

**UNIVERSIDADE REGIONAL DE BLUMENAU**  
**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**  
**CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO**

**FURBOT C: UMA ABORDAGEM CONSTRUCIONISTA PARA**  
**A CONSTRUÇÃO DO CONHECIMENTO EM**  
**PROGRAMAÇÃO**

**RAFAEL MAUS**

**BLUMENAU**  
**2011**

**2011/2-23**

**RAFAEL MAUS**

**FURBOT C: UMA ABORDAGEM CONSTRUCIONISTA PARA  
A CONTRUÇÃO DO CONHECIMENTO EM  
PROGRAMAÇÃO**

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 Matos - Orientador

**BLUMENAU  
2011**

**2011/2-23**

**FURBOT C: UMA ABORDAGEM CONSTRUCIONISTA PARA  
A CONSTRUÇÃO DO CONHECIMENTO EM  
PROGRAMAÇÃO**

Por

**RAFAEL MAUS**

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, Orientador - FURB

Membro: \_\_\_\_\_  
Prof. Dalton Solano dos Reis - FURB

Membro: \_\_\_\_\_  
Prof. Roberto Heinzle - FURB

Blumenau, 12 de dezembro de 2011

Dedico este trabalho a meus pais, irmã e a todos os amigos e professores, especialmente aqueles que me ajudaram diretamente na realização deste.

## **AGRADECIMENTOS**

A Deus, pelo seu imenso amor e graça.

À minha família, que muito me apoiou e me incentivou.

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

Para obter algo que você nunca teve, precisa  
fazer algo que nunca fez.

Chico Xavier

## **RESUMO**

O Furbot C 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 é possível gerar o código da inteligência do robô baseado nos comandos que foram executados, podendo ver e utilizar o código de uma possível solução para o exercício.

Palavras-chave: Furbot. Construtivismo. Linguagem Logo.

## **ABSTRACT**

The Furbot C is an extension to the Furbot framework that brings the possibility to the student solve exercises by keyboard controlling the robot. Allowing to move the robot, push, count or remove objects or type something that the robot should say. It is possible to generate the code of the robot intelligence based on the commands that were executed, allowing to see and use the code for a possible solution to the exercise.

Key-words: Furbot. Constructivism. Logo language.



## LISTA DE ILUSTRAÇÕES

Quadro 1 – Exemplo de programação da inteligência do Furbot.....	13
Quadro 2 – Exemplo de mundo em arquivo XML.....	14
Quadro 3 – Carga do arquivo XML.....	15
Figura 1 – Execução do Furbot .....	15
Figura 2 – Diagrama de classes do Furbot .....	16
Figura 3 – Robocode .....	19
Figura 4 – Diagrama de classes do pacote <code>br.furb.furbot.Exception</code> .....	21
Figura 5 – Diagrama de classes do pacote <code>br.furb.furbot.suporte</code> .....	21
Figura 6 – Diagrama de classes do pacote <code>br.furb.furbot</code> .....	22
Quadro 4 – Atributos adicionados a classe Furbot.....	23
Quadro 5 – assinatura dos métodos adicionados/modificados na classe Furbot.....	24
Quadro 6 – corpo do construtor .....	24
Quadro 7 – corpo do método <code>armazenaComando(int keyCode)</code> .....	25
Quadro 8 – corpo do método <code>contarObjetosDoTipo(String tipo, int local)</code> .....	26 e 27
Quadro 9 – corpo do método <code>somarObjetosDoTipo(String tipo, int local)</code> .....	28 e 29
Quadro 10 – corpo do método <code>Boolean ehPossivelEmpurrar(Direcao direcao)</code> .....	29
Quadro 11 – corpo do método <code>verificaDirEAdicionaCmd(Direcao direcao) ...</code> 30,31, 32 e 33	
Quadro 12 – corpo do método <code>somaIndividual(Direcao direcao)</code> .....	33
Quadro 13 – corpo do método <code>contIndividual(String tipo)</code> .....	34
Quadro 14 – corpo de método <code>movimenta(Direcao direcao)</code> .....	34
Quadro 15 – corpo do método <code>addmovimentoVerificado(Direcao direcao)</code> .....	35
Quadro 16 – corpo do método <code>Boolean proxCmdEhIgual(Integer proximo,Integer atual)</code> .....	35
Quadro 17 – Construtor da classe Mundo .....	36
Quadro 18 – Botões adicionados a classe MundoVisual.....	36 e 37
Quadro 19 – Enunciado do Exercício 1 .....	38
Figura 8 – Gabarito do exercício 1 .....	38
Figura 9 – Solução do exercício 1 .....	39
Quadro 20 – Enunciado do Exercício 4.....	39

Figura 10 – Gabarito do exercício 4 .....	40
Figura 11 – Solução do exercício 4 .....	41
Figura 12 – Opções de interação .....	41
Figura 13 – O que deve ser contado .....	41
Quadro 21 – Código gerado para contagem de objetos do tipo tesouro do mundo todo .....	42
Quadro 22 – Código gerado para contagem de objetos do tipo tesouro em colunas pares .....	43
Quadro 23 – Código gerado para contagem de objetos do tipo tesouro em colunas ímpares...	44
Quadro 24 – Código gerado para contagem de objetos do tipo tesouro em linhas pares .....	45
Quadro 25 – Código gerado para contagem de objetos do tipo tesouro em linhas ímpares ...	46
Figura 14 – O que deve ser acumulado .....	46
Quadro 26 – Código gerado para o acúmulo de objetos do tipo tesouro do mundo todo .....	47
Quadro 27 – Código gerado para o acúmulo de objetos do tipo tesouro em colunas pares ....	48
Quadro 28 – Código gerado para o acúmulo de objetos do tipo tesouro em colunas ímpares .	49
Quadro 29 – Código gerado para o acúmulo de objetos do tipo tesouro em linhas pares .....	50
Quadro 30 – Código gerado para o acúmulo de objetos do tipo tesouro em linhas ímpares ..	51
Quadro 31 – Código gerado para empurrar, remover, contar ou somar, informar a posição atual no mundo .....	52
Quadro 32 – corpo do método gerarCodigo() .....	56, 57, 58, 59, 60, 61, 62, 63 e 64
Quadro 33 – Lista de todos os comandos possíveis do atributo cmdsExecutados() .....	65

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>11</b>
1.1 OBJETIVOS DO TRABALHO .....	12
1.2 ESTRUTURA DO TRABALHO .....	12
<b>2 FUNDAMENTAÇÃO TEÓRICA .....</b>	<b>13</b>
2.1 <b>FRAMEWORK FURBOT.....</b>	<b>13</b>
2.2 PARADIGMAS DE APRENDIZAGEM.....	16
2.3 TRABALHOS CORRELATOS.....	18
2.3.1 Robocode .....	18
2.3.2 A linguagem de programação logo .....	19
<b>3 DESENVOLVIMENTO DA EXTENSÃO PARA O FRAMEWORK FURBOT .....</b>	<b>20</b>
3.1 <b>REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO .....</b>	<b>20</b>
3.2 <b>ESPECIFICAÇÃO .....</b>	<b>20</b>
3.3 <b>IMPLEMENTAÇÃO .....</b>	<b>22</b>
3.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS .....	22
3.3.1.1 CLASSE FURBOT .....	23
3.3.1.2 CLASSE MUNDO .....	35
3.3.1.3 CLASSE MUNDOVISUAL .....	36
3.3.1.4 CLASSE FURBOTCDIALOG .....	37
3.4. OPERACIONALIDADE DA IMPLEMENTAÇÃO .....	38
<b>4 RESULTADOS E DISCUSSÃO.....</b>	<b>53</b>
<b>5 CONCLUSÕES .....</b>	<b>54</b>
<b>6 EXTENSÕES .....</b>	<b>55</b>
<b>7 APÊNDICE A.....</b>	<b>56</b>
<b>8 APÊNDICE B.....</b>	<b>65</b>
<b>REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>66</b>

## 1. INTRODUÇÃO

A motivação para o estudo é a peça chave para a aprendizagem e está relacionada a três aspectos principais: conteúdo interessante, ambiente empolgante e didática dos professores, segundo Bergin et al (2001, p. 11).

O construtivismo se propõe a facilitar o processo de aprendizagem através de desafios que motivam o estudante a buscar novos conhecimentos. Segundo Pocrifka et al. (2009, p. 2472), a luz da teoria construtivista, um aprendiz informa ao computador o que deve ser feito. Isso difere da maioria dos softwares educacionais encontrados no mercado, que se constituem em programas prontos, fechados e baseados numa concepção instrucionista de aprendizagem, isso é, por meio de instruções, o programa diz para o aprendiz o que deve ser feito.

Dentre os *frameworks* existentes que buscam 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, p. 2). 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 ciências da computação e sistemas de informação da Universidade Regional de Blumenau (FURB) enfrentam na disciplina de programação de computadores. Conforme Vahldick e Mattos (2009), “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”.

O princípio que norteia o funcionamento 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 Vahldick e Mattos (2009).

Visto o acima, este Trabalho de Conclusão de Curso visa estender o Furbot de modo a possibilitar que a movimentação do robô traduza-se para um código fonte Java. Deste modo, o aprendizado de programação dar-se-á de uma forma inversa (modo construtivista) pela qual o Furbot foi originalmente concebido (modo instrucionista).

## 1.1. OBJETIVOS DO TRABALHO

O objetivo deste trabalho é estender o Furbot para permitir que a programação da inteligência do mesmo seja produzida a partir da movimentação do robô no mundo.

Os objetivos específicos do trabalho são:

- a) disponibilizar uma funcionalidade adicional para permitir coletar os movimentos do robô no mundo e a respectiva produção de código fonte que replique este movimento;
- b) validar a aplicação através de um estudo de caso.

## 1.2. ESTRUTURA DO TRABALHO

O presente trabalho está organizado em três capítulos. No capítulo 2 é descrita a fundamentação teórica que embasou este trabalho.

Na seção 2.1 é descrito o *Framework* Furbot, que é estendido pelo presente trabalho. Na seção 2.2 são apresentados paradigmas de aprendizagem. Na seção 2.3 fala-se a respeito de trabalhos correlatos em duas sub-seções a sub-seção Robocode e a sub-seção Linguagem de programação Logo. O capítulo 3 traz a especificação e implementação deste trabalho, relatando também as técnicas e ferramentas utilizadas, e a operacionalidade da implementação utilizada. O capítulo 4 contém a conclusão do trabalho.

## 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` (VAHLICK; MATTOS, 2009, p. 4). 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{
                //trata a situação em que existe um obstáculo na direção
                do furbot
            }
        }
    }
}
```

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

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

- 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)<sup>1</sup> 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.

```
<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>
    <qtidadeLin>8 </qtidadeLin>
    <qtidadeCol>8 </qtidadeCol>
    <explodir> true</explodir>
  </mundo>
  <robo>
    <x>0</x>
    <y>0</y>
  </robo>
</furbot>
```

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

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).

---

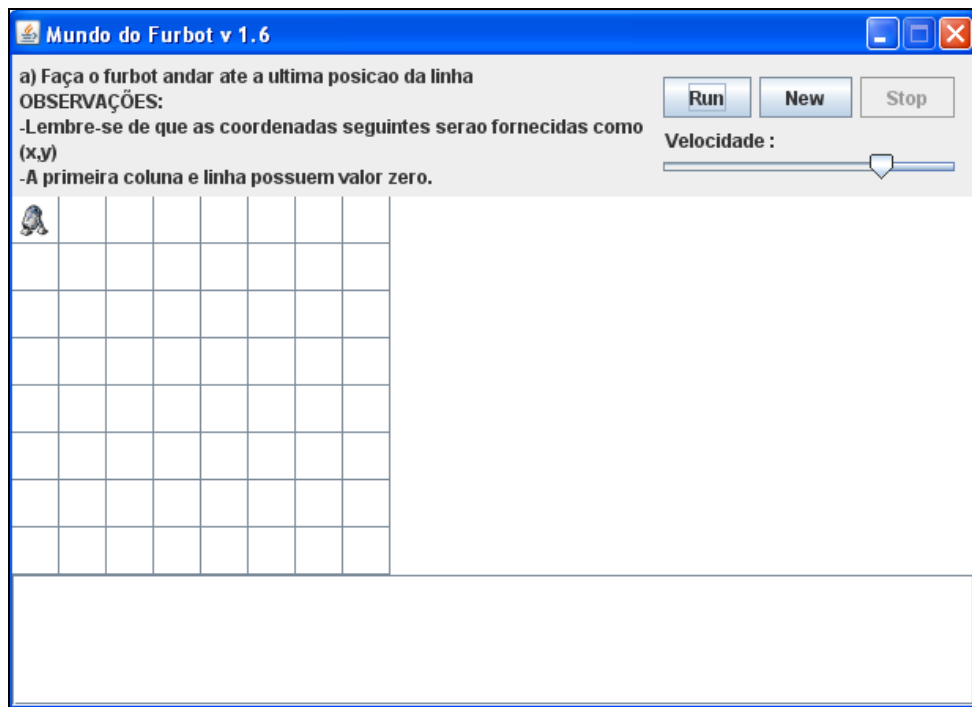
<sup>1</sup> JAR é um formato de arquivo utilizado por aplicações desenvolvidas na linguagem Java.

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

Fonte: Mattos, Vahldick e Hugo (2008), p. 7).

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.



Fonte: Mattos, Vahldick e Hugo (2008, p. 6).

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.

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*.



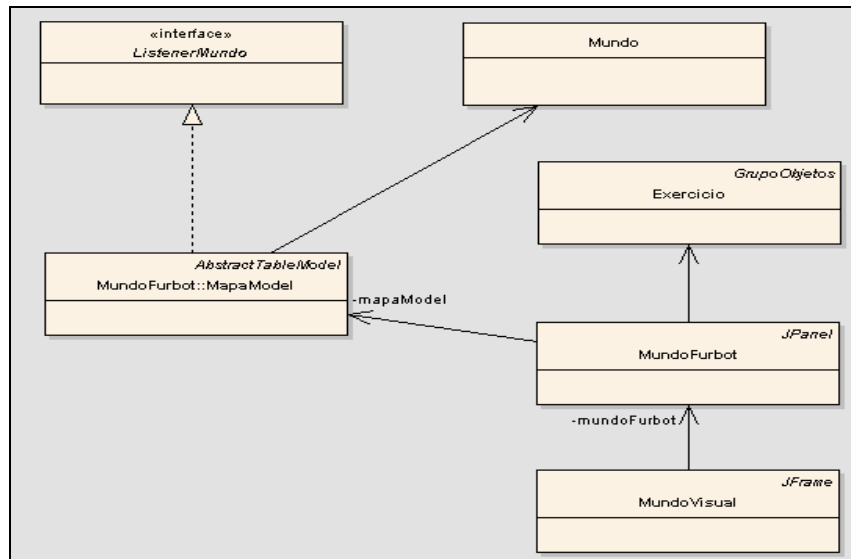


Figura 2 – Diagrama de classes do Furbot

## 2.2. PARADIGMAS DE APRENDIZAGEM

Conforme Morelato et al. (2011, p. 83), o processo de ensino ou aprendizagem apresenta diferentes enfoques que podem incorporar características, de um ou mais, dos paradigmas a seguir: instrucionismo, construtivismo e construcionismo.

O paradigma instrucionista, em sua forma mais ortodoxa (behaviorismo), baseia-se na teoria psicológica de Skinner, segundo a qual, um conhecimento pronto, hierarquizado e compartimentalizado é transferido do professor ao aprendiz, que funciona como um repositório de informações. Os conteúdos são apresentados de acordo com um plano prévio de ensino, definido e organizado pelos professores que, segundo este paradigma, têm mais experiência e capacitação para definir as necessidades de aprendizado de seus alunos. Nesta abordagem, os erros são normalmente punidos e o aprendiz tende a ser passivo no processo. Como o conhecimento resultante é desconectado, tende a manter-se inerte, podendo, posteriormente, vir a ser esquecido (NORMAN; SPOHRER; 1996, SOLOWAY; PRYOR, 1996; VALENTE, 1993).

O construtivismo opõe-se ao paradigma instrucionista. Segundo a teoria construtivista de Piaget, a criança possui um mecanismo de aprendizagem próprio, ou seja, uma estrutura cognitiva individual, antes de ir para a escola. Para Piaget a criança desenvolve sua capacidade intelectual interagindo com objetos do ambiente, sem ensino explícito, baseada na

exploração ativa, onde constrói o conhecimento a partir de um conjunto de problemas motivadores e realistas. Para o construtivismo, o conhecimento é função de como um indivíduo cria os significados a partir de suas experiências e não uma função do que alguém disse que é verdade (PAPERT, 1986, p. 20). O construtivismo defende que os erros ajudam a entender ações e conceitualizações. O aprendiz é ativo no processo.

O construcionismo proposto por Seymour Papert (PAPERT, 1986) é ao mesmo tempo uma teoria de aprendizagem baseada nos princípios de Jean Piaget (conhecimento é adquirido à medida que se pensa e age sobre o objeto maturação, experiência, transmissão social e equilíbrio). É uma estratégia de trabalho onde cada um se torna responsável por sua aprendizagem à medida que experimenta e constrói algo.

O objetivo de Papert ao criar a linguagem Logo foi de oportunizar as crianças a aprender com prazer a programar e assim potencializar a aprendizagem (PAPERT, 1997; SOUZA, 2004). Conforme Pocrifka et al. (2009), a linguagem Logo caracteriza-se como uma linguagem de programação que possibilita a uma criança dar instruções ao computador para que ele execute as ações determinadas por ela.

A linguagem permite um *feedback* imediato para as ações do aprendiz, o que permite ao mesmo comparar suas idéias iniciais com o resultado obtido no programa, analisar e refletir sobre seus acertos e erros, levantar hipóteses, fazer novas tentativas, verificar seus conceitos e, assim, construir novos conceitos (ALMEIDA, 2000; VALENTE, 1993; POCRIFKA et al.; 2009). Nesta dinâmica, o erro tem um papel fundamental, pois quando o programa não realiza a atividade desejada pelo aluno, significa que há algo conceitualmente errado. Então, em vez desta situação tornar-se um fator de punição ou vergonha, ela oportuniza o aprendiz a rever o seu raciocínio, descobrir as falhas existentes e buscar corrigi-las (VALENTE, 1993, p.16).

A principal característica do construcionismo é a utilização da tecnologia como uma ferramenta que auxilie os estudantes a construir seu próprio conhecimento. O computador nesta proposta torna-se um recurso auxiliar na busca de novas informações, procurando priorizar o interesse e estilo cognitivo do aprendiz. Em seguida, estas informações podem ser sistematizadas e transformadas em soluções para uma determinada situação-problema por meio da linguagem Logo. Conforme Almeida (2000, p. 33-34), no construcionismo, o conhecimento não é fornecido ao aluno para que ele dê as respostas. É o aluno que coloca o conhecimento no computador e indica as operações que devem ser executadas para produzir as respostas desejadas. O programa fornece importantes pistas sobre o pensamento do aluno, uma vez que o pensamento está descrito explicitamente e a resposta do computador permite

comparar o previsto com o obtido.

A concepção construcionista exige mudanças no processo educacional, uma vez que há maior ênfase na aprendizagem do que no ensino, na construção do conhecimento e não na instrução.

## 2.3. TRABALHOS CORRELATOS

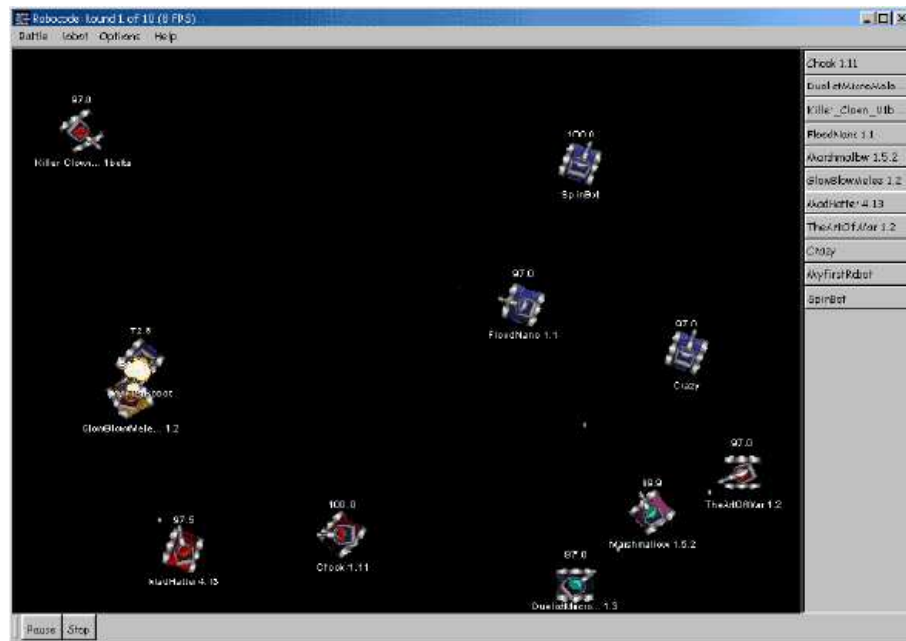
Existem aplicações que possuem características semelhantes ao proposto neste trabalho, tais como o Robocode (ROBOCODE HOME, 2008) e a linguagem Logo (LIMA; LEAL, 2010).

### 2.3.1. Robocode

Conforme Abrantes (2007, p. 41), o RoboCode da IBM (figura 3) é um dos mais interessantes e insólitos ambientes para aprender a programar na linguagem Java. Nada é fornecido ao jogador relacionado com a linguagem em si (ele terá de obter os manuais da linguagem por sua iniciativa), mas apenas o ambiente em que pode programar pequenos tanques de guerra e observar o seu comportamento numa batalha. Existem diversos modelos de comportamento de tanques que servem de base aos primeiros passos do jogador, que ele pode ir modificando conforme a sua vontade. A lógica do jogo é bastante invulgar, pois o jogador, em vez de programar um tanque, deve programar um exército de tanques. Assim, a tentação de controlar diretamente o seu tanque desvanece-se, pois o computador também vai lançar vários dos seus tanques no campo de batalha.

A batalha desenrola-se sem intervenção humana até ao aniquilamento total de um dos exércitos. Outro aspecto muito interessante desta abordagem é que, dada a multiplicidade de atores envolvidos (os tanques), emergem comportamentos imprevistos. O jogador, percebendo as limitações do seu programa pode ir acrescentando e sofisticando comportamentos, à medida que vai jogando. Pode ainda ter acesso ao código fonte dos tanques do computador, sendo livre de modificar o código à sua vontade. Quando os modelos utilizados pelo computador já não o satisfizerem, pode ir à Internet buscar outros modelos de tanques desenvolvidos por outros jogadores. Alguns desses estão abertos (com o código-fonte

disponível), mas muitos apenas são executáveis. Há torneios entre jogadores que permitem testar as competências obtidas (ABRANTES; 2007, p. 42).



Fonte: Abrantes (2007, p. 41).

Figura 3 – Robocode

### 2.3.2. A linguagem de programação logo

Papert e sua equipe do MIT desenvolveram a linguagem de programação Logo na segunda metade da década de 60. Do ponto de vista computacional, Logo foi concebida para ser inteligível e, desta forma, de fácil assimilação para iniciantes. O ambiente de programação Logo tipicamente disponibiliza uma tartaruga (um objeto visual que se desloca no mundo), uma criatura robótica, que é direcionada por meio dos comandos da linguagem (LIMA; LEAL, 2010, p. 2),

Conforme Lima e Leal (2010, p. 3), um dos ambientes que suportam Logo é o SuperLogo, o qual possui uma janela de comandos, que é uma área de interação para o usuário e usada para a escrita de comandos usando a linguagem. Há também uma janela gráfica que representa o mundo virtual por onde a tartaruga pode andar.

### 3. DESENVOLVIMENTO DA EXTENSÃO PARA O *FRAMEWORK* FURBOT

Este capítulo está dividido em três seções, na seção 3.1 são apresentados os requisitos levantados. Na seção 3.2 são demonstrados os diagramas de classe e na seção 3.3 é descrita toda a implementação realizada neste trabalho.

#### 3.1. REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A extensão proposta ao Furbot deve:

- a) possibilitar a resolução de problemas de programação no Furbot visualmente (Requisito Funcional - RF);
- b) permitir que o aluno possa resolver o exercício passo a passo e em tempo real, sem a necessidade de programar a inteligência do robô (RF);
- c) ao finalizar a solução dar a opção de gerar o código com a inteligência do robô (RF);
- d) executar em dispositivos que utilizem máquina virtual Java (Requisito Não-Funcional - RNF);
- e) implementar a extensão utilizando a linguagem de programação Java (RNF).

#### 3.2. ESPECIFICAÇÃO

O *framework* Furbot não sofreu mudanças significativas em sua estrutura de classes. A alteração mais profunda ocorreu na lógica de funcionamento como poderá ser visto nas seções seguintes. Os diagramas foram criados utilizando-se a ferramenta Enterprise Architect 9.1.910 Trial. Na Figura 4 é mostrado o diagrama de classes do pacote br.furb.furbot.exception, o que não sofreu modificações.

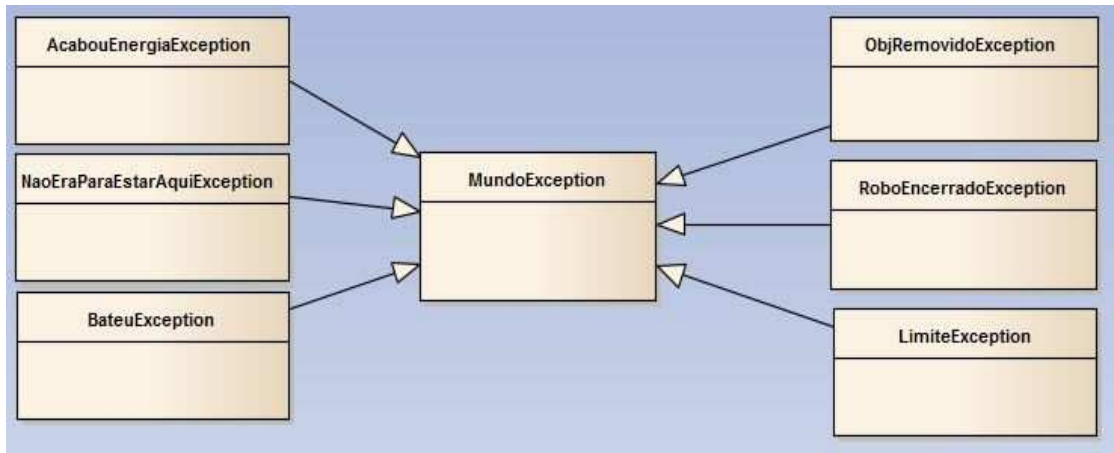


Figura 4 - Diagrama de classes do pacote `br.furb.furbot.Exception`

Na Figura 5 é mostrado o diagrama de classes do pacote `br.furb.furbot.suporte`, que sofreu modificações na classe `Mundo`, destacada em azul.

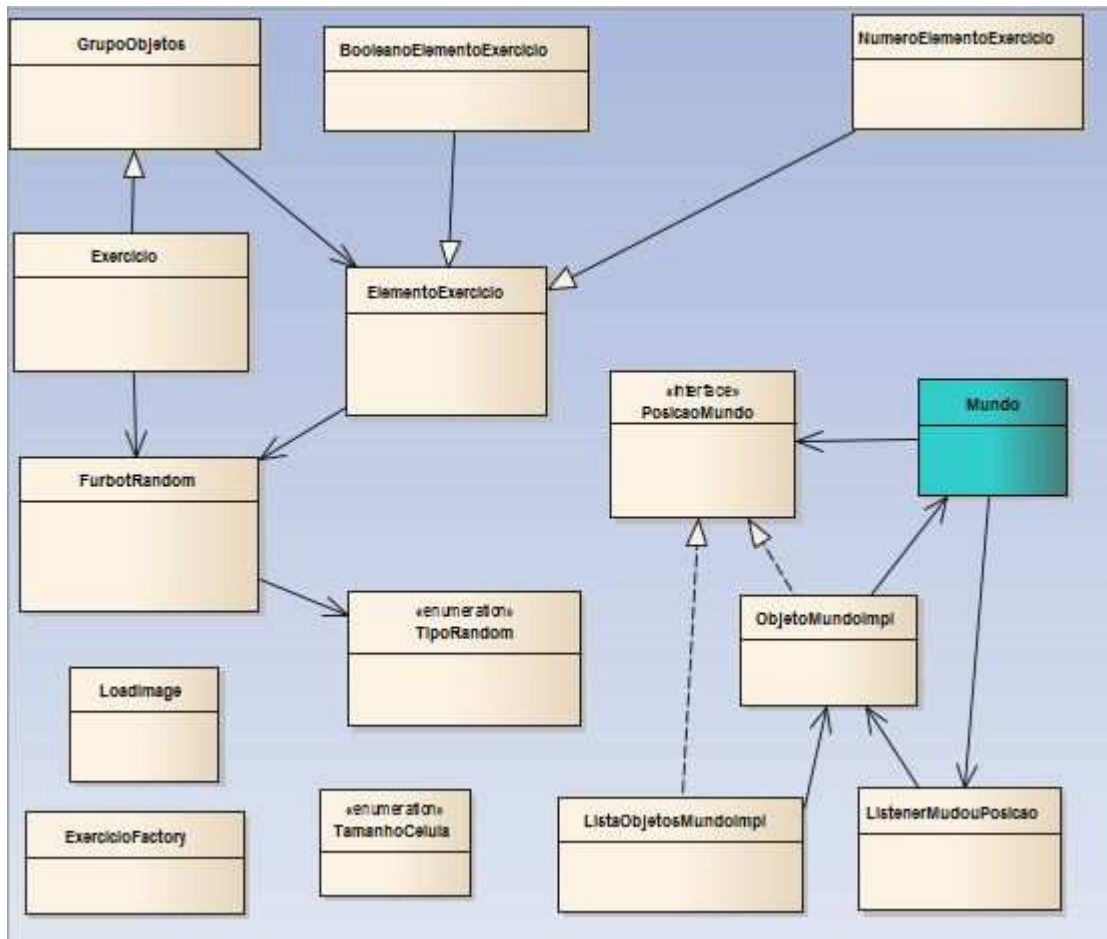


Figura 5 - Diagrama de classes do pacote `br.furb.furbot.suporte`

Na Figura 6 é mostrado o diagrama de classes do pacote `br.furb.furbot`, que sofreu modificações na classe `Furbot` e na classe `MundoVisual`, destacadas em verde. Foi adicionada a classe `FurbotCDialog` que tem o propósito de apresentar o código fonte gerado,

destacada em amarelo.

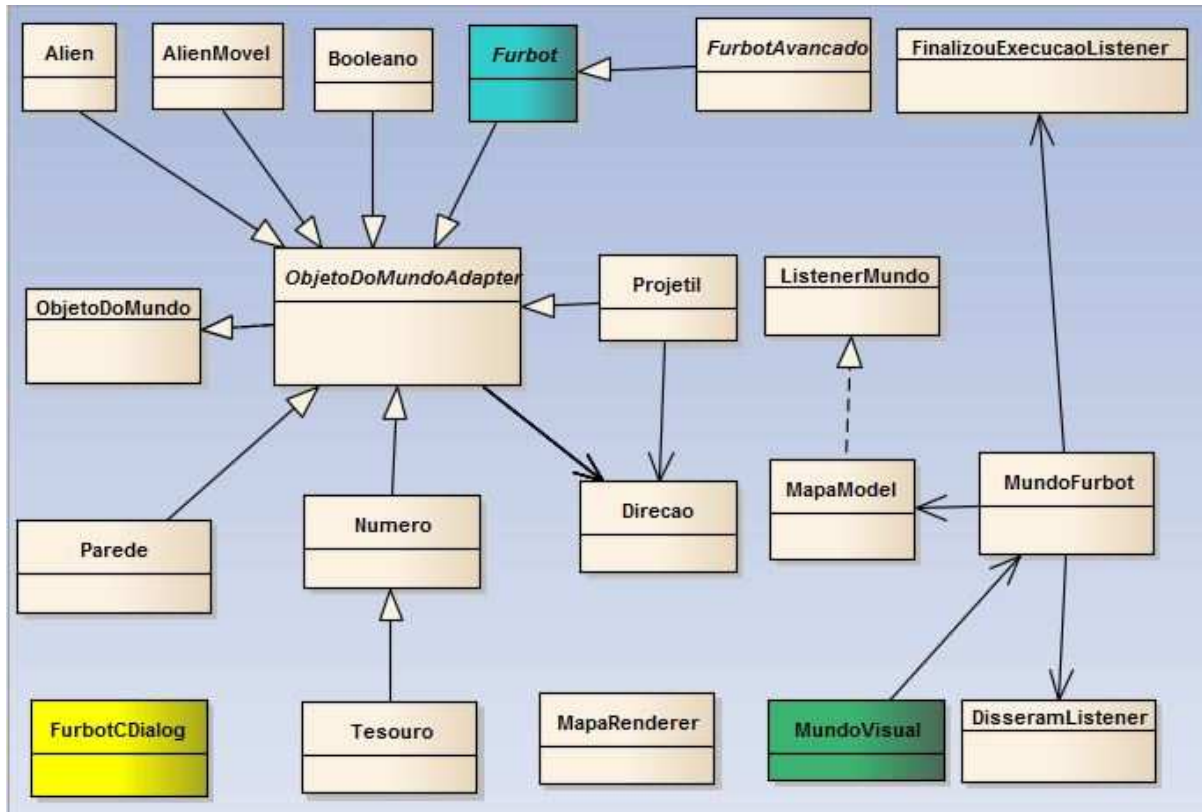


Figura 6 - Diagrama de classes do pacote br . furb . furbot

### 3.3. IMPLEMENTAÇÃO

A seguir são mostradas as técnicas, ferramentas utilizadas e a operacionalidade da implementação.

#### 3.3.1. Técnicas e ferramentas utilizadas

A extensão proposta neste trabalho foi implementada no projeto do Furbot em Java utilizando o NetBeans IDE 7.0.1. Serão descritas abaixo todas as funcionalidades e modificações implementadas no projeto.

### 3.3.1.1 Classe Furbot

A classe Furbot foi a classe que sofreu o maior número de alterações. No Quadro 4 são listados os atributos e em seguida é feita a descrição de cada um deles.

```
public static List<String> cmdsExecutados = new ArrayList<String>();
public static Boolean modoAjuda = false;
private static HashMap nomeTiposCont = new HashMap();
private static ArrayList qtdTiposCont = new ArrayList<String>();
private static HashMap nomeTiposSoma = new HashMap();
private static ArrayList qtdTiposSoma = new ArrayList<String>();
private static int qtdLinhas = 0;
private static int qtdColunas = 0;
```

Quadro 4 - Atributos adicionados a classe Furbot

- a) o atributo `cmdsExecutados` é uma lista que guarda todos os comandos que o aluno realizou ao fazer o robô executar. Todos os possíveis comandos que podem aparecer na lista estão descritos no Apêndice B quadro 33;
- b) o atributo `modoAjuda` determina se o exercício está sendo executado no modo de ajuda;
- c) o atributo `nomesTiposCont` é uma lista que guarda que tipo de objeto está sendo contado individualmente no exercício atual (Aliens, Tesouro, Parede etc.);
- d) o atributo `qtdTiposCont` é uma lista que guarda a quantidade de tipos de objetos que estão sendo contados individualmente no exercício atual;
- e) o atributo `nomesTiposSoma`: é uma lista que guarda quais tipos de objetos estão sendo somados individualmente no exercício atual;
- f) o atributo `qtdTiposSoma`: é uma lista que guarda a quantidade de tipos de objetos que estão sendo somados individualmente no exercício atual;
- g) o atributo `qtdLinhas`: quantidade de linhas no mundo atual;
- h) o atributo `qtdColunas`: quantidade de colunas no mundo atual.

No Quadro 5 está listado o construtor que recebeu uma pequena modificação e os métodos que foram adicionados.



```

public Furbot()
public final void armazenaComando(int keyCode)
public final void contarObjetosDoTipo(String tipo, int
local)
public final void somarObjetosDoTipo(String tipo, int
local)
public final Boolean ehPossivelEmpurrar(Direcao direcao)
public final void verificaDirEAdicionaCmd(Direcao
direcao)
public final void somaIndividual(Direcao direcao)
public final void contIndividual(String tipo)
public final void movimenta(Direcao direcao)
public final void addmovimentoVerificado(Direcao
direcao)
public static final void gerarCodigo()
public static Boolean proxCmdEhIgual(Integer proximo,
Integer atual)

```

Quadro 5 - assinatura dos métodos adicionados/modificados na classe Furbot

O método construtor da classe `Furbot` associa valores para os atributos `qtdLinhas` e `qtdColunas`, sendo estes a quantidade de linhas e colunas no mundo do exercício atual conforme Quadro 6.

```

public Furbot() {
    Furbot.qtdLinhas =
MundoVisual.getAtributo("QtdLinhas");
    Furbot.qtdColunas =
MundoVisual.getAtributo("QtdColunas");
}

```

Quadro 6 - corpo do construtor

No método `armazenaComando(int keyCode)` (Quadro 7) o parâmetro `keyCode` é analisado, este parâmetro contém o código da tecla pressionada, e de acordo com a análise um comando é executado e o mesmo é adicionado a lista `cmdsExecutados`.

```

public final void armazenaComando(int keyCode) {
    switch (keyCode) {
        case TECLAESQUERDA: {
            verificaDirEAdicionaCmd(ESQUERDA);
        }
        break;
        case TECLACIMA: {
            verificaDirEAdicionaCmd(ACIMA);
        }
        break;
        case TECLADIREITA: {
            verificaDirEAdicionaCmd(DIREITA);
        }
        break;
        case TECLABAIXO: {
            verificaDirEAdicionaCmd(ABAIXO);
        }
        break;
        case TECLA_D: {
            String frase =
                JOptionPane.showInputDialog(
                    null,
                    "O que devo dizer?: ",
                    "Escreva algo", 1);
            if (frase != null) {
                cmdsExecutados.add("9," + frase);
                this.diga("Aqui sera dito: " + frase);
            }
        }
        break;
        case TECLA_P: {
            cmdsExecutados.add("10");
            this.diga("Aqui sera informada a posição");
        }
        break;
    }
}

```

Quadro 7 - corpo do método armazenaComando(int keyCode)

O método `contarObjetosDoTipo(String tipo, int local)` (Quadro 8), é um método executado durante a solução do exercício, serve apenas para fazer a contagem de determinado objeto e mostrar o valor enquanto o exercício está sendo resolvido pelo aluno. O parâmetro `tipo` serve para determinar qual tipo de objeto esta sendo contado, o parâmetro `local` serve para determinar em quais locais devem ser contados os objetos do tipo encontrado, este parâmetro pode ter o valor 0 para contar todos os objetos, 1 para contar somente os objetos de colunas pares, 2 para colunas impares, 3 para linhas pares ou 4 para linhas impares.

```

public final void contarObjetosDoTipo(String tipo, int local) {
    int cont = 0;
    cmdsExecutados.add("18," + tipo + "," + local);
    this.diga("Contando objetos do tipo " + tipo + "...");
    for (int mY = 0; mY <= Furbot.qtdLinhas; mY++) {
        for (int mX = 0; mX <= Furbot.qtdColunas; mX++) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if (objeto.getSouDoTipo().equals(tipo)) {
                    switch (local) {
                        case 0: { // Todos
                            cont++;
                        }
                        break;
                        case 1: { // Colunas pares
                            if (mX % 2 == 0) {
                                cont++;
                            }
                        }
                        break;
                        case 2: { // colunas impares
                            if (mX % 2 != 0) {
                                cont++;
                            }
                        }
                        break;
                        case 3: { // Linhas pares
                            if (mY % 2 == 0) {
                                cont++;
                            }
                        }
                        break;
                        case 4: { // Linhas impares
                            if (mY % 2 != 0) {
                                cont++;
                            }
                        }
                        break;
                    }
                }
            }
        }
    }
    switch (local) {
        case 0: { // Todos
            this.diga("Foram encontrados " + cont + " objetos do tipo "
+ tipo);
        }
        break;
        case 1: { // Colunas pares
            this.diga("Foram encontrados " + cont + " objetos do tipo "
+ tipo + " em colunas pares");
        }
        break;
        case 2: { // Colunas impares
            this.diga("Foram encontrados " + cont + " objetos do tipo "
+ tipo + " em colunas impares");
        }
        break;
        case 3: { // Linhas pares
            this.diga("Foram encontrados " + cont + " objetos do tipo "

```

```
+ tipo + " em linhas pares");
    }
    break;
    case 4: { // Linhas impares
        this.diga("Foram encontrados " + cont + " objetos do tipo "
+ tipo + " em linhas impares");
        }
        break;
    }
}
```

Quadro 8 - corpo do método contarObjetosDoTipo(String tipo, int local)

O método `somarObjetosDoTipo(String tipo, int local)` (Quadro 9), é um método executado durante a solução do exercício, serve apenas para fazer a soma de determinado objeto e mostrar o valor enquanto o exercício está sendo resolvido pelo aluno. O parâmetro `tipo` serve para determinar qual tipo de objeto esta sendo somado, o parâmetro `local` serve para determinar em quais locais devem ser somados os objetos do tipo encontrado, este parâmetro pode ter o valor 0 para somar todos os objetos, 1 para somar somente os objetos de colunas pares, 2 para colunas impares, 3 para linhas pares ou 4 para linhas impares.

```

public final void somarObjetosDoTipo(String tipo, int local) {
    int soma = 0;
    cmdsExecutados.add("19," + tipo + "," + local);
    this.diga("Somaando objetos do tipo " + tipo + "...");
    for (int mY = 0; mY <= Furbot.qtdLinhas; mY++) {
        for (int mX = 0; mX <= Furbot.qtdColunas; mX++) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if (objeto.getSouDoTipo().equals(tipo)) {
                    switch (local) {
                        case 0: { // Todos
                            soma +=
Integer.parseInt(objeto.toString());
                            }
                            break;
                        case 1: { // Colunas pares
                            if (mX % 2 == 0) {
                                soma +=
Integer.parseInt(objeto.toString());
                            }
                            }
                            break;
                        case 2: { // colunas impares
                            if (mX % 2 != 0) {
                                soma +=
Integer.parseInt(objeto.toString());
                            }
                            }
                            break;
                        case 3: { // Linhas pares
                            if (mY % 2 == 0) {
                                soma +=
Integer.parseInt(objeto.toString());
                            }
                            }
                            break;
                        case 4: { // Linhas impares
                            if (mY % 2 != 0) {
                                soma +=
Integer.parseInt(objeto.toString());
                            }
                            }
                            break;
                    }
                }
            }
        }
    }
    switch (local) {
        case 0: { // Todos
            this.diga("A soma dos objetos do tipo " + tipo + " é de " +
soma);
        }
        break;
        case 1: { // Colunas pares
            this.diga("A soma dos objetos do tipo " + tipo + " em
colunas pares é de " + soma);
        }
        break;
        case 2: { // Colunas impares
            this.diga("A soma dos objetos do tipo " + tipo + " em

```

```

colunas impares é de " + soma);
    }
    break;
    case 3: { // Linhas pares
        this.diga("A soma dos objetos do tipo " + tipo + " em
linhas pares é de " + soma);
    }
    break;
    case 4: { // Linhas impares
        this.diga("A soma dos objetos do tipo " + tipo + " em
linhas impares é de " + soma);
    }
    break;
}
}
}

```

Quadro 9 - corpo do método somarObjetosDoTipo(String tipo, int local)

O método Boolean ehPossivelEmpurrar(Direcao direcao) (Quadro 10), é um método executado durante a solução do exercício, verifica se a célula para qual um objeto está sendo empurrado pode conter este objeto, a célula não pode ser o fim do mundo ou ter uma parede, caso contrário o objeto pode ser empurrado naquela direção, então o comando incluso na lista de cmdsExecutados.

```

public final Boolean ehPossivelEmpurrar(Direcao direcao) {
    // Não pode ser empurrado para o fim
    if (this.getObjeto(direcao).ehFim(direcao)) {
        return false;
    }
    //Uma parede não pode ser empurrada
    if (this.getObjeto(direcao).getSouDoTipo().equals("Parede")) {
        return false;
    }
    //Se não houver nada pode empurrar
    ObjetoDoMundoAdapter          objeto =
this.getObjeto(direcao).getObjeto(direcao);
    if (objeto == null) {
        return true;
    }
    // Não pode ser empurrado para uma parede
    if (objeto.getSouDoTipo().equals("Parede")) {
        return false;
    }

    return true;
}

```

Quadro 10 - corpo do método Boolean ehPossivelEmpurrar(Direcao direcao)

O método verificaDirEAdicionaCmd(Direcao direcao) (Quadro 11), é executado durante a solução do exercício, sempre que o aluno manda o robô se movimentar utilizando o teclado, este método é chamado passando como parâmetro a direção para a qual o robô será movido, se não houver nada na célula de destino o robô é movimentado para a mesma e um comando é adicionado a lista cmdsExecutados, caso contrário é mostrado um diálogo perguntando o que deve ser feito com o objeto encontrado, conseqüentemente o

comando referente a ação escolhida é adicionado a lista `cmdsExecutados`, seguido do comando de movimentação.

```

public final void verificaDirEAdicionaCmd(Direcao direcao) {
    if (ehFim(direcao)) {
        addmovimentoVerificado(direcao);
    } else if (ehVazio(direcao)) {
        switch (direcao) {
            case ESQUERDA: {
                cmdsExecutados.add("1");
                andarEsquerda();
                this.diga("Movimento para esquerda");
            }
            break;
            case ACIMA: {
                cmdsExecutados.add("2");
                andarAcima();
                this.diga("Movimento para cima");
            }
            break;
            case DIREITA: {
                cmdsExecutados.add("3");
                andarDireita();
                this.diga("Movimento para direita");
            }
            break;
            case ABAIXO: {
                cmdsExecutados.add("4");
                andarAbaixo();
                this.diga("Movimento para baixo");
            }
            break;
        }
    } else {
        String[] opcoes = {"Empurrar", "Remover", "Contar", "Acumular",
"Nada"};
        int opcao = JOptionPane.showOptionDialog(
            null, "O que deve ser feito com este objeto?",
            "Pergunta", JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null,
            opcoes, "Furbot");

        switch (opcao) {
            case 0: { //Empurrar
                if (ehPossivelEmpurrar(direcao)) {
                    switch (direcao) {
                        case ESQUERDA: {
                            cmdsExecutados.add("11");
                            this.getObjeto(direcao).andarEsquerda();
                        }
                        break;
                        case ACIMA: {
                            cmdsExecutados.add("12");
                            this.getObjeto(direcao).andarAcima();
                        }
                        break;
                        case DIREITA: {
                            cmdsExecutados.add("13");
                            this.getObjeto(direcao).andarDireita();
                        }
                        break;
                        case ABAIXO: {

```

```

                                cmdsExecutados.add("14");
                                this.getObjeto(direcao).andarAbaixo();
                                }
                                break;
                                }
                                diga("Objeto empurrado");
                                movimenta(direcao);
                                } else {
                                this.diga("Não é possível empurrar este objeto");
                                }
                                }
                                break;
                                case 1: { //Remover
                                if
(!((this.getObjeto(direcao).getSouDoTipo().equals("Parede")))) {
                                cmdsExecutados.add("15," + direcao);
                                removerObjetoDoMundo(this.getObjeto(direcao));
                                diga("Objeto removido");
                                movimenta(direcao);
                                } else {
                                this.diga("Não é possível remover este objeto");
                                }
                                }
                                break;
                                case 2: { // Contar
                                String tipo = this.getObjeto(direcao).getSouDoTipo();
                                String[] opcont = {"Todos", "Colunas pares", "Colunas
Impares", "Linhas pares", "Linhas impares", "Somente este"};
                                int opscont = JOptionPane.showOptionDialog(
                                null, "Quais objetos deste tipo devem ser
contados?",
                                "Pergunta",
                                JOptionPane.YES_NO_OPTION,
                                JOptionPane.QUESTION_MESSAGE, null, opcont, "Furbot");
                                switch (opscont) {
                                case 0: {
                                this.contarObjetosDoTipo(tipo, 0);
                                }
                                break;
                                case 1: {
                                this.contarObjetosDoTipo(tipo, 1);
                                }
                                break;
                                case 2: {
                                this.contarObjetosDoTipo(tipo, 2);
                                }
                                break;
                                case 3: {
                                this.contarObjetosDoTipo(tipo, 3);
                                }
                                break;
                                case 4: {
                                this.contarObjetosDoTipo(tipo, 4);
                                }
                                break;
                                case 5: {
                                contIndividual(tipo);
                                }
                                break;
                                }
                                if (!ehFim(direcao)) {
                                if
(!((this.getObjeto(direcao).getSouDoTipo().equals("Parede")))) {

```



```

        movimenta(direcao);
        addmovimentoVerificado(direcao);
    }
}
break;
case 3: {
    try {
        String                tipo                =
this.getObjeto(direcao).getSouDoTipo();
        String[] opsoma = {"Todos", "Colunas pares",
"Colunas Impares", "Linhas pares", "Linhas impares", "Somente este"};
        int opssoma = JOptionPane.showOptionDialog(
            null, "Quais objetos deste tipo devem ser
contados?",
            "Pergunta",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE, null, opsoma, "Furbot");
        switch (opssoma) {
            case 0: {
                this.somarObjetosDoTipo(tipo, 0);
            }
            break;
            case 1: {
                this.somarObjetosDoTipo(tipo, 1);
            }
            break;
            case 2: {
                this.somarObjetosDoTipo(tipo, 2);
            }
            break;
            case 3: {
                this.somarObjetosDoTipo(tipo, 3);
            }
            break;
            case 4: {
                this.somarObjetosDoTipo(tipo, 4);
            }
            break;
            case 5: {
                this.somaIndividual(direcao);
            }
            break;
        }
    } catch (Exception e) {
        this.diga("Este objeto não possui valor para ser
somado");
    }
    if (!ehFim(direcao)) {
        if
(!((this.getObjeto(direcao).getSouDoTipo().equals("Parede")))) {
            movimenta(direcao);
            addmovimentoVerificado(direcao);
        }
    }
}
break;
case 4: {
    if (!ehFim(direcao)) {
        if
(!((this.getObjeto(direcao).getSouDoTipo().equals("Parede")))) {
            movimenta(direcao);
            addmovimentoVerificado(direcao);
        }
    }
}

```

```

    }
    }
    }
    break;
}
}
}
}

```

Quadro 11 - corpo do método `verificaDirEAdicionaCmd(Direcao direcao)`

O método `somaIndividual(Direcao direcao)` (Quadro 12), é executado durante a solução do exercício, somente quando o aluno movimenta o robô em direção a algum objeto e opta por acumular e em seguida clica no botão `Somente este`, ele lista a soma deste tipo de objeto e adiciona o respectivo comando a lista `cmdsExecutados`.

```

public final void somaIndividual(Direcao direcao) {
    int valor = Integer.parseInt(this.getObjeto(direcao).toString());
    String tipo = this.getObjeto(direcao).getSouDoTipo();
    cmdsExecutados.add("17," + tipo + "," + direcao);
    if (qtdTiposSoma.contains(tipo)) {
        nomeTiposSoma.put(tipo,
            ((Integer.parseInt(nomeTiposSoma.get(tipo).toString()) + valor)));
    } else {
        qtdTiposSoma.add(tipo);
        nomeTiposSoma.put(tipo, valor);
    }
    this.diga("--- Soma de valores de objetos ---");
    for (int i = 0; i < qtdTiposSoma.size(); i++) {
        diga(qtdTiposSoma.get(i) + ":" + nomeTiposSoma.get(qtdTiposSoma.get(i)));
    }
}

```

Quadro 12 - corpo do método `somaIndividual(Direcao direcao)`

O método `contIndividual(String tipo)` (Quadro 13), é executado durante a solução do exercício, somente quando o aluno movimenta o robô em direção a algum objeto e opta por acumular e em seguida clica no botão `Somente este`, ele lista a contagem deste tipo de objeto e adiciona o respectivo comando a lista `cmdsExecutados`.

```

public final void contIndividual(String tipo) {
    cmdsExecutados.add("l6," + tipo);
    if (nomeTiposCont.containsKey(tipo)) {
        nomeTiposCont.put(
            tipo,
            (Integer.parseInt(
                nomeTiposCont.get(tipo).toString()) + 1));
    } else {
        nomeTiposCont.put(tipo, 1);
        qtdTiposCont.add(tipo);
    }
    diga("Contar objeto do tipo " + tipo);
    diga("--- Contagens Individuais---");
    for (int i = 0; i < qtdTiposCont.size(); i++) {
        diga(qtdTiposCont.get(i) + " : " +
nomeTiposCont.get(qtdTiposCont.get(i)));
    }
}
}

```

Quadro 13 - corpo do método contIndividual(String tipo)

O método movimentar(Direcao direcao)(quadro14), é executado durante a solução do exercício e apenas movimentar o robô na direção passada como parâmetro.

```

public final void movimentar(Direcao direcao) {
    switch (direcao) {
        case ESQUERDA: {
            this.andarEsquerda();
            this.diga("Movimento para esquerda");
        }
        break;
        case ACIMA: {
            this.andarAcima();
            this.diga("Movimento para cima");
        }
        break;
        case DIREITA: {
            this.andarDireita();
            this.diga("Movimento para direita");
        }
        break;
        case ABAIXO: {
            this.andarAbaixo();
            this.diga("Movimento para baixo");
        }
        break;
    }
}
}

```

Quadro 14- corpo de método movimentar(Direcao direcao)

O método adicionarMovimentoVerificado(Direcao direcao)(quadro15), é executado durante a solução do exercício e adiciona o respectivo comando, com verificação da célula de destino a lista cmdsExecutados de acordo com a direcao passada como parâmetro.

```

public final void addmovimentoVerificado(Direcao direcao) {
    switch (direcao) {
        case ESQUERDA:
            cmdsExecutados.add("5");
            break;
        case ACIMA:
            cmdsExecutados.add("6");
            break;
        case DIREITA:
            cmdsExecutados.add("7");
            break;
        case ABAIXO:
            cmdsExecutados.add("8");
            break;
    }
}

```

Quadro 15 - corpo do método addmovimentoVerificado(Direcao direcao)

O método `Boolean proxCmdEhIgual(Integer proximo, Integer atual)` (quadro16). Somente é chamado dentro do método `gerarCodigo()` e serve para verificar se existem comandos de movimentação iguais e em seqüência na lista `cmdsExecutados`, desta forma podem ser criados blocos de repetição.

```

public static Boolean proxCmdEhIgual(Integer proximo, Integer atual) {
    try {
        int valor = Integer.parseInt(cmdsExecutados.get(proximo));
        if (valor == atual) {
            return true;
        } else {
            return false;
        }
    } catch (Exception e) {
        return false;
    }
}

```

Quadro 16 - corpo do método `Boolean proxCmdEhIgual(Integer proximo, Integer atual)`

### 3.3.1.2 Classe Mundo

Na classe `Mundo` foi alterado o construtor para que os parâmetros de quantidade de linhas e colunas no mundo sejam passadas ao `MundoVisual` para que possam ser lidas na classe `Furbot`. As Linhas adicionadas estão destacadas em vermelho no Quadro 17.

```

public Mundo(int qtdadeLin, int qtdadeCol)
{
    imgExplosao = LoadImage.getInstance().getIcon("imagens/explosao.png");
    listeners = new ArrayList<ListenerMundo>();
    listenersMudouPosicao = new ArrayList<ListenerMudouPosicao>();
    modelo = new PosicaoMundo[qtdadeLin][qtdadeCol];
    this.qtdadeLin = qtdadeLin;
    this.qtdadeCol = qtdadeCol;
    MundoVisual.setAtributo("QtdLinhas", qtdadeLin);
    MundoVisual.setAtributo("QtdColunas", qtdadeCol);
}

```

Quadro 17 - Construtor da classe Mundo

### 3.3.1.3 Classe MundoVisual

Na classe `MundoVisual` foram adicionados dois botões e suas respectivas funcionalidades, o código adicionado pode ser visto no Quadro 18.

```

        jbAjuda = new JButton("Me ajude");
        jpBotoes.add(jbAjuda);
        jbAjuda.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                String info = "Modo de ajuda.\n\n" +
                    "Com ajuda você pode
controlar o\n" +
                    "seus comandos\n" +
                    "posteriormente \n" +
                    "controlar o robo.\n" +
                    "algum objeto, é possível\n" +
                    "ou acumular o valor do\n" +
                    "algo que deve ser dito.\n" +
                    "informe sua posição.\n" +
                    "Stop.\n" +
                    "gerar o codigo\n" +
                    "você fez o robo\n" +
                    info);
                ArrayList<String>() = new
                    "robo livremente, todos os
                    "serão armazenados, para
                    "gerar o código.\n" +
                    "Comandos: \n\n" +
                    " - Setas direcionais para
                    " Ao se mover na direção de
                    " empurrar, remover, contar
                    " objeto.\n" +
                    " - Tecla D para escrever
                    " - Tecla P para que o robo
                    "Ao finalizar clique no botão
                    "Então você tera a opção de
                    "baseado nos comandos que
                    "executar.";
                JOptionPane.showMessageDialog(null,
                    Furbot.cmdsExecutados = new
                    slider.setValue(10);

```

```
Furbot.modaAjuda = true;
                mundoAtual = 0;
executarProxMundo = false;
novaSequencia(exercicio);
executar(exercicio);
                jbCodigo.setEnabled(false);
    }
});
                jbCodigo = new JButton("Gerar Código");
                jbCodigo.setEnabled(false);
jpBotoes.add(jbCodigo);
jbCodigo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        Furbot.gerarCodigo();
    }
});
```

Quadro 18 - Botões adicionados a classe MundoVisual

#### 3.3.1.4 Classe FurbotCDialog

A classe FurbotCDialog refere-se ao diálogo mostrado ao clicar no botão Gerar Código após a finalização da possível solução de um exercício. Este diálogo pode ser visualizado na Figura 7.



Figura 7 - Diálogo gerado pelo botão Gerar Código

### 3.4. OPERACIONALIDADE DA IMPLEMENTAÇÃO

Para demonstrar todas as funcionalidades agregadas pela extensão ao Furbot implementada neste trabalho, a seguir apresenta-se a solução de alguns exemplos. Em cada exercício o aluno deverá, clicar no botão `Me Ajude`, comandar o robô pelo teclado, ao finalizar o exercício clicar no botão `Stop` e em seguida no botão `Gerar Código`.

No Quadro 19 está listado o enunciado do primeiro exercício.

```
<enunciado>
  Exercicio 1. <br> Faca o robo andar ate a ultima posicao da linha. <br>
                Lembre-se de que as coordenadas sempre
serao fornecidas como (x, y).<br> A primeira coluna e linhas sao a de
numero ZERO.
</enunciado>
```

Fonte: Mattos, Vahldick e Hugo (2008).

Quadro 19 - Enunciado do Exercício 1

Na Figura 8 tem-se o gabarito do exercício.

```
 diga("exercicio1");
  andarDireita();
  andarDireita();
  andarDireita();
  andarDireita();
  andarDireita();
  andarDireita();
  andarDireita();
  andarDireita();
```

Fonte: Mattos, Vahldick e Hugo (2008).

Figura 8 - Gabarito do exercício 1

Na Figura 9 é mostrada a solução obtida. Para tanto fez-se necessário pressionar a tecla "D", digitar a frase "Exercício 1", clicar em "Ok" e em seguida, pressionar a tecla direcional direita 8 vezes para obter a solução apresentada. Pode se observar que o código gerado será produzido de maneira otimizada sempre que for possível.

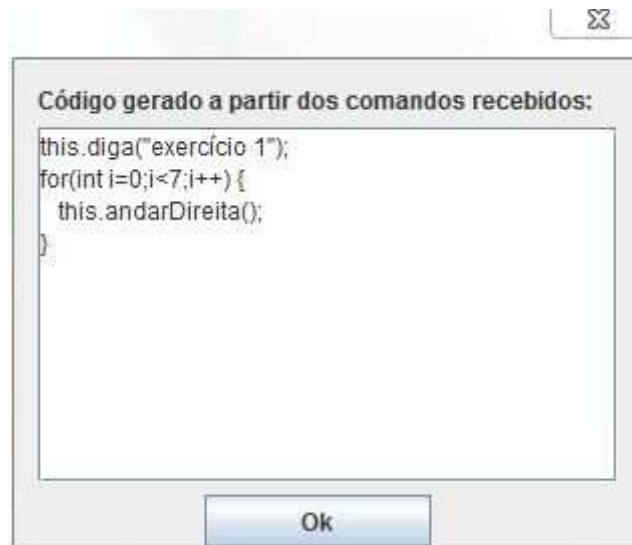


Figura 9 - Solução do exercício 1

No Quadro 20 está listado o enunciado do segundo exercício.

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

Fonte: Mattos, Vahldick e Hugo (2008)

Quadro 20 - Enunciado do Exercício 4

Na Figura 10 tem-se o gabarito do exercício.



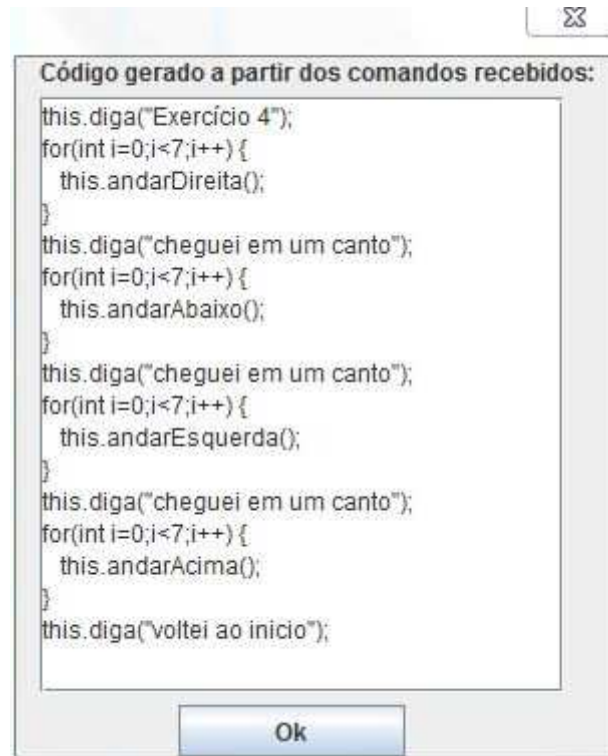
```
 diga("exercicio4");
 while (!ehFim(DIREITA)) {
   andarDireita();
 };
 diga ("cheguei num canto");
 while (!ehFim(ABAIXO)) {
   andarAbaixo();
 };
 diga ("cheguei num canto");
 while (!ehFim(ESQUERDA)) {
   andarEsquerda();
 };
 diga ("cheguei num canto");

 while (!ehFim(ACIMA)) {
   andarAcima();
 };
 diga ("retornei ao inicio");
```

Fonte: Mattos, Vahldick e Hugo (2008)

Figura 10 - Gabarito do exercício 4

Na Figura 11 é mostrada a solução obtida. Foi necessário pressionar a tecla "D", digitar a frase "Exercício 4" e clicar em "Ok". Em seguida pressionar 8 vezes cada tecla direcional na seqüência direita, abaixo, esquerda e acima, depois a tecla "D" e digitar a frase "Voltei ao início" para obter a solução apresentada.



```

Código gerado a partir dos comandos recebidos:
this.diga("Exercício 4");
for(int i=0;i<7;i++){
  this.andarDireita();
}
this.diga("cheguei em um canto");
for(int i=0;i<7;i++){
  this.andarAbaixo();
}
this.diga("cheguei em um canto");
for(int i=0;i<7;i++){
  this.andarEsquerda();
}
this.diga("cheguei em um canto");
for(int i=0;i<7;i++){
  this.andarAcima();
}
this.diga("voltei ao inicio");
  
```

Figura 11 - Solução do exercício 4

Além destas funções básicas de fazer o robô andar e dizer também é possível executar comandos para empurrar, remover, contar ou acumular o valor de objetos no mundo. Para que isso aconteça basta mover o robô em direção a algum objeto. Ao ser detectada a presença de algum objeto surgirá o dialogo mostrado na figura 12.



Figura 12 - Opções de interação

Quando o botão `Acumular` for clicado, o dialogo da Figura 13 será mostrado. Tem-se a opção de escolha da contagem do valor de todos objetos do tipo encontrado, em todo o mundo ou somente em colunas pares ou impares, bem como linhas pares ou impares.

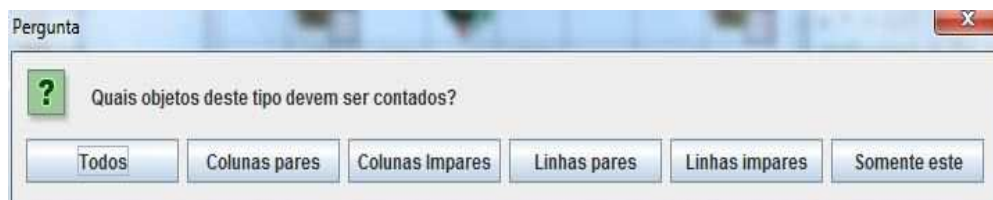


Figura 13 - O que deve ser contado

Para a contagem de todos os objetos, o código gerado será semelhante ao do Quadro

21.

```

this.diga("Contando todos os objetos do tipo Tesouro...");
int contTotalTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
        if (objeto != null) {
            if(objeto.getSouDoTipo().equals("Tesouro")) {
                contTotalTesouro++;
            }
        }
    }
}
this.diga("Foram encontrados "+contTotalTesouro+" objetos do tipo
Tesouro");
if(!ehFim(DIREITA)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(DIREITA);
    if (objeto == null) {        this.andarDireita();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarDireita();
        }
    }
}
}

```

Quadro 21 - Código gerado para contagem de objetos do tipo tesouro do mundo todo

Para a contagem de objetos em colunas pares o código gerado será semelhante ao do Quadro 22.

```

this.andarDireita();
this.diga("Contando objetos do tipo Tesouro em colunas pares...");
int contColParTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mX % 2 == 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    contColParTesouro++;
                }
            }
        }
    }
}
this.diga("Foram encontrados "+contColParTesouro+" objetos do tipo
Tesouro em colunas pares");
if(!ehFim(DIREITA)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(DIREITA);
    if (objeto == null) {        this.andarDireita();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarDireita();
        }
    }
}
}

```

Quadro 22 - Código gerado para contagem de objetos do tipo tesouro em colunas pares

Para a contagem de objetos em colunas ímpares o código gerado será semelhante ao do Quadro 23.

```

this.andarAbaixo();
this.diga("Contando objetos do tipo Tesouro em colunas impares...");
int contColImparTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mX % 2 != 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    contColImparTesouro++;
                }
            }
        }
    }
}
this.diga("Foram encontrados "+contColImparTesouro+" objetos do tipo
Tesouro em colunas impares");
if(!ehFim(ABAIXO)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(ABAIXO);
    if (objeto == null) {        this.andarAbaixo();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarAbaixo();
        }
    }
}
}

```

Quadro 23 - Código gerado para contagem de objetos do tipo tesouro em colunas impares

Para a contagem de objetos em linhas pares o código gerado será semelhante ao do Quadro 24.

```

this.andarDireita();
this.diga("Contando objetos do tipo Tesouro em linhas pares...");
int contLinParTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mY % 2 == 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    contLinParTesouro++;
                }
            }
        }
    }
}
this.diga("Foram encontrados "+contLinParTesouro+" objetos do tipo
Tesouro em colunas pares");
if(!ehFim(ABAIXO)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(ABAIXO);
    if (objeto == null) {        this.andarAbaixo();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarAbaixo();
        }
    }
}
}

```

Quadro 24 - Código gerado para contagem de objetos do tipo tesouro em linhas pares

Para a contagem de objetos em linhas impares o código gerado será semelhante ao do Quadro 25.

```

this.andarAbaixo();
this.diga("Contando objetos do tipo Tesouro em linhas impares...");
int contLinImparTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mY % 2 != 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    contLinImparTesouro++;
                }
            }
        }
    }
}
this.diga("Foram encontrados "+contLinImparTesouro+" do tipo Tesouro
em linhas impares");
if(!ehFim(DIREITA)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(DIREITA);
    if (objeto == null) {        this.andarDireita();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarDireita();
        }
    }
}
}

```

Quadro 25 - Código gerado para contagem de objetos do tipo tesouro em linhas impares

Quando o botão `Acumular` for clicado, o dialogo da Figura 14 será mostrado. Dessa forma ter-se a possibilidade de escolha da soma do valor de todos objetos do tipo encontrado, em todo o mundo ou somente em colunas pares ou impares, bem como linhas pares ou impares.



Figura 14 - O que deve ser acumulado

Para o acumulo de todos os objetos o código gerado será semelhante ao do Quadro 26.

```

this.andarDireita();
this.diga("Somando todos os objetos do tipo Tesouro...");
int somaTotalTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
        if (objeto != null) {
            if(objeto.getSouDoTipo().equals("Tesouro")) {
                somaTotalTesouro+=
Integer.parseInt(objeto.toString());
            }
        }
    }
}
this.diga("A soma de todos os objetos do tipo Tesouro é de
"+somaTotalTesouro);
if(!ehFim(ABAIXO)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(ABAIXO);
    if (objeto == null) {        this.andarAbaixo();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarAbaixo();
        }
    }
}
}

```

Quadro 26 - Código gerado para acumulo de objetos do tipo tesouro do mundo todo

Para o acumulo de objetos em colunas pares o código gerado será semelhante ao do Quadro 27.



```

this.diga("Somando objetos do tipo Tesouro em colunas pares...");
int somaColParTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mX % 2 == 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    somaColParTesouro+=
Integer.parseInt(objeto.toString());
                }
            }
        }
    }
}
this.diga("A soma dos objetos do tipo Tesouro em colunas pares é de
"+somaColParTesouro);
if(!ehFim(ABAIXO)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(ABAIXO);
    if (objeto == null) {        this.andarAbaixo();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarAbaixo();
        }
    }
}
}

```

Quadro 27 - Código gerado para o acúmulo de objetos do tipo tesouro em colunas pares

Para o acúmulo de objetos em colunas ímpares o código gerado será semelhante ao do Quadro 28.

```

this.andarDireita();
this.diga("Somando objetos do tipo Tesouro em colunas impares...");
int somaColImparTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mX % 2 != 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    somaColImparTesouro+=
Integer.parseInt(objeto.toString());
                }
            }
        }
    }
}
this.diga("A soma dos objetos do tipo Tesouro em colunas impares é de
"+somaColImparTesouro);
if(!ehFim(ABAIXO)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(ABAIXO);
    if (objeto == null) {        this.andarAbaixo();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarAbaixo();
        }
    }
}
}

```

Quadro 28 - Código gerado para o acúmulo de objetos do tipo tesouro em colunas ímpares

Para o acúmulo de objetos em linhas pares o código gerado será semelhante ao do Quadro 29.

```

this.andarDireita();
this.andarAbaixo();
this.diga("Somando objetos do tipo Tesouro em linhas pares...");
int somaLinParTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mY % 2 == 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    somaLinParTesouro+=
Integer.parseInt(objeto.toString());
                }
            }
        }
    }
}
this.diga("A soma dos objetos do tipo Tesouro em linhas pares é de
"+somaLinParTesouro);
if(!ehFim(ABAIXO)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(ABAIXO);
    if (objeto == null) {        this.andarAbaixo();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarAbaixo();
        }
    }
}
}

```

Quadro 29 - Código gerado para o acúmulo de objetos do tipo tesouro em linhas pares

Para o acúmulo de objetos em linhas ímpares o código gerado será semelhante ao do Quadro 30.

```

this.andarAbaixo();
this.diga("Somando objetos do tipo Tesouro em linhas impares...");
int somaLinImparTesouro = 0;for (int mY = 0; mY <= 8; mY++) {
    for (int mX = 0; mX <= 8; mX++) {
        if(mY % 2 != 0) {
            ObjetoDoMundoAdapter objeto = this.getObjetoXY(mX, mY);
            if (objeto != null) {
                if(objeto.getSouDoTipo().equals("Tesouro")) {
                    somaLinImparTesouro+=
Integer.parseInt(objeto.toString());
                }
            }
        }
    }
}
this.diga("A soma dos objetos do tipo Tesouro em linhas impares é de
"+somaLinImparTesouro);
if(!ehFim(DIREITA)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(DIREITA);
    if (objeto == null) {        this.andarDireita();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarDireita();
        }
    }
}
}

```

Quadro 30 - Código gerado para o acúmulo de objetos do tipo tesouro em linhas ímpares

Também é possível empurrar, remover, contar ou somar e informar a posição atual no mundo. O código gerado para estas funcionalidades está descrito no Quadro 31.

```

this.getObjeto(DIREITA).andarDireita();
removerObjetoDoMundo(this.getObjeto(ABAIXO));
int somaTesouro =
Integer.parseInt(this.getObjeto(DIREITA).toString());
if(!ehFim(DIREITA)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(DIREITA);
    if (objeto == null) {        this.andarDireita();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarDireita();
        }
    }
}
int contTesouro = 1;
if(!ehFim(ACIMA)) {
    ObjetoDoMundoAdapter objeto = this.getObjeto(ACIMA);
    if (objeto == null) {        this.andarAcima();
    } else {
        if (!objeto.getSouDoTipo().equals("Parede")) {
            this.andarAcima();
        }
    }
}
this.diga("Coluna: "+getX()+" | Linha: " + getY());
this.diga("Foram encontrados "+contTesouro+" objetos do tipo
Tesouro");
this.diga("A soma dos objetos do tipo Tesouro é de "+somaTesouro);

```

Quadro 31 - Código gerado para empurrar, remover, contar ou somar, informar a posição atual no mundo

#### 4. RESULTADOS E DISCUSSÃO

O presente projeto apresentou a descrição de uma extensão ao Furbot com o objetivo possibilitar a solução de exercícios sem a necessidade de codificar a inteligência do robô, para depois executar o programa e verificar se a solução está correta. Como demonstrado, o aluno pode executar o Furbot, visualizar o cenário e controlar o robô utilizando o teclado e o mouse, para que o exercício possa ser resolvido em tempo de execução. Ao finalizar as tarefas que o enunciado do exercício pede, o aluno tem a opção de gerar o código da inteligência do robô.

Espera-se que esta nova funcionalidade permita aqueles alunos que tem dificuldades em construir uma solução abstrata de um dado cenário possam exercitá-la concretamente e observar como seria o algoritmo que implementa a solução do problema. A extensão não resolve os exercícios propostos visto que não realiza otimização de passos. Esta etapa permanece sob responsabilidade do aluno. Um dos aspectos a ser melhorado refere-se ao módulo de geração de código.

Atualmente a análise da solução não considera a melhor solução possível para o exercício proposto. Eventualmente, dependendo do cenário construído, a adoção de algoritmos de busca poderiam trazer soluções melhores que aquela atualmente apresentada ao acadêmico. E de certa forma a automatização apresentada nesta extensão não é muito eficaz para a solução de exercícios com mundos dinâmicos, nos quais os objetos aparecem em locais diferentes em cada execução.

Ainda assim possibilita a geração de código para a contagem ou acúmulo de objetos que não dependem de um mundo estático. Neste caso cabe ao aluno desenvolver o raciocínio para resolver o exercício de forma que não dependa de objetos em posições específicas.

## 5. CONCLUSÕES

O desenvolvimento do trabalho atendeu a todos os objetivos propostos. Conforme os objetivos, foi desenvolvido uma extensão ao Furbot que possibilita que a programação da inteligência do mesmo aconteça de maneira simples, permitindo que o aluno controle o robô pelo teclado e resolva exercícios sem a necessidade de escrever o código para testar se a solução está correta fazendo o robô executá-la.

O aluno pode executar o exercício normalmente, ler quais são os objetivos e controlar o robô para que o exercício seja resolvido, ao resolver o exercício basta parar e gerar o código para a solução que o aluno acabou de criar.

Esta maneira agiliza muito o aprendizado em programação para alunos iniciantes que não possuem muita experiência com programação. Além disso o código gerado contribui para o aluno entenda a lógica da solução do problema, pois cria loops para comandos repetitivos e verifica a possibilidade da execução em trechos duvidosos no mundo em que o exercício acontece.

O embasamento teórico que sustenta a implementação baseou-se no conceito construcionista de aprendizado e contrasta com o conceito instrucionista que norteou o projeto original do Furbot. Isso porque, do ponto de vista de aprendizagem, programar o Furbot implica em desenvolver estratégias que viabilizam a movimentação de personagens em um mundo 2D. Do ponto de vista de ensino, programar o Furbot implica em construir e refinar o processo de desenvolvimento de algoritmos através da apresentação de problemas com complexidade crescente, utilizando para isso a metáfora de jogos.

A funcionalidade desenvolvida foi validada através de alguns estudos de caso onde ficou evidenciado a simplicidade e eficiência do projeto. Os trabalhos correlatos foram utilizados como base de consulta sobre aspectos de interação com o usuário e recursos disponibilizados ao usuário.

## 6. EXTENSÕES

Durante o desenvolvimento foram notadas algumas possibilidades de extensões deste trabalho. A primeira destas possibilidades é a inclusão de algoritmos de busca para encontrar soluções mais apropriadas em cenários dinâmicos.

Uma outra possibilidade seria a geração de um *log* de movimentos de tal forma a caracterizar para o aluno em que situação encontrava-se o robô em determinado momento e qual a decisão o algoritmo tomou para realizar o próximo movimento.



## 7. APÊNDICE A

O método `gerarCodigo()` é executado quando o aluno clica no botão Gerar Código. Este método verifica todos os comandos que foram adicionados a lista `cmdsExecutados` e concatena todo o código segundo cada elemento da lista, em seguida instância a classe `FurbotCDialog` e adiciona o código gerado a instância, para então ser exibido conforme o Anexo A.

```

public static final void gerarCodigo() {
    int cont = 1, prox = 0;
    String codigo = "";
    String parametro = "";
    Integer valor = 0;
    String dir = "";
    String local = "";
    ArrayList contHelper = new ArrayList<String>();
    ArrayList somaHelper = new ArrayList<String>();
    for (int i = 0; i < cmdsExecutados.size(); i++) {
        try {
            valor = Integer.parseInt(cmdsExecutados.get(i));
        } catch (Exception e) {
            String[] dado = cmdsExecutados.get(i).split(",");
            valor = Integer.parseInt(dado[0]);
            parametro = dado[1];
            if (valor == 17) {
                dir = dado[2];
            }
            if ((valor == 18) || (valor == 19)) {
                local = dado[2];
            }
        }
        prox = i + 1;
        switch (valor) {
            case 1: {
                if ((prox < cmdsExecutados.size()) &&
                    (proxCmdEhIgual(prox, valor) == true)) {
                    cont++;
                } else {
                    if (cont > 1) {
                        String.valueOf(cont) + ";i++) {\n";
                        codigo += "    this.andarEsquerda();\n}\n";
                        cont = 1;
                    } else {
                        codigo += "this.andarEsquerda();\n";
                    }
                }
            }
            break;
            case 2: {
                if ((prox < cmdsExecutados.size()) &&
                    (proxCmdEhIgual(prox, valor) == true)) {
                    cont++;
                } else {
                    if (cont > 1) {

```

```

        codigo += "for(int i=0;i<" +
String.valueOf(cont) + ";i++) {\n";
        codigo += "    this.andarAcima();\n";
        codigo += "}" + "\n";
        cont = 1;
    } else {
        codigo += "this.andarAcima();\n";
    }
    }
}
break;
case 3: {
    if ((prox < cmdsExecutados.size()) &&
(proxCmdEhIgual(prox, valor) == true)) {
        cont++;
    } else {
        if (cont > 1) {
String.valueOf(cont) + ";i++) {\n";
            codigo += "    this.andarDireita();\n}\n";
            cont = 1;
        } else {
            codigo += "this.andarDireita();\n";
        }
    }
}
break;
case 4: {
    if ((prox < cmdsExecutados.size()) &&
(proxCmdEhIgual(prox, valor) == true)) {
        cont++;
    } else {
        if (cont > 1) {
String.valueOf(cont) + ";i++) {\n";
            codigo += "    this.andarAbaixo();\n}\n";
            cont = 1;
        } else {
            codigo += "this.andarAbaixo();" + "\n";
        }
    }
}
break;
case 5: {
    codigo += "if(!ehFim(ESQUERDA)) {\n"
+ "    ObjetoDoMundoAdapter objeto =
this.getObjeto(ESQUERDA);\n"
+ "    if (objeto == null) {"
+ "        this.andarEsquerda();\n"
+ "    } else {\n"
+ "        if
(!objeto.getSouDoTipo().equals(\"Parede\")) {\n"
+ "            this.andarEsquerda();\n"
+ "        }\n    }\n}\n";
}
break;
case 6: {
    codigo += "if(!ehFim(ACIMA)) {\n"
+ "    ObjetoDoMundoAdapter objeto =
this.getObjeto(ACIMA);\n"

```

```

+ "    if (objeto == null) {"
+ "        this.andarAcima();\n"
+ "    } else {\n"
+ "        if
(!objeto.getSouDoTipo().equals(\"Parede\")) {\n"
+ "            this.andarAcima();\n"
+ "        }\n    }\n};
}
break;
case 7: {
    codigo += "if(!ehFim(DIREITA)) {\n"
+ "    ObjetoDoMundoAdapter objeto =
this.getObjeto(DIREITA);\n"
+ "    if (objeto == null) {"
+ "        this.andarDireita();\n"
+ "    } else {\n"
+ "        if
(!objeto.getSouDoTipo().equals(\"Parede\")) {\n"
+ "            this.andarDireita();\n"
+ "        }\n    }\n};
}
break;
case 8: {
    codigo += "if(!ehFim(ABAIXO)) {\n"
+ "    ObjetoDoMundoAdapter objeto =
this.getObjeto(ABAIXO);\n"
+ "    if (objeto == null) {"
+ "        this.andarAbaixo();\n"
+ "    } else {\n"
+ "        if
(!objeto.getSouDoTipo().equals(\"Parede\")) {\n"
+ "            this.andarAbaixo();\n"
+ "        }\n    }\n};
}
break;
case 9: {
    codigo += "this.diga(\"\" + parametro + "\");\n";
}
break;
case 10: {
    codigo += "this.diga(\"Coluna: \"+getX()+\" | Linha:
\" + getY());\n";
}
;
break;
case 11: {
    codigo +=
"this.getObjeto(ESQUERDA).andarEsquerda();\n";
}
break;
case 12: {
    codigo += "this.getObjeto(ACIMA).andarAcima();\n";
}
break;
case 13: {
    codigo +=
"this.getObjeto(DIREITA).andarDireita();\n";
}
break;
case 14: {
    codigo += "this.getObjeto(ABAIXO).andarAbaixo();\n";
}

```

```

    }
    break;
    case 15: {
        codigo += "removerObjetoDoMundo(this.getObjeto(" +
parametro.toUpperCase() + "));\n";
    }
    break;
    case 16: {
        if (contHelper.contains(parametro)) {
            codigo += "cont" + parametro + "++;\n";
        } else {
            codigo += "int cont" + parametro + " = 1;\n";
            contHelper.add(parametro);
        }
    }
    break;
    case 17: {
        if (somaHelper.contains(parametro)) {
            codigo += "soma" + parametro + " +=
Integer.parseInt(this.getObjeto(" + dir + ").toString());\n";
        } else {
            codigo += "int soma" + parametro + " =
Integer.parseInt(this.getObjeto(" + dir + ").toString());\n";
            somaHelper.add(parametro);
        }
    }
    break;
    case 18: {
        switch (Integer.parseInt(local)) {
            case 0: {
                codigo +=
objetos do tipo " + parametro + "...)\n"
                + "int contTotal" + parametro + " =
0;"
                + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
                + "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
                + "        ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
                + "        if (objeto != null) {\n"
                + "
                + "            contTotal" +
parametro + "++;\n"
                + "
                + "        }\n"
                + "    }\n"
                + "}"
                + "this.diga(\"Foram encontrados
\"+contTotal" + parametro + "+\" objetos do tipo " + parametro +
"\");\n";
            }
            break;
            case 1: {
                codigo +=
"this.diga(\"Contando objetos do tipo
" + parametro + " em colunas pares...)\n";
                + "int contColPar" + parametro + " =
0;"

```

```

+ "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
+ "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
+ "        if(mX % 2 == 0) {\n"
+ "            ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
+ "            if (objeto != null)
{\n"
+ "
+ "
if(objeto.getSouDoTipo().equals(\"" + parametro + "\")) {\n"
+ "                contColPar" +
parametro + "++;\n"
+ "
+ "                }\n"
+ "            }\n"
+ "        }\n"
+ "    }\n"
+ "}"
+ "this.diga(\"Foram encontrados
\"+contColPar" + parametro + "+\" objetos do tipo " + parametro + " em
colunas pares\");\n";
    }
    break;
    case 2: { // Colunas impares
        codigo +=
            "this.diga(\"Contando objetos do tipo
" + parametro + " em colunas impares...\");\n"
            + "int contColImpar" + parametro + "
= 0;"
            + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
            + "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
            + "        if(mX % 2 != 0) {\n"
            + "            ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
            + "            if (objeto != null)
{\n"
            + "
            + "
if(objeto.getSouDoTipo().equals(\"" + parametro + "\")) {\n"
            + "                contColImpar"
+ parametro + "++;\n"
            + "
            + "                }\n"
            + "            }\n"
            + "        }\n"
            + "    }\n"
            + "}"
            + "this.diga(\"Foram encontrados
\"+contColImpar" + parametro + "+\" objetos do tipo " + parametro + " em
colunas impares\");\n";
        }
        break;
        case 3: { // Linhas pares
            codigo +=
                "this.diga(\"Contando objetos do tipo
" + parametro + " em linhas pares...\");\n"
                + "int contLinPar" + parametro + " =
0;"
                + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
                + "    for (int mX = 0; mX <= " +

```

```

qtdColunas + "; mX++) {\n"
+ "          if(mY % 2 == 0) {\n"
+ "              ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
+ "          if (objeto != null)
{\n"
+ "
+ "          if(objeto.getSouDoTipo().equals(\"" + parametro + "\")) {\n"
+ "              contLinPar" +
parametro + "++;\n"
+ "          }\n"
+ "      }\n"
+ "  }\n"
+ "}"
+ "this.diga(\"Foram encontrados
\"+contLinPar" + parametro + "+\" objetos do tipo " + parametro + " em
colunas pares\");\n";
}
break;
case 4: { //linhas impares
    codigo +=
        "this.diga(\"Contando objetos do tipo
" + parametro + " em linhas impares...\");\n"
        + "int contLinImpar" + parametro + "
= 0;"
        + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
+ "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
+ "        if(mY % 2 != 0) {\n"
+ "            ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
+ "            if (objeto != null)
{\n"
+ "
+ "            if(objeto.getSouDoTipo().equals(\"" + parametro + "\")) {\n"
+ "                contLinImpar"
+ parametro + "++;\n"
+ "            }\n"
+ "        }\n"
+ "    }\n"
+ "}"
+ "this.diga(\"Foram encontrados
\"+contLinImpar" + parametro + "+\" do tipo " + parametro + " em linhas
impares\");\n";
}
}
}
break;
case 19: {
    switch (Integer.parseInt(local)) {
        case 0: {
            codigo +=
                "this.diga(\"Somando todos os objetos
do tipo " + parametro + "...\");\n"
                + "int somaTotal" + parametro + " =
0;"
                + "for (int mY = 0; mY <= " +

```

```

qtdLinhas + "; mY++) {\n"
                                + "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
                                + "        ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
                                + "        if (objeto != null) {\n"
                                + "
if(objeto.getSouDoTipo().equals(\"" + parametro + "\")) {\n"
                                + "                somaTotal" +
parametro + "+= Integer.parseInt(objeto.toString());\n"
                                + "                }\n"
                                + "            }\n"
                                + "        }\n"
                                + "    }\n"
                                + "}"
                                + "this.diga(\"A soma de todos os
objetos do tipo " + parametro + " é de \"+somaTotal" + parametro +
");\n";
        }
        break;
        case 1: { //Colunas pares
            codigo +=
                "this.diga(\"Somando objetos do tipo
" + parametro + " em colunas pares...\");\n"
                + "int somaColPar" + parametro + " =
0;"
                + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
                + "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
                + "        if(mX % 2 == 0) {\n"
                + "            ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
                + "            if (objeto != null)
{\n"
                + "
if(objeto.getSouDoTipo().equals(\"" + parametro + "\")) {\n"
                + "                somaColPar" +
parametro + "+= Integer.parseInt(objeto.toString());\n"
                + "                }\n"
                + "            }\n"
                + "        }\n"
                + "    }\n"
                + "}"
                + "this.diga(\"A soma dos objetos do
tipo " + parametro + " em colunas pares é de \"+somaColPar" + parametro +
");\n";
        }
        break;
        case 2: { // Colunas impares
            codigo +=
                "this.diga(\"Somando objetos do tipo
" + parametro + " em colunas impares...\");\n"
                + "int somaColImpar" + parametro + "
= 0;"
                + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
                + "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
                + "        if(mX % 2 != 0) {\n"
                + "            ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"

```

```

+ "                if (objeto != null)
{\n"
+ "
+ "
if(objeto.getSouDoTipo().equals(\"" + parametro + "\")) {\n"
+ "                somaColImpar"
+ parametro + "+= Integer.parseInt(objeto.toString());\n"
+ "                }\n"
+ "            }\n"
+ "        }\n"
+ "    }\n"
+ "}"\n"
+ "this.diga(\"A soma dos objetos do
tipo " + parametro + " em colunas impares é de \""+somaColImpar" +
parametro + ");\n";
    }
    break;
    case 3: { // Linhas pares
        codigo +=
            "this.diga(\"Somando objetos do tipo
" + parametro + " em linhas pares...\");\n"
            + "int somaLinPar" + parametro + " =
0;"
            + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
            + "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
            + "        if(mY % 2 == 0) {\n"
            + "            ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
            + "            if (objeto != null)
{\n"
            + "
            + "                somaLinPar" +
parametro + "+= Integer.parseInt(objeto.toString());\n"
            + "                }\n"
            + "            }\n"
            + "        }\n"
            + "    }\n"
            + "}"\n"
            + "this.diga(\"A soma dos objetos do
tipo " + parametro + " em linhas pares é de \""+somaLinPar" + parametro +
");\n";
        }
        break;
        case 4: { //linhas impares
            codigo +=
                "this.diga(\"Somando objetos do tipo
" + parametro + " em linhas impares...\");\n"
                + "int somaLinImpar" + parametro + "
= 0;"
                + "for (int mY = 0; mY <= " +
qtdLinhas + "; mY++) {\n"
                + "    for (int mX = 0; mX <= " +
qtdColunas + "; mX++) {\n"
                + "        if(mY % 2 != 0) {\n"
                + "            ObjetoDoMundoAdapter
objeto = this.getObjetoXY(mX, mY);\n"
                + "            if (objeto != null)
{\n"
                + "

```



```

if(objeto.getSouDoTipo().equals("\" + parametro + "\")) {\n"
+ "                                     somaLinImpar"
+ parametro + "+= Integer.parseInt(objeto.toString());\n"
+ "                                     }\n"
+ "                                     }\n"
+ "                                     }\n"
+ "                                     }\n"
+ "                                     }\n"
+ "                                     }\n"
+ "this.diga(\"A soma dos objetos do
tipo " + parametro + " em linhas impares é de \""+somaLinImpar" +
parametro + ");\n";
    }
    }
}

if (qtdTiposCont.size() > 0) {
    for (int it = 0; it < qtdTiposCont.size(); it++) {
        codigo += "this.diga(\"Foram encontrados \""+cont"
+ qtdTiposCont.get(it) + "+\" objetos do tipo "
+ qtdTiposCont.get(it) + "\");\n";
    }
}
if (qtdTiposSoma.size() > 0) {
    for (int it = 0; it < qtdTiposSoma.size(); it++) {
        codigo += "this.diga(\"A soma dos objetos do tipo "
+ qtdTiposSoma.get(it) + " é de \""+soma" +
qtdTiposSoma.get(it) + "\");\n";
    }
}
FurbotCDialog dialogo = new FurbotCDialog(null, true);
dialogo.add(codigo);
dialogo.setVisible(true);
}

```

Quadro 32 - corpo do método gerarCodigo()

## 8. APÊNDICE B

Todos os possíveis comandos que podem aparecer na lista `cmdsExecutados` estão descritos no Quadro 33.

Comando	Parâmetro adicional	Significado
1		Movimento a esquerda
2		Movimento acima
3		Movimento a direita
4		Movimento abaixo
5		Movimento verificado a esquerda
6		Movimento verificado acima
7		Movimento verificado a direita
8		Movimento verificado abaixo
9	String parâmetro com uma frase	Algo deve ser dito
10		A posição atual deve ser informada
11		Empurrar um objeto para a esquerda
12		Empurrar um objeto para cima
13		Empurrar um objeto para a direita
14		Empurrar um objeto para baixo
15	String parâmetro com a direção	Remover um objeto
16		Inicia ou continua a contagem individual de algum tipo de objeto
17		Inicia ou continua a soma individual de algum tipo de objeto
18	String tipo com o tipo a ser contado, String local com a localidade a ser verificada	Faz a contagem de objetos segundo o parâmetro tipo nas colunas/linhas determinadas pelo parâmetro local
19	String tipo com o tipo a ser somado, String local com a localidade a ser verificada	Faz a soma do valor de objetos segundo o parâmetro tipo nas colunas/linhas determinadas pelo parâmetro local

Quadro 33 - Lista de todos os comandos possíveis do atributo `cmdsExecutados` ( )

## REFERÊNCIAS BIBLIOGRÁFICAS

- ABRANTES, Steven L. **O uso dos jogos como estratégia de aprendizagem para alunos do 1º ciclo do ensino básico**. O caso do CD-ROM Escola Digital. 2007. 113 f. Dissertação (Mestrado em Gestão da Informação) – Departamento de Economia, Gestão e Engenharia, Industrial Universidade de Aveiro, Aveiro. Disponível em: <<http://ria.ua.pt/bitstream/10773/1514/1/2007001327.pdf>>. Acesso em: 29 set. 2011.
- ALMEIDA, Maria E. B. de. **Informática e formação de professores**. Secretaria de Educação a Distância. ProInfo- Brasília: Ministério da Educação, SEED, 2000.
- BERGIN, Joe et al. **Patterns for experiential learning**. [S.l.], 2001. Disponível em: <<http://csis.pace.edu/~bergin/patterns/ExperientialLearning.html>>. Acesso em: 20 set. 2008.
- LIMA, Márcio R. de; LEAL, Murilo C. Uso da linguagem Logo no ensino superior de programação de computadores. **Revista Eletrônica Multidisciplinar Pindorama do Instituto Técnico Federal de Educação, Ciência e Tecnologia da Bahia – IFBA**, Eunápolis, BA, n. 1, ano 1, p. 1-14, ago. 2010. Disponível em: <<http://revistapindorama.ifba.edu.br/files/MARCIO ROBERTO DE LIMA UFVJM.pdf>>. Acesso em: 29 set.2011.
- 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](http://www.inf.furb.br/poo/furbot/files/Apostila_FURBOT.pdf) >. Acesso em: 20 set. 2008.
- MORELATO, Leandro de A. et al. Avaliando diferentes possibilidades de uso da robótica na educação. **Revista de Ensino de Ciências e Matemática**, São Paulo, v. 1, f. 2, p. 80-96, jan. 2011.
- NORMAN, Donald A.; SPOHRER, James C. Learner-centered education. **Communications of the ACM**, New York, v. 39, n. 4, p. 24-27, Apr. 1996.
- PAPERT, Seymour. **Construcionism**: a new opportunity for elementary science education; a proposal to the National Science Foundation. Cambridge-Massachusetts: MIT, Media Lab., Epistemology and Learning Group, 1986.
- \_\_\_\_\_. **A família em rede**: ultrapassando a barreira digital entre gerações. Título original: The Connected Family: bridging the digital generation gap. Lisboa: Relógio D'Água Editores, 1997.
- POCRIFKA, Dagmar H. et al. Linguagem Logo e a construção do conhecimento. In: CONGRESSO NACIONAL DE EDUCAÇÃO – EDUCERE, 9., ENCONTRO SUL BRASILEIRO DE PSICOPEDAGOGIA, 3., 2009, Curitiba. **Anais...** Curitiba: Champagnat, 2009. p. 2470-2479.

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

SOLOWAY, Elliot; PRYOR, Amanda. The next generation in human-computer interaction. **Communications of the ACM**, New York, v. 39, n. 4, p. 16-18, Apr. 1996.

SOUZA, Ana de F. **A maior vantagem competitiva é a habilidade de aprender.** [S.l.], [2004]. Disponível em: <<http://www.dimap.ufrn.br/~jair/piu/artigos/seymour.html>>. Acesso em: 10 set. 2011.

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. Não paginado. Disponível em: <<http://sites.google.com/site/adilsonv77/icece09.pdf?attredirects=0>>. Acesso em: 29 set. 2011.

VAHLDICK, Adilson; HUGO, Marcel; MATTOS, Mauro M. **Apostila de FURBOT – versão 1.9.** Apostila de Departamento de Sistemas e Computação. Blumenau, 2008.

VALENTE, José A. Diferentes usos do computador na educação. IN: VALENTE, José A. (Org.). **Computadores e Conhecimento: repensando a educação.** Campinas: Gráfica da UNICAMP, 1993. p. 1-23.