogo(java.lang.String msg)
```

Atualiza o status do jogo de acordo com o quadro recebido do servidor

---

loopback

```
public void loopback()
```

Verifica a integridade entre os procedimentos atualizarStatusJogo e montarStatusJogo

---

setPenalidadeAplicada

```
public void setPenalidadeAplicada(boolean b)
```

Modifica o valor da variável PenalidadeAplicada

---

getPenalidadeAplicada

```
public boolean getPenalidadeAplicada()
```

Retorna o valor da variável penalidadeAplicada

---

getIndiceJogadorAtual

```
public int getIndiceJogadorAtual()
```

Retorna o índice do jogador atual

---

setIPServidor

```
public static void setIPServidor(java.lang.String IP)
```

Modifica o endereço ip do servidor

---

setPortaServidor

```
public static void setPortaServidor(java.lang.String porta)
```

Modifica a porta do servidor

---

getIPServidor

```
public static java.lang.String getIPServidor()
```

Retorna o endereço ip do servidor

---

getPortaServidor

```
public static java.lang.String getPortaServidor()
```

Retorna a porta do servidor

---

incrementaJogoAtual

```
public void incrementaJogoAtual()
```

Incrementa o índice do jogo atual

---

getJogoAtual

```
public int getJogoAtual()
```

Retorna o índice do jogo atual

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

jogo.apibasica

Class Dispositivo

```
java.lang.Object
  |
  +--jogo.apibasica.Dispositivo
```

public class **Dispositivo**
extends java.lang.Object

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis

Classe: jogo.apibasica.Dispositivo

Objetivo: Guarda informações sobre o dispositivo (largura da tela, altura da tela, coordenadas X e Y do centro da tela)

Version:
1.0

Author:
Rogério de Paula Aguilar

Constructor Summary

Dispositivo(javax.microedition.lcdui.Display display) Construtor

Method Summary

static int	**getAlturaTela**() Retorna a altura da tela
static int	**getCentroX**() Retorna a coordenada X do centro da tela
static int	**getCentroY**() Retorna a coordenada Y do centro da tela
static int	**getLarguraTela**()

	Retorna a largura da tela

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Dispositivo

`public` **`Dispositivo`**`(javax.microedition.lcdui.Display display)`

Construtor

Method Detail

getLarguraTela

`public static int` **`getLarguraTela`**`()`

Retorna a largura da tela

---

getAlturaTela

`public static int` **`getAlturaTela`**`()`

Retorna a altura da tela

---

getCentroX

`public static int` **`getCentroX`**`()`

Retorna a coordenada X do centro da tela

---

getCentroY

`public static int` **`getCentroY`**`()`

Retorna a coordenada Y do centro da tela

Package jogo.rede

Interface Summary

MontarArvore	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Rogério de Paula Aguilar 8NA Luiz Fernando P.S.

Class Summary

JogadorRede	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Rogério de Paula Aguilar 8NA Luiz Fernando P.S.
Sala	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Rogério de Paula Aguilar 8NA Luiz Fernando P.S.
Servidor	2003 - Faculdade Senac de Ciências Exatas e Tecnologia Rogério de Paula Aguilar 8NA Luiz Fernando P.S.

jogo.rede

Interface MontarArvore

All Known Implementing Classes:

Sala, Servidor

public interface **MontarArvore**

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Rogério de Paula Aguilar 8NA

Luiz Fernando P.S. Forgas 8NA

Trabalho de Conclusão de Curso: Técnicas de desenvolvimento de Jogos para dispositivos móveis

Pacote: jogo.rede

Interface: MontarArvore

Descrição: Interface utilizada para montar os nós da árvore de salas do servidor

Method Summary

`javax.swing.tree.DefaultMutableTreeNode`	**`retornarNo`**`(javax.swing.JTree arvore)` Método que deve ser implementado pelas classes que queiram fazer parte da árvore do servidor

Method Detail

retornarNo

```
public javax.swing.tree.DefaultMutableTreeNode
retornarNo(javax.swing.JTree arvore)
```

Método que deve ser implementado pelas classes que queiram fazer parte da árvore do servidor

jogo.rede

Class JogadorRede

```
java.lang.Object
  |
  +--jogo.rede.JogadorRede
```

All Implemented Interfaces:

java.io.Serializable

public class **JogadorRede**
extends java.lang.Object
implements java.io.Serializable

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Rogério de Paula Aguilar 8NA

Luiz Fernando P.S. Forgas 8NA

Trabalho de Conclusão de Curso: Técnicas de desenvolvimento de Jogos para dispositivos móveis

Pacote: jogo.rede

Classe: JogadorRede

Descrição: Guarda as informações sobre os jogadores da rede.

Version:

1.0

Author:

Rogério de Paula Aguilar

See Also:

Serialized Form

Constructor Summary

JogadorRede(java.lang.String nome, java.io.BufferedReader reader, java.io.PrintWriter writer, jogo.rede.Sala sala, java.net.Socket socket) Construtor.

Method Summary

void	**desconectarUsuario**()

	Envia uma mensagem para a aplicação cliente indicando que o servidor irá desconectar o usuário
void	**enviarMensagem**(java.lang.String str) Envia uma mensagem para o cliente
boolean	**equals**(java.lang.Object obj) Retorna se duas instâncias desta classe são iguais
void	**fechar**() Fecha a conexão com o jogador
java.lang.String	**getIp**() Retorna o ip da aplicação cliente
boolean	**getManterConexao**() Retorna o valor de Manter conexão
java.lang.String	**getNome**() Retorna o nome do jogador
java.lang.String	**getPorta**() Retorna a porta da aplicação cliente
int	**hashCode**() Retorna um inteiro para ser utilizado com índice de uma tabela hash
void	**iniciarThreadLeitura**() Inicializa a thread de leitura de comandos.
void	**retirarJogadorSala**() Remove o jogador da sala.
void	**setManterConexao**(boolean b) Modifica o valor de Manter conexão
java.lang.String	**toString**() Retorna uma representação deste jogador num objeto String

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

JogadorRede

```
public JogadorRede(java.lang.String nome,
                   java.io.BufferedReader reader,
                   java.io.PrintWriter writer,
                   jogo.rede.Sala sala,
                   java.net.Socket socket)
            throws java.io.IOException
```

Construtor. Não inicializa a thread de leitura. Esta deve ser inicializada manualmente através do método iniciarThreadLeitura

Method Detail

iniciarThreadLeitura

`public void **iniciarThreadLeitura**()`

Inicializa a thread de leitura de comandos.

---

getNome

`public java.lang.String **getNome**()`

Retorna o nome do jogador

---

getIp

`public java.lang.String **getIp**()`

Retorna o ip da aplicação cliente

---

getPorta

`public java.lang.String **getPorta**()`

Retorna a porta da aplicação cliente

---

equals

`public boolean **equals**(java.lang.Object obj)`

Retorna se duas instâncias desta classe são iguais

Overrides:

`equals` in class `java.lang.Object`

---

hashCode

`public int **hashCode**()`

Retorna um inteiro para ser utilizado com índice de uma tabela hash

Overrides:

`hashCode` in class `java.lang.Object`

---

fechar

`public void **fechar**()`

Fecha a conexão com o jogador

---

desconectarUsuario

`public void **desconectarUsuario**()`

Envia uma mensagem para a aplicação cliente indicando que o servidor irá desconectar o usuário

---

retirarJogadorSala

`public void **retirarJogadorSala**()`

Remove o jogador da sala. OBS: Não envia mensagem para o jogador. Sempre utilizar o método
enviarMensagem para informar a aplicação cliente que o jogador está sendo retirado da

sala
antes de chamar este método.

toString

`public java.lang.String **toString**()`

Retorna uma representação deste jogador num objeto String

Overrides:

`toString` in class `java.lang.Object`

setManterConexao

`public void **setManterConexao**(boolean b)`

Modifica o valor de Manter conexão

getManterConexao

`public boolean **getManterConexao**()`

Retorna o valor de Manter conexão

enviarMensagem

`public void **enviarMensagem**(java.lang.String str)`

Envia uma mensagem para o cliente

jogo.rede

Class Sala

```
java.lang.Object
  |
  +--jogo.rede.Sala
```

All Implemented Interfaces:
MontarArvore

public class **Sala**
extends java.lang.Object
implements MontarArvore

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Rogério de Paula Aguilar 8NA

Luiz Fernando P.S. Forgas 8NA

Trabalho de Conclusão de Curso: Técnicas de desenvolvimento de Jogos para dispositivos móveis

Pacote: jogo.rede

Classe: Sala

Descrição: Representa uma sala no servidor, onde os jogadores se encontram para jogar

Field Summary

static int	**NUMERO_JOGADORES** Número de jogadores na sala

Constructor Summary

Sala(java.lang.String id, jogo.rede.Servidor servidor) Construtor

Method Summary

void	**addAck**() Incrementa a variável ackAtualizacoes

boolean	**adicionarJogador**(jogo.rede.JogadorRede jogador) Adiciona um jogador na sala
void	**enviarMsgBroadcast**(java.lang.String msg) Envia uma mensagem a todos os jogadores da sala
int	**getAck**() Retorna o valor de ackAtualizacoes
java.lang.String	**getId**() Retorna o id da sala
jogo.rede.JogadorRede	**getJogador**(int id) Retorna um jogador pelo seu índice
int	**getJogadoresAtivos**() Retorna o número de jogadores ativos
boolean	**getJogando**() Retorna o valor da variável JOGANDO
jogo.rede.JogadorRede	**getPrimeiroJogador**() Retorna o primeiro jogador
jogo.rede.Servidor	**getServidor**() Retorna o servidor associado com esta sala
java.lang.Thread	**getThreadTimeoutJogo**() Retorna a thread que controla o tempo limite de fim de jogo
void	**log**(java.lang.String str) Adiciona uma entrada ao log do servidor
void	**removerJogador**(jogo.rede.JogadorRede jogador) Remove um jogador da sala
void	**resetar**() Reseta a sala, desconectando todos que estiverem nela
javax.swing.tree.DefaultMutableTreeNode	**retornarNo**(javax.swing.JTree arvore) Retorna o nó da árvore de salas correspondente a esta sala
void	**setJogando**(boolean jogando) Modifica o valor da variável JOGANDO
void	**setPrimeiroJogador**(jogo.rede.JogadorRede primeiroJogador Modifica o primeiro jogador
java.lang.String	**toString**() Retorna uma representação da sala num objeto String
void	**zerarAck**() Faz ackAtualizacoes = 0

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

NUMERO_JOGADORES

```
public static final int NUMERO_JOGADORES
```

Número de jogadores na sala

See Also:

Constant Field Values

Constructor Detail

Sala

```
public Sala(java.lang.String id,
            jogo.rede.Servidor servidor)
```

Construtor

Method Detail

addAck

```
public void addAck()
```

Incrementa a variável ackAtualizacoes

---

zerarAck

```
public void zerarAck()
```

Faz ackAtualizacoes = 0

---

getAck

```
public int getAck()
```

Retorna o valor de ackAtualizacoes

---

setJogando

```
public void setJogando(boolean jogando)
```

Modifica o valor da variável JOGANDO

---

getJogando

```
public boolean getJogando()
```

Retorna o valor da variável JOGANDO

---

resetar

```
public void resetar()
```

Reseta a sala, desconectando todos que estiverem nela

---

getId

```
public java.lang.String getId()
```

Retorna o id da sala

---

adicionarJogador

```
public boolean adicionarJogador(jogo.rede.JogadorRede jogador)
                         throws jogo.rede.JogadorDuplicadoException,
                                jogo.rede.NumeroJogadoresExcedidoException,
                                jogo.rede.JogandoException
```

Adiciona um jogador na sala

`jogo.rede.JogadorDuplicadoException`
`jogo.rede.NumeroJogadoresExcedidoException`
`jogo.rede.JogandoException`

---

removerJogador

```
public void removerJogador(jogo.rede.JogadorRede jogador)
                    throws jogo.rede.JogadorNaoEncontradoException
```

Remove um jogador da sala

`jogo.rede.JogadorNaoEncontradoException`

---

getJogadoresAtivos

```
public int getJogadoresAtivos()
```

Retorna o número de jogadores ativos

---

getJogador

```
public jogo.rede.JogadorRede getJogador(int id)
```

Retorna um jogador pelo seu índice

---

toString

```
public java.lang.String toString()
```

Retorna uma representação da sala num objeto String

Overrides:

`toString` in class `java.lang.Object`

---

retornarNo

```
public javax.swing.tree.DefaultMutableTreeNode
retornarNo(javax.swing.JTree arvore)
```

Retorna o nó da árvore de salas correspondente a esta sala

Specified by:

`retornarNo` in interface `MontarArvore`

---

setPrimeiroJogador

```
public void setPrimeiroJogador(jogo.rede.JogadorRede primeiroJogador)
```

Modifica o primeiro jogador

---

getPrimeiroJogador

```
public jogo.rede.JogadorRede getPrimeiroJogador()
```

Retorna o primeiro jogador

---

enviarMsgBroadcast

```
public void enviarMsgBroadcast(java.lang.String msg)
```

Envia uma mensagem a todos os jogadores da sala

---

log

```
public void log(java.lang.String str)
```

Adiciona uma entrada ao log do servidor

---

getServidor

```
public jogo.rede.Servidor getServidor()
```

Retorna o servidor associado com esta sala

---

getThreadTimeoutJogo

```
public java.lang.Thread getThreadTimeoutJogo()
```

Retorna a thread que controla o tempo limite de fim de jogo

---

jogo.rede

Class Servidor

```
java.lang.Object
  |
  +--jogo.rede.Servidor
```

All Implemented Interfaces:

MontarArvore

public class **Servidor**
extends java.lang.Object
implements MontarArvore

2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Rogério de Paula Aguilar 8NA

Luiz Fernando P.S. Forgas 8NA

Trabalho de Conclusão de Curso:

Técnicas de desenvolvimento de Jogos para dispositivos móveis

Pacote: jogo.rede

Classe: Servidor Descrição: Servidor que gerencia as salas do jogo on-line

Version:

1.0

Author:

Rogério de Paula Aguilar

Field Summary

`static int`	**`NUMERO_SALAS`** Quantidade de salas do servidor

Constructor Summary

`Servidor``(java.lang.String ip, int porta, double timeout, double timeoutjogo)` Construtor

Method Summary

void	**addUltimoNoSala**(javax.swing.tree.DefaultMutableTreeNode n Método utilizado para atualização da árvore
void	**atualizarArvore**() Atualiza a árvore de salas
java.lang.String	**entrarSala**(java.lang.String id, java.lang.String jogador, java.io.BufferedReader reader, java.io.PrintWriter writer java.net.Socket socket) Tenta adicionar um jogador em uma sala
jogo.rede.Sala	**getSala**(int id) Retorna uma sala pelo seu índice
double	**getTimeout**() Retorna o timeout para o início do jogo
double	**getTimeoutJogo**() Retorna o timeout para o término do jogo
void	**log**(java.lang.String msg) Adiciona uma entrada ao log
static void	**main**(java.lang.String[] args) Inicializa a execução do servidor
void	**parar**() Interrompe o servidor
javax.swing.tree.DefaultMutableTreeNode	**retornarNo**(javax.swing.JTree arvore) Retorna o nó principal da árvore de salas
java.lang.String	**retornarSalasDisponiveis**() Retorna a lista de salas disponíveis
java.lang.String	**toString**() Fornece uma representação do servidor num objeto string

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

NUMERO_SALAS

public static final int **NUMERO_SALAS**

Quantidade de salas do servidor

See Also:

Constant Field Values

Constructor Detail

Servidor

```
public Servidor(java.lang.String ip,
                int porta,
                double timeout,
                double timeoutjogo)
```

Construtor

Method Detail

atualizarArvore

```
public void atualizarArvore()
```

Atualiza a árvore de salas

---

toString

```
public java.lang.String toString()
```

Fornece uma representação do servidor num objeto string

Overrides:

`toString` in class `java.lang.Object`

---

parar

```
public void parar()
```

Interrompe o servidor

---

getSala

```
public jogo.rede.Sala getSala(int id)
```

Retorna uma sala pelo seu índice

---

retornarSalasDisponiveis

```
public java.lang.String retornarSalasDisponiveis()
```

Retorna a lista de salas disponíveis

---

entrarSala

```
public java.lang.String entrarSala(java.lang.String id,
                                   java.lang.String jogador,
                                   java.io.BufferedReader reader,
                                   java.io.PrintWriter writer,
                                   java.net.Socket socket)
```

Tenta adicionar um jogador em uma sala

---

retornarNo

```
public javax.swing.tree.DefaultMutableTreeNode
retornarNo(javax.swing.JTree arvore)
```

Retorna o nó principal da árvore de salas

Specified by:

`retornarNo` in interface `MontarArvore`

---

log

```
public void log(java.lang.String msg)
```

Adiciona uma entrada ao log

---

main

```
public static void main(java.lang.String[] args)
```

Inicializa a execução do servidor

---

addUltimoNoSala

```
public void addUltimoNoSala(javax.swing.tree.DefaultMutableTreeNode no)
```

Método utilizado para atualização da árvore

---

getTimeout

```
public double getTimeout()
```

Retorna o timeout para o início do jogo

---

getTimeoutJogo

```
public double getTimeoutJogo()
```

Retorna o timeout para o término do jogo

Apêndice B
Código do protótipo

Este apêndice apresenta o código da aplicação desenvolvida como protótipo. A pasta src do cd-rom possui o código da aplicação listado neste apêndice.

- **CLASSES DA APLICAÇÃO CLIENTE**

<u>Baralho.java</u>

/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis
Classe: jogo.Baralho
Responsabilidades: Representa o baralho

Alterações***************************
****

%%
%%
Data: 28/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criação da Classe
Status: OK
%%
%%

%%
%%
Data: 30/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Colocando imagens finais nas cartas
Status: OK
%%
%%

%%
%%
Data: 04/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Atualizando classe para exibir a carta atual
Status: OK
%%
%%

%%
%%

Data: 08/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando imagem com o verso da carta
Status: OK
%%
%%

%%
%%
Data: 11/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotinas para adicionar cartas ao baralho
Status: OK
%%
%%

%%
%%
Data: 14/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotinas para compra de cartas
Modificando rotinas de debug para utilizarem o método exibirDebugMsg da classe ControleJogo
Status: OK
%%
%%

%%
%%
Data: 19/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina que retorna o número de cartas do baralho
Status: OK
%%
%%

%%
%%
Data: 10/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando interface gráfica

Status: ok
%%
%%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Data: 22/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina para zerar o baralho

Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Data: 12/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando informação sobre o próximo naipe que deve ser jogado
                    quando a carta atual é uma valete

Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



*/


/*
@author  1.0
@param  1.0
@return  1.0
@throws  1.2
@version



*/


package jogo;

import java.util.*;
import jogo.apibasica.*;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

/**
```

```
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
<p>Classe: jogo.Baralho
<p>Responsabilidades: Representa o baralho do jogo

@author Rogério de Paula Aguilar
@version  1.0

*/


public class Baralho extends GameCanvas {

	/**
		Cartas do baralho
	*/
	public final static Carta[][] cartasBaralho = new Carta[4][14];

	/**
		Imagem do verso de uma carta
	*/

	private static Image imgVerso;

	/**
		Carta com o verso de uma carta
	*/
	public static Carta cartaVerso;

	/**
		Carta contendo o verso em miniatura
	*/
	public static Carta cartaVersoMiniatura;

	/**
		Número de cartas do baralho
	*/
	public static final byte NUMERO_CARTAS = 56;

	/**
		Número de cartas que é distribuído para cada jogador
	*/
	public static final byte CARTAS_PARA_CADA_JOGADOR = 7;

	/**
		Variável auxiliar para o desenho das setas de navegação
```

```
	*/
	protected byte deslocamentoSetaBaixo = 0;


	/**
		Cartas atuais do baralho
	*/

	private Stack cartasAtuais = new Stack();


	/**
		Tela de volta, utilizada quando o jogador seleciona a seta
		de navegação para baixo
	*/

	private Displayable telaVolta; //Tela para voltar

	static{
		try{
			Baralho.imgVerso = Image.createImage("/fundo.png");
			Baralho.cartaVerso = new Carta(imgVerso, Carta.VERSO, Carta.VERSO);
			Baralho.cartaVersoMiniatura = new
Carta(Image.createImage("/fundomini.png"), Carta.VERSO, Carta.VERSO);
		}catch(Exception e){
			System.out.println("Baralho>>static>>Erro ao carregar imagem>>" + e);
		}

	}

	/**
		Método invocado quando o jogador pressiona uma tecla na tela do baralho
	*/

	public void keyPressed(int keyCode){

		if(getKeyCode(DOWN) == keyCode){
			//Exibindo tela anterior
			ControleJogo.exibirDebugMsg("Baralho>>keyPressed>>Voltando à tela
anterior");
			deslocamentoSetaBaixo = 2;
		}
		repaint();

	}

	/**
```

```
		Método invocado quando o jogador solta uma tecla na tela do baralho
	*/

	public void keyReleased(int keyCode){
		if(getKeyCode(DOWN) == keyCode){
			//Exibindo tela anterior
			deslocamentoSetaBaixo = 0;
			repaint();
			CartasMidlet.getDisplay().setCurrent(telaVolta);
		}
		repaint();

	}

	/**
		Construtor padrão
	*/

	public Baralho(){
		super(false);


		Image img = null;

		try{
		  img = Image.createImage("/cartas.png");

		}catch(Exception e){
		  System.out.println("BARALHO>>Baralho()>>ERRO AO CARREGAR
IMAGEM PRINCIPAL DAS CARTAS");

		}



		//Inicializando cartas
		for(byte i = Carta.PAUS; i <= Carta.OUROS; i++){
			for(byte j = Carta.CORINGA; j <= Carta.AS; j++){

				try{


					cartasBaralho[i][j] = new Carta(
						Image.createImage(img, j *
Carta.LARGURA_CARTA, i * Carta.ALTURA_CARTA, Carta.LARGURA_CARTA,
Carta.ALTURA_CARTA, 0),
						i, j
```

```
                                        );

        ControleJogo.exibirDebugMsg(cartasBaralho[i][j].toString());

                                        }catch(Exception e){
                                                System.out.println("BARALHO>>Baralho()>>ERRO AO
CARREGAR IMAGEM");

                                        }

                                        cartasAtuais.push(cartasBaralho[i][j]);
                                }
                        }

                        if(ControleJogo.DEBUG_MODE){
                                ControleJogo.exibirDebugMsg("BARALHO>>Baralho()>>Cartas Atuais:
" + cartasAtuais.size());
                                imprimirCartasAtuais();
                        }
                        //embaralhar();
                        System.gc();

        }

        /**
                Imprime as cartas atuais
        */
        public void imprimirCartasAtuais(){
                System.out.println("*****Baralho>>imprimirCartasAtuais*****");
                for(int i =0; i < cartasAtuais.size(); i++){
                        Carta carta = (Carta)cartasAtuais.elementAt(i);
                        System.out.println(carta);
                }
                System.out.println("****************************************");
                System.gc();
        }

        /**
                Embaralha as cartas atuais
        */

        public  void embaralhar(){

                cartasAtuais = new Stack();
                Random r = new Random();

                while(cartasAtuais.size() != NUMERO_CARTAS){
```

```
                int proximoItemX = Math.abs(r.nextInt() % 4);
                int proximoItemY = Math.abs(r.nextInt() % 14);
                Carta carta = cartasBaralho[proximoItemX][proximoItemY];
                if(!(cartasAtuais.contains(carta))){
                        cartasAtuais.push(carta);
                }

        }

        if(ControleJogo.DEBUG_MODE){
                ControleJogo.exibirDebugMsg("*****Baralho>>Embaralhar*****");
                imprimirCartasAtuais();
                ControleJogo.exibirDebugMsg("*****************************");
        }

        System.gc();
    }

    /**
        Reseta o baralho
    */
    public void resetar(){
        ControleJogo.exibirDebugMsg("Baralho>>resetar()");

        cartasAtuais = new Stack();

        for(byte i = Carta.PAUS; i <= Carta.OUROS; i++){
            for(byte j = Carta.CORINGA; j <= Carta.AS; j++){

                    cartasAtuais.push(cartasBaralho[i][j]);
            }
        }


        System.gc();
    }


    /**
        Distribui as cartas para os jogadores
    */

    public void distribuirCartas(Jogador[] jogadores){
        ControleJogo.exibirDebugMsg("Baralho>>distribuirCartas()");
        if(jogadores.length != ControleJogo.NUMERO_JOGADORES){
            ControleJogo.exibirDebugMsg("Baralho>>distribuirCartas>>Nº de
jogadores inválido>>" + ControleJogo.NUMERO_JOGADORES);
```

```
                throw new IllegalArgumentException("Número de joadores inválido!");
        }

        ControleJogo.exibirDebugMsg("Baralho>>distribuirCartas()>>Começando a
distribuir cartas");

        //Distribuindo cartas aos jogadores
        for(byte i = 0; i < ControleJogo.NUMERO_JOGADORES; i++){
                for(byte j = 0; j < CARTAS_PARA_CADA_JOGADOR; j++){
                        Carta carta = null;
                        try{
                                carta = (Carta) cartasAtuais.pop();
                        }catch(EmptyStackException e){

    ControleJogo.exibirDebugMsg("Baralho>>distribuirCartas>>Impossível distribuir cartas,
baralho está vazio");
                        }
                        if(carta != null){
                                jogadores[i].adicionarCarta(carta);
                        }else{

    ControleJogo.exibirDebugMsg("Baralho>>distribuirCartas>>carta = null");
                        }
                }
        }
        ControleJogo.exibirDebugMsg("Baralho>>distribuirCartas()>>Cartas distribuídas
com sucesso");

    }

    /**
        Retorna a carta no topo do baralho

    */

    public Carta cartaAtual(){
        Carta carta = null;
        try{
                carta = (Carta) cartasAtuais.peek();
        }catch(EmptyStackException e){
                ControleJogo.exibirDebugMsg("Baralho>>cartaAtual>>baraalho está
vazio");
                throw e;
        }

        return carta;
```

```
	}

	/**
		Desenha a interface do baralho
	*/

	public void  paint(Graphics g){
		g.setColor(0, 190, 0);
		g.fillRect(0, 0, Dispositivo.getLarguraTela(), Dispositivo.getAlturaTela());

		try{
			Carta carta = cartaVerso;

			for(byte n = 0; n < 5; n++){
				carta.setPosition(((Dispositivo.getLarguraTela() - (
(carta.getWidth() * 2) + 10) ) / 2) + n,
					(Dispositivo.getAlturaTela() - carta.getHeight()) / 2 + n);

				carta.paint(g);
			}

			if(!(cartasAtuais.isEmpty())){
				g.setColor(0, 0, 255);

				g.drawString("Carta Atual" , (Dispositivo.getLarguraTela() -
Font.getDefaultFont().stringWidth("Carta Atual")) / 2, 0, 0);

				g.setColor(255,255,255);
				g.drawString(cartaAtual().getNaipeStr(),
					(Dispositivo.getLarguraTela() -
Font.getDefaultFont().stringWidth(cartaAtual().getNaipeStr())) / 2
					, Font.getDefaultFont().getHeight(), 0);
				g.drawString(cartaAtual().getValorStr(),
					(Dispositivo.getLarguraTela() -
Font.getDefaultFont().stringWidth(cartaAtual().getValorStr())) / 2

					, Font.getDefaultFont().getHeight() * 2, 0);

				if(cartaAtual().getValor() == Carta.VALETE){
					Ticker tk = getTicker();
					String naipeProximaJogada = "PRÓXIMO NAIPE:";

					switch(ControleJogo.NAIPE_ESCOLHIDO){
						case Carta.COPAS:
							naipeProximaJogada += "COPAS";
							break;
						case Carta.ESPADAS:
```

```
                                        naipeProximaJogada += "ESPADAS";
                                        break;
                                case Carta.PAUS:
                                        naipeProximaJogada += "PAUS";
                                        break;
                                case Carta.OUROS:
                                        naipeProximaJogada += "OUROS";
                                        break;
                        }

                        g.drawString(naipeProximaJogada,
                           (Dispositivo.getLarguraTela() -
Font.getDefaultFont().stringWidth(naipeProximaJogada)) / 2

                                , Font.getDefaultFont().getHeight() * 3, 0);


                        }
                }


                carta = cartaAtual();
                carta.setPosition( ((Dispositivo.getLarguraTela() - ( (carta.getWidth() * 2)
+ 10) ) / 2) + carta.getWidth() + 10,
                                (Dispositivo.getAlturaTela() - carta.getHeight()) / 2);
                carta.paint(g);

                //Seta de cima apontando para baixo (próximo jogador)
                Jogador.imgSeta.setTransform(Sprite.TRANS_ROT270);
                Jogador.imgSeta.setPosition( Dispositivo.getCentroX() ,
Dispositivo.getAlturaTela() - Jogador.imgSeta.getHeight() - 2 + deslocamentoSetaBaixo);
                Jogador.imgSeta.paint(g);
                //Boneco
                Jogador.imgJogador.setPosition( Dispositivo.getCentroX() -
Jogador.imgJogador.getWidth() , Dispositivo.getAlturaTela() - Jogador.imgJogador.getHeight() -
12);
                Jogador.imgJogador.paint(g);


        }catch(Exception e){
                System.out.println("ERRO: " + e);
        }
    }

    /**
        Muda a tela de volta
    */
```

```
	public void setTelaVolta(Displayable d){
		telaVolta = d;
	}


	/**
		Adiciona uma carta ao baralho
	*/
	public void adicionarCarta(Carta carta){
		cartasAtuais.push(carta);
		repaint();
	}

	/**
		Retorna uma carta para compra
	*/
	public Carta comprarCarta() throws EmptyStackException{
		ControleJogo.exibirDebugMsg("Baralho>>comprarCarta()>>retornando carta");
		Carta carta = null;
		if(cartasAtuais.size() >= 3){
			int indice = Math.abs((new Random().nextInt() % (cartasAtuais.size() -
1)));
			ControleJogo.exibirDebugMsg("Baralho>>comprarCarta()>>ìndice: " +
indice);
			carta = (Carta)cartasAtuais.elementAt(indice);
			cartasAtuais.removeElementAt(indice);
		}else if(cartasAtuais.size() == 2){
			carta = (Carta)cartasAtuais.elementAt(cartasAtuais.size()-2);
			cartasAtuais.removeElementAt(cartasAtuais.size()-2);
		}else{
			carta = ((Carta)cartasAtuais.pop());
		}
		return carta;

	}

	/**
		Retorna a quantidade de cartas que o baralho possui atualmente
	*/

	public int numeroCartas(){
		return cartasAtuais.size();
	}

	/**
		Retorna uma carta do baralho pelo seu índice
```

```
	*/

	public Carta carta(int indice){
		if(indice < 0 || indice > cartasAtuais.size())
			throw new IllegalArgumentException("Baralho>>carta>>Índice
inválido!");
		return (Carta)cartasAtuais.elementAt(indice);
	}


	/**
		Deixa o baralho sem nenhuma carta
	*/

	public void zerar(){
		ControleJogo.exibirDebugMsg("Baralho>>zerar()");

		cartasAtuais = new Stack();

		System.gc();
	}


}
```

__Carta.java__

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.Carta
Responsabilidades: Representa uma carta do baralho

****************************************Alterações****************************************


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 28/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criação da Classe
Status: OK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
```

%%%
%%%
%%
Data: 30/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Colocando imagens finais nas cartas
Status: OK
%%%
%%%
%%

%%%
%%%
%%
Data: 08/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Alterando classe para exibir as informações sobre o verso
Status: OK
%%%
%%%
%%

%%%
%%%
%%
Data: 14/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Modificando rotinas de debug para utilizarem o método exibirDebugMsg da classe ControleJogo
Status: OK
%%%
%%%
%%

%%%
%%%
%%
Data: 18/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando constantes

Status: OK

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%




*/
package jogo;

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;


/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.Carta
<p>Responsabilidades: Representa uma carta do baralho

@author Rogério de Paula Aguilar
@version  1.0

*/

public class Carta extends Sprite{
	/*
		OBS: Não modificar a ordem dos naipes e/ou cartas, pois
		a imagem das cartas será indexada segundo esta ordem.
		Se a ordem for modificada, a imagem deve ser modificada
		para refletir os novos índices.
	*/


	/**
		Naipes
	*/

	public final static byte PAUS = 0, COPAS = 1, ESPADAS = 2, OUROS = 3;

	/**
		Largura da imagem da carta
	*/
	public final static byte LARGURA_CARTA = 40;
```

```
	/**
		Altura da imagem da carta
	*/

	public final static  byte ALTURA_CARTA = 54;

	/*
		PAUS --> "árvore"
		COPAS --> Coração

	*/

	//valores das cartas --> Não mude aqui ! (utilizado como índice de vetores na classe baralho

	/**
		Valores das cartas
	*/

	public final static byte CORINGA = 0, DOIS = 1, TRES = 2, QUATRO = 3,
	CINCO = 4, SEIS = 5, SETE = 6, OITO = 7, NOVE = 8, DEZ = 9, DAMA = 10,
	VALETE = 11, REI = 12, AS = 13, VERSO = 14;


	/**
		Variáveis utilizadas em loops
	*/
	public final static byte PRIMEIRO_VALOR = CORINGA,
				ULTIMO_VALOR = AS;
	public final static byte PRIMEIRO_NAIPE = PAUS,
				ULTIMO_NAIPE = OUROS;

	//naipe e valor da instância atual
	/**
		Largura da imagem da carta
	*/

	/**
		Naipe da carta atual
	*/
	private byte naipe;

	/**
		Valor da imagem da carta
	*/
	private byte valor;
```

```
/**
	Construtor
*/

public Carta(Image imagem, byte naipe, byte valor){
	super(imagem);
	this.naipe = naipe;
	this.valor = valor;


}

/**
	Modifica o naipe da carta
*/

public void setNaipe(byte naipe){
	this.naipe = naipe;
}

/**
	Modifica o valor da carta
*/

public void setValor(byte valor){
	this.valor = valor;
}

/**
	Retorna o naipe da carta
*/
public byte getNaipe(){
	return naipe;
}

/**
	Retorna o valor da carta
*/
public byte getValor(){
	return valor;
}

/**
	Retorna uma representação do naipe da carta num objeto String
*/
```

```
public String getNaipeStr(){
        StringBuffer str = new StringBuffer(" Naipe: ");

        switch(naipe){
                case PAUS:
                        str.append("PAUS");
                        break;
                case COPAS:
                        str.append("COPAS");
                        break;
                case ESPADAS:
                        str.append("ESPADAS");
                        break;
                case OUROS:
                        str.append("OUROS");
                        break;
                case VERSO:
                        str.append("VERSO");
                        break;
                default:
                        str.append("INDEFINIDO");

        }

        return str.toString().trim();

}

/**
        Retorna uma representação do valor da carta num objeto String
*/
public String getValorStr(){
        StringBuffer str = new StringBuffer(" Valor: ");

        switch(valor){
                case CORINGA:
                        str.append("CORINGA");
                        break;
                case DOIS:
                        str.append("DOIS");
                        break;
                case TRES:
                        str.append("TRES");
                        break;
                case QUATRO:
                        str.append("QUATRO");
                        break;
```

```
                case CINCO:
                        str.append("CINCO");
                        break;
                case SEIS:
                        str.append("SEIS");
                        break;
                case SETE:
                        str.append("SETE");
                        break;
                case OITO:
                        str.append("OITO");
                        break;
                case NOVE:
                        str.append("NOVE");
                        break;
                case DEZ:
                        str.append("DEZ");
                        break;
                case DAMA:
                        str.append("DAMA");
                        break;
                case VALETE:
                        str.append("VALETE");
                        break;
                case REI:
                        str.append("REI");
                        break;
                case AS:
                        str.append("AS");
                        break;
                default:
                        str.append("INDEFINIDO");

        }

        return str.toString().trim();

}

/**
        Retorna uma representação da carta num objeto String
*/
public String toString(){
        StringBuffer str = new StringBuffer("Carta--> Naipe: ");

        switch(naipe){
                case PAUS:
```

```
                str.append("PAUS");
                break;
        case COPAS:
                str.append("COPAS");
                break;
        case ESPADAS:
                str.append("ESPADAS");
                break;
        case OUROS:
                str.append("OUROS");
                break;
        case VERSO:
                str.append("VERSO");
                break;
        default:
                str.append("INDEFINIDO");

}

str.append(" Valor: ");

switch(valor){
        case CORINGA:
                str.append("CORINGA");
                break;
        case DOIS:
                str.append("DOIS");
                break;
        case TRES:
                str.append("TRES");
                break;
        case QUATRO:
                str.append("QUATRO");
                break;
        case CINCO:
                str.append("CINCO");
                break;
        case SEIS:
                str.append("SEIS");
                break;
        case SETE:
                str.append("SETE");
                break;
        case OITO:
                str.append("OITO");
                break;
        case NOVE:
```

```
                        str.append("NOVE");
                        break;
                case DEZ:
                        str.append("DEZ");
                        break;
                case DAMA:
                        str.append("DAMA");
                        break;
                case VALETE:
                        str.append("VALETE");
                        break;
                case REI:
                        str.append("REI");
                        break;
                case AS:
                        str.append("AS");
                        break;
                case VERSO:
                        str.append("VERSO");
                        break;
                default:
                        str.append("INDEFINIDO");

        }

        return str.toString().trim();
    }

    /**
        Verifica a igualdade entre duas instâncias da classe
    */
    public boolean equals(Object obj){
        if(! (obj instanceof Carta))
                return false;

        else
                return (this.naipe == ((Carta)obj).naipe)
                && (this.valor == ((Carta)obj).valor);
    }



}
```

CartasMidlet.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos móveis
Classe: jogo.CartasMidlet
Responsabilidades: Classe principal, que inicializa os objetos de controle do jogo

**********Alterações**********

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Data: 22/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementação do formulário de ajuda e término da apresentação
Status: pendente (falta terminar o formulário de ajuda)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Data: 04/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: mudando display para ser estático e adicionando o método getDisplay
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



*/

package jogo;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import jogo.apibasica.*;


/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
<p>Classe: jogo.CartasMidlet
<p>Responsabilidades: Classe principal, que inicializa os objetos de controle do jogo
```

```
@author Rogério de Paula Aguilar
@version 1.0
*/


public class CartasMidlet extends MIDlet {

	/**
		Objeto contendo as características do dispositivo
		em que o jogo está sendo executado
	*/
	private Dispositivo dispositivo;

	/**
		Display do jogo
	*/
	private static Display display;
	/**
		Objeto que controla a execução do jogo
	*/
	private ControleJogo controleJogo;


	/**
		Inicializa o MIDlet
	*/

	public void startApp(){
		if(ControleJogo.DEBUG_MODE){
			System.out.println("CartasMidlet>>startAPP>>Endereço do Servidor: " +
getAppProperty("IP_SERVIDOR"));
			System.out.println("CartasMidlet>>startAPP>>Porta do Servidor: " +
getAppProperty("PORTA_SERVIDOR"));

		}
		display = Display.getDisplay(this);
		dispositivo = new Dispositivo(display);
		controleJogo = new ControleJogo(true, dispositivo, display, this,
getAppProperty("IP_SERVIDOR"), Integer.parseInt(getAppProperty("PORTA_SERVIDOR")));
		controleJogo.setIPServidor(getAppProperty("IP_SERVIDOR"));
		controleJogo.setPortaServidor(getAppProperty("PORTA_SERVIDOR"));

	}

	public void pauseApp(){}
```

```
	/**
		Sai da aplicação
	*/

	public void destroyApp(boolean b){
		System.gc();
		controleJogo.parar();


		if(ControleJogo.DEBUG_MODE)
			System.out.println("CartasMidlet>>destroyApp>>Saindo da aplicação...");

		notifyDestroyed();
	}

	/**
		Retorna o display do jogo
	*/
	public static Display getDisplay(){
		return display;
	}


}
```

<u>ControleJogo.java</u>

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.apibasica.ControleJogo
Responsabilidades: Controlar a execução do jogo e a atualização das telas de
apresentação
			e principal do mesmo. Controla o loop principal do jogo e a
navegabilidade
			da aplicação.


***********************************Alterações***********************************


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 09/08/2003
```

Responsável: Rogérrio de Paula Aguilar
Descrição: término da primeira parte da apresentação do jogo, adição de comandos
principais do jogo
Status: alterações OK
%%
%%
%%

%%
%%
%%
Data: 22/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementação do formulário de ajuda e término da apresentação
Status: pendente (falta terminar o formulário de ajuda)
%%
%%
%%

%%
%%
%%
Data: 30/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Associando imagens às cartas
Status: OK
%%
%%
%%

%%
%%
%%
Data: 05/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando métodos para criação do modo OpenMauMau
, atualizando tipos de jogador, adicionando formulário de opções
Status: OK
%%
%%
%%

%%
%%
%%

Data: 11/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina de verificação de jogada e classe Mensagem
Status: OK
%%
%%
%%

%%
%%
%%
Data: 14/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Continuando rotina de verificação de jogada e classe Mensagem
Adicionando rotina que verifica o vencedor do jogo
Modificando rotinas de debug para utilizarem o método exibirDebugMsg da classe ControleJogo
Adicionando variável que controla o número de cartas que o jogador deve comprar

Status:OK
%%
%%
%%

%%
%%
%%
Data: 15/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina que retorna o índice do próximo jogador
Adicionando rotina que retorna o índice do jogador anterior
Status:OK
%%
%%
%%

%%
%%
%%
Data: 19/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementando rotina que elege o vencedor do jogo
Testendo e corrigindo rotina verificarJogada
Status:OK

%%
%%
%%

%%
%%
%%
Data: 09/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Testando rotina de verificação de jogadas
Alterando mensagens para se tornarem mais claras para o usuário
Status:OK
%%
%%
%%

%%
%%
%%
Data: 10/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando interface gráfica
Criando variável JOGO_EM_ANDAMENTO para controlar
erros de sincronismo entre as threads de mensagens
IMPORTANTE: TODOS OS FORMULÁRIOS QUE VOLTAREM PARA ESA
TELA
DEVEM SETAR A VARIÁVEL JOGO_EM_ANDAMENTO PARA
FALSE,
PARA QUE A THREAD DE INÍCIO SEJA EXIBIDA
CORRETAMENTOE

Status: ok
%%
%%
%%

%%
%%
%%
Data: 13/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criando opção de jogo on-line
Status: ok

%%%
%%%
%%

%%%
%%%
%%
Data: 15/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Modificando protocolo (http para socket)
Status: ok
%%%
%%%
%%

%%%
%%%
%%
Data: 20/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina de jogo on-line
Status: ok
%%%
%%%
%%

%%%
%%%
%%
Data: 21/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina de jogo on-line
Status: ok
%%%
%%%
%%

%%%
%%%
%%
Data: 22/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina de jogo on-line
Adicionando rotina que trata da atualização do jogo, quando
o quadro com o status do jogo chega do servidor
Status: ok

%%
%%
%%

%%
%%
%%
Data: 26/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Terminando rotina de tratamento de quadros
Status: ok
%%
%%
%%

%%
%%
%%
Data: 26/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Refazendo rotina de jogada do usuário, para trocar mensagens e
corrigir problemas
Status: ok
%%
%%
%%

%%
%%
%%
Data: 05/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando tela de apresentação
Criando formulário de ajuda
Status: ok
%%
%%
%%

%%
%%
%%
Data: 12/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando problema da thread de jogada do computador
Status: ok

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 28/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando thread que fica enviando Acks para o servidor
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/
package jogo.apibasica;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.game.*;
import java.util.*;
import java.io.*;
import javax.microedition.io.*;
import jogo.*;


/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.apibasica.ControleJogo
<p>Responsabilidades: Controlar a execução do jogo e a atualização das telas de
apresentação
                    e principal do mesmo. Controla o loop principal do jogo e a
navegabilidade
                    da aplicação.

@author Rogério de Paula Aguilar
@version 1.0

*/

public class ControleJogo extends GameCanvas implements Runnable,
CommandListener{
```

```
	/**
		Indica se o modo debug está ativo ou não
	*/
	public final static boolean DEBUG_MODE = true;


	/**
		Indica se o jogador está no meio de uma partida
	*/

	public static boolean JOGO_EM_ANDAMENTO = false;


	/**
		Indica o índice do jogo atual contra o computador
	*/

	private int jogoAtual = 0;

	/**
		Indica se o jogador está conectado ao servidor
	*/

	public static boolean CONECTADO;


	/**
		Estado do jogo
	*/

	public static boolean EXIBIR_INFORMACAO_CARTA = false;
	public static boolean APRESENTACAO, ONLINE, TELASELECAO; //Estados do
jogo

	/**
		Tempo de espera para atualização da tela de jogo
	*/

	private byte DELAY = 100; //Tempo de espera para atualização da tela

	/**
		Objeto que contém as características do dispositivo onde o jogo está
		sendo executado
	*/

	private Dispositivo dispositivo;
```

```
/**
	Disply do jogo
*/

private Display display;
private boolean JOGANDO;

/**
	Variáveis que controlam o sentido do jogo
*/
public final static byte SENTIDO_NORMAL = 1, SENTIDO_INVERSO = -1;
public static byte SENTIDO_JOGO = SENTIDO_NORMAL;

/**
	Indica se a penalidade de uma carta hostil já foi aplicada ou não
*/
public static boolean PENALIDADE_APLICADA = false;

/**
	Quantidade de cartas para compra
*/
private static byte QUANT_COMPRA = 0;

/**
	Variável auxiliar para o desenho da tela de apresentação
*/
private int x_src = 5;

/**
	Imagem contando o logo do jogo
*/
private Sprite imgLogo; //Carta

/**
	Imagem contando o logo do jogo de forma escritaa
*/
private Image imgLogoEscrita;//Logo escrito

/**
	Formulário de ajuda
*/

private frmAjuda formularioAjuda; //Formulário de ajuda

/**
	Formulário de visualização de cartas
*/
```

```
        private frmVisualizadorCartas visualizadorCartas; //Visualização das cartas

        /**
                Formulário de Opções
        */
        private frmOpcoes frmopcoes; //Opções

        /**
                Formulário contendo a lista de salas (utilizado no jogo on-line)
        */
        private frmListaSalas frmlistasalas; //Lista de salas

        /**     Formulário de espera
        */
        public static frmEspera frmespera = new frmEspera("Aguarde comunicação com
o servidor...");

        /**
                Formulário de escolha de naipe (utilizado na jogada da valete)
        */
        private frmEscolhaNaipe frmescolhanaipe;

        /**
                Comando do menu principal
        */
        private Command cmdSair = new Command("Sair", Command.EXIT, 2);

        /**
                Comando do menu principal
        */
        private Command cmdIniciarJogo = new Command("Jogar", Command.SCREEN,
0);

        /**
                Comando do menu principal
        */
        private Command cmdIniciarJogoOnLine = new Command("Jogo on-line",
Command.SCREEN, 0);

        /**
                Comando do menu principal
        */
        private Command cmdOpcoes = new Command("Opções",
Command.SCREEN,0);

        /**
                Comando do menu principal
```

```
	*/
	private Command cmdAjuda = new Command("Ajuda", Command.SCREEN,0);

	/**
		Comando do menu principal
	*/
	private Command cmdVisualizadorCartas = new Command("Visualizador de
Cartas", Command.SCREEN,0);


	/**
		Midlet principal do jogo
	*/
	private CartasMidlet cartasMidlet;



	/**
		Variável auxiliar para desenho de cartas
	*/
	private int posicaoCartaX, posicaoCartaY;

	/**
		Variável auxiliar para desenho de cartas
	*/
	private byte direcao = 1;


	/**
		Baralho do jogo
	*/
	private Baralho baralhoAtual = new Baralho();
	private byte naipeAtual = 0, cartaAtual = 0;

	/**
		Número de jogadores
	*/
	public static int NUMERO_JOGADORES = 4;

	/**
		Jogadores
	*/
	Jogador jogadores[] = new Jogador[NUMERO_JOGADORES];

	/**
		Índice do jogador atual
	*/
```

```
private byte jogadorAtual;

/**
	Índice do jogador que está sendo exibido
*/
private byte jogadorExibido;


/**
	Indica se um jogador pode ver as cartas do outro
*/
private static boolean OPEN_MAU_MAU = false;

/**
	Indica se o jogador anterior foi pulado
*/
public boolean ANTERIOR_PULADO = false;

/**
	Gerador de números aleatórios
*/

private static Random random = new Random();


/**
	Objeto utilizado para exibir mensagens de erro

*/
private Mensagem msgErro;


/**
	Objeto que guarda a conexão com o servidor
*/
public static SocketConnection conexao;

/**
	Objeto utilizado para ler comandos que vem do servidor
*/
public static InputStream is;

/**
	Objeto utilizado para enviar mensagens para o servidor
*/
public static OutputStream os;
```

```
/**
	Objeto utilizado para enviar mensagens para o servidor
*/
public static PrintStream socketWriter;

/**
	Objeto utilizado para guardar os nomes dos jogadores
*/
public static String[] nomeJogadores = new String[NUMERO_JOGADORES];

/**
	Indica se o jogador é o primeiro
*/
public static boolean primeiroJogador;

/**
	Objeto que guarda o id da sala que o usuário está conectado
*/
public static String idSala = "";

/**
	Objeto  que guarda o nome do jogador atual
*/
public static String nomeJogador = "";

/**
	Guarda o índice do jogador na rede
*/
public static int indiceJogadorRede;

/**
	Indica se o usuário já recebeu o primeiro status do servidor
*/
private boolean primeiroStatusRecebido = false;

/**
	Indica se o jogador recebeu permissão para jogar
*/
private boolean RECEBEU_PERMISSAO_JOGADA;

/**
	Endereço do servidor
*/
private static String IP_SERVIDOR, PORTA_SERVIDOR;


private static String comando = "";
```

```
	/**
		Guarda o naipe escolhido para a próxima jogada (Jogada: Valete(*))
	*/
	public static byte NAIPE_ESCOLHIDO = 0;

	/**
		Listeners para formulários de espera
	*/
	private static CommandListener listenerOK, listenerEntradaJogador;

	/**
		Formulário de espera
	*/
	private static frmEspera frm;

	/**
		Comando cancelar do formulário de espera
	*/
	private static Command cmdCancelar = new Command("Cancelar",
Command.SCREEN, 0);

	/**
		Comando de começo de jogo do formulário de espera
	*/
	private static Command cmdComecarJogo = new Command("Começar jogo",
Command.SCREEN, 0);



	/**
		Thread que fica mandando acks para o servidor
	*/

	private static class threadAck implements Runnable{

		private ControleJogo controleJogo;

		public threadAck(ControleJogo controleJogo){
			this.controleJogo = controleJogo;
		}

		public void run(){
			while(CONECTADO){
				try{
					synchronized(controleJogo){
```

```
                                        exibirDebugMsg("Enviando ack");
                                        socketWriter.println();
                                        socketWriter.flush();

                                }
                                Thread.sleep(25000); //25 segundos
                        }catch(Exception e){
                                exibirDebugMsg("Erro ao enviar ack: " + e);
                        }
                }

                exibirDebugMsg("Encerrando thread de envios de acks");
        }

    }

    private static Thread tAck;

    /**
        Retorna o gerador de números aleatórios
    */
    public static int getNumeroAleatorio(){
        return random.nextInt();

    }

    public synchronized void setRecebeuPermissaoJogada(boolean b){
        RECEBEU_PERMISSAO_JOGADA = b;
    }

    public synchronized boolean getRecebeuPermissaoJogada(){
        return RECEBEU_PERMISSAO_JOGADA ;
    }

    /**
        Construtor
    */

    public ControleJogo(boolean s, Dispositivo d, Display display, CartasMidlet
cartasMidlet, String ip, int porta){


        super(s);

        dispositivo = d;
        this.display = display;
        JOGANDO = true;
```

```
                APRESENTACAO = true;
                TELASELECAO = false;
                try{

                        imgLogo = new Sprite(Image.createImage("/logo.png"));
                          imgLogoEscrita = Image.createImage("/logoEscr.png");
                          imgLogo = baralhoAtual.cartasBaralho[0][0];
                }catch(Exception e){
                         System.out.println("ERRO ao carregar imagem (ControleJogo): " +
e);

                }


                posicaoCartaX = -imgLogo.getWidth();

                formularioAjuda = new frmAjuda("Ajuda", this, display);
                visualizadorCartas = new frmVisualizadorCartas("Visualizador de
Cartas",this, display, baralhoAtual, d);
                frmopcoes = new frmOpcoes("Opções", this);

                display.setCurrent(this);
                new Thread(this).start();
                this.IP_SERVIDOR = ip;
                this.PORTA_SERVIDOR = "" + porta;
                //atualizarTela();

                this.cartasMidlet = cartasMidlet;

                //Adicionando comandos
                addCommand(cmdIniciarJogo);
                addCommand(cmdIniciarJogoOnLine);
                addCommand(cmdVisualizadorCartas);
                addCommand(cmdOpcoes);
                addCommand(cmdAjuda);
                addCommand(cmdSair);
                setCommandListener(this);




        }

        /**
                Loop do jogo (descontinuado)
        */
```

```
	public void mainLoop(){

	}

	/**
		Desenha a tela de apresentação
	*/

	public void atualizarTela(){
		Graphics g = getGraphics();
		g.setColor(0, 190, 0);
		g.fillRect(0,0,dispositivo.getLarguraTela(), dispositivo.getAlturaTela());
		int j = 220;
		x_src = 5;
		if(APRESENTACAO){//Exibindo apresentação do jogo
			//Exibindo logo
			for(int i = -imgLogo.getWidth(), i2 = dispositivo.getLarguraTela(); i <
dispositivo.getLarguraTela() + 10; i+=5, i2-=5){
				imgLogo.setPosition(i, (dispositivo.getAlturaTela()/2) -
imgLogo.getHeight());
				g.fillRect(0 ,0 ,dispositivo.getLarguraTela(),
dispositivo.getAlturaTela());
				if(j - 2 >= 190) j-=2;
				g.setColor(0, j, 0);
				imgLogo.paint(g);

				if(imgLogo.getX() > dispositivo.getCentroX() -
imgLogoEscrita.getWidth() && imgLogo.getX() <= dispositivo.getCentroX()){
				  //g.drawImage(imgLogoEscrita, dispositivo.getCentroX() -
imgLogoEscrita.getWidth(), (( imgLogo.getHeight() + imgLogo.getY() -
imgLogoEscrita.getHeight() - 12)),0);

				  //System.out.println("" + (imgLogo.getX() -
(dispositivo.getCentroX() - imgLogoEscrita.getWidth())));

				  g.drawRegion(imgLogoEscrita, 0, 0,
						imgLogo.getX() - (dispositivo.getCentroX() -
imgLogoEscrita.getWidth()),
						imgLogoEscrita.getHeight(), 0,
						dispositivo.getCentroX() -
imgLogoEscrita.getWidth(),
						(( imgLogo.getHeight() + imgLogo.getY() -
imgLogoEscrita.getHeight() - 12)), 0);


				}else{
```

```
                                    if(imgLogo.getX() > dispositivo.getCentroX()){
                                        imgLogo.setPosition(i,
(dispositivo.getAlturaTela()/2) - imgLogo.getHeight());
                                        g.drawImage(imgLogoEscrita,
dispositivo.getCentroX() - imgLogoEscrita.getWidth(), (( imgLogo.getHeight() +
imgLogo.getY() - imgLogoEscrita.getHeight() - 12)),0);
                                    }
                                }

                                imgLogo.setPosition(i2 , (dispositivo.getAlturaTela()/2));


                                if(imgLogo.getX() + imgLogo.getWidth() <
dispositivo.getCentroX() -10 + imgLogoEscrita.getWidth() && imgLogo.getX() +
imgLogo.getWidth() >= dispositivo.getCentroX() -10){
                                    g.drawRegion(imgLogoEscrita, //Image
                                            imgLogoEscrita.getWidth() - x_src, //x_src
                                            0, //y_src
                                            x_src, //width
                                            imgLogoEscrita.getHeight(), //height
                                            0, //transform
                                            dispositivo.getCentroX() - 10 +
imgLogoEscrita.getWidth() - x_src, //x_dest
                                            (( imgLogo.getHeight() + imgLogo.getY() -
imgLogoEscrita.getHeight() - 12)), //y_dest
                                            0); //anchor
                                    x_src+=5;


                                }else{
                                    if(imgLogo.getX() < dispositivo.getCentroX() - 10){
                                        g.drawImage(imgLogoEscrita,
dispositivo.getCentroX() - 10, (( imgLogo.getHeight() + imgLogo.getY() -
imgLogoEscrita.getHeight() - 12)),0);
                                    }

                                }

                                imgLogo.paint(g);

                                flushGraphics();
                                try{
                                    Thread.sleep(DELAY - 39);
                                }catch(Exception e){}
                            }
                            APRESENTACAO = false;
                            TELASELECAO = true;
```

```
            }else{
                    if(TELASELECAO){
                            g.fillRect(0 ,0 ,dispositivo.getLarguraTela(),
dispositivo.getAlturaTela());
                            //atualizando posição da carta que fica passando
                            imgLogo.setPosition(posicaoCartaX, posicaoCartaY);
                            posicaoCartaX += 10 * direcao;
                            imgLogo.paint(g);
                            if( (posicaoCartaX >= dispositivo.getLarguraTela()) ||
(posicaoCartaX <= -imgLogo.getWidth())){
                                    posicaoCartaY += imgLogo.getHeight();
                                    //posicaoCartaX = -imgLogo.getWidth();
                                    direcao = (byte)-direcao;
                                    naipeAtual++;
                                    if(naipeAtual == 4) naipeAtual = 0;
                                    cartaAtual++;
                                    if(cartaAtual == 14) cartaAtual = 0;
                                    try{
                                            imgLogo =
baralhoAtual.cartasBaralho[naipeAtual][cartaAtual];
                                    }catch(Exception e){


        System.out.println("ControleJogo>>TELASELECAO>>ERRO>>Imagem não
pode ser nula");
                                    }
                            }

                            if(posicaoCartaY >= dispositivo.getAlturaTela()){
                                    posicaoCartaY = 0;

                            }
                            imgLogo.setPosition(0, (dispositivo.getAlturaTela()/2) -
imgLogo.getHeight());
                            g.drawImage(imgLogoEscrita, dispositivo.getCentroX() -
imgLogoEscrita.getWidth(), (( imgLogo.getHeight() + imgLogo.getY() -
imgLogoEscrita.getHeight() - 12)),0);
                    imgLogo.setPosition(0 , (dispositivo.getAlturaTela()/2) );
                            g.drawImage(imgLogoEscrita, dispositivo.getCentroX() -10, ((
imgLogo.getHeight() + imgLogo.getY() - imgLogoEscrita.getHeight() - 12)),0);

                    }

            }
```

```
		System.gc();
		flushGraphics();

	}

	/**
		Loop de atualização da tela de apresentação do jogo
	*/
	public void run(){
		while(JOGANDO){

			atualizarTela();

			try{
				Thread.sleep(DELAY);
			}catch(Exception e){
				System.out.println("ERRO: " + e);
			}


		}

	}

	/**
		Para a execução do jogo

	*/
	public void parar(){
		exibirDebugMsg("ControleJogo>>parar>>Parando a execução do jogo");
		JOGANDO = false;
		JOGO_EM_ANDAMENTO = false;
		fecharConexao();
		System.gc();
	}


	/**
		Listener de eventos
	*/
	public void commandAction(Command command, Displayable displayable){
		exibirDebugMsg("ControleJogo>>commandAction");
		//Saindo da aplicação
		if(command == cmdSair){
			fecharConexao();
			cartasMidlet.destroyApp(true);
		}else if(command == cmdAjuda){
```

```
                        //Exibindo formulário de ajuda
                        exibirDebugMsg("ControleJogo>>CommandAction>>Exibindo
ajuda");
                        APRESENTACAO = false;
                        TELASELECAO = false;
                        display.setCurrent(formularioAjuda);
                }else if(command == cmdOpcoes){
                        //Exibindo formulário de opções
                        exibirDebugMsg("ControleJogo>>CommandAction>>Exibindo
opções");
                        APRESENTACAO = false;
                        TELASELECAO = false;
                        frmopcoes.setOpenMauMau(OPEN_MAU_MAU);

        frmopcoes.setExibirInformacoesCarta(EXIBIR_INFORMACAO_CARTA);
                        frmopcoes.setIPServidor(IP_SERVIDOR);
                        frmopcoes.setPortaServidor(PORTA_SERVIDOR);

                        display.setCurrent(frmopcoes);
                }else if(command == cmdVisualizadorCartas){
                        //Exibindo formulário de visualização de cartas
                        exibirDebugMsg("ControleJogo>>CommandAction>>Visualizador
de Cartas");
                        APRESENTACAO = false;
                        TELASELECAO = false;
                        display.setCurrent(visualizadorCartas);
                }else if(command == cmdIniciarJogo){
                        exibirDebugMsg("ControleJogo>>Inicializando novo jogo");
                        incrementaJogoAtual();
                        NUMERO_JOGADORES = 4;
                        ONLINE = false;
                        PENALIDADE_APLICADA = false;
                        APRESENTACAO = false;
                        TELASELECAO = false;
                        SENTIDO_JOGO = SENTIDO_NORMAL;
                        jogadorAtual = jogadorExibido = 0;
                        ANTERIOR_PULADO = false;
                        QUANT_COMPRA = 0;
                        JOGO_EM_ANDAMENTO = true;
                        resetarJogadores();
                        Graphics g = getGraphics();
                        g.setColor(0, 190, 0);
                        g.fillRect(0,0,dispositivo.getLarguraTela(),
dispositivo.getAlturaTela());resetarJogadores();
                        flushGraphics();
                        baralhoAtual.resetar();
                        baralhoAtual.embaralhar();
```

```
                    baralhoAtual.distribuirCartas(jogadores);

                    /*
                    //Teste da jogada Coringa

        jogadores[0].adicionarCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.CORIN
GA]);

                    //Teste da jogada Valete(*)

        baralhoAtual.adicionarCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.DOIS]);

        jogadores[0].adicionarCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.VALET
E]);
                    */

        //jogadores[0].adicionarCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.VALET
E]);

                    if(baralhoAtual.cartaAtual().getValor() == Carta.SETE){
                        setQuantidadeCompra((byte)3);

        ((JogadorHumano)jogadores[0]).setComandoCompra((byte)0);
                    }else{
                        if(baralhoAtual.cartaAtual().getValor() == Carta.DEZ &&
                          baralhoAtual.cartaAtual().getNaipe() == Carta.PAUS){
                            setQuantidadeCompra((byte)5);

        ((JogadorHumano)jogadores[0]).setComandoCompra((byte)0);
                        }
                    }
                    //((JogadorHumano)jogadores[0]).simularFimDeJogo(); //--> Apenas
para testar fim de jogo
                    display.setCurrent(jogadores[0]);

                    jogadorAtual = 0;
                    //loopback();


                }else if(command == cmdIniciarJogoOnLine){
                    exibirDebugMsg("ControleJogo>>Inicializando jogo on-line");
                    synchronized(this){
                        ONLINE = true;
                        primeiroStatusRecebido = false;
                        NUMERO_JOGADORES = 4;
                    }
```

```
                    //exibirDebugMsg("commandAction>>Jogo on-line>>" +
retornarMsgServidor(getHostServidor() + "listarSalasDisponiveis.jsp", "p=1", true));
                    CartasMidlet.getDisplay().setCurrent(frmespera);
                    Thread t = new Thread(){
                                public void run(){
                                        String strResultado =
retornarMsgServidor(getHostServidor(), "acao=LISTA_SALAS",false,
ControleJogo.this);
                                        if(strResultado.startsWith("ERRO")){
                                                ONLINE=false;
                                                exibirMsg("Não foi possível estabelecer a
conexão com o servidor. Verifique o endereço do servidor e a porta (através do menu
opções). Se estiverem corretos, tente novamente mais tarde.");

        exibirDebugMsg("ControleJogo>>commandAction>>ERRO: " + strResultado);
                                        }else{
                                                if(frmlistasalas == null)
                                                        frmlistasalas = new
frmListaSalas("Lista de salas", ControleJogo.this, strResultado, ControleJogo.this);
                                                else

                                                        frmlistasalas.reset(strResultado);


        CartasMidlet.getDisplay().setCurrent(frmlistasalas);
                                        }
                                }
                    };

                    t.start();

            }

        }

        /**
                Inicializa o estado do jogo para começar um novo jogo on-line
        */
        public synchronized void comecarJogoOnLine(boolean enviar){

        exibirDebugMsg("##################################################
######ControleJogo>>Inicializando novo jogo on-line>>JOGADOR:" + nomeJogador);
                primeiroStatusRecebido = true;
                ONLINE = true;
                APRESENTACAO = false;
                TELASELECAO = false;
                SENTIDO_JOGO = SENTIDO_NORMAL;
```

```
                jogadorAtual = jogadorExibido = 0;
                ANTERIOR_PULADO = false;
                QUANT_COMPRA = 0;
                JOGO_EM_ANDAMENTO = true;
                PENALIDADE_APLICADA = false;
                Graphics g = getGraphics();
                g.setColor(0, 190, 0);
                g.fillRect(0,0,dispositivo.getLarguraTela(), dispositivo.getAlturaTela());
                flushGraphics();
                int i = 0;
                exibirDebugMsg("ControleJogo>>Inicializando novo jogo on-
line>>Contando jogadores");

                for(i = 0; i < NUMERO_JOGADORES; i++){
                        if(nomeJogadores[i] == null){
                                break;
                        }else{

        exibirDebugMsg("ControleJogo>>comecarJogoOnLine>>NOMEJOGADORES[" +
i + "] --> " + nomeJogadores[i]);
                        }
                }
                NUMERO_JOGADORES = i;
                exibirDebugMsg("ControleJogo>>comecarJogoOnLine>>Nº de Jogadores:
" + NUMERO_JOGADORES);
                resetarJogadores();

                synchronized(baralhoAtual){
                        baralhoAtual.resetar();
                        baralhoAtual.embaralhar();
                        baralhoAtual.distribuirCartas(jogadores);
                }


                if(baralhoAtual.cartaAtual().getValor() == Carta.SETE){
                        setQuantidadeCompra((byte)3);
                        if(jogadores[0] instanceof JogadorHumano)

        ((JogadorHumano)jogadores[0]).setComandoCompra((byte)0);
                }else{
                        if(baralhoAtual.cartaAtual().getValor() == Carta.DEZ &&
                           baralhoAtual.cartaAtual().getNaipe() == Carta.PAUS){
                                setQuantidadeCompra((byte)5);
                                if(jogadores[0] instanceof JogadorHumano)

        ((JogadorHumano)jogadores[0]).setComandoCompra((byte)0);
                        }
```

```
                }

                /*jogadores[1].zerar(); //-->Para testar o fim do jogo

        jogadores[1].adicionarCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.CORINGA
]);

                baralhoAtual.zerar(); //-->Para testar o fim do jogo

        baralhoAtual.adicionarCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.CORING
A]);



                //Teste de jogada com valetes

        jogadores[0].adicionarCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.VALET
E]);

        jogadores[0].adicionarCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.VALETE])
;

        jogadores[0].adicionarCarta(Baralho.cartasBaralho[Carta.COPAS][Carta.VALETE
]);

        jogadores[0].adicionarCarta(Baralho.cartasBaralho[Carta.ESPADAS][Carta.VALE
TE]);
                */

                if(enviar){

                        if(frm == null){
                                frm = new frmEspera("Aguarde, inicializando jogo on-line");
                        }else{
                                frm.reset("Aguarde, inicializando jogo on-line");
                                frm.removeCommand(cmdCancelar);
                                frm.removeCommand(cmdComecarJogo);
                        }
                        frm.append(new StringItem("Aguarde: ", "Aguardando resposta do
servidor"));

                        display.setCurrent(frm);

                        enviarMsgServidor(
                                montarStatusJogo(
```

```
                                        new Mensagem("Iniciando jogo. O jogador " +
jogadores[0].getNome() + " fará a primeira jogada.", AlertType.INFO,
jogadores[jogadorAtual])
                                        , false));

                //}else{

                }
                System.gc();
        }

        /**
                Reseta o estado dos jogadores
        */
        public void resetarJogadores(){
                exibirDebugMsg("ControleJogo>>resetarJogadores()");
                jogadores = new Jogador[NUMERO_JOGADORES];

                if(!ONLINE){
                        jogadores[0] = new JogadorHumano("Player 1", dispositivo,
baralhoAtual, this);
                        jogadores[1] = new JogadorComputador("Player 2",  baralhoAtual,
this);
                        jogadores[2] = new JogadorComputador("Player 3",  baralhoAtual,
this);
                        jogadores[3] = new JogadorComputador("Player 4",  baralhoAtual,
this);
                }else{
                        for(int i = 0; i < NUMERO_JOGADORES; i++){
                           if(i == indiceJogadorRede){
                                jogadores[i] = new
JogadorHumano(nomeJogador.trim().toUpperCase(), dispositivo, baralhoAtual, this);
                           }else{
                                jogadores[i] = new JogadorComputador(
                                        nomeJogadores[i].trim().toUpperCase().substring(8,
nomeJogadores[i].length()).trim()
                                        , baralhoAtual, this);

                           }
                        }

                }
                System.gc();
        }

        public void setApresentacao(boolean b){
                        APRESENTACAO = b;
```

```
	}

	public void setTelaSelecao(boolean b){
			TELASELECAO = b;
			posicaoCartaX = -imgLogo.getWidth();
			posicaoCartaY = 0;
	}




	/**
	  Rotina que verifica qual é o próximo jogador e modfica o índice
	  do jogador atual para o próximo
	*/
	public int proximoJogador(){
			exibirDebugMsg("ControleJogo>>Exibindo próximo jogador");
			jogadorAtual += SENTIDO_JOGO;
			if(jogadorAtual < 0) jogadorAtual = (byte)(NUMERO_JOGADORES -
1);
			else if(jogadorAtual > (NUMERO_JOGADORES - 1)) jogadorAtual =
0;
			jogadorExibido = jogadorAtual;
			return jogadorAtual;
	}

	/**
		Rotina que retorna o índice do próximo jogador, mas não modifica o índice
do jogador atual
	*/
	public int getIndiceProximoJogador(){

	exibirDebugMsg("ControleJogo>>getIndiceProximoJogador()>>Retornando o
índice do próximo jogador");
			int proximoJogador = jogadorAtual + SENTIDO_JOGO;
			if(proximoJogador < 0) proximoJogador =
(byte)(NUMERO_JOGADORES - 1);
			else if(proximoJogador > (NUMERO_JOGADORES - 1))
proximoJogador = 0;
			return proximoJogador;
	}


	/**
		Rotina que retorna o índice do jogador anterior
	*/
	public int getIndiceJogadorAnterior(){
```

```
        exibirDebugMsg("ControleJogo>>getIndiceJogadorAnterior()>>Retornando o
índice do jogador anterior");
                        int jogadorAnterior = jogadorAtual - SENTIDO_JOGO;
                        if(jogadorAnterior < 0) jogadorAnterior =
(byte)(NUMERO_JOGADORES - 1);
                        else if(jogadorAnterior > (NUMERO_JOGADORES - 1))
jogadorAnterior = 0;
                        return jogadorAnterior;
        }


        /**
                Muda o jogador que está sendo exibido
        */
        public void exibirProximoJogador(){
                        jogadorExibido++;
                        if(jogadorExibido == NUMERO_JOGADORES) jogadorExibido = 0;
                        exibirDebugMsg("ContoleJogo>>exibirProximoJogador>>Exibindo
cartas dojogador:" + jogadores[jogadorExibido].getNome());
                        CartasMidlet.getDisplay().setCurrent(jogadores[jogadorExibido]);
        }

        /**
                Retorna o índice do jogador atual
        */
        public byte getJogadorAtual(){
                return jogadorAtual;
        }

        /**
                Retorna o jogador que está sendo exibido
        */
        public byte getJogadorExibido(){
                return jogadorExibido;
        }

        /**
                Seta Open MauMau
        */
        public static void setOpenMauMau(boolean b){
                OPEN_MAU_MAU = b;
        }

        /**
                Retorna o valor de OpenMauMau
        */
```

```
	public static boolean getOpenMauMau(){
		return OPEN_MAU_MAU;
	}

	/**
		Exibe informação de debug (apenas se o modo debug estiver acionado)
	*/
	public static void exibirDebugMsg(String msg){
		if(DEBUG_MODE)
			System.out.println(msg);
	}




	/**
		Verifica se a jogada que está sendo feita é válida ou não
	*/

	public Mensagem verificarJogada(Carta carta, JogadorHumano jogador){
		Carta cartaBaralho = null, cartaJogador = carta;

	exibirDebugMsg("ControleJogo>>verificarJogada(carta,jogador)>>INICIANDO
VERIFICAÇÃO DE JOGADA");

		Mensagem mensagem = null;
		try{
			cartaBaralho = baralhoAtual.cartaAtual();


	exibirDebugMsg("ControleJogo>>verificarJogada(carta,jogador)>>CARTA DO
BARALHO: " + cartaBaralho.toString());

	exibirDebugMsg("ControleJogo>>verificarJogada(carta,jogador)>>CARTA
JOGADA: " + cartaJogador.toString());

			/*****************************************************CARTAS QUE
NÃO DEPENDEM DA CARTA ATUAL DO
BARALHO*************************************************************************/
			/*Jogada: Coringa(*) --> Pode ser colocado sobre qualquer carta e
anula o efeito de qualquer carta especial*/
			exibirDebugMsg("ControleJogo>>verificarJogada(C*)>>Verificando
se carta jogada é um coringa");

			if(cartaJogador.getValor() == Carta.CORINGA){
```

```
                                exibirDebugMsg("ControleJogo>>verificarJogada(C*)>>Carta
jogada é um coringa");

                                setPenalidadeAplicada(false);
                                setQuantidadeCompra((byte)0);
                                String msgComplemento = "";
                                if(cartaBaralho.getValor() == Carta.SETE)
                                        msgComplemento = "O coringa anulou o efeito do sete
anterior.";
                                else if(cartaBaralho.getValor() == Carta.NOVE)
                                        msgComplemento = "O coringa anulou o efeito do
nove anterior.";
                                else if(cartaBaralho.getValor() == Carta.DEZ &&
cartaBaralho.getNaipe() == Carta.PAUS)
                                        msgComplemento = "O coringa anulou o efeito do 10
de paus anterior.";
                                baralhoAtual.adicionarCarta(cartaJogador);
                                mensagem = new Mensagem("O jogador " +
jogador.getNome() + " jogou um coringa, que pode ser jogado sobre qualquer carta e
anula o efeito de qualquer carta especial. " + msgComplemento + " O próximo jogador
será: " + jogadores[getIndiceProximoJogador()].getNome() + ". Movendo para o próximo
jogador.",
                                        AlertType.INFO, jogadores[proximoJogador()]);


                        }else{
                                /*Jogada: Valete(*): Pode ser colocada sobre qualquer carta
e anula o efeito de cartas especiais. Ao jogar um valete,
                                o jogador escolhe qual será o naipe da próxima carta que
deverá ser jogada*/

        exibirDebugMsg("ControleJogo>>verificarJogada(Valete*)>>VERIFICANDO SE
CARTA JOGADA É UM VALETE");
                                if(cartaJogador.getValor() == Carta.VALETE){

        exibirDebugMsg("ControleJogo>>verificarJogada(Valete*)>>Carta jogada é um
valete");
                                        setPenalidadeAplicada(false);
                                        setQuantidadeCompra((byte)0);
                                        String msgComplemento = "";
                                        if(cartaBaralho.getValor() == Carta.SETE)
                                                msgComplemento = "O valete anulou o efeito
do sete anterior.";
                                        else if(cartaBaralho.getValor() == Carta.NOVE)
                                                msgComplemento = "O valete anulou o efeito
do nove anterior.";
```

```
                                    else if(cartaBaralho.getValor() == Carta.DEZ &&
cartaBaralho.getNaipe() == Carta.PAUS)
                                        msgComplemento = "O valete anulou o efeito
do 10 de paus anterior.";
                                    baralhoAtual.adicionarCarta(cartaJogador);

                                    mensagem = new Mensagem("O jogador " +
jogador.getNome() + " jogou um valete, que pode ser jogada sobre qualquer carta e
anula o efeito de qualquer carta especial. " + msgComplemento,
                                    AlertType.INFO, jogadores[proximoJogador()]);

                                    frmescolhanaipe = new frmEscolhaNaipe(jogador, this,
mensagem);

        CartasMidlet.getDisplay().setCurrent(frmescolhanaipe);

                                }else{
                    /*********************************************************FIM DE CARTAS
QUE NÃO DEPENDEM DA CARTA ATUAL DO
BARALHO*************************************************************************************/
                    /*****************************************************DEFESA DE
CARTAS*************************************************************************************/

        exibirDebugMsg("ControleJogo>>verificarJogada(7*)>>VERIFICANDO SE
CARTA DO BARALHO É UM SETE");
                                    if(cartaBaralho.getValor() == Carta.SETE &&
!getPenalidadeAplicada()){

        exibirDebugMsg("ControleJogo>>verificarJogada(7*)>>CARTA DO BARALHO É
UM SETE");
                                        if(cartaJogador.getValor() == Carta.SETE){
                                            setPenalidadeAplicada(false);

        exibirDebugMsg("ControleJogo>>VerificarJogada(7)>>DOBRANDO A
QUANTIDADE DE COMPRA");
                                            baralhoAtual.adicionarCarta(carta);

        setQuantidadeCompra((byte)(getQuantidadeCompra() + 3));
                                            mensagem = new Mensagem(
                                            "O jogador " + jogador.getNome() + "
jogou um sete. O próximo jogador deverá comprar " + getQuantidadeCompra() + "
cartas, jogar outro 7, jogar um 2 do mesmo naipe do 7 atual, um coringa ou um valete.
O próximo jogador será: " + jogadores[getIndiceProximoJogador()].getNome() + ".
Movendo para o próximo jogador.",
                                                AlertType.INFO,
jogadores[proximoJogador()]);
                                        }else{
```

```
                                                    if(cartaJogador.getNaipe() ==
cartaBaralho.getNaipe() && cartaJogador.getValor() == Carta.DOIS){
                                                        baralhoAtual.adicionarCarta(carta);
                                                        setQuantidadeCompra((byte)0);
                                                        mensagem = new Mensagem(
                                                        "O jogador " + jogador.getNome()
+ " jogou um 2, que anulou o efeito do 7 anterior. O próximo jogador será: " +
jogadores[getIndiceProximoJogador()].getNome() + ". Movendo para o próximo
jogador.",
                                                        AlertType.INFO,
jogadores[proximoJogador()]);
                                                    }else{

        exibirDebugMsg("ControleJogo>>verificarJogada>>Jogada inválida
detectada[7]");
                                                        mensagem = new
Mensagem("Jogada inválida!",
                                                                AlertType.ERROR,
jogadores[jogadorAtual]);
                                                    }
                                                }


                                        }else{

        exibirDebugMsg("ControleJogo>>verificarJogada(9*)>>VERIFICANDO SE
CARTA DO BARALHO É UM NOVE");
                                                if(cartaBaralho.getValor() == Carta.NOVE &&
!getPenalidadeAplicada()){

        exibirDebugMsg("ControleJogo>>verificarJogada(9*)>>CARTA DO BARALHO É
UM NOVE");

                                                    if(cartaJogador.getValor() == Carta.DOIS
&& cartaJogador.getNaipe() == cartaBaralho.getNaipe()){

        exibirDebugMsg("ControleJogo>>verificarJogada(9*)>>DOIS. ANULADO O
EFEITO DO NOVE ANTERIOR...");
                                                        baralhoAtual.adicionarCarta(carta);
                                                        mensagem = new Mensagem("O
jogador " + jogador.getNome() + " jogou um dois do mesmo naipe do nove que estava
no baralho. Com isto, o efeito do nove foi cancelado. O próximo jogador será: " +
jogadores[getIndiceProximoJogador()].getNome() + ". Movendo para o próximo
jogador.",
                                                        AlertType.INFO,
jogadores[proximoJogador()]);
                                                    setQuantidadeCompra((byte)0);
```

```
                                                }else{

        exibirDebugMsg("ControleJogo>>verificarJogada(9*)>>CARTA INVÁLIDA");
                                                        mensagem = new
Mensagem("Jogada inválida! Você deve jogar um dois do mesmo naipe que o nove que
está no baralho, um coringa, um valete ou comprar uma carta.",
                                                                AlertType.ERROR,
jogadores[jogadorAtual]);


                                                }
                                        }else{

        exibirDebugMsg("ControleJogo>>VerificarJogada(10p)>>Verificando se a carta
do baralho é um 10 de paus");

                                                if(cartaBaralho.getNaipe() ==
Carta.PAUS && cartaBaralho.getValor() == Carta.DEZ && !getPenalidadeAplicada()){

        exibirDebugMsg("ControleJogo>>VerificarJogada(10p)>>Carta do baralho é um
10 de paus>>Verificando se a carta jogada é um 2 de paus");
                                                        setQuantidadeCompra((byte)5);
                                                        if(cartaJogador.getValor() ==
Carta.DOIS && cartaJogador.getNaipe() == Carta.PAUS){

        exibirDebugMsg("ControleJogo>>VerificarJogada(10p)>>Carta do baralho é um
10 de paus>>2 DE PAUS JOGADO");

        baralhoAtual.adicionarCarta(carta);
                                                                mensagem = new
Mensagem("O jogador " + jogador.getNome() + " jogou um dois de paus. Com isto, o
efeito do 10 de paus foi cancelado. O próximo jogador será: " +
jogadores[getIndiceProximoJogador()].getNome() + ". Movendo para o próximo
jogador.",

                                                                        AlertType.INFO,
jogadores[proximoJogador()]);

        setQuantidadeCompra((byte)0);
                                                        }else{

        exibirDebugMsg("ControleJogo>>VerificarJogada(10p)>>Carta do baralho é um
10 de paus>>Carta jogada é inválida!");
                                                                mensagem = new
Mensagem("Jogada inválida. Você deve jogar um dois de paus, um coringa, um valete
ou comprar 5 cartas.",
```

```
                                                                    AlertType.ERROR,
jogadores[jogadorAtual]);


                                                }

                                            }else{

        /*****************************************************FIM DA DEFESA DE
CARTAS*************************************************************************************/

        /*****************************************************CARTAS QUE SATISFAZEM A
CONDIÇÃO
BÁSICA*************************************************************************************/

        exibirDebugMsg("ControleJogo>>verificarJogada>>VERIFICANDO CONDIÇÃO
BÁSICA");

                                                    if( ( (cartaBaralho.getValor() !=
Carta.VALETE || (cartaBaralho.getValor() == Carta.VALETE &&
getPenalidadeAplicada()) ) && (cartaBaralho.getNaipe() == cartaJogador.getNaipe() ||
cartaBaralho.getValor() == cartaJogador.getValor()))
                                                        || (cartaBaralho.getValor() ==
Carta.VALETE && cartaJogador.getNaipe() == NAIPE_ESCOLHIDO &&
!getPenalidadeAplicada()) ){

                                                    /*Regra: A* --> Indica que o
jogador deve jogar novamente*/

        exibirDebugMsg("ControleJogo>>verificarJogada>>VERIFICANDO SE A CARTA
DO JOGADOR É UM AS");

                                                    if(cartaJogador.getValor()
== Carta.AS){

        exibirDebugMsg("ControleJogo>>verificarJogada>>CARTA DO JOGADOR É UM
AS");

        baralhoAtual.adicionarCarta(carta);
                                                                    mensagem = new
Mensagem("O jogador " + jogador.getNome() + " jogou um AS e tem direito a mais uma
jogada!",
                                                                    AlertType.INFO,
jogadores[jogadorAtual]);
                                                    }else{
```

```
                                                                    /*VerificandoRegra: 4
de espadas--> Todo mundo na mesa (menos quem jogou) compra uma
carta>>Indefensável*/

        exibirDebugMsg("Baralho>>VerificarJogada>>Verificando Regra: 4 de espadas--
> Todo mundo na mesa compra uma carta>>Indefensável");

        if(cartaJogador.getNaipe() == Carta.ESPADAS && cartaJogador.getValor() ==
Carta.QUATRO){

        exibirDebugMsg("ControleJogo>>verificarJogada(Carta)>>Jogador jogou um 4
de Espadas>>Todos os outros \ndevem comprar uma carta");

        baralhoAtual.adicionarCarta(carta);
                                                                    for(int i = 0 ; i <
NUMERO_JOGADORES; i++){

        if(!(jogadores[i].equals(jogador))){

        try{

        jogadores[i].adicionarCarta(baralhoAtual.comprarCarta());

        }catch(EmptyStackException e){

        exibirDebugMsg("Baralho>>verificarJogada(4 espadas)>>Baralho Vazio!");

        mensagem = elegerVencedor();

        //return mensagem;
                                                                            }
                                                                        }
                                                                    }
                                                                    mensagem =
new Mensagem("O jogador " + jogador.getNome() + " jogou um 4 de espadas e todos
ou outros jogadores compraram uma carta. O próximo jogador será: " +
jogadores[getIndiceProximoJogador()].getNome() + ". Movendo para o próximo
jogador.",

        AlertType.INFO, jogadores[proximoJogador()]);
                                                                    }else{

        if(cartaJogador.getValor() == Carta.CINCO){

        exibirDebugMsg("Baralho>>verificarJogada(5*)>>INVERTENDO O SENTIDO DO
JOGO");
```

```
        baralhoAtual.adicionarCarta(carta);

        inverterSentidoJogo();

        mensagem = new Mensagem("O jogador " + jogador.getNome() + " jogou um 5 e
inverteu o sentido do jogo. O próximo jogador será: " +
getJogador(getIndiceProximoJogador()).getNome() + ". Movendo para o próximo
jogador.",

        AlertType.INFO, jogadores[proximoJogador()]);
                                                                    }else{
                                                                        /*Regra
(Q(DAMA)*): pula o próximo jogador*/

        exibirDebugMsg("Baralho>>verificarJogada(Q*(DAMA))>>VERIFICANDO SE A
CARTA JOGADA É UMA DAMA");


        if(cartaJogador.getValor() == Carta.DAMA){

        exibirDebugMsg("Baralho>>verificarJogada(Q*(DAMA))>>CARTA JOGADA É
UMA DAMA>>PULANDO PRÓXIMO JOGADOR.");

        String nomeJogadorPulado = getJogador(getIndiceProximoJogador()).getNome();

        proximoJogador();

        mensagem = new Mensagem("O jogador " + jogador.getNome() + " jogou uma
dama e, como consequência, o jogador " + nomeJogadorPulado + " foi pulado. O
próximo jogador será: " + getJogador(getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, jogadores[proximoJogador()]);

        baralhoAtual.adicionarCarta(carta);



                                                                    }else{


        /*Verificando regra 8*: Se a que está sendo jogada for um 8, o jogador receberá
uma carta do próximo

        jogador e poderá jogar novamente, a não ser que o próximo jogador tenha
apenas uma carta.
```

```
        Neste caso, o 8 tem o efeito de uma carta normal.
                                                                                */

        exibirDebugMsg("ControleJogo>>VerificarJogada(8*)>>Verificando se carta
jogada é um 8");

        if(cartaJogador.getValor() == Carta.OITO){

        baralhoAtual.adicionarCarta(carta);

        exibirDebugMsg("ControleJogo>>VerificarJogada(8*)>>Carta jogada é um 8");

        if(jogadores[getIndiceProximoJogador()].numeroCartas() > 1){

                exibirDebugMsg("ControleJogo>>VerificarJogada(8*)>>Comprando uma
carta do próximo jogador");


        jogador.adicionarCarta(jogadores[getIndiceProximoJogador()].removerCarta());

                mensagem = new Mensagem("O jogador " + jogador.getNome() + " jogou
um 8. Com esta jogada, ele está recebendo uma carta do próximo jogador e pode jogar
novamente! Realizando nova jogada.",

                        AlertType.INFO, jogadores[jogadorAtual]);


        }else{

                exibirDebugMsg("ControleJogo>>VerificarJogada(8*)>>Não é possível
comprar uma carta do próximo jogador, pois o mesmo possui apenas uma carta!
Movendo para o próximo jogador.");


                mensagem = new Mensagem("O jogador " + jogador.getNome() + " jogou
um 8, porém não poderá receber uma carta do próximo jogador, pois o mesmo possui
apenas uma carta. O próximo jogador será: " +
getJogador(getIndiceProximoJogador()).getNome() + ". Movendo para o próximo
jogador.",

                AlertType.INFO, jogadores[proximoJogador()]);


        }

        }else{
```

```
        exibirDebugMsg("ControleJogo>>VerificarJogada(R*)>>VERIFICANDO SE
CARTA JOGADA É UM REI");



        /*Verificando jogada: R(*) --> Faz o jogador anterior comprar uma carta*/

        if(cartaJogador.getValor() == Carta.REI){

                exibirDebugMsg("ControleJogo>>VerificarJogada(R*)>>VERIFICANDO
SE CARTA JOGADA É UM REI>>CARTA JOGADA É UM REI");

                baralhoAtual.adicionarCarta(carta);


        jogadores[getIndiceJogadorAnterior()].adicionarCarta(baralhoAtual.comprarCarta
());

                mensagem = new Mensagem(

                        "O jogador " + jogador.getNome() + " jogou um rei e fez o jogador
anterior comprar uma carta. O próximo jogador será: " +
getJogador(getIndiceProximoJogador()).getNome() + ". Movendo para o próximo
jogador.",

                AlertType.INFO, jogadores[proximoJogador()]);

        }else{


                if(cartaJogador.getValor() == Carta.SETE){

                        setPenalidadeAplicada(false);

                        baralhoAtual.adicionarCarta(carta);

                        setQuantidadeCompra((byte)(getQuantidadeCompra() + 3));

                        mensagem = new Mensagem("O jogador " + jogador.getNome() + "
jogou um sete. O próximo jogador deve comprar " + getQuantidadeCompra() + " cartas,
jogar outro 7, jogar um 2 do naipe atual, um coringa ou um valete. O próximo jogador
```

```
será: " + getJogador(getIndiceProximoJogador()).getNome() +". Movendo para o
próximo jogador.",

                        AlertType.INFO, jogadores[proximoJogador()]);


                }else{

                        if(cartaJogador.getValor() == Carta.DEZ &&
cartaJogador.getNaipe() == Carta.PAUS){

                                setPenalidadeAplicada(false);

                                setQuantidadeCompra((byte)5);

                                baralhoAtual.adicionarCarta(carta);

                                mensagem = new Mensagem("O jogador " +
jogador.getNome() + " jogou um 10 de paus. O próximo jogador deve comprar " +
getQuantidadeCompra() + " cartas, jogar um 2 de paus, um coringa ou um valete. O
próximo jogador será: " + getJogador(getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                AlertType.INFO, jogadores[proximoJogador()]);


                        }else{


                                if(carta.getValor() == Carta.NOVE){

                                        setPenalidadeAplicada(false);

                                          setQuantidadeCompra((byte)1);

                                          baralhoAtual.adicionarCarta(carta);

                                          mensagem = new Mensagem("O jogador " +
jogador.getNome() + " jogou um 9. O próximo jogador deve comprar " +
getQuantidadeCompra() + " cartas, jogar um 2 do mesmo naipe do nove, um coringa ou
um valete. O próximo jogador será: " +
getJogador(getIndiceProximoJogador()).getNome() + ". Movendo para o próximo
jogador.",

                                                        AlertType.INFO,
jogadores[proximoJogador()]);
```

```
                    }else{

                        baralhoAtual.adicionarCarta(carta);


                        mensagem = new Mensagem("O jogador " +
jogador.getNome() + " fez uma jogada legal, porém não jogou nenhuma carta especial.
O próximo jogador será: " + getJogador(getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                        AlertType.INFO, jogadores[proximoJogador()]);

                    }

                }

            }




        }



                                                                    }


                                                                }

                                                            }

                                                        }
```

```
                                                        }//Fim do if AS

        /*****************************************************FIM DAS CARTAS QUE
SATISFAZEM A CONDIÇÃO
BÁSICA*************************************************************************************/

                                                    }else{

        exibirDebugMsg("ControleJogo>>verificarJogada>>Jogada inválida detectada");
                                                            mensagem = new
Mensagem("Jogada inválida! Você deve jogar uma carta do mesmo naipe ou do mesmo
valor da carta que está no baralho ou jogar uma das cartas especiais. Jogue
novamente.",
                                                                    AlertType.ERROR,
jogadores[jogadorAtual]);
                                                    }



                                                }
                                            }


                                        }

                                    }
                                }




                        if(mensagem.getType() != AlertType.ERROR &&
jogador.numeroCartas() == 0){
                            mensagem = new Mensagem("Fim de jogo. O vencedor foi: "
+ jogador.getNome()
                                    , AlertType.INFO, this);
                            mensagem.setTimeout(Jogador.ESPERA);

                        }else{


                            mensagem.setTimeout(Jogador.ESPERA);

                            if(mensagem.getType() != AlertType.ERROR){
```

```
                                        if( jogador.getDisseMauMau() &&
jogador.numeroCartas() > 2){
                                                mensagem.setString(mensagem.getString() + "
Você só deverá dizer mau-mau quando possuir apenas uma carta. Você receberá três
cartas como penalidade por dizer mau-mau na hora errada.");
                                                for(byte i = 0; i < 3; i++)

        jogador.adicionarCarta(baralhoAtual.comprarCarta());
                                        }else{
                                                if(jogador.numeroCartas() == 2 &&
!(jogador.getDisseMauMau())){


        mensagem.setString(mensagem.getString() + " Você possui uma carta e não
disse mau-mau, portanto, como penalidade, você receberá três cartas.");
                                                        for(byte i = 0; i < 3; i++)

        jogador.adicionarCarta(baralhoAtual.comprarCarta());

                                                }

                                        }


                                }
                        }
                }catch(EmptyStackException e){
                        //Se o baralho estiver vazio, elege o vencedor do jogo
                        exibirDebugMsg("Baralho>>verificarJogada>>Baralho Vazio!");
                        mensagem = elegerVencedor();

                }
                System.gc();

                return mensagem;
        }

        /**
                Elege o vencedor do jogo
        */
        public Mensagem elegerVencedor(){
                exibirDebugMsg("Baralho>>elegerVencedor()>>elegendo vencedor do
jogo");
                Mensagem mensagem = null;
                Jogador vencedor = jogadores[0];
                for(int i = 0; i < NUMERO_JOGADORES; i++){
```

```
                    if(jogadores[i].numeroCartas() < vencedor.numeroCartas()){
                        vencedor = jogadores[i];
                    }
                }
                if(baralhoAtual.numeroCartas() == 0){
                    mensagem = new Mensagem("O baralho ficou vazio! Neste caso, o
vencedor é o jogador com menos cartas. O vencedor foi: " + vencedor.getNome()
                            , AlertType.INFO, this);
                }else{
                    mensagem = new Mensagem("Fim de jogo. O vencedor foi: " +
vencedor.getNome()
                            , AlertType.INFO, this);
                }
                return mensagem;
        }


        /**
            Seta a quantidade de cartas que devem ser compradas
        */
        public static void setQuantidadeCompra(byte quantidade){
            if(quantidade >= 0)
                QUANT_COMPRA = quantidade;
        }

        /**
            Retorna a quantidade de cartas que devem ser compradas
        */
        public static byte getQuantidadeCompra(){
                return QUANT_COMPRA;
        }

        /**
            Classe que exibe mensagens na tela

            @version 1.0
            @author Rogério de Paula Aguilar
        */
        public class Mensagem extends Alert{
            /**
                Tela de volta
            */
            protected Displayable proximaTela;

            /**
                Construtor
            */
```

```
            public Mensagem(String titulo, AlertType tipo, Displayable proximaTela){
                super("", titulo, null, tipo);
                this.proximaTela = proximaTela;
                setTimeout(6000);
                setCommandListener(
                    new CommandListener(){
                        public void commandAction(Command c, Displayable
d){

        exibirDebugMsg("Mensagem>>commandAction");
                                    if(indiceJogadorRede == jogadorAtual &&
ONLINE && !getRecebeuPermissaoJogada()){
                                        MensagemAtualizacao msg = new
MensagemAtualizacao(getString(), getType(), Mensagem.this.proximaTela);
                                        msg.setTimeout(2000);

        CartasMidlet.getDisplay().setCurrent(msg);
                                        System.gc();
                                        return;
                                    }

                                    if(Mensagem.this.proximaTela instanceof
Jogador){

        ((Jogador)Mensagem.this.proximaTela).setDisseMauMau(false);

                                        if(Mensagem.this.proximaTela instanceof
JogadorHumano){
                                            try{
                                                if(getQuantidadeCompra() >
1){
                                                    if(!ONLINE)

        ((JogadorHumano)jogadores[0]).setComandoCompra((byte)0);
                                                    else

        ((JogadorHumano)jogadores[indiceJogadorRede]).setComandoCompra((byte)0);

                                                }else{
                                                    if(!ONLINE)

        ((JogadorHumano)jogadores[0]).setComandoCompra((byte)1);
                                                    else

        ((JogadorHumano)jogadores[indiceJogadorRede]).setComandoCompra((byte)1);

                                                }
```

```
                                                        }catch(EmptyStackException e){

        CartasMidlet.getDisplay().setCurrent(elegerVencedor());
                                                        }
                                                }
                                        }

        CartasMidlet.getDisplay().setCurrent(Mensagem.this.proximaTela);
                                        if(Mensagem.this.proximaTela instanceof
JogadorComputador){

        ((JogadorComputador)Mensagem.this.proximaTela).jogar(null);
                                        }


                                        if(Mensagem.this instanceof
MensagemAtualizacao && ONLINE && indiceJogadorRede != getIndiceJogadorAtual()){

        enviarMsgServidor("ACKATUALIZACAO");
                                        }
                                }

                        }
                );



                /*if(ControleJogo.this.JOGO_EM_ANDAMENTO)
                        CartasMidlet.getDisplay().setCurrent(this);*/

        }

        /**
                Modifica a tela de volta
        */
        public void setProximaTela(Displayable displayable){
                proximaTela = displayable;
        }
}

/**
        Mensagem especial, utilizada para indicar que o jogador atualizou
        o status do jogo em rede

        @author Rogério de Paula Aguilar
        @version 1.0
```

```
	*/
	class MensagemAtualizacao extends Mensagem{

		public MensagemAtualizacao(String titulo, AlertType tipo, Displayable
proximaTela){
			super(titulo, tipo, proximaTela);
		}

	}

	/**
		Retorna um jogador
	*/
	public Jogador getJogador(int indiceJogador){
		if(indiceJogador < 0 || indiceJogador > NUMERO_JOGADORES)
			throw new IllegalArgumentException("Jogador inválido!");
		else
			return jogadores[indiceJogador];
	}


	/**
		Inverte o sentido do jogo
	*/

	public void inverterSentidoJogo(){
			if(SENTIDO_JOGO == SENTIDO_NORMAL) SENTIDO_JOGO =
SENTIDO_INVERSO;
			else SENTIDO_JOGO = SENTIDO_NORMAL;

	}

	/**
		Mostra a tela informando o vencedor do jogo
	*/

	public void setVencedor(Jogador jogador){
			Mensagem mensagem = new Mensagem("Fim de jogo. O vencedor
foi: " + jogador.getNome()
					, AlertType.INFO, this);
			mensagem.setTimeout(Alert.FOREVER);
			if(!ONLINE)
				CartasMidlet.getDisplay().setCurrent(mensagem);
			else
				enviarMsgServidor(montarStatusJogo(mensagem, true));
	}
```

```
	/**
		Classe que fica esperando um comando do servidor. <p>Esta classe
sincroniza os comandos que vem do servidor
		e executa os mesmos
	*/
	private static class threadLeitura implements Runnable{
		/**
			Objeto utilizado para sincronização de comandos
		*/
		private static Object objSync = new Object(); //Objeto utilizado para
sincronização

		private ControleJogo controleJogo;

		/**
			Construtor
		*/
		public threadLeitura(ControleJogo controleJogo){
			new Thread(this).start();
			this.controleJogo = controleJogo;
		}

		/**
			Fica esperando um comando do servidor, sincroniza comandos e
executa os mesmos
		*/
		public void run(){
			exibirDebugMsg("ControleJogo>>threadLeitura>>Iniciando thread
de leitura");
			StringBuffer resultado = new StringBuffer();
			while(CONECTADO){
				resultado.delete(0, resultado.length());
				int ch = 0;

				try{
					while(ch != -1 && ((char)ch) != '\n') {
						ch = is.read();
						resultado.append((char)ch);
					}

					comando = resultado.toString().trim();
					exibirDebugMsg("threadLeitura(" + (nomeJogadores[0]
== null ? "null" : nomeJogadores[0]) + ")>>Comando>>" + comando);
					synchronized(controleJogo){
						if(comando.equals("SAIR_SERVIDOR")){
```

```
                                        synchronized(controleJogo){

        exibirDebugMsg("ControleJogo>>threadLeitura>>Jogador foi desconectado pelo
servidor");

        controleJogo.setApresentacao(true);
                                                controleJogo.setTelaSelecao(true);

        controleJogo.JOGO_EM_ANDAMENTO = false;
                                                controleJogo.ONLINE=false;
                                                controleJogo.exibirMsg("Você foi
desconectado pelo servidor. Reinicializando jogo.");
                                                fecharConexaoSemConfirmacao();
                                        }
                                }else if(comando.startsWith("ERRO")){
                                        synchronized(controleJogo){

        exibirDebugMsg("ControleJogo>>threadLeitura>>Não foi possível entrar na sala.
" + comando);

        controleJogo.setApresentacao(true);
                                                controleJogo.setTelaSelecao(true);

        controleJogo.JOGO_EM_ANDAMENTO = false;
                                                controleJogo.ONLINE=false;
                                                controleJogo.exibirMsg("Ocorreu
um erro. Mensagem do servidor: " + comando);

                                                //fecharConexao();
                                                fecharConexaoSemConfirmacao();
                                        }
                                }else if(comando.equals("OKPRIM") ||
comando.equals("OK")){
                                        frmEspera frm = null;
                                        primeiroJogador = false;
                                        if(comando.equals("OKPRIM")){
                                                primeiroJogador = true;
                                                if(frm == null){
                                                        frm = new frmEspera("Você
é o primeiro jogador a entrar nesta sala. Aguardando a entrada de novos jogadores.");
                                                }else{
                                                        frm.reset("Você é o primeiro
jogador a entrar nesta sala. Aguardando a entrada de novos jogadores.");
                                                }

                                        }else{
                                                if(frm == null){
```

```
                                                        frm = new frmEspera("Você
entrou na sala com sucesso. O responsável pelo início do jogo é o primeiro jogador da
sala. Aguardando o início do jogo.");
                                                }else{
                                                        frm.reset("Você entrou na
sala com sucesso. O responsável pelo início do jogo é o primeiro jogador da sala.
Aguardando o início do jogo.");

                                                }
                                        }
                                        frm.removeCommand(cmdCancelar);

        frm.removeCommand(cmdComecarJogo);
                                        frm.addCommand(cmdCancelar);

                                        if(comando.equals("OKPRIM")){
                                                frm.append(new
StringItem("Jogadores na sala " + idSala + ":", ""));
                                                frm.append(new StringItem("",
"\nJogador: " + nomeJogadores[0].toUpperCase()));


                                        }
                                        //frm.append("\nJogador: " +
nomeJogadores[0]);

                                        if(listenerOK == null){
                                                listenerOK = new
CommandListener(){

                                                        public void
commandAction(Command c, Displayable d){

        if(c.getLabel().equals("Cancelar")){

        synchronized(controleJogo){

        //if(CartasMidlet.getDisplay() != null){

        controleJogo.setApresentacao(true);

        controleJogo.setTelaSelecao(true);

        controleJogo.JOGO_EM_ANDAMENTO = false;

        if(CartasMidlet.getDisplay() != null)
```

```
                CartasMidlet.getDisplay().setCurrent(controleJogo);

        enviarMsgServidor("SAIR_CLIENTE");

        controleJogo.fecharConexao();
                                                                        //      }
                                                                        }
                                                                }
                                                        }
                                                };


                                        }


                                        frm.setCommandListener(
                                                listenerOK
                                        );

        CartasMidlet.getDisplay().setCurrent(frm);
                                    }else
if(comando.startsWith("ENTRADA_JOGADOR")){
                                        //if(!JOGO_EM_ANDAMENTO){
                                        System.out.println("CRIANDO
FORMULÁRIO DE LISTA DE USUÁRIOS");
                                        frmEspera frmEsperaTMP = frm;
                                        if(primeiroJogador){
                                                if(frmEsperaTMP == null){
                                                        frmEsperaTMP = new
frmEspera("Você é o primeiro jogador a entrar nesta sala. Aguardando a entrada de
novos jogadores.");
                                                }else{
                                                        frmEsperaTMP.reset("Você
é o primeiro jogador a entrar nesta sala. Aguardando a entrada de novos jogadores.");
                                                }
                                        }else{
                                                if(frmEsperaTMP == null){
                                                        frmEsperaTMP = new
frmEspera("Você entrou na sala com sucesso. O responsável pelo início do jogo é o
primeiro jogador da sala. Aguardando o início do jogo.");
                                                }else{
                                                        frmEsperaTMP.reset("Você
entrou na sala com sucesso. O responsável pelo início do jogo é o primeiro jogador da
sala. Aguardando o início do jogo.");
```

```
                                                }
                                            }

        frmEsperaTMP.removeCommand(cmdCancelar);

        frmEsperaTMP.removeCommand(cmdComecarJogo);


                                            int indice = comando.indexOf("]");
                                            int indiceInicial = 16;
                                            int i = 0;
                                            for(i = 0; i < NUMERO_JOGADORES;
i++)
                                                nomeJogadores[i] = null;
                                            int j = 0;
                                            System.out.println("FORMULÁRIO DE
LISTA DE USUÁRIOS CRIADO");

                                            comando =
comando.substring(indiceInicial, comando.length() -1 );
                                            exibirDebugMsg("COMANDO_PARCIAL:
" + comando);

                                            indice = comando.indexOf("][");

                                            while(indice != -1){
                                               nomeJogadores[j++] =
comando.substring(0, indice);
                                               comando = comando.substring(indice
+ 2, comando.length());

exibirDebugMsg("COMANDO_PARCIAL_ENTRADA: " + comando);
                                               indice = comando.indexOf("][");
                                               if( nomeJogadores[j-
1].trim().toUpperCase().substring(8, nomeJogadores[j-
1].length()).trim().equals(nomeJogador.trim().toUpperCase())){

        exibirDebugMsg("JOGADORREDE>>INDICEJOGADORREDE>>" + (j-1));
                                                   indiceJogadorRede = j - 1;

                                               }

                                            }
                                            exibirDebugMsg("COMANDO_TRIM: " +
comando.trim());
                                            nomeJogadores[j++] = comando.trim();
```

```
                                                    if( nomeJogadores[j-
1].trim().toUpperCase().substring(8, nomeJogadores[j-
1].length()).trim().equals(nomeJogador.trim().toUpperCase())){

        exibirDebugMsg("JOGADORREDE>>INDICEJOGADORREDE>>" + (j-1));
                                                            indiceJogadorRede = j - 1;

                                                    }



                                                    frmEsperaTMP.append(new
StringItem("Jogadores na sala " + idSala + ":", ""));


        exibirDebugMsg("###################LISTA DE USUÁRIOS ON-
LINE################");
                                                    for(i = 0; i < NUMERO_JOGADORES;
i++){
                                                            exibirDebugMsg("JOGADOR: " +
nomeJogadores[i]);
                                                            if(nomeJogadores[i] != null)
                                                                    frmEsperaTMP.append("\n"
+ nomeJogadores[i]);
                                                    }

        exibirDebugMsg("#####################################################
########");

                                                    //frm.append(new StringItem("Jogadores
na sala:", comando.substring(15, comando.length())));

                                                    if(primeiroJogador){

        frmEsperaTMP.addCommand(cmdComecarJogo);
                                                    }


                                                    //Command cmdCancelar = new
Command("Cancelar", Command.SCREEN, 0);

        frmEsperaTMP.addCommand(cmdCancelar);


                                                    if(listenerEntradaJogador == null){
                                                            listenerEntradaJogador = new
CommandListener(){
```

```
                                                    public void
commandAction(Command c, Displayable d){
        if(c.getLabel().equals("Cancelar")){
        synchronized(controleJogo){
        controleJogo.setApresentacao(true);
        controleJogo.setTelaSelecao(true);
        controleJogo.JOGO_EM_ANDAMENTO = false;
        controleJogo.ONLINE = false;
        if(CartasMidlet.getDisplay() != null)
        CartasMidlet.getDisplay().setCurrent(controleJogo);
        enviarMsgServidor("SAIR_CLIENTE");
        controleJogo.fecharConexao();
                                                            }
                                                        }else
if(c.getLabel().equals("Começar jogo")){
        controleJogo.comecarJogoOnLine(true);
                                                        }
                                                    }
                                                };


                                        }

                                        frmEsperaTMP.setCommandListener(
                                                listenerEntradaJogador

                                        );

                                        System.gc();
        CartasMidlet.getDisplay().setCurrent(frmEsperaTMP);

                                    //}
```

```
                                        }else if(comando.startsWith("ST")){
                                                synchronized(controleJogo){

        if(!controleJogo.primeiroStatusRecebido){

        controleJogo.comecarJogoOnLine(false);
                                                }

        controleJogo.atualizarStatusJogo(comando);

                                                }
                                        }else
if(comando.startsWith("VENCEDOR_JOGO")){
                                                synchronized(controleJogo){
                                                        Mensagem m = controleJogo.new
Mensagem("Fim de jogo. O vencedor foi: " + comando.substring(14, comando.length()) ,
AlertType.INFO, controleJogo);
                                                        m.setTimeout(10000);

        controleJogo.setApresentacao(true);
                                                        controleJogo.setTelaSelecao(true);

        controleJogo.JOGO_EM_ANDAMENTO = false;
                                                        controleJogo.ONLINE = false;
                                                        //fecharConexao();
                                                        fecharConexaoSemConfirmacao();
                                                        System.gc();

        CartasMidlet.getDisplay().setCurrent(m);
                                                }
                                        }else if(comando.equals("PODE_JOGAR")){
                                                if(controleJogo.indiceJogadorRede ==
controleJogo.getIndiceJogadorAtual()){

        controleJogo.exibirDebugMsg("Jogador>>" +
controleJogo.getJogador(controleJogo.getIndiceJogadorAtual()).getNome() +
">>CONFIRMAÇÃO DE JOGADA RECEBIDA");

        controleJogo.setRecebeuPermissaoJogada(true);
                                                }
                                        }else if(comando.startsWith("SAIR_SALA")){
                                                synchronized(controleJogo){

        controleJogo.setApresentacao(true);
                                                        controleJogo.setTelaSelecao(true);

        controleJogo.JOGO_EM_ANDAMENTO = false;
```

```
                                                        controleJogo.ONLINE = false;
                                                        controleJogo.exibirMsg("Você foi
desconectado pelo servidor. Mensagem do servidor: " + comando + " Reinicializando
jogo.");
                                                        //fecharConexao();
                                                        fecharConexaoSemConfirmacao();
                                                }
                                        }else
if(comando.startsWith("RECOMECO_ENTRADA_JOGADOR")){
                                                synchronized(controleJogo){
                                                        int indice = comando.indexOf("]");
                                                        int indiceInicial =
"RECOMECO_ENTRADA_JOGADOR".length() + 1;
                                                        int i = 0;
                                                        NUMERO_JOGADORES = 4;
                                                        for(i = 0; i <
NUMERO_JOGADORES; i++)
                                                                nomeJogadores[i] = null;
                                                        int j = 0;
                                                        comando =
comando.substring(indiceInicial, comando.length() -1 );
                                                        indice = comando.indexOf("][");
                                                        while(indice != -1){
                                                                nomeJogadores[j++] =
comando.substring(0, indice);
                                                                comando =
comando.substring(indice + 2, comando.length());

        exibirDebugMsg("COMANDO_PARCIAL_ENTRADA: " + comando);
                                                                indice =
comando.indexOf("][");
                                                                if( nomeJogadores[j-
1].trim().toUpperCase().substring(8, nomeJogadores[j-
1].length()).trim().equals(nomeJogador.trim().toUpperCase())){

        exibirDebugMsg("JOGADORREDE>>INDICEJOGADORREDE>>" + (j-1));
                                                                        indiceJogadorRede =
j - 1;
                                                                }

                                                        }
                                                        nomeJogadores[j++] =
comando.trim();
                                                        if( nomeJogadores[j-
1].trim().toUpperCase().substring(8, nomeJogadores[j-
1].length()).trim().equals(nomeJogador.trim().toUpperCase())){
```

```
	exibirDebugMsg("JOGADORREDE>>INDICEJOGADORREDE>>" + (j-1));
								indiceJogadorRede = j - 1;

							}

	exibirDebugMsg("@@@@@@@@@@@@@@@@@@LISTA DE
JOGADORES(RECOMEÇO)@@@@@@@@@@@@@@");
							for(i = 0; i <
NUMERO_JOGADORES; i++){
								if(nomeJogadores[i] != null)

	exibirDebugMsg(nomeJogadores[i]);

							}

	exibirDebugMsg("@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@");



	controleJogo.comecarJogoOnLine(true);
						}

					}
				}

			}catch(Exception e){
				synchronized(controleJogo){

	exibirDebugMsg("ControleJogo>>threadLeitura>>Erro ao ler>>Fechando
conexão>>" + e);
					controleJogo.setApresentacao(true);
					controleJogo.setTelaSelecao(true);
					controleJogo.JOGO_EM_ANDAMENTO = false;
					controleJogo.ONLINE = false;
					controleJogo.exibirMsg("Ocorreu um problema
de comunicação com o servidor. Isto ocorre quando há falhas no meio de comunicação
ou quando os outros jogadores da sala já saíram antes do início do jogo.");
					fecharConexaoSemConfirmacao();

				}
			}

			System.gc();
		}
```

```
                    exibirDebugMsg("ControleJogo>>threadLeitura>>Finalizando thread
de leitura");


                }
        }

        /**
                Conecta com o servidor e envia uma requisição. Se manterConexao for
igual a true, <p> mantém a conexão
                com o servidor aberta e inicializa a thread de leitura de comandos, <p>
senão fecha a conexão após receber
                a resposta ao comando enviado.
        */

        public static String retornarMsgServidor(String URL, String str, boolean
manterConexao, ControleJogo controleJogo){
                exibirDebugMsg("ControleJogo>>retornarMsgServidor>>" + URL + " " +
str);
                StringBuffer resultado = new StringBuffer("");

                fecharConexao();

                try{

                        conexao = (SocketConnection)Connector.open(URL);
                        conexao.setSocketOption(SocketConnection.KEEPALIVE, 1);
                        is  = conexao.openInputStream();
                        os = conexao.openOutputStream();
                        socketWriter = new PrintStream(os);
                        socketWriter.println(str);
                        socketWriter.flush();
                        int ch = 0;
                        if(!manterConexao){
                                while(ch != -1 && ((char)ch) != '\n') {
                                        ch = is.read();
                                        resultado.append((char)ch);
                                }
                                conexao.close();
                        }else{
                                setConectado(true);
                                new threadLeitura(controleJogo);
                                tAck = new Thread(new threadAck(controleJogo));
                                tAck.start();

                        }
```

```
		}catch(Exception e){
			exibirDebugMsg("ControleJogo>>retornarMsgServidor>>ERRO: " +
e);
			fecharConexao();
			return "ERRO";
		}
		return resultado.toString().trim();
	}



	/**
		Retorna a string de conexão com o servidor
	*/
	public static String getHostServidor(){
		return "socket://" + IP_SERVIDOR + ":" + PORTA_SERVIDOR;
	}

	/**
		Mostra uma mensagem na tela do usuário
	*/
	public void exibirMsg(String msg){
		msgErro = new Mensagem(msg, AlertType.INFO, this);
		setApresentacao(true);
		setTelaSelecao(true);
		JOGO_EM_ANDAMENTO = false;
		CartasMidlet.getDisplay().setCurrent(msgErro);

	}



	/**
		Fecha a conexão com o servidor
	*/

	public synchronized static void fecharConexao(){

				try{

					if (socketWriter != null){
						socketWriter.println("SAIR_CLIENTE");
					}
```

```
                        }catch(Exception e4){

        exibirDebugMsg("ControleJogo>>retornarMsgServidor>>ERRO AO FECHAR
CONEXÃO (socketWriter): " + e4);
                        }



                        try{

                                if (conexao != null)
                                conexao.close();

                        }catch(Exception e3){

        exibirDebugMsg("ControleJogo>>retornarMsgServidor>>ERRO AO FECHAR
CONEXÃO (C): " + e3);
                        }

                        setConectado(false);


        }



        /**
                Fecha a conexão com o servidor, mas não envia
                mensagem informando a saída do cliente (Utilizar somente
                quando processar msgs do servidor)
        */

        public synchronized static void fecharConexaoSemConfirmacao(){

                        /*try{

                                if (socketWriter != null){
                                        socketWriter.println("SAIR_CLIENTE");
                                }


                        }catch(Exception e4){

        exibirDebugMsg("ControleJogo>>retornarMsgServidor>>ERRO AO FECHAR
CONEXÃO (socketWriter): " + e4);
                        }*/
```

```
                        try{

                                if (conexao != null)
                                conexao.close();

                        }catch(Exception e3){

        exibirDebugMsg("ControleJogo>>retornarMsgServidor>>ERRO AO FECHAR
CONEXÃO (C): " + e3);
                        }

                        setConectado(false);


        }


        public static synchronized void setConectado(boolean b){
                CONECTADO = b;
                ONLINE=false;
        }

        public static synchronized boolean getConectado(){
                return CONECTADO;
        }

        /**
                Envia uma mensagem de texto para o servidor
        */
        public static synchronized void enviarMsgServidor(String msg){
                try{
                        socketWriter.println(msg);
                        socketWriter.flush();
                }catch(Exception e){
                        exibirDebugMsg("ControleJogo>>enviarMsgServidor>>Erro ao
enviarMsg>>" + e);
                        fecharConexao();
                }
        }

        /**
                Monta o quadro contendo o status do jogo
        */
        public synchronized String montarStatusJogo(Mensagem mensagem, boolean
fecharJogo){
```

```
		/*
			Formato do quadro:
				ST B NV NV ... NV J NV NV ... NV J NV NV ... NV J NV NV ...
NV J NV NV ... NV MM
				SJ MSG JA PA NE FJ FST
			ST --> Indica o tipo do quadro ("Status")
			NV --> Naipe e valor de cada carta do baralho
			B --> Início das cartas do baralho
			J -->Início da carta de um jogador
			OMM --> Indica se o jogo é OpenMauMau ou não
			SJ --> Sentido do jogo
			MSG --> Mensagem
			QC --> Quantidade de cartas para compra
			JA --> Jogador atual
			PA --> Penalidade aplicada
			NE --> Naipe Escolhido
			FJ --> Fechar Jogo
			FST --> Fim do quadro de status

		*/
		StringBuffer quadro = new StringBuffer("STB");
		int numeroCartas = baralhoAtual.numeroCartas();
		//Adiconando cartas do baralho
		if(numeroCartas > 0){
			for(int i = 0; i < numeroCartas; i++){
				Carta carta = baralhoAtual.carta(i);
				quadro.append("N" + carta.getNaipe());
				quadro.append("V" + carta.getValor());

			}
		}

		//Adicionando cartas dos jogadores
		for(int j = 0; j < NUMERO_JOGADORES; j++){
			if(jogadores[j] != null){
				Jogador jogadorAtual = jogadores[j];
				quadro.append("J");
				numeroCartas = jogadorAtual.numeroCartas();
				for(int i = 0; i < numeroCartas; i++){
					Carta carta = jogadorAtual.carta(i);
					quadro.append("N" + carta.getNaipe());
					quadro.append("V" + carta.getValor());
				}
			}
		}

		quadro.append("OMM" + getOpenMauMau());
```

```
                quadro.append("SJ" + SENTIDO_JOGO);
                quadro.append("MSG" + mensagem.getString());
                quadro.append("QC" + QUANT_COMPRA);
                quadro.append("JA" + jogadorAtual);
                quadro.append("PA" + getPenalidadeAplicada());
                quadro.append("NE" + NAIPE_ESCOLHIDO);

                quadro.append("FJ" + fecharJogo);

                quadro.append("FST");
                System.gc();
                exibirDebugMsg("ControleJogo>>montarStatusJogo(Mensagem)>>\n" +
quadro.toString().trim());
                return quadro.toString().trim();
        }

        /**
                Atualiza o status do jogo de acordo com o quadro recebido do servidor

        */
        public synchronized void atualizarStatusJogo(String msg){
                /*
                        Formato do quadro:
                                ST B NV NV ... NV J NV NV ... NV J NV NV ... NV J NV NV ...
NV J NV NV ... NV OMM
                                SJ MSG JA FJ FST
                        ST --> Indica o tipo do quadro ("Status")
                        NV --> Naipe e valor de cada carta do baralho
                        B --> Início das cartas do baralho
                        J -->Início da carta de um jogador
                        OMM --> Indica se o jogo é OpenMauMau ou não
                        SJ --> Sentido do jogo
                        MSG --> Mensagem
                        QC --> Quantidade de cartas para compra
                        JA --> Jogador atual
                        PA --> Penalidade Aplicada
                        NE --> Naipe Escolhido
                        FJ --> Fechar Jogo
                        FST --> Fim do quadro de status

                */
                RECEBEU_PERMISSAO_JOGADA = false;

                exibirDebugMsg("JOGADOR_REDE>>" +
jogadores[indiceJogadorRede].getNome() +
">>ControleJogo>>atualizarStatusJogo()>>Atualizando status do jogo");
```

```
		exibirDebugMsg("ControleJogo>>atualizarStatusJogo()>>Atualizando
status do jogo>>Atualizando baralho");
		String strCartasBaralho = msg.substring(msg.indexOf("B") + 1,
msg.indexOf("J"));
		exibirDebugMsg("ControleJogo>>atualizarStatusJogo()>>Atualizando
status do jogo>>Atualizando baralho>>" + strCartasBaralho);

		/*******************************************************Atualização do
baralho*****************************************************************/
		baralhoAtual.zerar();

		int naipe, valor;
		StringBuffer strNaipe = new StringBuffer();
		StringBuffer strValor = new StringBuffer();

		while(strCartasBaralho.length() > 0){
			int i = 1;
			strNaipe.setLength(0);
			strValor.setLength(0);

			while(strCartasBaralho.charAt(i) != 'V')
				strNaipe.append(strCartasBaralho.charAt(i++));

			i++;
			while(i < strCartasBaralho.length() && strCartasBaralho.charAt(i) !=
'N' )
				strValor.append(strCartasBaralho.charAt(i++));

			naipe = Integer.parseInt(strNaipe.toString().trim());
			valor = Integer.parseInt(strValor.toString().trim());

	exibirDebugMsg("ControleJogo>>atualizarStatusJogo()>>Atualizando status do
jogo>>Atualizando baralho");
			exibirDebugMsg("ADICIONANDO CARTA AO BARALHO " +
Baralho.cartasBaralho[naipe][valor].toString());

			baralhoAtual.adicionarCarta(Baralho.cartasBaralho[naipe][valor]);
			if( i < strCartasBaralho.length()){
				strCartasBaralho = strCartasBaralho.substring(i,
strCartasBaralho.length());
			}else{
				strCartasBaralho = "";
			}
		}
		exibirDebugMsg("ControleJogo>>atualizarStatusJogo()>>Atualizando
status do jogo>>Atualizando baralho>>BARALHO ATUALIZADO");
		//CartasMidlet.getDisplay().setCurrent(baralhoAtual);
```

```
        /**********************************************************************************************
*************************************************/


                /*******************************************************Atualização dos
jogadores*********************************************************************/
                msg = msg.substring(msg.indexOf("J"), msg.length());

                exibirDebugMsg("ControleJogo>>comecarJogoOnLine>>Atualizando
jogadores>>"+ msg);
                for(int j = 0; j < NUMERO_JOGADORES; j++){

        exibirDebugMsg("ControleJogo>>comecarJogoOnLine>>Atualizando
jogadores>>JOGADOR:" + jogadores[j].getNome());
                        jogadores[j].zerar();

                        int indiceProximoJogador = msg.indexOf("J", 2);

                        if(indiceProximoJogador == -1 || msg.charAt(indiceProximoJogador -
1) == 'S'){
                                indiceProximoJogador = msg.indexOf("O");
                                System.out.println("INDICEPROXIMOJOGADOR: " +
indiceProximoJogador);
                        }
                        String cartasJogadorAtual = msg.substring(0,
indiceProximoJogador);


        exibirDebugMsg("ControleJogo>>comecarJogoOnLine>>Atualizando
jogadores>>CARTASJOGADORATUAL>>" + cartasJogadorAtual);

                        int i = 1;
                        while(i < cartasJogadorAtual.length()){
                                i++;
                                strNaipe.setLength(0);
                                strValor.setLength(0);
                                while(cartasJogadorAtual.charAt(i) != 'V'){
                                        strNaipe.append(cartasJogadorAtual.charAt(i++));
                                }
                                i++;
                                while(i < cartasJogadorAtual.length() &&
cartasJogadorAtual.charAt(i) != 'N' )
                                        strValor.append(cartasJogadorAtual.charAt(i++));

                                naipe = Integer.parseInt(strNaipe.toString().trim());
```

```
                        valor = Integer.parseInt(strValor.toString().trim());

                        exibirDebugMsg("ADICIONANDO CARTA AO JOGADOR " +
jogadores[j].getNome() + ": " + Baralho.cartasBaralho[naipe][valor].toString());

        jogadores[j].adicionarCarta(Baralho.cartasBaralho[naipe][valor]);
                }

                msg = msg.substring(indiceProximoJogador, msg.length());

        exibirDebugMsg("ControleJogo>>comecarJogoOnLine>>Atualizando
jogadores>>CARTASJOGADORATUAL>>MSG>>" + msg);


        exibirDebugMsg("ControleJogo>>atualizarStatusJogo()>>Atualizando status do
jogo>>Atualizando JOGADORES>>JOGADORES ATUALIZADOS");




        }


        /**********************************************************************************************
**************************************************/


                /*************************************************************Status do
jogo************************************


                        OMM --> Indica se o jogo é OpenMauMau ou não
                        SJ --> Sentido do jogo
                        MSG --> Mensagem
                        QC --> Quantidade de cartas para compra
                        JA --> Jogador atual
                        PA --> Penalidade Aplicada
                        NE --> Naipe Escolhido
                        FJ --> Fechar Jogo
                        FST --> Fim do quadro de status
                */


                String OMM = msg.substring(3, msg.indexOf("SJ"));
                exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>OMM: " + OMM);
                if(OMM.equals("true"))
```

```
                OPEN_MAU_MAU = true;
            else
                OPEN_MAU_MAU = false;


            msg = msg.substring(msg.indexOf("SJ"), msg.length());
            exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>SJ: " +
msg.substring(2, msg.indexOf("MSG")));

            SENTIDO_JOGO = (byte)Integer.parseInt(msg.substring(2,
msg.indexOf("MSG")));

            msg = msg.substring(msg.indexOf("MSG"), msg.length());
            exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>MSG: " +
msg.substring(3, msg.indexOf("QC"))); //MSG
            String mensagem = msg.substring(3, msg.indexOf("QC"));

            msg = msg.substring(msg.indexOf("QC"), msg.length());
            exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>QC: " +
msg.substring(2, msg.indexOf("JA"))); //QC
            QUANT_COMPRA = (byte)Integer.parseInt(msg.substring(2,
msg.indexOf("JA")));

            msg = msg.substring(msg.indexOf("JA"), msg.length());
            exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>JA: " +
msg.substring(2, msg.indexOf("PA"))); //JA
            jogadorExibido = jogadorAtual = (byte)Integer.parseInt(msg.substring(2,
msg.indexOf("PA")));

            msg = msg.substring(msg.indexOf("PA"), msg.length());
            exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>PA: " +
msg.substring(2, msg.indexOf("NE"))); //PA
            setPenalidadeAplicada( msg.substring(2,
msg.indexOf("NE")).equals("true") ? true: false );

            msg = msg.substring(msg.indexOf("NE"), msg.length());
            exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>NE: " +
msg.substring(2, msg.indexOf("FJ"))); //NE
            NAIPE_ESCOLHIDO = (byte)Integer.parseInt(msg.substring(2,
msg.indexOf("FJ")));


            msg = msg.substring(msg.indexOf("FJ"), msg.length());
            exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>FJ: " +
msg.substring(2, msg.indexOf("FST"))); //FJ
```

```
                msg = msg.substring(msg.indexOf("FST"), msg.length());
                exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>FST: " + msg);
//FST

                int vencedor = mensagem.indexOf("vencedor");
                Mensagem m = null;
                if(vencedor != -1){ //Vencedor do jogo detectado
                        fecharConexao();
                        setTelaSelecao(true);
                        setApresentacao(true);
                        JOGO_EM_ANDAMENTO = false;
                        m = new MensagemAtualizacao(mensagem, AlertType.INFO, this);
                }else{
                        m = new MensagemAtualizacao(mensagem, AlertType.INFO,
jogadores[jogadorAtual]);

                }

                System.gc();


                //if(indiceJogadorRede != getIndiceJogadorAnterior())
                CartasMidlet.getDisplay().setCurrent(m);

                if(indiceJogadorRede == getIndiceJogadorAtual()){
                        enviarMsgServidor("ACKATUALIZACAO");
                }
                //CartasMidlet.getDisplay().setCurrent(m);
                exibirDebugMsg("ControleJogo>>atualizarStatusJogo>>Status atualizado
com sucesso");


        }

        /**
                Verifica a integridade entre os procedimentos atualizarStatusJogo e
montarStatusJogo
        */
        public void loopback(){
                atualizarStatusJogo(montarStatusJogo(new Mensagem("teste",
AlertType.INFO, jogadores[0]), false));

        }

        /**
                Modifica o valor da variável PenalidadeAplicada
        */
```

```
public synchronized void setPenalidadeAplicada(boolean b){
	if(b)
		setQuantidadeCompra((byte)0);
	PENALIDADE_APLICADA = b;
}

/**
	Retorna o valor da variável penalidadeAplicada
*/
public synchronized boolean getPenalidadeAplicada(){
	return PENALIDADE_APLICADA;
}

/**
	Retorna o índice do jogador atual
*/
public int getIndiceJogadorAtual(){
	return jogadorAtual;
}

/**
	Modifica o endereço ip do servidor
*/
public static void setIPServidor(String IP){
	IP_SERVIDOR = IP;
}

/**
	Modifica a porta do servidor
*/
public static void setPortaServidor(String porta){
	PORTA_SERVIDOR = porta;
}

/**
	Retorna o endereço ip do servidor
*/
public static String getIPServidor(){
	return IP_SERVIDOR;
}

/**
	Retorna a porta do servidor
*/
public static String getPortaServidor(){
	return PORTA_SERVIDOR;
}
```

```
	/**
		Incrementa o índice do jogo atual
	*/
	public synchronized void incrementaJogoAtual(){
		jogoAtual++;
		exibirDebugMsg("ControleJogo>>INCREMENTANDO ÍNDICE DO JOGO
ATUAL: " + jogoAtual);
	}

	/**
		Retorna o índice do jogo atual
	*/
	public synchronized int getJogoAtual(){
		return jogoAtual;
	}

}
```

<u>Dipositivo.java</u>

```
/*
Classe: jogo.apibasica.Dispositivo
Objetivo: Guarda informações sobre o dispositivo (largura da tela, altura da tela,
	coordenadas X e Y do centro da tela)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 04/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Tornando dados estáticos (OBS: A classe dave ser instanciada para
preencher estes dados
		corretamente)
Status: OK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/
package jogo.apibasica;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.game.*;
```

```
/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis

<p>Classe: jogo.apibasica.Dispositivo
<p>Objetivo: Guarda informações sobre o dispositivo (largura da tela, altura da tela,
        coordenadas X e Y do centro da tela)

@author Rogério de Paula Aguilar
@version 1.0
*/
public class Dispositivo{
        /**
                Display do dispositivo
        */
        private Display displayDispositivo;

        /**
                Coordenadas da tela
        */
        private static int X_CENTRO, Y_CENTRO, larguraTela, alturaTela;

        /**
                Canvas utilizado apenas para obter coordenadas da tela
        */
        private Canvas canvas = new GameCanvas(false){

                public void paint(Graphics g){}
        };

        /**
                Construtor
        */
        public Dispositivo(Display display){
                if(display == null){
                  throw new IllegalArgumentException("Dispositivo>>Dispositivo(Display
display)>>Display deve ser diferente de null!");
                }else{
                  displayDispositivo = display;
                  Displayable d = display.getCurrent();
                  display.setCurrent(canvas);
                  X_CENTRO = canvas.getWidth() / 2;
                  Y_CENTRO = canvas.getHeight() / 2;
                  larguraTela = canvas.getWidth();
```

```
                alturaTela = canvas.getHeight();
                display.setCurrent(d);


                d = null;
                canvas = null;
                System.gc();

            }
        }

        /**
            Retorna a largura da tela
        */
        public static int getLarguraTela(){
            return larguraTela;
        }

        /**
            Retorna a altura da tela
        */
        public static int getAlturaTela(){
            return alturaTela;
        }

        /**
            Retorna a coordenada X do centro da tela
        */

        public static int getCentroX(){
            return X_CENTRO;
        }

        /**
            Retorna a coordenada Y do centro da tela
        */

        public static int getCentroY(){
            return Y_CENTRO;
        }


}
```

frmAjuda.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.frmAjuda
Responsabilidades: Exibir a ajuda para o usuário

*************************************Alterações*************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 22/08/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementação do formulário de ajuda e término da apresentação
Status: pendente (falta terminar o formulário de ajuda)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/
package jogo;

import javax.microedition.lcdui.*;
import jogo.apibasica.*;


/**
<p>2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.frmAjuda
<p>Responsabilidades: Exibir a ajuda para o usuário
@author Rogério de Paula Aguilar
@version 1.0
*/

public class frmAjuda extends Form implements CommandListener{

	/**
		Tela de volta
	*/
	private Displayable telaVolta;

	/**
		Comando utilizado para voltar para a tela anterior
```

```
	*/
	private Command cmdVoltar = new Command("Voltar", Command.SCREEN, 0);

	/**
		Construtor
	*/

	public frmAjuda(String titulo, Displayable d, Display display){
		super(titulo);
		if(getTicker() == null)
			setTicker(new Ticker("2003 - Rogério de Paula Aguilar
(rogeriopaguilar@terra.com.br) - Luiz Fernando P.S. Forgas (luizforgas@terra.com.br) -
Trabalho de Conclusão de Curso"));
		if(d == null || display ==null)
			throw new IllegalArgumentException("Displayable e Display devem
ser diferentes de nul!");
		telaVolta = d;
		StringItem strRegras = new StringItem("Regras do jogo Mau-Mau", "");
		strRegras.setLayout(Item.LAYOUT_CENTER);
		append(strRegras);
		append(new StringItem("\nRegra geral: ", "O jogador deve jogar uma carta
do mesmo naipe ou valor da carta que estiver no baralho no momento da sua jogada,
ou jogar uma das cartas especiais. O jogador que ficar sem cartas primeiro vence o
jogo. Se o baralho ficar vazio, o jogador que tiver o menor número de cartas é
considerado vencedor."));
		append(new StringItem("Cartas especiais (* indica que a regra é válida
para todos os naipes)", ""));
		append(new StringItem(" A*"," o jogador que jogar um AS pode jogar
novamente."));
		append(new StringItem(" 2*"," carta utilizada para defender jogadas hostis
(ver próximas regras)"));
		append(new StringItem(" 4 de ESPADAS"," todo mundo (excluindo o
jogador que jogou esta carta) na mesa compra uma carta. Esta jogada é
indefensável."));
		append(new StringItem(" 5*"," esta carta inverte o sentido do jogo."));
		append(new StringItem(" 7*"," o próximo jogador deve comprar três cartas,
jogar outro sete (quando um sete é jogado sobre outro, a quantidade de cartas para
compra será a quantidade atual mais 3 e a penalidade é passada para o próximo
jogador). Pode ser defendido por um 2 do mesmo naipe do sete atual, por um coringa
ou por uma valete."));
		append(new StringItem(" 9*"," o próximo jogador deve comprar uma carta.
Pode ser defendido pelo dois do mesmo naipe do nove que está no baralho, por um
coringa ou por uma valete."));
		append(new StringItem(" 10 de PAUS"," o próximo jogador deve comprar
cinco cartas. Pode ser defendido pelo dois de paus, por um coringa ou por uma
valete."));
```

```
append(new StringItem(" C(CORINGA)*"," pode ser jogado sobre qualquer carta e anula o efeito de qualquer carta especial."));
append(new StringItem(" V(Valete)*"," pode ser jogado sobre qualquer carta e anula o efeito de qualquer outra carta especial. Ao jogar, o jogador escolhe o naipe que deve ser utilizado na próxima jogada."));
append(new StringItem(" D(DAMA)*"," pula o próximo jogador."));
append(new StringItem(" R(REI)*"," faz o jogador anterior comprar uma carta. Esta jogada é indefensável."));
append(new StringItem(" \n"," Quando o jogador tiver apenas duas cartas, antes de jogar sua penúltima carta ele deve dizer \"Mau-Mau\" ou receberá três cartas como penalidade. O jogador só pode dizer \"Mau-Mau\" nesta situação, senão também será penalizado com três cartas."));

append(new StringItem(" OBSERVAÇÃO:"," Se o jogador atual paga uma penalidade, a mesma não é passada para o próximo jogador. Exemplo: Se existe um sete de paus e o jogador atual compra três cartas (que é a penalidade do sete), o próximo jogador não precisa comprar três cartas, pois o efeito do sete já foi aplicado no jogador anterior. Nestes casos, basta seguir a regra geral."));



StringItem strDescricaoInterface = new StringItem("Descrição da interface de jogo", "");

strDescricaoInterface.setLayout(Item.LAYOUT_CENTER|Item.LAYOUT_NEWLINE_AFTER|Item.LAYOUT_NEWLINE_BEFORE);
append(strDescricaoInterface);
append(new StringItem("", "A tela do jogador apresenta as cartas do mesmo. Selecionando a tecla que aponta para cima, o jogador pode visualizar o baralho. Estando na tela do baralho, se o usuário selecionar a tecla baixo, é mostrada a tela anterior. Estando na tela de um jogador, se o usuário selecionar a tecla baixo, o jogo irá mostrar a tela do próximo jogador, e selecionando as teclas esquerda ou direita é possível navegar pelas cartas do jogador."));



StringItem strDescricaoMenus = new StringItem("Descrição dos menus", "");

strDescricaoMenus.setLayout(Item.LAYOUT_CENTER|Item.LAYOUT_NEWLINE_AFTER|Item.LAYOUT_NEWLINE_BEFORE);
append(strDescricaoMenus);
append(new StringItem("Jogar: ", "Inicia um novo jogo contra o computador."));
append(new StringItem("Jogar on-line: ", "Inicia um novo jogo on-line contra outras pessoas que estejam conectadas ao servidor."));
append(new StringItem("Visualizador de cartas: ", "Exibe um formulário onde é possível visualizar todas as cartas do baralho."));
```

```
            append(new StringItem("Opções: ", "Exibe opções de configuração do
jogo. O modeo Open Mau-Mau permite que os jogadores vejam as suas cartas e a dos
outros jogadores. Se esta opção não estiver selecionada, os jogadores podem ver
apenas as suas cartas. O endereço do servidor e a porta especificam onde o jogo
tentará se conectar para o modo jogo on-line."));
            append(new StringItem("Ajuda: ", "Exibe esta tela."));


            StringItem strDescricaoJogoOnLine= new StringItem("Iniciando um jogo
on-line", "");

        strDescricaoJogoOnLine.setLayout(Item.LAYOUT_CENTER|Item.LAYOUT_NEW
LINE_AFTER|Item.LAYOUT_NEWLINE_BEFORE);
            append(strDescricaoJogoOnLine);
            append(new StringItem("", "Para iniciar um jogo on-line, à partir da tela de
apresentação do jogo, selecione a opção jogo on-line. Se aparecer uma mensagem
dizendo que você estará entrando num ambiente de rede, apenas confirme esta
mensagem. O programa tentará se conectar ao servidor. Se ocorrer algum erro na
conexão, verifique, através do menu opções, se o endereço e a porta do servidor estão
corretos. Se estiverem, o servidor pode estar desligado ou com problemas. Neste caso,
tente novamente mais tarde. Se não ocorrerem problemas, será apresentado um
formulário contendo uma lista de salas disponíveis para o jogo. Selecione uma das
salas, digite o seu apelido e clique na opção entrar na sala. Aparecerá uma tela
informando quem está na sala. Se você foi o primeiro a entrar na sala de jogo, você
será notificado quando alguém entrar na sala, e poderá começar o jogo através da
opção começar jogo (esta opção só aparecerá a partir do momento que mais alguém
entrar na sala). Se você não foi o primeiro a entrar na sala, deve aguardar para que o
primeiro jogador que entrou na sala comece o jogo."));



            addCommand(cmdVoltar);
            setCommandListener(this);
        }

        /**
            Método invocado quando o jogador seleciona algum comando
        */

        public void commandAction(Command c, Displayable d){
            if(c == cmdVoltar)
                if(CartasMidlet.getDisplay() != null){
                    if(telaVolta instanceof ControleJogo){
                        ((ControleJogo)telaVolta).setApresentacao(true);
                        ((ControleJogo)telaVolta).setTelaSelecao(true);
                    }
                    System.gc();
```

```
                    CartasMidlet.getDisplay().setCurrent(telaVolta);

                }
        }

}
```

frmEscolhaNaipe.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.frmfrmEscolhaNaipe
Responsabilidades: permite que o usuário escolha o naipe da próxima carta (Valete(*))

*/
package jogo;


import javax.microedition.lcdui.*;
import jogo.apibasica.*;
import java.util.*;
import java.io.*;
import javax.microedition.io.*;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.frmfrmEscolhaNaipe
<p>Responsabilidades: permite que o usuário escolha o naipe da próxima carta
(Valete(*))

@author Rogério de Paula Aguilar
@version 1.0
*/


public class frmEscolhaNaipe extends Form implements CommandListener{

        /**
                Tela de volta
        */
        private JogadorHumano telaVolta;

        /**
```

```
		Comando utilizado para confirmar o naipe escolhido
	*/
	private Command cmdEscolherNaipe = new Command("Escolher Naipe",
Command.SCREEN, 0);

	/**
		Mensagem
	*/
	private ControleJogo.Mensagem m;

	/**
		Mensagem que fica aparecendo na tela do usuário
	*/
	private Ticker ticker = new Ticker("Quando você joga uma valete, deve escolher o
naipe da próxima jogada. Selecione o naipe e clique em Escolher Naipe");

	/**
		Exibe os naipes para serem escolhidos
	*/
	private ChoiceGroup choice;

	private ControleJogo controleJogo;

	/**
		Construtor
	*/
	public frmEscolhaNaipe(JogadorHumano d, ControleJogo c,
ControleJogo.Mensagem m){
		super("");
		setTicker(ticker);
		if(d == null)
			throw new IllegalArgumentException("Displayable deve ser diferente
de null!");
		telaVolta = d;
		addCommand(cmdEscolherNaipe);
		setCommandListener(this);
		choice = new ChoiceGroup("Naipes:", Choice.EXCLUSIVE);
		choice.append("COPAS", null);
		choice.append("PAUS", null);
		choice.append("ESPADAS", null);
		choice.append("OUROS", null);
		this.m = m;
		append(choice);
		controleJogo = c;

	}
```

```
	/**
		Método invocado quando o jogador seleciona algum comando
	*/
	public void commandAction(Command c, Displayable d){
		String strNaipe = "";
		if(choice.getSelectedIndex() == 0){
			controleJogo.NAIPE_ESCOLHIDO = Carta.COPAS;
			strNaipe = "COPAS";
		}else if(choice.getSelectedIndex() == 1){
			controleJogo.NAIPE_ESCOLHIDO = Carta.PAUS;
			strNaipe = "PAUS";
		}else if(choice.getSelectedIndex() == 2){
			controleJogo.NAIPE_ESCOLHIDO = Carta.ESPADAS;
			strNaipe = "ESPADAS";
		}else if(choice.getSelectedIndex() == 3){
			controleJogo.NAIPE_ESCOLHIDO = Carta.OUROS;
			strNaipe = "OUROS";

		}
		System.out.println("COMMAND_ACTION" + m.getString() +  " O próximo
jogador será: " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + " e
deverá jogar uma carta do naipe: " + strNaipe + ". Movendo para o próximo jogador.");
		m.setString(m.getString() +  " O próximo jogador será: " +
controleJogo.getJogador(controleJogo.getJogadorAtual()).getNome() + " e deverá jogar
uma carta do naipe: " + strNaipe + ". Movendo para o próximo jogador.");
		telaVolta.jogar(new Object[]{m});

	}


}
```

frmEspera.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.frmEspera
Responsabilidades: Exibir uma tela para o usuário enquanto este aguarda o término de
algum processo do jogo

**************************************Alterações**************************************
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 15/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementação do formulário de espera
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/
package jogo;

import javax.microedition.lcdui.*;
import jogo.apibasica.*;


/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.frmEspera
<p>Responsabilidades: Exibir uma tela para o usuário enquanto este aguarda o término
de algum processo do jogo

@author Rogério de Paula Aguilar
@version 1.0

*/


public class frmEspera extends Form {

	/**
		Mensagem que fica aparecendo na tela do usuário
	*/
	private Ticker tk = new Ticker("Aguarde. Enviando informações para o servidor.");

	private ImageItem img;


	/**
		Construtor
	*/
	public frmEspera(){
```

```
                super("");
                setTicker(tk);
                try{
                        img = new
ImageItem("",Image.createImage("/conexao.png"),ImageItem.LAYOUT_CENTER,"");
                }catch(Exception e){
                        ControleJogo.exibirDebugMsg("frmEspera>>static>>Erroa ao
carregar conexao.png");
                }
                append(img);

        }

        /**
                COnstrutor
        */
        public frmEspera(String msg){
                super("");
                tk = new Ticker(msg);
                setTicker(tk);

                try{
                        img = new
ImageItem("",Image.createImage("/conexao.png"),ImageItem.LAYOUT_CENTER,"");
                }catch(Exception e){
                        ControleJogo.exibirDebugMsg("frmEspera>>static>>Erroa ao
carregar conexao.png");
                }

                append(img);
        }

        /**
                Retorna a imagem associada a esta tela de espera
        */
        public ImageItem getImage(){
                return img;
        }


        /**
                Retira todos os itens do formulário
        */
        public void reset(String msg){
                deleteAll();
                tk = getTicker();
```

```
            if(tk == null){
                tk = new Ticker(msg);
            }else{
                tk.setString(msg);
            }
            setTicker(tk);


            append(img);


        }


}
```

frmListaSalas.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.frmListaSalas
Responsabilidades: Exibe as salas para o usuário

**************************************Alterações*************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Data: 13/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementação do formulário
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Data: 15/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando opção de entrar na sala
           Adicionando campo apelido
Status: ok
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/
package jogo;


import javax.microedition.lcdui.*;
import jogo.apibasica.*;
import java.util.*;
import java.io.*;
import javax.microedition.io.*;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.frmListaSalas
<p>Responsabilidades: Exibe as salas para o usuário e um campo para digitar o
apelido, quando o
                              <br>jogador estiver começando um jogo on-line

@author Rogério de Paula Aguilar
@version 1.0
*/

public class frmListaSalas extends Form implements CommandListener{

        /**
                Tela de volta
        */
        private Displayable telaVolta;

        /**
                Comando utilizado para retornar para a tela anterior
        */
        private Command cmdVoltar = new Command("Voltar", Command.SCREEN, 0);

        /**
                Comando utilizado para entrar na sala selecionada
        */
        private Command cmdEntrarSala = new Command("Entrar na sala",
Command.SCREEN, 0);

        /**
```

```
		Mensagem que fica aparecendo na tela do jogador
	*/
	private Ticker ticker = new Ticker("Selecione a sala, digite o apelido e clique na
opção entrar na sala");

	/**
		Vetor que guarda a lista de salas
	*/
	private Vector vetorSalas;

	/**
		Exibe a lista de salas
	*/
	private ChoiceGroup choice;

	/**
		Texto para o jogador digitar o apelido
	*/
	private TextField txtApelido = new TextField("Apelido:", "", 15, TextField.ANY);

	/**
		Mensagem de erro
	*/
	private Alert alertaApelido;

	/**
		Mensagem de erro
	*/
	private Alert alertaErro;
	private ControleJogo controleJogo;

	/**
		Construtor
	*/
	public frmListaSalas(String titulo, Displayable d, String salas, ControleJogo c){
		super(titulo);
		setTicker(ticker);
		if(d == null || salas == null)
			throw new IllegalArgumentException("Displayable e Display devem
ser diferentes de null!");
		telaVolta = d;
		addCommand(cmdEntrarSala);
		addCommand(cmdVoltar);
		setCommandListener(this);
		vetorSalas = new Vector();
		int i = 0, j = 0;
		while( (i = salas.indexOf("-", j)) != -1){
```

```
                String salaAtual = salas.substring(j, i);
                j = i + 1;
                vetorSalas.addElement(salaAtual);
            }
            vetorSalas.addElement(salas.substring(j, salas.length()));
            vetorSalas.trimToSize();
            choice = new ChoiceGroup("", Choice.EXCLUSIVE);
            for(i = 0; i < vetorSalas.size(); i++){
                String strSala = (String)vetorSalas.elementAt(i);
                int indice = strSala.indexOf("[");
                choice.append( strSala.substring(0, indice) + " - " +
strSala.substring(indice + 1, strSala.indexOf("])) + " jogadores", null);
            }
            append(txtApelido);
            append(choice);
            controleJogo = c;

        }

        /**
            Apaga o formulário coloca a nova lista de salas
        */
        public void reset(String salas){
            deleteAll();
            vetorSalas = new Vector();
            int i = 0, j = 0;
            while( (i = salas.indexOf("-", j)) != -1){
                String salaAtual = salas.substring(j, i);
                j = i + 1;
                vetorSalas.addElement(salaAtual);
            }
            vetorSalas.addElement(salas.substring(j, salas.length()));
            vetorSalas.trimToSize();
            choice = new ChoiceGroup("", Choice.EXCLUSIVE);
            for(i = 0; i < vetorSalas.size(); i++){
                String strSala = (String)vetorSalas.elementAt(i);
                int indice = strSala.indexOf("[");
                choice.append( strSala.substring(0, indice) + " - " +
strSala.substring(indice + 1, strSala.indexOf("])) + " jogadores", null);
            }
            append(txtApelido);
            append(choice);
            System.gc();
        }

        /**
            Método que é invocado quando o jogador seleciona algum comando
```

```
	*/
	public void commandAction(Command c, Displayable d){
		if(c == cmdVoltar){
			if(CartasMidlet.getDisplay() != null){
				if(telaVolta instanceof ControleJogo){
					((ControleJogo)telaVolta).setApresentacao(true);
					((ControleJogo)telaVolta).setTelaSelecao(true);
					ControleJogo.JOGO_EM_ANDAMENTO = false;
				}
				CartasMidlet.getDisplay().setCurrent(telaVolta);
			}
		}else if(c == cmdEntrarSala){
			if(txtApelido.getString().trim().length() == 0){
				if(alertaApelido == null)
					alertaApelido = new Alert("", "Digite o apelido!", null,
AlertType.ERROR);

				CartasMidlet.getDisplay().setCurrent(alertaApelido);
			}else{

	CartasMidlet.getDisplay().setCurrent(ControleJogo.frmespera);
				//Thread thread = new Thread(){
				//	public void run(){
						String sala =
(String)vetorSalas.elementAt(choice.getSelectedIndex());
						int indice = sala.indexOf("[");
						sala = sala.substring(0, indice).trim();
						try{
							String strResultado =
ControleJogo.retornarMsgServidor(ControleJogo.getHostServidor(),
"acao=ENTRAR_SALA&jogador=" +
txtApelido.getString().replace('[','a').replace(']','b').replace('!','c').replace('@','d').replace('#
','e').replace('$','f').replace('%','g').replace('¨','h').replace('&','i').replace('*','j').trim() +
"&sala=" + sala, true, frmListaSalas.this.controleJogo);

							ControleJogo.nomeJogadores[0] =
	txtApelido.getString().replace('[','a').replace(']','b').replace('!','c').replace('@','d').rep
lace('#','e').replace('$','f').replace('%','g').replace('¨','h').replace('&','i').replace('*','j').trim();

							ControleJogo.nomeJogador =
txtApelido.getString().replace('[','a').replace(']','b').replace('!','c').replace('@','d').replace('#
','e').replace('$','f').replace('%','g').replace('¨','h').replace('&','i').replace('*','j').trim();

							ControleJogo.idSala = sala;

	ControleJogo.exibirDebugMsg("ControleJogo>>commandAction>>" +
strResultado);
```

```
                                        }catch(Exception e){

                                             if(alertaErro == null)
                                                  alertaErro = new Alert("", "Erro ao
estabelecer conexão (SK) " + e, null, AlertType.INFO);
                                             else
                                                  alertaErro.setString("Erro ao
estabelecer conexão (SK) " + e);


     CartasMidlet.getDisplay().setCurrent(alertaErro, frmListaSalas.this);


                                        }
                              //        }
                              /*};
                              try{
                                   thread.start();
                                   thread.join();
                              }catch(Exception e){}*/

                         }

                    }
          }

}
```

<u>frmOpcoes.java</u>

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.frmOpcoes
Responsabilidades: Exibir opções para o usuário

**************************************Alterações**************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Data: 05/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementação do formulário de opções
Status: ok
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 10/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando opção para exibir informações sobre as cartas
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 05/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando opção para inserir ip e porta do servidor
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/
package jogo;

import javax.microedition.lcdui.*;
import jogo.apibasica.*;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.frmOpcoes
<p>Responsabilidades: Exibir opções para o usuário, entre elas opção de
habilitar/desabilitar o modo open mau mau,<br>
                        opção de habilitar/desabilitar a visualização de informações sobre a
carta selecionada.<br>
                        Também exibe campos onde o jogador deve digitar o endereço e a
porta do servidor.
*/
public class frmOpcoes extends Form implements CommandListener{
```

```
	/**
		Tela de volta
	*/
	private Displayable telaVolta;

	/**
		Comando utilizado para retornar para a tela anterior
	*/
	private Command cmdVoltar = new Command("Voltar", Command.SCREEN, 0);

	/**
		Exibe as opções de habilitar/desabilitar modo open mau mau e <br>
		opção de habilitar/desabilitar a visualização de informações sobre a carta
selecionada.<br>
	*/
	private ChoiceGroup grupo;

	/**
		Campo onde o jogador deve especificar o endereço IP do servidor
	*/
	private TextField txtIP = new TextField("Endereço do servidor: ", "", 32,
TextField.URL);

	/**
		Campo onde o jogador deve especificar a porta do servidor
	*/
	private TextField txtPorta = new TextField("Porta do servidor: ", "", 8,
TextField.NUMERIC);


	/**
		Construtor
	*/
	public frmOpcoes(String titulo, Displayable d){
		super(titulo);
		//setTicker(new Ticker("Ajuda e Regras do Jogo"));
		if(d == null)
			throw new IllegalArgumentException("Displayable deve ser diferente
de null!");
		telaVolta = d;

		grupo = new ChoiceGroup("Opções", Choice.MULTIPLE , new
String[]{"Open Mau Mau", "Exibir informações sobre a carta selecionada"} , null);
		append(grupo);
		append(txtIP);
		append(txtPorta);
```

```
        addCommand(cmdVoltar);
        setCommandListener(this);
}


/**
        Seta a opção open Mau Mau
*/
public void setOpenMauMau(boolean b){
        grupo.setSelectedIndex(0,  b);
}

/**
        Seta a opção Exibir informações
*/
public void setExibirInformacoesCarta(boolean b){
        grupo.setSelectedIndex(1, b);
}

/**
        Modifica o ip do servidor
*/
public void setIPServidor(String ip){
        txtIP.setString(ip);
}

/**
        Modifica a porta do servidor
*/
public void setPortaServidor(String porta){
        txtPorta.setString(porta);
}


/**
        Método que é invocado quando o jogador seleciona algum comando
*/
public void commandAction(Command c, Displayable d){
        if(c == cmdVoltar)
                if(CartasMidlet.getDisplay() != null){
                  if(telaVolta instanceof ControleJogo){
                        ((ControleJogo)telaVolta).setApresentacao(true);
                        ((ControleJogo)telaVolta).setTelaSelecao(true);
                  }
                  if(ControleJogo.DEBUG_MODE){
                        if(grupo.isSelected(0)){
```

```
        System.out.println("frmOpcoes>>commandAction>>Modo open mau mau
ativado");
                                }else{

        System.out.println("frmOpcoes>>commandAction>>Modo open mau mau
desativado");
                                }
                        }

                        ControleJogo.setOpenMauMau(grupo.isSelected(0));
                        ControleJogo.EXIBIR_INFORMACAO_CARTA =
grupo.isSelected(1);
                        String ip = txtIP.getString().trim();
                        String porta = txtPorta.getString().trim();
                        Alert alertaErro = null;
                        if(ip.length() == 0){
                                alertaErro = new Alert("", "Digite o ip do servidor!", null,
AlertType.INFO);
                                CartasMidlet.getDisplay().setCurrent(alertaErro);
                        }else if(porta.length() == 0){
                                alertaErro = new Alert("", "Digite a porta do servidor!", null,
AlertType.INFO);
                                CartasMidlet.getDisplay().setCurrent(alertaErro);
                        }else if(Integer.parseInt(porta) < 0){
                                alertaErro = new Alert("", "Porta do servidor inválida!", null,
AlertType.INFO);
                                CartasMidlet.getDisplay().setCurrent(alertaErro);
                        }else{
                                ControleJogo.setIPServidor(ip);
                                ControleJogo.setPortaServidor(porta);

                                CartasMidlet.getDisplay().setCurrent(telaVolta);
                        }
                }
        }

}
```

frmVisualizadorCartas.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para dispositivos
móveis
Classe: jogo.frmAjuda
```

```
Responsabilidades: Exibir as cartas do baralho

************************************Alterações************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 01/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criação da classe
Status: OK
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/
package jogo;

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

import jogo.*;
import jogo.apibasica.*;


/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.frmAjuda
<p>Responsabilidades: Exibir as cartas do baralho

@author Rogério de Paula Aguilar
@version 1.0
*/

public class frmVisualizadorCartas extends Canvas implements CommandListener{

	/**
		Tela de volta
	*/
	private Displayable telaVolta;

	/**
		Comando utilizado para retornar à tela anterior
	*/
```

```
	private Command cmdVoltar = new Command("Voltar", Command.SCREEN, 0);

	/**
		Tela do jogo
	*/
	private Display display;

	/**
		Informação sobre a carta que está sendo exibida
	*/
	private byte naipe, valor;

	/**
		Comando utilizado para mover para a próxima carta
	*/
	private Command cmdProximaCarta = new Command("Próxima Carta",
Command.SCREEN, 0);

	/**
		Comando utilizado para mover para a carta anterior
	*/
	private Command cmdCartaAnterior = new Command("Carta anterior",
Command.SCREEN, 0);

	/**
		Imagem da carta atual
	*/
	private ImageItem imagemAtual;

	/**
		Baralho
	*/
	private Baralho baralho;

	/**
		Objeto utilizado para obter informações sobre o dispositivo
	*/
	private Dispositivo dispositivo;

	/**
		Construtor
	*/
	public frmVisualizadorCartas(String titulo, Displayable telaVolta, Display display,
Baralho baralho, Dispositivo d){
		//super(false);

		this.display = display;
```

```
            this.telaVolta = telaVolta;
            this.baralho = baralho;
            dispositivo = d;
            //imagemAtual = new ImageItem(baralho.cartasBaralho[naipe][valor],
ImageItem.LAYOUT_CENTER, "");

            addCommand(cmdProximaCarta);
            addCommand(cmdCartaAnterior);
            addCommand(cmdVoltar);
            setCommandListener(this);

        }

        /**
            Método que é invocado quando o usuário seleciona algum comando
        */
        public void commandAction(Command c, Displayable d){
            if(c == cmdVoltar){
                if(display != null){
                   ((ControleJogo)telaVolta).setApresentacao(true);
                   ((ControleJogo)telaVolta).setTelaSelecao(true);
                   display.setCurrent(telaVolta);
                }
            }else if(c == cmdProximaCarta){
                valor +=1;
                if(valor == 14){
                   naipe += 1;
                   if(naipe == 4) naipe = 0;
                   valor = 0;

                }
                repaint();

            }else if(c == cmdCartaAnterior){
                valor -=1;
                if(valor == -1){
                   naipe -= 1;
                   if(naipe == -1) naipe = 3;
                   valor = 13;

                }
                repaint();

            }


        }
```

```
	/**
		Desenha a carta atual e a s informações sobre a mesma
	*/
	public void paint(Graphics g){
		//super.paint(g);
		g.setColor(0, 190, 0);
		g.fillRect(0, 0, dispositivo.getLarguraTela(), dispositivo.getAlturaTela());

		g.setColor(255, 255, 255);

		g.setFont(Font.getFont(Font.STYLE_BOLD ));

		g.drawString(baralho.cartasBaralho[naipe][valor].getNaipeStr() , 0, 0, 0);
		g.drawString(baralho.cartasBaralho[naipe][valor].getValorStr() , 0,
Font.getFont(Font.STYLE_BOLD ).getHeight() + 2, 0);


		baralho.cartasBaralho[naipe][valor].setPosition(
(dispositivo.getLarguraTela() - baralho.cartasBaralho[naipe][valor].getWidth()) / 2,
									(dispositivo.getAlturaTela() -
baralho.cartasBaralho[naipe][valor].getHeight()) / 2 );
		baralho.cartasBaralho[naipe][valor].paint(g);
	}

}
```

Jogador.java

```
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 01/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criação da classe
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 05/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Movendo rotinas de teclado que são comuns a todos os tipos de jogadores
```

(visualizar o baralo e exibir próximo jogador)
Status: ok
%%%
%%%
%%

%%%
%%%
%%
Data: 11/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando novas rotinas de exclusão de cartas

Status: ok
%%%
%%%
%%

%%%
%%%
%%
Data: 14/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina que retorna o número de cartas do jogador
Adicionando método equals
Adicionando variável que verifica se o jogador disse mau mau
Adicionando método dizer Mau Mau
Status: ok
%%%
%%%
%%

%%%
%%%
%%
Data: 18/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina procurar carta
Status: ok
%%%
%%%
%%

%%%
%%%
%%

```
Data: 19/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: movendo rotina de movimentação de carta de jogadorhumano para jogador
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 09/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: movendo rotina de movimentação de carta de jogadorhumano para jogador
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 20/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: adicionando rotina que retorna uma carta pelo índice
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 22/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: adicionando rotina que zera o baralho do jogador
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%



*/
package jogo;

import javax.microedition.lcdui.*;
```

```
import javax.microedition.lcdui.game.*;
import java.util.*;
import jogo.*;
import jogo.apibasica.*;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.Jogador
<p>Responsabilidades: Guarda as informações e ações que são comuns aos
jogadores, independente se estes <br>
                        são jogadores humanos ou o computador

@author Rogério de Paula Aguilar
@version  1.0

*/


public abstract class Jogador extends GameCanvas implements CommandListener{

        /**
                Nome do jogador
        */
        private String nome;

        /**
                Cartas do jogador
        */
        protected Stack cartasAtuais;

        /**
                Baralho
        */
        protected Baralho baralho;

        /**
                Indica que tela está sendo exibida atualmente
        */
        private byte telaAtual;

        /**
                Tempo de espera
        */
        public final static int ESPERA = 10000; //tempo de espera
```

```
protected ControleJogo controleJogo;

/**
	Índice da carta selecionada
*/
protected int cartaSelecionada = 0;

/**
	Posição inicial para o desenho das cartas
*/
protected int posicao = 12;//posição inicial de desenho das cartas

/**
	Indica se o jogador disse mau-mau ou não
*/
protected boolean DISSE_MAU_MAU = false;

/**
	Comando utilizado para sair
*/
protected Command cmdSair = new Command("Sair", Command.SCREEN, 0);

/**
	Imagem que aparece na tela do jogador que está jogando
*/
public static Sprite imgPensar;

/**
	Imagem do jogador
*/
public static Sprite imgJogador;

/**
	Transformação atual (utilizada para o desenho das setas de navegação)
*/
protected int transformacaoAtual = Sprite.TRANS_NONE;

/**
	Imagem da seta de navegação
*/
public static Sprite imgSeta;

protected byte deslocamentoSetaEsquerda = 0;
protected byte deslocamentoSetaDireita = 0;
protected byte deslocamentoSetaCima = 0;
protected byte deslocamentoSetaCimaBaixo = 0;
```

```
	/**
		Construtor
	*/
	public Jogador(String nome, Baralho baralho, ControleJogo controleJogo){
		super(false);
		this.nome = nome;
		this.baralho = baralho;
		cartasAtuais = new Stack();
		this.controleJogo = controleJogo;
		//setTicker(new Ticker("Mesa: " + baralho.cartaAtual().toString()) );
		addCommand(cmdSair);
		setCommandListener(this);
		try{
			imgPensar = new Sprite(Image.createImage("/pensar.png"));
			imgSeta = new Sprite(Image.createImage("/seta.png"));
			imgJogador = new Sprite(Image.createImage("/jogador.png"));

		}catch(Exception e){
			ControleJogo.exibirDebugMsg("Jogador>>Jogador(...)>>Erro ao
carregar .png>>" + e);
		}

	}

	/**
		Método invocado quando o jogador seleciona algum comando do jogo
	*/
	public void commandAction(Command c, Displayable d){
	    synchronized(controleJogo){
		if(c == cmdSair){
			controleJogo.setApresentacao(true);
			controleJogo.incrementaJogoAtual();
			controleJogo.setTelaSelecao(true);
			ControleJogo.JOGO_EM_ANDAMENTO = false;

			ControleJogo.exibirDebugMsg("JOGO_EM_ANDAMENTO: " +
ControleJogo.JOGO_EM_ANDAMENTO);
			CartasMidlet.getDisplay().setCurrent(controleJogo);
			if(controleJogo.ONLINE){
				controleJogo.enviarMsgServidor("SAIR_CLIENTE");

				controleJogo.fecharConexao();
				ControleJogo.ONLINE = false;
```

```
                }
                System.gc();
        }
        }
}


/**
        Adiciona uma carta para este jogador
*/
public void adicionarCarta(Carta carta){
        cartasAtuais.push(carta);
}

/**
        Remove uma carta do jogador
*/
public Carta removerCarta(){
        return (Carta)(cartasAtuais.pop());
}

/**
        Remove uma carta do jogador pelo índice da carta
*/
public Carta removerCarta(int i){
        Carta carta = (Carta)cartasAtuais.elementAt(i);
        cartasAtuais.removeElementAt(i);
        return carta;
}



/**
        Método de jogo
*/
public abstract void jogar(Object args[]);

/**
        Retorna o nome do jogador
*/
public String getNome(){
        return nome;
}

/**
        Retorna uma representação do jogador no formato de um objeto String
*/
```

```
	public String toString(){
		StringBuffer str = new StringBuffer("*****Jogador>>Nome: " + nome +
"*****");
		for(int i = 0; i < cartasAtuais.size(); i++){
			str.append("\n" + (Carta)cartasAtuais.elementAt(i));
		}
		return str.toString().trim();

	}

	/**
		Modifica o nome do jogador
	*/
	public void setNome(String nome){
		this.nome = nome;
	}


	/**
		Método invocado quando o ujogador pressiona alguma tecla
	*/
	public void keyPressed(int keyCode){
		synchronized(controleJogo){

		if(getKeyCode(UP) == keyCode){
			deslocamentoSetaCima = -2;
		}else if(getKeyCode(DOWN) == keyCode){
			//Exibindo cartas do próximo jogador
			deslocamentoSetaCimaBaixo = 2;
		}else if(getKeyCode(LEFT) == keyCode){
			//Move a carta para a esquerda
			cartaSelecionada -= 1;
			if(cartaSelecionada < 0)
				cartaSelecionada = cartasAtuais.size() - 1;
			deslocamentoSetaEsquerda = -2;

		}else if(getKeyCode(RIGHT) == keyCode){
			//Move a carta para a direita
			cartaSelecionada += 1;
			if(cartaSelecionada > (cartasAtuais.size() - 1))
				cartaSelecionada = 0;
			deslocamentoSetaDireita = 2;

		}
		repaint();
		}
	}
```

```
	/**
		Método invocado quando o usuário solta uma tecla que estava
pressionada
	*/
	public void keyReleased(int keyCode){
	    synchronized(controleJogo){
	      if(getKeyCode(LEFT) == keyCode)
			deslocamentoSetaEsquerda = 0;
	    else if(getKeyCode(RIGHT) == keyCode)
			deslocamentoSetaDireita = 0;
	    else if(getKeyCode(UP) == keyCode){
			//Exibindo Baralho
			deslocamentoSetaCima = 0;
			repaint();

	ControleJogo.exibirDebugMsg("JogadorHumano>>keyPressed>>Exibindo
Baralho");
			baralho.setTelaVolta(this);
			CartasMidlet.getDisplay().setCurrent(baralho);

	     }else if(getKeyCode(DOWN) == keyCode){

	ControleJogo.exibirDebugMsg("JogadorHumano>>keyPressed>>Exibindo cartas
do próximo jogador");
			deslocamentoSetaCimaBaixo = 0;
			repaint();
			controleJogo.exibirProximoJogador();


		}
		repaint();
		}
	}

	/**
		Retorna o número de cartas do jogador
	*/
	public int numeroCartas(){
		return cartasAtuais.size();
	}

	/**
		Indica se duas instâncias de um jogador são iguais
	*/
	public boolean equals(Object obj){
```

```
		boolean b = false;
		if(obj instanceof Jogador){
			Jogador jogador = (Jogador)obj;
			b = nome.equals(jogador.nome);
			if(b){
				if(cartasAtuais.size() == jogador.cartasAtuais.size()){
					for(int i = 0; i < cartasAtuais.size(); i++){
						Carta carta1 = (Carta)cartasAtuais.elementAt(i);
						Carta carta2 =
(Carta)jogador.cartasAtuais.elementAt(i);
						b = b && carta1.equals(carta2);
					}
				}else{
					return false;
				}
			}else{
				return false;
			}
		}
		return b;
	}

	/**
		Seta a opção disse mau mau
	*/
	public void setDisseMauMau(boolean d){
		DISSE_MAU_MAU = d;
	}

	/**
		Retorna o valor de DISSE_MAU_MAU
	*/
	public boolean getDisseMauMau(){
		return DISSE_MAU_MAU;
	}

	/**
		Retorna o índice de determinada carta
	*/

	public int getIndiceCarta(Carta carta){
		return cartasAtuais.indexOf(carta);
	}


	/**
```

```
		Desenha os elementos da interface gráfica que são comuns a todos os
tipos de
		jogadores
	*/
	public void paint(Graphics g){



		g.setColor(0, 190, 0);
		g.fillRect(0, 0, Dispositivo.getLarguraTela(), Dispositivo.getAlturaTela());


		if(controleJogo.getJogadorAtual() == controleJogo.getJogadorExibido()){
			imgPensar.setPosition(Dispositivo.getLarguraTela() - 70, 25);
			imgPensar.paint(g);
		}


		//Seta da esquerda
		imgSeta.setTransform(Sprite.TRANS_NONE);
		imgSeta.setPosition( 5 + deslocamentoSetaEsquerda,
Dispositivo.getCentroY() + (Carta.ALTURA_CARTA / 2) + 10);
		imgSeta.paint(g);
		//Seta da direita
		imgSeta.setTransform(Sprite.TRANS_MIRROR);
		imgSeta.setPosition( Dispositivo.getLarguraTela() - 5 - imgSeta.getWidth()
+ deslocamentoSetaDireita, Dispositivo.getCentroY() + (Carta.ALTURA_CARTA / 2) +
10);
		imgSeta.paint(g);
		//Seta de cima (baralho)
		imgSeta.setTransform(Sprite.TRANS_ROT90);
		imgSeta.setPosition( 2, 5 + deslocamentoSetaCima);
		imgSeta.paint(g);
		//Seta de cima apontando para baixo (próximo jogador)
		imgSeta.setTransform(Sprite.TRANS_ROT270);
		imgSeta.setPosition( Dispositivo.getLarguraTela() - imgSeta.getWidth() - 2
, 5 + deslocamentoSetaCimaBaixo);
		imgSeta.paint(g);
		//Boneco
		imgJogador.setPosition( Dispositivo.getLarguraTela() - imgSeta.getWidth()
- 12, 5);
		imgJogador.paint(g);
		//Baralho em miniatura
		for(byte n = 0; n < 3; n++){
			Baralho.cartaVersoMiniatura.setPosition(8 + imgSeta.getWidth() +
n, 4 + n);
			Baralho.cartaVersoMiniatura.paint(g);
```

```
                }

                //Informações sobre a carta
                if(ControleJogo.EXIBIR_INFORMACAO_CARTA && (this instanceof
JogadorHumano || (this instanceof JogadorComputador &&
controleJogo.getOpenMauMau()))){

                        if(cartaSelecionada >= 0 && cartaSelecionada <
cartasAtuais.size()){
                                Ticker tk = getTicker();
                                if(tk != null){

        tk.setString(((Carta)cartasAtuais.elementAt(cartaSelecionada)).toString());
                                }else{
                                        tk = new
Ticker(((Carta)cartasAtuais.elementAt(cartaSelecionada)).toString());
                                        setTicker(tk);
                                }

                                /*g.setColor(0, 0, 255);
                                g.drawString(
((Carta)cartasAtuais.elementAt(cartaSelecionada)).toString(), 0, 0, 0);
                                */

                        }
                }


        }

        /**
                Retorna uma carta do baralho
        */

        public Carta carta(int indice){
                if(indice < 0 || indice > cartasAtuais.size())
                        throw new IllegalArgumentException("Baralho>>carta>>Índice
inválido!");
                return (Carta)cartasAtuais.elementAt(indice);
        }

        /**
                Deixa o jogador sem nenhuma carta
        */
        public void zerar(){
                cartasAtuais = new Stack();
```

```
                System.gc();
        }

}
```

<u>JogadorComputador.java</u>

```
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 05/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criação da classe
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 08/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando rotina de exibição
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 14/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Começando rotina jogar()
            Criando thread que fará a jogada pelo computador
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 18/09/2003
```

Responsável: Rogério de Paula Aguilar
Descrição: Implementando rotina de jogo
Status: ok
%%
%%
%%

%%
%%
%%
Data: 19/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Implementando rotina de jogo (jogar)
Status: ok
%%
%%
%%

%%
%%
%%
Data: 09/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Testando rotina de jogo
Alterando mensagens para se tornarem mais claras para o jogador
Inserindo jogada 8* que estava faltando
Adicionando rotina para verificar se o jogador disse mau-mau ou não
Status: ok
%%
%%
%%

%%
%%
%%
Data: 10/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando interface gráfica
Resolvendo problemas de sincronismo da thread que joga pelo computador

Status: ok
%%
%%
%%

%%
%%
%%
Data: 12/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Corrigindo problema com a thread de jogada

Status: ok
%%
%%
%%

%%
%%
%%
Data: 03/12/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando mensagens e adicionando jogada Valete

Status: ok
%%
%%
%%

*/

```
package jogo;
import jogo.apibasica.*;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.util.*;



/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.JogadorComputador
<p>Responsabilidades: Guarda as informações sobre um jogador controlado pelo
computador e realiza a
                         a jogada do computador
@author Rogério de Paula Aguilar
@version  1.0
```

```
*/

public class JogadorComputador extends Jogador{


	/**
		Classe que faz a jogada pelo computador
	*/
	class threadJogada implements Runnable{

			/**
				Índice do jogo atual
			*/

			private int indiceJogoAtual;

			/**
				Modifica o índice do jogo atual
			*/
			public void setJogoAtual(int jogoAtual){
				indiceJogoAtual = jogoAtual;
			}

			/**
				Método que faz a jogada pelo computador
			*/
			public void run(){
					try{


	Thread.sleep(JogadorComputador.this.ESPERA);

	ControleJogo.exibirDebugMsg("**********JogadorCOmputador>>jogar()>>threadJ
ogada>>Continuando jogada**********");

							ControleJogo.Mensagem mensagem =
null;

							int indiceCarta = -1;
							try{
								/*Verificando jogada: 7* */

	ControleJogo.exibirDebugMsg("JogadorComputador>>Testando se carta do
baralho é um sete...");
```

```
                                                    if(baralho.cartaAtual().getValor()
== Carta.SETE && !controleJogo.getPenalidadeAplicada()){

ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Procurando sete");
                                                    indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.SETE]);
                                                    if(indiceCarta == -1) indiceCarta
= getIndiceCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.SETE]);
                                                    if(indiceCarta == -1) indiceCarta
= getIndiceCarta(Baralho.cartasBaralho[Carta.COPAS][Carta.SETE]);
                                                    if(indiceCarta == -1) indiceCarta
= getIndiceCarta(Baralho.cartasBaralho[Carta.ESPADAS][Carta.SETE]);
                                                    if(indiceCarta == -1){

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Procurando sete>>sete não encontrado>>procurando 2 do mesmo naipe");
                                                        indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][Carta.DOIS]);
                                                        if(indiceCarta == -1){

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Procurando sete>>sete não encontrado>>2 do mesmo naipe não encontrado");

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Procurando sete>>sete não encontrado>>2 do mesmo naipe não
encontrado>>Procurando um coringa");
                                                            indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.CORINGA]);
                                                            if(indiceCarta == -1)
        indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.CORINGA]);
                                                            if(indiceCarta == -1)
        indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.COPAS][Carta.CORINGA]);
                                                            if(indiceCarta == -1)
        indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.ESPADAS][Carta.CORINGA]);
                                                            if(indiceCarta == -1){

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Procurando sete>>sete não encontrado>>Coringa não encontrado>>Comprando
cartas");
                                                            for(int i = 0; i <
controleJogo.getQuantidadeCompra(); i++){

        adicionarCarta(baralho.comprarCarta());
```

```
                                                                    }
                                                                    mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " comprou " +
controleJogo.getQuantidadeCompra() + " cartas. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));


        controleJogo.setPenalidadeAplicada(true);

        controleJogo.setQuantidadeCompra((byte)0);
                                                                }else{

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Procurando sete>>sete não encontrado>>Coringa encontrado");
                                                                mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " jogou um coringa, que anula
o efeito de qualquer carta especial e pode ser jogado sobre qualquer carta. O coringa
anulou o efeito do sete anterior. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);

        controleJogo.setQuantidadeCompra((byte)0);


                                                                }
                                                        }else{

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Procurando dois do mesmo naipe>>dois encontrado");

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);

        controleJogo.setQuantidadeCompra((byte)0);
                                                                mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " jogou um dois que anulou o
efeito do sete anterior. O próximo jogador será " +
```

```
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));

                                                                }
                                                        }else{

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
sete>>Sete encontrado");

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);

        controleJogo.setPenalidadeAplicada(false);

        controleJogo.setQuantidadeCompra((byte)(controleJogo.getQuantidadeCompra()
+ 3));
                                                                mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " jogou um sete. O próximo
jogador deve comprar" + controleJogo.getQuantidadeCompra() + " cartas ou jogar outro
7, um valete, um dois do mesmo naipe ou um coringa. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",
                                                                        AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));
                                                                }


                                                        }else{
                                                          /*Verificando jogada: 9* */

ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando se a carta atual é um
nove");
                                                                if(baralho.cartaAtual().getValor()
== Carta.NOVE && !controleJogo.getPenalidadeAplicada()){

        ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando se a carta
atual é um nove>>Carta atual é um nove>>Procurando por um dois do naipe " +
baralho.cartaAtual().toString());
                                                                        indiceCarta =
getIndiceCarta(baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][Carta.DOIS]);
                                                                        if(indiceCarta == -1){

ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando se a carta atual é um
```

```
nove>>Carta atual é um nove>>Procurando por um dois do naipe>>DOIS NÃO
ENCONTRADO>>Procurando um coringa");
                                                                    indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.CORINGA]);
                                                                    if(indiceCarta == -1)
indiceCarta = getIndiceCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.CORINGA]);
                                                                    if(indiceCarta == -1)
indiceCarta = getIndiceCarta(Baralho.cartasBaralho[Carta.COPAS][Carta.CORINGA]);
                                                                    if(indiceCarta == -1)
indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.ESPADAS][Carta.CORINGA]);
                                                                    if(indiceCarta == -1){

        ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando se a carta
atual é um nove>>Carta atual é um nove>>Procurando por um dois do naipe>>DOIS
NÃO ENCONTRADO>>Procurando um coringa>>Coringa não encontrado");

        adicionarCarta(baralho.comprarCarta());

        controleJogo.setQuantidadeCompra((byte)0);

        controleJogo.setPenalidadeAplicada(true);
                                                                    mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " comprou uma carta, pois
não tinha um 2 do mesmo do 9 que estava no baralho para se defender e não possuia
um coringa. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));
                                                                    }else{

        ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando se a carta
atual é um nove>>Carta atual é um nove>>Procurando por um dois do naipe>>DOIS
NÃO ENCONTRADO>>Procurando um coringa>>CORINGA ENCONTRADO");
                                                                    mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " jogou um coringa, que anula
o efeito de qualquer carta especial e pode ser jogado sobre qualquer carta. O coringa
anulou o efeito do 9 anterior. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);
```

```
        controleJogo.setQuantidadeCompra((byte)0);


                                                                }

                                                              }else{

ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando se a carta atual é um
nove>>Carta atual é um nove>>Procurando por um dois do naipe>>DOIS
ENCONTRADO");

baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

cartasAtuais.removeElementAt(indiceCarta);

controleJogo.setQuantidadeCompra((byte)0);
                                                                    mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " jogou um 2 do mesmo naipe
do nove anterior, anulando o efeito deste. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",
                                                                        AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));


                                                                }
                                                            }else{

                                                                /*Verificando jogada: 10 de
PAUS*/

        if(baralho.cartaAtual().getNaipe() == Carta.PAUS &&
baralho.cartaAtual().getValor() == Carta.DEZ &&
!controleJogo.getPenalidadeAplicada()){

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é 10p
>>Procurando um 2 de Paus");
                                                                        indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.DOIS]);
                                                                        if(indiceCarta == -1){
                                                                        indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.CORINGA]);
                                                                            if(indiceCarta
== -1) indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.CORINGA]);
```

```
                                                                    if(indiceCarta
== -1) indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.COPAS][Carta.CORINGA]);
                                                                    if(indiceCarta
== -1) indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.ESPADAS][Carta.CORINGA]);
                                                                    if(indiceCarta
== -1){

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é
10p>>Procurando um 2 de paus>>2 de paus não encontrado>>Comprando 5 cartas");
                                                                        for(int i
= 0; i < controleJogo.getQuantidadeCompra(); i++){

        adicionarCarta(baralho.comprarCarta());

                                                                        }

        mensagem = controleJogo.new Mensagem("O jogador " + getNome() + "
comprou " + controleJogo.getQuantidadeCompra() + " cartas. O próximo jogador será "
+ controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));

        controleJogo.setPenalidadeAplicada(true);

        //controleJogo.setQuantidadeCompra((byte)0);


                                                                    }else{

        ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando se a carta
atual é um 10p>>Carta atual é um 10p>>Procurando por um dois do naipe>>DOIS NÃO
ENCONTRADO>>Procurando um coringa>>CORINGA ENCONTRADO");

        mensagem = controleJogo.new Mensagem("O jogador " + getNome() + " jogou
um coringa, que anula o efeito de qualquer carta especial e pode ser jogado sobre
qualquer carta. O coringa anulou o efeito do 10 de paus anterior. O próximo jogador
será " + controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome()
+ ". Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);
```

```
        controleJogo.setQuantidadeCompra((byte)0);

                                                                    }

                                                                }else{

        ControleJogo.exibirDebugMsg("JogadorComputador>>A carta atual  é um
10p>>Dois de paus encontrado>>Efeito do 10 anulado");

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);

        controleJogo.setQuantidadeCompra((byte)0);
                                                                mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " jogou um 2 de paus, o que
anulou o efeito do 10 de paus. A quatidade de cartas para compra agora é: " +
controleJogo.getQuantidadeCompra() + " cartas. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));
                                                                }

                                                            }else{

        ControleJogo.exibirDebugMsg("JogadorComputador>>Carta do baralho não é
especial");

        ControleJogo.exibirDebugMsg("JogadorComputador>>ProcurandoCarta>>Carta
atual do baralho: " + baralho.cartaAtual().toString());
                                                                boolean encontrou =
false;
                                                                indiceCarta = -1;


        ControleJogo.exibirDebugMsg("JogadorComputador>>Procurando
carta>>Procurando um AS");
                                                                //Procurando um AS

                                                                if(indiceCarta == -1)

        if(baralho.cartaAtual().getNaipe() == Carta.OUROS)
```

```
        indiceCarta = getIndiceCarta(baralho.cartasBaralho[Carta.OUROS][Carta.AS]);

                                                                    if(indiceCarta == -1)

        if(baralho.cartaAtual().getNaipe() == Carta.PAUS)

        indiceCarta = getIndiceCarta(baralho.cartasBaralho[Carta.PAUS][Carta.AS]);
                                                                    if(indiceCarta == -1)

        if(baralho.cartaAtual().getNaipe() == Carta.COPAS)

        indiceCarta = getIndiceCarta(baralho.cartasBaralho[Carta.COPAS][Carta.AS]);
                                                                    if(indiceCarta == -1)

        if(baralho.cartaAtual().getNaipe() == Carta.ESPADAS)

        indiceCarta = getIndiceCarta(baralho.cartasBaralho[Carta.ESPADAS][Carta.AS]);

                                                                    if(indiceCarta != -1){

        ControleJogo.exibirDebugMsg("JogadorComputador>>Procurando
carta>>Procurando um AS>>As encontrado");
                                                                        mensagem =
controleJogo.new Mensagem("O jogador " + getNome() + " jogou um AS e poderá jogar
novamente. Realizando nova jogada.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.getJogadorAtual()));


baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);

                                                                    }else{
                                                                        indiceCarta = -
1;

        if(baralho.cartaAtual().getNaipe() == Carta.PAUS){

        ControleJogo.exibirDebugMsg("JogadorComputador>>Naipe atual é
PAUS>>Procurando um 10 de paus");

        indiceCarta = getIndiceCarta(baralho.cartasBaralho[Carta.PAUS][Carta.DEZ]);

        if(indiceCarta != -1){
```

```
        ControleJogo.exibirDebugMsg("JogadorComputador>>Naipe atual é
PAUS>>Procurando um 10 de paus>>Dez de paus encontrado");

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);

        controleJogo.setPenalidadeAplicada(false);

        controleJogo.setQuantidadeCompra((byte)5);

        mensagem = controleJogo.new Mensagem("O jogador " + getNome() + " jogou
um 10 de PAUS. O próximo jogador deverá comprar " +
controleJogo.getQuantidadeCompra() + " cartas, jogar um 2 de PAUS, um coringa ou
um valete. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()));



                                                                    }
                                                                }
                                                                if(indiceCarta
== -1){

        //Procurando um sete do naipe atual

        indiceCarta = -1;

        ControleJogo.exibirDebugMsg("JogadorComputador>>Procurando um 7 do naipe
atual");

        indiceCarta =
getIndiceCarta(baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][Carta.SETE]);

        if(indiceCarta != -1){

        ControleJogo.exibirDebugMsg("JogadorComputador>>Procurando um 7 do naipe
atual>>7 encontrado");

        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

        cartasAtuais.removeElementAt(indiceCarta);
```

```
        controleJogo.setPenalidadeAplicada(false);

        controleJogo.setQuantidadeCompra((byte)(controleJogo.getQuantidadeCompra()
+ 3));

        mensagem = controleJogo.new Mensagem("O jogador " + getNome() + " jogou
um 7. O próximo jogador deverá comprar " + controleJogo.getQuantidadeCompra() + "
cartas, jogar um outro 7, um dois do mesmo naipe do sete, um coringa ou um valete. O
próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

        AlertType.INFO, controleJogo.getJogador(controleJogo.proximoJogador()) );

                                                                        }else{

        //Procurando um quatro de espadas, caso o naipe atual seja espadas

        indiceCarta = -1;

        if(baralho.cartaAtual().getNaipe() == Carta.ESPADAS){

        indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.ESPADAS][Carta.QUATRO]);

        if(indiceCarta != -1){

                //4 de espadas encontrado

                baralho.adicionarCarta( (Carta)cartasAtuais.elementAt(indiceCarta) );

                cartasAtuais.removeElementAt(indiceCarta);


                for(int i = 0 ; i < ControleJogo.NUMERO_JOGADORES; i++){

                        if( ! ( controleJogo.getJogador(i).equals(this)) ){


        controleJogo.getJogador(i).adicionarCarta(baralho.comprarCarta());

                        }

                }
```

```
                mensagem = controleJogo.new Mensagem("O jogador " + getNome() + "
jogou um 4 de espadas e, como consequência, todos ou outros jogadores compraram
uma carta. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                        AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));


        }
                                                                                }


        if(indiceCarta == -1){



        //Se o próximo jogador tem um número menor de cartas que o jogador anterior,
tentar inverter o sentido do jogo

        int proximoJogador = controleJogo.getIndiceProximoJogador();

        int jogadorAnterior = controleJogo.getIndiceJogadorAnterior();

        if(controleJogo.getJogador(proximoJogador).numeroCartas() <
controleJogo.getJogador(jogadorAnterior).numeroCartas()){

                indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][Carta.CINCO]);

                if(indiceCarta != -1){

                        //Cinco encontrado

                        baralho.adicionarCarta( (Carta)cartasAtuais.elementAt(indiceCarta)
);

                        cartasAtuais.removeElementAt(indiceCarta);

                        controleJogo.inverterSentidoJogo();

                        mensagem = controleJogo.new Mensagem("O jogador " +
getNome() + " jogou um 5 e inverteu o sentido do jogo. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",
```

```
                    AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));

                }

        }

        if(indiceCarta == -1){



                //Procurando um  9, que faz o próximo jogador comprar uma carta



                indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][Carta.NOVE]);

                if(indiceCarta != -1){

                        //9 encontrado

                        baralho.adicionarCarta( (Carta)cartasAtuais.elementAt(indiceCarta)
);

                        cartasAtuais.removeElementAt(indiceCarta);

                        controleJogo.setPenalidadeAplicada(false);

                        mensagem = controleJogo.new Mensagem("O jogador " +
getNome() + " jogou um 9. O próximo jogador deve comprar uma carta, jogar um 2 do
mesmo naipe do nove, um coringa ou um valete. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));



                }
```

```
            if(indiceCarta == -1){

                    //Tentando encontrar um Rei, que faz o jogador anterior comprar
uma carta

                    indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][Carta.REI]);

                    if(indiceCarta != -1){


        controleJogo.getJogador(controleJogo.getIndiceJogadorAnterior()).adicionarCart
a(baralho.comprarCarta());

                            baralho.adicionarCarta(
(Carta)cartasAtuais.elementAt(indiceCarta) );

                            cartasAtuais.removeElementAt(indiceCarta);

                            mensagem = controleJogo.new Mensagem("O jogador " +
getNome() + " jogou um Rei e, como consequência, o jogador anterior comprou uma
carta. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                    AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));




                    }

            }


            if(indiceCarta == -1){

                    //Se o próximo jogador tem menos que 5 cartas, tenta encontrar
uma dama para pular o jogador


        if(controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).numeroCart
as() < 5 && baralho.cartaAtual().getValor() != Carta.DAMA){
```

```
                                indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][Carta.DAMA]);

                                if(indiceCarta != -1){

                                        baralho.adicionarCarta(
(Carta)cartasAtuais.elementAt(indiceCarta) );

                                        cartasAtuais.removeElementAt(indiceCarta);

                                        controleJogo.proximoJogador();

                                        mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " jogou uma Dama e, como consequência, o próximo jogador foi
pulado. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));



                                }



                        }

                }


                if(indiceCarta == -1){

                        //Procurando carta do mesmo naipe


                        for(int i = Carta.PRIMEIRO_VALOR ; (i <= Carta.ULTIMO_VALOR)
&& !encontrou; i++){

                                indiceCarta =
getIndiceCarta(baralho.cartasBaralho[baralho.cartaAtual().getNaipe()][i]);

                                if(indiceCarta != -1){
```

```
                        encontrou = true;

                    }
                }
                if(!encontrou){
                    //Procurando carta do mesmo valor
                    for(int i = Carta.PRIMEIRO_NAIPE; (i <=
Carta.ULTIMO_NAIPE) && (!encontrou); i++){
                        indiceCarta =
getIndiceCarta(baralho.cartasBaralho[i][baralho.cartaAtual().getValor()]);
                        if(indiceCarta != -1){
                            encontrou = true;

                        }
                    }
                }
                if(!encontrou){
                    //Se não encontrou carta de mesmo valor ou naipe, procura
um coringa
                    indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.OUROS][Carta.CORINGA]);
                    if(indiceCarta == -1) indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.PAUS][Carta.CORINGA]);
                    if(indiceCarta == -1) indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.COPAS][Carta.CORINGA]);
                    if(indiceCarta == -1) indiceCarta =
getIndiceCarta(Baralho.cartasBaralho[Carta.ESPADAS][Carta.CORINGA]);
```

```
                        if(indiceCarta == -1){

                                mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " comprou 1 carta. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                        AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));

                                adicionarCarta(baralho.comprarCarta());

                        }else{

                                //Coringa encontrado

                                mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " jogou um coringa, que anula o efeito de qualquer carta especial e
pode ser jogado sobre qualquer carta. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                        AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));


        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

                                cartasAtuais.removeElementAt(indiceCarta);

                                controleJogo.setQuantidadeCompra((byte)0);



                        }


                }else{

                        if( ((Carta)cartasAtuais.elementAt(indiceCarta)).getValor() ==
Carta.DAMA){

                                String nomeJogadorPulado =
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome();
```

```
                                controleJogo.proximoJogador();

                                mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " jogou uma Dama e, como consequência, o próximo jogador foi
pulado. O próximo jogador será: " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));

                            }else

                            if( ((Carta)cartasAtuais.elementAt(indiceCarta)).getValor() ==
Carta.CINCO){

                                controleJogo.inverterSentidoJogo();

                                mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " jogou um 5 e alterou o sentido do jogo. O próximo jogador será: " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));



                            }else{

                                if(
((Carta)cartasAtuais.elementAt(indiceCarta)).getValor() == Carta.CORINGA ){

                                    mensagem = controleJogo.new Mensagem("O
jogador " + getNome() + " jogou um coringa, que pode ser jogado sobre qualquer carta
e anula o efeito de qualquer carta especial. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                    AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));



                                }else{
```

```
                                                    if(
((Carta)cartasAtuais.elementAt(indiceCarta)).getValor() == Carta.OITO){

                                                        //Verifica se a carta escolhida foi um oito


        if(controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).numeroCart
as() > 1){


        ControleJogo.exibirDebugMsg("JogadorComputador>>VerificarJogada(8*)>>Co
mprando uma carta do próximo jogador");


        adicionarCarta(controleJogo.getJogador(controleJogo.getIndiceProximoJogador()
).removerCarta());

                                                            mensagem = controleJogo.new
Mensagem("O jogador " + getNome() + " jogou um 8. Com esta jogada, ele está
recebendo uma carta do próximo jogador e pode jogar novamente!",

                                                            AlertType.INFO,
controleJogo.getJogador(controleJogo.getJogadorAtual()));



                                                    }else{


        ControleJogo.exibirDebugMsg("JogadorComputador>>VerificarJogada(8(*))>>Pr
óximo jogador possui apenas uma carta...");

                                                            mensagem = controleJogo.new
Mensagem("O jogador " + getNome() + " jogou um 8. Com esta jogada, ele estaria
recebendo uma carta do próximo jogador e poderia jogar novamente, porém o próximo
jogador possui apenas uma carta. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                                            AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));
```

```
                                        }


                                    }else{

                                            mensagem = controleJogo.new
Mensagem("O jogador " + getNome() + " fez uma jogada legal e não jogou nenhuma
carta especial. O próximo jogador será " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",

                                                    AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()));

                                    }

                                }

                            }


        baralho.adicionarCarta((Carta)cartasAtuais.elementAt(indiceCarta));

                                cartasAtuais.removeElementAt(indiceCarta);




                    }

                }

        }
                                                                        }
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }

                                                }


                                            }catch(EmptyStackException e){
```

```
        ControleJogo.exibirDebugMsg("JogadorComputador>>jogar()>>Verificando
jogada>>Baralho está vazio!");
                                                        mensagem =
JogadorComputador.this.controleJogo.elegerVencedor();
                                                    }

        JogadorComputador.this.setDisseMauMau(false);

        if(ControleJogo.JOGO_EM_ANDAMENTO){ //Verifica se o jogador ainda está
jogando

        ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando
andamento>>JOGO_EM_ANDAMENTO>>" +
ControleJogo.JOGO_EM_ANDAMENTO);
                                                        if(numeroCartas() > 0){
                                                            //Fazendo o computador
dizer mau-mau ou não de acordo com o número de cartas
                                                        if(numeroCartas() == 1)
setDisseMauMau( ControleJogo.getNumeroAleatorio() > 5);
                                                        if(numeroCartas() == 1 &&
getDisseMauMau() == false){

        mensagem.setString(mensagem.getString() + " O jogador possui apenas uma
carta e não disse mau-mau, portanto ele comprou três cartas. O próximo jogador será "
+ controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".");
                                                                try{
                                                                    for(byte i = 0; i
< 3; i++)

        adicionarCarta(baralho.comprarCarta());

        }catch(EmptyStackException e){


        ControleJogo.exibirDebugMsg("JogadorComputador>>jogar()>>Verificando
jogada (disse mau-mau)>>Baralho está vazio!");
                                                                    mensagem =
JogadorComputador.this.controleJogo.elegerVencedor();

                                                                }catch(Exception e){

        ControleJogo.exibirDebugMsg("JogadorComputador>>Verificando Disse Mau-
Mau>>Erro>> " + e);
                                                                }
                                                        }
```

```
                                                        if(indiceJogoAtual ==
controleJogo.getJogoAtual())

        CartasMidlet.getDisplay().setCurrent(mensagem);
                                                        else

        controleJogo.exibirDebugMsg("JogadorComputador>>threadJogada>>THREAD
É DO JOGO ANTERIOR");
                                                    }else{
                                                        if(indiceJogoAtual ==
controleJogo.getJogoAtual())

        controleJogo.setVencedor(JogadorComputador.this);
                                                        else

        controleJogo.exibirDebugMsg("JogadorComputador>>threadJogada>>THREAD
É DO JOGO ANTERIOR");

                                                    }

                                                }

        ControleJogo.exibirDebugMsg("******************************************************")
;

                                            }catch(Exception e){

        ControleJogo.exibirDebugMsg("JogadorComputador>>jogar()>>ERRO: " + e);

                                            }


                                        }









        }

        public threadJogada jogada = new threadJogada();


        /**
```

```
		Construtor
	*/
	public JogadorComputador(String nome, Baralho baralho, ControleJogo
controleJogo){
		super(nome, baralho, controleJogo);
	}


	/**
		Método invocado para realizar a jogada do computador
	*/
	public void jogar(Object arg[]){
		if(ControleJogo.ONLINE){
			/*
				Rotina de jogo para jogo ONLINE -->Nesta versão o jogador
computador
				não participa do jogo on-line
			*/

	ControleJogo.exibirDebugMsg("JogadorComputador>>AGUARDANDO JOGADA
DO PRóXIMO JOGADOR");
		}else{
			//Rotina de jogo para jogo OFFLINE
			ControleJogo.exibirDebugMsg("JogadorComputador>>jogar()");
			jogada.setJogoAtual(controleJogo.getJogoAtual());
			Thread thread = new Thread(jogada);
			thread.start();

			System.gc();
		}
	}

	/**
		Desenha a interface gráfica do jogador
	*/
	public void paint(Graphics g){
		super.paint(g);

		if(ControleJogo.getOpenMauMau()){
			//Desenha as cartas
			posicao = 12;

			if( ( (cartasAtuais.size() + 2) * 12) < Dispositivo.getLarguraTela()){
				if(ControleJogo.DEBUG_MODE)
					System.out.println("Jogador
Humano>>paint>>Ajustando posição: nº de cartas < tamTela");
```

```
                        posicao = (Dispositivo.getLarguraTela() - (
(cartasAtuais.size() + 2) * 12)) / 2;
                    }else{
                        if( ((cartaSelecionada + 2) * 12) + posicao >
Dispositivo.getLarguraTela()){
                            if(ControleJogo.DEBUG_MODE)
                            System.out.println("Jogador
Humano>>paint>>Ajustando posição: nº de cartas > tamTela");
                            posicao = - ( ((cartaSelecionada + 2) * 12) -
Carta.LARGURA_CARTA );
                        }

                    }

                    for(int i = 0 ; i < cartasAtuais.size(); i++, posicao += 12){
                        Carta carta = (Carta)cartasAtuais.elementAt(i);
                        if(i != cartaSelecionada){
                            carta.setPosition(posicao, (Dispositivo.getAlturaTela() -
carta.getHeight()) / 2);
                        }else{
                            carta.setPosition(posicao, ((Dispositivo.getAlturaTela()
- carta.getHeight()) / 2) - 15);

                        }
                        carta.paint(g);
                    }



                }else{
                    //Exibir apenas o fundo da carta e o número decartas
                    Baralho.cartaVerso.setPosition(((Dispositivo.getLarguraTela() -
Baralho.cartaVerso.getWidth()) / 2) ,
                                                    (Dispositivo.getAlturaTela() -
Baralho.cartaVerso.getHeight()) / 2);
                    Baralho.cartaVerso.paint(g);
                }

                g.setColor(255,255,255);
                g.drawString("Nº de Cartas: " + cartasAtuais.size(),
(Dispositivo.getLarguraTela() - Font.getDefaultFont().stringWidth("Nº de Cartas: " +
cartasAtuais.size())) / 2,
                        (Dispositivo.getCentroY() + Baralho.cartaVerso.getHeight()/2)
+ 1, 0);
                g.drawString(getNome(),
                        (Dispositivo.getLarguraTela() -
Font.getDefaultFont().stringWidth(getNome()) ) / 2,
```

```
                                (Dispositivo.getCentroY() + Baralho.cartaVerso.getHeight()/2)
+ 1 + Font.getDefaultFont().getHeight() + 1,0);



        }

        /**
                Método invocado quando o jogador pressiona alguma tecla
        */
        public void keyPressed(int keyCode){
                synchronized(controleJogo){
                super.keyPressed(keyCode);
                repaint();
                }
        }

}
```

<u>JogadorHumano.java</u>

```
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 01/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criação da classe
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
Data: 14/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando comando para comprar carta do baralho
          Adicionando comando para dizer mau mau
          Modificando rotinas de debug para utilizarem o método exibirDebugMsg da
                classe ControleJogo
Status: ok
```

%%
%%
%%

%%
%%
%%
Data: 19/09/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina de tratamento de commandos

Status: ok
%%
%%
%%

%%
%%
%%
Data: 09/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina para verificar se o usuário falou "mau-mau"
Adicionando rotinas de simulação do fim do jogo.
Status: ok
%%
%%
%%

%%
%%
%%
Data: 20/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando rotina de jogo on-line
Status: ok
%%
%%
%%

%%
%%
%%
Data: 26/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando mensagens
Criando lógica para penalidades

```
Status: ok
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%


*/

package jogo;

import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;
import java.util.*;
import jogo.apibasica.*;



/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
<p>Projeto de conclusão de curso: Técnicas de desenvolvimento de jogos para
dispositivos móveis
<p>Classe: jogo.JogadorHumano
<p>Responsabilidades: Guarda as informações sobre o jogador que está utilizando o
dispositivo móvel

@author Rogério de Paula Aguilar
@version  1.0

*/
public class JogadorHumano extends Jogador implements CommandListener{

	/**
		Classe que guarda informações sobre o dispositivo
	*/
	private Dispositivo dispositivo;

	/**
		Comando utilizado para comprar uma carta
	*/
	private Command cmdComprarCarta = new Command("Comprar Carta",
Command.SCREEN, 0);

	/**
		Comando que o usuário seleciona quando quer dizer mau-mau
	*/
```

```
	private Command cmdDizerMauMau = new Command("Dizer \"Mau Mau \"",
Command.SCREEN, 0);

	/**
		Comando para compra de três ou mais cartas
	*/
	private Command cmdComprarTresCartas;


	/**
		Simula o fim de jogo
	*/
	public void simularFimDeJogo(){
		cartasAtuais = new Stack();

	cartasAtuais.push(baralho.cartasBaralho[Carta.COPAS][Carta.CORINGA]);
		cartasAtuais.push(baralho.cartasBaralho[Carta.PAUS][Carta.DOIS]);

	}

	/**
		Construtor
	*/
	public JogadorHumano(String nome, Dispositivo dispositivo, Baralho baralho,
ControleJogo controleJogo){
		super(nome, baralho, controleJogo);
		this.dispositivo = dispositivo;
		addCommand(cmdComprarCarta);
		addCommand(cmdDizerMauMau);
		setCommandListener(this);

	}

	/**
		Adiciona o comando de compra de cartas na tela do usuário
	*/
	public void setComandoCompra(byte tipo){
		//0 --> Adiciona commando para comprar n cartas
		//1 --> Adiciona commando para comprar 1 carta
		removeCommand(cmdComprarTresCartas);
		removeCommand(cmdComprarCarta);
		if(tipo == 0){
			cmdComprarTresCartas = new Command("Comprar " +
ControleJogo.getQuantidadeCompra() + " Cartas.", Command.SCREEN, 0);
			addCommand(cmdComprarTresCartas);
		}else{
			addCommand(cmdComprarCarta);
```

```
                }
                if(numeroCartas() > 1) addCommand(cmdDizerMauMau);
                cartaSelecionada = 0;

        }

        /**
                Chama o método verificarJogada de ControleJogo para verificar a jogada,
exibe
                uma mensagem para o usuário<br> e envia a atualização para o servidor,
caso o jogo seja on-line
        */
        public void jogar(Object args[]){
                ControleJogo.exibirDebugMsg("JogadorHumano>>jogar");
                ControleJogo.Mensagem m = null;

                if(args == null){
                        m = controleJogo.verificarJogada((
(Carta)cartasAtuais.elementAt(cartaSelecionada) ), this);
                }else{
                        if(args[0] instanceof ControleJogo.Mensagem){
                                m = (ControleJogo.Mensagem)args[0];
                        }
                }
                int strValete = m.getString().indexOf("jogou um valete");
                System.out.println("STRVALETE: " + strValete);
                if(strValete == -1 || args != null && args[0] instanceof
ControleJogo.Mensagem){
                        if(m.getType() != AlertType.ERROR)
                                removerCarta(cartaSelecionada);
                        m.setTimeout(ESPERA);


                        //Verificando vencedor
                        if(m.getType() != AlertType.ERROR && numeroCartas() == 0){

        controleJogo.exibirDebugMsg("JogadorHumano>>jogar>>JOGADOR HUMANO
VENCEU!!!!!");
                                m = controleJogo.new Mensagem("Fim de jogo. O vencedor
foi: " + getNome()
                                , AlertType.INFO, controleJogo);
                                m.setTimeout(10000);
                                controleJogo.setApresentacao(true);
                                controleJogo.setTelaSelecao(true);
                                controleJogo.enviarMsgServidor("VENCEDOR_JOGO:" +
getNome());
```

```
                        controleJogo.fecharConexaoSemConfirmacao();
                    }

                    if(controleJogo.ONLINE && m.getType() != AlertType.ERROR)
                        m.setTimeout(300000);

                    CartasMidlet.getDisplay().setCurrent(m);

                    if(controleJogo.ONLINE && m.getType() != AlertType.ERROR &&
numeroCartas() != 0){

        controleJogo.enviarMsgServidor(controleJogo.montarStatusJogo(m, false));
                        System.gc();
                    }
                }



        }

        /**
                Método invocado quando o usuário pressiona alguma tecla
        */
        public void keyPressed(int keyCode){
              synchronized(controleJogo){
                super.keyPressed(keyCode);


                if(getKeyCode(FIRE) == keyCode){
                    if(controleJogo.getJogadorAtual() !=
controleJogo.getJogadorExibido()){

        ControleJogo.exibirDebugMsg("JogadorHumano>>keyPressed>>Jogador está
exibindo cartas de outro jogador");
                    }else{
                        jogar(null);
                    }
                }



                repaint();
              }

        }

        /**
```

```
		Desenha a interface gráfica do jogador
	*/
	public  void paint(Graphics g){
		super.paint(g);
		posicao = 12;

		if( ( (cartasAtuais.size() + 2) * 12) < dispositivo.getLarguraTela()){
			ControleJogo.exibirDebugMsg("Jogador
Humano>>paint>>Ajustando posição: nº de cartas < tamTela");
			posicao = (dispositivo.getLarguraTela() - ( (cartasAtuais.size() + 2) *
12)) / 2;
		}else{
			if( ((cartaSelecionada + 2) * 12) + posicao >
dispositivo.getLarguraTela()){
				ControleJogo.exibirDebugMsg("Jogador
Humano>>paint>>Ajustando posição: nº de cartas > tamTela");

				posicao = - ( ((cartaSelecionada + 2) * 12) -
Carta.LARGURA_CARTA );
			}

		}

		Carta objcartaSelecionada = null;
		for(int i = 0 ; i < cartasAtuais.size(); i++, posicao += 12){
			Carta carta = (Carta)cartasAtuais.elementAt(i);
			if(i != cartaSelecionada){
				carta.setPosition(posicao, (dispositivo.getAlturaTela() -
carta.getHeight()) / 2);

			}else{
				carta.setPosition(posicao, ((dispositivo.getAlturaTela() -
carta.getHeight()) / 2) - 15);
				objcartaSelecionada = carta;
			}
				carta.paint(g);

		}
		//objcartaSelecionada.paint(g);

		g.setColor(0,0,255);
				g.drawString("Nº de Cartas: " + cartasAtuais.size(),
(Dispositivo.getLarguraTela() - Font.getDefaultFont().stringWidth("Nº de Cartas: " +
cartasAtuais.size())) / 2,
						(Dispositivo.getCentroY() +
Baralho.cartaVerso.getHeight()/2) + 1, 0);
```

```
                    g.drawString(getNome(),
                                (Dispositivo.getLarguraTela() -
Font.getDefaultFont().stringWidth(getNome()) ) / 2,
                                (Dispositivo.getCentroY() +
Baralho.cartaVerso.getHeight()/2) + 1 + Font.getDefaultFont().getHeight() + 1,0);

        }

        /**
                Procedimento chamado quando o jogador aciona algum comando
        */
        public void commandAction(Command c, Displayable d){
                synchronized(controleJogo){
                ControleJogo.Mensagem mensagem = null;
                super.commandAction(c, d);
                if(controleJogo.getJogadorAtual() == controleJogo.getJogadorExibido()){


                  try{
                        if(c == cmdComprarCarta){
                                controleJogo.setPenalidadeAplicada(true);

        ControleJogo.exibirDebugMsg("JogadorHumano>>commandAction()>>Compran
do uma carta");
                                if(getDisseMauMau()){
                                        mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " comprou uma carta e receberá mais três por ter dito mau-mau na
hora errada. O próximo jogador será: " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",
                                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()) );
                                        for(byte i = 0; i < 4; i++)
                                                adicionarCarta(baralho.comprarCarta());

                                }else{
                                        mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " comprou uma carta. O próximo jogador será: " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",
                                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()) );
                                        adicionarCarta(baralho.comprarCarta());
                                }

                        }else if(c == cmdComprarTresCartas){
```

```
        ControleJogo.exibirDebugMsg("JogadorHumano>>commandAction()>>Compran
do " + controleJogo.getQuantidadeCompra() + " cartas");
                                for(int i = 0; i < controleJogo.getQuantidadeCompra(); i++){
                                        adicionarCarta(baralho.comprarCarta());
                                }

                                if(getDisseMauMau()){
                                        for(int i = 0; i < 3; i++){
                                                adicionarCarta(baralho.comprarCarta());
                                        }


                                        mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " comprou " + controleJogo.getQuantidadeCompra() + " cartas e
receberá mais três por ter dito mau-mau na hora errada. O próximo jogador será: " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",
                                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()) );


                                }else{
                                        mensagem = controleJogo.new Mensagem("O jogador
" + getNome() + " comprou " + controleJogo.getQuantidadeCompra() + " cartas. O
próximo jogador será: " +
controleJogo.getJogador(controleJogo.getIndiceProximoJogador()).getNome() + ".
Movendo para o próximo jogador.",
                                                AlertType.INFO,
controleJogo.getJogador(controleJogo.proximoJogador()) );

                                }
                                controleJogo.setPenalidadeAplicada(true);

                                /*if(controleJogo.getQuantidadeCompra() % 3 == 0)
                                        controleJogo.setQuantidadeCompra((byte)3);*/


                        }else if(c == cmdDizerMauMau){
                                setDisseMauMau(true);
                                removeCommand(cmdDizerMauMau);

                        }
                }catch(EmptyStackException e){
                        mensagem = controleJogo.elegerVencedor();

                }
```

```
                //if(!ControleJogo.ONLINE)
                        CartasMidlet.getDisplay().setCurrent(mensagem);
                if(ControleJogo.ONLINE && mensagem != null)

        controleJogo.enviarMsgServidor(controleJogo.montarStatusJogo(mensagem,
false));

          }
         }
        }
}
```

- **CLASSES DA APLICAÇÃO SERVIDOR**

JogadorDuplicadoException.java

```
package jogo.rede;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Interface: JogadorDuplicadoException

<p>Descrição: Exception que é lançada quando o usuário tenta entrar com um apelido
                que já existe na sala
*/

class JogadorDuplicadoException extends Exception{
        /**
                Construtor
        */
        public JogadorDuplicadoException(String msg){
                super(msg);
        }

}
```

JogadorNaoEncontradoException.java

```
package jogo.rede;
/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Interface: JogadorNaoEncontradoException

<p>Descrição: Exception que é lançada quando um jogador não é encontrado
*/

class JogadorNaoEncontradoException extends Exception{
	/**
		Construtor
	*/
	public JogadorNaoEncontradoException(String msg){
		super(msg);
	}

}
```

JogandoException.java

```
package jogo.rede;


/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Interface: JogandoException

<p>Descrição: Exception que é lançada quando o usuário tenta entrar numa sala
```

```
                em que o jogo está em andamento
*/
class JogandoException extends Exception{

        /**
                Construtor
        */
        public JogandoException(String msg){
                super(msg);
        }
}
```

NumeroJogadoresExcedidoException.java

```
package jogo.rede;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Interface: NumeroJogadoresExcedidoException

<p>Descrição: Exception que é lançada quando o usuário tenta entrar numa sala
                que está cheia

*/

class NumeroJogadoresExcedidoException extends Exception{
        /**
                Construtor
        */
        public NumeroJogadoresExcedidoException(String msg){
                super(msg);
        }
}
```

MontarArvore.java

```
package jogo.rede;

import javax.swing.*;
```

```
import javax.swing.tree.*;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Interface: MontarArvore

<p>Descrição: Interface utilizada para montar os nós da árvore
                de salas do servidor
*/

public interface MontarArvore{

        /**
                Método que deve ser implementado pelas classes que queiram fazer parte
da árvore do servidor
        */
        public DefaultMutableTreeNode retornarNo(JTree arvore);
}
```

JogadorRede.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Rogério de Paula Aguilar 8NA
Luiz Fernando P.S. Forgas 8NA

Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

Pacote: jogo.rede
Classe: JogadorRede

Descrição: Guarda as informações sobre os jogadores
da rede.
*/

package jogo.rede;
```

```
import java.io.*;
import java.net.*;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Classe: JogadorRede

<p>Descrição: Guarda as informações sobre os jogadores
da rede.

@author Rogério de Paula Aguilar
@version 1.0
*/

public class JogadorRede implements java.io.Serializable{

	/**
		Nome do jogador
	*/
	private String nome;

	/**
		Objeto utilizado para enviar mensagens para o jogador
	*/
	private PrintWriter writer;

	/**
		Objeto utilizado para ler mensagens que vem do cliente
	*/
	private BufferedReader reader;

	/**
		Indica se a conexão deve ser mantida
	*/
	private boolean MANTER_CONEXAO;

	/**
		IP da aplicação cliente
```

```
	*/
	private String IP;

	/**
		Porta da aplicação cliente
	*/
	private String PORTA;

	/**
		Objeto utilizado para estabelecer a comunicação
	*/
	private Socket socket;

	/**
		Sala associada com este jogador
	*/
	private Sala sala;

	/**
		Classe que fica esperando por comandos da aplicação cliente e processa
estes comandos
	*/
	private Thread threadLeitura = new Thread(){
		public void run(){
			System.out.println("Jogador:" + getNome() + ">>Inicializando thread
de leitura.");

			while(JogadorRede.this.MANTER_CONEXAO){
				try{
					String str = reader.readLine();
					synchronized(JogadorRede.this){
					if(str != null){
						sala.log("Jogador:" + getNome() +
">>MENSAGEM RECEBIDA: " + str);

						if(str.startsWith("SAIR_CLIENTE")){
							Thread threadEnviarSaida = new
Thread(){
								public void run(){
									synchronized(sala){
										retirarJogadorSala();

	sala.enviarMsgBroadcast("SAIR_SALA: Jogador " + getNome() + " saiu da sala.
O jogo foi cancelado.");
										fechar();
										sala.resetar();
								}
```

```
                                        }
                                    };
                                    try{
                                        threadEnviarSaida.start();
                                        threadEnviarSaida.join(300000);
                                    }catch(Exception e){
                                    }

                                }else if(str.startsWith("ST")){
                                    synchronized(sala){
                                        sala.setJogando(true);

                                        int indiceJN = str.indexOf("JN");
                                        int quantJogadoresStatus = 0;
                                        while(indiceJN != -1){
                                            quantJogadoresStatus++;
                                            if(indiceJN + 1 < str.length())
                                                indiceJN =
str.indexOf("JN", indiceJN + 1 );
                                            else
                                                indiceJN = -1;
                                        }
                                        if(quantJogadoresStatus <
sala.getJogadoresAtivos()){
                                            sala.log("Sala>>" +
sala.getId() + ">>Jogador>>" + getNome() + ">>Quantidade de jogadores na mensagem
de status != JogadoresAtivos>>Recomeçando jogo");
                                            StringBuffer buffer = new
StringBuffer();
                                            for(int i = 0; i <
sala.NUMERO_JOGADORES; i++){

        if(sala.getJogador(i) != null){

        buffer.append("[Jogador: " + sala.getJogador(i).getNome() + "]");
                                                    }
                                            }
                                            sala.log("Enviando lista de
jogadores para o cliente para recomeço de jogo: " + buffer.toString().trim());

        sala.getPrimeiroJogador().enviarMensagem("RECOMECO_ENTRADA_JOGADO
R" + buffer.toString().trim());

                                        }else{
```

```
                                                    sala.log("Sala>>" +
sala.getId() + ">>Jogador>>" + getNome() + ">>Quantidade de jogadores na mensagem
de status: " + quantJogadoresStatus);
                                                    sala.log("Sala>>" +
sala.getId() + ">>Jogador>>" + getNome() + ">>Quantidade de jogadores na sala:" +
sala.getJogadoresAtivos());

                                                    //if(!sala.getJogando()){

                                                    //}
                                                    sala.log("Sala>>" +
sala.getId() + ">>Jogador:" + getNome() + ">>MENSAGEM DE STATUS RECEBIDA: "
+ str);

        sala.enviarMsgBroadcast(str);
                                                }

                                            }
                                        }else if(str.startsWith("VENCEDOR_JOGO:")){
                                            class threadVencedor implements
Runnable{
                                                private String str;
                                                private Sala sala;

                                                public threadVencedor(String str,
Sala sala){
                                                    this.str = str;
                                                    this.sala = sala;
                                                }

                                                public void run(){


                                                    sala.log("Sala>>" +
sala.getId() + ">>Jogador:" + getNome() + ">>VENCEDOR DETECTADO:" + str);
                                                    retirarJogadorSala();

        sala.enviarMsgBroadcast(str);
                                                    fechar();

                                                    sala.resetar();
                                                }
                                            }
                                            try{
                                                Thread tV = new Thread(new
threadVencedor(str, sala));
                                                tV.start();
```

```
                                        tV.join(300000);
                                    }catch(Exception e){
                                    }



                                }else if(str.startsWith("ACKATUALIZACAO")){
                                    synchronized(sala){
                                        sala.addAck();
                                        if(sala.getAck() ==
sala.getJogadoresAtivos()){
                                            sala.zerarAck();
                                            sala.log("Sala>>" +
sala.getId() + ">>Jogador:" + getNome() + ">>Confirmação de atualizações nos
clientes");

        sala.enviarMsgBroadcast("PODE_JOGAR");
                                        }

                                    }
                                }

                            }
                            else if(socket != null && !(socket.isConnected())){

                                System.out.println("Jogador>>" + getNome() +
">>Fechando conexão");
                                synchronized(sala){
                                    retirarJogadorSala();
                                    sala.enviarMsgBroadcast("SAIR_SALA:
Jogador " + getNome() + " saiu da sala. O jogo foi cancelado.");
                                    fechar();
                                    sala.resetar();
                                }

                            }
                            Thread.sleep(100);
                        }
                    }catch(Exception e){
                        sala.log("Jogador:" + getNome() + ">>ERRO NA
THREAD DE LEITURA: " + e);
                        synchronized(sala){
                            retirarJogadorSala();
                            sala.enviarMsgBroadcast("SAIR_SALA:
Jogador " + getNome() + " saiu da sala. O jogo foi cancelado.");
                            fechar();
                            sala.resetar();
```

```
                                }

                        }


                }

                System.out.println("Jogador:" + getNome() + ">>Encerrando thread
de leitura.");

        }
    };


    /**
        Construtor. Não inicializa a thread de leitura. Esta deve ser inicializada
manualmente
        através do método iniciarThreadLeitura
    */
    public JogadorRede(String nome, BufferedReader reader, PrintWriter writer, Sala
sala, Socket socket) throws IOException{
        this.nome = nome.toUpperCase().trim();
        MANTER_CONEXAO = true;
        if(reader == null || writer == null)
            throw new IllegalArgumentException("JogadorRede>>Socket deve
ser diferente de null");

        //Thread threadInicializar = new Thread(){
            this.socket = socket;
            this.reader = reader;
            this.writer = writer;

            this.PORTA = new Integer(socket.getPort()).toString();
            this.IP = socket.getInetAddress().getHostAddress();
        //}
        //threadLeitura.start();
        this.sala = sala;

    }


    /**
        Inicializa a thread de leitura de comandos.
    */
    public void iniciarThreadLeitura(){
        threadLeitura.start();
        try{
```

```
                        socket.setSoTimeout(180000); //Servidor pode ficar no máximo 3
minutos sem receber mensagens do cliente
                }catch(Exception e){
                        sala.log("Sala>>" + sala.getId() + ">>Jogador>>" + getNome() +
">>Erro ao setar timeout para o socket>>" + e);
                }
        }


        /**
                Retorna o nome do jogador
        */
        public String getNome(){
                return nome;
        }

        /**
                Retorna o ip da aplicação cliente
        */
        public String getIp(){
                return IP;
        }

        /**
                Retorna a porta da aplicação cliente
        */
        public String getPorta(){
                return PORTA;
        }

        /**
                Retorna se duas instâncias desta classe são iguais
        */
        public boolean equals(Object obj){
                return (obj instanceof JogadorRede) &&
((JogadorRede)obj).nome.equals(this.nome);
        }

        /**
                Retorna um inteiro para ser utilizado com índice de uma tabela hash
        */
        public int hashCode(){
                return nome.hashCode();
        }

        /**
                Fecha a conexão com o jogador
```

```
	*/
	public void fechar() {
		MANTER_CONEXAO = false;



		try{
			if(socket != null) socket.close();
		}catch(Exception e2){}


	}

	/**
		Envia uma mensagem para a aplicação cliente indicando que o servidor
irá desconectar o usuário
	*/
	public void desconectarUsuario(){
		try{
			if(writer != null){
				writer.println("SAIR_SERVIDOR");
		}
		}catch(Exception e2){}

	}

	/**
		Remove o jogador da sala. OBS: Não envia mensagem para o jogador.
Sempre utilizar o método<br>
		enviarMensagem para informar a aplicação cliente que o jogador está
sendo retirado da sala<br>
		antes de chamar este método.
	*/
	public void retirarJogadorSala(){
		try{
			sala.removerJogador(this);
		}catch(Exception e){
			System.out.println("Erro ao remover jogador: " + e);
		}

	}

	/**
		Retorna uma representação deste jogador num objeto String
	*/
	public String toString(){
		return "JogadorRede: " + getNome();
```

```
	}


	/**
		Modifica o valor de Manter conexão
	*/
	public synchronized void setManterConexao(boolean b){
		MANTER_CONEXAO = b;
	}

	/**
		Retorna o valor de Manter conexão
	*/
	public synchronized boolean getManterConexao(){
		return MANTER_CONEXAO;
	}

	/**
		Envia uma mensagem para o cliente
	*/
	public synchronized void enviarMensagem(String str){
		try{
			writer.println(str);
			writer.flush();
		}catch(Exception erro){
			/*retirarJogadorSala();
			sala.enviarMsgBroadcast("SAIR_SALA: Jogador " + getNome() + "
saiu da sala. O jogo foi cancelado.");
			fechar();
			sala.resetar();*/
			sala.log("Sala>>" + sala.getId() + ">>Jogador>>" + getNome() +
">>Erro ao enviar mensagem para o jogador>>" + erro);
		}
	}




}
```

Sala.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia
```

Rogério de Paula Aguilar 8NA
Luiz Fernando P.S. Forgas 8NA

Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

Pacote: jogo.rede
Classe: Sala

Descrição: Representa uma sala no servidor, onde
os jogadores se encontram para jogar

%%
%%
%%
Data: 21/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando thread que controla o timeout de início de jogo
Status: ok
%%
%%
%%

%%
%%
%%
Data: 26/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Adicionando mensagem que detecta o fim do jogo
Status: ok
%%
%%
%%

%%
%%
%%
Data: 04/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando thread de timeout
Status: ok
%%
%%
%%

```
*/

package jogo.rede;

import java.util.*;
import javax.swing.*;
import javax.swing.tree.*;


/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Classe: Sala

<p>Descrição: Representa uma sala no servidor, onde
os jogadores se encontram para jogar

*/

public class Sala implements MontarArvore{

	/**
		Número de jogadores na sala
	*/
	public static final int NUMERO_JOGADORES = 4;

	/**
		Jogadores da sala
	*/
	private JogadorRede jogadores[] = new JogadorRede[NUMERO_JOGADORES];

	/**
		Primeiro jogador que entrou na sala
	*/
	private JogadorRede primeiroJogador;

	/**
		Id da sala
	*/
```

```
private String id;

/**
	Jogadores ativos
*/
private int jogadoresAtivos = 0;

/**
	Indica se existe algum jogo em andamento nesta sala
*/
private boolean JOGANDO = false;

/**
	Objeto servidor
*/
private Servidor servidor;

/**
	Controla o tempo máximo de espera pelo início de um novo jogo
*/
private Thread timeout;

/**
	Controla o tempo máximo de espera pelo final de uma partida
*/

private Thread timeoutJogo;

/**
	Variável que controla a sincronização do jogo nos clientes
*/
private int ackAtualizacoes = 0; //Para sincronizar as atualizações nos clientes


/**
	Incrementa a variável ackAtualizacoes
*/
public synchronized void addAck(){
	ackAtualizacoes++;

}

/**
	Faz ackAtualizacoes = 0
*/
public synchronized void zerarAck(){
	ackAtualizacoes = 0;
```

```
	}

	/**
		Retorna o valor de ackAtualizacoes
	*/
	public synchronized int getAck(){
		return ackAtualizacoes;
	}


	/**
		Classe que controla o timeout para o início do jogo

	*/
	private class threadTimeout implements Runnable{
		private int TIMEOUT = 60000 * 15;//15 minutos


		public void run(){
			TIMEOUT = (int)(60000 * servidor.getTimeout());
			if(TIMEOUT <= 0) TIMEOUT = 60000 * 15;
			try{
				servidor.log("Sala>>" + id + ">>Inicializando thread
timeout>>TIMEOUT: " + (TIMEOUT/1000) + " segundos.");
				Thread.sleep(TIMEOUT);
				synchronized(Sala.this){
					if(!Sala.this.getJogando()){
						servidor.log("Sala>>" + id + ">>TIMEOUT:
Tempo para o início do jogo expirado>>Retirando jogadores e reinicializando sala");
						Sala.this.enviarMsgBroadcast("ERRO: O tempo
para o início de jogo expirou. Você foi desconectado pelo servidor. Reinicializando
jogo.");
						Sala.this.resetar();
						servidor.atualizarArvore();
					}

				}
			}catch(Exception e){
				servidor.log("Sala>>" + id + ">>Erro em threadTimeout>>" +
e);
				servidor.atualizarArvore();
			}
		}

	}
```

```
	/**
		Classe que controla o tempo limite de duração de um jogo
	*/
	private class threadTimeoutJogo implements Runnable{
		private int TIMEOUT = 60000 * 60;//1 hora

		public void run(){
			TIMEOUT = (int)(60000 * servidor.getTimeoutJogo());
			if(TIMEOUT <= 0) TIMEOUT = (int)60000 * 60;
			try{
				servidor.log("Sala>>" + id + ">>Inicializando thread de
timeout do JOGO>>TIMEOUT: " + (TIMEOUT/1000) + " segundos.");
				Thread.sleep(TIMEOUT);
				synchronized(Sala.this){
					if(Sala.this.getJogando()){
						servidor.log("Sala>>" + id + ">>TIMEOUT:
Tempo para o término do jogo expirado>>Retirando jogadores e reinicializando sala");
						Sala.this.enviarMsgBroadcast("ERRO: O tempo
para o término de jogo expirou. Você foi desconectado pelo servidor. Reinicializando
jogo.");
						Sala.this.resetar();
						servidor.atualizarArvore();
					}

				}
			}catch(Exception e){
				servidor.log("Sala>>" + id + ">>Exception em
threadTimeoutJogo>>" + e);
				servidor.atualizarArvore();
			}
		}

	}



	/**
	*/
	void notifyObservers(String str){
		System.out.println(str);
	}

	/**
		Construtor
	*/
	public Sala(String id, Servidor servidor){
```

```
		//notifyObservers("Criando sala. ID = " + id);
		this.id = id.trim();
		this.servidor = servidor;
	}

	/**
		Modifica o valor da variável JOGANDO
	*/
	public synchronized void setJogando(boolean jogando){
		JOGANDO = jogando;
		if(JOGANDO){
			if(timeout != null) timeout.interrupt();
			if(timeoutJogo != null) timeoutJogo.interrupt();
			timeoutJogo = new Thread(new threadTimeoutJogo());
			timeoutJogo.start();

		}else{
			if(timeoutJogo != null) timeoutJogo.interrupt();
		}
	}

	/**
		Retorna o valor da variável JOGANDO
	*/
	public synchronized boolean getJogando(){
		return JOGANDO;
	}

	/**
		Reseta a sala, desconectando todos que estiverem nela
	*/
	public synchronized void resetar(){
		servidor.log("Sala>>" + id + ">>Resetando sala");
		for(byte i = 0; i < NUMERO_JOGADORES; i++){
			if(jogadores[i] != null){
				try{
					notifyObservers("Sala>>ID = " + id + ">>Fechando
jogador " + i);
					jogadores[i].desconectarUsuario();
					jogadores[i].fechar();

				}catch(Exception e){
					notifyObservers("\nSala>>ID = " + id + ">>ERRO AO
FECHAR JOGADOR " + i + ": " + e);
				}
				jogadores[i] = null;
```

```
                }

            }

            jogadoresAtivos = 0;
            JOGANDO = false;
            if(timeout != null) timeout.interrupt();
            if(timeoutJogo != null) timeoutJogo.interrupt();
            ackAtualizacoes = 0;
            servidor.atualizarArvore();
            notifyObservers("Sala>>ID = " + id + ">>Sala resetada");

        }

        /**
            Retorna o id da sala
        */
        public String getId(){
            return id;
        }


        /**
            Adiciona um jogador na sala

        */
        public synchronized boolean adicionarJogador(JogadorRede jogador) throws
JogadorDuplicadoException, NumeroJogadoresExcedidoException, JogandoException{

            if(jogador == null){
                throw new IllegalArgumentException("Jogador deve ser diferente de
null");
            }
            if(JOGANDO){
                jogador.setManterConexao(false);
                throw new JogandoException("Sala>>ID = " + id + ">>Não é
possível adicionar jogador. Jogo em andamento");
            }
            for(byte i = 0; i < NUMERO_JOGADORES; i++){

                if(jogadores[i] != null && jogadores[i].equals(jogador)){
                    notifyObservers("Sala>>ID = " + id + ">>Não é possível
adicionar jogador " + jogador + ". Já existe um jogador com este nome");

                    jogador.setManterConexao(false);
```

```
                                throw new JogadorDuplicadoException("Sala>>ID = " + id +
">>Não é possível adicionar jogador " + jogador + ". Já existe um jogador com este
nome");


                        }

                }

                if(jogadoresAtivos == NUMERO_JOGADORES){
                        notifyObservers("Sala>>ID = " + id + ">>Não é possível adicionar
jogador " + jogador + ". Número de jogadores excedido.");
                        jogador.setManterConexao(false);
                        throw new NumeroJogadoresExcedidoException("Sala>>ID = " + id
+ ">>Não é possível adicionar jogador " + jogador + ". Número de jogadores
excedido.");

                }else{
                        if(jogadoresAtivos == 0){
                                primeiroJogador = jogador;
                                //servidor.log("INICIALIZANDO THREAD TIMEOUT");
                                timeout = new Thread(new threadTimeout());
                                timeout.start();
                        }
                        //primeiroJogador.enviarMensagem("ENTRADA_JOGADOR" +
jogador.getNome());


                        jogadores[jogadoresAtivos++] = jogador;
                        jogador.iniciarThreadLeitura();

                }
                if(jogadoresAtivos == 1)
                        return true;
                return false;
        }


        /**
                Remove os jogadores que não estão ativos
        */
        private void removerNulls(){
                for(byte i = 0; i < NUMERO_JOGADORES; i++){
                        if(jogadores[i] == null && i < NUMERO_JOGADORES - 1){
                                jogadores[i] = jogadores[i+1];
                                jogadores[i+1] = null;
                        }
```

```
                }

        }

        /**
                Remove um jogador da sala
        */

        public synchronized void removerJogador(JogadorRede jogador) throws
JogadorNaoEncontradoException{
                if(jogador == null)
                        throw new IllegalArgumentException("Sala>> ID = " + id +
">>removerJogador>>Jogador deve ser diferente de null");
                for(byte i = 0; i < NUMERO_JOGADORES; i++){

                        if(jogadores[i] != null && jogadores[i].equals(jogador)){
                          notifyObservers("Sala>>ID = " + id + ">>Removendo jogador:" +
jogador);
                          /*try{
                                jogadores[i].fechar();
                          }catch(Exception e){

                          }*/
                          jogadores[i] = null;
                          jogadoresAtivos--;
                          if(jogadoresAtivos == 0){
                                setJogando(false);
                                timeout.interrupt();
                          }
                          removerNulls();
                          servidor.atualizarArvore();
                          return;
                        }

                }


                throw new JogadorNaoEncontradoException("Sala>> ID = " + id +
">>removerJogador: " + jogador + ". Jogador não encontrado.");

        }

        /**
                Retorna o número de jogadores ativos
        */
        public synchronized int getJogadoresAtivos(){
                return jogadoresAtivos;
```

```
	}


	/**
		Retorna um jogador pelo seu índice
	*/
	public JogadorRede getJogador(int id){
		if(id >= NUMERO_JOGADORES)
			throw new IllegalArgumentException("getJogador>>Índice
inválido");
		else
			return jogadores[id];

	}

	/**
		Retorna uma representação da sala num objeto String
	*/
	public String toString(){
		StringBuffer buffer = new StringBuffer("Sala>>ID = " + id + "\n");
		for(int i = 0; i < NUMERO_JOGADORES; i++){
			if(jogadores[i] != null)
				buffer.append(">>>>" + jogadores[i] + "\n");
			else
				buffer.append(">>>>Jogador " + i  + " = vazio\n");

		}
		buffer.append("Jogadores ativos: " + jogadoresAtivos);

		return buffer.toString().trim();
	}

	/**
		Retorna o nó da árvore de salas correspondente a esta sala
	*/
	public DefaultMutableTreeNode retornarNo(JTree arvore){
		DefaultMutableTreeNode sala = new DefaultMutableTreeNode(getId());
		DefaultMutableTreeNode jogadoresAtivos = new
DefaultMutableTreeNode("Jogadores Ativos: " + getJogadoresAtivos());
		servidor.addUltimoNoSala(jogadoresAtivos);

		sala.add(jogadoresAtivos);
		for(int i = 0; i < NUMERO_JOGADORES; i++){
			if(jogadores[i] != null){
				DefaultMutableTreeNode no = new
DefaultMutableTreeNode(jogadores[i].toString());
```

```
                                no.add(new DefaultMutableTreeNode("IP: " +
jogadores[i].getIp()));
                                no.add(new DefaultMutableTreeNode("PORTA: " +
jogadores[i].getPorta()));
                                sala.add(no);


                        }
                        else
                                sala.add(new DefaultMutableTreeNode("Jogador " + i + ":
vazio"));

                }



                return sala;
        }


        /**
                Modifica o primeiro jogador
        */
        public void setPrimeiroJogador(JogadorRede primeiroJogador){
                this.primeiroJogador = primeiroJogador;
        }

        /**
                Retorna o primeiro jogador
        */
        public JogadorRede getPrimeiroJogador(){
                return primeiroJogador;
        }

        /**
                Envia uma mensagem a todos os jogadores da sala
        */
        public synchronized void enviarMsgBroadcast(String msg){
                for(int i = 0; i < NUMERO_JOGADORES; i++){
                        if(jogadores[i] != null)
                                jogadores[i].enviarMensagem(msg);

                }

        }
```

```
	/**
		Adiciona uma entrada ao log do servidor
	*/
	public void log(String str){
		servidor.log(str);
	}

	/**
		Retorna o servidor associado com esta sala
	*/
	public Servidor getServidor(){
		return servidor;
	}

	/**
		Retorna a thread que controla o tempo limite de fim de jogo
	*/

	public synchronized Thread getThreadTimeoutJogo(){
		return timeoutJogo;

	}


}
```

Servidor.java

```
/*
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

Rogério de Paula Aguilar 8NA
Luiz Fernando P.S. Forgas 8NA

Trabalho de Conclusão de Curso:
Técnicas de desenvolvimento de Jogos para dispositivos móveis

Pacote: jogo.rede
Classe: Servidor

Descrição: Servidor que gerencia as salas do jogo on-line
```

%%
%%
%%
Data: 13/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Criação da classe
Status: ok
%%
%%
%%

%%
%%
%%
Data: 15/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Modificação do protocolo (http para socket)
Status: ok
%%
%%
%%

%%
%%
%%
Data: 24/10/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando interface gráfica
Implementando ações relacionadas aos botões
Status: ok
%%
%%
%%

%%
%%
%%
Data: 04/11/2003
Responsável: Rogério de Paula Aguilar
Descrição: Arrumando interface gráfica
Status: ok
%%
%%
%%

```
*/


package jogo.rede;

import java.util.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.awt.*;
import java.awt.event.*;

import javax.swing.text.*;
import java.io.*;
import java.net.*;
import javax.net.*;
import java.util.prefs.*;

/**
2003 - Faculdade Senac de Ciências Exatas e Tecnologia

<p>Rogério de Paula Aguilar 8NA
<p>Luiz Fernando P.S. Forgas 8NA

<p>Trabalho de Conclusão de Curso:
<p>Técnicas de desenvolvimento de Jogos para dispositivos móveis

<p>Pacote: jogo.rede
<p>Classe: Servidor

Descrição: Servidor que gerencia as salas do jogo on-line

@author Rogério de Paula Aguilar
@version 1.0

*/
public class Servidor implements MontarArvore{



	/**
		Classe que implementa a interface gráfica com o usuário
	*/
	private class ServidorGUI extends JFrame implements Observer{

		/**
			Objeto servidor
		*/
```

```
		private Servidor servidor;

		/**
			Objeto de interface gráfica
		*/
		private JPanel painelStatus = new JPanel();

		/**
			Objeto de interface gráfica
		*/
		private JLabel lblStatus = new JLabel("Pronto", SwingConstants.LEFT);

		/**
			Objeto de interface gráfica que exibe o log da aplicação
		*/
		private JTextArea txtLog = new JTextArea();

		/**
			Objeto de interface gráfica que mostra uma árvore contendo as
salas e os usuário que etão conectados nestas salas
		*/
		private JTree arvore;

		/**
			Ação que implementa o mecanismo de sair do servidor
		*/
		private class acaoSair extends AbstractAction{

			public acaoSair(String acao, Icon icon){
				super(acao, icon);
			}
			public void actionPerformed(ActionEvent evt){
					parar();
					System.exit(0);
			}
		}


		/**
			Ação que implementa o mecanismo de atualização da árvore de
salas
		*/

		private class acaoAtualizarArvore extends AbstractAction{

			public acaoAtualizarArvore(String acao, Icon icon){
				super(acao, icon);
```

```
		}
		public void actionPerformed(ActionEvent evt){
			servidorGUI.atualizarArvore();
		}
	}


	/**
		Ação que implementa o mecanismo de limpar o log da aplicação
	*/

	private class acaoLimparLog extends AbstractAction{

		public acaoLimparLog(String acao, Icon icon){
			super(acao, icon);
		}
		public void actionPerformed(ActionEvent evt){
			txtLog.setText("");
		}
	}

	/**
		Ação que implementa o mecanismo de mudança de aparência da
interface gráfica
	*/
	private class mudarLookAndFeel implements ActionListener{
		public void actionPerformed(ActionEvent evt){
			try{
				if(evt.getSource() == menuAparenciaGTK)
					UIManager.setLookAndFeel(lookAndFeel[0]);

				else if(evt.getSource() == menuAparenciaWINDOWS)
					UIManager.setLookAndFeel(lookAndFeel[2]);
				else if(evt.getSource() == menuAparenciaMetal)
					UIManager.setLookAndFeel(lookAndFeel[1]);


	SwingUtilities.updateComponentTreeUI(ServidorGUI.this);
			}catch(Exception e){
				JOptionPane.showMessageDialog(null, "Erro ao
mudar a aparência! Erro:" + e);
			}
		}

	}

	/**
```

```
		Objeto ação
	*/
	private AbstractAction aSair = new acaoSair("Sair", new
ImageIcon("imagens/sair.gif"));

	/**
		Objeto ação
	*/
	private AbstractAction aAtualizarArvore = new
acaoAtualizarArvore("Atualizar árvore", new ImageIcon("imagens/arvore.gif"));

	/**
		Objeto ação
	*/
	private AbstractAction aLimparLog = new acaoLimparLog("Limpar Log",
new ImageIcon("imagens/limpar.gif"));

	private mudarLookAndFeel mudar = new mudarLookAndFeel();

	/**
		Objeto da interface gráfica
	*/
	private JButton cmdSair = new JButton(aSair);

	/**
		Objeto da interface gráfica
	*/
	private JMenuBar menus = new JMenuBar();

	/**
		Objeto da interface gráfica
	*/
	private JMenu menuOpcoes = new JMenu("Opções");

	/**
		Objeto da interface gráfica
	*/
	private JMenuItem menuSobre = new JMenuItem("Sobre");

	/**
		Objeto da interface gráfica
	*/
	private JMenuItem menuAtualizarArvore = new
JMenuItem(aAtualizarArvore);

	/**
		Objeto da interface gráfica
```

```
		*/
		private JMenuItem menuSair = new JMenuItem(aSair);

		/**
			Objeto da interface gráfica
		*/
		private JToolBar toolbar = new JToolBar(SwingConstants.HORIZONTAL);

		/**
			Objeto da interface gráfica
		*/
		private JMenuItem menuLimparLog = new JMenuItem(aLimparLog);

		/**
			Objeto da interface gráfica
		*/
		private JMenu menuAparencia = new JMenu("Aparência");

		/**
			Objeto da interface gráfica
		*/
		private JRadioButtonMenuItem menuAparenciaGTK = new
JRadioButtonMenuItem("Motif");

		/**
			Objeto da interface gráfica
		*/
		private JRadioButtonMenuItem menuAparenciaWINDOWS = new
JRadioButtonMenuItem("Windows");

		/**
			Objeto da interface gráfica
		*/
		private JRadioButtonMenuItem menuAparenciaMetal = new
JRadioButtonMenuItem("Metal", true);

		/**
			Objeto da interface gráfica
		*/
		private ButtonGroup grupoBotoes = new ButtonGroup();

		/**
			Objeto da interface gráfica
		*/
		private JScrollPane scrollLog;

		/**
```

```
		Aparências
	*/

	private String lookAndFeel[] = new
String[]{"com.sun.java.swing.plaf.motif.MotifLookAndFeel",

"javax.swing.plaf.metal.MetalLookAndFeel",

"com.sun.java.swing.plaf.windows.WindowsLookAndFeel"};



	/**
		Construtor
	*/
	public ServidorGUI(Servidor servidor){

		//Configurando descrição das ações
		aSair.putValue(Action.SHORT_DESCRIPTION, "Sai da aplicação");
		aAtualizarArvore.putValue(Action.SHORT_DESCRIPTION,
"Atualiza a árvore");
		aLimparLog.putValue(Action.SHORT_DESCRIPTION, "Limpa o log
da aplicação");


			//Configurando disposição da janela
			Toolkit tk = Toolkit.getDefaultToolkit();
			Dimension d = tk.getScreenSize();
			setIconImage(tk.getImage("imagens/MauMau.png"));
			setSize(d.width / 2, d.height / 2);
			setTitle("Mau-Mau Server - 1.0");

			this.servidor = servidor;
			txtLog.setEditable(false);

			painelStatus.setLayout(new FlowLayout());
			painelStatus.add(lblStatus, FlowLayout.LEFT);


			Container contentPane = getContentPane();
			//contentPane.setLayout(new GridBagLayout());
			contentPane.setLayout(new BorderLayout());

			//GridBagConstraints con = new GridBagConstraints();

			//Configurando e adicionando a toolbar
			toolbar.add(aAtualizarArvore);
```

```
                    toolbar.add(aLimparLog);
                    toolbar.add(aSair);

                    /*con.weightx = 1.0;
                    con.weighty = 1.0;

                    con.gridx = 0;
                    con.gridy = 0;
                    con.gridwidth = 1;
                    con.gridheight = 1;
                    con.fill = GridBagConstraints.BOTH;
                    contentPane.add(toolbar, con);*/
                    contentPane.add(toolbar, BorderLayout.NORTH);



                    //Configurando e adicionando a árvore
                    TreeNode raiz = servidor.retornarNo(arvore);
                    arvore = new JTree(raiz);
                    DefaultTreeSelectionModel modeloSelecao = new
DefaultTreeSelectionModel();

        modeloSelecao.setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECT
ION);
                    arvore.setSelectionModel(modeloSelecao);
                    JScrollPane scrollArvore = new JScrollPane(arvore);

                    scrollArvore.setPreferredSize(new Dimension(getWidth() / 2 ,
getHeight()));
                    contentPane.add(scrollArvore, BorderLayout.WEST);


                    JPanel painelLog = new JPanel();
                    painelLog.setLayout(new BorderLayout());
                    painelLog.add(new JLabel("Log:"), BorderLayout.NORTH);
                    scrollLog = new JScrollPane(txtLog);
                    //s.setAutoscrolls(true);
                    painelLog.setPreferredSize(new Dimension(getWidth(),
getHeight()));

                    painelLog.add(scrollLog, BorderLayout.CENTER);
                    /*con.gridx = 5;
                    con.gridy = 1;
                    con.gridwidth = 4;
                    con.gridheight = 9;
                    con.fill = GridBagConstraints.BOTH;
```

```
contentPane.add(painelLog, con);*/

contentPane.add(painelLog, BorderLayout.CENTER);




JPanel painelBaixo = new JPanel();
painelBaixo.setLayout(new FlowLayout(FlowLayout.RIGHT));
painelBaixo.add(cmdSair);
contentPane.add(painelBaixo, BorderLayout.SOUTH);



grupoBotoes.add(menuAparenciaGTK);
grupoBotoes.add(menuAparenciaWINDOWS);
grupoBotoes.add(menuAparenciaMetal);

//Menu

menuOpcoes.add(menuAtualizarArvore);
menuOpcoes.add(menuLimparLog);
menuOpcoes.addSeparator();
menuOpcoes.add(menuSair);
menuAparencia.add(menuAparenciaGTK);
menuAparencia.add(menuAparenciaWINDOWS);
menuAparencia.add(menuAparenciaMetal);
menuAparenciaGTK.addActionListener(mudar);
menuAparenciaWINDOWS.addActionListener(mudar);
menuAparenciaMetal.addActionListener(mudar);
menus.add(menuOpcoes);
menus.add(menuAparencia);


menuSobre.addActionListener(

        new ActionListener(){
                public void actionPerformed(ActionEvent evt){
                        JOptionPane.showMessageDialog(null , "2003 -
Faculdade Senac de Ciências Exatas e Tecnologia\n\n >>Trabalho de Conclusão de
Curso:\nTécnicas de Desenvolvimento de Jogos para Dispositivos Móveis\nServidor do
jogo Mau-Mau - versão 1.0\n\n>>Alunos:\nRogério de Paula Aguilar -
rogeriopaguilar@terra.com.br \nhttp:\\\\www.rogerioaguilar.cjb.net\n\nLuiz Fernando
P.S. Forgas - luizforgas@terra.com.br\n\n\"O raciocínio lógico leva você de A a B. A
imaginação leva você a qualquer lugar\"\n\n","Sobre",
JOptionPane.INFORMATION_MESSAGE, null);
```

```
                        }

                    }

            );
            JMenu mnuInt = new JMenu("?");
            menus.add(mnuInt);
            mnuInt.add(menuSobre);
            setJMenuBar(menus);

            addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent evt){
                    parar();
                    System.exit(0);
                }

            });

            pack();
            setLocation( (d.width - getWidth()) / 2, (d.height - getHeight()) / 2);



    }

    /**
            Modifica o status da aplicação
    */
    public void setStatus(String texto){
            lblStatus.setText(texto);
    }

    public void update(Observable obs, Object args){
            if(args instanceof String){
                    setStatus((String)args);
            }
            log("OBSERVADOR: " + args.toString());
            JOptionPane.showMessageDialog(null, "teste");

    }

    /**
            Adiciona uma entrada ao log da aplicação
    */
    public void log(String str){
            System.out.println("\n" + str);
            txtLog.append("\n" + str + "\n");
```

```
                JScrollBar barraHorizontal = scrollLog.getVerticalScrollBar();
                barraHorizontal.setValue(barraHorizontal.getMaximum());
        }

        /**
                Atualiza a árvore da aplicação
        */

        public void atualizarArvore(){
                //synchronized(this){
                        try{
                                noAtual = 0;
                                arvore.setModel(new
DefaultTreeModel(servidor.retornarNo(arvore)));
                                arvore.putClientProperty("JTree.lineStyle", "Angled");
                                DefaultTreeCellRenderer renderSala = new
DefaultTreeCellRenderer();
                                renderSala.setLeafIcon(new
ImageIcon("imagens/jogador.png"));
                                renderSala.setClosedIcon(new
ImageIcon("imagens/maumau.png"));
                                renderSala.setOpenIcon(new
ImageIcon("imagens/maumau.png"));

                                arvore.setCellRenderer(renderSala);

                                for(int i = 0; i < ultimoNo.length; i++){
                                        if(ultimoNo[i] != null){
                                                TreeNode nos[] =
((DefaultTreeModel)arvore.getModel()).getPathToRoot(ultimoNo[i]);
                                                TreePath path = new TreePath(nos);
                                                arvore.makeVisible(path);
                                        }
                                }
                        }catch(Exception e){
                                log("Exception ao atualizar árvore>>" + e);

                        }
                //}

        }



}
```

```
	/**
		Thread que fica esperando por conexões
	*/
	private class threadConexao implements Runnable{

		/**
			Socket que fica aguardando novas conexões
		*/
		private ServerSocket server;

		/**
			Construtor
		*/
		public threadConexao(ServerSocket server){
			this.server = server;
			new Thread(this).start();
		}


		/**
			Método que inicializa as variáveis e fica aguardando por conexões
		*/
		public void run(){
			servidorGUI.log("threadConexao>>Inicializando
threadConexao.Aguardando requisições...");
			while(SERVIDOR_RODANDO){
				try{
					Socket s = server.accept();
					s.setKeepAlive(true);
					servidorGUI.log("threadConexao>>Requisição
recebida...");

					new threadRequisicao(s);
				}catch(Exception e){
					servidorGUI.log("threadConexao>>ERRO: " + e);
				}
				try{
					Thread.sleep(200);
				}catch(Exception e){}

			}
			servidorGUI.log("threadConexao>>Encerrando threadConexao");
			System.out.println("threadConexao>>Encerrando threadConexao");


		}
```

```
	}

	/**
		Thread que atende uma requisição de um cliente
	*/
	private class threadRequisicao implements Runnable{

		private String requisicao = "";
		private Socket socket;
		private PrintWriter writer;
		private BufferedReader reader;

		/**
			COnstrutor
		*/
		public threadRequisicao(Socket socket) throws IOException{
			this.socket = socket;
			if(socket == null)
				throw new
IllegalArgumentException("threadRequisicao>>ERRO: socket = null");
			Thread threadAbrir = new Thread(){
				public void run(){
					try{
					    writer = new PrintWriter(new
OutputStreamWriter(threadRequisicao.this.socket.getOutputStream()), true);
					    reader = new BufferedReader(new
InputStreamReader(threadRequisicao.this.socket.getInputStream()));
					}catch(Exception e){
						//throw e;
					}
				}
			};
			threadAbrir.start();
			try{
				threadAbrir.join();
				new Thread(this).start();

			}catch(Exception e){

			}

		}

		/**
			Método que processa a requisição do usuário
		*/
```

```
		public void run(){
			servidorGUI.log("threadRequisição>>Atendendo requisição: IP --> "
+ socket.getRemoteSocketAddress()  + " PORTA --> " + socket.getPort());
			String tipoRequisicao = "";

			try{
				requisicao = reader.readLine();
				servidorGUI.log("threadRequisição>>Atendendo
requisição>>Tipo:" + requisicao);

			}catch(IOException e){
				servidorGUI.log("threadRequisicao>>Impossível ler
requisição");
			}
			if(requisicao != null){
				int indice = requisicao.indexOf("acao=");
				if(indice != -1){
					tipoRequisicao = requisicao.substring(
(requisicao.indexOf("=") + 1), (requisicao.indexOf("&") == -1 ? requisicao.length() :
requisicao.indexOf("&")) );
					System.out.println("TIPO DE REQUISIÇÃO: " +
tipoRequisicao);
					tipoRequisicao = tipoRequisicao.trim().toUpperCase();
					if(tipoRequisicao.equals("LISTA_SALAS")){
						//Retornando lista de salas

	servidorGUI.log("threadRequisicao>>Retornando lista de salas");
						try{
							writer.println(retornarSalasDisponiveis());
							writer.flush();

	servidorGUI.log("threadRequisicao>>Lista de salas enviada com sucesso");

						}catch(Exception e){
							servidorGUI.log("threadRequisicao>>Não
é possível enviar lista de salas>>ERRO: " + e);

						}finally{
							try{
								socket.close();
							}catch(Exception e){

	servidorGUI.log("threadRequisicao>>ERRO ao fechar socket:" + e);

							}
```

```
                                        }
                                    }else if(tipoRequisicao.equals("ENTRAR_SALA")){
                                        //Jogador está tentando entrar numa sala
                                        servidorGUI.log("threadRequisicao>>Tentando
entrar na sala");
                                        StringTokenizer tokens = new
StringTokenizer(requisicao, "&");
                                        if(tokens.countTokens() != 3){

        servidorGUI.log("threadRequisicao>>ENTRAR_SALA>>PArâmetros faltando");
                                            try{
                                                socket.close();
                                            }catch(Exception e){

        servidorGUI.log("threadRequisicao>>ERRO ao fechar socket:" + e);

                                            }


                                        }else{

        servidorGUI.log("threadRequisicao>>ENTRAR_SALA>>Numero de tokens ok");

                                            tokens.nextToken();
                                            String nomeJogador =
(String)tokens.nextToken();
                                            String sala = (String)tokens.nextToken();
                                            indice = nomeJogador.indexOf("=");
                                            if(indice == -1){

        servidorGUI.log("threadRequisicao>>ENTRAR_SALA>>PArâmetros faltando");
                                                try{
                                                    socket.close();
                                                }catch(Exception e){

        servidorGUI.log("threadRequisicao>>ERRO ao fechar socket:" + e);

                                                }
                                            }else{

                                                nomeJogador =
nomeJogador.substring( indice + 1, nomeJogador.length() );

        servidorGUI.log("threadRequisicao>>Token nomeJogador encontrado");

                                                indice = sala.indexOf("=");
                                                if(indice == -1){
```

```
servidorGUI.log("threadRequisicao>>ENTRAR_SALA>>PArâmetros faltando");
                                                try{
                                                        socket.close();
                                                }catch(Exception e){

servidorGUI.log("threadRequisicao>>ERRO ao fechar socket:" + e);

                                                }
                                        }else{

                                                sala = sala.substring( indice
+ 1, sala.length() );
                                        }

servidorGUI.log("threadRequisicao>>Token SALA encontrado");


servidorGUI.log("threadRequisicao>>Chamando rotina de entrada");
                                        synchronized(sala){
                                                String resultado =
entrarSala(sala, nomeJogador, reader, writer, socket);
                                                resultado = resultado.trim();

System.out.println(resultado);
                                                writer.println(resultado);
                                                if(resultado.equals("OK")){
                                                        Sala s =
procurarSala(sala);
                                                        StringBuffer buffer =
new StringBuffer("ENTRADA_JOGADOR");
                                                        for(int i = 0; i <
s.NUMERO_JOGADORES; i++)

if(s.getJogador(i) != null)

buffer.append("[Jogador: " + s.getJogador(i).getNome() + "]");
                                                                log("Enviando
lista de jogadores para o cliente: " + buffer.toString().trim());

s.enviarMsgBroadcast(buffer.toString().trim());

                                                }


if(resultado.startsWith("ERRO")){
```

```
servidorGUI.log("threadRequisicao>>Fechando socket: " + resultado);

                                                try{
                                                        socket.close();
                                                }catch(Exception e){

servidorGUI.log("threadRequisicao>>ERRO ao fechar socket:" + e);

                                                }

                                                        }
                                                }

                                        }

                                }
                        }else{
                                servidorGUI.log("threadRequisicao>>Formato de
requisição inválido. Requisição:" + requisicao);

                                }
                        }else{
                                try{
                                        socket.close();
                                }catch(Exception e){
                                        servidorGUI.log("threadRequisicao>>ERRO ao fechar
socket:" + e);

                                }
                        }
                        servidorGUI.log("threadRequisição>>Atendendo
requisição>>Requisição atendida");


                }
        }

        /**
                Quantidade de salas do servidor
        */
        public final static int NUMERO_SALAS = 10; //Quantidade de salas do servidor

        /**
                Salas
```

```
*/
private Sala sala[] = new Sala[NUMERO_SALAS];

/**
	Objeto utilizado para sincronização de mensagens
*/
private Object syncMsg = new Object(); //Para sincronizar as mensagens

/**
	IP do servidor
*/
private String ip;

/**
	Porta do servidor
*/
private int porta;


/**
	Timeout para início de jogo
*/

private double timeout;


/**
	Timeout para fim de jogo
*/

private double timeoutjogo;


/**
	Objeto de interface gráfica
*/
private ServidorGUI servidorGUI;

/**
	Objeto para conexão em rede
*/
private ServerSocketFactory factory = ServerSocketFactory.getDefault();
/**
	Objeto para conexão em rede
*/
private ServerSocket serverSocket;
```

```
	/**
		Porta default do servidor
	*/

	private int PORTA_SERVIDOR = 5656;

	/**
		Indica se o servidor está funcionando ou não
	*/
	private boolean SERVIDOR_RODANDO = true;

	/**
		Objeto para atualização da árvore de salas
	*/
	private DefaultMutableTreeNode ultimoNo[] = new
DefaultMutableTreeNode[NUMERO_SALAS];

	/**
		Variável utilizada na atualização da árvore de salas
	*/

	private int noAtual = 0;

	/**
		Construtor
	*/
	public Servidor(String ip, int porta, double timeout, double timeoutjogo){
		servidorGUI = new ServidorGUI(this);
		//addObserver(servidorGUI);

		this.ip = ip;
		this.porta = porta;
		this.timeout = timeout;
		this.timeoutjogo = timeoutjogo;

		try{
			servidorGUI.log("Criando ServerSocket...");
			serverSocket = factory.createServerSocket(porta);
			servidorGUI.log("ServerSocket criado: " + ip + ":" + porta);
			servidorGUI.log("Timeout para o início de um jogo: " + getTimeout()
+ " minutos.");
			servidorGUI.log("Timeout para o final de um jogo: " +
getTimeoutJogo() + " minutos.");

			new threadConexao(serverSocket);
			System.out.println(serverSocket);
```

```
                }catch(IOException e){

                        JOptionPane.showMessageDialog(null, "Já existe algum processo
utilizando a porta "+ PORTA_SERVIDOR + ". Feche este processo e tente
novamente!");
                        System.exit(1);
                }


                for(int i = 0; i < NUMERO_SALAS; i++){
                        sala[i] = new Sala("Sala " + i, this);
                        //sala[i].addObserver(servidorGUI);
                        //notifyObservers(sala[i].getId() + " criada");
                        servidorGUI.log(sala[i].toString());
                        servidorGUI.log("\n" + sala[i].getId() + " criada\n");
                }

                servidorGUI.atualizarArvore();
                servidorGUI.show();

        }

        /**
                Atualiza a árvore de salas
        */
        public void atualizarArvore(){
                servidorGUI.atualizarArvore();

        }

        /**
                Fornece uma representação do servidor num objeto string
        */
        public String toString(){
                StringBuffer buffer = new StringBuffer("Servidor: IP --> " + ip + " Porta: " +
porta + "\n");
                for(int i = 0; i < NUMERO_SALAS; i++){
                        buffer.append(">>" + sala[i] + "\n");

                }
                return buffer.toString().trim();

        }

        /**
                Interrompe o servidor
        */
```

```
	public void parar(){
		SERVIDOR_RODANDO = false;
		for(int i = 0; i < NUMERO_SALAS; i++){
			if(sala[i] != null) sala[i].resetar();
			sala[i] = null;
		}
		servidorGUI.dispose();

	}

	/**
		Retorna uma sala pelo seu índice
	*/
	public Sala getSala(int id){
		if(id >= NUMERO_SALAS)
			throw new IllegalArgumentException("Servidor>>getSala>>Índice
inválido");
		else
			return sala[id];

	}


	/**
		Retorna a lista de salas disponíveis
	*/
	public String retornarSalasDisponiveis(){
		StringBuffer buffer = new StringBuffer("");
		for(int i = 0; i < NUMERO_SALAS; i++){
			if(sala[i] != null && (sala[i].getJogadoresAtivos() <
Sala.NUMERO_JOGADORES) && !(sala[i].getJogando()))
				buffer.append(sala[i].getId() + "[" +
sala[i].getJogadoresAtivos() + "]" + ( i < (NUMERO_SALAS - 1) ? "-" : "") );

		}
		return buffer.toString().trim();


	}

	/**
		Procura uma sala
	*/
	private Sala procurarSala(String id){
		servidorGUI.log("Procurando sala: " + id);
		for(int i = 0; i < NUMERO_SALAS; i++){
```

```
                        if(sala[i] != null &&
sala[i].getId().trim().toUpperCase().equals(id.trim().toUpperCase())){
                                servidorGUI.log("Procurando sala: " + id + ". Sala
encontrada.");

                                return sala[i];

                        }
                }
                servidorGUI.log("Procurando sala: " + id + ". Sala não encontrada.");
                return null;
        }

        /**
                Tenta adicionar um jogador em uma sala
        */
        public String entrarSala(String id, String jogador, BufferedReader reader,
PrintWriter writer, Socket socket){
                servidorGUI.log("Jogador " + jogador + " quer entrar na sala " + id);
                String resultado = "";
                Sala s = procurarSala(id);
                if(s == null){ //Sala não encontrada
                        resultado = "ERRO: Id de sala inválido! Obtenha novamente a lista
de salas, provavelmente o nome de alguma sala do servidor foi modificado.";
                        servidorGUI.log("Jogador>>" + jogador + ">>ERRO: Id de sala(" + id
+ ") inválido! Obtenha novamente a lista de salas, provavelmente o nome de alguma
sala do servidor foi modificado.");
                }else{
                        try{
                                boolean primeiro = s.adicionarJogador(new
JogadorRede(jogador, reader, writer, s, socket));
                                if(primeiro){
                                  resultado = "OKPRIM"; //Jogador foi o primeiro a entrar
nesta sala
                                  servidorGUI.log("Jogador>>" + jogador + " entrou na sala "
+ id + " e é o primeiro usuário desta sala.");

                                }else{
                                  resultado = "OK"; //Já exitiam jogadores nesta sala
                                  servidorGUI.log("Jogador>>" + jogador + " entrou na sala "
+ id + ".");
                                  /*synchronized(s){
                                        if(s.getJogadoresAtivos() ==
Sala.NUMERO_JOGADORES)
                                                s.setJogando(true);
                                  }*/
                                }
```

```
                        servidorGUI.atualizarArvore();
                    }catch(NumeroJogadoresExcedidoException e2){
                        resultado = "ERRORET: A sala escolhida está cheia.
Selecione outra sala e tente novamente.";
                        servidorGUI.log("Jogador>>" + jogador + " não conseguiu
entrar na sala " + id + ", pois a mesma está cheia");

                    }catch(JogandoException e3){
                        resultado = "ERRORET: O jogo já foi iniciado na sala
escolhida. Selecione outra sala e tente novamente.";
                        servidorGUI.log("Jogador>>" + jogador + " não conseguiu
entrar na sala " + id + ", pois o jogo já está em andamento nesta sala.");

                    }catch(JogadorDuplicadoException e1){
                        resultado = "ERRO: Já existe um jogador na sala com o
apelido que você escolheu. Troque o apelido e tente novamente.";
                        servidorGUI.log("Jogador>>" + jogador + " não conseguiu
entrar na sala " + id + ", pois já existe um jogador com este apelido na sala.");

                    }catch(IOException e4){
                        resultado = "ERRO: Não é possível criar a conexão.";
                        servidorGUI.log("Jogador>>" + jogador + " não conseguiu
entrar na sala " + id + ", pois não foi possível criar uma conexão.\n" + e4);


                    }

                }

                return resultado;

        }

        /**
                Retorna o nó principal da árvore de salas
        */
        public DefaultMutableTreeNode retornarNo(JTree arvore){
                DefaultMutableTreeNode raiz = new DefaultMutableTreeNode("Servidor
Mau-Mau");
                DefaultMutableTreeNode salas = new DefaultMutableTreeNode("Salas");
                /*raiz.add(new DefaultMutableTreeNode("Log"));
                raiz.add(new DefaultMutableTreeNode("Configurações"));*/


                for(int i = 0; i < NUMERO_SALAS; i++){
                        if(sala[i] != null){
                                synchronized(sala[i]){
```

```
                                        DefaultMutableTreeNode noSala =
sala[i].retornarNo(arvore);
                                        salas.add(noSala);
                                }
                        }
                }

                raiz.add(salas);



                return raiz;
        }

        /**
                Adiciona uma entrada ao log
        */
        public void log(String msg){
                servidorGUI.log(msg);
        }

        /**
                Inicializa a execução do servidor
        */

        public static void main(String[] args){
                int PORTA_SERVIDOR = 5656;
                double TIMEOUT = 15, TIMEOUTJOGO = 60;

                try{
                        Preferences.userRoot().node("servidormaumau").removeNode();

                }catch(Exception e1){}

                try{
                        Preferences.importPreferences(new FileInputStream("config.xml"));
                        Preferences p = Preferences.userRoot();
                        System.out.println("**********Arquivo de configuração
encontrado**********");
                        System.out.println("COnfigurações:");
                        PORTA_SERVIDOR = p.node("servidormaumau").getInt("PORTA",
5656);
                        TIMEOUT = p.node("servidormaumau").getDouble("TIMEOUT",
15.0);
                        TIMEOUTJOGO =
p.node("servidormaumau").getDouble("TIMEOUTJOGO", 60.0);
                        System.out.println("PORTA_SERVIDOR: " + PORTA_SERVIDOR);
```

```
                        System.out.println("TIMEOUT: " + TIMEOUT);
                        System.out.println("TIMEOUT_JOGO: " + TIMEOUTJOGO);

        System.out.println("*****************************************************");
                        if(PORTA_SERVIDOR <= 0) PORTA_SERVIDOR = 5656;
                        if(TIMEOUT <= 0) TIMEOUT = 15;
                        if(TIMEOUTJOGO <= 0) TIMEOUTJOGO = 60;


                }catch(Exception e){
                        System.out.println("Erro ao carregar arquivo de
configuração(config.xml)>> " + e + ">>Utilizando opções default.");


                }
                Servidor s = new Servidor("socket://localhost", PORTA_SERVIDOR,
TIMEOUT, TIMEOUTJOGO);

        }

        /**
                Método utilizado para atualização da árvore
        */
        public synchronized void addUltimoNoSala(DefaultMutableTreeNode no){
                if(noAtual < NUMERO_SALAS)
                        ultimoNo[noAtual++] = no;

        }

        /**
                Retorna o timeout para o início do jogo
        */
        public double getTimeout(){
                return timeout;
        }

        /**
                Retorna o timeout para o término do jogo
        */
        public double getTimeoutJogo(){
                return timeoutjogo;
        }



}
```