DESENVOLVIMENTO DE JOGO CASUAL SOBRE A PLATAFORMA J2ME

Edson Toshio N. T. da Silva, Diogo de O. Alves, Osvaldo A. da Silveira e Rodrigo H. Yamashiro

Universidade Nove de Julho/Departamento de Informática, São Paulo, Brasil

e-mail: toshio@uninove.br

Resumo: Com o crescente aumento da utilização do telefone celular, esse dispositivo móvel passou a fazer parte do cotidiano de bilhões de pessoas ao redor do mundo. Devido à evolução da tecnologia empregada nesse tipo de aparelho ser cada vez maior, hoje, o seu uso não está restrito apenas à comunicação, mas também ao entretenimento, ao marketing e até mesmo para o pagamento de contas. Este artigo apresenta um estudo sobre a tecnologia Java sobre a plataforma J2ME, demonstrando como resultado o desenvolvimento de um jogo casual para dispositivos móveis, chamado DangerDove, totalmente funcional e portátil, apresentando ótima interface gráfica, desafiando os jogadores a ultrapassarem os seus recordes e promovendo a sua diversão.

***Palavras Chaves*: Java, J2ME, telefonia celular, jogo casual, entretenimento**

I. Introdução

Devido ao grande crescimento do mercado de jogos eletrônicos, de maneira que hoje os seus lucros ultrapassam os da bilionária indústria cinematográfica, muitas tecnologias têm sido desenvolvidas a fim de darem aos jogos características cada vez mais realistas, aumentando sobremaneira a procura dos jogadores por diversões eletrônicas como forma de entretenimento. Naturalmente, o uso dessas tecnologias exige a utilização de consoles e/ou computadores cada vez mais sofisticados e com maiores recursos quando comparados com equipamentos mais antigos, além do maior capital investido pelo jogador e de sua disponibilidade de ficar, pelo menos, algumas horas se dedicando exclusivamente a determinado jogo.

Na contramão dessa realidade, segundo (CERCHIARO e SANTOS 2006), o desenvolvimento de jogos para plataformas móveis, mais especificamente os aparelhos de telefonia celular, deve encontrar maneiras de adaptar seus projetos às limitações tecnológicas destes aparelhos. Por isso, as características desses jogos são muito mais simples em relação aos supracitados, mas garantem a diversão e a distração de seu jogador, mesmo que por um pequeno intervalo de tempo e a um preço realmente acessível. De acordo com uma pesquisa realizada pela Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos (ABRAGAMES), a produção nacional de jogos para telefones celulares cresceu significativamente (ABRAGAMES 2008) .

Este trabalho tem o propósito de demonstrar como um jogo simples e casual para aparelhos de telefonia móvel pode ser desenvolvido utilizando técnicas diferentes daquelas empregadas em desenvolvimento de sistemas e aplicativos informatizados ou mesmo dos métodos usados para a criação de jogos para consoles ou computadores.

O desenvolvimento do artigo está dividido da seguinte forma: além da Seção I que trata da sua introdução, são utilizadas as seguintes seções: Seção II, onde são descritos os materiais e os métodos utilizados para o desenvolvimento; a Seção III, dedicada aos resultados obtidos e a Seção IV, na qual se encontra a conclusão do trabalho.

II. Materiais e Métodos

O Mercado de Jogos Eletrônicos

O consumo de jogos eletrônicos tem crescido exponencialmente nos últimos anos. Fabricantes de consoles, como Microsoft, Nintendo e Sony tem conseguido atingir lucros bilionários com a venda de seus produtos pelo mundo. Tudo isto demonstra a seriedade e o poder por trás dessa indústria e como as pessoas, de diferentes faixas etárias, se interessam em investir na compra daquilo que lhes forneça entretenimento.

Conforme descreve (CERCHIARO e SANTOS 2006), ao mesmo tempo em que os PC‘s respondem por mais de 60% dessas vendas, mais de 20% desse mercado é dominado pelos jogos para celulares e vêm crescendo a cada ano. O restante do mercado é ocupado pelos jogos destinados aos consoles.

O que contribui para o crescimento das vendas para telefonia celulares, é o valor destinado ao desenvolvimento para plataformas móveis, que segundo (DALMAZO 2009), custa 1,5% do valor médio de um jogo tradicional, normalmente estimado em 10 milhões de dólares, podendo ser produzidos em maior volume. Além disso, o preço de venda dos jogos eletrônicos para telefonia celular gira em torno de 10 reais, enquanto os downloads para PC’s custam até 45 reais. Sem contar o valor dos jogos para consoles, que podem chegar a mais de 200 reais cada.

A Situação Brasileira

Com a missão de promover o mercado de games no Brasil, foi fundada em abril de 2004 a Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos, a ABRAGAMES. Entre os incentivos para o setor está a realização da BRGAMES, um programa em parceria com o Ministério da Cultura, que tem como objetivo fomentar o desenvolvimento dos jogos eletrônicos e a criação de ambientes propícios no Brasil, além da sua participação no mercado externo.

Devido à crescente pirataria no Brasil, principalmente no que tange os jogos para consoles, as vendas para a Internet, via download e para o mercado móvel, ou seja, para telefones celulares, têm conseguido atingir lucros cada vez maiores, haja vista que devido ao seu valor ser muito menor em comparação com os jogos para consoles e PC's, também propiciam entretenimento satisfatório a seus usuários.

Dentro desse nicho, o que mais tem chamado a atenção dos desenvolvedores, até mesmo de grandes empresas como a Nintendo e a Tec Toy, são os jogos casuais. Estes são jogos simples e despretensiosos, mas cativam seu público pela jogabilidade e possibilidade de os utilizarem como passa tempo em momentos de lazer.

O mercado de desenvolvimento de jogos casuais ainda é pouco explorado em comparação ao da indústria dos jogos eletrônicos para consoles. Isso se deve às limitações existentes no hardware dos dispositivos móveis, pois recursos como memória, armazenamento permanente e energia são abundantes em computadores tradicionais, mas preciosos em dispositivos pequenos, conforme (KEOGH 2003), dificultando a criação de jogos com maior qualidade competitiva.

Este trabalho trata exatamente deste segmento de jogos casuais para telefonia celular, onde será apresentado o desenvolvimento de um jogo simples, mas capaz de atrair um grande número de usuários, contribuindo dessa maneira para a expansão de artigos acadêmicos neste tema ainda não tão estudado no Brasil.

Plataformas de Desenvolvimento

Diversas tecnologias e plataformas têm sido construídas a fim de auxiliar os desenvolvedores a criarem os seus jogos bem como outros aplicativos para dispositivos móveis, dentre elas destacam-se: WAP, i-mode, BREW e J2ME.

Devido a sua portabilidade e maior aceitação no mercado, a plataforma J2ME é a mais utilizada para tais objetivos, destacando-se grandemente em relação às outras. Como este artigo trata do desenvolvimento de um jogo casual utilizando o J2ME, discorreremos a seguir acerca das principais características dessa plataforma.

Java to Micro Edition (J2ME)

A J2ME oferece um ambiente robusto e flexível para aplicações que rodem em telefones celulares e outros dispositivos integrados, como assistentes digitais pessoais (PDA's), conversores de sinal digital e impressoras. Além disso, tais aplicações são portáveis para vários dispositivos, permitindo assim alavancar as capacidades nativas de cada dispositivo.

Segundo a (SUN MICROSYSTEMS 2009), pode-se apontar como principais componentes da J2ME o CDC (Conneted Device Configurations, ou Configurações para Dispositivos Conectados), o CLDC (Connected Limited Device Configurations, ou Configurações para Dispositivos com Conexão Limitada) e o MIDP (Mobile Information Device Profiles, ou Perfis de Informações de Dispositivos Móveis) entre outras ferramentas e tecnologias inerentes. Assim, é correto afirmar que a arquitetura da J2ME é dividida em Configurations (Configurações), Profiles (Perfis) e API's opcionais.

O CDC abrange dispositivos com maior poder de processamento e memória, além de melhor conexão de rede, como exemplo temos os laptops e alguns tipos de PDA's, como os Pocket PC's da Microsoft.

O CLDC é específico para dispositivos pequenos, como telefones celulares, PDA's e pagers, ou seja, aparelhos com baixo poder de processamento, limitações de memória e largura de banda.

O MIDP é a camada que contém API's Java para conexões de usuários em rede, persistência de armazenamento e a interface com o usuário. Em conjunto com o CLDC é possível obter a funcionalidade do aplicativo principal exigidas pelas aplicações móveis.

As API's opcionais são funcionalidades adicionais específicas que não serão encontradas em todos os dispositivos, mas que possuem sua devida importância.

Para (KEOGH 2003), escrever programas em Java é semelhante à codificação em C++, onde o programador constrói seu código-fonte em um ambiente de desenvolvimento integrado ou em um editor específico. Após este processo, o programa é compilado. Nesta etapa, surgem as diferenças das duas linguagens. O programa escrito em C, após compilado, torna-se um executável que pode rodar em uma máquina apropriada, enquanto o compilador Java converte o código-fonte em bytecode, o qual será executado pela Máquina Virtual Java (JVM). Instruções específicas de máquina não estão incluídos no bytecode. Em vez disso, elas já residem na JVM, a qual é uma máquina específica. Embora o compilador Java gere bytecode que deve ser interpretado pela JVM em tempo de execução, o número de instruções que precisam de tradução são geralmente mínimos e já foram otimizados pelo compilador Java.

No caso da J2ME, especificamente na CLDC, a máquina virtual utilizada é a KJava Virtual Machine (KVM), que faz parte do menor ambiente de tempo de execução (JRE – Java Runtime Environment) e é usada em dispositivos com memória e potência de CPU

limitadas. O KVM é semelhante ao JVM, pelo fato de ser o mecanismo que executa o aplicativo e o mini-aplicativo desenvolvidos com a tecnologia Java. A diferença é que o KVM é usado em telefones celulares e dispositivos móveis, enquanto o JVM é usado em computadores.

Figura 1 - Arquitetura do J2ME.

O Java Micro Edition não contém todos os pacotes e classes existentes na Java Standard Edition ou em outras versões de Java destinadas ao desenvolvimento de aplicativos para desktop e/ou servidores. Dessa forma, o aplicativo a ser criado em J2ME deve ser desenvolvido sem tais recursos ou implementados diretamente pelo desenvolvedor.

Um exemplo da falta de recursos é a ausência de suporte a números em ponto flutuante, algo importantíssimo para a criação de jogos eletrônicos. Em outras palavras, não é possível efetuar cálculos decimais ou utilizar coisas tais como seno, coseno e tangente. Mas segundo (LAM 2004) , é possível superar este tipo de problema utilizando a chamada matemática de ponto-fixo (fixed-point math), a qual consiste em mapear os números decimais em inteiros. Se desejar, por exemplo, uma precisão de 3 (três) casas decimais como 1.000, basta representar o número como 1000. Observa-se que devido ao não suporte a tipos como float e double em J2ME, efetua-se o cálculo através do uso de números com tipos integer ou long. Isto é possível com a aplicação de alguns métodos disponiveis na plataforma de desenvolvimento. Assim, embora o desenvolvedor não possa realizar a operação 1.575 * 2.893, ele pode fazê-la através da representação 1575 * 2893, contanto que se lembre da localização exata do ponto decimal fixo.

Com a maneira descrita acima, é possível utilizar os valores referentes ao seno, coseno e tangente. Sendo que para movimentar um sprite, que será descrito com mais detalhes à frente neste artigo, na tela de um telefone celular de maneira fluente é preciso a definição de 15 direções de ângulos, desconsiderando 0º e 360º, já que possuem mesma direção. Como exemplo, o seno de 22.5º é igual a 0.383, com o uso do ponto-fixo é ajustado para 383. A tabela abaixo apresenta os respectivos valores de seno e coseno em relação aos graus estabelecidos.

Tabela 1 – Valores de seno e coseno em J2ME.

Graus	Seno (Valor Decimal * 1000)	Coseno (Valor Decimal * 1000)
0	0	1000
22.5	383	924
45	707	707
67.5	924	383
90	1000	0
112.5	924	-383
135	707	-707
157.5	383	-924
180	0	-1000
202.5	-383	-924
225	-707	-707
247.5	-924	-383
270	-1000	0
292.5	-924	383
315	-707	707
337.5	-383	924
360	0	1000

O Desenvolvimento do Jogo Casual

Um jogo casual é direcionado a uma audiência em massa, conhecida como jogadores casuais. Estes se diferem dos outros jogadores, voltados aos consoles por exemplo, pelo fato de possuírem tempo ou interesse limitados em relação ao jogo quando comparados àqueles. Assim, os jogos desenvolvidos para tal público, são compostos de regras simples e facilidade de jogo, com o objetivo de agradar pessoas de qualquer faixa etária ou nível de habilidade. O primeiro jogo casual de sucesso foi o Paciência da Microsoft, que acompanhava o software de Sistema Operacional Windows e teve mais de 400 milhões de pessoas que o jogaram desde a sua criação em 1990. De acordo com (DALMAZO 2009), uma pesquisa realizada pela Casual Games Association confirma que a maior parte do público que se interessa por jogos casuais é formado por mulheres, atingindo algo em torno de 51%. Além disso, 62% desse público tem idade superior a 35 anos, impulsionando a criação de jogos contendo animais estimação e até mesmo noivas como personagens principais.

Para o desenvolvimento de um jogo é necessário prévio conhecimento da linguagem a ser utilizada, no caso Java, além de noções de programação, como lógica, algoritmos, estruturas de dados e conceitos de orientação a objetos.

Em primeiro lugar, quando se projeta e desenvolve um jogo eletrônico, este deve ser codificado sem qualquer preocupação com o seu desempenho. O foco primordial é tornar o código limpo, compreensível e funcional para depois fazer a avaliação do desempenho e realizar os ajustes que forem necessários. De acordo com (LAM 2004), isso faz com que se evite o desperdício de tempo com a implementação de táticas de melhoria de desempenho desnecessárias.

Em relação ao pré-requisito de conhecimento do paradigma de linguagem de programação orientada a objetos, é preciso observar que no mundo da telefonia móvel a programação procedural é algumas vezes adotada como método preferido. Isto ocorre devido a limitação inerente ao hardware dos próprios equipamentos, principalmente no que tange a sua memória, pois na orientação a objetos existe a tendência de se utilizar muitas classes, podendo aumentar com o uso de frameworks e design patterns. Dessa forma, a memória, já escassa em dispositivos móveis, fica cada vez mais comprometida. Por isso, deve existir um equilíbrio na maneira em que o jogo eletrônico é codificado, com enfoque nas melhores práticas e padrões sugeridos para a implementação em aparelhos de telefonia celular, a fim de garantir o bom uso da memória e, consequentemente, o seu desempenho.

As aplicações em J2ME são referidas como uma MIDlet que pode rodar em praticamente qualquer dispositivo de comunicação móvel que implemente a máquina virtual Java e a MIDP. Este fato encoraja os desenvolvedores a investirem tempo e dinheiro em aplicações para tais dispositivos sem que aquelas sejam dependentes destes.

As MIDlets são controladas através de um sofware de gestão do aplicativo, conhecido como Aplication Management Software (AMS). Ele é construído pela fabricante dos aparelhos de telefonia celular e neles implementados. A AMS interage com as operações nativas do dispositivo móvel e controla o ciclo de vida da MIDlet, sendo responsável por iniciar, gerenciar e parar a execução da MIDlet.

Programar uma MIDlet é similar à criação de um aplicativo na plataforma Java to Standard Edition (J2SE), em que se define uma classe e os métodos relacionados à ela. Contudo, uma MIDlet é menos robusta do que uma aplicação J2SE devido às restrições impostas pelos dispositivos computacionais pequenos.

Conforme (KEOGH 2003), uma MIDlet é uma classe que extende a classe MIDlet e é a interface entre as declarações do aplicativo e o ambiente de execução, o qual é controlado pelo AMS. Uma classe MIDlet deve conter três métodos necessários ao seu ciclo de vida.

No método startApp() é feita a aquisição de recursos, inicializando a execução. O método pauseApp() libera os recursos em um modo de espera, como por exemplo, quando é necessário atender o telefone em uma chamada. Após isso, o AMS chama novamente o método startApp(). Por fim, caso ocorra alguma exceção durante a execução desse método, a aplicação é destruída automaticamente e o terceiro método, destroyApp(), é chamado, liberando os recursos utilizados pela MIDlet a fim de terminar o aplicativo.

O diagrama abaixo ilustra como funciona o ciclo de vida de uma MIDlet.

Diagrama 1 – Ciclo de vida de uma MIDlet.

Com o intuito de atingir os objetivos anteriormente explanados, utilizar-se-á programas de desenvolvimento para aplicativos móveis, tais como:

- Java (™) ME Platform SDK 3.0 – inclui as ferramentas avançadas encontradas no Java Wireless Toolkit 2.5.2, para CLDC e no Sun Java Toolkit 1.0 para CDC. A sua capacidade expandida inclui emulação over-the-air (OTA), além de suporte a GPS, sensores e muitos outros recursos novos.

- IDE NetBeans 6.5 – ambiente de desenvolvimento integrado utilizado para o desenvolvimento de aplicativos Java.

- JUDE Community – Aplicativo para gerenciamento de projetos.

- Photoshop CS4 e Nero WaveEditor – Editores de imagem e som.

Através desses programas, será possível a realização da codificação e de testes do jogo a ser desenvolvido, no caso deste artigo, o jogo DangerDove, sendo que tais softwares podem ser baixados gratuitamente do site www.sun.com, da empresa Sun Microsystems, recentemente adquirida pela empresa Oracle.

A IDE NetBeans é uma das plataformas preferidas pelos programadores iniciantes no desenvolvimento de aplicativos móveis, pois possui o Mobility Pack, um conjunto de recursos que auxilia grandemente na criação de tais programas. Através dele é possível

utilizar o Java (™) ME Platform SDK no próprio NetBeans, sendo permitido o método de clicar e arrastar os componentes visuais na tela do celular, além de realizar a depuração do projeto. Este é o motivo pelo qual optou-se por este ambiente de desenvolvimento.

O Mobility Pack possui um recurso específico para a construção de jogos para dispositivos móveis, em especial, auxiliando no desenvolvimento de suas interfaces gráficas. Este recurso chama-se Game Builder, que permite ao programador interagir de forma gráfica e através de wizards com os cenários e personagens que há de criar, possibilitando mais tempo à lógica do jogo e ao desenvolvimento do seu código.

Nas primeiras versões da plataforma Java ME, os jogos, independente do estilo adotado, fossem esporte, puzzle, arcade, RPG entre outros, eram desenvolvidos através da codificação pura, pois não existiam classes que implementassem características comuns, como a imagem do jogo, o tamanho do personagem e sua respectiva posição na tela. Essa situação acontecia quando utilizada a MIDP 1.0.

Cientes da dificuldade existente, com o intuito de suprir essa deficiência, a Sun disponibilizou na versão MIDP 2.0 a Game API, que possui uma biblioteca poderosa de classes voltadas exclusivamente ao desenvolvimento de jogos. Isto possibilitou não apenas uma maior produtividade para os desenvolvedores, mas também a diminuição do tamanho dos jogos gerados em J2ME, pois os códigos que gerenciam os personagens e os cenários começaram a fazer parte da biblioteca de classe, deixando-os mais enxutos e assim, economizando espaço e memória.

Dos recursos oferecidos pelo Game API, destacam-se:

- GameCanvas – evolução da classe Canvas, existente na MIDP 1.0, é projetada especificamente para o desenvolvimento de jogos. Com ela é possível controlar todo o jogo com apenas uma Thread, o que era feito em uma separada na MIDP 1.0. Através do método getKeyStates pode-se capturar eventos do teclado, até mesmo captura de teclas simultâneas e daquelas que são mantidas pressionadas.

- Sprite – esta classe é um elemento gráfico do jogo, normalmente usada para os personagens. Com os seus métodos é possível verificar colisões, transformar imagens e visualizar uma sequência delas, facilitando em muito o trabalho do programador.

- LayerManager – esta classe é utilizada para a criação de cenários. Através dela pode-se obter e gerenciar todos os componentes visuais do jogo, organizando-os em camadas. Quando inserido no LayerManager, o componente visual recebe uma prioridade, a qual é representada pela letra z. Dessa forma, os elementos gráficos com maior prioridade poderão sobrescrever aqueles com prioridade menor quando necessário.
- TiledLayer – classe usada para a construção de cenários de jogos. Através dela é possível montar imagens a partir da repetição de pequenas imagens, criando um cenário maior e mais complexo.

Estudo de Caso

Para o desenvolvimento do DangerDove, foi escolhido como base para sua criação o jogo FightFight, devido à sua simplicidade, boa jogabilidade e à sua semelhança ao clássico Asteróides.

O FightFight trata-se, basicamente, de uma nave espacial responsável por destruir os asteróides que vem contra ela durante sua viagem no espaço sideral. O projeto base foi desenvolvido por Hoang Tuanbs, podendo o seu código ser acessado pelo endereço http://www.codeproject.com/KB/java/GameProgrammingInJ2ME.aspx. Serão feitas aqui certas modificações no código a fim de melhorar alguns aspectos relacionados à velocidade do jogo, a quantidade de disparos permitidos pela nave, além do acréscimo do menu de opções para o jogo.

Mostra-se a seguir os diagramas de classes referentes ao jogo FightFight:

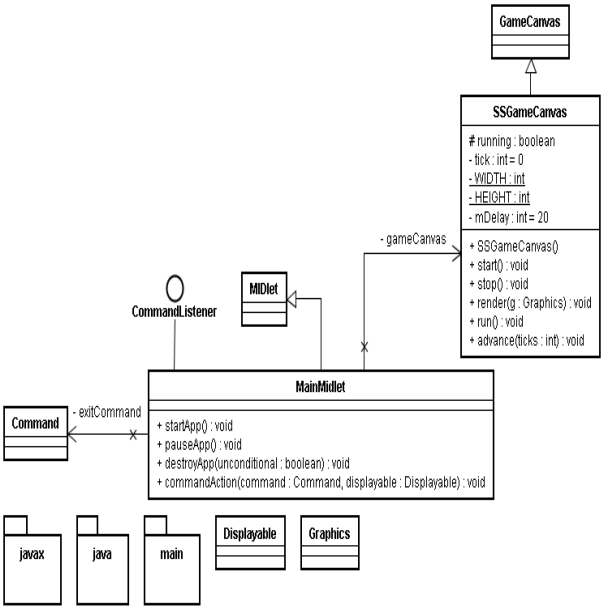

Diagrama 2 – Diagrama de classe da MainMidlet.

Este diagrama de classe descreve a MainMidlet, ou seja, a MIDlet principal do jogo, mostrando suas relações com as demais classes do projeto. Por ser a principal MIDlet, é nela que estão contidos os três métodos básicos supracitados, necessários ao seu ciclo de vida, que são: startApp(), pauseApp() e destroyApp().

Logo abaixo, são exibidos os diagramas detalhados de cada uma das classes relacionadas à MainMidlet, com suas respectivas identificações, atributos, métodos

e ligações de uma para com as demais, incluindo nisto as suas heranças.

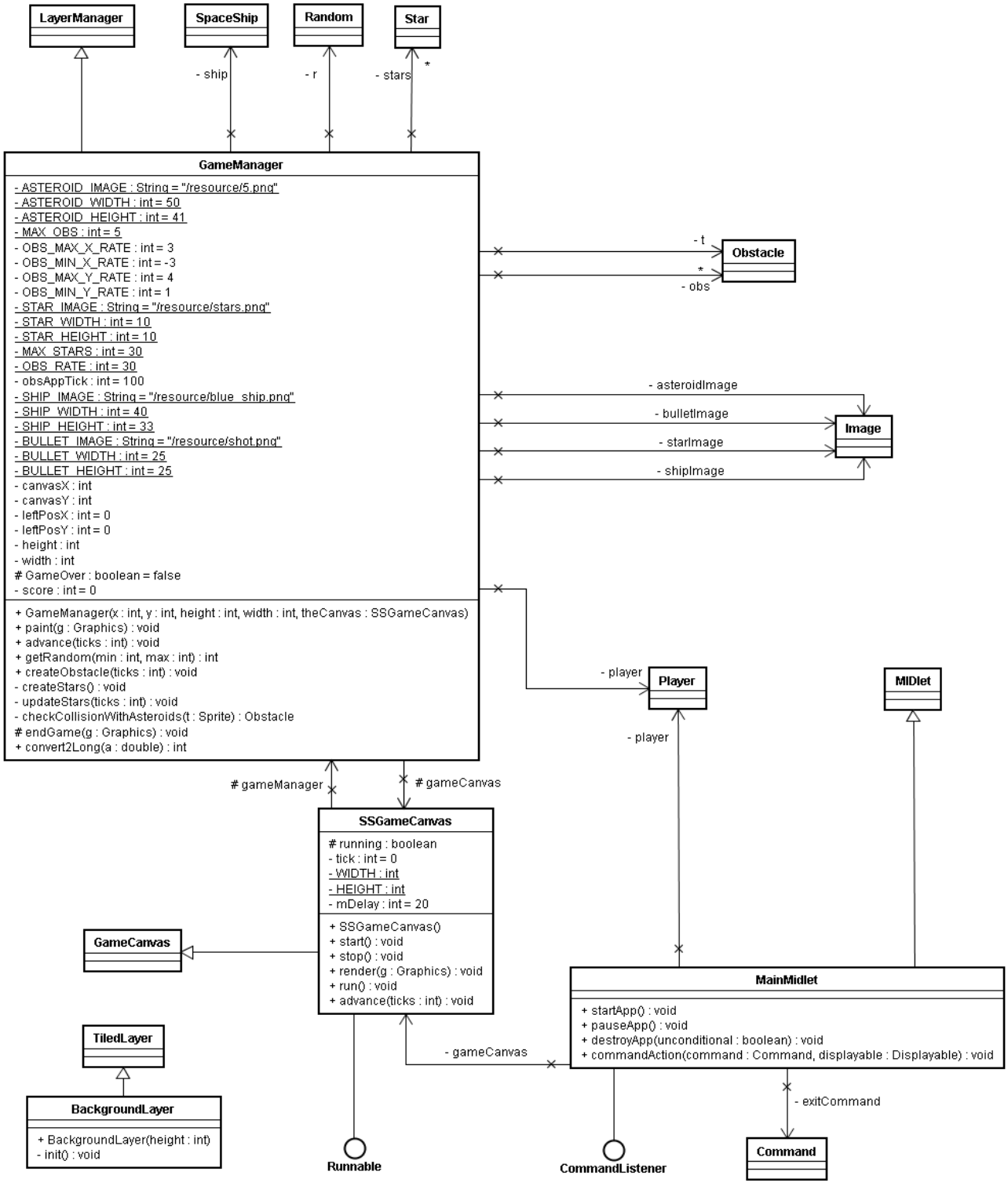

Diagrama 3 – Diagrama de classe detalhado 1.

Demonstra-se neste diagrama o relacionamento com a classe SSGameCanvas, que herda recursos importantes da classe GameCanvas. É também responsável pela implementação da interface Runnable, já que Java não permite que uma classe herde mais de uma classe ao mesmo tempo, necessária para a execução da thread necessária para o jogo. Além disso, nessa classe é chamado o método responsável pela renderização dos gráficos. A classe GameManager implementa a criação dos asteróides, da nave espacial,

das estrelas e das balas dos disparos, trabalhando com a posição dos sprites na tela e suas colisões.

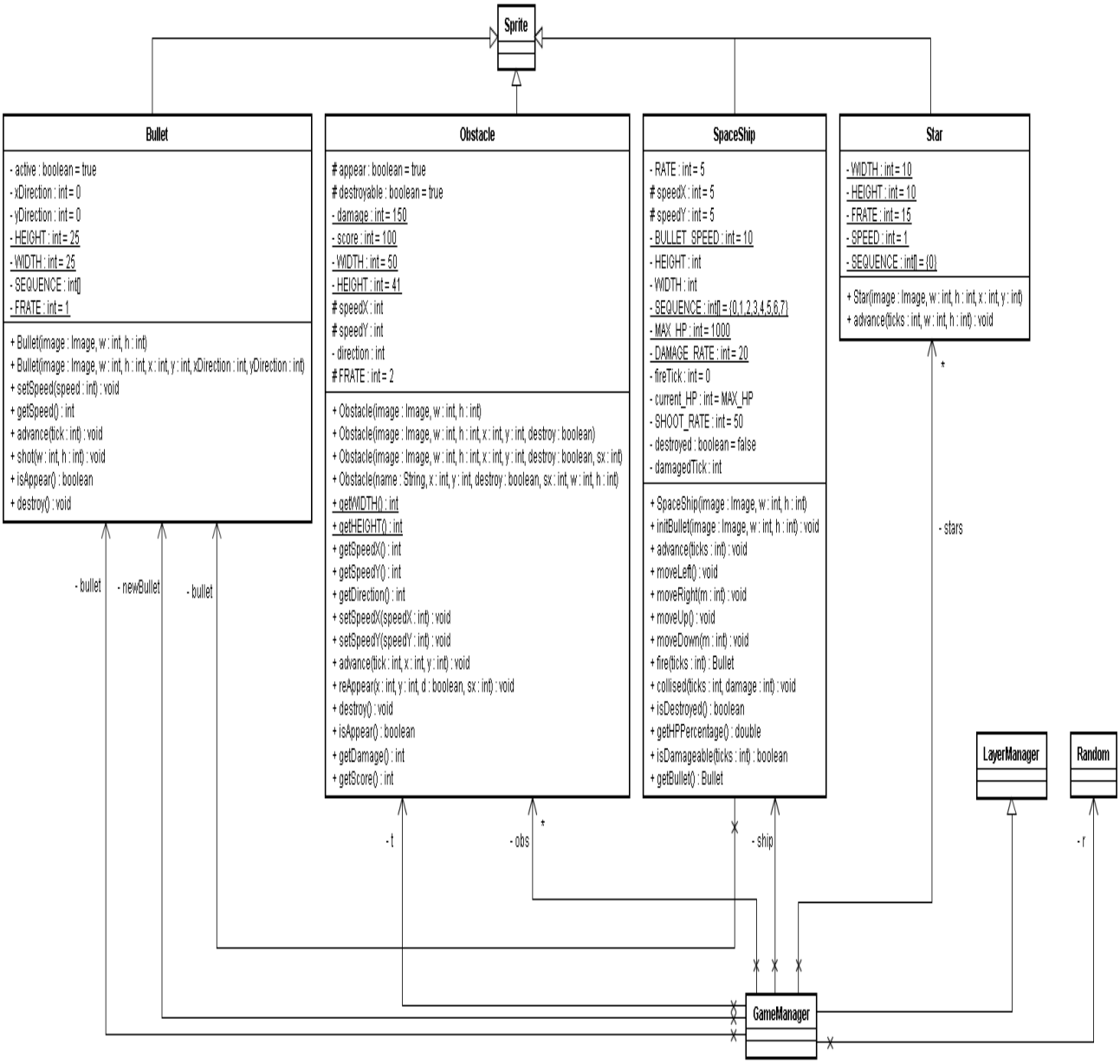

Diagrama 4 – Diagrama de classe detalhado 2**.**

Este diagrama já demonstra como a classe GameManager adquire os parâmetros e valores das classes acima para a criação dos sprites citados. Mostra também como a classe Bullet, responsável pela bala disparada, a classe Obstacle, responsável pelos asteróides, a classe SpaceShip, responsável pela nave espacial e a classe Star, responsável pelas estrelas herdam da classe Sprite os seus atributos e métodos, além daqueles específicos a elas.

Segue ao lado o jogo rodando no emulador do NetBeans. Neste caso, foi preciso utilizar o Java Wireless Toolkit 2.5.2, pois o jogo foi desenvolvido nesta versão e apresentava algumas inconsistências na mais recente.

Figura 2 – Jogo FightFight

Conforme observado durante os testes do código e sua relação com o funcionamento do jogo, notou-se a excelência da qualidade gráfica quando comparado ao que se pode obter de dispositivos móveis, devido aos seus parcos recursos de hardware e software. Isto foi possível devido aos programas utilizados para codificação e testes, além da otimização do código necessária para que a memória do dispositivo não se sobrecarregasse.

Podemos relatar também que os resultados obtidos em primeira instância, ou seja, sem as modificações propostas no item anterior, foram:

- Não há um menu onde possam ser verificadas informações sobre o jogo, escolha de nível para o jogador, visualização dos recordes e nem mesmo opção para iniciar o jogo.
- O usuário deve esperar aproximadamente 15 segundos para o início do jogo.
- Após cada tiro disparado, é necessário que o usuário aguarde um pequeno momento para que possa disparar novamente, possibilitando que o jogador seja atingido pelos asteróides sem poder destruí-los.
- Não há novas fases nem aumento do nível ou velocidade do jogo após atingir determinado número de pontos.
- Após o término do jogo, não existe a possibilidade de registrar o nome do jogador e sua quantidade de pontos para montagem do recorde.

Devido às observações acima explanadas, foram sugeridas que tais modificações fossem implementadas no código do jogo, tendo em vista o seu aperfeiçoamento e consequentemente, o aumento de interesse por parte do jogador em utilizá-lo como meio de entretenimento nos momentos vagos que este dispuser. Além disso, a temática do jogo será mudada para uma pomba que deve combater naves alienígenas que tentam invadir a Terra.

III. Resultados

Para que todas as transformações, melhorias e adaptações necessárias ao jogo fossem inseridas, utilizou-se a ferramenta de edição de imagens Photoshop CS4, a fim de criar o novo cenário, a pomba, as nuvens e os inimigos.

Adaptação do código-fonte de FightFight

Nesta parte do trabalho se faz importante salientar que este não teve o caráter de desenvolver um jogo por completo desde seu início, mesmo porque pouquíssimos jogos são produzidos dessa forma. No desenvolvimento de jogos de grande porte, para consoles e PCs, é extremamente difundido o conceito de motores de jogos, os conhecidos Game Engines, de forma que diversos jogos são produzidos sobre uma mesma estrutura de código pré-desenvolvida, que lhes fornece simulação de física, renderização, detecção de colisões, entre outros. Aplicando este mesmo conceito a plataforma J2ME, pode-se criar diversos jogos, com temática diferenciada, utilizando como base o código de um jogo funcional, lembrando ainda que a reusabilidade é um dos principais aspectos da linguagem Java.

FightFight vs. DangerDove

Enquanto compartilham o gênero Shoot' em up, jogos cujo objetivo é atirar contra elementos que aparecem pelo cenário, os dois jogos divergem em sua temática, o que levou às principais adaptações do código. Enquanto FightFight se passa no espaço sideral, com uma nave evitando a colisão com asteróides, DangerDove retrata uma pomba que voa pelo céu disparando contra discos voadores que investem em sua direção.

Plano de fundo

O código original de Hoang Tuanbs fazia menção de utilizar uma classe TiledLayer para compor o fundo do cenário, no caso, corpos celestes distantes. Esta classe constrói uma imagem a partir da combinação dos frames de um sprite, de forma que um mesmo sprite pode gerar figuras diferentes e estas sejam modificadas dinamicamente.

DangerDove, neste quesito, tem necessidades mais humildes, precisando apenas de um plano de fundo azul celeste que represente o céu diurno em que a ação do jogo se dá. Para isto, a classe Background do projeto original foi excluída, e uma simples modificação dos valores passados ao método setColor(), na classe SSGameCanvas, bastou para que o objetivo fosse alcançado. O método setColor() recebe um ternário no padrão RGB e aplica a cor referente a tela da aplicação.

Novo Protagonista

Apesar de parecerem bem distintos, a troca do personagem principal foi um dos implementos mais fáceis, pois compartilham a mesma estrutura, mudando apenas as figuras de seus sprites. Para facilitar o processo de transição, o novo sprite foi construído com dimensões e composição iguais às do original, ou seja, oito frames de 40x33 pixels, mas agora representando uma pomba batendo suas asas, não sendo necessárias alterações no código-fonte além da referência ao arquivo-fonte. Para o tiro disparado pelo protagonista, foi apenas editado o sprite referente, não necessitando nenhuma modificação na codificação. Isto prova o conceito de reusabilidade de código, pois apenas mudando os elementos representados nos sprites pode-se obter um jogo completamente diferente.

Velocidade dos tiros

Observou-se que a taxa com que os tiros podiam ser disparados era relativamente baixa no projeto original, o que causava frustração no jogador. A fim de melhorar este aspecto, houve um incremento na taxa de disparos e um decremento do tempo entre estes, alterando as constantes RATE e SHOOT_RATE da classe SpaceShip.

Adaptação das estrelas às nuvens

Em DangerDove, haverá nuvens surgindo no céu, diferentemente do FightFight que apresenta estrelas. Desta forma, adaptou-se o código gerador das estrelas para tal objetivo.

A primeira modificação necessária foi criar novos sprites condizentes ao tema, o que foi feito com a ajuda do editor mencionado anteriormente. Porém, as imagens referentes às nuvens eram maiores do que as originais, fazendo-se necessária a modificação do código da classe GameManager.

As constantes STAR_WIDTH e STAR_HEIGHT são utilizadas pela classe Star, que herda as características da classe Sprite, para definir a área da figura total que representa um sprite. Portanto, tiveram seus valores modificados a fim de abrangerem os novos limites.

A segunda dificuldade encontrada nesta implementação foi a geração diferenciada de nuvens. No espaço sideral representado em FightFight as estrelas eram todas iguais, porém é estranho imaginar um céu com nuvens exatamente iguais no contexto de DangerDove. Por isso, foi acrescentada à assinatura do método construtor da classe Star uma nova variável incumbida de definir qual Sprite da nova figura, agora com três modelos de nuvens, seria utilizado na construção dinâmica do objeto.

Para gerar o valor desta variável forma estudadas duas aproximações:

- Utilizar um objeto Random já implementado na classe GameManager para gerar um número aleatório entre 0 e 2, referentes aos modelos de nuvens dos sprites. Esta abordagem se mostrou falha, pois nem sempre todos modelos eram utilizados devido ao caráter imprevisível da classe random.

- Utilizar o contador i do método createStars() e o operador %, que representa o resto da divisão inteira, para que a cada objeto gerado seja utilizado um sprite diferente. Esta abordagem foi a utilizada pois garante a utilização dos três modelos de nuvens. A tabela a seguir demonstra este processo:

Tabela 2 – Contador i da classe Star.

i % 3	**Resultado**
0 % 3	0
1 % 3	1
2 % 3	2
3 % 3	0
4 % 3	1
5 % 3	2
6 % 3	0
7 % 3	1
8 % 3	2
9 % 3	0
10 % 3	1

Inimigos animados

No jogo original, os inimigos eram representados por imagens estáticas de asteróides que vinham em rota de colisão contra a nave do jogador. É conveniente que rochas espaciais, a não ser pelo seu movimento natural provocado pela inércia, são objetos bem estáticos, o que não era o caso dos discos voadores projetados para DangerDove. Para tal, foi desenhado um novo sprite composto por oito frames que representavam o movimento giratório dos OVNI's, bem como um certo molejo em sua movimentação. Além de alterações na classe GameMananger para suportar o novo formato do sprite, como no caso das nuvens, foram necessários alguns incrementos na classe Obstacle, representante dos inimigos, a fim de torná-la compatível com objetos animados, assemelhando-se a classe SpaceShip, que representa o protagonista. Para tanto, foi acrescentado aos seus construtores o método setFrameSequence(), bem como um vetor à classe com a seqüência de frames a ser passada a este método. Por fim, foi acrescentado ao método advance(), responsável pela atualização do objeto na tela, o método nextFrame(), que verifica qual o próximo frame da animação, conforme definido por setFrameSequence() e atualiza a imagem, criando a ilusão de movimento.

Splash Screen e Menu principal

Splash Screen é o termo utilizado para descrever a tela de abertura que geralmente aparece por alguns segundos antes no menu principal. Em geral, contém o nome do jogo e/ou informações resumidas do desenvolvedor. O menu tem como objetivo dar ao jogador outras opções além de iniciar o jogo em si, tais como configurar preferências, consultar recordes, acessar instruções sobre o jogo, etc.

Devido a abordagem simplista de Tuanbs, seu jogo não possuía estes dois elementos, carregando diretamente o cenário ao se executar a aplicação. Assim, uma de nossas metas foi adicioná-los a DangerDove, seguindo o objetivo geral de aprimorar o jogo original.

O primeiro passo foi a criação de nossa Splash, o que foi alcançado com o auxilio do editor de imagens Photoshop CS4, também utilizado na criação dos sprites e wallpaper do jogo.

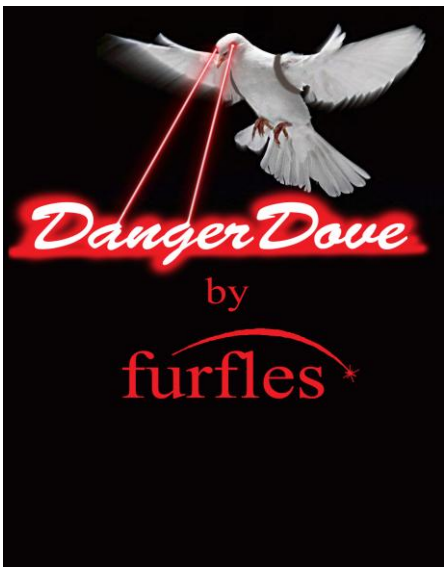

Figura 3 – Splash do jogo DangerDove.

Adotou-se a estrutura simples de Splash Screen e Menu apresentada por (Lam 2004). Assim, foram adicionados ao projeto as classes SplashScreen, MainMenuScreen e CountDown, bem como as adaptações necessárias à nossa classe principal, a MainMidlet.

Desta forma, o método startApp() da classe MainMidlet não mais inicia a construção do ambiente do jogo, mas sim instancia um novo objeto de SplashScreen. Este, por sua vez, recebe entre seus parâmetros a instância de MainMenuScreen, a imagem a ser exibida e o tempo para exibi-la. A contagem deste tempo é fornecida pela classe CountDown e após passado, a tela de menu é exibida. Nota-se que o objeto referente ao menu já havia sido instanciado por startApp() e SplashScreen apenas o torna visível através do método setCurrent(). A construção do ambiente de jogo passa a ser feita pelo novo método gameScreenShow() adicionado à classe principal e podendo ser invocado pelo menu. Este método executa todos os procedimentos antes pertencentes a startApp().

No caso de DangerDove, o menu exibirá as opções Novo Jogo, Ajuda e Sobre. Através de uma estrutura de controle switch-case, o método processMenu() identifica a opção selecionada pelo jogador e executa o método ligado a ela.

- Novo jogo: Invoca o método scnNewGame(), que por sua vez invoca gameScreenShow(), dando início ao jogo propriamente dito.

- Ajuda: Invoca o método scnHelp(), que exibirá informações sobre o tema do jogo e instruções acerca de como jogá-lo.

- Sobre: Invoca o método scnHelp(), que exibirá informações sobre a versão da aplicação e sobre a equipe de desenvolvimento.

Som de início

FightFight apresenta um som de tiros para indicar o início do jogo. Como em DangerDove seria mais interessante que tal indicação ocorresse junto à tela de Splash, as funções relativas foram movidas para startApp() na classe principal. Foi também substituído o arquivo de som por um que proclama enfaticamente o nome do jogo, fazendo referência a jogos clássicos dos anos 90. O arquivo, que apresenta um efeito de reverberação, foi editado com a ajuda do software Nero WaveEditor.

Ajuste da Splash Screen à Tela do dispositivo

Ao testar o jogo em diversos aparelhos, percebeu-se que a imagem exibida pela Splash Screen ultrapassava os limites da tela de alguns dispositivos. Este problema ocorre devido às diferenças de tamanho e resolução entre os modelos. O problema foi contornado adicionando um novo método à classe principal MainMidlet, com a função de converter a imagem ao tamanho nativo da tela em questão. O método resizeImage() deve receber como parâmetro a imagem a ser redimensionada e necessitará ainda da largura e altura da tela. Para tanto é instanciado um objeto de SSGameCanvas em startApp() que não chega a ser exibido, pois tem o simples objetivo de alimentar as variáveis screenWidth e screenHeight, usadas pelo método supracitado. Isto é feito utilizando os métodos getWidth() e getHeight() que retornam, respectivamente, a largura e a altura da tela.

Com isso a classe SplashScreen não mais recebe um objeto do tipo imagem, mas sim o retorno do método resizeImage(), podendo exibir a tela de abertura no tamanho correto.

Novo Diagrama de Classes

Naturalmente, houve algumas alterações nos diagramas de classes devido às novas implementações. Estas são exibidas abaixo com o intuito de demonstrar como ficaram depois de encerrada a criação do jogo DangerDove.

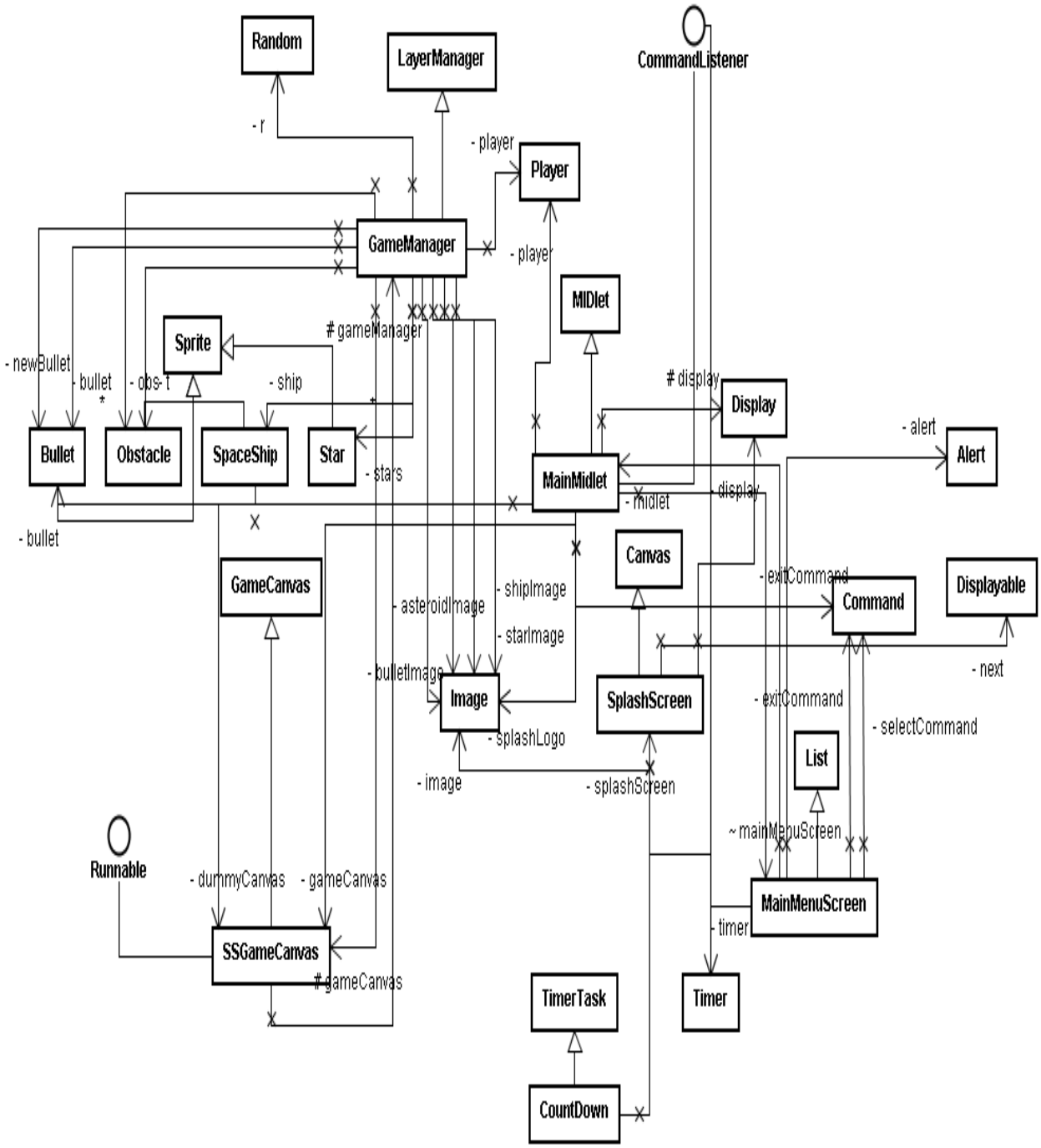

Diagrama 3 – Diagrama de classe detalhado 1.

O diagrama acima mostra o relacionamento das classes anteriormente existentes do projeto de Hoang Tuanbs com a nova classe CountDown, que herda recursos da classe Timer Task, a classe MainMenuScreen, que herda de List e a classe Splash Screen, que herda de Canvas os recursos necessários para seus objetivos.

O diagrama de classes detalhado da classe GameManager e sua relação com as classes Bullet, Obstacle, SpaceShip e Star são exibidas abaixo, juntamente com as alterações nelas introduzidas. Entre essas mudanças está a taxa de disparos da classe SpaceShip que, conforme citado anteriormente, foi aumentada a fim de proporcionar ao jogador maior controle sobre o jogo. Nota-se também o acréscimo do método SEQUENCE na classe Obstacle, responsável pela geração das naves e seu movimento giratório.

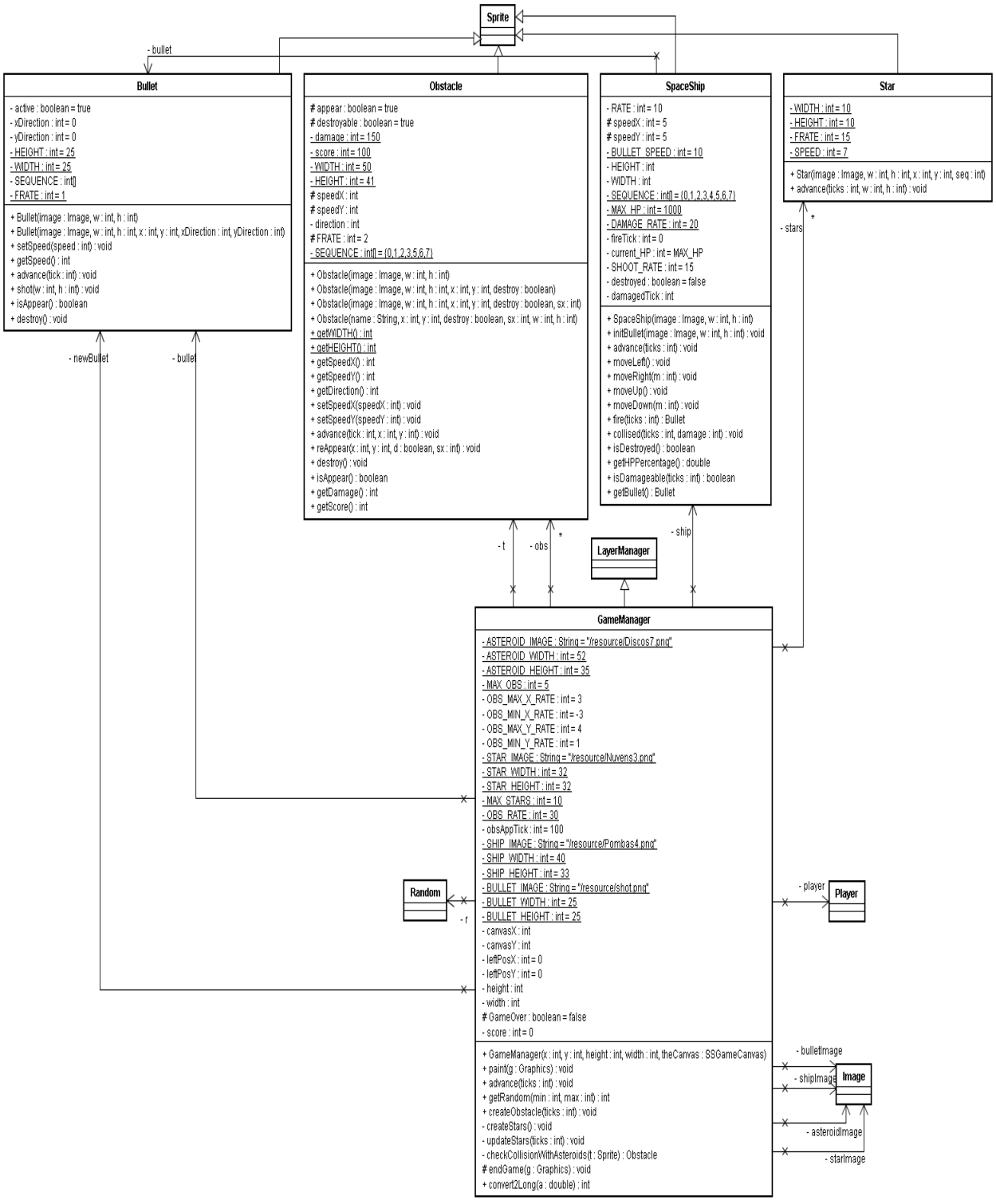

Diagrama 6 – Novo Diagrama de classe 2. (detalhado)

Além da Codificação

Neste ponto do artigo, cabe a observação de que a confecção de um jogo exige muito mais além do conhecimento de programação, sendo necessárias habilidades artísticas variadas e conhecimento de outras ferramentas, além daquelas ligadas a codificação. Isto se mostra particularmente verdadeiro quanto a criação de sprites. Tamanha a relevância deste problema, (Lam 2004) aponta três possíveis soluções com relação aos sprites:

- Procurar Sprites gratuitos disponíveis na Internet.
- Contratar profissionais ou obter sua ajuda gratuitamente dependendo do tipo de projeto.
- Aprender a confeccioná-los por conta própria, se você não se importar em gastar o tempo necessário.

Uma vez que não foram encontrados modelos gratuitos condizentes ao tema em questão e contratar profissionais seria inviável, optou-se pelo terceiro item acima. Inicialmente parecia que o aplicativo Paint da Microsoft seria suficiente para as necessidades do jogo, porém, notou-se a falta de suporte à transparência, essencial na confecção de sprites. Dessa forma, utilizou-se o mais conceituado software em edição de imagens, o Photoshop CS4, mas sua interface pouco amigável mostrou-se um empecilho para se realizar ações que pareciam simples. Ainda houve experimentos com o software aberto Gimp, mas este também se mostrou mais complexo do que o esperado. Por fim, o software escolhido foi o Photoshop CS4 que, graças à abundância de tutoriais e materias disponíveis na Internet, conseguiu-se o conhecimento necessário para atingir os objetivos do jogo, auxiliando na criação dos sprites e do splash screeen.

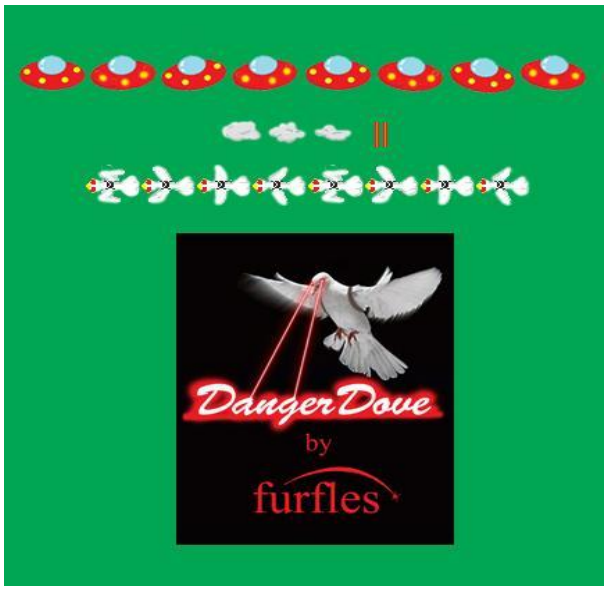

Figura 4 – Sprites e Splash Screen criados e usados no jogo.

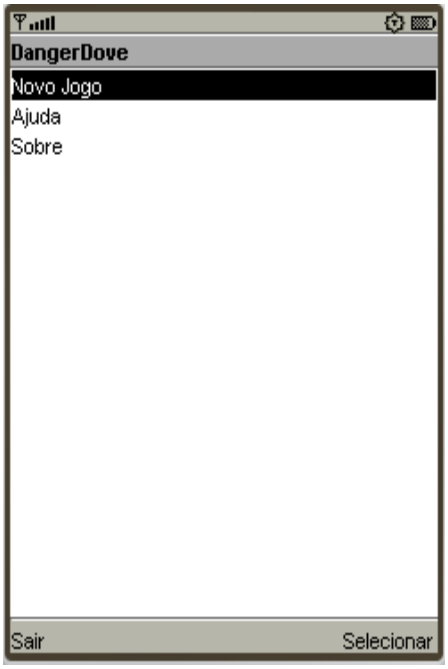

Figura 5 – Tela do menu de DangerDove.

Figura 6 – Tela do jogo DangerDove.

IV. Conclusão

Através do desenvolvimento de DangerDove, verificou-se a eficiência da plataforma J2ME para a criação de jogos para dispositivos móveis, proporcionando aumento de produtividade quando usado em conjunto com o NetBeans e seus poderoso recursos, em especial o Mobility Pack.

Conclui-se que a utilização da tecnologia Java é totalmente satisfatória, seja para os objetivos do presente trabalho como para a criação de outras aplicações voltadas a estes tipos de aparelho, auxiliando

o mercado com aplicações que geram retorno considerável. Esse retorno, salienta-se, não é apenas o financeiro, mas também o do conhecimento adquirido nos campos científico e tecnológico, os quais podem ser implementados em áreas como a inteligência artificial, engenharia de software, computação gráfica, entre outras e suas interações com setores não necessariamente ligados à Tecnologia da Informação, mas essenciais à conclusão e excelência dos trabalhos exigidos.

V. Agradecimentos

Agradecemos a Deus que nos deu a vida e também a sabedoria necessária para que realizássemos este artigo. Agradecemos à Universidade Nove de Julho – Uninove por nos proporcionar o apoio necessário para a obtenção de todas as informações para a conclusão deste trabalho.

VI. Referências

[1] CERCHIARO, D.R.; SANTOS, E.V.F. (2006). *O Mercado de Games.* (On-line). Disponível: http://www.ceset.unicamp.br/~vladimir/ST067/ST566%20Monografia%20-%20O%20Mercado%20de%20Games.pdf, (25/09/2009).

[2] ABRAGAMES. (2008). *A Indústria Brasileira de Jogos Eletrônicos – Um Mapeamento do Crescimento do Setor nos Últimos 4 Anos.* (On-line). Disponível: http://www.abragames.org/docs/Abragames-Pesquisa2008.pdf, (25/09/2009).

[3] DALMAZO, Luiza. (2009). *Passatempos Digitais.* (On-line). Disponível: http://portalexame.abril.com.br/revista/exame/edicoes/0937/tecnologia/passatempos-digitais-422200.html, (27/09/2009).

[4] KEOGH, James. *J2ME: The Complete Reference*. California: McGraw-Hill/Osborne, 2003.

[5] SUN MICROSYSTEMS (2009). *O que é J2ME?* (On-line). Disponível: http://www.java.com/pt_BR/download/faq/whatis_j2me.xml, (27/09/2009).

[6] LAM, Jason (2004). *J2ME & Gaming.* (On-line). Disponível: http://www.jasonlam604.com/v2/index.php?action=books#, (19/10/2009).