UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

DESENVOLVIMENTO DE JOGOS PARA DISPOSITIVOS MÓVEIS UTILIZANDO MIDP: IMPLEMENTAÇÃO DO JOGO TETRIS

CRISTIANO ROSSETTO

BLUMENAU
2005

2005/1-10

CRISTIANO ROSSETTO

DESENVOLVIMENTO DE JOGOS PARA DISPOSITIVOS MÓVEIS UTILIZANDO MIDP: IMPLEMENTAÇÃO DO JOGO TETRIS

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

Prof. Mauricio Capobianco Lopes - Orientador

BLUMENAU
2005

2005/1-10

DESENVOLVIMENTO DE JOGOS PARA DISPOSITIVOS MÓVEIS UTILIZANDO MIDP: IMPLEMENTAÇÃO DO JOGO TETRIS

Por

CRISTIANO ROSSETTO

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

Presidente:	Prof. Mauricio Capobianco Lopes, Msc. – Orientador, FURB
Membro:	Prof. Paulo César Rodacki Gomes, Dr. – FURB
Membro:	Prof. Marcel Hugo, Msc. – FURB

Blumenau, 05 de julho de 2005.

Dedico este trabalho aos meus pais, que me deram a base, para a construção da minha personalidade.

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

À toda minha família, que sempre me apoiou nas decisões mais importantes de minha vida e me ajudou a não desistir, especialmente ao meu irmão Fabiano, que muito me ajudou no início da faculdade.

À minha esposa Aline, pelo amor e carinho, que mesmo não gostando de minha ausência durante o desenvolvimento deste trabalho, soube aceitar.

Aos meus amigos, que entenderam as minhas faltas e me apoiaram no desenvolvimento deste.

Ao meu orientador, Mauricio Capobianco Lopes, por ter acreditado na conclusão deste trabalho.

O gênio é um por cento de inspiração e noventa e nove de transpiração.

Thomas A. Edison

RESUMO

Este trabalho apresenta um estudo sobre o desenvolvimento de jogos para dispositivos móveis, principalmente celulares, mostrando as principais tecnologias de desenvolvimento de jogos com ênfase em J2ME e MIDP. Também são apontados os principais problemas no desenvolvimento de jogos multiplataforma e as técnicas aplicadas para a solução dos mesmos. Apresenta as ferramentas utilizadas desde o projeto até a instalação em um dispositivo real. Como exemplo é apresentado o desenvolvimento do jogo Tetris.

Palavras-chave: J2ME. Jogos. Dispositivos móveis. MIDP. Tetris. CLDC. Java.

ABSTRACT

This work presents a study of game development for mobile devices, mainly cell-phones, showing the main technologies of game development with emphasis in J2ME and MIDP. Also main problems are pointed in the developing of multiplatform games and the techniques that applied regarding the solution of the same ones. It presents the tools used since the project until the installation in a real device. Such as the development of the Tetris game.

Key-Words: J2ME. Games. Mobile devices. MIDP. Tetris. CLDC. Java.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Com a enorme quantidade de dispositivos móveis, principalmente celulares, que suportam a tecnologia Java, a procura por jogos para esses "brinquedos modernos" vem crescendo de forma exponencial. Segundo o Jornal da Globo (2004) o mercado mundial de jogos cresce a uma velocidade de 25% ao ano. No Brasil, estima-se que o faturamento anual seja de aproximadamente 100 milhões de reais.

Jogos para dispositivos móveis, como celulares, costumam ser bem mais simples que jogos para PC's ou consoles atuais, cuja criação muitas vezes exige investimentos de milhões de dólares, equipes de dezenas de pessoas e anos de trabalho (SABINO, 2003). Devido às limitações de hardware destes aparelhos, os jogos tendem a ser menos complexos, tornando-se possível o desenvolvimento de um jogo por equipes pequenas e com um custo muito menor.

Entretanto, desenvolver um jogo não é uma tarefa trivial, pois envolve o conhecimento de diversas áreas da computação. Em ambientes restritos, com limitações de memória e processamento, esta tarefa é ainda mais difícil, uma vez que operações como detecções de colisão, animação de *sprites* e renderização, são necessárias na maioria dos jogos.

Pode se desenvolver jogos para dispositivos móveis usando várias tecnologias. As mais usuais segundo Almeida, Loureiro e Nogueira (2005) são: *Binary Runtime Enviroment for Wireless* (BREW), *Execution Engine* (ExEn), Mophun e *Java 2 Micro Edition* (J2ME). Entre elas, J2ME se destaca sendo a plataforma mais suportada por dispositivos móveis no mercado.

Como será visto em maiores detalhes neste trabalho, J2ME possui várias extensões, ou API's, cada uma voltada para um certo grupo de aparelhos eletrônicos. Entre elas a *Mobile Information Device Profile* (MIDP) é a principal e mais difundida, pois é neste grupo que se

encontram a maioria dos aparelhos de telefone celular.

Portanto, será abordada neste trabalho a tecnologia necessária para o desenvolvimento completo de um jogo multiplataforma para dispositivos móveis, mais especificamente para dispositivos que implementam o MIDP na versão 2.0.

Também serão apontados os principais problemas no desenvolvimento de jogos multiplataforma e as técnicas aplicadas para a solução dos mesmos, desde o projeto até a instalação, e como exemplo é desenvolvido um estudo de caso baseado no jogo Tetris.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver o jogo Tetris para dispositivos móveis com ênfase em celulares, utilizando a tecnologia MIDP.

Os objetivos específicos do trabalho são:

a) pesquisar técnicas de desenvolvimento de jogos;

b) aplicar a tecnologia J2ME, no desenvolvimento de jogos multiplataforma;

c) dividir os conhecimentos adquiridos na concepção deste trabalho com a comunidade Java e acadêmica, publicando a sua implementação e os resultados nos fóruns da comunidade Java, já que a maioria das bibliotecas utilizadas no desenvolvimento de jogos atualmente são proprietárias.

1.2 ESTRUTURA DO TRABALHO

O seguinte trabalho está organizado em cinco capítulos.

Após o capítulo inicial, que expôs uma visão geral sobre os assuntos e os objetivos que estão sendo abordados neste trabalho, o segundo capítulo apresenta alguns conceitos do

desenvolvimento de jogos para dispositivos móveis, e as principais tecnologias atualmente usadas neste segmento.

O terceiro capítulo trata exclusivamente de J2ME, conceitos e características. Neste capítulo também é abordado o perfil MIDP e sua API específica para o desenvolvimento de jogos.

O quarto capítulo apresenta o desenvolvimento do jogo Tetris, sua especificação através de diagramas da UML, características de sua implementação e seu funcionamento através de algumas telas.

O quinto e último capítulo apresenta as conclusões obtidas no desenvolvimento deste trabalho e algumas sugestões para a sua continuidade.

2 DESENVOLVIMENTO DE JOGOS PARA DISPOSITIVOS MÓVEIS

O desenvolvimento de jogos para celulares é bastante recente. Inicialmente os jogos eram criados pelos fabricantes dos celulares e cada dispositivo saia de fábrica com um conjunto de jogos predefinidos. Segundo Barros (2003) os primeiros jogos criados eram simples, tais como *Snakes* e Memória.

Com a evolução tecnológica tornou-se possível instalar aplicações nos celulares. Isto mudou radicalmente o futuro dos jogos nestes dispositivos, pois agora não é só o fabricante de celulares que desenvolve jogos, mas outros programadores e empresas de desenvolvimento de jogos passaram a investir nesta nova plataforma.

2.1 DIFERENÇA ENTRE JOGOS PARA DISPOSITIVOS MÓVEIS E JOGOS PARA CONSOLES E PC'S

Segundo Nascimento (2003), os jogos desenvolvidos para dispositivos móveis são limitados pelas características dos próprios dispositivos. Comparando com o desenvolvimento para consoles ou para PC's, os aparelhos celulares, por exemplo, representam um grande desafio para os programadores devido às grandes restrições existentes quando comparadas com o desenvolvimento comum de jogos. Outro aspecto complicador é a variedade de modelos com telas e recursos diferentes e as diferenças de implementação das plataformas de cada um deles. Estas diferenças e restrições geram a necessidade da adoção de soluções criativas e otimizadas.

Além das restrições de recursos, outra grande característica que limita os jogos desenvolvidos para aparelhos celulares é a forma de interação com o jogador. Os aparelhos não foram feitos para jogar e por isso o tamanho de tela, as cores apresentadas, a falta de suporte a som dificultam o estabelecimento de uma boa relação entre o jogo e o jogador. A

entrada de dados também é muito prejudicada por causa do teclado reduzido e da falta de suporte para o tratamento de mais de uma tecla pressionada ao mesmo tempo.

2.2 TECNOLOGIAS

Segundo Almeida, Loureiro e Nogueira (2005), atualmente, a plataforma mais difundida para o desenvolvimento de jogos para dispositivos móveis é J2ME, a versão simplificada da linguagem Java que é suportada, pelo menos nominalmente, por todos os fabricantes de aparelhos celulares.

Porém os jogos podem ser desenvolvidos usando várias outras plataformas. Uma breve abordagem das principais plataformas é mostrada em seguida.

2.2.1 BREW

Binary Runtime Environment for Wireless (Ambiente de desenvolvimento Binário para Dispositivos Móveis), é a plataforma da Qualcomm para desenvolvimento de aplicativos, inclusive jogos, para dispositivos móveis. Segundo Barros (2003), as aplicações são desenvolvidas em C/C++, compiladas e rodam em um chip separado do processador principal do dispositivo, o que possibilita uma boa performance.

BREW possui um bom suporte para o desenvolvimento de jogos. É possível desenhar primitivas gráficas e imagens. Também têm suporte a som, criação de temporizadores, conectividade e persistência de dados, conforme Qualcomm (2005).

Além da API, a proposta da Qualcomm foi estabelecer um completo processo para o desenvolvimento de aplicações para aparelhos celulares que possibilitasse criar um canal entre o desenvolvedor e as operadoras de telefonia dando garantias de qualidade dos aplicativos

construídos e do pagamento pela comercialização dos mesmos.

2.2.2 ExEn

Execution Engine da In-Fusion, foi a primeira plataforma dedicada à execução e desenvolvimento de jogos para dispositivos móveis voltada ao mercado massivo europeu, conforme Amaro (2003). A In-Fusion, empresa que hoje lidera o mercado europeu de jogos para dispositivos móveis (PHONEGAMEREVIEW.COM, 2004), fornece toda a infra-estrutura necessária à publicação, desenvolvimento e licenciamento de jogos para a ExEn.

Segundo Almeida, Loureiro e Nogueira (2005), as aplicações ExEn são escritas na linguagem Java. Ainda segundo estes autores, a plataforma conta com uma máquina virtual, cujo desempenho pode superar em até 30 vezes uma genérica, e permite, além da execução, o *download* de jogos através das operadoras telefônicas. O módulo gráfico da plataforma suporta até 65K de cores e contém, além dos mecanismos básicos para a construção de jogos em 2D, pacotes com suporte a gráficos 3D que inclui funções para operações em vetores e matrizes 3D.

2.2.3 Symbian OS

Symbian OS é um sistema operacional aberto desenvolvido pela empresa Symbian, um consórcio da Nokia, Motorola, Sony-Ericsson e outras gigantes da indústria de comunicação sem fio, segundo Nascimento (2003). Por ser um sistema operacional, não pode ser comparado diretamente com outras plataformas. Mas não se pode ignorar sua importância no desenvolvimento de jogos.

As aplicações para Symbian OS são desenvolvidas em C++ e possuem acesso a todos

os recursos do dispositivo móvel. A API do Symbian OS fornece um *framework* de desenvolvimento de aplicações, desta forma, é necessário apenas herdar algumas classes, reimplementado alguns métodos.

2.2.4 Mophun

Solução desenvolvida pela empresa Synergenix, que visa principalmente aproximar o desenvolvimento de jogos para dispositivos móveis do desenvolvimento de jogos tradicional. Segundo Obigo (2005), é considerada a mais avançada e completa solução de desenvolvimento de jogos para dispositivos móveis.

A plataforma, cujos jogos são desenvolvidos em C ou C++, ocupa 50Kb de memória ROM, necessitando de mais 100Kb quando incluída uma biblioteca 3D otimizada. Seu módulo gráfico possui funções de renderização que estão disponíveis tanto para gráficos em 2D quanto em 3D. A plataforma ainda provê um bom suporte a multimídia.

Porém, mesmo com todo o avanço, a plataforma está disponível em uma quantidade muito pequena de dispositivos, ficando seu mercado bastante comprometido se comparada com outras plataformas.

2.2.5 Comparação Entre as Tecnologias

Para melhor detalhar as diferenças entre as tecnologias apresentadas acima, Almeida, Loureiro e Nogueira (2005) fazem uma análise comparativa com os seguintes parâmetros:

a) custo para desenvolvimento: indica qual o custo que deve ser arcado pelo desenvolvedor para que seja possível iniciar o desenvolvimento de um jogo;
b) modelo de negócios: neste ponto é analisado que meios são disponibilizados para o

desenvolvedor publicar seu jogo e a que custo esta operação é realizada;

c) carga: mostra a quantidade de recursos que a plataforma utiliza do dispositivo, assim como a quantidade que é disponibilizada pela mesma para a execução de jogos;

d) linguagem de desenvolvimento: indica a linguagem utilizada como base para o desenvolvimento na plataforma;

e) segurança: indica o grau de segurança da plataforma contra falhas nos aplicativos em execução, aplicativos mal-intencionados, etc.;

f) suporte a elementos de jogo: quais os tipos de elementos de jogo suportados pelas plataformas, tais como a manipulação de gráficos 2D e 3D, reprodução de vídeos, sons, etc.

Os resultados da comparação são apresentados no Quadro 1. Lembrando que o Symbiam OS não pode ser comparado, pois trata-se de um sistema operacional.

	J2ME	**BREW**	**ExEn**	**Mophun**
Custo para desenvol-vimento	Completamente gratuito.	- Pago para disponibilização do jogo; - IDE's pagas.	Completamente gratuito.	- Plataforma gratuita; - IDE's pagas.
Modelo de negócios	Desenvolvedor responsável por todas as etapas, do desenvolvimento à distribuição.	Empresa realiza testes e disponibiliza as aplicações.	Programas específicos dão suporte para implementação, testes e distribuição.	Ferramentas distribuídas com a plataforma auxiliam na distribuição.
Carga	- Interpretado; - 128Kb à 512Kb de memória (RAM e ROM).	- Aplicativos compilados; - 90Kb de ROM.	- Rápido, pois utiliza a API nativa do dispositivo; - 100Kb de ROM e 32K de RAM.	- Processamento rápido; - 50Kb à 150Kb na memória.
Linguagem de desenvol-vimento	Java	C++	Java	C / C++
Segurança	- Os jogos não possuem o funcionamento garantido; - A plataforma protege o dispositivo de aplicações mal intencionadas.	- Aplicativos testados pela Qualcomm e assinados digitalmente; - Previne os dispositivos contra falhas nos jogos e ações maliciosas.	- Previne os dispositivos contra falhas nos jogos e ações maliciosas.	- Os jogos são assinados digitalmente e protegidos contra pirataria; - Cada jogo é exaustivamente testado antes de ser liberado.
Suporte	- Utilização de camadas facilitando a inserção de elementos visuais; - Verificação automática de colisão entre *sprites*; - Construção simplificada de labirintos com a classe *TiledLayer*.	- Utilização de arquivos de recursos facilita a gerência de elementos do jogo; - Maior desempenho ao utilizar a linguagem nativa.	- Várias operações nativas para tratamento de gráficos 3D.	- Detecção automática entre *sprite* e *background*; - 3 módulos providos com o SDK: *sprites*, *background* e *collision*; - API 3D provendo vários tipos de materiais e aplicações de luz.

Fonte: Almeida, Filho e Nogueira (2005, p. 7).

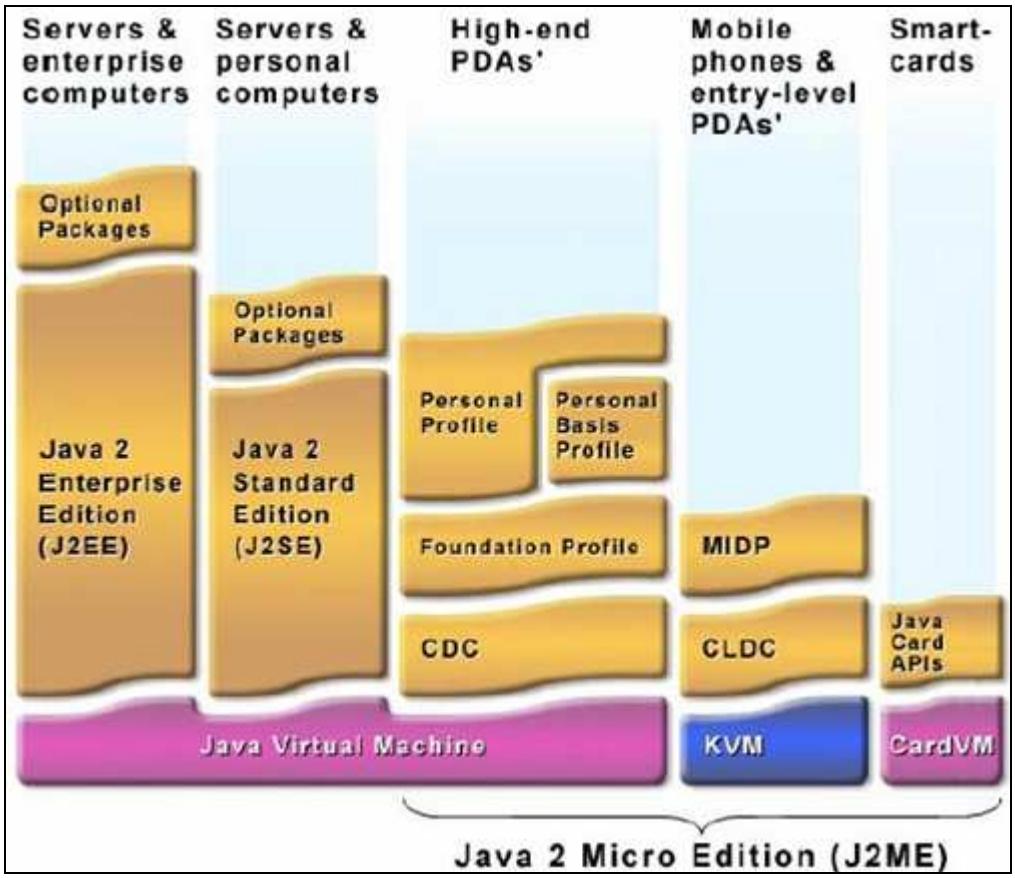
Quadro 1 – Comparação entre as plataformas de desenvolvimento de jogos

Além desses dados, a escolha da plataforma para o desenvolvimento de um jogo depende ainda da estratégia a ser adotada pelo desenvolvedor, suas necessidades e até mesmo seu orçamento. Como já mencionado a plataforma J2ME possui o maior número de aparelhos no mercado. Sendo assim, pode-se verificar melhor a portabilidade do jogo a ser desenvolvido neste trabalho. J2ME ainda possui a vantagem de possuir mais de uma ferramenta gratuita para desenvolvimento.

3 J2ME

Segundo Miranda (2002a), J2ME é um conjunto de especificações que define uma *Java Virtual Machine* (JVM), ou seja, um conjunto de API's especializadas para pequenos dispositivos. Esta arquitetura foi criada para ser simples e enxuta, retirando-se partes do *Java 2 Standard Edition* (J2SE) não aplicáveis ou características difíceis de suportar em certos dispositivos, como movimento e tratamento de eventos do *mouse* e algumas bibliotecas para programação gráfica.

Conforme Caldeira (2002) o J2ME é dividido em *Configurations* (Configurações) e *Profiles* (Perfis). Estes oferecem informações sobre as APIs de diferentes famílias de equipamentos (Figura 1).



Fonte: Carniel e Teixeira (2003, p. 2).

Figura 1 – Arquitetura da plataforma Java

Uma configuração é uma API que contém os requisitos mínimos de funcionamento e é

destinada a uma larga faixa de dispositivos, com certas características importantes, tais como: capacidade de processamento, memória, comunicação, consumo de energia e interface com o usuário. Atualmente são definidas duas configurações: *Connected Device Configuration* (CDC) e *Connected Limited Device Configuration* (CLDC).

3.1 CDC

Configuração para dispositivos conectados. É destinada a equipamentos com no mínimo 512 Kb de memória ROM, 256 Kb de memória para alocação em tempo de execução (RAM), e algum tipo de conexão, possivelmente persistente e com uma alta taxa de transmissão. A CDC suporta a máquina virtual completa e é voltada para equipamentos com sistema de navegação via satélite, TVs com recursos de rede e PDA’s com maior capacidade.

3.2 CLDC

Configuração para dispositivos conectados e limitados. É destinada a equipamentos que possuem entre 128 e 512 Kb de memória ROM, 32 Kb de RAM e uma interface com usuários restrita, além de conexão de rede tipicamente sem fio, com baixa taxa de transmissão e acesso intermitente. É a tecnologia mais popular e abordada, pois é nela que se encaixam os telefones celulares, *pagers*, *mini-games* e outros equipamentos com tamanho similar.

3.3 O PERFIL MIDP

Os perfis são mais específicos que as configurações, pois contém bibliotecas de classes para que desenvolvedores possam escrever código para um dispositivo particular, ou para

categoria restrita de dispositivos. O principal e mais abordado perfil para CLDC é o *Mobile Information Device Profile* (MIDP) ou Perfil de Dispositivo de Informação Móvel.

O Perfil MIDP acrescenta, além das APIs definidas pela CLDC, suas próprias APIs voltadas para a interface com o usuário, tais como: APIs para entrada de dados, manipulação de eventos, persistência, suporte a rede e temporização. Tudo isso levando em consideração as limitações gráficas e de memória dos dispositivos móveis.

3.3.1 Recursos para jogos

A primeira versão do MIDP (MIDP 1.0) não possuía suporte para importantes ações relacionadas com jogos, como suporte a transformações geométricas (escalas, rotações), suporte a som e gerenciamento de objetos do jogo.

Com o lançamento de MIDP 2.0, algumas dessas deficiências foram corrigidas com a introdução de um novo pacote voltado para a produção de jogos. Segundo Nascimento (2003, p11) as classes deste pacote são:

a) *GameCanvas*: uma extensão da classe *Canvas* de MIDP 1.0, responsável por controlar a tela. Esta classe possui um *buffer off-line*, onde as operações de desenho são feitas e depois este *buffer* é desenhado na tela. Outra funcionalidade desta classe é permitir o acesso ao estado das teclas, facilitando identificar quais estão pressionadas, sem ter que ficar dependendo do tratamento dos eventos e possibilitando a identificação de mais de uma tecla pressionada ao mesmo tempo;
b) *Layer*: classe abstrata que representa um elemento do jogo. Pode ser utilizada através das classes filhas *TiledLayer* e *Sprite*;
c) *TiledLayer*: classe que representa uma matriz de células conhecidas como mapa de *tiles*, e um conjunto de imagens, os *tiles*. Cada célula da matriz está associada com

uma imagem, criando um mosaico usado como cenário para o jogo;

d) *Sprite*: classe que representa uma imagem, formada por diversos *frames*. Esses *frames* são usados para construir uma animação para representar o objeto. A classe ainda possui métodos para verificar colisão com outros *layers*;

e) *LayerManager*: possui uma lista ordenada dos *layers*. Gerencia o desenho correto dos *layers* e evita o desenho de áreas que serão encobertas por outros *layers*.

Além do novo pacote para jogos, alguns outros recursos foram implementados em MIDP 2.0 que auxiliam o desenvolvedor de jogos. Segundo Freire (2004) existe a possibilidade de representar as imagens como um *arrays* de bytes, dando acesso às informações de cores de cada *pixel*. Com o acesso direto aos *pixels*, fica possível a implementação de rotinas de colisão mais precisas e de outros efeitos gráficos, como efeitos com partículas. É possível tocar arquivos de músicas e utilizar um gerador de tons para tocar até 4 sons simultaneamente. Foi ainda acrescido o suporte a conexões através de *sockets*, criando melhores condições para implementação de jogos multi-usuários.

3.3.2 Recursos de desenho 2D

No desenvolvimento de jogos, freqüentemente é necessário desenhar diretamente na tela e para isso é usada a classe *Canvas*, que possui altura e largura específicas, e nela é desenhado o que o usuário final verá. A classe *Canvas* também fornece métodos para tratamento de eventos, como eventos de teclado e de toque na tela, se o dispositivo suportar.

Para desenhar em uma tela utiliza-se o objeto *Graphics*. Essa classe tem métodos para desenhar linhas, arcos, retângulos e texto. Através dela pode-se ainda especificar cor aos objetos a serem desenhados e preferência de fonte.

3.3.3 Persistência de dados

O MIDP disponibiliza um mecanismo para que as aplicações possam guardar dados e lê-los mais adiante. É o chamado *Record Management System* (RMS) ou sistema de gerenciamento de registros.

Segundo Muchow (2004) o RMS utiliza memória não volátil para armazenar informações como uma alternativa ao uso de um sistema de arquivos. Esse banco de dados orientado a registros, freqüentemente referido como arquivo puro, pode ser imaginado como uma série de fileiras em uma tabela, com um identificador exclusivo para cada fileira, conforme o Quadro 2.

Id de registro	Dados
1	*Array* de bytes
2	*Array* de bytes
3	*Array* de bytes
...	...

Fonte: Muchow (2004).

Quadro 2 – Armazém de registros (RMS)

3.3.4 Timers

Aplicações que necessitem agendar tarefas para um tempo futuro podem utilizar as classes *Timer* e *TimerTask*, que incluem funções para uma execução, ou várias execuções, em determinados intervalos de tempo. No desenvolvimento de jogos esta classe pode facilitar o gerenciamento de chamadas a *Threads* em *background*.

3.3.5 Multimídia

J2ME possui um pacote opcional chamado *Mobile Media API* (MMAPI), definida

através do *Java Specification Request* (JSR) 135, que foi concebida para uso com a configuração CLDC e projetada de maneira a ser independente de protocolo e formatos. A MMAPI contém toda a funcionalidade necessária para acessar, controlar e gravar dados multimídia.

Segundo Miranda (2002b), é possível dividir a API, em um subconjunto para suportar apenas um tipo de formato multimídia, somente áudio, por exemplo. Assim, perfis J2ME que não são capazes de suportar toda a API podem apenas implementar parte dela.

3.3.6 Rede

A especificação MIDP 1.0 requer que dispositivos compatíveis implementem apenas o protocolo *Hyper Text Transfer Protocol* (HTTP), sendo que outros protocolos podem ou não ser implementados a gosto do fabricante. O MIDP 2.0 adiciona a exigência do HTTP sobre o *Secure Sockets Layer* (HTTPS), possibilitando o desenvolvimento de aplicações seguras, segundo Freire (2004).

A especificação também recomenda que sejam implementados protocolos de conexão baseados em datagramas (para envio de pacotes de dados sem garantia de entrega), *sockets* (para comunicação ponto-a-ponto) e *server sockets* (que possibilita que o dispositivo abra uma porta e aguarde conexões externas, funcionando como um servidor) .

4 DESENVOLVIMENTO DO JOGO

Com objetivo de aplicar a tecnologia J2ME no desenvolvimento de jogos multiplataforma entre dispositivos móveis e apresentar os recursos e as restrições da tecnologia, será apresentado a seguir o desenvolvimento completo de um jogo.

O jogo escolhido para exemplificar é o Tetris, pois é um jogo muito conhecido e com boas características didáticas.

4.1 O JOGO TETRIS

Tetris é um dos jogos mais populares da história, conseguindo ser ao mesmo tempo muito simples e desafiador. Segundo Atari Gaming (2005), o jogo foi criado por Alexey Pazhitnov, em 1985, no centro de Ciências da Computação da academia de Moscou.

Trata-se de um jogo de encaixar peças, que são exibidas aleatoriamente. Atualmente existem muitas variantes do jogo para diversas consoles e PC’s. Na Figura 2 são apresentadas algumas dessas variações do jogo.

Figura 2 – Variantes do jogo Tetris

4.2 REQUISITOS PRINCIPAIS DO JOGO

Como Tetris é um jogo que possui muitas variantes, este trabalho pretende implementar a versão clássica, que entre seus requisitos funcionais destacam-se:

a) o jogo deve suportar mudança de níveis ou fases;

b) o jogo deve prever a contagem de pontos. Sempre que uma linha estiver preenchida de quadrados, serão incrementados 10 pontos ao jogador;

c) a cada 30 pontos o jogador muda de fase;

d) o jogo deve suportar infinitos níveis, sendo que a cada nível a velocidade do jogo deve aumentar. Isso quer dizer que as peças cairão mais rápidas, a cada nível, até a velocidade que o jogador não poderá controlar as peças;

e) deve ser possível movimentar a peça corrente, a fim de encaixá-la, da melhor

forma possível no tabuleiro;

f) o jogo deve suportar quatro tipos de sons: a cada movimento de descida da peça, quando a peça encaixar no tabuleiro, quando o jogador fizer pontos e quando o jogador passar de fase;

g) o jogo deve terminar quando o número de quadrados atingirem a parte superior da tela;

h) o jogo deve apresentar um *ranking* com os três melhores resultados já atingidos.

Entre os requisitos não funcionais do jogo, destacam-se:

a) o jogo deve rodar em diversos aparelhos que suportem a tecnologia J2ME e o perfil MIDP 2.0, como celulares e *Personal Digital Assistants* (PDAs);

b) o jogo deve possuir interface colorida, mas deve rodar também em aparelhos em que o *display* seja monocromático;

c) deve haver a possibilidade de habilitar e desabilitar o som do jogo.

4.3 ESPECIFICAÇÃO

Para fazer a especificação do jogo foi utilizada uma técnica orientada a objetos, representada em diagramas da *Unified Modeling Language* (UML). A ferramenta usada para fazer a modelagem foi o *JUDE Community 1.5.2* JUDE (2005), ferramenta gratuita para modelagem de projetos orientados a objetos com suporte a Java e UML que na versão atual possui suporte a UML 1.4.

4.3.1 Casos de uso

Os casos de uso têm como propósito principal descrever de forma conceitual a

estrutura do software, segundo Furlan (1998, p. 169). Na Figura 3, são apresentados os principais casos de uso do jogo.

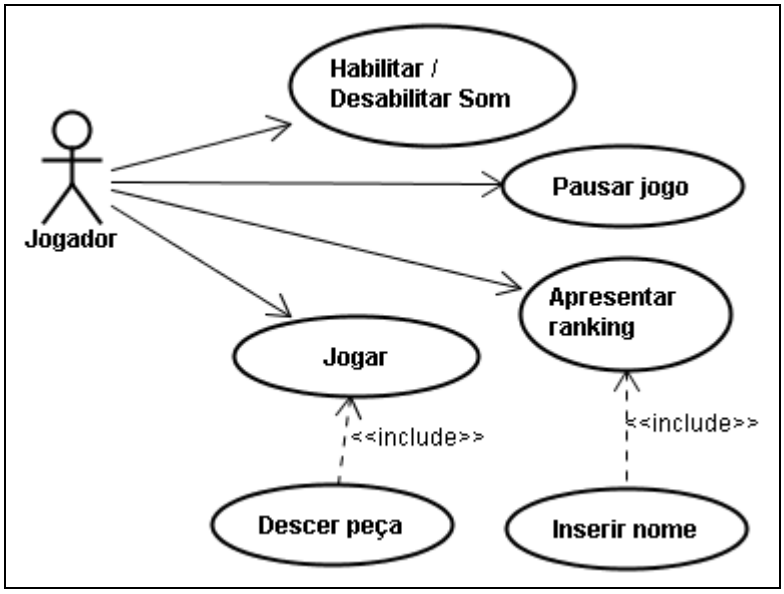


Figura 3 – Principais casos de uso do jogo

No Quadro 3, estão descritos os casos de uso do jogo, com seus respectivos atores e a descrição de cada um deles.

Caso de uso	**Descrição**
Jogar	O jogador irá movimentar as peças que surgem no tabuleiro com intenção de encaixá-las de forma que preencha a maior quantidade de linhas, sem deixar espaços incompletos. A cada linha completada o jogador irá ganhar 10 pontos. A cada 30 pontos o jogador irá mudar de nível, sendo que as peças irão descer mais rápidas a cada nível. O jogo irá terminar, quando as peças do tabuleiro atingirem a superfície da tela.
Descer peça	O jogo irá descer a peça corrente do tabuleiro, uma posição, a cada intervalo de tempo determinado pelo nível atual do jogo. A cada movimento de descida da peça será executado um som.
Habilitar / Desabilitar Som	O jogador poderá, a qualquer momento, habilitar ou desabilitar o som do jogo.
Pausar o jogo	O jogador poderá pausar a execução do jogo, sendo que o mesmo permanecerá em estado de espera por tempo indeterminado até que o jogador desabilite a opção de pausa.
Apresentar ranking	No final do jogo é apresentado um ranking com os nomes dos três melhores jogadores.
Inserir nome	O jogador poderá deixar seu nome registrado no ranking do jogo, desde que consiga uma pontuação maior que os três primeiros lugares.

Quadro 3 – Casos de uso do jogo

4.3.2 Diagrama de classes

Segundo Furlan (1998, p. 91), o diagrama de classes é utilizado para representar a estrutura lógica de um sistema detalhando cada elemento como: pacotes, classes, tipos, atributos e métodos. A Figura 4 apresenta as dependências entre os pacotes do jogo.

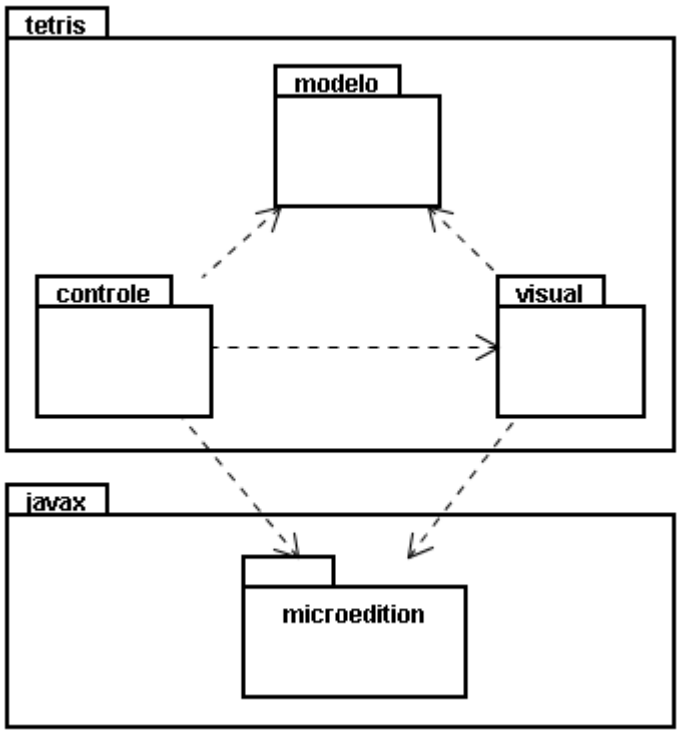


Figura 4 – Dependência entre pacotes e arquitetura MVC

O pacote do jogo (tetris) é subdividido em três pacotes: modelo, visual e controle, análogo ao *design pattern* MVC, que segundo Freitas e Rezende (2003), determina a separação de responsabilidades de uma aplicação em camadas que são:

a) modelo (*model*): agrupa toda a lógica do jogo, informando à camada de visualização, através da camada de controle, quando o estado interno do jogo for alterado;

b) visualização (*view*): desenha o estado atual do modelo, na tela do dispositivo, definindo como o modelo deve ser apresentado para o jogador;

c) controle (*controller*): interpreta as ações do jogador e as mapeia para chamadas do

modelo. Implementa interface para receber mensagens do modelo.

O pacote *microedition* do *javax* está representado neste modelo pois é onde estão concentradas as bibliotecas específicas do J2ME. Como visto na Figura 4, o pacote modelo não depende do pacote *microedition*, tornando o modelo portável para outras plataformas que suportem Java como é o caso do ExEn ou BREW, por exemplo.

A seguir serão apresentados os diagramas de classes dos pacotes controle (Figura 5), modelo (Figura 6) e visual (Figura 7).

A Figura 5 mostra as classes do pacote controle e suas relações com as classes do pacote *microedition* do J2ME.

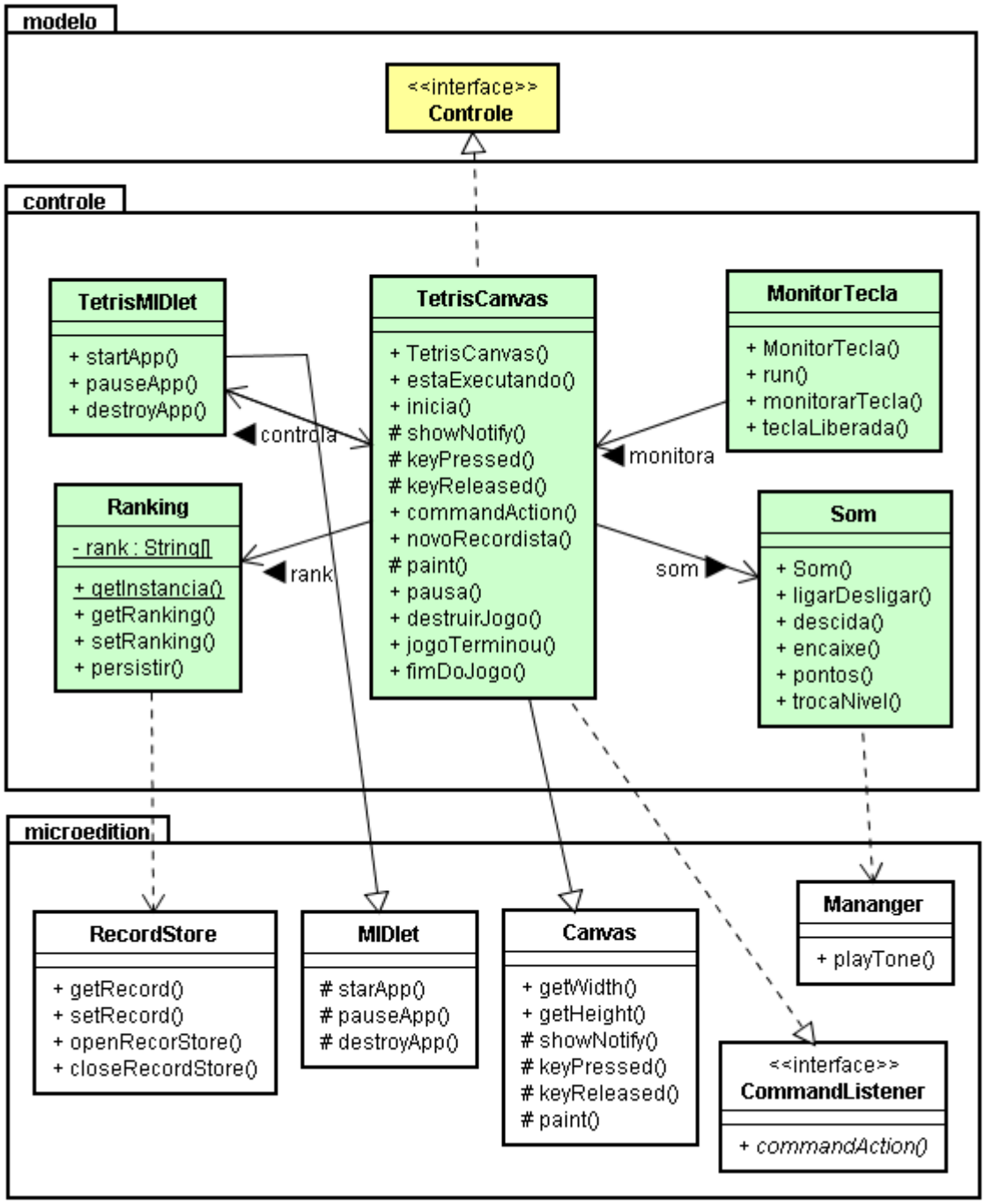


Figura 5 – Classes do pacote controle

Para facilitar o entendimento, alguns atributos, métodos e classes, menos importantes foram omitidos dos diagramas, assim como alguns pacotes do J2ME.

A seguir está um resumo do funcionamento das principais classes da camada de controle:

a) TetrisMIDlet: classe inicial do jogo. Assim como todo aplicativo móvel (*midlet*) a

classe inicial do jogo (ou classe *main*) deve estender a classe *MIDlet* do MIDP, para que o gerenciador de aplicativos do dispositivo possa reconhecer a aplicação e iniciá-la através do método abstrato *startApp()*;

b) TetrisCanvas: classe principal do controle do jogo, estende o *Canvas* que permite desenhar a interface do jogo no dispositivo através do método abstrato *paint*() e tratar os eventos do jogo. Esta classe também implementa a interface Controle, mostrada na Figura 6, permitindo que o pacote modelo, interaja com os demais componentes do jogo. Além disso ela implementa também a interface *CommandListener* que monitora os eventos de menu do jogo.

c) MonitorTecla: esta classe é uma *thread*, cuja função é monitorar se o jogador permanece com as teclas do dispositivo pressionada por um determinado tempo. Em caso positivo informa a classe TetrisCanvas para repetir o evento de tecla pressionada, usado para fazer a peça atual do jogo "correr rapidamente" de um lado para outro, em dispositivos que não suportam repetição de eventos;

d) Ranking: responsável por persistir o ranking no dispositivo mesmo que este seja desligado ou sua bateria trocada, usando-se para isso a classe *RecordStore* do RMS (explicado na seção 3.3.3);

e) Som: responsável por executar os diversos sons do jogo, utilizando-se para isso dos recursos da MMAPI (explicada na seção 3.3.5) utilizando a classe *Mananger* que disponibiliza métodos para criar *players* para diversos tipos de arquivos.

Na Figura 6, são apresentadas as principais classes do pacote modelo.

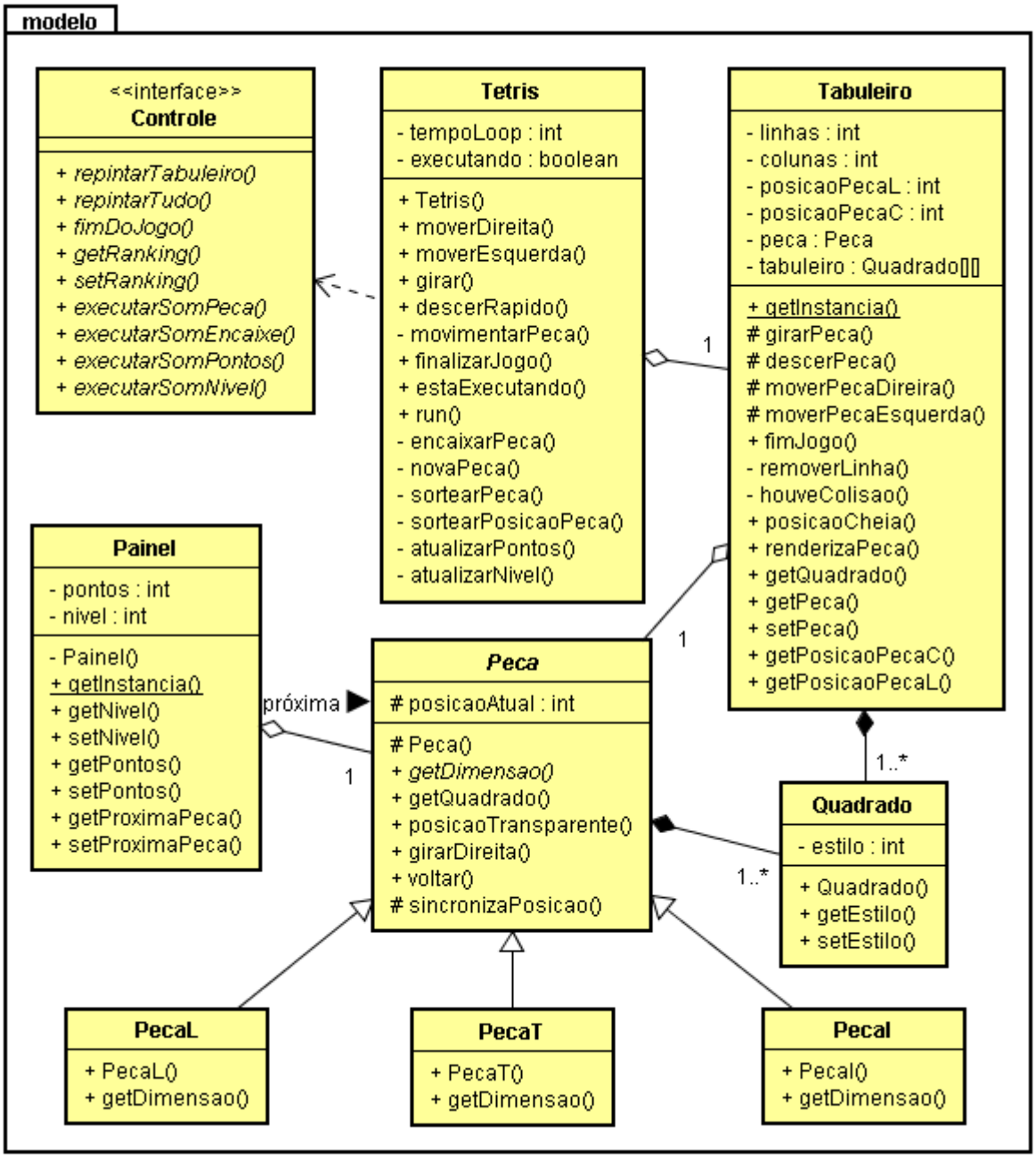


Figura 6 – Classes do pacote modelo

Entre as principais classes do pacote modelo, destacam-se:

a) Tetris: classe que agrupa a maior parte da lógica do jogo. Trata-se de uma *thread*, sendo que no método *run* é implementado o *looping* do jogo;

b) Controle: interface pela qual o modelo se comunica com outras camadas do jogo. Através desta interface o modelo do jogo torna-se independente das bibliotecas do J2ME;

c) Tabuleiro: representa o tabuleiro do jogo onde as peças se movem. Possui métodos para movimentar e fixar as peças dentro do tabuleiro;

d) Painel: representa o painel do jogo onde são colocados os pontos, o nível e a próxima peça do jogo;

e) Peca: classe abstrata que contém a estrutura física de uma peça. Todas as demais peças estendem esta classe.

f) Quadrado: representa a menor parte do tabuleiro ou de uma peça. Possui um atributo chamado “estilo”, que quando verificado pela camada de visualização, irá determinar suas características visuais.

Na Figura 7, são apresentadas as principais classes do pacote de visualização.

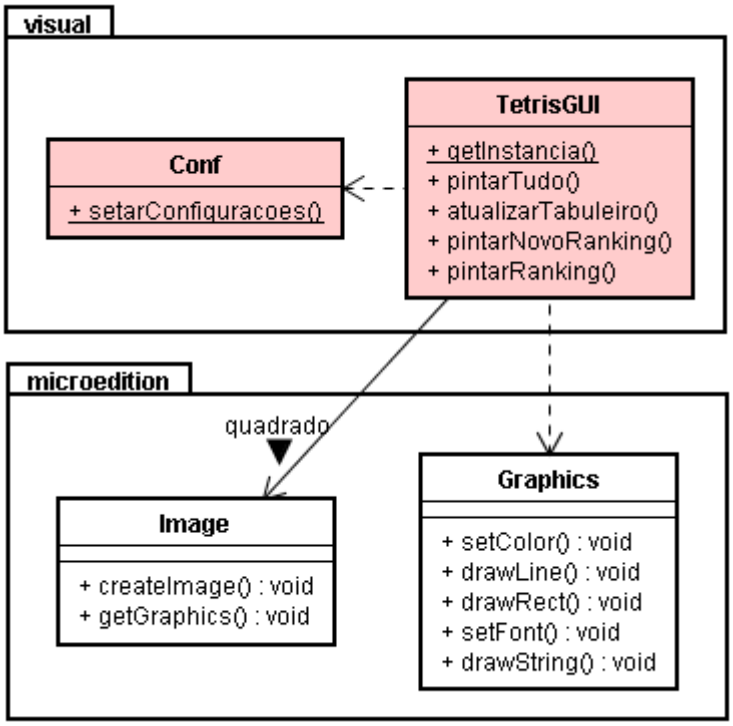


Figura 7 – Classes do pacote visual

A classe TetrisGUI é responsável por desenhar no *display* do dispositivo todos os objetos do jogo de acordo com seu estado atual, obtido através do modelo. Para isso ele utiliza as classes *Graphics* e *Image* do MIDP. Conf é uma classe utilitária do jogo, onde estão mapeadas todas as configurações referentes ao visual do jogo, como: cores, nomes de menus,

fontes, etc...

4.3.3 Diagrama de atividades

Um diagrama de atividades é essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra (BOOCH, RUMBAUGH e JACOBSON, 2000). Através do diagrama de atividades é apresentado o *looping* principal do jogo na Figura 8.

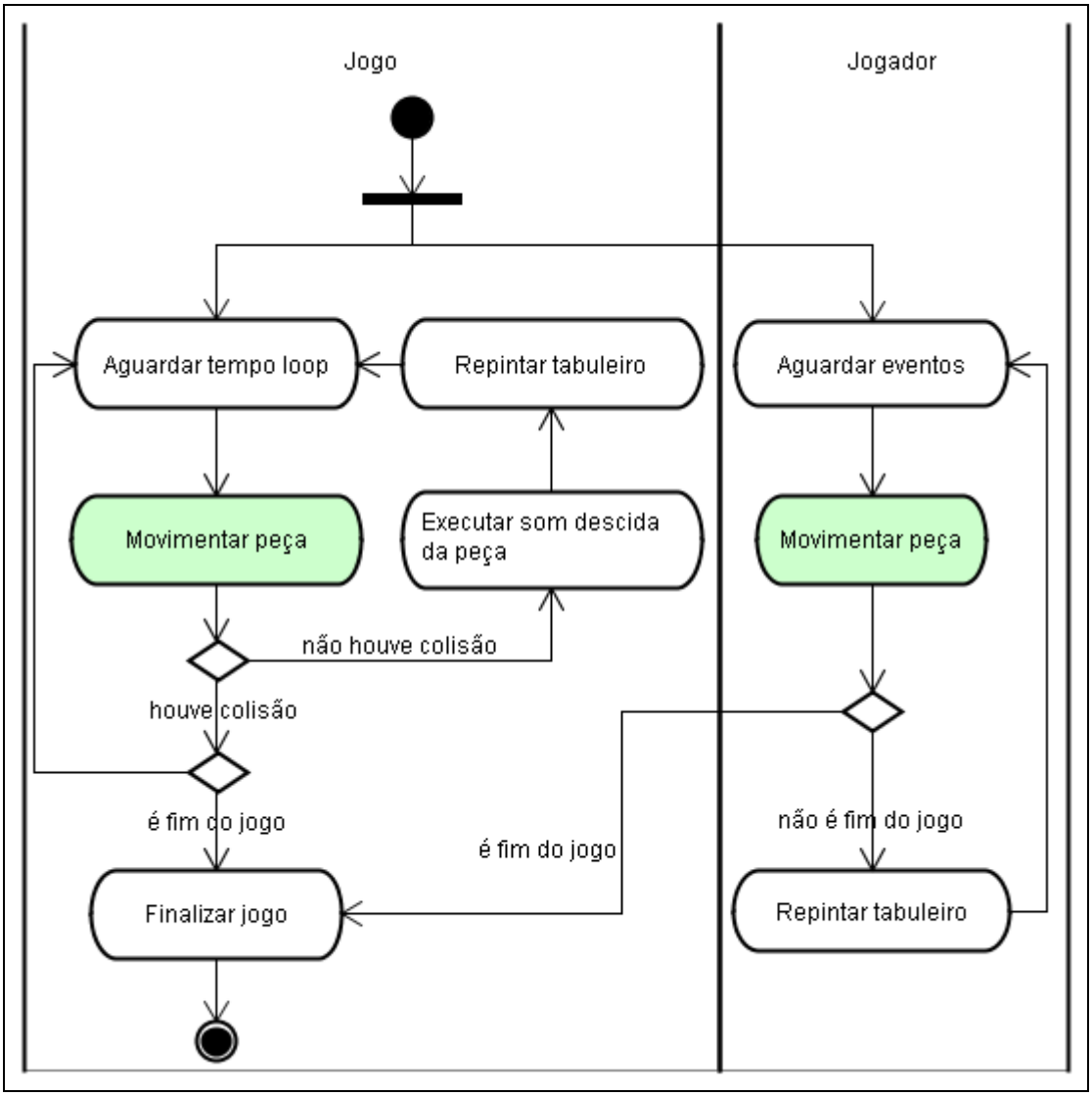


Figura 8 – Diagrama de atividades, *looping* principal do jogo

Pode-se observar no diagrama da Figura 8, que as atividades “Movimentar peça” e “Repintar tabuleiro”, podem ser executadas simultaneamente, através da *thread* do jogo que

controla a descida das peças ou através de um evento gerado pelo jogador. Não há problemas em repintar o tabuleiro simultaneamente, uma vez que isto não influencia no processamento do modelo do jogo. Porém, o movimento simultâneo das peças pode ocasionar problemas se executada ao mesmo tempo, como será visto a seguir. Desta forma deve-se sincronizar a execução desta atividade.

Na Figura 9, a atividade “Movimentar peça” é apresentada detalhadamente.

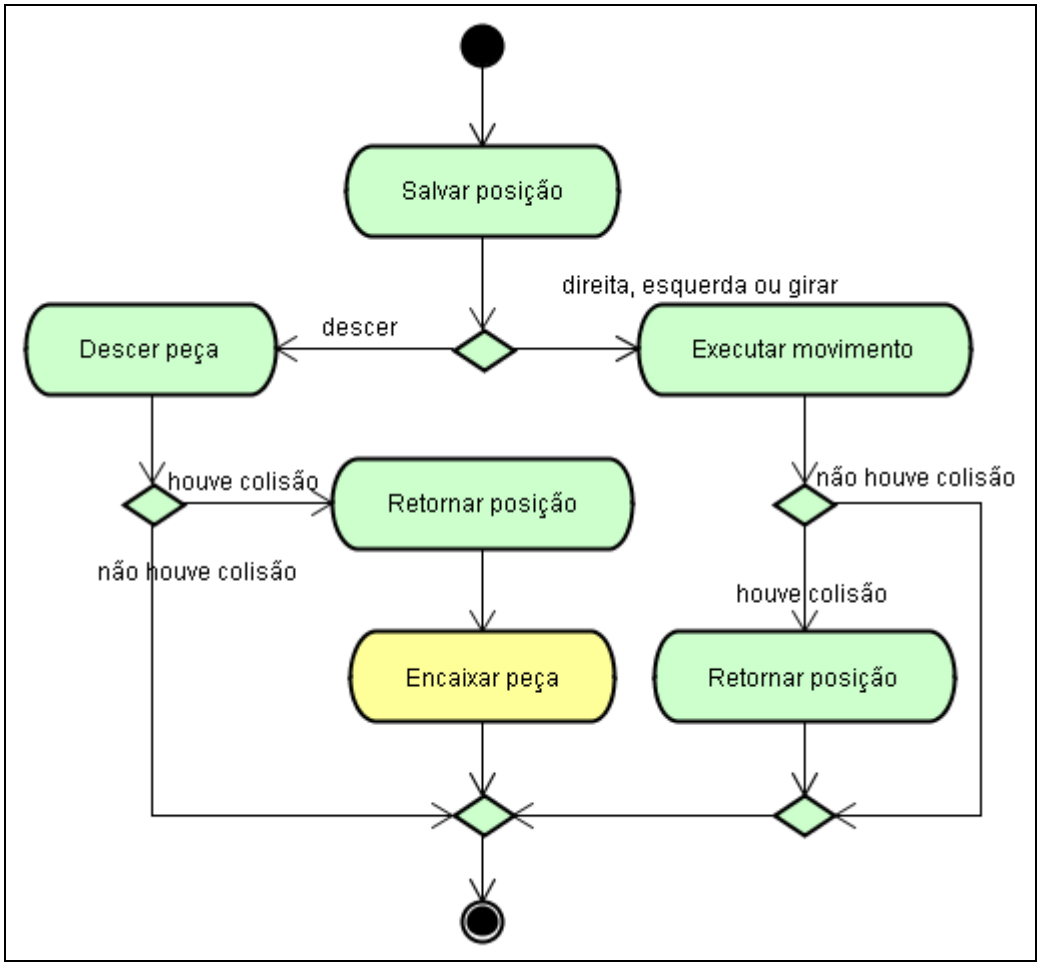


Figura 9 – Detalhes da atividade “Movimentar peça”

Uma técnica interessante no desenvolvimento de jogos quando se necessita que um objeto (geralmente um *layer*) não sobreponha outro na tela é apresentada na Figura 9. Trata-se de salvar as posições do objeto antes de executar o movimento e, se há colisão, retorná-lo à posição anterior. Normalmente isso é melhor do que verificar se vai haver colisão e então executar o movimento. A maioria das bibliotecas de jogos citadas neste trabalho possuem

detecção de colisão, mas elas só apontam a colisão se os objetos (ou *layers*) estiverem com alguma área sobreposta.

A atividade encaixar peça engloba uma grande parte de lógica do jogo, por isso ela é mais bem detalhada na Figura 10.

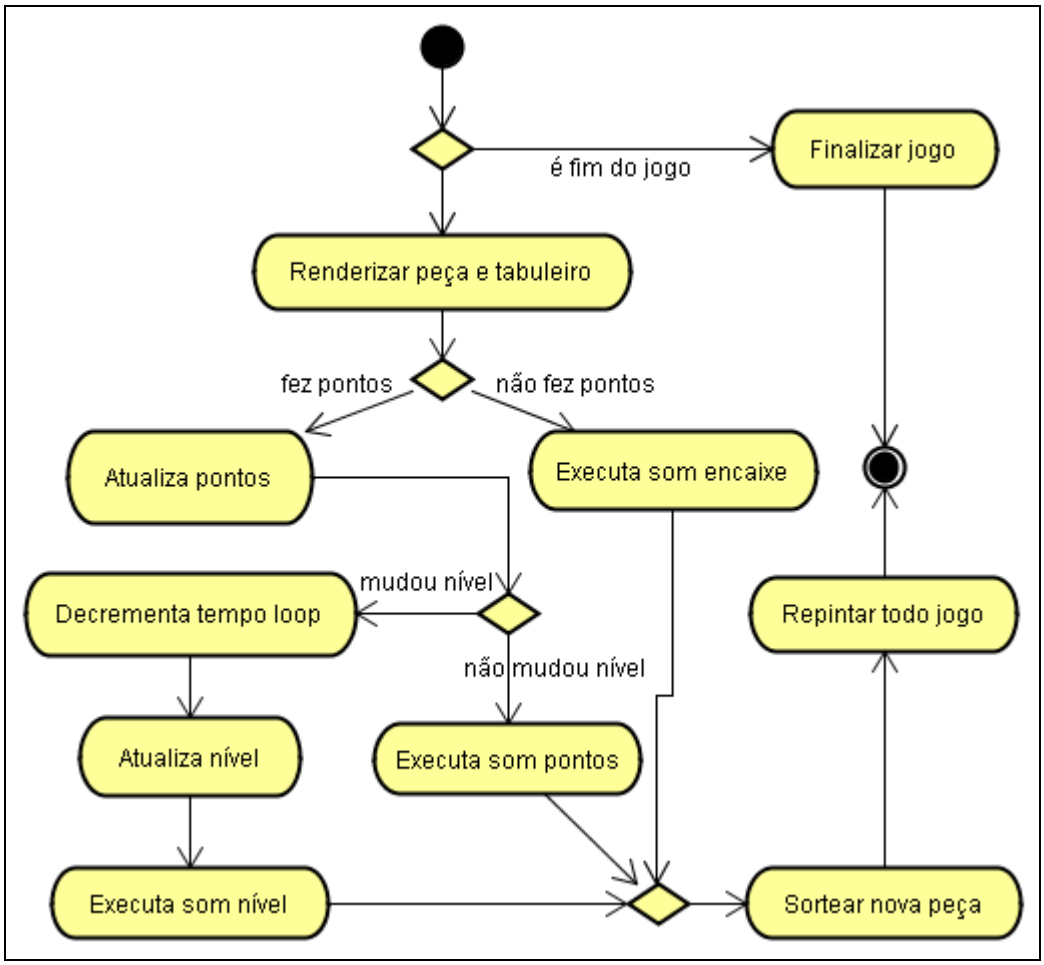


Figura 10 – Detalhes da atividade “Encaixar peça”

Devido à atividade “Renderizar peça e tabuleiro”, mostrada na Figura 10, é que a atividade “Movimentar peça” (Figura 8) não pode ser executada simultaneamente, pois pode ocorrer que, em um momento antes de executar a atividade, a peça poderia ser movimentada novamente e logo após ser renderizada em um local inadequado.

4.4 IMPLEMENTAÇÃO DO JOGO

Nas próximas seções são abordadas as ferramentas utilizadas no desenvolvimento do jogo e maiores detalhes sobre sua implementação e funcionamento.

4.4.1 Ferramentas de desenvolvimento

Como visto anteriormente, para a implementação do jogo, foi utilizada a plataforma J2ME para dispositivos CLDC que suportem o perfil MIDP 2.0.

Para a compilação e emulação do jogo, foi utilizado o J2ME Wireless Toolkit 2.2 (WTK), ferramenta da Sun, para aplicações *wireless* que possui vários emuladores de dispositivos móveis. Através dele pode-se emular o jogo perfeitamente, pois a ferramenta apresenta várias opções de configuração como, por exemplo, a simulação de velocidade de processamento da máquina virtual, gerenciamento de memória e conexão de rede entre outros. A Figura 11 mostra a tela principal do WTK.

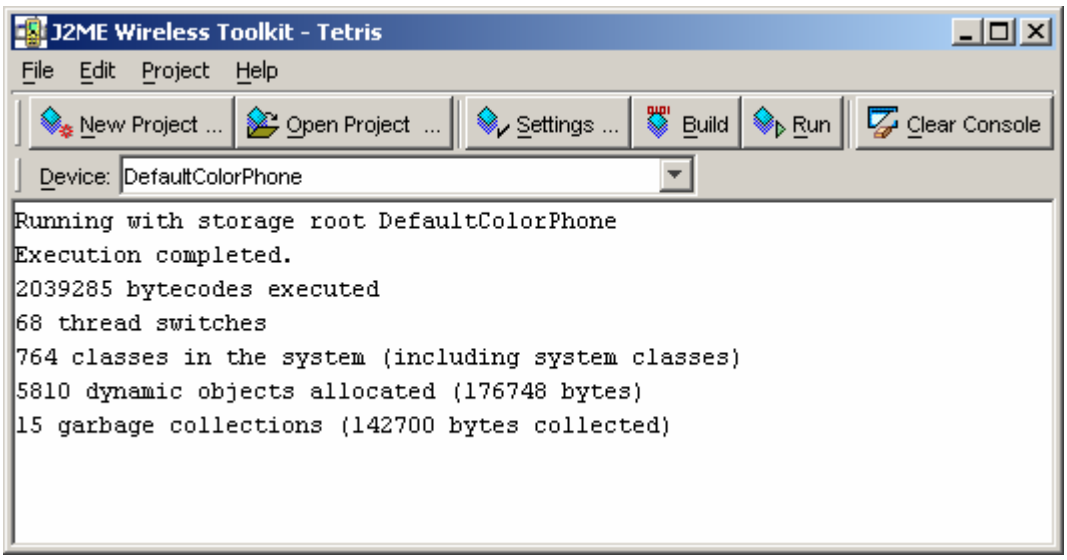


Figura 11 – Tela principal do J2ME Wireless Toolkit

Ao criar um projeto no WTK deve-se informar o nome da classe que estende a classe MIDlet, que neste caso é “br.furb.tcc.tetris.controle.TetrisMIDlet”. O WTK cria uma estrutura de diretórios dentro da pasta “*apps*”, com o nome do projeto, no local onde a ferramenta foi instalada. Dentro desta estrutura está a pasta “*src*”, onde devem ser adicionados os fontes do jogo.

O WTK 2.2 da Sun não possui um editor acoplado em sua instalação, sendo utilizado para isso o Eclipse 3.0, que tem um editor bastante completo em recursos. No projeto do jogo do Eclipse, foram adicionadas, no *classpath* somente as bibliotecas do CLDC e MIDP, como mostrado na Figura 12, assim podem-se usar todos os recursos do editor como o assistente de código, marcação de erros, *debug*, entre outros.

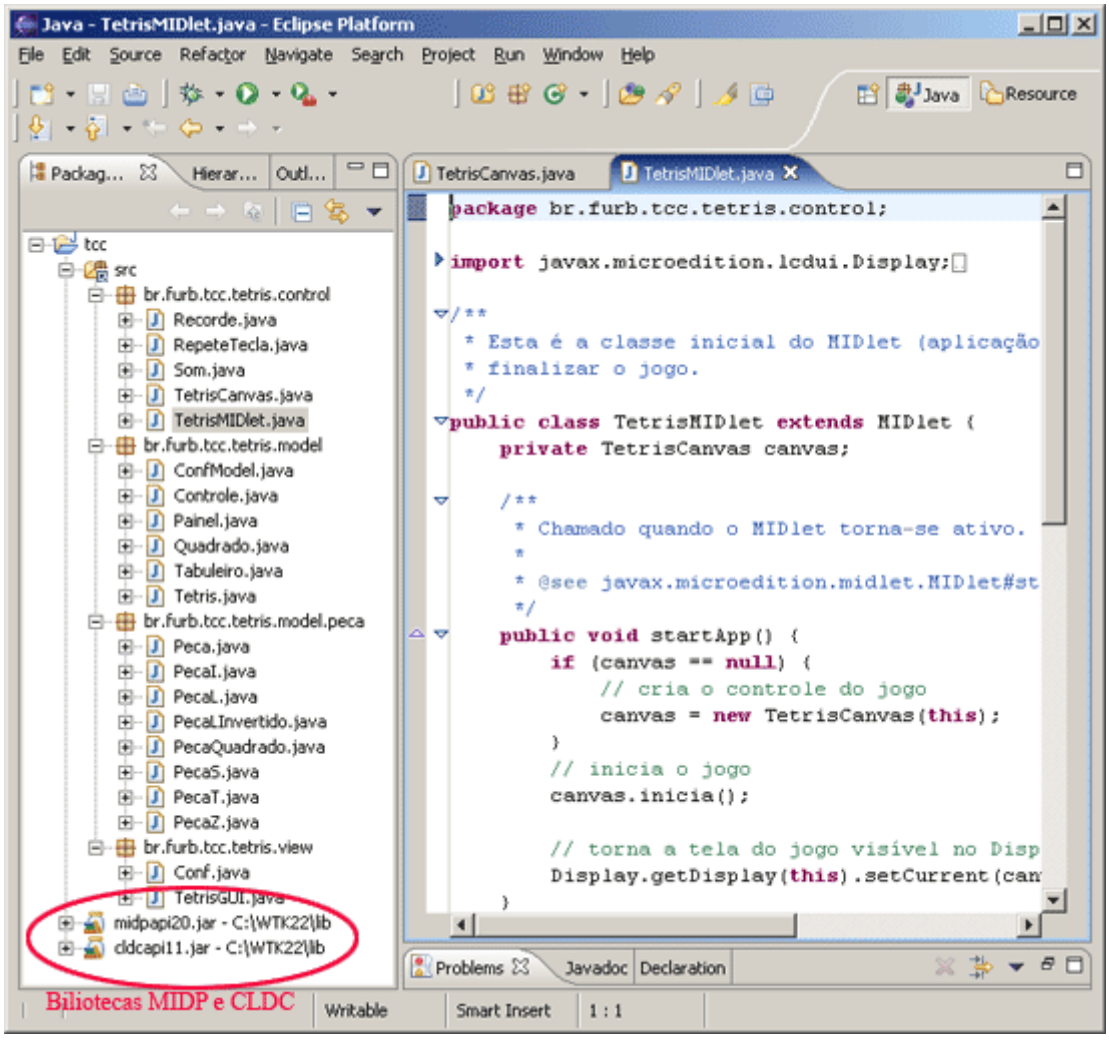


Figura 12 – Eclipse 3.0

4.4.2 Operacionalidade da implementação

Na Figura 13 é apresentado o jogo em execução, no emulador padrão do WTK.

Figura 13 – Jogo Tetris em execução no emulador padrão do WTK.

Para jogar podem ser usadas as seguintes teclas do dispositivo:

a) tecla de direção (↑) *up* ou tecla número 2: faz a peça atual do tabuleiro girar no sentido horário;

b) tecla de direção (↓) *down* ou tecla com número 8: faz a peça descer rapidamente até encaixar no tabuleiro;

c) tecla de direção (←) *left* ou tecla com número 4: faz a peça movimentar-se uma posição para a esquerda. Caso esta tecla continuar pressionada a peça irá movimentar-se rapidamente para a esquerda;

d) tecla de direção (→) *right* ou tecla com número 6: faz a peça movimentar-se uma posição para a direita. Caso esta tecla continuar pressionada a peça irá movimentar-se rapidamente para a direita.

E ainda as teclas do menu que podem:

a) pausar o jogo;

b) habilitar e desabilitar o som do jogo;

c) sair do jogo.

O jogador ganha 10 pontos quando uma linha estiver completa de quadrados. Para fins de demonstração, o jogo foi configurado para trocar de nível a cada 30 pontos, sendo que o tempo de descida da peça é iniciado em 0,7 segundos e cada troca de nível este tempo é diminuído em 0,06 segundos. O jogo termina quando as novas peças colidem com as já existentes no momento de sua criação, ou em outras palavras quando as colunas do meio do tabuleiro estiverem completas de quadrados.

Quando o jogo termina, se o número de pontos do jogador for maior que os três melhores resultados já alcançados, o jogador tem a opção de deixar seu nome registrado no *ranking* do jogo e em seguida é apresentado o *ranking* com os três melhores resultados já alcançados (Figura 14).

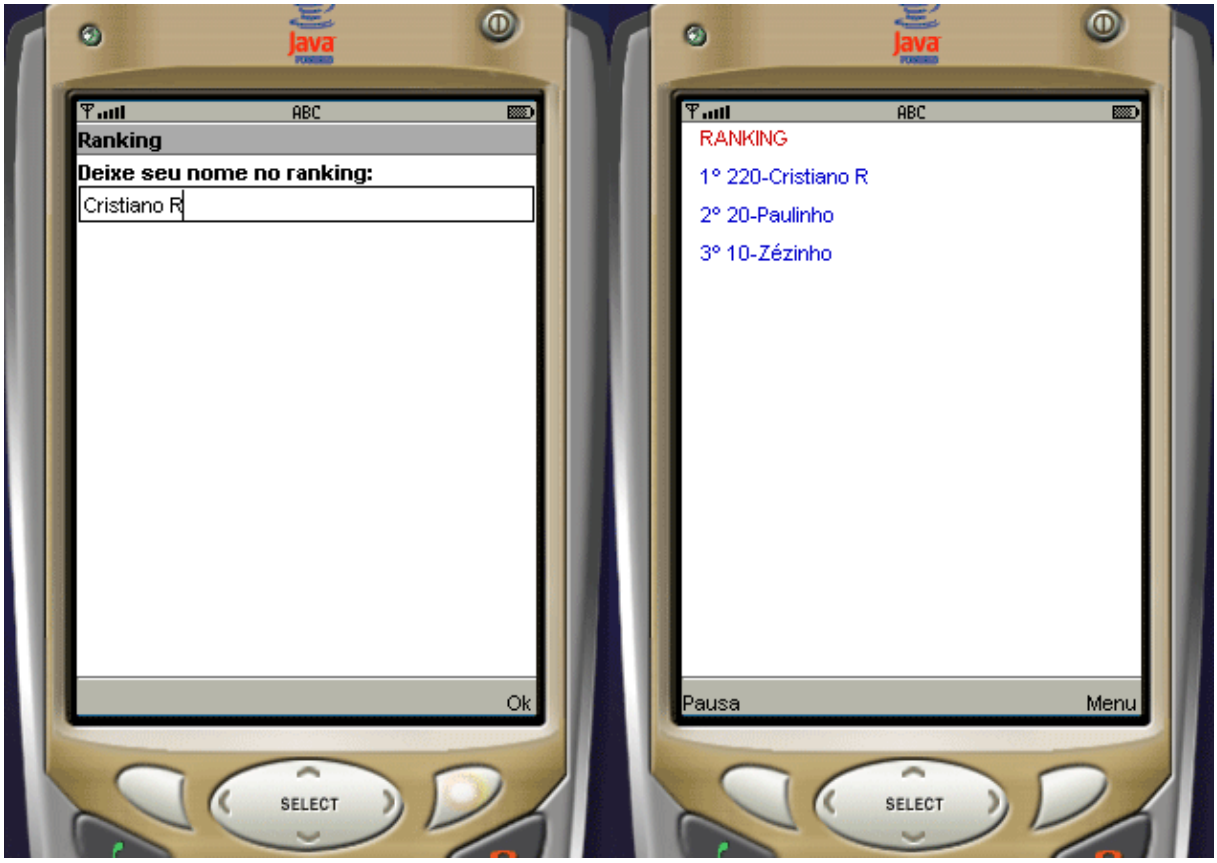


Figura 14 – Opção para deixar o nome registrado no *ranking*.

O *ranking* com os três melhores resultados fica armazenado no dispositivo enquanto o jogo estiver instalado no mesmo. O Quadro 4 mostra o código para persistir um registro utilizando o RMS do MIDP.

```
/**
 * Grava o dado no banco de registros.
 * @param posicao do dado.
 * @param dado a ser gravado.
 */
private void gravar(int posicao, String dado) {
    try {
        //transforma a String em um array de bytes
        byte[] rec = dado.getBytes();
        //se posicao maior que o número de registros do banco
        if (posicao > banco.getNumRecords()) {
            //adiciona novo registro
            banco.addRecord(rec, 0, rec.length);
        } else {
            //sobreescreve o registro da posição no banco
            banco.setRecord(posicao, rec, 0, rec.length);
        }
    }catch (Exception e) {
        e.printStackTrace();
    }
}
```

Quadro 4 – Persistência de registro utilizando o RMS

4.4.3 Portabilidade do jogo

O jogo foi desenvolvido para ser o mais portável possível. Em relação à plataforma pode-se afirmar que roda em todos os dispositivos com suporte a MIDP 1.0 e a MMAPI (explicada na seção 3.3.5).

4.4.3.1 Tamanho do *display*

Em relação ao tamanho da tela deve-se considerar que a maioria dos dispositivos móveis não possui um *display* de 240 por 289 *pixels*, que é o tamanho do emulador padrão do WTK, e sim em torno de 128 por 96 que é o caso do celular Nokia modelo 7210, apresentado na Figura 15.

Figura 15 – Jogo Tetris rodando em emulador do celular Nokia 7210

Como visto anteriormente, uma das maiores dificuldades no desenvolvimento de jogos para dispositivos móveis é a preocupação com a diversidade de *displays* em que o jogo irá rodar. Para que o jogo ficasse idêntico e ao mesmo tempo atraente nos mais diversos tipos de dispositivos móveis, foi necessário desenvolver uma técnica para ajustar o tamanho dos componentes do jogo, fazendo com que eles se redimensionassem de acordo com o tamanho da tela dos aparelhos.

Assim, antes de iniciar o jogo é obtida a largura e altura do *display* do dispositivo através dos métodos da classe *Canvas* *getWidth()* e *getHeight()* respectivamente. Estes parâmetros são passados para o método, mostrado no Quadro 5, que configura uma série de atributos estáticos. Esses atributos servem de referência para a montagem e disposição dos objetos na tela.

```
/**
 * Configura o tamanho do jogo
 * e dos quadrados de acordo com o tamanho
 * do display do dispositivo
 * @param l largura do display
 * @param a altura do display
 */
public static void setarConfiguracoes(int l, int a) {
    TELA_LARGURA = l;
    TELA_ALTURA = a;
    if (l < a) {
        QUADRADO_TM = (l - 5) / (TABULEIRO_COLUNAS + PECA_MAX_TM);
    } else {
        QUADRADO_TM = (a - 1) / TABULEIRO_LINHAS;
    }
    PAINEL_ALTURA = (TABULEIRO_LINHAS * QUADRADO_TM);
    PAINEL_PX = (TABULEIRO_COLUNAS * QUADRADO_TM) + 2;
    PAINEL_PX_LABEL_NIVEL = PAINEL_PX + 3;
    PAINEL_PY_LABEL_NIVEL = QUADRADO_TM * 7;
    PAINEL_PX_NIVEL = PAINEL_PX + 3;
    PAINEL_PY_NIVEL = PAINEL_PY_LABEL_NIVEL + 12;
    PAINEL_PX_LABEL_PONTOS = PAINEL_PX + 3;
    PAINEL_PY_LABEL_PONTOS = PAINEL_PY_NIVEL + 15;
    PAINEL_PX_PONTOS = PAINEL_PX + 3;
    PAINEL_PY_PONTOS = PAINEL_PY_LABEL_PONTOS + 12;
    PAINEL_PROX_PECA_PX = PAINEL_PX + 3;
    PAINEL_PROX_PECA_PY = 3;
}
```

Quadro 5 – Método que seta a configuração dos objetos conforme o tamanho do *display*

Dentro do método “setarConfiguracoes”, é calculado o tamanho em *pixels* do quadrado do tabuleiro (QUADRADO_TM), que é proporcional ao tamanho da tela e do número de linhas ou colunas configuradas para o jogo. O restante dos atributos servem como referência para o posicionamento dos objetos na tela e são calculados através do tamanho do quadrado.

Os resultados deste redimensionamento automático podem ser vistos na Figura 16, onde a mesma versão do jogo é apresentada em 4 dispositivos com tamanho de telas diferentes.

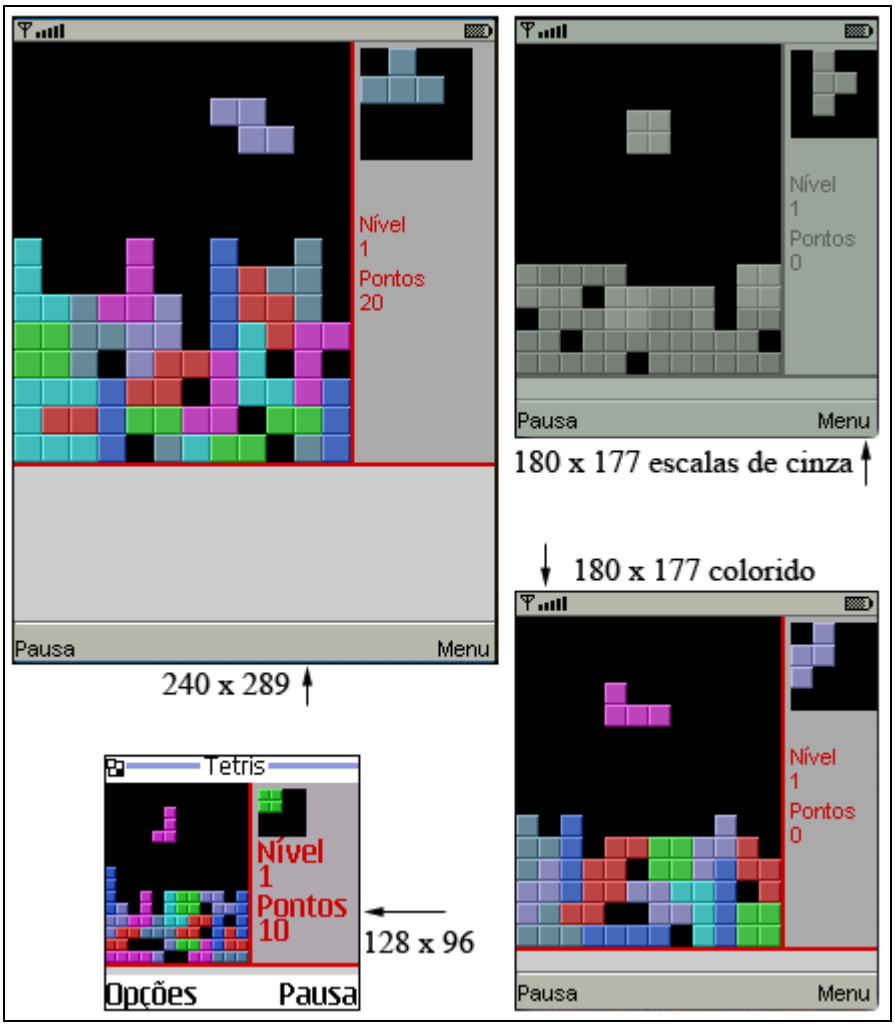


Figura 16 – Tetris em vários dispositivos

Também foram aplicados efeitos de sombra para exibição das peças, sendo somada uma constante no valor da cor base do quadrado. Para obter o efeito reflexo foi diminuída da cor base do quadrado a mesma constante, como mostrado no Quadro 6.

```
/**
 * Método responsável por desenhar um quadrado dentro de uma imagem
 * @param estilo do quadrado a ser desenhado.
 * @return a imagem do quadrado desenhado.
 */
private static Image criarImagem(int estilo) {
    //cria uma imagem dinâmica
    Image ret = Image.createImage(Conf.QUADRADO_TM, Conf.QUADRADO_TM);
    Graphics gr = ret.getGraphics();

    //recupera o tamanho do quadrado
    int t = Conf.QUADRADO_TM - 1;
    int cor = getCor(estilo);

    //pinta o efeito reflexo
    gr.setColor(cor + 0x333333);
    gr.drawLine(0, t - 1, 0, 0);
    gr.drawLine(1, 0, t, 0);

    //pinta miolo
    gr.setColor(cor);
    gr.fillRect(1, 1, t - 1, t - 1);

    //pinta o efeito de sombra
    gr.setColor(cor - 0x333333);
    gr.drawLine(0, t, t, t);
    gr.drawLine(t, 1, t, t);

    return ret;
}
```

Quadro 6 – Criação da imagem do quadrado

O método “criarImagem”, encontra-se dentro da classe “TetrisGUI” da camada visual. Ele desenha um quadrado de uma peça de acordo com seu estilo e retorna um objeto *Image*. Assim, se houver necessidade de criar uma imagem diferente de um quadrado basta alterar a implementação deste método.

Ainda em relação a portabilidade, um requisito importante é garantir que o jogo rode na mesma velocidade, em dispositivos com processadores diferentes. Para isso é aplicada a técnica mostrada no Quadro 7.

```
/**
 * Neste método fica o looping de descida da peça.
 * @see java.lang.Runnable#run()
 */
public void run() {
    long inicio, fim, duracao;
    executando = true;
    try {
        while (executando) {
            //obtém o tempo no inicio da iteração
            inicio = System.currentTimeMillis();
            if (movimentarPeca(ConfModel.DESCE)){
                control.executarSomPeca();
                control.repintarTabuleiro();
            }
            //obtem o tempo no fim da iteração
            fim = System.currentTimeMillis();
            duracao = (fim - inicio);
            if (duracao < tempoLoop) {
                //subtrai o tempo do loop com a execução do jogo
                //para garantir que a peça caia na mesma velocidade
                //em processadores diferentes
                Thread.sleep(tempoLoop - duracao);
            }
        }
    } catch (Exception ie) {
        ie.printStackTrace();
        finalizarJogo();
    }
}
```

Quadro 7 – Looping de descida da peça

Esta técnica calcula o tempo que o processador leva para executar o jogo, chamado de “duracao”. A diferença entre o tempo em que a peça deve cair “tempoLoop” e a “duracao” é que determina quanto o jogo deve esperar até que haja uma nova iteração. Sem a aplicação desta técnica, qualquer otimização na implementação do jogo, torna-o mais rápido e vice-versa.

4.4.4 Instalação

Há várias maneiras de instalar um jogo, ou qualquer aplicativo, em dispositivos móveis, o que geralmente depende de cada equipamento. Normalmente nos celulares, que

suportam J2ME, basta copiar o jogo para um local do aparelho destinado exclusivamente para jogos e aplicativos. As maneiras para fazer esta cópia também são diversas e dependem dos tipos de conexões que o dispositivo suporta, podendo ser do seguinte modo:

a) *download* via *browser* do aparelho;

b) cabo serial, ou equivalente;

c) infravermelho;

d) *bluetooth*.

Neste trabalho o jogo Tetris foi instalado em um celular Nokia modelo 7250i. Para fazer a comunicação entre o celular e o PC foi utilizado um dispositivo infravermelho para o PC desenvolvido pelo autor deste trabalho (ver apêndice A) e da aplicação *Nokia PC Suite* 5, também instalada no PC.

O *Nokia PC Suite* é um pacote de aplicativos desenvolvido para ser usado com celulares Nokia. Ele permite editar, armazenar e sincronizar dados do celular em um PC (NOKIA, 2005). Entre este pacote de aplicativos está o *Nokia Application Installer* (Figura 17), que é utilizado para a instalação de aplicativos, ou jogos, nos celulares.

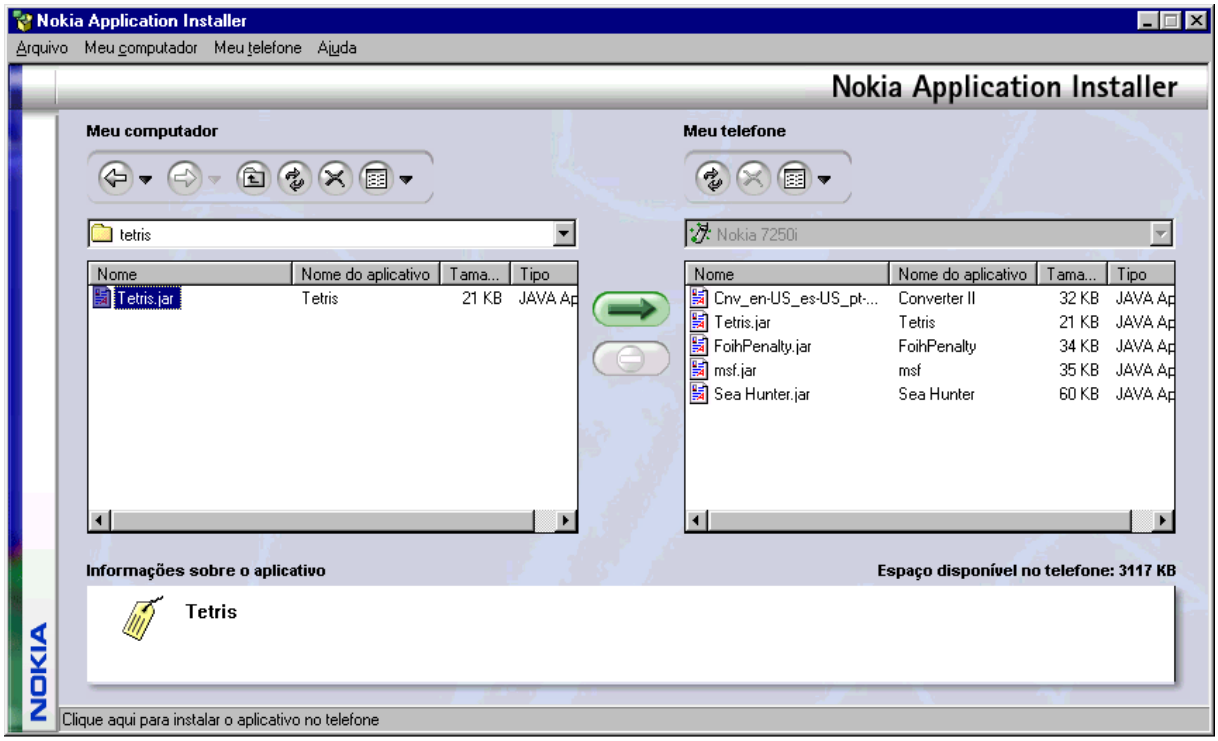


Figura 17 – *Nokia Application Installer*, utilizado para instalar o jogo no celular Nokia

4.5 RESULTADOS E DISCUSSÃO

Como visto, o jogo Tetris foi instalado e testado em um dispositivo real, e observou-se que o mesmo funciona perfeitamente. A diferença entre testar um jogo em um emulador e testar em um dispositivo real é que no aparelho pode-se verificar com maior clareza aspectos como jogabilidade e performance.

Para que o jogo pudesse ser testado em um dispositivo real, foi necessário desenvolvê-lo utilizando apenas recursos do MIDP 1.0, pois no Brasil, os poucos dispositivos que suportam MIDP 2.0 possuem ainda um custo muito elevado. Isso dificultou bastante o desenvolvimento, pois o MIDP 1.0 não possui API específica para jogos e muitos recursos tiveram que ser implementados a fim de suprir suas limitações.

Importante ressaltar que o celular 7250i da Nokia suporta MIDP 1.0, porém não suporta o pacote opcional MMAPI, sendo que para fazer os testes, compilou-se uma versão do jogo utilizando as API’s definidas pela Nokia para execução de som. A fim de testar a o processamento do som do jogo em um dispositivo real.

Outro complicador foi redimensionar o tamanho dos objetos do jogo conforme o tamanho da tela, que apesar dos bons resultados, em telas muito altas ou muito largas o jogo não faz um bom aproveitamento do espaço.

5 CONCLUSÕES

Através das diversas tecnologias citadas neste trabalho, voltadas para o desenvolvimento de jogos e aplicações para dispositivos móveis, algumas ainda muito recentes, pode-se concluir que este é um mercado em expansão e com grande potencial para o desenvolvimento de aplicações, sobretudo jogos.

O presente trabalho mostrou os maiores desafios no desenvolvimento de jogos para dispositivos móveis, como limitações de *hardware*, problemas de interação com o jogador e diferenças entre dispositivos. Mesmo com essas dificuldades, é possível desenvolver um jogo completamente portável, simples e mesmo assim muito atraente, como foi o caso do jogo desenvolvido.

A arquitetura MVC sob a qual o jogo foi desenvolvido, é mais trabalhosa, pois, seria mais prático fazer todo o modelo ligado à interface, como tem sido feito tradicionalmente no desenvolvimento dos jogos. Entretanto, verificou-se que esta arquitetura permite maiores vantagens como: portabilidade, escalabilidade e manutenibilidade. Pois, a medida que a complexidade do jogo aumenta, fatores como esses são imprescindíveis.

O trabalho também contribuiu apresentando as ferramentas utilizadas para o desenvolvimento de um jogo, desde a fase de projeto até a implantação, verificando que elas atendem as necessidades do desenvolvedor e ainda possuem a vantagem de serem todas gratuitas. A metodologia UML também demonstrou-se adequada para a especificação do jogo.

O projeto completo do jogo foi publicado em dois importantes fóruns da tecnologia Java nacionais, são eles Portal Java (PORTALJAVA, 2005) e Java Free (JAVAFREE, 2005). Possibilitando assim que outros desenvolvedores que se interessam por essa tecnologia, aprendam com esta experiência e possam dar continuidade ao trabalho.

Estudos mais específicos sobre a tecnologia J2ME foram realizados durante o desenvolvimento deste trabalho. Esta tecnologia demonstrou ser muito interessante e adequada para o desenvolvimento de aplicações para dispositivos móveis. Foram desenvolvidas e aplicadas diversas técnicas de desenvolvimento de jogos, tais como, renderização em diferentes tipos de displays, efeitos de sombra nos desenhos, efeitos sonoros, entre outros, contribuindo desta forma, para futuras pesquisas nesta área. Pode-se dizer então, que o trabalho atingiu seus objetivos.

5.1 EXTENSÕES

Durante o desenvolvimento surgiram várias idéias para a extensão do jogo. O mesmo, poderia ser implementado em outras plataformas, como o ExEn por exemplo, sendo que a camada de modelo poderia ser completamente reaproveitada, já que o ExEn suporta Java. Outros recursos do jogo poderiam também ser aperfeiçoados, como:

a) implementar uma versão em três dimensões, onde as peças poderiam se movimentar como se estivessem em um tabuleiro tri-dimensional;
b) implementar opção para girar o jogo em 90º para que em telas mais largas os jogadores pudessem utilizar o dispositivo em posição horizontal;
c) implementar opção para configurar as teclas de controle do jogo;
d) adicionar mais recursos de sons;
e) adicionar mais recursos gráficos ao jogo, como por exemplo, piscar as linhas preenchidas antes de apagá-las;
f) implementar opção para o jogador selecionar o nível do jogo, como: fácil, médio e difícil.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, H.O.;LOUREIRO, E.C.F.;NOGUEIRA, W.F. Plataformas para Desenvolvimento de Jogos para Celulares. **INFOCOMP: Journal of Computer Science**, Lavras, v. 4, n. 1, 2005, p. 53-61. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v4.1/art07.pdf>. Acesso em: 02 maio 2005.

AMARO, Pedro Henrique Simões. **The clash of mobile platforms**: J2ME, ExEn, Mophun and WGE, Coimbra, 2003. Disponível em: <http://www.gamedev.net/reference/business/features/clashmobile/>. Acesso em: 05 maio 2005.

ATARI GAMING: **The tetris saga**. [S.1], [1996?]. Disponível em: < http://atarihq.com/tsr/special/tetrishist.html>. Acesso em 11 maio 2005.

BARROS, Tiago Guedes Ferreira. **SymbyG(r)aF – symbyan games framework**. 2003. 51 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Informática, Universidade Federal de Pernambuco, Pernambuco.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuario : o mais avançado tutorial sobre Unified Modeling Language (UML). Rio de Janeiro: Campus, 2000.

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

CARNIEL, Juliano.;TEIXEIRA, Clóvis. **Apostila de J2ME**. [S.l.], 2003. Disponível em <http://www.portaljava.com.br/home/modules.php?name=News&file=article&sid=1653>. Acesso em: 29 maio 2005.

FURLAN, Jose Davi. **Modelagem de objetos através da UML-The Unified Modeling Language**. São Paulo: Makron Books, 1998.

FREIRE, Herval. Revisitando o MIDP 2.0: prepare-se para o novo padrão. **Java Magazine**, Grajaú, n. 16, p. 42-47, 2004.

FREITAS, João Bosco de Barros; REZENDE, Elton Ricelli Ferreira. **Importância da padronização no reuso de código**: design pattern MVC - model view controller. [S.l], [2003?]. Disponível em <http://www.inf.unirondon.br/~elton/arquivos/mvc.pdf>. Acesso em: 06 jul 2005.

GAMMA, Erich et al. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos. Tradução Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

GAILLAND, Alain. **Transmissor infravermelho para PC**. [S.l.], [2004?]. Disponível em: <http://www.clubedohardware.com.br/download/misc/IrDA_xp.zip>. Acesso em 10 nov. 2004.

JAVAFREE: **Exemplos e mini-programasl**. [S.l.] 2005. Disponível em: <http://www.javafree.org/javabb/viewtopic.jbb?t=850196>. Acesso em: 02 jul 2005.

JORNAL DA GLOBO: **Criação digital**. [S.l.], 2004. Disponível em: <http://jg.globo.com/JGlobo/0,19125,VTJ0-2742-20040913-61397,00.html>. Acesso em 02 nov. 2004.

JUDE: **A Java/UML object-oriented design tool**. [S.l.], [2005]. Página oficial do produto. Disponível em: <http://www.esm.jp/jude-web/en/index.html>. Acesso em: 29 maio 2005.

MIRANDA, Cláudio. J2ME no JavaOne 2002: multimídia e jogos no Java Micro Edition. **Java Magazine**, Grajaú, n. 1, p. 16-18, 2002a.

_____. Multimídia no celular: mobile media API (MMAPI). **Java Magazine**, Grajaú, n. 2, p. 24-26, 2002b.

MUCHOW, John W. **Core J2ME**: tecnologia e MIDP. Tradução João Eduardo Nóbrega Tortello. São Paulo: Makron Books, 2004.

NASCIMENTO, Ivo Frazão. **Desenvolvimento de um framework para jogos sobre a plataforma Brew**. 2003. 41 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Informática, Universidade Federal de Pernambuco, Pernambuco.

NOKIA. **Nokia PC suíte.** [S.l.], [2005]. Página oficial do produto. Disponível em <http://www.nokia.com.br/nokia/0,,58322,00.html>. Acesso em: 30 maio 2005.

OBIGO: **Mophun Gaming Platform**. [S.l.], [2005?] Disponível em: <http://www.obigo.com/PSUser/servlet/com.ausys.ps.web.user.servlet.AuthorizedPageServlet?nodeid=1708&pageversion=2>. Acesso em: 10 maio 2005.

PHONEGAMEREVIEW.COM: **In-Fusio's EGE gaming solution for orange signature phones across europe**. [S.l.], 2004. Disponível em: <http://www.phonegamereview.com/phone-game-news/article.asp?a=714>. Acesso em: 10 maio 2005.

PORTALJAVA: **J2ME**. [S.l.] 2005. Disponível em: <http://www.portaljava.com.br/home/modules.php?name=Forums&file=viewtopic&t=22212&sid=a9e125528af63b2432695185ed84ab63>. Acesso em: 02 jul 2005.

QUALCOMM: **A solução BREW**. [S.l.], 2005. Página oficial do produto. Disponível em: <http://brew.qualcomm.com/brew/pt/developer/resources/gs/brew_solution.html> Acesso em: 10 jun 2005.

SABINO, Vanessa Cristiana. Game API: simplicidade e poder em jogos para celulares. **Java Magazine**, Grajaú, n. 10, p. 42-47, 2003.

APÊNDICE A – Interface infravermelho para PC

Para conectar o PC ao celular Nokia, modelo 7250i, a fim de instalar o jogo Tetris, utilizou-se de uma interface infravermelho (IrDA), cujo projeto e montagem são descritos a seguir.

Este projeto trata-se de uma interface genérica, capaz de conectar qualquer equipamento que contenha infravermelho ao PC, desde que a placa mãe do PC contenha um conector infravermelho que pode ser também chamado de "IR", "IRCON", "SIR" ou "SIRCON" dependendo da marca da placa. Isso pode ser verificado através do manual da placa mãe ou pelas configurações do *setup*. Esta interface pode ser encontrada pronta em casas especializadas, porém não é fácil encontrá-la e seu custo costuma ser elevado.

Os componentes necessários para a montagem são:

a) 1 LED infravermelho (emissor);

b) 1 Foto-diodo (receptor);

c) 2 transistores BC 548;

d) 1 capacitor 10nF;

e) 1 resistor 4K7Ω 1/8W;

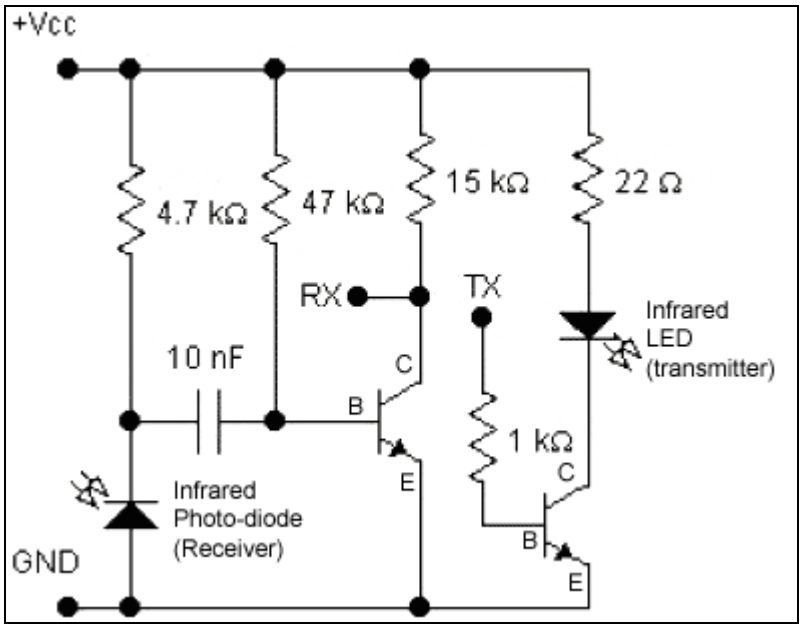
f) 1 resistor 47KΩ 1/8W;

g) 1 resistor 15KΩ 1/8W;

h) 1 resistor 22Ω 1/8W;

i) 1 resistor 1KΩ 1/8W.

O diagrama esquemático do circuito é apresentado na Figura 18.



Fonte: Gailland (2004).

Figura 18 – Diagrama esquemático do circuito

Na montagem do circuito, deve-se ter cuidado especial para posicionar o transmissor (LED) e receptor (foto-diodo) lado a lado. Após a montagem a interface deve ser ligada a placa-mãe observando-se os quatro pontos: Vcc, Gnd, TX e RX. É importante também configurar no *setup* da placa-mãe a interface IR para operar em modo *full-duplex*, para um melhor desempenho, mas caso haja problemas, também pode ser usada em modo *half* .