UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

DESENVOLVIMENTO DE UM JOGO MULTIPLAYER PARA CELULAR

VILSON TRUPEL

BLUMENAU
2008

2008/2-27

VILSON TRUPEL

DESENVOLVIMENTO DE UM JOGO MULTIPLAYER PARA CELULAR

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos, Dr. - Orientador

BLUMENAU
2008

2008/2-27

DESENVOLVIMENTO DE UM JOGO MULTIPLAYER PARA CELULAR

Por

VILSON TRUPEL

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
	_____
Membro:	Prof. Paulo César Rodacki Gomes, Dr. – FURB
	_____
Membro:	Prof. Miguel Alexandre Wisintainer, Ms. – FURB

Blumenau, 10 de Fevereiro de 2009

Dedico este trabalho a meus pais e minha família pelo apoio e incentivo, à minha noiva e a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

Aos meus pais e minha família, por tudo.

À minha noiva, pela paciência e compreensão nas ausências.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Mauro Marcelo Mattos, por ter acreditado e confiado na conclusão deste trabalho.

Pensar é mais interessante do que saber, mas não é mais interessante do que agir.

Johann Wolfgang Von Goethe

RESUMO

Este trabalho apresenta a especificação e o desenvolvimento de um jogo *multiplayer* para celular através da plataforma J2ME utilizando MIDP. Para o desenvolvimento, foi escolhido o jogo Batalha Naval. Apresenta também, a especificação e implementação da aplicação servidora bem como um protocolo para comunicação entre eles, utilizando para isto, *sockets*.

Palavras-chave: Jogos *multiplayer*. J2ME. MIDP. Dispositivos móveis.

ABSTRACT

This paper presents the specification and development of a multiplayer game for mobile platform by using J2ME MIDP. For development, it was chosen the game Naval Battle. It also presents the specification and implementation of application server as well as a protocol for communication between them, using for this, socket.

Key-words: Multiplayer games. J2ME. MIDP. Mobile devices.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

BREW – *Binary Runtime Environment for Wireless*

CDC – *Connected Device Configuration*

CLDC – Connected Limited Device Configuration

HTTP – *Hyper Text Transfer Protocol*

JAD – Java Archive

JAR – Java Application Descriptor

JEE – Java Enterprise Edition

J2ME – Java Standard Edition

J2SE – Java Enterprise Edition

JSP – Java Server Pages

JVM – Java Virtual Machine

KVM – *Kilobyte Virtual Machine*

MMO – *Massively Multiplayer*

MMMOG ou 3MOG – *Massively Mobile Multiplayer Online Games*

PDA – *Personal Digital Assistant*

XML – *eXtensible Markup Language*

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

LISTA DE SÍMBOLOS

- sustenido

SUMÁRIO

1 INTRODUÇÃO

O mercado de dispositivos móveis desponta como sendo um dos mais promissores e rentáveis dos últimos tempos, devido à facilidade de aquisição de um aparelho celular por qualquer pessoa.

Apoiado neste mercado e através do avanço tecnológico contínuo dos dispositivos móveis, associados ao crescimento da infra-estrutura das redes de telecomunicação e de plataformas de desenvolvimento de aplicações móveis como o J2ME e o BREW, tornou-se possível o desenvolvimento de softwares mais complexos e de jogos com qualidade cada vez mais superior.

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).

Através de um aparelho celular é possível, hoje, acessar a rede mundial de computadores, a Internet, permitindo com isso, o desenvolvimento de jogos *multiplayer* para celulares.

De acordo com Menezes (2006, p. 1), desenvolver um jogo *multiplayer* para dispositivos móveis não é uma tarefa simples, pois devem ser superados vários desafios tecnológicos, como a alta latência das redes de telefonia móvel e a baixa capacidade de processamento dos dispositivos móveis.

Diante do exposto, propõem-se desenvolver um jogo de Batalha Naval (YAHOO GAMES, 2008) para celular em modo *multiplayer* onde um usuário, poderá ter como adversário, o servidor do jogo ou outro cliente. O cliente poderá optar ainda por jogar individualmente contra outro jogador ou em duplas. No jogo em duplas, o servidor deverá formar as duplas e juntar as duplas adversárias.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver o jogo Batalha Naval em modo *multiplayer* para dispositivos móveis (telefone celular).

Os objetivos específicos do trabalho são:

a) disponibilizar um cliente do jogo Batalha Naval para celular em plataforma J2ME;
b) disponibilizar um servidor que possa mediar a comunicação com os demais clientes do jogo;
c) permitir que uma partida do jogo seja composta de dois e no máximo quatro jogadores.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado em quatro capítulos, possibilitando uma melhor compreensão do mesmo. No primeiro capítulo, é apresentada a introdução, na qual estão dispostas as razões que levaram ao desenvolvimento deste trabalho, contendo seus objetivos e relevância. No segundo capítulo, é feito um aprofundamento nos conceitos, tecnologias, ferramentas e jogos em modo *multiplayer* para dispositivos móveis que se fizeram necessários para a compreensão e confecção do tema proposto. No terceiro capítulo, são expostos o desenvolvimento e os resultados obtidos com o mesmo. Finalizando, no quarto capítulo, são apresentadas as conclusões a respeito do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções seguintes serão apresentados alguns aspectos teóricos relacionados aos recursos e tecnologias empregados no desenvolvimento deste trabalho. Será abordado o tema da tecnologia Java, seguido de um aprofundamento na tecnologia J2ME. São tratadas na seqüência, os assuntos de jogos para celular, jogos *multiplayer* e a arquitetura de servidores. Posteriormente, são abordados a descrição das características do jogo Batalha Naval e os trabalhos correlatos ao trabalho desenvolvido.

2.1 TECNOLOGIA JAVA

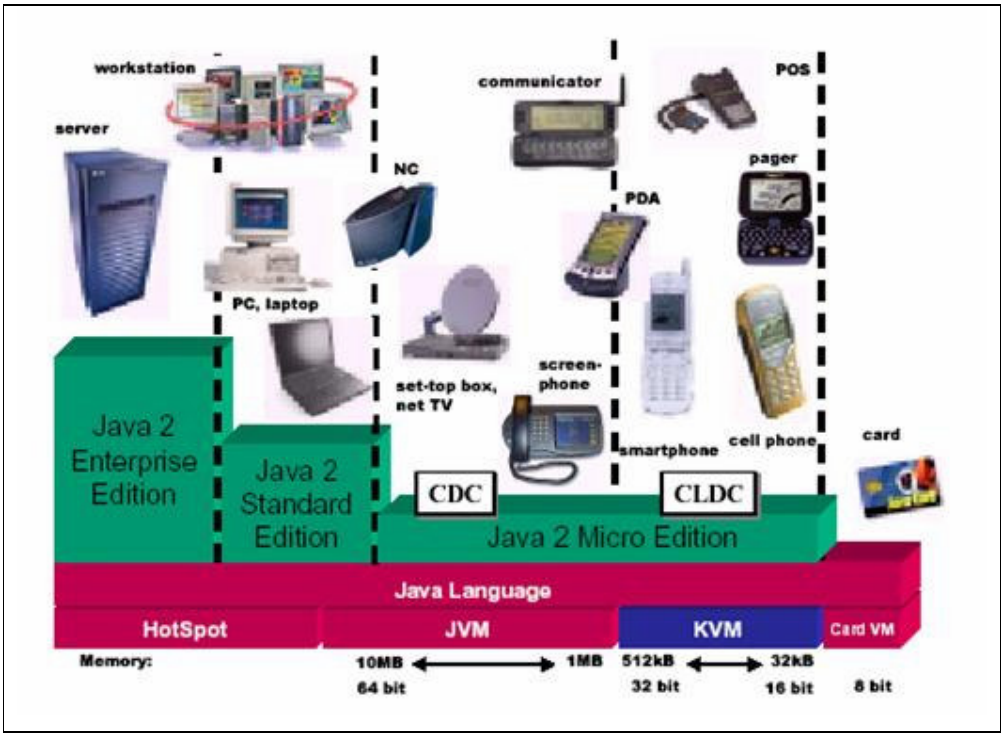
Conforme Sun (2007), a tecnologia Java define uma linguagem de programação e uma plataforma de software, cujo principal objetivo é a portabilidade do código.

Os programas desenvolvidos em Java rodam em diferentes ambientes graças a um componente da plataforma chamado JVM, que é um tipo de tradutor de código Java para instruções específicas de cada sistema e dispositivo (CARMO, 2008, p.4).

De acordo com Muchow (2006, p. 2), a tecnologia Java compreende três edições:

a) J2SE: projetada para a execução em máquinas simples de computadores pessoais e estações de trabalho;
b) J2EE: com suporte interno para servlets, JSP e XML, essa edição é destinada a aplicativos baseados no servidor;
c) J2ME: projetada para dispositivos com memória, vídeo e poder de processamento limitados.

A figura 1 mostra a plataforma Java em suas três edições.



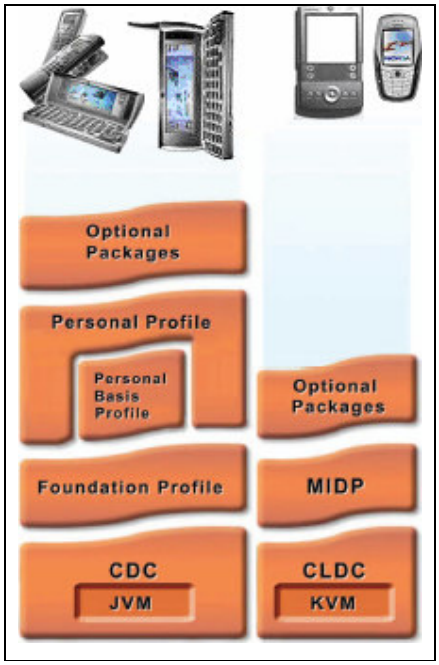
Fonte: Ferraz (2006).

Figura 1 – Plataforma Java

2.2 PLATAFORMA J2ME

Conforme Gomes (2008), a plataforma J2ME tem por objetivo definir um conjunto de tecnologias centradas no desenvolvimento de soluções para dispositivos cujos recursos são limitados, mas que por carregarem algum tipo de micro-processador são potencialmente capazes de realizar operações computacionais. A linha de dispositivos capazes de suportar a tecnologia J2ME varia desde *pagers* e celulares até grandes aparelhos domésticos como televisores digitais, refrigeradores e automóveis.

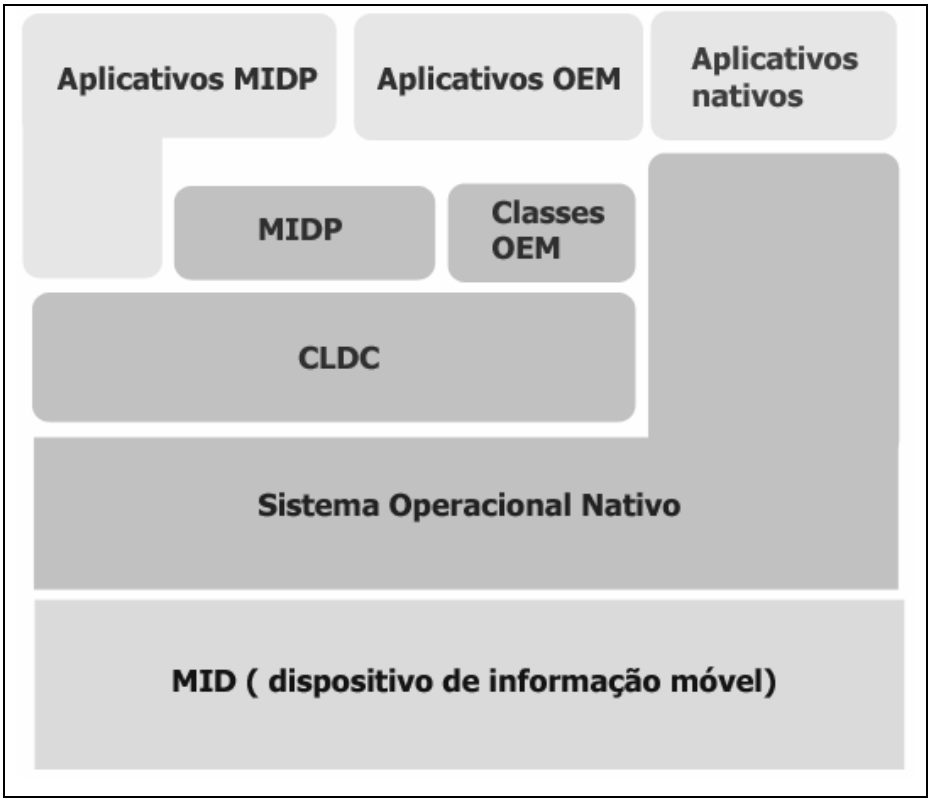
Conforme Caldeira (2002), o J2ME é dividido em *Configurations* (Configurações) e *Profiles* (Perfis). Estes oferecem informações sobre as APIs de diferentes famílias de equipamentos (Figura 2).



Fonte: Andriolli (2008).

Figura 2 – Plataforma J2ME

Na figura 3 é apresentada a arquitetura para o desenvolvimento de aplicações J2ME ao qual, na camada mais básica encontramos o componente chamado de Dispositivo de Informação Móvel (MID). Em uma camada mais acima encontramos o sistema operacional nativo. Acima da camada que representa o sistema operacional nativo temos a Configuração de Dispositivo Conectado Limitado (CLDC) e a Configuração de Dispositivo Conectado (CDC). O Perfil de Dispositivo de Informação Móvel (MIDP) é o componente que, em termos de software, representa as limitações impostas por uma configuração. Como complemento ao perfil MIDP, ainda podemos considerar as classes específicas do Fabricante de Equipamento Original (OEM). Na última camada dos componentes da arquitetura J2ME temos três subcomponentes: os aplicativos MIDP, os aplicativos específicos do OEM e os aplicativos nativos.



Fonte: Cisneiros (2005).

Figura 3 – Arquitetura das aplicações J2ME

2.2.1 Configurações

Segundo Muchow (2006, p. 5), para suportar uma ampla variedade de produtos que se encaixam dentro do escopo do J2ME, a Sun introduziu o conceito de "configuração". Uma configuração define uma plataforma Java para uma ampla variedade de dispositivos. A linha divisória de aplicação de uma configuração é, de modo geral, baseada na memória, no vídeo, na conectividade de rede e no poder de processamento disponível em um dispositivo. Há basicamente duas configurações típicas: CDC e CLDC.

a) *Connected Device Configuration* (CDC): esta configuração é voltada para dispositivos com no mínimo 512KB de memória para executar a máquina Java, 256KB de memória dinâmica para armazenar variáveis, além de recursos de conectividade de rede e largura de banda possivelmente persistente e alta;
b) *Connected Limited Device Configuration* (CLDC): esta configuração é voltada para dispositivos com 128KB de memória para executar a máquina Java, 32 KB de memória dinâmica para armazenar variáveis, possuindo interface restrita com o

> usuário, baixo poder de processamento, normalmente alimentado por bateria, conectividade de rede geralmente sem fio, largura de banda baixa e com acesso intermitente. Ela define uma máquina virtual Java reduzida, chamada de KVM e um conjunto também reduzido de APIs.

Em termos mais concretos, uma configuração define uma máquina virtual, um conjunto mínimo de bibliotecas e os recursos da linguagem Java que estão disponíveis para os dispositivos de uma determinada categoria (PEREIRA, 2004, p. 2).

2.2.2 Perfil MIDP

Um perfil atende às demandas específicas de uma certa família de dispositivos. Enquanto uma configuração visa aparelhos que possuem recursos de *hardware* semelhantes, um perfil é definido para dispositivos que executam tarefas semelhantes. Ao contrário de uma configuração, perfis incluem bibliotecas mais específicas e vários deles podem ser suportados pelo mesmo dispositivo. Além disso, as classes que fazem parte de um perfil tipicamente estendem aquelas definidas para uma configuração. O MIDP foi o primeiro perfil definido para a plataforma J2ME, tendo sido lançado em novembro de 1999. Esse perfil foi implementado sobre a configuração CLDC. Também para a configuração CDC foi desenvolvido um perfil, o qual ficou conhecido como *Foundation Profile* (VALENTE, 2003, p. 2).

O perfil MIDP, fornece funcionalidades para o desenvolvimento de aplicativos de uma ampla variedade de celulares, utilizando para isso, a configuração CLDC.

2.2.3 Midlet

Midlet é uma aplicação que implementa o perfil MIDP e roda sobre a máquina virtual KVM proposta pela configuração CLDC. Um Midlet pode utilizar tanto funções oferecidas no perfil MIDP como funções que o MIDP herda da CLDC. Uma aplicação MIDP deve conter no mínimo uma classe Midlet, podendo em alguns casos conter mais de um, sendo chamada de Midlet Suíte. A aplicação que segue o MIDP, para ser instalada no dispositivo, precisa ser empacotada, criando um Java Archive (JAR) que obrigatoriamente contém um arquivo de manifesto, englobando as informações sobre o Midlet contido no pacote JAR. Este arquivo

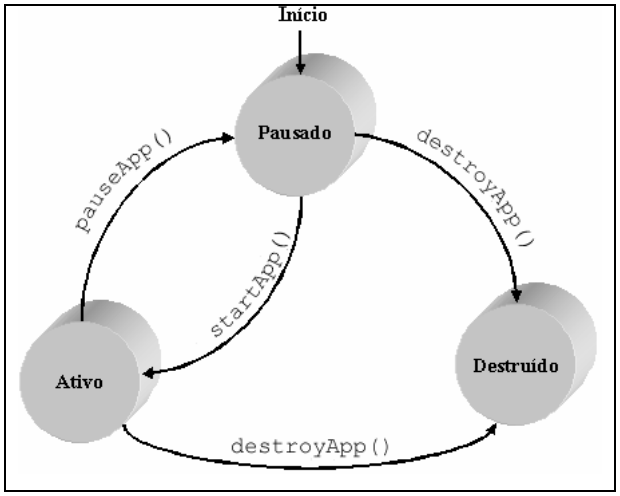
contendo as informações sobre a aplicação, configuração e perfil utilizando na mesma, é denominado JAD (SHMITT JUNIOR, 2004, p.23).

Para Martins (2004, p. 20), um Midlet é uma aplicação que pode ser gerenciada por um aparelho que suporte o MID Profile.

De acordo com Ferreira (2005, p.36), os elementos de um Midlet Suíte, cujo conjunto forma o software de gerenciamento da aplicação, e dos quais se espera que implementem as funções necessárias para instalar, selecionar, rodar e remover Midlets, são:

a) ambiente de execução: é compartilhado por todos os Midlets que estão na mesma Midlet Suíte, e qualquer Midlet pode interagir com outro que esteja no mesmo pacote;
b) empacotamento do Midlet suíte: um ou mais Midlets podem ser empacotados num único arquivo JAR, que contém as classes compartilhadas e os arquivos de recursos pelos Midlets, além de um arquivo manifesto descrevendo seu conteúdo. Existem vários atributos pré-definidos que permitem identificação de um Midlet, com nome, versão, tamanho de dados, descrição, etc.
c) descritor de aplicação: é utilizado para gerenciar o Midlet e é usado pela própria Midlet para atributos de configuração específica. O descritor permite que seja verificado se o Midlet é adequado ao aparelho antes de se carregar todo o arquivo JAR do Midlet Suíte. Ele também permite que parâmetros sejam passados para os Midlets sem modificar os arquivos JAR.
d) ciclo de vida da aplicação: um Midlet não deve possuir um método *public void static void main()*. O software de gerenciamento de aplicação deve suprir a classe inicial necessária pelo CLDC para iniciar o Midlet. Quando um Midlet é instalado, ele é mantido no aparelho e fica pronto para uso. Quando é rodado, uma instância é criada através de seu construtor público sem argumentos, e seus métodos são chamados para passar pelos estados do Midlet. Quando ele é destruído, os recursos utilizados podem ser recuperados, incluindo os objetos criados e suas classes.

A Midlet compreende três estados, onde cada um representa o estado da aplicação. Os estados são: ativo, pausado e destruído (Figura 4).



Fonte: Andriolli (2008).

Figura 4 – Estados de uma Midlet

No quadro 1 tem-se uma aplicação básica, onde esta implementa uma Midlet. Para implementar uma Midlet, é necessário importar as bibliotecas de *javax.microedition.midlet.**. A classe deverá então estender a superclasse Midlet implementando os métodos *startApp()* (para iniciar a aplicação), *pauseApp* (para pausar a aplicação) e *destroyApp()* (para destruir a aplicação).

```
import javax.microedition.midlet.*;
public class MidletBasica extends MIDlet  {
   public void MidletBasica()  {
      System.out.println("Construtor");
   }
   public void startApp()  {
      System.out.println("Início");
   }
   public void pauseApp()  {
      System.out.println("Pausa");
   }
   public void destroyApp(boolean unconditional)  {
      System.out.println("Fim");
   }
}
```

Quadro 1 – Exemplo de uma Midlet básica

2.2.4 Máquina Virtual K

A fim de prover a plataforma J2ME com um ambiente no qual as aplicações pudessem ser executadas, a SUN desenvolveu em 1999 celular uma máquina virtual específica, denominada *Kilobyte Virtual Machine* (KVM). Esta máquina virtual foi desenvolvida para executar em dispositivos dotados de um processador de 16 ou 32 bits e que não dispõem de mais que algumas centenas de *kilobytes* de memória. Estas especificações aplicam-se a uma vasta gama de aparelhos, como por exemplo, telefones celulares digitais e *pagers*, dispositivos de áudio e vídeo portáteis e também pequenos terminais de consulta ou pagamento de débitos (PEREIRA, 2004, p. 2).

O termo K da sigla KVM surgiu para fazer alusão aos poucos *kilobytes* necessários para que a máquina virtual execute uma aplicação na configuração CLDC (JOHNSON, 2006, p. 4).

2.2.5 API de jogos com MIDP

A MIDP 2.0 inclui o pacote *javax.microedition.lcdui*. De acordo com Rosetto (2005 apud NASCIMENTO, 2003, p. 11), as classes deste pacote são:

a) GameCanvas: uma extensão da classe Canvas da MIDP 1.0, responsável por controlar a tela. Esta classe possui um *buffer off-line*, onde as operações de desenho são feitas e depois este *buffer* é desenhado na tela. Outra funcionalidade desta classe é permitir o acesso ao estado das teclas, facilitando identificar quais estão pressionadas, sem ter que ficar dependendo do tratamento dos eventos e possibilitando a identificação de mais de uma tecla pressionada ao mesmo tempo;
b) *Layer*: classe que representa uma matriz de células conhecidas como mapa de *tiles*, e um conjunto de imagens, os *tiles*. Cada célula da matriz está associada a uma imagem, criando um mosaico usado como cenário para o jogo;
c) *Sprite*: classe que representa uma imagem, formada por diversos *frames*. Esses *frames* são usados para construir uma animação para representar o objeto. A classe ainda possui métodos para verificar colisão com outros *layers*;
d) *LayerManager*: possui uma lista ordenada dos *layers*. Gerencia o desenho correto dos *layers* e evita o desenho de áreas que serão encobertas por outros *layers*.

Além destes recursos, foi implementado na MIDP 2.0 o suporte a conexões através de *sockets*, criando melhores condições para implementação de jogos *multiplayer*.

2.3 JOGOS PARA CELULAR

Segundo Computerworld (2008), o mercado mundial de jogos para aparelhos móveis, deve faturar US$ 4,5 bilhões de dólares em 2008 contra US$ 3,9 bilhões de dólares em 2007. A previsão é que esse mercado apresente um crescimento de 10,2%. Até 2011 os gastos com jogos para dispositivos móveis chegarão a US$ 6,3 bilhões de dólares.

Este mesmo site afirma ainda que a Ásia deve liderar o mercado. O faturamento no país, que foi de US$ 80 milhões de dólares em 2007, pode chegar a US$ 450 milhões até 2012.

De acordo com Symbianbr (2008), no Brasil, o faturamento de jogos para celular foi de aproximadamente US$ 30 milhões e cresce na casa dos 30% anuais. Os jogos para celular são os principais propulsores do mercado brasileiro de jogos.

> Mas aos poucos, o Brasil ultrapassa o mercado interno e ganha espaço lá fora, abrindo novas frentes. Atualmente ocupa o 13º lugar na lista dos maiores países em desenvolvimento de software para celular e tem como carro-chefe a produção de jogos eletrônicos, indústria que movimenta algo em torno de R$18 milhões, segundo dados da Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos (Abragames). O lugar que mais abriga empresas de desenvolvimento é o Paraná, com 33% seguido de São Paulo, 30%, e Rio de Janeiro, 12%. Das 55 empresas em atividade no país, 22% focam em aplicações para telefones móveis. O que significa mais vagas na área de desenvolvimento de jogos. (UNIVERSIA, 2005, p. 6, grifo do autor).

O desenvolvimento de jogos para celular requer um conhecimento específico das limitações dos aparelhos devido às diferenças em relação à maioria das aplicações. Por ser um sistema computacional, herda diversas características no que diz respeito ao processo de desenvolvimento, porém possui particularidades na documentação envolvida, na definição da arquitetura e métodos de programação bem particulares.

Segundo Rabelo, Hahn e Barbosa (2005, p. 3) atualmente existe uma grande variedade de dispositivos móveis, os quais possuem configurações individuais, com capacidades distintas orientadas as mais variadas tarefas. Jogos para esses dispositivos devem buscar um denominador comum, respeitando os limites de cada plataforma. Sendo assim, os jogos com pouca ação e baixa utilização de recursos de rede são ideais, pois requerem pouco poder de processamento, geralmente não necessitam de atualizações de telas freqüentes e cabem em

uma memória de armazenamento limitada.

Para Junior (2008, p. 6), a variedade de aparelhos celulares provoca um trabalho adicional aos desenvolvedores de software para esses dispositivos, porque se torna necessário ter versões diferentes da mesma aplicação para atender a estes diferentes dispositivos. É comum, por exemplo, ter várias versões de um mesmo jogo para celular, cada um com mudanças apenas no tamanho das imagens a serem apresentadas. Este problema é conhecido como fragmentação e tem se revelado como uma das maiores dificuldades para o desenvolvedor de aplicações para dispositivos móveis.

2.4 JOGOS MULTIPLAYER

Para Pedrosa (2006, p. 20), sendo aplicações de espaço compartilhado, é fundamental para jogos *multiplayer* manter o estado global consistente entre os jogadores. Para isto, as ações de cada jogador devem ser repassadas entre os jogadores de forma rápida, confiável e ordenada, de modo a garantir a correta execução dos eventos no jogo. Porém, a separação física dos componentes e jogadores decorrente da natureza distribuída de ambientes virtuais em rede impõe obstáculos para a manutenção de um estado global consistente entre os jogadores.

Um dos obstáculos encontrados é o acesso a qualquer parte do mundo do jogo, por parte do jogador, que idealmente deveria fazê-lo independentemente do tipo de dispositivo utilizado. Porém esse requisito gera problemas de adaptação do jogo devido às diferentes características de cada dispositivo, como memória, poder de processamento, latência de conexão (PEREIRA, 2006, p. 35).

De acordo com Jábali (2004, p. 4), pode-se analisar um sistema de jogos *multiplayer* considerando quatro importantes pontos:

a) latência fim-a-fim: em jogos de computadores *online*, muitas vezes faz-se necessário que a latência seja baixa o suficiente para que o usuário não perceba atrasos que o façam tomar decisões antes mesmo que o evento seja recebido. Jogos com temáticas de esportes e *first-person-shooters*, são exemplos que requerem baixa latência na rede. Já os jogos de estratégia, como xadrez e damas, são exemplos que não necessitam de latência tão baixa, uma vez que o usuário tem um grande tempo disponível para pensar em sua ação;

b) largura de banda: em jogos massivos, o número de usuários simultaneamente *online* pode aumentar, teoricamente, sem limites. O estado do jogo é aumentado de acordo com o número de usuários jogando, além de outros itens. Sendo assim, a quantidade de dados que trafega na rede aumenta de acordo com o número de jogadores *online*. Estes jogos (massivos) requerem uma alta largura de banda de cada computador conectado ao jogo, ou estratégias que diminuam a quantidade de dados que necessitem ser trafegados;
c) consistência de estados: manter consistentes os estados de um jogo significa garantir que todos os jogadores concordem com cada acontecimento que ocorre no jogo. Quando dois jogadores tentam pegar uma mesma arma que está no chão de uma sala do jogo, apenas o primeiro deles conseguirá e o outro deve concordar com o fato, para que o estado do jogo permaneça consistente - caso contrário, uma nova arma seria criada e colocada na mão do segundo jogador sem justificativa existencial;
d) controle administrativo: o controle administrativo de um jogo consiste num conjunto de fatores que viabilizam a publicação e a manutenção do produto. Cita-se como exemplo a criação de mundos virtuais persistentes, redução de trapaças (*cheating*), facilidade de atualizações, eliminação de pirataria e pagamento por tempo de jogo.

Pode-se dizer que no cenário de jogos *multiplayer* para dispositivos móveis, há a necessidade de interação entre as considerações anteriormente apresentadas com a jogabilidade, o perfil de jogadores e a interface do jogo em questão.

Com relação à jogabilidade e a interface dos jogos, estes devem ser desenvolvidos de forma a utilizar poucos recursos dos dispositivos, além de garantir que a forma de interação com o usuário leve em conta características como o tamanho do teclado e do d*isplay* e também a conectividade, que nesses dispositivos, são fatores comumente preocupantes no desenvolvimento de um jogo *multiplayer*.

Quanto aos jogadores, segundo Menezes (2008, p. 2) do ponto de vista cultural, há uma mudança na postura e no perfil dos mesmos, que passam a buscar neste tipo de jogos, uma rápida distração de forma que tenham recompensas imediatas e em momentos e lugares rotineiros.

O cenário atual, dos jogos *multiplayer* para dispositivos móveis, ainda enfrenta dificuldades, porém, a tecnologia para este fim está progredindo de forma que problemas como a latência, qualidade dos jogos e interatividade já podem ser contornados.

2.5 ARQUITETURA DE SERVIDORES

Segundo Kozovits e Feijó (2003, p. 5), existem algumas arquiteturas padrão para suportar jogos multiplayer, sendo elas:

a) *peer-to-peer unicast:* não empregam uma boa escalabilidade, pois, cada mudança no estado de um avatar, por exemplo, terá que ser comunicada as N-1 estações de trabalho participantes da simulação;

b) *broadcast* e *multicast*: basicamente ainda uma arquitetura *peer-to-peer*, mas ao invés de ocorrer o envio de mensagens para cada participante, apenas uma única mensagem *broadcast* será enviada a cada atualização de cada participante, reduzindo o número de mensagens requeridas;

c) cliente/servidor: separa as aplicações clientes de aplicações servidoras, sendo interligadas entre si, geralmente utilizando uma rede de computadores. Cada instância de um cliente pode enviar requisições de dados para algum dos servidores conectados na rede e esperar pela resposta. Por sua vez, algum dos servidores disponíveis pode aceitar tais requisições, processá-las e retornar o resultado para o cliente.

A arquitetura cliente/servidor, utilizada para o desenvolvimento deste trabalho, pode ser representada pela figura 5.

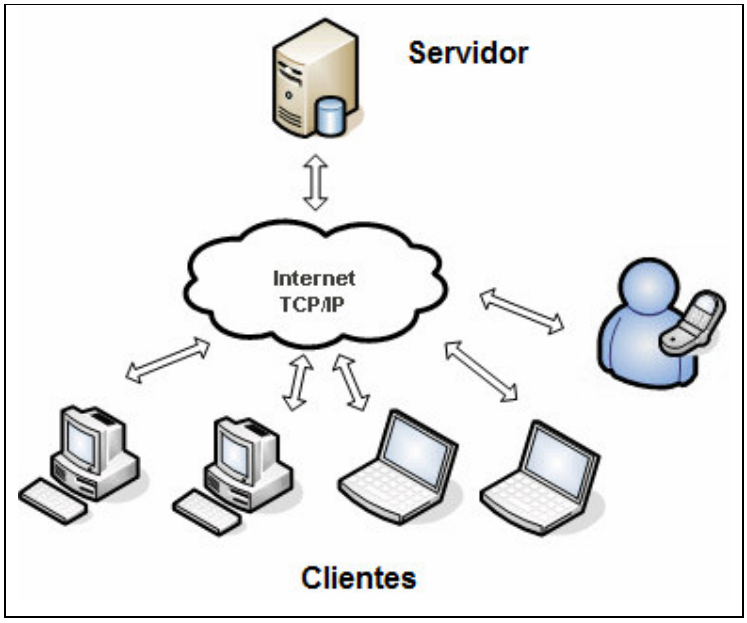


Figura 5 – Arquitetura cliente/servidor

2.6 CARACTERÍSTICAS DO JOGO BATALHA NAVAL

De acordo com Yahoo Games (2008), Batalha Naval é um jogo para dois jogadores onde cada jogador tenta destruir a frota de navios do outro. O jogo pode ser dividido em três modos (selecionáveis ou não pelo jogador):

a) clássico: regras padrão. Cada jogador dispara um tiro por vez;
b) tiro extra ao acertar: toda vez que o jogador acertar um navio inimigo, ganha outra jogada. Quando erra, passa a vez para o adversário;
c) atire tantas vezes enquanto tiver navios: os jogadores têm permissão para continuar atirando enquanto houver navios em suas frotas.

Uma vez que o modo de jogo é definido, devem-se colocar os navios no mapa. Cada jogador começa com cinco navios dos quais, podem ser:

a) porta-aviões: necessita de cinco tiros para ser abatido;
b) destroyer: necessita de quatro tiros para ser abatido;
c) cruzador: necessita de quatro tiros para ser abatido;
d) submarino: necessita de três tiros para ser abatido;
e) barco patrulha: necessita de dois tiros para ser abatido.

A disposição de cada navio no tabuleiro pode ser na horizontal ou vertical.

Depois que o jogador posicionar todos os cinco navios, poderá então ser iniciado o jogo. A escolha do primeiro jogador a jogar é randômica. Uma jogada consiste em selecionar uma região do mapa sendo que se a região selecionada contiver um pedaço de um navio, o jogador poderá jogar novamente até errar e a vez é dada ao adversário. O primeiro jogador a afundar os cinco navios do adversário, será o campeão.

2.7 TRABALHOS CORRELATOS

Jogos *multiplayer* para celular ainda é um assunto tratado como novidade devido às circunstâncias levantadas em seções anteriores. Alguns jogos possuem funcionalidades semelhantes às propostas neste trabalho. Na seqüência, é apresentado o jogo Cellmons, assim como o desenvolvimento de jogos para dispositivos móveis utilizando MIDP: implementação do jogo Tetris (ROSETTO, 2005) e também a arquitetura servidora de jogos de celular online

massivamente *multiplayer* (CEREJA, 2008).

2.7.1 Cellmons

Segundo Palermo et al. (2006, p. 4), Cellmons é um jogo experimental desenvolvido em um projeto apoiado pela Financiadora de Estudos e Projetos (FINEP) e caracteriza-se por ser um jogo de batalha em turnos. Os jogadores assumem papéis de treinadores que devem evoluir suas mascotes (os Cellmons) com cuidados e treinamentos. Menezes et al. (2006, p. 4), afirma que dentre os serviços implementados, podem-se citar dois grupos: serviços gerais, inerentes a qualquer jogo *multiplayer* e específicos, próprios do jogo. Do primeiro grupo, destacam-se o gerenciamento de contas de usuário e jogadores, pontuação e classificação (*ranking*), serviços de mensagens entre jogadores e notícias; do segundo, o serviço de lógica de batalhas e treinamentos.

Para evitar o excessivo consumo de processamento e memória dos dispositivos móveis, foi descartado o uso de cenários interativos reapresentando cada um dos lugares do jogo. Ao invés disso, foram utilizados menus contextualizados com o intuito de passar ao jogador a impressão de seu personagem estar realmente em um determinado local. Esta técnica torna intuitiva e fácil a interação do usuário com o jogo, pois deixa claro que ações o jogador pode tomar em cada local (PALERMO et al., 2006, p. 5).

De acordo com Palermo (2006, p. 5), o cenário do jogo é um mundo dividido em quatro locais (Figura 6):

a) a *Arena*, onde os jogadores se encontram para trocar mensagens e desafios;
b) o *Correio*, para ler e enviar mensagens aos demais jogadores e também receber notícias relacionadas ao mundo do jogo;
c) o *Parque*: onde os Cellmons podem descansar sempre que necessitar a recuperação da energia gasta nos combates e treinamentos;
d) o *Ginásio*, para que os Cellmons realizem treinamentos para evoluir seus atributos, como forças e disposição.

Fonte: Pallermo et all (2006).

Figura 6 – Cellmons

2.7.2 Desenvolvimento de jogos para dispositivos móveis utilizando MIDP: Implementação do jogo Tetris

Rosetto (2005) desenvolveu o jogo Tetris para celular sobre a plataforma J2ME utilizando MIDP.

As principais funcionalidades do jogo são:

a) o jogo apresenta mudança de níveis ou fases;
b) o jogo apresenta um *ranking* com os três melhores lugares atingidos;
c) o jogo deve suportar quatro tipos de sons;
d) o jogo deve terminar quando o número de quadrados atingirem a parte superior da tela;
e) deve ser possível movimentar a peça corrente.

O autor testou com sucesso as funcionalidades do jogo em um emulador e em um aparelho celular verificando assim, as diferenças entre estes. Além disso, na época do desenvolvimento deste trabalho, para que o jogo fosse testado em um aparelho celular, foi necessário desenvolvê-lo utilizando MIDIP 1.0, pois segundo o autor, no Brasil, havia poucos aparelhos que suportavam MIDIP 2.0 e estes possuíam um custo elevado dificultando assim, o desenvolvimento.

2.7.3 Arquitetura servidora de jogos para celular online massivamente multiplayer

Cereja (2008) desenvolveu uma arquitetura servidora para jogos massivamente *multiplayer* bem como um cliente do jogo *Galaxy Navigation* para testar esta arquitetura. As principais funcionalidades são:

a) efetuar *login* de clientes do jogo devidamente instalados em telefones celulares;
b) permitir a conexão de todos os clientes de celulares que efetuarem *login*, até um limite máximo estabelecido pelo administrador do servidor;
c) persistir os perfis de cada jogador na base de dados;
d) controlar as regras do jogo estabelecidas;
e) suportar a interação dos jogadores humanos com personagens NPC;
f) repassar aos servidores de jogo as ações pretendidas pelo usuário;
g) atualizar e exibir o ambiente em que se encontra o usuário, a partir dos dados recebidos dos servidores distribuídos de jogo.

Os resultados obtidos neste trabalho foram que o autor, apesar de não ter feito um teste massivo sobre o trabalho desenvolvido a fim de testar a arquitetura e de utilizar somente algumas instâncias de um emulador ao invés de aparelhos de celular, todos os dados dos jogadores foram persistidos de modo que o cliente do jogo *Galaxy Navigation* pode ser descrito como um exemplo de jogo MMO para celulares.

3 DESENVOLVIMENTO

Nesta seção é descrito o desenvolvimento do jogo cliente/servidor para celular utilizando para isto, a tecnologia J2ME. O jogo escolhido é o Batalha Naval.

Para tanto, esta seção tem a finalidade de detalhar os principais requisitos que o jogo deve atender bem como descrever os passos e os resultados obtidos com a implementação. Na seção abaixo, são apresentados os requisitos do trabalho desenvolvido.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Nesta seção são apresentados os requisitos funcionais e não funcionais que detalham as principais funcionalidades que a aplicação deverá ter para que se torne viável o jogo *multiplayer* em celulares:

a) o jogador deverá informar seu nome e sua preferência pelo jogo individual (contra outro jogador) ou em duplas (Requisitos Funcionais – RF);
b) o servidor deverá monitorar uma porta específica para o registro de novos jogadores (RF);
c) o servidor deverá monitorar uma porta para cada jogador que já esteja registrado (RF);
d) o servidor deverá associar os jogadores aos adversários conforme a preferência escolhida por cada um (RF);
e) o servidor não deverá implementar o conceito de *ranking* (RF);
f) o jogador deverá submeter seus dados ao servidor a cada nova rodada do jogo (RF);
g) o jogo deverá permitir uma interação entre o jogador e seu avatar (RF);
h) o servidor do jogo deverá ser implementado na linguagem de programação Java (Requisito Não Funcional – RNF);
i) o jogo deverá ser implementado na plataforma J2ME (RNF);
j) deverá ser utilizado no desenvolvimento do jogo a configuração CLDC e o perfil MIDP 2.0 (RNF);
k) a comunicação entre o servidor e o cliente do jogo, deverá ser feito utilizando

socket (RNF).

3.2 ESPECIFICAÇÃO

Para fazer a especificação do servidor e do cliente do jogo, foram utilizados diagramas da UML. Os diagramas escolhidos para melhor apresentar as funcionalidades do sistema são: diagrama de casos de uso, digrama de seqüência, diagrama de classes, diagrama de atividades e o diagrama de disposição. A ferramenta adotada para fazer estes diagramas foi o *Enterprise Architect*.

3.2.1 Diagrama de casos de uso

Segundo Debone (2004), um caso de uso descreve um objetivo que um ator externo ao sistema tem com o sistema. Um ator pode ser um elemento humano ou não que interage com o sistema. O ator se encontra fora do escopo de atuação do sistema, enquanto o conjunto de casos de uso forma o escopo do sistema.

A figura 7 tem por finalidade especificar as ações que o jogador faz sobre o cliente do jogo.

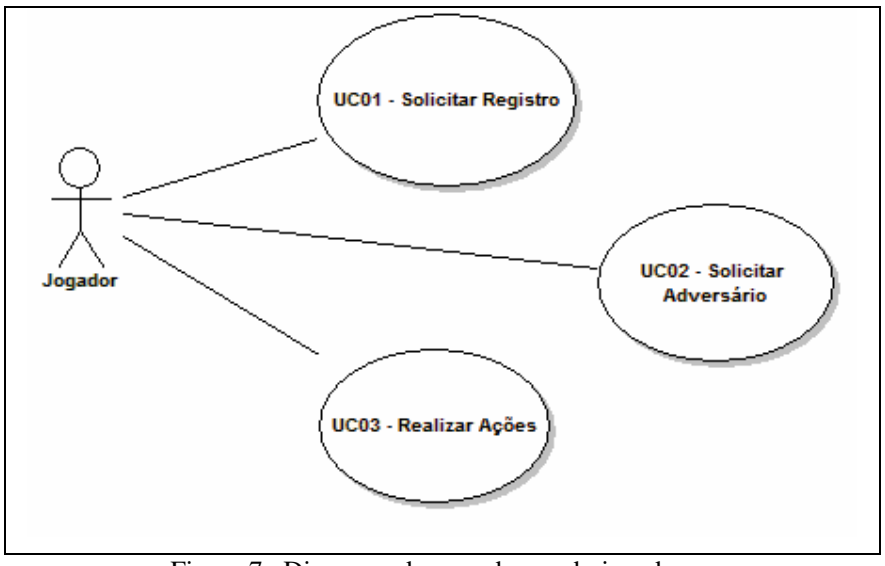


Figura 7– Diagrama de caso de uso do jogador

Nos quadros 2, 3 e 4 estão descritos os cenários dos casos de uso que dizem respeito à interação do jogador com o cliente do jogo e deste, com o servidor. Será detalhada a forma com que o jogador atua sobre o cliente do jogo e este solicita ações para o servidor, fazendo com que este procure por um adversário ou simplesmente atualize os dados do adversário na instancia mantida do mesmo no servidor.

UC01-Solicitar Registro
Descrição: Neste cenário, o jogador deverá informar alguns dados e o cliente do jogo fará uma solicitação de registro do jogador ao servidor.
Ator Principal: Jogador
Cenário Principal: Solicitar Registro
1. o sistema apresenta a tela de entrada do jogo;
2. o jogador seleciona “Ok”;
3. o sistema apresenta a tela de entrada de dados;
4. o jogador informa seu nome e preferência de jogo e seleciona “Ok”.
5. o cliente do jogo gera uma chave temporária para o jogador e envia uma mensagem para o servidor contendo os dados informados pelo jogador e a chave gerada pelo cliente, solicitando o registro do jogador;
6. o servidor recebe a mensagem do cliente e seleciona uma porta, instancia uma thread exclusiva para tratar a comunicação deste cliente, enviando na seqüência a porta de comunicação, o código do jogador e a chave enviada pelo cliente;
7. o cliente do jogo recebe os dados do servidor (código do jogador, chave e porta) e associa ao jogador, apresentando na tela a mensagem: “Aguardando adversário”.

Cenário de Exceção:
1. no passo 2 se o jogador selecionar “Sair”, o sistema deverá encerrar a execução;
2. no passo 4 se o jogador selecionar “Voltar”, o sistema deverá apresentar a tela de entrada;
3. no passo 5 caso seja a primeira conexão do cliente do jogo, o jogador é questionado se deseja permite que a aplicação tenha acesso a rede para enviar e receber dados. Se selecionar “Yes”, continua no fluxo senão, se selecionar “No”, a aplicação será encerrada.

Pré-condições:
1. o jogador ter instalado em seu celular o cliente do jogo;
2. o servidor tem que estar iniciado.

Pós-condições:
1. o cliente do jogo solicita um adversário para o jogador.

Quadro 2 – Cenário do Caso de Uso UC01-Solicitar Registro

UC02-Solicitar Adversário
Descrição: A partir do momento que é feito o registro do jogador no servidor, o cliente do jogo fica solicitando um adversário para o jogador até que o servidor retorne um adversário.
Ator Principal: Cliente do jogo
Cenário Principal: Solicitar Adversário
1. o cliente do jogo envia uma mensagem para o servidor solicitando um adversário para o jogo;
2. o servidor verifica se há algum jogador na situação "Livre" e com a mesma preferência do jogador como adversário ao jogador solicitante e vice-versa.
3. o servidor envia o código e o nome do adversário para ambos os jogadores;
4. o cliente do jogo recebe os dados do servidor e apresenta a tela de jogo com os personagens já posicionados.

Cenário de Exceção:
1. no passo 1, caso o jogador tenha escolhido a preferência de jogo em duplas, o cliente do jogo envia uma mensagem ao servido de comunicação solicitando um parceiro para compor a dupla e uma dupla de jogadores adversários.

Pré-condições:
1. o jogador tem que estar previamente registrado no servidor;
2. o servidor tem que estar iniciado.

Pós-condições:
o jogador tem um adversário no caso da preferência de jogo "Individual" ou um parceiro e uma dupla de jogadores adversários para o caso da preferência de jogo em duplas.

Quadro 3 – Cenário do Caso de Uso UC02-Solicitar Adversário

UC03-Realizar Ação
Descrição: Neste cenário, o jogador deverá movimentar seu personagem e atirar no adversário. Estas ações terão que ser enviadas pelo cliente do jogo para o servidor a fim de atualizar seu personagem no adversário.
Ator Principal: Jogador
Cenário Principal: Realizar Ação
1. o cliente envia uma mensagem de atualização dos dados do jogador para o servidor;
2. o servidor recebe as informações e atualiza a instancia do jogador que tem armazenado, enviando na seqüência uma mensagem para o jogador, contendo os dados de atualização de seu adversário;
3. o cliente do jogo recebe os dados de atualização de adversário e volta ao passo 1.

Cenário de Exceção:
1. no passo 4 se o cliente do jogo receber uma mensagem de seu adversário informando que o jogo acabou, é apresentada a tela informando da vitória/derrota e a pontuação obtida pelo jogador e pelo adversário.

Pré-condições:
1. ter um adversário ou um parceiro e uma dupla de adversários;
2. o servidor tem que estar iniciado.

Pós-condições:
1. voltar ao cenário UC01-Solicitar Registro.

Quadro 4 – Cenário do Caso de Uso UC03-Realizar Ação

3.2.2 Diagrama de classes

Os diagramas de classes descrevem as classes que formam a estrutura do sistema e suas relações. As relações entre as classes podem ser associações, agregações ou heranças. As classes possuem além de um nome, os atributos e as operações que desempenham para o sistema. Uma relação indica um tipo de dependência entre as classes, essa dependência pode ser forte como no caso da herança ou da agregação ou mais fraca como no caso da associação, mas indicam que as classes relacionadas cooperam de alguma forma para cumprir um objetivo para o sistema (DEBONE, 2004).

A figura 8 representa o diagrama de classes da aplicação cliente do jogo Batalha Naval. Neste diagrama a principal classe é a `ServidorGUI`. Quando iniciado o servidor nesta interface, será instanciada e apresentada a interface da classe `JogadorGUI` e também será instanciada a classe `ServidorConexao` e solicitado a execução do método `iniciarServidor()`, ao qual instancia a classe `ServidorConexaoThread` para receber e registrar novos jogadores.

Quando da identificação de um novo jogador, através do método `receberMensagem()`, é instanciada a classe `ServidorComunicacao` e `Jogador`. Na classe `ServidorComunicacao` é armazenada uma lista de jogadores aos quais quando um jogador termina uma partida, o jogo é encerrado e o vínculo de cada instância da classe `Jogador` no servidor é destruído (incluindo a instância da classe `ServidorComunicacao` exclusiva deste jogador).

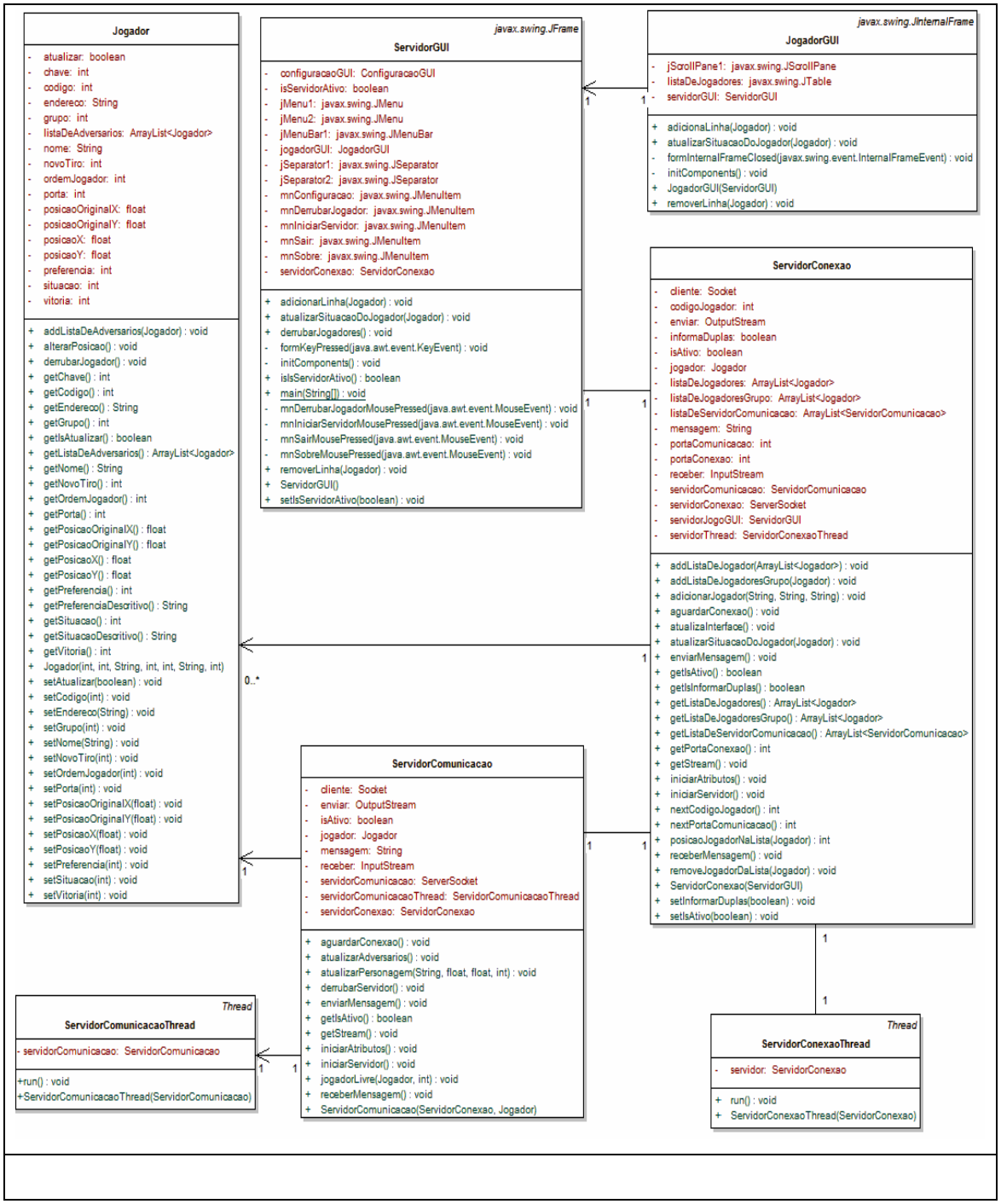


Figura 8 – Diagrama de classes do servidor do jogo

A figura 9 representa o diagrama de classes da aplicação cliente do jogo Batalha Naval. Neste diagrama, principal classe do cliente do jogo é a classe `ClienteGUI`. Nesta classe é apresentada a tela de entrada e um formulário para o jogador. Esta classe possui o método `solicitarRegistro()` que com o auxílio de uma *Thread*, implementada na classe pelo *Runnable*, faz uma solicitação de registro do jogador ao servidor. O retorno do servidor é tratado pelos métodos `desempacotarDados` e `tratarDados()`. Ao receber o retorno, é

instanciada a classe `ClienteCanvas` através do método `apresentarJogo()`. Na classe cliente `ClienteCanvas` é instanciado tantos personagens quantos forem a preferência do jogador (incluindo o próprio).

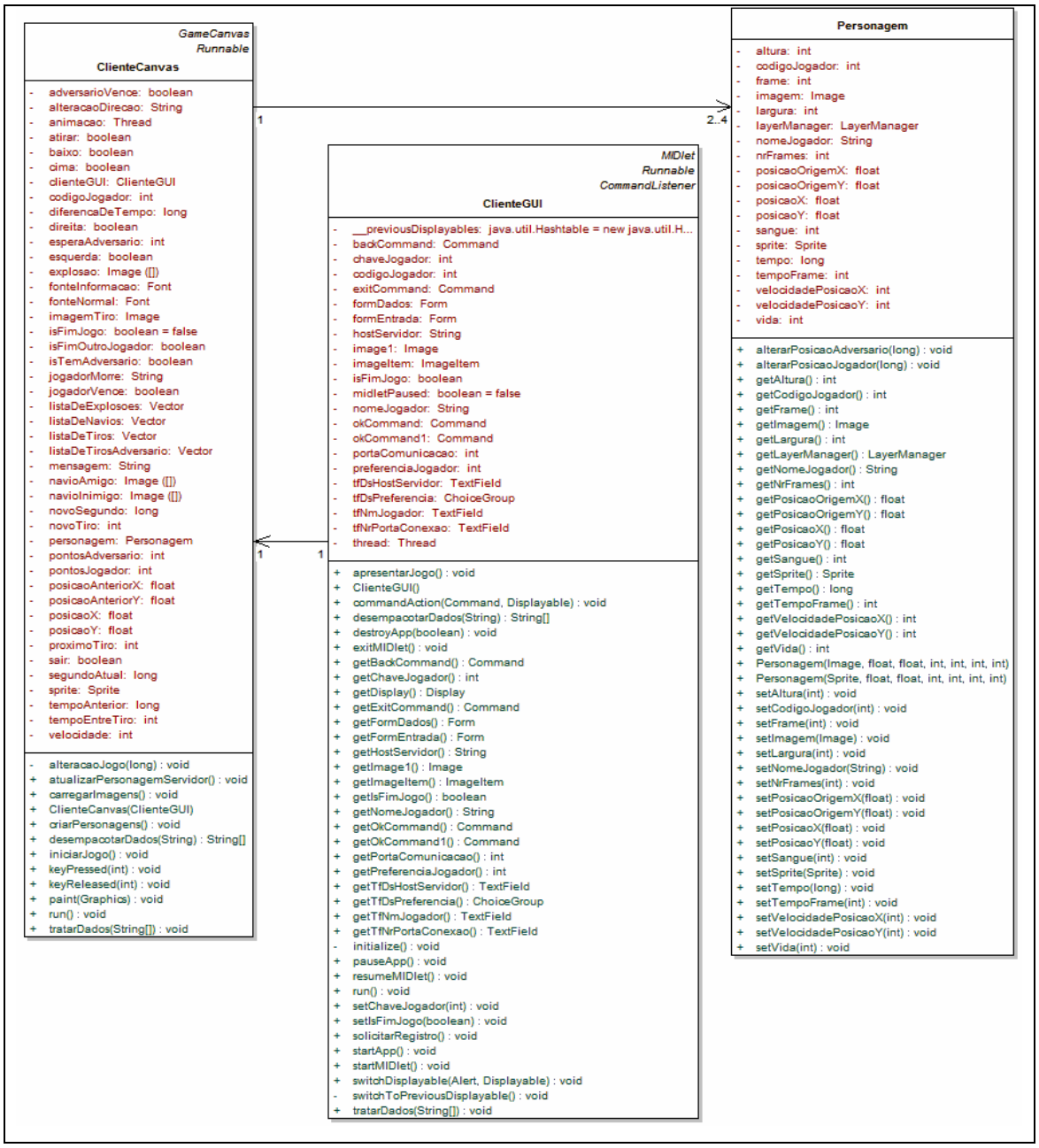


Figura 9 – Diagrama de classes do cliente do jogo

3.2.3 Diagrama de seqüência

De acordo com Debone (2004), o diagrama de seqüência de eventos permite modelar

processo através da troca de mensagens (eventos) entre os objetos do sistema. Os objetos são representados por linhas verticais e as mensagens como setas que partem do objeto que invoca outro objeto. As setas podem ser cheias para indicar uma mensagem de chamada ou tracejadas para indicar uma mensagem de retorno.

No diagrama de seqüência apresentado pela figura 10, é definida a troca de mensagens entre os objetos, seguindo o cenário do diagrama de caso de uso UC01-Solicitar registro. Neste cenário, é apresentada a solicitação de registro do cliente do jogo para o servidor.

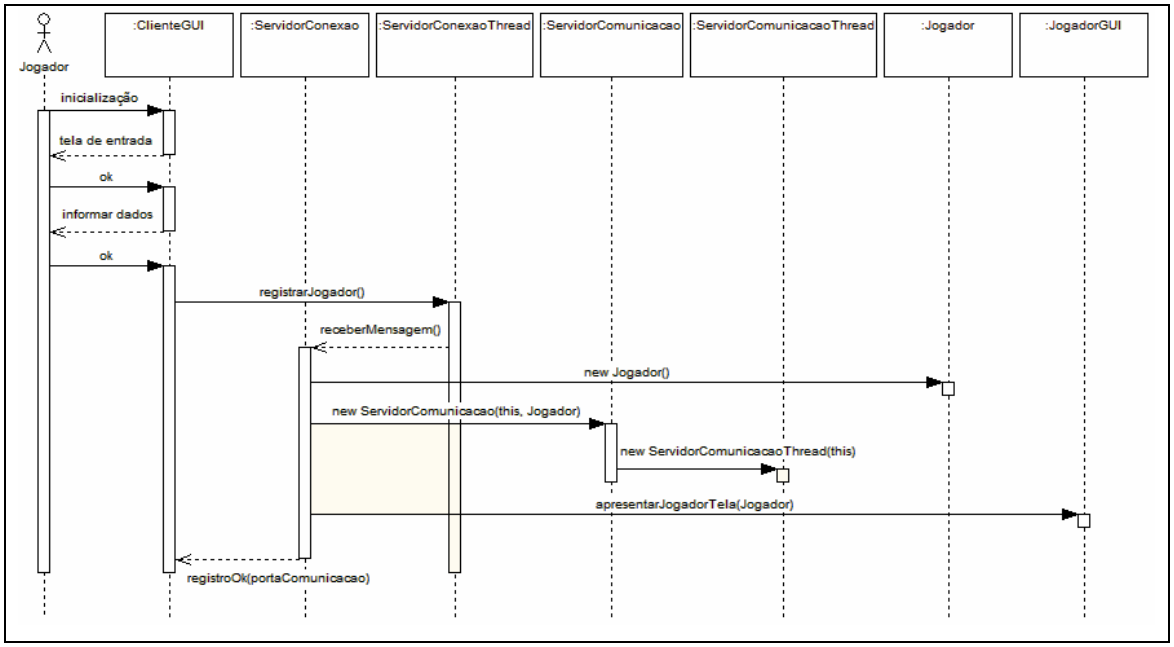


Figura 10 – Diagrama de seqüência referente ao caso de uso UC01

No diagrama de seqüência apresentado pela figura 11, é definida a troca de mensagens entre os objetos, seguindo o cenário do diagrama de caso de uso UC02-Solicitar adversário. Neste cenário, é apresentada a solicitação de um adversário feita pelo cliente do jogo para o servidor. Este faz uma pesquisa para verificar da disponibilidade de algum jogador e então, responde ao cliente informando se tem ou não adversário disponível no momento.

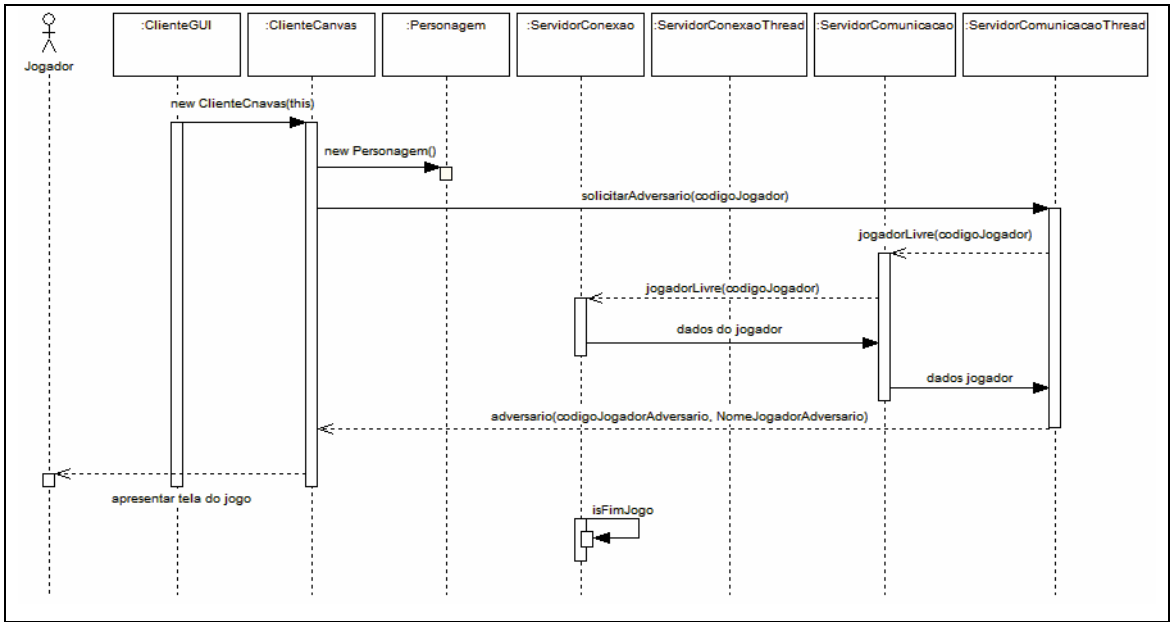


Figura 11 – Diagrama de seqüência referente ao caso de uso UC02

No diagrama de seqüência apresentado pela figura 12, é definida a troca de mensagens entre os objetos, seguindo o cenário do diagrama de caso de uso UC03-Realizar ação. Neste cenário, é apresentada a forma com que o jogador realiza uma ação e esta é enviada ao servidor que faz a atualização em uma instância ao qual podemos chamar de "global" do jogador por estar no servidor e enviar como resposta, a atualização de seu adversário. Esta troca de mensagens é realizada até que o jogo se encerre.

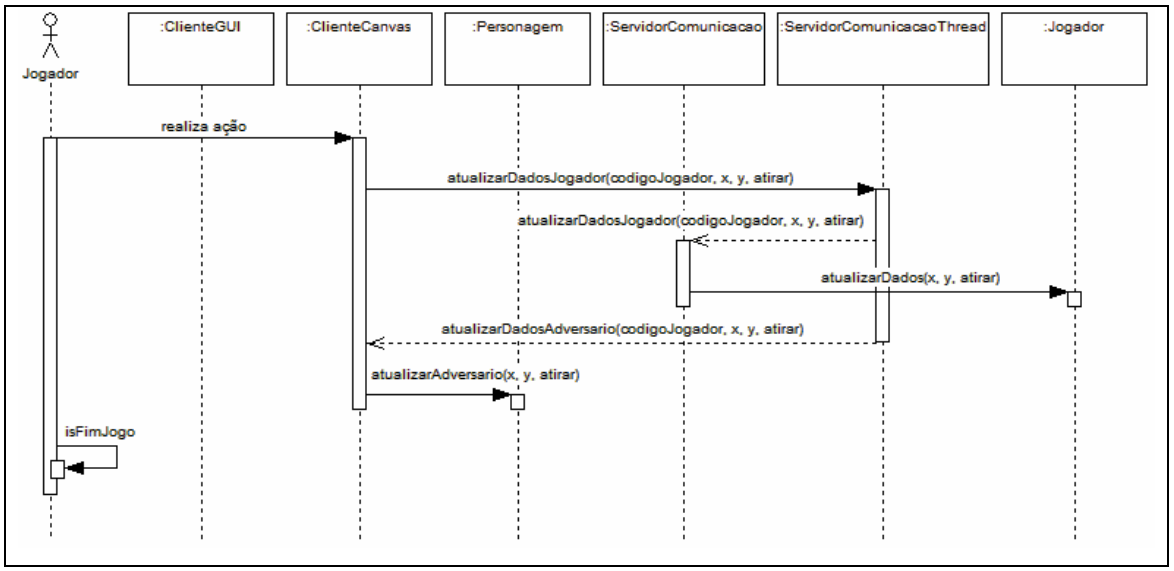


Figura 12 – Diagrama de seqüência referente ao caso de uso UC03

3.2.4 Diagrama de atividade

De acordo com Rosetto (2005, apud BOOCH, RUMBAUGH e JACOBSON, 2000), um diagrama de atividades é essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra.

Na figura 13 é apresentado o fluxo de execução do servidor e do cliente do jogo.

Em um primeiro momento, quando o servidor do jogo é iniciado, ele monitora constantemente uma porta específica determinada nas configurações (foi utilizada a porta 50000 para testes). É através dessa porta que o servidor recebe as requisições de registro enviadas pelos clientes do jogo. O servidor para de monitorar esta porta, somente quando é encerrado.

A partir do momento em que um cliente do jogo faz uma solicitação de registro de um jogador para o servidor através da mesma porta monitorada pelo servidor, este identifica a solicitação e armazena uma instância do jogador com os dados enviados pelo cliente do jogo. Depois, instancia uma *Thread* de comunicação específica para o cliente do jogo a fim de enviar e receber respostas somente deste cliente através de uma porta única determinada pelo servidor do jogo.

Feito o registro do jogador, o servidor do jogo envia uma resposta para o cliente, informando a porta de comunicação (foi utilizada a porta 20000 até 49999 para testes) que deverá ser utilizada pelo mesmo para a troca de mensagens entre eles.

De posse da porta para comunicação, o cliente solicita um ou uma dupla de adversários (dependendo da preferência de jogo selecionada pelo jogador) para o jogador. Quando o servidor achar um ou a dupla de adversários, a cada um, é associado o código de seu adversário e então, o servidor informa ao cliente os dados do adversário do jogador. O cliente, então, apresenta a tela de jogo com as imagens dos navios nas posições iniciais. O jogo terá chegado a seu fim, somente quando um adversário tiver perdido suas três vidas, ou seja, o jogador destruir três vezes o navio do adversário.

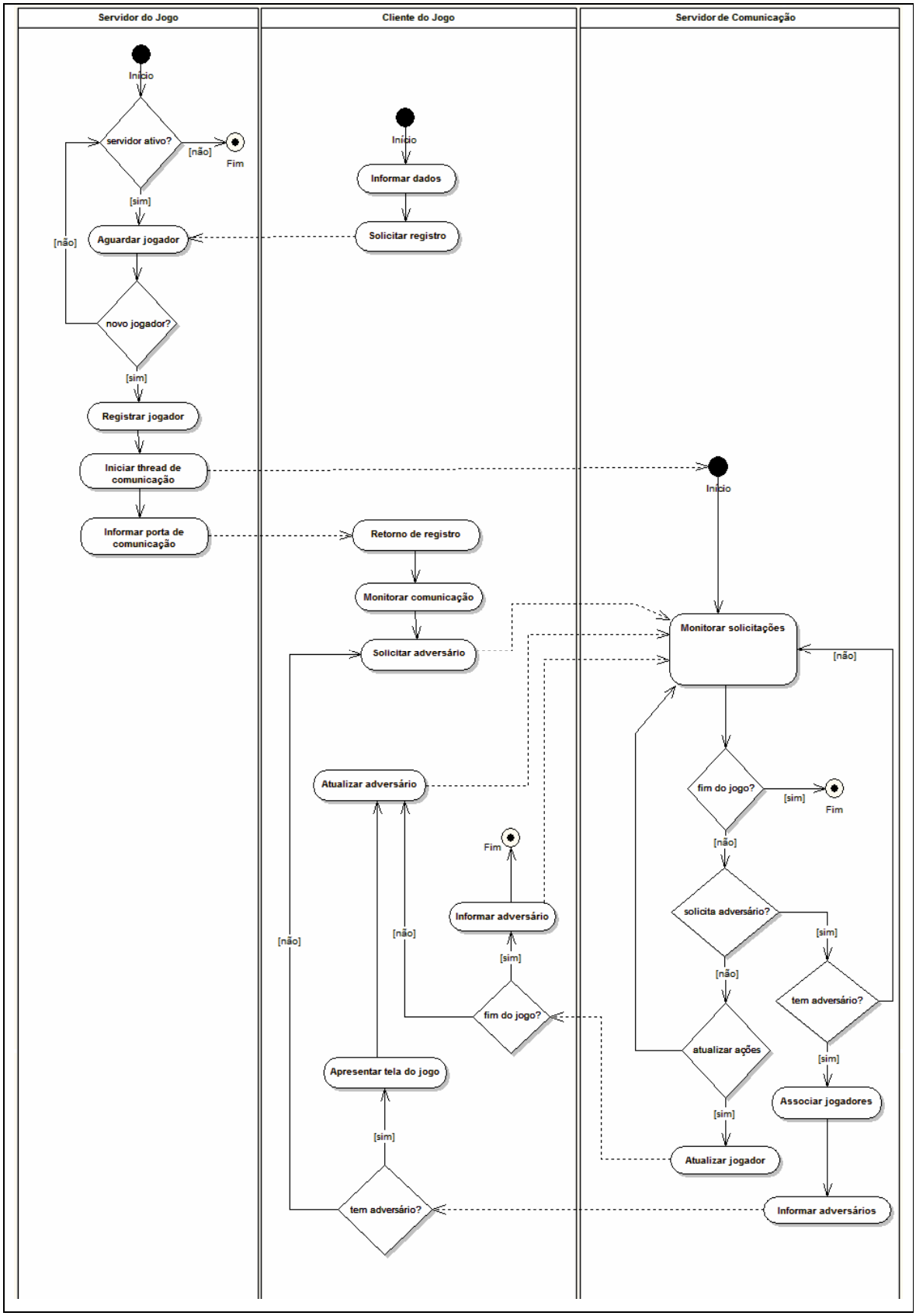


Figura 13 – Diagrama de atividades

3.2.5 Diagrama de disposição

Segundo Medeiros (2004, p. 207), um diagrama de disposição modela o inter-relacionamento entre recursos de infra-estrutura, de rede ou artefatos de sistema. Este diagrama é normalmente usado para representar servidores. Estes recursos são chamados de *nodes* ou nós. Cada nó é uma máquina física que encerra um ou vários componentes. Outros dispositivos podem ser apresentados com o estereótipo de <<dispositivo>> ou <<device>>.

O diagrama de disposição apresentado na figura 14 representa a arquitetura da aplicação. O diagrama nos dá uma perspectiva de como foi implementado e implantado o jogo de forma que a aplicação servidora é executada sobre um microcomputador PC enquanto que a aplicação cliente é executada em um celular. A comunicação entre elas é feita utilizando-se de *socket* sobre o protocolo TCP/IP.

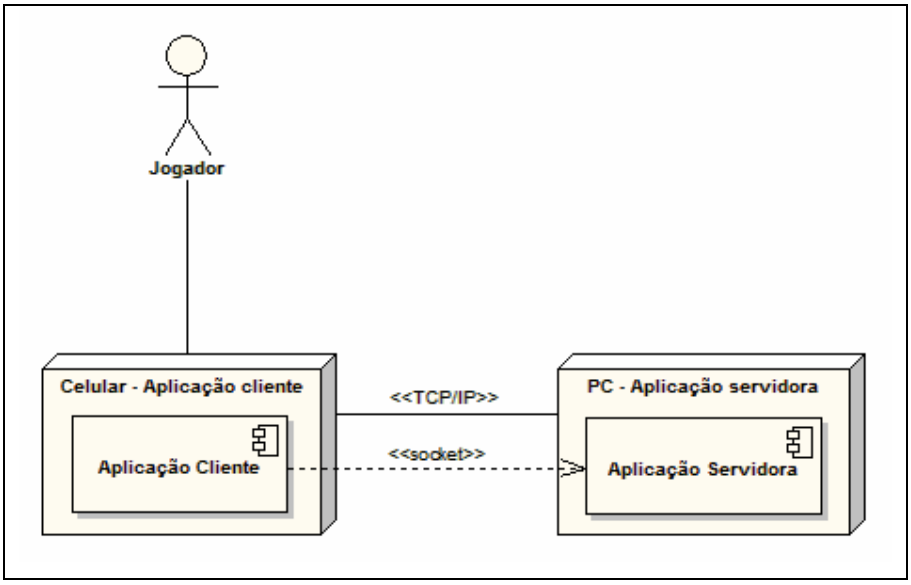


Figura 14 – Diagrama de disposição

3.2.6 Protocolo de comunicação

Para que este trabalho tivesse êxito, tornou-se necessário o desenvolvimento de uma aplicação servidora para receber e controlar os jogadores.

A aplicação cliente e a servidora, comunicam-se através de *socket*, tendo em vista que no MIDP 2.0, há suporte para a comunicação tanto para *socket* quanto para *http* e datagrama. Optou-se pela comunicação com *socket* pela simplicidade no desenvolvimento, haja vista que

na conexão *http*, por exemplo, é necessário a passagem de um cabeçalho padrão do protocolo.

No quadro 5 têm-se um exemplo das formas de conexão com o MIDP 2.0.

```
Connector.open(“http://127.0.0.1”);
Connector.open(“socket://127.0.0.1:50000”);
Connector.open(“datagram://127.0.0.1:50000”);
```

Quadro 5 – Comunicação MIDP

Para fazer com que ambas as aplicações pudessem interpretar quais comandos executar através da mensagem recebida, foi criado um “protocolo de comunicação”, que reflete a troca de mensagens entre as aplicações fazendo com que as mesmas tenham um “significado” para a aplicação que a receba.

No quadro 6 é apresentado uma descrição dos campos obrigatórios em uma mensagem do protocolo de comunicação. Nela pode-se observar que sempre ao formular-se uma mensagem, esta deve conter no primeiro campo, o código do jogador (na mensagem de solicitação de registro do cliente é gerado um número randômico e posto no lugar do código do jogador, sendo que o mesmo ainda não possui o código que é distribuído pelo servidor) e no segundo campo, a ação que deve ser tomada pelo outro lado ao receber a mensagem. Os atributos posteriores a estes são opcionais podendo ter até um total de 19 atributos em uma mensagem.

Caso a mensagem tenha sido enviada pelo servidor, esta deve conter um ponto e vírgula no fim, pois o cliente ao desempacotar a mensagem, precisa necessariamente saber onde a ela termina devido ao tratamento por caractere sendo que o ponto e vírgula é o delimitador de atributos da mensagem. Já o cliente não precisa enviar a mensagem com um ponto e vígula no fim, pois o servidor já separa seus atributos em um vetor sem a necessidade de tratar os caracteres.

```
{codigoJogador;acao;}
- codigoJogador - código do jogador
- acao - ação que indica o que deve ser feito
```

Quadro 6 – Definição do protocolo

Na figura 15 é apresentado, através de um diagrama de seqüência, o protocolo de comunicação entre o servidor e o cliente do jogo.

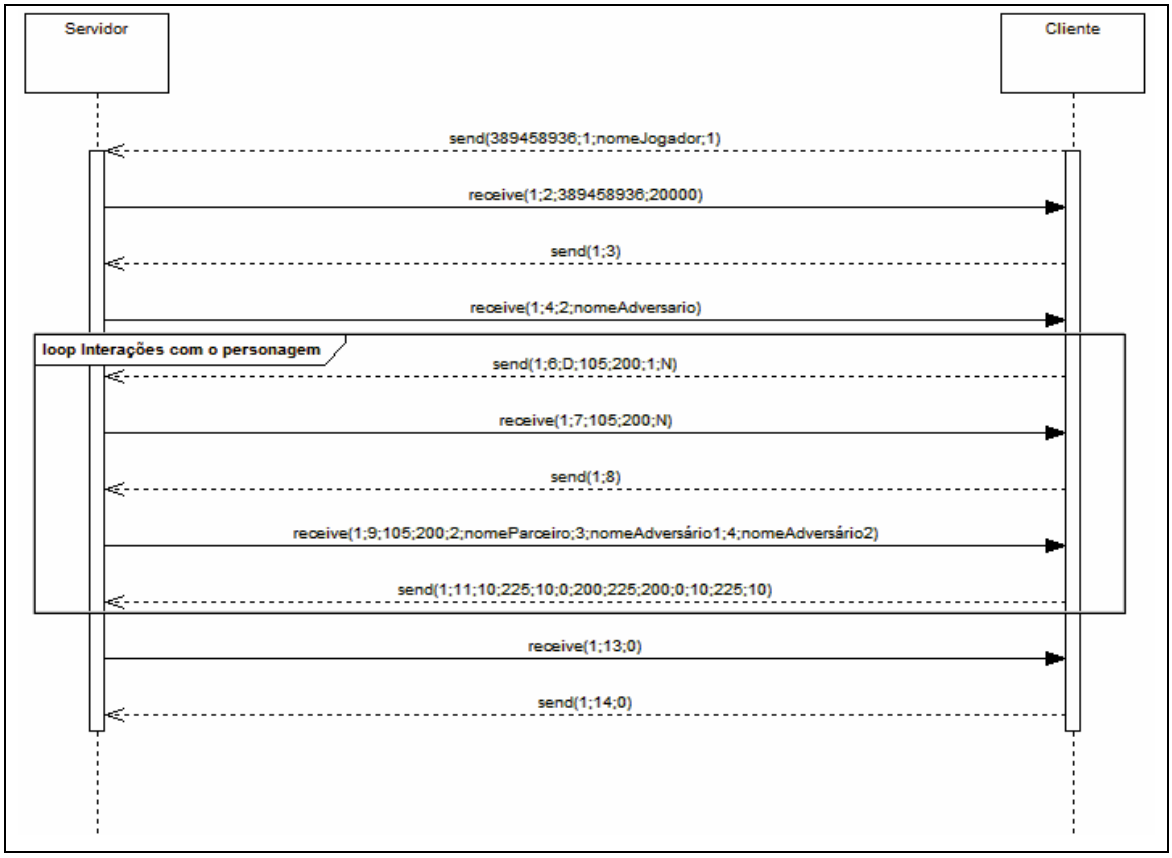


Figura 15 – Diagrama de seqüência do protocolo de comunicação

A seguir, é apresentado o protocolo desenvolvido mostrando cada ação por meio de quadros separados.

No quadro 7 é apresentado o envio da solicitação de registro do cliente para o servidor. Nesta mensagem, é enviado um número randômico (ao qual foi chamado de chave de identificação, para que o cliente identifique quando o servidor enviar a resposta desta solicitação), a ação que deve ser tratada, o nome do jogador e sua preferência de jogo (0 - individual ou 1 - em dupla).

{389458936;1;nomeJogador;1}
- 389458936 - chave de identificação
- 1 - identificador da ação
- nomeJogador - nome do jogador
- 1 - preferência de jogo

Quadro 7 – Ação de solicitação de registro do jogador

O quadro 8 é apresenta a mensagem de resposta do servidor para o cliente do jogo. Nesta mensagem, o servidor envia ao cliente, o código de registro (que será único para cada cliente), o identificador da ação, a chave de identificação gerada pelo cliente e a porta ao qual

será monitorada pelo servidor, exclusivamente para as próximas trocas de mensagem entre o servidor e este cliente em específico.

{1;2;389458936;20000;} - 1 - código do jogador - 2 - identificador da ação - 389458936 - chave de identificação - 20000 - porta de comunicação com o servidor

Quadro 8 – Ação de resposta do servidor a solicitação de registro

O quadro 9 apresenta a mensagem de solicitação de adversário que parte do cliente para o servidor a partir do momento que este, recebe a resposta de solicitação de registro (apresentada no quadro 8). Esta mensagem é composta pelo código do jogador e pelo código da ação. Este código indica ao servidor do jogo, que o jogador solicita um adversário para um jogo individual.

{1;3} - 1 - código do jogador - 3 - ação de solicitação de adversário para jogo individual

Quadro 9 – Ação de solicitação de adversário para jogo individual

O quadro 10 é apresentada a mensagem de envio de adversário do servidor para o cliente do jogo. Esta mensagem é composta pelo código do jogador, identificador da ação, código do jogador adversário e nome do adversário.

{1;4;2;nomeAdversario;} - 1 - código do jogador - 4 - identificador da ação - código do adversário - nomeAdversario - nome do jogador adversário

Quadro 10 – Ação de retorno de adversário

O quadro 11 apresenta a mensagem enviada pelo cliente para atualização do personagem do jogador no servidor. Esta mensagem é composta pelo código do jogador, identificador da ação, a direção de movimento do personagem, as novas coordenadas X e Y e se o jogador atirou ou não. Esta mensagem é enviada constantemente para o servidor, após o cliente ter recebido o adversário para o jogador.

{1;6;D;105;200;1;N} - 1 - código do jogador - 6 - identificador da ação - D - posição de alteração do personagem (D;E;C;B) - 105;200 - posições X e Y - 0/1 - jogador atirou - S/N - jogador morreu (caso houver latência na conexão)

Quadro 11 – Ação de atualização dos dados do jogador no servidor

No quadro 12, temos a mensagem de envio do servidor para o cliente, com dados de atualização de seu adversário. A mensagem é composta pelo código do jogador, o identificador da ação, as posições X e Y atuais e se o adversário atirou ou não.

{1;7;105;200;N;} - 1 - código do jogador - 7 - identificador da ação - 105;200 - posições X e Y - S/N - Identifica se o jogador atirou

Quadro 12 – Ação de atualização dos dados do jogador no adversário

O quadro 13 apresenta a mensagem enviada pelo cliente ao servidor, solicitando um parceiro para o jogador e uma dupla adversária. Esta mensagem somente é enviada, quando o jogador escolher a preferência de um jogo em dupla. A mensagem enviada, é composta pelo código do jogador e pelo identificador da ação.

{1;8} - 1 - código do jogador - 8 - identificador da ação

Quadro 13 – Ação de solicitação de um parceiro e outra dupla de jogadores

O quadro 14 apresenta a mensagem de retorno do servidor para o cliente do jogo, informando os dados do parceiro do jogador e da dupla adversária. A mensagem é composta pelo código do jogador, o identificador da ação, as coordenadas de início do personagem do jogador, o código e nome do parceiro, o código e nome do adversário um e código e nome do adversário 2.

{1;9;105;200;2;nomeParceiro;3;nomeAdversário1;4;nomeAdversário2;} - 1 - código do jogador; - 9 - identificador da ação - 105;200 - podições originais X e Y - 2 - código do parceiro - nomeParceiro - nome do parceiro - 3 - código do adversário 1 - nomeAdversário1 - nome do adversário 1 - 4 - código do adversário 2 - nomeAdversário2 - nome do adversário 2

Quadro 14 – Ação de retorno de um parceiro e uma dupla adversária para o jogador

O quadro 15 apresenta o envio dos dados de atualização dos jogadores apresentados no quadro 14 para cada jogador que compõem o jogo. Esta mensagem é composta pelo código do jogador, identificador da ação e pelas posições X e Y dos jogadores e se atiraram ou não.

{1;11;10;225;10;0;200;225;200;0;10;225;10} - 1 – código do jogador - 11 – identificador da ação - 10;225;10;0;200;225;200;0;10;225;10 – posições X e Y dos outros jogadores

Quadro 15 – Ação de atualização dos dados dos adversários para um jogo em duplas

O quadro 16 apresenta a mensagem de atualização enviada pelo servidor para o cliente da atividade do jogo informando que o adversário venceu o jogo. Esta mensagem é necessária para casos de latência na rede. A mensagem é composta pelo código do jogador, identificador da ação e se o jogador venceu.

{1;13;0;} - 1 - código do jogador - 13 - identificador da ação - 0/1 - informa se a vitória foi do jogador ou do adversário

Quadro 16 – Ação de atualização da atividade do jogo

O quadro 17 apresenta a mensagem de atualização enviada pelo cliente ao servidor referente à atividade do jogo. Esta mensagem, assim como a apresentada no quadro 13, é necessária para casos de latência na rede. A mensagem é composta pelo código do jogador, identificador da ação e se o jogador venceu.

{1;14;0} - 1 - código do jogador - 14 - identificador da ação - 0/1 - informa se a vitória foi do jogador ou do adversário

Quadro 17 – Ação que informa ao servidor que o jogo acabou

3.3 IMPLEMENTAÇÃO

Nesta seção, serão mostradas as técnicas e ferramentas utilizadas para o desenvolvimento da aplicação servidora bem como a aplicação cliente.

3.3.1 Técnicas e ferramentas utilizadas

Para detalhar e especificar os diagramas descritos neste que representam as aplicações foi utilizado à ferramenta *Enterprise Architect*. Para o desenvolvimento das aplicações, optou-se por utilizar a ferramenta Netbeans 6.1 devido a maior estabilidade e suporte no desenvolvimento de aplicações MIDP.

Para a comunicação entre o cliente e o servidor do jogo, utilizou-se *socket*. O quadro 18 representa o método de inicialização do servidor de conexão. Para efeito de testes, utilizou-se a porta 50000 (representada pelo atributo `portaConexao`), do servidor, para aguardar a solicitação de registro dos jogadores.

```
public void iniciarServidor() throws IOException, ClassNotFoundException  {
      try {
            this.servidorConexao = new ServerSocket(this.portaConexao);
      }catch(Exception e) {
         System.out.println("Erro no Servidor: " + e.getMessage());
      }

      this.servidorThread = new ServidorConexaoThread(this);
      this.servidorThread.start();
}
public synchronized void aguardarConexao() throws IOException {
      this.cliente = this.servidorConexao.accept();
}
public synchronized void getStream() throws IOException, ClassNotFoundException {
      this.enviar  = this.cliente.getOutputStream();
      this.receber = this.cliente.getInputStream();
}
```

Quadro 18 – Inicialização do servidor de conexão

No quadro 19 são apresentados os métodos de envio e recebimento de mensagem do servidor de conexão. No método de recebimento de mensagens, caso identifique que a ação seja 1 então, é acionado o método `adicionarJogador()`. Esse método cria uma nova instancia das classes `Jogador` e `ServidorComunicacao`. O servidor de comunicação instanciado fará o controle de mensagens recebidas e mensagens a serem enviadas para o cliente do jogo instanciado. A comunicação se dará através de uma porta específica determinada pelo servidor de conexão. Para testes, foi delimitado que a porta para comunicação se dará a partir da 20000. Esta porta é repassada para o servidor de conexão no momento da instancia e para o cliente do jogo como resposta a solicitação de registro no servidor ao qual envia juntamente com esta, o código do jogador no servidor e a chave ao qual o cliente identifica quando for uma mensagem específica para ele, pois neste momento, ainda não possui o código de identificação.

```
public synchronized void enviarMensagem() throws IOException {
      if (this.mensagem == null) {
         this.mensagem = "#;1;";
      }
      this.enviar.write(this.mensagem.getBytes());
      this.enviar.flush();
      this.enviar.close();
      this.mensagem = null;
}
public synchronized void receberMensagem() throws IOException, ClassNotFoundException
{
      int caracter;
         StringBuffer buffer = new StringBuffer();
      while (((caracter = this.receber.read()) != -1)) {
         buffer.append((char)caracter);
      }
      String[] dados = buffer.toString().trim().split(";");
      if (dados[1].equals("1")) {
         this.adicionarJogador(dados[0], dados[2], dados[3]);
      }
}
public synchronized void adicionarJogador(String chave, String nome, String preferencia) {
      this.jogador = new Jogador(this.nextCodigoJogador(),
                                 Integer.parseInt(chave),
                                 nome,
                                 1,
                                 Integer.parseInt(preferencia),
                                 this.cliente.getInetAddress().getHostAddress(),
                                 this.nextPortaComunicacao());
                                 this.listaDeJogadores.add(this.jogador);
                                 this.servidorJogoGUI.adicionarLinha(this.jogador);
                  this.servidorComunicacao = new ServidorComunicacao(this, this.jogador);
                           this.listaDeServidorComunicacao.add(this.servidorComunicacao);
      try {
         this.servidorComunicacao.iniciarServidor();
      }catch (IOException e) {
         System.out.println("Erro no servidor de comunicação: " + e.getMessage());
      }catch (ClassNotFoundException e) {
         System.out.println("Erro no servidor de comunicação: " + e.getMessage());
      }
      this.mensagem = this.jogador.getCodigo() + ";" +
                      2 + ";" +
                      this.jogador.getChave() + ";" +
                      this.jogador.getPorta() + ";";
   }
```

Quadro 19 – Envio e recebimento de mensagem do servidor de conexão

No quadro 20 é apresentada a forma pela qual o servidor de comunicação identifica uma solicitação de adversário por um cliente do jogo. Se o servidor identificar uma ação do

tipo 3, entende-se que o jogador optou pela preferência do jogo “individual”. Para isto, o servidor terá que pesquisar por um jogador que esteja “livre”, ou seja, não esteja jogando e que também tenha escolhido a preferência de jogo “individual”.

Quando o servidor achar um adversário compatível, as duas instâncias dos jogadores são atualizadas no servidor como jogando e nelas, é associado o código de seus adversários. Internamente a instancia da classe jogador no servidor, além de armazenar os atributos de um jogador, armazena a localização real e a posição no adversário uma vez que neste, o personagem do jogador é apresentado de forma invertida e vice-versa.

Para o caso do servidor identificar uma ação do tipo 8, entende-se que o jogador optou pela preferência do jogo em “duplas”. Para isso, o servidor terá que pesquisar primeiramente um parceiro para este jogador e em seguida, formar outra dupla para compor um jogo.

A seleção dos jogadores é feita de forma que o jogador terá que estar na situação “livre” e ter selecionado a opção de jogo em “duplas”. O servidor sempre retornará o primeiro jogador da lista que atenda estes critérios.

```
if (dados[1].equals("3")) {
        if (this.jogador.getListaDeAdversarios().size() > 0) {
           this.mensagem = this.jogador.getCodigo() + ";" +
                        4 + ";" +
                        this.jogador.getListaDeAdversarios().get(0).getCodigo() + ";" +
                        this.jogador.getListaDeAdversarios().get(0).getNome() + ";";
                       this.servidorConexao.atualizarSituacaoDoJogador(this.jogador);
        }else {
            this.jogadorLivre(this.jogador, 1);
        }
}
if (dados[1].equals("8")) {
        (this.jogador.getListaDeAdversarios().size() > 2)) {
        If
(this.servidorConexao.getListaDeJogadores().get(this.servidorConexao.posicaoJogadorNaLi
sta(this.jogador)).getListaDeAdversarios().size() == 3) {
                        this.mensagem = jogador.getCodigo() + ";" +
                        9 + ";" +
                        this.jogador.getPosicaoOriginalX() + ";" +
                        this.jogador.getPosicaoOriginalY() + ";" +
                        this.jogador.getListaDeAdversarios().get(0).getCodigo() + ";" +
                        this.jogador.getListaDeAdversarios().get(0).getNome() + ";" +
                        this.jogador.getListaDeAdversarios().get(1).getCodigo() + ";" +
                        this.jogador.getListaDeAdversarios().get(1).getNome() + ";" +
                        this.jogador.getListaDeAdversarios().get(2).getCodigo() + ";" +
                        this.jogador.getListaDeAdversarios().get(2).getNome() + ";";
                        this.servidorConexao.atualizarSituacaoDoJogador(this.jogador);
                        this.jogador.setAtualizar(false);
                   }else {
                       this.jogadorLivre(this.jogador, 2);
                   }
}
```

Quadro 20 – Formação de adversários no servidor de comunicação

No quadro 21 é apresentada a criação dos personagens do jogo que representam os navios (avatares) dos jogadores. A criação destes personagens, esta diretamente relacionada à preferência de jogo selecionada pelo jogador. Caso o jogador tenha selecionado a preferência "individual", são criadas duas instâncias de personagens. Uma representando o navio do jogador e a outra, do adversário.

Na classe `ClienteCanvas`, a cada nova iteração do método `run()` da *Thread*, é feita uma chamada ao método `alteracaoJogo`. Neste método são feitas todas as validações sobre os dados atualizados do personagem do jogador e de seus adversários fazendo com que seja atualizada a posição do avatar na tela e sua lista de tiros incluindo explosões caso algum destes tiros acerte o avatar do inimigo terminando o jogo caso algum jogador tenha perdido três vidas.

```
public void criarPersonagens() {
      if (this.clienteGUI.getPreferenciaJogador() == 0) {
         this.personagem = new Personagem(this.navioAmigo[0], 105, 225,
this.navioAmigo[0].getWidth(), this.navioAmigo[0].getHeight(), 1, 100);
         this.listaDeNavios.addElement(this.personagem);

         this.personagem = new Personagem(this.navioInimigo[0], 105, 0,
this.navioInimigo[0].getWidth(), this.navioInimigo[0].getHeight(), 1, 100);
         this.listaDeNavios.addElement(this.personagem);
      }else {
         this.personagem = new Personagem(this.navioAmigo[0], 10, 225,
this.navioAmigo[0].getWidth(), this.navioAmigo[0].getHeight(), 1, 100);
         this.listaDeNavios.addElement(this.personagem);
         this.personagem = new Personagem(this.navioAmigo[1], 200, 225,
this.navioAmigo[1].getWidth(), this.navioAmigo[1].getHeight(), 1, 100);
         this.listaDeNavios.addElement(this.personagem);
         this.personagem = new Personagem(this.navioInimigo[0], 10, 0,
this.navioInimigo[0].getWidth(), this.navioInimigo[0].getHeight(), 1, 100);
         this.listaDeNavios.addElement(this.personagem);
         this.personagem = new Personagem(this.navioInimigo[1], 200, 0,
this.navioInimigo[1].getWidth(), this.navioInimigo[1].getHeight(), 1, 100);
         this.listaDeNavios.addElement(this.personagem);
      }
   }
```

Quadro 21 – Criação dos personagens na classe `ClienteCanvas`

3.3.2 Operacionalidade da implementação

Para que seja possível jogar uma partida do jogo Batalha Naval, é necessário que o servidor esteja ativo e que tenha uma única porta para receber as conexões dos usuários.

Na figura 16 é apresentado o servidor de forma que os jogadores que conectam-se, são apresentados de forma seqüencial na lista.

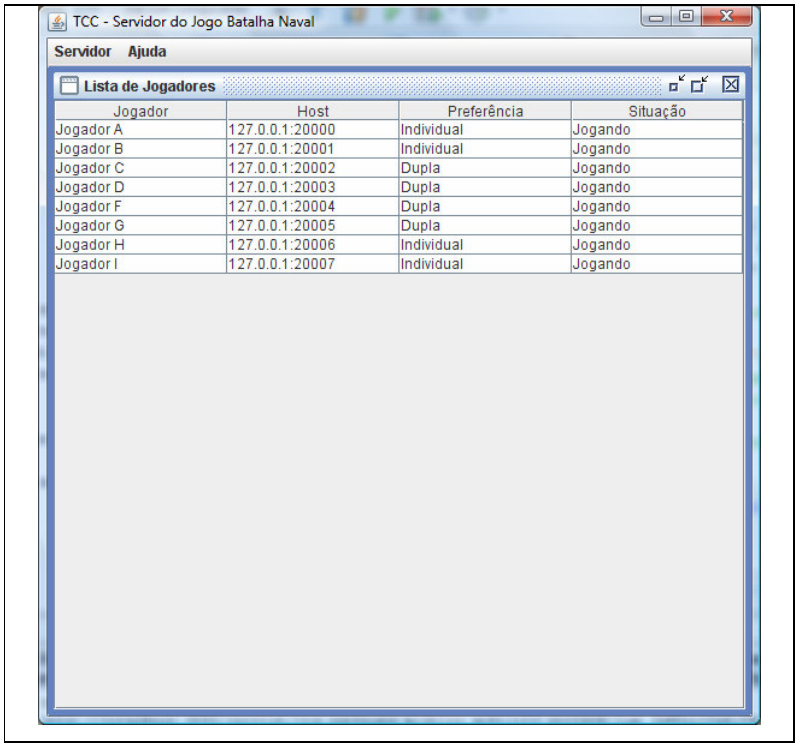


Figura 16 – Servidor do jogo

Quando o jogador inicia o cliente do jogo no celular, é apresentada uma imagem de apresentação do jogo (figura 16). Nesta tela, o jogador deverá escolher prosseguir (selecionando *Ok*) ou sair (selecionando *Sair*).

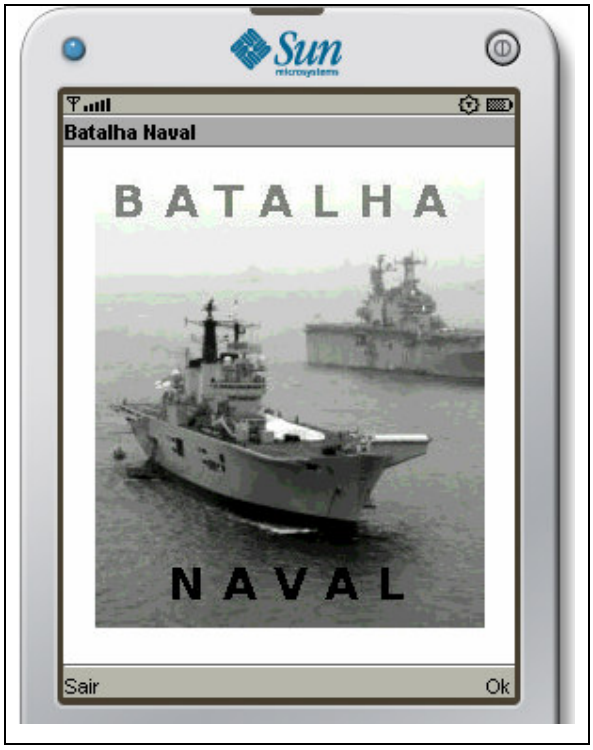


Figura 17 – Tela de entrada do cliente do jogo

Caso o jogador selecione *Ok*, é apresenta a tela de entrada de dados (figura 17). Nesta tela, o jogador deverá informar seu nome (para que seja informado em sua tela e na tela do adversário) e a preferência do jogo. A informação da preferência do jogo é um fator muito importante para que o servidor, com base nesta informação, selecione um adversário para o jogador. O jogador poderá selecionar a preferência "individual", caso este queira jogar contra outro jogador ou selecionar a preferência "duplas", caso este deseje jogar em parceria com outro jogador contra outra dupla de jogadores.

O jogador poderá alterar o *host* do servidor bem como a porta de conexão. Para efeito de testes, foi utilizado o endereço 127.0.0.1 e a porta 50000.

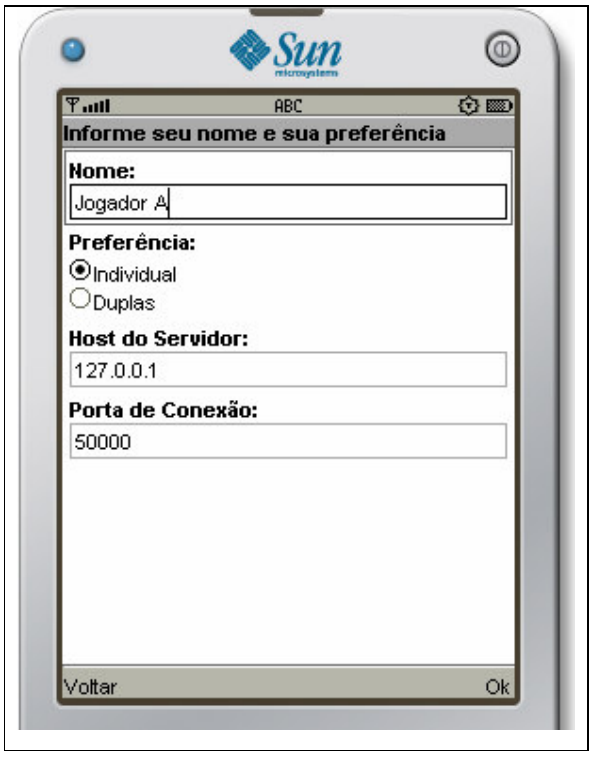


Figura 18 – Tela de entrada de dados

Quando o jogador selecionar a opção *Ok*, o jogador ficará aguardando um adversário (caso este tenha selecionado a opção "individual" senão, ficará aguardando outros três jogadores que também queiram jogar em duplas). Na figura 18 é mostrado a tela que é apresentada ao jogador para que este aguarde um adversário.

Figura 19 – Jogador aguarda adversário

Na figura 19 é apresentada a disposição inicial dos personagens do jogo com os jogadores jogando individualmente.

O personagem do jogador é sempre apresentado na parte inferior da tela de forma centralizada. O personagem deste jogador é apresentado na parte superior da tela do adversário. Para melhor visualizar este conceito, observe as disposições dos personagens A e B. Para o jogador, o personagem aparece na parte inferior da tela enquanto que para seu adversário, aparece na parte superior da tela.

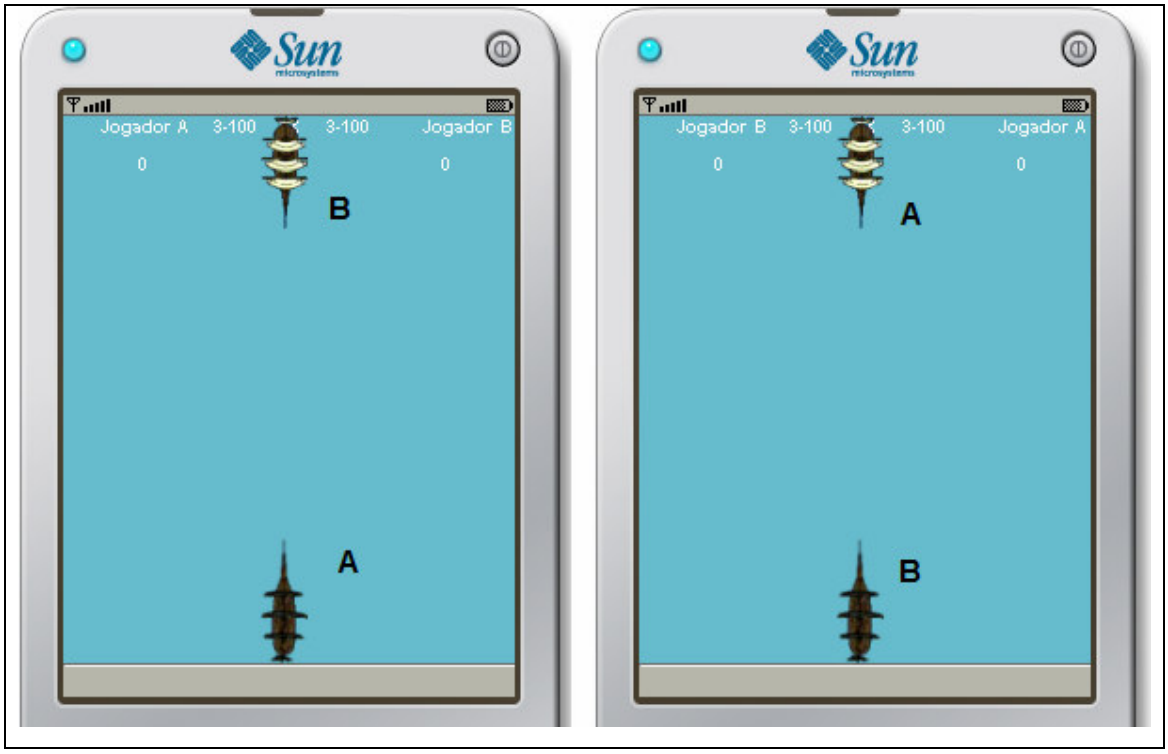


Figura 20 – Disposição inicial dos personagens para um jogo individual

O personagem do jogo pode ser movimentado através das teclas 2, 4, 6 e 8 (para cima, para a esquerda, para a direita, para baixo). Para atirar, o jogador poderá utilizar a tecla 0.

Um personagem é composto de três vidas sendo que em cada vida ele poderá ser atingido cinco vezes, para então, perder uma vida.

Quando o personagem é atingido, é apresentada uma animação pequena de explosão e descontado 20 pontos de sua energia (figura 20). Quando a energia chegar a 0, é apresentada uma animação maior e o personagem então, perde uma vida.

Figura 21 – Movimentação dos personagens

Quando um personagem perder as três vidas, é apresentada uma mensagem indicando que o jogador perdeu e sua respectiva pontuação e outra mensagem simultaneamente para o adversário indicando que este, ganhou o jogo e a pontuação alcançada (figura 17).

Figura 22 – Fim de jogo individual

Quando o jogo for em duplas, a disposição dos personagens muda, sendo que aos olhos do o jogador e de seu parceiro, seus personagens sempre estarão na parte inferior da tela ao passo que seus adversários aparecerão na parte superior da tela.

Na figura 22 é apresentado o jogador A e seu parceiro, o jogador B (na parte inferior da tela) contra o jogador C e seu parceiro o jogador D (na parte superior da tela).

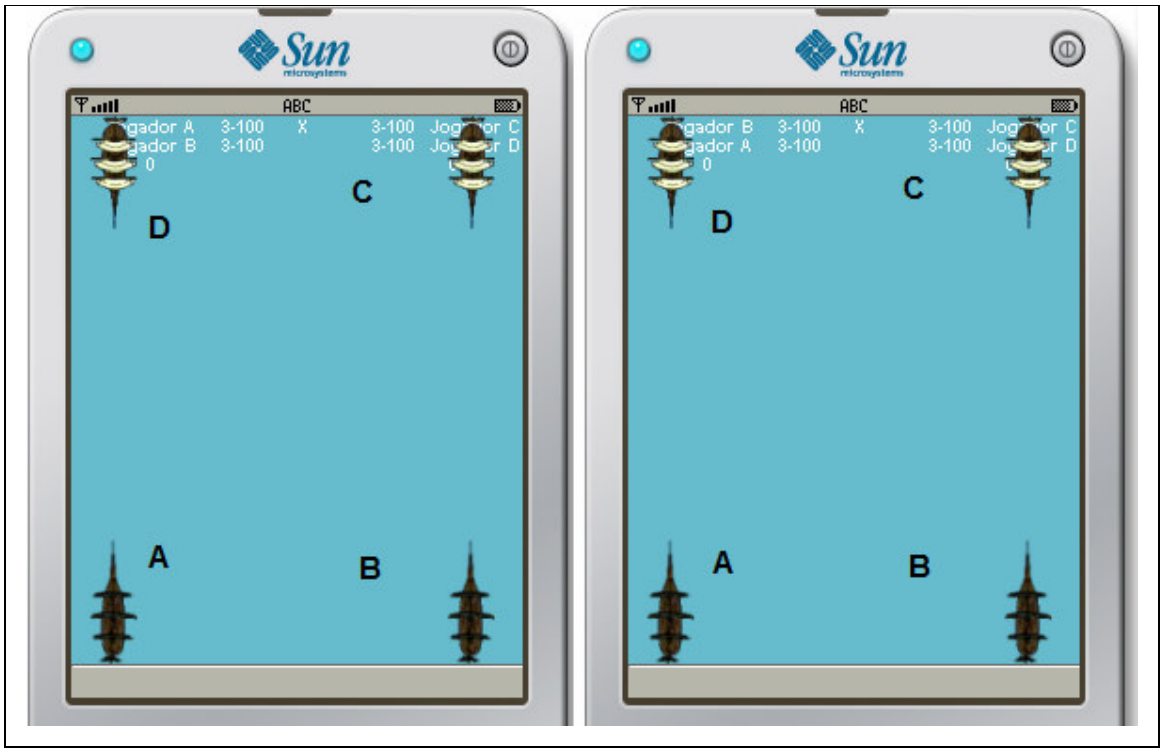


Figura 23 – Jogo em duplas (jogador A e jogador B contra os jogadores C e D)

No mesmo jogo, é apresentado na figura 23, o jogador C e seu parceiro, o jogador D (na parte inferior da tela) contra o jogador A e seu parceiro o jogador B (na parte superior da tela).

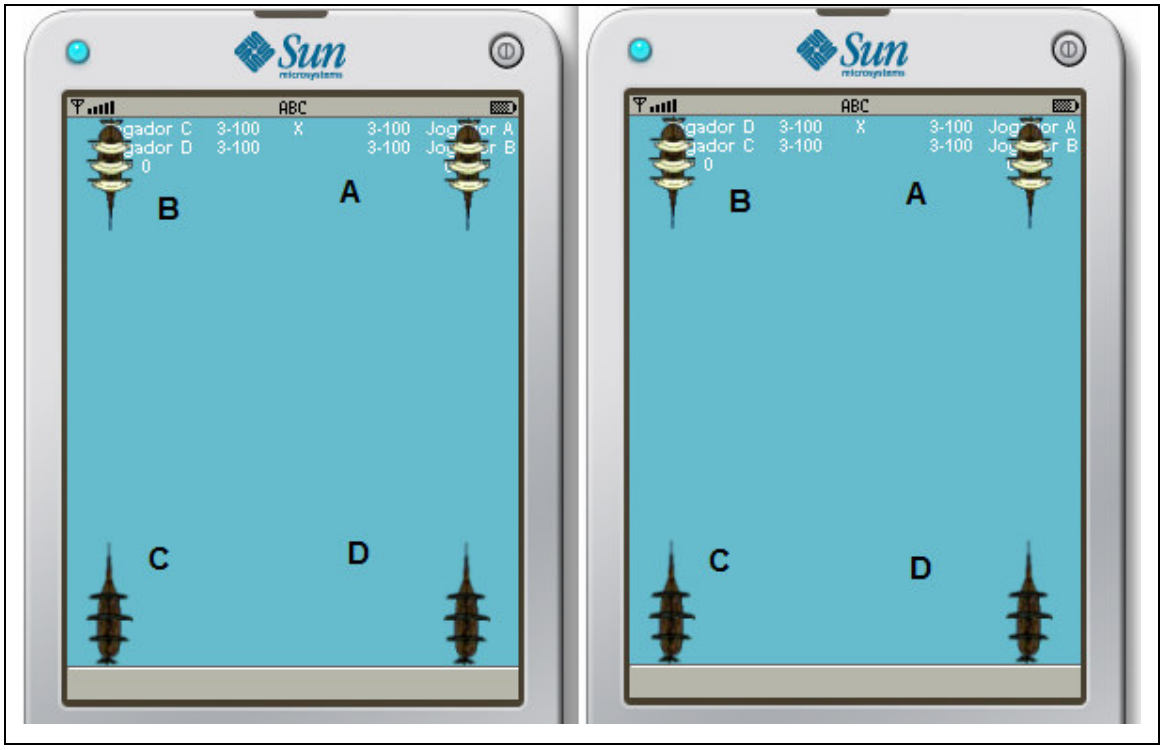


Figura 24 – Jogo em duplas (jogador C e jogador D contra os jogadores A e B)

Nas figuras 24 e 25 são apresentadas as primeiras movimentações dos personagens. Quanto às disposições dos nomes dos jogadores, pontos e energia, procurou-se apresentar sempre o jogador e seu parceiro no canto superior esquerdo enquanto que os nomes dos adversários, no canto superior direito.

O nome do jogador atual é apresentado em cima enquanto que o de seu parceiro, em baixo do seu nome.

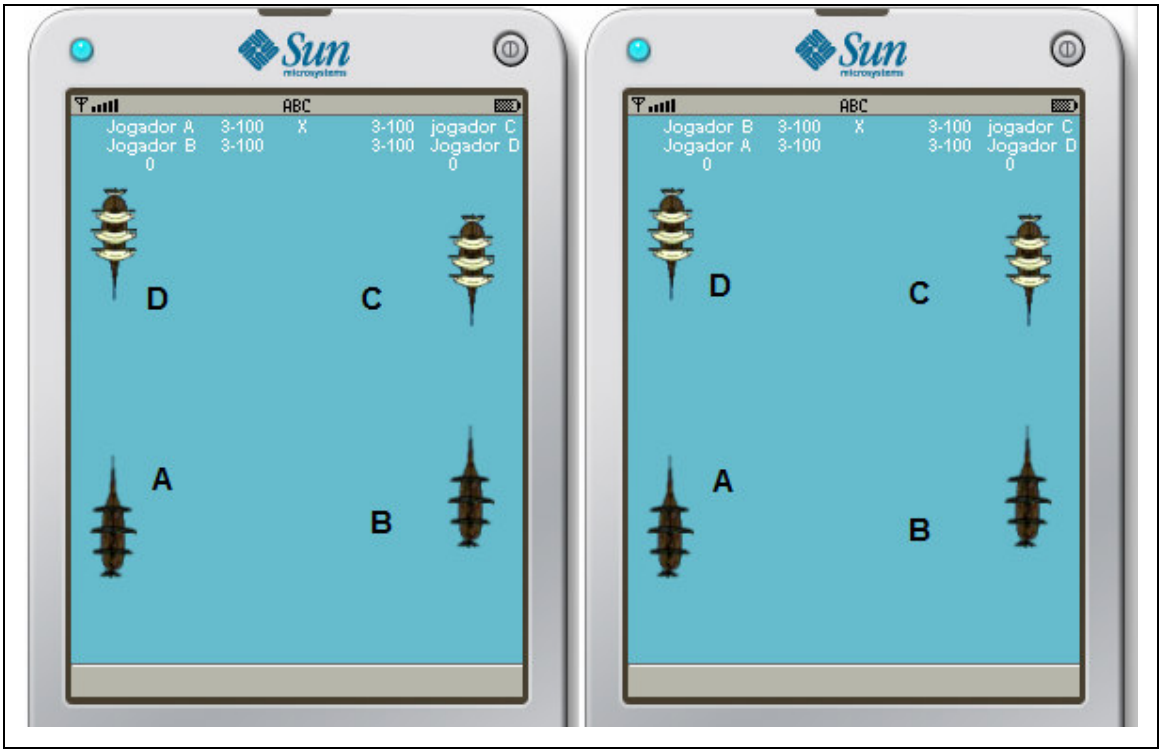


Figura 25 – Jogo em duplas – disposições (jogador A e jogador B contra os jogadores C e D)

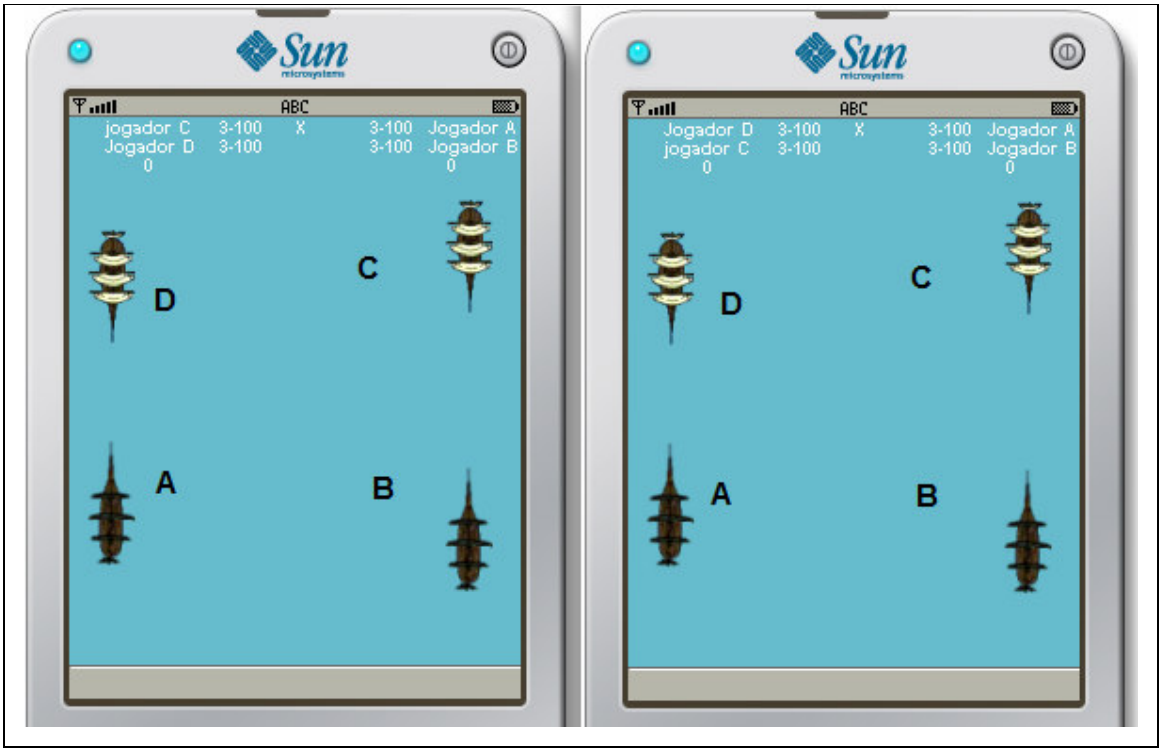


Figura 26 – Jogo em duplas – disposições (jogador C e jogador D contra os jogadores A e B)

3.4 RESULTADOS E DISCUSSÃO

Inicialmente a idéia de desenvolver um jogo *multiplayer* para celular era de fazê-lo rodar também em celulares. Porém, ao estudar o problema proposto, viu-se da necessidade do desenvolvimento de um servidor para armazenar e controlar as informações pertinentes aos jogos e aos jogadores que estavam conectados.

Um pouco mais a frente, viu-se também da necessidade de desenvolver um protocolo que permitisse fazer com que o cliente e o servidor do jogo pudessem se comunicar. Neste ponto, o desenvolvimento do jogo tornou-se um processo não tão trivial. Problemas de entendimento da plataforma J2ME fizeram com que o desenvolvimento da comunicação entre as partes demorasse um pouco mais do que o previsto atrasando assim, o desenvolvimento de outras partes importantes do projeto.

Os resultados almejados através do desenvolvimento deste trabalho foram alcançados. Apesar de não terem sido feitos os testes com telefones celulares, os testes foram feitos em simuladores com até oito máquinas ligadas em rede sendo que o servidor estava em uma máquina própria.

Os testes iniciais foram emulados em uma única máquina fazendo com que o jogo tivesse uma latência muito alta devido à falta de recursos da máquina em questão. Posteriormente, foram conectados, em rede, quatro jogadores dos quais formaram duplas de jogadores e estas enfrentaram-se. Enquanto isto conectavam-se mais quatro máquinas sendo que agora, formariam-se mais dois jogos individuais com dois jogadores cada. A latência, que era uma preocupação nestes casos, não foi percebida.

No quadro 22 é feita uma comparação em termos de estrutura dos trabalhos correlatos ao trabalho desenvolvido. Para tanto, é possível perceber que somente o jogo *Galaxy Navigator* transpõe a estrutura comumente proposta neste trabalho por se tratar de um jogo massivamente *multiplayer* ao ponto que o que fora proposto neste trabalho, foi o de desenvolver um jogo somente *multiplayer* utilizando a arquitetura cliente/servidor sem distribuição.

	Multiplayer	MMO	Plataforma	Arquitetura
Batalha Naval	Sim	Servidor: Sim Jogo: Não	Celular	Cliente/Servidor
Cellmons	Sim	Não	Celular	Centralizado
Galaxy Navigator	Não	Sim	Celular	Distribuído
Tetris	Não	Não	Celular	Centralizado

Quadro 22 – Comparação entre os trabalhos correlatos

4 CONCLUSÕES

Como a proposta deste trabalho foi de desenvolver um jogo cliente/servidor para celular, teve-se que implementar todo o software servidor para controlar os clientes do jogo e tornar possível a conclusão deste trabalho.

Através do desenvolvimento deste, pode-se concluir que os jogos *multiplayer* para dispositivos móveis como o celular, que apesar de ainda possuírem baixos recursos de processamento e memória, são altamente viáveis. Isto nos remete a pensar que com o crescente avanço nos recursos dos celulares, e com a disseminação destes dispositivos na sociedade, é possível tornar o desenvolvimento de jogos *multiplayer* um grande nicho de mercado apesar de hoje ainda serem tímidos os desenvolvimentos de jogos nessa modalidade.

Quanto à tecnologia J2ME empregada no desenvolvimento deste trabalho, pode-se afirmar que com a chegada do perfil MIDP 2.0 agregado a configuração CLDC, de uma forma geral, a viabilidade de jogos mais complexos e interativos. Os jogos *multiplayer* enquadram-se neste contexto devido ao suporte dado pelo MIDP para conexões *socket* e *http*.

Pode-se afirmar que os objetivos deste trabalho foram alcançados. Dois jogadores conseguem jogar de forma individual através de simuladores. Quatro jogadores também conseguem jogar através de simuladores sendo que para esta "preferência" de jogo é tratada no servidor como um jogo em duplas.

Caso a aplicação servidora estivesse em um celular, a conexão por *http* e o protocolo de conexão por *socket* teria a mesma funcionalidade que a apresentada neste.

Uma preocupação no desenvolvimento deste trabalho, foi com a latência. Durante o desenvolvimento deste trabalho, foi possível identificar com clareza a latência na atualização dos dados de um jogador em seu adversário (isto para uma única máquina). Nos testes com uma rede de computadores, o jogo foi simulado em oito máquinas fazendo um jogo em duplas (quatro jogadores) e outros dois jogos contendo dois jogadores cada. Nesta simulação, não houve problemas com a latência. Verificou-se com isto, que apesar de não terem sido feitos testes com um telefone celular, somente com simuladores, os jogos *multiplayer* para este fim, podem vir a tornar-se em um futuro não tão distante, uma realidade em nosso dia-a-dia.

4.1 EXTENSÕES

Sugere-se para trabalhos futuros, são dispostas as seguintes sugestões:

a) o desenvolvimento de uma base de dados para registrar os jogadores implementando um controle de *login* para identificar os jogadores logados;
b) desenvolver um *ranking* para deixar disponível para consulta aos jogadores, aqueles que tem o maior número de pontos e também uma consulta individual no *ranking*.
c) a inteligência artificial do jogador-computador pode ser melhorada;
d) pode ser implementado uma opção para que o jogador possa jogar *sigle-player* sem a necessidade da conexão com o servidor do jogo;
e) o servidor pode ser implementado em um celular.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDRIOLLI, Gabriel F. J2ME MIDP: os primeiros passos para o desenvolvimento de aplicativos para celulares. **J2ME**. [S.l.], 2008. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=120&hl=>. Acesso em: 11 nov. 2008.

CALDEIRA, Hyalen M. Java 2 Micro Edition: do Pager ao PDA, com J2ME. **Java Magazine**, Grajaú, n. 1, p. 32-35, 2002.

CARMO, Sandro D. do. **Curso de introdução ao Java**. [S. l.], 2008. Disponível em: <http://www.inpe.br/twiki/bin/viewfile/Competencia/SunInpe?rev=1;filename=Aula-01-Fabio.pdf>. Acesso em: 12 nov. 2008.

CEREJA, Joni B. **Arquitetura de jogos de celular online massivamente multiplayer**. 2008. 48 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

CISNEIROS, Ronaldo. **Especificação comportamental de um subconjunto da plataforma J2ME**, v.154, p. 1-154, Recife, 2005. Disponível em: <http://www.dsc.upe.br/tcc/20052/RonaldoCisneiros.pdf>. Acesso em: 12 nov. 2008.

COMPUTERWORLD. **Mercado mundial de jogos**. São Paulo. 2008. Disponível em: <http://idgnow.uol.com.br/telecom/2008/06/26/mercado-de-jogos-para-celular-fatura-us-4-5-bi-este-ano-diz-gartner/>. Acesso em: 21 nov. 2008.

DEBONE, José E. Z. **Introdução aos diagramas UML**. [S. l.], 2004. Disponível em: <http://www.voxxel.com.br/pages/introdiauml.html>. Acesso em: 16 nov. 2008.

FERRAZ, Felipe et al. **Fundamentos do J2ME**. Santa Catarina, 2006. Disponível em: <http://www.dei.unicap.br/~almir/seminarios/2006.1/ns06/J2ME/>. Acesso em: 14 nov. 2008.

FERREIRA, Geraldo A. **Aplicações MIDP em aparelhos móveis celulares e monitoramento remoto de bio-sinais:** considerações e desenvolvimento de uma solução. 2005. 98 f. Dissertação de Mestrado em Ciências da Computação – Universidade de Minas Gerais. Disponível em: <http://www.bibliotecadigital.ufmg.br/dspace/bitstream/1843/SLBS-6GUNZ4/1/geraldoantonioferreira.pdf>. Acesso em: 17 nov. 2008.

GOMES, Alexandre. **Tutorial J2ME**. [S.l], 2008. Disponível em: <http://www.mundooo.com.br/php/mooartigos.php?pa=showpage&pid=9>. Acesso em: 15 abr. 2008.

JOHNSON, Thienne M. et al. Integrando a tecnologia J2ME no âmbito acadêmico. **J2ME.** Belém. 2006. Disponível em: <http://www.unibratec.com.br/jornadacientifica/diretorio/NovoINT.pdf>. Acesso em: 27 out. 2008.

JUNIOR, Eloi. Utilizando SVG com Java ME. **Web Mobili**, Rio de Janeiro, 2008, v. 16, n. 2, p. 6-14, mar. 2008.

MARTINS, Guilherme. **Jogos para telefones celulares:** estudo prático de desenvolvimento utilizando J2ME e MIDP 2.0. 2004, 81 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Universidade Federal de Santa Catarina, Florianópolis.

MEDEIROS, Ernani. **Desenvolvendo software com UML 2.0**. São Paulo: Makron Books, 2004

MENEZES, Andrea et al. Impacto da mobilidade no game design de 3MOGs. **Jogos móveis**. Recife. 2006. Disponível em: <http://www.cesar.org.br/files/file/2006-1.pdf>. Acesso em: 18 nov. 2008.

MUCHOW, John W. **Core J2ME:** tecnologia & MIDP. São Paulo: Pearson Education do Brasil, 2006.

PALERMO, Bruno et al. Impacto da mobilidade no game design de 3MOGs. **Jogos Móveis Massivamente Multiplayer**, [S.l], 2006. Disponível em: <http://www.cesar.org.br/files/file/2006-1.pdf>. Acesso em: 15 abr. 2008.

PEDROSA, Davi de V. **Jogos multiplataforma multiusuário**. 2006. 80 f. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) – Universidade Federal de Pernambuco, Recife. Disponível em: <http://cin.ufpe.br/~tg/2006-1/dvp.pdf>. Acesso em: 21 nov. 2008.

PEREIRA, Daniel A. **Jogos oblíquos com bluethooth**. 2006. 60 f. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) – Universidade Federal de Pernambuco, Recife. Disponível em: <http://cin.ufpe.br/~tg/2005-2/dap.pdf>. Acesso em: 19 nov. 2008.

PEREIRA, Fernando M. Q. et al. Chamada remota de métodos na plataforma J2ME/CLDC. **J2ME**, Minas Gerais, v.11, p. 1-11, 2004. Disponível em: <http://compilers.cs.ucla.edu/fernando/publications/journals/Inatel2004.pdf>. Acesso em: 13 nov. 2008.

SHMITT JUNIOR, A. J. **Protótipo de front end de controle de acesso usando J2ME**. 2004. 70 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

RABELO, Solon A.; HAHN, Rodrigo M.; BARBOSA, Jorge. BlueGame: um jogo móvel multi-jogador baseado em comunicação *bluetooth*. **Jogos Móveis,** [S.l], 2005. Disponível em: <http://inf.unisinos.br/~barbosa/textos/WJOGOS_2005.pdf>. Acesso em: 12 abr. 2008.

ROSETTO, Cristiano. **Desenvolvimento de jogos para dispositivos móveis utilizando MIDP: implementação do jogo Tetris**. 2005. 59 f. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SUN. **Java tecnology.** [S.l.], 2007. Disponível em: <http://java.sun.com/about>. Acesso em: 14 nov. 2008.

SYMBIANBR. 2008. **O mercado de jogos para celular cresce**. [S.l.], 2008. Disponível em: <http://www.symbianbr.com.br/modules/news/article.php?storyid=179>. Acesso em: 20 nov. 2008.

UNIVERSIA. **Mais emprego para criadores de jogos para celular**. [S. l.], 2005. Disponível em: <http://www.universia.com.br/html/noticia/noticia_clipping_cgeje.html>. Acesso em: 21 nov. 2008.

VALENTE, Marco T. O. et al. Chamada remota de métodos na plataforma J2ME/CLDC. **J2ME**, Minas Gerais, v.12, p. 1-12, 2003. Disponível em: <http://compilers.cs.ucla.edu/fernando/publications/papers/5oWCSF.pdf>. Acesso em: 09 nov. 2008.

WISNIEWSKI, D. et al. 2005 Mobile Games White Paper. In: GAME DEVELOPERS CONFERENCE, 2005. [S.l]. **Anais.** [S.l], 2005. Disponível em: <http://www.igda.org/online/>. Acesso em: 05 nov. 2008.

YAHOO GAMES. **Batalha naval**. [S.l], 2008. Disponível em: <http://br.play.yahoo.com/games/rules/fleet/rules.html?page=flt>. Acesso em: 10 jun. 2008.