UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

MOBILE-FURBOT: UMA VERSÃO DO FURBOT PARA CRIAÇÃO DE JOGOS EM DISPOSITIVOS MÓVEIS

DANIEL SEVERO ESTRÁZULAS

BLUMENAU
2009

2009/2-04

DANIEL SEVERO ESTRÁZULAS

MOBILE-FURBOT: UMA VERSÃO DO FURBOT PARA CRIAÇÃO DE JOGOS EM DISPOSITIVOS MÓVEIS

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

Prof. Mauro Marcelo Mattos, Dr. - Orientador

BLUMENAU
2009

2009/2-04

MOBILE-FURBOT: UMA VERSÃO DO FURBOT PARA CRIAÇÃO DE JOGOS EM DISPOSITIVOS MÓVEIS

Por

DANIEL SEVERO ESTRÁZULAS

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

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Adilson Vahldick, Ms. – FURB
Membro:	Prof. Roberto Heinzle, Ms. – FURB

Blumenau, 15 de dezembro de 2009

AGRADECIMENTOS

À minha família, que mesmo longe, sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

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

Ao professor, Adilson Vahldick, pelo auxílio e pelas dúvidas tiradas com relação ao *framework* Furbot.

A bravata não é sinônimo de bravura. A bravata e a violência são uma questão menos de forma que de espírito. O homem bravo é consciente de seus deveres e da justiça. Sabe bater-se pelas idéias fazendo dos obstáculos não uma respectiva derrota, mas sim um fator estímulo.

Ginshin Funakoshi

RESUMO

O presente trabalho tem por finalidade construir um ambiente de execução do Furbot onde aplicativos desenvolvidos para versão *desktop* que possam vir a ser rodados em dispositivos móveis compatíveis com a plataforma *Java Micro Editon* (JME). Neste trabalho são apresentados os elementos da arquitetura do Furbot que servem como base para a criação de uma arquitetura voltada para dispositivos móveis. Também é explicado sobre as diferenças de desenvolvimento entre as arquiteturas *destkop* e dispositivos móveis, as quais foram relevantes para viabilizar a construção deste trabalho. O resultado alcançado foi a implementação de uma arquitetura específica para funcionamento em dispositivos móveis através da conversão dos elementos da arquitetura *desktop* do Furbot.

Palavras-chave: Furbot. J2ME. Jogos.

ABSTRACT

The present work aims to build an execution environment where applications developed for the desktop version of Furbot that may be run on mobile devices compatible with the platform Java Micro Edition(JME). This work presents the architecture elements of Furbot that serve as basis for the creation of an architecture aimed at mobile devices. It is also explained about the differences in development between the desktop and mobile architectures which were relevant to facilitate the construction of this work. The result achieved was the implementation of a specific architecture to run on mobile devices through the conversion of the elements of the Furbot desktop architecture.

Key-words: Furbot. J2ME. Games.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

JAD – *Java Application Description*

JAR – *Java Archive*

JME – *Java Micro Edition*

JSE – *Java Standard Edition*

MGBL – *Mobile Game-Based Learning*

MVC – *Model View Controller*

UML – *Unified Modeling Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais são criadas técnicas e metodologias de apoio ao ensino de lógica de programação, porém muitas destas maneiras de ensino acabam por não se tornarem atrativas a ponto de despertar a vontade e a motivação do aluno em aprender e a explorar o desenvolvimento de problemas e algoritmos. De acordo com Bergin (2001, p. 11), “o aprendizado é mais eficiente quando o aluno é motivado. Esta motivação depende de um conteúdo interessante, de um ambiente empolgante e da didática dos professores”.

Dentre alguns *frameworks* existentes para facilitar o aprendizado de lógica de programação, o Furbot caracteriza-se por ter sido concebido para tentar diminuir as dificuldades de aprendizagem e ensino na lógica de programação através de um forte apelo à área de jogos, criando assim uma atmosfera facilitadora ao aprendizado (VAHLDICK; MATTOS, 2009).

A proposta do Furbot surgiu a partir de reflexões dos professores Adilson Vahldick e Mauro Marcelo Mattos relativamente à dificuldade que os acadêmicos dos cursos de Ciência da Computação e Sistemas de Informação da Universidade Regional de Blumenau (FURB) enfrentam na disciplina de programação de computadores.

Conforme Vahldick e Mattos (2009, p. 1), “A experiência desses professores mostra que os alunos não se sentem motivados em resolver exercícios com enunciados como "digite cinco nomes e notas de alunos e mostre a média da sala, a maior e menor nota". Apesar da solução não ser trivial para os alunos iniciantes, eles não se sentem desafiados”.

De acordo com Vahldick e Mattos (2009, p. 3), “o elemento central do Furbot é a programação de um robô que vive num mundo bidimensional junto de outros tipos de objetos, que também podem ser programados”. Sobre este mundo, o aluno desenvolve atividades de movimentação em 4 direções e detecção de obstáculos.

Todos esses aspectos motivaram o desenvolvimento de uma versão do Furbot onde classes desenvolvidas para a versão *desktop* possam ser executadas em dispositivos móveis compatíveis com a plataforma *Java Micro Editon* (JME)[1].

Para permitir a criação do Mobile Furbot foi utilizado o mesmo padrão *Model View*

[1] JME é uma plataforma Java de desenvolvimento voltada para dispositivos móveis ou portáteis, como telefones celulares e *palmtops* (OGLIARI, 2008).

Control (MVC)[2] adotado pela versão original do Furbot, em que a finalidade é separar as camadas de desenvolvimento, para facilitar a utilização de outras formas visuais que farão o desenho do mundo e dos seus objetos. No entanto, a adaptação ao Mobile Furbot fez a arquitetura original sofrer mudanças na estrutura das classes para permitir a execução do novo ambiente em dispositivos móveis.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é criar um ambiente de execução do Furbot em dispositivos móveis baseados em JME.

Os objetivos específicos do trabalho são:

a) adequar a estrutura do Furbot tornando-a compatível para utilização em ambientes móveis;
b) permitir a criação de jogos bidimensionais para celulares;
c) viabilizar a criação e controle de elementos do jogo através da utilização de `threads`.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado em quatro capítulos. No capítulo seguinte é descrita a fundamentação teórica que embasou este trabalho. Na seção 2.1 está a fundamentação sobre aprendizagem baseada em jogos para dispositivos móveis; na seção 2.2 é mostrado uma explicação sobre os componentes e a funcionalidade do *framework* Furbot e por fim na seção seguinte (2.3) são explicados componentes existentes na plataforma JME e como eles são benéficos para a área de jogos. O capítulo 2 é finalizado com os trabalhos correlatos.

O capítulo 3 traz a especificação e implementação do *framework*. Ao final do capítulo

[2] O padrão MVC serve para separar dados ou lógica de negócios (*Model*) da interface do usuário (*View*) e do fluxo da aplicação (*Control*). A idéia é permitir que uma mensagem lógica de negócios possa ser acessada e visualizada através de várias interfaces (PEREIRA, 2004).

são apresentados os resultados alcançados a partir dos testes realizados.

O capítulo 4 contém a conclusão do trabalho, juntamente com sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados alguns aspectos teóricos relacionados ao trabalho, tais como: aprendizagem baseada em jogos, informações sobre o *framework* Furbot, conceitos sobre JME, além do uso desta tecnologia para desenvolvimento de jogos para celular. Na ultima seção são apresentados alguns trabalhos correlatos.

2.1 APRENDIZAGEM BASEADA EM JOGOS *MOBILE*

Atualmente, as funções mais utilizadas nos celulares são as que se referem ao entretenimento. A procura por aplicativos de personalização do aparelho, *ring tones*, imagens, *chats* e principalmente jogos vem aumentando significativamente.

De acordo com Bachmair e Pachler (2009, p. 1), “Se é o caso que dispositivos móveis com suas estruturas tecnológicas sociais e práticas culturais tornaram-se uma parte integral da vida diária, então o campo educacional deve reagir”.

A *m-learning* (*mobile learning*) é uma extensão do *e-learning* (*electronic learning*) e é praticado através de dispositivos móveis, como celulares e *smartphones*, permitindo assim uma maior condição de acesso a recursos pedagógicos, independentemente de tempo e lugar (SIAU; HOONAH, 2008).

Neste contexto, o processo não ocorre em locais fixos, e sim em qualquer lugar, no qual o aprendiz vai usar da tecnologia que tem em mãos para criar uma situação de aprendizagem. Seguindo essa perspectiva, o estudante tem a seu favor toda a interatividade proporcionada pelo objeto de aprendizagem aliada às vantagens da mobilidade, permitindo, assim, progredir seu estudo conforme seu ritmo de aprendizagem (FRANCISCATO; MEDINA, 2009).

Tecnologias móveis na educação podem proporcionar benefícios tanto aos alunos quanto para os professores. Aos alunos é proporcionada uma maior flexibilidade na aprendizagem, sendo que o material está acessível através de seus dispositivos móveis, permitindo-lhes aprender como e quando for necessário, não importando onde estejam, mesmo que em movimento. Aos educadores é fornecido um novo meio de disponibilização do material pedagógico, como também um novo meio de interação com o aluno (SIAU;

HOONAH, 2008).

O uso de tecnologias digitais pelo aluno, como ferramenta de apoio pedagógico para construção de novos conhecimentos, deve favorecer os níveis de interação entre o aprendiz e a máquina, propiciando-se situações onde o aprendiz estabeleça uma via de mão dupla, e o mesmo exerça ciclos de ação, reflexão e depuração (VALENTE, 2003).

Com um mercado em ampla expansão e que conta com milhões de consumidores, o desenvolvimento de jogos para celulares já é um negócio altamente rentável e que tende a ser ainda mais nos próximos anos (COGOI; SANGIORGI; SHAHIN, 2006).

> Um grupo crescente de pesquisa indica que as ferramentas das tecnologias móveis podem ser eficazes para alunos na era digital e há sinais de forte motivação e possíveis ganhos de aprendizagem através de jogos jogados em dispositivos móveis por público jovem e adulto. (COGOI; SANGIORGI; SHAHIN, 2006).

O mercado de jogos para dispositivos móveis é uma área importante para o crescimento da indústria de jogos. Este mercado está previsto para crescer rapidamente com a convergência das tecnologias e aplicações móveis cada vez menos restringidas por limitações dos dispositivos. As últimas gerações de dispositivos apresentam altas definições de cores, vídeo, memória reforçada e muitas novas funcionalidades que estão fazendo os aplicativos móveis cada vez mais atraentes e com um menor custo de desenvolvimento se comparado aos aplicativos para as plataformas tradicionais (COGOI; SANGIORGI; SHAHIN, 2006).

Diante destes cenários, a utilização em sala de aula do dispositivo móvel como ferramenta de auxílio pedagógico tornaria as aulas mais dinâmicas e os materiais didáticos seriam compartilhados com os alunos em tempo real. Não seria necessário o aluno recorrer em casa a um computador, pois a utilização do próprio celular, artefato cada vez mais acessível junto às pessoas de diversas classes sociais, poderia ser feita para acessar os conteúdos de aula, resolver os problemas e construir conhecimentos. Além disso, a interatividade entre o aluno e o dispositivo móvel pode ser facilitada pela familiaridade que se apresenta em sua utilização, sobretudo no caso do telefone celular (MARÇAL et al., 2008).

2.2 *FRAMEWORK* FURBOT

Conforme citado anteriormente, a dificuldade percebida no ensino de programação de computadores levou ao desenvolvimento do Furbot. O projeto vem sendo desenvolvido desde

o primeiro semestre de 2008.

O elemento central do Furbot é a implementação do método denominado `inteligencia`. É neste método que os alunos desenvolvem a lógica de programação de um personagem. Os comandos de movimentação do personagem são expressos em Java, mas remetem o aluno a utilizar nomes em português, tais como `andarAcima`, `ehVazio` e `diga` (VAHLDICK; MATTOS, 2009). O Quadro 1 exemplifica a programação da inteligência do Furbot.

```
import br.furb.furbot.Furbot;

public class ExemploFurbot extends Furbot {

     public void inteligencia() {

        while(!ehFim(DIREITA))
        //desloca o furbot até o limite direito do mundo
        {

             if(!ehVazio(DIREITA))
             //se não houver obstáculo a direita
             {
                 andarDireita();//desloca o furbot para a próxima tela

             }else{

                 //tratar aqui a situação em que existe um obstáculo na
                 //direção do furbot
             }
        }

}
```

Fonte: Vahldick e Mattos (2009).

Quadro 1 – Exemplo de programação da inteligência do Furbot

O *framework* Furbot permite a criação de ambientes de jogos 2D. O Furbot foi concebido e construído com a linguagem Java, baseado no padrão *Model View Control* (MVC), tornando o código flexível à adaptação de outros tipos de interfaces gráficas.

Com o resultado da criação do Furbot pode-se destacar como principais características:

a) facilidade de implementação do código utilizando um fluxo sequencial simples, onde o aluno pode construir passo a passo a lógica de seu personagem simplesmente utilizando da estrutura facilitadora que o Furbot mantém para suportar as funções destes personagens;
b) a codificação é feita independente da *Integrated Development Environment* (IDE) Java, por meio de um arquivo no formato *Java ARchive* (JAR)[3] que pode ser importado para a IDE de programação Java de escolha do aluno;
c) as atividades a serem desenvolvidas pelos alunos são definidas pelo professor,

[3] JAR é um formato de arquivo utilizado por aplicações desenvolvidas na linguagem Java.

através de um arquivo *eXtensible Markup Language (*XML), que contém informações como dimensões do mundo e os elementos que compõem o cenário do jogo.

No Quadro 2 é apresentado a especificação de um problema no formato XML onde o mundo contém 8 linhas por 8 colunas e estabelece que o robô inicia na posição 0 (zero) para a coordenada X e 0 (zero) para a coordenada Y.

```
<furbot>

   <enunciado>
      Exercício 1. &lt;br&gt;
        Faça o robô andar até a ultima posição da linha.
        &lt;br&gt;
        -Lembre-se de que as coordenadas sempre serão
        fornecidas como (x,y), &lt;br&gt;
        - A primeira coluna e linhas possuem valor ZERO.
   </enunciado>

   <mundo>
                  <qtdadeLin>8 </qtdadeLin>
                  <qtdadeCol>8 </qtdadeCol>
                  <explodir>true</explodir>
   </mundo>

   <robo>
                   <x>0</x>
                   <y>0</y>
   </robo>

</furbot>
```

Fonte: Vahldick e Mattos (2008).

Quadro 2 – Exemplo de mundo em arquivo XML

Após a criação e especificação do arquivo XML (Quadro 2), o aluno deve incluir um método `main` na aplicação, identificando o arquivo XML que será utilizado (Quadro 3).

```
public static void main (String args[])
{

   MundoVisual.iniciar("ExemploFurbot.xml");

}
```

Fonte: Vahldick e Mattos (2008).

Quadro 3 – Carga do arquivo XML

Ao executar o método *`main`*, é apresentada ao aluno uma janela (Figura 1), contendo os elementos definidos no arquivo XML, além dos botões `Run`, `New`, `Stop` e um componente *Slider* para estabelecer a velocidade de execução.

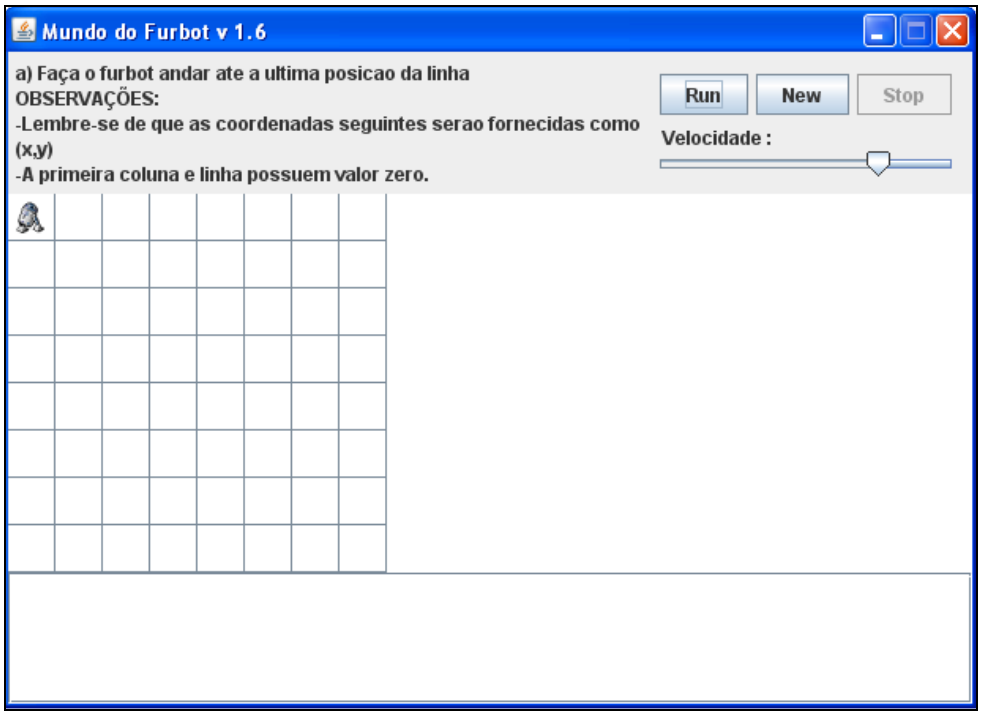


Figura 1 – Execução do Furbot

Ao clicar no botão `Run`, o Furbot executa o método inteligência. O botão `New` permite gerar uma nova disposição dos componentes já que existe a possibilidade de configuração de posicionamento aleatório no arquivo XML e o botão `Stop` encerra a execução.

2.2.1 A arquitetura do *Framework*

Na sequência são exibidos os principais pacotes de classes do Furbot com finalidade de mostrar a estrutura interna utilizada como base para este trabalho.

O principal pacote do *framework* é o pacote `br.furb.furbot`. A estrutura deste pacote é apresentada na Figura 2.

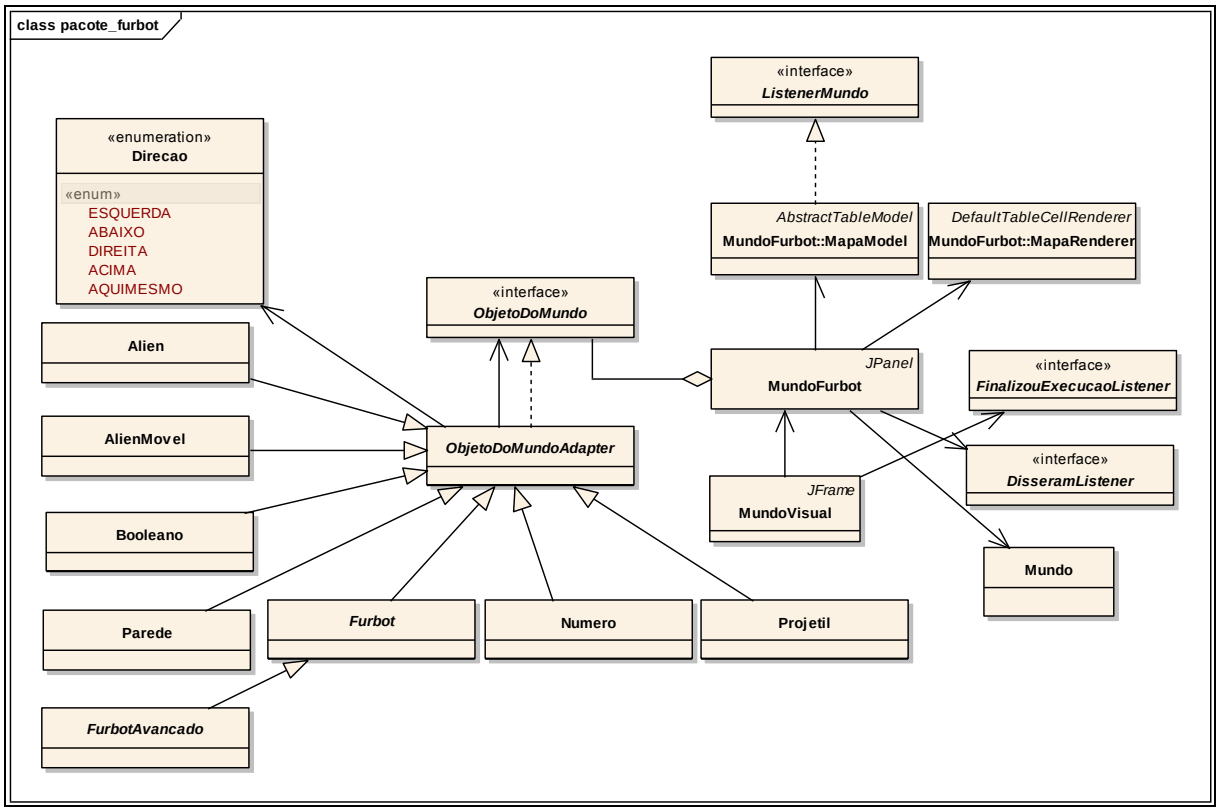


Figura 2 – Diagrama de classes do pacote `br.furb.furbot`

Neste pacote estão definidas as principais classes do *framework*, tais como:

a) `MundoFurbot`, classe que faz a interação do mapa modelo do mundo Furbot, com os componentes de interface gráfica da `swing`;

b) `MundoVisual`, classe responsável por inicializar uma instância de `MundoFurbot` através da chamada do método `iniciar`, recebendo os parâmetros dos exercícios definidos pelo arquivo XML;

c) `ObjetoDoMundo`, interface base para criação de qualquer objeto do mundo Furbot. Esta interface é implementada pela classe `ObjetoDoMundoAdapter` onde são definidos todos os atributos que são em comum para cada objeto.

Outro pacote de classes do Furbot é o pacote `br.furb.furbot.suporte` o qual agrupa classes para criar, manipular e desenhar os objetos do mundo Furbot. Este pacote pode ser visualizado na Figura 3.

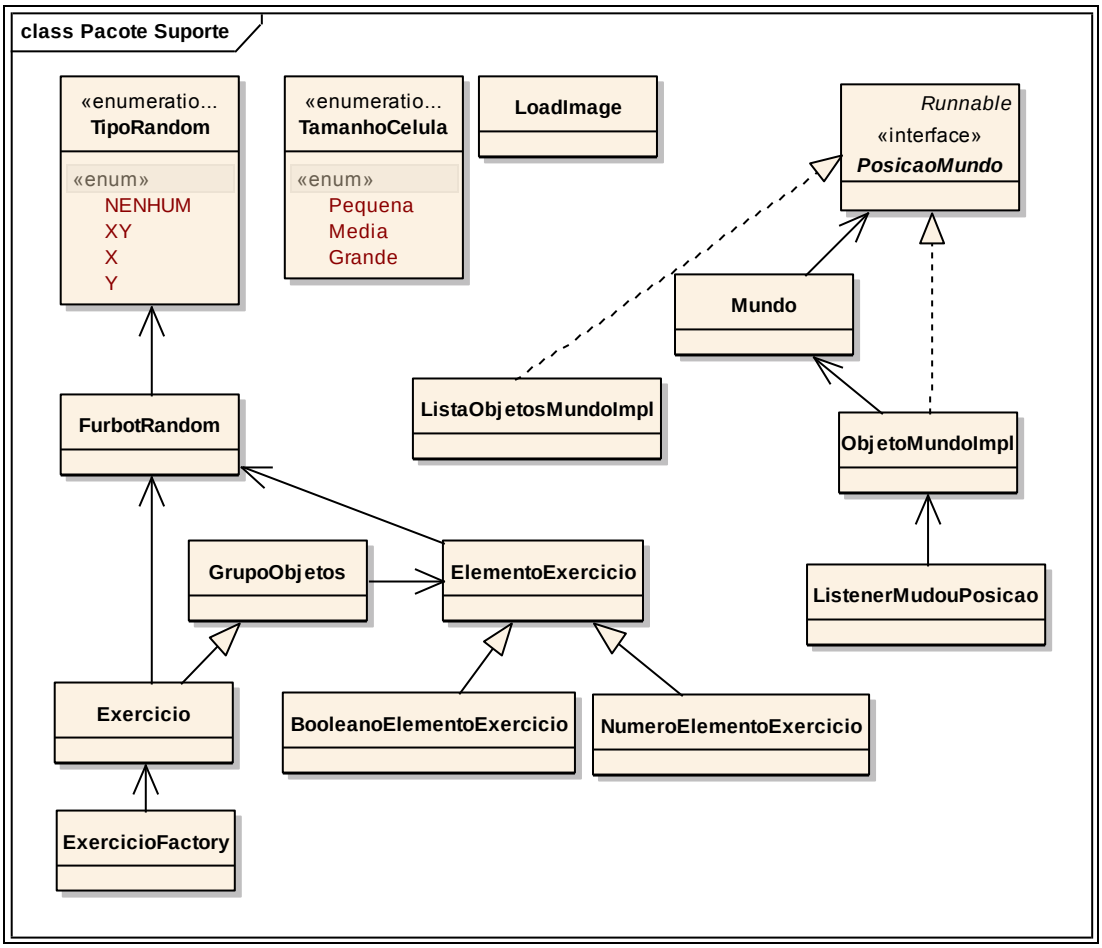


Figura 3 – Diagrama de classes do pacote `br.furb.furbot.suporte`

Neste pacote destacam-se as seguintes classes de suporte:

a) `ExercicioFactory`, faz a criação de um exercício modelado em XML para inicializar o `MundoFurbot`, quer irá fazer a criação do `Mundo` e dos seus elementos;

b) `Exercicio`, guarda informações de enunciado, tamanho de mundo e a posição onde os objetos do mundo serão criados inicialmente;

c) `Mundo`, mantém o status de cada posição do mundo e os objetos que estão nela. É nesta classe também onde são feitas alterações nas posições dos objetos ao se movimentarem pelo mapa modelo.

Outro pacote encontrado no *framework* é o pacote `br.furb.furbot.exceptions`, o qual mantém as classes tratadoras de exceções que podem ocorrer durante a execução do mundo Furbot. Este pacote pode ser exemplificado na Figura 4.

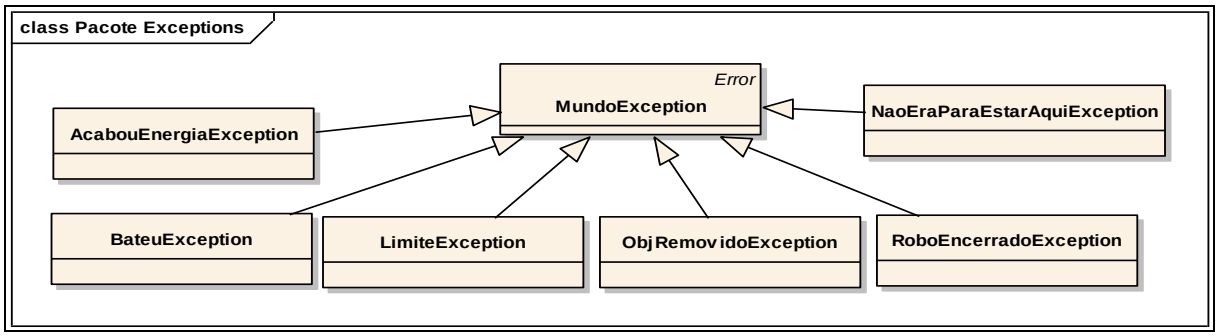


Figura 4 – Diagrama de classes do pacote `br.furb.furbot.exceptions`

A interligação entre os pacotes e as classes do Furbot pode ser visualizada no diagrama de classes completo apresentado na Figura 5. A seguir será explicado brevemente como funciona o relacionamento de classes do Furbot.

A classe `MundoVisual` inicializa a interface gráfica, um objeto `MundoFurbot` e também controla a finalização da execução do Furbot através da criação de um objeto do tipo `FinalizouExecucaoListener`.

O `MundoFurbot` é também o responsável por criar o `grid` diante das propriedades do objeto `Mundo` criado a partir da classe `MapaModel` a qual mantém e atualiza as informações do mapa do mundo.

A classe `Mundo` possui um `array` bidimensional de objetos de `PosicaoMundo` que é uma interface para acessar informações de uma posição no tabuleiro, esta interface é implementada pela classe `ObjetoMundoImpl` que faz o controle das ações dos objetos dentro do `Mundo`. Todos os elementos de jogo do Furbot herdam da classe `ObjetoDoMundoAdapter` a qual possui uma instancia de `ObjetoMundoImpl` utilizada para que este objeto tenha acesso a funções de interação com a classe `Mundo`.

Os elementos disponíveis para criação de jogos são: `Alien`, `Booleano`, `Numero`, `Projetil`, `Parede`, `Furbot` e `FurbotAvancado`. Todos esses elementos utilizam da classe `LoadImage` para carregamento de imagens. Eles também possuem o acesso ao tipo enumeração `Direcao` para tratarem internamente o sentido em que estão.

A qualquer momento as ações que os objetos fazem perante o jogo podem criar exceções. Estas exceções podem ser geradas por qualquer classe do framework através da classe `MundoException` a qual será responsável por emitir uma mensagem de erro dependendo de qual foi o lugar e a forma de seu lançamento.

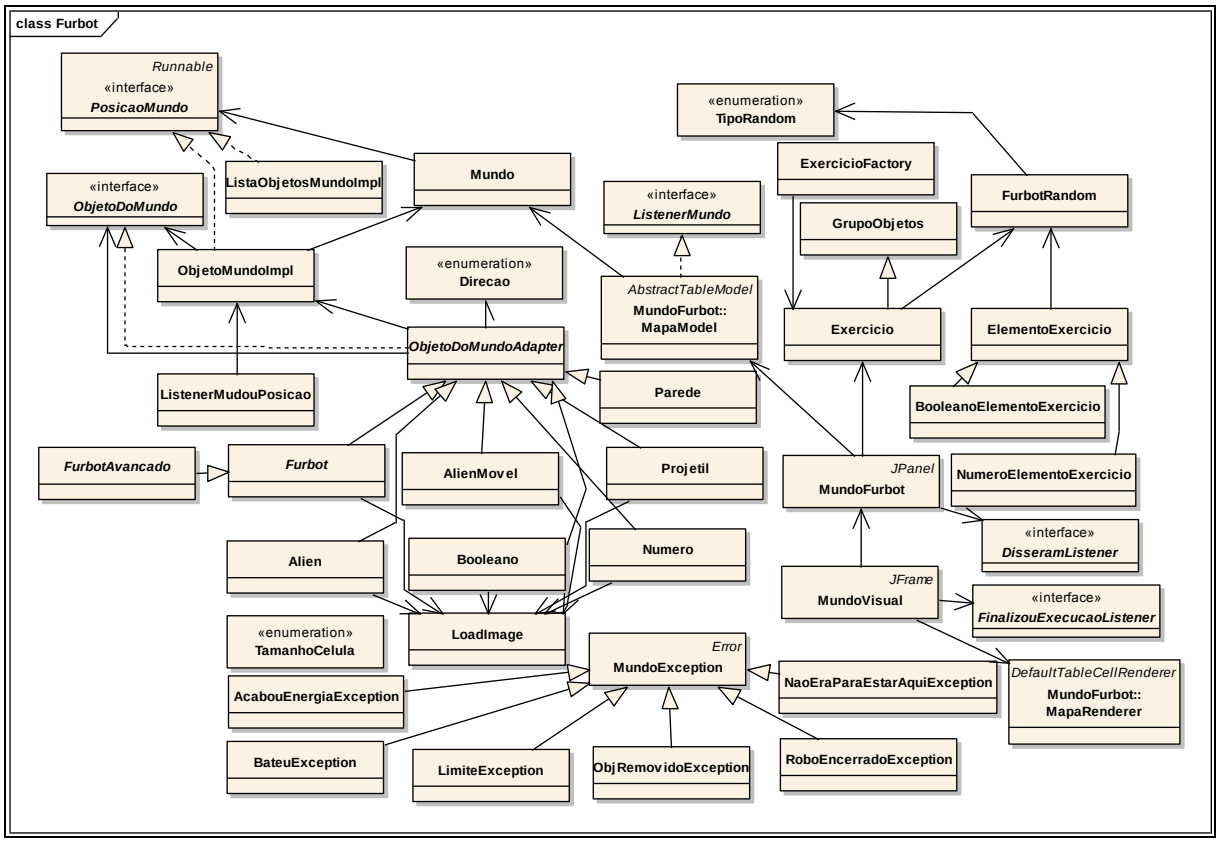


Figura 5 – Diagrama completo das classes do Furbot

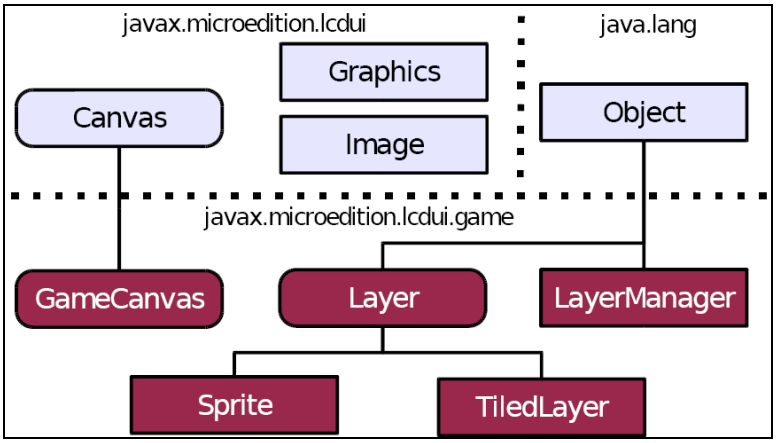
2.3 JME PARA JOGOS

> Originalmente a tecnologia Java ME foi criada com objetivo para trabalhar com restrições associadas com a construção de aplicações para dispositivos de pequeno porte. Por este propósito a Sun definiu as bases para a tecnologia Java ME se enquadrar em ambientes limitados e possibilitar a criação de aplicações Java rodando em dispositivos de pequeno porte com memória limitada, vídeo e pouco poder de processamento. (SUN MICROSYSTEMS, 2008, tradução nossa).

O desenvolvimento de aplicações JME na área de jogos também vem ganhando cada vez mais espaço no mercado de jogos e para suprir e adicionar novas necessidades dos desenvolvedores existe o *Mobile Information Device Profile* (MIDP) desenvolvido pela Sun MicroSystems.

O MIDP consiste em um conjunto de componentes do JME voltados para a criação de aplicações em terminais menos avançados como celulares e na sua versão 2.0 apresenta um novo pacote nomeado `Game` que é exclusivamente orientado a facilitar o desenvolvimento de jogos (OGLIARI, 2008). Este pacote adicional facilita o desenvolvimento de jogos, podendo

ajudar o programador através do uso de algumas classes melhoradas em relação à versão anterior 1.0 do MIDP. Para desenvolver o aspecto visual de mundo em um jogo, existe a classe `GameCanvas`, que fornece funções melhoradas da classe `Canvas`, possibilitando trabalhar com mundos virtuais em jogos. Para trabalhar com os objetos visuais no `canvas` existe a classe `Sprite`, que ajuda no trabalho de criação e manipulação de uma ou mais imagens que farão parte dos objetos do jogo. Na Figura 6 é mostrada a estrutura do pacote *`Game`* incluído no MIDP 2.0.



Fonte: Ogliari (2008).

Figura 6 – Estrutura do pacote game do MIDP 2.0

A popularidade do JME possibilitou que uma grande quantidade de fabricantes criassem classes customizadas para suportar o desenvolvimento interno. Isso acabou por gerar grandes problemas de portabilidade entre as tecnologias de celulares existentes. Este também é um dos motivos principais do MIDP 2.0 ter sido criado (LAM, 2006, p. 34).

2.4 TRABALHOS CORRELATOS

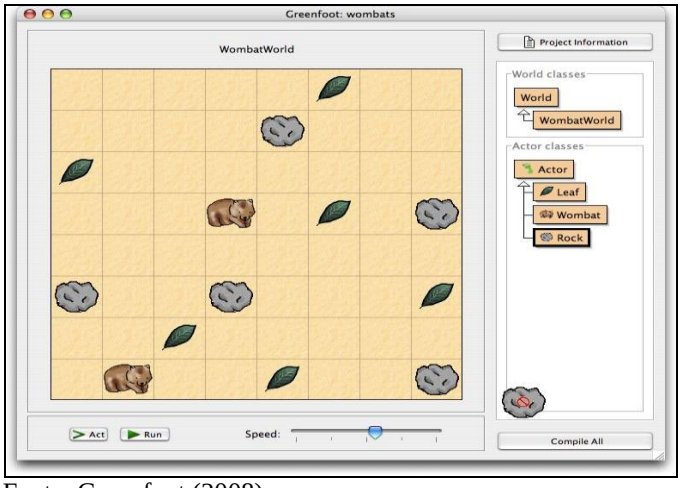
Existem aplicações que possuem características semelhantes ao proposto neste trabalho, tais como o Greenfoot (GREENFOOT, 2008), Robocode (ROBOCODE HOME, 2008) e a plataforma MGBL (MGBL, 2008).

2.4.1 Greenfoot

O Greenfoot é uma combinação entre um *framework* de criação de mundos 2D em Java e um ambiente de desenvolvimento que integra ferramentas como visualizador de classes, editor, compilador e execução. Este *framework* pode ser usado com abordagem de jogos que possam ser visualizados e construídos em mundos 2D, como em jogos populares de tabuleiro (GREENFOOT, 2008).

> O aluno desenvolve duas classes: o mundo e o ator. O mundo é uma matriz bidimensional onde podem ser distribuídos vários objetos (chamados de atores). Os atores são os elementos centrais nesse ambiente. O foco nos exercícios é a implementação dos atores. Um ator pode se mover pelo mundo, adicionar e remover outros atores, e "enxergar" o mundo. Uma imagem precisa ser atribuída a um ator, ela pode ser trocada durante o tempo de vida do ator. (VAHLDICK; MATTOS, 2009).

A Figura 7 ilustra a interface gráfica do Greenfoot e alguns objetos do mundo. A grande área quadriculada representa o mundo. Ao lado direito estão relacionadas as classes deste mundo e atores, abaixo do mundo estão os controles de execução do mundo.



Fonte: Greenfoot (2008).

Figura 7 – Ambiente do Greenfoot

2.4.2 Robocode

O Robocode é uma ferramenta de ensino de linguagens de programação, desenvolvida pela AlphaWorks, onde é possível a criação de robôs programáveis na linguagem Java. Os

objetivos do Robocode são proporcionar uma ferramenta que facilita o aprendizado do paradigma orientado a objetos com foco motivacional em relação à diversão por meio de um jogo de batalhas entre robôs que possam ter seus códigos modificados e melhorados para obter melhores resultados dentro de um campo de batalha (AGUIAR, 2007).

No Robocode só existe um tipo de unidade, os robôs, onde cada instância é uma *thread*[4] Java que possui os métodos específicos que formam a inteligência deste objeto. A partir do momento em que a batalha começa, as unidades são disparadas fazendo com que cada robô tenha sua lógica de programação colocada em ação (ROBOCODE HOME, 2008). A Figura 8 mostra o ambiente do Robocode, com o campo de batalha e alguns robôs.

Fonte: Robocode Home (2008).

Figura 8 – Ambiente do Robocode

2.4.3 *Mobile Game-Based Learning* (MGBL)

A MGBL é uma plataforma que permite o desenvolvimento de jogos de auxílio na aprendizagem em telefones celulares com as tecnologias JME e MIDP. O foco principal da MGBL leva em consideração a criação de jogos onde sejam envolvidas questões de escolha de decisões em situações críticas da vida real. Estas decisões são transformadas para o mundo virtual de um *game* partindo de estudos e pesquisas a um grupo de pessoas de uma determinada idade, que procuram desenvolver conhecimentos e habilidades em assuntos escolares e acadêmicos levantados por professores (MGBL, 2008).

Existem três tipos de jogos que podem ser criados a partir da utilização dos recursos da MGBL (MGBL, 2008):

a) jogos de questionários, onde são designadas perguntas ao aluno que deve escolher uma ou mais respostas válidas em um determinado tempo visando adquirir uma melhor pontuação com objetivo de atingir maiores níveis;
b) jogos de aventura em ambientes 2D, em que o aluno pode andar pelo mundo livremente e interagir com objetos ao seu redor. Quando um objeto é tocado, eventos são disparados para fornecer conteúdos multimídia ou questionários;
c) jogos de interação através de troca de mensagens: este tipo de jogo propõe ao aluno mensagens com objetivos ou questões definidas pelo professor. As respostas a estas mensagens são analisadas pela plataforma MGBL, que incrementa ou não a pontuação ao estudante.

As construções destes jogos são feitas através de uma interface *on-line*. Esta interface é acessada pelos professores, que indicam temas, perguntas e atividades para o tipo de jogo escolhido a um grupo específico de pessoas. A plataforma MGBL faz a criação destes jogos com a especificação *on-line* fornecida pelo professor (MGBL, 2008).

A Figura 9 mostra um jogo de questionário com controle de tempo e níveis. O jogo foi desenvolvido na plataforma MGBL e está rodando em um simulador JME Wireless Toolkit.

Fonte: MGBL (2008).

Figura 9 – Jogo exemplo de questionários

[4] *Threads* são estruturas de execução pertencentes a um processo e assim compartilham os segmentos de código e dados e os recursos alocados ao sistema operacional pelo processo. *Threads* também podem ser criadas pelas linguagens de programação para divisão de linhas de execução de uma aplicação (SAUVÉ, 2000).

3 DESENVOLVIMENTO

Este capítulo tem por objetivo detalhar os principais requisitos que a versão deve atender, assim como descrever os passos e os resultados obtidos com a implementação.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O ambiente proposto deverá:

a) possibilitar a criação de jogos bidimensionais com forma de tabuleiro ou *grid*[5], para celulares, com uma estrutura similar ao Furbot original (Requisito Funcional - RF);
b) viabilizar a alimentação das classes pelo desenvolvimento de métodos para execução das ações dos elementos do jogo;
c) ter como meio de edição de propriedades do mundo, a utilização de leitura de arquivos XML (Requisito Não-Funcional - RNF);
d) permitir execução do ambiente proposto em dispositivos móveis (RNF);
e) implementar controle de objetos e mundo através de *threads* (RNF);
f) ser implementada utilizando a tecnologia JME (RNF);
g) executar em dispositivos que utilizem máquina virtual Java, perfil MIDP 2.0 (RNF).

3.2 ESPECIFICAÇÃO

A finalidade desta seção é especificar o *framework* Mobile Furbot desenvolvido através de imagens e diagramas definidos pela *Unified Modeling Language* (UML). A ferramenta Enterprise Architect foi utilizada no desenvolvimento destes diagramas. Com intuito de explicar o funcionamento do *framework*, os diagramas escolhidos são os de classes

[5] *Grid* é um sistema de coordenadas projetado sobre uma superfície plana, onde são delimitadas zonas quadradas para medir e destacar posições e o seu conteúdo (SULCOM, 2007).

e de casos de uso.

3.2.1 Diagrama de Classes

Nesta seção apresenta-se o diagrama de classes completo do *framework* Furbot com as classes adicionadas para possibilitar a criação da versão Mobile Furbot (Figura 10).

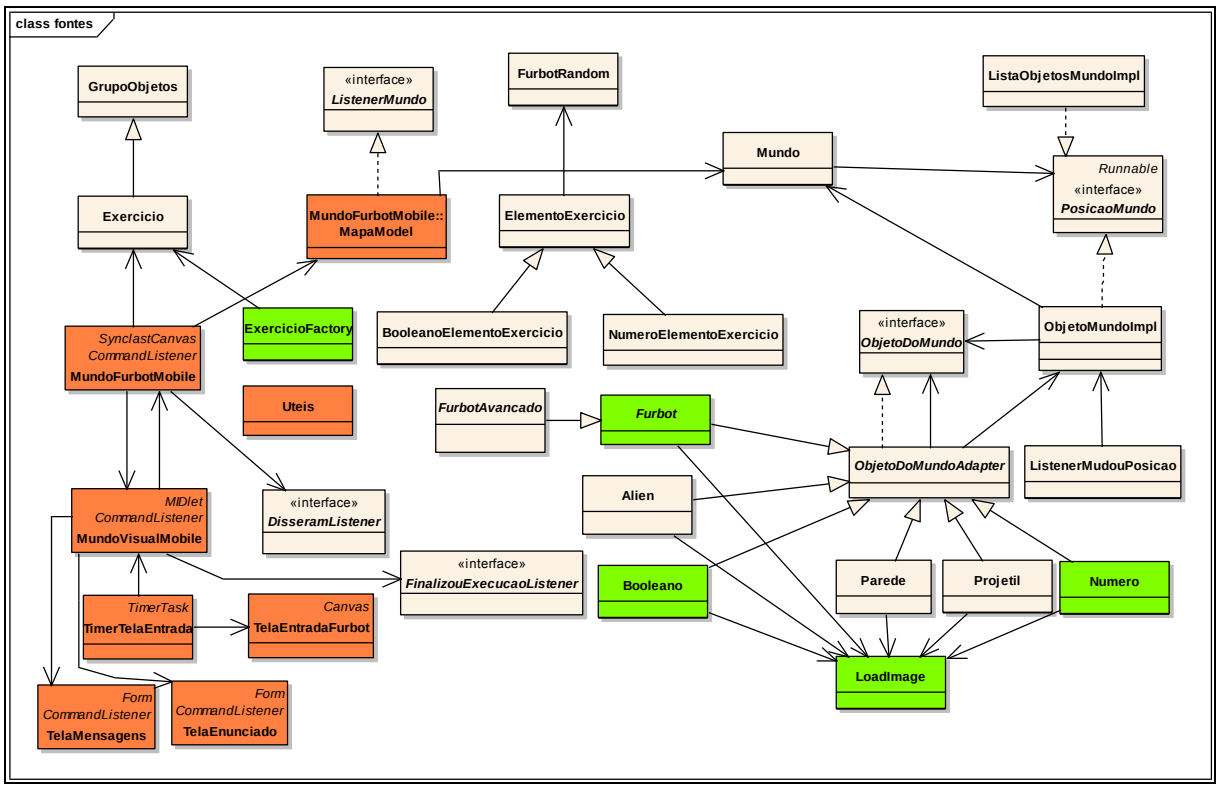


Figura 10 – Diagrama de classes do Mobile Furbot

Todas as classes sofreram modificações por utilizarem de recursos específicos da plataforma utilizada e também pela utilização de bibliotecas não compatíveis a JSE. As classes que se encontram na cor bege são as classes que já existiam no *framework* Furbot e sofreram adaptações para serem convertidas para a plataforma JME. As classes em verde são as classes que sofreram modificações pela utilização de bibliotecas específicas para desenvolvimento em aplicativos móveis. As classes em laranja são as classes novas criadas para o *framework* Mobile Furbot.

As classes `Furbot`, `Numero`, `Booleano` e `LoadImage`, fazem a utilização de métodos de desenho e edição de imagens e estas foram adaptadas para a plataforma JME. A classe `ExercicioFactory` sofreu reconstrução para portar leitura XML com a biblioteca `KXML`.

Para viabilizar a criação deste trabalho, foram criadas as classes descritas em vermelho na Figura 10. A seguir apresenta-se uma breve explicação destas classes:

a) `MundoVisualMobile`, é a classe principal que é um `MIDlet` e mantém todos os comandos para iniciar um `MundoFurbotMobile` através do exercício criado a partir da leitura de um arquivo XML;
b) `MundoFurbotMobile`, é a classe responsável por criar um mundo Furbot, inicializar e atualizar os componentes gráficos do `canvas` para desenho do `grid` e também por controlar as chamadas de ações possíveis a serem utilizadas;
c) `MundoFurbotMobile::MapaModel`, responsável por fazer a interface de atualização entre o modelo do mundo Furbot e os componentes gráficos do `canvas`. Esta classe está localizada dentro da classe `MundoFurbotMobile`;
d) `TelaEntradaFurbot`, faz a inicialização do *framework* Mobile Furbot. Esta classe exibe uma imagem de abertura e dá inicio a um `timer` que ao finalizar chama o menu principal controlado pela classe `MundoFurbotMobile`;
e) as classes `TelaEnunciado` e `TelaMensagens` são formulários para entrada das informações do exercício e das mensagens lançadas pelos objetos do mundo Furbot;
f) a classe `Uteis` foi criada para manter constantes e suprir alguns métodos tais como: `split` para quebrar `Strings` e `reescalaArray` para redimensionar uma imagem via seu `array` de *pixels*.

3.2.2 Diagramas de casos de uso

O diagrama de casos de uso apresentado na Figura 11, tem por finalidade especificar quais as funcionalidades e ações o aluno pode realizar para solucionar e modelar um problema proposto em forma de exercício por um professor. A seguir serão explanados os casos de uso “Formular exercício” no Quadro 4, “Programar e executar o Mobile Furbot” no Quadro 5 e por fim o caso de uso “Testar em ambiente compatível com JME” no Quadro 6.

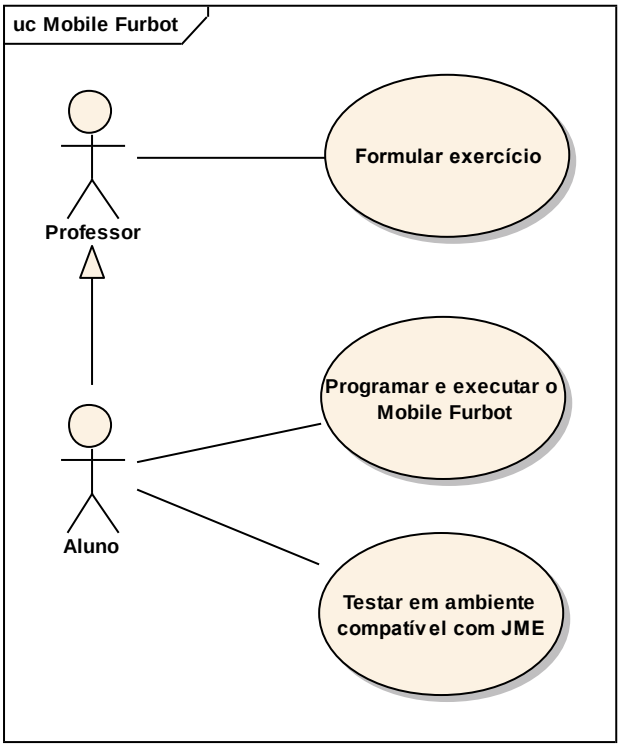


Figura 11 – Diagrama de casos de uso

UC01* - *Formular exercício

UseCase: O professor de programação define o tema e objetivo do Mobile Furbot através de um arquivo XML.

Scenarios

Define tags do xml {Principal}.

1. O professor digita o objetivo do enunciado na tag <enunciado>
2. O professor define as propriedades de mundo na tag <mundo>
3. O professor inicializa o mundo com alguns obstáculos iniciais.
4. O professor disponibiliza o arquivo XML aos alunos.

Inicializa mundo vazio {Alternativo}.

No passo 3, caso o professor não tenha inicializado nenhum elemento de obstáculo, o professor pode optar para que o aluno gere dinamicamente seus próprios obstáculos.

Informa path inválido das classes {Exceção}.

No passo 3, caso o professor tenha digitado incorretamente o caminho das classes dos objetos, o framework exibirá erro ao tentar criar o elemento da classe referenciada erroneamente.

Quadro 4 – Casos de Uso `Formular exercício`

UC02 - Programar e executar o Mobile Furbot

UseCase:

Constraints

- *Pre-condition.* Ter o arquivo XML do exercício proposto.
- *Pre-condition.* Ter as classes do framework devidamente importadas.
- *Post-condition.* O aluno pode compilar as classes dos requisitos do exercício.

Scenarios

Constrói lógica do exercício no framework {Principal}.

1. O aluno recebe o XML com o enunciado do exercício.
2. O aluno cria a MIDlet para iniciar o MundoFurbot.
3. O aluno cria as classes que irão implementar a inteligência do Furbot em um ambiente de programação desktop.
4. O aluno executa o Mobile Furbot para ver o campo inicial proposto pelo professor e o seu enunciado.
5. O framework apresenta o grid e as opções de menu para desenvolvimento do exercício.
6. O aluno formula e refina a inteligência de seus elementos criados.
7. O aluno chega a solução para resolver o objetivo do enunciado.
8. O framework para a execução dos elementos.

Finaliza sem alcançar a solução {Alternativo}.

No passo 7, o aluno encontra-se com situação de insucesso na solução , então este volta ao passo 3 tentando uma nova forma de programação da suas classes.

Falha ao criar os objetos do XML {Exceção}.

No passo 4, o aluno tenta executar um arquivo de XML onde o framework não consegue achar as classes indicadas. Neste caso o arquivo XML recebe alterações nas referências das classes e tenta novamente a execução.

Quadro 5 – Casos de Uso `Programar e executar o Mobile Furbot`

UC03 - Testar em ambiente compatível com JME

UseCase:

Constraints

- *Pre-condition.* Ter elementos do mundo programados.
- *Pre-condition.* Ter as classes do Mobile Furbot.
- *Pre-condition.* Ter as bibliotecas de leitura XML.
- *Post-condition.* Exercício executado e solucionado.

Scenarios

Controla as ações no mundo Furbot {Principal}.

1. O aluno exporta as classes para rodar em um dispositivo móvel.
2. O aluno com suas classes implementadas visualiza a tela do *grid* do mundo Furbot em um ambiente suportado pela plataforma JME.
3. O framework inicia os objetos do mundo chamando seus métodos de inteligência.
4. O aluno visualiza mensagens expedidas pelos objetos do mundo.
5. O aluno aumenta e diminui a velocidade de execução dos objetos do mundo via teclado de celular.
6. O sistema altera a velocidade da thread do mundo.
7. O aluno para a execução manualmente pelo teclado ou aguarda finalização da execução os objetos.

Quadro 6 – Casos de Uso `Testar em ambiente compatível com JME`

3.3 IMPLEMENTAÇÃO

Esta seção apresenta primeiramente as técnicas e ferramentas utilizadas na programação do Mobile Furbot. Em seguida é mostrado o modo como o usuário pode interagir com o *framework*.

3.3.1 Técnicas e ferramentas utilizadas

Para viabilizar a implementação do *framework* Mobile Furbot foi utilizado o ambiente de programação Eclipse Pulsar que é uma versão do Eclipse Galileu específica para desenvolvimento para dispositivos móveis. Juntamente com este ambiente foi utilizado o simulador da JME Wireless ToolKit 2.5.2. Para testes em ambiente móvel real, foi utilizado um celular Nokia N95 com suporte a JME e MIDP 2.0.

3.3.1.1 Biblioteca KXML

Com intuito de trabalhar com leitura de arquivos XML foi utilizada a biblioteca `KXML` que facilitou na leitura das `tags` de exercícios em XML do Mobile Furbot. Nesta biblioteca cada `tag` é interpretada separadamente para inicializar as propriedades de cada exercício definido em um arquivo XML.

```
private void parseMundoTag(KXmlParser parser, Exercicio exercicio)
{
    if(parser.getName().equals(TAG_QTDADELIN))
        exercicio.setQtdadeLin(Integer.parseInt(parser.nextText().trim()));
    if(parser.getName().equals(TAG_QTDADECOL))
        exercicio.setQtdadeCol(Integer.parseInt(parser.nextText().trim()));
    if(parser.getName().equals(TAG_TAMANHOCEL))
        exercicio.setTamanhoCel(parser.nextText().trim());
```

```
<mundo>
<qtdadeLin>12</qtdadeLin>
<qtdadeCol>11</qtdadeCol>
<tamanhoCel>P</tamanhoCel>
<explodir>true</explodir>
</mundo>
```

Figura 12 – Exemplo de leitura com a biblioteca XML

A Figura 12 mostra um exemplo de como os elementos da `tag` mundo são lidos através do *`parser`* da `KXML`. As setas mostram que para testar qual a `tag` está sendo lida, basta utilizar o método `getName` da classe `KxmlParser`. Assim é possível atribuir os valores das propriedades de mundo indicadas no exercício.

3.3.1.2 Biblioteca `Synclast`

Para viabilizar a implementação do desenho do *grid* no mundo Mobile Furbot foi utilizada a biblioteca `Synclast` para desenho de interfaces gráficas em aplicativos desenvolvidos para JME.

Esta biblioteca facilita a criação e posicionamento de elementos visuais através de uma organização baseada em *layouts* similar aos do pacote *`swing`* encontrado nas versões do *Java Standard Edition* (JSE). No Mobile Furbot foi utilizado um elemento de *layout* em forma de tabela para desenhar os quadrados que compõem o tabuleiro do mundo Furbot e logo abaixo deste foi utilizado um elemento de texto para exibir a última mensagem disparada por um objeto do Furbot.

Um exemplo destes componentes pode ser observado na Figura 13, onde foi criado um mundo de 11 colunas por 12 linhas com um tamanho de célula pequeno e logo abaixo ao *grid* está a última mensagem que foi gerada no mundo Furbot, indicando a posição em que o elemento Furbot está no tabuleiro.

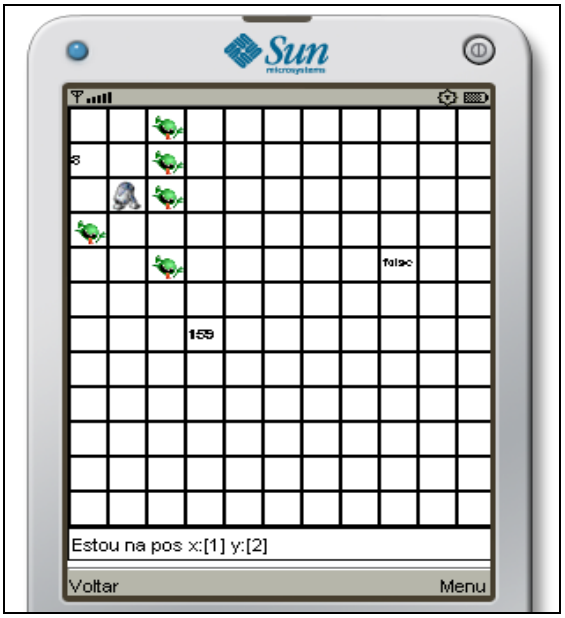


Figura 13 – *Grid* desenhado com a biblioteca `Synclast`

3.3.2 Análise Comparativa

Para que haja uma melhor compreensão das modificações feitas em relação ao Furbot *desktop* e o Mobile Furbot, nesta seção será apresentada uma análise comparativa focada em diversos recursos e componentes alterados para permitir a criação desta versão para dispositivos móveis em JME.

3.3.2.1 Versão do Java

As diferenças entre as plataformas Java utilizadas levaram a consideráveis mudanças em todas as classes originais do Furbot, já que muitos dos recursos encontrados na plataforma JSE do Java não são suportados pela JME. Alguns destes recursos ainda não foram construídos na versão *Micro Edition* do Java, para favorecer ganho de desempenho assim como, um melhor aproveitamento de recursos de memória e processamento disponíveis em dispositivos móveis.

O Furbot em sua versão 1.7 utilizando a plataforma JSE como meio de programação, este utiliza de muitos recursos auxiliares como: Tipo `Enumeração`, `Generics`, `ArrayList`, `Foreach` e outras classes de comum uso, existentes nos pacotes *Standard* do Java.

No Mobile Furbot utilizando a plataforma JME não foi possível utilizar estes recursos auxiliares, porém como alternativas os tipos enumeração transformaram-se em constantes de inteiros pré-definidos, `ArrayList` foram substituídos por objetos `Vector` e as estruturas de `Generics` e `Foreach` foram substituídas pelas estruturas clássicas de programação como: Iterações através de construções `for` simples com `casts` para acessar e armazenar os tipos reais dos `Objects`.

3.3.2.2 Generics e estruturas de dados auxiliares.

No Furbot os tipos `List` e `ArrayList` são utilizadas para criação de uma lista dinâmica juntamente com a definição `Generics` para indicar que nesta lista somente irão ser contemplados objetos do tipo que foi declarado para a lista. Um exemplo de código fonte pode ser visto no Quadro 7.

```
private List<DisseramListener> disseramListeners = new ArrayList<DisseramListener>();
```

Quadro 7 – Exemplo de lista dinâmica no Furbot

No Mobile Furbot toda lista dinâmica é criada através da utilização da classe `Vector`, porém não é possível definir um tipo `Generic` para utilização dentro do `Vector`, possibilitando a ocorrência de erros de `casts` incorretos durante a execução. Um exemplo de código fonte pode ser visto no Quadro 8.

```
private Vector disseramListeners = new Vector();
```

Quadro 8 – Exemplo de lista dinâmica no Mobile Furbot

3.3.2.3 Leitura de XML

Um dos requisitos não funcionais do Furbot desde a sua primeira versão é a definição de exercícios através de um arquivo XML. Para isso foram utilizadas bibliotecas externas tanto na versão original quanto na versão *mobile*.

O Furbot utiliza a biblioteca `Digester` 1.8. Esta biblioteca faz a inicialização de objetos do exercício mediante a definição de regras para cada `tag` ou propriedades a serem lidas em um arquivo XML.

No Quadro 9 da linha 7 até a 11 está um exemplo de como são definidas as regras referentes as `tags` do mundo no Furbot.

```
1  Digester d = new Digester();
2  exercicio = new Exercicio(nomeArquivoXML);
3  d.push(exercicio);
4
5  d.addSetProperties("furbot");
6  d.addBeanPropertySetter("*/enunciado", "enunciado");
7  d.addBeanPropertySetter("*/mundo/qtdadeLin");
8  d.addBeanPropertySetter("*/mundo/qtdadeCol");
9  d.addBeanPropertySetter("*/mundo/explodir");
10 d.addBeanPropertySetter("*/mundo/usarLinhasNaGrade");
11 d.addBeanPropertySetter("*/mundo/tamanhoCel");
```

Quadro 9 – Leitura com a biblioteca `Digester`

Assim quando o arquivo XML é lido, a classe `Digester` chama os métodos `setters` da classe `Exercicio` para atribuir valores às propriedades do mundo.

O *framework* Mobile Furbot utiliza a biblioteca externa `KXML`, que trabalha com leitura

de uma `tag` por vez. Durante o `parsing` sempre é solicitada a próxima parte que deve ser processada até chegar ao fim. Este processo é feito dentro de um *loop* principal o qual termina na última `tag </furbot>`. Esta biblioteca é projetada para ambientes restritos, como os definidos para JME.

Um exemplo de leitura utilizando a `KXML` pode ser observado no Quadro 10.

```
1  InputStream entrada = getClass().getResourceAsStream("/"+nomeArquivoXML);
2  InputStreamReader inptStream = new InputStreamReader(entrada);
3  KXmlParser parser = new KXmlParser();
4
5  parser.setInput(inptStream);
6
7  //Ciclo principal , até o final do arquivo
8  do
9  {
10
11    int pullParser = parser.next();
12
13    switch(pullParser)
14    {
15        //se encontrar uma tag de abertura no XML cai aqui o parsing
16        case XmlPullParser.START_TAG:
17        {
18
19            //trata tag com o enunciado
20            parseEnunciadoTag(parser,exercicio);
21
22            //trata tags com informações de mundo
23            parseMundoTag(parser,exercicio);
24
25            break;
26         }
27
28         //sai do arquivo para de parsear
29         case XmlPullParser.END_DOCUMENT:
30         {
31            fimArquivo = true;
32            break;
33         }
34
35    }//switch
36 }//do
37 while(!fimArquivo);
```

Quadro 10 – Leitura com a biblioteca `KXML`

3.3.2.4 Manipulação de Imagens

A manipulação de imagens no Furbot foi estabelecida através da centralização de uma classe responsável por carregar os arquivos de imagem e cada objeto do mundo deve implementar um método `buildImage` que pode ou não fazer alterações nas imagens carregadas. Nos parágrafos abaixo serão demonstrados os exemplos de criação e manipulação destas imagens no Furbot e no Mobile Furbot.

No Furbot original, uma imagem é carregada através de uma criação de um

`ImageIcon`. Este `ImageIcon` é do pacote `javax.swing`, e ele aceita imagens nos formatos GIF, JPG e PNG, que podem ser editadas e alteradas via manipulações geométricas através da classe `Graphics2D` do pacote `java.awt`.

Um exemplo de construção e manipulação de imagens é apresentado no Quadro 11. Este cria uma instância de um `ImageIcon` (linha 2). Em seguida é criada uma nova imagem com modificações através da classe `Graphics2D` (linha 10 até a linha 27). Por fim a imagem gerada é enviada ao retorno da função `buildImage`, que é a responsável por criar as imagens de cada objeto no mundo Furbot.

```
1  public ImageIcon buildImage() {
2  ImageIcon image = LoadImage.getInstance().getIcon("imagens/r2d2-icon.gif");
3
4   if (ehDependenteEnergia())
5   {
6     //cria uma imagem em branco de 50x50
7     BufferedImage img = new BufferedImage(50, 50, 1);
8
9     //cria uma instancia de Graphics2D para permitir desenhar dentro da imagem
10    Graphics2D g = img.createGraphics();
11    g.setColor(Color.white);
12    g.fillRect(0, 0, 50, 50);
13
14    //preenche com a imagem do robo
15    g.drawImage(image.getImage(), 2, 2, null);
16
17    //calcula a porcentagem restante de energia
18    float fator = getEnergia()/(getMaxEnergia() * 1.0F);
19    int altura = Math.round(40F * fator);
20
21    //cria status da energia em vermelho
22    g.setColor(Color.red);
23    g.fillRoundRect(40, 45 - altura, 5, altura, 5, 5);
24
25    //cria status da energia em preto
26    g.setColor(Color.black);
27    g.drawRoundRect(40, 5, 5, 40, 5, 5);
28
29    //cria uma Instancia da imagem alterada e retorna para a função buildImage
30    image = new ImageIcon(img);
31  }
32 return image;
33 }
```

Quadro 11 – Criação de imagens no Furbot

No Mobile Furbot, da mesma forma é necessário implementar o método `buildImage`, porém a classe para manipulação de imagem é a classe `Image` do pacote `javax.microedition.lcdui`. Esta inicialmente não pode ser alterada a menos que se crie uma imagem mutável através do no método `getIconMuttable` criado para o Mobile Furbot na classe `LoadImage`. Este processo é demonstrado no Quadro 12.

```
1   public Image buildImage() {
2      //cria a imagem a partir do arquivo
3      Image image = LoadImage.getInstance().getIcon("r2d2-icon-mob.gif");
4
5      //cria uma imagem em branco para alteração
6      Image energiaRobo = LoadImage.getInstance().getIconMuttable();
7
8      if (ehDependenteEnergia())
9      {
10       //pega o acesso para alteração através do método getGraphics
11       Graphics g = energiaRobo.getGraphics();
12
13       //trabalha a imagem para mostrar a barra de energia restante
14       g.setColor(Color.WHITE);
15       g.fillRect(0, 0, 20, 20);
16
17       g.drawImage(image, 2, 2, 0);
18
19       float fator = (float) getEnergia()/ ((float) getMaxEnergia() * 1);
20       float altura = 20F * fator;
21
22       g.setColor(Color.RED);
23
24       g.fillRoundRect(20,(int) (25 - altura), 5, (int)altura, 5, 5);
25                     g.setColor(Color.BLACK);
26                     g.drawRoundRect(20, 5, 5, 20, 5, 5);
27                     image = energiaRobo;
28     }
29
30     return image;
31  }
```

Quadro 12 – Criação de imagens no Mobile Furbot

3.3.2.5 Acesso aos recursos e arquivos

Também se observou necessário efetuar alterações nos diretórios onde os recursos de arquivos são lidos, pelas diferenças de acesso a arquivos entre as plataformas JSE e JME.

No Furbot os arquivos de imagens do *framework* Furbot ficam no diretório `imagens` e são acessados através do método `getIcon` da classe `LoadImage`, já os arquivos XML são lidos do diretório raiz onde está o aplicativo Furbot.

No Mobile Furbot todos os recursos são lidos a partir do diretório `res`, um padrão estabelecido por aplicativos desenvolvidos para a plataforma JME. A classe `LoadImage` do Mobile Furbot irá buscar as imagens com o método `getIcon`.

3.3.2.6 Objetos do arquivo XML

Todas as `tags` originais do Furbot foram mantidas na versão Mobile Furbot, porém

como foi preciso criar uma estrutura de diretórios de classes totalmente separada do Furbot original, esta nova estrutura afetou na identificação da localidade das classes a serem utilizadas pelos objetos definidos nos arquivos XML.

No Furbot as classes dos objetos ficam localizadas dentro pacote `br.furb.furbot` e a criação desse objetos via arquivo XML por ser observada na linha 1 do Quadro 13.

```
1  <objeto class="br.furb.furbot.Alien">
2  <id>4</id>
3  <!-- posição inicial fixa do alien -->
4  <x>2</x>
5  <y>2</y>
6  </objeto>
```

Quadro 13 – Referência aos objetos no XML do Furbot

As classes dos objetos no Mobile Furbot ficam localizadas dentro pacote `br.furb.furbot.mobile.objetos` e a criação desse objetos via arquivo XML por ser observada na linha 1 do Quadro 14.

```
1  <objeto class="br.furb.furbot.mobile.objetos.Alien">
2  <x>2</x>
3  <y>2</y>
4  </objeto>
```

Quadro 14 – Referência aos objetos no XML do Mobile Furbot

Como observado no Quadro 14, foi criado um novo diretório `objetos` para separar as classes dos objetos básicos originais do Furbot.

Já que não foi possível manter os antigos diretórios, pelas inúmeras modificações das classes, resolveu-se separar estas classes para facilitar a implementação e a legibilidade desta versão.

3.3.2.7 Estrutura de resolução do exercício

As chamadas para os métodos disponíveis para resolver os exercícios foram mantidas visando padronizar as classes, o que torna possível a utilização das mesmas classes desenvolvidas para a versão *desktop* no Mobile Furbot, porem estas classes devem utilizar somente de recursos que estejam disponíveis para ambas as plataformas JSE e JME.

O aluno deve utilizar dos mesmos comandos do Furbot original sem precisar se preocupar em alterações a modo de *framework*, somente com arquitetura da plataforma utilizada.

Um exemplo de classe desenvolvida para a versão Furbot contendo elementos

específicos da plataforma JSE pode ser visto no Quadro 15.

```
public void inteligencia() throws Exception {
   setTempoEspera(0);

   int nascAlien = 0;

   Direcao ultDirecao = DIREITA;
   while (vivo) {

     int tecla = getUltimaTeclaPress();

     // os inimigos nascem a cada 1 milhao de vezes em que passa por aqui
     if (nascAlien == 1000000) {  // TODO altere 1000000 para valores menores
         nascerAlien();
         nascAlien = 0; // reinicia a contagem
     }

     nascAlien++;

     switch (tecla) {

        case TECLACIMA:
          ultDirecao  = andar(ACIMA);
          break;

       case TECLABAIXO:
          ultDirecao = andar(ABAIXO);
          break;
```

Quadro 15 – Método inteligência da classe do exercício na versão Furbot

A mesma classe desenvolvida apresentada no Quadro 15 pode ser vista na nova versão compatível para dispositivos móveis no Quadro 16.

```
public void inteligencia() throws Exception {
   setTempoEspera(0);

   int nascAlien = 0;

   int ultDirecao = DIREITA;
   while (vivo) {

     int tecla = getUltimaTeclaPress();
     // os inimigos nascem a cada 1 milhao de vezes em que passa por aqui
     if (nascAlien == 2000) {  // TODO altere 1000000 para valores menores
         nascerAlien();
         nascAlien = 0; // reinicia a contagem
     }

     nascAlien++;

     switch (tecla) {

        case TECLACIMA:
          ultDirecao  = andar(ACIMA);
          break;

       case TECLABAIXO:
          ultDirecao = andar(ABAIXO);
          break;
```

Quadro 16 – Método inteligência da classe do exercício na versão Mobile Furbot

A única diferença entre as classes é que na apresentada no Quadro 16 está no tipo enumeração `Direcao` foi trocado por uma variável `int`.

3.3.2.8 Iniciar classe do exercício

O método `iniciar` é o responsável por começar um mundo novo no Furbot, este é chamado através da classe que irá rodar a parte visual do Furbot.

No mundo Furbot para começar um novo mundo basta implementar um método `main` que será o responsável por ativar a classe `MundoVisual`. Este processo pode ser visto no Quadro 17.

```
public static void main(String[] args) {
   MundoVisual.iniciar("Interacao.xml");
}
```

Quadro 17 – Iniciar um novo mundo visual no Furbot

No mundo do Mobile Furbot, o usuário precisa criar uma classe que irá herdar a `MIDlet MundoVisualMobile` , em seguida chamar o construtor pai passando o arquivo XML desejado e a classe que contém a resolução do exercício. Este processo pode ser visto no Quadro 18.

```
public class InteracaoJMEStarter extends MundoVisualMobile{

      public InteracaoJMEStarter() {
            super(Interacao.class, "Interacao.xml");
      }

}
```

Quadro 18 – Iniciar um novo mundo visual no Mobile Furbot

A `MIDlet` citada acima é a responsável por inicializar o ambiente do dispositivo móvel ou emulador no qual o aplicativo Mobile Furbot irá rodar.

3.3.2.9 Tamanho das células no *grid*

Com a finalidade de obter um melhor aproveitamento na tela os tamanhos das células foram alterados e diminuídos com relação a versão original do Furbot.

O Furbot possui quatro tamanhos de células diferentes: “P” com células de 32 por 32

pixels, “M” com células de 40 por 40 pixels, “A” com células de 20 por 20 pixels e “B” com células de 10 por 10 pixels.

O Mobile Furbot Possui três tamanhos de células diferentes: “P” com 20 por 20 pixels, “M” com 32 por 32 pixels e “G” com 37 por 37 pixels.

3.3.2.10 Restrições de resolução de tela

A restrição de tela encontrada em dispositivos móveis é muito maior do que em um dispositivo *desktop*, isso criou a necessidade de eliminar da tela principal muitos dos elementos visuais originais do Furbot, como a área de log de mensagens, área de exibição do enunciado e a área das ações de controle do Furbot. Estas áreas foram transferidas para um sub-menu deixando espaço na tela principal para alocar o tabuleiro de forma integral.

As restrições de tela do Furbot dependem da resolução do sistema operacional em que está rodando, permitindo muitas células no tabuleiro.

Já para o Mobile Furbot existe uma grande restrição devida falta de espaço em tela, o que levou a análise de uma melhor combinação de linhas e colunas dependendo do tamanho da célula do mundo Furbot Mobile.

Utilizando o emulador *DefaultColorPhone* do *Wireless ToolKit* (WTK) versão 2.5.2, foi identificado que para construir um tabuleiro com células tamanho “P”, o ideal é que o tabuleiro tenha 12 por 11 colunas, para o tamanho “M”, o ideal é 8 linhas por 7 colunas e para o tamanho “G” o ideal é 7 linhas por 7 colunas. Estes tamanhos podem ser observados na Figura 14.

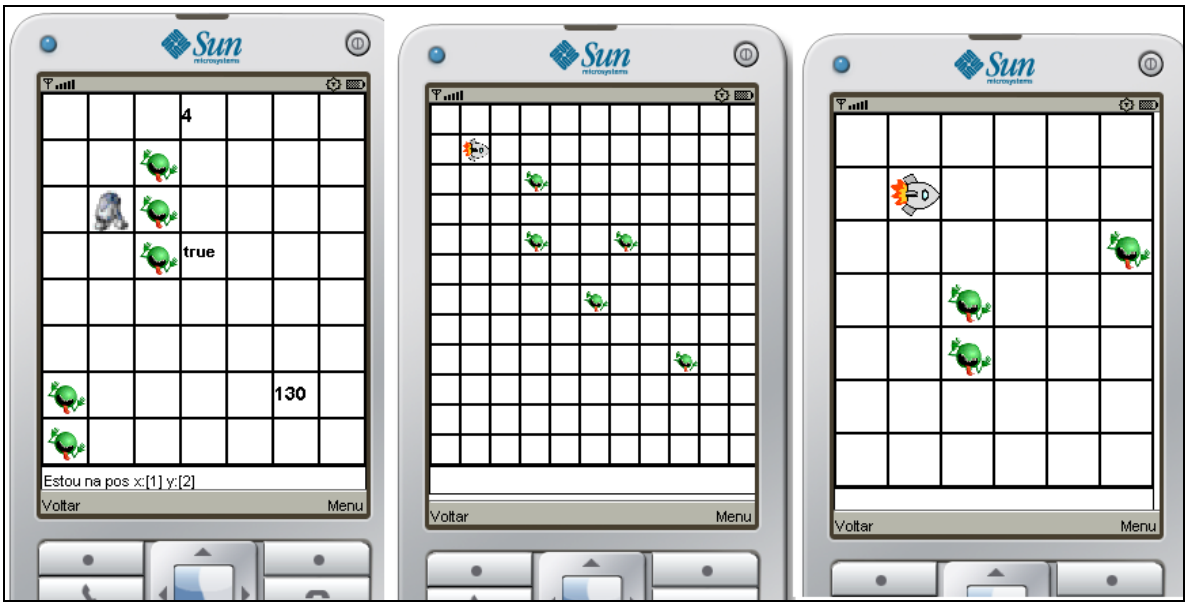


Figura 14 – Melhor aproveitamento dos tamanhos de células

Também é possível melhorar um pouco o espaço do mundo se a tag `usarLinhasNaGrade` estiver habilitada nas propriedades do mundo do XML. Um exemplo de mundo com tamanho de células "P" sem linhas na grade pode ser visto na Figura 15 com um ganho de algumas colunas já que os `pixels` ocupados pelo *grid* foram eliminados.

A configuração de linhas e colunas do arquivo XML pode variar dependendo do tamanho do dispositivo utilizado para testes, porém cabe ao utilizador do *framework* definir qual o tamanho desejado do mundo para o seu aplicativo.

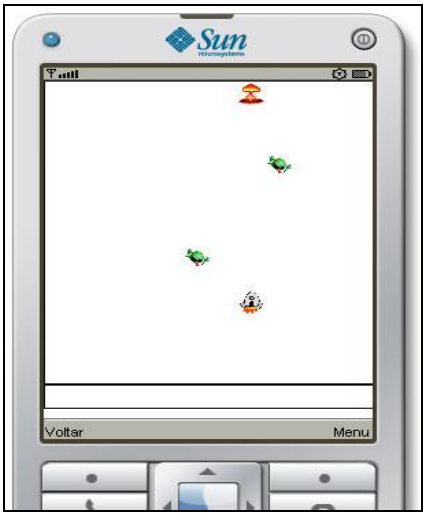


Figura 15 – Exemplo de mundo sem o *grid*

3.3.2.11 Biblioteca de interface com o usuário

Nesta seção serão apresentadas as diferenças entre os componentes visuais da camada

de interface com o usuário em ambos os *frameworks*.

A interface do Furbot é constituída de um `JFrame` implementado na classe `MundoVisual`, onde são criados os componentes `JPanel`, `JButton`, `JSlider`, `JTable`, `JTextArea` e `JLabels`. Todos estes componentes estão definidos dentro da classe `MundoVisual` e ambos são provenientes do pacote `swing` do JSE. A Figura 16 mostra a tela principal do Furbot e seus componentes.

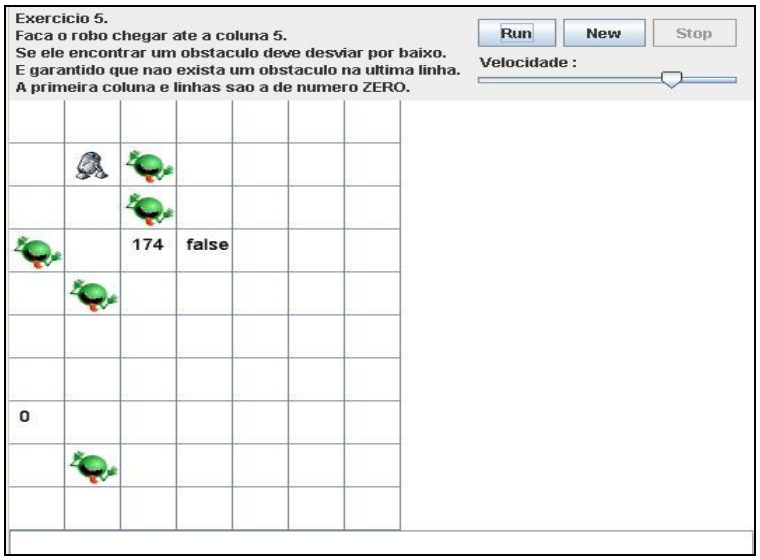


Figura 16 – Furbot e a interface `swing`

Já no caso do Mobile Furbot, inicialmente é apresentado um `Form` com a imagem de apresentação do *framework*. E em seguida é aberto outro `Form` para que o usuário acesse o jogo através de opções de escolha com o componente `List`. Na primeira opção do `List` o usuário tem acesso ao exercício que será carregado mediante ao XML do exercício indicado no método iniciar da classe `MundoVisualMobile`.

Ao entrar no mundo Furbot Mobile, componentes `TableContainer` e `Input` da biblioteca `Synclast` são carregados para desenhar o tabuleiro do mundo e mostrar a área da ultima mensagem disparada. Estes componentes são inicializados na classe `MundoFurbotMobile`, que contém o `canvas` para interação com o usuário.

3.3.2.12 Tratamento de eventos da interface gráfica

No Furbot os eventos são tratados através de *listeners* que ficam observando eventos de ações aos componentes visuais. Todos os métodos tratadores de *listeners* são definidos durante a inicialização da classe `MundoVisual`. O Quadro 19 mostra a inicialização do tratador do *listener* responsável por reiniciar o mundo Furbot.

```
jbRenovar.addActionListener(
             new ActionListener() {
                  public void actionPerformed(ActionEvent arg0)
```

Quadro 19 – Tratamento de ações no Furbot

No Mobile Furbot os eventos são tratados a partir de um único método `commandAction`, cada `Form` ou `Canvas` com botões disponíveis têm este método devidamente implementado. Este método identifica o comando de interface acionado pelo usuário e cada comando é tratado separadamente. O Quadro 20 mostra o método `commandAction` da classe `MundoFurbotMobile`.

```
/** Comandos disponíveis no grid do mobile furbot */
public void commandAction(Command c, Displayable d) {
   if (c == comandoVoltar) {
      this.baseStarterMidlet.mostrarMenuPrincipal();
   } else if (c == comandoPausar) {
      mapaModel.stopRobo();
   } else if (c == comandoEnunciado) {
      new TelaEnunciado(baseStarterMidlet.getTelaFurbot(),this ,
exercicio);
   } else if (c == comandoReiniciar){
      this.parar();
      new TelaEnunciado(baseStarterMidlet.getTelaFurbot(),this ,
exercicio);
      try {
         this.setExercicio(exercicio);
         this.reiniciar();
         this.setTempoEspera(velocidadeAtual);
      } catch (Exception e) {
         e.printStackTrace();
      }
}
```

Quadro 20 – Tratamento de comandos no Mobile Furbot

3.3.2.13 Exibição do Enunciado, Área de Mensagens e Controle de Velocidade

No Furbot todas as funcionalidades de visualização de enunciados, área de mensagens e controle de velocidades, podem ser encontradas na tela principal, e todos eles são carregados e alimentados por meio da classe `MundoVisual`.

No Mobile Furbot, estas funcionalidades tiveram de ser removidas da tela principal para dar espaço ao tabuleiro, por este motivo elas agora se encontram no menu secundário a direita do *grid* do mundo. A Figura 17 apresenta a tela com o menu e um exemplo de como é exibida a tela da área do enunciado proposto no XML.

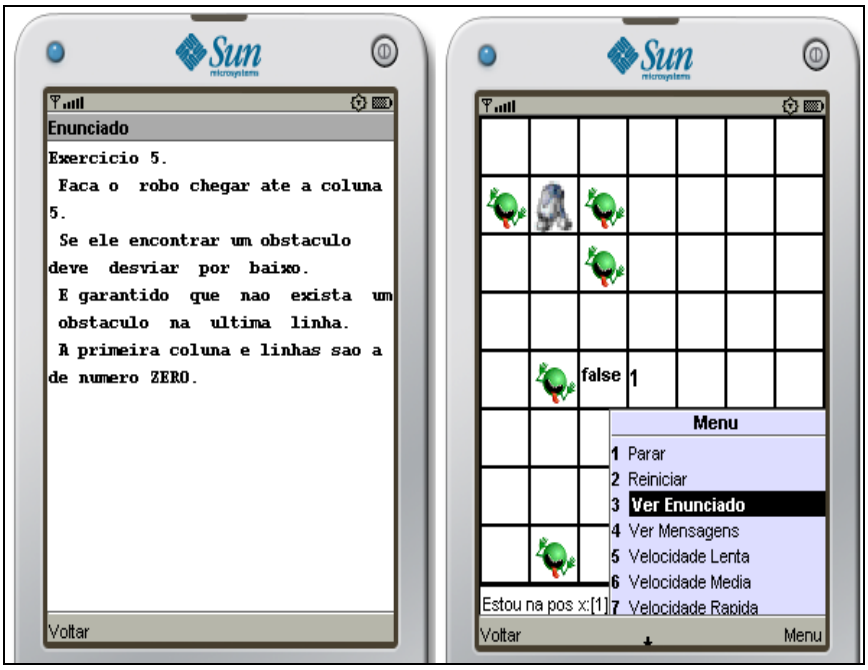


Figura 17 – Enunciado e menu secundário no Mobile Furbot

Com intuito de manter um `log` de mensagens lançadas pelos objetos do mundo, foi criado neste menu secundário a opção número 4 para exibir um `Form` com todas as mensagens apresentadas desde o início do mundo Furbot Mobile.

Como alternativa a falta de um componente `JSlider`, foram implementados 3 novas opções no menu que fazem a mudança entre 3 velocidades predefinidas, estas são: velocidade lenta, média e rápida. Ambas as opções utilizam o método `setTempoEspera` que altera o tempo em milissegundos de espera entre o próximo ciclo das *Threads* dos objetos do mundo.

3.3.2.14 Forma de atualização do mundo

No Furbot a atualização é feita através de um `JTable` que é ligado a um `MapaModel`. Neste `MapaModel` são chamadas as funções `fireTableCellUpdated` e `fireTableDataChanged` que sinalizam que algum elemento foi alterado para então poder re-desenhar o `JTable` com o conteúdo do mapa no mundo Furbot.

Já no Mobile Furbot, foi mantido o mesmo padrão adotado pelo Furbot, criando um método chamado `atualizaContainer` que recebe a instância do `MapaModel`, que quando ativado cria novamente todo o `canvas Synclast`, tendo como base o estado atual do mapa e dos objetos deste mapa no mundo do Mobile Furbot.

A descrição do método `atualizaContainer` pode ser vista no Quadro 21.

```
/**
 * Cria um container para desenhar ou redesenhar todo o mundo furbot
 *
 * @param modelo
 *            - modelo do mundo furbot
 */
public void atualizaContainer(MapaModel modelo) {
      furbotContainer = null;
      System.gc();

      // atualizar a grid
      furbotContainer = new FlowContainer(Graphics.TOP);

      // crio a tabela com o tabuleiro e a área de texto
      criaAmbienteFurbot(modelo, furbotContainer);

      // atribuir o estilo do canvas
      setStyleSheet(local);

      // redefine o container do furbot para redesenhar
      setContainer(furbotContainer);

      // atualizar este canvas
      container.repaint();
}
```

Quadro 21 – Desenho e atualização do `canvas Synclast`

Outro método relevante na atualização do mundo no Mobile Furbot é o método `criaAmbienteFurbot`. É nele em que as posições do modelo do mundo são lidas para encontrar objetos ativos ou inativos. As posições no mundo são iteradas e testadas para que caso for encontrado um objeto do mundo válido, será exibida a sua imagem no tabuleiro, ou caso contrário o bloco da posição do tabuleiro será apagado.

3.3.2.15 Interação com o teclado

Nesta seção serão apresentadas as diferenças de tratamento e definição de teclas a serem utilizadas no mundo Furbot.

O Furbot utiliza o tratamento através de `KeyAdapters` que são criados para tratar o evento `KeyPressed`. Depois de identificado um evento de tecla pressionada, o método `pressionadaTecla` da classe `Mundo` é chamado passando o código da tecla.

As constantes que definem as ações do mundo podem ser mostradas no Quadro 22.

```
      public static final int TECLACIMA = 38;
      public static final int TECLABAIXO = 40;
      public static final int TECLAESQUERDA = 37;
      public static final int TECLADIREITA = 39;
      public static final int TECLAESPACO = 32;
```

Quadro 22 – Constantes no Furbot

Já para a programação no Mobile Furbot foi utilizado do método `keyPressed` incluído no próprio `canvas` JME. Este método chama igualmente o método do `Mundo` `pressionaTecla`, porém os códigos das constantes foram alterados para valores correspondentes das setas direcionais em teclado de celulares (Quadro 23).

```
public static final int TECLACIMA = 50;
public static final int TECLABAIXO = 56;
public static final int TECLAESQUERDA = 52;
public static final int TECLADIREITA = 54;
public static final int TECLAESPACO = 53;
```

Quadro 23 – Constantes no Mobile Furbot

3.3.2.16 Adaptações nas imagens utilizadas

A seguir são apresentadas as imagens básicas utilizadas no Furbot. Estas tiveram de sofrer alterações em seu tamanho original para a versão Mobile Furbot.

As imagens base para criação de um aplicativo Furbot podem ser vistas na Figura 18. Os tamanhos delas são relativamente grandes, porém adequados a plataforma *desktop* a que se aplica.

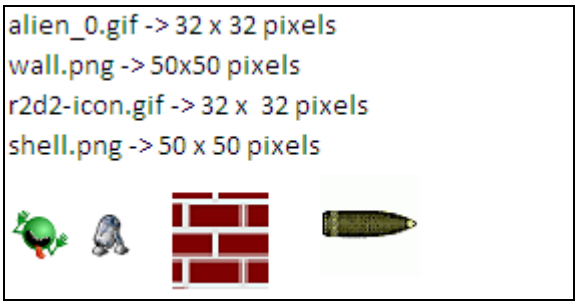


Figura 18 – Tamanho de imagens do Furbot

No Mobile Furbot visando possibilitar a inclusão destas imagens sem grandes distorções, foi necessário editar as imagens diminuindo-as em alguns `pixels` favorecendo a visualização em tabuleiros com células tamanho “P”. A Figura 19 mostra as imagens modificadas e seus tamanhos.

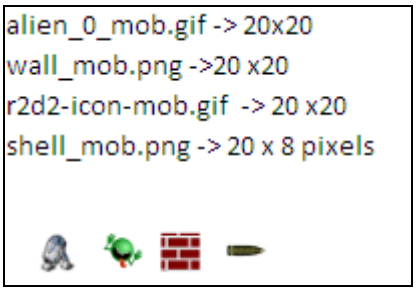


Figura 19 – Tamanho de imagens do Mobile Furbot

3.3.2.17 Funcionalidade

Para destacar a funcionalidade do sistema foi utilizado um jogo criado pelo professor Adilson Vahldick. Este jogo foi apresentado no evento Interação Furb 2009. Neste jogo é apresentado um ambiente com uma nave espacial se encontra no meio de um ataque massivo de inimigos *aliens* com movimentação aleatória.

Os objetos *aliens* são criados aleatoriamente para tentar cercar a nave do Furbot, muitos inimigos são criados e cada objeto do tipo `Inimigo` faz testes para verificar quem está na posição em que ele está andando. Quando encontrado o objeto nave ele aciona o método matar e elimina o personagem principal.

Sobre o personagem principal, ele pode lançar tiros nas 4 direções disponíveis no Furbot, cada tiro possui em sua `Thread` um método de inteligência que fica em loop até atingir um objeto Inimigo ou sair do cenário.

Este jogo foi criado no *framework* Furbot utilizando de recursos únicos do JSE, tais como: `ArrayList` e `Foreach`. O jogo rodando em uma plataforma *desktop* através da JSE apresentou normalidade em questões de desempenho.

Já para permitir o porte do jogo no Mobile Furbot estes recursos específicos utilizados da JSE foram alterados. `ArrayList` foram substituídos por `Vector` e as iterações via `Foreach` foram substituídas por construções `For` nativas da linguagem Java. Na Figura 20 apresenta-se o jogo inicializado nas versões *mobile* e *desktop* do Furbot.

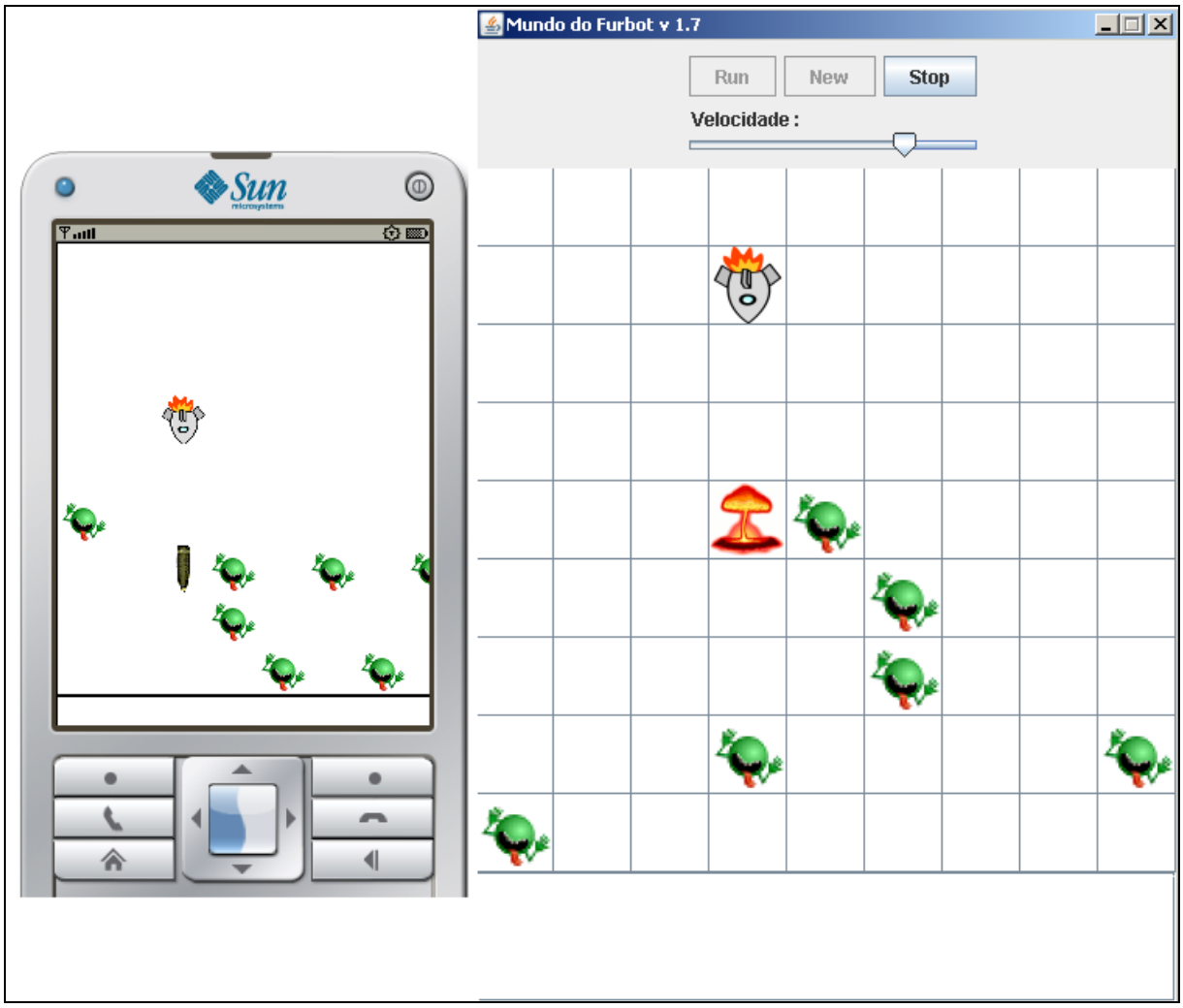


Figura 20 – Jogo apresentado no evento Interação FURB 2009

O desempenho do jogo no Mobile Furbot diminuiu à medida que novos objetos foram sendo criados aleatoriamente e o poder de processamento das múltiplas `threads` foi decrescendo até fazer com que o jogo ficasse muito lento.

A quantidade de objetos criados por um jogo mais complexo fez com que o *framework* inicialize muitas `threads` de elementos no mundo, fazendo com que o número de controles, testes, criações de objetos, chamadas de funções e iterações fossem duplicadas a cada novo elemento criado. Assim fazendo com que nesta situação o ambiente de dispositivo móvel, mais restrito de processamento e memória, apresenta-se uma queda em questões de desempenho e tempo de resposta as ações do usuário, por não conseguir atender a todas as requisições internas feitas pelo *framework* e as funcionalidades criadas para o jogo.

3.3.3 Operacionalidade da implementação

Para executar o *framework* Mobile Furbot é necessário ter o pacote de classes do *framework*, `mobileFurbot1.jar`, assim como os arquivos `kxml2-2.3.0.jar` e `synclast-ui.jar`. Também é necessário possuir um ambiente para testes com suporte a plataforma JME com MIDP 2.0. Este ambiente poderá ser um simulador ou um dispositivo móvel compatível.

Após a preparação do ambiente é necessária a criação de um arquivo XML válido para a versão Mobile Furbot com o conteúdo inicial necessário para definir a quantidade de linhas a serem utilizadas no mundo e a posição inicial do *robô* que representa o personagem principal criado pelo aluno. Um exemplo de arquivo XML aceito pelo Mobile Furbot pode ser visualizado no Quadro 24.

```
<furbot>
<enunciado>Exercicio 2.
 Faca o  robo chegar ate a coluna 5.
 Se ele encontrar um obstaculo  deve  desviar  por  baixo.  &lt;br&gt;
 E garantido  que  nao  exista  um  obstaculo  na  ultima  linha.
&lt;br&gt;
 A primeira coluna e linhas sao a de numero ZERO.
</enunciado>
<mundo>  <!—Propriedades do mundo-->
<qtdadeLin>8</qtdadeLin> <!-- Y -->
<qtdadeCol>8</qtdadeCol> <!-- X -->
<tamanhoCel>M</tamanhoCel>
<explodir>true</explodir>
</mundo>
<robo>
<x>1</x>
<y>1</y>
<energia>400</energia>
</robo>
<!-- Cria aliens -->
<objeto class="br.furb.furbot.mobile.objetos.Alien">
<id>4</id>
<random limiteSupX="2"/>
</objeto>
<objeto class="br.furb.furbot.mobile.objetos.Alien">
<x>2</x>
<y>1</y>
</objeto>
<objeto class="br.furb.furbot.mobile.objetos.Alien">
<id>3</id>
<random limiteSupX="2"/>
</objeto>
<objeto class="br.furb.furbot.mobile.objetos.Alien">
<id>3</id>
<random limiteSupX="2"/>
</objeto>
<booleano><!-- Cria booleano -->
<valor random="true"/>
<random />
</booleano>
```

```
<numero><!-- Cria numero -->
<!-- Sortear valores de numero entre 1 e 10 -->
<valor  randomInf="0" randomSup="4"/>
<random limiteSupX="3"/>
</numero>
<numero>
<!-- Sortear valores de numero entre 1 e 10 -->
<valor  randomInf="100" randomSup="200"/>
<random limiteSupX="5"/>
</numero>
</furbot>
```

Quadro 24 – Exemplo de arquivo XML

Para iniciar o `MundoFurbotMobile` é necessária a criação de uma classe responsável por inicializar a MIDlet `MundoVisualMobile`. Um exemplo desse passo pode ser visto no Quadro 18.

Posteriormente a preparação da estrutura do XML e das bibliotecas necessárias o Mobile Furbot estará pronto para ser executado com auxílio de uma IDE configurada para a plataforma JME. Um exemplo de como a estrutura final deve ficar apresenta-se na Figura 21.

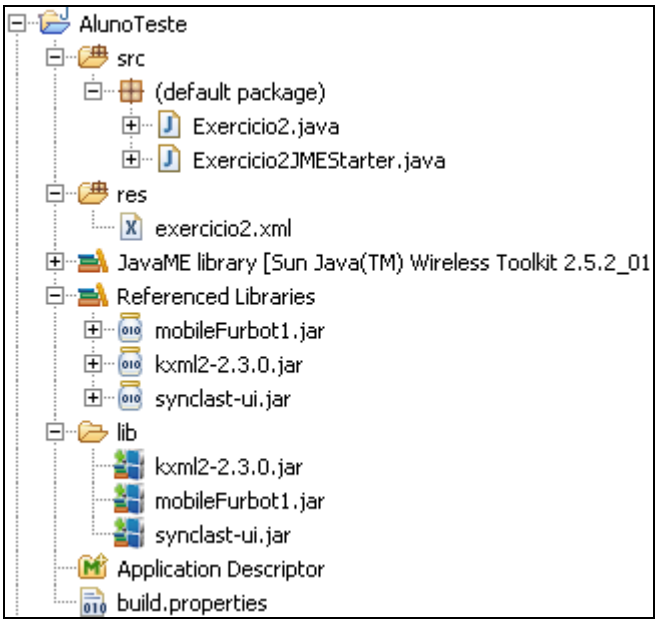


Figura 21 – Estrutura final para execução

A ferramenta Eclipse Pulsar poderá facilitar na geração do pacote executável o qual poderá ser exportado para o dispositivo móvel desejado. Um exemplo de como fazer a criação do pacote para o projeto selecionado na ferramenta Eclipse pode ser visto na Figura 22.

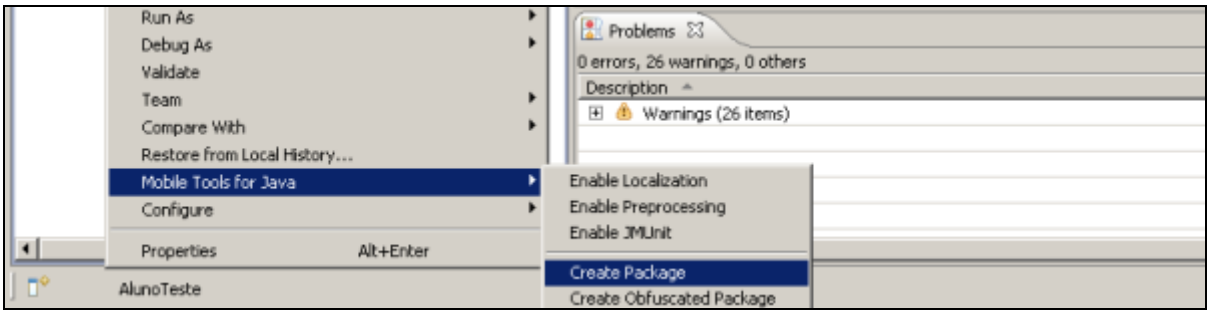


Figura 22 – Criação de pacotes com Eclipse Pulsar

Após o procedimento mostrado na Figura 22, o Eclipse cria um novo diretório chamado `deployed`, com um arquivo de extensão *Java Application Description* (JAD) e outro de extensão JAR. O arquivo JAD é o arquivo de informações e configurações sobre o programa JME e o arquivo JAR é o arquivo que conterá os recursos e classes do Mobile Furbot e as do aplicativo desenvolvido. Porém esta versão do Eclipse não indica no JAD a classe principal a ser executada, sendo necessário indicá-la, para isto é preciso editar o arquivo JAD adicionando a propriedade `MIDlet-1` (Quadro 25).

```
MIDlet-Jar-Size: 421100
MIDlet-Jar-URL: FurbotMobileParser.jar
MIDlet-Name: FurbotMobileParser MIDlet Suite
MIDlet-Vendor: MIDlet Suite Vendor
MIDlet-Version: 1.0.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0

MIDlet-1:InteracaoJMEStarter,/r2d2-icon-
mob.png,br.furb.furbot.mobile.exercicios.InteracaoJMEStarter
```

Quadro 25 – Configuração do arquivo JAD

Em seguida é necessário copiar os arquivos JAD e JAR para o dispositivo móvel e executar o arquivo JAD, assim o sistema operacional do dispositivo fará a instalação automática do aplicativo para execução.

Após executado o aplicativo Mobile Furbot, o acesso ao `grid` inicial do mundo é dado através de acesso por teclado pela a opção “Novo Mundo Furbot”. Durante a execução do exercício é possível chamar pelo menu as opções de parar, reiniciar, aumentar a velocidade, ver enunciado e mensagens do Furbot. Essas opções são apresentadas na Figura 17.

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos foram satisfatórios executando exemplos simples com poucos objetos simultaneamente no mundo. Porém foram observados alguns travamentos tanto em

ambiente de simulação quanto em um celular N95, quando testado um exemplo mais completo com muitos objetos e instruções sendo executadas ao mesmo tempo.

Com relação aos trabalhos correlatos, GreenFoot e RoboCode, o *framework* desenvolvido manteve o conceito de facilidade de criação dos personagens do jogo através da construção de classes para representar estes objetos. Em comparação com a plataforma MGBL o Mobile Furbot utilizou de recursos disponíveis para a plataforma JME via criação das regras e personagens definidos pelo professor através de um arquivo XML. Já na plataforma MGBL as regras e questões possíveis para resolução são definidas previamente, porém por cadastro *online* com foco em questionários pedagógicos.

O padrão de desenvolvimento adotado pelo *framework* Furbot utilizou de muitos recursos existentes somente para algumas versões da plataforma JSE, ocasionando uma grande incompatibilidade e muitas dificuldades na conversão das classes de modelo e de controle para o ambiente desenvolvido em JME. A incompatibilidade entre os dois *frameworks* faz com que os códigos fontes de futuras alterações precisem ser desenvolvidos novamente para as duas versões. Com intuito de eliminar este problema seria necessário que fosse feita uma reformulação dos códigos originais do Furbot para se adaptarem a qualquer plataforma Java existente, restando apenas à necessidade de trabalhar separadamente na camada de visão com as bibliotecas específicas de cada plataforma.

Pode-se perceber que o trabalho desenvolvido é relevante pela adaptação a uma realidade *desktop* com sua versão totalmente dividida em camadas MVC para uma realidade muito mais restrita em recursos de hardware e de programação como a plataforma JME. Esta adaptação tornou-se possível após o estudo das restrições existentes, criação de funções alternativas e utilização de outros tipos de bibliotecas externas para procurar manter os requisitos do *framework* Furbot compatíveis com os requisitos do *framework* Mobile Furbot.

4 CONCLUSÕES

Pode-se afirmar que o objetivo proposto para o trabalho foi alcançado. A arquitetura original do Furbot sofreu várias alterações para permitir a construção desta versão compatível para dispositivos móveis.

As ferramentas utilizadas mostraram terem sido adequadas para o desenvolvimento deste trabalho. Os ambientes de desenvolvimento e especificação também foram adequados para o sucesso do mesmo. O ambiente do simulador e o celular N95 apareceram com ótimas opções e ferramentas para facilitar nos testes e na execução não só deste trabalho, mas como em outros aplicativos utilizados como base adicional de conhecimento para esta aplicação desenvolvida em JME.

Tendo em vista a construção deste trabalho acadêmico, é possível chegar à conclusão de que é possível a conversão de uma arquitetura *desktop* para uma arquitetura *mobile*, porém muitas mudanças precisaram ser feitas para contornar as questões das trocas de bibliotecas externas e da plataforma de desenvolvimento. Este trabalho demonstra a necessidade de explorar ainda mais a área de aprendizagem em dispositivos móveis, por servir como um diferencial comercial e atrativo para ajudar no desenvolvimento do aprendizado entre professores e alunos.

Existem limitações que podem ser citadas em relação ao trabalho desenvolvido. Uma delas é o tempo de resposta em exemplos mais complexos com muitos objetos simultaneamente. Outra limitação é a dificuldade de manutenção na classe de leitura de XML, já que não existe uma biblioteca baseada em regras para a plataforma JME, acabando por dificultar a atribuição de novas funcionalidades ao exercício para o aluno.

Um dos aspectos a destacar e que valoriza o trabalho desenvolvido é que um resumo do mesmo foi submetido ao SEMINCO/2009 na forma de um artigo científico e selecionado para publicação e apresentação oral dentre as 80 submissões realizadas.

4.1 EXTENSÕES

As extensões encontradas para este trabalho foram:

a) implementar uma biblioteca de comunicação em rede para viabilizar a construção de exercícios e jogos *multiplayer* tanto para a versão *desktop* como para a versão *mobile*;
b) implementação de `Sprites` de animação, elementos de áudio e efeitos visuais para melhorar aspectos motivacionais de jogos e aplicativos criados;
c) criar uma nova versão do *framework* com a utilização dos pacotes nativos de manipulações de `Sprites` da JME com intuito de evitar a utilização de `Threads`, o que diminuiria o processamento utilizado visando obter um ambiente com melhor desempenho e de tempo menor de resposta entre as ações dos elementos do mundo;
d) construir um editor gráfico de mundo para posicionar os objetos, ditar enunciados e simplificar a definição da localidade das classes dos objetos a serem usados no XML;
e) construir uma versão do Furbot com gráficos 3D;
f) fazer um estudo de engenharia de software para tratar da migração de softwares para a plataforma JME.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, Adriano. **Batalhando e aprendendo com robocode.** [S.l.], 2007. Disponível em: <http://www.javafree.org/news/view.jf?idNew=276>. Acesso em: 20 set. 2008.

BACHMAIR, Ben; PACHLER, Norbert J. **An analysis of mobile expertise, structures and emerging cultural practices.** Kassel, 2009. Disponível em: <http://www.medienpaed.com/2009/bachmair0903.pdf >. Acesso em: 04 nov. 2009.

BERGIN, Joe et al. **Patterns for experiental learning.** [S.l.], 2001. Disponível em: <http://csis.pace.edu/~bergin/patterns/ExperientialLearning.html>. Acesso em: 20 set. 2008.

COGOI, Cristina; SANGIORGI, Daniele; KUSSAI, Shahin. **Perspectives and usage in learning and career guidance topics.** Itália, 2006. Disponível em: <http://www.elearningeuropa.info/files/media/media10911.pdf>. Acesso em: 29 out. 2009.

FRANCISCATO, Fabio T.; MEDINA, Roseclea D. **Sistema de gerenciamento de objetos de aprendizagem para dispositivos móveis.** Santa Maria, 2009. Disponível em: <http://www.cinted.ufrgs.br/renote/jul2009/artigos/9b_fabio.pdf >. Acesso em: 29 out. 2009.

GREENFOOT. **What is greenfoot?** Melbourne, 2008. Disponível em: <http://www.greenfoot.org/about/whatis.html>. Acesso em: 20 set. 2008.

LAM, Jason. **J2ME & gaming.** [S.l.], 2006. Disponível em: <http://www.jasonlam604.com/books.php>. Acesso em: 20 set. 2008.

MARÇAL, Edgar et al. **O uso de dispositivos móveis para auxiliar a aprendizagem significativa na geometria espacial.** Fortaleza, 2008. Disponível em: <http://www.sbc.org.br/bibliotecadigital/download.php?paper=1263>. Acesso em: 27 out. 2009.

MGBL. **MGBL game models.** Áustria, 2008. Disponível em: <http://www.mg-bl.com/fileadmin/downloads/Overview_game_models_1-3.pdf>. Acesso em: 2 nov. 2008.

OGLIARI, Ricardo S. **Jogos com MIDP 2.0.**: personagens. Passo Fundo, 2008. Disponível em: <http://www.javafree.org/content/view.jf?idContent=173>. Acesso em: 17 set. 2008.

PEREIRA, Nice M. **Artigo sobre MVC (Model View Controller).** Taquara, 2004. Disponível em: <http://anacarol.blog.br/old/aulas/artigos_uteis/modelo_visualizacao_controle.pdf>. Acesso em: 23 set. 2008.

ROBOCODE HOME. **Online help.** [S.l.], 2008. Disponível em: <http://testwiki.roborumble.org/w/index.php?title=Robocode_Basics>. Acesso em: 1 nov. 2008.

SAUVÉ, Jacques P. **Threads em Java.** Aracaju, 2000. Disponível em: <http://inf.unisul.br/~marmed/SENAI/PROG_WEB/Material_Pesquisa/JAVA/Apostila_Java_Exemplos/Threads.doc>. Acesso em: 23 set. 2008.

SIAU, Keng; HOONAH, Fiona. **Application of mobile technology in education.** Nebraska, 2008. Disponível em: <http://ait.unl.edu/fnah/IEEE_Editorial.pdf >. Acesso em: 27 out. 2009.

SULCOM. **Glossário de termos cartográficos.** Porto Alegre, 2007. Disponível em: <http://www.sulcom.com.br/g/glossario_de_cartografia_e_navegacao.shtml>. Acesso em: 1 nov. 2008.

SUN MICROSYSTEMS. **Java ME technology.** [S.l.], [2008?]. Disponível em: <http://java.sun.com/javame/technology/index.jsp>. Acesso em: 15 set. 2008.

VAHLDICK, Adilson; MATTOS, Mauro M. Aprendendo programação de computadores com experiências lúdicas. In: INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION, 6., 2009, Buenos Aires. **Anais…** Buenos Aires: ICECE, 2009. p. 1-6.

VALENTE, Jose A. **Formação de educadores para o uso da informática na escola.** Campinas, 2003. Disponível em: <http://www.anped.org.br/reunioes/28/textos/gt08/gt081345int.rtf >. Acesso em: 01 nov. 2009.