Desenvolvimento de Jogos para Celular usando Java ME: o Jogo Batalha Terrestre

Andrea Fernanda Fontes Bezerra
Centro Federal de Educação Tecnológica da Paraíba
Unidade Acadêmica de Informática
e-mail: websapiens@gmail.com

Fausto Véras Maranhão Ayres
Centro Federal de Educação Tecnológica da Paraíba
Unidade Acadêmica de Informática
e-mail: fausto.ayres@gmail.com

RESUMO: Com o avanço da tecnologia de telecomunicações entre dispositivos móveis e a redução dos custos de tais equipamentos, tornou-se popular a utilização da telefonia celular para transmissão de voz e para o entretenimento - em particular, para os jogos. O desenvolvimento de jogos para celular requer a utilização de uma plataforma específica de programação, tal como a plataforma *Java Microedition* (Java ME), que se encontra disponível gratuitamente na web. Este artigo apresenta um estudo sobre desenvolvimento de jogos para celular na plataforma *Java ME* e a implementação do jogo Batalha Terrestre que foi idealizado pelos autores. O jogo foi testado no emulador *Java Wireless Toolkit 2.5.2* para CLDC, com dispositivo *Default Color Phone*. Espera-se que este trabalho contribua para aumentar o interesse dos estudantes no desenvolvimento de jogos para celular.

Palavras-chave: Desenvolvimento de jogos para celular, plataforma *Java ME*, dispositivos móveis.

ABSTRACT: *Cell phones became very popular for uses in entertainment area (e.g. games) and in voice transmission due to the advances of the telecommunications technology and the cost reduction in mobile devices. Developing games for cell phones requires the use of a specific programming platform, such as the Java Microedition platform (Java ME), which is available for free on the World Wide Web. This article presents an overview of the development of games for cell phones using Java ME, and a complete implementation of the Batalha Terrestre Game, which was designed by the authors and was tested on Java Wireless ToolKit 2.5.2 emulator for CLDC using the Default Color Phone device. We hope that this work can contribute to increase the interest of the students for on development of cell phone games.*

Keywords: *Cell phone game development, Java ME platform, mobile devices.*

1. Introdução

Com o avanço da tecnologia de telecomunicações entre dispositivos móveis (*Personal Digital Assistant* - PDA, *smart phones*, telefones celulares) e a redução dos custos de tais equipamentos, tornou-se popular a utilização de aparelhos de telefone celular para outros fins, especialmente para o entretenimento e, em particular, para os jogos. Atualmente, existem no mundo mais de 3,3 bilhões de assinaturas de telefones móveis (AHONEN, 2008) e este número, possivelmente, tende a aumentar, conforme o avanço da tecnologia e da necessidade crescente do ser humano do mundo moderno se manter atualizado, tendo acesso à informação a qualquer hora e lugar. Jogos para celular já movimentaram no mundo em 2008 cerca de 4,5 bilhões de dólares, confirmando um crescimento de 16,1 % em relação a 2007 (TIINSIDE, 2008). No Brasil, o faturamento das empresas desenvolvedoras de jogos para celular em 2007 foi de aproximadamente US$ 30 milhões e aumentará cerca de 30% anuais (BAUDISCH, 2007).

A habilidade de se programar para celulares vem sendo cada vez mais requerida para o profissional de informática (BRANDEL, 2007). Existem diversas plataformas para o desenvolvimento de aplicações para celular, tais como: Java ME (SUN, 2008), BREW (QUALCOMM, 2009) e Android (RUBIN, 2007). Além disso, os fabricantes de celulares fornecem seu próprio conjunto de bibliotecas específico para programação de seus dispositivos móveis. Para o programador, o desenvolvimento de tais aplicações, em geral, é desafiador, devido às restrições de memória, processamento e armazenamento destes dispositivos. Outro desafio é a portabilidade de aplicações entre dispositivos diferentes, pois existe uma grande quantidade de celulares diferentes no mercado para atender a diversos segmentos com necessidades e poderes aquisitivos distintos. Neste caso, o desenvolvedor se vê, muitas vezes, obrigado a modificar seus jogos para que os mesmos atendam aos requisitos específicos de cada telefone alvo (SAMPAIO *et al*, 2004).

O objetivo deste artigo é fornecer uma introdução à plataforma Java ME direcionada ao desenvolvimento de jogos para celular, apresentando a implementação completa do jogo Batalha Terrestre, um jogo simples e didático desenvolvido pelos autores com a finalidade de despertar o interesse dos estudantes de computação no desenvolvimento de jogos para celular.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta as principais características da plataforma Java ME; a Seção 3 foca nos aspectos do desenvolvimento de jogos para celular; a Seção 4 discute a implementação e teste do jogo da Batalha Terrestre; e a Seção 5 apresenta as considerações finais.

2. A plataforma Java ME

A plataforma Java ME (SUN, 2008) foi lançada em 1999 pela empresa Sun Microsystems com a intenção de permitir desenvolvimento de aplicações para celular. Considerando as limitações de *hardware* de tais dispositivos, a implementação da sua Máquina Virtual Java (JVM) foi reduzida significativamente, se comparada com a JVM da plataforma Java SE (*Standard Edition*), usada em aplicações *desktop,* pois, a plataforma Java ME não contém todas as classes da edição Java SE. A plataforma Java ME disponibilizada para diversos sistemas operacionais, dentre eles:

- *Symbian OS* – foi desenvolvido pela Symbian e projetado para usar o mínimo possível de recursos de hardware, tais como memória e bateria. Como sua arquitetura é modular, ela permite que cada fabricante implemente um projeto específico para a interface gráfica. O Symbian OS suporta também aplicações desenvolvidas em Symbian C/C++, Python, dentre outras linguagens (SYMBIAN, 2008).
- *Windows Mobile* – foi desenvolvido pela Microsoft© Corporation e é voltado para dispositivos móveis de maior capacidade, como PDA e *smartphones*. Existem algumas JVM para Windows Mobile, dentre elas a versão gratuita Mysaifu JVM e a CrEme, versão melhorada da JVM criada pela NSIcom e comercializada pela Symbol (SYMBOL, 2008).
- *Android* – foi desenvolvido pela Google e é um sistema operacional baseado em Linux que inclui uma interface gráfica e aplicações (RUBIN, 2007). Para esta plataforma, uma nova tecnologia de JVM, chamada Dalvik, foi desenvolvida pela

Google, a qual executa mais rapidamente programas Java em dispositivos com restrições de hardware (SHANKLAND, 2007).

2.1. Arquitetura Java ME

Para possibilitar a utilização da plataforma Java ME em diferentes tipos de dispositivos, a sua arquitetura foi dividida em camadas, conforme a Tab. (1). A primeira camada (camada de mais baixo nível) é a Máquina Virtual Java, que é instalada no sistema operacional do dispositivo. Acima desta camada está a camada de configuração, que fornece diversas bibliotecas para implementações de funcionalidades essenciais para a categoria do dispositivo, para padronizar o ambiente de execução. Por último, encontra-se a camada de perfil (camada de mais alto nível).

- Um perfil é uma extensão da camada de configuração e possui bibliotecas relacionadas ao tipo de dispositivo. Um perfil está associado a apenas uma configuração. Para dispositivos móveis de pequena capacidade, tal como o celular, utiliza-se o perfil MIDP na configuração CLDC.

2.2. O Perfil MIDP

O perfil MIDP oferece pacotes de bibliotecas para implementação de diversos tipos de aplicações, desde jogos até aquelas que usam comunicação em rede com criptografia (SUN, 2006), fornecendo recursos adequados para manipulação de desenhos em interfaces gráficas, monitoramento e gerenciamento de eventos de teclado. Os pacotes disponíveis são:

- *User Interface Package* (UIP), para criação de interface para o usuário;
- *Persistence Package,* para armazenamento de dados e posterior recuperação dos mesmos pela aplicação MIDlet;
- *Application Lifecycle Package* (ALP), para gerenciamento do ciclo de vida da aplicação, define as interações entre a aplicação e o ambiente no qual a mesma é executada;
- *Network Package,* para conexões de rede, incluindo conexões *HTTP* (*Hyper-Text Transfer Protocol*), *HTTPS* (*Secure HTTP*) e conexões via *sockets*;
- *Audio Package,* para execução de arquivos de audio e video. Suporta ampla faixa de tipos de multimídia;
- *Public Key Package,* para estabelecimento de troca segura de informações;
- *Core Packages,* para funcionamento mínimo de uma aplicação, como entrada/saída de dados, manipulação de data e hora, dentre outras.

As aplicações para celular que utilizam o perfil MIDP são conhecidas como *MIDlets*, pois, são implementações da classe abstrata *javax. microedition.midlet.MIDlet* (pacote ALP), que gerencia o ciclo de vida da aplicação.

2.3. Ciclo de vida dos *MIDlets*

A classe abstrata *MIDlet* determina o ciclo de vida da aplicação, que pode assumir três estados: pausado, ativo e destruído, como ilustrado no esquema da Fig. (1), onde:

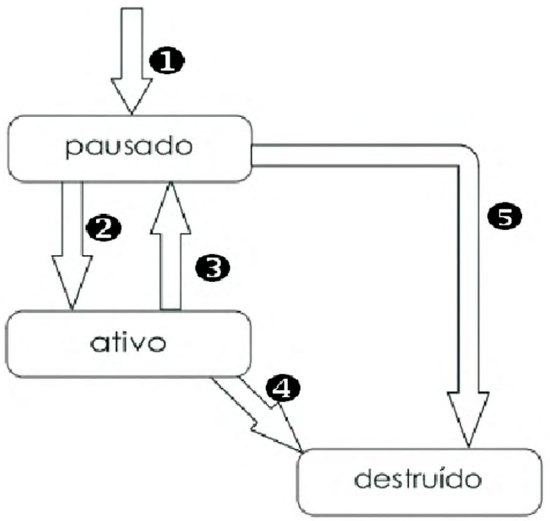

Figura 1. Esquema do ciclo de vida de um MIDlet (Adaptado de ORTIZ, 2004).

❶ Após uma chamada ao método construtor do *MIDlet*, o mesmo entra em estado pausado;
❷ Após chamada ao método *startApp()*, o *MIDlet* entra em estado ativo;
❸ Do estado ativo, pode voltar ao estado pausado, através da chamada ao método *pauseApp();*

❹ Através da chamada ao método *destroyApp()*, o *MIDlet* em estado ativo pode ser destruído, finalizando, assim, sua execução.
❺ Alternativamente, o *MIDlet* em estado pausado pode também ser finalizado pela chamada ao método *destroyApp()*.

Tais métodos devem ser implementados pelo desenvolvedor da aplicação *MIDlet*. O método *pauseApp()* é útil quando há necessidade de pausar a aplicação quando ocorrer um evento no celular que tenha maior prioridade que o *MIDlet* em execução, como, por exemplo o recebimento de uma chamada telefônica. Contudo, isto pode ocasionar a perda de dados da aplicação interrompida se não houver implementação adequada dos métodos *startApp()* ou *pauseApp()*.

2.4. Interface gráfica para *MIDlets*

O diagrama UML (*Unified Modeling Language)* da Fig. (2) apresenta, dentre outras, as principais classes de interface gráfica do perfil MIDP, que estão no pacote *javax.microedition. lcdui,* as quais são descritas a seguir e serão utilizadas, posteriormente, no desenvolvimento do jogo Batalha Terrestre (ver Seção 4).

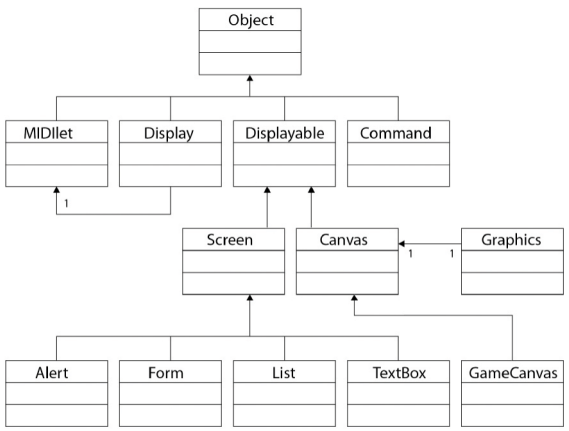

Figura 2. Diagrama UML das principais classes de interface gráfica da plataforma Java ME.

Uma aplicação *MIDlet* deve conter apenas uma referência para um objeto da classe *javax.microedition.lcdui.Display* (pacote UIP) que representa o *hardware* do equipamento e que pode exibir diferentes elementos de interface (telas, por exemplo), sejam eles implementações das classes abstratas *javax.microedition.lcdui.Screen* ou *javax.microedition.lcdui.Canvas,* ambas subclasses da classe abstrata *javax.microedition. lcdui.Displayable*.

O usuário interage com o *MIDlet* através de objetos da classe *javax.microedition.lcdui. Command* (pacote UIP), que correspondem aos comandos invocados pelo usuário. Os comandos podem ser adicionados em objetos originados da classe *Displayable*. Na interface, os comandos são associados a botões ou a itens de menu.

A classe abstrata *Screen* fornece subclasses para construção de interfaces de alto nível, tais como *Alert*, *Form*, *List* e *TextBox*. Um objeto da classe *Alert* é um tipo de tela que é usado para exibir uma mensagem ao usuário e aparece durante um período pré-definido de tempo. Um objeto da classe *Form* é um outro tipo de tela que pode receber outros objetos de outras classes (comandos, campos de entrada de texto, imagens, etc.), para composição de um formulário, por exemplo. Um objeto da classe *List* se constitui em uma tela que oferece opções em forma de lista ao usuário. A lista é navegável e o usuário pode selecionar e escolher uma ou mais opções. Um *TextBox* é um tipo de tela que permite a entrada e edição de texto.

A classe abstrata *Canvas* e sua subclasse abstrata *GameCanvas* são específicas para construção de interfaces gráficas de baixo nível e muito utilizadas na implementação de jogos. A classe *Canvas* fornece métodos para gerenciar eventos de teclado e de apontadores, e métodos para detectar se os dispositivos suportam tais eventos. Nesta classe, existe uma referência para um objeto da classe *javax.microedition.lcdui. Graphics* (pacote UIP), subclasse direta de *Object,* que permite o desenho de formas geométricas de duas dimensões, textos e imagens sobre um objeto *Canvas*, possibilitando um controle sobre os *pixels* da tela do dispositivo – o sistema de coordenadas de *pixels* atribui o ponto (0,0) no canto superior esquerdo da tela.

Para implementação de jogos, a classe *GameCanvas* oferece mais vantagens sobre a sua superclasse *Canvas*, pois, provê recursos específicos, como a habilidade de detectar o estado das teclas e *buffer* gráfico *off-screen*, que impede que imagens parciais sejam exibidas durante a composição de cenas de jogo. A classe *GameCanvas* deve ser estendida na aplicação.

2.5. Estrutura de uma aplicação *MIDlet*

Nas aplicações *MIDlets*, as classes Java são compiladas num arquivo *.jar* (Java Archive). A coleção de recursos e arquivos de um *MIDlet* é chamada de *MIDlet* Suite, que consiste de:

- Arquivo Java Archive (*.jar*): classes Java compiladas;
- Arquivo Java Application Descriptor (*.jad*): arquivo que descreve o MIDlet, armazenado fora do *.jar ;*
- Arquivo manifest (*.mf*): descreve caracte-rísticas da aplicação, sendo empacotado dentro do próprio *.jar;*
- Recursos para a aplicação (imagens, arquivos de audio, etc).

Tabela 1. Arquitetura Java ME em camadas

Camada	Dispositivos de menor capacidade	Dispositivos de maior capacidade
Perfil	MIDP (*Mobile Information Device Profile*) – apresenta funcionalidades como suporte a conexões HTTP e *sockets*, API para jogos, interface gráfica simplificada.	PP (*Personal Profile*) – perfil mais completo da configuração CDC, provê suporte a *applets* e a interface gráfica AWT (*Abstract Window Toolkit*).
Configuração	CLDC (*Connected Limited Device Configuration*) – oferece bibliotecas reduzidas, para uso em dispositivos móveis de baixa capacidade computacional, como celulares.	CDC (*Connected Device Configuration*) – definida na *Java Specification Requirement* (JSR) 218, foi planejada para dispositivos móveis de capacidade maior que a dos celulares, como *palmtops*.
Máquina Virtual Java	KVM (*Kilo Virtual Machine*)	*HotSpot Implementation*

3. Desenvolvimento de jogos para celular

Jogos para computadores, em geral, têm evoluído bastante nos últimos anos, principalmente com relação à interface gráfica, incluindo animações 3D, já que computadores pessoais possuem amplos recursos de *hardware* que permitem aplicações mais sofisticadas. No entanto, o desenvolvimento de aplicações para celular é desafiador, devido às restrições de memória, tamanho de tela, processamento e armazenamento destes dispositivos.

Apesar da plataforma Java ME ser portável, o desenvolvedor de jogos para celular deve observar que cada tipo de dispositivo suporta um determinado conjunto de bibliotecas e pode ser que o código desenvolvido para um não seja totalmente portável para outro, principalmente, entre dispositivos de diferentes fabricantes (SAMPAIO et al, 2004). Contudo, diferentemente dos jogos tradicionais para computadores *desktop*, o desenvolvimento de jogos para celular requer menor tempo e menor orçamento, pois, devido às limitações de memória e processamento destes dispositivos, a implementação do jogo deve ser mais simples e, portanto, mais rápida.

Um jogo para computador pode ser definido como um sistema que consiste de três componentes fundamentais (BATTAIOLA et al, 2001) que são:

- *Enredo* – define o tema e o(s) objetivo(s) do jogo, que deve(m) ser completado(s) pelo jogador para ganhar o jogo e/ou receber pontuações.
- *Motor* – controla o estado do jogo após a execução de cada jogada ou comando e, dependendo do tipo de jogo, controla a execução de uma nova jogada pelo computador. Jogos mais complexos requerem algoritmos de inteligência artificial.
- *Interface interativa* – reporta ao jogador as alterações realizadas após a execução de uma jogada ou comando. O projeto de interface envolve aspectos técnicos (como plataforma de desenvolvimento e codificação), os aspectos cognitivos (percepção correta da informação apresentada pela interface) e artísticos (imagens, gráficos, efeitos de som e músicas, etc).

3.1. Classificação dos jogos

Segundo o objetivo, os jogos podem ser classificados como (BATTAIOLA et al, 2001):

- *Aventura* – jogos que misturam ações baseadas em reflexo rápido e raciocínio lógico para que o jogador consiga ultrapassar fases do jogo até alcançar o objetivo final.
- *Simuladores* – jogos que exigem reflexos do jogador que está inserido em um ambiente que tenta simular a realidade.
- *Estratégia* – jogos que exigem o alcance de uma meta através de estratégias e táticas, como jogos de xadrez.
- *Role Playing Game (RPG)* – jogos em que o jogador assume papéis e enfrenta situações para ganhar experiência.
- *Esporte* – jogos que implementam partidas com regras de esportes convencionais (futebol, vôlei, basquete etc.).
- *Passatempo* – jogos sem história relacionada, onde a meta é a execução de uma tarefa, como montar um quebra-cabeça ou fazer mais pontos executando uma atividade proposta.

No entanto, essa classificação não é totalmente rígida e os jogos podem ser implementados com características de mais de uma delas, para inovar e despertar interesse dos usuários.

Adicionalmente, um jogo pode ser monousuário ou multiusuário. Existe uma tendência no desenvolvimento de jogos multiusuários, principalmente, com o advento das redes de telefonia 2G e 3G (NETSIZE, 2008). Contudo, grande parte deles é para um número reduzido de jogadores, por questões de limitação de comunicação em rede, principalmente de longa distância, que pode causar atrasos perceptíveis na atualização da interface após a conclusão de cada jogada, o que afeta o desempenho daquelas aplicações que necessitam de simulação em tempo real.

4. Estudo de caso: o jogo Batalha Terrestre

Nesta seção apresentamos o desenvolvimento completo do jogo Batalha Terrestre que foi idealizado pelos autores e implementado na plataforma Java ME/MIDP 2.0. Classificamos o jogo como monousuário e de passatempo. O jogo foi desenvolvido usando a IDE NetBeans 6.1 e testado no emulador Java Wireless Toolkit 2.5.2 para CLDC 1.1, com dispositivo *Default Color Phone*.

4.1. Descrição do jogo

O jogo é composto por um tabuleiro 10 x10 com 100 células, sendo 5 delas aleatoriamente escolhidas pela aplicação como alvos a serem identificados pelo jogador, o qual terá 30 tiros (tentativas) para acertar todos os cinco alvos e será informado na interface se o mesmo acertou algum alvo ou se acertou apenas terra, de acordo com a localização do tiro. Além disso, caso o tiro seja próximo a algum alvo, em uma das oito posições vizinhas à célula escolhida no tabuleiro, uma mensagem informativa será exibida ("Há alvo vizinho!"). Caso contrário, nenhuma mensagem é mostrada.

4.2. Implementação do jogo

A Fig. (3) mostra o modelo de classes definido para o jogo Batalha Terrestre (BT), tomando como base o diagrama da Fig.(2). Para maior compreensão desta implementação, recomenda-se que o leitor tenha acesso a documentação da API Java MIDP 2.0, disponível no *site* da Sun Microsystems (SUN, 2006).

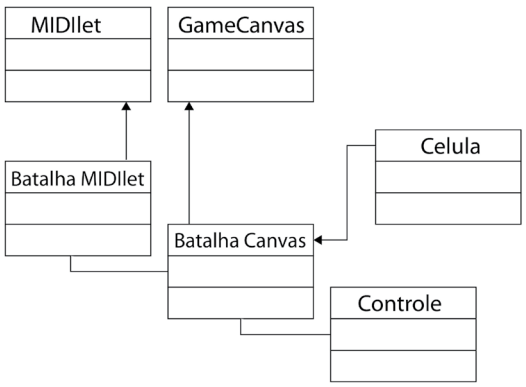

Figura 3. Diagrama de classes do jogo.

O código-fonte completo do jogo está disponível no anexo deste artigo. Por simplicidade, os métodos *getters* e *setters,* para atributos, foram omitidos nas classes *Controle* e *Célula*. Assume-se que todas as classes Java ME citadas nesta seção pertencem ao pacote *javax.microedition.lcdui*, exceto quando indicado.

4.3. Implementação do MIDlet

A classe *BatalhaMidlet* é subclasse da classe *javax.microedition.midlet.MIDlet* e gerencia o ciclo de vida da aplicação. Deve possuir uma referência para um objeto da classe *Display*, pois existe apenas uma instância desta classe por *MIDlet*, que pode ser obtida pela chamada ao método estático *Display.getDisplay()*. *Display*, neste caso, representa o gerenciador da tela do dispositivo. A partir da referência ao *Display*, é possível determinar qual tela deve ser exibida num determinado momento, já que só é possível mostrar uma por vez. O método responsável por isto é o *setCurrent(Displayable d),* onde o argumento pode ser qualquer objeto de uma subclasse concreta da classe abstrata *Displayable*. No caso da aplicação do jogo BT, existem seis telas que serão exibidas em situações e momentos distintos, de acordo com os comandos acionados pelo usuário ou conforme o andamento do jogo (Figura 4).

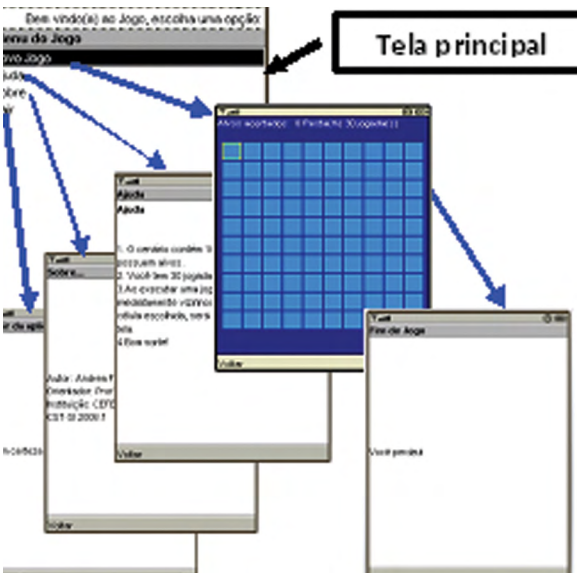

Figura 4. As opções da tela principal levam a outras telas, sendo a do tabuleiro que corresponde à classe *BatalhaCanvas*.

Para realizar a ação determinada pelos objetos do tipo *Command*, a classe *BatalhaMidlet* implementa a interface *CommandListener*, e, consequentemente, o método *commandAction(Command c, Displayable d)*.

Para o jogo BT, um objeto do tipo *List* foi definido como a tela inicial, onde existe uma lista de opções a serem escolhidas pelo usuário ("**Novo Jogo**", "**Ajuda**", "**Sobre**", "**Sair**"). Cada uma delas leva à exibição de uma nova tela. No caso do *List*, cada seleção realizada é considerada um comando, sendo isto implementação nativa do perfil MIDP 2.0. Todas as telas do jogo BT (*Displayable*) e comandos são construídos por instanciação tardia nos métodos *getters*, para evitar sobrecarga da memória limitada do dispositivo. A tela do jogo propriamente dito, disponível na opção "**Novo Jogo**", corresponde à instância da classe *BatalhaCanvas*, que é uma subclasse de *GameCanvas* (pacote *javax.microedition.lcdui.game*). A maior parte da interação com o usuário no jogo é através desta tela, com exibição de mensagens e alteração das cores das células que o compõem.

4.4. Desenho do tabuleiro

No construtor do *BatalhaCanvas*, é chamado o método *comporInterface()* que, além de chamar os métodos para o desenho do tabuleiro, determina, na célula [0][0], a posição do cursor, que é uma borda quadrada de **cor alaranjada** na célula selecionada. Esse cursor mudará de posição de acordo com a direção indicada pelo usuário sobre a tecla multidirecional do dispositivo (emulador), exibido na Fig. (5).

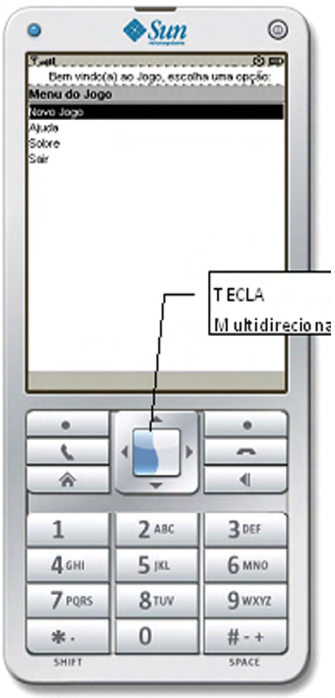

Figura 5. Captura da tela inicial do jogo no emulador.

Para a composição do tabuleiro, foi usada uma matriz quadrada 10 x 10 de objetos da classe *Celula*. Sua dimensão foi definida numa constante no *BatalhaCanvas*. Essa matriz é construída na memória do dispositivo (método *inicializarTabuleiro()*) e a cada posição é atribuída uma instância da classe *Celula*, criada para o jogo BT. Para sua exibição, é invocado o método *desenharTabuleiro()*, que confere **cor azul escura** ao plano de fundo do tabuleiro e faz cada célula invocar seu método *pintar(Graphics g)*, que renderiza a mesma de acordo com seu *status*, sendo que todas, no início do jogo, recebem a **cor azul clara.**

4.5. Descrição das células do tabuleiro

Cada objeto *Celula* possui referências à sua localização no tabuleiro (x, y), à sua cor e à sua dimensão, determinada dinamicamente no *BatalhaCanvas* com base na largura e altura da tela do dispositivo (métodos *getWidth()* e *getHeight()*, da classe *Displayable*). Além ter de características físicas, cada célula pode assumir quatro possíveis *status,* definidos como constantes numéricas inteiras de 0 a 3:

- *OCULTA_VAZIA* = 0 (célula oculta sem alvo)
- *OCULTA_ALVO* = 1 (célula oculta com alvo)
- *EXPLICITA_VAZIA* = 2 (célula escolhida pelo jogador, porém sem alvo)
- *EXPLICITA_ALVO* = 3 (célula escolhida pelo jogador, contendo alvo).

Considera-se oculta aquela célula que não recebeu tiro (não escolhida pelo jogador), ao contrário da explícita. Toda célula é construída com *status*=0. Uma vez definido pelo jogo, através de sorteio (classe *Controle*, método *definirAlvos()*), que uma célula terá alvo (*status*=1), seu *status* só poderá mudar para 3, quando o jogador selecioná-la, acertando o alvo. Analogamente, para uma célula sem alvo, seu *status* mudará de 0 para 2, após ser escolhida pelo jogador. A célula pode informar através do método *estaDisponivel()* se pode ser realizada alguma jogada sobre si própria e ainda informa quanto à presença ou não de um alvo em si (método *temAlvo()*).

4.6. Gerenciamento dos eventos de tecla

No *BatalhaCanvas*, os eventos de tecla são detectados através dos métodos *keyPressed(int keyCode), keyRepeated(int keyCode).* O primeiro detecta quando uma tecla é pressionada, o segundo, quando uma tecla é pressionada por longo intervalo de tempo (repetida). Ambos foram implementados no *BatalhaCanvas*, pois suas implementações na classe ancestral abstrata *Canvas* são vazias. O argumento *keyCode* (código da tecla) é fornecido à aplicação pela máquina virtual Java do dispositivo (emulador) e pode ser traduzido em ações de jogo, representadas pelas constantes definidas no perfil MIDP: *FIRE*, *UP*, *DOWN*, *LEFT*, *RIGHT*, que foram usadas na implementação destes métodos no *BatalhaCanvas* para monitoramento da ação do jogador sobre a tecla multidirecional. Por causa da implementação do método *keyRepeated*(*int keyCode)*, o usuário pode manter a tecla multidirecional pressionada em uma direção até que o cursor se posicione sobre a célula desejada.

O método *moverCursor(int, int),* adaptado de NOKIA 2002, recebe dois argumentos: variação da linha e da coluna na matriz de acordo com a direção escolhida (*UP*, *DOWN*, *LEFT*, *RIGHT*), realiza a remoção do cursor da célula anterior, repintando a mesma com sua cor atual, e o coloca sobre a última célula selecionada, dando a impressão de movimento. Novamente o método *pintar(Graphics g)* do objeto célula é invocado. Em seguida, o método *repaint(int*, *int*, *int*, *int)* da superclasse *Canvas* é chamado. Os quatro argumentos determinam o ponto de início da pintura da tela (x,y), largura e altura, que, neste caso, correspondem à posição e dimensão somente da célula a ser "repintada", realizando processamento apenas sobre esta pequena área. Existe outro método *repaint()* sem argumentos*,* para *Canvas*, que equivale ao anterior com os argumentos 0,0, *getWidth()* (largura da tela), *getHeight()* (altura da tela)*,* e realiza nova pintura de toda a área da mesma.

O jogador, ao pressionar o centro da tecla multidirecional (*FIRE*), dispara a jogada sobre a célula selecionada, através do método *executarJogada(),* que altera o *status* da célula escolhida para um dos seguintes:

- *EXPLICITA_VAZIA*, caso a mesma não possua alvo, invocando método *pintar(Graphics g)* da célula para atribuir-lhe nova cor, a mesma do plano de fundo do tabuleiro (**azul escuro**), de acordo com seu novo *status*. Neste caso, é informado ao jogador se há alvo vizinho à sua ultima célula

escolhida. A dica "**Há alvo vizinho!**" é exibida no rodapé da tela;

- *EXPLICITA_ALVO*, caso a mesma possua alvo, invocando método *pintar(Graphics g)* da célula para atribuir-lhe nova cor (**azul anil**), de acordo com seu novo *status*, neste caso, atribuindo um acerto ao jogador.

As jogadas sobre células anteriormente escolhidas não causam efeito sobre o jogo.

4.7. Controlador do jogo

A classe *Controle* contém constantes que definem o número máximo de alvos e de jogadas, realiza contagem dos acertos e do número de jogadas executadas pelo usuário. No início de cada jogo, define os alvos por sorteio (método *definirAlvos()*).

Destaca-se aqui o método *temAlvoVizinho()*, que determina a presença ou não de pelo menos um alvo em uma das oito células vizinhas à célula escolhida, após uma jogada sem acerto em alvo. Este método faz uso de outro, *avaliarPosicao*(*int, int),* que recebe como argumentos os valores da linha e da coluna da célula escolhida, incrementados ou decrementados em uma unidade, na intenção de se fazer uma varredura nas oito posições possíveis vizinhas à célula da jogada atual. Este método tenta obter o *status* da célula correspondente na matriz do tabuleiro com base nos argumentos fornecidos. Contudo, se a jogada for executada nas células mais externas da matriz, ocorrerá para algumas das oito possíveis posições avaliadas a exceção de *java.lang. ArrayIndexOutOfBoundsException*, já que não haverá oito células vizinhas para tais células mais externas, retornando falso, fazendo com que a próxima linha do método *temAlvoVizinho* seja excutada, até que seja encontrado um alvo vizinho ou não.

4.8. Teste do jogo

Para iniciar o jogo, o jogador deve selecionar a opção "Novo jogo" conforme a Fig. (5). Em seguida, aparece o tabuleiro, como mostra a Fig.(6a), sendo que cinco células contém os alvos ocultos que o jogador deverá identificar.

A cada jogada, o jogador é informado sobre a quantidade de acertos e jogadas restantes e, no final do jogo, é exibida uma tela de encerramento com o resultado final (vitória ou derrota), conforme a Fig. (6c).

Na classe *BatalhaMidlet*, o método *startApp*() foi implementado de modo que, após uma chamada telefônica, o jogador pudesse retomar o jogo do ponto de onde parou. Para isso, uma variável booleana foi declarada para identificar quando o jogo está em execução. Essa variável é sempre testada no método *startApp()* que é invocado quando a aplicação inicia e também quando retorna após chamada ao *pauseApp()* que ocorre quando uma chamada telefônica é recebida pelo celular. Para testar esta situação seleciona-se, durante uma partida em andamento, a opção *Pause*

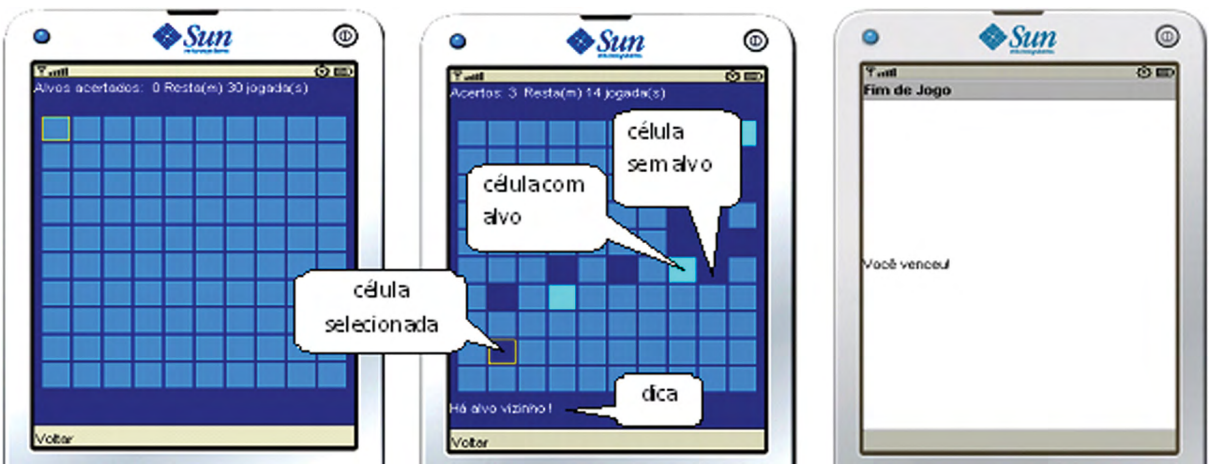

Figura 6. Telas do jogo: (a) início do jogo com o cursor em alaranjado sobre a primeira célula; (b) tabuleiro com alvos identificados pelo jogador e (c) mensagem final.

do menu MIDlet do emulador, que, então, exibe a mensagem *Incoming Call*. Em seguida, seleciona-se a opção *Resume* do menu MIDlet mostrada na Fig. (7). Neste momento, o jogo retorna ao estado anterior ao da simulação da chamada telefônica.

Figura 7. Simulação de uma chamada telefônica.

5. Considerações finais

Este artigo apresentou uma visão geral da plataforma Java ME – uma das plataformas mais utilizadas para o desenvolvimento de jogos para celular – através da implementação do jogo Batalha Terrestre (BT), o qual foi idealizado pelos autores, sendo classificado como um jogo monousuário e de passatempo. A escolha desse jogo foi baseada na sua similaridade com o jogo Batalha Naval, bastante conhecido no mundo dos jogos. Porém, o jogo BT é muito mais simples de usar e de implementar, possibilitando um maior entendimento por parte dos leitores.

A técnica apresentada para a criação da interface gráfica do jogo, controlando os *pixels* para desenhar o tabuleiro, não é a única possível, tendo sido utilizada neste trabalho por ser mais simples de implementar e requerer menor número de classes Java ME para a construção da interface gráfica. Interfaces mais avançadas podem ser construídas a partir de outras classes do Java ME tais como, as classes *Layer*, *TiledLayer*, *Sprite (javax.microedition.lcdui.game)* e *Image (javax.microedition.lcdui)*.

A implementação do jogo BT com uso da tecla multidirecional, para realização das jogadas e movimentação do cursor, confere maior usabilidade ao jogo, pois, o jogador precisa usar apenas uma tecla durante toda a execução do mesmo.

6. Referências

AHONEN, Tomi. **When there is a mobile phone for half the planet**, 2008. Disponível para Internet:<http://communities-dominate.blogs.com/brands/2008/01/when-there-is-a.html>Acesso em: 23/05/2008.

BATTAIOLA, A. *et al.* **Desenvolvimento de Jogos em Computadores e Celulares**, 2001 Disponível para Internet: <http://www.inf.ufrgs.br/~revista/docs/rita08/rita_v8_n2_p7a46.pdf> Acesso em: 13/05/2008.

BAUDISCH, A. **Informações sobre o mercado de jogos para celular**, 2007. Disponível para Internet:<http://desenvolvedormovel.aurium-mobile.com/index.php/2007/10/25/informacoes-sobre-o-mercado-de-jogos-para-celular/>Acesso em: 06/07/2008.

BRANDEL, M. **12 skills that employers can't say no to**, 2007. Disponível para Internet: <http://www.javaworld.com/javaworld/jw-07-2007/jw-07-skills.html?page=1> Acesso em: 01/06/2008.

MICROSOFT. **Windows Mobile Development**, 2008. Disponível para Internet: <http://www.microsoft.com/windowsmobile/developers/default.mspx> Acesso em: 01/06/2008.

NETSIZE Mobile Business & Entertainment. **The NetSizeGuide**, 2008. Disponível para

Internet: <http://www.netsize.com/downloads/registration.aspx Acesso: em 24/04/2008.

NOKIA. **Brief Introduction to MIDP Graphics v1.0. Forum Nokia, 2002.** Disponível para Internet:
http://sw.nokia.com/id/9f1ca05b-9959--47ac-a7ee-78d974a7d96f/Brief_Introduction_to_MIDP_Graphics_v1_0.pdf Acesso em 01/06/2008.

NOKIA. **How to develop a game - Part 3.** Fórum Nokia.com, 2007. Disponível para Internet: <http://wiki.forum.nokia.com/index.php/How_to_develop_a_game_-_Part_3> Acesso em 26/12/2008.

ORTIZ, C. E. **Managing the MIDlet Life-Cycle with a Finite State Machine.** Sun Developer Network, 2004. Disponível para Internet: <http://developers.sun.com/mobility/midp/articles/fsm/> Acesso em 26/12/2008.

QUALCOMM. **Developing with Brew**, 2009. Disponível para Internet: < http://brew.qualcomm.com/brew/en/developer/overview.html> Acesso em 08/04/2009

RUBIN, Andy. **Where's my Gphone?**, 2007. Disponível para Internet: <http://google-blog.blogspot.com/2007/11/wheres-my-gphone.html#links> Acesso em: 23/05/2008.

SAMPAIO, P.; DAMASCENO, A. ; SAMPAIO, I. *et al.* **Portando Jogos em J2ME: Desafios, Estudo de Caso e Diretrizes.** Revista SCIENTIA, vol. 15, n° 1, 2004. Disponível para Internet: <http://twiki.cin.ufpe.br/twiki/pub/SPG/AspectProductLine/scientia05.pdf> Acesso em: 07/07/2008.

SHANKLAND, Stephen. **Google's Android parts ways with Java industry group**, 2007. Disponível para Internet: <http://news.cnet.com/8301-13580_3-9815495-39.html> Acesso em: 23/05/2008.

SONY-ERICSSON. **XPERIA X1,** 2008. Disponível para Internet: <http://www.sonyericsson.com/cws/products/mobilephones/specifications/x1?cc=us&lc=en> Acesso em: 01/06/2008.

SUN Microsystems. **Java ME Platform Overview**, 2008. Disponível para Internet: <http://java.sun.com/javame/technology/index.jsp> Acesso em: 04/05/2008.

SUN Microsystems. **MID Profile**, 2006. Disponível para Internet: <http://java.sun.com/javame/reference/apis/jsr118> Acesso em: 04/05/2008.

SYMBIAN. **Company Overview**, 2008. Disponível para Internet: <http://www.symbian.com/about/index.html> Acesso em: 23/05/08.

SYMBOL. **CrEme Plus v.3.25,** 2008. Disponível para Internet: <http://software.symbol.com/detail.cfm?prod=1786> Acesso em: 23/05/08.

TIINSIDE. **Jogos para celulares já movimentaram US$ 4,5 bi neste ano**, 2008. Disponível para Internet: <http://www.tiinside.com.br/Filtro.asp?C=265&ID=90125> Acesso em: 06/07/2008.

ANEXO – Código-fonte da aplicação

```
/*****************************************
Classe BatalhaMidlet
*****************************************/

public class BatalhaMidlet extends MIDlet implements
CommandListener{

    private Display display = null;
    private List formPrincipal = null;
    private Ticker ticker  = null;
    private String help =
            "\n1. O cenário contém 100 células,
das quais " + Controle.TOTAL_ALVOS + " possuem al-
vos...";
    private Form formAjuda = null;
  private StringItem ajuda = null;
  private Alert alertSobre = null;
  private Alert alertSair = null;
  private BatalhaCanvas batalhaCanvas = null;
  private Command comandoOK = null;
  private Command comandoCancel = null;
  private Command comandoBack = null;
  private boolean emJogo = false;

public BatalhaMidlet(){}

public void startApp() {
   if (emJogo) {
       exibir(get_Canvas());
       return;
```

```
   }
   if (display == null)
       display = Display.getDisplay(this);
       display.setCurrent(get_FormPrincipal());
   }

public void pauseApp() {}

public void destroyApp(boolean unconditional) {
        notifyDestroyed();
}

public  Displayable get_Canvas(){
  if (batalhaCanvas == null){
      batalhaCanvas = new BatalhaCanvas(false,
this);
      batalhaCanvas.addCommand(get_ComandoBack());
      batalhaCanvas.setCommandListener(this);
      }

  return batalhaCanvas;
}

public  Displayable get_FormPrincipal(){
   if (formPrincipal == null){
       formPrincipal = new List("Menu do Jogo",
                                      Choice.IM-
PLICIT);
       ticker = new Ticker("Escolha uma opção:");
       formPrincipal.append("Novo Jogo",null);
       formPrincipal.append("Ajuda",null);
       formPrincipal.append("Sobre",null);
       formPrincipal.append("Sair",null);
       formPrincipal.setTicker(ticker);
       formPrincipal.setCommandListener(this);
    }
    return formPrincipal;
 }
 public  Displayable get_FormAjuda() {
     if (formAjuda == null) {
         formAjuda = new Form("Ajuda");
         ajuda = new StringItem("Ajuda\n", help);
         formAjuda.append(ajuda);
         formAjuda.addCommand(get_ComandoBack());
         formAjuda.setCommandListener(this);
        }
     return formAjuda;
 }

public  Displayable get_AlertSobre() {
 if (alertSobre == null) {
     alertSobre = new Alert("Sobre...",
     "Autor: Andrea\nOrientador: Prof. Fausto
      Ayres\nInstituição: IFPB\nCurso: CST-SI",
      null, AlertType.INFO);
      alertSobre.addCommand(get_ComandoBack());
      alertSobre.setCommandListener(this);
      alertSobre.setTimeout(6000);
  }
  return alertSobre;
}

public  Displayable get_AlertSair() {
  if (alertSair == null) {
      alertSair = new Alert("Sair da aplicação:",
            "Tem certeza que deseja sair?",
     null,
     AlertType.CONFIRMATION);

    alertSair.addCommand(get_ComandoOK());
    alertSair.addCommand(get_ComandoCancel());
    alertSair.setCommandListener(this);
 }
 return alertSair;
}

public  Command get_ComandoBack() {
 if(comandoBack == null) {
    comandoBack = new Command("Voltar",
                                          Command.BACK,
0);
 }
 return comandoBack;
}


public  Command get_ComandoCancel() {
  if (comandoCancel == null) {
      comandoCancel = new Command("Cancelar",
                                              Command.
CANCEL, 0);
  }
  return comandoCancel;
}

public  Command get_ComandoOK() {
  if(comandoOK == null) {
     comandoOK = new Command("OK", Command.OK, 1);
  }
  return comandoOK;
}

public  void finalizarJogo(boolean resultado){
  if (resultado)
        exibir(new Alert("Fim de Jogo",
        "Você venceu!",
        null,AlertType.INFO),get_FormPrincipal());
  else
        exibir(new Alert("Fim de Jogo",
        "Você perdeu!",
        null,AlertType.WARNING),get_FormPrinci-
        pal());

        emJogo = false;
}

public  void exibir(Displayable dp){
    Display.getDisplay(this).setCurrent(dp);
}


public  void exibir(Alert a, Displayable proximo){
    Display.getDisplay(this).setCurrent(a, proximo);
}


public  void commandAction(Command c, Displayable d)
{

  switch (c.getCommandType()){

  case Command.BACK:
   if (d.equals(formAjuda) || d.equals(alertSobre))
       exibir(formPrincipal);
       else if (d.equals(batalhaCanvas)){
                batalhaCanvas = null;
                emJogo = false;
                exibir(get_FormPrincipal());
                }
   break;

   case Command.CANCEL:
      if (d.equals(alertSair))
          exibir(formPrincipal);
   break;

   case Command.OK:
      if (d.equals(alertSair)){
          destroyApp(true);
      }
   break;

   default:
   if (c.equals(List.SELECT_COMMAND)){
    if (d.equals(formPrincipal)){

     switch(formPrincipal.getSelectedIndex()){
      case 0:
```

```
        if (batalhaCanvas!=null)
                    batalhaCanvas = null;

                    emJogo = true;
                    exibir(get_Canvas());
      break;

      case 1: exibir(get_FormAjuda()); break;
      case 2: exibir(get_AlertSobre()); break;
      case 3: exibir(get_AlertSair()); break;
                    }
                }
            }
        }
    }
}

/*****************************************
Classe BatalhaCanvas
*****************************************/

public  class BatalhaCanvas extends GameCanvas{
    private int larguraTela = getWidth();
    private int alturaTela = getHeight();
    static final int DIMENSAO_TABULEIRO = 10;
    static final int COR_MEDIA = 0x1C94C7;
    static final int COR_MENSAGENS = 0xFFFFFF;
    static final int COR_TABULEIRO = 0x000099;
    static final int Y_POSICAO_TABULEIRO = 30;
    static final int X_POSICAO_TABULEIRO = 4;
    static final int INTERVALO_CELULAS = 2;
    private final int Y_POS_DICA = alturaTela - 10;
    private final int X_POS_DICA = 5;
    final String[] dicas = {"", "Há alvo vizinho! "};

    private int dimensao;
    private BatalhaMidlet midlet;
    private Celula[][] celulas;
    private Controle jogo;
    private int antigaPosCursor;
    private int atualPosCursor;
    private String dica = "";
    private Graphics g = null;
    private Font fonte = null;

protected BatalhaCanvas(boolean suppressKeyEvents,
BatalhaMidlet midlet) {
        super(suppressKeyEvents);
        this.midlet = midlet;
        jogo = new Controle(this);
        g = getGraphics();
        comporInterface();
        jogo.definirAlvos();
}

ublic  void comporInterface(){
        inicializarTabuleiro();
        desenharTabuleiro(g);
        antigaPosCursor = atualPosCursor = 0;
        celulas[0][0].setCursor(true);
        celulas[0][0].pintar(g);
}

private void inicializarTabuleiro(){

 this.fonte = Font.getFont(Font.FONT_STATIC_TEXT);
 this.dica = dicas[0];
 this.celulas =
  new Celula[DIMENSAO_TABULEIRO][DIMENSAO_TABULEI-
RO];

 Celula celula = null;

  dimensao =  (larguraTela - 2)/DIMENSAO_TABULEIRO ;

 for(int i=0;i<DIMENSAO_TABULEIRO;i++){
   for (int j = 0; j < DIMENSAO_TABULEIRO; j++){
    celula = new Celula(COR_MEDIA,
                        dimensao - INTERVALO_CELU-
LAS,
                        Celula.OCULTA_VAZIA);

    celula.setY(i*dimensao +
                        BatalhaCanvas.Y_POSICAO_
TABULEIRO);
    celula.setX(j*dimensao +
                        BatalhaCanvas.X_POSICAO_
TABULEIRO);

    celulas[i][j] = celula;
    celula = null;
    }
  }
}

private void desenharTabuleiro(Graphics g){
 g.setColor(COR_TABULEIRO);
 g.fillRect(0, 0, larguraTela, alturaTela);
for (int i=0; i<BatalhaCanvas.DIMENSAO_TABULEIRO; i++){
for (int j=0;j<BatalhaCanvas.DIMENSAO_TABULEIRO;j++)
{
      celulas[i][j].pintar(g);
  }
 }
 exibirMensgensNaTela("");
}

protected void keyPressed(int keyCode){

  int gameAction = getGameAction(keyCode);
     switch (gameAction){
       case UP:    moverCursor(-1, 0);  break;
       case LEFT:  moverCursor(0, -1);  break;
       case RIGHT: moverCursor(0, 1);   break;
       case DOWN:  moverCursor(1, 0);   break;
       case FIRE:  executarJogada();    break;
    }
}

protected void keyRepeated(int keyCode){
   if (!hasRepeatEvents()) return;
   int gameAction = getGameAction(keyCode);
     switch(gameAction){
           case UP:
           case DOWN:
           case LEFT:
           case RIGHT:
           keyPressed(keyCode);
       }
 }

private void executarJogada(){
  int linha, coluna;
  linha =  (int) atualPosCursor/DIMENSAO_TABULEIRO;
  coluna = atualPosCursor % DIMENSAO_TABULEIRO;

  if (celulas[linha][coluna].estaDisponivel()){
     String aviso = "";
     this.dica  = dicas[0];
     jogo.incContadorJogadas();
     if (celulas[linha][coluna].temAlvo()){
       celulas[linha][coluna].
                        setStatus(Celula.EXPLICI-
TA_ALVO);
       aviso = " >>Acertou!";
       jogo.incAcertos();
       dica = dicas[0];
     }

     else{
      celulas[linha][coluna].
                        setStatus(Celula.EXPLICI-
TA_VAZIA);
      aviso = " >>Terra!";
         if (jogo.temAlvoVizinho(linha, coluna))
             dica = dicas[1];
         else dica = dicas[0];
```

```
   }

   celulas[linha][coluna].pintar(g);
   exibirMensgensNaTela(aviso);
   repaint();

  if (jogo.acabou())
      midlet.finalizarJogo(jogo.getStatusJogador());

   }
}

private void apagarDica(){

g.setColor(COR_TABULEIRO);
int altura_da_fonte = this.fonte.getBaselinePosi-
tion();

int largura_da_string = fonte.stringWidth(this.
dica);

g.fillRect(X_POS_DICA,
                    Y_POS_DICA - altura_da_fonte,
                    largura_da_string,
                    alturaTela - Y_POS_DICA);

        repaint(X_POS_DICA,
                    Y_POS_DICA - altura_da_fonte,
                    largura_da_string,
                    alturaTela - Y_POS_DICA);

this.dica = dicas[0];

}

private void exibirMensgensNaTela(String msg){
g.setColor(COR_TABULEIRO);
g.fillRect(0, 0, larguraTela, 15);

int altura_da_fonte = fonte.getBaselinePosition();
int largura_da_string = fonte.stringWidth(this.
dica);

g.fillRect(0,
                    Y_POS_DICA - altura_da_fonte,
            largura_da_string,
                 alturaTela - Y_POS_DICA);

g.setColor(COR_MENSAGENS);

g.drawString("Acertos: " +
              jogo.getAcertos() +
               msg + "  Resta(m) " +
               jogo.getJogadasRestantes() +
               " jogada(s)" , 0, 0,
               Graphics.TOP|Graphics.LEFT);

g.drawString(this.dica,
                     X_POS_DICA,
             Y_POS_DICA,
             Graphics.BOTTOM|Graphics.LEFT);
}

private void moverCursor(int deltaLinha, int delta-
Coluna){

 apagarDica();
 int novaPosCursor = atualPosCursor +
        deltaColuna + 10 * deltaLinha;

  if ((novaPosCursor >= 0) &&
                (novaPosCursor < DIMENSAO_TABULEI-
RO*
                        DIMENSAO_TABULEIRO)){

      antigaPosCursor = atualPosCursor;
      int l = (int)antigaPosCursor/10;
      int c = antigaPosCursor % 10;
      celulas[l][c].setCursor(false);
      celulas[l][c].pintar(g);

      repaint(celulas[l][c].getX(),
              celulas[l][c].getY(),
              dimensao, dimensao);

      atualPosCursor = novaPosCursor;

      int l2 = (int)novaPosCursor/10;
      int c2 = novaPosCursor % 10;
      celulas[l2][c2].setCursor(true);
      celulas[l2][c2].pintar(g);

      repaint(celulas[l2][c2].getX(),
              celulas[l2][c2].getY(),
              dimensao, dimensao);

      exibirMensgensNaTela("");
    }
 }

public  Celula getCelula(int l, int c){
      return this.celulas[l][c];
 }

}

/*****************************************
Classe Celula
*****************************************/

public  class Celula {
static final int OCULTA_VAZIA    = 0;
static final int OCULTA_ALVO     = 1;
static final int EXPLICITA_VAZIA = 2;
static final int EXPLICITA_ALVO  = 3;

private static final int ESPESSURA_CURSOR = 1;
private int COR_EXPLICITA_VAZIA = BatalhaCanvas.
COR_TABULEIRO;

private final int COR_EXPLICITA_ALVO = 0x00CCFF;
private final int COR_CURSOR  = 0xDDCC66;

private boolean cursorOn;
private int status;
private int cor;

private int dimensao;
private int x = 0;
private int y = 0;

public  Celula(int cor, int dimensao, int status) {
        this.cor = cor;
        this.dimensao = dimensao;
        this.status   = status;
}

public  void setCursor(boolean c){
        this.cursorOn = c;
}

public  void pintar(Graphics g){
  switch(this.status) {
     case OCULTA_VAZIA:
     case OCULTA_ALVO: break;

     case EXPLICITA_VAZIA:
         setColor(COR_EXPLICITA_VAZIA);
     break;

     case EXPLICITA_ALVO:
        setColor(COR_EXPLICITA_ALVO);
     break;
 }

  g.setColor(this.cor);
  g.fillRect(x, y, dimensao, dimensao);
```

```
 if (cursorOn){
    g.setColor(COR_CURSOR);
    g.drawRect(x, y,
                dimensao - ESPESSURA_CURSOR,
                dimensao - ESPESSURA_CURSOR);
    }
}

public  boolean estaDisponivel(){
 if (this.status == OCULTA_VAZIA ||
     this.status == OCULTA_ALVO)
     return true;

 return false;
}

public  boolean temAlvo(){
   if (this.status == OCULTA_ALVO) return true;
   return false;
 }

}

/*****************************************
Classe Controle
*****************************************/
public class Controle {

static final int TOTAL_ALVOS = 5;
static final int MAX_JOGADAS = 30;
private BatalhaCanvas batalhaCanvas;
private int acertos;
private int contador_jogadas;
private boolean ganhou;

public  Controle(BatalhaCanvas batalhaCanvas) {
        this.batalhaCanvas = batalhaCanvas;
        this.acertos = 0;
        this.contador_jogadas = 0;
        this.ganhou = false;
   }

public  void definirAlvos(){
  int alvo = -1;
  int k = 0;
  int linha, coluna;
  Random random = new Random();
  Celula celula = null;

int faixa = BatalhaCanvas.DIMENSAO_TABULEIRO * Bata-
lhaCanvas.DIMENSAO_TABULEIRO;

while (k < TOTAL_ALVOS){
 alvo = random.nextInt(faixa);
 linha = (int)(alvo/
                        BatalhaCanvas.DIMENSAO_TA-
BULEIRO);
 coluna = (int)( alvo %
                        BatalhaCanvas.DIMENSAO_TA-
BULEIRO);

 celula = batalhaCanvas.getCelula(linha, coluna);

  if (celula.getStatus() == Celula.OCULTA_VAZIA) {
       celula.setStatus(Celula.OCULTA_ALVO);
       k++;
  }
 }
   celula = null;
}


public  boolean acabou(){
    if (acertos == TOTAL_ALVOS){
        ganhou = true;
        return true;
    }
    if (contador_jogadas == MAX_JOGADAS){
        ganhou = false;
        return true;
    }

        return false;
}


public  boolean temAlvoVizinho(int l, int c){

     if(avaliarPosicao(l, c + 1))
                return true;
     if(avaliarPosicao(l - 1, c + 1))
                return true;
     if(avaliarPosicao(l + 1, c + 1))
                return true;
     if(avaliarPosicao(l + 1, c))
                return true;
     if(avaliarPosicao(l + 1, c - 1))
                return true;
     if(avaliarPosicao(l, c - 1))
                return true;
     if(avaliarPosicao(l - 1, c - 1))
                return true;
     if(avaliarPosicao(l - 1, c))
                return true;

      return false;
 }

private boolean avaliarPosicao(int a, int b){
  int status_celula = -1;

  try{
    status_celula =
                batalhaCanvas.getCelula(a,
b).getStatus();
        }catch(ArrayIndexOutOfBoundsException
aiobe){
                return false;
        }
   if (status_celula == Celula.OCULTA_ALVO)
                return true;

        return false;
}

public  void incContadorJogadas(){
   this.contador_jogadas++;
}

 public  void incAcertos(){
  this.acertos++;
 }

 public  int getJogadasRestantes(){
   return Controle.MAX_JOGADAS - contador_jogadas;
 }

public  boolean getStatusJogador(){
   return this.ganhou;
 }
}
```