UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

AFURBOT – MIGRAÇÃO DO FRAMEWORK FURBOT PARA A PLATAFORMA ANDROID

ALEXANDRE RODRIGUES COELHO

BLUMENAU
2013

2013/1-04

ALEXANDRE RODRIGUES COELHO

AFURBOT – MIGRAÇÃO DO FRAMEWORK FURBOT PARA A PLATAFORMA ANDROID

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 - Orientador

BLUMENAU
2013

2013/1-04

AFURBOT – MIGRAÇÃO DO FRAMEWORK FURBOT PARA A PLATAFORMA ANDROID

Por

ALEXANDRE RODRIGUES COELHO

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, Doutor – Orientador, FURB
	_____
Membro:	Prof. Alexander Roberto Valdameri, Mestre – FURB
	_____
Membro:	Prof. Roberto Heinzle, Doutor – FURB

Blumenau, 02 de julho de 2013

Dedico este trabalho a minha família que soube apoiar em todos os momentos, e aos professores que apoiaram e me incentivaram para a conclusão desse trabalho.

AGRADECIMENTOS

À minha família, que sempre se esforçaram para me auxiliar no que fosse preciso.

Aos meus amigos, pelo apoio e ajuda que me deram.

Ao meu orientador, Mauro Marcelo Mattos, por ter acreditado na conclusão deste trabalho e principalmente me incentivou e orientou para que o trabalho fosse concluído.

Se quiser por a prova o caráter de um homem, dê-lhe poder.

Abraham Lincoln

RESUMO

Este trabalho descreve a construção de um ambiente de execução do Furbot que permita a execução de aplicativos em dispositivos móveis compatíveis com a plataforma *Android*. Neste trabalho são apresentados os elementos da arquitetura do Furbot que servem como base para a criação de uma arquitetura voltada para dispositivos móveis e 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. Android. Ferramenta de ensino de programação.

ABSTRACT

This work describes the develpment of a Furbot execution environment that allows the execution of applications developed for the Android platform. We present the elements of architecture Furbot that serve as the basis for creating an architecture focused on mobile devices and is explained on developmental differences between the architectures Destkop and mobile devices, 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 by converting the elements of architecture desktop Furbot.

Key-words: Furbot. Android. Programming teaching tool.

LISTA DE QUADROS

LISTA DE FIGURAS

LISTA DE SIGLAS

ADT – *Android Development Tools*

API – *Application Programming Interface*

DVM – *Dalvik Virtual Machine*

EDGE – *Enhanced Data-Rates for Global Evolution*

IDE – *Integrated Development Environment*

JME – *Java Micro Edition*

MVC – *Model View Control*

SDK – *Software Development Kit*

XML - *eXtensible Markup Language*

Wi-Fi – *Wireless Fidelity*

SUMÁRIO

1 INTRODUÇÃO

A cada ano novos alunos entram na universidade com o objetivo de aprender a programar e evoluírem profissionalmente. O aluno dá os primeiros passos na área através da disciplina de introdução a programação, onde exige que o mesmo aprenda sintaxe, projeto, lógica e depuração de um programa (PHELPS; EGERT; BAYLISS, 2009, p. 4-5). Segundo Kumar (2003 apud GONDIM; AMBRÓSIO, 2008, p. 1), "problemas com esta disciplina são apontados como responsáveis pelo grande número de reprovações e desistências em cursos de Computação".

Cada vez mais são criadas técnicas e metodologias de apoio ao ensino de lógica de programação, porém muitas destas técnicas não são atrativas a ponto de despertar 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 (MATTOS; VAHLDICK; HUGO, 2008). 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.

A proposta deste TCC chamada AFurbot visa permitir que aplicações desenvolvidas para a versão *desktop* do Furbot possam vir a ser executadas em dispositivos móveis compatíveis com a plataforma Android.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é criar um ambiente de execução do Furbot em dispositivos móveis como *tablets* e *smartphones* Android.

Os objetivos específicos do trabalho são:

a) disponibilizar o ambiente Furbot para a plataforma Android;
b) validar a aplicação através de um estudo de caso utilizando exercícios aplicados em sala de aula.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos. No primeiro capítulo é apresentado a introdução do trabalho, o objetivo principal, os objetivos específicos e uma descrição sobre a estrutura do trabalho.

O segundo capítulo contempla a fundamentação teórica do trabalho onde é descrito uma explicação sobre os componentes e a funcionalidade do *framework* Furbot. São explicados os componentes da arquitetura Android e por fim é apresentado as ferramentas utilizadas no desenvolvimento e os trabalhos correlatos.

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

2.1 FRAMEWORK FURBOT

Segundo Mattos; Vahldick; Hugo (2008, p. 3), 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, p. 4). O Quadro 1 exemplifica a programação da inteligência do Furbot.

Quadro 1 - Exemplo de 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{
             //trata a situação em que existe um obstáculo na direção do furbot
          }
      }
}
```

Fonte: Mattos, Vahldick e Hugo (2008, p. 4).

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 (VAHLDICK; MATTOS, 2009, p. 3).:

a) facilidade de implementação do código utilizando um fluxo seqüencial 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)[1] 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, através de um arquivo *eXtensible Markup Language (*XML), que contém informações como dimensões do mundo e os elementos que compõem o cenário do jogo.

Uma especificação de problema é composta basicamente por três seções: enunciado, caracterização das coordenadas do mundo e caracterização dos objetos que populam o mundo. No Quadro 2 é apresentado uma 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.

Quadro 2 - Exemplo de mundo em arquivo XML

```
<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: Mattos, Vahldick e Hugo (2008, p. 5).

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).

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

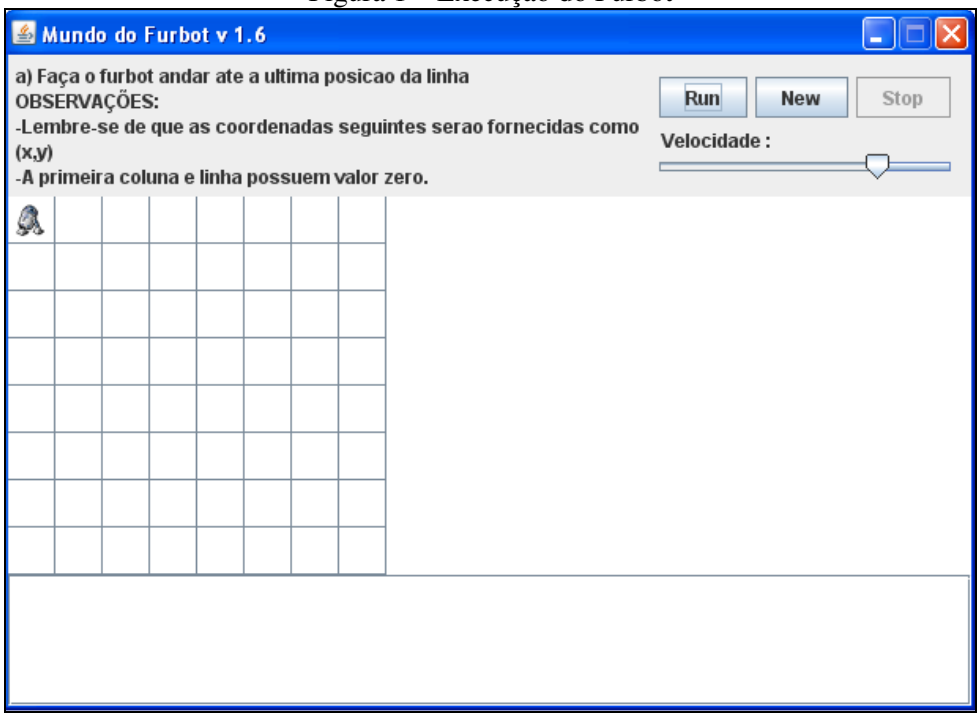
Quadro 3 – Carga do arquivo XML

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

Fonte: Mattos, Vahldick e Hugo (2008).

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



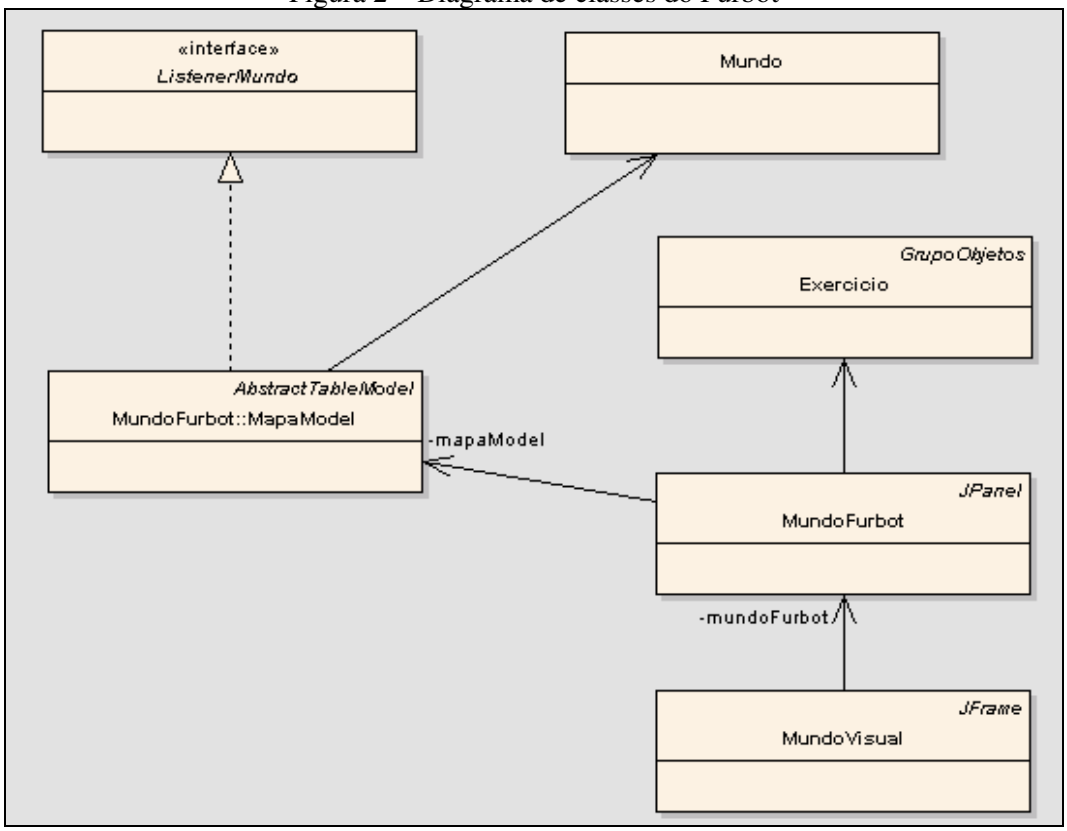
Fonte: Mattos, Vahldick e Hugo (2008).

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.

A Figura 2 apresenta um diagrama de classes em uma visão macro da arquitetura da versão atual do Furbot. A classe `MundoVisual` utiliza componentes gráficos do Java `Swing` para controle de um `MundoFurbot` que associa à um `Exercicio` (que contém informações do exercício obtidas a partir de um arquivo XML).

O `MundoVisual` é acessado através da implementação de uma interface `ListenerMundo`. A classe `Mundo` contém os tratadores de eventos do `MundoFurbot` e ela é instanciada pela classe `MundoFurbot`.

Figura 2 – Diagrama de classes do Furbot



Fonte: Mattos, Vahldick e Hugo (2008).

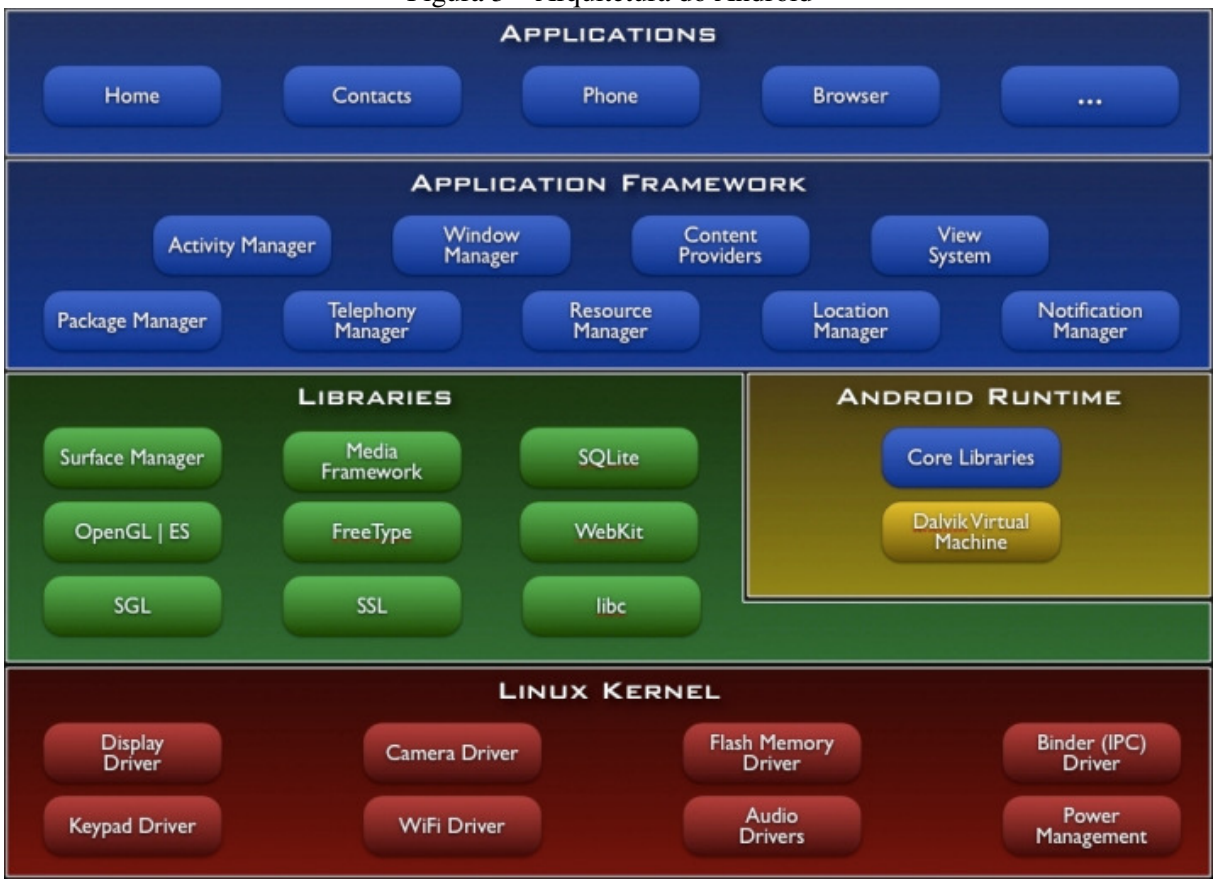
2.2 ANDROID

O diagrama da Figura 3 mostra os principais componentes do sistema operacional Android. Segundo Aquino (2007, p. 5), ele é formada por camadas, onde os níveis mais baixos são expostos para os níveis mais altos através de interfaces. A primeira camada é o núcleo do sistema (*kernel*) que funciona como uma camada de abstração entre o hardware e o restante de softwares da plataforma. Utilizando Linux 2.6.27, ele é responsável pelo gerenciamento de memória, processos, *threads*, pilha de protocolos de rede, módulo de segurança e vários módulos do núcleo de infraestrutura.

A segunda camada contém um conjunto de bibliotecas C/C++ usada por vários componentes do sistema e a *Dalvik Virtual Machine* (DVM). A DVM foi escrita de forma que um dispositivo possa executar múltiplas máquinas virtuais concorrentemente de maneira eficiente. Ela usa o *kernel* do Linux para prover a funcionalidade de múltiplas *threads* e gerenciamento de memória de baixo nível. Cada aplicação do Android roda em um processo

separado, com sua própria máquina virtual, número do processo e dono do processo. Isso garante que caso a aplicação apresente erros, ela possa ser isolada e removida da memória sem comprometer o resto do sistema. Ao desenvolver as aplicações em Java, a DVM compila o *bytecode* (`.class`) e converte para o formato *Dalvik Executable* (`.dex`), que representa a aplicação do Android compilada. Depois disso, os arquivos `.dex` e outros recursos do projeto são compactados em um único arquivo com a extensão *Android Package File* (`.apk`), que representa a aplicação final.

Figura 3 – Arquitetura do Android



Fonte: Google (2012).

A terceira camada é composta por *frameworks* de aplicações. Os desenvolvedores têm acesso completo a mesma interface de programação de aplicativos que é usada pelas aplicações essenciais da plataforma. Esta camada fornece um conjunto de serviço tais como *Activity Manager*, que gerencia o ciclo de vida das aplicações; *Package Manager*, que mantêm quais aplicações estão instalaadas no dispositivo; *Window Manager,* que gerencia as janelas das aplicações; *Telephony Manager*, que são os componentes para acesso aos recursos de telefonia; *Content Providers*, que permitem que as aplicações acessem dados de outras aplicações ou compartilhem os seus próprios dados; *Resource Manager*, que fornece acesso aos recursos gráficos e arquivos de *layout*; *View System*, que é um conjunto rico e extensível

de componentes de interface do usuário; *Location Manager*, que gerencia a localização do dispositivo; *Notification Manager*, que permite que todas as aplicações exibam alertas na barra de *status*. A quarta camada é composta pelas aplicações propriamente ditas.

A plataforma Android oferece suporte a diversas tecnologias de conectividade tais como *Bluetooth*, *Enhanced Data-Rates for Global Evolution* (EDGE), 3G e *Wireless Fidelity* (Wi-Fi). Este suporte disponibiliza a comunicação com a internet, podendo ser usado o conceito *sockets*, através da linguagem Java que é compilada e executada na DVM para realizar esta conexão com a rede mundial de computadores.

2.3 TRABALHOS CORRELATOS

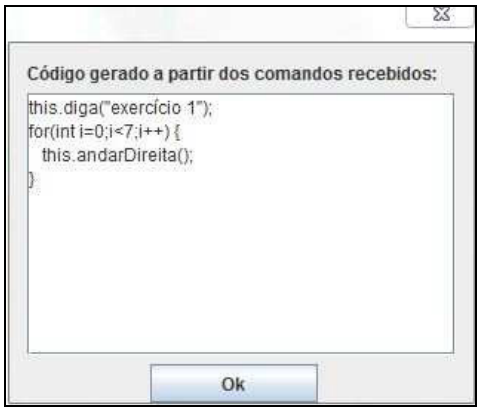
Algumas ferramentas desempenham papel semelhante ao proposto. Dentre elas, foram selecionadas: Furbot-C (MAUS, 2011), Mobile Furbot (ESTRAZULAS, 2009) e iFurbot (CORRÊA, 2009).

2.3.1 Furbot-C

O Furbot-C (MAUS, 2011) é uma extensão para o *framework* Furbot que traz a possibilidade para o aluno resolver exercícios controlando o robô pelo teclado, podendo movimentar o robô, empurrar, contar e remover objetos ou digitar algo para que o robô diga. Após a solução o aluno tem a opção de gerar o código da inteligência do robô baseado nos comandos que foram executados, onde o aluno pode ver e utilizar o código para uma possível solução para o exercício.

A figura 4 apresenta um exemplo de um código gerado pelo Furbot-C.

Figura 4 – Código gerado a partir dos comandos recebidos



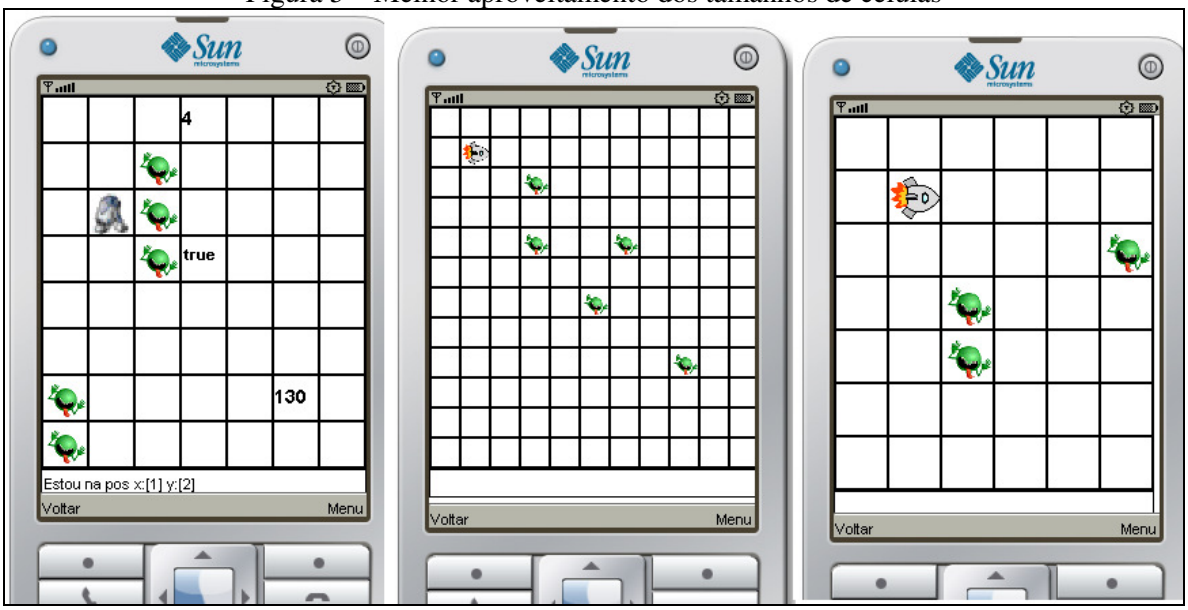
Fonte: Maus (2011).

2.3.2 Mobile Furbot

Estrázulas (2009) descreve a construção de um ambiente de execução do Furbot onde aplicativos desenvolvidos para versão *desktop* podem ser executados 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.

A figura 5 apresenta um exemplo de execução do Mobile Furbot com diferentes tamanhos de células do *grid*.

Figura 5 – Melhor aproveitamento dos tamanhos de células



Fonte: Estrázulas (2009).

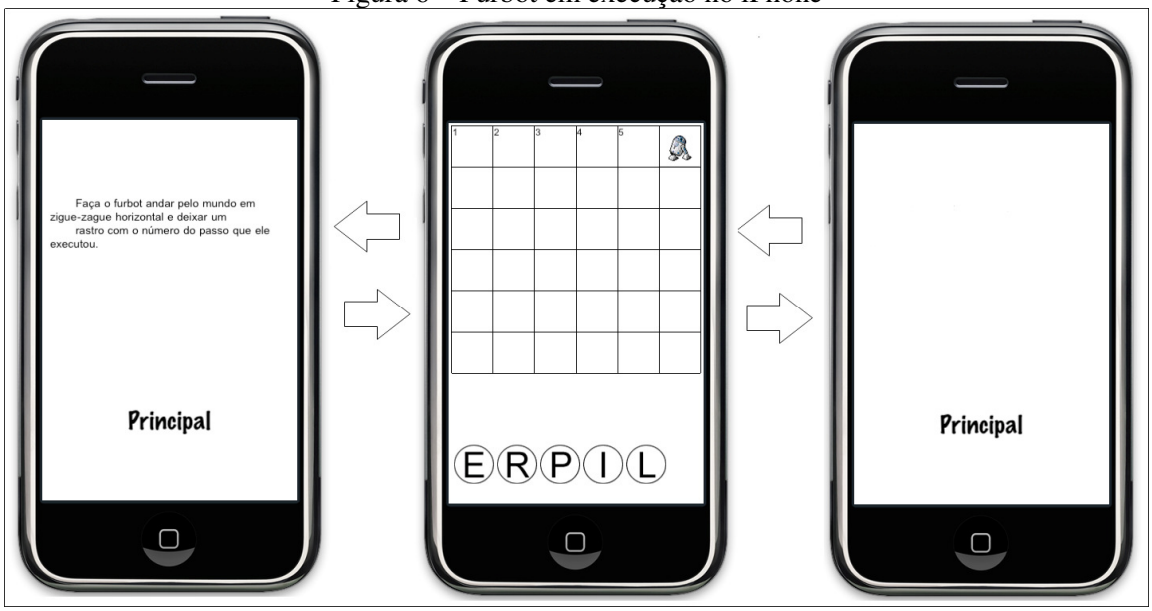
Conforme Estrázulas (2009, p. 56), a arquitetura original do Furbot sofreu várias alterações para permitir a construção desta versão compatível para dispositivos móveis. Existem limitações que podem ser citadas em relação ao trabalho desenvolvido. Uma delas é o tempo de resposta em exemplos mais complexos 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.

2.3.3 iFurbot

Correa (2009) apresenta a conversão do *framework* Furbot desenvolvido na linguagem de programação Java para a linguagem Objective-C, tornando possível sua utilização no desenvolvimento de aplicações para iPhone. São utilizados recursos como acelerômetro e *multi-touch*, disponíveis no iPhone, para atender as funcionalidades já existentes na versão do Furbot em Java. O *framework* Furbot utilizado em disciplinas de introdução a programação busca facilitar o aprendizado de lógica e linguagens de programação, utilizando como

incentivo o desenvolvimento de aplicações envolvendo jogos. A figura 6 apresenta um exemplo de execução do Furbot no ambiente de simulação IPhone.

Figura 6 – Furbot em execução no iPhone



Fonte: Corrêa (2009).

Conforme Corrêa (2009, p. 49) as linguagens Java e Objective-C possuem várias semelhanças, principalmente devido a sua origem ser a linguagem C e C++. A complexidade encontrada em Objective-C deveu-se a sua sintaxe diferenciada. Enquanto em Java é utilizada uma sintaxe de declarações de métodos que é muito semelhante às funções C, Objective-C, por outro lado, utiliza uma sintaxe que é muito mais próxima de Smalltalk utilizando o conceito de mensagens.

Ainda, em Objective-C há a necessidade de um controle mais rígido da memória alocada com o uso das diretivas, `retain` e `release`, o que dificulta a conversão já que em Java este tratamento não é necessário por utilizar *garbage collection*. Em algumas situações onde não houve interação *multi-thread*, foi utilizado o objeto `NSAutoReleasePool` para gerenciamento da memória.

Durante a conversão do Furbot levou-se em consideração que haveriam manutenções em ambas as versões além de futuras implementações. Sendo assim, foi mantida a nomenclatura das classes, métodos e variáveis, tentando minimizar a complexidade das alterações, deixando apenas a sintaxe e os objetos oferecidos pelas linguagens como diferencial.

3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas de desenvolvimento do projeto. Primeiramente são descritos os requisitos do trabalho. Em seguida são apresentadas a especificação e implementação, destacando e explicando as principais classes e funções do aplicativo. Por fim, são apresentados os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O ambiente proposto deverá:

a) permitir a criação de jogos bidimensionais para Android (Requisito Funcional - RF);
b) permitir a formulação de exercícios a serem executados (RF);
c) utilizar arquivos do tipo XML para armazenar propriedades de configuração do mundo e seus personagens (Requisito Não Funcional - RNF);
d) permitir a criação de jogos bidimensionais, considerando as limitações de tela do Android (RF);
e) converter os controles e objetos do mundo do Furbot para a estrutura de *Activities* do Android (RNF);
f) permitir executar aplicações desenvolvidas utilizando o *framework* convertido em plataformas móveis baseadas em Android (RNF).

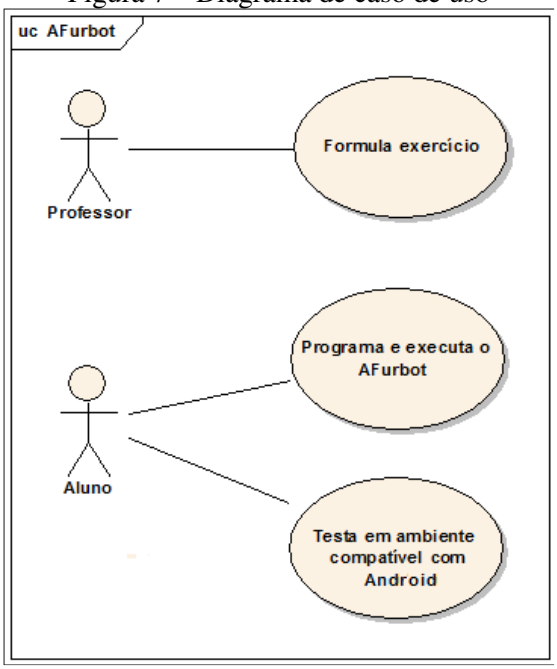
3.2 ESPECIFICAÇÃO

Esta seção demonstra o diagrama de classes, diagrama de sequencia e de uso de caso. Os diagramas foram criados utilizando a ferramenta Enterprise Architect 10.0.1007 versão trial.

3.2.1 Diagrama de caso de uso

O diagrama de caso uso apresentado na Figura 7, 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.

Figura 7 – Diagrama de caso de uso



O Quadro 4 apresenta o detalhamento do caso de uso “Formula exercício”.

Quadro 4 – Caso de Uso Formula exercício

UC01 - Formula exercício
Caso de Uso: O professor de programação define o tema e objetivo do AFurbot através de um arquivo XML. ***Cenários*** 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.

O Quadro 5 apresenta o caso de uso “Programa e executa o AFurbot”.

Quadro 5 – Caso de Uso Programa e executa o AFurbot

UC02 - Programa e executa o AFurbot
Caso de Uso:
Restrições ▪ *Pré-condição.* Ter o arquivo XML do exercício proposto. ▪ *Pré-condição.* Ter as classes do framework devidamente importadas. ▪ *Pós-condição.* O aluno pode compilar as classes dos requisitos do exercício.
Cenários Constrói lógica do exercício no framework {Principal}. 1. O aluno recebe o XML com o enunciado do exercício. 2. O aluno implementar o método inteligência do AFurbot em um ambiente de programação desktop. 3. O aluno executa o AFurbot para ver o campo inicial proposto pelo professor e o seu enunciado. 4. O framework apresenta o grid e as opções de menu para desenvolvimento do exercício. 5. O aluno formula e refina a inteligência de seus elementos criados. 6. O aluno chega a solução para resolver o objetivo do enunciado. 7. O framework para a execução dos elementos. Finaliza sem alcançar a solução {Alternativo}. No passo 6, o aluno encontra-se com situação de insucesso na solução , então este volta ao passo 2 tentando uma nova forma de programação da suas classes.

O Quadro 6 apresenta o caso de uso “Testa em ambiente compatível com AFurbot”.

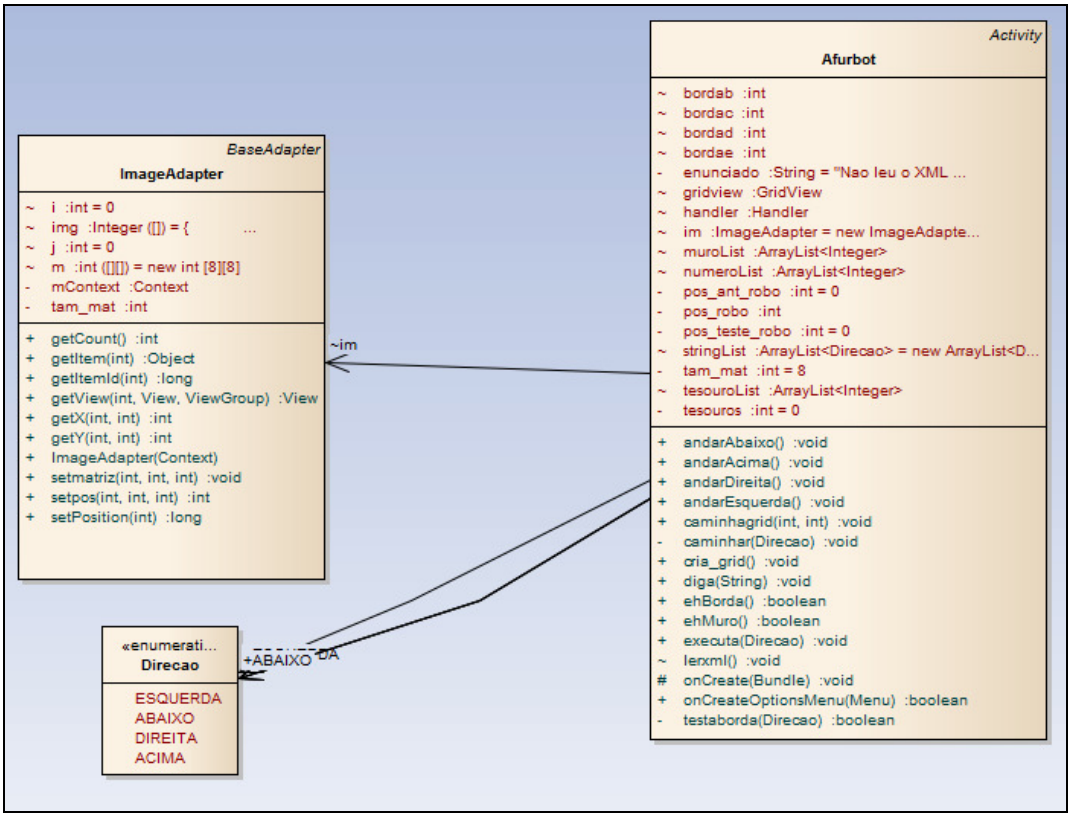
Quadro 6 – Caso de Uso Testa em ambiente compatível com AFurbot

UC03 - Testa em ambiente compatível com AFurbot
Caso de Uso:
Restrições ▪ *Pré-condição.* Ter elementos do mundo programados. ▪ *Pré-condição.* Ter as classes do AFurbot. ▪ *Pós-condição.* Exercício executado e solucionado.
Cenários Controla as ações no mundo AFurbot {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 AFurbot em um ambiente suportado pela plataforma Android. 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 aguarda finalização da execução os objetos.

3.2.2 Diagrama de classes

A Figura 8 demonstra um diagrama de classes com as principais classes contidas no pacote `AFurbot`.

Figura 8– Diagrama de classes do pacote furbot em Android



3.3 IMPLEMENTAÇÃO

Esta seção descreve técnicas e ferramentas utilizadas para o desenvolvimento do projeto. São detalhadas as principais classes, apresentando trechos de código para facilitar o entendimento. Por fim, é apresentada a operacionalidade dos aplicativos.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento de todos os aplicativos foi utilizada a linguagem de programação Java em conjunto com o *Android Software Development* Kit (SDK), que disponibiliza as APIs e ferramentas necessárias para desenvolvimento para a plataforma Android. Como *Integrated Development Environment* (IDE), foi utilizado o Eclipse, juntamente com o *plugin Android Development Tools* (ADT).

O Android SDK disponibiliza várias bibliotecas e ferramentes essencias para o desenvolvimento de aplicativos para o sistema operacional Android, incluindo um emulador e um *debbuger* para facilitar os testes. O desenvolvimento da aplicação foi realizado utilizando dois modelos de dispositivos móveis, sendo eles um *smartphone* Samsung Galaxy S III, da

fabricante Samsung com o sistema operacional *Android Jelly Bean*, e um *tablet* Motorola Xoom II ME, da fabricante Motorola com o sistema operacional *Ice Cream Sandwich*.

3.3.2 Configuração da aplicação

Nas aplicações para Android, é comum haver um arquivo de *manifest,* normalmente nomeado de `AndroidManifest.xml`, onde são determinadas todas as configurações da aplicação, como por exemplo as activities, layouts, permissões, entre outros. O quadro 7 apresenta o conteúdo do `AndroidManifest.xml` da aplicação do AFurbot.

Quadro 7– AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.furb.AFurbot"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/r2d2"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="br.furb.AFurbot.Afurbot"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Além do arquivo de *Manifest*, o Android possui outros arquivos XML os quais foram utilizados na criação da interface gráfica. Nessa implementação foi utilizada somente uma *activity*, a qual está representada no quadro 8. Esse XML é responsável por criar a interface gráfica do AFurbot, onde possui elementos de grid, de texto e de botões.

Quadro 8 – activity_afurbot.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".Afurbot" >
    <TextView
        android:id="@+id/textoinicio"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="top"
        android:text="@string/texto_inicio" />
    <GridView
        android:id="@+id/gridview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:columnWidth="30dp"
        android:gravity="center"
        android:horizontalSpacing="3dp"
        android:numColumns="8"
        android:stretchMode="columnWidth"
        android:layout_below="@+id/textoinicio"
        android:verticalSpacing="3dp" />
    <TextView
        android:id="@+id/textodiga"
        android:layout_alignParentBottom="true"
        android:layout_toRightOf="@+id/button01"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="10dp"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/texto_diga" />
    <Button
        android:id="@+id/button01"
        android:layout_alignParentBottom="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="run"
        android:text="@string/botao_run" />
</RelativeLayout>
```

3.3.3 Operacionalidade da implementação

Como funcionalidade do próprio Furbot, o AFurbot faz a leitura de um arquivo XML com as informações referentes ao posicionamento dos objetos do mundo e executa instruções programadas pelo aluno. Nesta seção é descrito todos os passos para a implementação do Furbot em Android.

O primeiro passo na execução do programa é a leitura do XML de configuração do Furbot. Existem algumas maneiras de fazer a leitura de um XML no Android, mas a utilizada foi *DocumentBuilder* pela simplicidade e agilidade na implementação visto que o XML de configuração do Furbot também é simples. No quadro 9 segue um exemplo da leitura do XML.

Quadro 9 - Carregamento do arquivo XML e leitura da tag tesouros

```
private NodeList nodeList;
Document doc;

XML (Context context){
    try {
        InputStream url=context.getResources().openRawResource(R.raw.furbot);
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        doc = db.parse(new InputSource(url));
        doc.getDocumentElement().normalize();

        nodeList = doc.getElementsByTagName("furbot");
    } catch (Exception e) {
        System.out.println("XML Pasing Excpetion = " + e);
    }

}

 public ArrayList<Integer> tesouros (){
     nodeList = doc.getElementsByTagName("tesouro");
     ArrayList<Integer> tesouro = new ArrayList<Integer>();
     for (int i = 0; i < this.nodeList.getLength(); i++){
         tesouro.add(Integer.parseInt(nodeList.item(i).getFirstChild().getNodeValue()));
     }
     return tesouro;
 }
```

Após o carregamento do XML do Furbot é criado o mundo baseado nos valores lidos neste XML.

A interface utilizado foi o *Layout Gridview* para manter a similaridade com o Furbot. O *gridview* é criado através do XML activity_afurbot.xml, mas alterado no inicio da execução para adequar o número de colunas e iniciar o carregamento das imagens para cada célula do grid conforme o quadro 10.

Quadro 10 - Criação do Grid informando o número de colunas

```
public void cria_grid (){
    gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));
    gridview.setNumColumns(col);
}
```

Para o carregamento das imagens referentes ao AFurbot em cada célula (*view*) do *grid* utilizou-se a class *Image Adapter* conforme o quadro 11. Cada célula é preenchida primeiramente com uma imagem em branco definida em um array de imagens.

Quadro 11 - Inserção da imagem em cada célula

```
public View getView(int position, View convertView, ViewGroup parent) {
    ImageView imageView;
    if (convertView == null) {
        imageView = new ImageView(mContext);
        imageView.setLayoutParams(new GridView.LayoutParams(80, 80));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(1,1,1,1);
    } else {
        imageView = (ImageView) convertView;
        setmatriz(position,8,0);
    }

    imageView.setImageResource(img[position]);
    return imageView;
}
```

A insersão dos outros elementos do AFurbot como o robô, muros e tesouros é feita após a criação do *grid*, sendo portanto alterado dinamicamente através do código conforme o quadro 12. Para fazer essa alteração utiliza-se a classe "R", criada pelo próprio ambiente de desenvolvimento, para alterar o vetor de imagens. Após a alteração é necessário avisar o Android que deve ser feita uma atualização nos elementos do *grid*.

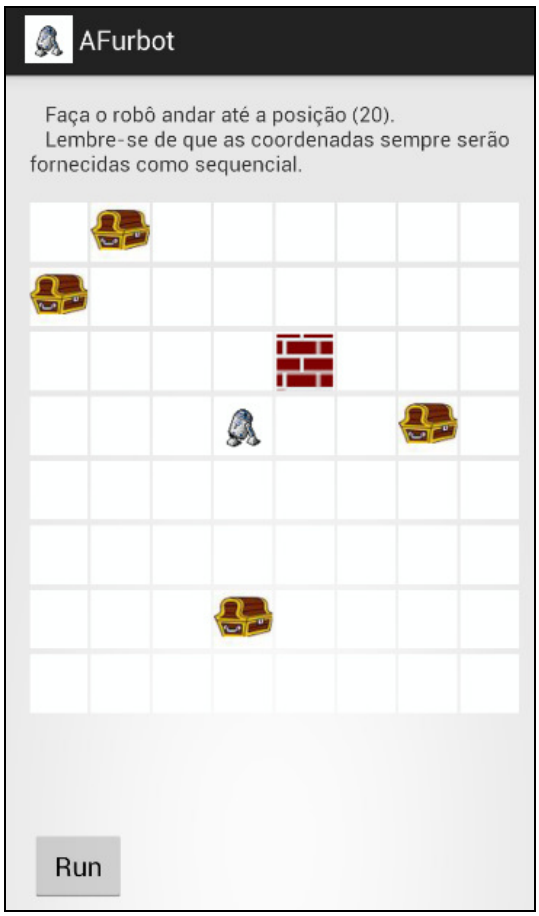
Quadro 12 – Alteração da imagem em uma posição específica

```
public void pos_matriz (int position, int tipo) throws InterruptedException{
    switch (tipo){
    case 1: im.img[position]=R.drawable.r2d2; break;
    case 2: break;
    case 3: im.img[position]=R.drawable.wall;break;
    case 101: im.img[position]=R.drawable.tesouro;break;
    case 201: im.img[position]=R.drawable.tesouro;break;  //numeros
    }

    im.notifyDataSetChanged();
    gridview.setAdapter(im);
    gridview.invalidateViews();
    im.setmatriz(position,tam_mat,tipo);
}
```

Após o carregamento do XML e da montagem do Grid com as imagens lidas através do XML de configuração, é apresentado o Furbot ao usuário conforme a figura 9.

Figura 9 - AFurbot



Ao pressionar o botão Executar o robô irá percorrer o passos criados pelo aluno.

Para realizar o caminhamento do robô através do mundo algumas construções foram necessárias conforme descrito abaixo:

a. escolha da direção que o robô seguirá sem ultrapassar as barreiras do fim do mundo conforme o quadro 13;

Quadro 13 - Método caminhar

```
private void caminhar(Direcao direcao) throws InterruptedException {
    bordad = 0; bordab = 0; bordac = 0; bordae = 0;
    switch (direcao) {
    case ABAIXO:
            if (im.getX(pos_robo, tam_mat)==col-1){
                bordab=1;
            }else x++;
        break;

    case ACIMA:
            if (im.getX(pos_robo, tam_mat)==0){
                bordac=1;
            }else x--;
        break;

    case DIREITA:
            if (im.getY(pos_robo, tam_mat)==col-1){
                bordad=1;
            }else{
                y++;
            }
        break;

    case ESQUERDA:
            if (im.getY(pos_robo, tam_mat)==0){
                bordae=1;
            }else{
                y--;
            }
        break;
    }
```

b. posicionamento no *Gridview*. Esse foi um dos problemas encontrados pois o posicionamento no *gridview* é realizado me maneira sequencial e o posicionamento do Furbot é feito através de uma matriz. Então foi necessário a implementação de métodos de conversão entre os dois diferentes tipos de posicionamento conforme o quadro 14;

Quadro 14 - Conversão xy e posição

```
public int setpos(int x, int y, int tam){
    return ((x*tam)+y);
}

public int getY(int position, int tammatriz){
    return (position%tammatriz);
}

public int getX(int position, int tammatriz){
    return (position/tammatriz);
}
```

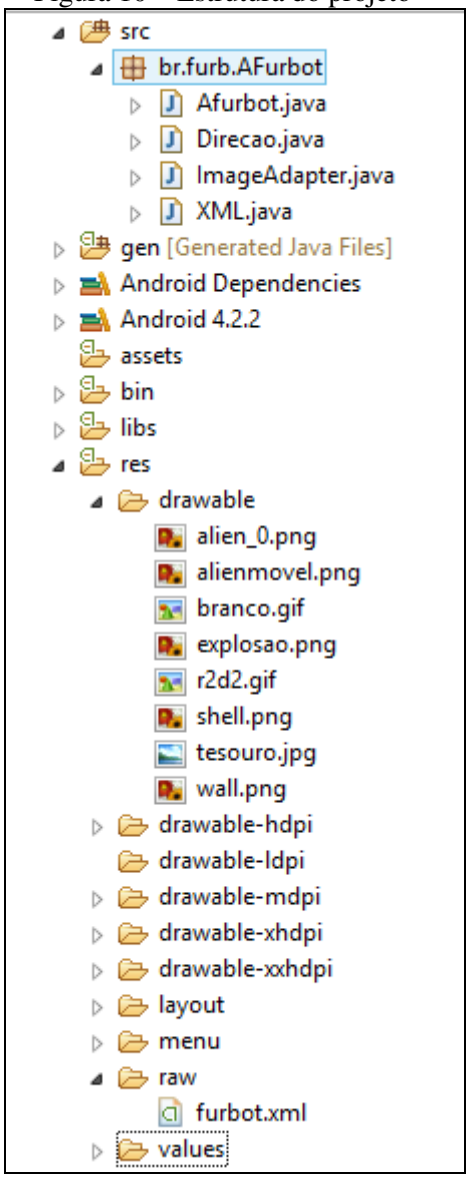
c. execução dos comandos de caminhamento. Para a execução dos comandos definidos pelo aluno era necessário que o robô caminhasse célula por célula para que o aluno visualizasse o resultado, mas para isso não era possível simplesmente caminhar de célula a célula esperando um determinado tempo entre uma e outra. Por motivos de performance o Android não permite o acesso livre à Thread principal, ou seja, a Activity principal. Como era necessário, para cada mudança de célula do robo, uma atualização gráfica completa do gridview, a Thread principal não permitia essa atualização a não ser no final da execução. A solução encontrada foi criar um *arraylist* de comandos e executá-los cada um em uma thread de forma recursiva. Para isso foi necessário também criar um método que simulasse o caminhamento antes de adicionar o mesmo ao arraylist evitando *loops* infinitos. O quadro 15 apresenta o código para a execução das *threads*;

Quadro 15 - Executa os comandos criados pelo aluno

```
public void executa(final Direcao a) {
    handler.postDelayed(new Runnable() {
       public void run() {
          try {
            caminhar(a);
            aux_tam++;
            if (aux_tam < tam_list){
                executa(stringList.get(aux_tam));
            }
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
       }
     }, 500);
}
```

d. para executar o framework AFurbot necessita-se do eclipse juntamente com o SDK do Android e importar o projeto. A figura 10 demonstra o ambiente preparado com o AFurbot.

Figura 10 – Estrutura do projeto



Após a preparação do ambiente é necessário a criação no arquivo XML chamado “furbot.xml” na pasta “res/raw” dentro da estrutura de pastas do projeto, o qual possui as informações referentes ao posicionamento dos elementos do mundo do Furbot. Como cenário de teste foi utilizado o exemplo 4 da apostila do Furbot conforme descrito na quadro 16.

Quadro 16 – Exercicio 4 da apostila do Furbot

```
<furbot>
<enunciado>Exercicio 4
Faca o robo andar ate os extremos do mundo retornando a posicao inicial.
Cada vez que o robo atingir um dos extremos, faca-o informar que ele
chegou ate aquela posicao.
Lembre-se de que as coordenadas sempre serao fornecidas como (x, y).
A primeira coluna e linhas sao a de numero ZERO.</enunciado>
<mundo>
  <qtdadeLin>8</qtdadeLin>
  <qtdadeCol>8</qtdadeCol>
  <explodir>false</explodir>
</mundo>
<robo>
  <x>0</x>
  <y>0</y>
</robo>
</furbot>
```

Diferentemente do Furbot para desktop, onde o método `inteligencia()` é criado em outra classe herdando o Furbot, nessa implementação para Androi deve-se utilizar o método `inteligencia()` na própria classe `Afurbot`, mantendo a chamada explicita da execução dos comandos (`executa (stringList.get(aux_tam))`) conforme o quadro 17.

Quadro 17 – Implementação do método inteligência

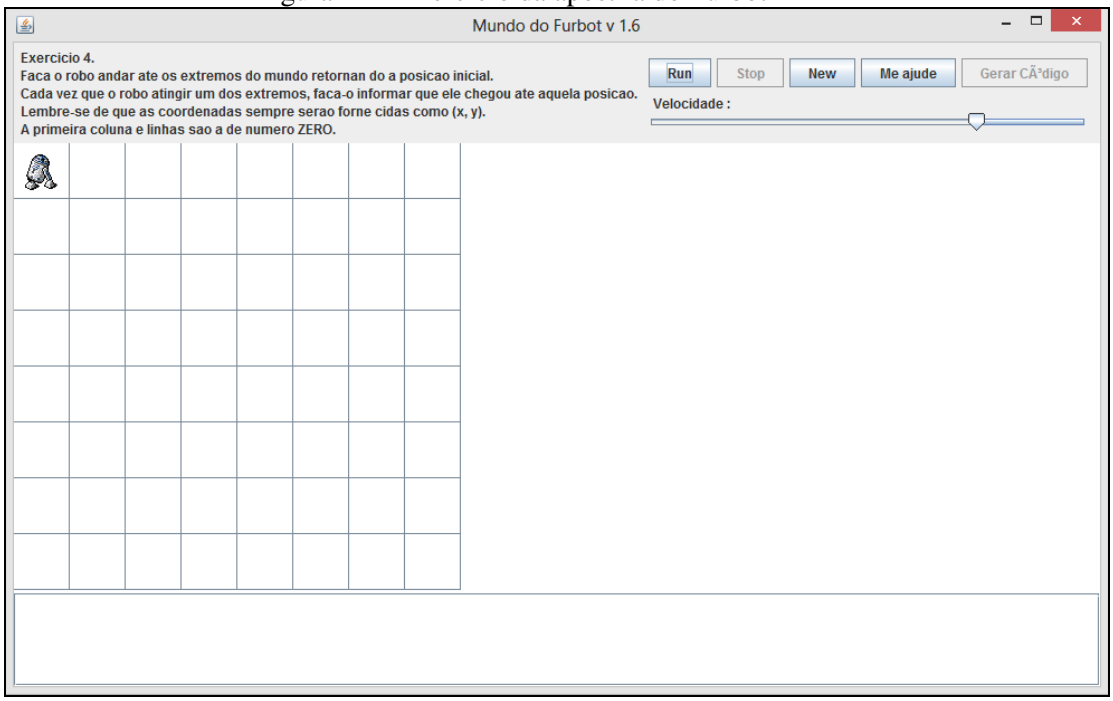
```
public void inteligencia (View view) throws InterruptedException {

    diga("exercicio4");
    while (!ehFim(DIREITA))     {
    //anda até o canto superior direito
        andarDireita();
    };
    diga ("cheguei no canto superior direito");
    while (!ehFim(ABAIXO)) {
    //anda até o canto inferior direito
        andarAbaixo();
    };
    diga ("cheguei no canto inferior direito");
    while (!ehFim(ESQUERDA)) {
    //anda até o canto inferior esquerdo
        andarEsquerda();
    };
    diga ("cheguei no canto inferior esquerdo");
    while (!ehFim(ACIMA)) {
    //anda até o canto superior esquerdo
        andarAcima();
    };
    diga ("retornei ao inicio");

    //====== Fim inteligencia AFurbot
    tam_list = stringList.size();
    executa (stringList.get(aux_tam));
}
```

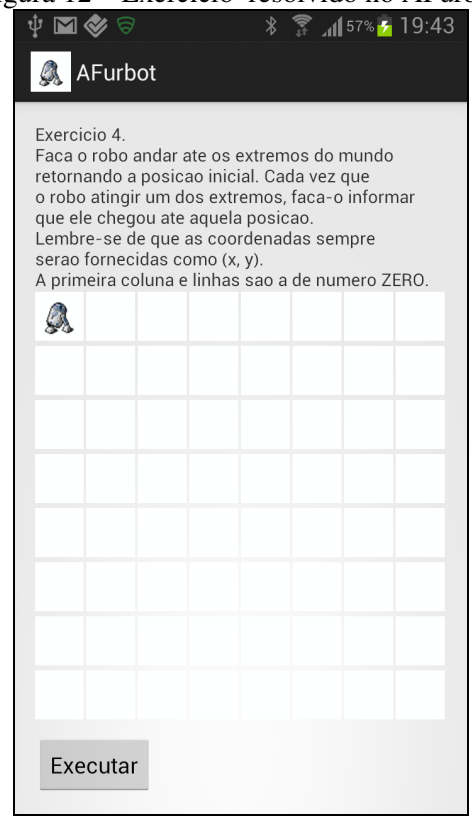
A figura 11 apresenta a imagem da execução do Furbot utilizando o exercício 4 da apostila original.

Figura 11 – Exercício da apostila do Furbot



Para fins comparativos, a figura 12, representa a imagem da execução do mesmo exercício 4 da apostila original do Furbot, no AFurbot.

Figura 12 – Exercício resolvido no AFurbot



3.3.4 Análise Comparativa

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

Quadro 18 – Implementação do método inteligência

Furbot	AFurbot
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	Os arquivos de imagens do framework AFurbot ficam no diretório "/res/drawable", um padrão estabelecido para o desenvolvimento para Android. A classe `findViewById` juntamente com a classe `R.drawable` é utilizada para fazer a busca do arquivo nesse diretório.
Atualmente em sua versão 1.7 utilizando a plataforma JSE como meio de programação, este utiliza de muitos recursos auxiliares como: Tipo Enumeração, `ArrayList` e outras classes de comum uso, existentes nos pacotes Standard do Java.	No AFurbot não foi possível utilizar alguns recursos que o Furbot utiliza na versão desktop, mas puderam ser substituídos por outras estruturas mais adequeadas a arquitetura, como *thread* foi substituída por *handler*.
Os elementos gráficos, como grid, botões, paineis, entre outros devem ser criados explicitamente via comandos.	Os elementos gráficos podem ser todos criados através de uma definição XML, mas também podem ser criados e manipulados de maneira explícita no código do projeto.

O quadro 19 apresenta uma análise comparativa referente aos trabalhos correlatos, identificando algumas diferenças entre os trabalhos.

Quadro 19 – Implementação do método inteligência

Trabalho	Ambiente de desenvolvimento	Linguagem de programação	Interface Gráfica
Furbot	Eclipse	Java	AWT
Mobile Furbot	Eclipse	Java ME	AWT
IFurbot	Xcode	Objective-C	Cocos2d
AFurbot	Eclipse	Java	XML

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtivos foram satisfatórios levando em consideração que a linguagem e o ambiente utilizado para o desenvolvimento para android foram os mesmos utilizados para a criação do Furbot *desktop*. A diferença no desenvolvimento, encontrada nesse projeto, difere principalmente na interface gráfica, na manipulação de Threads e leitura de arquivos XML.

Durante a migração do Furbot para android levou-se em consideração a possibilidade de futuras implementações e manutenções. Sendo assim, foi minimizado a diferença entre as classes e métodos limitados à nova arquitetura.

A forma de caminhamento do robo através do mundo teve que ser alterado devido a restrições da arquitetura referente as alterações em sua interface principal (*Activity*), através de outras classes, privilegiando assim a performance do aplicativo. Para isso foi utilizado a classe *Handler* para simular a espera em cada célula do grid.

A utilização da ferramenta eclipse juntamente com o SDK do Android, permitiu uma adaptação mais ágil e efetiva para o desenvolvimento para essa arquitetura.

4 CONCLUSÕES

O objetivo desse trabalho de converter o *framework* Furbot utilizando a tecnologia Java para a plataforma Android foi concluído com êxito. Com o *framework* já convertido, é possível a criação de aplicações para dispositivos móveis através da codificação do método `inteligencia` do robô e o uso do emulador Android para efetuar testes na aplicação.

As ferramentas utilizadas mostraram-se adequadas para o desenvolvimento deste trabalho. Os ambientes de desenvolvimento e especificação também foram adequados para o sucesso do mesmo. O desenvolvimento foi realizado utilizando-se a transferencia direta do código para um dispositivo Samsung Galaxy S III eliminando-se desta forma a necessidade do uso do simulador.

Através do desenvolvimento desse trabalho acadêmico, chegou-se à conclusão de que é possível a conversão de uma arquitetura *desktop* para uma arquitetura *mobile* baseada em Android, porém muitas mudanças precisaram ser feitas para contornar as questões das trocas 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. Este problema também foi relatado em Estrázulas (2009) na plataforma JME. Outra limitação é a rotação da tela a qual faz-se necessário uma outra configuração completa dos componentes gráficos.

Em relação aos trabalhos de Estrázulas (2009) e Correa (2009) pode-se afirmar que o *framework* FURBOT cumpre a finalidade em cada uma das plataforma móveis: JME, IOS e Android. Contudo esforços adicionais são necessários para tornar a plataforma mais robusta e genérica em relação aos recursos disponibilizados em cada uma delas. Em relação ao trabalho de Maus (2011), as funcionalidades não foram implementadas e são consideradas como uma extensão ao presente trabalho.

4.1 EXTENSÕES

As extensões sugeridas para este trabalho são:

a) implementar uma biblioteca de comunicação em rede para viabilizar a construção

de exercícios e jogos *multiplayer* tanto para a versão *desktop* como para as versões *mobile* (JME, IOS e Android);

b) implementação de `Sprites` de animação, elementos de áudio e efeitos visuais para melhorar aspectos motivacionais de jogos e aplicativos criados;

c) 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;

d) construir uma versão do Furbot com gráficos 3D;

e) implementar a extensão descrita em Maus (2011).

REFERÊNCIAS BIBLIOGRÁFICAS

AQUINO, Juliana F. S. **Plataformas de desenvolvimento para dispositivos móveis**. 2007. 14 f. Monografia (Pós Graduação em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

BERGIN, Joe et al. **Patterns for experiental learning.** [S.l.], 2001. Disponível em: <http://csis.pace.edu/~bergin/patterns/ExperientialLearning.html>. Acesso em: 18 jun. 2012.

CORRÊA, Marco A. **IFURBOT –** migração do framework furbot para a plataforma do iphone. 2009. 59 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Sistemas e Computação, Universidade Regional de Blumenau, Blumenau.

ESTRÁZULAS, Daniel S. **Mobile Furbot:** uma versão do Furbot para a criação de jogos em dispositivos móveis. 2009. 60 f.Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Sistemas e Computação, Universidade Regional de Blumenau, Blumenau.

GONDIM, Haller W. A. S.; AMBRÓSIO, Ana P. **Esboço de fluxogramas no ensino de algoritmos**. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 28., 2008, Belém. **Anais...** Belém: SBC, 2008. p. 1-9.

GOOGLE, **App framework**. Disponível em: <http://developer.android.com/about/versions/index.html>. Acesso em: 26 mar. 2013.

MATTOS, Mauro M.; VAHLDICK, Adilson; HUGO, Marcel. **Apostila de OO e FURBOT**. Blumenau, 2008. Disponível em: <http://www.inf.furb.br/poo/furbot/files/Apostila_FURBOT.pdf >. Acesso em: 26 mar. 2013.

MAUS, Rafael. **Furbot C**: Uma abordagem construtivista para a construção do conhecimento em programação. 2011. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Sistemas e Computação, Universidade Regional de Blumenau, Blumenau.

PHELPS, Andrew M.; EGERT, Christopher A.; BAYLISS, Jessica D. Media impact games in the classroom: using games as a motivator for studying computing: part 1. **IEEE MultiMedia**, [S.l.], v. 12, n. 2, p. 4-8, Apr./June 2009.

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