André Ribeiro Barros

Estudo sobre a utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Ricardo Martins Abreu Silva

Lavras
Minas Gerais - Brasil
2005

André Ribeiro Barros

Estudo sobre a utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *17 de Janeiro de 2005*

Prof. Guilherme Bastos Alvarenga

Prof. André Luis Zambalde

Prof. Ricardo Martins Abreu Silva
(Orientador)

Lavras
Minas Gerais - Brasil

Agradecimentos

Agradeço a Deus pela vida. Meus pais e irmãos por acreditarem em mim. Aos amigos verdadeiros pelos momentos únicos. Ao Professor Ricardo pela orientação. E a todos aqueles que de alguma forma me ajudaram.

Dedico este trabalho a meus pais que sempre me colocaram no caminho do estudo, não importando a barreira, seja ela econômica, de saúde ou sentimental.

Resumo

Estudo sobre a utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário

O objetivo deste trabalho é apresentar o estudo sobre a utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário. Para este estudo foi desenvolvido um jogo de estratégia distribuído multi-usuário denominado Panzer utilizando a tecnologia J2ME e J2SE. Este estudo envolve várias áreas da ciência da computação como, física, computação gráfica, redes, engenharia de software, e sistemas móveis. É descrito neste trabalho a implementação do jogo, apresentando informações sobre sua modelagem, interface com o usuário, arquitetura de comunicação utilizada e controle do estado e lógica do jogo. Por fim será apresentada uma breve discussão sobre as dificuldades e facilidades encontradas no decorrer do trabalho, com propostas de melhorias capazes de torná-la comercialmente viável.

Palavras-chave: tecnologia wireless, aplicação distribuída, multi-usuário

Abstract.

Study on the use of the technology wireless in the development of multi-user distributed applications

The objective of this work is to present the study on the use of wireless technology in the development of multi-user distributed applications. For this study was developed a distributed multi-user strategy game called Panzer, using technology J2ME and J2SE. This study involves some areas of computer science as physic, graphical computation, nets, software engineer, and mobile systems. The game´s implementation was described in this work, presenting information on its modeling, user interface, used communication architecture and the state control and logical of the game. Finally it will be presented a brief quarrel about the difficulties and easinesses found in elapsing of the work, with improvements proposals of capable to become it commercially viable.

Key-words:technology wireless, distributed application, multi-user

Sumário

Lista de Figuras

Capítulo 1

Introdução

O objetivo deste trabalho é estudar a utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário. Este estudo tem como foco específico a área de jogos.

A área de jogos tem crescido e mudado acentuadamente desde seu surgimento com o primeiro exemplar em 1962 - denominado *Spacewar*. *Hardware* e *software* têm avançado desde então, e os jogos têm beneficiado e contribuído para esta evolução na computação. Novas tecnologias têm revolucionado muitos aspectos de jogos, e sua popularidade e lucro tem crescido em paralelo com sua crescente sofisticação.

Figura 1.1: Venda de jogos eletrônicos nos U.S.A. (Dados: Electronic Industries Association)

Segundo Arnold [ARNOLD, 2002], o rendimento on-line de jogos nos Estados Unidos crescerá de $138 milhões em 2002 para $2,3 bilhões em 2005. Embora a quantidade média gasta em aparelhos domésticos on-line tende a cair de $167 para $157 em 2005, o número de usuários deverá aumentar substancialmente. A indústria de jogos tem se igualado a de música e cinema em vendas.

As desvantagens dos dispositivos móveis atuais em aplicações de entretenimento, em especial em jogos, estão no seu poder de processamento e no *design* [1]. O poder de processamento destes dispositivos é na grande maioria reduzida devido a seu limitado tamanho e propósito. A jogabilidade pode ser prejudicada em jogos mais vibrantes, devido ao desconforto causado pelo *design* do aparelho móvel. O propósito destes dispositivos é a comunicação, assim os dispositivos móveis mais populares (os celulares) são na grande maioria projetados para digitar números para uma ligação, e no máximo redigir uma pequena mensagem. Mas este cenário poderá mudar. Com o aumento da criação de jogos multi-usuário para dispositivos móveis e havendo uma grande aceitação por parte dos usuários, fabricantes de *hardware* de dispositivos móveis podem se interessar e melhorar o processamento e *design* de seus produtos em resposta a esta demanda.

Os dispositivos móveis atuais mais populares incluem celulares, PDAs [2], pagers e laptops. Hoje, tem-se celulares que além de sua função básica (ligações), enviam SMS [3] e e-mail, tocam músicas no formato mp3, tiram fotos, incluem agenda, calculadora, *chat*, entre outros. Havendo assim uma tendência à união das funções de todos os tipos dispositivos móveis em um único. Isto será uma realidade num futuro próximo, pois todo dia avanços tecnológicos fornecem novos *softwares*, *hardwares* cada vez menores e mais potentes, e protocolos de comunicação.

[1] Diz respeito a forma e tamanho de um aparelho, a disposição dos botões no mesmo, e o tamanho de sua tela.

[2] *Personal Digital Assistant*, ou assistente digital particular

[3] *Short Messaging Service*, ou serviço de mensagem curta

Capítulo 2

Referêncial Teórico

2.1 Arquitetura de Sistemas Distribuídos Cliente/Servidor

Ainda que a arquitetura cliente/servidor seja a líder na indústria, não há consensos em o que o termo atualmente significa [RDJ, 1996] [1]. Como o nome implica, cliente e servidor são entidades lógicas separadas que trabalham juntas sobre uma rede pra realizar uma tarefa. A seguir serão mostradas algumas características que distingue a arquitetura cliente/servidor de outras arquiteturas de sistemas distribuídos:

- **Serviço**: Primeiramente, cliente/servidor é uma relação entre processos, os quais podem estar em máquinas separadas ou em uma única. O processo servidor é um provedor de serviços. O cliente é um consumidor de serviços. Na essência, cliente/servidor fornece uma separação de funções baseado na idéia de serviços.

- **Compartilhamento de serviços**: Um servidor pode servir muitos clientes ao mesmo tempo e regula os acessos para compartilhar recursos.

- **Protocolos assimétricos**: Há uma relação de muitos para um entre clientes e servidor. Clientes sempre iniciam o diálogo por requisitar um serviço. Servidores ficam passivamente esperando requisições dos clientes.

- **Transparência de localização**: O servidor é um processo que pode residir na mesma máquina que está o cliente ou em uma diferente máquina através da rede. Um *software* cliente/servidor geralmente esconde a localização do servidor do cliente por redirecionar o serviço chamado quando necessário. Um programa pode ser um cliente, um servidor, ou ambos.

[1] Está seção é baseada no capítulo dois do livro *The Essential Cliente/Server Survival Guide.*

- **Mistura e combinação**: O *software* cliente/servidor ideal é independente de *hardware* ou plataforma de sistema operacional. Devemos ser capaz de misturar e combinar plataformas de cliente e servidor.

- **Troca baseada em mensagem**: Clientes e servidores são sistemas livremente acoplados que interagem através de um mecanismo de passagem de mensagem. A mensagem é o mecanismo de entrega para o serviço de requisição e resposta.

- **Encapsulamento de serviços**: O servidor é um "especialista". Uma mensagem diz ao servidor qual serviço é requisitado; o servidor deve então processar tal requisição e entregar o trabalho feito num certo formato. Servidores podem ser atualizados sem afetar o cliente, dado que a interface pública de mensagens não muda.

- **Escalabilidade**: Sistemas cliente/servidor podem ser horizontalmente ou verticalmente escaláveis. Escalar horizontalmente significa adicionar ou remover estações de trabalho clientes com somente um leve impacto de desempenho. Escalar verticalmente significa migrar para uma máquina servidora ampla e rápida ou multi-servidora.

- **Integridade**: O código e dados do servidor são mantidos centralizados, o que resulta em uma manutenção barata e a garantia de integridade dos dados compartilhados. Ao mesmo tempo, o cliente fica personalizado e independente.

As características de cliente/servidor descritas aqui permitem que a informação seja facilmente distribuída através da rede. Estas características também fornecem uma estrutura para o projeto de aplicações baseadas em redes.

2.1.1 Modelos Cliente-Servidor

Muitos sistemas com muitas diferentes arquiteturas têm sido chamados "cliente/servidor". Vendedores de sistemas freqüentemente usam cliente/servidor como se o termo pudesse somente ser aplicado para seus específicos pacotes. Por exemplo, vendedores de servidor de arquivo juram que eles foram os primeiros a inventar o termo. Adicionaremos à lista de tecnologias cliente/servidor faladas: objetos distribuídos, monitores TP, *groupware* e a Internet. Assim, qual destas tecnologias é a correta? Qual é realmente cliente/servidor? A resposta para ambas as perguntas é todas as acima citadas.

A idéia de dividir uma aplicação numa arquitetura cliente/servidor tem sido usada durante os últimos vinte anos para criar formas de solução de *software* sobre

uma LAN [2]. Cada uma destas soluções é distinguida pela natureza dos serviços oferecida aos clientes, como é mostrado nas seguintes seções.

Servidores de Arquivo

Com um servidor de arquivo, o cliente (tipicamente um PC) passa requisições de registros de arquivo sobre uma rede para o servidor de arquivo. Isto é uma forma muito primitiva de serviços de dados que necessita de muitas trocas de mensagens sobre a rede para encontrar o requisitado dado. Servidores de arquivo são úteis para compartilhar arquivos através da rede. Eles são indispensáveis por criar repositórios compartilhados de documentos, imagens, desenhos de engenharia e outros volumosos objetos de dados.

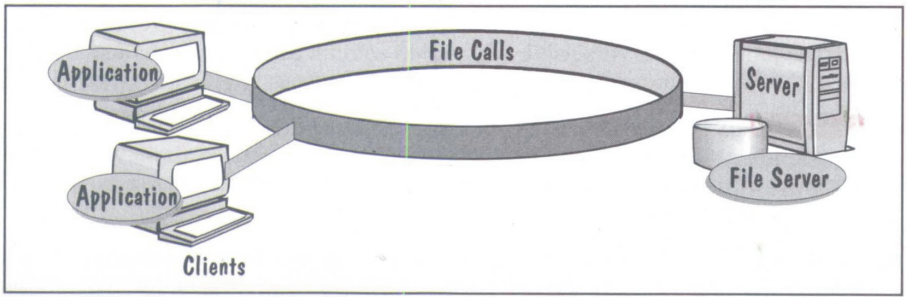

Figura 2.1: Cliente/servidor com servidor de arquivo

Servidores de Banco de Dados

Com um servidor de banco de dados, o cliente passa requisições SQL [3] como mensagens para o servidor de banco de dados (figura 2.2). Os resultados de cada comando SQL são retornados sobre a rede. O servidor usa seu próprio poder de processamento para encontrar os dados requisitados em vez de passar todos os registros de volta para o cliente e então deixa-lo encontrar por si próprio os dados, como foi o caso do servidor de arquivos. O resultado é um muito mais eficiente uso do distribuído poder de processamento. Com este método, o código do servidor é empacotado pelo vendedor. Mas freqüentemente é necessário escrever código para a aplicação cliente (ou pode-se comprar pacotes clientes como Quest ou Paradox). Servidores de banco de dados fornecem a base para sistemas de suporte a decisão que requerem consultas "*ad hoc*"e relatórios flexíveis.

[2] *Local Area Network*, ou área local de rede
[3] *Structured Query Language*, ou linguagem de consulta estruturada

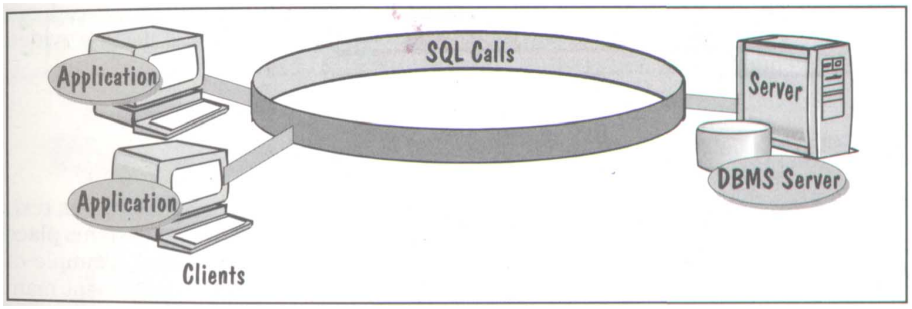

Figura 2.2: Cliente/servidor com banco de dados

Servidores de Transação

Com um servidor de transação, o cliente invoca métodos remotos que residem no servidor com um motor de banco de dados SQL (figura 2.2). Estes métodos remotos no servidor executam um grupo de instruções SQL. A troca na rede consiste de uma simples mensagem de requisição/resposta (oposto ao método do servidor de banco de dados onde temos uma mensagem de requisição/resposta para cada instrução SQL na transação). A instrução SQL é toda bem sucedida ou falha toda a unidade (conjunto de instruções SQL). Estas instruções SQL agrupadas são chamadas "*transactions*"(transações).

Com um servidor de transações criamos uma aplicação cliente/servidor por escrever o código em ambos componentes cliente e servidor. O componente cliente geralmente inclui uma GUI [4]. O componente servidor geralmente consiste da "*transaction*"(transação) SQL contra um banco de dados. Estas aplicações são chamadas OLTP [5]. Elas tendem a ser para aplicações de missão crítica que requerem um tempo de resposta de um a três segundos. Aplicações OLTP também requerem um controle firme sobre a segurança e a integridade dos bancos de dados. Duas formas mais conhecidas de OLTP são *TP Lite*, baseada em métodos de armazenamento fornecidos por vendedores de banco de dados; e *TP Heavy*, baseada em monitores TP fornecidos por vendedores de OLTP.

Servidores *Groupware*

Groupware é voltado para o gerenciamento de informação semi-estruturada tal como texto, imagem, e-mail, quadro de avisos, e o fluxo de trabalho. Este sistema cliente/servidor coloca pessoas em contato direto com outras pessoas. *Lotus Notes* é o exemplo líder de tal sistema, embora um número de outras aplicações - incluindo gerência de documentos, imagem, fluxo de trabalho, etc - são voltadas algumas para as mesmas necessidades. *Software groupware* especializado pode

[4] *Graphical User Interface*, ou interface gráfica com o usuário
[5] *Online Transaction Processing*, ou processo de transação on-line

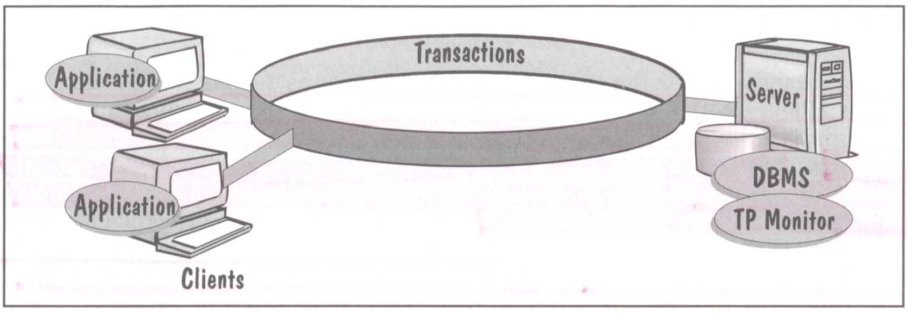

Figura 2.3: Cliente/servidor com servidor de transação

ser embutido no topo dos pacotes de APIs [6] cliente/servidor dos vendedores. Em muitos casos, aplicações são criadas usando uma linguagem de *script* e interface baseada em formulário fornecida pelo vendedor. O *Middleware* [7] de comunicação entre o cliente e o servidor é específico do vendedor (figura 2.4). Eventualmente, a Internet tornará o *Middleware* de escolha para o *Groupware*.

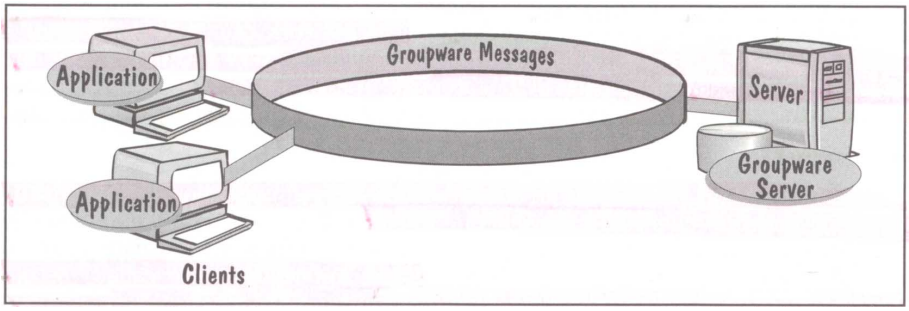

Figura 2.4: Cliente/servidor com servidor *groupware*

Servidores de Objeto

Com um servidor de objeto, a aplicação cliente/servidor é escrita como um conjunto de objetos comunicantes (figura 2.5). Objetos clientes comunicam com objetos servidores usando um ORB [8]. O cliente invoca um método em um objeto remoto. O ORB localiza uma instância daquela classe de objeto servidor, invocando o método requisitado, e retornando o resultado para o objeto cliente. Objetos servidores devem fornecer suporte para concorrência e compartilhamento. Depois de anos de incubação, alguns comerciais ORBs práticos, que deixaram de ser protóti-

[6] *Aplications Programming Interface*, ou interface de programas aplicativos
[7] uma ferramenta ou meio que faz o intermédio de um cliente e um servidor
[8] *Object Request Broker*, ou corretor de requisição de objeto

pos, estão agora em produção. Exemplos de ORBs comerciais que cumpriram com o padrão CORBA do *Object Management Group* incluem *ObjectBroker* da Digital, *SOM 3.0* da IBM, *NEO* da Sun, *ORB Plus* da HP, *PowerBroker* da Expersoft, *Orbix* da Iona, e *BlackWindow* da PostModern.

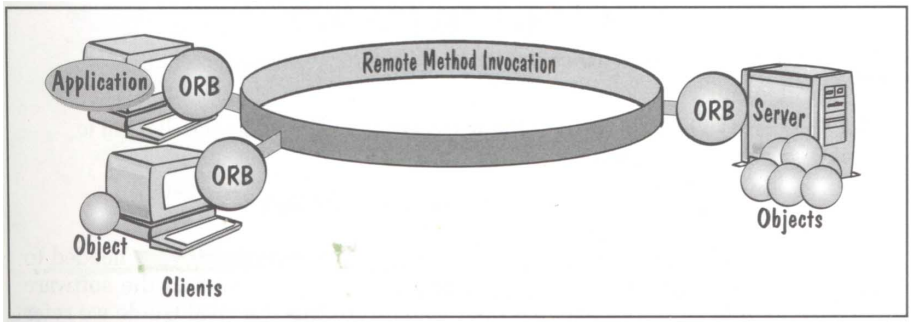

Figura 2.5: Cliente/servidor com servidor de objeto

Servidores *Web*

A *Word Wide Web* [9] é a primeira verdadeira intergaláctica aplicação cliente/servidor. Este novo modelo de cliente/servidor consiste de um cliente leve, portável, "universal"que conversa com servidores muito pesados. Em seu modo mais simples, um servidor *Web* retorna documentos quando uma cliente pergunta por seu nome (figura 2.6). O cliente e o servidor comunicam usando um protocolo chamado HTTP. Este protocolo define um simples conjunto de comandos; parâmetros são passados como *strings* [10], sem fornecimento dos tipos de dados. A *Web* tem sido estendida para fornecer mais formas interativas de computação cliente/servidor. Em adição, a *Web* e objetos distribuídos têm sido iniciados juntos. *Java* é a primeira manifestação deste novo objeto *Web*.

2.2 Tecnologias de Desenvolvimento de Sistemas Móveis

Esta seção trata das mais recentes linguagens para desenvolvimento de aplicações em sistemas móveis, tais como *J2ME*, *BREW*, *.NET*, *WAP/WML*, e *i-mode*. Será aqui apresentada uma breve introdução, dando uma geral noção das características de cada uma. Na seção **??** será mostrado em mais detalhes a linguagem J2ME, a qual foi escolhida para o desenvolvimento deste trabalho.

[9] Rede de documentos em formato HTML que estão interligados e espalhados em servidores do mundo inteiro

[10] Seqüência ou série de caracteres que são processados como uma unidade de informação

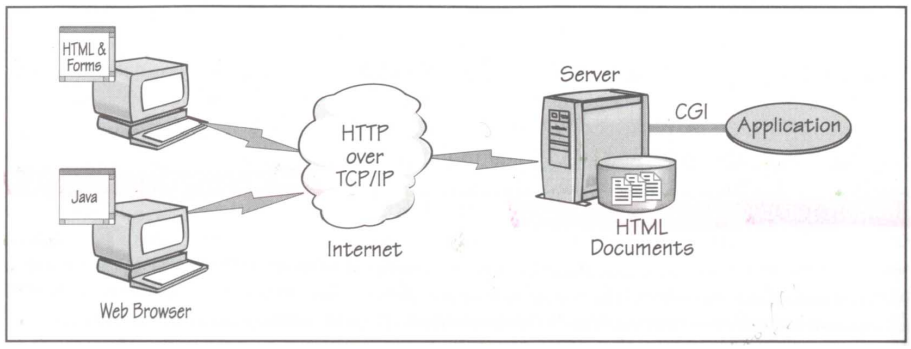

Figura 2.6: Cliente/servidor com servidor *web*

2.2.1 *J2ME*

J2ME é dividido em *configurations*, *profiles* e APIs opcionais [JAMES, 2002] [11]. Para entendermos melhor, pensemos da seguinte maneira, *Profiles* são mais específicos que *configurations*, e fazendo analogia a um velho exemplo temos uma abstração sobre o que é um carro e como ele é fabricado (*configuration*) e como um Ford é fabricado (*profile*), mais tecnicamente falando *profile* é baseado em *configuration* e ainda acima dos *profiles* estão as APIs que na nossa analogia seria um modelo específico da Ford.

Existem dois "*configurations*", um *configuration* é o CLDC [12], que rege as configurações para aparelhos bem pequenos como celulares ou PDAs, o qual fica acima das diretrizes J2ME juntamente com CDC [13] o que rege as configurações para aparelhos um pouco maiores, mas mesmo assim pequenos.

Pode haver vários *Profiles*, vamos citar dois aqui os quais são os mais importantes para este estudo, MIDP [14] e também o PDAP [15] e ambos estão acima do CLDC.

CLDC como já vimos, rege as configurações para aparelhos extremamente pequenos, ele foi desenvolvido para dispositivos de 160KB a 512KB com memória valida para Java. A quantidade de memória é realmente pequena, o que nos faz pensar muito em termos de aplicações que podem rodar neles e nos traz de volta a certas programações para DOS no que diz respeito à memória. O processamento é também muito fraco, por volta de 10Mhz, e isto se tratando de aparelhos topo de linha, o que nos faz analisar melhor códigos e métodos usados, os quais despendem muito processamento e uso de memória. E também a conexão lenta, tipicamente de 9.600bps.

[11] Seção baseada no livro *Java 2 Micro Edition - Java in small things*.
[12] Connected, Limited Device Configuration
[13] Connected Device Configuration
[14] Mobile Information Device Profile
[15] Personal Digital Assistant Profile

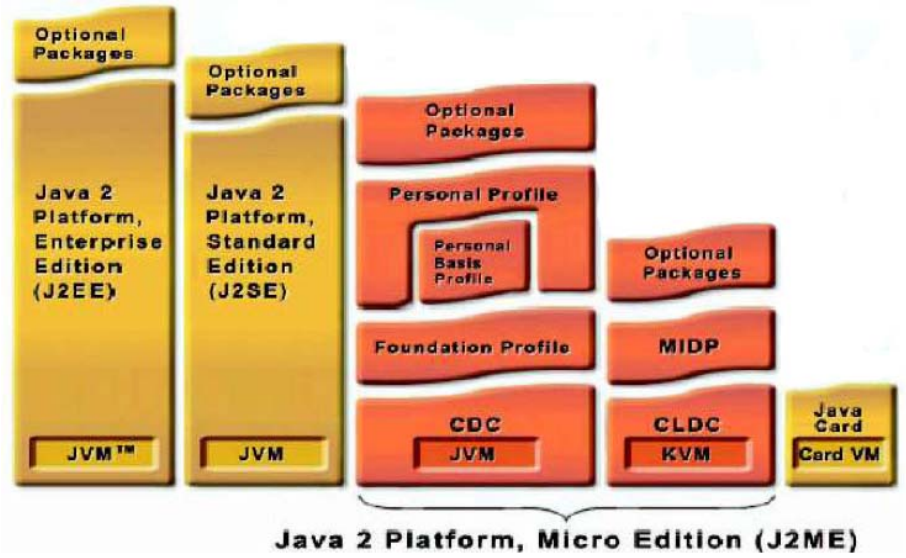

Figura 2.7: Camadas da arquitetura J2ME

MIDP tem as seguintes características:

- 128kB de memória não-volátil para JAVA.
- 32KB de memória volátil para tempo de execução.
- 8KB de memória não-volátil de para armazenamento de dados.
- Uma tela de pelo menos 96x54 pixels.
- Capacidade de entrada de dados seja por teclado (do celular), teclado externo ou mesmo *Touch-screen*.
- Possibilidade de enviar e receber dados em conexão.

Interface de usuário do *MIDP* em baixo nível

Enquanto que há poucas classes e instâncias para negociar com a interface de usuário em baixo nível do MIDP, o desenvolvedor pode negociar com muitos detalhes em baixo nível, tais como sistemas de coordenadas de pixels, fontes, formas geográficas para desenhar e renovar a tela.

O objeto *Canvas* - Diferente da API de alto nível, há somente uma subclasse *Displayable* para usar quando criado uma interface de usuário em baixo nível. Está é a classe *Canvas*. Contudo, a classe *Canvas* é abstrata e requer que a aplicação a estenda para usá-la. Formas gráficas bidimensionais e/ou textos são mostrados num objeto *Canvas* através de um mecanismo de desenho, o qual é feito

chamando o objeto *Graphics*. Em uma instância da subclasse *Canvas* é passao um objeto *Graphics* ao método *paint(Graphics g)* do canvas. Cada subclasse *Canvas* deve implementar este método abstrato, e somente durante a duração do método *paint(Graphics g)* podem aplicações desenhar gráficos de baixo nível na tela. Contudo, a aplicação nunca invoca este método diretamente. Este trabalho é deixado ao aparelho.

O objeto *Graphics* - Uma instância do objeto *Graphics* faz todos os desenhos da interface em baixo nível do MIDP. Ele fornece vários métodos de desenho para mostrar caracteres ou strings, imagens, linhas retangulares, retângulos com os cantos arredondados, e arcos. retângulos, retângulos com os cantos arredondados, e arcos podem ser preenchidos ou não. O objeto *Graphics* não tem que ser criado. Particularmente, uma nova instância do objeto *Graphics* é criada e passada para o objeto *Canvas* através do método *paint(Graphics g)* deste objeto *Canvas*. Isto permite gráficos serem mostrados diretamente na próxima vez que o método *paint(Graphics g)* do objeto *Canvas* for invocado pelo sistema.

Todas as operações de desenho fazem substituição de pixels. Em outras palavras, qualquer operação de desenho especificada em um objeto *Graphics* que coloca um valor pixel, recoloca o prévio valor. Não existe capacidade de combinar ou fundir valores de pixels como é fornecido em muitos sistemas de desenhos sofisticados.

Os objetos *Graphics* suportam 24-bits de cor. As cores componentes vermelho, verde e azul (RGB) são para cada uma alocada 8-bits. Contudo, nem todos os aparelhos suportam 24-bits de cor. Neste caso, o sistema tentará mapear cores disponíveis tão apropriadas quanto possível para as cores requisitadas pela aplicação.

Todos os métodos de desenho geométrico no objeto *Graphics* fazem uso de um sistema de coordenadas. O sistema de coordenadas padrão assume que o canto superior esquerdo da tela do aparelho é o canto (0,0) - figura 2.8. O sistema de coordenadas realmente representa a localização entre cada pixel. Por exemplo, as seguintes coordenadas limitam o primeiro pixel do lado superior esquerdo da tela: (0,0), (1,0), (0,1), (1,1). Cada incremento das coordenadas X e Y representam mover um pixel na tela.

2.2.2 *BREW*

BREW [16] é uma nova plataforma para aplicações de software desenvolvido pela *Qualcomm* [DEITEL, 2001] [17]. A plataforma permite desenvolvedores de *software* criar aplicações que podem ser acessíveis através de uma variedade de dispositivos móveis. *BREW* é uma camada de código que trabalha com *chips Qualcomm*

[16] *Binary Run-Time Environment for Wireless*, ou ambiente binário em tempo de execução para tecnologia sem fio

[17] Está seção foi baseada no capítulo vinte e nove do livro *Introduction to the Internet, World Wide Web and Wireless Communications*.

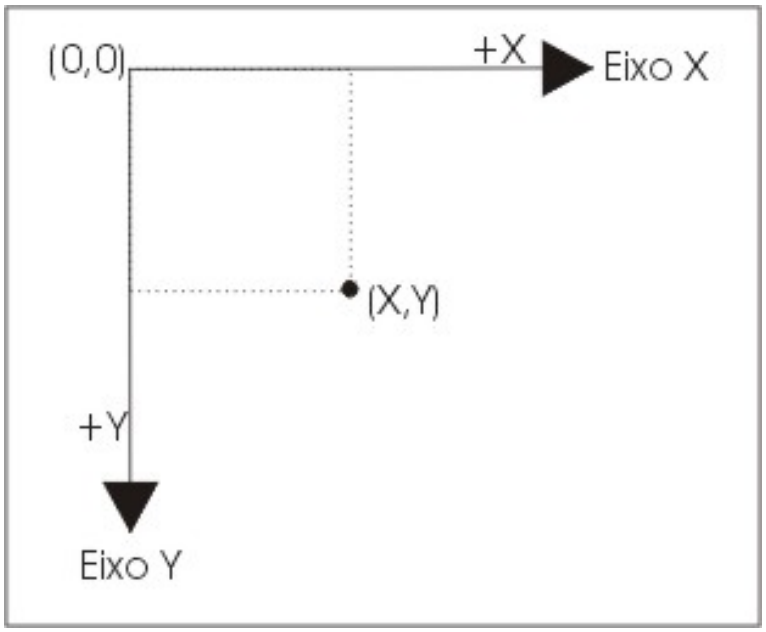

Figura 2.8: Sistema de coordenadas MIDP. As unidades são medidas em pixels

e vários sistemas operacionais para celulares, permitindo-os executar aplicações que são escritas usando ferramentas de desenvolvimento distribuídas pela *Qualcomm*.

Qualcomm desenvolveu *BREW* para padronizar o ambiente de execução em diferentes dispositivos. A padronização da plataforma *BREW* criada permite usuários baixar aplicações celulares sobre a rede dos provedores. Antes do *BREW*, companhias de desenvolvimento de *software* para dispositivos *wireless* tinham que programar informações específicas para cada tipo de dispositivo *wireless*. *BREW* simplificou tarefas como atualizar ou mudar *browsers* e integrou maquina virtual, ou seja, ele ficou independente do sistema operacional.

Qualcomm desenvolveu o *BREW SDK* [18] para facilitar a criação ou melhoramento das existentes características de dispositivos *wireless*.

BREW foi projetado para simplificar integração de aplicações celulares, ou seja, e-mail, mensagem instantânea, *browsers* de internet, e máquinas virtuais em dispositivos móveis.

Integração do *BREW* com Aplicações

Muitos dispositivos móveis suportam somente uma versão de aplicação, tais como um *browser*, de um provedor. Códigos de aplicações desenvolvidos para satisfazer

[18] *BREW Standard Development kit*, ou ferramenta de desenvolvimento padrão do BREW

dispositivos específicos tomam tempo e, como resultado, pode limitar a funcionalidade da aplicação do dispositivo. *BREW* usa uma API uniforme, fornecida pela *Qualcomm*, onde o código da aplicação pode ser escrito diretamente. Quando aplicações são escritas usando a API, *BREW* pode integrar as aplicações com *software* já instalado no dispositivo. A figura 2.9 ilustra as camadas da arquitetura do *BREW* com uma aplicação.

Figura 2.9: Camadas da arquitetura do BREW com uma alicação

Integração do *BREW* com Máquina Virtual

Atualizar ou incluir máquinas virtuais dentro de dispositivos *wireless* pode ser problemático. *Qualcomm* projetou *BREW* para integrar a máquina virtual no topo e ao lado da camada do *BREW* no dispositivo (figura 2.10). Esta estrutura permite *BREW* interpretar a máquina virtual como classes de extensões, ou seja, os métodos e classes da máquina virtual são acessíveis ao *BREW*. A máquina virtual também ganha acessos a ferramentas que o *BREW* e o dispositivo possuem.

Figura 2.10: Camadas da arquitetura do BREW com máquina virtual

2.2.3 *.NET*

Será apresentado aqui o ambiente *Microsoft .NET Mobile Framework* para criação de aplicações *web* móveis no lado do servidor [DEITEL, 2001] [19]. O *.NET Mobile Framework SDK* é uma poderosa ferramenta para a criação de páginas *Web* para dispositivos *wireless*. Usando *.NET*, um desenvolvedor pode criar conteúdo *web* móvel acessível usando linguagens tais como *C#*, *Visual Basic* e *Visual C++*. Em tempo de execução, o *Mobile Framework* traduz estas linguagens para *WML* [20] (falado na seção 2.2.4).

Softwares requeridos para o desenvolvimento e execução de aplicações em *.NET*:

- *Windows 2000 Server*, ou *Windows 2000 Advanced Server*, ou *Windows 2000 Professional* ou versões superiores a estas;
- *Internet Explorer 5.5* ou superior;
- *Service Packet 1*;
- *Internet Information Service (IIS)* ou *Apache*;
- *.NET Framework SDK Beta 1*;
- *.NET Mobile Web SDK Beta 1*;
- *Openwave Mobile Brower (UP.SDK 4.1 Simulator)*.

Usando *Microsoft .NET* nós podemos criar conteúdo *Web wireless* que contém vários formulários na mesma página, e pode ser suportado em vários dispositivos móveis. Esta tecnologia também pode ser programada para produzir diferentes saídas dependendo do corrente dispositivo. Programar conteúdo *Web wireless* é também muito fácil usando *ASP .NET*, porque *WML* (seção 2.2.4) é gerado por um código fácil de entender em outra linguagem. Este código é gerado no servidor e considerado dinâmico, porque ele pode mudar.

2.2.4 *WAP/WML*

O *HDML* [21] foi a inicial linguagem de marcação *wireless* [DEITEL, 2001] [22]. *HDML* foi implementado em milhões de dispositivos quando ele foi apresentado

[19] Seção baseada no capítulo vinte e oito do livro *Introduction to the Internet, World Wide Web and Wireless Communications*.

[20] *Wireless Markup Language*, ou linguagem de marcação sem fio

[21] *Handheld Device Markup Language*, ou linguagem de marcação para dispositivos portáteis

[22] Seção baseada no capítulo treze do livro *Introduction to the Internet, World Wide Web and Wireless Communications*.

pela primeira vez, mas tem sido substituído pelo *WAP/WML*. O *WAP* [23] habilita vários tipos de dispositivos *wireless* comunicarem e acessarem a Internet usando o *WML* [24]. Caracteres (*tags*) *WML* são usados para "*mark up*"(remarcar) uma página *Web* para especificar como ela deve ser formatada em dispositivos *wireless*. *WML* não é um uma linguagem de programação *procedural* como *C*, *Fortran*, *Cobol* ou *Pascal*. *WML* é uma linguagem de marcação. A linguagem identifica os elementos de um documento assim que o *browser wireless*, tal como *mobile browser* da *Openwave*, pode formatar (*renderizar*) o documento no dispositivo *wireless*.

Linguagem de marcação

WML formata texto e imagens. Formatar ("*mark up*") informações com *WML* não deve ser confundido com escrever programas em linguagens de programação tradicionais. Linguagens de programação são usadas para produzir aplicações ou *scripts* que realizam ações específicas pelo programador. *WML* formata conteúdo e *links* de documentos juntos. Ele também fornece mecanismos, tais como *soft keys* [25], que facilita simples interações do usuário.

Em *WML*, o texto é formatado com elementos delimitadores chamados *tag* [26]. Por exemplo, o próprio elemento **wml** indica que o documento é um documento *WML* para ser formatado (*renderizado*) por um *browser wireless*. O documento *WML* inicia com a *tag* **<wml>** e termina com a *tag* **</wml>**. *Tags WML* são sensíveis ao contexto (*case sensitive*) e devem ser escritas em letras minúsculas por ser um padrão.

2.2.5 *i-mode*

NTT DoCoMo (www.nttdocomo.com) é um dos líderes do mundo na indústria de telecomunicação *wireless* [DEITEL, 2001] [27]. *DoCoMo*, que significa "em qualquer lugar"em japonês, é a subsidiária *wireless* da companhia de telecomunicações japonesa, *Nippon Telephone* e *Telegraph (NTT)*. *DoCoMo* foi fundada em 1991, e antes de fevereiro de 1999 a companhia estreou seu serviço núcleo, o fenômeno *i-mode*.

i-mode é um serviço de assinatura para acesso a Internet *wireless* disponível pra assinantes de telefones celulares da *DoCoMo*. O número de assinantes de *i-mode* aumentou mais de 24 (vinte e quatro) milhões até junho de 2001, apenas 28 (vinte e oito) meses desde sua estréia em fevereiro de 1999.

[23] *Wireless Aplication Protocol*, ou protocolo de aplicações sem fio

[24] *Wireless Markup Language*, ou linguagem de marcação sem fio

[25] *soft keys*, é o nome que se dá aos dois botões que estão exatamente abaixo da tela do dispositivo, e que tem rótulos associados a eles na tela.

[26] Palavras chave que contém um par de símbolos maior (>) e menor (<)

[27] Seção baseada no capítulo vinte e dois do livro *Introduction to the Internet, World Wide Web and Wireless Communications*.

DoCoMo procurou novos modelos para criar um serviço que muitos de seus mercados alvos comprariam. A tecnologia que *DoCoMo* desenvolveu para suportar *i-mode* não é nova, mas a aplicação da tecnologia é.

A tecnologia *i-mode*

Telefones celulares no Japão usam a tecnologia *PDC* [28]. *PDC* é um padrão de tecnologia celular digital *TDMA* [29] que é também a base para outros populares sistemas celulares mundiais, incluindo a tecnologia *GSM* [30] da Europa. *DoCoMo* está transformando sua rede *PDC* para *WCDMA* [31] em preparação para o lançamento de sua *3G* [32]. *WCDMA* oferecerá mais altas velocidades habilitando uma maior quantidade de aplicações com serviços sofisticados.

A rede *i-mode* do *DoCoMo*, *PDC-P* [33], corre em paralelo com a rede *PDC*. Como na Internet, *PDC-P* a tecnologia de troca de pacotes para transferir informações. Assinantes do *i-mode* têm a vantagem de trabalhar com pacotes e não com tempo de conexão. Isto significa que quando um usuário baixa um arquivo, não importa se aquele arquivo tomou um ou dez minutos para baixar, o usuário paga pelo arquivo e não o tempo gasto para baixá-lo. Isto é tão lento quanto a velocidade da primeira geração de *modems*. O serviço compensa, de certa forma, para transmissões de baixa velocidade, porque *i-mode* está em um estado "sempre conectado", os usuários não tem que discar para conectar ao serviço de rede. Cada aparelho de telefone *i-mode* tem um botão "i"que o usuário deve pressionar para conectar ao serviço. Telefones celulares que suportam *WAP* também provê serviços de dados, mas assinantes *WAP* devem discar para sua rede para conectar ao serviço. Usuários *WAP* pagam por minuto em suas conexões. Embora *WAP* seja disponível mundialmente, seus assinantes são pouco comparados aos do *i-mode*.

2.3 Tecnologias de comunicação wireless MIPD disponíveis

Esta seção mostra quais as tecnologias disponíveis atualmente para se realizar uma comunicação *MIDP* em dispositivos móveis. Há aqui uma breve descrição de como estas tecnologias trabalham.

---

[28] *Personal Digital Cellular*, ou celular digital particular.

[29] *Time Division Multiple Access*, ou acesso múltiplo por divisão de tempo

[30] *Global System for Mobile Communications*, ou sistema global para comunicação móvel.

[31] *Wideband Code Division Multiple Access*, ou acesso múltiplo por divisão de código em banda larga.

[32] terceira geração da tecnologia *wireless*

[33] *Personal Digital Cellular Packets*, ou pacotes para celular digital particular

2.3.1 Tecnologia de comunicação MIDP 1.0

Aqui é demonstrado apenas as tecnologias de comunicação disponíveis para a versão 1.0 do *profile MIDP*.

Compartilhando um celular

De longe, o mais simples método para um jogo "multi-usuário"é através do compartilhamento de um único aparelho pelos jogadores. Isto pode ser feito bem em jogos de turno, onde com a chegada da vez (turno) de um jogador, este pode se apossar do aparelho para jogar. Tal mecanismo torna o processo simples e funcional. Jogos baseados em turnos são confiáveis e não causam nenhuma sobrecarga de dados.

Hypertext Transport Protocol (HTTP)

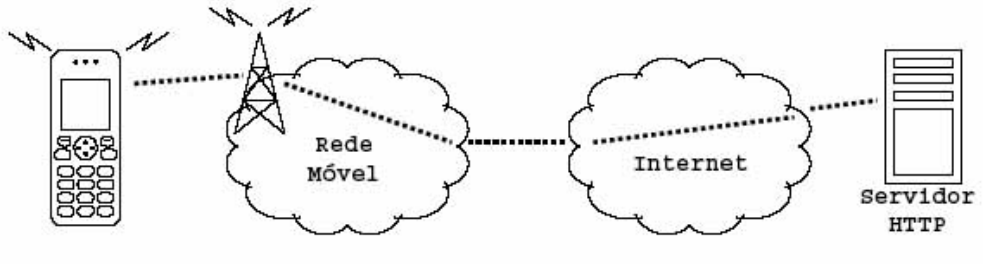

Figura 2.11: MIDP HTTP networking

O protocolo HTTP [34] é suportado por todos os telefones MIDP. Este suporte obrigatório ao MIDP é apenas um suporte para cliente HTTP, não há extensões para servidor. Isto significa que o MIDlet (nome que se dá a uma aplicação MIDP) atua como um *Web browser*. O MIDlet envia uma requisição HTTP ao servidor HTTP, e recebe de volta uma resposta HTTP. Isto é tudo que o padrão HTTP garante, e tudo que um MIDlet deve contar. Em particular, desenvolvedores MIDlet algumas vezes comete o erro de contar com as seguintes características da implementação HTTP para internet, a qual geralmente não trabalha em redes móveis:

- **HTTP *streaming***: devido a implementações HTTP para internet serem construídas no topo dos protocolos de *streaming* (Fluxo corrente de dados), implementações HTTP para internet podem pegar vantagens disto e escrever *streams* de dados contínuos sobre HTTP. Entretanto, redes móveis podem usar uma implementação não TCP, e esta implemetação pode não usar *streaming*. Muitas redes móveis acumulam em uma memória auxiliar (*buffer*)

[34] Hypertext Transport Protocol, ou protocolo de transferência de hipertexto

uma requisição ou resposta inteira antes de enviá-la, e assim *streaming* HTTP de MIDlets tipicamente falham quando são realizadas em telefones reais nestas redes. Por isso, não da para enviar um pouco de dado em um certo tempo: nada será recebido até que todos os dados tenham sido enviados.

- **Respostas atrasadas**: outro truque que desenvolvedores MIDlet tipicamente tentam quando eles descobrem que streaming HTTP não funcionam é atrasar o envio da resposta HTTP do servidor até que algum evento aconteça (ex.: o outro jogador se move). Uma conexão aberta pode ser custosa em uma rede móvel, e são encontradas muito mais *time-outs* (tempo onde o computador tenta se conectar a um dispositivo ou a um outro computador mas não consegui) em redes móveis do que em redes não móveis.

- **Múltiplas conexões HTTP**: telefones móveis freqüentemente não têm recursos para suportar múltiplas conexões HTTP abertas, as quais podem ser completamente custosas em termos de estruturas de dados e *buffers* de dados.

A mensagem que fica viva de todas as acima citadas é que se queremos que uma aplicação MIDlet seja confiável e portável, fixe em enviar uma requisição e imediatamente enviar uma resposta. Um servidor HTTP não tem modos de iniciar uma conexão para um cliente HTTP; se o servidor HTTP deve notificar a aplicação MIDlet de um evento (ex.: quando o outro jogador pega seu turno), o MIDlet deve periodicamente enviar uma requisição HTTP para o servidor. Redes móveis HTTP não são caras para uma pequena quantidade de dados. Caso que geralmente acontece em jogos.

Outros Protocolos

MIDP 1.0 não tem suporte específico para nenhum protocolo de rede a não ser HTTP. Contudo, alguns telefones MIDP 1.0 incluem suporte não-padrão a outros protocolos como *sockets* TCP. Se estiver certo que o alvo é somente estes telefones, pode-se sacrificar a portabilidade para pegar vantagens destas características.

2.3.2 Tecnologia de comunicação MIDP 2.0

Aqui é demonstrado apenas as tecnologias de comunicação disponíveis para a versão 2.0 do *profile MIDP*.

Secure Hypertext Transport Protocol (HTTPS)

HTTPS é o "HTTP seguro", como usado por serviços de comercio eletrônico (*e-commerce*), etc., deve ser suportado por implementações MIDP 2.0. HTTPS não será falado aqui.

Transmission Control Protocol (TCP)

Transmission Control Protocol, ou protocolo de controle de transmissão, ou protocolo orietado a conexão é um dos fundamentais protocolos da internet. Ele é um protocolo confiável, orientado a conexão, que significa:

- Uma vez que um dado enviado ele sempre é recebido, na correta ordem, livre de erros (há não ser que a conexão eventualmente falhe).
- A conexão é estabelecida entre as duas máquinas comunicantes e mantida enquanto durar a comunicação.

Uma analogia comum é que uma conexão TCP é como uma ligação por telefone entre duas máquinas que se comunicam. Na internet, HTTP é geralmente implementado usando TCP.

Um "*socket*"refere-se a um ponto final de uma conexão TCP. Um "*server socket*"em servidor TCP aceita novas requisições de conexão e cria um novo *socket* para cada.

A segurança do TCP freqüentemente simplifica muito o projeto de programas; contudo, quando ele é usado sobre uma rede não-confiável ele pode se apresentar significativamente pior do que o *User Datagram Protocol* (UDP, veja abaixo). Se um pacote é perdido, o TCP tentará reenviar-lo e não entregará subseqüentes pacotes até o pacote perdido tenha sido reenviado com sucesso. Se não é preciso desta confiança de reenvio, pode se achar que o protocolo UDP seja o mais adequado.

MIDP 1.0 não inclui suporte para TCP. MIDP 2.0 especifica suporte TCP, mas faz opcional a inclusão deste por parte dos fabricantes.

User Datagram Protocol UDP

User Datagram Protocol, ou protocolo de pacote de dados de usuário, ou protocolo não-orietado a conexão é outro protocolo fundamental da internet. Ele é um protocolo não confiável, orientado a pacote, que significa:

- Uma vez que um pacote de dados é enviado ele pode ser recebido: uma vez, mais de uma vez, nunca, ou mesmo na ordem errada.
- Não há conexão entre as máquinas comunicantes: cada pacote é individualmente endereçado.

Aqui, uma analogia comum é que UDP é como um correio postal, com pacotes de dados sendo cartas ou cartões postais enviados entre máquinas que se comunicam. O termo "*datagram*"simplesmente significa um pacote de dados.

UDP adiciona muito pouco *overhead* (sobrecarga) para a comunicação e é "geralmente"a mais eficiente tecnologia de rede. Muitas aplicações não necessitam da confiança do TCP; por exemplo, quando o dado é sensível ao tempo, como para vídeo *streaming* (fluxo continuo de dados), é freqüentemente melhor simplificar a perda de um pacote do que segurar subseqüentes pacotes enquanto tenta recuperar um pacote que perdeu sua validade.

É amplamente discutido se UDP ou TCP é o protocolo mais adequado para jogos. A improvável latência (tempo gasto para o envio e recebimento de um dado) das redes móveis tende a enfatizar a diferença entre as duas.

Duas soluções comuns são:

- Use ambos: *datagrams* UDP para mensagens que não necessitam ser confiáveis, e uma conexão TCP para mensagens que deve ser confiáveis.
- Use somente UDP, mas implemente uma fina camada no topo que permita algumas mensagens serem enviadas com confiança e algumas não.

Quando implementado sobre um protocolo de pacote de dados tal como GPRS (*General Packet Radio Service* - Serviço de comunicação que usa a tecnologia de pacotes para enviar dados via rede.), rede UDP móvel não é cara por ser pequena a quantidade de dados que jogos geralmente enviam. Dado a latência de redes móveis, ali geralmente não existe necessidade de enviar pacotes a uma freqüência maior que uma vez a cada segundo. Se estiver ocorrendo o envio de pacotes nesta freqüência, tenha atenção para seu tamanho, e tente não enviar pacotes desnecessariamente.

MIDP 1.0 não inclui suporte para UDP. MIDP 2.0 especifica suporte UDP, mas faz opcional a inclusão deste por parte dos fabricantes.

Cabo Serial

Uso de um cabo serial permite dois jogadores (cada um com um telefone) jogarem junto, desde que eles estejam em um mesmo local físico. Em MIDP 2.0, comunicação por cabo serial entre dois telefones pode ser suportado através da *interface* `SerialPortConnection`. Porém, no tempo de escrita do documento [NOKIA01, 2003], esta interface ainda não era suportada por telefones MIDP da Nokia.

Infra-Vermelho (IrDA)

O uso de infravermelho também permite dois jogadores (cada um com um telefone) jogarem junto, desde que eles estejam em um mesmo local físico. Os tele-

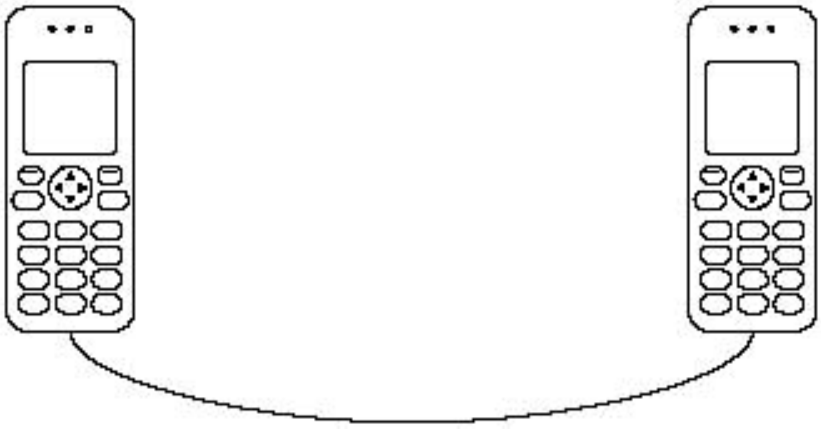

Figura 2.12: Comunicação entre Telefones por cabo serial

Figura 2.13: Comunicação entre dois telefones por infravermelho

fones devem ser segurados numa posição fixa, com suas portas seriais apontando uma para a outra. Isto pode não funcionar bem se o jogo for excitante. Alguns telefones têm uma porta infravermelha no topo, outros na esquerda e ainda outros na direita. Encontrar uma posição de jogo confortável pode tornar-se difícil para dois jogadores.

Em MIDP 2.0, a comunicação infravermelha entre telefones pode ser suportada através da *interface* `SerialPortConnection`. No tempo de escrita do documento [NOKIA01, 2003], esta interface ainda não era suportada por telefones MIDP da Nokia.

2.3.3 Tecnologia de comunicação com "Pacotes Opcionais"MIDP

Telefones MIDP 1.0 e MIDP 2.0 podem incluir vários "pacotes opcionais"com funcionalidade extra. Esta seção cobre as tecnologias de comunicação destes pacotes opcionais.

Bluetooth

Bluetooth é uma tecnologia de comunicação de curta freqüência de rádio que permite até oito dispositivos comunicarem juntos sobre um faixa por volta de dez

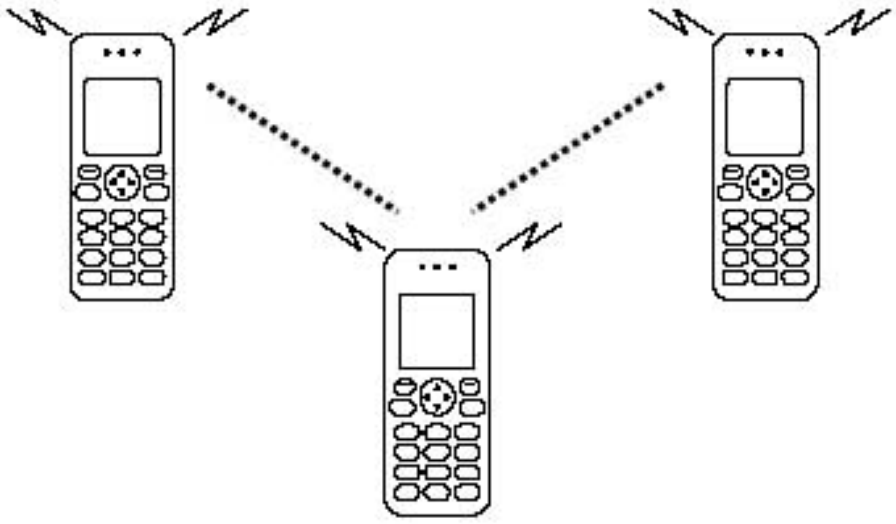

Figura 2.14: Comunicação Bluetooth entre telefones

metros. *Bluetooth* tem baixa latência e é muito adequado para jogos multi-usuário - ele é usado intensivamente em jogos multi-usuário no andar de jogos móveis da *Nokia N-Gage*^{TM}. Os telefones não tem que ser apontados um para o outro, pois *Bluetooth* transmite em todas as direções.

Em telefones *MIDP*, *Bluetooth* deve ser suportado por pacotes opcionais das APIs Java para *Bluetooth*. Estas APIs não serão cobertas aqui.

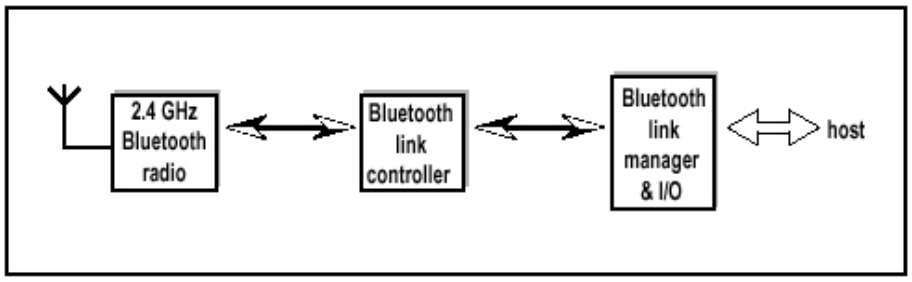

Figura 2.15: Comunicação *Bluetooth* entre telefones

O sistema *Bluetooth* consiste de uma unidade de rádio (*Bluetooth Radio*), uma unidade controladora de *link* (*Bluetooth link controller*), e uma unidade de suporte para gerenciamento de *link* e funções de interface de terminal de *host* (*Bluetooth link manager & I/O*) - figura 2.15. A interface controladora do *host* (*HCI*) - figura 2.16 - fornece um significado para um dispositivo *host* acessar as capacidades do *hardware Bluetooth*. Por exemplo, um computador *laptop* pode ser um dispositivo *host* e um *PC card* inserido no *PC* o dispositivo *Bluetooth*. Todos os comandos

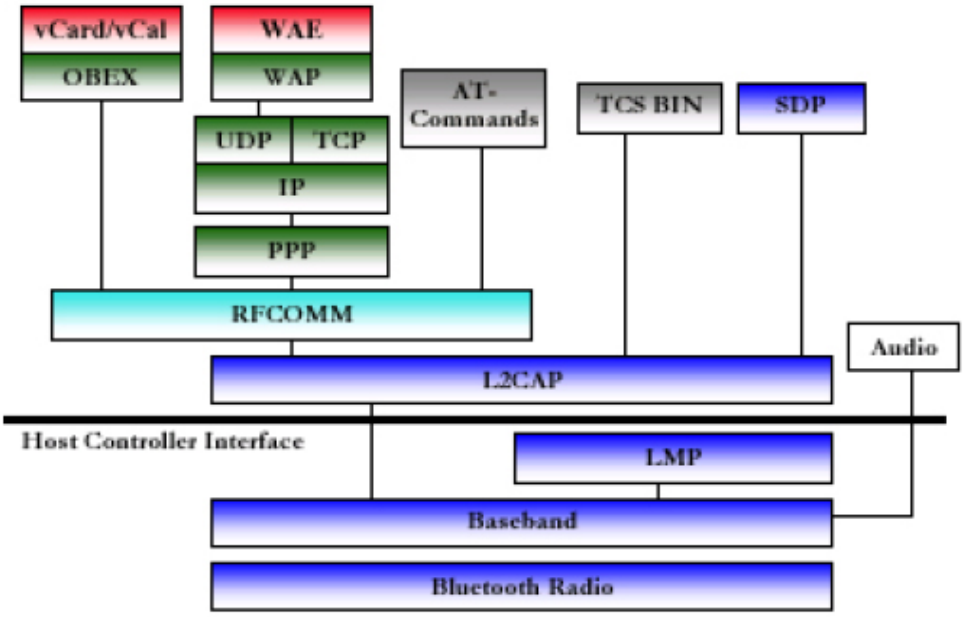

Figura 2.16: Pilha de protocolos *Bluetooth* v1.1

do *host* para o módulo *Bluetooth* e eventos do módulo para o *host* ocorrem através da interface *HCI*. A pilha de protocolos que está acima dos *hardwares* radio e controladora de *link*, residem parcialmente na unidade *Bluetooth* e parcialmente no dispositivo *host* (figura 2.16).

***Short Message Service* (SMS)**

A tecnologia *Short Message Service*, ou serviço de mensagem curta permite um telefone móvel trocar mensagens curtas com outros telefones ou com um computador servidor tal como um servidor de jogo. SMS é muito familiar em sua forma de mensagem de texto, mas ela também pode ser usada para enviar dados binários tais como toque de som ou dados de jogo. SMS é uma tecnologia de "armazenagem e encaminhamento", isto é, mensagens são enviadas a um dispositivo chamado *Short Message Service Center* (SMSC) ou centro de serviço de mensagem curta, o qual armazena estas mensagens em uma fila e encaminha-as posteriormente para o recipiente. Se o recipiente é um servidor ou um telefone que está ligado e tem recepção, e o SMSC não está pesadamente carregado, as mensagens SMS são geralmente recebidas em poucos segundos depois que elas foram enviadas, caso contrário, elas podem levar dias para serem entregues, ou no final das contas não serem entregues de nenhuma maneira.

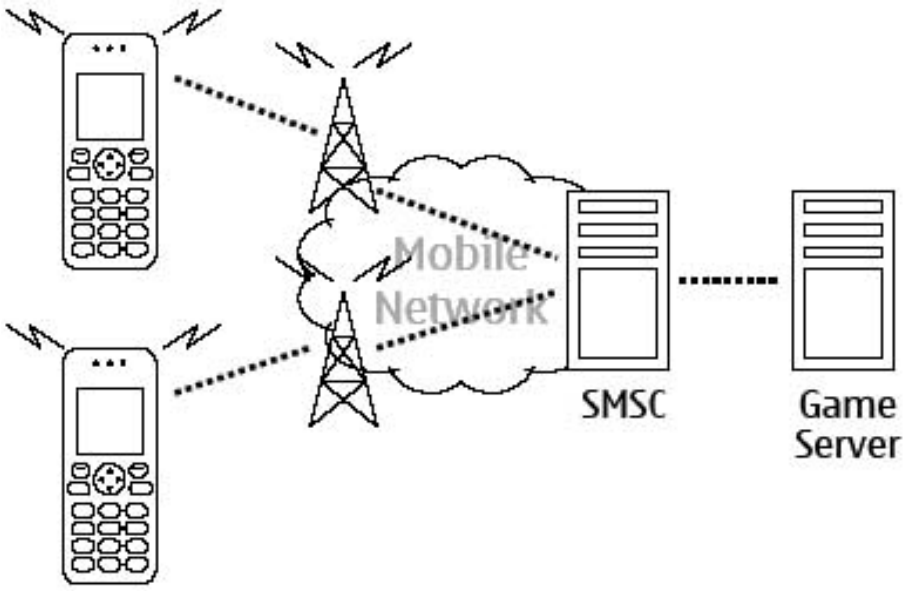

Figura 2.17: Comunicação SMS entre telefones e servidores

Em telefones MIDP, SMS pode ser suportado por pacotes opcionais da API *Wireless Messaging*. Isto ainda não era suportado por muitos telefones no tempo de escrita do documento [NOKIA01, 2003], os únicos telefones NOKIA que suportavam esta API eram os telefones Nokia 3650 e as recentes versões do telefone Nokia 3410. Todas as versões do telefone Nokia 3410 suportam uma API Nokia muito similar chamada SMS API.

Ao usar SMS para jogos MIDP, tenha em mente que cada mensagem custa como uma simples mensagem de texto normal. Um jogo de Xadrez envolvendo 50 turnos pode sair muito caro para o jogador.

Multimedia Messaging Service (MMS)

A tecnologia *Multimedia Messaging Service*, ou serviço de mensagem multimídia é uma atualização, melhoramento, da versão SMS que permite mensagens terem várias partes, incluindo texto, figuras, som, e vídeo. Como SMS, MMS é uma tecnologia de "armazenagem e encaminhamento", e as mesmas notas a respeito de tempo de entrega são aplicadas.

MMS é implementada usando a combinação de SMS e HTTP. Se o jogo MIDP envolver comunicação com telefones sem MIDP MMS, provavelmente ele será mais bem servido por SMS ou HTTP, ou por uma própria combinação de ambos.

Em telefones MIDP, MMS pode ser suportado por pacotes opcionais da API *Wireless Messaging* 2.0. No tempo de escrita do documento [NOKIA01, 2003],

isto estava ainda sendo especificado. Assim, não entraremos em mais detalhes aqui.

2.3.4 Tecnologias para servidor de jogo

Aqui, é demonstrado as tecnologias de comunicação mais apropriadas para a parte servidora se comunicar com a parte cliente *MIDP*.

HTTP e HTTPS

No lado do servidor, pode-se usar qualquer tecnologia que normalmente é usada para servidores de HTTP, ou seja, paginas *Web* estáticas, CGI, ASP, servlets Java, e JSP.

TCP e UDP

Atualmente não existe na plataforma Java 2 nenhum modelo de aplicação *servlets* para os protocolos TCP e UDP. Se usado a linguagem de programação Java para escrever um servidor de jogo que use TCP ou UDP, é necessário ter ou seu próprio servidor escrito, por exemplo usando J2SE (*Java 2 Standard Edition* - Edição padrão do Java 2), ou um servidor de jogo proprietário.

SMS e MMS

O *Mobile Games Interoperability Forum* (MGIF) ou fórum de interoperabilidade de jogos móveis, que tem tornado o grupo de trabalho de serviços de jogos do *Open Mobile Alliance* (OMA) ou livre aliança móvel, tem publicado um padrão de especificações para servidores de jogos chamado *MGIF Platform Specification* [MGIF-SPEC], o qual inclui suporte para ambos SMS e MMS. Porém, no tempo de escrita do documento [NOKIA01, 2003] nenhuma implementação deste padrão tinha sido anunciada ainda.

2.4 Diferença entre um jogo de estratégia e um jogo de ação

A principal diferença entre um jogo de estratégia e um de ação está em que num jogo de estratégia o jogador deve pensar antes de executar uma jogada, e lhe deve ser dado um tempo para isto (ex.: jogo de xadrez), enquanto que em um jogo de ação o jogador usa mais de sua agilidade do que da lógica pra executar uma ação, pois ele não tem um tempo razoável para bolar uma estratégia e até as circunstâncias do jogo não o permitem (os usuários podem-se movimentar todos ao mesmo

tempo). Devido a maioria dos jogos de estratégia serem *baseados em turno* [35] os jogadores têm o tempo do turno para bolar sua estratégia, e também, enquanto ele tem o turno ele pode executar ações enquanto o(s) outro(s) não, podendo assim se despreocupar e fazer sua estratégia sem perigo e com calma. Em jogos de ação todos os jogadores podem estar ao mesmo tempo interagindo, e assim, mesmo se o jogador bolar uma estratégia ela pode dar errado devido à dinamicidade do jogo.

2.5 Lançamento de Projéteis

Podemos observar que quando um projétil é lançado com um determinado ângulo com a horizontal, este descreve no ar uma trajetória que é uma parábola [USP, 2004] [36].

O que acontece com a velocidade inicial do projétil?

Quando o projétil está subindo, a sua velocidade inicial vai diminuindo até atingir um valor mínimo no ponto mais alto da trajetória (vértice da parábola) e vai aumentando quando está descendo até atingir o solo (alcance do projétil).

Por que a velocidade do projétil tem esta variação?

Para que haja variação da velocidade, precisa haver forças atuando; desprezando a resistência do ar, a força que está atuando no projétil é a força peso. A força peso atua na vertical de cima para baixo, comunicando ao projétil uma aceleração denominada aceleração da gravidade. Esta aceleração, para corpos próximos à superfície da Terra, vale aproximadamente 9,8 m/s2.

Quando o projétil está subindo, a força peso, sendo para baixo, faz com que a velocidade diminua (movimento retardado) e quando o projétil está descendo, a força peso, atuando no mesmo sentido, faz com que a velocidade aumente (movimento acelerado).

2.5.1 Princípio da Independência dos Movimentos (Galileu)

O movimento do projétil é um movimento bidimensional, sendo realizado nas direções horizontal (X) e vertical (Y); este movimento é composto de dois tipos de movimentos:

- Movimento uniforme na direção horizontal (X);
- Movimento uniformemente variado na direção vertical (Y);

Galileu já sabia disto no século XVI, e baseando-se em fatos experimentais, enunciou o **Princípio da Independência dos Movimentos**, que diz o seguinte:

[35] Em jogos baseados em turnos cada jogador tem sua vez de jogar, esta "vez de jogar"são os turnos, os quais são alternados pelos jogadores

[36] Esta seção foi baseada em material colhido no site *http://educar.sc.usp.br/fisica*

"Quando um móvel realiza um movimento composto cada um dos movimentos componentes se realiza como se os demais não existissem."

No nosso caso este princípio se aplica, porque o movimento na direção horizontal se realiza uniformemente, independente do movimento na vertical que é uniformemente variado.

2.5.2 Análise vetorial / Movimento de projéteis

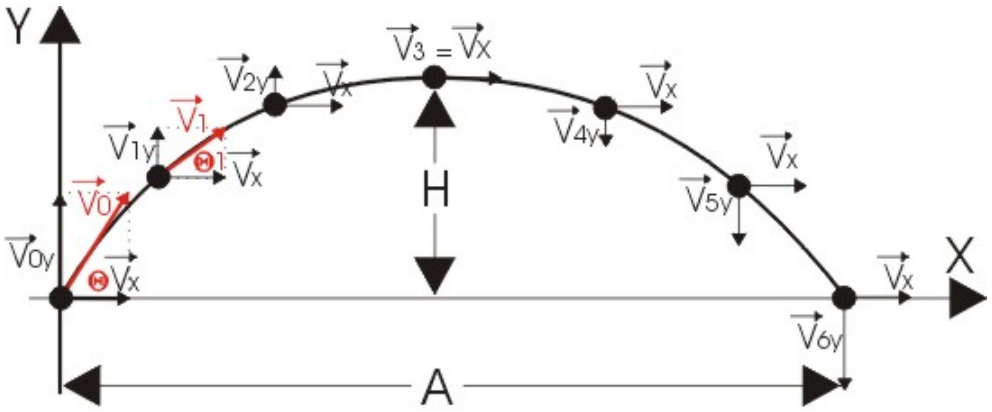

Figura 2.18: Trajetória de um projétil

A figura 2.18 mostra a trajetória de um projétil. Foram traçados os vetores velocidade, V_1, V_2, V_3, V_4, V_5 e V_6, que são tangentes a cada ponto da trajetória. Na figura também está indicado o alcance, A, e a altura máxima do projétil, H.

Estes vetores velocidade apresentam as componentes, V_x e V_y, para cada posição, nas direções X e Y (figura 2.18). Como na direção X o movimento é uniforme, o valor da componente V_x será constante, ou seja, $V_{1x} = V_{2x} = ... = V_{nx} = V_x$. Na direção Y o movimento é uniformemente variado, portanto cada componente V_y terá um valor. Observe que, vetorialmente, o valor de V_y diminui na subida, anula-se no vértice da parábola (altura máxima) e aumenta na descida.

O projétil foi lançado a partir de O (origem), fazendo um ângulo com a horizontal (figura 2.19). Para determinar as componentes V_x e V_{0y}, sendo conhecidos o ângulo e a velocidade V_0, basta projetar o vetor V_0 nas duas direções X e Y, obtendo:

$$V_x = V_0 cos\theta$$

$$V_{0y} = V_0 sen\theta$$

$$V_{1y} = V_1 sen\theta$$

e analogamente determina-se V_{2y}, V_{3y}, ...

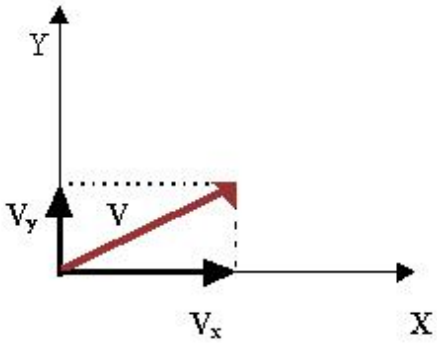

Figura 2.19: Vetor velocidade V e as componentes V_x e V_y.

O vetor resultante V (figura 2.19) é dado pela soma dos dois vetores V_x e V_y:

$$V = V_x + V_y$$

Pode-se determinar o módulo do vetor velocidade, V, para cada posição, sendo conhecidos os módulos das componentes, V_x e V_y (figura 2.19), obtendo:

$$V_2 = V_{2x} + V_{2y}$$

2.5.3 Determinação da aceleração da gravidade

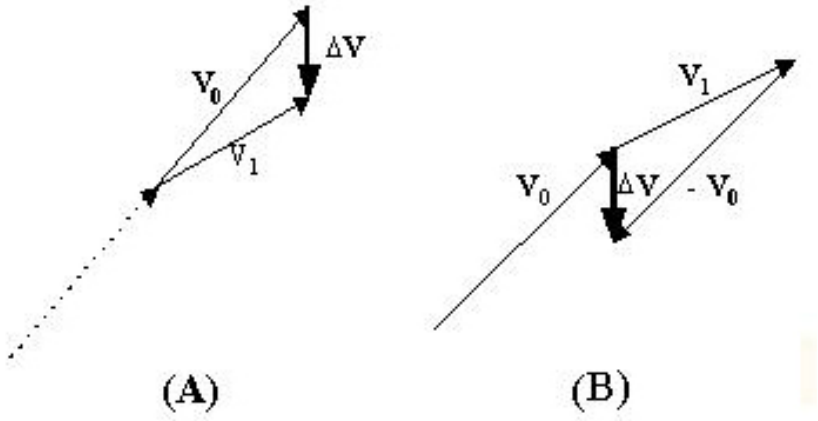

Figura 2.20: Diferença entre os dois vetores velocidade para duas posições sucessivas. (A) Método do paralelogramo; (B) Método da triangulação.

Considerando os vetores velocidade da figura 2.18, V_0 e V_1, por exemplo, e colocando as origens destes vetores coincidentes (parte A da figura 2.20) ou colo-

cando a origem do vetor oposto, $-V_0$, coincidente com a extremidade do vetor V_1 (parte B da figura 2.20), obtém-se a diferença entre dois vetores velocidade (ΔV) para duas posições sucessivas. Fazendo o mesmo procedimento para todas as posições, para intervalos de tempo iguais, observa-se que esta diferença de velocidade é constante, para quaisquer duas posições, ou seja, a aceleração é constante:

$$a = \frac{\Delta V}{\Delta t} = constante$$

$$a = -g$$

Onde g é a aceleração da gravidade. O sinal para g é considerado negativo porque a trajetória é orientada positiva para cima e o vetor g atua para baixo.

Capítulo 3

O jogo Panzer

Neste capítulo será apresentado o funcionamento em alto nível do jogo de estratégia multi-usuário nomeado Panzer, o qual será destinado a aparelhos móveis que utilizam a tecnologia *J2ME* com a *configuration CLDC 1.1* e o *profile MIDP 2.0*. Como será uma apresentação de alto nível do jogo, não será explanado detalhes da implementação do jogo.

3.1 O jogo

O nome do jogo, Panzer, é uma homenagem a um tanque alemão que ficou famoso na segunda guerra mundial.

O jogo Panzer é um jogo de batalha de tanques onde o usuário deve utilizar de seus conhecimentos físicos em lançamento de projéteis para atingir seu adversário. Ele deve ajustar o ângulo do canhão, a velocidade inicial do tiro, e a posição do tanque para projetar seu tiro. Para movimentar o tanque ele devem usar as teclas 4 (quatro) e 6 (seis), para movimentar o ângulo do canhão as teclas 2 (dois) e 8 (oito), e para ajustar a velocidade inicial do tiro as teclas 1 (um) e 3 (três), quando o usuário achar que está com sua melhor configuração para o tiro ele deve apertar a tela 5 (cinco) para fazê-lo (figura 3.1).

O jogo Panzer é um jogo de estratégia com vista de perfil (figura 3.2) onde cada usuário tem o domínio de um tanque. Para entrar no jogo o usuário deve fornecer o endereço e a porta do servidor ao qual o cliente conectará. O primeiro usuário a entrar no jogo pega o tanque azul de número 1 (um) e o segundo usuário pega o tanque vermelho de número 2 (dois). Quando o primeiro usuário entra no jogo ele deve esperar que o segundo usuário entre para dar inicio ao jogo, neste momento uma tela de aviso é mostrada ao usuário dizendo que ele deve esperar um adversário entrar.

Para executar o jogo Panzer deve-se ter um mínimo e máximo de dois jogadores conectados. Um mínimo de dois, devido a um duelo só ocorrer se houver

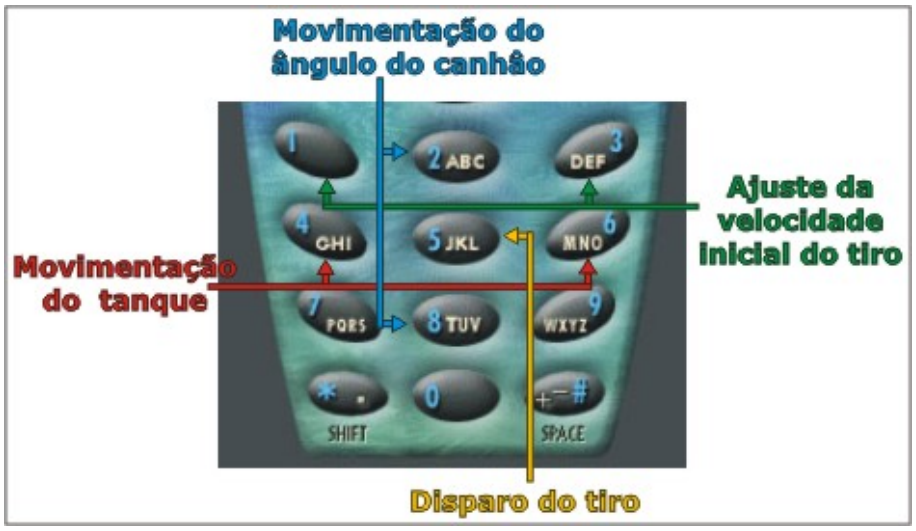

Figura 3.1: Botões de controles do jogo

pelo menos dois jogadores em confronto. Um máximo de dois, devido à própria limitação da disposição gráfica do jogo, que se mostra de perfil - visão 2D. A área de movimentação dos tanques fica entre o canto da tela e o obstáculo que os separa. De um modo geral, os cenários do jogo panzer são compostos de um obstáculo que fica no meio da tela separando os dois tanques, este obstáculo pode ser uma montanha, um precipício, um lago, uma floresta, um prédio, etc.

Com os dois usuários conectados inicia-se o jogo, como o jogo é de estratégia teremos a cada rodada um usuário obtendo o turno, quando um usuário está com o turno apenas ele pode se mexer, o outro apenas observa a movimentação de seu adversário. O último usuário a entrar inicia com o turno e assim tem o direito de ajustar os parâmetros para dar o tiro. O usuário com o turno deve procurar a melhor configuração de força do tiro, ângulo do canhão e posição do taque, considerar a direção e velocidade do vento (que pode ser percebida por uma nuvem que representa tal velocidade) para tentar acertar o adversário que está do outro lado da tela separado por um obstáculo (ex.: uma montanha). Quando o usuário com turno achar que está pronto para dar o tiro ele aperta a tela 5 (cinco) para executá-lo. Neste momento ocorrerá o desenho (a renderização) do tiro, estando os dois clientes sem a capacidade de se mexerem, podendo apenas observarem a movimentação do tiro. Esta acaba quando o tiro (a bola) encontra um obstáculo, que pode ser o solo, a montanha ou um dos tanques. Se o tiro acertar um dos tanques, aquele tanque que foi atingido terá um decréscimo em sua energia. Após este estágio, o turno é passado para o outro cliente e o jogo volta para o estágio de configuração dos parâmetros do tiro. O desenrolar do jogo continua até que a energia de um dos tanques acabe, finalizando assim o jogo.

Figura 3.2: Visão do jogo

O menu principal do jogo é composto dos seguintes itens:

- Iniciar jogo - Onde o usuário inicia um jogo com outro usuário que já está conectado, ou cria um novo jogo e espera a entrada de um adversário.
- Ajuda - Uma breve explicação do funcionamento do jogo e da ação das telas.
- Créditos - Lista as pessoas envolvidas no desenvolvimento do jogo.
- Sair - O usuário deixa o jogo, o jogo é fechado.

3.2 Porque Panzer é um jogo de estratégia?

Como explanado na seção 2.4, um jogo de estratégia permite ao jogador arquitetar seu plano, isto por ser, geralmente, baseado em turnos. Como o jogo Panzer é baseado em turnos ele dá ao jogador este tempo e também não há a dinamicidade de um jogo de estratégia, pois enquanto um jogador manipula seu tanque o outro não pode fazê-lo. Assim, logo após o jogador ter ajustado todos os parâmetros do tiro, e apertado o botão para executá-lo, inicia-se o lançamento do mesmo, neste momento como o outro adversário não pode se mexer toda a estratégia (a configuração dos parâmetros) bolada pode agora ser executada, caracterizando-o assim como um jogo de estratégia. Se no momento do lançamento do tiro o adversário pudesse se mexer, toda a estratégia bolada seria perdida, e teríamos assim um jogo de ação, pois o adversário usaria também de sua astúcia para se desviar do tiro.

Capítulo 4

Implementação

Neste capítulo será apresentado como foi feita a implementação do jogo Panzer, com descrições de como foi projetada a interface com o usuário, os estágios de execução do jogo, a arquitetura de comunicação utilizada, bem como a lógica implementada no combate do jogo. Para realizar esta implementação foi utilizada a tecnologia J2ME (seção 2.2.1), o motivo da utilização desta tecnologia é descrito na seção 4.1. É necessária a leitura da seção 2.2.1 para o entendimento da seção 4.5.

4.1 Por que utilizar J2ME?

Motivações:

- O suporte à tecnologia na maioria dos aparelhos móveis;
- Por ser uma linguagem poderoso;
- Ter um razoável domínio sobre a tecnologia Java;
- Ser um fã, amante da tecnologia Java;

4.2 Interface

Esta seção apresentará a interface de interação entre o usuário e o jogo Panzer. Esta interação inicia após a criação do jogo e termina com a finalização do mesmo. Há algumas etapas a se seguir até o início do combate, e outras como ajuda e créditos. Estas etapas serão demonstradas utilizando-se de figuras para uma melhor ilustração de como interar-se com elas.

A interface apresentada ao usuário pode ter uma aparência diferente para cada tipo de dispositivo móvel, pois diferentes fabricantes de dispositivos implementam

os elementos de interação (*widgets*) da forma que desejarem. Assim, as ilustrações do jogo Panzer apresentadas são da interface do dispositivo padrão de 65mil cores disponibilizado pelo emulador *Wireless Toolkit 2.1* [1].

4.2.1 Executando a aplicação

Para que a aplicação possa ser executada ela deve já estar armazenada (instalada) no dispositivo móvel. A forma mais comum de carregar uma aplicação para dentro do aparelho é pela *Web* via *WAP* [2]. Pois existe uma forte recomendação para que esta instalação seja feita *on the air* [3] [SUM, 2003], o qual facilita a distribuição da aplicação pela Web.

Figura 4.1: (a)Tela de apresentação do jogo Panzer; (b) Tela de menu do jogo Panzer;

Considerando que a aplicação já esteja armazenada no dispositivo móvel, após o usuário carrega-lá, a execução é iniciada e a tela de apresentação é mostrada (parte a da figura 4.1), com dois botões *soft keys* [4]:

- Sair - Aplicação é fechada e descarregada;
- Menu - É apresentado um menu de opções do jogo Panzer;

[1] Encontra-se no endereço http://java.sun.com/products/j2mewtoolkit

[2] *Wireless Aplication Protocol*, ou protocolo de aplicações sem fio

[3] OTA - pelo ar

[4] Nome que se dá aos dois botões que estão exatamente abaixo da tela do dispositivo, e que tem rótulos associados a eles na tela.

Tendo o usuário escolhido a opção Menu, a aplicação mostra três opções ao usuário (parte b da figura 4.1) e dois botões *soft keys*:

- Voltar - Retorna à tela de apresentação;
- Selecionar - Executa uma das três opções selecionada pelo usuário;
- Iniciar jogo - Mostra tela de conexão com o servidor;
- Ajuda - Mostra tela de ajuda;
- Créditos - Mostra tela de créditos;

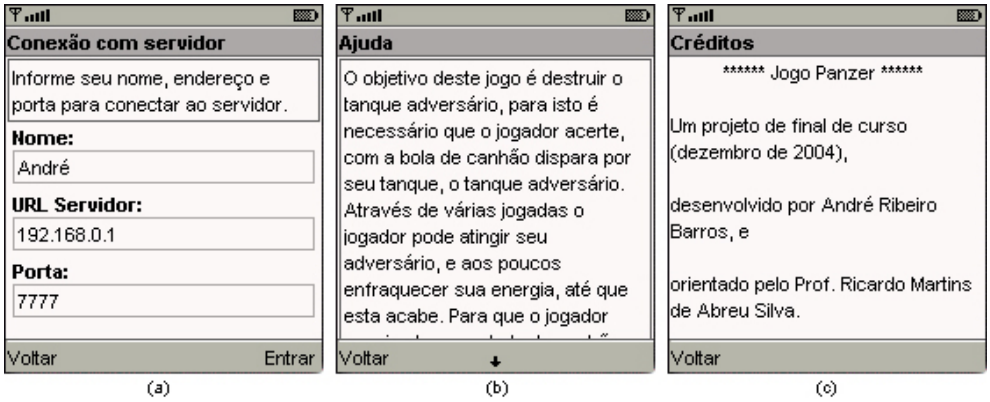

Figura 4.2: (a)Tela de entrada de dados para conexão com servidor; (b)Tela de ajuda do jogo Panzer; (c)Tela de créditos

A tela de conexão com servidor (parte a da figura 4.2) pede que o usuário entre com seu nome, com o endereço do *host* servidor, e a porta no servidor usada para a comunicação. Nesta tela há campos para a entrada destes dados, e dois botão *soft key*:

- Voltar - Retorna a tela de menu;
- Entrar - Inicia/Entra no jogo;

A tela de Ajuda (parte b da figura 4.2) contém uma breve explicação do objetivo, funcionamento, e comandos do jogo Panzer. Nesta tela são apresentados a ajuda e um botão *soft key* para que o usuário possa voltar a tela de menu.

A tela de Créditos (parte c da figura 4.2) apresenta o desenvolvedor do jogo Panzer e seu orientador. Nesta tela são apresentados os créditos e um botão *soft key* para que o usuário possa voltar a tela de menu.

Figura 4.3: (a)Tela de espera, onde o jogador que inicia o jogo aguarda a entrada de seu adversário; (b)Tela do jogo Panzer, jogador com tanque dois ajusta os parâmetros para o tiro;

Após o primeiro jogador iniciar o jogo uma tela de espera é exibida para que ele aguarde a entrada de seu adversário, esta tem uma imagem e um *ticker* [5] que passa o aviso de aguardo (parte a da figura 4.3). Esta tela permanecerá até o momento em que seu adversário entra no jogo. Após isto, o jogo se inicia e o segundo jogador a entrar, o qual tem posse do tanque dois, ganha o turno. Com a posse do turno ele inicia os ajustes dos parâmetros para executar o lançamento da bola de canhão. Caso um dos jogadores queira deixar o jogo, ele deve usar o botão *soft key* a esquerda da tela para abandonar o jogo (parte b da figura 4.3).

Quando o jogador abandona o combate o jogo é finalizado, não podendo o jogador retoma-lo. Com o fim ou abandono do jogo o usuário retorna a tela de menu, podendo: iniciar/entrar em um novo jogo, sair da aplicação, etc.

4.3 Estágios do jogo

Nessa sessão serão apresentadas as etapas do jogo Panzer, estas etapas são divididas em estágios e transições. Há dois estágios fundamentais, mais um complementar e as transições entre estes:

[5]Texto que corre no topo da tela, semelhante às *tickers* de mercados de ações, e semelhante ao <*marquee*> de HTML. O recurso de *Ticker* serve para algum texto informativo que possa ser importante, ou que deve ser lembrando a toda momento.

- **Estágio A (Ajuste de parâmetros para o tiro)**: Neste estágio o cliente que tem o turno deve ajustar os parâmetros (valores) para que ocorra o tiro. Estes parâmetros são: posição do tanque, ângulo do canhão, e velocidade inicial do tiro. Os dados posição do tanque e ângulo do canhão são enviados periodicamente via UDP para o servidor, enquanto isto o cliente que não tem o turno apenas ouve do servidor tais parâmetros, também via UDP. Ao término dos ajustes o cliente com o turno deve apertar um botão autorizando repassar as configurações do tiro ao servidor (isto é o tiro), estes parâmetros são passados via TCP. A ação de apertar o botão do tiro representa a transição do estágio 'A' para o estágio 'B'. Neste estágio o cliente que mantém o turno é quem o controla, quando ele aperta o botão de tiro ele permite ao servidor informar ao outro cliente a passagem de estágio.

- **Estágio B (Processamento do tiro)**: Neste estágio não há interação do usuário, os clientes ficam por conta de renderizar o tiro. Neste estágio não há transição de informações na rede. Os clientes devem somente faz a renderização do tiro com base nas informações que foram passadas com o termino do estágio anterior. Neste estágio são os clientes quem o controlam, assim ao final da renderização os eles devem informar o fim do mesmo para que haja a sincronização do jogo.

- **Estágio de espera** - É necessário este estágio para que haja uma sincronização na transição do estágio A para o B e vice-versa. Na transição do estágio A pro B os clientes deve iniciar juntos a renderização para ocorrer a sincronização das movimentações das imagens nos clientes, pois a renderização dos mesmos são realizadas independentes do servidor, ficando a cargo de cada cliente a renderização. Na transição de B pra A é também necessário que os clientes iniciem juntos o estágio, para que o cliente sem o turno não perca a movimentação do cliente com turno.

4.3.1 Fluxograma com as transições

Aqui será mostrado um fluxograma completo com os estágios e transições do jogo Panzer. As transições são enumeradas e explicadas em detalhes, fazendo correspondência com as figuras 4.4 e 4.5.

Os itens enumerados a seguir dizem respeito as transições processadas pelo cliente que tem o turno (figura 4.4).

1. Dados enviados periodicamente pelo cliente ao servidor (via UDP):
    - Posição do tanque;
    - Ângulo do tiro;
2. Tarefas realizadas:

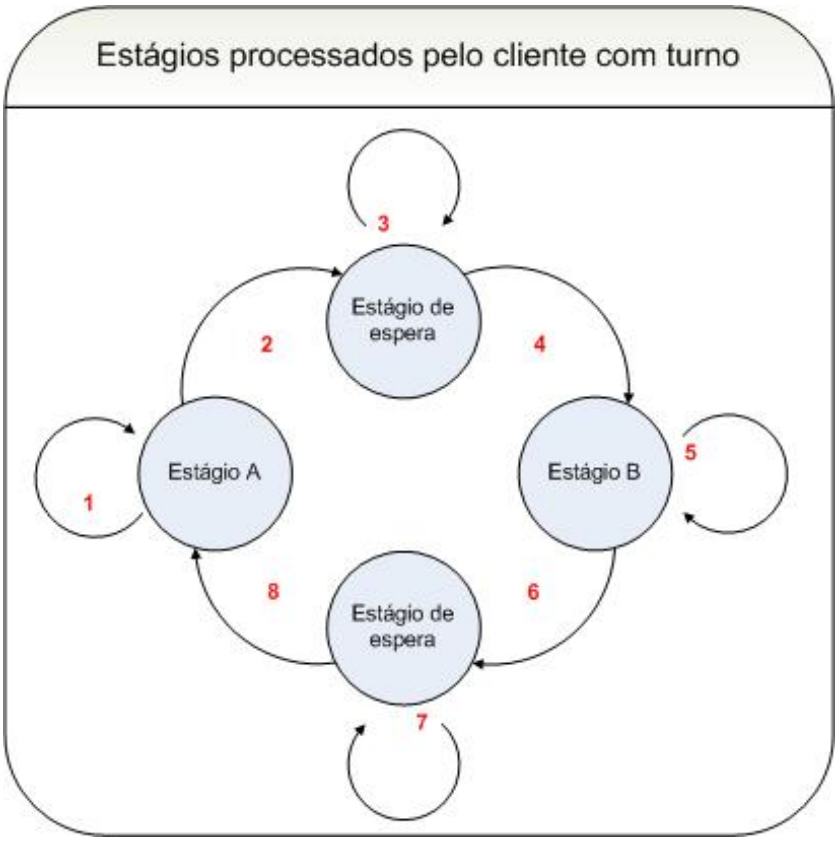

Figura 4.4: Estágios processados pelo cliente com turno

- Envia para o servidor os dados necessários a renderização do tiro (via TCP);
- Se coloca em estado de espera;

3. Espera notificação do servidor para passar para estágio B (via TCP);
4. Tarefas realizadas:
   - Recebe notificação do servidor para passar para estágio B (via TCP);
   - Decompõe a velocidade inicial do tiro para posterior renderização do mesmo;
5. Usa da Física de lançamento de projéteis para calcular a posição do tiro.
6. Tarefas realizadas:
   - Troca turno;

- Gera dados do vento e envia-os ao servidor (via TCP)
- Envia notificação de que está pronto para passar pro próximo estagio (via TCP);

7. Espera uma notificação do servidor para passar pro estágio A (via TCP).
8. Recebe notificação do servidor para passar pro estágio A (via TCP).

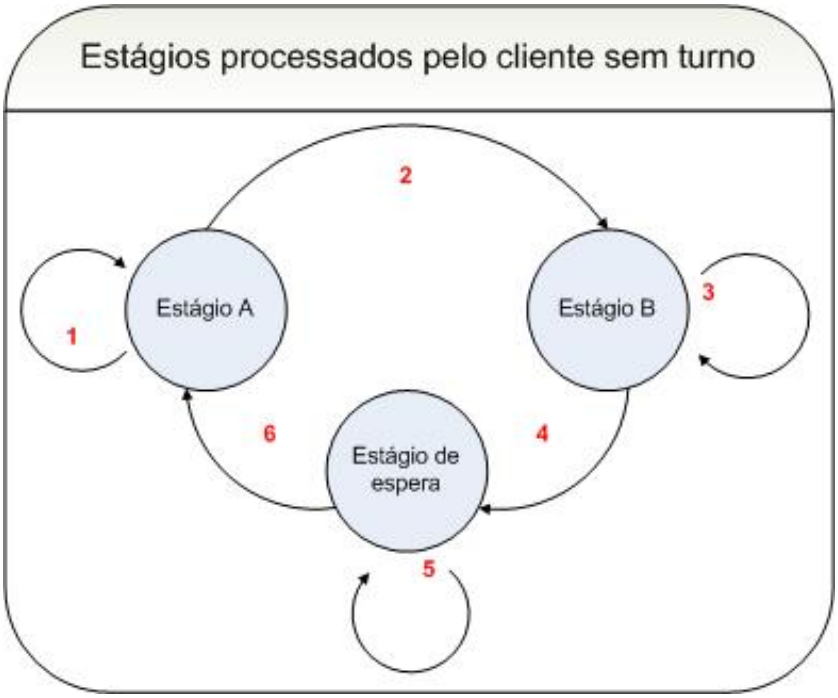

Figura 4.5: Estágios processados pelo cliente sem turno

Os itens enumerados a seguir dizem respeito aos estágios processados pelo cliente que não tem o turno. Eles têm correspondência na figura 4.5.

1. Dados recebidos (via UDP):
   - Posição do tanque;
   - Ângulo do canhão;
2. Tarefas realizadas:
   - Recebe do servidor (via TCP):
     - Informação do novo estágio;
     - Posição do tanque;

    - Ângulo do canhão;
    - Força do tiro;
  - Decompõe a velocidade inicial do tiro;

3. Usa da Física de laçamento de projéteis para calcular a posição do tiro.

4. Tarefas realizadas:

   - Troca turno;
   - Recebe dados do vento (via TCP);
   - Envia notificação de que está pronto para passar pro próximo estagio (via TCP);

5. Espera notificação do servidor para passar pro estágio A (via TCP)

6. Recebe notificação do servidor para passar para estágio A (via TCP)

4.4 Arquitetura de comunicação

A arquitetura de comunicação usada no jogo Panzer é a cliente-servidor. A disposição física desta arquitetura é bem flexível por usarmos aparelhos móveis, mas sua estrutura é basicamente como a mostrada na figura 4.6.

Nesta arquitetura o servidor tem um papel somente de informante, ele não faz qualquer processamento de dados. Ele mantém o estado do jogo que é controlado pelo cliente que tem o turno, como este cliente é quem sempre realiza as tarefas que alterar o estado do jogo, basta que este cliente forneça os dados alterados ao servidor para que o estado do jogo também seja mantido atualizado, pois a qualquer alteração de dados o servidor notifica ao cliente sem o turno a alteração.

4.4.1 Comunicação Cliente-Servidor

A comunicação entre a parte cliente e a parte servidora do jogo Panzer é feita usando os protocolos TCP e UDP. Baseando-se nos fluxogramas apresentados na figura 4.4 e 4.5 da seção 4.3.1, o protocolo TCP é usado em todos as transições dos estágios, exceto em uma onde o UDP é utilizado. O porque do uso das duas diferentes tecnologias de comunicação é explicado da seção 4.4.3. Será mostrado aqui como é feita esta comunicação a nível de mensagens.

Os dados do jogo que trafegam na rede e seus tipos estão na tabela 4.1:

Os dados enviados por TCP pelo cliente ao servidor variam de acordo com o tipo de mensagem. Para cada tipo de mensagem a um número inteiro associado que é o primeiro dado a ser enviado. Este número inteiro serve para que o servidor saiba de antemão qual a quantidade e seqüência de tipos de dados estará chegando.

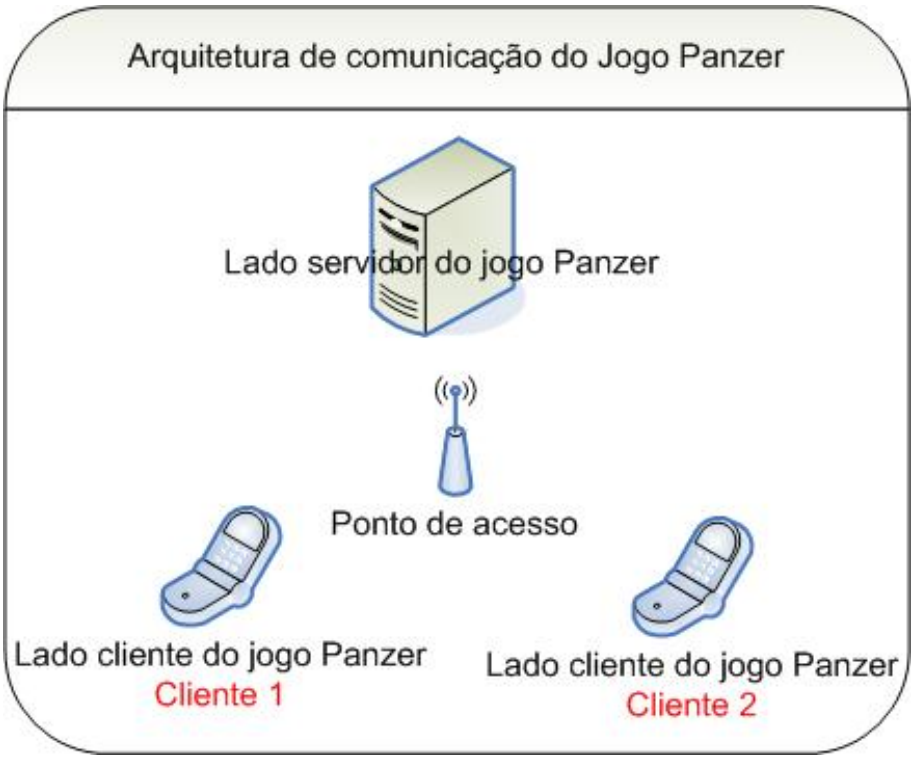

Figura 4.6: Arquitetura de comunicação do Jogo Panzer

Os tipos de mensagem, o que significam e os dados enviados/recebidos por eles estão na tabela 4.2.

Os dados enviados, via TCP, são empacotados e enviados um a um, de acordo com seu tipo de dado (Inteiro, Caracter, etc), em vez de serem colocados em uma mensagem única do tipo String. Isto é feito para não se ter o anterior e posterior trabalho de processamento das substrings. Devido a este modo como foi implementado o envio/recebimento das mensagens TCP, os dados recebidos pelo servidor são lidos da mensagem e recolocados em uma outra mensagem para envio.

Como os dados enviados por UDP só ocorrem em uma etapa do jogo (Ajustes dos parâmetros do tiro 4.4), temos assim sempre os mesmos dados a serem enviados, podendo a mensagem ter um formato único, o qual é apresentado na figura 4.7. Quando o servidor recebe está mensagem UDP do cliente com turno, ele nem a processa, somente a repassa (em forma bruta) para o cliente sem turno.

4.4.2 Criação cruzada de ouvintes TCP e UDP

A velocidade na comunicação entre a parte cliente e a parte servidora do jogo Panzer deve ser rápida e deve causar o mínimo possível de tráfego na rede. As

Dado	Tipo de dado
Número do tipo de mensagem	integer
Número de clientes conectados	integer
Número do tanque com turno	integer
Direção do vento	integer
Velocidade do vento	byte
Estágio	char
Posição x do tanque	integer
Posição y do tanque	integer
Posição x da ponta do canhão	integer
Posição y da ponta do canhão	integer
Ângulo do canhão	integer
Velocidade inicial do tiro	integer

Tabela 4.1: Dados do jogo que trafegam na rede e seus tipos

transações devem ser rápidas devido ao jogo trabalhar com movimentação em tempo real, a movimentação de um cliente não pode chegar atrasada no outro. O tráfego na rede deve ser o mínimo, devido a arquitetura física do jogo envolver dispositivos móveis de pouco poder de processamento (por volta de 10Mhz) e conexão lenta (9.600bps) - citado na seção 2.2.1. Devido a estes requisitos foi usado a criação cruzada de ouvintes TCP e UDP para tentar minimizar o tráfego na rede, e aumentar a velocidade de comunicação. Será explicado aqui como isto foi possível usando esta criação cruzada de ouvintes de dados.

Quando usamos uma conexão cruzada de ouvinte TCP ou UDP, tiramos a necessidade de um cliente ter que periodicamente perguntar ao servidor se uma dada informação chegou. Para ilustração, vejamos uma situação real que ocorre no jogo Panzer onde o cliente com o tanque número um quer saber se o tanque número dois já enviou os dados necessários a renderização do tiro. Imaginemos que não foi implementado uma conexão cruzada de ouvintes TCP, que só tivéssemos um ouvinte no lado servidor, teríamos a seguinte situação:

- Estando o cliente de tanque número dois com o turno, o mesmo estaria ajustando suas configurações de tiro para quando tivesse decido delas as enviasse para o servidor. Como o cliente com o tanque número um não sabe em qual momento seu adversário terá colocado os dados "definitivos"do tiro, para buscá-los, ele deve periodicamente perguntar ao servidor se seu adversário já o fez. Além do pesado tráfego causado por esta estratégia, para que ela ocorra deve haver um tráfego nos dois sentidos da rede, pois o cliente deve enviar um dado de pergunta ao servidor para receber a resposta (outro dado). Estas duas características são intrínsecas quando se tem apenas um socket ouvinte, causando um grande tráfego na rede.

Tipo de Mensagem	**Informa**	**Dado(s)enviado(s) pelo cliente**	**Dado(s) enviado(s) pelo Servidor**
Processa número de clientes	Cliente quer saber o número de clientes conectados, caso o mesmo não foi atingido, o servidor deve incrementá-lo;	Apenas o número inteiro do tipo de mensagem;	Número de clientes conectados
Número de clientes conectados	Cliente quer saber o número de clientes conectados no jogo;	Apenas o número inteiro do tipo de mensagem;	Número de clientes conectados;
Decremento de clientes	Cliente pede para decrementar a variável que guarda o número de clientes conectados ao jogo;	Apenas o número inteiro do tipo de mensagem;	Nenhum dado;
Atualização de turno	Cliente pede para atualizar o tanque que está com o turno, isto é feito trocando o número do tanque atual pelo número do outro;	Apenas o número inteiro do tipo de mensagem;	Nenhum dado;
Número do tanque com turno	Cliente quer saber o número do tanque que tem o turno;	Apenas o número inteiro do tipo de mensagem;	Número do tanque que está com o turno;
Receber dados da nuvem	Servidor está enviando os dados da nuvem;	Nenhum dado;	Número inteiro do tipo de mensagem, direção e velocidade do vento;
Enviar dados da nuvem	Cliente está enviando os dados da nuvem;	Número inteiro do tipo de mensagem, direção e velocidade do vento;	Nenhum dado;
Receber estágio	Servidor está enviando o estágio atual;	Nenhum dado;	Número inteiro do tipo de mensagem e estágio;
Enviar estágio	Cliente está enviando o estágio atual;	Número inteiro do tipo de mensagem e estágio;	Nenhum dado;
Receber dados da transição para estágio B	Cliente recebe os dados da transição para o estágio B. Estes são todos os dados necessários a realização do lançamento da bola de canhão;	Nenhum dado;	Número inteiro do tipo de mensagem, posição x e y do tanque, posição x e y da ponta do canhão, ângulo do canhão, velocidade inicial do tiro;
Enviar dados da transição para estágio B	Cliente envia os dados da transição para o estágio B. Estes são todos os dados necessários a realização do lançamento da bola de canhão;	Número inteiro do tipo de mensagem, posição x e y do tanque, posição x e y da ponta do canhão, ângulo do canhão, velocidade inicial do tiro;	Nenhum dado;

Tabela 4.2: Dados do jogo que trafegam na rede e seus tipos

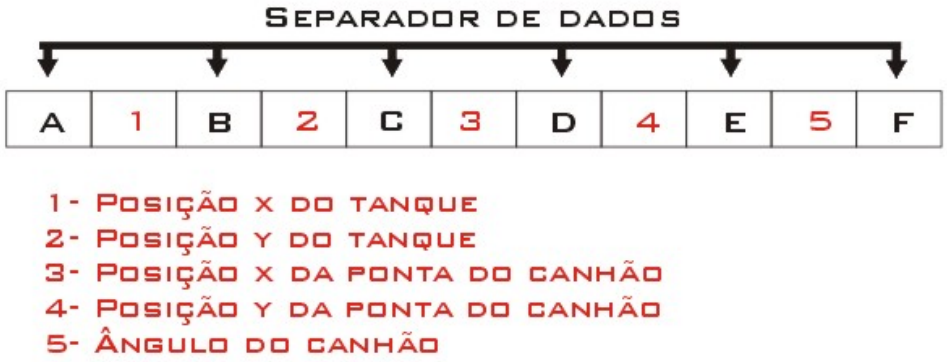

Figura 4.7: Formato da mensagem UDP, a qual envia os parâmetros do tiro

Este desnecessário tráfego pode ser resolvido criando uma conexão cruzada de ouvintes TCP. Assim, com a mesma situação anterior, porém com *sockets* ouvintes também nos clientes, teremos:

- Agora, como o cliente de tanque um tem um ouvinte TCP ele não precisa mais perguntar periodicamente ao servidor se seu adversário já enviou os dados do tiro, pois assim que o adversário o fizer, com a chegada dos dados o servidor imediatamente reenvia-os ao cliente de tanque número um. Também não há um tráfego de ida e vinda de dados, o cliente apenas recebe o dado de imediato.

O mesmo pode ser dito para conexões UDP. Assim, o jogo Panzer usa esta criação cruzada de ouvintes tanto em conexões TCP quanto em UDP.

4.4.3 Porque usar TCP e UDP?

As características destes dois protocolos de comunicação foram apresentadas na seção 2.3.2. Resumidamente, usa-se o protocolo TCP (orientado a conexão) quando se necessita de confiabilidade na conexão, ou seja, quando queremos ter certeza de que um dado chegou a seu destino, e se precisamos de rapidez em vez de confiança, optamos pelo protocolo UDP. Com o protocolo UDP (não-orientado a conexão) não há tratamento de chegada de um pacote a seu destino, assim um pacote é enviado e não se importa mais com ele a partir do momento que ele sai para a rede, isto torna a protocolo UDP muito mais rápido do que o TCP.

Na implementação do jogo Panzer a confiabilidade do protocolo TCP foi usada para dados que precisávamos ter certeza de que chegaria a seu destino, como a entrada do jogador ao jogo, os dados do tiro "confirmados", os dados do vento, etc. Se alguns destes dados não chegar ao outro cliente, o mesmo, ou não conseguirá

entrar no jogo, ou estará trabalhando com dados desatualizados no turno, podendo, por exemplo, no estágio de tiro um cliente estar rederizando um tiro que acerta o tanque enquanto que o outro que trabalha com o dado atualizando estaria errando o mesmo, causando um descompasso no decorrer do jogo; ou não conseguindo finalizar a sessão de jogo causando inconsistência no mesmo. Daí uma necessidade do uso do protocolo TCP.

O protocolo UDP foi usado no jogo Panzer no estágio de ajustes de parâmetros para o tiro. Neste estágio, o jogador que tem o turno, ao se movimentar com o tanque e o canhão, precisa enviar os dados que representam estes movimentos, como esta movimentação deve ser feita em tempo real para o outro cliente, tem-se que ter uma certa rapidez no tráfego destes dados, e daí o uso do protocolo UDP. A perda de algum pacote ocasiona apenas uma pequena pausa (*flic*) na movimentação, se usássemos TCP poderia ocorrer subsequentes pausas na movimentação e não algumas poucas que são o caso no UDP.

4.5 A lógica do jogo

Nesta seção será descrito como foi realizado a lógica de implementação dos estágios fundamentais do jogo Panzer, tais como: ajuste dos parâmetros do tiro (posicionamento do tanque, ajuste do canhão e da velocidade inicial do tiro), o desenho da movimentação do tiro (decomposição da velocidade inicial do tiro, desenho do lançamento e forças atuantes no lançamento), e a representação do vento.

4.5.1 O ajuste dos parâmetros do tiro

Para que o jogador possa atingir a bola de canhão em seu adversário ele deve anteriormente ajustar alguns parâmetros, como posição do tanque, ângulo do canhão, e velocidade inicial do tiro. Neste estágio, o jogador que tem o turno é quem faz estes ajustes, o que não tem o turno apenas observa a movimentação do adversário. Será apresentado nas seções seguintes como o ajuste destes parâmetros são implementados.

Posicionando o tanque

O posicionamento (movimentação) do tanque é realizado quando o jogador aperta os botões quatro (movimento para esquerda) e seis (movimento para direita). Quando o jogador aperta o botão quatro um evento é gerado e tratado para mover o tanque para a esquerda. Este tratamento consiste de decrementar a posição x do tanque de um pixel, assim na próxima vez que a tela for redesenhada teremos a impressão de que o tanque moveu-se para esquerda, pois a imagem do tanque foi redesenhada em uma nova posição. O mesmo ocorre quando o usuário aperta o botão seis, com a diferença de a posição x do tanque ser incrementada de um pixel.

O limite das posições dos tanques são a montanha e o canto da tela (canto esquerdo para o tanque dois e canto direito para o tanque um). Quando o tanque um colide com a montanha sua posição é incrementada de um fazendo com que ele recue para a direita, e quando sua posição ultrapassa o limite da tela seu pixel não é incrementado, permanecendo o tanque estável. O mesmo é aplicado ao tanque dois só que em modo inverso. É feito o incremento ou decremento da posição dos tanques quando estes colidem com a montanha para dar a impressão (em imagem) de que o tanque realmente trombou com a montanha, uma reação de impacto.

Ajustando o canhão

Os valores limites do ângulo do canhão do tanque são zero (0^o) e noventa (90^o) graus, o aumento ou diminuição deste ângulo é feito de cinco em cinco unidades. O ajuste (movimentação) do canhão é realizado quando o jogador aperta os botões dois (aumenta o ângulo) e oito (diminui o ângulo). Quando o jogador aperta o botão dois um evento é gerado e tratado para aumentar o ângulo do canhão do tanque. Este tratamento consiste em incrementar o valor do ângulo de cinco graus e de representar este aumento na tela (representação gráfica).

A representação gráfica deste aumento/diminuição é feita calculando as variações da nova posição da ponta do canhão nos eixos x e y. Estas variações são somadas à antiga posição da ponta do canhão nos dois eixos. Como o canhão é representado por uma reta, devemos passar as posições x e y de sua base e ponta para que possa ser desenhado. Assim, devemos apenas redesenhar o canhão com sua nova posição da ponta, uma vez que a posição da base só se altera com a movimentação do tanque.

O cálculo das variações da ponta do canhão em x e y é realizado da seguinte maneira:

- Para o tanque um, que fica no canto direito da tela:

    1. O ângulo do canhão é guardado em uma variável auxiliar;
    2. O novo ângulo é incrementado de cinco;
    3. A nova posição da ponta do canhão em x é a soma da posição anterior da ponta em x com a variação causada pelo aumento do ângulo, o qual é calculado pela formula:

    $$v = (cos\theta_1)c - (cos\theta_2)c$$

    Onde:

    - v - Variação da posição da ponta do canhão;
    - θ_1 - Ângulo anterior a mudaça;
    - θ_2 - Novo ângulo;

- c - Comprimento do canhão;

4. A nova posição da ponta do canhão em y é a subtração da posição anterior da ponta do canhão em x com a variação causada pelo aumento do ângulo, o qual é calculado pela formula:

$$v = (cos(90 - \theta_1))c - (cos(90 - \theta_2))c$$

Onde:

- v - Variação da posição da ponta do canhão;
- θ_1 - Ângulo anterior a mudaça;
- θ_2 - Novo ângulo;
- c - Comprimento do canhão;

- Para o tanque dois, que fica no canto esquerdo da tela é feito o mesmo processamento, a diferença está que, em vez de somar a variação da ponta do canhão no eixo x, ela é subtraída.

O mesmo processo é realizado para a diminuição do ângulo, porém de forma inversa.

O processamento da variação gráfica é feito pelo cliente que tem o turno, e enviado ao cliente sem turno, que tem somente o trabalho de redesenho do canhão na tela.

Ajustando a velocidade inicial

O ajuste da velocidade inicial do lançamento (tiro) é realizado usando os botões um (diminui o valor) e três (aumenta o valor). O valor da velocidade inicial do lançamento fica entre zero e vinte, o aumento ou diminuição deste valor é feito de um em um. Para ajustar aos valores reais do jogo, devido a proporções dimensionais do cenário, este valor é multiplicado por uma constante para obter um valor condizente. Quando o jogador aperta o botão três um evento é gerado e tratado para aumentar o valor da velocidade inicial do lançamento. Este tratamento consiste em incrementar o valor da velocidade inicial. O inverso é feito quando a velocidade inicial é diminuída pelo jogador. A representação da velocidade inicial é realizada desenhando um retângulo que tem uma altura fixa e uma largura que é proporcional ao valor da velocidade inicial passado pelo usuário.

4.5.2 O tiro (lançamento de projéteis)

O tiro realizado pelo tanque é baseado na física de lançamento de projéteis, onde é necessário se ter um ângulo e uma velocidade inicial de lançamento. O objeto lançado em nosso caso é uma bola de canhão dispara pelo tanque, a qual deve ser

lançada usando um movimento parabólico para que possa conseguir chegar até o tanque adversário. Sem um movimento parabólico não seria possível ultrapassar o obstáculo que separa os dois tanques, daí o uso de lançamento de projéteis na construção do jogo.

Quando este estágio é iniciado os dois clientes já estão com todos os dados necessários a renderização do tiro em mãos. Aqui, não se tem tráfego na rede, todo o cálculo da posição da bola de canhão é realizado pelo próprio cliente segundo as fórmulas de lançamento de projéteis. Todas as etapas do desenho da movimentação do tiro serão apresentadas nesta seção.

Decomposição da velocidade inicial do tiro

A decomposição da velocidade inicial do tiro nos eixos x e y é uma das etapas executadas pelo jogo para realizar a movimentação parabólica do tiro. Como o usuário passa apenas a velocidade inicial e ângulo do tiro ao jogo, o mesmo deve pegar está velocidade inicial e decompo-la no eixos x e y, com base no ângulo do tiro, para que o lançamento de projéteis possa ser realizado. Para fazer esta decomposição o jogo usa da física de lançamento de projéteis (citado na seção 2.5).

A decomposição é realizada a partir dos dados da velocidade inicial (V_i) e ângulo do tiro (θ) passados pelo usuário. A velocidade inicial no eixo X (V_x) é obtida multiplicando o coseno do ângulo pela velocidade inicial do tiro. A velocidade inicial no eixo Y (V_y) é obtida multiplicando o seno do ângulo pela velocidade inicial do tiro.

$$V_x = V_0 cos\theta$$

$$V_y = V_0 sen\theta$$

A próxima etapa faz o desenho do lançamento da movimentação da bola de canhão que será explicada na seção seguinte.

O desenho do lançamento

O desenho no cenário da movimentação da bola de canhão é realizado redesenhando de tempos em tempos a bola de canhão em sua nova posição. Este redesenho é realizado de cinqüenta em cinqüenta milisegundos, o que dá vinte quadros por segundo. O redesenho da bola de canhão, vinte vezes por segundo, da a impressão de que este movimento é um movimento de lançamento de projétil real.

A bola de canhão é desenha na tela com base em suas posições nos eixos x e y. Estas posições são com o passar do tempo alteradas devido às influências das velocidades inicias em x e y (citado na seção 2.5). A influencia das forças atuantes (seção 4.5.2) nas velocidades inicias da bola de canhão caraterizam o movimento

de lançamento de projéteis. Na próxima seção será falada em detalhes a influência das forças atuantes nas velocidades em x e y da bola de canhão.

O desenho do movimento da bola de canhão termina quando o quadrado ao qual a imagem da bola se encontra colide, ou com um pixel da imagem da montanha (não com o quadrado ao qual a imagem está, pois a colisão é feito a nível de pixel [6]), ou com o quadrado ao qual a imagem de um dos tanques está, ou quando o ponto superior esquerdo da imagem da bola de canhão passa do nível do solo [7].

Forças atuantes no lançamento

A influencia das forças gravidade e vento nas velocidades inicias (V_x e V_y) da bola de canhão caraterizam o movimento de lançamento de projéteis. A força da gravidade atua no eixo y, causando uma desaceleração da velocidade da bola neste eixo. A força vento atua no eixo x, e pode causar tanto uma aceleração como uma desaceleração na velocidade da bola neste eixo, se o sentido do vento for o mesmo da velocidade da bola então haverá uma aceleração da velocidade neste eixo, se os sentidos forem opostos teremos uma desaceleração da velocidade neste eixo.

A força da gravidade é uma constante, não variando durante todo o desenrolar do jogo. Já a força e sentido do vento variam a cada turno do jogo, e são aleatoriamente geradas, podendo a força variar na proporção de zero (sem vento) a vinte (máximo valor do vento), e o sentido sendo ou -1 (um negativo) ou +1 (um positivo). O valor negativo do vento lhe dá uma velocidade no sentido para esquerda e o positivo no sentido contrário (para direita).

4.5.3 A representação do vento

A velocidade e sentido do vento são características da força vento que o jogador não pode desprezar, pois elas influenciam muito nos ajustes de tiros feitos pelo jogador. Os valores dessas características do vento mudam a cada turno e para que o jogador possa percebê-la é desenhada uma nuvem que representa esta movimentação. Não foram colocados os valores da nuvem diretamente devido à exatidão dos mesmos levar o usuário a memorizar qual a melhor configuração de tiro para tais instâncias de valores do vento. Assim, tais valores podem ser percebidos pela movimentação da nuvem, mas não com a mesma precisão de quando se tem posse dos valores exatos, não permitindo assim que o usuário memorize a melhor situação. Desta forma o usuário deve usar de sua percepção para prever a melhor configuração para o tiro.

[6] Um recurso da classe "GameCanvas", onde existe uma função que verifica a colisão entre duas imagem. Está colisão pode ser feita a nível de pixel ou não. A nível de pixel é verificado se as duas imagens colidem algum de seus pixels opacos, e sem a colisão a nível de pixel é verificado se o quadrado ao qual as imagens encontram colidem.

[7] Se for um solo de relevo regular, e estando o solo a um nível y de pixels do canto superior esquerdo da tela, então verifica se a bola ultrapassou este nível.

A geração de números aleatórios pelo computador não é totalmente fortuita, pois uma seqüência de números será sempre a mesma se gerada a partir de certa semente. Ou seja, se usarmos a todo o momento a mesma semente, a seqüência gerada será sempre a mesma. Assim, para que esta seqüência seja realmente aleatória, o jogo Panzer, na sua criação, instância uma semente baseada num número inteiro que é a milésima parte de um segundo. Posteriormente, a cada turno, o próximo número aleatório desta semente instanciada é obtido, e assim usado para modular a velocidade e direção do vento.

Capítulo 5

Considerações Finais

Este capítulo apresenta as considerações finais do estudo da utilização da tecnologia J2ME no desenvolvimento de uma aplicação distribuída multi-usuário e também algumas sugestões de melhoria da aplicação para trabalhos futuros. A seção 5.1 descreve os resultados alcançados, bem como as dificuldades e facilidades encontradas com o desenvolvimento da lógica do jogo e com o uso da tecnologia J2ME. A seção 5.2 cita algumas sugestões para a melhoria da aplicação.

5.1 Comentários

O objetivo deste trabalho foi o estudo da utilização da tecnologia wireless no desenvolvimento de aplicações distribuídas multi-usuário, este estudo foi realizado com a implementação de um jogo com as seguintes características:

- Utilização de uma tecnologia wireless - Foi utilizada a tecnologia de desenvolvimento J2ME (seção 3.1);
- Multi-usuário - O *game* deve ser jogado por um mínimo e máximo de dois usuários (seção 3.1).
- Distribuída - O jogo utiliza a arquitetura de comunicação cliente-servidor (seção 4.4).

Assim, a aplicação com todos os requisitos propostos foi desenvolvida sobre um jogo de estratégia denominado Panzer. As dificuldades e facilidades encontradas neste desenvolvimento são descritas a seguir.

As dificuldades foram as seguintes:

- A grande dificuldade encontrada no desenvolvimento foi devido a aplicação ser destinada a dispositivos móveis, que são aparelhos com reduzido poder de processamento, memória, e conexão lenta (seção 2.2.1). Com isto, foi

necessário uma minuciosa análise do código, visando a otimização do uso da memória e do processamento.

- Uma aplicação distribuída exige uma constante troca de informações entre suas partes. Isto também foi um desafio ao se considerar uma aplicação destinada a dispositivos móveis.

- Dificuldade no início do projeto em decidir qual protocolo usar na comunicação entre as partes cliente e a parte servidora, por razões de falta de conhecimento dos protocolos de comunicação existentes. No começo, todas as etapas do jogo usavam do protocolo TCP para fazer esta comunicação, posteriormente no estágio de ajuste dos parâmetros do tiro verificou-se que a utilização do protocolo UDP era mais apropriada por se tratar de uma movimentação em tempo real (isto foi discutido na seção 4.4.3).

- A inexistência do tipo de dado ponto flutuante na versão 1.0 da *configuration* CLDC levou ao uso da versão 1.1 no desenvolvimento do projeto. Devido à pequena resolução da tela foi necessário trabalhar com números de ponto flutuante para trazer a movimentação gráfica do jogo o mais perto possível da realidade física.

- Bolar uma lógica para fazer a sincronização do jogo foi uma dificuldade por não encontrar material a respeito do assunto, foi realizado de modo experimental.

A utilização da tecnologia J2ME se mostrou muito adequada na construção do jogo Panzer. O *profile* MIDP2.0 facilitou muito devido a utilização das classes localizadas no pacote *javax.microedition.lcdui.game*, as quais são direcionadas ao desenvolvimento de jogos. Uma destas classes que se destaca é a *Sprite*, por conter um método de detecção de colisão entre uma imagem e outra.

5.2 Trabalhos futuros

O jogo Panzer foi desenvolvido a partir de alguns requisitos básicos, assim muitos outros surgiram com o decorrer do desenvolvimento do mesmo. Estes novos requisitos podem ser acrescidos à aplicação para melhora-lá, de forma a tornar viável sua comercialização. A seguir são listadas estas melhorias:

- Devido ao jogo Panzer se mostrar em uma visão de perfil, ele tem um limite de dois jogadores duelando-se ao mesmo tempo (discutido em 3.1). Uma outra versão de jogo poderia ser feita tendo uma visão área do campo de batalha.

- Quando a bola de canhão atinge o obstáculo (Ex.: montanha), este não sofre qualquer alteração. Assim, uma melhoria seria a alteração parcial da forma da montanha à medida que esta fosse atingida pelas bolas de canhão.

- O jogo se encontra com poucas opções de campos de batalha, cenários com uma aparência diferente e também com formas de obstáculo diferentes trariam mais empolgação aos combatentes. Por exemplo, a criação de solos irregulares, com diferenças de desnível; criação de obstáculos de tipos diferentes: um precipício, um rio;

- Uma maior movimentação do campo de batalha, o cenário se encontra fixo. Quando a bola de canhão é disparada a tela não a acompanha se esta sair do cenário.

- A inclusão de efeitos sonoros ao jogo. Quando a bola de canhão fosse disparada, quando ela atingisse um obstáculo ou o adversário, quando o tanque fosse movimentado, músicas de fundo, etc.

Estas são as propostas que surgiram com o desenvolvimento do trabalho, novas surgirão com o desenvolvimento dos trabalhos futuros, e assim surgem também as expectativas de tornar aplicação um produto que possa realmente ser comercializado.

Referências Bibliográficas

[DEITEL, 2001] DEITEL, H. M. DEITEL, P. J. *Introduction to the Internet, World Wide Web and Wireless Communications*. 25 de junho de 2001.

[NOKIA01, 2003] NOKIA. *Multi-Player MIDP Game Programming*. http://forum.nokia.com. 29 de outubro 2003.

[LAMOTHE, 1999] LAMOTHE, André. *Tricks of the Windows Game Programming Gurus*. 1999.

[ARNOLD, 2002] ARNOLD, Martin. *Microsoft sees Xbox playing by different rules*. http://www.ft.com. 25 de janeiro de 2002

[RDJ, 1996] Orfali, Robert. Harkey, Dan. Edwards, Jeri. *The Essential Cliente/Server Survival Guide - Second Edition*. 1996

[JAMES, 2002] JAMES, P. White. DAVID, A. Hemphill. *Java 2 Micro Edition - Java in small things*. 2002.

[USP, 2004] Site USP/SC. http://educar.sc.usp.br/fisca . Acessado em 15 de dezembro de 2004

[SUM, 2003] *Over The Air User Initiated Provisioning Recommended Pratices for the Mobile Information Device Profile*. http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf. Acessado em 05 de junho de 2003