UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOGO DE ESTRATÉGIA MULTI-JOGADOR PARA TELEFONES CELULARES

TRABALHO DE GRADUAÇÃO

N° 222

Fernando Bevilacqua

Santa Maria, RS, Brasil

2007

JOGO DE ESTRATÉGIA MULTI-JOGADOR PARA TELEFONES CELULARES

por

Fernando Bevilacqua

Trabalho de Graduação apresentado ao Curso de Ciência da Computação
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profª Andrea Schwertner Charão

Co-orientador: Profª Iara Augustin

Santa Maria, RS, Brasil

2007

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

JOGO DE ESTRATÉGIA MULTI-JOGADOR PARA TELEFONES CELULARES

elaborado por
Fernando Bevilacqua

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Profª Andrea Schwertner Charão
(Presidente/Orientador)

Prof. Cesar Tadeu Pozzer (UFSM)

Prof. João Carlos Damasceno Lima (UFSM)

Santa Maria, 2 de março de 2007.

DEDICATÓRIA

À minha mãe, ao meu pai e à minha irmã.

AGRADECIMENTOS

Agradeço primeiramente a Deus por me possibilitar estar nessa condição, escrevendo esse trabalho. Agradeço a Ele pela família maravilhosa que me deu, que sem a qual eu não teria chegado onde cheguei.

Agraceço a minha mãe, meu pai e minha irmã, por me ajudarem e darem força nos momentos difíceis. De nada me serve o conhecimento se a minha vida não tivesse sido abençoada com vocês. Obrigado!

Agradeço a minha orientadora, a Profª Andrea Schwertner Charão, por sua grande ajuda no desenvolvimento desse trabalho. Ela me mostrou que um orientador não deve ter só conhecimento e competência, mas compreensão, paciência e bom senso. Obrigado, professora, por me ajudar e entender que o trabalho de graduação não era a minha única prioridade. Agradeço, também, a minha co-orientadora, a Profª Iara Augustin, que soube ajudar não somente com o trabalho de graduação, mas também com assuntos acadêmicos diversos.

O meu obrigado a todos os meus colegas e amigos que, de uma forma ou de outra, me ajudaram com o trabalho. Um obrigado especial a aqueles que me ajudaram a fazer o Latex compilar o texto de uma maneira aceitável (valeu, Daronco). Obrigado também a todos os meus professores que me deram o conhecimento necessário para que esse trabalho fosse escrito.

Obrigado também a meus colegas de trabalho, que sempre estavam dispostos a conversar sobre a composição de capítulos, normas do MDT, número de referências e outros vários assuntos. Grato por me ajudarem com prazos para entrega das avaliações e dos documentos quando eu estava concentrado com outras tarefas.

Finalmente, meu obrigado às pessoas que me ajudaram e por falta de ajuda de minha memória, não relato aqui.

“Concede-me, Senhor, a serenidade necessária
para aceitar as coisas que não posso modificar,
coragem para modificar aquelas que posso
e sabedoria para distinguir uma das outras.”

ORAÇÃO DA SERENIDADE

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

JOGO DE ESTRATÉGIA MULTI-JOGADOR PARA TELEFONES CELULARES

Autor: Fernando Bevilacqua
Orientador: Profª Andrea Schwertner Charão
Co-orientador: Profª Iara Augustin

Local e data da defesa: Santa Maria, 2 de março de 2007.

As aplicações para dispositivos móveis evoluíram ao longo do tempo, fazendo com que aumentasse a demanda por aplicativos mais complexos, como jogos multi-jogador. A deficiência de meios de comunicação sofisticados implementados na maioria dos dispositivos e, também, o estado ainda pouco consolidado de recursos em plataformas de desenvolvimento para a criação de jogos, especialmente os multi-jogador, é um problema corrente. Neste sentido, as experiências de análise e utilização das ferramentas existentes constituem bases importantes para o avanço e a disseminação de conhecimentos nesta área. Baseado nisso, este trabalho se propõe a levantar subsídios informacionais sobre a plataforma J2ME através da implementação de um jogo multi-jogador como caso prático de uso dessa plataforma, a fim de prover informações úteis a outros desenvolvedores sobre a criação de jogos através dessa plataforma. Fundamentado nas experiências vivenciadas durante o desenvolvimento do caso de uso, o trabalho mostra as vantages, desvantagens e peculiaridades da plataforma J2ME no que se refere à criação de jogos para dispositivos móveis ao estilo do jogo implementado. Traça-se, também, um paralelo das vantagens e desvantagens do J2ME com outras duas plataformas, BREW e Flash Lite, a fim de enriquecer a análise. Além disso, apresenta-se durante o trabalho, e executa-se na implementação do caso de uso, técnicas ligadas à redução do consumo de recursos, como otimização de imagens e obscurecimento de código, a fim de atenuar o problema de desenvolver aplicações para ambientes com pouca memória e baixo poder de processamento.

Palavras-chave: Jogo multi-jogador; Telefone celular; J2ME; Multiplayer; Mobile; Game; Flash Lite; BREW.

ABSTRACT

Trabalho de Graduação
Course of Computer Science
Universidade Federal de Santa Maria

STRATEGY MULTIPLAYER GAME FOR MOBILE DEVICES
Author: Fernando Bevilacqua
Advisor: Prof[a] Andrea Schwertner Charão
Coadvisor: Prof[a] Iara Augustin

The applications for mobile devices have increased their complexity level over the years, resulting in more sophisticated software, like multiplayer games. The inability or absence of robust network connection methods and the non-strongly based state of resources for games development in mobile plataforms is a current problem. In this way, the experience with game development using existing tools is an important topic to continue the evolution of such development area. Surrounded by this scenario, this work intends to raise useful information about J2ME development experience, through the implementation of a real game as a practical use case of the plataform. Based on the experiences resulted of the implementation of the practical use case, this work shows the advantages, disadvantages and characteristics of J2ME, concerning multiplayer games development. This work also compares the advantages and disadvantages of J2ME with other two development platforms, BREW and Flash Lite, intending to improve the J2ME analysis. During the implementation of the use case, optimization techniques are used to reduce the game memory usage; those techniques are focused in image optimization and code obfuscation.

Keywords: Multiplayer game; Mobile; J2ME; Multiplayer; Game; Flash Lite; BREW.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Os jogos para telefones celulares estão evoluindo em um ritmo acelerado nos últimos anos. Como inicialmente os aparelhos possuíam poucos recursos, os primeiros jogos eram rudimentares e pouco numerosos. Com a crescente incorporação de características aos aparelhos, os jogos para celular foram se tornando mais complexos, com gráficos e sons mais elaborados e uma interatividade maior do que seus antecessores, possibilitando inclusive a interação entre múltiplos jogadores. A demanda e a oferta de jogos também aumentaram, de modo que atualmente o mercado de jogos para celulares constitui uma área rentável e em plena expansão (WISNIEWSKI et al., 2005).

Mesmo com os recentes avanços em jogos para celulares, percebe-se que os jogos multi-jogador ainda não são amplamente explorados nestes dispositivos móveis. Dentre os motivos que podem explicar esse fato, estão a deficiência de meios sofisticados para conexões entre aparelhos (apenas os protocolos mais básicos de comunicação estão disponíveis na maioria dos dispositivos) e o estado ainda pouco consolidado das plataformas de desenvolvimento deste tipo de jogo. Neste sentido, as experiências de análise e utilização das ferramentas existentes constituem subsídios importantes para o avanço e a disseminação de conhecimentos nesta área.

Considerando o cenário acima delineado, este trabalho visa o desenvolvimento de um jogo de estratégia multi-jogador para telefones celulares, utilizando uma das principais ferramentas de desenvolvimento para dispositivos móveis: a plataforma J2ME (Java 2 Micro Edition) (SUN MICROSYSTEMS, 2006). O jogo escolhido para este estudo de caso é o Stratego (WIKIPEDIA, 2006), um jogo de estratégia popular e, até onde foi possível constatar, ainda inexistente para telefones móveis. Com esse trabalho, pretende-se reunir informações relevantes sobre o desenvolvimento desse tipo de aplicação nessa plataforma, a fim de contribuir com outros desenvolvedores que atuam nesta área. Além

disso, os resultados deste trabalho serão úteis para a empresa Decadium Studios, situada na Incubadora Tecnológica de Santa Maria. Esta empresa já atua no desenvolvimento de jogos digitais para microcomputadores e, futuramente, com base neste trabalho, poderá estender suas atividades ao mercado de jogos para celulares.

Este trabalho de graduação apresenta a seguinte organização: o Capítulo 2 discute a temática dos multi-jogador para telefones celulares, apresentando as vantagens deste tipo de jogo, as características dos aparelhos que têm impacto na execução desses jogos, as plataformas de desenvolvimento e alguns trabalhos recentes nesta área. O Capítulo 3 apresenta uma visão geral do jogo e a forma como ele foi planejado e desenvolvido, destacando-se as principais decisões tomadas em cada etapa de implementação. O Capítulo 4 contém uma análise da plataforma de desenvolvimento J2ME, salientando o consumo de recursos por parte do jogo e técnicas para amenizar seu impacto, as vantagens e desvantagens da plataforma constatadas durante o desenvolvimento de Stratego e um paralelo de comparação com outras duas plataformas: Flash Lite e BREW. Finalmente o Capítulo 5 apresenta a conclusão do presente trabalho, frisando as experiências de desenvolvimento e resultados obtidos durante e após a implementação do jogo, além de apresentar sugestões de trabalhos futuros.

2 JOGOS MULTI-JOGADOR EM TELEFONES CELULARES

2.1 Vantagens dos jogos multi-jogador

A utilização de telefones celulares se tornou algo comum na sociedade moderna. O celular passou a integrar o conjunto de objetos que as pessoas carregam quando deixam suas casas para ir ao trabalho, ao supermercado ou ao cinema. Além de serem usados para comunicação, os aparelhos atuais possuem várias outras funções, incluindo formas de entretenimento. Para uma pessoa que espera na fila do cinema, por exemplo, o tempo pode "passar mais rápido" se ela mantiver sua atenção voltada para um jogo em seu celular. Pessoas que dispõem de tempo livre e que desejam se divertir através de seus celulares não precisam mais ir a uma loja para comprar um cartão com um jogo: elas podem efetuar *download* de novos jogos para os seus aparelhos. As possibilidades de entretenimento são grandes, porém há um problema: como fazer para que um jogo continue atrativo aos jogadores?

A maioria dos jogos para um único jogador (*singleplayer*) apresenta adversários com inteligência artificial, programados para imitar o comportamento humano e aumentar o nível de desafio no jogo. Esses adversários, porém, são baseados em regras fixas e, não raramente, têm seus movimentos previstos pelo jogador, à medida que esse conhece as táticas utilizadas. Com o tempo, os inimigos com inteligência artificial passam a ser previsíveis em demasia, o que acaba com o desafio que o jogador procura (HEDLUNG; MAXE, 2004), fazendo com que esse busque outro jogo.

Em contrapartida, jogos multi-jogador (*multiplayer*) são desenvolvidos para que mais de uma pessoa possa interagir ao mesmo tempo, o que aumenta as possibilidades de desafio. Se duas ou mais pessoas se confrontarem em um jogo, cada uma delas, provavelmente, apresentará o seu estilo de jogo e utilizará táticas diferentes para alcançar os ob-

jetivos que o jogo propõe. Diferentemente de um agente programado com inteligência artificial, um jogador humano pode facilmente utilizar seus conhecimentos prévios para formular táticas novas com base na sua análise dos oponentes, reagindo de forma variada ao longo do tempo. O jogador busca o aprimoramento de suas táticas, a fim de que essas se tornem melhores que as dos adversários. Dessa forma, as pessoas envolvidas em uma partida multi-jogador são desafiadas de forma contínua, o que pode aumentar a sua diversão e vontade de continuar jogando.

Além disso, o comportamento social do ser humano pode fazer com que uma pessoa que está sozinha com seu celular se sinta confortável ao interagir com outras pessoas que estão na mesma situação (HEDLUNG; MAXE, 2004). Essas pessoas podem gastar seu tempo em um jogo multi-jogador no qual devem, por exemplo, agir em conjunto para atingir um objetivo comum. Essa interação não poderia ser realizada em jogos *singleplayer*.

Através de sua constante evolução e aumento de conhecimento, as pessoas aprimoram suas táticas de jogo e se tornam desafiantes em potencial para outros jogadores. A interação entre eles pode garantir uma seção de jogo divertida sem que o jogo apresente uma lógica complexa, com alto custo de CPU, para implementar inimigos com inteligência artificial que imitem humanos. Se um usuário gostar de um jogo *multiplayer* e encontrar adversários que o desafiem e o mantenham entretido, ele pode jogar o mesmo jogo, divertindo-se, por muito tempo.

2.2 Características dos aparelhos

Em relação às características de *hardware* dos telefones celulares, há várias limitações que inexistem quando se desenvolve jogos para a plataforma PC. A primeira delas é o tamanho reduzido da tela. Se um jogo apresentar um cenário muito grande, torna-se necessário o uso de barras de rolagem, caso contrário os elementos que serão mostrados na tela serão pequenos demais para que o jogador possa distingui-los. A utilização de barras de rolagem, porém, pode tornar o jogo pouco ergonômico, uma vez que elas limitam a capacidade de resposta do jogador às situações do jogo.

Outra limitação é a bateria dos aparelhos. O uso intensivo de CPU, algo comum em jogos, pode descarregar rapidamente a bateria em um dispositivo móvel (NOKIA CORPORATION, 2003a). Em jogos para PC, a bateria não é um fator de risco e, salvo raras exceções, não é o ponto crítico do projeto. Além da limitação de energia, existem

limitações sérias em relação à memória e à capacidade de processamento. Embora as características dos aparelhos evoluam constantemente, é comum encontrar exemplos de dispositivos com processadores de 16 bits e 160Kb a 512Kb de memória total disponível.

No que concerne à capacidade de comunicação dos dispositivos, esta caracteriza-se por uma alta latência e uma baixa taxa de transmissão de dados (HÄGGLUND, 2004). Em dispositivos GSM (*Global System for Mobile Communication*), por exemplo, a taxa de transmissão é limitada a 9,6kbits ou 14,4kbits por segundo e por *slot* de tempo, dependendo do que é provido pelo aparelho e pela operadora (LILJESTAM, 2004). A latência para comunicação entre os aparelhos, por sua vez, pode variar de 750 a 1050 milissegundos (HÄGGLUND, 2004). Essas características limitam as oportunidades para jogos multi-jogador, tornando proibitiva a transferência de uma grande quantidade de dados.

Quanto ao *software* básico, os dispositivos móveis apresentam um sistema operacional próprio, responsável por prover recursos multimídia, gráficos, serviços relacionados a manipulação de dados, etc. Dentre os sistemas operacionais apropriados para jogos, destacam-se o Symbian e o Windows Mobile (HEDLUNG; MAXE, 2004). O Symbian é fruto de uma iniciativa conjunta de fabricantes de celulares para criar um sistema operacional com suporte a recursos multimídia, comunicação de dados e interfaces gráficas de qualidade. No desenvolvimento de jogos para este sistema operacional, as linguagens comumente utilizadas são Java e C++. O Windows Mobile é a versão para dispositivos móveis do consolidado sistema operacional Windows, da Microsoft. Esse sistema operacional apresenta certa compatibilidade com as aplicações que são desenvolvidas para o Windows voltado a PCs. A linguagem de programação nativa para o Windows Mobile é C++.

2.3 Plataformas de desenvolvimento

Seguindo o crescimento tecnológico dos dispositivos portáteis, que passaram a fornecer mais recursos para a execução de aplicações, os jogos para celular evoluíram para aplicações mais sofisticadas, fazendo uso de recursos como gráficos 3D, sons e comunicação em rede. Cada uma dessas características, porém, é dependente do *hardware* onde o jogo se encontra, como quantidade de memória e capacidade de processamento disponíveis.

Para aumentar o nível de abstração entre as peculiaridades de cada dispositivo e o jogo em si, uma camada de *software* é inserida entre esses dois elementos: a plataforma de de-

senvolvimento. Em síntese, as plataformas para desenvolvimento de *software* para telefones móveis podem ser classificadas em dois grandes grupos: as plataformas baseadas na linguagem C e as plataformas baseadas na linguagem Java.

As seções que seguem apresentam, respectivamente, uma breve descrição da plataforma BREW, que é baseada em C/C++, uma descrição da plataforma Flash Lite, que é uma das plataformas mais recentes, e uma descrição mais detalhada da plataforma J2ME, que é utilizada neste trabalho.

2.3.1 BREW

A plataforma BREW (*Binary Runtime Environment for Wireless*) (QUALCOMM INCORPORATED, 2006) foi criada e lançada em 2001 pela empresa Qualcomm. Esta plataforma destina-se ao desenvolvimento de *software* para aparelhos celulares com tecnologia CDMA (*Code Division Multiple Access*). Nessa plataforma, os desenvolvedores podem escrever aplicações na linguagem nativa (C/C++). Estas aplicações serão compiladas e executadas em um *chip* à parte do processador principal do aparelho, o que garante um maior desempenho de execução. Além de C/C++, o programador também pode utilizar outra linguagem cujo ambiente de execução seja suportado, como Java e Flash (ADOBE SYSTEMS INCORPORATED, 2006a).

Antes do lançamento dessa plataforma, os desenvolvedores de aplicações para celular necessitavam programar características específicas para cada aparelho (BARROS, 2005). Com o lançamento da BREW, a Qualcomm uniformizou e padronizou o ambiente de execução das aplicações em diferentes dispositivos. Essa unificação se deu também no modelo de negócios adotado pela Qualcomm, que passou a oferecer recursos para divulgação e venda dos jogos. Nesse modelo, os jogos criados por desenvolvedores são primeiramente enviados à Qualcomm, que realiza testes sobre o jogo. Depois que a aplicação está testada e possui garantia de funcionamento e qualidade comprovada, ela é enviada para comercialização na rede de distribuição, composta pelas operadoras de telefonia celular filiadas à Qualcomm.

Os desenvolvedores de jogos que utilizam a BREW ficam obrigatoriamente vinculados ao modelo de negócios da Qualcomm e das operadoras da rede de distribuição. Embora esse modelo de negócios diminua a porcentagem de pequenos desenvolvedores investindo nessa plataforma, há algumas vantagens tanto para o usuário final quanto para

a empresa que veicula seus jogos. A principal delas é que o *download* de todos os jogos é efetuado através da rede de distribuição, que é um ambiente comprovado e seguro, no qual o usuário pode confiar na hora de realizar suas compras. Esse meio permite, também, um controle maior sobre o que é instalado nos aparelhos celulares, o que previne a pirataria de jogos.

2.3.2 Flash Lite

A plataforma Flash Lite (ADOBE SYSTEMS INCORPORATED, 2006b) é uma parte da plataforma Flash, da empresa Adobe Systems Incorporated, voltada a dispositivos móveis. Ao contrário do Flash Player, responsável pela execução de conteúdo Flash em microcomputadores (ADOBE SYSTEMS INCORPORATED, 2006a), a plataforma Flash Lite oferece um ambiente mais otimizado em relação à utilização de recursos, como memória e processamento. Firmada sobre as capacidades do Flash Player 7, a plataforma Flash Lite, em sua versão 2.1, apresenta recursos como comunicação por *sockets*, manipulação de XML, utilização de gráficos vetoriais e recursos multimídia (como visualização de imagens, áudio e vídeo), armazenamento persistente de dados e suporte a Unicode (ADOBE SYSTEMS INCORPORATED, 2007).

Conforme análise divulgada por (LINSALATA; SLAWSBY, 2005), a barreira criada entre desenvolvedores inexperientes ou com pouco conhecimento em desenvolvimento de aplicações para dispositivos móveis pode ser contornada com o uso do Flash Lite (LINSALATA; SLAWSBY, 2005). Fazendo uso do ambiente integrado para desenvolvimento de conteúdo Flash, desenvolvedores que já possuem conhecimento e habilidades na plataforma para PC podem, também, utilizar esse conhecimento para desenvolverem conteúdo para dispositivos móveis. Essa redução da quantidade de conhecimento técnico necessário para o desenvolvimento de aplicações para aparelhos móveis aumenta o número de potenciais desenvolvedores que possam utilizar a plataforma, como é o caso de *designers* com conhecimento na plataforma Flash (LINSALATA; SLAWSBY, 2005).

2.3.3 J2ME

A plataforma J2ME (SUN MICROSYSTEMS, 2006) é um conjunto de tecnologias e especificações voltadas para o desenvolvimento de aplicações para sistemas embarcados com recursos limitados, como celulares e PDAs (*Personal Digital Assistants*) . Esta

plataforma compreende interfaces de usuário, modelos de segurança, protocolos de comunicação em rede e outras funcionalidades que, quando combinadas, constituem um ambiente de execução Java otimizado para uso de memória, processamento e operações de entrada e saída.

Lançada em 1999, a J2ME é composta basicamente de duas camadas dispostas sobre a máquina virtual Java: a camada de configurações (*configurations*) e a camada de perfis (*profiles*). A primeira se refere à configuração mínima exigida por um conjunto de dispositivos que apresentem características de *hardware* semelhantes. Aparelhos fortemente distintos, como geladeiras e telefones celulares, têm suas características agrupadas e definidas em diferentes configurações. A camada de perfis provê o suporte para as necessidades específicas de dispositivos de uma determinada família, garantindo interoperabilidade das aplicações entre estes aparelhos.

2.3.3.1 Configurações

Uma configuração define a plataforma Java para um grupo de aparelhos com características de *hardware* semelhantes. Estão incluídas nessa definição os recursos da linguagem Java, os recursos da JVM e, também, as APIs que são suportadas. Existem dois tipos de configurações atualmente definidas: a CLDC (*Connected Limited Device Configuration*) e a CDC (*Connected Device Configuration*). A primeira define uma parcela da plataforma Java destinada a aparelhos com restrições significativas de memória e processamento, enquanto a segunda é voltada para aparelhos com maior capacidade de processamento e maior disponibilidade de recursos.

A CLDC apresenta várias limitações nas funcionalidades do Java em comparação à plataforma J2SE (*Java 2 Standard Edition*), como inexistência de números de ponto-flutuante, limitações para manusear e tratar erros (a aplicação não pode se recuperar de alguns erros devido a especificidades de *hardware*), nenhum suporte a JNI (*Java Native Interface*) ou a reflexão. Além dessas limitações, as APIs disponíveis contêm menos classes que as APIs do J2SE, uma vez que elas ocupam uma grande quantidade de memória que não está disponível no ambiente dos dispositivos. Apenas um grupo reduzido e essencial de classes está disponível.

A CDC também possui limitações, mas como destina-se a dispositivos com uma quantidade maior de recursos, tais limitações são menores. Exemplo disso é que a CDC suporta

importantes funcionalidades da linguagem Java, como números de ponto flutuante, JNI, RMI (*Remote Method Invocation*) e reflexão.

2.3.3.2 Perfis

Como já mencionado, os perfis oferecem o suporte necessário para um família específica de dispositivos. Cada configuração pode apresentar diferentes tipos de perfis.

A CLDC apresenta, atualmente, apenas um perfil, chamado MIDP (*Mobile Information Device Profile*). O MIDP é o perfil destinado a dispositivos portáteis, como celulares e PDAs, apresentando algumas características como:

- suporte a um subconjunto do protocolo HTTP, implementado sobre protocolos baseados em IP (TCP/IP, por exemplo) ou não baseados em IP (WAP, por exemplo);
- mecanismos para armazenamento persistente de dados no dispositivo;
- definição de um modelo de aplicação.

A CDC também apresenta, atualmente, apenas um perfil, o *Foundation Profile*. Esse perfil apresenta uma implementação completa das funcionalidades da J2SE, mas com algumas modificações. A mais notável delas é o tamanho reduzido das telas (*displays*), o que torna dispensável o uso de pacotes gráficos complexos relacionados a interfaces de usuário, como por exemplo o pacote `java.awt`.

2.3.3.3 Comunicação

A API para comunicação disponível na plataforma J2ME é provida pelo *Generic Connection Framework (GCF)* (ORTIZ, 2003). O GCF surgiu para uso no CLDC 1.1 como alternativa aos pacotes `java.net` e `java.io`, ambos pertencentes à família J2SE, porque esses pacotes são grandes em demasia para serem usados em ambientes de pouca memória, como é o caso de dispositivos móveis.

O GCF é composto por uma hierarquia de classes e interfaces, as quais definem métodos de conexão e execução de I/O. No topo da hierarquia, encontra-se a interface `Connection`, que é estendida por todas as demais classes. Descendo-se na hierarquia, as classes tornam-se mais complexas, específicas para determinado tipo de conexão, porém uma fábrica de conexões garante a uniformidade de acesso. Através de uma URL, o programador pode criar um canal de comunicação sobre um dos tipos de conexões

disponíveis, utilizando a fábrica de conexões, expressa pela chamada de `Connection.open(URL)`. A URL é criada no formato `scheme://endereço:parâmetros`, onde `scheme` define o tipo de conexão a ser usado. A Tabela 2.1 mostra alguns dos `schemes` e tipos de conexão disponíveis (ORTIZ, 2003).

Tabela 2.1: Tipos de `schemes` e tipos de conexão do GCF

URL Scheme	Conectividade	Tipo de conexão GCF	Definido por
btl2cape	Bluetooth	L2CAPConnection	JSR 82. Suporte é opcional.
datagram	Datagram	DatagramConnection	Todos profiles baseados em CLDC e CDC, como MIDP, Foundation e relacionados; e por JSR 197, J2SE. Suporte é opcional.
file	File Access	FileConnection, InputConnection	JSR 75. Suporte é opcional.
http	HyperText Transport Protocol	HttpConnection	MIDP 1.0, MIDP 2.0, Foundation Profile, J2SE (JSR 197). Suporte é requerido.
https	Secure HTTP	HttpsConnection	MIDP 2.0. Suporte é requerido.
comm	Serial I/O	CommConnection	MIDP 2.0. Suporte é opcional.
sms mms cbs	Short Messaging Service	MessageConnection	JSR 120, JSR 205. Suporte é opcional.
apdu jcrmi	Security Element	APDUConnection, JavaCardRMIConnection	JSR 177. Suporte é opcional.
socket socketserver	Socket	SocketConnection, ServerSocketConnection	JSR118 (MIDP 2.0). Suporte é opcional.
datagram	UDP Datagram	UDPDatagramConnection	JSR118 (MIDP 2.0). Suporte é opcional.

Embora existam vários tipos de conexões disponíveis para uso, algumas das classes que as implementam não são requeridas nos dispositivos móveis como parte indispensável do MIDP ou de outro *profile*, como mostra a coluna *Definido por*, da Tabela 2.1. Mesmo que a disponibilidade de comunicação por HTTP e HTTPS seja requerida no MIDP 1.0 ou MIDP 2.0, o que garante ao programador a existência desses recursos em dispositivos

com tais *profiles*, a sua utilização pode não ser a melhor forma de comunicação para determindos casos.

A utilização de *sockets*, por exemplo, permite ao desenvolvedor uma comunicação de rede de baixo nível (MAHMOUD, 2003), uma vez que os dados transmitidos não estão vinculados a nenhum protocolo pré-definido. Sendo assim, um protocolo próprio pode ser criado, o que evita *overheads* de processamento de cabeçalhos de pedido-resposta textuais, como é o caso do HTTP, por exemplo. No caso de jogos, algumas informações podem ser perdidas durante a comunicação sem grandes prejuízos ao andamento do jogo. Para tirar proveito disso, o desenvolvedor pode basear a transmissão de dados sobre UDP, que não é orientado a conexão (não há garantia de entrega do datagrama), porém apresenta melhor desempenho em relação a TCP/IP (orientado a conexão), uma vez que não há processamento extra para estabelecimentos de conexões de verificação.

Outro tipo de conexão que não é obrigatória, porém aumenta as possibilidades de comunicação, é o Bluetooth. O Bluetooth é uma tecnologia de custo baixo, pouco consumo de energia e com curto alcance de rádio-frequência (de 1m a 100m), criado para substituir os cabos nas conexões entre dispositivos portáteis (KLINGSHEIM, 2004). A comunicação é feita em uma banda de 2.4 Gz, utilizando-se uma técnica de sinalização chamada Frequency Hopping Spread Spectrum (FHSS). Nessa técnica, a banda de rádio é divida em 79 sub-canais, nos quais o rádio Bluetooth alterna o uso a cada 625 milissegundos. Os saltos entre os sub-canais são usados para reduzir a interferência causada por dispositivos Bluetooth próximos uns dos outros. A Java APIs for Bluetooth Wireless Technology (JABWT) define um pacote J2ME, chamado `javax.bluetooth`, para comunicação utilizando Bluetooth. Em se tratando de jogos, dispositivos móveis podem formar uma rede *ad-hoc* através de Bluetooth para permitir que seus usuários possam participar de uma partida *multiplayer*, por exemplo. Conforme testes de Klingshein (2004), a taxa de transferência de dados entre dois dispositivos Nokia 6600, utilizando Bluetooth, foi de 10 a 11 KB/s. Ao contrário da comunicação por HTTP ou por *sockets*, a comunicação via Bluetooth não possui custos de transmissão de dados *over the air*, uma vez que a comunicação é feita diretamente entre um dispositivo e outro. Isso pode se tornar uma opção atrativa de comunicação entre jogadores que não desejam gastar recursos financeiros com a transmissão de dados entre seus dispositivos.

2.4 Exemplos de jogos e trabalhos relacionados

Os jogos multi-jogador para telefones celulares, bem como a análise entre plataformas de desenvolvimento, têm sido tema de trabalhos e publicações recentes. Alguns desses trabalhos foram analisados e são brevemente comentados a seguir.

O primeiro trabalho analisado foi o jogo BlueGame (RABELO; HAHN; BARBOSA, 2005). Trata-se de um jogo de combate que pode ser jogado por duas pessoas. Os jogadores controlam um personagem que possui pontos de vida (*Health Points*) e pontos de ação (*Action Points*), que indicam o combustível do personagem. A cada turno, os jogadores escolhem cartas que indicam o comportamento do personagem. As cartas possuem ações como combater, mover-se para cima, mover-se para baixo, etc., tendo um custo de ação associado. Dessa forma, o jogador pode realizar as ações que seus pontos de ação possibilitarem (cartas de movimento não possuem custo). O jogo foi implementado na linguagem C#, com a plataforma de desenvolvimento .NET Compact Framework. Este jogo utiliza um modelo *peer-to-peer* para comunicação, com transmissão de dados serial através da tecnologia Bluetooth. Não está dentro do escopo do presente trabalho realizar análises sobre a implementação de jogos utilizando-se a linguagem C#, porém uma análise sobre utilização de Bluetooth foi citada na seção 2.3.3.3.

O segundo trabalho analisado foi o jogo *The Right Way* (HEDLUNG; MAXE, 2004). Este é um jogo de estratégia que pode ser jogado por várias pessoas. Cada pessoa controla um personagem, que está posicionado em um labirinto. O labirinto é mostrado com uma visão do topo para baixo, porém todos os caminhos que o personagem ainda não percorreu estão escuros. À medida que o personagem caminha pelo labirinto carregando uma tocha, o caminha vai sendo descoberto. O objetivo do jogo é encontrar a saída do labirinto antes dos demais jogadores. O jogador pode ver os demais jogadores passeando pelo labirinto e é incentivado a explorar o cenário em busca não só da saída, mas também de elementos que o auxiliem a encontrá-la (um mapa, por exemplo). O jogo utiliza um modelo cliente-servidor para comunicação, através do protocolo HTTP 1.1. O cliente e o servidor foram implementados utilizando-se a linguagem Java. Sobre os recursos utilizados para o desenvolvimento de *The Right Way*, existe um estudo realizado por (KARLSSON et al., 2003), que mostra uma comparação entre as plataformas J2ME, Symbiam e BREW, fundamentada nas experiências de desenvolvimento do grupo de pesquisas CIn/CESAR, em relação ao uso dessas plataformas para o desenvolvimento de jogos. No trabalho, são mostradas

comparações entre as diversas características de cada uma das plataforma, como gerenciamento de I/O, processamento gráfico, processamento de som e conectividade. A análise possui um cunho técnico e é voltada, conforme informações do próprio texto, aos desenvolvedores de jogos para sistemas móveis, informando sobre peculiaridades encontradas em cada uma das plataformas, como disponibilidade de execução do pressionamento de mais de uma tecla do dispositivo simultaneamente, por exemplo.

O terceiro trabalho analisado é uma proposta de arquitetura que dá suporte ao cenário de pequenos jogos para dispositivos móveis (PEREIRA, 2006), utilizando Bluetooth para comunicação. Conforme explanado neste trabalho, os jogos multi-jogador para celulares são uma evolução dos jogos dessa categoria para PC, o que é comprovado pelo lançamento de iniciativas comerciais nesse sentido, como o aparelho Nokia NGage (NOKIA CORPORATION, 2006). Através de um paralelo com jogos MMOG (*Massively Multiplayer Online Game*), o autor cita que a união desses com jogos móveis cria um novo conceito de jogo, os MMMOGs (*Massively Mobile Multiplayer Online Games*). Outros estudos também foram realizados sobre jogos multi-jogador para dispositivos móveis, comprovando a existência da uma preocupação com tópicos dessa área, como é o caso de (SACRAMENTO; ROCHA; ESMIN, 2005), que propõe um *framework* para o desenvolvimento de jogos desse estilo (multi-jogador para dispositivos móveis). O objetivo principal do referido trabalho é proporcionar aos desenvolvedores de jogos um acesso transparente entre os aparelhos, a fim de que não haja preocupação em questões específicicas no processo de comunicação, como a forma com que as informações são trocas entre os dispositivos. O *framework* é desenvolvido em Java e provê um conjunto de funcionalidades sobre o protocolo HTTP, como envio de mensagens, validação por login e senha, validação de usuário por grupo, envio de requisições, obtenção de um lista de informações pertinentes (pontos, posição no *ranking*, apelido, etc).

3 JOGO DESENVOLVIDO

As seções seguintes apresentam uma visão geral do jogo de estratégia escolhido para estudo de caso neste trabalho, além da arquitetura da aplicação que implementa este jogo em telefones celulares.

3.1 Visão geral do jogo

Conforme mencionou-se anteriormente, o jogo escolhido para este trabalho foi o Stratego[1]. Neste jogo, cada um dos dois participantes possui um conjunto de peças ao seu dispor, cada uma delas com uma graduação (um número que indica a patente da peça). O jogador consegue ver apenas a graduação de suas peças, sendo que a graduação das peças inimigas permanece oculta. Cada vez que duas peças se encontram em uma mesma posição do tabuleiro, elas são mostradas a ambos os jogadores e a peça de maior graduação ganha, fazendo com que a peça perdedora deixe o tabuleiro (as duas peças saem em caso de empate). Há uma peça especial, chamada bandeira. Se um dos jogadores capturar a bandeira inimiga, ele é o vencedor. A Figura 3.1 ilustra o tabuleiro em uma sessão de jogo.

Em uma sessão de jogo, primeiramente os jogadores dispõem as suas peças da maneira que lhes convier estrategicamente. Em seguida, um dos jogadores inicia movendo uma peça. Depois que já realizou o movimento, ele passa a vez para que o outro jogador possa mover uma peça e assim por diante. Não há limite de tempo para uma sessão.

Stratego foi escolhido para este trabalho por motivos relacionados aos recursos de *hardware* necessários para suportá-lo:

1. O número de jogadores é reduzido a apenas duas pessoas. Tendo em vista que a

[1]Frisa-se que a utilização de Stratego é apenas um caso de uso para a análise da plataforma J2ME, logo as regras do jogo não são descritas em sua totalidade aqui. É possível encontrar mais informações sobre as regras em (WIKIPEDIA, 2006)

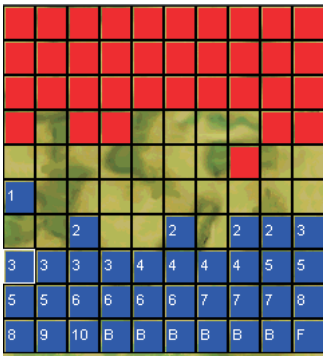

Figura 3.1: Tabuleiro com as peças organizadas para uma sessão de jogo de Stratego

comunicação de dados entre aparelhos celulares é custosa e lenta, manter um grande número de jogadores se comunicando ocasionaria uma sobregarga de comunicação muito grande ao aplicativo.

2. O jogo não necessita de comunicação em tempo real entre os jogadores, o que reduz consideravelmente a troca de informações entre os aparelhos celulares. Esse não seria o caso de um jogo de corrida, por exemplo, no qual a posição de um jogador na pista deveria ser atualizada e propagada a todos os jogadores a cada instante, a fim de sincronizar ações como ultrapassagens, colisões, etc. Para o jogo Stratego, importa apenas quando o jogador terminou sua ação e qual peça ele moveu. De posse dessa informação, o sistema pode calcular os acontecimentos aplicáveis e, então, passar a vez ao próximo jogador. Assim que esse jogador terminar suas ações e informar qual peça moveu, o ciclo se repete.

3. Não é necessária a utilização de grande quantidade de gráficos para tornar o jogo atrativo. Uma vez que Stratego é um jogo essencialmente de estratégia, acredita-se que um tabuleiro e um conjunto de peças bem elaborados graficamente sejam suficientes para garantir a atratividade do jogo. De fato, não há necessidade de animações em todos os momentos, como no caso de jogos de corrida, caçadas espaciais etc. Isso reduz a quantidade de imagens a serem utilizadas. Uma vez que a quantidade de memória disponível em aparelhos celulares é pequena, poupar memória em gráficos sem que o jogo perca qualidade é um aspecto importante.

3.2 Arquitetura da aplicação

Visando à escalabilidade do código, a aplicação foi dividida em três camadas: camada lógica, camada de comunicação e camada gráfica. A camada lógica contém todas as classes que implementam a lógica do jogo, como o que acontece quando duas peças se encontram, o que acontece quando a bandeira do adversário é capturada, etc. A camada de comunicação contempla as classes que implementam a troca de dados entre um celular e outro. Por fim, a camada gráfica abriga a classe que realiza qualquer desenho na tela. Somente essa classe está autorizada a desenhar na tela, carregar gráficos e executar funções desse tipo. Cada uma das camadas que compõem a aplicação são explanadas em detalhes nas seções seguintes.

3.2.1 Camada lógica

A camada lógica da aplicação tem por objetivo isolar e centralizar elementos lógicos da aplicação, vinculando-as com as demais camadas do jogo. Para tal função, essa camada necessitou de um elemento que pudesse controlar todas regras do jogo e, também, o ciclo de vida da aplicação, conforme as especificações do J2ME. Partindo do fato que o dispositivo no qual o jogo irá rodar possui baixo poder de processamento e pouca memória, conforme já citado, a camada lógica teve de prover uma abstração entre a lógica do jogo e o ciclo de vida da aplicação sem, no entanto, gerar um conjunto muito grande de classes para esse fim, visando reduzir o consumo de memória. Para centralizar as variáveis que controlam a aplicação, como sinalizadores de avisos, indicadores de pausa e afins, bem como o ciclo de vida do programa, criou-se a classe `StrategoEngine`, que é a base da aplicação. Para prover uma centralização das regras do jogo, especificamente, criou-se a classe `GameBoard`. Estas classes, e as suas respectivas classes-base, são apresentadas na figura 3.2.

A classe `StrategoEngine` constitui a base de uma aplicação MIDP, contendo os métodos necessários para controle de seu ciclo de vida e resposta aos avisos feitos pelo dispositivo, como interrupção do jogo em virtude de uma chamada telefônica. Estão contidas nessa classe todos os atributos importantes ao jogo e à aplicação, como estado atual da partida (pausada, jogando, dispondo peças no tabuleiro, etc.), informações sobre os dados recebidos do outro dispositivo, quem joga no turno corrente, etc. Todas essas informações são utilizadas pelas outras classes da aplicação para guiar o seu comportamento.

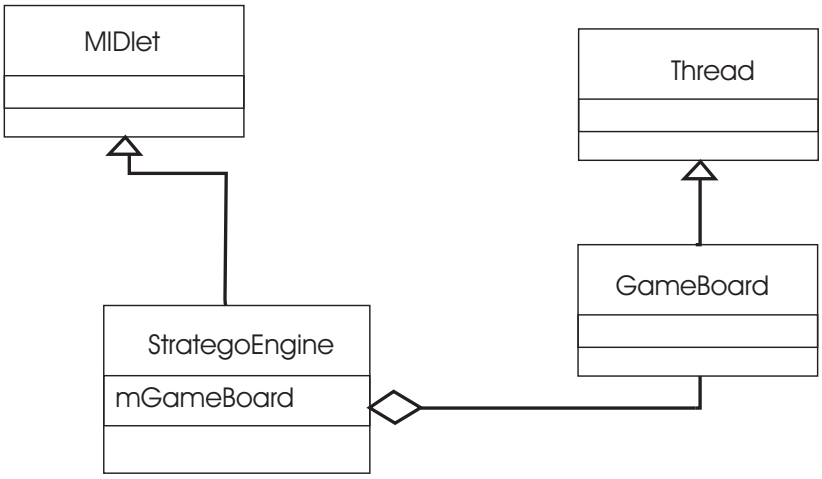

Figura 3.2: Diagrama de classes simplificado da camada lógica

No momento em que o usuário dispara a aplicação, a classe `StrategoEngine` é instanciada. Ela, então, desenha no *display* uma tela de boas vindas e solicita algumas informações ao usuário, como o modo de atuação (cliente ou servidor). Com base nos dados informados pelo usuário, a `StrategoEngine` instancia as demais classes da aplicação, inicializando-as apropriadamente e criando as *threads* necessárias. Todas as *threads* criadas continuarão executando até que `StrategoEngine` seja finalizada de forma explícita, ou seja, por um comando do jogador para terminar a aplicação ou através de um sinal de que a partida corrente terminou.

O outro componente da camada lógica é a classe `GameBoard`, que detém toda a lógica do jogo, desde o tabuleiro com a disposição das peças até os métodos que resolvem o combate entre elas. No momento em que é inicializada pela `StrategoEngine`, a `GameBoard` executa em uma *thread* em separado, que é o laço principal do jogo. Esse laço realiza, dentre outras funções, o controle das atividades ligadas principalmente ao tempo, como fechar as conexões de rede se o outro dispositivo está há muito tempo sem enviar quaisquer informações. Todas as entradas de dados efetuadas para o dispositivo, como acionamento de uma tecla ou recebimento de uma mensagem de outro aparelho, são interpretadas pelo `GameBoard`. Essa foi uma decisão de projeto para garantir a centralização do processamento das diversas entradas que o jogo possa receber, a fim de concentrar pontos de falha críticos em poucos métodos de uma classe específica.

Um dos principais atributos da `GameBoard` é o campo de batalha, armazenado através de uma matriz 10x10, na qual cada elemento representa uma posição do tabuleiro. Para garantir um uso otimizado de memória, cada peça é representada por um byte, o que

limita o tamanho do tabuleiro em no máximo 100 bytes. A `GameBoard` utiliza os 7 bits menos significativos do byte de uma peça para representar a sua graduação. O bit mais significativo é usado para indicar o time da peça, que pode ser vermelho ou azul.

3.2.2 Camada de comunicação

A camada de comunicação desempenha uma tarefa crítica no jogo, pois uma comunicação rápida é imprescindível para que uma sessão de jogo seja satisfatória para o jogador. Visando flexibilidade, a camada de comunicação foi projetada para garantir que um jogador, de posse da aplicação, pudesse realizar todas as ações relacionadas ao jogo sem ter que realizar qualquer tipo de *download* de componentes extras ou manter em seu dispositivo várias versões do jogo. Stratego está fundamentado na idéia de cliente e servidor e, para que o jogador não necessite de *downloads* extras ou de mais de uma versão do jogo, a camada de comunicação teve de conter classes que se comuniquem de forma híbrida: hora como cliente, hora como servidor. Isso garante que o jogador não necessite manter duas versões do jogo, como uma versão "Stratego cliente"e outra versão "Stratego servidor", por exemplo. Para a camada de comunicação criou-se, então, a classe `GameConnection`, que pode atuar como cliente ou como servidor, e a classe `Sender`, que realiza tarefas de suporte em relação ao envio de dados. Estas classes, e as suas respectivas classes-base, são apresentadas na figura 3.3.

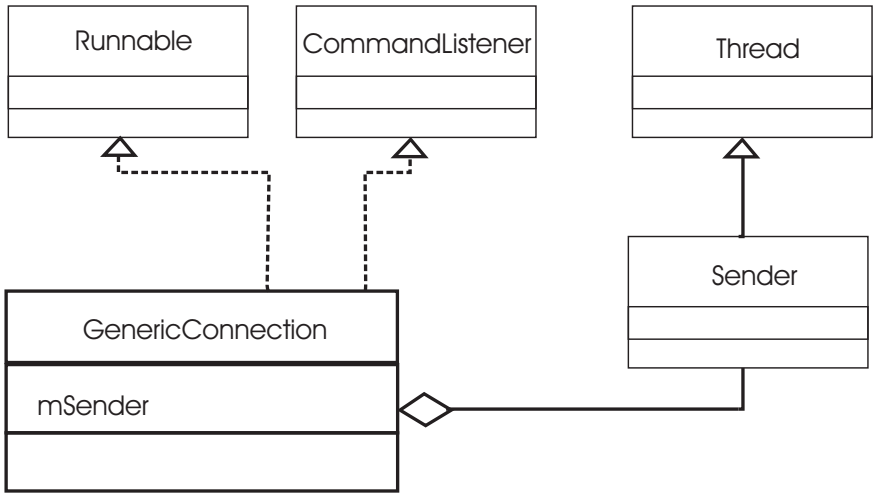

Figura 3.3: Diagrama de classes simplificado da camada de comunicação

A classe `GameConnection` foi projetada para possuir características de servidor e de cliente, porém sem que as demais classes da aplicação precisem utilizar uma interface diferente para realizar a comunicação. Independentemente do comportamento atual

(cliente ou servidor), a classe `GameConnection` disponibiliza a mesma interface de comunicação para as demais classes do jogo.

Depois que a aplicação é lançada, a `StrategoEngine` instancia um objeto da classe `GameConnection`, inicializando-o com o modo de atuação (cliente ou servidor) escolhido pelo jogador. Se `GameConnection` está atuando como servidor, ela escuta pedidos de conexão vindos de outro dispositivo. Assim que recebe um pedido de conexão, um canal para comunicação é aberto entre os dois dispositivos. Se `GameConnection` está atuando como cliente, ela abre um canal de comunicação na porta que o servidor está escutando e esperando por novas conexões. Assim que o servidor sinaliza que o pedido foi aceito, um canal de comunicação é estabelecido, da mesma forma que no modo de atuação como servidor.

Para simplificar o trabalho, foram utilizados endereços de conexão e portas fixas nas classes de comunicação, todos criados em tempo de compilação. Julgou-se safistisfatória a análise dos resultados de comunicação obtidos com a implementação dos endereços e portas dessa forma, o que torna o aprimoramento desse processo (como utilização de endereços dinâmicos, por exemplo) um tópico fora do escopo do presente trabalho. Ressalta-se que a implementação de Stratego é um caso de uso para a análise da plataforma, logo fazer com que o jogo utilize recursos de operadoras de celular, por exemplo, para encontrar outros celulares, abrindo um canal de comunicação, enfatiza a análise tão somente de recursos de rede e ,possivelmente, peculiaridades de operadoras, não os demais assuntos da plataforma.

A alta latência de comunicação entre os dois dispositivos faz com que o jogador tenha que esperar parcelas de tempo consideráveis entre o envio e resposta de uma mensagem. Para garantir que a comunicação de dados seja tratada de forma elegante e possa ser notada pelo jogador (NOKIA CORPORATION, 2003b), uma tela de espera é usada. Essa tela informa ao jogador que os dados estão sendo enviados e impedem que o jogo fique sem resposta durante a transmissão de dados, o que faz com que o jogador não saiba se o sistema está enviando os dados ou se repentinamente parou de funcionar. Para conseguir enviar os dados e ainda manter a aplicação respondendo à interação do jogador, uma *thread* é utilizada. Essa *thread* é gerenciada por uma instância da classe `Sender`, criada quando `GameConnection` se inicializa. A *thread* da classe `Sender` tem como único objetivo enviar uma quantidade de bytes para o destino informado por

`GameConnection`.

Para receber uma mensagem, a classe `GameConnection` mantém-se ouvindo o canal de comunicação estabelecido com o outro dispositivo. Assim que um byte é recebido, ele é colocado em um *buffer* e o processo de recebimento de uma mensagem é iniciado. `GameConnection` recebe cada byte enviado pelo outro dispositivo e o guarda em um *buffer*, até que receba um caracter de quebra de linha, indicando que a mensagem terminou. Nesse momento, o *buffer* contém toda a sequência de bytes recebida, ou seja, uma mensagem completa. De posse dessa mensagem e partindo da premissa que o processamento das entradas é feito pela classe `GameBoard`, `GameConnection` invoca o método `onMessage()` de `GameBoard`, para que a mensagem seja tratada adequadamente.

3.2.2.1 Protocolo de comunicação

Os dois dispositivos trocam vários tipos de mensagens durante uma partida, como sinais que indicam que o tabuleiro foi configurado, que uma peça se moveu, que a partida terminou, etc. Para padronizar essa forma de comunicação, criou-se um protocolo com mensagens simples e pequenas que podem ser rapidamente traduzidas e transmitidas. A estrutura básica de uma mensagem é descrita na Figura 3.4

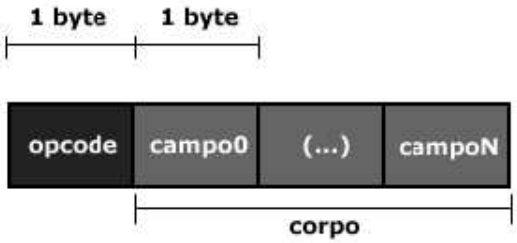

Figura 3.4: Estrutura básica de uma mensagem do protocolo utilizado pelo jogo

A mensagem possui N campos de um byte cada, sendo o primeiro campo um código de operação (`opcode`), que descreve o tipo da mensagem. O número de campos varia para cada tipo de `opcode`. A Tabela 3.1 descreve todos os tipos de mensagens possíveis.

Para fins de comunicação, o tabuleiro do jogo é indexado de forma sequencial, sendo a posição (0,0) da matriz o número 0, a posição (0, 1) o número 1, a posição (0, 2) a posição 2 e assim por diante. Dessa forma, tanto a mensagem de `opcode B` quanto a de `opcode M` utilizam essa indexação para informarem a posição de alguma peça no tabuleiro.

Tabela 3.1: Estrutura de todos os tipos de mensagens

Opcode	Total de campos (sem contar o opcode)	Descrição
B	80	Configuração, sincronização dos tabuleiros. O corpo é interpretado como sendo um conjunto de 40 pares peça-posição, que descrevem a distribuição completa das peças do adversário em seu tabuleiro.
M	2	Movimento de peça. O corpo possui duas coordenadas: a origem do movimento e onde a peça parou.

A mensagem de `opcode B` é enviada assim que o jogador termina de dispor as peças no seu tabuleiro. O dispositivo que envia uma mensagem de `opcode B` está apenas esperando a confirmação do outro dispositivo para iniciar a batalha de uma partida de jogo, já que o seu tabuleiro está pronto. O dispositivo que recebe uma mensagem desse tipo primeiramente salva o seu conteúdo e, em seguida, verifica a sua condição de configuração do tabuleiro. Os casos possíveis de configuração são os seguintes:

- O seu tabuleiro está pronto, o que indica que ambos os tabuleiros estão configurados. O dispositivo, então, sincroniza o seu tabuleiro local com as informações recebidas e, então, envia uma mensagem de `opcode B` para o outro dispositivo, fazendo com que ele também sincronize seu tabuleiro local com as informações enviadas, a fim de que o jogo possa começar. Por convenção, o dispositivo que está atuando como servidor inicia jogando.

- O seu tabuleiro não está pronto. Nesse momento, o dispositivo possui as informações sobre como o tabuleiro inimigo está configurado (isso foi informado na mensagem recém-chegada), mas o tabuleiro local ainda não está pronto para o jogo. Assim que o jogador termina de configurar o tabuleiro local, as informações que foram recebidas antes são inseridas nele (o tabuleiro local é sincronizado) e, então, uma mensagem de `opcode B` é enviada ao outro dispositivo. O outro dispositivo, como já está pronto, apenas sincronizará o seu tabuleiro local e iniciará o combate. Se ele estiver atuando como cliente, irá esperar o servidor jogar, uma vez que é esse quem joga primeiro; se for servidor, iniciará jogando.

A mensagem de `opcode M` é enviada sempre que o jogador efetua um movimento no tabuleiro. Depois de jogar, são armazenados os dois números que indicam, respectiva-

mente, a origem e o fim do movimento. Em seguida esses valores são concatenados com o `opcode M` e, então, empacotados em uma mensagem e enviados. Como ambos os aparelhos estão com o tabuleiro sincronizado no momento que mensagens de `opcode M` são enviadas, o dispositivo que recebe uma mensagem desse `opcode` realiza o movimento que ela indica, repetindo os mesmos cálculos que o dispositivo que enviou a mensagem realizou, o que garante que ambos os jogadores vejam os mesmos acontecimentos (avisos na tela, sons, etc.) após o movimento de uma peça.

3.2.3 Camada gráfica

Para garantir que a camada gráfica do jogo apresentasse o mínimo de impacto possível em outras classes da aplicação se alguma mudança fosse aplicada às suas classes, todos os seus elementos foram projetados para possuirem pouca ou nenhuma lógica de jogo. Partindo da regra que somente as classes da camada gráfica estão autorizadas a desenhar no *display* e, também, garantindo que essas classes não contenham lógica da aplicação, as classes dessa camada podem sofrer alterações drásticas de código. Por exemplo, se algum dispositivo móvel necessitar de comportamentos e métodos especiais para que o desenho seja feito de maneira correta, grandes alterações poderão ser aplicadas às classes dessa camada, até mesmo a sua completa substituição por classes com os mesmos métodos públicos, sem que isso implique alterações nas outras classes ou lógica da aplicação. Isso é possível porque nenhuma lógica do jogo está contida nas classes da camada lógica, apenas lógica relacionada ao desenho.

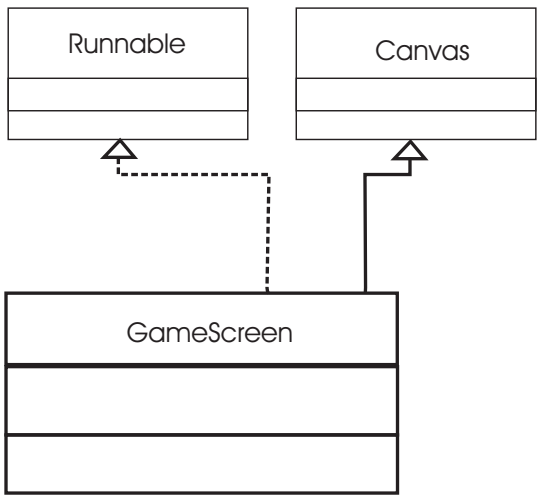

Figura 3.5: Diagrama de classes simplificado da camada gráfica

A classe que compõe a camada gráfica é a `GameScreen`, conforme ilustrado pela

figura 3.5. Assim que é inicializada por `StrategoEngine`, a classe `GameScreen` obtém informações do ambiente em que se encontra, como a largura e altura máxima do *display*. Em seguida, realiza-se alguns cálculos para que o tamanho de cada quadrado do tabuleiro do jogo possa ser desenhado da maior maneira possível, a fim de que o usuário tenha uma visão clara do campo de batalha. Em seguida, `GameScreen` cria uma *thread*, que ficará ativa enquanto o jogo estiver executando. Essa *thread* tem como função realizar os desenhos na tela e, também, ouvir as entradas feitas pelo teclado. Ela é responsável por ler as entradas do teclado porque, na plataforma J2ME, a classe `Canvas` (extentida por `GameScreen`) é a responsável por manipular eventos do teclado e de ponteiros, se o dispositivo suportar (SUN MICROSYSTEMS, INC., 2006a), decodificando-os. Todas as entradas que são realizadas pelo jogador são ouvidas pela `GameScreen`, que então decodifica a tecla pressionada e a traduz para um código de tecla de jogo, como `GAME_A`, `GAME_B`, `UP` e `DOWN`, por exemplo (estas constantes que designam as teclas são pré-definidas na plataforma J2ME). Cada dispositivo pode possuir o seu conjunto de teclas que representem movimentos e ações em jogos. Dessa forma, se em um dispositivo a tecla para `FIRE` for a tecla de número 5 e, em outro dispositivo, ela for a tecla de número 9, a aplicação receberá apenas o código `FIRE`, não `KEY_NUM5` ou `KEY_NUM9`, o que garante portabilidade entre os dispositivos.

Depois que `GameScreen` decodifica qual tecla foi pressionada e qual é o seu comando de jogo, ela repassa essa informação para a classe `GameBoard`, a fim de que essa trate a entrada e responda adequadamente.

A figura 3.6 ilustra o uso de todas as classes em uma partida de Stratego entre dois dispositivos.

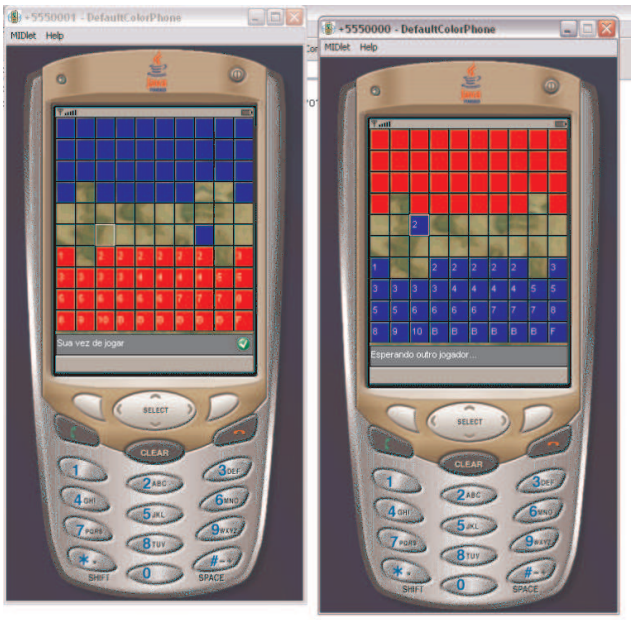

Figura 3.6: Sessão de jogo entre dois dispositivos

4 AVALIAÇÃO DA PLATAFORMA J2ME

O desenvolvimento do jogo Stratego constituiu um caso prático de utilização da plataforma J2ME para telefones celulares. Com base nesta experiência, este capítulo apresenta uma avaliação de algumas decisões tomadas durante o processo de desenvolvimento e, também, uma discussão sobre os pontos fortes e fracos da plataforma J2ME para a área de jogos em telefones celulares.

4.1 Uso de recursos

Conforme citado anteriormente, os dispositivos móveis apresentam limitações de *hardware* consideráveis, como pouca memória disponível, o que faz com que o programador use os recursos da maneira mais otimizada possível. Se o jogo utilizar uma grande quantidade de memória para seu funcionamento, maior será a memória miníma exigida, o que pode reduzir o número de disposivos aptos a rodarem o jogo.

Existem duas técnicas que o programador pode utilizar para reduzir a quantidade de recursos em seus jogos: a otimização de imagens e o obscurecimento de código (SUN MICROSYSTEMS, INC., 2006b). Além dessa preocupação, durante o desenvolvimento de Stratego, várias decisões de projeto foram tomadas, todas norteadas pela utilização de forma otimizada dos recursos disponíveis no aparelho móvel. As duas principais questões de projeto tomadas foram a utilização do menor número de classes possível, visando a redução no uso de memória para a execução do jogo, e troca de poucas informações entre os dispositivos.

As seções seguintes explanam sobre cada uma dessas decisões de projeto e, também, cada uma das técnicas de otimização, mostrando os dados obtidos com as medições realizadas.

4.1.1 Otimização de imagens

A otimização de imagens consiste no processo de remover do arquivo de imagem informações que sejam dispensáveis à aplicação, ou seja, elementos que se forem removidos não causarão alterações drásticas na visualização da imagem. Imagens PNG, por exemplo, possuem blocos de informações internos, chamados `chunks`, que podem conter diversas informações sobre a imagem. Algumas dessas informações podem não ser relevantes para a renderização da imagem, uma vez que foram inseridas por programas de edição para controle de camadas e paletas de cores, por exemplo. Os otimizadores de imagens procuram por `chunks` irrelevantes e os removem. Além disso, os otimizadores de imagens também tendem a paletizar a imagem, o que faz com que o número de cores seja mapeado em uma tabela com tamanho pré-definido, o que reduz a quantidade de cores na imagem e, consequentemente, o seu tamanho.

O tipo de imagem escolhido para ser utilizado em Stratego foi o PNG, pelo fato de que esse formato é obrigatoriamente implementado pelos dispositivos que suportam aplicações da plataforma J2ME (SUN MICROSYSTEMS, 2006). Embora imagens no formato JPG ou GIF, por exemplo, sejam menores, não há garantia de que todos os dispositivos móveis tenham suporte a esses formatos.

Para otimizar as imagens de Stratego, foram utilizados dois *softwares*: o PNGout (ARDFRY IMAGING, LLC, 2007) e o OptiPNG (TRUTA, 2007). A Tabela 4.1 ilustra o tamanho do arquivo resultante do empacotamento do jogo e, também, a utilização de memória durante a execução, antes e depois da otimização das imagens.

Tabela 4.1: Informações do jogo sobre otimização das imagens (resultados em bytes)

Otimização imagens	Tamanho do JAR	Memória max. cliente	Memória max. servidor
Não	372.308	700.104	699.476
Sim	92.434	191.972	192.860

Conforme pode ser observado na Tabela 4.1, o impacto da otimização das imagens é considerável. O tamanho do arquivo resultante do empacotamento de todos os elementos que compõem a aplicação (classes e imagens) sofreu um decréscimo de aproximadamente 75%. Quanto menor o arquivo JAR resultante, menos dados o usuário terá de transferir para o seu dispositivo via *download* para conseguir obter o jogo.

Em relação à utilização de mémória durante a execução do jogo, as colunas "Memória

max. cliente"e "Memória max. servidor"indicam o pico máximo de utilização de memória da aplicação agindo como cliente e como servidor, respectivamente. Antes da otimização das imagens, o pico máximo registrado de consumo de memória foi de aproximadamente 683Kb. Após a otimização das imagens, tanto por parte do jogo atuando como cliente ou como servidor, a redução foi de aproximadamente 72%, o que resultou num pico máximo em torno de 184Kb de memória usada. Esses dados de utilização de memória durante a execução foram obtidos com o *Memory Monitor*, ferramenta integrada ao *J2ME Wireless Toolkit*, kit utilizado para o desenvolvimento de Stratego.

Embora Stratego faça uso de poucas imagens, a redução do tamanho do arquivo JAR resultante e também a redução da utilização de memória em virtude da otimização de imagens ficou em torno de 73%. Aplicações com uma grande quantidade de imagens PNG podem vir a reduzir drasticamente o seu consumo de memória através do uso de ferramentas de otimização.

4.1.2 Obscurecimento de código

O obscurecimento de código consiste em executar sobre o código-fonte da aplicação alguma ferramenta cuja função é obscurecer o conteúdo analisado, tornando-o pouco compreensível ou ilegível para humanos. Embora essa técnica seja utilizada para proteção dos arquivos compilados, afim de evitar ou dificultar a descompilação, ela também pode ser usada para reduzir a quantidade de recursos utilizados pela aplicação. Uma vez que o obscurecedor substitui o nome original das classes por códigos, como a troca do nome `GameBoard` por `a`, o tamanho dos arquivos tente a ser reduzido. Isso gera, por conseguinte, byte-codes menores, o que fará com que a aplicação utilize menos memória do dispositivo móvel.

Utilizando-se o ProGuard 3.7 (LAFORTUNE, 2007), o código de Stratego (já com imagens otimizadas) foi obscurecido. A Tabela 4.2 ilustra o tamanho do arquivo resultante do empacotamento do jogo e, também, a utilização de memória durante a execução, antes e depois do obscurecimento do código-fonte.

Depois que o código-fonte da aplicação foi obscurecido, o arquivo JAR gerado teve o seu tamanho reduzido em 2,78%. Em relação à utilização de memória durante a execução do jogo, o pico máximo de consumo medido antes do obscurecimento sofreu uma redução de aproximadamente 4,60% com a utilização da técnica. Comparando-se os dados da

Tabela 4.2: Informações do jogo sobre obscurecimento do código-fonte (resultados em bytes)

obscurecimento código-fonte	Tamanho do JAR	Memória max. cliente	Memória max. servidor
Não	92.434	191.972	192.860
Sim	90.230	183.688	183.372

aplicação em seu pior caso (imagens não otimizadas e código-fonte não obscurecido) com o melhor caso (imagens otimizadas e código-fonte obscurecido), a redução do arquivo JAR resultante chegou a 75,76%; a redução do uso de memória durante a execução do jogo chegou a 73,77%.

4.1.3 Eliminação de métodos get/set

Para buscar um menor consumo de memória para a execução do jogo, decidiu-se não utilizar métodos `get/set` nas classes de Stratego, pois a utilização de tais métodos tende a aumentar o tamanho do *heap* de execução e o *overhead* de processamento das classes (FASTERJ.COM, 2007). Para atenuar esses efeitos ou os eliminar completamente, todas as classes de Stratego foram implementadas com propriedades públicas, sem utilização de métodos `set` para sua alteração.

Para analisar os resultados dessa decisão de projeto, foram realizadas medições no tamanho do arquivo JAR resultante e na quantidade de memória utilizada para a execução do jogo, tanto na forma de atuação cliente como na forma de atuação servidor. A Tabela 4.3 mostra dados referentes ao jogo com classes utilizando apenas propriedades públicas, sem uso de métodos `get/set`. Em contrapartida, a Tabela 4.4 mostra os mesmos atributos medidos na Tabela 4.3, porém com classes utilizando apenas propriedades privadas (propriedades estáticas foram deixadas públicas), com métodos `get/set` para sua manipulação.

Tabela 4.3: Informações do jogo não utilizando métodos get/set (resultados em bytes)

obscurecimento código-fonte	Otimização imagens	Tamanho do JAR	Memória max. cliente	Memória max. servidor
Não	Não	372.308	700.104	699.476
Nao	Sim	92.434	191.972	192.860
Sim	Não	370.104	694.548	692968
Sim	Sim	90.230	183.688	183.372

Para uma melhor compreensão dos dados obtidos, a comparação entre as Tabelas 4.3

Tabela 4.4: Informações do jogo utilizando métodos get/set (resultados em bytes)

obscurecimento código-fonte	Otimização imagens	Tamanho do JAR	Memória max. cliente	Memória max. servidor
Não	Não	374.410	691.392	691.804
Não	Sim	94.536	181904	183.484
Sim	Não	370.711	700.240	700.688
Sim	Sim	90.837	191.232	191.508

e 4.4 está explicitada na Tabela 4.5. Essa tabela foi gerada subtraíndo-se os campos da Tabela 4.4 com os campos da Tabela 4.3. Dessa forma, resultados positivos encontrados na Tabela 4.5 indicam que o valor do campo em questão aumentou da primeira para a segunda tabela, ao passo que resultados negativos indicam uma redução no valor.

Tabela 4.5: Subtração dos dados da Tabela 4.4 pelos dados da Tabela 4.3 (resultados em bytes)

obscurecimento código-fonte	Otimização imagens	Tamanho do JAR	Memória max. cliente	Memória max. servidor
Não	Não	2.102	-8.712	-7.672
Não	Sim	2.102	-10.068	-9.376
Sim	Não	607	5.692	7.720
Sim	Sim	607	7.544	8.136

Todas as porcentagens indicadas a seguir são em relação ao tamanho da aplicação sem utilização de métodos `get/set`. Conforme mostrado na Tabela 4.5 e partindo da análise dos casos em que as imagens da aplicação foram otimizadas, nota-se um aumento no tamanho do arquivo JAR resultante. A segunda linha da Tabela 4.5 mostra um aumento de aproximadamente 2,27% no tamanho do arquivo JAR resultante, da versão sem métodos `get/set` em relação à sua equivalente, com tais métodos. A quarta linha da Tabela 4.5 também mostra um aumento do JAR resultante em aproximadamente 0,67%.

Em relação à utilização de memória durante a execução da aplicação, a porcentagem de variação é maior. A análise da primeira e segunda linha da Tabela 4.5 mostra que, para as versões do código que não foram obscurecidas, a inclusão de métodos `get/set` reduziu o consumo de memória durante a execução. A redução mais significativa ocorreu no caso do código-fonte não obscurecido e imagens otimizadas, chegando a uma queda no uso de memória de aproximadamente 5%.

Com relação ao obscurecimento de código, que reduziu o consumo de memória da aplicação em sua versão sem os métodos `get/set`, conforme mostrado na subseção

4.1.2, o mesmo resultado não foi obtido com a adição dos métodos `get/set`. A terceira linha da Tabela 4.5 mostra um aumento no consumo de memória de aproximadamente 0,96% no código de Stratego alterado (obscurecido, sem imagens otimizadas, com `get/set`) em relação ao código de Stratego original (obscurecido,sem imagens otimizadas, sem `get/set`). A quarta linha da Tabela 4.5, que compara a versão com código-fonte obscurecido e imagens otimizadas, mostra um aumento mais significativo do consumo de memória. De acordo com os dados obtidos, a versão alterada da aplicação (obscurecido, imagens otimizadas, com `get/set`) apresentou um aumento no consumo de memória de aproximadamente 4,30% em relação ao seu equivalente original (obscurecido, imagens otimizadas, sem `get/set`).

4.2 Vantagens e desvantagens

Estão expostas a seguir as principais vantagens e desvantagens, *até onde foi possível constatar*, das funcionalidades da plataforma J2ME para o desenvolvimento de jogos ao estilo Stratego. Para avaliar se uma característica é uma vantagem ou desvatagem, consideraram-se elementos como:

- Facilidade de customização de elementos: programador poder decidir exatamente a localização de um elemento na tela, por exemplo;
- Facilidade para execução de tarefas ligadas à aplicação: desenhar imagens na tela, obter dados através de elementos de interface, etc;
- Recursos da linguagem: ferramentas e artifícios que a linguagem de programação apresenta ao programador para facilitar no desenvolvimento do jogo, como estruturas de dados (matriz, vetor, *buffers*);
- Qualidades da API: uniformidade e clareza nos métodos/funções da API, afim de tornar a lógica da aplicação simples e sem obscuridades;

4.2.1 Vantagens

Umas das principais vantagens do J2ME é a existência de um pacote de classes voltado exclusivamente para o desenvolvimento de material para jogos *wireless*: o `javax.microedition.lcdui.game`. As classes desse pacote são estruturadas e implementadas para permitirem o uso de código nativo, aceleração de *hardware* e formatos de

dados específicos do dispositivo, o que aumenta o desempenho da aplicação e reduz o seu tamanho (SUN MICROSYSTEMS, INC., 2006a). Através do `javax.microedition.lcdui.game`, o programador possui à disposição classes com funcionalidades destinadas à tarefas comuns no desenvolvimento de jogos, como consultar o estado de uma tecla em um determinado instante, possibilidade de criar camadas para a apresentação de contéudo, criação e gerência de animações (baseadas na renderização de uma sequência de imagens) e gerenciamento e criação de grandes áreas de conteúdo gráfico (como mapas), gerenciados através de *tiles*.

Outra vantagem da plataforma é a existência de *threads*. Embora os dispositivos móveis apresentem poucos recursos disponíveis, como baixa capacidade de processamento e pouca memória, a JVM disponibiliza ao programador o uso de *threads*. Esse recurso é útil para realizar tarefas concorrentes na aplicação, como o envio dos dados que é feito em Stratego, no qual uma *thread* envia as informaçõs ao mesmo tempo que a aplicação recebe e interpreta aos comandos do jogador vindos do teclado. Além de utilizar *threads* para o envio de dados, Stretago também faz uso desse recurso para desenhar gráficos: quando a tela de jogo é criada, uma *thread* é também criada e associada à tarefa, realizando as atualizações a cada ciclo do laço principal do jogo.

Uma outra vantagem encontrada é o conjunto de funcionalidades para comunicação. Conforme explanado na seção 2.3.3.3, o J2ME apresenta o *Generic Connection Framework (GCF)*. Através dele, o programador tem acesso a recursos de comunicação em rede ou acesso a arquivos de uma forma clara e uniforme. Estão disponíveis para uso no J2ME conexões como HTTP, HTTPS, SMS, Datagramas UDP, *sockets* e Bluetooth. Através de *sockets*, por exemplo, que são utilizados em Stratego, é possível que a aplicação atue como cliente de uma conexão ou como servidora, comunicando dados através de um protocolo próprio. Isso aumenta as possibilidades de desenvolvimento por parte do programador, que não fica atrelado a modelos de comunicação ou protocolos pré-definidos pela plataforma, podendo escolher aquele que melhor se adapte às suas necessidades, conforme citado na seção 2.3.3.3.

Finalmente outra vantagem encontrada na plataforma J2ME são os vários tipos e estruturas de dados disponíveis ao programador. Existem classes que provêem uma abstração à manipulação de dados brutos, como a classe `StringBuffer`, que implementa uma sequência variável de caracteres, permitindo a concatenação de caracteres em difer-

entes momentos, sem que o programador precise se preocupar com o tamanho do buffer disponível. Essa classe gerencia, também, possíveis problemas de concorrência derivados, por exemplo, da ação de multiplas *threads* sobre um único buffer. Além dessa classe, estão disponíveis outras classes que implementam tipos de dados como `Byte`, `Double`, `Short` e `Integer`. Estão disponíveis também, vinculados a essas classes, métodos úteis ao tratamento e manipulação de dados, como o método `getBytes()`, que retorna uma sequência de bytes que descreve uma *string*. Esse método foi utilizado em Stratego para a comunicação de dados entre os dispositivos, para que a *thread* de envio pudesse transmitir um byte da mensagem por vez, até que toda a *string* fosse completamente transmitida.

4.2.2 Desvantagens

A primeira desvantagem encontrada na plataforma J2ME durante o desenvolvimento de Stratego foi a forma como gráficos e eventos de teclado estão, de certa forma, atrelados. Para que uma classe seja hábil a ser desenhada na tela do dispositivo, ela precisa estender a classe `Displayable`. Para a implementação de Stratego, utilizou-se a classe `Canvas`, que estende `Displayable`. Conforme (SUN MICROSYSTEMS, INC., 2006a), `Canvas` é a classe base para o desenvolvimento de aplicações que manipulam eventos de baixo nível e utilizam chamadas para desenhar gráficos na tela. É através dos métodos da classe `Canvas` que o programador pode manipular ações de jogos e eventos de teclas. A combinação de eventos relacionados a gráficos e manipulação de teclado associadas a uma mesma hierarquia de classes torna a lógica da aplicação confusa do ponto de vista técnico. Em Stratego, por exemplo, todas as ações relacionadas a renderização de elementos na tela foram limitados à classe `GameScreen` (estende `Canvas`) que, por sua vez, também recebe e repassa os eventos de teclado realizados pelo jogador. Conforme descrito ao longo desse trabalho, `GameScreen` é a única classe autorizada a desenhar na tela, tendo, para isso, o mínimo de lógica possível, para que alterações na classe não causem efeitos sobre a lógica de outras classes. Isso, porém, não é plenamente alcançado, uma vez que `GameScreen` deve manter uma parcela de lógica da aplicação para poder manipular eventos de teclado e, então, os repassar aos cuidados de `GameBoard`. No contexto de Stretego, faz mais sentido que `GameBoard` ou `StrategoEngine` recebam eventos de teclado, porém `GameScreen` é quem realiza essa tarefa.

Outra desvantagem encontrada na plataforma foi a pouca abstração para a criação de animações. Para que o programador crie animações, é possível utilizar a classe `Sprite`, integrante do pacote de classes destinadas ao desenvolvimento de jogos. Essa classe renderiza os vários quadros encontrados em uma única imagem, de forma a criar uma animação. Os quadros na imagem são definidos pela sua divisão em células de tamanho igual, na qual cada célula resultante será interpretada como um quadro da animação. A largura e a altura de cada célula é definida pelo programador. Uma vez instanciado, o objeto `Sprite` pode ser renderizado. Para que o próximo quadro seja renderizado, é necessário que o programador chame explicitamente um método que indica que o quadro atual da animação deve ser incrementado. Para que alguns quadros da animação sejam renderizados por mais tempo, como no caso do programador desejar uma parada na animação, é possível dizer ao objeto `Sprite` qual a ordem dos quadros a ser renderizada, através de um vetor. Nesse vetor, o programador pode fazer com o que o quadro da parada seja renderizado três vezes, por exemplo, para só então o próximo quadro ser renderizado. Um exemplo para ilustrar esse processo seria a utilização do vetor `0,1,1,1,2,3` para guiar a animação; nesse exemplo, a animação iniciaria no quadro `0` da imagem, passando depois para o quadro `1` (três vezes), em seguida para o quadro `2` e, finalmente, para o quadro `3`. Isso dará ao usuário a sensação de que a animação ficou pausada por alguns instantes no quadro `1`. Até onde foi possível constatar, a plataforma J2ME, em si, não apresenta ferramentas que auxiliem o programador na criação de animações de uma forma sistemática e automatizada. Isso pode vir a aumentar o tempo de produção de um jogo em virtude da elaboração de animações com pouco nível de abstração do processo, conforme descrito. A seção 4.3 mostra outras plataformas de desenvolvimento com recursos mais aprimorados em relação à criação e gerência de animações.

Outra desvantagem encontrada no desenvolvimento de jogos ao estilo Stratego é a forma como elementos de GUI (*Graphic User Interface*), ou elementos gráficos, são tratados. Conforme citado anteriormente, para que um elemento seja mostrado na tela do dispositivo, ele precisa estender `Displayable`. Qualquer elemento gráfico que extenda essa classe pode, então, ser desenhado na tela. A plataforma J2ME apresenta vários elementos de GUI para auxiliar no desenvolvimento de aplicações, como a classe `Alert`, que mostra um aviso na tela, e a classe `ChoiceGroup`, que mostra um grupo de opções de escolha. Se o programador precisar mostrar vários desses elementos na tela, eles de-

vem, então, ser agregados a um objeto da classe `Form` (que estende `Displayable`), para que sejam desenhados todos juntos. A forma de desenho na tela é ditada pelo algoritmo de layout da classe `Form` (SUN MICROSYSTEMS, INC., 2006a). Essa estruturação dos elementos, durante o desenvolvimento de Stratego, trouxe algumas complicações ao código relacionado à interface com o usuário. Quando o jogo se inicia, a classe `GameScreen` é setada como sendo o elemento que deve ser desenhado na tela. Nesse momento, a aplicação tem total controle sobre aquilo que é apresentado ao usuário, como a imagem de fundo, a imagem das peças do jogo, etc. Em determinados momentos, é necessário mostrar ao jogador elementos de interface, como grupos de escolha ou campos de texto. Como `GameScreen` é a classe corrente que está sendo desenhada, não é possível que outros elementos (como o `ChoiceGroup`) sejam mostrados sem que `GameScreen` interrompa suas tarefas. Para que o `ChoiceGroup` seja desenhado, uma instância da classe `Form` deve ser setada como o elemento que deve ser desenhado na tela (`focus` de desenho), o que faz com que `GameScreen` interrompa seus desenhos. Para desenhar elementos de interface sobre a tela de jogo, o programador pode, por exemplo, 1) abrir mão da tela de jogo e, então, instanciar um novo `Form`, agregar elementos de GUI a ele e instruir o dispositivo a desenhar o form (fazendo com que o jogador não veja mais qualquer elemento do jogo, como o mapa de combate, no caso de Stratego); ou 2) implementar seus próprios componentes de GUI, a fim de que esses possam ser desenhados pela classe que está desenhando no momento (`GameScreen`, no caso de Stratego). Há, porém, alguns elementos de interface que podem ser mostrados no momento que `GameScreen` está desenhando, como um objeto da classe `Alert`. Quando instanciado, o programador indica ao construtor dessa classe qual elemento deve passar a ser desenhado depois que o aviso desaparecer. Mesmo com essa abordagem, `GameScreen` cede a tarefa de renderizar todos os elementos da tela para que o objeto da classe `Alert` possa ser desenhado. Nesse momento, não haverá mais elementos do jogo na tela, apenas o aviso (se o programador desejar que uma parcela do jogo continue sendo mostrada, ele deverá implementar seu próprio aviso, como no caso do `ChoiceGroup`).

4.3 Paralelo com outras plataformas de desenvolvimento

Afim de aumentar a análise da plataforma J2ME e abordar de forma mais técnica os tópicos levantados na seção 4.2, a seguir é apresentado um paralelo entre as funcionali-

dades da plataforma utilizada para o desenvolvimento de Stratego com as funcionalidades semelhantes encontradas em outras plataformas. Não é o objetivo dessa seção prover uma análise profunda das plataformas citadas, mas sim enriquecer a análise das vantagens e desvantagens do J2ME, citadas anteriormente.

4.3.1 Paralelo entre as vantagens

Uma das vantagens citadas da plataforma J2ME é a existência de um pacote destinado exclusivamente ao desenvolvimento de jogos. Comparando-se com a plataforma Flash Lite, em sua versão 2.0, até onde foi possível constatar, não existe um pacote de classes explicitamente declarado para o desenvolvimento de jogos. Embora a plataforma Flash Lite apresente vários recursos gráficos aprimorados, que serão citados à frente, não há a centralização desses em um pacote explícito como aquele visto no J2ME. Conforme já descrito, o pacote `javax.microedition.lcdui.game` utiliza classes que são implementadas e desenhadas para tirarem vantagem do *hardware* do dispositivo, aumentando o desempenho dos processos relacionados.

Outra vantagem citada anteriormente foi a existência de *threads* em J2ME, que aumentam a capacidade da aplicação em executar tarefas de forma concorrente. Até onde foi possível constatar, na plataforma Flash Lite não está disponível ao programador a criação e utilização de *threads*. É possível, porém, simular a execução concorrente de operações através do uso da função `setInterval()`, no qual o programador define um método de uma classe para ser executado repetidamente, com um intervalo customizável entre as chamadas. Através das propriedades da classe, é possível manter estados do processo, simulando a preservação de contexto de *threads*. Essa abordagem, porém, é limitada, uma vez que programador não detém um fluxo exclusivo de execução, como aquele proporcionado pelas *threads* do J2ME. Conforme testes realizados, métodos que sejam chamados por `setInterval()` não podem conter laços infinitos, como um `while(true)`, uma vez que eles priorizam a execução somente desse método, pois ele não retorna, fazendo com que `setInterval()` aguarde sua finalização indeterminadamente, o que faz com que a aplicação não responda mais (resultando em um *crash*). Em relação à plataforma BREW, a partir do SDK 3.0, está disponível ao programador recursos como a `IThread`, que implementam a execução de tarefas de forma paralela (QUALCOMM INCORPORATED, 2006).

Uma vantagem da plataforma J2ME também citada é o grande conjunto de recursos para comunicação em rede. Em um paralelo com a plataforma Flash Lite, constatou-se que essa apresenta meios para troca de informações em rede limitados a protocolos pré-definidos, em sua grande maioria. A plataforma possibilita ao programador o carregamento de arquivos externos (como documentos de texto ou arquivos XML) através do protocolo HTTP (se o arquivo for remoto), utilização de SMS e envio de um conjunto de informações (como campos de um formulário) para um servidor, utilizando HTTP para a comunicação (ADOBE SYSTEMS INCORPORATED, 2006b). No conjunto de meios de comunicação com protocolos mais flexíveis (com características definidas pelo próprio programador, por exemplo), está disponível a classe `XMLSocket`, que permite ao programador abrir um canal de dados com algum servidor e, através de XML, trocar informações. Há várias restrições a serem consideradas nesse processo, como a porta na qual o *socket* é aberto, que deve ser, obrigatoriamente, maior que 1024; e que cada mensagem enviada é um documento XML terminado por um byte zero (o servidor que receber a mensagem deve entender essa convenção). Em comparação com as funcionalidades do *Generic Connection Framework* do J2ME, que apresenta uma API uniforme e clara para comunicação em rede, através de vários protocolos e formas de transmissão de dados, as possibilidades de comunicação da plataforma Flash Lite são mais limitadas.

Em um paralelo com a plataforma BREW, é possível encontrar algumas semelhanças em termos de funcionalidades de comunicação. BREW disponibiliza bibliotecas como `ITAPI`, que possibilitam acesso a serviços de voz e SMS, `ISocket` que permite a manipulação de conexões tanto TCP quanto UDP, `IWeb` que facilita a condução de transações web sobre vários protocolos e `IPort` que disponibiliza uma comunicação genérica com várias interfaces.

Outra vantagem citada sobre o J2ME são os vários tipos e estruturas de dados disponíveis ao programador. Conforme já explanado, J2ME apresenta um grande conjunto de tipos nativos de dados, como `int`, `char` e `byte`, além de classes que implementam estruturas de dados úteis à programação, como `StringBuffer`. Em um paralelo com a plataforma BREW, um grande conjunto de tipos de dados também é fornecido, assim como aqueles que são encontrados na linguagem de programação C++ (QUALCOMM INCORPORATED, 2006). Em ambas as plataformas, a linguagem de programação apresenta tipagem forte, o que evita que uma variável tenha seu tipo alterado ao longo

do tempo. Na plataforma Flash Lite, até onde foi possível constatar, existem também estruturas de dados já implementadas úteis à programação, como `Array`, `String` e `Boolean`, porém a linguagem de programação é fracamente tipada e não apresenta distinção entre certos tipos nativos de dados. Em relação à tipagem, conforme testes realizados, é facultativa a definição do tipo de uma variável, o que permite que uma variável tenha o seu tipo alterado ao longo do tempo. Se o programador explicimente especificar o tipo de uma variável, forçando uma tipagem forte, o compilador acusa erro de incompatibilidade de tipos se houver a tentativa de utilização de algum tipo não compatível. Em relação à distinção de tipos nativos de dados, não existe distinção entre números inteiros (com sinal ou sem sinal) e números de ponto flutuante. Se o programador deseja tipar uma variável como sendo numérica, ele possui à disposição apenas o tipo `Number`, que engloba todos os tipos de dados numéricos existentes na plataforma.

A tabela 4.6 apresenta um resumo da comparação das vantagens da plataforma J2ME com as outras plataformas.

4.3.2 Paralelo entre as desvantagens

Em relação às desvantagens encontradas na plataforma J2ME, a primeira delas é a forma como a manipulação de eventos de teclado e a geração de gráficos são tratados de forma atrelada, conforme citado anteriormente. Em um paralelo com plataforma BREW, até onde foi possível constatar, a manipulação de eventos do teclado e geração de gráficos são elementos distintos tratados por interfaces diferentes. Para tratar eventos do teclado, a aplicação da plataforma BREW deve implementar o método `nomeClasse_HandleEvent()`, onde `nomeClasse` é o nome da classe que descreve a aplicação. Para desenhar na tela, a aplicação deve fazer uso, por exemplo, de `IDISPLAY_Update()`. Em relação à plataforma Flash Lite, conforme testes realizados e até onde foi possível constatar, não está ao alcance do programador a realização do controle da renderização na tela do dispositivo; essa tarefa é feita de forma transparente pelo *Flash Player* do dispositivo móvel. Em relação à manipulação de eventos do teclado, o programador deve criar e vincular um objeto a um *listener* de teclado, através da chamada de `Key.addListener()`. O *listener* informa seus ouvintes cada vez que o *status* do teclado é alterado, como no caso de uma tecla ter sido pressionada. Tanto quanto na plataforma BREW quanto na plataforma Flash Lite, não há um forte atrelamento entre manipulação de eventos do

teclado e geração de gráficos, assim como é feito em J2ME.

Outra desvantagem encontrada no J2ME foi a pouca abstração para a criação e gerência de animações, o que pode aumentar o tempo gasto para o desenvolvimento de um jogo. Conforme já explanado, J2ME utiliza a classe `Sprite` para renderizar os vários quadros de uma imagem composta, dando ao usuário a noção de movimento. Na plataforma BREW, o programador pode fazer uso da ferramenta *BCI Authoring*, que possibilita a conversão de imagens para o formato BCI (*BREW Compressed Image*) e, também, a criação de animações. Através dessa ferramenta, o programador pode abrir várias imagens e, então, criar uma animação (as imagens são empacotadas, criando a animação, na ordem em que foram abertas). O programador pode customizar o intervalo entre um quadro e outro da animação, ainda fazendo uso da ferramenta *BCI Authoring*. Na plataforma Flash Lite, conforme testes realizados, é possível criar animações através de códigos ou através da IDE de programação da plataforma. Utilizando a IDE, o programador pode definir quadros-chave da animação, criar interpolações de movimento entre eles e definir o tempo em que um quadro é mostrado, por exemplo. Através da interpolação, a IDE da plataforma baseia-se em dois quadros-chave informados pelo programador, criando os quadros intermediários para que o primeiro quadro-chave seja animado no segundo quadro-chave. As animações são representadas pela classe `MovieClip`, que apresenta métodos intuitivos para controle do fluxo da animação, como `play()`, `stop()`, `gotoAndPlay()`, `gotoAndStop()`, `prevFrame()` e `nextFrame()`. Depois de iniciada com `play()`, a animação é renderizada até o seu final sem interferência do programador.

Finalmente outra desvantagem encontrada na plataforma J2ME foi como elementos de GUI (*Graphic User Interface*), ou elementos gráficos, são tratados. Conforme explanado, em J2ME, para que um elemento seja desenhado na tela, ele deve estender alguma classe que extenda `Displayable` e, também, deve ganhar o *focus* de desenho. Em Stratego, não foi possível mostrar elementos de GUI ao mesmo tempo que a tela de jogo era mostrada, porque somente um elemento por vez poderia possuir o *focus* de desenho (eles não poderiam ser desenhados ao mesmo tempo). Em um paralelo com a plataforma Flash Lite, tanto elementos de GUI quanto outros elementos (como animações) são desenhados de forma diferenciada. Conforme testes realizados, o `Flash Player` do dispositivo realiza o desenho dos elementos que estão no palco (área visível

do *display*). O programador pode utilizar a IDE de programação da plataforma para inserir elementos gráficos manualmente no palco, dispondo-os conforme suas necessidades, ignorando preocupações relacionadas ao *focus* de desenho, como no caso da plataforma J2ME. O programador pode, também, inserir elementos gráficos em tempo de execução, contanto que esses elementos extendam `MovieClip`. Uma vez adicionado ao palco, um elemento pode ser posicionado, rotacionado ou ter suas proporções alteradas, afim de atender às necessidades do programador. Através dessa estrutura, o programador pode colocar elementos na tela em tempo de execução, como avisos ou caixas de texto, sem que a tela de jogo deixe de ser desenhada ou que o jogador perca contato visual com ela. Há, também, vários elementos de GUI já implementados e disponíveis para uso na IDE de programação da plataforma Flash Lite, como `ChoiceGroups` e `Alerts`, o que não obriga o programador a desenvolver seus próprios elementos de interface para que eles possam ser mostrados ao mesmo tempo que a tela de jogo.

A tabela 4.7 apresenta um resumo da comparação das desvantagens da plataforma J2ME com as outras plataformas.

Tabela 4.6: Resumo da comparação das vantagens entre as plataformas J2ME, BREW e Flash Lite

Elemento analisado	J2ME	BREW	Flash Lite
Existência de pacote com classes voltadas ao desenvolvimento. de jogos	Sim	Não	Não
Existência de *threads*	Sim	Sim, a partir do SDK 3.0	Não
Meios de comunicação em rede	Grande conjunto através do GCF: HTTP, Bluetooth, HTTPS, *socket*, UPDDatagram, SMS e comunicação serial.	Conjunto considerável: HTTP, *socket*, HTTPS, SMS, comunicação serial.	Conjunto reduzido: HTTP, SMS e XMLSocket (*socket* com diversas limitações).
Linguagem de programação fortemente tipada	Sim	Sim	Não
Tipos nativos de dados	Grande conjunto, como `int`, `byte`, `char` e `short`.	Os mesmos tipos de dados nativos da linguagem C, como `int`, `float`, `char`, `long` e `short`.	Inexistentes. A plataforma utiliza apenas classes para representar dados.
Classes e estruturas de dados	Grande conjunto de classes úteis já implementadas, como `StringBuffer` `String`, `Float`, `Vector` e `Hashtable`	*(Não analisado)*	Conjunto considerável de classes, como `String`, `Array`, `Boolean` e `Number`.

Tabela 4.7: Resumo da comparação das desvantagens entre as plataformas J2ME, BREW e Flash Lite

Elemento analisado	J2ME	BREW	Flash Lite
Manipulação de eventos do teclado	Métodos da classe `Canvas`	Através do método `nomeClasse_HandleEvent()`	Através de um *listener* de teclado
Desenhos no display	Métodos da classe `Canvas`	Método `IDISPLAY_Update()`	Feita de forma transparente pelo *Flash Player*. O programador não tem meios para interferir no processo.
Gerenciamento e criação de animações	Manipulação de baixo nível, através da classe `Sprite`. A plataforma, por si só, não possui ferramentas para criação de animações.	Disponibilizado através da ferramenta BCI Authoring, da própria plataforma.	Manipulação de alto nível, através da classe `MovieClip`. Criação de animações através da IDE do Macromedia Flash.
Desenho de elementos de interface (GUI)	*Focus* de desenho exclusivo: elementos GUI da plataforma não podem ser desenhados ao mesmo tempo que telas de jogo criadas pelo programado (se a tela de jogo não for um `Form`)	*(Não analisado)*	Não há *focus* de desenho. Elementos GUI da plataforma podem ser desenhados ao mesmo tempo que as telas de jogo.

5 CONCLUSÃO

Este trabalho apresentou uma exposição das funcionalidades das plataformas J2ME, BREW e Flash Lite, com maior ênfase e aprofundamento na primeira. Através da implementação de um caso de uso, o jogo multi-jogador Stratego, utilizando-se a plataforma J2ME, mostrou-se as vantages, desvantagens e peculiaridades da plataforma no que se refere ao desenvolvimento de jogos para dispositivos móveis. Com base nas experiências vivenciadas durante a implementação do caso de uso, traçou-se um paralelo das vantagens e desvantagens do J2ME com outras duas plataformas, a BREW e Flash Lite, a fim de enriquecer a análise e prover subsídios a outros desenvolvedores sobre a criação de jogos através da utilização da plataforma J2ME.

Uma das principais constribuições desse trabalho foi o levantamento e execução de técnicas ligadas à reduçao do consumo de recursos por parte de jogos para dispositivos móveis, afim de atenuar o problema de desenvolver aplicações para ambientes com pouca memória e baixo processamento. Realizou-se otimização de imagens, obscurecimento de código e redução do uso de um número elevado de classes e de métodos `get/set`. A execução dessas técnicas na implementação de Stratego possibilitou a coleta de informações importantes, como a quantidade de memória consumida por classes que utilizavam métodos `get/set` e o consumo de classes que não utilizavam tais métodos, através de tabelas de dados e comparações. Tais dados podem ser utilizados, por exemplo, como base para analisar a técnica de otimização utilizada, analisar a plataforma e suas funcionalidades ou analisar o impacto no consumo de recursos em aplicações semelhantes a Stratego.

Como o principal foco do trabalho estava na análise de assuntos relevantes ligados à plataforma a partir de um caso prático de uso, tópicos relacionados ao jogo Stratego foram deixados de fora do escopo de implementação e análise. Nesse sentido, pode-se desenvolver trabalhos futuros em relação a esses tópicos não contemplados, como aper-

feiçoamento da interface do jogo, através do armazenamento da disposição de peças, afim de evitar que o jogador tenha que posicionar todas as peças a cada partida (ele pode simplesmente escolher uma organização que já tenha feito e armazenado).

Outro trabalho possível é a utilização de diferentes formas de comunicação entre os dispositivos, como Bluetooth ou HTTP, afim de analisar seus problemas e vantagens de forma prática. Embora a análise dos recursos de comunicação disponíveis no J2ME tenham sido cobertos pelo presente trabalho, não de uma forma mais profunda em virtude da cobertura de outros assuntos, a utilização de outros meios de comunicação poderá disponibilizar informações úteis sobre esse tópico crítico no desenvolvimento de jogos, que é a comunicação entre os dispositivos.

Outra possibilidade seria a implementação do jogo Stratego em uma das outras duas plataformas de desenvolvimento citadas, BREW e Flash Lite. Através dessa implementação, a análise das vantagens e desvantagens da plataforma J2ME seria maximizada e aumentaria os subsídios informacionais gerados, o que poderia contribuir com estudos e análises futuras entre as plataformas abordadas.

REFERÊNCIAS

ADOBE SYSTEMS INCORPORATED. **Adobe Flash Player**. Disponível em: <http://www.adobe.com/products/flashplayer/>.

ADOBE SYSTEMS INCORPORATED. **Adobe Flash Lite**. Disponível em: <http://www.adobe.com/products/flashlite/>.

ADOBE SYSTEMS INCORPORATED. **Adobe Flash Lite 2.1 Datasheet**. Disponível em: <http://www.adobe.com/products/flashlite/productinfo/overview/datasheet.pdf>.

ARDFRY IMAGING, LLC. **PNGout**. Disponível em: <http://www.ardfry.com/pngoutwin/>.

BARROS, A. R. **Estudo sobre a utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário**. Lavras: Universidade Federal de Lavras, 2005.

FASTERJ.COM. **Java Performance Tuning**. 2007.

HEDLUNG, M.; MAXE, R. **Multiplayer Games in Mobile Terminals**. 2004. Dissertação (Mestrado) — Lulea University of Technology, Sweden.

HÄGGLUND, G. . TeliaSonera, Sundsvall, Sweden: [s.n.], 2004.

KARLSSON, B. F. F.; NASCIMENTO, I. F.; BARROS, T. G. F.; SANTOS, A. L. M.; RAMALHO, G. L. Análise de tecnologias de desenvolvimento de jogos para dispositivos móveis. In: WORKSHOP BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL (WJOGOS), 2003. **Anais...** [S.l.: s.n.], 2003.

KLINGSHEIM, A. N. **J2ME Bluetooth Programming**. 2004. Dissertação (Mestrado) — University of Bergen, Norway.

LAFORTUNE, E. **ProGuard**. Disponível em: <http://proguard.sourceforge.net/>.

LILJESTAM, L. . TeliaSonera, Sundsvall, Sweden: [s.n.], 2004.

LINSALATA, D.; SLAWSBY, A. **Addressing Growing Handset Complexity with Software Solutions**. Disponível em: <http://www.adobe.com/mobile/news_reviews/articles/2005/idc_whitepaper.pdf>.

MAHMOUD, Q. H. **J2ME Low-Level Network Programming with MIDP 2.0**. Disponível em: <http://developers.sun.com/techtopics/mobility/midp/articles/midp2network/>.

NOKIA CORPORATION. **Designing Single-player Mobile Games. Version 1.01**. [S.l.: s.n.], 2003.

NOKIA CORPORATION. **Developer Platform 1.0 for Series 40 and Series 60**: known issues. Version 1.1. [S.l.: s.n.], 2003.

NOKIA CORPORATION. **Ngage.com**. Disponível em: <http://www.ngage.com>.

ORTIZ, C. E. **The Generic Connection Framework**. Disponível em: <http://developers.sun.com/techtopics/mobility/midp/articles/genericframework/>.

PEREIRA, D. A. **Jogos Ubíquos com Bluetooth**. Recife: Universidade Federal de Pernambuco, 2006.

QUALCOMM INCORPORATED. **Qualcomm Brew Developer Home**. [S.l.: s.n.], 2006. Disponível em: <http://brew.qualcomm.com/brew/en/developer/overview.html>.

RABELO, S.; HAHN, R. M.; BARBOSA, J. BlueGame: um jogo móvel multijogador baseado em comunicação bluetooth. In: WORKSHOP BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL (WJOGOS), 2005, São Paulo. **Anais...** [S.l.: s.n.], 2005. n.4.

SACRAMENTO, D. A. A.; ROCHA, G.; ESMIN, A. A. A. Proposta de um framework para construção de jogos multi-usuários para telefones celulares. In: WORKSHOP BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL (WJOGOS), 2005. **Anais...** [S.l.: s.n.], 2005.

SUN MICROSYSTEMS. **The Java ME Platform**. [S.l.: s.n.], 2006. Disponível em: <http://java.sun.com/j2me/>.

SUN MICROSYSTEMS, INC. **OverView (MID Profile)**. [S.l.: s.n.], 2006.

SUN MICROSYSTEMS, INC. **J2ME Development**. [S.l.: s.n.], 2006.

TRUTA, C. **OptiPNG**. Disponível em: <http://optipng.sourceforge.net/>.

WIKIPEDIA. **Stratego**. Disponível em: <http://en.wikipedia.org/wiki/Stratego>.

WISNIEWSKI, D. et al. 2005 Mobile Games White Paper. In: GAME DEVELOPERS CONFERENCE, 2005. **Anais...** [S.l.: s.n.], 2005. Disponível em: <http://www.igda.org/online/>.