PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

CÂMPUS URUGUAIANA.

FACULDADE DE ADMINISTRAÇÃO, CONTABILIDADE E INFORMÁTICA.

DEPARTAMENTO DE INFORMÁTICA.

ESTUDO E APLICAÇÃO DAS TECNOLOGIAS ENVOLVIDAS NO DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS ATRAVÉS DA IMPLEMENTAÇÃO DE UM PROTÓTIPO DE JOGO 2D PARA TELEFONES CELULARES

Autor
DIEISON ANTONELLO DEPRÁ

Orientador
MARCUS KINDEL

TRABALHO DE CONCLUSÃO

Uruguaiana, julho de 2004.

AGRADECIMENTOS

Agradeço a Deus por ter me permitido vencer uma grande etapa de minha vida.

Agradeço em especial aos meus pais e irmão, pelo apoio nas horas mais difíceis, nos momentos em que mais precisei, por sua compreensão com meus dias de tensão e mau humor, sempre colaborando em todas as suas possibilidades para que eu pudesse alcançar este objetivo.

Agradeço ao meu orientador Prof. Ms. Marcus Kindel por ter aceitado a proposta deste trabalho e me dispensado toda atenção e suporte necessários.

Aos meus colegas e amigos, pelo companheirismo e complacências com que superamos mais uma etapa.

Agradeço em especial aos meus grandes amigos Alex Jacques da Costa e Flávio Zacarias Fagundes.

Agradeço em especial a toda equipe da RPS SISTEMAS, pelos anos de apoio e compreensão com as dificuldades enfrentadas.

SUMÁRIO

LISTA DE ABREVIATURAS

API - *Application Programming Interface*
CLCD - *Connected Limited Configuration Device*
J2ME - *Java 2 Micro Edition*
JAM - *Java Application Manager*
JVM - *Java Virtual Machine*
MIDP - *Mobile Information Device Profile*
MMA - *MultiMedia API*
MMS - *MultiMedia Message API*
SMS **-** *Short Message Service*
SDK **-** *Software Developer Kit*
TCP **-** *Transmission Control Protocol*
WAP - *Wireless Application Protocol*
WMA - *Wireless Message APII*
WML - *Wireless Markup Language*

LISTA DE FIGURAS

RESUMO

A gigantesca disseminação dos dispositivos móveis, impulsionada principalmente pelos telefones celulares, proporcionou as condições necessárias para o desenvolvimento tecnológico destes dispositivos e de todo mercado que gira em torno deles. Este fato possibilitou a criação de um novo segmento no desenvolvimento de *software* voltado ao ambiente móvel, que gradativamente passa de um simples atrativo ou diferencial, a ser explorado pelo *marketing* das empresas, para uma necessidade.

Paralelamente ao desenvolvimento do ambiente de computação móvel, observamos a explosão na busca por entretenimento, cada vez mais procurado por pessoas de todas as idades, que entre outras coisas fomenta o crescimento exponencial do mercado de jogos computacionais.

O encontro destas duas "ondas" resultou na abertura de um novo nicho mercadológico a ser explorado, com enorme demanda por produtos e profissionais.

Neste contexto, este trabalho apresenta o desenvolvimento de um protótipo de jogo 2D voltado ao ambiente dos dispositivos móveis, denominado AmazonGuard, utilizando a plataforma J2ME, que é um subconjunto de Java2 e conta com inúmeras bibliotecas adequadas ao desenvolvimento de aplicações para este ambiente.

Serão apresentados os conceitos básicos sobre dispositivos móveis e jogos, as ferramentas utilizadas, a plataforma, o protótipo, a implementação além das otimizações e resultados obtidos. Assume-se que o leitor possua conhecimentos em Java.

1. INTRODUÇÃO

No mundo globalizado em que vivemos, a comunicação, objetivando a troca constante de informações, é uma necessidade irrefutável para manter-se atualizado num contexto dinâmico, onde o conhecimento é o principal produto. [1].

Nesse sentido, o advento dos dispositivos móveis constitui o principal recurso tecnológico capaz de proporcionar às pessoas a conectividade desejada. Os *notebooks*, computadores portáteis, foram os pioneiros nessa área e responsáveis por injetar no mercado o ânimo e os investimentos necessários ao desenvolvimento deste nicho mercadológico que deu origem a toda uma linha de dispositivos móveis, assim como: telefones celulares, *Palmtops* [2] e PDAs [3].

A popularização da Internet e do telefone celular, aliada à evolução tecnológica nas áreas de *hardware*, *software* e telecomunicações, proporcionaram as condições necessárias para surgimento de uma nova gama de aplicações, voltadas ao mercado de dispositivos móveis, tanto na área de *e-commerce* [4] como comunicação e entretenimento. Neste cenário, as empresas do setor direcionam seu foco aos debates e pesquisas, investindo pesado para obter o domínio das tecnologias e ferramentas que possibilitem suprir as exigências peculiares às aplicações deste ambiente.

Na área de entretenimento supracitada, em especial a subárea de jogos merece uma atenção especial, pois essa indústria está em pleno desenvolvimento, tem movimentado cifras espantosas no mercado mundial [5] e será o pano de fundo deste trabalho.

Pode-se observar que o aumento no interesse do público por jogos é uma constante desde a época dos primeiros *videogames* comerciais como Atari [6], Megadrive [7] e Nintendo [8], passando pelos computadores pessoais (PCs) e modernos videogames até chegar aos "recentes" dispositivos portáteis.

A construção e disponibilidade de jogos em dispositivos portáteis como telefones celulares começou timidamente com dois objetivos básicos:

a) Ilustrar as funcionalidades dos dispositivos.

b) Apelo de *marketing* das empresas com intuito de ganhar mercado.

No entanto, as limitações encontradas nestes jogos, na maioria decorrentes da carência de recursos dos próprios dispositivos, decepcionaram usuários acostumados com o alto nível e a qualidade de acabamento dos jogos para PCs e *videogames* contemporâneos. Atentos a este fato e tendências da atualidade, os fabricantes vêm investindo fortemente em pesquisa e desenvolvimento de novas arquiteturas com capacidade de atender aos anseios e expectativas de seus consumidores. Como resultado desse processo, os últimos lançamentos das empresas líderes de segmento, como Motorola [9], Nokia [10], Siemens [11] e SonyEricsson [12] contam com inovações como tela colorida, *vibracall*, tons polifônicos, comando de voz, câmera digital, MP3 *players* e suporte a tecnologias como WAP [13] e J2ME [14].

Neste cenário, o desenvolvimento de aplicações para dispositivos móveis aparece como uma das mais promissoras áreas dentro da indústria de *software* para os próximos anos [15]. A evolução do *hardware* desses dispositivos e a adoção de padrões, está proporcionando um substancial incremento nas possibilidades de desenvolvimento de aplicações para essas arquiteturas [16].

Apoiado nessas realizações e nas perspectivas de mercado, o ramo de desenvolvimento de jogos para dispositivos móveis, principalmente para telefones celulares, está ganhando força e passa a ser motivo de debate nos principais fóruns de discussão da área, despertando o interesse das principais empresas da indústria [17].

Entretanto, segundo Fernando A. M. Trinta:

> Apesar do sucesso na área industrial/comercial, o estudo sobre jogos é um campo pouco explorado na área acadêmica, podendo-se dizer que havia até um certo preconceito com a área. Esta situação era criada por uma série de fatores como a falta de instituições de pesquisa interessadas no tema, a escassez de livros e congressos, bem como de especialistas no meio acadêmico. Porém, este cenário começa a mudar. De fato, pode-se dizer que poucas são as áreas da computação onde é possível se explorar tantos domínios do conhecimento quanto na área de jogos [18].

Sendo assim, sintonizado com as questões da atualidade, o presente trabalho consiste na apresentação do estudo realizado sobre as tecnologias envolvidas no desenvolvimento de aplicações com capacidade de operar sobre uma arquitetura de dispositivos móveis, mais especificamente telefones celulares, através da implementação de um protótipo de jogo 2D.

1.1 OBJETIVOS

Este trabalho possui uma série de objetivos distintos, alguns genéricos, outros mais restritos, que serão classificados, para facilitar sua abordagem, em gerais e específicos.

1.1.1 Objetivos Gerais

Este projeto teve como objetivo geral o estudo das principais questões relacionadas ao desenvolvimento de aplicações para dispositivos móveis através de aplicação prática na implementação de um protótipo de jogo 2D para telefones celulares. Dentre estas principais questões podemos citar:

- Identificar a plataforma alvo para desenvolvimento;
- Dominar as tecnologias a serem utilizadas;
- Descobrir e dominar as ferramentas disponíveis que possam auxiliar o desenvolvimento do projeto;

- Atingir o estado da arte no desenvolvimento de jogos 2D voltados para dispositivos móveis.

A metodologia para análise e seleção definitiva destas questões levou em consideração aspectos como: disponibilidade do dispositivo no mercado regional, formas de comunicação e transmissão de dados, ambientes e SDKs oferecidos, documentação e especificações técnicas disponíveis, além das possibilidades de consolidação do dispositivo perante o público alvo.

Partindo das análises anteriores, tem-se como objeto final um jogo 2D para telefones celulares, que proporcionasse a aplicação prática das tecnologias e ferramentas pesquisadas, além da familiarização com problemas inerentes ao desenvolvimento de jogos.

1.1.2 Objetivos Específicos

Medindo a complexidade e a diversidade dos problemas envolvidos no desenvolvimento de um jogo digital, seja ele para *videogames*, PCs, ou dispositivos móveis, parece sensato traçar uma lista de objetivos específicos, que aponte desafios especiais a serem superados durante a missão em busca do objetivo maior.

Os objetivos específicos são de extrema importância, pois servem como marcos referenciais, indicando etapas imediatas a serem enfrentadas e, quando atingidos, permitem avaliar de forma mais clara o estágio atual dentro do projeto como um todo. A seguir serão destacados alguns destes objetivos.

Um dos primeiros objetivos específicos a ser superado foi a análise e compreensão das alternativas existentes tanto em termos de tecnologia de suporte ao desenvolvimento de aplicações para dispositivos móveis, como das arquiteturas, especificações técnicas e disponibilidade de mercado dos telefones celulares.

Outro estágio, que se buscou atingir, foi o de lucidez sobre as diversas questões de complexidade técnica envolvidas no desenvolvimento de um jogo, como

modelagem do mundo virtual, detecção de colisões, técnicas de inteligência artificial, otimizações de código, entre outras.

Obter capacitação técnica para aplicações de novas tecnologias em um novo e promissor mercado são fonte de vantagem competitiva para qualquer profissional e, também, constitui um dos objetivos específicos deste trabalho.

1.2 MOTIVAÇÃO E JUSTIFICATIVA

Posso dizer que concordo com minha ex-colega Cintya Bortolazzo [19] quando diz que a "[...] principal fonte de motivação para se alcançar qualquer objetivo seja a paixão [...]", pois quando nos dedicamos plenamente aos nossos objetivos encontramos a força necessária para atravessar os momentos difíceis, a criatividade indispensável para transpor obstáculos que se erguem e principalmente a persistência inquestionável, sempre acreditando na vitória final mesmo quando perdemos algumas batalhas. Desde minha infância, quando tive os primeiros contatos com jogos digitais nos saudosos tempos do Atari, cultivo a curiosidade em descobrir como os jogos são construídos e alimento o desejo de criar meus próprios jogos.

Agora, deixando de lado fatores sentimentais, posso tranqüilamente afirmar que a possibilidade de aplicar, num trabalho prático, uma diversidade fantástica de conceitos abordados ao longo do curso de graduação, aliado ao ar desafiador de encarar a complexidade inerente ao desenvolvimento de jogos, me faz sentir motivado uma vez que põe a prova minha capacidade de adaptação, sendo uma excelente forma de preparação para enfrentar um novo mercado, o qual foge do meu cotidiano.

Minhas justificativas para escolha do tema estão baseadas na observação das tendências tecnológicas nas áreas de microeletrônica, telecomunicações, dispositivos móveis e entretenimento, além da constatação dos surpreendentes resultados obtidos com o programa de pesquisa e desenvolvimento de jogos para dispositivos móveis implementado pelo CESAR [20].

Não obstante isso, a exploração de novas tecnologias, novas arquiteturas, novas plataformas e ferramentas de trabalho abrem possibilidade de capacitação

profissional aplicada, formando as bases de sustentação para futuros trabalhos voltados para fins comerciais.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho é formado por onze capítulos, sendo o primeiro a presente introdução. Os demais capítulos abordam os seguintes assuntos:

- **Capítulo 2, Conceitos Básicos**: onde são apresentadas as características fundamentais dos dispositivos de comunicação móvel, bem como os conceitos básicos dos jogos computacionais. Tais características envolvem principalmente recursos e formas de comunicação. Já os conceitos, abordam questões como classificação de jogos em: tipos, gêneros e categorias.

- **Capítulo 3, Plataforma**: onde é apresentada de forma resumida a plataforma selecionada para desenvolvimento do protótipo, bem como as tecnologias oferecidas pela plataforma para auxiliar o desenvolvimento de aplicações para o ambiente de computação móvel.

- **Capítulo 4, Ferramentas**: traz a apresentação dos diferentes tipos de ferramentas utilizadas para o desenvolvimento do protótipo. Estas ferramentas incluem ambientes de programação, manipulação de imagens, avaliação de desempenho, entre outras.

- **Capítulo 5, AmazonGuard**: discute os principais aspectos da modelagem do protótipo. Fornece uma visão geral dos componentes e funcionamento do jogo.

- **Capítulo 6, Implementação**: comenta os principais recursos empregados, as técnicas para detecção de colisão, controle de fim de jogo, inteligência dos inimigos, além dos possíveis estados do jogo.

- **Capítulo 7, Simulação**: debate a simulação do protótipo em diferentes simuladores disponíveis, analisando portabilidade, robustez, desempenho e usabilidade.

- **Capítulo 8, Otimizações**: apresenta diversas otimizações realizadas no protótipo, abordando as principais implicações de cada tipo de otimização.

- **Capítulo 9, Resultados**: relata os problemas encontrados, comenta as alternativas e as soluções adotadas.

- **Capítulo 10, Considerações finais**: apresenta as últimas considerações a respeito do trabalho como um todo, dos estudos realizados e da aplicação prática de conhecimento para desenvolvimento do protótipo.

Por fim, no último capítulo, são apresentadas as referências bibliográficas utilizadas como suporte ao presente trabalho.

2. CONCEITOS BÁSICOS

Este trabalho fez uso de duas linhas básicas de pesquisa, uma relacionada aos dispositivos de comunicação móvel e outra voltada aos jogos computacionais, sendo que ambas apresentam uma série de peculiaridades fundamentais que afetam diretamente o projeto e desenvolvimento de aplicações que envolvam estas duas linhas. Dessa forma, este capítulo está dividido em duas seções, que serão apresentadas na seqüência, cada uma abordando as principais características e implicações de cada um dos temas.

2.1 DISPOSITIVOS MÓVEIS

O termo "dispositivo móvel" atualmente tem sido empregado para designar uma série de dispositivos portáteis equipados com algum tipo de processador. Estes dispositivos podem variar de computadores (*notebooks*) até telefones celulares, passando pelos *PALMs* e uma infinidade de outros *PDAs* e *handhelds* disponíveis no mercado. Para o escopo deste trabalho, a expressão será adotada, mais especificamente, como sinônimo de telefones celulares e *PDAs*.

Com a crescente demanda por novas aplicações e as diversas inovações tecnológicas nas áreas de *hardware* e de telecomunicações, o projeto de desenvolvimento de *software* para o ambiente móvel deve ao menos levar em consideração quatro questões básicas: a arquitetura alvo, a plataforma de desenvolvimento, a tecnologia e a forma de comunicação.

2.1.1 Arquitetura

Os dispositivos móveis são constituídos por arquiteturas proprietárias e não padronizadas que variam muito de um fabricante para outro e mesmo entre dois modelos de um mesmo fabricante, portanto a especificação e a análise das arquiteturas não serão apresentadas aqui e fogem ao escopo deste trabalho.

Mesmo evitando a apresentação de uma análise sobre as arquiteturas existentes, há uma série de questões relacionadas aos recursos disponíveis (principalmente a falta destes) [21], que afetam diretamente o desenvolvimento de aplicações para este ambiente e serão sucintamente descritos a seguir [22]:

- **Portabilidade**: esses dispositivos, como o próprio nome já diz são móveis, podendo ser facilmente transportados de um lugar para outro, trocando de cidade, estado ou até mesmo de país; logo, deve haver uma preocupação especial com a internacionalização das aplicações;

- **Conectividade**: característica primordial desses dispositivos, sendo assim, esse fator deve ser levado em consideração a fim de aproveitar uma característica inerente ao meio;

- **Processador**: para desenvolvedores acostumados com processadores de computadores pessoais com freqüências na faixa dos *gigahertz*, entrar nesse ambiente, onde as freqüências oscilam em poucas dezenas de *megahertz*, é realmente um desafio e exige a reavaliação do desenvolvimento de *software*.

- **Memória**: para os desenvolvedores, acostumados com computadores *desktop*, espaço de memória não é um grande problema. No entanto, no ambiente móvel é um ponto vital e decisivo para a viabilidade de qualquer *software*. Aqui os equipamentos possuem tipicamente entre 128KB e 512KB de memória disponível.

- **Tela**: a apresentação da aplicação na tela deve receber uma atenção especial dos desenvolvedores, uma vez que pode ser afetada tanto pelo

tamanho da tela, que pode variar de 95x65 até 320x240 *pixels*, quanto pela capacidade de suporte a cores.

- **Latência**: a largura de banda disponível nos telefones celulares, apesar dos esforços e inovações tecnológicas na área de redes de telecomunicações, ainda é extremamente estreita e o tratamento adequado do intervalo de tempo entre uma requisição (via rede) e o recebimento de sua resposta deve ser cuidadosamente planejado durante a elaboração do projeto da aplicação.

- **Interface com o usuário**: aspecto fundamental para interação humana com qualquer elemento computacional, a interface com o usuário nos dispositivos móveis ainda enfrenta alguns problemas, como a utilização de um teclado otimizado que dificulta a digitação de textos e uma resolução de tela limitada que dificulta a apresentação e organizações das informações.

2.1.2 Comunicação

Atualmente os telefones móveis são os dispositivos computacionais de comunicação em redes mais naturais, pois enviar e receber mensagens constitui sua funcionalidade primordial.

No ambiente de computação móvel, existem várias formas de realizar a comunicação entre dois dispositivos, utilizando-se para isso diversas tecnologias distintas, presentes na maior parte dos aparelhos de última geração. A seguir, serão enumeradas as principais tecnologias suportadas [23]:

- **SMS e MMS**: troca de mensagens de texto e/ou multimídia entre os dispositivos sobre as redes das operadoras.

- **TCP/IP**: comunicação utilizando a rede de comunicação das operadoras como meio de acesso a Internet.

- **Interface serial**: ligação física entre dois dispositivos, ou entre um dispositivo e um PC através de um cabo serial.

- **Infravermelho**: ligação lógica entre dois dispositivos, ou entre um dispositivo e um PC através de sinal infravermelho.

- ***Bluetooth***: ligação entre dispositivos via ondas de rádio de curto alcance.

Estas características são comuns a maioria dos dispositivos móveis e foram consideradas as que mais afetam o desenvolvimento de *software* para estes dispositivos. Torna-se evidente que existem muitas outras características importantes, no entanto, sua presença está vinculada a fabricantes ou a modelos específicos de dispositivos móveis e, portanto, não serão abordadas.

2.2 JOGOS COMPUTACIONAIS

Como dito anteriormente, o desenvolvimento de jogos computacionais é uma atividade multidisciplinar de grande complexidade. As contingências envolvem o conhecimento e o domínio de técnicas, teorias e conceitos das mais diversas áreas, "[...] no campo da computação, podemos citar como exemplo as de computação gráfica, inteligência artificial, computação musical, redes de computadores, engenharia de *software*, entre outras" [18].

Segundo Marcelo Johann,

> Jogos de computador apresentam uma necessidade acentuada de poder computacional. Os recursos visuais, sonoros e de controle desejados estão sempre além da tecnologia disponível. Por esta razão, os programadores de jogos sempre extraíram o máximo da tecnologia existente, por exemplo, usando modos de operação especiais das placas de vídeo, programando em linguagem *assembly* para ter código mais otimizado e principalmente usando tabelas de resultados pré-computados [24].

Apesar das diferenças existentes entre o desenvolvimento de jogos convencionais e os jogos para dispositivos móveis, existe uma gama de conceitos e características comuns que permeiam todo processo de desenvolvimento de jogos.

Nas subseções a seguir são apresentados os componentes básicos de um jogo, as diversas questões relacionadas e as etapas do projeto de um jogo.

2.2.1 Componentes

Os jogos computacionais são compostos por pelo menos quatro partes básicas: modelagem, visualização, controle, além das ações e movimentos. Cada uma destas partes deve ser cuidadosamente observada durante a elaboração de um jogo.

Como diria Grady Booch [25] "[...] um modelo é uma simplificação da realidade [...]". A modelagem, também pode ser interpretada como a construção de todo o universo virtual. Este é um processo artístico e muito trabalhoso que envolve a elaboração de todos os objetos que irão compor os cenários com base na história do jogo [26].

A visualização trata das visões da cena, ou de parte dela, do ponto de vista do observador. As visões englobam todo o processo de seleção, recorte, eliminação de faces ocultas e pintura que deve ser realizada a cada vez que a cena é exibida, normalmente entre 15 e 30 vezes por segundo [18].

As ações de um jogo são comportamentos atribuídos a objetos segundo determinados eventos, enquanto os movimentos envolvem uma série de problemas típicos de jogos, como por exemplo, animações de personagens, detecção de colisões e simulação de física [26].

O mecanismo de controle envolve a conjugação da modelagem lógica com a modelagem física do jogo, dando funcionalidade ao mesmo. Algumas das questões abordadas pelo mecanismo de controle são as movimentações do personagem principal, os objetivos, os comportamentos e a jogabilidade [26].

2.2.2 Fatores relacionados

Existe uma gama de diferentes fatores implícitos ao desenvolvimento de jogos, mas que também merecem uma atenção especial. Dentre estes fatores podemos citar [27]:

- **Tipo**: que se refere ao número de jogadores para o qual o jogo é projetado, podendo ser basicamente dois: de um único jogador (*single-player*) ou de múltiplos jogadores (*multi-player*);

- **Modalidade**: quanto à forma de realização do jogo, existem jogos baseados na troca de mensagens, jogos para trabalhar através de navegadores, além dos jogos de console simples ou em rede;

- **Gênero**: relaciona-se com o tipo de interação ou desafio que o jogo proporciona. Não existe uma taxonomia uniforme e bem definida a este respeito, entretanto, pode-se apresentar como exemplos os jogos de ação, de esportes, de aventura, de estratégia, de enigmas, corrida, entre outras.

Além destes fatores, existem outros diversos, como a preocupação com as questões sociais, políticas, religiosas, financeiras e éticas que fazem parte desse contexto.

2.2.3 Etapas do projeto de um jogo

A elaboração de um jogo é um processo longo e trabalhoso que normalmente envolve uma série de etapas como definições lógicas e semânticas, plataforma alvo, modelagem, interface, representação gráfica, além da elaboração dos algoritmos.

A construção de um jogo passa necessariamente pela criação de uma história, um enredo, que deve ser cuidadosamente elaborado e conter toda trama de definições semânticas necessárias à implementação, bem como todo detalhamento que possibilite a compreensão da lógica do mesmo.

A compreensão exata da plataforma alvo é fundamental, pois jogos sempre estão no extremo da utilização dos recursos existentes. Logo, sua construção deve levar em consideração a plataforma sobre a qual o mesmo vai ser executado, a fim de garantir a alocação correta e o uso eficiente dos recursos disponíveis.

Mapear os objetos do mundo real, componentes da definição lógica e semântica, em elementos do universo virtual constitui uma etapa fundamental do

projeto de jogos, sendo assim, a correção da modelagem influencia de forma direta no êxito ou fracasso do projeto.

Um dos principais aspectos da construção de jogos é a definição dos mecanismos através dos quais o usuário poderá interagir com o jogo. Sua especificação deve levar em consideração os recursos oferecidos pela plataforma, além de questões ergonômicas e cognitivas [27].

Em um primeiro momento, a visão ou concepção que o usuário tem de um jogo é dada através da observação e da representação visual/gráfica do mesmo. Normalmente são despendidos grandes esforços, num trabalho praticamente artístico, para alcançar uma boa qualidade gráfica [26].

A diversidade de problemas inerentes ao desenvolvimento de jogos computacionais, torna a tarefa de elaboração e implementação de bons algoritmos uma tarefa extremamente complexa. Tais problemas podem envolver, entre outros, técnicas de combate dos adversários do jogador, características de movimentação dos personagens e efeitos da aplicação das leis da física sobre os mesmo, além de rotinas de IA responsáveis pelo comportamento dos oponentes do jogador.

Finalmente, porém não menos importante, aparece a documentação que deve ser redigida de forma clara, de acordo com padrões bem definidos e conhecidos por toda a equipe, devendo ser seguida metodicamente.

3. PLATAFORMA

Atualmente existem várias plataformas que oferecem suporte ao desenvolvimento de aplicações para dispositivos móveis, dentre as quais se destacam: J2ME, Symbian OS [28], Linux, Palm OS [29] e Windows CE.

Conforme pesquisa e seleção realizada, durante o desenvolvimento deste trabalho, a plataforma alvo para desenvolvimento do protótipo foi J2ME. Na próxima subseção será apresentada a plataforma.

3.1 J2ME

O J2ME é um subconjunto da plataforma Java2, otimizada para o desenvolvimento de aplicação sob dispositivos móveis, com grandes limitações técnicas em termos de recursos quando comparados com PCs. [30].

A plataforma J2ME compreende um conjunto de padrões e tecnologias que devem ser suportados pelos dispositivos para que os mesmos possam ser classificados como compatíveis com a plataforma. [30].

Esses padrões e tecnologias definem as bibliotecas que devem ser suportadas pela arquitetura, além de exigir a presença de uma série de recursos e funcionalidades mínimas e máximas nos dispositivos. Uma visão geral da arquitetura que compõe a plataforma J2ME pode ser observada na Figura 3.1:

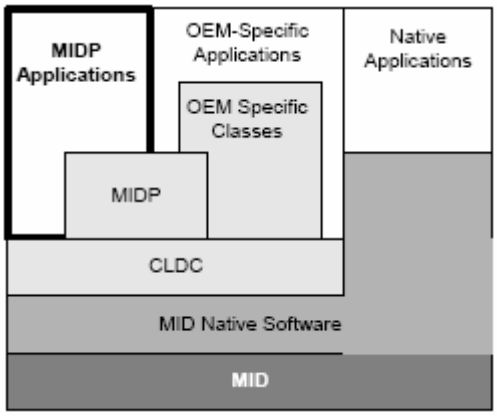

Figura 3.1 - Arquitetura geral da plataforma J2ME (retirada de [34]).

Estas definições são especificadas com maior separação e riqueza de detalhes nos padrões CLDC [31] e MIDP [32], os quais definem os recursos em termos de *software* e *hardware* respectivamente e serão apresentados nas duas próximas subseções. É importante salientar que a plataforma e os padrões, CLCD e MIDP, definem todos os recursos necessários à execução do protótipo desenvolvido.

3.2 CLDC

O *Connected Limited Device Configuration*, ou simplesmente CLDC, constitui um padrão de indústria, que define o suporte obrigatório a uma máquina virtual, conhecida como KVM (*Kilobyte Virtual Machine*), responsável pela interpretação e tradução dos *bytecodes* das aplicações Java para o conjunto de instruções da arquitetura, além de um conjunto mínimo de recursos e funcionalidades que o *hardware* deve suprir para garantir a correta execução das aplicações destinadas a essa plataforma [33].

3.3 MIDP

O *Mobile Information Device Profile*, ou simplesmente MIDP, também constitui um padrão de indústria, que por sua vez exige a existência de um *software* responsável pelo gerenciamento das aplicações Java, conhecido como JAM (*Java Application Manager*), rodando sobre a arquitetura, além de uma biblioteca com um conjunto de classes e pacotes mínimos que devem ser providos pelo dispositivo [34].

4. FERRAMENTAS

Para realização de qualquer tarefa, a utilização de um conjunto de ferramentas adequadas é fundamental para que a execução da mesma seja a mais eficiente possível. No desenvolvimento de *software,* este fato é ainda mais importante devido a diversos fatores, tais como: a grande complexidade envolvida, as regras, métodos e padrões que devem ser empregados de forma metódica e recorrente.

A seleção das ferramentas a serem utilizadas neste trabalho levou em consideração aspectos como a familiaridade com a ferramenta, a adequação com a tarefa, a disponibilidade e a forma de distribuição, além de intensa pesquisa na Internet sobre relatos de usuários.

Com base nos fatos acima mencionados, as ferramentas utilizadas neste trabalho foram agrupadas em cinco categorias, de acordo com sua finalidade e serão apresentadas nas próximas seções.

4.1 AMBIENTE DE DESENVOLVIMENTO

O carro chefe das ferramentas de desenvolvimento de *software* é o ambiente de desenvolvimento. Portanto, sua escolha pode influenciar diretamente na qualidade, no tempo e nos custos de desenvolvimento.

O Eclipse, ferramenta escolhida como ambiente de desenvolvimento para este projeto, na verdade é muito mais que um simples ambiente de desenvolvimento de

software, ele constitui uma plataforma completa, com capacidade de operar sobre diversos sistemas operacionais.

Uma apresentação mais detalhada sobre todos os *plug-ins*, nativos ou não, que agregam recursos e facilidades a plataforma Eclipse, pode ser encontrada na própria documentação do *software* [35], que também serviu como base para descrição dos itens considerados mais importantes e que são enumerados a seguir:

- **Editor de texto**: o Eclipse conta com um poderoso editor de texto que pode reconhecer diversas sintaxes (como HTML, Java, C, JSP, EJB, XML, Cobol) e conta com recursos como auto complementação e verificação de sintaxe em tempo real;

- **Organização do projeto**: os projetos podem ser bem organizados em estruturas de árvores, separando código fonte, recursos, documentação e código final;

- **Perspectivas**: o usuário pode personalizar a disposição, quantidade e conteúdo das ferramentas oferecidas para gerenciamento do projeto, criando perfis que possam se adaptar ás diferentes realidades de cada projeto;

- **GUI e RAD**: possibilita o desenvolvimento de aplicações a partir de componentes visuais com os quais o usuário deve interagir, favorecendo o desenvolvimento rápido das aplicações;

- **Compilação**: permite a configuração dos compiladores e bibliotecas necessárias à compilação integrada dos programas;

- ***Plugin* Eclipse ME**: conta com um *plugin* especialmente desenvolvido para oferecer suporte ao desenvolvimento de aplicações para a plataforma J2ME.

Além de todos os recursos e facilidades mencionadas deve-se acrescentar o fato deste *software* ter licença pública, sendo distribuído de forma gratuita [36].

4.2 MANIPULAÇÃO DE IMAGENS

A criação de um jogo passa obrigatoriamente pela etapa de criação dos cenários, elaboração dos personagens e demais elementos visuais que compõem o mundo virtual. Dessa forma, a utilização de ferramentas para manipulação de imagens é uma peça chave no processo de desenvolvimento de aplicações com refinada representação gráfica.

Devido a falta de domínio total sobre todas as funcionalidades de uma boa ferramenta para manipulação de imagens, foi necessário utilizar três *softwares* diferentes para obter os resultados desejados. Estas ferramentas foram:

- **CorelDraw**: foi utilizado para criação dos personagens e aplicação de efeitos de transformação, como rotação e inversões [37];

- **Adobe PhotoShop**: utilizado principalmente na elaboração do cenário, na aplicação de recorte e na manipulação de camadas [38];

- **Macromedia FireWorks**: aplicado no acabamento e aplicação de transparência ao fundo das imagens, além de ser imprescindível para exportação das imagens para o formato PNG, com otimização significativa no tamanho das mesmas [39].

Provavelmente o conhecimento aprofundado de apenas uma destas ferramentas apresentadas já permitisse a eliminação das demais. No entanto, a utilização combinada das mesmas foi a alternativa encontrada para superar a deficiência de conhecimentos específicos.

4.3 SIMULAÇÃO

O desenvolvimento de *software* sempre exige uma fase de testes, onde o funcionamento de uma aplicação é colocado a prova, para verificar se a mesma consegue responder de forma adequada às especificações.

Testar uma aplicação na própria plataforma onde esta deverá funcionar é a maneira mais segura de obter resultados convincentes. Este fato caracteriza um problema para aqueles que desenvolvem aplicações para dispositivos móveis, uma vez que a plataforma de desenvolvimento difere da plataforma de execução. Para sorte dos desenvolvedores, existe atualmente uma boa quantidade de sistemas que fazem a simulação fiel de todas as características de um dispositivo móvel, permitindo inclusive a simulação de outros sistemas.

Para realização deste trabalho procurou-se utilizar todos os sistemas de simulação disponíveis de forma gratuita, que em sua grande maioria foram desenvolvidos e disponibilizados pelos próprios fabricantes de dispositivos móveis, fato que propõe credibilidade a sua qualidade e fidelidade com o dispositivo real. A seguir serão enumerados os sistemas utilizados:

- ***Wireless ToolKit 2.1***: este *software* é considerado o estado da arte das ferramentas de simulação voltadas a plataforma J2ME; desenvolvido pela Sun Microsystens como referência para os fabricantes de dispositivos móveis [40];

- ***Siemens Mobility ToolKit***: *software* desenvolvido pela Siemens para simulação e depuração de aplicações voltadas à plataforma J2ME sobre suas arquiteturas [41];

- ***iDEN SDK for J2ME***: *software* desenvolvido pela Motorola para simulação e depuração de aplicações voltadas à plataforma J2ME sobre suas arquiteturas [42];

- ***Nokia Developer´s Suite 2.1 for J2ME***: *software* desenvolvido pela Nokia para simulação e depuração de aplicações voltadas à plataforma J2ME sobre suas arquiteturas [43];

- ***Sony Ericsson WTK 2.1***: extensão ao *Wireless ToolKit 2.1* da Sun desenvolvido pela Sony Ericsson com intuito de personalizar o *software* para atender as peculiaridades de suas arquiteturas [44].

Os *softwares* acima mencionados proporcionam, além da simulação e depuração, algumas funcionalidades e/ou extensões que permitem a assinatura e avaliação de desempenho das aplicações.

4.4 OBFUSCADORES

Os obfuscadores são um tipo de ferramenta de extrema importância para os desenvolvedores de aplicações para a plataforma Java. Estes sistemas trabalham sobre o código fonte, analisando-o e gerando uma versão intermediária deste código para depois realizar o processo de compilação.

Durante a geração desta versão intermediária do código, o obfuscador troca os nomes de identificadores, incluindo aí as classes, os métodos e as variáveis ou objetos. Esse processo afeta apenas os arquivos de *bytecodes* (os arquivos com extensão ".class"), tendo dois propósitos: o primeiro de deixar os *bytecodes* mais confusos, dificultando assim, a ação dos decompiladores e o segundo, de reduzir o tamanho final do código objeto diminuindo o número de caracteres utilizados.

O Proguard 2.1 foi o obfuscador escolhido para este trabalho. Entretanto, podem ser obtidos na internet diversos obfuscadores [45].

5. AMAZONGUARD

AmazonGuard foi o nome dado ao protótipo de jogo 2D para dispositivos móveis desenvolvido durante a realização deste trabalho. O jogo pode ser classificado como sendo do tipo de único jogador, gênero aventura, da modalidade console, uma vez que é totalmente executado sobre o dispositivo móvel.

O enredo do jogo nos leva a um contexto onde o personagem principal, um avião de guerra da Força Aérea Brasileira (FAB), é comandado pelo jogador na missão de defesa do espaço aéreo nacional sobre a região da Amazônia, combatendo os traficantes de drogas e possíveis invasores.

Nas próximas seções, o jogo será apresentado de acordo com sua modelagem, em três diferentes visões dos aspectos que o afetam, que são a modelagem conceitual, a modelagem lógica e a modelagem gráfica.

5.1 MODELAGEM CONCEITUAL E SEMÂNTICA

A modelagem conceitual e semântica de um jogo, deve-se levar em consideração todas as questões relacionadas às regras, aos objetivos, à movimentação e aos comportamentos através dos quais o jogo e seus personagens respondem ou devem reagir.

O objetivo do jogo é que o jogador proteja o espaço aéreo na região amazônica, eliminando os invasores que vão aparecendo. Para atingir seu objetivo, o

jogador deve se preocupar com a eliminação do chefe, dos inimigos e, principalmente, manter-se vivo, acabando momentaneamente com a ameaça.

O jogador inicia o jogo com três vidas, ou três chances. A cada colisão de seu personagem contra os invasores ou tiros do chefão, ele perde uma vida. Caso o número de vidas do jogador atinja a quantidade zero, o jogo entra no estado encerrado e o jogador perde a disputa.

Para possibilitar melhor qualidade de desafio e disputa, o jogo conta com um sistema graus de dificuldade, dividido em três níveis: fácil, médio e difícil. A alteração dos graus afeta algumas variáveis do jogo, como velocidade de deslocamento dos inimigos, número máximo de inimigos simultâneos, o número mínimo de inimigos que devem ser destruídos antes de enfrentar o chefão e o multiplicador de pontuação.

Cada inimigo atingido pelo jogador reverte-se em um número de pontos, que varia de acordo com o grau de dificuldade selecionado e que são adicionados ao placar do jogo.

O avião do jogador pode usar três tipos de armamentos: tiros de metralhadora, mísseis e laser. Apenas um tipo de armamento pode ser utilizado a cada instante. O jogador inicia com a metralhadora e a cada dez (10) inimigos destruídos consecutivamente, sem deixar nenhum escapar, um novo tipo de arma é gerado e lançado ao cenário, até que se atinja a arma de maior poder de destruição, o laser. Cabe ao jogador colidir contra este armamento que vai se deslocando pelo cenário, girando 360 graus.

Depois de obter a arma do tipo laser, a cada trinta (30) inimigos destruídos em seqüência, sem que nenhum escape, uma nova vida é lançada ao cenário, a título de bônus, sendo que ela obedece ao mesmo conjunto de regras das armas sorteadas.

Ao atingir o estágio de enfrentamento com o chefão do jogo, os inimigos param de ser gerados e é travado um duelo entre o jogador e o chefão, que pode disparar tiros direcionados. Após a destruição do chefão, o jogo entra no estágio “jogo encerrado”, exibindo ao jogador a mensagem “jogo vencido”.

5.2 MODELAGEM LÓGICA

A modelagem lógica do jogo, por se tratar de um projeto que utiliza o paradigma orientado a objetos, consiste da definição das classes, métodos e atributos, que foram implementados a fim de atender as definições conceituais e semânticas.

Como a discussão individual de cada um desses elementos seria extensa e cansativa, optou-se pela apresentação do diagrama de classes simplificado do projeto, que pode ser observado na Figura 5.1, seguido de pequena explanação sobre o relacionamento entre as entidades que o compõem.

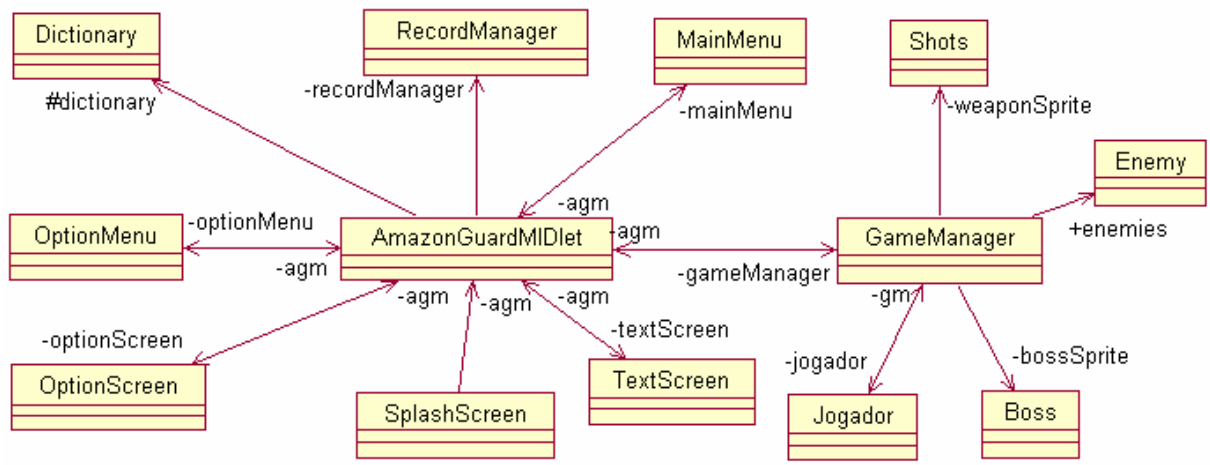

Figura 5.1 - Diagrama simplificado de classes do projeto.

Fica evidente que o diagrama de classes apresentado não é o melhor do ponto de vista da modelagem de *software* orientado a objetos. Entretanto, a definição deste modelo teve como principal objetivo reduzir os efeitos colaterais da orientação a objetos, como a subclassificação e a sobrecarga de métodos e atributos, que tem impacto direto sobre a utilização de recursos tão escassos no ambiente dos dispositivos móveis, como memória e a capacidade de processamento.

A classe responsável pela inicialização da aplicação é a classe AmazonGuardMIDlet. Após a aplicação ser selecionada pelo usuário para executar, um objeto desta classe é criado pelo gerenciador de aplicações Java (JAM), o qual também chama o método "*Startapp( )*" da referida classe. Este método, por sua vez, cria uma instância da classe SplashScreen, que é responsável pela exibição da tela de

apresentação do jogo, a qual é executada em paralelo com o método de inicialização da aplicação. As tarefas do método de inicialização incluem a criação de instâncias das classes Dictionary (responsável pela internacionalização das mensagens exibidas ao usuário), RecordManager (responsável pela gravação e recuperação das configurações do jogo na memória não volátil do dispositivo), GameManager (responsável pelo gerenciamento e controle do jogo), além de uma instância da classe MainMenu que é encarregada da apresentação do menu principal ao usuário.

De acordo com a seleção do usuário no menu principal um conjunto específico de tarefas é executada. Dentre estas opções a principal é a opção "Novo jogo", pois nesse caso o método "*init( )*" da classe GameManager é acionado. A execução desse método cria todos os objetos necessários à execução do jogo, colocando-o em funcionamento.

A abordagem superficial do relacionamento entre as entidades que compõem o projeto foi adotada, pois a análise exaustiva de cada relação envolve uma quantidade de minúcias que complicariam a compreensão global do jogo.

5.3 MODELAGEM GRÁFICA

Em um primeiro momento, a percepção de qualidade do jogo por parte dos usuários se dá em relação à representação visual que o mesmo oferece. Apesar de não garantir o sucesso ou satisfação dos usuários com o jogo, a qualidade de apresentação dos elementos gráficos pode ser decisiva para determinar a rejeição do mesmo, ao primeiro contato.

Atento a esse fato procurou-se manter o máximo de aproximação entre a forma dos objetos que compõem o mundo virtual com suas fontes de inspiração no mundo real.

Para atender as definições presentes no enredo do jogo foram elaboradas imagens para representar o cenário, o avião do jogador, os aviões inimigos, o avião do chefão do jogo, os tiros disparados pelas armas do jogador e do chefão, além de algumas outras.

Logo na abertura da aplicação, na tela de apresentação, é exibida uma imagem de identificação do jogo. Essa imagem pode ser observada logo abaixo, na Figura 5.2.

Figura 5.2 - Imagem de identificação e apresentação do jogo.

Na elaboração do cenário utilizou-se uma seqüência de imagens, as quais representam uma floresta e seus rios, que quando combinadas e distribuídas sobre um mapa formam a representação gráfica do cenário. Essas imagens separadamente e o mapa que constituem combinadas podem ser observadas na Figura 5.3.

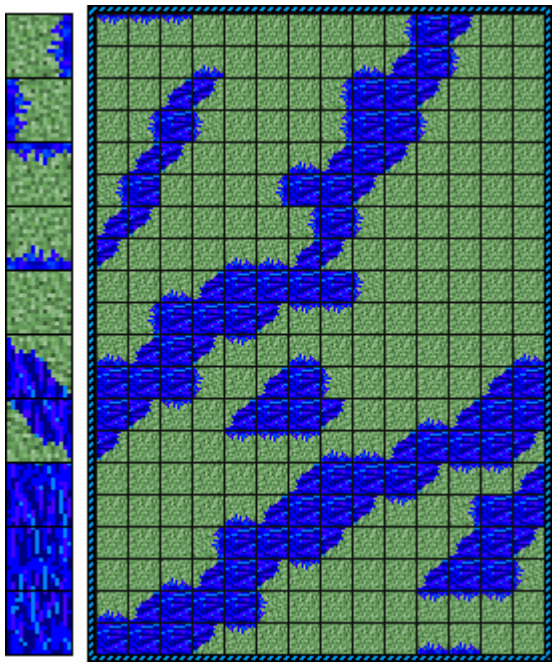

Figura 5.3 - Seqüência de imagens e o mapa que formam para representar o cenário.

A elaboração dos personagens foi uma das etapas que exigiu maior esforço dentro do processo de *design* gráfico, fundamentalmente por dois motivos: o primeiro porque foi uma decisão de projeto não utilizar nenhuma imagem da Internet para representar os personagens; e a segunda, devido ao reduzido tamanho das imagens, em torno de 20x20 *pixels*, fato que torna extremamente difícil a definição de bordas.

Com intuito de manter a fidelidade e a qualidade dos elementos visuais, foi empregada uma metodologia para elaboração dos personagens que consistia de três passos: realizar primeiramente uma pesquisa sobre modelos de aviões reais mais adequados para servir de base ao processo de desenho, depois criar manualmente cada um dos personagens com base nos modelos selecionados e, finalmente, testar o efeito visual da combinação de todos os componentes dentro do cenário. Cabe ressaltar que diversas tentativas foram realizadas até que fosse atingido um resultado considerado satisfatório.

A imagem do avião utilizado como modelo para criação do avião do jogador e ele próprio podem ser observadas, respectivamente, na Figura 5.4. Entretanto, cabe salientar que o tamanho e as proporções reais das imagens foram alterados.

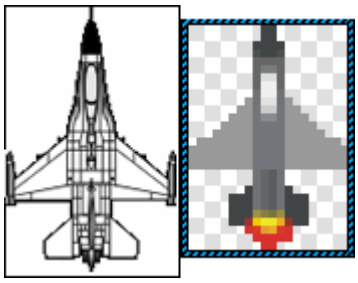

Figura 5.4 - Imagem do avião do jogador (direita) e o modelo utilizado para sua criação (esquerda).

A seguir serão apresentados, na Figura 5.5, o avião utilizado como modelo para criação do chefão do jogo e o resultado desta criação. Novamente, tamanho e proporções reais das imagens foram alterados.

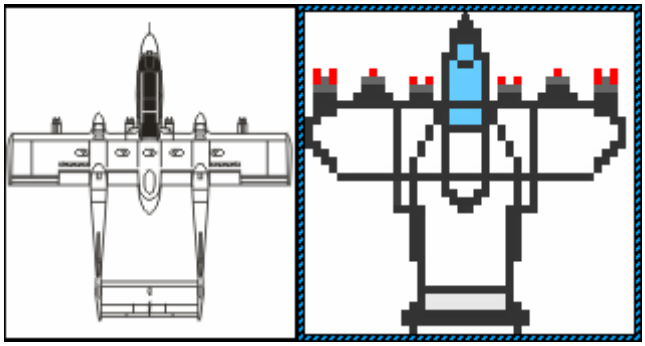

Figura 5.5 - Imagem do chefão do jogo (direita) e o modelo utilizado para sua criação (esquerda).

A imagem que representa os inimigos que "vão surgindo" no decorrer do jogo foi a última, entre os personagens, a ser criada, fato ocasionado pela dificuldade de encontrar um modelo que atendesse a contento, bem como pelo tamanho, o menor entre os personagens. O resultado está exposto na Figura 5.6.

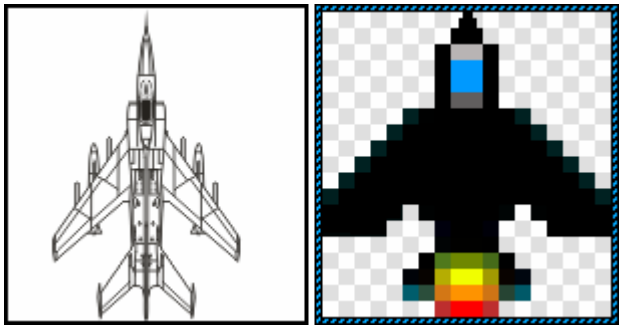

Figura 5.6 - Imagem dos inimigos (direita) e o modelo utilizado para sua criação (esquerda).

Os demais itens que complementam a representação gráfica do jogo foram criados a partir de imagens encontradas na Internet, sob as quais foram aplicadas transformações e modificações para adaptação de tamanho, coloração e disposição. Todas estas imagens foram agrupadas na Figura 5.7.

Figura 5.7 - Imagem da face, seqüência de explosões e conjunto de armas.

6. IMPLEMENTAÇÃO

A implementação do jogo AmazonGuard teve preocupação não apenas com os aspectos relacionados aos jogos, mas também levou em consideração questões inerentes ao desenvolvimento de aplicações voltadas aos dispositivos móveis.

Para orientar a implementação do jogo foi construído um diagrama, apresentado na Figura 6.1, que mostra os principais estados que o jogo pode assumir e as transições entre estes.

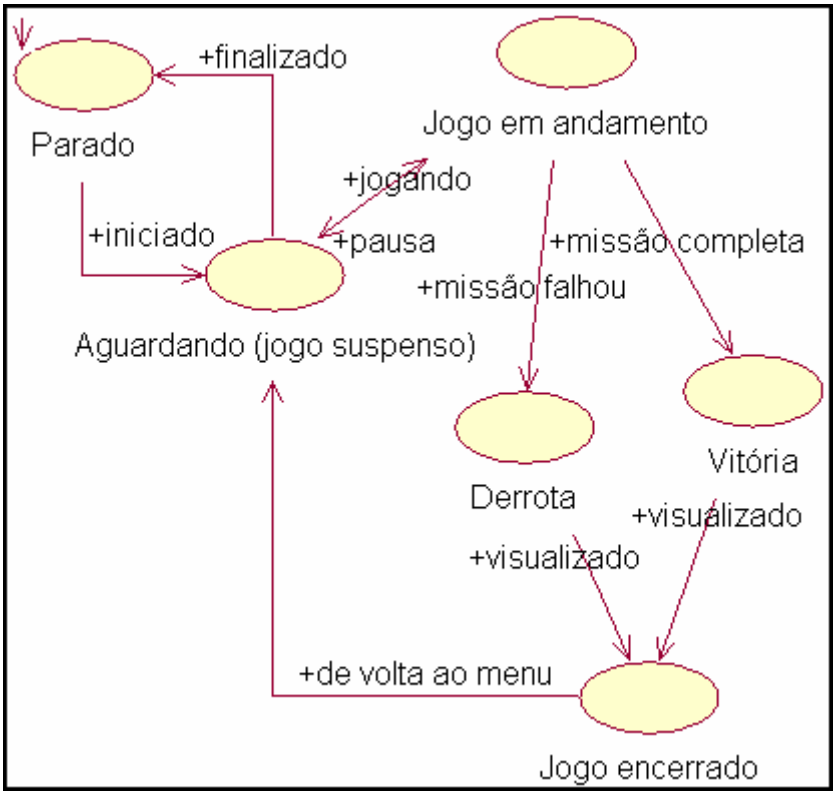

Figura 6.1 - Diagrama de estados do jogo.

As classes envolvidas nessas trocas de estado são a classe "AmazonGuardMIDlet", a classe "MainMenu" e a classe "GameManager".

A partir do momento que aplicação é colocada em atividade, a classe "AmazonGuardMIDlet" entra em execução e chama o objeto "mainMenu" iniciando a aplicação e trocando o estado do jogo para "Aguardando". O jogo permanecerá no estado "Aguardando" até que o usuário selecione a opção "Novo Jogo". Nesse momento, o objeto "gameManager" será inicializado, alterando o estado do jogo para "Jogo em andamento".

Dentro do estado em andamento, uma série de ações são executadas. Estas ações podem ser dividas em três grupos e a seqüência em que ocorrem pode ser observada na Figura 6.2. Explicações mais detalhadas sobre este conjunto de ações serão apresentadas com maior riqueza de detalhes nas próximas seções.

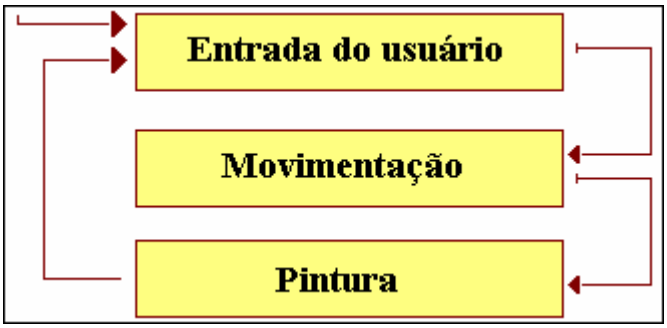

Figura 6.2 - Seqüência de ações executadas no estado jogo em andamento.

Depois de entrar no estado "Jogo em andamento", a aplicação pode efetuar três transições de estado diferentes:

- **Pausa**: quando o usuário deliberadamente interrompe a aplicação por algum motivo qualquer ou quando algum evento externo a aplicação ocorre, como por exemplo, recebimento de uma chamada, levando ao estado de "Aguardando", onde o jogo fica suspenso;

- **Missão falhou**: atingido quando o jogador não consegue suprir as exigências mínimas (como manter um número de vidas superior a zero) ou contemplar as regras do jogo (como impedir mais de trinta inimigos

escaparem da área de visualização do cenário). Esta transição leva ao estado de “Derrota”;

- **Missão concluída**: esta transição ocorre quando o jogador consegue lograr êxito no alcance dos objetivos propostos pelo jogo. Essa transição leva ao estado de “Vitória”.

Uma vez atingidos os estágios de “Derrota” ou “Vitória”, a mensagem correspondente é apresentada ao usuário e o jogo passa ao estado de “Jogo encerrado”, permanecendo nele até que o usuário interaja com o jogo, retornando ao estado “Aguardando”, o qual leva ao menu principal. Estando no menu principal o usuário pode escolher se repete o ciclo ou finaliza a aplicação, levando-a ao estado de “Parada”.

Torna-se óbvia a existência de outros estados e transições entre estes, porém sua omissão tem como objetivo manter a simplicidade de apresentação e foco nos pontos chave do projeto.

A seguir serão apresentados os principais recursos oferecidos pela plataforma J2ME e que foram empregados no desenvolvimento do AmazonGuard.

6.1 RECURSOS EMPREGADOS

A plataforma J2ME oferece uma gama de recursos e bibliotecas definidas em padrões e API´s que facilitam o desenvolvimento de aplicações para dispositivos móveis.

A implementação do jogo AmazonGuard deu-se sobre o padrão MIDP 2.0, porque este conta com um pacote específico para o desenvolvimento de jogos. Esse pacote permite ao desenvolvedor utilizar um nível maior de abstração em relação as ações comuns que devem ser realizadas em diferentes jogos, concentrando assim, sua atenção e esforços no que realmente interessa, que são as regras e definições específicas para o domínio de aplicação do seu jogo.

Dentre os recursos aplicados no desenvolvimento do AmazonGuard podemos citar como mais relevantes os seguintes:

- **RMS**: o *Record Manager Store* (RMS) é um pacote que fornece ao desenvolvedor um conjunto de classes e métodos para possibilitar a gravação e leitura de dados na memória não-volátil do dispositivo, recurso essencial para guardar e recuperar dados como configurações do usuário;

- **Som**: o pacote Media conta com recursos que possibilitam a manipulação de efeitos sonoros como a criação de tons polifônicos, além da reprodução de arquivos gravados nos padrões MIDI e WAV, sendo que foi utilizado no jogo para dar a sonoridade aos disparos realizados pelo jogador, desde que a configuração correspondente tenha sido ativada no jogo pelo usuário;

- **Luz negra**: efeito de luminosidade dado ao contorno das teclas, tela e outras partes do dispositivo, que pode ser controlado pelo programador e foi utilizado no jogo para salientar a entrada do chefão da fase no cenário, desde que a configuração correspondente tenha sido ativada;

- **Vibração**: este recurso faz com que o dispositivo "trema" por alguns instantes e foi utilizado no jogo para dar ênfase às colisões ocorridas com o jogador e a destruição do chefão no final do jogo, desde que a configuração correspondente tenha sido ativada.

A seguir serão apresentadas as principais tarefas que o jogo precisa atender quando se encontra no estado em andamento.

6.2 ENTRADA DO USUÁRIO

A classe GameManager mantém um laço principal através de um *thread* obtido a partir da interface *runnable*, que é executado em paralelo com a recepção de eventos, disparados pelos mecanismos de interação do dispositivo, como por exemplo,

o pressionamento de teclas. As teclas pressionadas são armazenadas num *buffer* que é analisado a cada iteração do laço principal.

O conteúdo do *buffer* é um número inteiro que representa, ao mesmo tempo, as teclas que foram manipuladas e qual evento cada uma delas sofreu. Esse número será utilizado pelo método "*tick( )*" da classe Jogador para determinar qual movimento o personagem principal deve efetuar.

6.3 MOVIMENTOS

As técnicas de movimentação são fundamentais dentro do universo dos jogos. São estas técnicas que definem o comportamento dos personagens e determinam a jogabilidade de um jogo.

O conjunto de regras e técnicas definidas para realizar a movimentação dos elementos visuais do projeto AmzonGuard podem ser classificados em seis grupos distintos de acordo com o tipo de objeto que afetam:

- **Cenário**: a regra definida para movimentação do cenário consiste da rolagem circular do mapa na tela do dispositivo no ritmo de um (1) *pixel* a cada três (3) quadros, o que dá em torno de sete (7) *pixels* por segundo. Para implementação desta regra adotou-se a utilização de dois mapas, um contendo o mapa lógico original e outro com o mapa visual que é alterado com base em variáveis de controle;

- **Inimigos**: para reduzir a complexidade do controle de colisões a técnica de movimentação adotada para os inimigos é muito rudimentar, sendo formada basicamente pelo sorteio aleatório da posição inicial no eixo X e depois apenas incrementa-se a posição do eixo Y, de acordo com nível de dificuldade do jogo, gerando um deslocamento vertical em linha reta;

- **Bônus**: os bônus são vidas extras e armas lançadas no cenário como prêmio ao jogador por cumprir alguma especificação do jogo. A movimentação desses bônus é idêntica a movimentação dos inimigos

acrescentando-se apenas o fato deles também realizarem uma rotação de 90° em sentido horário a cada deslocamento;

- **Disparos**: os disparos são tiros do jogador e do chefão e tem sua movimentação condicionada ao personagem que efetuou o disparo. Quando este é o jogador, eles partem da posição onde se localiza o ponto de referência do mesmo e se deslocam com uma velocidade constante no eixo Y sem mudar sua posição no eixo X. Quando partem do chefão, iniciam com seu ponto Y na mesma posição do eixo Y do chefão e a sua posição no eixo X é sorteada aleatoriamente dentro de um intervalo que compreende os limites da imagem do chefão. A partir daí se deslocam numa velocidade constante no eixo Y e vão ajustando sua posição no eixo X com o objetivo de atingir a localização do jogador;

- **Chefão**: esta é a movimentação mais requintada de todos os elementos, ela tem sua posição inicial definida com os valores de zero para o eixo Y e um número sorteado aleatoriamente para o eixo X. A partir deste momento, ele começa a se movimentar da esquerda para direita, de cima pra baixo até que atinja o limite lateral ou ultrapasse a metade da altura da tela. Então ele muda seu comportamento e passa a se deslocar de baixo pra cima e da direita pra esquerda até que atinja o limite inferior no eixo Y ou o limite lateral no eixo X retornando ao comportamento anterior;

- **Jogador**: a movimentação do personagem que representa o jogador obedece aos comandos impostos pelo jogador, podendo se movimentar em todas direções, limitado-se apenas ao limites da área de exibição do jogo.

6.4 PINTURA

A pintura da cena e dos objetos nela contidos na tela do dispositivo é uma etapa que normalmente consome bastante poder computacional e, conseqüentemente, demanda tempo para ser realizada. Logo, a aplicação racional dos métodos de pintura constitui fator decisivo para o desempenho do jogo.

A representação visual do estado atual do jogo em andamento é atualizada para tela em média de 20 vezes por segundo (também denominados quadros por segundo ou simplesmente FPS).

O protótipo AmazonGuard se beneficia de três recursos oferecidos pelo padrão MIDP 2.0 que possibilitam otimização e automatização do processo de pintura, sendo região de recorte, gerenciador de camadas e o *buffer* de tela síncrono [32].

A utilização da região de recorte permite restringir os efeitos da pintura a uma determinada área selecionada minimizando a necessidade de manipulação de toda área de exibição.

O gerenciador de camadas se encarrega de verificar quais são as áreas visíveis e quais são as áreas encobertas, pintando apenas as áreas visíveis, reduzindo assim o número de operações.

Finalmente, a utilização de um *buffer* síncrono garante a atualização da tela uniformemente e apenas devolve o controle à aplicação depois de toda a pintura ser realizada, evitando problemas de sincronismo entre a aplicação e a área de exibição.

A utilização dos recursos acima mencionados combinados com implantação de variáveis de controle e métodos personalizados de pintura permitiu um bom equilíbrio entre resultado e desempenho.

6.5 CONTROLE DO JOGO

Controle é essencial. No ambiente de jogos, o controle de todas as ações envolve uma diversidade de tarefas que visam garantir manutenção das regras definidas, bem como, verificar ações a serem tomadas frente aos possíveis eventos, realizando as transições necessárias e atualizando o estado do jogo.

Dentro do jogo AmazonGuard, as principais tarefas de controle foram organizadas em três categorias: detecção de colisões, geração de inimigos e fim de jogo. Estas categorias serão discutidas nas próximas subseções.

6.5.1 Detecção de colisões

O controle de detecção de colisões tem a finalidade de garantir que a lei da física, que diz "[...] dois corpos não podem ocupar o mesmo lugar no espaço ao mesmo tempo", seja respeitada. Este tarefa consiste basicamente no processo de verificar se dois elementos visuais fazem intersecção entre si.

O padrão MIDP 2.0 define dois métodos para detecção de colisões, uma é detecção por retângulos e a outra é a por *pixel*.

Na detecção de colisões por retângulo, apenas é comparado se os limites dos retângulos fazem intersecção entre si, como pode ser observado na Figura 6.3.

Figura 6.3 - Detecção de colisões por retângulos.

Esta técnica tem a vantagem de ser muito rápida quando comparada com a técnica de detecção por *pixel*, pois consegue determinar a colisão apenas comparando as posições e limites dos retângulos de cada imagem. Na Figura 6.3, os retângulos em azul delimitam a área total ocupada por cada uma das imagens enquanto o retângulo em verde destaca a área de intersecção entre estes dois retângulos, apontando uma colisão entre os mesmos.

Esta técnica seria a solução ideal para todos os testes de colisões se não fosse pelo fato das imagens não serem sempre retangulares. Para possibilitar a criação de imagens que não tenham uma representação visual retangular o formato de imagem PNG [46], que é o formato de imagens utilizados neste trabalho, implementa a técnica de transparência pelo canal alfa.

Com a inserção de transparências nas imagens, a detecção de colisões por retângulos pode levar a resultados visuais imprecisos, pois mesmo que as imagens não tenham representação visual retangular, elas continuam sendo retangulares. A imagem exibida na Figura 6.4 ilustra está situação.

Figura 6.4 - Detecção de colisões por retângulos, transparência.

Na Figura 6.4, as áreas com quadriculado em cinza e branco representam as regiões transparentes nas imagens e pode-se observar que, mesmo havendo intersecção entre os retângulos, os objetos representados por ambas permanecem inalterados. Em situações como esta, a utilização da técnica de detecção de colisões por *pixel* é mais adequada.

Na detecção de colisões por *pixel*, cada *pixel* de uma imagem é comparado com todos os *pixels* da outra imagem e a colisão só é detectada se um *pixel* não opaco ou transparente de uma imagem estiver sobreposto a outro *pixel* não transparente.

Esta técnica resolve o problema da detecção de colisões sobre imagens com transparências, porém sua adoção prejudica extremamente o desempenho da aplicação, devido ao excessivo número de comparações e deve ser aplicada apenas nos casos realmente necessários.

Neste trabalho utilizou-se a técnica de detecção de colisão por retângulos para controlar as colisões dos inimigos com os disparos realizados pelo jogador. Fazendo uso de um método oferecido pelo padrão MIDP 2.0, que permite a definição do retângulo para teste de colisões independentemente dos limites da imagem, pode-se

amenizar o defeito visual ocasionado pela detecção de colisões sobre imagens com transparências, sem perder desempenho.

O resultado da utilização desse método pode ser observado na Figura 6.5, os retângulos com bordas em azul e sem preenchimento destacam a área utilizada para detectar colisão em cada uma das imagens.

Figura 6.5 - Recuo do retângulo para detecção de colisões.

Para detectar a colisão do personagem principal com os inimigos, com o chefão e com os tiros disparados pelo chefão optou-se pela utilização da técnica de detecção por *pixel*, a fim de garantir a representação e controle adequado das condições do jogo.

6.5.2 Geração de inimigos

Determinar quando, quantos e em que posição os inimigos devem ser criados e dispostos no cenário é uma tarefa que envolve uma certa complexidade. É necessário encontrar um ponto de equilíbrio entre a quantidade a ser gerada, o momento em que são lançados ao cenário e a dificuldade que isto pode acarretar ao jogo.

Para determinar o número máximo de inimigos que podem estar simultaneamente no cenário e manter uma certa proporção com o grau de dificuldade selecionado foi estabelecida a seguinte regra: o número máximo de inimigos simultâneos deve ser o resultado da multiplicação de uma constante pelo grau de desafio do jogo, sendo que o nível fácil tem o valor um (1), médio vale dois (2) e difícil vale três (3). Desta forma, o número de inimigos varia de acordo com o grau de desafio escolhido.

A disposição dos inimigos gerados dentro do cenário gerou o problema da superposição. Este problema ocorreu porque o posicionamento de cada inimigo gerado

é definido como sendo o valor zero (0) para o eixo Y e um valor sorteado aleatoriamente podendo variar de zero até o limite da área de exibição da tela para o eixo X. Mesmo sendo a posição no eixo X definida aleatoriamente aconteciam vários casos onde dois sorteios consecutivos geravam números idênticos ou muito próximos. Para contornar esta situação foi inserida uma variável para controlar a liberação de novos inimigos, esta variável só permite a liberação de um novo inimigo se o último inimigo gerado já tenha se deslocado no eixo Y, um número de *pixels* maior ou igual a sua própria altura.

A cada iteração do laço principal, a classe GameManager verifica se o número de inimigos ativos é igual ao máximo permitido e então testa a variável de controle para determinar se um novo inimigo pode ser sorteado ou não.

6.5.3 Fim de jogo

O constante acompanhamento do estado do jogo é fundamental para determinar as situações onde o final do jogo é atingido.

No AmazonGuard o final de jogo pode ser atingido em três situações especificas:

- **Determinação arbitrária**: acontece quando o usuário resolve interromper permanentemente o jogo em execução e seleciona a opção de sair. Isto pode ocorrer indiferente de qualquer estado interno do jogo;

- **Missão falhou**: quando o usuário não consegue garantir a manutenção das regras básicas de sobrevivência no jogo, que são a regra de não chegar a um número de vidas igual a zero e a regra de não deixar um número maior ou igual a trinta (30) inimigos escaparem da área de exibição do jogo;

- **Missão concluída**: atingido quando o usuário alcança os objetivos determinados pelo jogo sem violar as regras de sobrevivência.

7. SIMULAÇÃO

A utilização de simuladores de arquitetura é indispensável para o desenvolvimento e teste de aplicações voltadas ao ambiente dos dispositivos móveis.

Os simuladores contam com um conjunto de parâmetros que podem ser manipulados pelo usuário e permitem a validação da aplicação sobre diversas circunstâncias. A modificação do valor destes parâmetros possibilita a simulação de comportamentos específicos, que podem variar de dispositivo para dispositivo de acordo com os recursos ou facilidades oferecidas por cada um.

Atualmente encontra-se disponível na Internet uma grande variedade de sistemas de simulação para as arquiteturas de dispositivos móveis, a grande maioria fornecida de forma gratuita pelos próprios fabricantes destes dispositivos. Estes sistemas contam com uma série de funcionalidades específicas que visam simular com a maior precisão possível o comportamento da aplicação no dispositivo real.

A avaliação do desempenho da aplicação em diferentes simuladores é aconselhável, pois permite analisar seu comportamento em diferentes arquiteturas. Portanto, nas próximas seções serão apresentados os sistemas de simulação utilizados para validação do protótipo AmazonGuard.

7.1 SIMULADOR DA SUN

Batizado de WTK (*Wireless ToolKit*), este sistema desenvolvido pela Sun é um simulador padrão para dispositivos móveis compatíveis com a plataforma J2ME.

Atualmente o WTK está na versão 2.2, entretanto, a versão utilizada neste trabalho foi a versão 2.1 [40].

O WTK é considerado pela própria Sun o estado da arte dos sistemas de simulação para a plataforma J2ME. Ele é composto por quatro diferentes dispositivos para simulação além de ferramentas auxiliares para avaliação de desempenho e utilização de memória. Dentre os recursos oferecidos pode-se citar o suporte aos padrões CLCD e MIDP além das API´s opcionais como MMA [47] e WSA [48]. Suas funcionalidades incluem as capacidades de criação, compilação, depuração, pré-verificação, empacotamento e assinalamento de projetos J2ME.

Na Figura 7.1 é exibida a seqüência normal de execução do AmazonGuard, para o simulador padrão de telefones celulares coloridos, oferecido pelo WTK 2.1, desde a seleção da aplicação para execução, passando pelas etapas de apresentação e menu principal até chegar ao jogo propriamente dito.

Figura 7.1 - Seqüência de execução do AmazonGuard no WTK 2.1 utilizando o simulador padrão para telefones celulares coloridos.

A existência de quatro simuladores diferentes dentro do WTK 2.1 permite ao desenvolvedor realizar o teste de sua aplicação em quatro arquiteturas distintas dentro do mesmo sistema. A Figura 7.2 apresenta o jogo AmazonGuard sendo executado

sobre o simulador para telefones celulares sem suporte a cores e no simulador de dispositivos no padrão *comunicator*.

Figura 7.2 - Execução do AmazonGuard sobre simulador em tons de cinza e sobre o simulador *comunicator*.

Na imagem acima pode-se observar que o jogo se adapta perfeitamente à área de exibição disponível na tela do dispositivo. No caso do simulador em tons de cinza o cenário do jogo é mais largo que a área disponível para exibição, então o jogo coloca o canto superior esquerdo do cenário no ponto (0, 0) da área de exibição da tela e faz o recorte do tamanho que exceder o tamanho máximo aceito pelo dispositivo. No segundo simulador, a largura da área de exibição disponível na tela é maior que a largura do cenário, neste caso, o jogo faz o cálculo da área excedente e com uma translação centralizando o cenário na área de exibição.

7.2 SIMULADOR DA NOKIA

Dentre os principais fabricantes de dispositivos móveis a Nokia é seguramente quem oferece o mais completo suporte ao desenvolvedor. Ela disponibiliza um portal com notícias, eventos, ferramentas, documentação e fóruns para debate [43].

O sistema da Nokia, denominado de *Nokia Developer´s Suíte 2.1 for J2ME*, é um gerenciador de simuladores que fornece as bases para instalação de simuladores específicos para cada modelo de dispositivo das suas diferentes séries que ofereçam suporte a plataforma J2ME [43].

Simuladores para vários modelos foram instalados e testados, sendo que alguns apresentaram falhas na execução do jogo. Modificações no jogo foram realizadas a fim de sanar os problemas ocorridos, novamente testou-se a execução do jogo e pôde-se constatar que algumas falhas persistiam.

Uma análise mais cuidadosa foi realizada onde ficou evidente ao menos duas anomalias no comportamento do simulador, que não condizem com as especificações do padrão MIDP 2.0:

- **Repetição de teclas**: uma vez que o usuário pressione duas vezes a mesma tecla em intervalos de tempo muito pequenos o simulador fica num laço enviando o sinal para aplicação de que a tecla em questão está pressionada, este laço só é encerrado quando o usuário pressiona e solta novamente a mesma tecla;

- **Estouro de tempo**: em alguns casos a execução da aplicação ficava muito lenta, o programa de teste implementado consistia de um laço que deveria realizar um certo número de operações dentro de um intervalo de tempo pré-determinado e a cada iteração o tempo de execução era calculado, ao final do teste pôde-se observar que o tempo real em execução era muito maior que o somatório dos tempos calculados a cada iteração.

As anomalias acima descritas afetam diretamente a execução da aplicação, pois levam a comportamentos imprecisos e demandam atrasos inaceitáveis.

A Figura 7.3, demonstra o funcionamento do AmazonGuard sobre simulador para os dispositivos da série 40 da Nokia, sendo que esse simulador teve desempenho dentro do padrão e resultados muito parecidos com os obtidos no simulador da Sun.

Figura 7.3 - Execução do AmazonGuard no simulador para dispositivos da série 40 da Nokia.

7.3 SIMULADOR DA MOTOROLA

A Motorola também conta com um suporte muito bom ao desenvolvedor de aplicações para dispositivos móveis. A exemplo da Nokia, também mantêm um portal especializado com notícias, documentação, fóruns e ferramentas [42].

Apesar de oferecer suporte a um número de dispositivos limitado e bem inferior ao da Nokia, os simuladores da Motorola se mostraram muito estáveis e em conformidade com os padrões.

O único comentário negativo a ser feito sobre os simuladores da Motorola fica por conta do tempo de demora para alocação de memória para instanciação de novos objetos, fato que, se não tratado pelo desenvolvedor pode ocasionar “travadas” momentâneas na aplicação.

A Figura 7.4, ilustra a execução do jogo sobre os simuladores para os dispositivos i730 e i285 respectivamente.

Figura 7.4 - AmazonGuard sendo jogado nos simuladores da Motorola i730 e i285.

7.4 SIMULADOR DA SIEMENS

A Siemens, a exemplo de outros fabricantes, também lançou um portal destinado aos desenvolvedores de aplicações para seus dispositivos [41]. Entretanto, o suporte ao padrão MIDP 2.0 para os seus dispositivos é bastante recente, sendo que, até o momento existem simuladores para apenas dois modelos. Contudo, a primeira impressão foi muito positiva já que o teste de execução do jogo AmazonGuard no simulador CX65 foi excelente, não necessitando nenhuma consideração adicional. A Figura 7.5 mostra o jogo sendo executado neste simulador.

Figura 7.5 - AmazonGuard sendo jogado no simulador da Siemens CX65.

7.5 SIMULADOR DA SONY

O que a Sony disponibiliza não é bem um sistema de simulação, na verdade é uma extensão ao WTK 2.1 que estende e adapta as funcionalidades deste sistema para atender as peculiaridades dos dispositivos da Sony.

Apesar da interface dos simuladores da Sony não encorajar muito a aquisição de um de seus dispositivos, conforme a Figura 7.6, o desempenho destes foi extremante satisfatório comportando-se de forma análoga ao simulador da Sun.

Figura 7.6 - AmazonGuard sendo jogado nos simuladores da Sonyericsson P900 e Z500.

Na próxima seção será abordado o tema do teste de distribuição da aplicação para o dispositivo real nos ambientes de simulação disponíveis.

7.6 DISTRIBUIÇÃO

Uma aplicação desenvolvida para plataforma J2ME pode ser facilmente distribuída pela Internet através de um servidor WEB. Para instalar a aplicação no dispositivo final basta que o usuário acesse a Internet através da WAP de seu dispositivo móvel e indique o endereço e o caminho onde se encontra a aplicação que o JAM se encarregará de fazer o *download* e a instalação do aplicativo no dispositivo.

A distribuição do AmazonGuard para o dispositivo real pode ser realizada por qualquer uma das formas apresentadas na Seção 2.1.2. Entretanto, pela simplicidade do processo será apresentado como a distribuição de uma aplicação a partir de um servidor WEB pode ser simulada dentro do sistema WTK 2.1.

O processo para realizar a instalação do jogo para um dispositivo móvel é simulado passo a passo e pode ser acompanhado em detalhes na Figura 7.7.

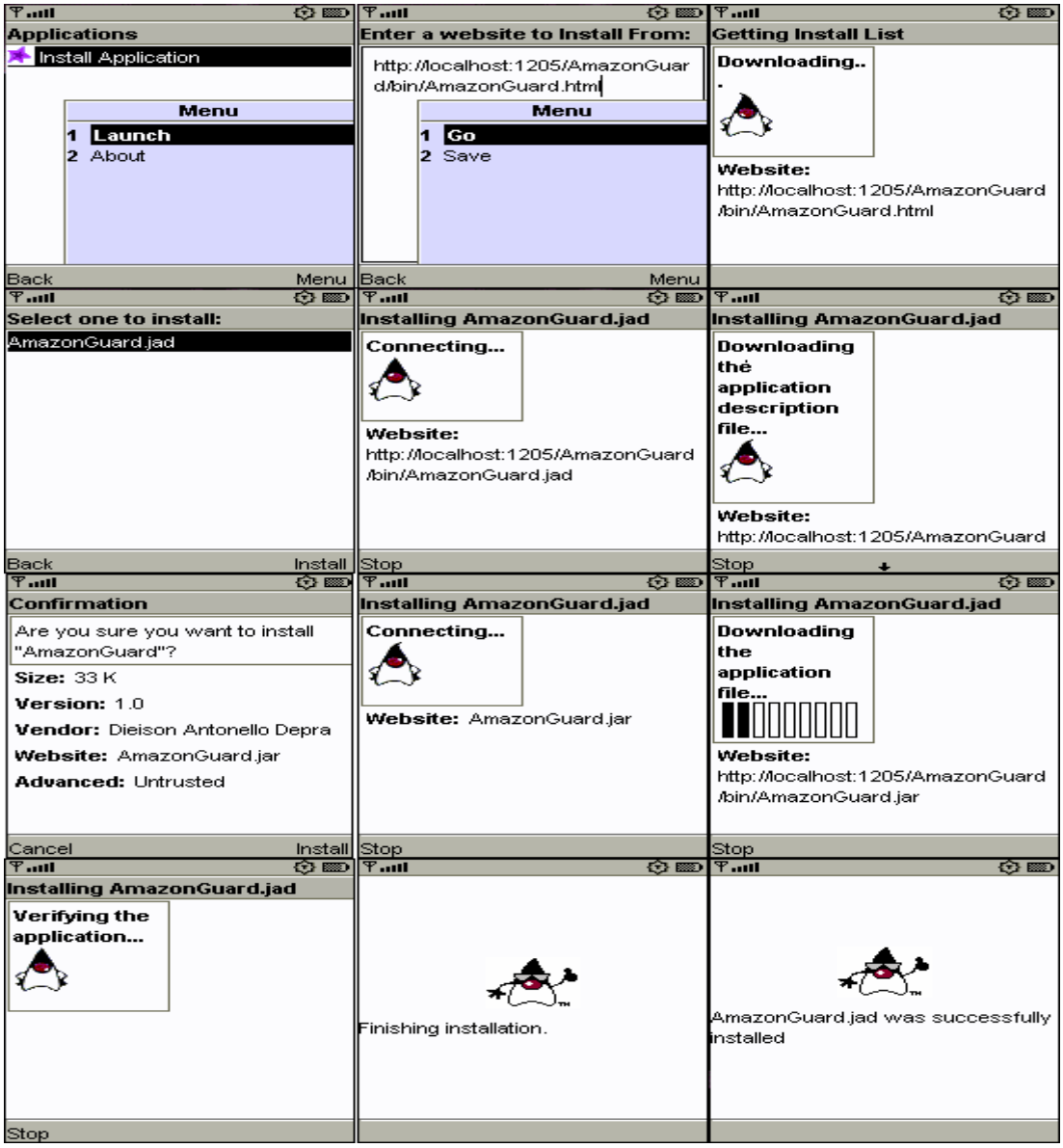

Figura 7.7 – Simulação processo de instalação do jogo AmazonGuard no dispositivo real a partir de um servidor WEB.

O processo todo consiste em informar o endereço e caminho da aplicação, depois o dispositivo conecta-se ao servidor WEB, verifica a existência do arquivo com informações de descrição da aplicação e faz *download* do mesmo para verificar se os requisitos solicitados são contemplados pelo dispositivo. Logo depois, as informações sobre tamanho, versão, fabricante e assinalamento da aplicação são exibidas ao usuário que deve confirmar ou rejeitar a instalação do aplicativo.

8. OTIMIZAÇÕES

A relativa simplicidade no desenvolvimento de algumas aplicações e a disponibilidade de recursos encontrados nos computadores pessoais muitas vezes leva o desenvolvedor a uma certa acomodação, fazendo com que ele contente-se com soluções eficazes. Entretanto, num contexto de aplicações complexas como da área de jogos, combinados a um ambiente extremamente carente de recursos de *hardware,* como nos dispositivos móveis, uma solução apenas eficaz pode não ser aceitável. Nestes casos, é necessário buscar um nível de excelência, onde apenas soluções eficientes sejam consideradas.

Neste sentido, a etapa de otimização do jogo teve que ser divida em diferentes frentes a fim de atuar nos principais obstáculos identificados, podendo-se mencionar três itens:

- **Tamanho**: o padrão MIDP 2.0 estabelece um tamanho máximo de 64KB para todo pacote da aplicação. Neste pacote deve-se incluir além dos arquivos de classe todos os recursos necessários à execução da aplicação, como arquivos de som, imagens e dados;

- **Memória**: a quantidade de memória de execução disponível é extremamente limitada, logo, a gerência de memória deve ser preocupação constante durante a implementação para evitar o esgotamento da memória e necessidade de executar a coleta de lixo;

- **Processamento**: a capacidade de processamento é outro fator preocupante, logo, a otimização dos algoritmos deve ser efetuada a fim de garantir que a aplicação possa ser executada dentro do intervalo de tempo desejado.

As técnicas aplicadas para realizar a otimização do jogo foram agrupadas em três diferentes categorias que serão apresentadas nas próximas seções.

8.1 IMAGENS

A otimização das imagens utilizadas no jogo permitiu a redução do tamanho total do pacote e, conseqüente, da utilização de memória de execução.

Todo o processo de otimização das imagens teve quatro passos básicos que foram aplicados a todas as imagens:

- Primeiramente foi utilizado o recurso do FireWorks que permite definir exatamente o tamanho da paleta de cores da imagem durante a exportação da mesma para o formato PNG;

- Num segundo passo procurou-se reduzir o número de tonalidades diferentes de uma mesma cor, reduzindo assim o número de cores na paleta;

- Na terceira etapa, imagens de tamanho iguais foram agrupadas dentro de um mesmo arquivo;

- Finalmente, no último estágio, as imagens muito grandes foram divididas em vários quadros, os quadros repetidos foram eliminados e a montagem da imagem completa passou a ser realizada em tempo de execução através da combinação dos quadros isolados.

Ao final desse processo o tamanho total das imagens passou de 11KB para 6.5KB representando uma redução de quase 41% do tamanho original, reduzindo consideravelmente a utilização da memória de execução.

8.2 CÓDIGO

A otimização do código normalmente é o processo que tem mais impacto no desempenho do sistema como um todo. As modificações realizadas no código do jogo permitiram reduzir a utilização de memória, o tamanho da aplicação e a força de processamento necessária.

Algumas técnicas de otimização que dizem respeito a características próprias da linguagem de programação empregada podem ser aplicadas de forma genérica, sem a necessidade de conhecimento preciso sobre qual o trecho deva ser melhorado. Na teoria, a linguagem Java livra o programador da preocupação com a gerência de memória, porém em se tratando de desenvolvimento de jogos e ambientes de dispositivos móveis todo cuidado pode ser pouco.

O foco aqui será mostrar como se encontraram trechos que necessitavam ser otimizados e quais foram às modificações realizadas nesses trechos. Uma relação mais detalhada sobre as diferentes técnicas genéricas de otimização que foram aplicadas neste trabalho e podem ser estendidas para quaisquer aplicações J2ME podem ser encontradas no artigo de Mike Shivas [49].

Descobrir qual método está consumindo maior tempo de processamento ou desperdiçando mais memória é uma tarefa extremamente difícil. Contudo, para a sorte dos desenvolvedores de aplicações para plataforma J2ME, o WTK 2.1 dispõe de uma ferramenta auxiliar conhecida como *profiler* que permite ao desenvolvedor monitorar e avaliar o desempenho de cada método da aplicação.

Após a execução da aplicação, o *profiler* exibe cada um dos métodos executados, quantas vezes ele foi chamado durante a execução, quantos ciclos de CPU sua execução consumiu e qual o percentual de ciclos em relação ao total da aplicação. Com estas informações é perfeitamente possível descobrir qual método representa o gargalo do sistema e assim concentrar as técnicas de otimização neste trecho.

A Figura 8.1 mostra que em apenas cinco (5) chamadas ao método "*System.gc( )*" estavam sendo gastos 66% dos ciclos de CPU e que as chamadas ao método "*drawString( )*" despediam 26% dos ciclos.

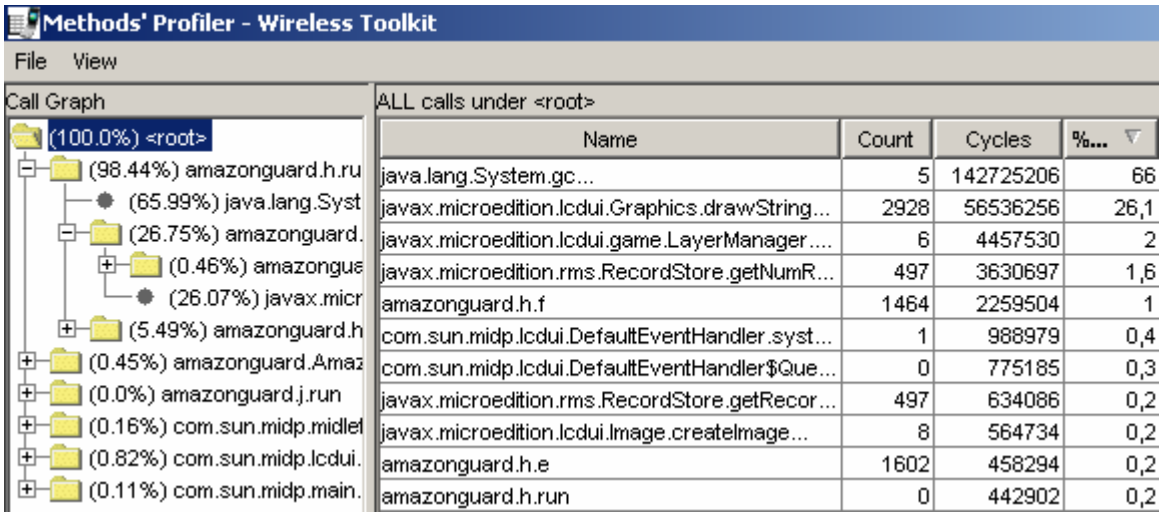

Figura 8.1 - Monitoramento do desempenho dos métodos *System.gc()* e *drawString()*.

As chamadas a esses métodos estavam representando um consumo de 92% do total de ciclos de CPU, evidenciando a necessidade de otimização nos dois. Porém os métodos "*System.gc( )*" e "*drawString( )*" são métodos da biblioteca Java e não é possível fazer modificações nesse código. A solução adotada foi reduzir o número de chamadas ao primeiro método (melhorando a gerência de objetos, conseqüentemente reduzindo a geração de lixo) e substituir o segundo método por outro similar.

Para redução da quantidade de lixo gerada foi necessário implementar uma série de modificações no código, as principais são descritas a seguir:

- Restringir a utilização de instâncias da classe *string* apenas a objetos finais, eliminando a modificação dinâmica de *strings* que tanto lixo produz;

- Modificar o mecanismo de gerência dos inimigos ativos, substituindo o vetor dinâmico por um estático de tamanho igual ao tamanho máximo de inimigos ativos simultaneamente;

- Utilizar o modificador de acesso final na declaração dos objetos bônus e nas instâncias da Classe "*Image*", reduzindo o número de chamadas aos métodos construtores destas classes.

Para substituir o método "*drawString( )*" foram realizadas duas tentativas. Na primeira foi implementado um método personalizado que pintava os valores desejados caractere a caractere. A segunda tentativa lançou mão de outro método da API Java, o "*drawChars( )*" e lançou uso do objeto da classe "*String*" por um da classe "*StringBuffer*".

Na prática, a utilização do método personalizado apresentou desempenho muito similar ao método "*drawChars( )*", sendo preterido a este para manter padronização e reduzir o tamanho do código. A Figura 8.2 mostra o desempenho do jogo após todas as otimizações realizadas.

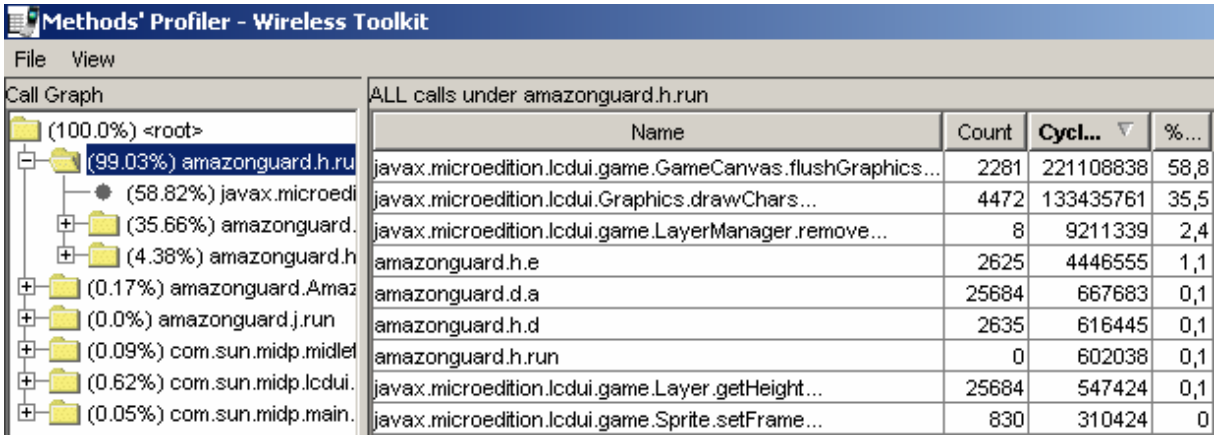

Figura 8.2 - Monitoramento do desempenho do jogo depois das otimizações.

Agora o método "*System.gc( )*" não foi chamado nenhuma vez e o método "*flushGraphics( )*" passou a ser responsável por 58% dos ciclos de CPU. Aparentemente pode dar a impressão que apenas se transferiu o problema de lugar, no entanto, a medição do tempo necessário para execução do código após as otimizações ficou 68% abaixo da marca sem as otimizações. Este fato comprova a importância e impacto das modificações implementadas no desempenho geral da aplicação.

Em síntese a adoção de boas técnicas de otimização combinadas com a utilização de ferramentas que permitam o monitoramento dos recursos da plataforma formam o conjunto ideal para qualquer desenvolvedor obter o máximo desempenho de sua aplicação, mas vale lembrar que, como diria Mike Shivas [49], "[...] a melhor ferramenta que existe para otimização encontra-se entre suas orelhas – o cérebro[...]".

8.3 COMPRESSÃO

A compressão, como o próprio nome induz, fornece uma redução no tamanho do código objeto. As ferramentas que realizam esse trabalho de compressão de arquivos de código objeto Java são conhecidas como obfuscadores.

Um obfuscador atinge a redução no tamanho do código objeto através da substituição de nomes de métodos e variáveis por nomes mais curtos que necessitem de menos caracteres para serem representados. Durante esse processo, como efeito colateral da compressão, essas ferramentas acabam embaralhando o código objeto o que torna muito mais complexa a tarefa de decompilação do mesmo. Este fato torna o efeito colateral um fator positivo.

9. RESULTADOS

Como resultado final deste trabalho chegou-se a um protótipo de jogo 2D destinado ao ambiente dos dispositivos móveis, capaz de ser executado sobre qualquer arquitetura que ofereça suporte a plataforma J2ME e ao padrão MIDP 2.0.

Contudo, durante o desenvolvimento do protótipo enfrentaram-se algumas dificuldades que merecem destaque devido, principalmente, ao tempo que suas soluções exigiram, sendo discutidas nas próximas seções.

9.1 QUADROS POR SEGUNDO

Um dos primeiros problemas a ser enfrentado foi o número de quadros a serem exibidos a cada segundo.

O primeiro passo em busca da solução foi descobrir o tempo exato gasto por uma determinada arquitetura para realizar todas as operações necessárias à exibição de apenas um quadro da cena. Para identificar este valor foi necessário calcular a média aritmética e o desvio padrão para o tempo de execução em cada uma das 10.000 iterações realizadas.

A partir destes dados foi possível definir um número inicial de 14 quadros por segundo e depois através do método de tentativa e erro foi atingido o valor definitivo de vinte (20) quadros por segundo, que demonstrou boa relação entre ás variáveis: tempo de processamento, tempo de espera e resposta ao usuário.

9.2 DETECÇÃO DE COLISÕES

O segundo conjunto de problemas a ser solucionado dizia respeito às técnicas de detecção de colisões empregadas.

Como descrito na Seção 6.5.1, existem duas técnicas básicas para detecção de colisões oferecidas pelas bibliotecas do padrão MIDP 2.0 da plataforma J2ME: uma por retângulos e a outra por *pixel*. Entretanto, a primeira prioriza o desempenho em detrimento do efeito visual produzido quando aplicada sobre imagens com transparência e a outra faz exatamente o contrário. Era necessário então encontrar um meio termo entre desempenho e coerência visual.

A solução a ser aplicada foi divida em dois casos distintos. No primeiro foi aplicada a detecção de colisão por *pixel* onde o nível de detalhe e coerência visual era indispensável, como nas colisões do personagem principal e do chefão do jogo. No outro caso, aplicado a todos os outros tipos de colisões, foi empregada uma versão modificada da detecção de colisões por retângulos para situações mais corriqueiras e que não exigissem nível de detalhe tão grande. Para obter-se esta solução foi necessário recuar o retângulo utilizado para detectar a colisão, das bordas da imagem para uma região mais próxima ao centro do objeto, minimizando assim o efeito visual impreciso e mantendo a eficiência da técnica.

9.3 OTIMIZAÇÕES

A otimização foi o problema mais agudo de todos, pois teve que ser atacado em várias frentes. Sua resolução envolveu manipulação de imagens, modificações em algoritmos e estruturas de dados, além da utilização de ferramentas que atuassem sobre o código objeto.

Contudo, ao final do processo de otimizações, obteve-se um protótipo leve em tamanho e rápido na execução, sendo principalmente eficiente, já que cumpre seu objetivo com economia de recursos, sem acarretar problemas de legibilidade ou manutenibilidade do código fonte.

10. CONSIDERAÇÕES FINAIS

Quando um novo trabalho é iniciado muitos desafios são lançados, muitos planos são traçados e muita expectativa é criada e, mesmo com todo empenho, dedicação e competência empregada, algumas vezes o resultado final pode fugir ao previsto, podendo, em alguns momentos nos surpreender de forma negativa e em outros de forma positiva. Entretanto, o que realmente importa e deve ficar, é a experiência vivida e os ensinamentos que dela podem ser tirados.

Ao longo do desenvolvimento deste trabalho muita pesquisa foi realizada e, por mais detalhista que se fosse, seria praticamente impossível enumerar tudo que se aprendeu e foi aplicado ao mesmo, no decorrer do tempo.

Com o desenvolvimento do protótipo AmazonGuard para dispositivos móveis, procurou-se, acima de tudo, validar o estudo realizado e comprovar a aplicação prática das tecnologias disponíveis para o desenvolvimento de aplicações para o ambiente dos dispositivos móveis.

Também se pode constatar o surgimento de um novo nicho de mercado, carente por produtos e serviços, mas principalmente por profissionais capacitados a atender suas expectativas e necessidades.

O presente trabalho também desempenha um papel social, uma vez que traz à comunidade acadêmica, fatos e tendências que devem ser observados a fim de auxiliar na preparação, direcionamento e, principalmente, formação dos profissionais de amanhã, que devem sempre permanecer conectados com as questões da atualidade.

Chegado ao término deste trabalho, é importante relatar que o desenvolvimento do protótipo AmazonGuard superou as expectativas iniciais, pois não limitou-se a atingir os objetivos traçados. Como resultado final, obteve-se um *software* funcional, robusto, portável, personalizável, eficiente, de boa usabilidade, com suporte a internacionalização, com código de boa qualidade e fácil manutenibilidade e, principalmente, que não fica devendo aos produtos comerciais desenvolvidos com a mesma tecnologia.

Como trabalhos futuros, seria uma boa proposta fazer as extensões e aperfeiçoamentos necessários para tornar este protótipo em um jogo acabado com boas possibilidades comerciais. Outra sugestão, no entanto um pouco ousada, seria desenvolver um protótipo de jogo 3D para o mesmo ambiente, pois as tecnologias necessárias para tal desenvolvimento começam a chegar ao mercado e certamente, em pouco tempo, dispositivos com o suporte adequado a esta tecnologia ganharão a grande massa, tornando-se mais uma febre de consumo.

11. REFERÊNCIAS BIBLIOGRÁFICAS

[1] MASSI, Domenico Di; **Ócio Criativo e Futuro do trabalho - A sociologia do trabalho.** Emissora TVE/Rede Brasil, Programa Roda Vida, 1998.

[2] O que são PalmTop´s? Capturado em 12 de setembro de 2003. On-line. Disponível em: http://www.palm.com/home.html.

[3] O que são Pda´s? Capturado em 18 de setembro de 200. On-line. Disponível em: http://www.codetel.net.do/datamovil/preguntas%20frecuentes.htm#7.

[4] O que é E-commerce? Capturado em 16 de setembro de 2003. On-line. Disponível em: http://www.pinguimdan.hpg.ig.com.br/e_commerce.htm.

[5] O Mercado de Entretenimento – Jogos. Capturado em 10 de setembro de 2003. On-line. Disponível em: http://outerspace.terra.com.br/especial/materias/jogodoano2002/introducao.asp.

[6] ATARI Inc. Capturado em 17 de setembro de 2003. On-line. Disponível em: http://www.atari2600.com/.

[7] SEGA Inc. Capturado em 16 de setembro de 2003. On-line. Disponível em: http://www.sega.com/index_2.jhtml.

[8] NINTENDO Inc. Capturado em 16 de setembro de 2003. On-line. Disponível em: http://www.nintendo.com/index.jsp.

[9] MOTOROLA Inc. Capturado em 16 de setembro de 2003. On-line. Disponível em: http://www.motorola.com.

[10] NOKIA Inc. Capturado em 16 de setembro de 2003. On-line. Disponível em: http://www.nokia.com.

[11] SIEMENS Inc. Capturado em 16 de setembro de 2003. On-line. Disponível em: http://www.siemens.com.

[12] SONY Inc. Capturado em 21 de setembro de 2003. On-line. Disponível em: http://www.sony.com.

[13] *Wireless Application Protocol Forum.* "**WAP – *Wireless Application Protocol Specification***". Capturado em 22 de setembro de 2003. On-line. Disponível em: http://www.wapforum.org/what/technical.htm.

[14] JAVA 2 Platform, Micro Edition (J2ME). Capturado em 25 de setembro de 2003. On-line. Disponível em: http://java.sun.com/j2me/index.jsp.

[15] JOGOS e Outras Aplicações Java para Dispositivos Móveis. Telemoveis. Capturado em 5 de outubro de 2003. On-line. Disponível em: http://www.telemoveis.com/articles/item.asp?ID=354.

[16] PEREIRA, Rui F.; SILVA, Mário J. **Descoberta de Serviços em Ambientes Móveis.** Faculdade de Ciências, Departamento de Informática da Universidade de Lisboa. Capturado em 10 de outubro de 2003. On-line. Disponível em: http://gsd.di.uminho.pt/epcm99/pdf/sessao4/servicos.pdf.

[17] Diversão no Celular. Correio Brasiliense. Capturado em 30 de setembro de 2003. Disponível em: http://www2.correioweb.com.br/cw/EDICAO_20030408/sup_info_080403_22.htm.

[18] TRINTA, Fernando A. M.; MIRANDA, Oscar Gomes de; RAMALHO, Geber Lisboa. **Jogos isométricos em dispositivos móveis.** Universidade Federal de

Pernambuco. Capturado em 15 de outubro de 2003. On-line. Disponível em: www.cin.ufpe.br/~famt/docs/cientific_production/CIJ.pdf.

[19] BORTOLAZZO, Cintya. **Proposta de Trabalho de Conclusão – Desenvolvimento de um Jogo 3D.** Pontifícia Universidade Católica do Rio Grande do Sul. Não publicado. 2002.

[20] CESAR – Centro de Estudos Avançados do Recife. Capturado em 2 de outubro de 2003. On-line. Disponível em: http://www.cesar.org.br/.

[21] FORUM; Nokia. "**MIDP and Game UI".** Capturado em 5 de novembro de 2003. On-line. Disponível em: http://www.forum.nokia.com.

[22] FORUM; Nokia. "**Developing Java™ Games for Platform Portability - Case Study: Miki's World".** Capturado em 2 de novembro de 2003. On-line. Disponível em: http://www.forum.nokia.com.

[23] FORUM; Nokia. "**Multi-Player MIDP Game Progaming".** Capturado em 3 de novembro de 2003. On-line. Disponível em: http://www.forum.nokia.com.

[24] JOHANN, Marcelo; KINDEL, Marcus; BORTOLAZZO, Cintya. **Experiência de um Curso Prático de Introdução à Programação em Linguagem C++ Aplicada ao Desenvolvimento de Jogos em 3D.** Pontifícia Universidade Católica do Rio Grande do Sul – Campus Universitário II. Capturado em 5 de novembro de 2003. On-line. Disponível em: http://www.inf.ufrgs.br/~johann/papers.htm.

[25] BOOCH, Grady; RUNBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário.** Rio de Janeiro. Editora Campus Ltda. 2000.

[26] BORTOLAZZO, Cintya. **PANDEMONIUM –Um protótipo de jogo em 3D.** Pontifícia Universidade Católica do Rio Grande do Sul – Campus Universitário II. Trabalho de Conclusão II. 2003.

[27] FORUM; Nokia. "**Introduction to the Mobile Games Business".** Capturado em 11 de novembro de 2003. On-line. Disponível em: http://www.forum.nokia.com.

[28] SYMBIAN OS Tecnology. Capturado em 22 de novembro de 2003. On-line. Disponível em: http://www.symbian.com/technology/technology.html.

[29] PALM OS. Capturado em 27 de novembro de 2003. On-line. Disponível em: http://www.palmsource.com/palmos/.

[30] FOX, David; VERHOSEK, Roman. **Micro Java™ Game Developement.** Addison Wesley, 2002.

[31] CONNECT LIMITED DEIVCE CONFIGURATION (CLDC) Specification 1.1. Capturado em 10 de fevereiro de 2004. On-line. Disponível em: http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html.

[32] MOBILE INFORMATION DEVICE PROFILE (MIDP) Specification 2.0. Capturado em 15 de janeiro de 2004. On-line. Disponível em: http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html.

[33] "The CLDC HotSpot™ Implementation Virtual Machine". Capturado em 23 de outubro de 2003. On-line. Disponível em: http://java.sun.com/products/cldc/wp/CLDC_HotSpot_WhitePaper.pdf.

[34] "Datasheet Mobile Information Device Profile". Capturado em 23 de outubro de 2003. On-line. Disponível em: http://java.sun.com/products/midp/midp-ds.pdf.

[35] "Eclipse Platform Technical Overview". Capturado em 17 de março de 2004. On-line. Disponível em: http://www.eclipse.org/articles/index.html.

[36] "Eclipse.org Software User Agreement". Capturado em 21 de março de 2004. On-line. Disponível em: http://www.eclipse.org/downloads/index.php.

[37] "Corel Draw". COREL inc. Capturado em 05 de maio de 2004. On-line. Disponível em: http://www.corel.com.

[38] "Adobe PhotoShop". ADOBE inc. Capturado em 05 de maio de 2004. On-line. Disponível em: http://www.adobe.com/products/photoshop/main.html.

[39] "Macromedia FireWorks". Macromedia inc. Capturado em 07 de maio de 2004. On-line. Disponível em: http://www.macromedia.com/software/fireworks/.

[40] "Datasheet Java™ 2 Platform, Micro Edition (J2ME™) Wireless Toolkit". Capturado em 24 de outubro de 2003. On-line. Disponível em: http://java.sun.com/j2me/docs/j2me_wireless_toolkit.pdf.

[41] "SMTK for Java™ Development - Emulator Packs". Siemens Developers Portal. Capturado em 15 de junho de 2004. On-line. Disponível em: https://communication-market.siemens.de/portal/main.asp.

[42] "iDen SDK for J2ME™". Motorola Developers Portal. Capturado em 18 de junho de 2004. On-line. Disponível em: http://idenphones.motorola.com/idenDeveloper//developer/developer_home.jsp.

[43] "Nokia Developer´s Suite 2.1 for J2ME™". Nokia Developers Portal. Capturado em 17 de junho de 2004. On-line. Disponível em: http://www.forum.nokia.com/main/0,6566,034-2,00.html.

[44] "Sony Ericsson WTK 2.1". Sonyericsson Developers Portal. Capturado em 19 de junho de 2004. On-line. Disponível em: http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp.

[45] "ProGuard Obfuscador". ProGuard Manual. Capturado em 27 de maio de 2004. On-line. Disponível em: http://proguard.sourceforge.net/manual/.

[46] "Portable Network Graphics (PNG) Specification and Extensions". Capturado em 10 de junho de 2004. On-line. Disponível em: http://www.libpng.org/pub/png/spec/.

[47] "Mobile Media API (MMAPI); JSR 135". Capturado em 7 de junho de 2004. On-line. Disponível em: http://java.sun.com/products/mmapi/index.jsp.

[48] "J2ME Web Services APIs (WSA), JSR 172". Capturado em 8 de junho de 2004. On-line. Disponível em: http://java.sun.com/products/wsa/.

[49] SHIVAS, Mike. **J2ME Game Optimization Secrets.** Portal Desenvolvedores Java. Capturado em 15 de novembro de 2003. On-line. Disponível em: http://www.microjava.com/articles/techtalk/optimization.