

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE

André Maranhão de Miranda

**DESENVOLVIMENTO DE UM PROTÓTIPO DE UM JOGO DA VELHA PARA
CELULAR UTILIZANDO JME E BLUETOOTH**

NATAL/RN

2006

ANDRÉ MARANHÃO DE MIRANDA

**DESENVOLVIMENTO DE UM PROTÓTIPO DE UM JOGO DA VELHA PARA
CELULAR UTILIZANDO JME E BLUETOOTH**

Monografia apresentada ao curso de Bacharelado em Sistemas de Informação da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador:

Gleydson de Azevedo Ferreira Lima

NATAL/RN

2006



004.43: 621.39

M64d

2006

Ex: 01

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Miranda, André Maranhão de

Desenvolvimento de um protótipo de um jogo da velha para celular utilizando JME e bluetooth / André Maranhão de Miranda. – Natal: FARN, 2006.
90 f.

Orientador: Gleydson de Azevedo Ferreira Lima (M.Sc.)

Monografia (Bacharelado em Sistemas de Informação) – Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte.

1. Tecnologia Móvel – Monografia. 2. Desenvolvimento de Jogos Eletrônicos – Monografia 3. Java Micro Edition – Monografia 4. Bluetooth – Monografia I. Título.

RN/FARN/BC

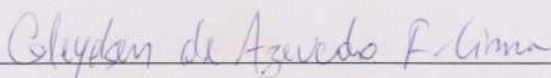
CDU 621.39:004.43

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE

**DESENVOLVIMENTO DE UM PROTÓTIPO DE UM JOGO DA VELHA PARA
CELULAR UTILIZANDO JME E BLUETOOTH**

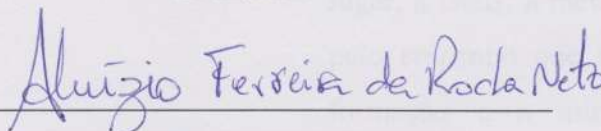
André Maranhão de Miranda

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Faculdade
Natalense para o Desenvolvimento do Rio Grande do Norte e aprovado em 16 de dezembro
de 2006 pela banca examinadora composta pelos seguintes membros:



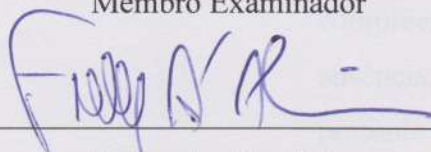
Gleydson de Azevedo Ferreira Lima

Orientador



Aluizio Ferreira da Rocha Neto

Membro Examinador



Fellipe Araújo Aleixo

Membro Examinador

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus, pelo caminho da vida, pela força de superar as dificuldades e por proporcionar-me este grande momento.

A minha família, pelo apoio que sempre me deu durante minha vida e que agora me dá a força para seguir em frente.

A meu pai e minha mãe, pelo exemplo que sempre me deu, pela força de superar as dificuldades e por estar sempre ao meu lado, por tudo o que me ensinaram e por sempre estarem comigo.

Especialmente, a minha amada noiva Daniele, por sua paciência e compreensão nos momentos de ausência, pelo incentivo e por estar sempre ao meu lado, por tudo o que me ensinaram e por sempre estarem comigo.

Aos meus amigos, pelo apoio que sempre me deu e por estarem sempre ao meu lado, por tudo o que me ensinaram e por sempre estarem comigo.

Aos meus professores, pelo conhecimento que me transmitiram e por estarem sempre ao meu lado, por tudo o que me ensinaram e por sempre estarem comigo.

A todos os que me ajudaram e apoiaram neste trabalho, por tudo o que me ensinaram e por sempre estarem comigo.

A todos os que me ajudaram e apoiaram neste trabalho, por tudo o que me ensinaram e por sempre estarem comigo.

A todos os que me ajudaram e apoiaram neste trabalho, por tudo o que me ensinaram e por sempre estarem comigo.

Dedico este trabalho, em primeiro lugar, a Deus, a meu pai e minha mãe, pelo empenho que tiveram em minha formação e a minha amada noiva Daniele, por sua paciência e compreensão, nos momentos de ausência, pelo incentivo e por estar presente nos bons e maus momentos sempre ao meu lado.

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus, pela minha vida, pela força de superar as dificuldades e por proporcionar-me essa grande conquista.

A minha família, pelo apoio que sempre me deu durante minha vida e, em especial, nesta fase.

A meu pai e minha mãe, pelo empenho que tiveram em minha formação como pessoa, pelo esforço que fizeram para contribuir para a minha conclusão, por todo apoio que me deram e pelo amor que têm por mim.

Especialmente, a minha amada noiva Daniele, por sua paciência e compreensão, nos momentos de ausência, pelo incentivo e por estar presente nos bons e maus momentos sempre ao meu lado.

Ao meu irmão Jack, por nossa amizade e sua torcida pelo meu crescimento pessoal e intelectual.

Aos meus amigos Bruno, Elza e José de Arimatea, companheiros de estudos e trabalhos, pela amizade, força e auxílio durante nosso caminho, pelas horas de estudos muitas vezes na casa de Bruno.

A minha amiga Luana, que foi fundamental neste trabalho, por nossa amizade e pelo apoio prestado no desenvolvimento desta pesquisa.

Aos professores pelos conhecimentos que nos passaram, ao professor Gleydson de Azevedo Ferreira Lima, meu orientador, pela paciência durante o trabalho, pela oportunidade e confiança depositadas, pelo incentivo e conhecimentos repassados.

Às pessoas que contribuíram indiretamente e diretamente neste trabalho.

RESUMO

Como todas as áreas da Tecnologia da Informação, a Tecnologia Móvel vem sofrendo um grande avanço ao longo dos anos, com destaque para os telefones celulares, que atualmente, em algumas empresas, ocupam a maior parte do faturamento. Com este crescimento surgiu a necessidade de aplicações para os mesmos, destacando-se os toques para celulares, serviço de envio de mensagens e vídeos, banco via celular, localização de aparelhos e entretenimento via celular, dentre outras. Diante do contexto, a presente pesquisa tem como objetivo a produção de uma aplicação de entretenimento, que consiste em um jogo da velha para dispositivos móveis (celulares ou PDAs), realizando um estudo sobre a Plataforma Java para dispositivos móveis no desenvolvimento de aplicações para celular, utilizando a tecnologia Java Micro Edition (JME) e bluetooth. O jogo será utilizado por dois participantes, em dispositivos distintos, conectados por uma WPAN (Wireless Private Area Network). Para atender estes requisitos e realizar as comprovações das teorias abordadas foi criado um protótipo do jogo, que foi testado em celulares de fabricantes distintos e em emuladores. A aplicação é desenvolvida para poder realizar duas funções, tanto de servidor quanto de cliente da conexão, sendo abstraída para o usuário em qual modo ela opera. Cada dispositivo monta o seu tabuleiro do jogo, controlando a marcação das jogadas, transmitindo para o outro jogador as ações realizadas, através de um mecanismo de passagem de token. Com JME e o bluetooth o desenvolvedor consegue obter as vantagens da plataforma Java unida à tecnologia móvel. Obtendo um desenvolvimento mais prático com a utilização de uma tecnologia mais fácil e robusta, havendo possibilidade de portabilidade para diversos dispositivos móveis. Conseguindo também a interligação dos dispositivos de uma forma segura e sem necessidade de fios. Porém durante o desenvolvimento problemas surgiram, como o comprometimento da portabilidade, devido a API bluetooth ser um pacote opcional não sendo encontrado em alguns aparelhos que possuem bluetooth, gerando um problema para o desenvolvedor. A maneira de tratamento dos eventos realizados no teclado, bem como o tamanho da tela, varia de dispositivo para dispositivo, exigindo do desenvolvedor uma grande atenção e dificultando ainda mais a portabilidade. De fabricante para fabricante as API's disponíveis mudam tornando necessário a utilização de API's básicas para obtenção de uma boa portabilidade. Apesar das dificuldades encontradas, as tecnologias demonstraram-se como uma grande ferramenta para desenvolvimento e de comunicação.

Palavras-chaves: Plataforma Java; TM - Tecnologia Móvel; JME - *Java Micro Edition*; *bluetooth*; WPAN; jogo da velha; canvas; Java 2D; MIDP; CLDC.

ABSTRACT

As all the areas of the Technology of the Information, the Mobile Technology comes suffering a great advance throughout the years, with prominence for the cellular telephones, that currently, in some companies, occupy most of the invoicing. With this growth the necessity appeared of applications for the same ones, being distinguished the touches for cellular, service of sending of messages and videos, bank saw cellular, localization of devices and entertainment saw cellular, amongst others. Ahead of the context, the present research has as objective to the production of an entertainment application, that consists of a game of the old one for mobile devices (cellular or PDAs), carrying through a study on the Java Platform for mobile devices in the development of applications for cellular, using the technology Java Micro Edition (JME) and bluetooth. The game will be used by two participants, in distinct devices, connected for a WPAN (Wireless Private Network Area). To take care of these requirements and to carry through the evidences of the boarded theories an archetype of the game was created, that was tested in cellular of distinct manufacturers and emulators. The application is developed to be able to carry through two functions, as much of server how much of customer of the connection, being abstracted for the user in which way it operates. Each device mounts its tray of the game, controlling the marking of the plays, transmitting for the other player the carried through actions, through a mechanism of ticket of token. With JME and bluetooth the desenvolvedor obtains to get the advantages of the Java platform joined to the mobile technology. Getting a development more I practise with the use of a more easy and robust technology, having possibility of portabilidade for diverse mobile devices. Also obtaining the interconnection of the devices of one it forms insurance and without wire necessity. However during the development problems they had appeared, as the comprometimento of the portabilidade, which had the API bluetooth to be an optional package not being found in some devices that possess bluetooth, generating a problem for the desenvolvedor. The treatment way them events carried through in the keyboard, as well as the size of the screen, still more varies of device for device, demanding of the desenvolvedor a great attention and making it difficult the portabilidade. Of manufacturer for manufacturer the available API 's move becoming necessary the use of basic API 's for attainment of a good portabilidade. Although the found difficulties, the technologies had been demonstrated as a great tool for development and of communication.

Word-keys: Java platform; TM - Mobile technology; JME - Java Edition Micron; bluetooth; WPAN; game of the old one; canvas; Java 2D; MIDP; CLDC.

LISTA DE SIGLAS

TI - Tecnologia da Informação	14
TM - Tecnologia Móvel	15
JME - <i>Java Micro Edition</i>	16
BREW - <i>Binary Runtime Environment for Wireless</i> /Ambiente Binário de Execução para sem-fio	17
SDK - Kit para Desenvolvimento de Software	18
BDS - <i>Delivery System</i> / Sistema de Distribuição BREW	19
PDA - <i>Personal Data Assistants</i> / assistentes digitais pessoais	19
J2SE - <i>Java 2 Standart Edition</i>	20
J2EE - <i>Java 2 Enterprise Edition</i>	21
APIs - <i>Applications Programming Interfaces</i> / interfaces de programas aplicativos	23
JVM - <i>Java Virtual Machine</i> / máquina virtual Java	23
CDC - <i>Connected Device Configuration</i> / Configuração de Dispositivo Conectado	24
CLDC - <i>Connected Limited Device Configuration</i> / Configuração de Dispositivo Conectado Limitado	24
MIDP - <i>Mobile Information Device Profile</i> / perfil de dispositivo de informação móvel	25
IMP - <i>Information Module Profile</i>	26
FP - <i>Foundation Profile</i>	26
PBP - <i>Personal Basis Profile</i>	26
PP - <i>Personal Profile</i>	27
RPG - <i>Role Playing Game</i>	27
IEEE - <i>Institute of Electrical and Electronics Engineers</i>	28
SIG - <i>Special Interest Group</i>	28
WPAN - <i>Wireless Private Área Network</i>	29
L2CAP - <i>Logical Link Control and Adaptation Protocol</i>	30
RFCOMM - <i>Radio Frequency Communication</i>	31
UUID - <i>Universally Unique Identifiers</i>	34
GPRS - <i>General Packet Radio Service</i>	34

3.4.1.2. Tipos de Jogos em Pontos de Objetos

3.4.1.3. Tipos de Jogos em Espaço de Trabalho de Usuários

3.4.2. Anotações de Desenvolvimento

SUMÁRIO

1. INTRODUÇÃO	14
1.1. PROBLEMA DE PESQUISA	15
1.2. OBJETIVOS	16
1.2.1. Objetivo Geral	16
1.2.2. Objetivos Específicos	17
1.3. JUSTIFICATIVA	17
2. REFERENCIAL TEÓRICO	19
2.1. TECNOLOGIA MÓVEL	19
2.2. BLUETOOTH	20
2.2.1. Aplicações do Bluetooth	22
2.2.2. Características de Operação	23
2.2.2.1 Estabelecendo Conexão	23
2.2.2.2 Solicitação e Paginação	25
2.2.2.3 Redes de Difusão	26
2.2.3 Dispositivos	26
2.3. PLATAFORMAS DE DESENVOLVIMENTO	29
2.3.1. Symbian	29
2.3.2. Brew	29
2.3.3. Java	29
2.3.4. Jme	32
2.3.4.1. Configuração	33
2.3.4.2. Perfis	32
2.3.4.3. Midlet	37
2.3.4.4. Desenho 2d (Java 2d)	40
2.3.4.5. Api Bluetooth (Jsr-82)	46
2.4. DESENVOLVIMENTO DE JOGOS	53
2.4.1. Tipos de Jogos	54
2.4.1.1. Tipos de Jogos em Função da Interface	54
2.4.1.2. Tipos de Jogos em Função do Objetivo	54
2.4.1.3. Tipos de Jogos em Função do Número de Usuários	55
2.4.2. Antes do Desenvolvimento	55

2.4.3. Desenvolvimento	56
2.5. JOGO DA VELHA	56
3. METODOLOGIA	58
3.1. TIPOS DE PESQUISA	58
3.1.2. Quanto aos Objetivos	58
3.1.3. Quanto aos Procedimentos	58
3.1.4. Quanto ao Objeto	59
4. RESULTADOS	60
4.1. LÓGICA DO JOGO	63
4.1.1. Jogo	63
4.1.2. Classe Partida	64
4.1.3. Classe Pontomatriz	65
4.1.4. Classe Ponto	66
4.2. MIDLET DA APLICAÇÃO	66
4.2.1. Classe Velhamidlet	66
4.2.1.1 Ações do Ciclo de Vida	66
4.3. CLASSE JOGODAVELHACANVAS	68
4.3.1. Utilização de Java 2d	69
4.3.2. Tratamento de Eventos	73
4.4. COMUNICAÇÃO	76
4.4.1. Servidor sendo Encontrado	78
4.4.2. Servidor não Encontrado	82
4.5. SCREENSHORTS DA APLICAÇÃO	83
5. CONCLUSÕES	88
REFERÊNCIAS	89

FIGURAS

Figura 1 – Aparelhos Moveis (celulares, PDAs, <i>Palmtop</i>) (PALUDO, 2003)	20
Figura 2 – Tecnologia <i>bluetooth</i> sendo usada para conectar aparelhos eletrônicos domésticos (LIMA; MEDEIROS, 2005)	22
Figura 3 – Diagrama de uma piconet (LIMA e MEDEIROS, 2005)	24
Figura 4 – Diagrama de estados dos dispositivos <i>bluetooth</i> (LIMA e MEDEIROS, 2005)	25
Figura 5 – Rede de difusão (<i>Scatternet</i>) (LIMA e MEDEIROS, 2005)	26
Figura 6 – cartão PCMCIA que torna um notebook ou <i>palmtop</i> compatíveis com o padrão (LIMA e MEDEIROS, 2005)	27
Figura 7 – Primeiro <i>headset Bluetooth</i> do mundo, juntamente com sua estação base (LIMA e MEDEIROS, 2005)	27
Figura 8 – Interface USB, que permite conexão <i>Bluetooth</i> (LIMA e MEDEIROS, 2005)	28
Figura 9 – Relógio e caneta compatíveis com <i>Bluetooth</i> (LIMA e MEDEIROS, 2005).	28
Figura 10 – Telefone/PDA <i>Communicator</i> da Ericsson, compatível com <i>Bluetooth</i> (LIMA e MEDEIROS, 2005)	28
Figura 11 – Plataformas Java (J2EE, J2SE, JME, <i>Java Card</i>)	31
Figura 12 – Plataformas Java (J2EE, J2SE, JME, <i>Java Card</i>) (JODE et al, 2004)	32
Figura 13 – Métodos Mínimos de uma <i>MIDlet</i> (MUCHOW, 2002)	38
Figura 14 – Métodos de uma Classe <i>Canvas</i>	41
Figura 15 – Coordenadas em uma <i>Canvas</i>	42
Figura 16 – Exemplo de código de um desenho na tela	44
Figura 17 – Execução do código da figura 13	44
Figura 18 – Exemplo de utilização da classe <i>LocalDevice</i> (PÓVOA, 2005)	47
Figura 19 – Exemplo de disponibilização de um serviço	49
Figura 20 – Estrutura dos métodos <i>deviceDiscovered()</i> e <i>inquiryCompleted()</i>	50
Figura 21 – Estrutura do método <i>serviceSearchCompleted()</i> e utilização do <i>searchServices()</i>	51
Figura 22 – Métodos de conexão	52
Figura 23 – Tabuleiro do jogo da velha	56
Figura 24 – Exemplo de configurações válidas para vitória	57
Figura 25 – Camadas da aplicação	61

Figura 26 – Diagrama da aplicação	62
Figura 27 – Diagrama de estado da aplicação	63
Figura 28 – Classe Jogo	64
Figura 29 – Classe Partida	65
Figura 30 – Classe PontoMatriz	65
Figura 31 – Classe Ponto	66
Figura 32 – Código da inicialização da <i>MIDlet</i>	67
Figura 33 – Método <i>destroyApp()</i>	67
Figura 34 – Iniciando a busca por dispositivos	68
Figura 35 – Exemplo da classe <i>JogoDaVelhaCanvas</i>	68
Figura 36 – Código que desenha o tabuleiro na <i>JogoDaVelhaCanvas</i>	69
Figura 37 – Tabuleiro do jogo e o marcador de posição	70
Figura 38 – Código que desenha as marcações no tabuleiro na <i>JogoDaVelhaCanvas</i>	71
Figura 39 – Marcações dos jogadores	71
Figura 40 – Desenhando a marcação da vitória	72
Figura 41 – Marcação da vitória	72
Figura 42 – Código da Marcação da vitória	72
Figura 43 – Desenho das informações	73
Figura 44 – Tratamento dos eventos realizados nas teclas	74
Figura 45 – Tratamento dos comandos	75
Figura 46 – Tratamento de eventos com som	75
Figura 47 – Início da busca	77
Figura 48 – Método para adicionar os dispositivos encontrados	78
Figura 49 – Código do tratamento de erro da busca e de cancelamento	78
Figura 50 – Código do tratamento de busca completa	79
Figura 51 – Código da verificação do serviço	80
Figura 52 – Término da busca	80
Figura 53 – Início da conexão cliente	81
Figura 54 – Método executar da classe <i>BluetoothConexao</i>	81
Figura 55 – Disponibilização do serviço	82
Figura 56 – Telas iniciais jogador 1 e Tela inicial do jogador 2	83
Figura 57 – Tela do menu do jogador 1 e do Jogador 2	83
Figura 58 – Busca por dispositivos do Jogador 1	84

Figura 59 – Criação do Servidor no Jogador 1	84
Figura 60 – Jogador 1 Aguardando um cliente para conexão e Servidor encontrado pelo Jogador 2	85
Figura 61 – Criando Cliente no Jogador 2	85
Figura 62 – Início do jogo para o Jogador 1 e Jogador 2	86
Figura 63 – Marcação das jogadas do Jogador 1 e do Jogador 2	86
Figura 64 - Marcação da vitória do Jogador 1 no display dos Jogadores	87
Figura 65 – Marcação do empate no display dos Jogadores	87

Tabela 1 – Os perfis do <i>bluetooth</i> (LIMA; MEDEIROS, 2005)	23
Tabela 2 – API <i>MIDlet</i> (MUCHOW, 2002)	39
Tabela 3 – API <i>Display</i> (MUCHOW, 2002)	39
Tabela 4 – API <i>Displayable</i> (MUCHOW, 2002)	39
Tabela 5 – API <i>Canvas</i> (MUCHOW, 2002)	43
Tabela 6 – Alguns métodos da API <i>Graphics</i> (MUCHOW, 2002)	45
Tabela 7 – Métodos da API <i>LocalDevice</i> (PÓVOA, 2005)	47
Tabela 8 – Métodos Construtores da API <i>UUID</i> (PÓVOA, 2005)	48
Tabela 9 – Métodos do Java 2D utilizados na <i>JogoDaVelhaCanvas</i>	69

1. INTRODUÇÃO

Como todas as áreas da Tecnologia da Informação (TI), a Tecnologia Móvel (TM) vem sofrendo um grande avanço ao longo dos anos, destacando-se os telefones celulares, os quais atualmente ocupam uma grande fatia do mercado de telefonia, sendo em algumas empresas a linha mais importante, ocupando a maior parte do faturamento.

Segundo a Nokia (2006), O volume combinado de dispositivos móveis do grupo de negócios *Mobile Phones* (Telefones Celulares), *Multimedia* (Multimídia) e *Enterprise Solutions* (Soluções Empresariais) no quarto trimestre de 2005 foi um recorde de 84 milhões de unidades. Obteve um crescimento de 26% e de 27% comparado ao mesmo período no ano passado. O estímulo gerado pelo crescimento do mercado, incentivou as empresas a fornecerem equipamentos mais avançados, com poder de processamento cada vez maior, obtendo múltiplas funções e com tecnologias mais novas e avançadas.

Com a sofisticação dos aparelhos e a infinidade de modelos existentes, surge a necessidade de aplicações mais complexas. Atualmente, podem-se destacar os toques para celulares (*ringtones*), serviços de papo on-line (*chat*), serviço de envio de mensagens e vídeos, *internet banking* (banco na internet) via celular, localização de aparelhos e entretenimento via celular, dentre outras.

De acordo com Ferrari (2006), o Banco do Brasil apostou no *mobile banking* (Banco Móvel) como uma nova estratégia de negócios, tendo em vista os mais de 88 milhões de aparelhos celulares e uma rede integrada com as principais operadoras de telefonia móvel, cobrindo 99% do território nacional. Interessado em reduzir custos e fornecer uma nova maneira de atendimento, o banco conseguiu fazer o número de transações por mês subir de 500 mil para 950 mil na primeira semana de divulgação do serviço na mídia.

O desenvolvimento de entretenimento vem aumentando devido a todo esse avanço tecnológico a que os dispositivos móveis estão sendo submetidos. A procura por este produto vem crescendo e forçando as indústrias de telefonia celular a investirem no desenvolvimento dos mesmos, pressionando as grandes Organizações de entretenimento a migrarem seus produtos para celular.

Os jogos de computadores pessoais e consoles estão migrando para o celular e em breve a convergência entre estas plataformas será possível. Hoje os jogos para celular estão cada vez mais complexos e num futuro próximo direcionarão a fabricação dos aparelhos, como fazem com os computadores pessoais (FALCO, 2005).

Estimulado pela grande adoção dos entretenimentos para celular e por causa da limitação dos dispositivos (de processamento e interfaces gráficas), despontou à necessidade no mercado: de profissionais de TI, que desenvolvam em plataformas que sejam compatíveis e que atendam às exigências dos aparelhos, e de entretenimento que estimulem o interesse dos usuários e possibilitem um consumo satisfatório para as empresas.

Com o crescimento da indústria de celulares, que em 2004 faturou US\$ 200 bilhões, os estudantes de ciência da computação e de desenho industrial foram atraídos para atuarem na criação de aplicativos para dispositivos móveis. Cada vez mais, é valorizado o engenheiro de *software*, tendo em vista a oferta de vagas hoje ser maior do que o número de profissionais qualificados para preenchê-las. Prova disso é que os salários para recém-formados variam entre R\$ 1,5 mil e R\$ 5 mil (GIARDINO, 2005).

Diante do contexto, a presente pesquisa tem como finalidade atender a essas necessidades desenvolvendo um jogo de interação. Aborda a tecnologia de desenvolvimento de aplicações para dispositivos móveis *Java Micro Edition* (JME), uma das tecnologias mais utilizadas para desenvolvimento em dispositivos móveis e a tecnologia de comunicação entre dispositivos sem fio (*wireless bluetooth*).

1.1 PROBLEMA DE PESQUISA

A utilização de dispositivos móveis vem crescendo demasiadamente no mercado mundial, principalmente pelo uso de celulares, que atualmente trazem um conjunto de outras tecnologias agregadas como, por exemplo: gravações de vídeo e áudio, fotos digitais, MP3 *player*, infravermelho, *bluetooth* e outras tecnologias. Cada vez mais, aumentam o seu poder de processamento e a quantidade de serviços agregados.

Balieiro (2005, pg. 82) afirma que nenhum outro dispositivo sem fio ganhou tantos adeptos no Brasil quanto o celular. Hoje, 70 milhões de pessoas possuem seu telefone móvel. Os “tijolões” saíram de cena. Os celulares ficaram menores, ganharam MP3 *players* e câmeras digitais, passaram a conversar com o computador pessoal.

Diante deste crescimento o mercado de TI foi estimulado, fazendo com que surgissem diversas tecnologias, bem como outras já existentes passassem a ser utilizadas nos dispositivos, com destaque para os entretenimentos. Despertando, assim, o interesse de grandes empresas de TI para o mercado de tecnologia móvel.

A Qualcomm (empresa de desenvolvimento de Tecnologia Móvel) no Brasil precisa defender a tecnologia desenvolvida pela companhia para os celulares, um mercado

que agora passa por uma inovação, na qual os aparelhos estão se transformando em verdadeiras centrais de entretenimento móveis, capazes de receber, por exemplo, sinais de emissoras de TV, ou de funcionar como rádios digitais móveis (Fernades, 2006).

Tendo em vista as limitações impostas pelos dispositivos, são poucas as plataformas de desenvolvimento encontradas no mercado, como por exemplo: *Symbian C++*, *Brew* e *JME*. Dentre elas, podemos destacar a *JME*, uma poderosa ferramenta de desenvolvimento para dispositivos móveis, que é compatível com a maioria dos modelos de celulares encontrados no mercado.

Diferente do que ocorre em outros mercados, em Natal são poucas, ou quase nenhuma, as empresas voltadas para este tipo de mercado, dificultando o incentivo e o surgimento de novos profissionais nesta área, bem como a proliferação do *JME*. Porém, com o crescimento que vem apresentando, logo será necessário encontrar pessoas para atender à demanda em Natal.

O mercado de Recife tem obtido um grande crescimento, causado pelo Porto Digital, o qual atraiu grandes empresas como Motorola, Nokia, LG e Samsung. A Motorola foi a primeira a se estabelecer na região. O projeto mais bem-sucedido desenvolvido em Recife é o Vivo Agenda, que permite a assinantes da operadora de telefonia transferir a agenda dos celulares Motorola para novos aparelhos automaticamente. A Motorola inaugurou no Porto Digital seu centro mundial de testes de software, batizado de *Brazil Test Center* (GOMES, 2005).

Diante desse fato, a necessidade de se obter novas aplicações e novos profissionais no desenvolvimento para tecnologias móveis caminha para o seguinte questionamento no campo desta pesquisa: **Como desenvolver uma Aplicação Multimídia Interativa para dispositivos limitados e conectados?**

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Esta Monografia refere-se a um trabalho de pesquisa, que tem como objetivo geral desenvolver uma aplicação multimídia interativa para dispositivos limitados e conectados, utilizando *JME* e *bluetooth*, possibilitando a divulgação da tecnologia e servindo de apoio para pessoas que não os conheçam.

1.2.2 Objetivos Específicos

A Monografia está dividida em quatro objetivos específicos, que serão discriminados abaixo:

- Estudar a plataforma JME para desenvolvimento de dispositivos móveis;
- Analisar o desenvolvimento de jogos para celular;
- Estudar a tecnologia *bluetooth*;
- Implementar uma aplicação para dispositivos móveis com comunicação *bluetooth*;

1.3 JUSTIFICATIVA

Com o advento da tecnologia móvel os usuários passaram a ter acesso a diversas tecnologias na palma da mão, em qualquer lugar e a qualquer hora. Atualmente, é possível fotografar, filmar, ouvir música, acessar a internet, consultar saldos bancários, enviar e-mail e mensagens e se entreter, tudo através de um celular.

Cada vez mais a utilização de dispositivos móveis, bem como da tecnologia móvel, vem apresentando um grande crescimento, dentro de empresas e na vida das pessoas, tornando-se um atrativo para as organizações de desenvolvimento e para os desenvolvedores.

Com intuito de atender à necessidade deste mercado a *Sun Microsystems*, desenvolvedora da plataforma Java, criou uma edição Java para dispositivos móveis, a JME, uma plataforma atualmente integrada à maioria dos celulares, que tornou-se uma das plataformas mais utilizadas por desenvolvedores.

Com a utilização dos dispositivos móveis surgiu a necessidade de sua interligação. Sendo desenvolvido, por intermédio de um consórcio de empresas de tecnologia, um meio de comunicação sem fio, chamado *bluetooth*, que vem ganhando espaço no mercado e expandindo suas fronteiras para outros dispositivos, como relógios, impressoras, televisores, sons, *headset*, canetas dentre outros.

Uma das tecnologias para dispositivos móveis que mais cresce no mercado, principalmente no Brasil, e que está ganhando as atenções de grandes empresas nacionais e multinacionais, é a de entretenimento para celulares, com destaque para jogos, monousuário ou multi-usuários.

Contudo, a idéia de desenvolver um jogo para celular, que una todas estas tecnologias, vem a atender um nicho de mercado existente e a servir como base para pesquisas futuras.

A presente monografia visa o desenvolvimento de um protótipo de jogo da velha, para dispositivos móveis, focando o celular, utilizando a plataforma de desenvolvimento JME, com suporte à *bluetooth*. Possibilitando desta maneira pessoas interagirem com os dispositivos ou com outras pessoas, disponibilizando desta maneira uma forma de entretenimento, fazendo uso de tecnologias de destaque atualmente.

2. REFERENCIAL TEÓRICO

2.1. TECNOLOGIA MÓVEL

Tecnologia móvel é toda tecnologia que permite seu uso durante a movimentação do usuário. Por ter alcançado o dia a dia das pessoas e estar modificando suas rotinas, ela já faz parte da suas vidas podendo ser considerada uma revolução. Hoje em dia pessoas já não conseguem viver sem celular, através desta ferramenta podem ser encontradas em qualquer lugar a qualquer hora, alguns exemplos de aparelhos são demonstrados na figura 1. Toda essa mudança só foi possível através da evolução da tecnologia móvel (WIKIPÉDIA, 2006).

Para Wikipédia (2006), mobilidade iniciou como uma facilidade, mas hoje em dia tornou-se uma necessidade. Mas, por que a evolução deste segmento foi tão rápida? Simples. Porque este tipo de tecnologia permite o acesso a dados e informações em qualquer momento e em qualquer lugar. Isto se torna um poderoso atrativo.

O uso da Tecnologia móvel vem crescendo tanto que já está disponível em várias áreas do mercado, com seu crescimento surgiu um novo tipo de comércio o “*mCommerce*” (comércio móvel) e possibilitou a utilização de novos aplicativos para os dispositivos móveis (COMÉRCIO, 2006).

Os serviços móveis de varejo, como a compra de produtos pelo celular, auxiliarão o mercado de dispositivos móveis a atingir um valor superior a US\$ 63 bilhões em receitas até 2010. A conclusão é de uma pesquisa realizada pela *Juniper Research*, consultoria inglesa especializada no setor de telecomunicações (COMÉRCIO, 2006).

A *Royal Philips Electronics* anunciou hoje a sua primeira solução “TV no celular” para o mercado norte-americano, apenas três meses após o lançamento da solução européia. A Philips é pioneira nesta nova tendência em entretenimento multimídia móvel. Estima-se que o crescimento será de 10 milhões de clientes sem fio nos Estados Unidos até 2010 (*ABI Research*) (TECNOLOGIA, 2006).



Figura 1 – Aparelhos Moveis (celulares, PDAs, *Palmtop*) (PALUDO, 2003)

2.2. BLUETOOTH

Com o crescimento da tecnologia móvel surgiu a necessidade de comunicação entre: dispositivos móveis\dispositivos móveis e dispositivos móveis\demais dispositivos. Com a finalidade de solucionar este problema e livrar os dispositivos de cabos, as empresas L. M. Ericsson, IBM, Intel, Nokia e Toshiba, em 1998, formaram um consórcio denominado SIG (*Special Interest Group*). O SIG tinha o intuito de desenvolver um padrão de comunicação para dispositivos e acessórios, utilizando ondas de rádio de curto alcance, baixa potência e baixo custo, que foi denominado *bluetooth*, sendo gerada a primeira especificação do projeto (versão 1.0), em julho de 1999. (BLUETOOTH, 2003).

Bluetooth (2003) afirmam que, o projeto recebeu o nome *Bluetooth* em homenagem a Harald Blaatand II, mais conhecido como Harald Bluetooth (940-981), um rei Viking que unificou a Dinamarca e a Noruega.

Billo (2003) argumenta que *bluetooth* é uma tecnologia para comunicação de dispositivos eletrônicos (não apenas computadores) que surgiu, inicialmente, com o intuito de substituir o cabeamento necessário para interconexão de tais dispositivos.

Bluetooth é um link de rádio de curta distância destinado a substituir as conexões por cabo (LIMA; MEDEIROS, 2005).

Segundo Bluetooth (2003), de uma forma geral, o padrão *bluetooth* visa facilitar as transmissões em tempo real de voz e dados, permitindo conectar quaisquer aparelhos eletrônicos, fixos ou móveis, que estejam de acordo com a tecnologia.

Devido ao sucesso do *bluetooth*, o grupo de padrões do IEEE (*Institute of Electrical and Electronics Engineers*), que examinava as redes especiais sem fio (IEEE 802.15), adotou o documento do *bluetooth* (versão 1.1) como base e começou a modificá-lo (LIMA; MEDEIROS, 2005).

A tecnologia expandiu-se tanto que passou a ocupar espaço das redes sem fio, tornando-se mais útil, e gerando uma competição pelo mercado, com o padrão de rede sem fio 802.11. Esta competição acarretou ainda em interferências elétricas um com o outro, devido à utilização da mesma faixa de frequência (2.4 a 2,48 GHz) (LIMA; MEDEIROS, 2005), na Figura 2 é demonstrada a tecnologia *bluetooth* sendo usada para conectar aparelhos eletrônicos domésticos.

Lima e Medeiros (2005) argumentam que o padrão *bluetooth*, pretende, no mínimo:

- Eliminar fios e cabos entre dispositivos fixos e móveis separados por pequenas distâncias (10 metros);
- Facilitar comunicação de dados e de voz;
- Ativar redes *ad hoc* e fornecer sincronização automática entre vários dispositivos compatíveis com a tecnologia *bluetooth*.

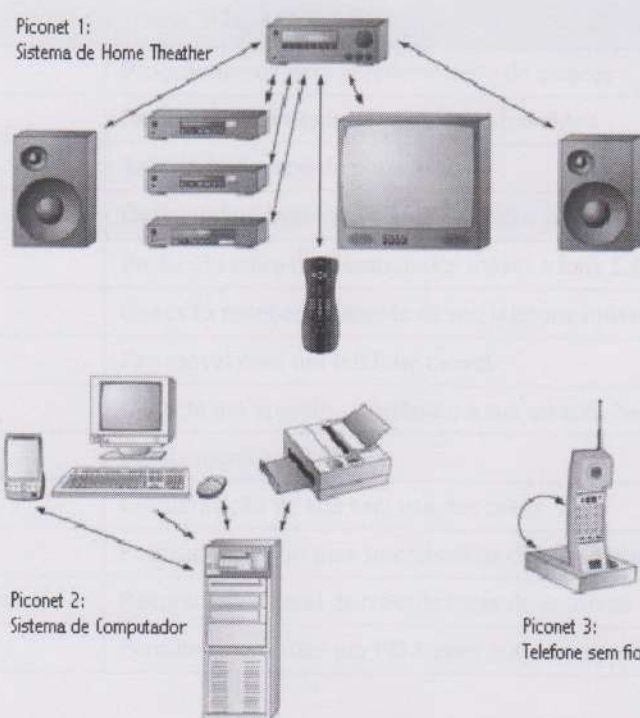


Figura 2 – Tecnologia *bluetooth* sendo usada para conectar aparelhos eletrônicos domésticos (LIMA; MEDEIROS, 2005).

2.2.1. Aplicações do *Bluetooth*

Conforme Lima e Medeiros (2005), enquanto os outros protocolos só fornecem canais (a ligação) entre entidades que se comunicam, o *bluetooth* identifica 13 aplicações específicas que serão admitidas e fornece diferentes pilhas de protocolos para cada uma, demonstradas na Tabela 1:

NOME	DESCRIÇÃO
Acesso genérico	Procedimentos para gerenciamento de enlaces
Descoberta de Serviço	Protocolo para descobrir serviços oferecidos
Porta serial	Substitui um cabo de porta serial
Intercâmbio genérico de objetos	Define relacionamento cliente/servidor para movimentação de objetos
Acesso de LAN	Protocolo entre um computador móvel e uma LAN fixa
Rede dial-up	Conexão notebook através de um telefone móvel
Fax	Fax móvel com um telefone móvel
Telefonia sem fio	Conecta um aparelho telefônico à sua estação-base local
Intercomunicador	Intercomunicação digital
Fone de ouvido	Comunicação de voz sem uso das mãos
Push de objetos	Fornece um meio para intercambiar objetos simples
Transferência de arquivos	Recurso mais geral de transferência de arquivos
Sincronização	Permite sincronizar um PDA com outro computador

Tabela 1 – Os perfis do *bluetooth* (LIMA; MEDEIROS, 2005).

2.2.2. Características de Operação

Para Lima e Medeiros (2005) o *bluetooth* possui as seguintes características de operação: estabelecendo conexão, solicitação e paginação, redes de difusão.

2.2.2.1 Estabelecendo Conexão

Para melhor entendimento do estabelecimento de conexão, necessita-se entender as *piconets*. Segundo *Bluetooth* (2003), a *piconet* é uma espécie de rede pessoal, que pode ter no máximo oito dispositivos conectados, é gerenciada por um *máster* (mestre), sendo os demais dispositivos da *piconet* denominados de escravos, a Figura 3 mostra um diagrama de uma *piconet*.

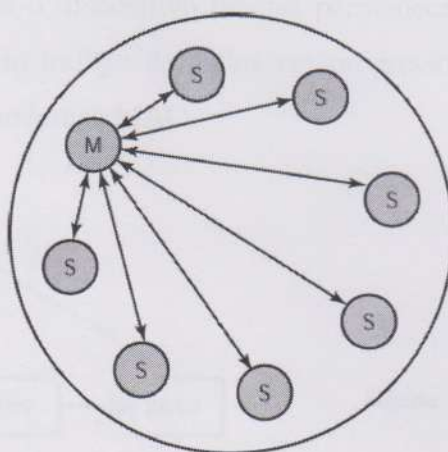


Figura 3 – Diagrama de uma piconet (LIMA e MEDEIROS, 2005).

De acordo com Lima e Medeiros (2005), dois dispositivos *bluetooth* quaisquer que estejam dentro de um raio de 10 metros podem estabelecer uma conexão ponto-a-ponto e/ou ponto-a-multiponto (como uma rede pessoal) “imediatamente”.

Um dispositivo *bluetooth* pode operar em dois estados de conexão principais: *connection* e *standby*. O dispositivo está no estado *connection* se ele está conectado a um outro dispositivo envolvido em atividades correntes (LIMA e MEDEIROS, 2005).

Se o dispositivo não estiver conectado, ou estiver conectado, mas não envolvido ativamente com outros dispositivos, ele estará automaticamente no estado *standby*. Estando neste estado, o dispositivo monitora mensagens de outros dispositivos a cada 1,28 segundos (LIMA e MEDEIROS, 2005).

Quando o dispositivo muda de estado (*standby* para *connection*) ele pode assumir quatro modos possíveis de conexão, o diagrama de estados dos dispositivos *bluetooth* é mostrado na Figura 4 (LIMA e MEDEIROS, 2005):

Active: participa ativamente da *piconet*, seja transmitido ou recebendo informação. Escravos são sincronizados com o mestre.

Sniff: monitora a *piconet* a uma taxa reduzida, diminuindo assim seu consumo de energia. A taxa de sniff é programável e varia de uma aplicação para outra;

Hold: dentro de uma *piconet*, as unidades mestre podem colocar as unidades escravas em modo *Hold*. É apenas mantido o timer interno do dispositivo ativo. Um exemplo de aplicação seriam sensores de temperatura;

Park: usado quando o dispositivo precisa permanecer conectado a uma *piconet*, mas não necessita participar do tráfego de dados em progresso. Permanece sincronizado à *piconet*, porém, suspende seu endereço MAC.

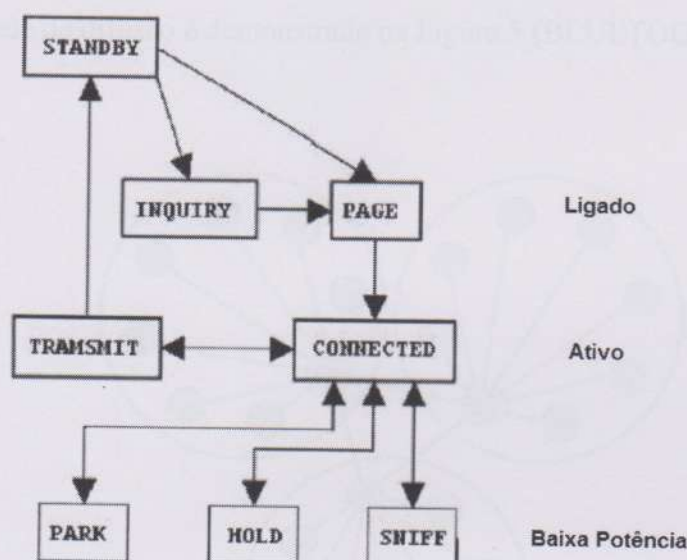


Figura 4 – Diagrama de estados dos dispositivos *bluetooth* (LIMA e MEDEIROS, 2005).

2.2.2.2 Solicitação e Paginação

Dispositivos *bluetooth* são iniciados no estado de baixo consumo de energia *standby*, ficam procurando um outro, ao perceberem o outro é iniciada a conexão, a unidade que encontrou a conexão assume a função de mestre da mini-rede (LIMA e MEDEIROS, 2005).

De acordo com Lima e Medeiros (2005), no procedimento de início de uma rede os dispositivos podem executar dois tipos diferentes de comandos, a solicitação e a página. Solicitação é emitida quando o número de identificação (endereço) do outro dispositivo ainda não é conhecido. Quando se conhece o endereço do dispositivo é emitido o comando página, que serve para “despertar” a outra unidade e estabelecer uma conexão plena entre os dois dispositivos.

2.2.2.3 Redes de Difusão

Para conexão de mais de oito dispositivos, utiliza-se uma rede de difusão (*scatternet*), que consiste em criar diversas *piconet*, depois conectar os mestres de cada *piconet*, uma rede de difusão é demonstrada na Figura 5 (BLUETOOTH, 2003).

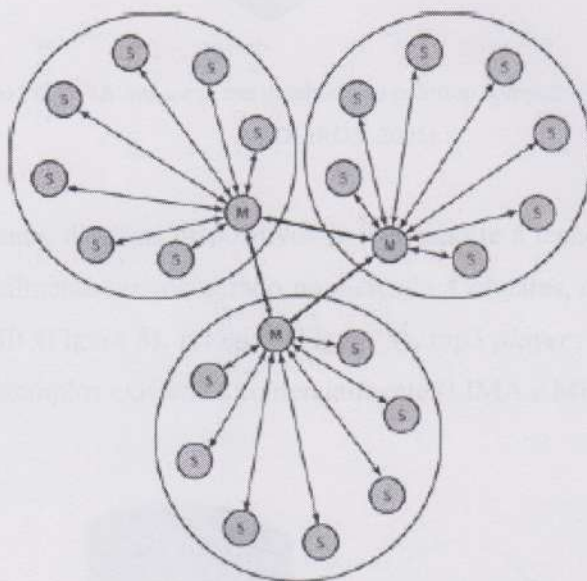


Figura 5 – Rede de difusão (*Scatternet*) (LIMA e MEDEIROS, 2005).

Toda comunicação entre *piconets* diferentes são filtradas pelos dispositivos mestres das *piconets* individuais. Uma rede de difusão suporta no máximo 10 *piconets* (80 dispositivos *Bluetooth* diferentes), se o número for maior, a rede fica saturada, visto que a tecnologia só utiliza 79 frequências no total (LIMA e MEDEIROS, 2005).

2.2.3 Dispositivos

Segundo Lima e Medeiros (2005), os primeiros produtos com a tecnologia Bluetooth no mercado eram dispositivos complementares que adicionavam recursos da tecnologia a produtos existentes.

Por exemplo, com um cartão PCMCIA Bluetooth, é possível para tornar um notebook ou palmtop compatíveis com o padrão, como ilustra a Figura 6.



Figura 6 – cartão PCMCIA que torna um notebook ou palmtop compatíveis com o padrão (LIMA e MEDEIROS, 2005).

Atualmente, diversos dispositivos já dão suporte à tecnologia Bluetooth de forma nativa, podendo facilmente ser encontrado no mercado. Celulares, *notebooks*, *headsets* (Figura 7), adaptadores USB (Figura 8), relógios (Figura 9), mp3 *players* e PDA's (Figura 10) são alguns dos vários exemplos existentes comercialmente (LIMA e MEDEIROS, 2005).



Figura 7 – Primeiro *headset Bluetooth* do mundo, juntamente com sua estação base (LIMA e MEDEIROS, 2005).



Figura 8 – Interface USB, que permite conexão *Bluetooth* (LIMA e MEDEIROS, 2005).



Figura 9 – Relógio e caneta compatíveis com *Bluetooth* (LIMA e MEDEIROS, 2005).



Figura 10 – Telefone/PDA *Communicator* da Ericsson, compatível com *Bluetooth* (LIMA e MEDEIROS, 2005).

2.3. PLATAFORMAS DE DESENVOLVIMENTO

“É uma estrutura digital feita especialmente para construir programas de computador, ou seja, um software que monta softwares. A plataforma funciona com métodos e padrões de trabalho bem definidos” (FRAMEWORK, 2006).

2.3.1. Symbian

Bourbon (2005, p. 12) argumenta que líderes da indústria de comunicação sem-fio, Nokia, Motorola, Siemens, Sony Ericsson, Matsushita (Panasonic), Psion e Siemens, criaram uma empresa privada, em 1998, denominada *Symbian*. Com a finalidade de criar um sistema operacional mais complexo que garante uma plataforma confiável e versátil para empresas de desenvolvimento de software. A *Symbian* é fornecedora de um sistema operacional aberto, avançando e padronizado para celulares chamado *Symbian OS*. O *Symbian OS* é baseado no sistema operacional EPOC da Psion e também é uma plataforma de desenvolvimento.

2.3.2. Brew

A BREW (*Binary Runtime Environment for Wireless* - Ambiente Binário de Execução para sem-fio), foi desenvolvida pela QUALCOMM, é uma solução completa para o desenvolvimento de aplicações sem-fio (*wireless*). A solução completa BREW é composta do BREW SDK (Kit para Desenvolvimento de Software) para desenvolvedores de aplicativos, do software cliente BREW, de ferramentas para o *porting* (portabilidade) para fabricantes de celulares, e do BREW *Delivery System* (BDS - Sistema de Distribuição BREW), que é controlado e gerenciado pelas operadoras (de telefonia), permitindo-lhes facilmente colocar aplicações de desenvolvedores no mercado e coordenar o processo de cobrança e pagamento (BOURBON, 2005).

2.3.3. Java

Para Deitel e Deitel (2001, p. 56), Java é uma linguagem de programação de computador Orientada à Objeto que atende a iniciantes, mas é também uma poderosa ferramenta de desenvolvimento para programadores experientes.

A *Sun Microsystems* finalizava em 1991, uma pesquisa corporativa com o codinome *Green*, que resultou em uma linguagem de programação baseada em C e C++ , voltada a dispositivos eletrônicos, batizada de *OAK* (carvalho) por James Gosling seu criador. Recebeu este nome em homenagem a uma árvore que dava para a janela de seu escritório na *Sun* (DEITEL; DEITEL, 2001).

De acordo com Topley (2002, p. 9), a *Sun Microsystems* criava uma nova linguagem de programação chamada *OAK* voltada para dispositivos eletrônicos. O primeiro protótipo do *OAK* foi um controle portátil de casa batizado de *start7* utilizando rede sem fio e infravermelho. Ele poderia ser usado como controle remoto para televisão e possuía determinadas funções que hoje são utilizadas em PDA (*Personal Data Assistants* - assistentes digitais pessoais). Era um software que não exigia muita memória.

Descobriu-se mais tarde que já existia uma linguagem com o mesmo nome, sendo rebatizada para Java (cidade de origem de um tipo de café importado), quando uma equipe da *Sun* freqüentava uma cafeteria local (DEITEL; DEITEL, 2001).

No início a linguagem Java não teve o impacto esperado, porém em 1993, com o surgimento da *World Wide Web*, os integrantes da *Sun* viram o potencial de Java para a criação de páginas *Web* com conteúdo dinâmico (conteúdos que ficam se modificando). Em maio de 1995, Java era anunciada formalmente em uma conferência importante e gerou interesse imediato na comunidade comercial (DEITEL; DEITEL, 2001).

Muchow (2002, p. 1) argumenta que tudo começou com uma versão Java, conhecida como *Java 2 Standart Edition* (J2SE), e a frase "Write once, run Anywhere " (escreva uma vez, execute em qualquer lugar). A linguagem Java foi desenvolvida para ser executada em qualquer plataforma que suportasse a Máquina Virtual Java com o código escrito uma única vez.

Desde seu lançamento a linguagem Java vem se destacando, e ultrapassou o seu alcance, indo bem além das máquinas simples e dos computadores pessoais, e passou a fornecer suporte para aplicativos em nível empresarial de larga escala com a edição *Java 2 Enterprise Edition* (J2EE), depois voltando à sua origem, dando suporte às TVs habilitadas para *internet*, celulares e PDAs, com o *Java Micro Edition* (JME) (MUCHOW, 2002).

A linguagem Java consiste em partes chamadas classes, que possuem outras partes chamadas métodos. Os métodos realizam tarefas e ao completá-las retornam informações. Em Java o desenvolvedor pode fazer suas próprias classes ou aproveitar as muitas existentes em bibliotecas de classes Java, mais conhecidas como APIs (*Applications Programming Interfaces* – interfaces de programas aplicativos) (DEITEL; DEITEL, 2001).

Para Deitel e Deitel(2001, p. 66), a linguagem Java diferentemente de outras linguagens, como C e C++, possibilita ao programador desenvolver sistemas com atividades ocorrendo em paralelo, quer dizer, que podem ocorrer ao mesmo tempo.

Java é uma linguagem portátil que funciona em quase todas as Plataformas, necessitando apenas que tenha a Máquina Virtual Java instalada para que os programas funcionem. Sua plataforma está dividida em quatro edições: *Java 2 Standard Edition* (J2SE), *Java 2 Enterprise Edition* (J2EE), *Java Micro Edition* (JME) e a mais nova *Java Card*.

Muchow (2002, p. 2) argumenta que as edições são reunidas como, as figuras 11 e 12 exemplificam as edições:

- Standard Edition (J2SE): projetada para execução em máquinas simples de computadores pessoais e estações de trabalho;
- Enterprise Edition (J2EE): com suporte interno para *servlets*, JSP (*Java servlet Page*), essa edição é destinada a aplicativos baseados no servidor;
- Micro Edition (JME): projetada para dispositivos com memória, vídeo e poder de processamentos limitados;
- Java Card: Tecnologia Java para ser usada em dispositivos com enormes restrições de Recursos – *Smart Cards* (projetada para cartões).



Figura 11 – Plataformas Java (J2EE, J2SE, JME, *Java Card*)

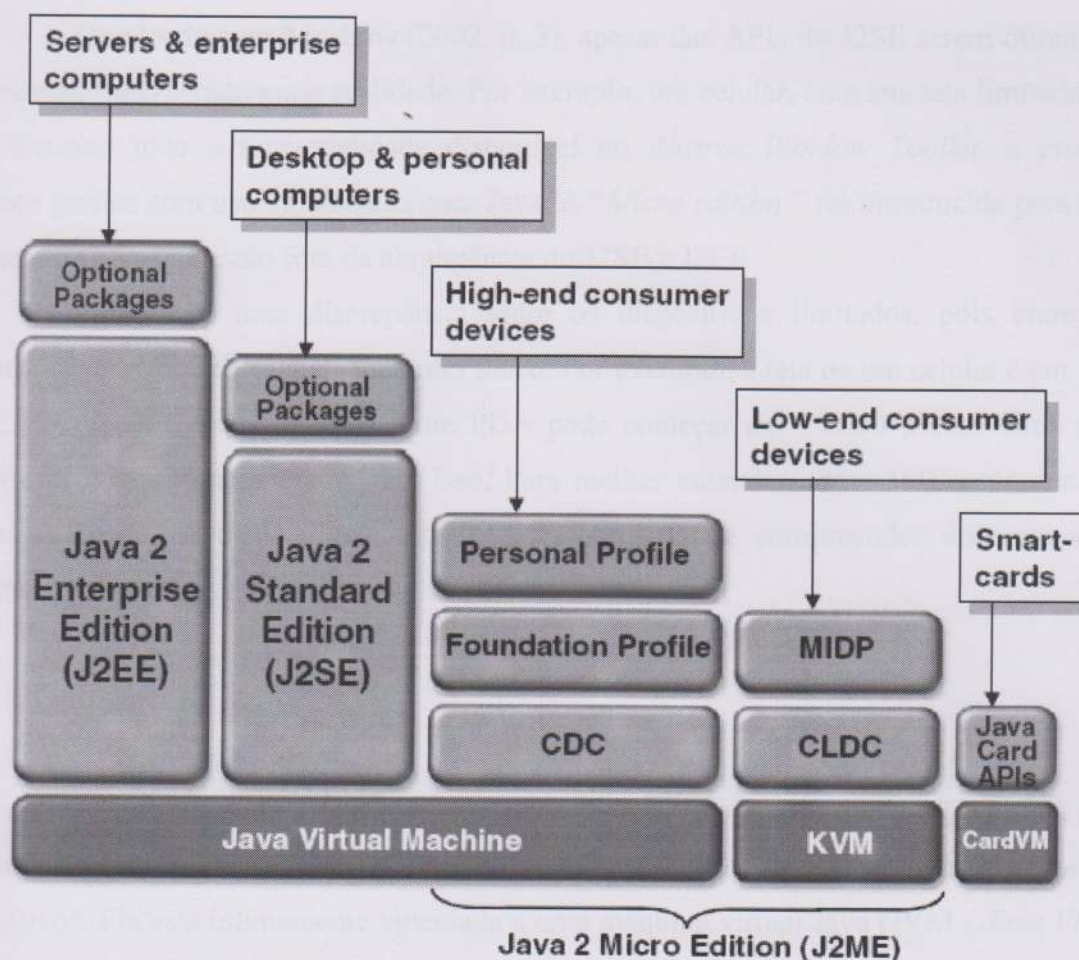


Figura 12 – Plataformas Java (J2EE, J2SE, JME, Java Card) (JODE et al, 2004)

Java agora retorna a sua origem e passa a ser utilizada para desenvolvimento de aplicações para dispositivos com poder de processamento, de vídeo e de memória limitados, como TVs, celulares, PDA, com a edição JME.

2.3.4. JME

O desenvolvimento em JME destaca-se por si próprio, pois tem suporte a uma ampla variedade de dispositivos e portabilidade entre plataformas, e a aceitação entre os fabricantes e provedores de serviço tem sido enorme (MUCHOW, 2002).

JME é diretamente destinado aos dispositivos consumidores com poder limitado. Muitos desses dispositivos não têm opção de *download* (baixar arquivo) e *software* de instalação, além dos que foram configurados durante o processo de fabricação. Com a introdução do JME os aparelhos deixaram de ser estáticos e passaram a ter a opção de *download* e de instalar aplicativos Java e conteúdos (MUCHOW, 2002).

De acordo com Muchow (2002, p. 3), apesar das APIs do J2SE serem ótimas, em dispositivos “micro” não eram realidade. Por exemplo, um celular, com sua tela limitada, não pode fornecer toda a funcionalidade disponível no *Abstract Window Toolkit*, a primeira interface gráfica com usuário lançada com Java. A “*Micro edition*” foi introduzida para tratar das necessidades que estão fora da abrangência do J2SE e J2EE.

Porém, há uma discrepância entre os dispositivos limitados, pois entre eles encontra-se diferença de tela, de tamanho físico. Por exemplo, a tela de um celular é em torno de 12.288 pixels (pontos da tela) e um PDA pode começar em 20.000 pixels. Uma única plataforma Java não acomodará tudo isso. Para melhor entender como JME acomoda uma ampla variedade de dispositivos, tem-se a necessidade de compreender dois conceitos: configuração e perfil.

2.3.4.1. Configuração

A Sun, com o intuito de atender à diversidade de produtos que tem suporte à JME, introduziu a Configuração, que define uma plataforma Java para uma ampla variedade de dispositivos. Ela está intimamente vinculada a uma máquina virtual Java (JVM - *Java Virtual Machine*). Na verdade, uma configuração define os recursos da linguagem Java e as bibliotecas Java básicas da JVM para essa configuração em particular (MUCHOW, 2002).

Para Jode et al (2004, p. 5), a configuração consiste em uma combinação da máquina virtual e as classes mínimas de uma biblioteca Java para prover a base de funcionalidades para uma série de equipamentos com características similares, como conexão de rede, poder de processamento e memória.

A linha divisória de aplicação de uma configuração é, de modo geral, baseada na memória, no vídeo e na conectividade de rede (ou limitações disso) e no poder de processamento disponível em um dispositivo (MUCHOW, 2002).

A configuração é dividida basicamente em duas, *Connected Device Configuration* (CDC- Configuração de Dispositivo Conectado) e *Connected Limited Device Configuration* (CLDC - Configuração de Dispositivo Conectado Limitado).

CONNECTED DEVICE CONFIGURATION

Segundo Muchow (2002), CDC Destina-se a dispositivos menos limitados como TV interativa, telemetria de veículos.

Exigência mínima de *hardware*:

- 512 kilobytes (no mínimo) de memória para executar o Java.
- 256 kilobytes (no mínimo) de memória para alocação de memória em tempo de execução.
- Conectividade de rede, largura de banda possivelmente persistente e alta.

CONNECTED LIMITED DEVICE CONFIGURATION

De acordo com Muchow (2002), CLDC destina-se a dispositivos mais limitados como celulares, PDAs.

Exigência mínima de *hardware*:

- 128 kilobytes de memória para executar a JVM e as bibliotecas CLDC (esta memória deve preservar o conteúdo, mesmo quando o aparelho for desligado).
- 32 kilobytes para alocação de memória em tempo de execução.
- Interface restrita com o usuário.
- Baixo poder, normalmente alimentado por bateria.
- Conectividade de rede, normalmente dispositivos sem fio com largura de banda baixa e acesso intermitente.

Exigência mínima de software:

- Seleção e ativação de aplicativos;
- A capacidade de remover aplicativos Java do dispositivo.

O CLDC não dá suporte à matemática de pontos flutuantes, nem ao método *finalize()*, utilizado em Java para colocar o código que faz a limpeza de recursos alocados na memória. Suporta ainda um conjunto limitado de exceções de tratamento de erro. O CLDC não tem esse suporte tendo em vista a exigência de um grande consumo de processamento dessas funcionalidades (MUCHOW, 2002).

O CLDC “próxima geração” pode incluir suporte a números de ponto flutuante, recursos de tratamento de erro adicionais e um gerenciador de segurança mínimo (MUCHOW, 2002).

Conforme Muchow (2002, p. 4), à medida que a tecnologia nos oferecer mais poder de processamento, com mais memória e recursos de tela, a sobreposição entre essas categorias se tornará maior. Isso leva diretamente à discussão sobre os perfis.

2.3.4.2. Perfis

Um perfil é uma extensão de uma configuração. Ele fornece as bibliotecas para um desenvolvedor escrever aplicativos para um tipo em particular de dispositivo (MUCHOW, 2002). O perfil fornece suporte mais específico para os dispositivos, tendo em vista que mesmo dentro de uma configuração pode-se ter dispositivos ainda mais distintos.

Para Muchow (2002, p. 4), apesar de apresentarem especificações de hardware parecidas, utilizando a mesma configuração, pode-se constatar que, o que parece limitante para um dispositivo em uma configuração pode parecer uma abundância para outro. Por exemplo, a analogia do tamanho da tela do telefone celular em relação à de um PDA. Para tratar dessa ampla variação de recursos e para dar mais flexibilidade à medida que a tecnologia muda, a *Sun* introduziu o conceito de perfil na plataforma JME.

O perfil se divide em duas grandes categorias, baseados em CDC e baseados em CLDC. Podemos destacar dois perfis baseados em CLDC o *Mobile Information Device Profile* (MIDP – Perfil de Dispositivo de Informação Móvel) e *Information Module Profile* (IMP). Podem-se destacar três perfis baseados em CDC, *Foundation Profile* (FP), *Personal Basis Profile* (PBP) e *Personal Profile* (PP).

MOBILE INFORMATION DEVICE PROFILE

A mais famosa camada do JME é uma extensão do CLDC.

De acordo com Muchow (2002, p. 21), exigência mínima de *hardware*:

- A tela deve suportar pelo menos 96 x 54 pixels;
- Deve ter pelo menos um dispositivo de entrada: teclado de uma mão (teclado de telefone), teclado de duas mãos (teclado comum) ou uma tela de toque;
- 128 *kilobytes* de memória para executar o componente *Mobile Information Device* (MID – dispositivo de informação móvel);
- Pelo menos 8 *kilobytes* de memória para os aplicativos armazenarem dados persistentes, como configurações e dados do aplicativo;
- 32 *kilobytes* de memória para executar o Java.
- Conectividade de rede sem fio.

O MIDP 2.0 (nova geração) pode incluir suporte a HTTPS, soquetes e datagramas, extensões para interface de baixo nível e um analisador XML (MUCHOW, 2002).

Um MIDlet é um aplicativo Java (classe Java) projetado para ser executado em um dispositivo móvel. Mais especificamente, uma MIDlet tem como classe básica a CLDC e o MIDP (MUCHOW, 2002).

INFORMATION MODULE PROFILE

Projetado para dispositivos sem mecanismos de entrada/saída (I/O), fornece os mesmos serviços do MIDP, porém sem os serviços de I/O, e não possui uma interface gráfica rica com o usuário.

FOUNDATION PROFILE (FP)

O perfil FP é o mais baixo nível de perfil para CDC. É utilizado para implementações fortemente embarcadas, fornecendo uma implementação CDC com capacidade de rede e sem interface gráfica. Podendo ser também com o Personal Basis Profile e o Personal Profile para dispositivos que requerem uma interface gráfica com o usuário (GUI) (BOURBON, 2005).

PERSONAL PROFILE (PP)

Bourbon (2005) argumenta que o PP é utilizado em dispositivos como PDAs de última geração, aparelhos de comunicação e videogames, dispositivos que exigem uma Interface (GUI) completa ou tenha suporte para *Applets* de Internet. PP substitui a tecnologia PersonalJava™ e fornece um claro caminho para migração de aplicativos PersonalJava para a plataforma JME.

PERSONAL BASIS PROFILE (PBP):

Segundo Bourbon (2005), o PBP da mesma maneira que o PP operam em camadas sob o CDC e PF. Ele é utilizado principalmente em receptores de TV, sistemas veiculares e quiosques de informações, pois são dispositivos que possuem conexão à rede e suportam um nível básico de apresentação gráfica.

Para melhor entender o desenvolvimento em JME existe a necessidade de se entender sobre os MIDlets.

2.3.4.3. Midlet

Segundo Muchow (2002), uma *MIDlet* é uma aplicação *Java* projetada para ser executada em um dispositivo móvel. O gerenciador de aplicativos do dispositivo, através de seus métodos, controla a aplicação (por exemplo, pausando a aplicação, permitindo que uma chamada telefônica seja atendida e, posteriormente, a reiniciando).

CICLO DE VIDA DA MIDLET:

“Uma MIDlet passa por várias fases como parte de seu ciclo de vida, e é sempre considerada como estando em um de três estados:” (MUCHOW, 2002)

Uma *MIDlet* pode estar em três estados: pausa, quando depois de estar em execução tiver de ser parada para, posteriormente, reiniciar (a aplicação ir para o estado de pausa, para que uma chamada seja atendida); ativa, quando estiver em execução; destruída, quando a *MIDlet* liberar todos os recursos utilizados e for desligada pelo gerenciador (MUCHOW, 2002).

Uma *MIDlet* estende a classe MIDlet (uma classe abstrata que possui três métodos abstratos), como mostra a Figura 13.

```

public class MidletExemplo extends MIDlet {

    //método construtor não é obrigatório
    public MidletExemplo() {

    }

    //chamado pelo gerenciador para iniciar a Midlet
    public void startApp() {

    }

    //chamado pelo gerenciador para pausa da Midlet
    public void pauseApp() {

    }

    //chamado pelo gerenciador para desligamento da Midlet
    public void destroyApp(boolean unconditional) {

    }

}

```

Figura 13 – Métodos Mínimos de uma *MIDlet* (MUCHOW, 2002)

Muchow (2002) afirma que, para colocar uma *MIDlet* em estado ativo o gerenciador de aplicativos chama o método *startApp()*, que pode ser chamado várias vezes durante o ciclo de vida de uma *MIDlet*, sendo de grande importância o código utilizado nesse método. Quando há necessidade de se colocar o *MIDlet* no estado de pausa é utilizado o método *pauseApp()*, que notifica a *MIDlet* que ela vai modificar de estado e deve liberar o máximo de recurso possível. Por último, para que todos os recursos utilizados sejam liberados e a *MIDlet* seja desligada é chamado o método *destroyApp()*.

Cada *MIDlet* possui um objeto *Display*, gerenciador da tela, que controla o que é exibido na tela e quando é exibido, o qual é responsável por recuperar informações sobre a tela atual e contém métodos para exibição na tela. Os objetos exibidos na tela são do tipo *Displayable* (o objeto que é visto na tela). O MIDP inclui duas subclasses de *Displayable*: *Screen* e *Canvas*. Onde os *Screens* (TextBox, Alert, List e Form) são componentes de alto nível da interface com o usuário e o *Canvas* é utilizado para desenhos gráficos personalizados e tratamento de evento de baixo nível, muito utilizado para o desenvolvimento de jogos. Através da *Displayable* pode-se adicionar ou remover objetos *Command*, que fornecem um meio de interação para o usuário (MUCHOW, 2002). Na tabela 2 é demonstrada a API *MIDlet*, na Tabela 3 a API *Display* e a Tabela 4 a API *Displayable*.

Classe MIDlet	
Método	Descrição
Comunicação do gerenciador de aplicativos com a MIDlet	
abstract void destroyApp (boolean unconditional)	A MIDlet está para ser desligada
abstract void pauseApp()	A MIDlet está para ser colocada no estado de pausa.
abstract void startApp()	A MIDlet foi colocada no estado ativo
Comunicação da MIDlet com o gerenciador de aplicativos	
final void notifyDestroyed()	A MIDlet está pedindo para ser desligada
final void notifyPaused()	A MIDlet está pedindo para fazer uma pausa
final void resumeRequest()	A MIDlet está pedindo para tornar ativa (após uma pausa)
Pedido de atributo da MIDlet para o gerenciador de aplicativos	
final String getAppProperty(String key)	Obtém atributos dos arquivos

Tabela 2 – API MIDlet (MUCHOW, 2002)

Classe Display	
Método	Descrição
Static Display getDisplay(MIDlet m)	Obtém o objeto Display dessa MIDlet
Displayable getCurrent()	Obtém o objeto Displayable corrente
void setCurrent(Alert alerta, Displayable próximo)	Mostra um alerta, seguido do objeto Displayable especificado
void setCurrent(Displayable próximo)	Mostra um novo objeto Displayable
boolean isColor()	Verifica se o dispositivo suporta cor
Int numColors()	Verifica a quantidade de cores suportada
void callSerially(Runnable r)	Pede para que um objeto executável seja chamado depois da repintagem

Tabela 3 – API Display (MUCHOW, 2002)

Classe Displayable	
Método	Descrição
void addCommand (Command comando)	Adiciona um comando no objeto Displayable
void removeCommand (Command comando)	Remove um comando no objeto Displayable
void setCommandListener (CommandListener listen)	Adiciona um CommandListener no objeto Displayable
boolean isShown()	Verifica se o Displayable está visível na tela

Tabela 4 – API Displayable (MUCHOW, 2002)


```
public class ExemploCanvas extends Canvas implements CommandListener {  
    /**  
     * construtor  
     */  
    public ExemploCanvas() {  
        try {  
            // definindo o canvas para ouvir evento  
            setCommandListener(this);  
            // adicionando o comando sair com prioridade 1  
            addCommand(new Command("Sair", Command.EXIT, 1));  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
  
    /**  
     * método de pintura  
     */  
    public void paint(Graphics g) {  
    }  
  
    /**  
     * método de tratamento de eventos (utilizado quando uma tecla é pressionada)  
     */  
    protected void keyPressed(int keyCode) {  
    }  
  
    /**  
     * método de tratamento de eventos (utilizado quando uma tecla é solta)  
     */  
    protected void keyReleased(int keyCode) {  
    }  
  
    /**  
     * método de tratamento de eventos (utilizado quando uma tecla é repetida)  
     */  
    protected void keyRepeated(int keyCode) {  
    }  
  
    /**  
     * método de tratamento de eventos (utilizado quando o ponteiro é arrastado)  
     */  
    protected void pointerDragged(int x, int y) {  
    }  
  
    /**  
     * método de tratamento de eventos (utilizado quando o ponteiro é pressionado)  
     */  
    protected void pointerPressed(int x, int y) {  
    }  
  
    /**  
     * método de tratamento de eventos (utilizado quando o ponteiro é liberado)  
     */  
    protected void pointerReleased(int x, int y) {  
    }  
  
    /**  
     * tratamento de eventos de comandos (utilizado quando um comando é  
     * pressionado)  
     */  
    public void commandAction(Command command, Displayable displayable) {  
    }  
}
```

Figura 14 – Métodos de uma Classe *Canvas*

Outro conceito importante para o desenho é o conceito de coordenadas, que tem como origem do desenho o canto superior esquerdo da tela, sendo a coordenada 0,0, como mostra a Figura 15. Os valores do eixo x aumentam para a direita e os do eixo y aumentam para baixo (MUCHOW, 2002).

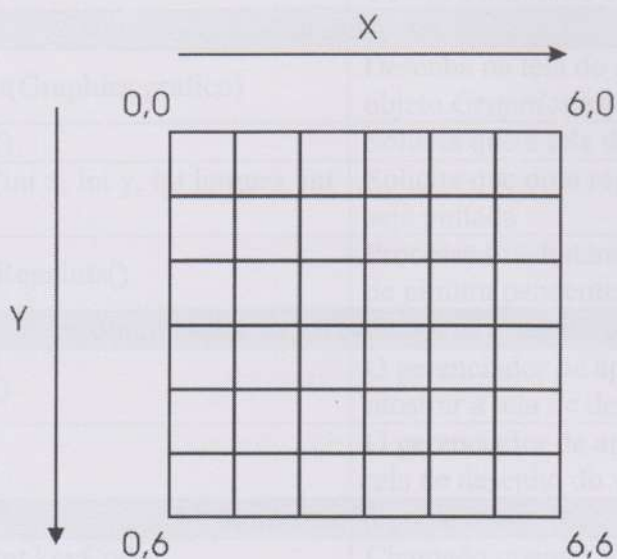


Figura 15 – Coordenadas em uma Canvas

<code>int getActionType(int keyCode)</code>	Retorna o código de ação de uma ação de teclado
<code>int getCharAction(int keyCode)</code>	Retorna o código de ação de uma ação de caractere
<code>String getKeyText(int keyCode)</code>	Retorna o texto de uma ação de teclado
<code>boolean hasFocus()</code>	Verifica se a plataforma possui o foco
<code>boolean hasMouseMotionEvent()</code>	Verifica se a plataforma possui o movimento do mouse
<code>void paint(Graphics g)</code>	Chamado quando a plataforma é pintada
<code>void paint(Graphics g, int x, int y)</code>	Chamado quando a plataforma é pintada em uma posição específica
<code>void paint(Graphics g, int x, int y, int w, int h)</code>	Chamado quando a plataforma é pintada em uma área específica

Tabela 5 – API Canvas (MUCHOW, 2002)

Na maioria das vezes, a classe Canvas é utilizada em conjunto com a classe Graphics para desenhar na tela. Um código de um desenho é exibido na Figura 7 e demonstrado como funciona no aplicativo na Figura 8.

Classe Canvas	
Método	Descrição
Métodos de coordenada	
Int getWidth()	Obtém a largura da tela de desenho
Int getHeight()	Obtém a altura da tela de desenho
Métodos paint	
abstract void paint(Graphics grafico)	Desenha na tela de desenho usando o objeto <i>Graphics</i> especificado
final void repaint()	Solicita que a tela de desenho seja pintada
final void repaint(int x, int y, int largura, int altura)	Solicita que uma região da tela de desenho seja pintada
final void serviceRepaints()	Processa imediatamente todos os pedidos de pintura pendentes
Comunicação do gerenciador de aplicativos	
void showNotify()	O gerenciador de aplicativos está para mostrar a tela de desenho no vídeo
void hideNotify()	O gerenciador de aplicativos removeu a tela de desenho do vídeo
Métodos de código de tecla	
void keyPressed(int keyCode)	Chamado quando a tecla é pressionada
void keyReleased(int keyCode)	Chamado quando a tecla é solta
void keyRepeated(int keyCode)	Chamado quando a tecla é repetida
String getKeyName(int keyCode)	A String de texto que representa o código de tela
Métodos de ação de jogo	
Int getKeyCode(int gameAction)	Determina o código de tecla de uma ação de jogo
Int getGameAction(int keyCode)	Obtém a ação de jogo, se houver, do código de tecla
String getKeyName(int keyCode)	Obtém o nome de um código de tela
Métodos de ponteiro	
boolean hasPointerEvents()	Verifica se a plataforma suporta ponteiro
boolean hasPointerMotionEvents()	Verifica se a plataforma detecta movimento de ponteiro (clica/aponta e arrasta)
void pointerDragged(int x, int y)	Chamado quando o ponteiro é arrastado
void pointerPressed(int x, int y)	Chamado quando o ponteiro é pressionado
void pointerReleased(int x, int y)	Chamado quando o ponteiro é liberado

Tabela 5 –API Canvas (MUCHOW, 2002)

No método *paint* da classe *Canvas* é utilizado o objeto *Graphics* para desenhar na tela, um código de um desenho é exibido na Figura 7 e demonstrado como fica no aparelho na Figura 8.

```

public void paint(Graphics g) {
    //definindo a cor como branco
    g.setColor(255,255,255);
    //desenhando um retângulo para limpar a tela
    g.fillRect(0,0,getWidth(),getHeight());
    //definindo a cor como preto
    g.setColor(0,0,0);
    //desenhando uma linha
    g.drawLine(0,0,20,20);
    //desenhando um retângulo
    g.drawRect(30,30,20,20);
    //desenhando um retângulo sólido
    g.fillRect(52,50,20,20);
    //desenhando um círculo
    g.drawArc(72,70,20,20,0,360);
    //desenhando um círculo sólido
    g.fillArc(92,90,20,20,0,360);
}

```

Figura 16 – Exemplo de código de um desenho na tela

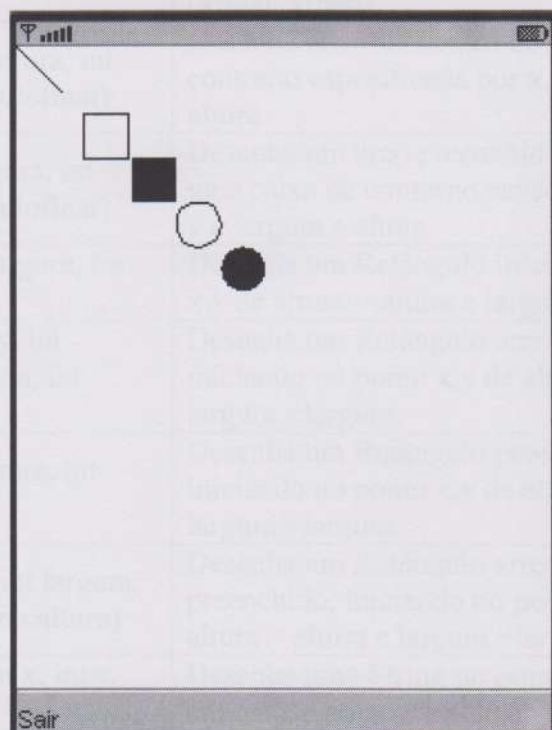


Figura 17 – Execução do código da figura 13

Na Tabela 6 são demonstrados alguns métodos da API *Graphics*.

Classe Graphics	
Método	Descrição
Suporte para cor	
void setColor(int vermelho,int verde, int azul)	Configura a cor, especificando cada componente (vermelho, verde e azul) para obter a cor
void setColor(int RGB)	Configura a cor, combinando cada componente (vermelho, verde e azul) em um único valor inteiro
Int getColor()	Obtém a cor corrente como um único valor inteiro
Int getBlueComponente()	Obtém o componente azul da cor atual
Int getGreenComponente()	Obtém o componente verde da cor atual
Int getRedComponente()	Obtém o componente vermelho da cor atual
void setGrayScale(int valor)	Configura a escala de tons
Int getGrayScale ()	Obtém a escala de tons atual
Métodos de traçado	
Int getStrokeStyle()	Obtém o estilo de traçado atual
void setStrokeStyle (int style)	Configura o estilo de traçado (podendo ser SOLID – linha cheia, DOTTED – linha tracejada)
Métodos de desenho	
void drawLine(int xinicio, int yinicio, int xfinal, int yfinal)	Desenha uma linha iniciada no ponto (xinicio,yinicio) e que termina no ponto (xfinal, yfinal)
void drawArc(int x, int y, int largura, int altura, int ânguloInicial, int angulofinal)	Desenha um arco dentro de uma caixa de contorno especificada por x, y e largura e altura
Void fillArc(int x, int y, int largura, int altura, int ânguloInicial, int angulofinal)	Desenha um arco preenchido, dentro de uma caixa de contorno especificada por x, y e largura e altura
void drawRect(int x, int y, int largura, int altura)	Desenha um Retângulo iniciando no ponto x,y de altura = altura e largura =largura
void drawRoundRect(int x, int y, int largura, int altura, int arcoLargura, int arcoaltura)	Desenha um Retângulo arredondado iniciando no ponto x,y de altura = altura e largura =largura
void fillRect(int x, int y, int largura, int altura)	Desenha um Retângulo preenchido, iniciando no ponto x,y de altura = altura e largura =largura
void fillRoundRect(int x, int y, int largura, int altura, int arcoLargura, int arcoaltura)	Desenha um Retângulo arredondado preenchido, iniciando no ponto x,y de altura = altura e largura =largura
void drawString(String texto, int x, int y, int ancora)	Desenha uma String no pont x,y alinhado em reação ao ponto pelo ancora

Tabela 6 – Alguns métodos da API *Graphics* (MUCHOW, 2002)

2.3.4.5. Api Bluetooth (JSR-82)

API Bluetooth Wireless Technology (JABWT) recebeu um nome formal JSR-82. É a API que define a arquitetura de uma aplicação JME com *Bluetooth* e as classes e métodos que devem ser implementados pelas aplicações compatível com a JSR-82. A API baseia-se nas configurações CLDC e opera na camada CLDC da arquitetura de implementação da plataforma JME, mas na prática, ela é usada como uma extensão de funcionalidades da camada de *profiles* do JME (LIMA, 2005).

A JSR-82 não é parte integrante do MIDP, sendo implementada pelo fabricante, podendo ou não ser encontrada nos aparelhos. Atualmente, apenas alguns dispositivos com MIDP 2.0, que têm a tecnologia *bluetooth*, possuem a API implementada. Apesar de ser uma API desenvolvida para dispositivos móveis, já existem empresas implementando-a para *desktop* (PÓVOA, 2005).

Segundo Pova (2005), a JSR-82 é composta de dois pacotes `javax.bluetooth.*` e `javax.obex.*`, o segundo não é suportado nos dispositivos. O pacote `javax.bluetooth.*` fornece dois tipos de protocolos (L2CAP - *Logical Link Control and Adaptation Protocol* e RFCOMM - *Radio Frequency Communication*) para comunicação.

Para utilização da API existe a necessidade de se entender duas classes que representam o dispositivo *LocalDevice*, que é demonstrado o código para definição de um objeto na Figura 18 e os métodos da API na Tabela 7, e *RemoteDevice*. Pova (2005) argumenta que, para a utilização do dispositivo *bluetooth*, existe a necessidade de se criar o objeto *LocalDevice*, que representa o dispositivo *bluetooth*, só podendo existir um único *LocalDevice*. Este gerencia várias propriedades do dispositivo (endereço do dispositivo e o nome do dispositivo, através dos métodos: `getBluetoothSddress()` e `getFriendlyName()`).

```

try {
    //define o dispositivo bluetooth
    LocalDevice dispositivoLocal = LocalDevice.getLocalDevice();
    //guardando o modo inicial
    int modoInicial = dispositivoLocal.getDiscoverable();
    //define o DiscoveryAgent
    DiscoveryAgent agente = dispositivoLocal.getDiscoveryAgent();
} catch (BluetoothStateException ex){
    ex.printStackTrace();
}

```

Figura 18 – Exemplo de utilização da classe LocalDevice (PÓVOA, 2005)

Classe LocalDevice	
Método	Descrição
LocalDevice getLocalDevice()	Obtém o LocalDevice do dispositivo
String getFriendlyName()	Obtém o nome do dispositivo
String getBluetoothAddress()	Obtém o endereço do dispositivo
int getDiscoverable()	Obtém o modo no qual o dispositivo encontra-se
DiscoveryAgent getDiscoveryAgent()	Obtém o DiscoveryAgent
void setDiscoverable(int modoDeDescoberta)	Define o modo de descoberta do dispositivo (podendo ser DiscoveryAgent.LIAC – torna o dispositivo possível de ser descoberto por um minuto; DiscoveryAgent.GIAC - torna o dispositivo possível de ser descoberto até que seja modificado o estado; DiscoveryAgent.NOT_DISCOVERABLE)

Tabela 7 – Métodos da API LocalDevice (PÓVOA, 2005)

Com a utilização da classe *RemoteDevice* consegue-se a representação do dispositivo *bluetooth* remoto, obtendo várias informações a respeito do dispositivo remoto (endereço, nome), além de poder configurar uma conexão quanto à autenticação, autorização e encriptação. O objeto *RemoteDevice* é adquirido no término do *inquiry* (LIMA, 2005).

Para se trabalhar com uma conexão *bluetooth* é necessário dois módulos, o Servidor da conexão e o Cliente, ambos utilizam a classe *LocalDevice*.

DISPONIBILIZANDO UM SERVIÇO:

Para disponibilizar um serviço *bluetooth* existe a necessidade de se entender dois pontos, como o serviço será identificado ou diferenciado (pois um dispositivo poderá ser servidor de vários serviços) e como serão gerenciados os serviços. Para a identificação a JSR-82 utiliza a classe UUID (*Universally Unique Identifiers* - Identificador Universal Único), que são identificadores que serão “únicos durante todo espaço de tempo”, e devem ser conhecidos tanto pelo servidor quanto pelo cliente da conexão, a Tabela 8 exemplifica os Métodos Construtores da API UUID. Eles são representados por números sem sinal de 16, 32 ou 128 bits (PÓVOA, 2005).

Classe UUID	
Método	Descrição
Métodos Construtores	
Public UUID(log valor)	Cria UUID's de 16 ou 32, recebe um atributo do tipo long
Public UUID(String valor, boolean pequeno)	Cria UUID's de 16 ou 32 se o pequeno for true e de 128 se for false, recebe um atributo String

Tabela 8 – Métodos Construtores da API UUID (PÓVOA, 2005)

Póvoa (2005) afirma que, para disponibilizar o serviço é necessário colocar o dispositivo em um dos modos de descoberta (GIAC ou LIAC) para que o servidor seja descoberto, como demonstra a Figura 19. Devendo, posteriormente, registrar o serviço através do método `Connector.open()`.

```

try{
    //definindo o dispositivo local
    dispositivoLocal = LocalDevice.getLocalDevice();
    //armazenando o modo inicial do dispositivo, para no término do jogo
    // voltar o dispositivo para o modo
    modoInicialDescoberta = dispositivoLocal.getDiscoverable();
    //definindo o modo do dispositivo para ser descoberto (no modo GIAC,
    // para ser descoberto a qualquer hora)
    dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
    //registrando o serviço
    notificador = (StreamConnectionNotifier) Connector.open(
        "btspp://localhost:00ABCDE0000000000000000AF2006BCEF");
    //aguardando a conexão
    conexao = notificador.acceptAndOpen();
    //chamar o método executar da classe BluetoothConexao (classe pai)
    executar(conexao);
} catch(Exception e){
    cancelar();
}

```

Figura 19 – Exemplo de disponibilização de um serviço

PROCURA POR DISPOSITIVOS (*INQUIRY*):

A JSR-82 oferece meios simples para a procura por dispositivos, através das classes *DiscoveryAgent* (responsável pela procura dos dispositivos e serviços) e *DiscoveryListener* (classe notificada quando um dispositivo ou serviço são encontrados ou não). A busca inicia através do método *startInquiry()*, que quando encontra um dispositivo chama o método *deviceDiscovered()* da *DiscoveryListener*, como mostra a Figura 20, passando o dispositivo encontrado e que tipo de dispositivo é (celular, PDA). No término da busca é chamado o método *inquiryCompleted()*, podendo a busca ser: *INQUIRY_COMPLETED* – indicando que a busca foi concluída normalmente, *INQUIRY_TERMINATED* – indicando que a busca foi cancelada e *INQUIRY_ERROR* – indicando que a busca terminou com erro (LIMA, 2005).

Depois da descoberta dos dispositivos existe a necessidade de verificar se o mesmo disponibiliza o serviço desejado ou não.

```

public void deviceDiscovered(RemoteDevice remoteDevice,
    DeviceClass deviceClass) {
    //adicionar o dispositivo em um list
}

public void inquiryCompleted(int caso) {

    switch (caso){
        //busca foi concluída normalmente
        case INQUIRY_COMPLETED:
            /*
             *
             *
             *
             */

            // busca foi cancelada
            case INQUIRY_ERROR:
                /*
                 *
                 *
                 *
                 */

                //busca terminou com erro
                case INQUIRY_TERMINATED:
                    /*
                     *
                     *
                     *
                     */
            }
    }
}

```

Figura 20 – Estrutura dos métodos deviceDiscovered() e inquiryCompleted()

PROCURA POR SERVIÇOS:

Devido ao *Inquiry* retornar todos os dispositivos *bluetooth* encontrados, existe a necessidade de se verificar qual dispositivo possui o serviço desejado. Através do método `searchServices()` é realizada a verificação do serviço, e chama o método `servicesDiscovered()`, quando um serviço especificado é encontrado (LIMA, 2005).

Segundo Póvoa (2005), quando a busca pelo serviço é terminada o método `serviceSearchCompleted()` é notificado, recebendo o status da busca que pode ter os seguintes valores, como mostra a Figura 21:

- `DiscoveryListener.SERVICE_SEARCH_COMPLETED` – Busca concluída normalmente.

- DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE – Busca não pode ser concluída, pois o dispositivo está fora de alcance.
- DiscoveryListener.SERVICE_SEARCH_ERROR – Busca terminou com erro.
- DiscoveryListener.SERVICE_SEARCH_NO_RECORDS – Busca terminou e nenhum serviço foi encontrado.
- DiscoveryListener.SERVICE_SEARCH_TERMINATED – Busca foi cancelada.

```

public void verificarServico(int selec){
    UUID[] uuidset = new UUID[]{new UUID(
        "00ABCDE000000000000000AF2006BCEF", false)};
    try{
        agente.searchServices(null,uuidset,rd[selec],this);
    }catch(IOException ex){
        midlet.mostrarAlerta("Erro na busca por serviços",AlertType.INFO);
    }
}

public void serviceSearchCompleted(int i, int i0) {
    switch(i0){
        //Busca foi cancelada.
        case SERVICE_SEARCH_TERMINATED:
            /*
             .
            */

            //Busca terminou com erro.
            case SERVICE_SEARCH_ERROR:

            //Busca terminou e nenhum serviço foi encontrado.
            case SERVICE_SEARCH_NO_RECORDS:

            //Busca não pôde ser concluída, pois o dispositivo está fora de alcance.
            case SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
                /*
                 .
                */

            //concluída normalmente.
            case SERVICE_SEARCH_COMPLETED:
                /*
                 .
                */
            }
    }
}

```

Figura 21 – Estrutura do método serviceSearchCompleted() e utilização do searchServices()

Depois de se verificar que o dispositivo dispõe do serviço desejado existe a necessidade de se estabelecer a comunicação entre os dispositivos.

COMUNICAÇÃO:

A conexão é representada pelo objeto `StreamConnection` gerado do método `Connector.open()`, que recebe um texto com a URL desejada. Os conhecidos objetos `InputStream` e `OutputStream` são usados para ler e escrever dados nessa conexão, e o método `flush()` para envio dos dados, a Figura 22 mostra os métodos de uma conexão.

```
//criação da url por meio do objeto ServiceRecord obtido
String url = getRecord().getConnectionURL(
ServiceRecord.NOAUTHENTICATE_NOENCRYPT,false);
//criação da conexão, representado pelo StreamConnection
conexao = (StreamConnection) Connector.open(url);

//envio e recebimento
//objeto de saída
private DataOutputStream saida;
//objeto de entrada
private DataInputStream entrada;

//método para enviar
public void enviar(String texto){
    if (saida != null){
        try {
            saida.writeStringt(texto);
            saida.flush();
        } catch (IOException ex) {
            cancelar();
        }
    }
}

//método para receber
public String receber() throws IOException{
    return entrada.readString();
}
```

Figura 22 – Métodos de conexão

2.4. DESENVOLVIMENTO DE JOGOS

O desenvolvimento de jogos é uma mistura de produção de *software* com criação artística, o que torna seus processos de criação muito variados e extremamente flexíveis. O desenvolvedor de jogos encontra grandes desafios e muitas novidades tecnológicas a serem vencidas (JUNIOR; NASSU; JONACK, 2002).

Para Bargallo (2004), existem diversas formas de desenvolvimento para jogos, encontrando-se no mercado diversas opiniões sobre o assunto. O método de desenvolvimento depende do tipo de jogo, da equipe de desenvolvimento, dentre outros aspectos. Porém, há uma coesão para a maioria dos desenvolvedores, de que os jogos para dispositivos móveis são mais simples de se desenvolver do que os jogos para computador pessoal. Em alguns momentos apenas um programador e um designer são necessários para a produção de um título para dispositivo móvel.

Na maioria das vezes, o desenvolvimento de um jogo para computador pessoal é realizado no período de 18 a 38 meses, já um projeto para dispositivo móvel leva de 1 a 3 meses para ser desenvolvido (BARGALLO, 2004).

Paludo (2003) afirma que, no desenvolvimento de um jogo para dispositivos móveis, devem ser analisadas diversas etapas: o jogo propriamente dito, como a definição da história, personagens, características do jogo, objetivo, regras, a interfaces, sons, gráficos; as características de *hardware*, que se diferenciam em muitos aspectos, como tamanho do visor, memória, fabricante, número de cores suportadas, além da capacidade de processamento; as características de *software* como a plataforma de desenvolvimento e o sistema operacional.

Um jogo eletrônico é uma aplicação interativa voltada ao entretenimento. É um produto de *software*, e como tal passa por diversas fases durante o seu desenvolvimento (JUNIOR; NASSU; JONACK, 2002).

De acordo com Junior, Nassu e Jonack (2002), o processo de desenvolvimento de jogos é similar ao processo de desenvolvimento de um *software* qualquer, seguindo as fases da engenharia de *software*, como análise, projeto, implementação e teste, ou fases análogas a estas. Porém devido ao jogo ser bastante subjetivo, especialmente em requisitos a serem satisfeitos, surge à necessidade dos processos de desenvolvimento não serem tão rígidos. Normalmente no processo de desenvolvimento não são seguidas às fases em uma ordem única, elas se entrelaçam durante todo o andamento do desenvolvimento, com algumas assumindo um papel mais significativo em certos momentos.

A engenharia de *software* é o estabelecimento de uso de sólidos princípios de engenharia para que se possa obter economicamente um *software* que seja confiável e que funcione eficientemente em máquinas reais (PRESSMAN, 1995).

2.4.1. Tipos de Jogos

Segundo Battaiola et al (2001), antes de se definir os tipos de jogos, existe a necessidade de se entender as formas de atuação do usuário com o jogo, que pode ser em terceira pessoa (descrita na literatura como *God's Eye* -olhar de Deus), usada geralmente em narrações, é uma visão superior de todo o sistema. Por exemplo, os jogos de luta, onde o usuário vê o jogo e seu personagem. Outra atuação, é a em primeira pessoa (a cena exibida representa o que os olhos do usuário vêem), como em jogos de tiro, que o jogador caminha em cenários atirando em inimigos.

2.4.1.1. Tipos de Jogos em Função da Interface

Quando a interface é em terceira pessoa, o personagem que representa o usuário na cena é denominado de ator, o personagem é parte integrante da cena e deve ser bem caracterizado, exigindo um visual sofisticado e bem elaborado. Quando é em primeira pessoa, o personagem é denominado de *avatar*, o personagem simplesmente marca o usuário na cena e a sua sofisticação deve ir até o limite em que não prejudique a ilusão de imersão do usuário no jogo (BATTAIOLA et al, 2001).

2.4.1.2. Tipos de Jogos em Função do Objetivo

Battaiola et al (2001) afirma que existem vários tipos de jogos segundo os objetivos, porém atualmente existe uma tendência de combinar as características de cada um, para gerar jogos diferentes. Os principais tipos segundo o objetivo são:

Estratégia: jogos idealizados com o objetivo do usuário gerencia recursos a fim de conquistar objetivos usando estratégias e táticas;

Simuladores: jogos que requerem reflexo do usuário que está imerso em um ambiente que tenta retratar a realidade;

Aventura: jogos que combinam ações baseadas em raciocínio e reflexo onde o jogador deve ultrapassar estágios que envolvam a solução de enigmas para chegar ao final;

Infantil: jogos que enfocam quebra-cabeças educativos ou histórias simples com o objetivo de divertir a criança;

Passatempo: jogos simples, como quebra-cabeças rápidos e sem nenhuma história relacionada, cujo objetivo essencial é atingir uma pontuação alta;

RPG (Role Playing Game): versão computadorizada do RPG convencional, onde o jogador, assumindo uma personalidade, enfrenta situações para ganhar experiência;

Esporte: jogos de habilidade em esportes populares, como os jogos de futebol, futebol americano, vôlei, basquete, boxe, baseball, etc.;

Educação/Treinamento: Jogos com fins educacionais que podem envolver as características de qualquer um dos jogos anteriores.

(BATTAIOLA et al, 2001)

2.4.1.3. Tipos de Jogos em Função do Número de Usuários

De acordo com Battaiola et al (2001), os jogos podem ser classificados em função do número de jogadores em monousuário e múltiplos usuários, onde no monousuário apenas um usuário participa do jogo, enquanto que no múltiplos usuários vários usuários participam do jogo, podendo estar geograficamente separados, interligados apenas por uma rede de longa distância.

2.4.2. Antes do Desenvolvimento

Junior, Nassu e Jonack (2002) argumentam que, antes de iniciar o processo de criação do jogo propriamente dito, há a necessidade de se definir alguns pontos exteriores ao desenvolvimento, mas extremamente importantes: concepção inicial, os publicadores e distribuidores, a procura por investidores, análise do público-alvo e a escolha da plataforma sobre a qual o jogo irá ser executado.

O primeiro passo para a criação do jogo é a concepção inicial, que é a idéia básica por trás do jogo. Outras definições necessárias são as figuras do publicador (responsável pela replicação em larga escala do software, pela impressão dos manuais e caixas, ou seja, pela transformação do jogo em um produto), do distribuidor (responsável pela comercialização e distribuição do jogo e também por parte da publicidade), do investidor (responsável pelo financiamento do projeto) e do público alvo (usuários do jogo) (JUNIOR; NASSU; JONACK, 2002).

2.4.3. Desenvolvimento

No início, o desenvolvimento de jogos era relativamente simples, por causa da limitação imposta pelo hardware. Forçava os desenvolvedores a se preocuparem com a eficiência na execução do jogo e com o tamanho do código gerado. O código tinha de ser bastante eficiente para que as máquinas rodassem o jogo de forma satisfatória, além de que o tamanho do código não poderia ser muito grande ou ele não caberia na memória disponível no computador (BATTAIOLA et al, 2001).

Junior, Nassu, Jonack (2002) argumentam que, o desenvolvimento de um jogo é realizado em sete fases, designe (análise), projeto, cronograma e orçamento, implementação, codificação, teste e manutenção.

2.5. JOGO DA VELHA

O jogo da velha é jogado por dois participantes, podendo ser jogador humano versus jogador humano ou jogador humano versus jogador artificial (celular). O jogo é realizado em turnos alternados entre dois jogadores. O jogo ocorre em um tabuleiro de 3x3, como ilustrado na Figura 23, posições no quais os jogadores vão selecionando suas posições.

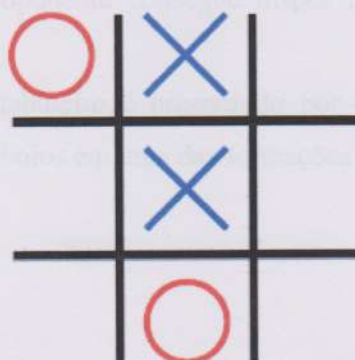


Figura 23 – Tabuleiro do jogo da velha

Os jogadores utilizam os símbolos “X” e “O”, um para cada. O jogador pode colocar um símbolo em qualquer lugar do tabuleiro em sua vez, desde que não esteja marcado. Quando for selecionado o local a vez passa para o outro jogador, que realiza o

mesmo procedimento até que a partida seja vencida, ou o tabuleiro seja completamente preenchido, situação na qual ocorre empate.

O jogador vencerá quando conseguir colocar três símbolos em seqüência, seja em linha, coluna ou nas diagonais do tabuleiro, como demonstrado na Figura 24. O objetivo dos jogadores, então, é colocar três marcas numa das configurações válidas, enquanto evitam que o seu oponente consiga fazer as três marcas antes dele.

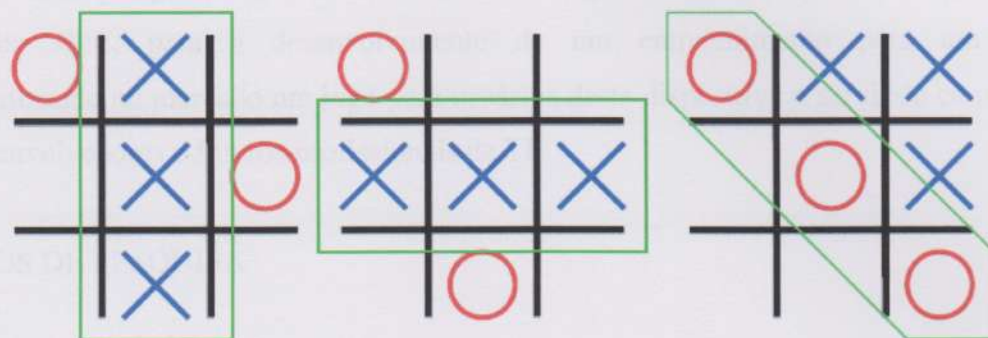


Figura 24 – Exemplo de configurações válidas para vitória

Critérios de avaliação:

- Vitória: o jogador consegue dispor três símbolos em uma das formações válidas.
- Derrota: o oponente consegue dispor três símbolos em uma das formações válidas.
- Empate: o tabuleiro é preenchido por completo e nenhum dos participantes conseguiu dispor três símbolos em uma das formações válidas.

3. METODOLOGIA

O presente trabalho trata-se de uma pesquisa aplicada, que segundo Andrade (1998), “é uma pesquisa que visa aplicações práticas, com o objetivo de atender às exigências da vida moderna, sendo o objetivo contribuir para fins práticos pela busca de soluções para problemas concretos”.

Esta pesquisa visa a utilização de tecnologias móveis, rede sem fio *bluetooth* e a plataforma JME, para o desenvolvimento de um entretenimento para um celular, disponibilizando no mercado um jogo para usuários deste dispositivo e servindo como acervo para desenvolvedores e futuros profissionais de TI.

3.1. TIPOS DE PESQUISA

3.1.2. Quanto aos Objetivos

A pesquisa foi classificada quanto aos objetivos como exploratória, que de acordo com Andrade (1998) é o primeiro passo a ser feito em todo trabalho científico, tendo como finalidades proporcionar maiores informações sobre o assunto; facilitar a delimitação do tema; definir os objetivos ou formular as hipóteses de uma pesquisa ou descobrir novo tipo de enfoque para o trabalho que se tem em mente. “Portanto, a pesquisa exploratória, na maioria dos casos, constitui um trabalho preliminar ou preparatório” (ANDRADE, 1998).

3.1.3. Quanto aos Procedimentos

O trabalho trata-se de uma pesquisa bibliográfica com intuito de obter dados suficientes para a realização do projeto. Pesquisa bibliográfica tem como finalidade “colocar o pesquisador em contato direto com tudo o que foi escrito, dito ou filmado sobre determinado assunto” (LAKATOS; MARCONI, 1991).

Segundo Lakatos e Marconi (1991), a pesquisa bibliográfica abrange toda a bibliografia já tornada pública, referente ao assunto de estudo, como publicações avulsas, boletins, jornais, revistas, livros, pesquisas, monografias, teses, etc., até meios de comunicação orais.

4. RESULTADOS

Como resultado da pesquisa foi desenvolvido um protótipo do jogo da velha, utilizando a plataforma *Java Micro Edition* (JME) e a API Java *bluetooth* (JSR-82). Como ferramentas de desenvolvimento utilizou-se a IDE *netbeans* 5.0 com o *netbeans mobile pack* 5.0, a máquina virtual Java 5.0 e o Java Wireless Toolkit 2.5, ambos da *Sun Microsystem*. Devido à necessidade de se utilizar desenhos para realização do jogo, foi utilizado o *Canvas*, para desenhar o tabuleiro, as marcações das jogadas e demais marcações do jogo (vez do jogador, vitórias e derrotas).

A aplicação foi desenvolvida em uma arquitetura na qual está dividida em camadas (controlador, modelo ou serviço de comunicação - enlace), por exemplo, possibilitando que seja adicionada uma outra forma de conexão sem necessidade de se refazer toda a aplicação, demonstrado na Figura 25. Para se modificar a aplicação basta alterar as classes de cada pacote, sem comprometer as outras classes. O diagrama da figura 36 ilustra como as classes da aplicação se relacionam.

Todo o código utiliza APIs básicas para atender à portabilidade (poder ser utilizados em diversos dispositivos, de diversos fabricantes). Está distribuído em nove classes (Ponto, PontoMatriz, Jogo, Partida, BluetoothConexao, Cliente, Servidor, VelhaMidlet, JogoDaVelhaCanvas) que estão divididas em três pacotes (jogo, *bluetooth* e apresentacao), o diagrama da aplicação é ilustrado na Figura 26:

- Pacote jogo: Estão as classes responsáveis pela lógica do jogo (Ponto, PontoMatriz, Jogo, Partida);
- Pacote *bluetooth*: Estão as classes responsáveis pela conexão *bluetooth* (BluetoothConexao, Cliente, Servidor);
- Pacote apresentacao: Estão as classes responsáveis pela apresentação da aplicação (o que aparece para o usuário - VelhaMidlet, JogoDaVelhaCanvas).

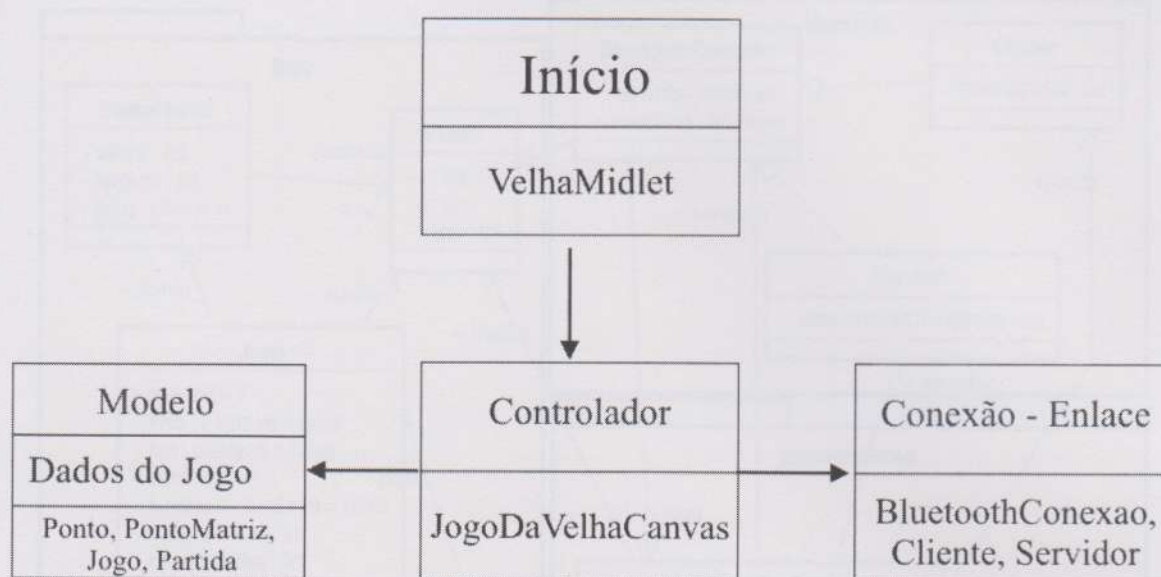


Figura 25 – Camadas da aplicação

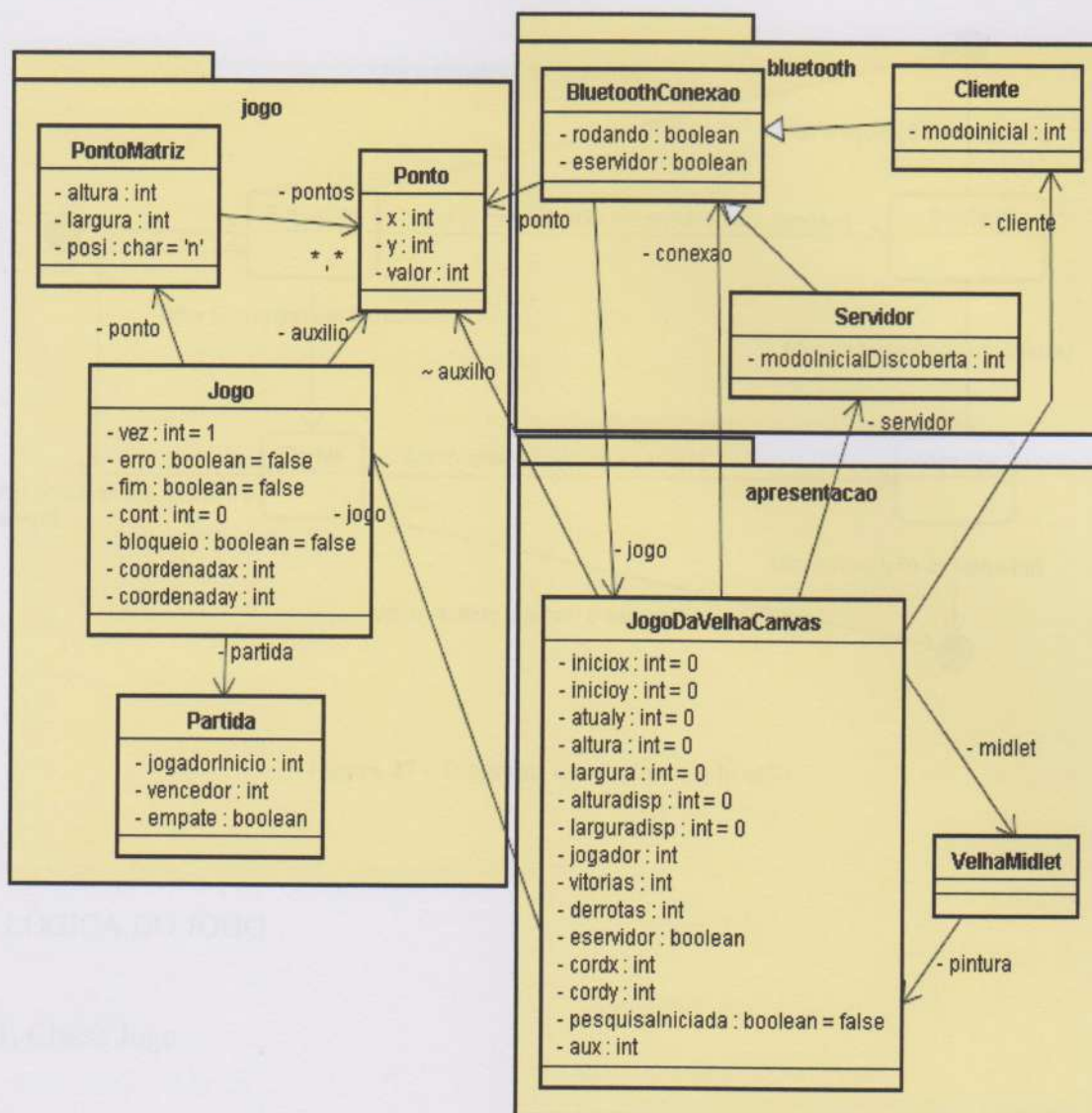


Figura 26 – Diagrama da aplicação

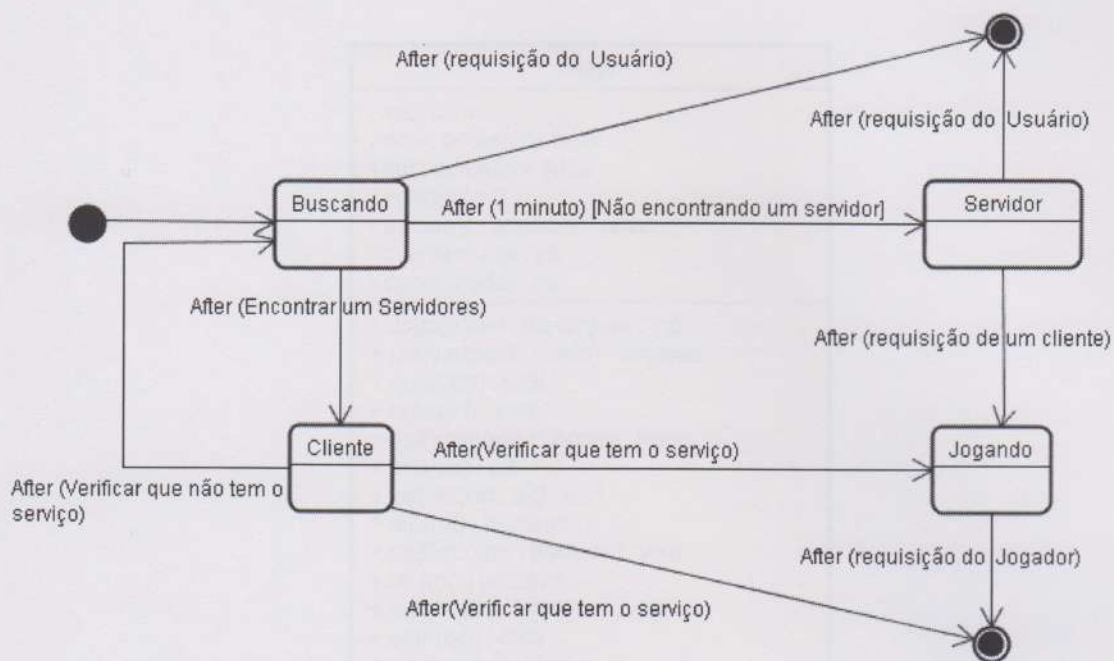


Figura 27 – Diagrama de estado da aplicação

4.1. LÓGICA DO JOGO

4.1.1. Classe Jogo

Classe responsável pelo gerenciamento do jogo, que é demonstrada na Figura 28:

- Controla a vez do jogador, fazendo a mudança da vez, após receber o token;
- Limpa as jogadas para que o jogo seja reiniciado;
- Monta a matriz com as coordenadas, chamando a classe PontoMatriz;
- Controla as partidas (fim e início).

Jogo
- vez : int = 1 - erro : boolean = false - fim : boolean = false - cont : int = 0 - bloqueio : boolean = false - coordenadax : int - coordenaday : int
+ Jogo(altura : int, largura : int) + marcar(pont : Ponto) : boolean + modVez() : void + resetar() : void + getPonto(pont : Ponto) : Ponto + getVez() : int + setVez(vez : int) : void + isErro() : boolean + setErro(erro : boolean) : void + isFim() : boolean + setFim(fim : boolean) : void + getPosi() : char + getPartida() : Partida + setPartida(partida : Partida) : void + isBloqueio() : boolean + setBloqueio(bloqueio : boolean) : void + getCoordenadax() : int + setCoordenadax(coordenadax : int) : void + getCoordenaday() : int + setCoordenaday(coordenaday : int) : void

Figura 28 – Classe Jogo

4.1.2. Classe Partida

Classe que representa a partida, armazena o jogador de início (que inicia a jogada), se a partida foi empate ou quem foi o vencedor, para o controlador utilizar as informações com o intuito de exibi-las na tela, demonstrada a Figura 29.

Partida
- jogadorInicio : int - vencedor : int - empate : boolean
+ Partida() + getJogadorInicio() : int + setJogadorInicio(jogadorInicio : int) : void + getVencedor() : int + setVencedor(vencedor : int) : void + isEmpate() : boolean + setEmpate(empate : boolean) : void

Figura 29 – Classe Partida

4.1.3. Classe Pontomatriz

Classe que representa o tabuleiro e possui a matriz onde estão as marcações do jogo, mostrada na Figura 30. Armazena a altura e largura do dispositivo para montar uma matriz própria para cada dispositivo, possibilitando a portabilidade e a compatibilidade (entre dois dispositivos distintos e de tamanho de telas diferentes). Armazena em que posição foi a vitória, para que sejam marcados na tela os pontos que a possibilitaram.

PontoMatriz
- altura : int - largura : int - posi : char = 'n'
+ PontoMatriz(altura : int, largura : int) + restart() : void + getPonto(ponto : Ponto) : Ponto + setPontos(ponto : Ponto) : void + elivre(ponto : Ponto) : boolean + getAltura() : int + setAltura(altura : int) : void + getLargura() : int + setLargura(largura : int) : void + eVitoria(pont : Ponto) : boolean + getPosi() : char + setPosi(posi : char) : void

Figura 30 – Classe PontoMatriz

4.1.4. Classe Ponto

Classe que representa o ponto da matriz, que é exibida na Figura 31, contendo as coordenadas x e y do ponto e, caso o mesmo esteja marcado, contém o jogador que o selecionou.

Ponto
- x : int - y : int - valor : int
+ Ponto() + getX() : int + setX(x : int) : void + getY() : int + setY(y : int) : void + getValor() : int + setValor(valor : int) : void

Figura 31 – Classe Ponto

4.2. MIDLET DA APLICAÇÃO

4.2.1. Classe Velhamidlet

É a *MIDlet* da aplicação. Contém dois objetos *List*, um para iniciar a aplicação e o outro para exibição da lista dos dispositivos *bluetooth* encontrados. Possui também um objeto *Alert*, através do qual exibe os alertas ao usuário, e um objeto *Ticker* para exibir algumas informações. É responsável pela comunicação da aplicação com o gerenciador de aplicativos do dispositivo.

4.2.1.1 Ações do Ciclo de Vida

PAUSA:

Os objetos que estão em uso são colocados em pausa, ficando, deste modo, a maioria dos recursos utilizados liberados para que outras operações possam ser realizadas.

ATIVA:

Quando a *MIDlet* é iniciada é chamado o método `startApp()`, que chama o método `initialize()`, como é demonstrado na Figura 32, onde é definido como *Screen* da aplicação a lista menu, com a opção de iniciar o jogo.

```
public void startApp() {
    initialize();
}

private void initialize() {
    getDisplay().setCurrent(get_menu());
}
```

Figura 32 – Código da inicialização da *MIDlet*

DESTRUÍDA:

Quando a *MIDlet* é destruída, todos os recursos utilizados são liberados, sendo fechada a conexão existente, como mostra a Figura 33.

```
public void destroyApp(boolean unconditional) {
    if (unconditional==true){
        if(pintura.getConexao()!=null){
            //cancelando conexão
            pintura.getConexao().cancelar();
        }
    }
}
```

Figura 33 – Método `destroyApp()`

Quando o jogo é iniciado através do *list* menu, selecionando iniciar, a *Screen* da aplicação passa a ser a *ListaDispositivos*, sendo criado um objeto *JogadaVelhaCanvas*

(controlador da aplicação), iniciando-se a busca por dispositivos, por intermédio do *JogoDaVelhaCanvas*, o código é ilustrado na Figura 34.

```

}else if (command == menu.SELECT_COMMAND) {
    switch (get_menu().getSelectedIndex()) {
        case 0:
            //torna o display como sendo a lista
            getDisplay().setCurrent(get_listaDispositivos());
            // Criando o objeto JogoDaVelhaCanvas
            pintura = new JogoDaVelhaCanvas(this);
            pintura.iniciaBusca();
            break;
    }
}

```

Figura 34 – Iniciando a busca por dispositivos

4.3. CLASSE JOGODAVELHACANVAS

Classe responsável pelo controle da aplicação, por disponibilizar o serviço *bluetooth* ou buscá-lo, bem como criar as conexões. Gerencia todo o jogo e é responsável pela aparência do jogo (tabuleiro, marcações e as exibições para o usuário). A classe *JogoDaVelhaCanvas* estende a classe *Canvas* e implementa *CommandListener* (para gerenciamento dos comandos) e *DiscoveryListener* (para o processo de busca por dispositivos e gerenciamento do dispositivo *bluetooth*), como mostra a Figura 35.

```

public class JogoDaVelhaCanvas extends Canvas implements
CommandListener, DiscoveryListener {
    /**
     * .....
     */
}

```

Figura 35 – Exemplo da classe *JogoDaVelhaCanvas*

4.3.1. Utilização de Java 2d

A classe utiliza figuras geométricas básicas para desenhar na tela (círculos, retângulos e retas) e desenhos de textos para mostrar informações ao usuário, os métodos utilizados são demonstrados na Tabela 9 e o código que desenha o tabuleiro na Figura 36.

Códigos Java 2D utilizados	
Método	Descrição
setColor()	Utilizado para definir a cor
fillRect()	Utilizado para limpar a tela e desenhar o tabuleiro
fillArc()	Utilizado para desenhar as peças dos jogadores
drawRect()	Utilizado para desenhar o marcador de posição e de jogada errada e o marcador das posições que geraram a vitória.
drawString()	Utilizado para desenhar mensagens na tela, mostrar o jogador e as vitórias e derrotas
repaint()	Utilizado para pintar a tela

Tabela 9 – Métodos do Java 2D utilizados na JogoDaVelhaCanvas

```
//tabuleiro do jogo
//limpa toda a tela (branco)
g.setColor(255,255,255);
//limpa a tela com um retângulo branco
g.fillRect(0,0,getWidth(),getHeight());
//pinta a tela de verde
g.setColor(100,160,80);
g.fillRect(0,0,getWidth(),alturadisp);
g.setColor(0,0,255);
//linhas horizontais
g.fillRect(0,altura,larguradisp,2);
g.fillRect(0,2*altura,larguradisp,2);
//linhas verticais
g.fillRect(largura,0, 2, alturadisp);
g.fillRect(2*largura,0, 2, alturadisp);
g.setColor(0,0,0);
//fim do tabuleiro
```

Figura 36 – Código que desenha o tabuleiro na JogoDaVelhaCanvas

Para desenhar o tabuleiro, primeiramente, foi desenhado um retângulo preenchido, do tamanho da tela, na cor branca, com a finalidade de limpar a tela, que será preenchida, posteriormente, por um retângulo na cor desejada para o tabuleiro, menor um pouco na altura, possibilitando um local para a exibição das informações (jogador, vez, vitórias, derrotas e situação do jogo), como mostra a Figura 37.

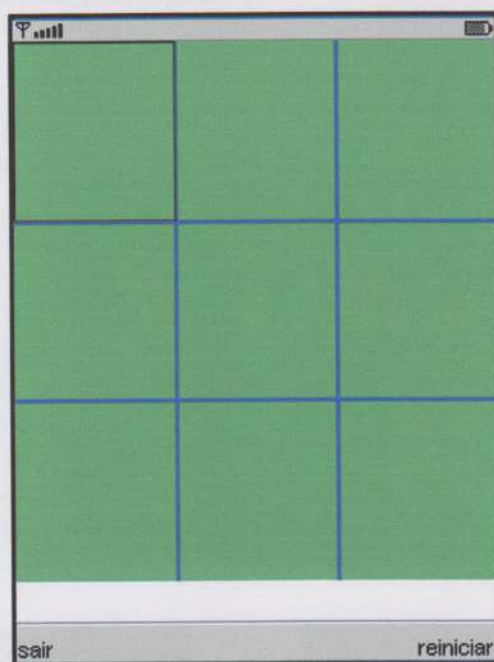


Figura 37 – Tabuleiro do jogo e o marcador de posição

No desenho das marcações dos jogadores, foram utilizados círculos preenchidos, na cor preta para jogador 1 (um) e na cor vermelha para a o jogador 2 (dois), onde o código é exemplificado na Figura 38 e é exibido o resultado na Figura 39.

```

//desenha as peças dos jogadores no tabuleiro
for (int i=0;i<3;i++){
    for (int j=0;j<3;j++){
        Ponto atual =new Ponto();
        atual.setX(i);
        atual.setY(j);
//verifica se qual a marcação do ponto
//se não for branco
        if(getJogo().getPonto(atual).getValor()!=0){
//se for marcado pelo jogador 1
            if(getJogo().getPonto(atual).getValor()==1){
//desenha um circulo preenchido na cor preta
                g.fillArc(getJogo().getPonto(atual).getX(),getJogo().
                    getPonto(atual).getY(),(largura/2),(largura/2),0,600);
            }
//se for marcado pelo jogador 2
            //desenha um circulo preenchido na cor preta
            if(getJogo().getPonto(atual).getValor()==2){
                g.setColor(255,0,0);
                g.fillArc(getJogo().getPonto(atual).getX(),getJogo().
                    GetPonto(atual).getY(),(largura/2),(largura/2),0,600);
                g.setColor(0,0,0);
            }
        }
    }
}

```

Figura 38 – Código que desenha as marcações no tabuleiro na JogoDaVelhaCanvas

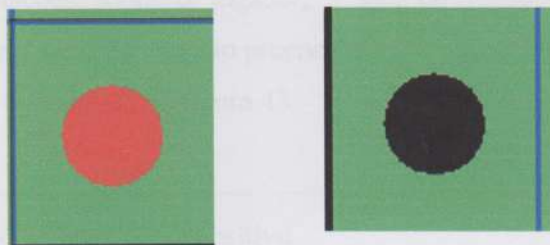


Figura 39 – Marcações dos jogadores

Na marcação da vitória é verificada a posição em que ocorreu e são desenhados três retângulos sem preenchimento nas posições correspondentes, o código está exposto na Figura 40 e o resultado exibido na Figura 41.

```

//desenhando a marcação da vitória
//definindo a cor
g.setColor(0,255,0);
//desenhando os quadros marcando as posições que de finiram a vitória
g.drawRect(0,altura*2,largura,altura);
g.drawRect(largura,altura,largura,altura);
g.drawRect(largura*2,0,largura,altura);
g.setColor(0,0,0);

```

Figura 40 – Desenhando a marcação da vitória

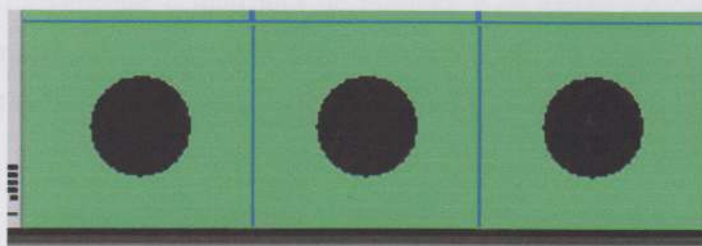


Figura 41 – Marcação da vitória

As informações sobre o jogador, a vez do jogador, vitórias e derrotas são desenhadas em texto no local da tela não preenchido pelo tabuleiro, corresponde ao código da Figura 42 e é exibido o resultado na figura 43.

```

//desenha o jogador do dispositivo
g.drawString(" j: "+getJogador(),0,alturadisp, Graphics.TOP|Graphics.LEFT);
//desenha o número de vitórias e derrotas
g.drawString(" v: "+vitorias+" d:" +derrotas,larguradisp,alturadisp,
Graphics.TOP|Graphics.RIGHT);
//desenha de quem é a vez
g.drawString(" vez jogador " +getJogo().getVez(),getWidth()/2,
alturadisp,Graphics.TOP|Graphics.HCENTER);

```

Figura 42 – Código da Marcação da vitória

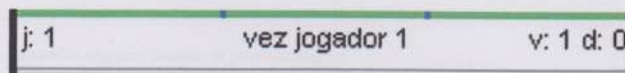


Figura 43 – Desenho das informações

4.3.2. Tratamento de Eventos

Durante toda a aplicação, a classe `JogoDaVelhaCanvas` trata diversos eventos, como a seleção de alguma tecla ou comando, e produz sons quando ocorrem determinados eventos (erro de marcação, vitória e empate). As teclas 2, 6, 8, 9, UP, DOWN, LEFT e RIGHT são tratadas para movimentar o marcador de posição, e as teclas 5 e SELECT realizam o acionamento da marcação, a Figura 44 demonstra o código do tratamento dos eventos de teclado, a Figura 45 o do tratamento dos comandos e a Figura 46 do tratamento de som.

```

* Chamado quando algum botão é pressionado
*/
protected void keyPressed(int keyCode) {
//leitura dos comandos para movimentação na tela
//verifica se está bloqueado
if(!getJogo().isBloqueio()){
//botão 6 ou RIGHT
if((keyCode == 54)|| (getGameAction(keyCode)==RIGHT)){
if(avalido((iniciox+largura),(inicioy))){
iniciox+=largura;
}

repaint();
//botão 4 ou LEFT
} else if((keyCode == 52)|| (getGameAction(keyCode)==LEFT)){
if(avalido((iniciox-largura),(inicioy))){
iniciox-=largura;
}

repaint();
//botão 8 ou DOWN
} else if((keyCode == 56)|| (getGameAction(keyCode)==DOWN)){
if(avalido((iniciox),(inicioy+altura))){
inicioy+=altura;
}

repaint();
//botão 2 ou UP
} else if((keyCode == 50)|| (getGameAction(keyCode)==UP)){
if(avalido((iniciox),(inicioy-altura))){
inicioy-=altura;
}

repaint();
//botão 9
} else if((keyCode == 57){
if(avalido((iniciox+largura),(inicioy+altura))){
inicioy+=altura;
iniciox+=largura;
}

repaint();
//botão 7
} else if((keyCode == 55){
if(avalido((iniciox-largura),(inicioy+altura))){
inicioy+=altura;
iniciox-=largura;
}

repaint();
//botão 3
} else if((keyCode == 51){
if(avalido((iniciox+largura),(inicioy-altura))){
inicioy-=altura;
iniciox+=largura;
}

repaint();
//botão 1
} else if((keyCode == 49){ //1
if(avalido((iniciox-largura),(inicioy-altura))){
inicioy-=altura;
iniciox-=largura;
}

repaint();
//botão 5 ou select
} else if((keyCode == 53)|| (getGameAction(keyCode)==FIRE)){
auxilio.setX(iniciox);
auxilio.setY(inicioy);
auxilio = cordenada(auxilio);
auxilio.setValor(getJogo().getVez());
conexao.enviar(auxilio.getX(),auxilio.getY(),getJogo().getVez());
getJogo().marcar(auxilio);

repaint();
}
}
}
}

```

Figura 44 – Tratamento dos eventos realizados nas teclas

```
public void commandAction(Command command, Displayable displayable) {  
    //se o comando for sair  
    if(command == cmExit){  
        //aciona o método exitMIDlet() do MIDlet  
        getMidlet().exitMIDlet();  
        //se o comando for reiniciar  
    }else if(command == jogarNo){  
        //resetar o jogo  
        getJogo().resetar();  
        //enviar padrão de reinicio  
        conexao.enviar(10,10,10);  
        //pintar  
        repaint();  
        setAux(0);  
    }  
}
```

Figura 45 – Tratamento dos comandos

```
//método de emissão de som  
public void som(int tipo){  
    switch (tipo){  
        case 1://erro  
            (new Thread(){  
                public void run(){  
                    try{  
                        AlertType.ERROR.playSound(getMidlet().getMostrador());  
                    }catch(Exception e){  
                        }  
                }  
            }).start();  
            break;  
        case 2://vitoria  
            (new Thread(){  
                public void run(){  
                    try{  
                        AlertType.INFO.playSound(getMidlet().getMostrador());  
                    }catch(Exception e){  
                        }  
                }  
            }).start();  
            break;  
        case 3://empate  
            (new Thread(){  
                public void run(){  
                    try{  
                        AlertType.CONFIRMATION.playSound(getMidlet().getMostrador());  
                    }catch(Exception e){  
                        }  
                }  
            }).start();  
            break;  
    }  
}
```

Figura 46 – Tratamento de eventos com som

4.4. COMUNICAÇÃO

Para a criação de uma comunicação *bluetooth* é necessário a existência de pelo menos um servidor e um cliente. No protótipo, para o usuário é abstraído o módulo no qual a aplicação está operando. Toda a aplicação ao iniciar trabalha no modo cliente, procurando possíveis dispositivos servidores. No início da comunicação pode ocorrer dois casos:

- O dispositivo encontra alguns dispositivos: Os dispositivos são exibidos em um objeto *list*, para que possa ser escolhido aquele com o qual se deseja realizar a verificação do serviço e a conexão;
- O dispositivo não encontra um servidor: O cliente é fechado e a aplicação passa a trabalhar no modo servidor, aguardando um cliente para conexão, sem que o usuário necessite interagir.

Na Figura é exibido o código do início da busca.

```

// cria o objeto controlador
pintura = new JogoDaVelhaCanvas(this);
//inicia a busca
pintura.iniciaBusca();

//método para início da busca dos dispositivos
public void iniciaBusca(){
    //removendo elementos do objeto list
    midlet.removeElementos();
    //exibe a informação de que a busca iniciou
    midlet.getTicker1().setString("Iniciando a busca...");
    try {
        //define o dispositivo blutthoth
        dispositivoLocal = LocalDevice.getLocalDevice();
        //define o DiscoveryAgent
        agente= dispositivoLocal.getDiscoveryAgent();
    } catch (BluetoothStateException ex) {
        ex.printStackTrace();
    }
    //define o vetor de retrieveDevices como nulo
    rd=null;
    //verifica se existe algum dispositivo armazenado em cache
    rd=agente.retrieveDevices(javax.bluetooth.DiscoveryAgent.CACHED);
    //verifica se não existe algum
    if(rd==null){
        //se não tiver dispositivos, verifica se existe algum dispositivo conhecido
        rd=agente.retrieveDevices(javax.bluetooth.DiscoveryAgent.PREKNOWN);
    }
    if((rd!=null)&&(rd.length > 0)){
        //adiciona os dispositivos na lista
        Midlet.addEle(rd);
    }else{
        //exibindo o texto informando que está executando a busca
        midlet.getTicker1().setString("Iniciando a busca...");
        pesquisaIniciada=true;
        //criando um vetor para armazenar os dispositivos remotos
        dispositivosRemotos = new Vector();
        try {
            //iniciando a busca
            agente.startInquiry(javax.bluetooth.DiscoveryAgent.GIAC,this);
        } catch (IOException ex) {
            //exibir alerta de erro de pesquisa
            midlet.mostrarAlerta("Erro no início da pesquisa!",AlertType.ERROR);
        }
    }
}

```

Figura 47 – Início da busca

4.4.1. Servidor sendo Encontrado

Quando um dispositivo é encontrado ele é inserido em um vetor para que, posteriormente, seja exibido ao jogador, que no jogo é realizado pelo código da Figura 48.

```
//método para adicionar os dispositivos descobertos que contém dispositivos bluetooth
// ativos
public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass) {
    if(!dispositivosRemotos.contains(remoteDevice)){
        //adicionado o dispositivo
        dispositivosRemotos.addElement(remoteDevice);
    }
}
```

Figura 48 – Método para adicionar os dispositivos encontrados

A busca pode ser terminada, se for cancelada, se ocorrer erro ou se for completa:

- Se for cancelada, é exibido um *Alert* com a informação de que a operação foi cancelada, como mostra o código da Figura 49;
- Se ocorrer um erro, é exibido um *Alert* com a informação de que ocorreu um erro na operação, como mostra o código da Figura 49;
- Se a busca for completa, o vetor com os dispositivos é exibido em um objeto *list*, para que o usuário selecione o dispositivo desejado, e ocorra a busca pelo serviço (o jogo da velha) , como mostra o código da Figura 50.

```
case INQUIRY_ERROR:
    //exibir o alerta
    midlet.mostrarAlerta("Ocorreu um erro durante a busca",AlertType.ERROR);
    break;

case INQUIRY_TERMINATED:
    //exibir o alerta
    midlet.mostrarAlerta("Busca cancelada",AlertType.INFO);
    break;
```

Figura 49 – Código do tratamento de erro da busca e de cancelamento

```

case INQUIRY_COMPLETED:
    //limpar a lista de dispositivos do midlet lista
    midlet.removeElementos();
    //verifica se existe dispositivos no vetor
    if(dispositivosRemotos.size() > 0){
        //exibindo o texto informando que a busca foi concluída e os dispositivos encontrados
        midlet.getTicker1().setString("Busca terminada...");
        //criando um array de RemoteDevice para receber os elementos do vetor de dispositivos
        rd=new RemoteDevice[dispositivosRemotos.size()];
        //copiando as informações dos dispositivos remotos para para o array de RemoteDevice
        dispositivosRemotos.copyInto(rd);
        //laço para percorrer o array de RemoteDevice
        for(int i=0;i<rd.length;i++){
            try{
                //adicionar os objetos devicelist.append(rd[i].getFriendlyName(false),icon);
                //adicionar os objetos na lista
                midlet.get_list1().append(rd[i].getFriendlyName(false),null);

            }catch(IOException ex){
                //exibir o alerta
                midlet.mostrarAlerta("Erro na obtenção do nome do dispositivo!",AlertType.ERROR);
                return;
            }
        }

    }else{
        //código para o erro
    }
    break;

```

Figura 50 – Código do tratamento de busca completa

A verificação do serviço é realizada quando o usuário seleciona um dos dispositivos da lista, executando o método para verificar se o dispositivo possui o UUID desejado, como mostra o código da Figura 51. A busca por serviço pode terminar concluída normalmente; não pôde ser concluída, pois o dispositivo está fora de alcance; terminar com erro; nenhum serviço for encontrado; foi cancelada. Quanto à busca de serviço pode observar as seguintes situações:

- Busca concluída normalmente: é criado um objeto Cliente da conexão, que estende a classe BluetoothConexao e implementa a classe *Runnable*, e inicia a comunicação.
- Busca não pôde ser concluída, pois o dispositivo está fora de alcance: é exibido um *Alert* para informar ao usuário que a busca não foi concluída.

- Busca terminou com erro: é exibido um *Alert* para informar ao usuário que ocorreu um erro.
- Busca terminou e nenhum serviço foi encontrado: é exibido um *Alert* para informar ao usuário que o dispositivo não possui o serviço.
- Busca foi cancelada: é exibido um *Alert* para informar ao usuário que a busca foi cancelada.

A Figura 52 mostra código de quando a busca é terminada normalmente, a Figura 53 exibe o código do início da conexão cliente e a Figura 54 o código do método executar da classe BluetoothConexao.

```
//método para verificar se o dispositivo tem o jogo rodando
public void verificarServico(int selec){
    //criando um vetor de UUIDs
    UUID[] uuidset = new UUID[]{new UUID(
        "00ABCDE000000000000000AF2006BCEF", false)};
    try{
        //busca pelo serviço
        agente.searchServices(null,uuidset,rd[selec],this);
    } catch(IOException ex){
        midlet.mostrarAlerta("Erro na busca por serviços",AlertType.INFO);
    }
}
```

Figura 51 – Código da verificação do serviço

```
case SERVICE_SEARCH_COMPLETED:
    setCliente(new Cliente());
    getCliente().setJogo(this);
    getCliente().setRecord(record);
    getCliente().run();
    break;
```

Figura 52 – Término da busca

```

//classe cliente da conexão Subclasse da BluetoothConexao
//iniciando o dispositivo local
dispositivoLocal= LocalDevice.getLocalDevice();
//armazenando o modo inicial do dispositivo
modoInicial = dispositivoLocal.getDiscoverable();
//definindo o modo do dispositivo para ser descoberto (no modo NOT_DISCOVERABLE,
// para que não seja descoberto)
dispositivoLocal.setDiscoverable(DiscoveryAgent.NOT_DISCOVERABLE);
String url = getRecord().getConnectionURL(
ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
conexao = (StreamConnection) Connector.open(url);
//método da classe mãe
executar(conexao);

```

Figura 53 – Início da conexão cliente

```

//Método executar da classe mãe (BluetoothConexao)
//método para executar a conexão
public void executar(StreamConnection conexao) throws IOException{
    try{
        //definindo o jogo como rodando
        setRodando(true);
        //iniciando o objeto de envio
        saida= conexao.openDataOutputStream();
        //iniciando o objeto de recebimento
        entrada = conexao.openDataInputStream();
        //chama dispositivo
        getJogo().iniciarJogo(eservidor,this);
        ponto = new Ponto();
        while(isRodando()){
            if (isRodando()){
                receber();
            }
        }
    } finally{
        desconectar();
    }
}

```

Figura 54 – Método executar da classe BluetoothConexao

4.4.2. Servidor não Encontrado

Quando nenhum dispositivo é encontrado, é criado um objeto Servidor, que estende a classe BluetoothConexao e implementa a classe *Runnable*, e a aplicação fica aguardando uma aplicação cliente para realizar a conexão, conforme código da Figura 55. Ao receber uma requisição de um cliente é executado o método executar da classe mãe (BluetoothConexao) e a comunicação é iniciada.

```
//Definindo o dispositivo local
dispositivoLocal = LocalDevice.getLocalDevice();
//armazenando o modo inicial do dispositivo, para no término do jogo voltar
// o dispositivo para o modo
modoInicialDescoberta = dispositivoLocal.getDiscoverable();
//definindo o modo do dispositivo para ser descoberto (no modo GIAC, para
// ser descoberto a qualquer hora)
dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
//registrando o serviço
notificador = (StreamConnectionNotifier) Connector.open(
"btspp://localhost:00ABCDE00000000000000AF2006BCEF");
//aguardando a conexão
conexao = notificador.acceptAndOpen();
//chamar o método executar da classe BluetoothConexao (classe pai)
executar(conexao);
```

Figura 55 – Disponibilização do serviço

4.5. SCREENSHORTS DA APLICAÇÃO

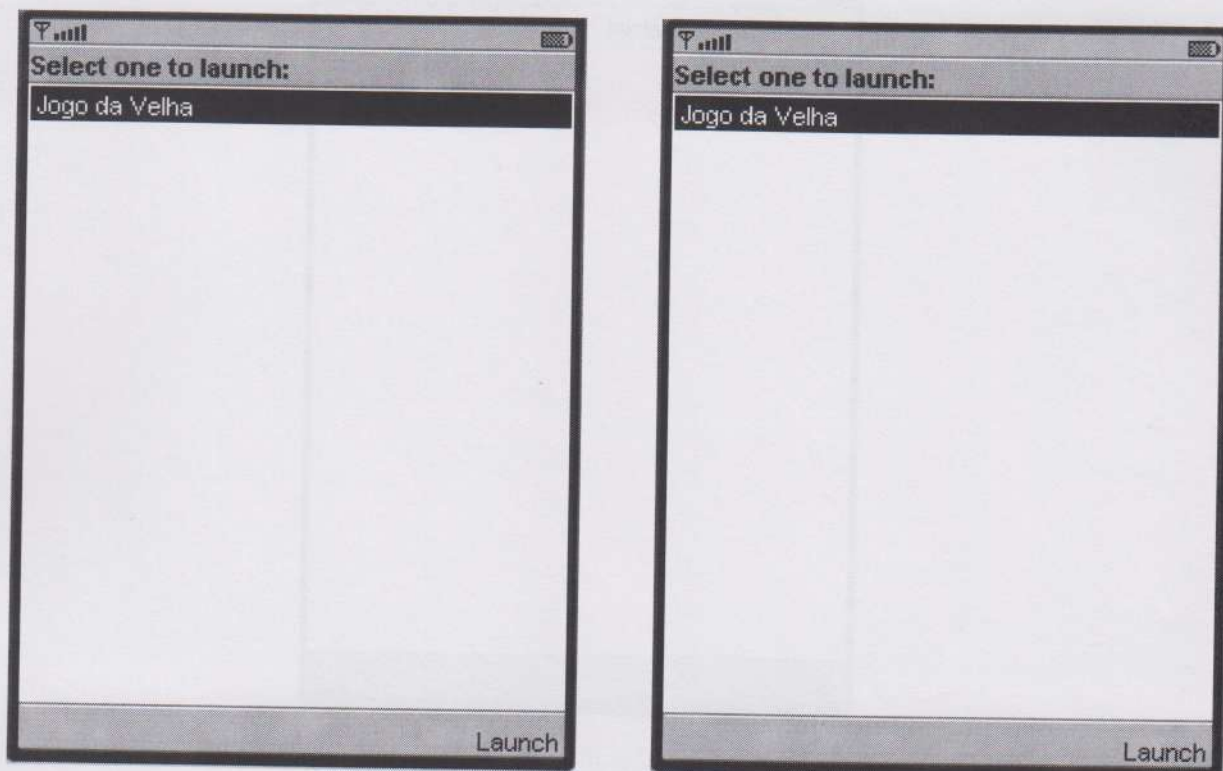


Figura 56 – Telas iniciais jogador 1 e Tela inicial do jogador 2

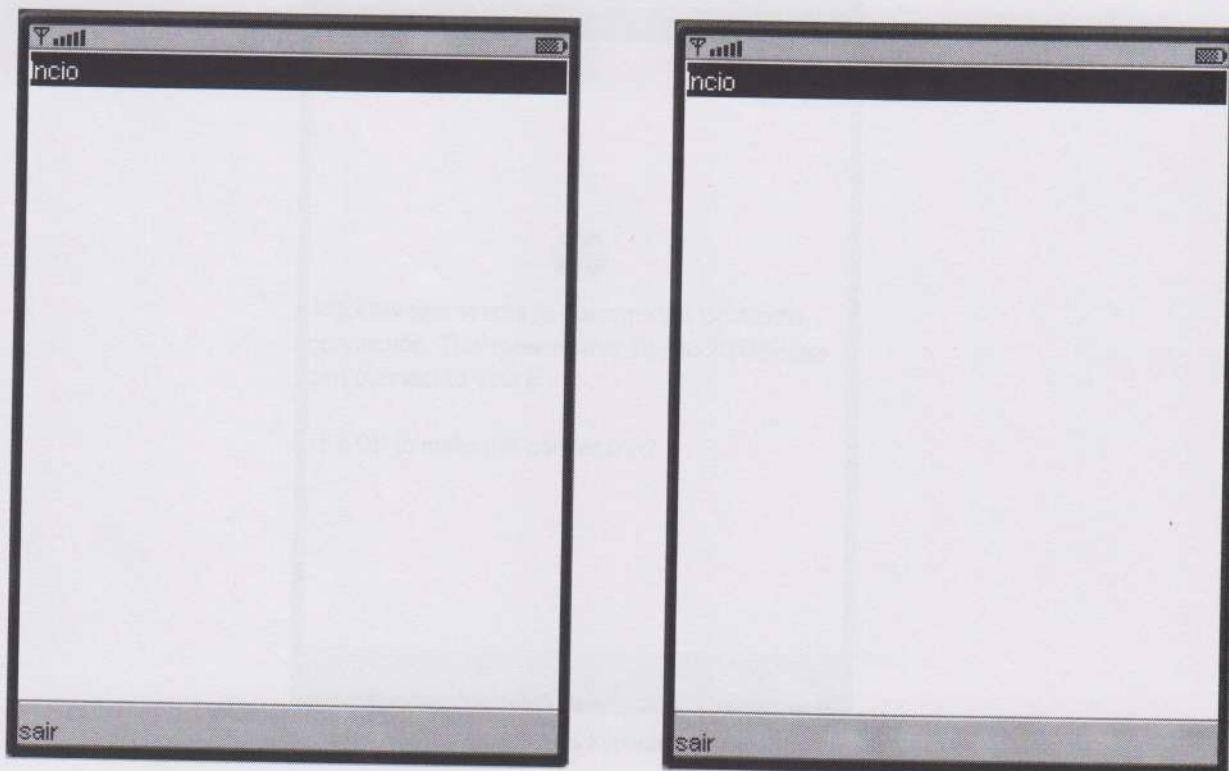


Figura 57 – Tela do menu do jogador 1 e do Jogador 2

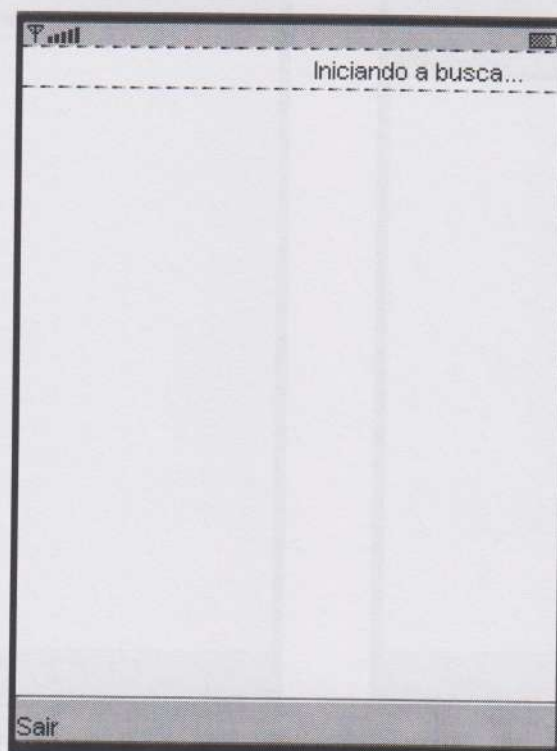


Figura 58 – Busca por dispositivos do Jogador 1

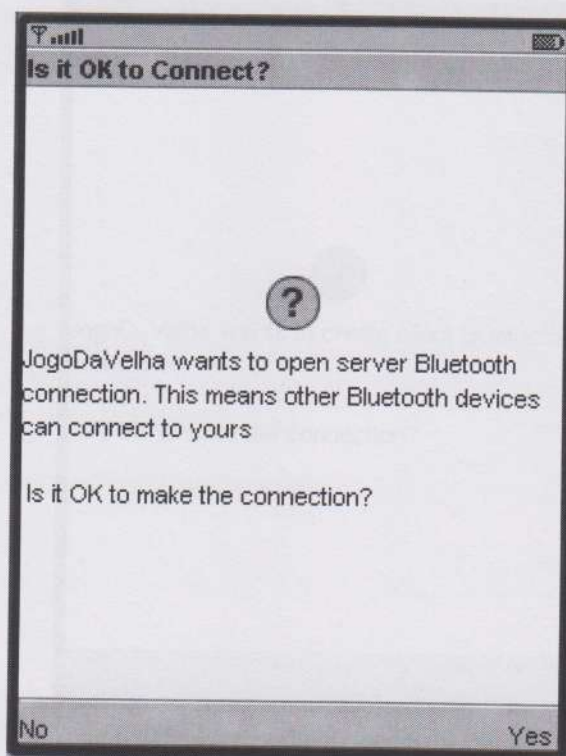


Figura 59 – Criação do Servidor no Jogador 1

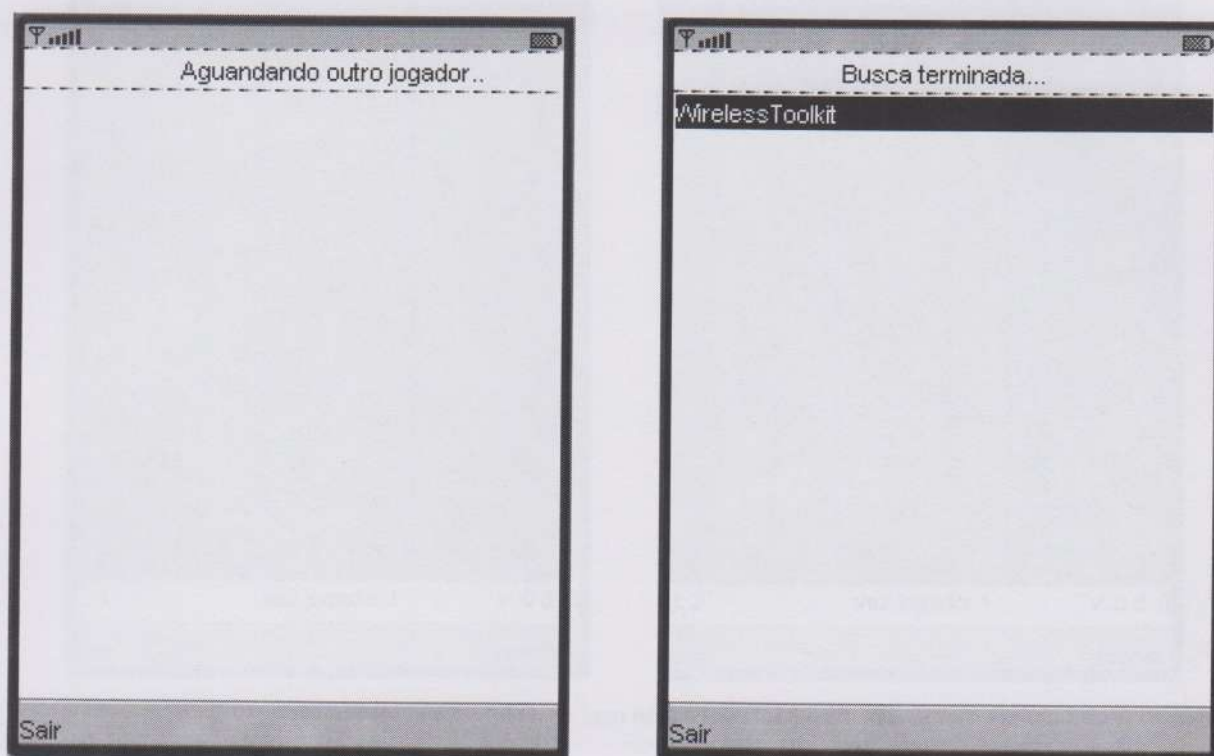


Figura 60 – Jogador 1 Aguardando um cliente para conexão e Servidor encontrado pelo Jogador 2

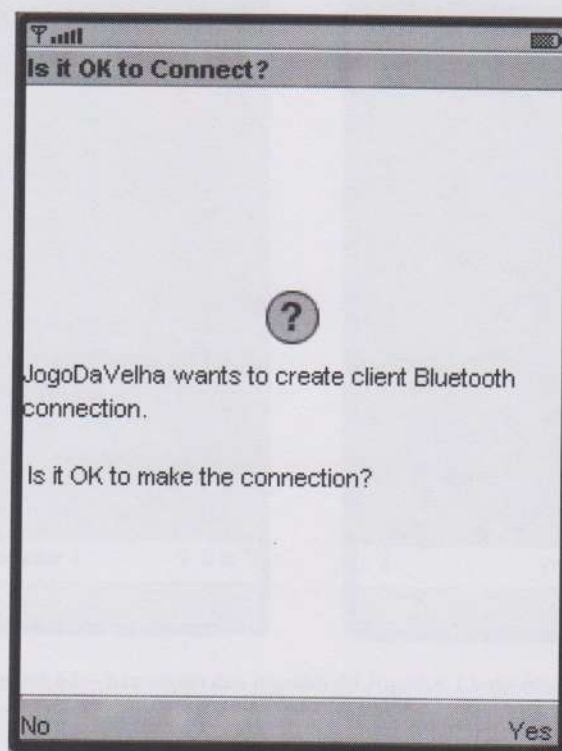


Figura 61 – Criando Cliente no Jogador 2

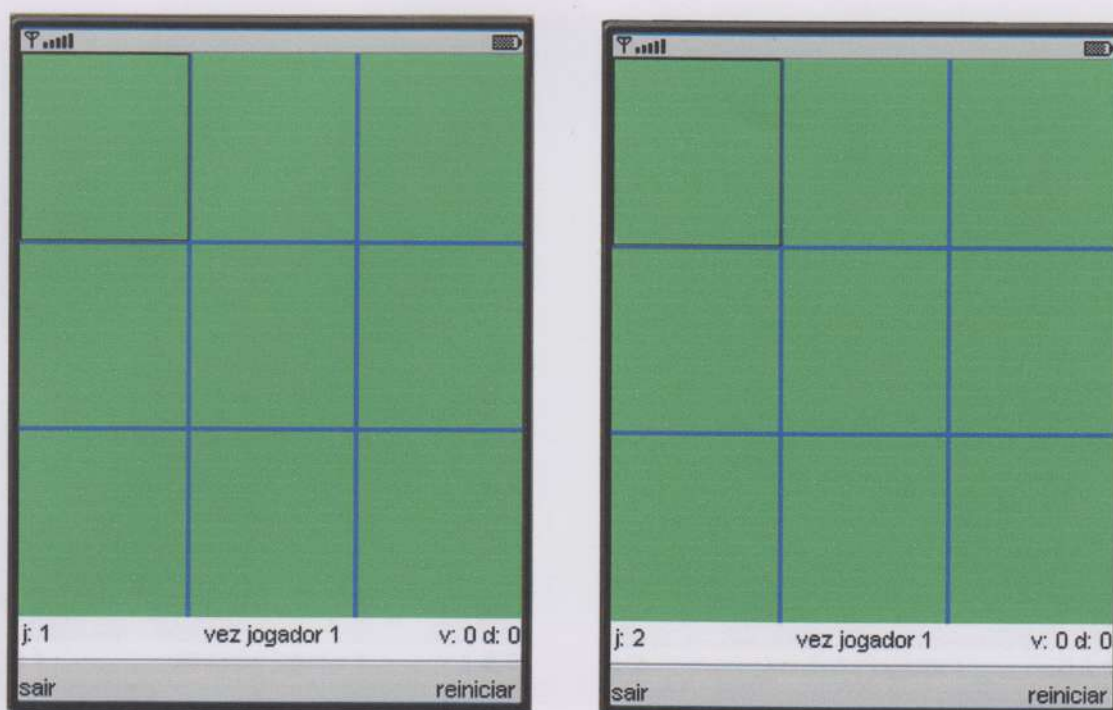


Figura 62 – Início do jogo para o Jogador 1 e Jogador 2

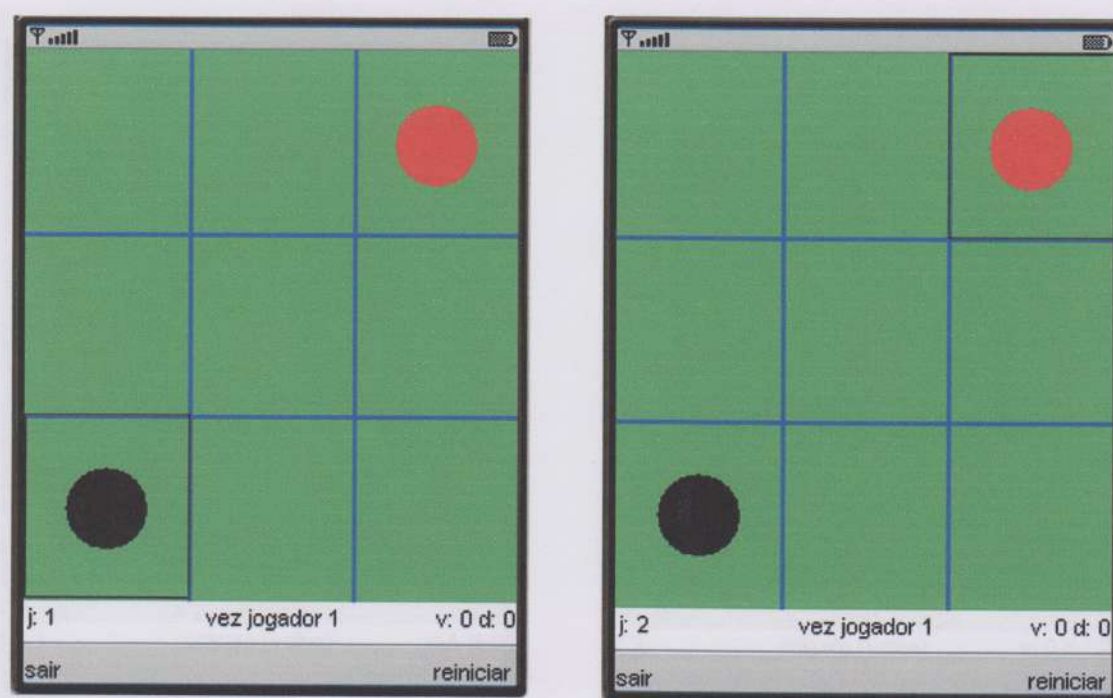


Figura 63 – Marcação das jogadas do Jogador 1 e do Jogador 2

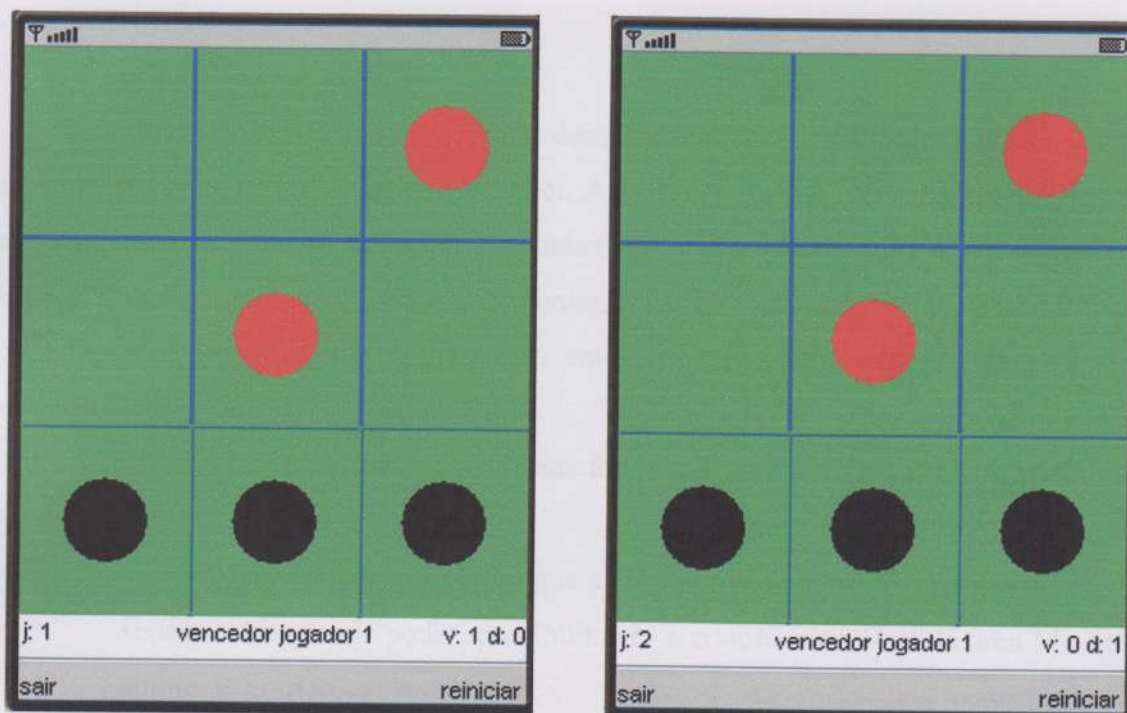


Figura 64 - Marcação da vitória do Jogador 1 no display dos Jogadores

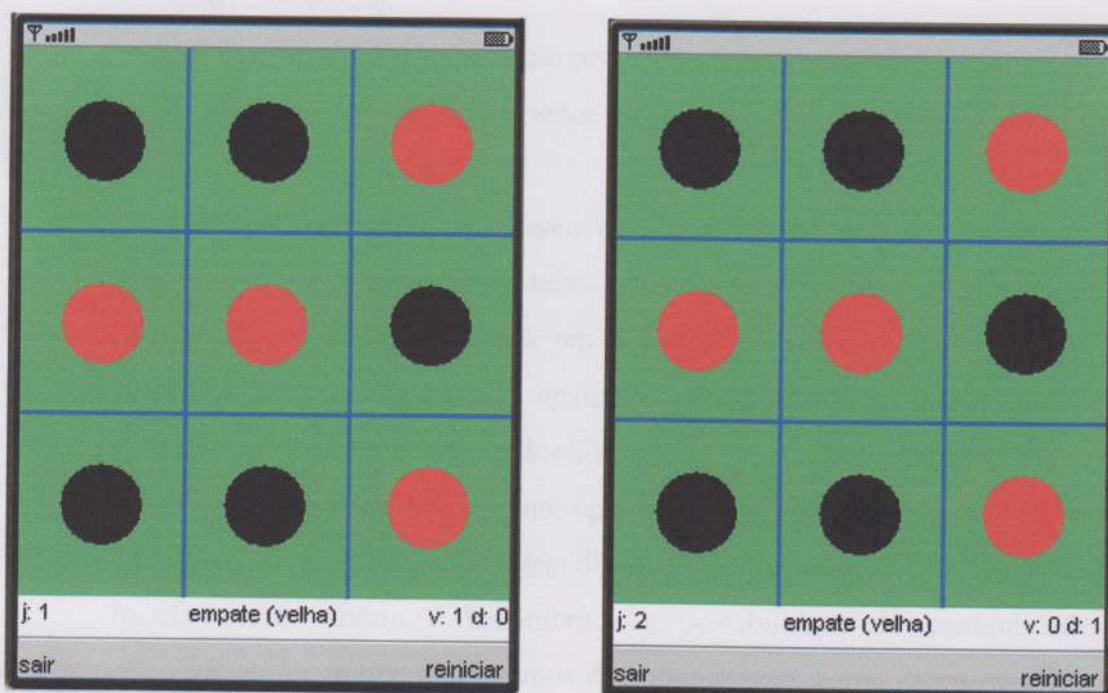


Figura 65 - Marcação do empate no display dos Jogadores

5. CONCLUSÃO

Com o uso da tecnologia JME o desenvolvedor consegue obter todas as vantagens da plataforma Java unida à tecnologia móvel. A JME demonstrou ser uma grande linguagem para o desenvolvimento de aplicações voltadas para dispositivos móveis, possibilitando um desenvolvimento simples e com uma abrangência de recursos, tendo possibilidade de portabilidade e de utilização de sistemas em dispositivos que não possuem opções de instalação de software.

Encontram-se no mercado diversas ferramentas para o desenvolvimento em JME, com destaque para:

- as IDE Netbeans e Eclipse, que se destacaram como ferramentas de utilização simples e de grande poder, possibilitando a criação mais fácil de uma MIDlet e de ambiente gráfico.
- O *Wireless Toolkit*, que emula os dispositivo móveis e seus acessórios, possibilitando a simulação da aplicação no computador para teste, emulando até o dispositivo *bluetooth*.

Porém durante o desenvolvimento problemas foram encontrados:

- A JME não tem suporte a todos os recursos dos dispositivos, limitando um pouco o desenvolvedor;
- Para obter portabilidade o desenvolvedor necessita ter uma atenção redobrada, pois fatores como tamanho de telas, o reconhecimento dos eventos das teclas e APIs suportadas são diferentes de um fabricante para outro.
- Operadoras configuram os aparelhos para permitirem apenas instalação de aplicações através de determinados sites (normalmente sites das operadoras).
- A API *bluetooth* é um pacote opcional, não estando disponível em todos os aparelhos, mesmo nos que possuem dispositivos *bluetooth*.

Imerso nesse cenário é vislumbrada a possibilidade de continuidade desta pesquisa: com a adoção de outros mecanismos de comunicação, como infravermelho, GPRS (*General Packet Radio Service*) e web com controlador; com o uso de outra interface gráfica, com adoção de tecnologia 3D (terceira dimensão); e colocação de áudio. Desta maneira, esta pesquisa servirá como base para pesquisas futuras e agirá como meio de divulgação das tecnologias Java, JME e *bluetooth*.

REFERÊNCIAS

- ANDRADE, Maria Margarida de. **Introdução à metodologia do trabalho científico**. 3. ed. São Paulo: Atlas, 1998.
- BALIEIRO, Silvia. Mais Recursos no alô. Revista Info Exame. Ano 20 n. 231. 2005
- BARBAGALLO, Ralph. **Wireless Game Development in Java with MIDP 2.0**. 1. ed. Texas: Wordware Publishing, Inc, 2004.
- BATTAIOLA, André Luiz et al. Desenvolvimento de Jogos em Computadores e Celulares. Revista de Informática Teórica e Aplicada. Volume 8 n. 2. 2002.
- BILLO, Eduardo Afonso. Uma pilha de protocolos *Bluetooth* adaptável à aplicação. Florianópolis:UFSC, 2003. **Monografia** (obtenção do grau de Bacharel em Ciência da Computação) - Universidade Federal de Santa Catarina. Florianópolis, 2003.
- BLUETOOTH.2003. Disponível em: <<http://www.gta.ufrj.br/~apaulo/seminarios/adhoc-e-bluetooth/bluetooth.html>>. Acesso em 17/junho/2006.
- BOURBON, Bruno Costa. Um *Framework* para desenvolvimento de aplicativos em Windows Móble. Recife:UFPE, 2005. **Monografia** (Graduação em Ciência da Computação) - Universidade Federal de Pernambuco. Recife, 2001.
- COMÉRCIO móvel atingirá US\$ 10 bi até 2010. 2006. Disponível em: <<http://www.decisionreport.com.br/publique/cgi/cgilua.exe/sys/start.htm?infoid=118&sid=20>>. Acesso em 17/maio/2006
- DEITEL, H. M.; DEITEL, P.J. **Como Programar Java**. 3. ed. Porto Alegre: Bookman, 2001.
- FALCO, Alessandra. Desenvolvimento de jogos para celular no Brasil. 2005. Disponível em: <<http://webinsider.uol.com.br/vernoticia.php/id/2429>>. Acesso em 17/maio/2006.
- FERNANDES, Manoel. O Celular GSM vai acabar. 2005. Disponível em: <<http://www.terra.com.br/istoedinheiro/410/entrevista/index.htm>>. Acesso em 17/maio/2006
- FERRARI, Bruno. Banco de bolso. 2006. Disponível em: <<http://www.decisionreport.com.br/publique/cgi/cgilua.exe/sys/start.htm?infoid=186&sid=1>>. Acesso em 17/maio/2006
- FRAMEWORK da Celepar aumenta produtividade no desenvolvimento de sistemas. 2006. Disponível em: <<http://www.pr.gov.br/batebyte/edicoes/2006/bb149/plataforma.shtml>>. Acesso em 11/jul/2006.
- GIARDINO, Andréa. Mais emprego para criadores de jogos para celulares. 2005. Disponível em: <http://www.universia.com.br/html/noticia/noticia_clipping_cgeje.html>. Acesso em 17/maio/2006

GOMES, João Paulo. Meantime Games: "Os nerds pernambucanos deram certo". 2005. Disponível em: <http://www.fircapital.com/ri/noticias/exibe_noticia.asp?idNoticia=16>. Acesso em 04/maio/2006

JODE, Martin et al. **Programming Java 2 Micro Edition on Symbian OS A developer's guide to MIDP 2.0**. 1. ed. West Sussex: John Wiley & Sons, Ltd, 2004.

JUNIOR, Ademar de Souza Reis; NASSU, Bogdan T.; JONACK, Marco Antonio. Um Estudo Sobre os Processos de Desenvolvimento de Jogos Eletrônicos (Games). Curitiba: UFPR, 2002. **Trabalho de disciplina** (Graduação em em Ciência da Computação)- Universidade Federal do Paraná.

LAKATOS, Eva Maria; MARCONI, Mariana de Andrade. **Fundamentos de metodologia científica**. 3. ed. São Paulo: Atlas, 1991.

LIMA, André Luiz de Sousa. Identificação e Exemplificação de Aspectos da Computação Distribuída no Desenvolvimento de Aplicações para Celulares Usando J2ME e *Bluetooth*. Recife: UFPE, 2005. **Monografia** (Graduação em em Ciência da Computação) - Universidade Federal de Pernambuco. Recife, 2005.

LIMA, Gleydson de Azevedo Ferreira; MEDEIROS, Ronaldo Maia de. A Tecnologia Bluetooth. Natal: UFRN, 2005. **Trabalho de disciplina** (Pós-Graduação em Engenharia Elétrica) - Universidade Federal do Rio Grande do Norte. Natal, 2005.

MUCHOW, John W. **Core J2ME Tecnologia & MIDP**. 1. ed. São Paulo: Markron Books, 2002.

NOKIA encerra 2005 com vendas líquidas de EU\$ 34,2 bilhões 31 de Janeiro de 2006. 2006. Disponível em: <<http://www.nokia.com.br/nokia/0,,87515,00.html>>. Acesso em 17/maio/2006.

PALUDO, Lauriana. Um Estudo Sobre as Tecnologias Java de Desenvolvimento de Aplicações Móveis. Florianópolis: UFSC, 2003. **Monografia** (Título de especialista em Ciência da Computação) - Universidade Federal de Santa Catarina. Florianópolis, 2003.

PÓVOA, Fábio Mesquita. Uma introdução à JSR-82 e alguns conceitos básicos sobre bluetooth. Revista Web Mobile. Edição 04. Ano I. 2005.

PRESSMAN, Roger S. **Engenharia de Software**. 1. ed. São Paulo: Markron Books, 1995.

TECNOLOGIA de transmissão móvel dá aos clientes acesso à programação da TV em qualquer hora, em qualquer lugar. 2006. Disponível em: <<http://www.philips.com.br/about/news/press/article-15577.html>>. Acesso em 17/maio/2006.

TOPLAY, Kim. **J2ME in a Nutshell**. 1. ed. O'Reilly, 2002.

WIKIPÉDIA. Tecnologia Móvel. 2006. Disponível em: <http://pt.wikipedia.org/wiki/Tecnologia_m%C3%B3vel>. Acesso em 09/maio/2006.