UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

ROBOTOY: FERRAMENTA PARA USO DE ROBÓTICA NO ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS

MARIA GABRIELA TORRENS

BLUMENAU
2014

2014/2-15

MARIA GABRIELA TORRENS

ROBOTOY: FERRAMENTA PARA USO DE ROBÓTICA NO ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Profa. Joyce Martins, Mestre – Orientadora

BLUMENAU
2014

2014/2-15

ROBOTOY: FERRAMENTA PARA USO DE ROBÓTICA NO ENSINO DE PROGRAMAÇÃO PARA CRIANÇAS

Por

MARIA GABRIELA TORRENS

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

Presidente:	Profa. Joyce Martins, Mestre – Orientadora, FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB
Membro:	Prof. Maurício Capobianco Lopes, Doutor – FURB

Blumenau, 10 de dezembro de 2014

AGRADECIMENTOS

Aos meus pais, José Renato Torrens e Ivone Rodrigues Torrens, por terem me apoiado ao longo do curso tanto moralmente quanto financeiramente.

Ao meu noivo, Thiago Alexandre Gesser, por ter me apresentado ao mundo da computação e ter me aconselhado ao longo desses anos nos assuntos pertinentes à área.

À minha orientadora, Joyce Martins, a quem admiro muito pela competência e profissionalismo demonstrados ao longo deste trabalho, que soube me orientar com eficiência nos momentos de dificuldade.

Ao coordenador de curso, Aurélio Faustino Hoppe, por ter me auxiliado e me concedido a devida atenção durante a escolha do meu tema, permitindo que eu pudesse trabalhar em algo de meu interesse e ao qual me identifico.

A persistência é o caminho do êxito.

Charles Chaplin

RESUMO

Este trabalho descreve a especificação de uma ferramenta de programação desenvolvida para crianças, que abrange o ambiente da robótica educacional. Para tanto, foi utilizada a linguagem Java para construir o ambiente de programação, bem como o Gerador de Analisadores Léxicos e Sintáticos (GALS) para definir a linguagem elaborada, além do Lego Java *Operating System* (leJOS) para realizar a implantação de programas na plataforma LEGO Mindstorms NXT. Não existem restrições quanto à montagem do robô, isto é, o usuário pode montar o modelo de sua preferência levando em consideração algumas regras que são mencionadas no decorrer do trabalho. Atualmente, a ferramenta suporta a implantação de programas apenas na plataforma LEGO Mindstorms NXT, mas foi desenvolvida de forma a permitir que outras plataformas venham a ser incluídas. Embora não tenham sido realizados testes com crianças, pode-se afirmar que a linguagem pode ser usada por elas por assemelhar-se a outra ferramenta correlata utilizada no ensino fundamental. No entanto, é necessário realizar adaptações no que diz respeito a aspectos visuais e de usabilidade, com o propósito de tornar a ferramenta mais atrativa.

Palavras-chave: Linguagens de programação. Ensino de programação. Robótica educacional. LEGO Mindstorms NXT. leJOS. Arduino.

ABSTRACT

This paper describes the specification of a programming tool developed for children, comprehending the environment of educational robotics. The Java language was used to build the development environment, GALS (Syntatical and Lexical Analyzer Generator) was used to define the developed language and the Lego Java Operating Systems (leJOS) was used to deploy the programs in the LEGO Mindstorms NXT platform. There are no restrictions in building the robot, allowing the user to build his preferred model following some rules mentioned in this paper. Currently, the tool supports the deployment of programs only in the LEGO Mindstorms NXT platform, but was developed to allow the inclusion of other platforms. Although kids were not involved during the tests, we can affirm that the language can be used by them due to its similarity with other related tool used in elementary school. However, some adaptations must be made in the visual and usability aspects, with the aim to make the tool more compelling.

Key-words: Programming languages. Programming learning. Educacional robotics. LEGO Mindstorms NXT. leJOS. Arduino.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

API – *Application Programming Interface*

FURB – Universidade Regional de Blumenau

GALS – Gerador de Analisadores Léxicos e Sintáticos

IDE – *Integrated Development Environment*

leJOS – Lego Java *Operating System*

MIT – *Massachusetts Institute of Technology*

RF – Requisito Funcional

RNF – Requisito Não Funcional

UCP – Unidade Central de Processamento

USB – *Universal Serial Bus*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Durante a fase escolar espera-se que uma criança desenvolva várias competências cognitivas. A realização de projetos na área da robótica educacional pode contribuir para o desenvolvimento dessas competências, pois a sua prática instiga os alunos a planejar, projetar, desenvolver e avaliar (FERREIRA, 2005). A teoria do construtivismo, desenvolvida por Jean Piaget, defende que o conhecimento não é algo que possa ser transmitido, mas sim construído pelo sujeito em conjunto com o ambiente que lhe é proporcionado (CHELLA, 2002, p. 20). Um ambiente de programação educativo se encaixa exatamente nesse cenário, pois o aluno aprende na prática por tentativa e erro, tirando suas próprias conclusões.

Segundo Oliveira, Silva e Andrade (2013), a robótica educacional é uma candidata em potencial para desenvolver o raciocínio lógico de uma criança, pois estimula a capacidade de questionar, formular e resolver problemas. No entanto, para programar é necessário definir o ambiente de desenvolvimento que será utilizado e, muitas vezes, ambientes desapropriados e monótonos para fins educacionais dificultam a adoção da programação na prática. Sendo assim, adotar a prática do lúdico aliada à programação no processo de ensino-aprendizagem pode ser uma alternativa para agilizar o desenvolvimento do raciocínio lógico, uma vez que as crianças tendem a ser mais receptivas com atividades recreativas (MARCON et al., 2012). Nesse sentido, a robótica educacional apresenta um ambiente mais atrativo, que abrange diretamente a programação, além de promover a criatividade, a curiosidade e a multidisciplinaridade (BENITTI et al., 2010a).

Em 1986, ao perceber a oportunidade de popularizar a robótica no meio educacional, a LEGO Group decidiu investir fortemente nesta área, o que resultou no lançamento do *kit* LEGO Mindstorms NXT, viabilizando a montagem e automação de robôs (LEGO GROUP, 2013a). Os robôs podem ser programados utilizando-se a API leJOS. No entanto, é necessário que o programador esteja familiarizado com a linguagem Java e o ambiente de desenvolvimento, muito complexos para uma criança.

Diante do acima exposto, foi desenvolvida uma ferramenta que possui uma linguagem de programação simplificada, permitindo que crianças possam desenvolver programas para a plataforma LEGO Mindstorms NXT. Desta forma, é possível que o indivíduo se concentre na solução de um problema específico, relevando detalhes da linguagem de programação e do ambiente de desenvolvimento.

1.1 OBJETIVOS

O objetivo principal deste trabalho é disponibilizar uma ferramenta para programação de robôs LEGO Mindstorms NXT.

Os objetivos específicos são:

a) disponibilizar uma linguagem que permita programar robôs;
b) traduzir os programas construídos para a API leJOS;
c) enviar o programa traduzido ao robô, para que o mesmo possa ser executado.

1.2 ESTRUTURA

O trabalho está dividido em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusão. No capítulo a seguir é apresentada a fundamentação teórica, que diz respeito aos assuntos pertinentes a esse trabalho. Na sequência é apresentado o desenvolvimento da ferramenta, que contempla a especificação e implementação da mesma e a montagem, especificação e implementação do robô. Por fim, são apresentados os resultados obtidos e a conclusão.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está dividido em quatro seções. A seção 2.1 aborda o tema ensino da programação. A seção 2.2 traz uma visão geral sobre robótica educacional, uma extensão do ensino da programação que inclui robôs. Já na seção 2.3 é feita uma contextualização acerca do tema compiladores. Por fim, na seção 2.4 são apresentados três trabalhos correlatos a este.

2.1 ENSINO DE PROGRAMAÇÃO

Segundo Setzer (2007), é muito comum as pessoas confundirem o ensino da programação com aprender uma linguagem de programação. No entanto, a linguagem de programação nada mais é do que um instrumento que facilita o processo de ensino-aprendizagem. Basicamente, a programação consiste em aprender a especificar a solução de um problema em termos de algoritmos. Há milhares de anos os seres humanos definem algoritmos sem a existência de computadores, por exemplo, quando somam dois números com muitos algarismos (SETZER, 2007).

No trabalho desenvolvido por Borges Neto e Borges (2007) foram estudadas algumas teorias relacionadas à educação infantil, como a teoria do construtivismo proposta por Jean Piaget. Diante do estudo dessas teorias, foram evidenciados os benefícios proporcionados a uma criança quando esta realiza atividades na área da informática, sendo um dos benefícios mencionados o desenvolvimento do raciocínio lógico.

Para Marcon et al. (2012), as ferramentas computacionais são potencializadoras do ensino, pois materializam o conhecimento e seduzem as crianças. As crianças que têm a oportunidade de interagir com a informática através da programação conseguem assimilar melhor as relações matemáticas, sobretudo pela experiência, o que torna o processo de ensino-aprendizagem enriquecedor (MARCON et al., 2012).

Existem diversos meios efetivos de ensinar programação para crianças, utilizando para isso ferramentas que foram especialmente projetadas para elas, como:

a) Logo: linguagem desenvolvida para crianças baseada na filosofia de uma educação não diretiva, de inspiração piagetiana, em que a criança aprende explorando o seu ambiente com regras que ela mesma impõe (CREATIVE COMMONS, 2009);
b) Scratch: linguagem de programação visual projetada para ser divertida, educativa e fácil de aprender. Com ela é possível criar histórias, jogos, artes e simulações. A parte gráfica da linguagem se assemelha a um jogo de quebra-cabeça, onde as peças representam comandos que podem ser conectados uns aos outros (LIFELONG KINDERGARTEN GROUP, 2014);

c) ToonTalk: linguagem de programação onde o próprio código-fonte é animado (KAHN, 1996). Neste ambiente a criança voa sobre uma cidade, podendo pousar e entrar nas casas para manipular objetos de programação e ferramentas de controle (MORGADO; CRUZ, 2004, p. 4).

2.2 ROBÓTICA EDUCACIONAL

A robótica educacional é caracterizada por promover no meio educacional a disseminação de projetos que envolvem a construção e manipulação de robôs. Resumidamente, um ambiente de aprendizagem é colocado à disposição do aluno, ambiente esse que favorece o desenvolvimento de habilidades e competências básicas, como o raciocínio lógico, a criatividade, a autonomia no aprendizado, o trabalho em equipe e a multidisciplinaridade (CASTILHO, 2009).

Diante dessa definição, vale a pena destacar o trabalho desenvolvido em sala de aula pela professora Andrade (2011). Neste trabalho foi dada a oportunidade dos alunos aplicarem em um robô o conhecimento teórico que adquiriram nas disciplinas de Física e Matemática (ANDRADE, 2011, p. 27). Os exercícios propostos em sala de aula tinham relação com assuntos que eram ensinados nessas disciplinas. Desta forma, os alunos acabavam absorvendo melhor o que haviam estudado pelo desafio de ver o robô funcionando efetivamente. Durante os exercícios, era possível perceber a motivação e o grande interesse demonstrado pelos alunos, bem como a persistência, a aprendizagem por tentativa e erro, a cooperação entre grupos e a partilha de resultados (ANDRADE, 2011, p. 46).

Além disso, com o uso da robótica educacional a aprendizagem não se limita exclusivamente ao que o professor ensina: o aluno é um elemento ativo neste processo, portanto, precisa aprender a questionar, a pesquisar, a experimentar e a partir disso construir o seu próprio conhecimento. É neste cenário que a robótica educacional vem, de forma lúdica e desafiadora, preencher um espaço existente entre as atividades desenvolvidas em sala de aula e o dia-a-dia (CASTILHO, 2009).

Resumindo, quando bem trabalhada, a adoção da robótica educacional pode tornar o processo de ensino-aprendizagem agradável e enriquecedor para ambos os lados, tanto para o aluno quanto para o professor. Sendo assim, a robótica educacional é uma opção viável para o professor que busca uma metodologia de ensino prática e lúdica, no intuito de motivar os alunos. Na robótica educacional existem várias plataformas que permitem a construção e automação de robôs, como os *kits* LEGO Mindstorms NXT e Arduino, mencionados a seguir.

2.2.1 LEGO Mindstorms NXT

A LEGO Group é uma empresa fabricante de brinquedos que, com a ajuda de professores e especialistas em educação, tenta levar soluções inovadoras para as salas de aula, a fim de transformar a maneira de aprender (LEGO GROUP, 2014). Em 1988, em parceria com o MIT, a LEGO criou um minicomputador no intuito de promover a aprendizagem da robótica em seus brinquedos, uma iniciativa que rendeu o melhor produto já feito na história da empresa (LEGO GROUP, 2013a). Com a criação do minicomputador, foi possível lançar no mercado os *kits* de robótica, conhecidos como LEGO Mindstorms.

Existem vários *kits* LEGO Mindstorms, dentre eles a derivação NXT 2.0, composta por 612 peças, sendo: 2 sensores ultrassônicos para detecção de distância; 2 sensores de toque; 1 sensor de cor que também detecta luz; 3 motores que permitem o deslocamento e o NXT *Brick*, que é a UCP de 32 bits. O NXT *Brick* é alimentado por 6 pilhas e possui 4 portas de entrada e 3 portas de saída, onde os sensores e motores podem ser conectados. Além disso, a UCP dispõe também de 1 *display* e 4 teclas de *input*, conforme ilustrado na Figura 1.

Figura 1 - NXT com motores e sensores conectados

Fonte: LEGO Group (2013b).

Há inúmeras formas de programar para a plataforma LEGO Mindstorms. Uma das opções é através da API leJOS. O leJOS é um mini sistema operacional baseado em Java, dividido em três partes: uma máquina virtual Java para execução do Java *bytecode*; uma API para programar para o LEGO Mindstorms e ferramentas de *software* adicionais (SCHOLZ, 2006). O leJOS NXJ, direcionado para o *kit* NXT, suporta as seguintes funcionalidades herdadas do Java (GRIFFITHS et al., 2009): programação orientada a objetos; *threads*; *arrays*, incluindo os multidimensionais; recursão; sincronização; exceções e classes Java simplificadas, contidas no `classes.jar` do leJOS.

O leJOS faz uso do compilador do Java para compilar seus programas. No entanto, essa compilação é feita com algumas adaptações. Basicamente, a biblioteca do Java é substituída pela biblioteca do leJOS, que contém um Java simplificado embutido, a partir da qual a compilação é realizada.

Por essa razão, o leJOS provê uma linha de comando conhecida como `nxjc`. Esse comando realiza a compilação levando em consideração as adaptações que foram mencionadas. Em seguida, para realizar a implantação no robô é necessário executar outros dois comandos: o `nxjlink`, que gera a partir do arquivo `.class` um arquivo binário com a extensão `.nxj`, e o `nxjupload`, que realiza a implantação do arquivo `.nxj` no robô. Existe ainda um quarto comando que resume o `nxjlink` e o `nxjupload` em um comando só, que é conhecido como `nxj`. Diante disso, a compilação, o *link* e a implantação de um programa podem ser feitos através de dois comandos: `nxjc` e `nxj` (GLASSEY; GRIFFITHS, 2012).

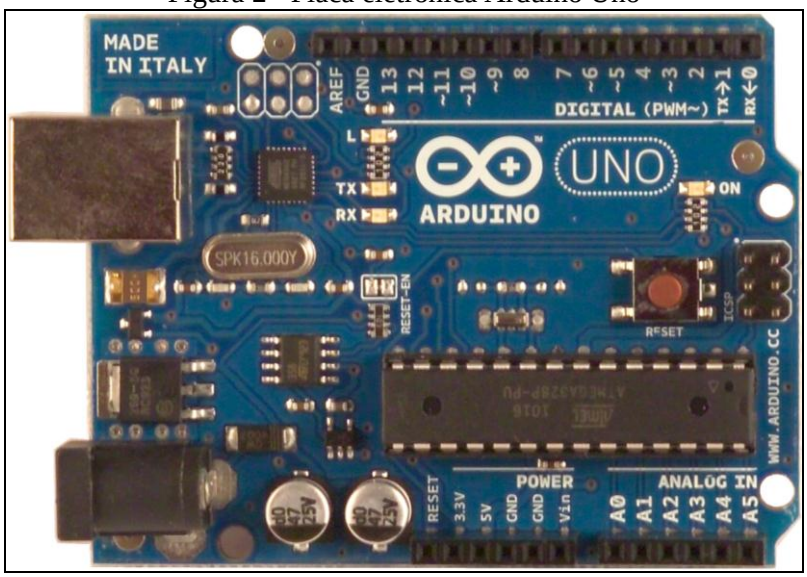
Por fim, para que a implantação seja realizada com sucesso é necessário que o robô esteja conectado ao computador através de um cabo USB. Uma segunda opção é enviar via *bluetooth*. Após isso, é possível executar os programas via *display*, com o auxílio das teclas de *input*.

2.2.2 Arduino

O projeto Arduino foi criado na Itália em 2005, com o objetivo de fornecer uma plataforma de prototipagem que fosse simples e de baixo custo, o que resultou numa placa eletrônica acompanhada de um *software* que viabiliza a programação (SILVA, 2014, p. 55). Através dessa plataforma, é possível montar uma variedade de circuitos, sendo que a linguagem de programação utilizada, conhecida como Processing, é simples e se assemelha muito ao Java e ao C++ (GIOPPO et al., 2009, p. 9).

É importante ressaltar que o Arduino não se limita exclusivamente a uma placa. Existem vários modelos de placas, cada qual com características que diferem entre si (GOMES; TAVARES, 2013). A placa precursora é conhecida como Arduino Uno, conforme ilustra a Figura 2.

Figura 2 - Placa eletrônica Arduino Uno



Fonte: Gomes e Tavares (2013).

Existe uma vasta comunidade em torno do projeto Arduino, envolvendo técnicos e desenvolvedores de diversas áreas, assim como alunos, professores e pessoas que interagem através da *internet* e eventos patrocinados mundo afora. Essa comunidade existe pelo fato do projeto como um todo ser de acesso público e gratuito, o que torna o Arduino muito atraente para pessoas que estão conhecendo essa plataforma (SOUZA et al., 2011, p. 1).

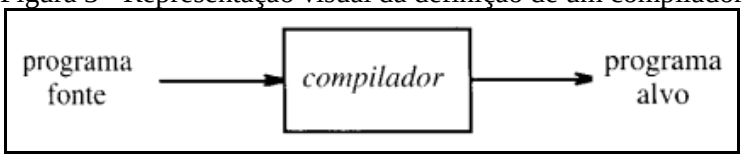
Segundo Souza et al. (2011, p. 2), o Arduino é ideal para a criação de dispositivos que interagem com o ambiente, que utilizam sensores de temperatura, luz, som, *leds*, motores, *displays*, alto-falantes, entre outros componentes. Diante disso, é possível afirmar que construir um robô com Arduino é algo extremamente versátil.

Assim como o LEGO Mindstorms, o Arduino também pode ser programado em Java utilizando-se a HaikuVM. A HaikuVM é uma máquina virtual Java desenvolvida para a programação de microcontroladores, que viabiliza a programação em Java no Arduino. É importante ressaltar que essa máquina virtual foi desenvolvida utilizando o leJOS (RUSTON, 2014).

2.3 COMPILADORES

Pode-se dizer de forma simplificada que um compilador é um programa que tem como objetivo realizar a leitura de outro programa escrito numa linguagem – a linguagem fonte – para traduzir num programa equivalente em outra linguagem – a linguagem alvo (AHO; SETHI; ULLMAN, 1995, p. 1). Na Figura 3 é ilustrada a representação visual da definição de um compilador.

Figura 3 - Representação visual da definição de um compilador

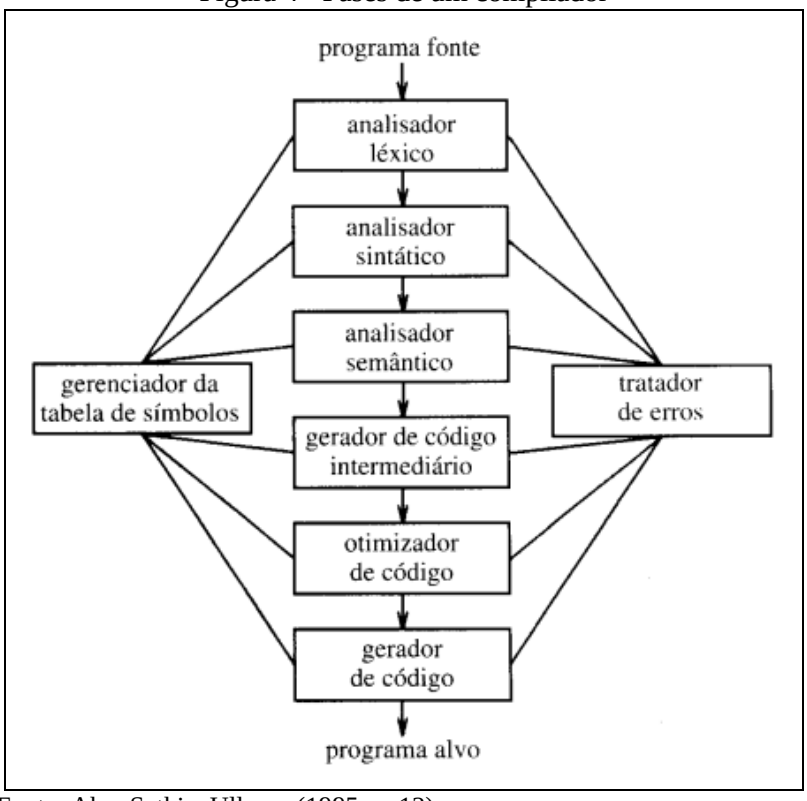


Fonte: Adaptado de Aho, Sethi e Ullman (1995, p. 8).

De forma geral, existem duas etapas que são executadas durante o processo de compilação: a análise e a síntese. A análise é responsável por dividir o programa em partes, para, a partir daí, criar uma representação intermediária do mesmo. Por sua vez, a síntese é responsável por construir o programa alvo desejado levando em consideração a representação intermediária (AHO; SETHI; ULLMAN, 1995, p. 1).

Conceitualmente, um compilador opera em fases, as três primeiras fases dizem respeito à etapa de análise do compilador, composta pelos analisadores léxico, sintático e semântico. As três últimas fases dizem respeito à etapa de síntese, constituída pelo gerador de código intermediário, o otimizador de código e o gerador de código (AHO; SETHI; ULLMAN, 1995, p. 5). Na Figura 4 são ilustradas as fases de um compilador.

Figura 4 - Fases de um compilador



Fonte: Aho, Sethi e Ullman (1995, p. 12).

O analisador léxico possui a função de efetuar a leitura de um fluxo de caracteres que constituem um programa, a fim de agrupá-los em *tokens*. Os *tokens* são sequências de caracteres que podem representar símbolos especiais, identificadores, números, palavras reservadas, entre outros (AHO; SETHI; ULLMAN, 1995, p. 2). Segundo Aho, Sethi e Ullman

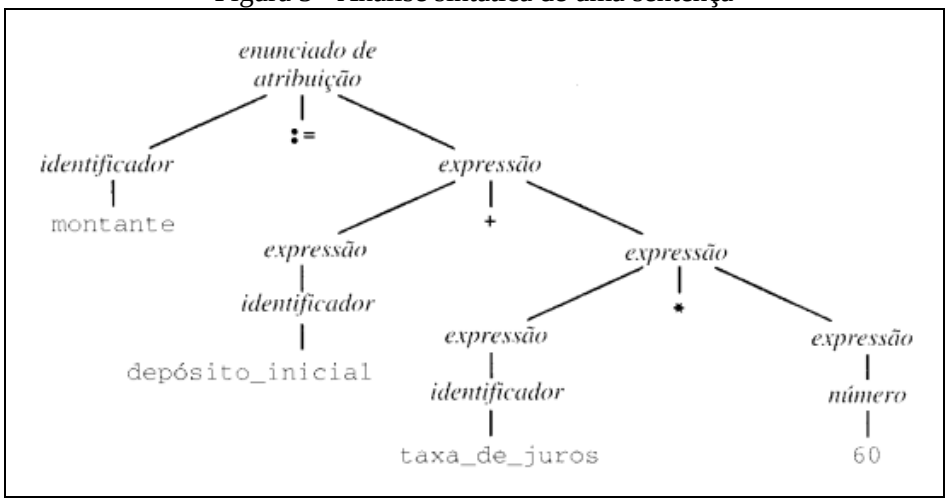
(1995, p. 3), a análise léxica da sentença `montante := depósito_inicial + taxa_de_juros * 60` resultaria na classificação apresentada no Quadro 1.

Quadro 1 - Classificação dos *tokens* de uma sentença

token	tipo
`montante`	identificador
`:=`	símbolo de atribuição (símbolo especial)
`depósito_inicial`	identificador
`+`	sinal de adição (símbolo especial)
`taxa_de_juros`	identificador
`*`	sinal de multiplicação (símbolo especial)
`60`	número

No analisador sintático, os *tokens* são agrupados numa estrutura hierárquica conhecida como árvore sintática, na qual cada nó representa uma operação e o filho do nó representa o argumento da operação (AHO; SETHI; ULLMAN, 1995, p. 1). Na Figura 5 é apresentada a análise sintática da sentença utilizada como exemplo.

Figura 5 - Análise sintática de uma sentença



Fonte: Aho, Sethi e Ullman (1995, p. 10).

Por sua vez, o analisador semântico verifica se as construções sintáticas estão semanticamente corretas. Para tanto, verifica a coerência entre a declaração e o uso de identificadores e extrai informações para assegurar a compatibilidade de operadores e operandos das expressões e sentenças, para que a fase de geração de código possa ser realizada sem problemas (AHO; SETHI; ULLMAN, 1995, p. 4).

Por fim, é realizada a etapa de geração de código, que consiste em traduzir o programa definido em alto nível para a linguagem de máquina. Em geral, a geração de código não ocorre diretamente para a linguagem *assembly*. Normalmente é realizada a geração de código para uma máquina abstrata, com uma linguagem semelhante ao *assembly* e independente de processadores, para daí então ser traduzida para a linguagem *assembly* desejada. Desta forma, é possível reaproveitar parte do compilador para trabalhar com diferentes tipos de

processadores (RICARTE, 2004, p. 128). Também é possível traduzir a linguagem de alto nível para outra linguagem de programação, como Java ou C.

2.4 TRABALHOS CORRELATOS

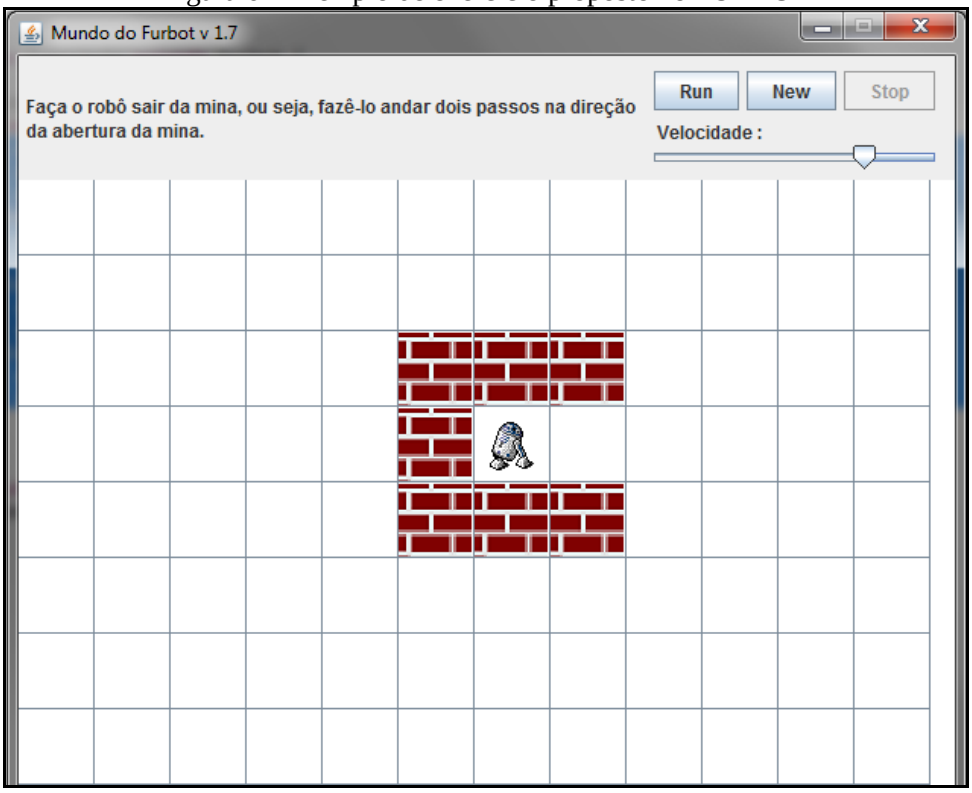
Nesta seção são descritos os trabalhos correlatos ao trabalho proposto, todos referentes a ferramentas que contribuem para o desenvolvimento da lógica através da programação.

2.4.1 FURBOT

O FURBOT é um ambiente que foi desenvolvido em 2008 na FURB, cujo objetivo é auxiliar as disciplinas iniciais de programação no processo de ensino-aprendizagem. Este ambiente possui um forte apelo na área de jogos, pois possibilita que os personagens sejam controlados pelo programador de tal forma a criar uma atmosfera facilitadora de aprendizagem (VAHLDIK et al., 2008a).

O mundo do FURBOT se resume a uma matriz bidimensional, onde personagens, inimigos, obstáculos e objetos de interesse são inseridos no cenário a fim de determinar metas que os alunos deverão alcançar. Na Figura 6 é ilustrado um exemplo de exercício aplicado em sala de aula. Neste cenário o objetivo do robô é sair da mina, considerando que a direção de saída pode variar entre uma execução e outra, estando virada para cima, para baixo, para a direita ou para a esquerda.

Figura 6 - Exemplo de exercício proposto no FURBOT



No Quadro 2 é apresentado um exemplo de algoritmo que resolve o problema do exercício em questão. Nesta solução o robô procura a direção que não possui obstáculos. Ao encontrá-la, se movimenta naquela direção duas vezes, atingindo assim a meta definida pelo exercício.

Quadro 2 - Solução FURBOT para o problema do robô perdido na mina

```
if (ehVazio(ACIMA)) {
        andarAcima();
        andarAcima();
} else if (ehVazio(ABAIXO)) {
        andarAbaixo();
        andarAbaixo();
} else if (ehVazio(DIREITA)) {
        andarDireita();
        andarDireita();
} else {
        andarEsquerda();
        andarEsquerda();
}
```

O FURBOT é composto basicamente por uma biblioteca Java. Basta criar um projeto Java num ambiente de desenvolvimento e adicionar no *classpath* do projeto a biblioteca do FURBOT mais as suas dependências (VAHLDIK et al., 2008b). Em seguida, para elaboração do cenário é necessário que o professor esteja familiarizado com a biblioteca, pois a construção do cenário depende do mapeamento de objetos a partir de um arquivo XML.

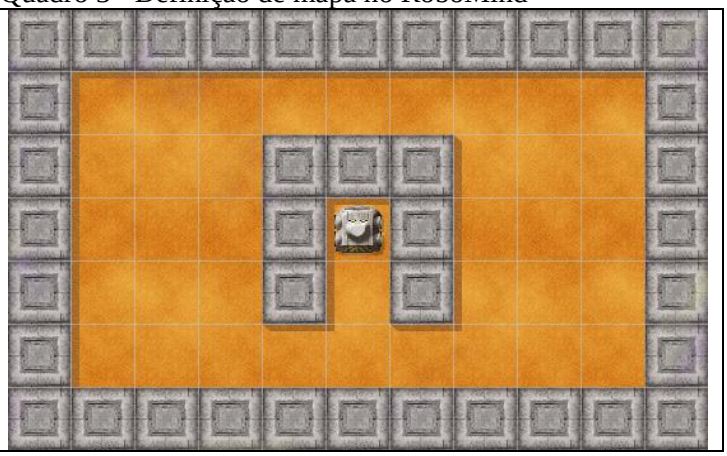
2.4.2 RoboMind FURB

O RoboMind é um ambiente de programação de robôs, desenvolvido por Arvid Hama na Universidade de Amsterdam, que faz uso da linguagem Robo, inspirada na linguagem Logo. Esta linguagem é extremamente simples e se mostrou capaz de promover o ensino-aprendizagem dos conceitos básicos de programação, inteligência artificial e robótica. O ambiente virtual do RoboMind define o sujeito programável como sendo um robô, que possui as ações de andar, olhar em volta, mover objetos e pintar. Tudo isso é realizado num cenário bidimensional, composto por várias células que podem ser preenchidas com alguns tipos de objetos (ROBOMIND ACADEMY, 2014).

No RoboMind é possível definir mapas através de um arquivo com a extensão `.map`. As células do cenário podem ser preenchidas com vários tipos de objetos, os quais são: o próprio robô, paredes, piscinas, plantas, caixas ou esferas que podem ser capturadas pelo robô (HALMAF, 2008). No Quadro 3 é definida a configuração de um cenário semelhante ao apresentado na seção anterior, sendo que no lado esquerdo é apresentado o conteúdo do arquivo `.map`, enquanto que no lado direito é apresentado o cenário gerado.

Quadro 3 - Definição de mapa no RoboMind

```
.map

AAAAAAAAAAA
A         A
A   AAA   A
A   A@A   A
A   A A   A
A         A
AAAAAAAAAAA
```



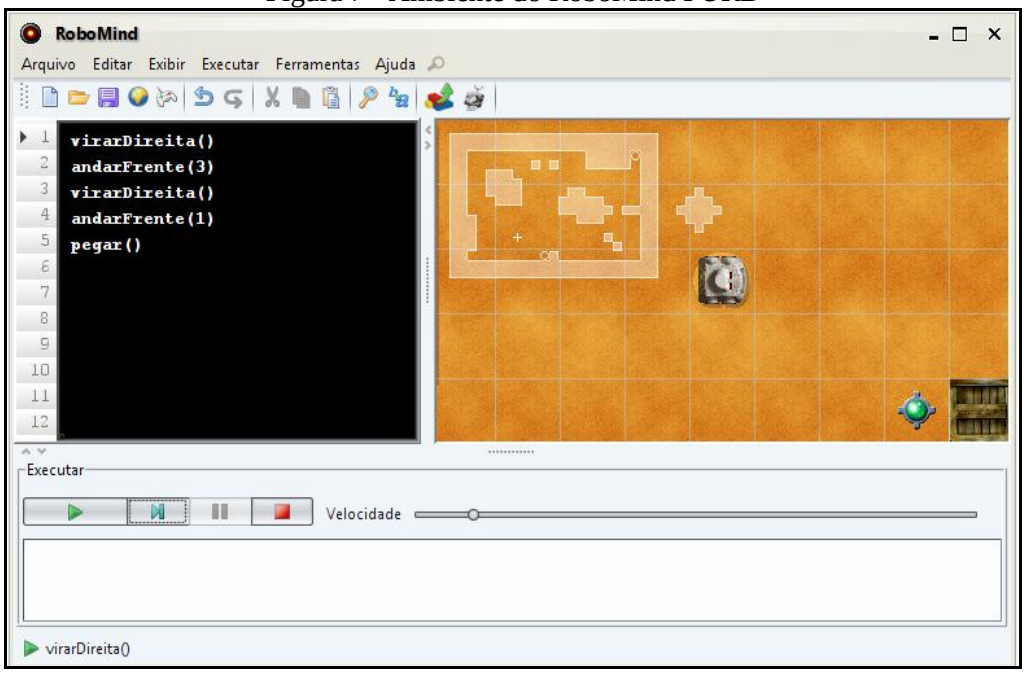
No Quadro 4 é apresentado um algoritmo em RoboMind que soluciona o problema do robô perdido na mina. Neste algoritmo, enquanto houver obstáculo o robô vira para a esquerda, no momento em que a passagem estiver livre, o robô anda duas vezes para frente.

Quadro 4 - Solução RoboMind para o problema do robô perdido na mina

```
repetirEnquanto(temObstáculoFrente) {
    virarEsquerda
}
andarFrente(2)
```

Por sua vez, o RoboMind FURB é uma adaptação do RoboMind original, que fornece um *plugin* para que os usuários possam testar seus programas em robôs físicos, montados com o LEGO Mindstorms NXT. Sendo assim, o aluno deve escolher um modelo de robô suportado pelo ambiente, para em seguida programar utilizando as ações que são disponibilizadas para aquele modelo de robô (BENITTI et al., 2010a). Na Figura 7 é apresentado o ambiente do RoboMind FURB.

Figura 7 - Ambiente do RoboMind FURB

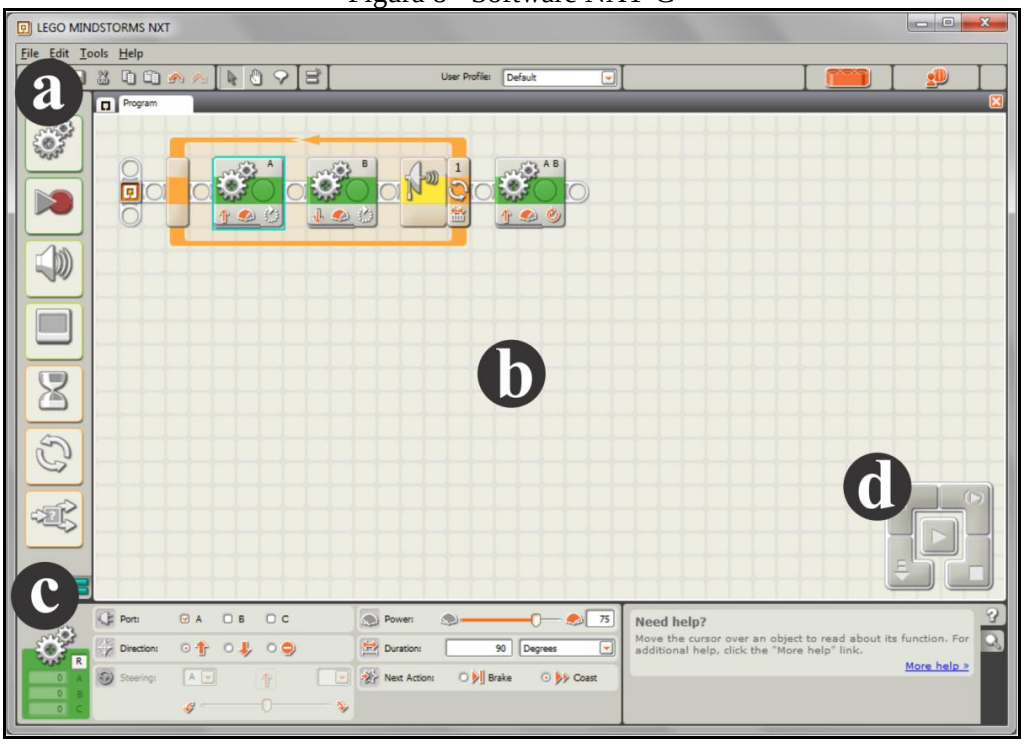


O ambiente é dividido entre o editor de código, o cenário em que o robô está inserido e o *console*, que dispõe de botões para iniciar, depurar, pausar e interromper o código em execução. Na barra de ferramentas é possível acessar uma janela para fazer a seleção do tipo de robô, que inclusive informa ao usuário as ações disponíveis para cada modelo.

2.4.3 NXT-G

O NXT-G é uma linguagem gráfica desenvolvida para o *kit* NXT. Com ela é possível criar programas e realizar a implantação do código para o robô, que executa o algoritmo em seguida. Essa linguagem, graficamente falando, faz alusão aos jogos de quebra-cabeça, em que as peças representam comandos que podem ser encaixados uns nos outros para formar algoritmos (KELLY, 2010, p. 5). Na Figura 8 é apresentada a interface do NXT-G, onde é possível visualizar: (a) paleta de comandos; (b) editor de código gráfico; (c) painel de propriedades do comando selecionado e (d) painel para execução do código.

Figura 8 - Software NXT-G



A paleta de comandos disponibiliza comandos de diversos tipos, sendo que os mais frequentemente utilizados são os comandos de: movimentação, gravação e execução de ações, emissão de sons, exibição de texto no *display*, espera, controle de fluxo (equivalente ao `while` em outras linguagens de programação) e seleção (equivalente ao `switch`).

A seleção de uma peça no editor de código implica na exibição do painel de propriedades, onde estão contidas as possibilidades de configuração da peça selecionada. Na Figura 9 é apresentado o painel de propriedades do comando de movimentação, onde é

possível, por exemplo, configurar os motores que serão rotacionados, a direção de locomoção, a duração do movimento, entre outros. Além disso, as configurações realizadas permanecem perceptíveis mesmo que a seleção do comando seja desfeita, pois o visual da peça no editor de código se molda de acordo com as propriedades, o que facilita a compreensão do que o comando faz sem ter que selecionar a peça para rever as propriedades.

Figura 9 - Painel de propriedades do comando de movimentação

Como exemplo de programa, também foi resolvido no NXT-G o exercício do robô perdido na mina. Na Figura 10 é apresentado o algoritmo adotado para solucionar o problema. O retângulo com borda laranjada representa um comando `while`, no qual, no canto direito, existe o desenho de uma antena que define a condição de parada. Essa antena, através de um valor definido no painel de propriedades, verifica a distância do robô frente a um obstáculo. Enquanto o obstáculo existir, o robô executa os comandos internos ao `while`. O primeiro comando faz com que a roda direita do robô rotacione para frente, em seguida o segundo comando faz com que a roda esquerda rotacione para trás, possibilitando que o robô vire para a esquerda a fim de encontrar o caminho com a passagem livre. Quando o robô encontra a passagem, o laço do `while` é encerrado, o que desencadeia a execução do último comando. O último comando aciona as rodas direita e esquerda, fazendo com que elas girem para frente permitindo que o robô saia da mina.

Figura 10 - Solução NXT-G para o problema do robô perdido na mina

3 DESENVOLVIMENTO DA FERRAMENTA

O capítulo está dividido em cinco seções, onde são apresentados:

a) os requisitos principais da ferramenta desenvolvida;
b) a especificação da linguagem, envolvendo os aspectos léxico, sintático e semântico;
c) a definição de um robô suportado, envolvendo a definição genérica do robô, além da implementação e montagem do robô LEGO Mindstorms NXT programado com a API leJOS;
d) a especificação da ferramenta;
e) a implementação, as ferramentas utilizadas e a operacionalidade da implementação;
f) os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

A ferramenta de programação deve:

a) disponibilizar uma linguagem de programação para que o usuário possa programar (Requisito Funcional - RF);
b) disponibilizar um *software* para que o usuário possa manipular os programas definidos na linguagem (RF);
c) disponibilizar uma camada intermediária que viabilize o suporte a várias plataformas de robôs (RF);
d) possuir a especificação da linguagem definida através do GALS (Requisito Não Funcional - RNF);
e) permitir que programas definidos na linguagem possam ser implantados em robôs LEGO Mindstorms NXT (RNF);
f) realizar a implantação de código no robô LEGO Mindstorms NXT através da API leJOS (RNF);
g) possuir a inteligência do robô LEGO Mindstorms NXT implementada com a API leJOS (RNF);
h) disponibilizar uma tela para configuração do robô LEGO Mindstorms NXT (RNF).

3.2 ESPECIFICAÇÃO DA LINGUAGEM ROBOTOY

Inicialmente, nesta seção são apresentadas as especificações léxica, sintática e semântica da linguagem definida, bem como o detalhamento dos comandos e o código correspondente em Java.

3.2.1 Especificação da linguagem: aspectos léxicos

A especificação léxica da linguagem conta com cinco definições regulares, que determinam o que é dígito, letra, alfanumérico e comentários de linha e bloco. No Quadro 5 são apresentadas as definições regulares.

Quadro 5 - Definições regulares

```
digito : [0-9]
letra : [A-z Á Ã Â À É Ê Í Ó Õ Ô Ú Ç á ã â à é ê í ó õ ô ú ç]
alfanumerico : {digito} | {letra}
comentario_de_linha : / / ([^\n])*
comentario_de_bloco : / \* ([^ \* /])* \* /
```

Um dígito é qualquer número de 0 a 9. A letra é qualquer letra maiúscula ou minúscula, incluindo acentuações. O alfanumérico é qualquer dígito ou letra. O comentário de linha é iniciado com os caracteres `//`, seguido de zero ou mais caracteres que não correspondam à quebra de linha. O comentário de bloco é demarcado pelos símbolos `/*` e `*/`, contendo em seu interior zero ou mais caracteres, incluindo quebra de linha e excluindo `*` e `/`.

Existem alguns componentes que são ignorados quando encontrados. No caso da linguagem Robotoy, são os espaços em branco, as tabulações e os comentários de linha e bloco. Geralmente a quebra de linha também é ignorada em outras linguagens, mas esse não é o caso pelo fato do símbolo terminal ser a quebra de linha, enquanto que em outras linguagens normalmente é utilizado o ponto e vírgula.

A linguagem faz uso de expressões regulares para constituir tipos numéricos e literais, além de identificadores. No Quadro 6 são apresentadas as expressões regulares da linguagem.

Quadro 6 - Expressões regulares

```
numerico : {digito}+ (, {digito}+)?
literal : \"[^\n\"]*\"
identificador : {letra}+ {alfanumerico}*
```

A expressão regular `numerico` define o que é um número na linguagem, equivalente ao `double` no Java, que pode ser tanto um número inteiro ou decimal, com várias casas, onde o separador decimal é denotado por vírgula, diferente do Java que é denotado por ponto. Na sequência tem-se a expressão regular `literal`, equivalente à `String` no Java, onde o `literal` é demarcado por `"` e `"`, contendo em seu interior zero ou mais caracteres, com exceção da quebra de linha. Por fim, é apresentada a expressão regular `identificador`, que

define o padrão de formação para escrever palavras reservadas, nome de variável e nome de rotina, sendo que o nome atribuído às variáveis e rotinas não podem competir com as palavras reservadas da linguagem. Um `identificador` pode ser qualquer sequência que inicia com uma ou mais letras, minúsculas ou maiúsculas, com inclusão das acentuações, que pode conter zero ou mais caracteres alfanuméricos. No Quadro 7 são apresentadas as palavras reservadas da linguagem, baseadas na expressão regular `identificador`.

Quadro 7 - Palavras reservadas

a	da	esquerda	número	rotina	vermelha
amarela	de	fim	obstáculo	se	vermelho
amarelo	direita	frente	ou	senão	
andar	do	girar	para	som	
azul	e	identificada	parar	tem	
branca	emitir	motor	preta	texto	
branco	enquanto	multiuso	preto	trás	
cor	escrever	não	roda	verde	

Os operadores lógicos, relacionais, aritméticos e de atribuição utilizados pela linguagem estão relacionados no Quadro 8.

Quadro 8 - Operadores da linguagem

Operador	Descrição
+	adição
-	subtração
*	multiplicação
/	divisão
%	resto da divisão
(	abre parênteses
)	fecha parênteses
>	maior que
<	menor que
>=	maior ou igual
<=	menor ou igual
=	igual
=/	diferente de
<-	operador de atribuição
.	operador de concatenação de literais

3.2.2 Especificação da linguagem: aspectos sintáticos

A linguagem é composta por uma lista de comandos, seguida de uma lista de rotinas que também podem conter comandos. Os comandos são separados por quebra de linha. Desta forma, não há necessidade de utilizar um símbolo especial para indicar o fim do comando, como no caso do ponto e vírgula normalmente usado em outras linguagens. Os comandos disponibilizados pela linguagem podem ser classificados nos seguintes tipos: controle de fluxo, que depende da definição de uma sentença condicional para ser executado; declaração de variável, que permite o armazenamento de informações; definição e invocação de rotinas, que permitem evitar a duplicação de código além de organizar as partes que compõem o programa e, por último, comandos direcionados ao robô, que permitem controlar o robô no

cenário em que foi inserido. No Apêndice A encontra-se a gramática com todas as construções sintáticas da linguagem.

3.2.3 Especificação da linguagem: aspectos semânticos e geração de código

A especificação semântica garante a correta geração de código. Tem-se as seguintes validações sob responsabilidade do analisador semântico:

a) o uso de variáveis que não foram declaradas;
b) a declaração duplicada de variáveis;
c) o uso de rotinas que não foram declaradas;
d) a declaração duplicada de rotinas;
e) a compatibilidade de tipos na atribuição de variáveis: `número` só pode ser atribuído a `número` ou `texto`; `cor` só pode ser atribuído a `cor` ou `texto` e `texto` só pode ser atribuído a `texto`;
f) o uso dos sinais unários, cujos operandos devem ser expressões numéricas;
g) a quantidade de sinais unários usados limitada a um;
h) o uso do operador de concatenação de literais . (ponto), cujos operandos devem ser expressões literais;
i) o uso dos operadores aritméticos, cujos operandos devem ser expressões numéricas;
j) a compatibilidade de tipos dos operandos dos operadores relacionais: `número` só pode ser comparado com `número`; `cor` só pode ser comparado com `cor` e `texto` só pode ser comparado com `texto`, sendo que `cor` e `texto` só podem ser comparados com = (igual) ou =/ (diferente de);
k) o tipo de parâmetro passado aos comandos de ação do robô: as ações, com exceção do comando `escrever`, devem receber uma expressão numérica, enquanto que o comando `escrever` deve recebe uma expressão literal.

Uma vez definidas as validações semânticas, cada estrutura sintática da linguagem, seguindo essas validações, é traduzida para código Java. Os comandos da linguagem em geral dependem de expressões para serem reconhecidos, sendo que essas expressões podem variar bastante. As expressões podem ser classificadas de acordo com o exposto no Quadro 9, que apresenta também o respectivo código em Java, gerado pela ferramenta.

Quadro 9 - Expressões da linguagem

<valor numérico>

Descrição:
Denota um número inteiro ou decimal, baseado na definição da expressão regular `numerico`.

Exemplo:
```
10,75
```

Código correspondente em Java:
```
(double) 10.75
```

<variável numérica>

Descrição:
Faz referência a uma variável numérica através do nome da variável.

Exemplo:
```
número valor <- 10
valor <- valor + 1
```

Código correspondente em Java:
```
valor = (double) 10;
valor = (double) valor + 1;
```

<expressão numérica>

Descrição:
Relaciona valores numéricos e variáveis numéricas utilizando operadores aritméticos, assim como os sinais unários, além de parênteses para priorizar a ordem das operações.

Exemplo:
```
valor <- (10 * 2) % valor
```

Código correspondente em Java:
```
valor = (double) (10 * 2) % valor;
```

<valor literal>

Descrição:
Denota um literal, baseado na definição da expressão regular `literal`.

Exemplo:
```
"exemplo"
```

Código correspondente em Java:
```
"exemplo"
```

<variável literal>

Descrição:
Faz referência a uma variável literal através do nome da variável.

Exemplo:
```
texto nome <- "mundo"
escrever nome
```

Código correspondente em Java:
```
nome = "mundo";
_ROBOT.write(nome);
```

<expressão literal>

Descrição:
Concatena valores independente de tipo, utilizando para isso o operador de concatenação de literal . (ponto), sendo que também é possível utilizar parênteses na concatenação de valores.

Exemplo:
```
nome <- "Olá " . nome . " " . (verde)
```

Código correspondente em Java:
```
nome = "Olá " + nome + " " + "verde";
```

Como já mencionado, os comandos da linguagem podem ser comandos direcionados ao robô ou comandos de controle de fluxo, declaração de variáveis e definição e invocação de rotinas.

Os comandos direcionados ao robô foram classificados nos seguintes tipos:

a) comunicação: permite que o robô possa se comunicar através da escrita ou de sons;
b) orientação: permite que o robô possa se orientar no meio em que foi inserido através da verificação de obstáculos;
c) detecção: permite que o robô possa detectar cores no meio em que foi inserido;
d) locomoção: permite que o robô possa se movimentar no meio em que foi inserido.

A seguir são detalhados os tipos de comandos direcionados ao robô. Primeiramente, são apresentados os comandos de comunicação, incluídos no Quadro 10. Na sequência são apresentados o comando de orientação, incluído no Quadro 11, e o comando de detecção, incluído no Quadro 12.

Quadro 10 - Comandos de comunicação do robô

`escrever <texto>`

Descrição:
Este comando permite que o robô escreva um texto no *display*, sendo que:
`<texto>::= <valor literal> | <variável literal> | <expressão literal>`

Exemplo:
```
escrever "Olá mundo."
```

Código correspondente em Java:
```
_ROBOT.write("Olá mundo.");
```

`emitir som`

Descrição:
Este comando permite que o robô emita um som.

Exemplo:
```
emitir som
```

Código correspondente em Java:
```
_ROBOT.beep();
```

Quadro 11 - Comando de orientação do robô

`<negação> tem obstáculo`

Descrição:
Este comando faz com que o robô verifique a existência de um obstáculo com auxílio do sensor de distância, sendo que:
`<negação>::= ε | não`
Os comandos `<se>` e `<enquanto>` estão diretamente relacionados a esse comando, ou seja, o comando só pode ser utilizado em conjunto com um dos comandos mencionados.

Exemplo:
```
enquanto não tem obstáculo
    andar para frente 1
fim do enquanto
```

Código correspondente em Java:
```
while (!_ROBOT.hasObstacle()) {
    _ROBOT.walkForward(Math.abs(1));
}
```

Quadro 12 - Comando de detecção do robô

`cor identificada`

Descrição:
Este comando permite que o robô identifique a cor capturada pelo sensor de cor. É possível utilizar esse comando na declaração de variáveis ou na comparação de cores nos comandos `<se>` e `<enquanto>`. As cores suportadas pela linguagem são branco, preto, azul, verde, vermelho e amarelo. Caso seja identificada uma cor diferente das citadas, é realizada uma verificação para determinar qual das cores suportadas se assemelha mais à `cor identificada`. Diante disso, o robô considera a cor semelhante como `cor identificada`.
Exemplo:
`cor tom <- cor identificada`
`se tom = cor identificada`
`  virar para a esquerda 1`
`fim do se`
Código correspondente em Java:
`tom = _ROBOT.detectColor();`
`if (tom.getColor() == _ROBOT.detectColor().getColor()) {`
`  _ROBOT.turnLeft(Math.abs(1));`
`}`

O Quadro 13 traz os comandos de locomoção do robô.

Quadro 13 - Comandos de locomoção do robô

`andar para <direção> <quantidade>`

Descrição:
Este comando faz com que o robô ande numa direção, sendo que:
`<direção>::= frente | trás`
`<quantidade>::=ε| <valor numérico> | <variável numérica> | <expressão numérica>`
Caso seja informado `ε`, o robô andará constantemente na `<direção>` informada.
Exemplo:
`andar para frente 5`
Código correspondente em Java:
`_ROBOT.walkForward(Math.abs(5));`

`parar de andar`

Descrição:
Faz com que o robô pare de andar caso o mesmo esteja andando.
Exemplo:
`parar de andar`
Código correspondente em Java:
`_ROBOT.stopWalking();`

`virar para a <direção> <quantidade>`

Descrição:
Este comando faz com que o robô vire 90° para uma direção, sendo que:
`<direção>::= esquerda | direita`
`<quantidade>::=ε| <valor numérico> | <variável numérica> | <expressão numérica>`
Caso seja informado `ε`, o robô virará constantemente para a `<direção>` informada.
Exemplo:
`virar para a direita 2 + 3`
Código correspondente em Java:
`_ROBOT.turnRight(Math.abs(2 + 3));`

`parar de virar`

Descrição:
Faz com que o robô pare de virar caso o mesmo esteja virando.
Exemplo:
```
parar de virar
```
Código correspondente em Java:
```
_ROBOT.stopTurning();
```

`virar motor multiuso para a <direção> <quantidade>`

Descrição:
Este comando faz com que o motor multiuso vire 90° para uma direção, sendo que:
```
<direção>::= esquerda | direita
<quantidade>::= <valor numérico> | <variável numérica> | <expressão numérica>
```
Neste comando o ε não é suportado porque é permitido o encaixe de um sensor de cor ou distância no motor, o que acarreta na inclusão de um cabo para conectar o sensor ao *brick*. Se fosse permitido ao motor virar constantemente, a estrutura física do robô seria comprometida quando a quantidade de giros superasse o comprimento do cabo conectado ao sensor. Apesar de o motor não virar constantemente, nada impede que o usuário declare uma alta quantia de giros para o comando. Diante disso, é necessário que o usuário esteja ciente dessa limitação e manuseie o comando com cuidado.
Exemplo:
```
virar motor multiuso para a esquerda 1
```
Código correspondente em Java:
```
_ROBOT.turnMultiuseMotorLeft(Math.abs(1));
```

`girar roda <lado> para <direção>`

Descrição:
Este comando faz com que o robô gire constantemente uma das rodas para uma direção, sendo que:
```
<lado>::= esquerda | direita
<direção>::= frente | trás
```
Exemplo:
```
girar roda esquerda para trás
```
Código correspondente em Java:
```
_ROBOT.spinLeftWheelBackward();
```

`parar de girar`

Descrição:
Este comando faz com que o robô cesse o giro das rodas em movimento.
Exemplo:
```
parar de girar
```
Código correspondente em Java:
```
_ROBOT.stopSpinning();
```

Além dos comandos direcionados ao robô, existem os comandos para: declaração de variáveis, atribuição de valores a variáveis, comandos para controle de fluxo e declaração e invocação de rotinas. A seguir são detalhados os tipos de comandos citados. Primeiramente, são apresentados os comandos de declaração de variáveis, incluídos no Quadro 14.

Quadro 14 - Declaração de variáveis

`cor <nome> <- <valor>`

Descrição:

Este comando realiza a declaração de uma variável do tipo `cor`, sendo que:

```
<nome>::= identificador
<valor>::= branco | branca | preto | preta | amarelo | amarela | vermelho |
           vermelha | verde | azul | cor identificada | <variável do tipo cor>
```

Exemplo:

```
cor tom <- amarela
```

Código correspondente em Java:

```
tom = new ColorDefinition("amarela", Color.YELLOW);
```

`número <nome> <- <valor>`

Descrição:

Este comando realiza a declaração de uma variável do tipo `número`, sendo que:

```
<nome>::= identificador
<valor>::= <valor numérico> | <variável numérica> | <expressão numérica>
```

Na declaração de variáveis numéricas é possível elaborar operações aritméticas, onde os operadores aritméticos são `+` (adição), `-` (subtração), `/` (divisão), `*` (multiplicação) ou `%` (resto da divisão). Além desses operadores, também é possível adicionar parênteses para definir a ordem de execução das operações aritméticas.

Exemplo:

```
número valor <- 5
número outroValor <- (valor + 5) * 2,5
```

Código correspondente em Java:

```
valor = (double) 5;
outroValor = (double) (valor + 5) * 2.5;
```

`texto <nome> <- <valor>`

Descrição:

Este comando realiza a declaração de uma variável do tipo `texto`, sendo que:

```
<nome>::= identificador
<valor>::= <valor literal> | <variável literal> | <expressão literal> |
           todos os valores válidos para variáveis do tipo cor |
           todos os valores válidos para variáveis do tipo número, onde
           expressões devem ser colocadas entre parênteses
```

A variável do tipo `texto` é a mais completa entre todos os tipos, pois aceita cores e valores numéricos na sua declaração. Essa característica é fundamental, pois não seria possível exibir textos elaborados no *display* caso a linguagem não realizasse a conversão de tipos não literais para literais.

Além disso, a declaração de variáveis do tipo `texto` conta com o apoio de um operador de concatenação, denotado por `.` (ponto). Esse operador possibilita a junção de valores literais, cores e valores numéricos.

Exemplos:

```
número valor <- -10,0
texto umTexto <- "A quantidade é: " . valor

cor tom <- cor identificada
texto umTexto <- "A cor identificada é " . tom . "."

texto umTexto <- "O valor encontrado é: " . ((10 + 20) * 2)
```

Código correspondente em Java:

```
valor = (double) -10.0;
umTexto = "A quantidade é" + Double.toString(valor);

tom = _ROBOT.detectColor();
umTexto = "A cor identificada é" + tom.getName() + ".";

umTexto = "O valor encontrado é: " + Double.toString((10 + 20 * 2));
```

Na sequência é apresentado o comando de atribuição de variáveis, incluído no Quadro 15.

Quadro 15 - Atribuição de variáveis

```
<nome> <- <valor>
```

Descrição:

A atribuição de variáveis está diretamente associada à declaração de variáveis, só é possível fazer uma atribuição se a variável em questão estiver declarada, caso contrário o comando resultará num erro semântico, sendo que:

```
<nome>::= <identificador>
<valor>::= valor compatível com o tipo da variável já declarada, se essa condição não for respeitada acarretará num erro semântico
```

Exemplos:

```
número valor <- 25,25
valor <-  5 * 2

cor tom <- cor identificada
tom <- verde

texto palavra <- "Olá "
palavra <- palavra . "mundo."
```

Código correspondente em Java:

```
valor = (double) 25.25;
valor = (double) 5 * 2;

tom = _ROBOT.detectColor();
tom = new ColorDefinition("verde", Color.GREEN);

palavra = "Olá ";
palavra = palavra + "mundo.";
```

Os comandos para controle de fluxo encontram-se no Quadro 16.

Quadro 16 - Comandos para controle de fluxo

```
se <condição>
      <lista de comandos>
senão
      <lista de comandos>
fim do se
```

Descrição:

O comando `<se>` depende de uma condição para ser executado, se a `<condição>` for verdadeira, a lista de comandos é executada. Por outro lado, caso a `<condição>` seja falsa, a lista de comandos subsequente ao `senão` é executada. A cláusula `senão <lista de comandos>` é opcional. Portanto, o usuário pode declarar o comando `<se>` com ou sem o `senão`, sendo que:

```
<condição>::= <comparação de números> | <comparação de cores> |
              <comparação de textos> | <verificação de obstáculo>
<lista de comandos>::= ε | <se> | <enquanto> | <chamada de rotina> |
                       <comando do robô> | <declaração de variável> |
                       <atribuição de variável>
```

É necessário respeitar algumas regras durante a definição da condição, tais como:

a) o operador relacional para comparação de números, que pode ser `<` (menor que), `>` (maior que), `<=` (menor ou igual), `>=` (maior ou igual), `=` (igual a) ou `=/` (diferente de);
b) o operador relacional para comparação de cores ou textos, que pode ser `=` ou `=/`.

Além disso, é possível adicionar condições lógicas usando os operadores lógicos `e` ou `ou`. No caso do `e`, para que a lista de comandos seja executada, é necessário que as duas condições ligadas pelo operador lógico sejam verdadeiras, no caso do `ou` basta que apenas uma condição seja verdadeira.

Exemplos:

```
se tem obstáculo
    andar para frente 1
senão
    andar para trás 1
fim do se

se não tem obstáculo e cor identificada = verde
    andar para frente 1
fim do se

número valor <- 5
se valor > 5 ou valor < 10
    emitir som
fim do se

texto cumprimento <- "Olá"
se cumprimento = "Oi" ou cumprimento = "Olá" e cor identificada =/ amarela
    escrever "Olá mundo."
fim do se
```

Código correspondente em Java:

```
if (_ROBOT.hasObstacle()) {
    _ROBOT.walkForward(Math.abs(1));
} else {
    _ROBOT.walkBackward(Math.abs(1));
}

if (!_ROBOT.hasObstacle() && _ROBOT.detectColor().getColor() == Color.GREEN) {
    _ROBOT.walkForward(Math.abs(1));
}

valor = (double) 5;
if (valor > 5 || valor < 10) {
    _ROBOT.beep();
}

cumprimento = "Olá";
if (cumprimento.equals("Oi") || cumprimento.equals("Olá") &&
_ROBOT.detectColor().getColor() != Color.YELLOW) {
    _ROBOT.write("Olá mundo.");
}
```

```
enquanto <condição>
      <lista de comandos>
fim do enquanto
```

Descrição:

O comando `<enquanto>` segue os mesmos princípios do comando `<se>`, o que difere é a quantidade de vezes que a lista de comandos pode ser executada, enquanto que o `<se>` executa somente uma vez, o `<enquanto>` executa repetidamente até que o valor `<condição>` seja falso.

Exemplos:

```
número célulasPercorridas <- 0
enquanto não tem obstáculo
    célulasPercorridas <- célulasPercorridas + 1
fim do enquanto

número valor <- 5
enquanto valor =/ 0
    valor <- valor - 1
fim do enquanto

enquanto não tem obstáculo e cor identificada =/ branco
    andar para trás 1
fim do enquanto
```

Código correspondente em Java:

```
célulasPercorridas = (double) 0;
while (!_ROBOT.hasObstacle()) {
    célulasPercorridas = (double) célulasPercorridas + 1;
}

valor = (double) 5;
while (valor != 0) {
    valor = (double) valor - 1;
}

while (!_ROBOT.hasObstacle() && _ROBOT.detectColor().getColor() != Color.
WHITE) {
    _ROBOT.walkBackward(Math.abs(1));
}
```

Na sequência é apresentada a declaração de rotinas, incluída no Quadro 17.

Quadro 17 - Declaração de rotinas

```
rotina <nome>
      <lista de comandos>
fim da rotina
```

Descrição:

As rotinas, que em outras linguagens atendem pelo nome de métodos ou *procedures*, consistem no agrupamento de uma sequência de comandos comumente invocados ao longo do código. Desta forma, é possível desenvolver um código mais organizado, legível, sem duplicações e com menos incidência de erros, uma vez que parte da lógica do programa se concentra em apenas um lugar. A declaração de rotinas na linguagem é relativamente simples, pois dispensa a passagem de parâmetros, sendo que:

```
<nome>::= identificador
<lista de comandos>::= ε | <se> | <enquanto> | <chamada de rotina> |
                       <comando do robô> | <declaração de variável> |
                       <atribuição de variável>
```

Para invocar uma rotina na linguagem basta escrever numa linha o nome atribuído à rotina durante a sua declaração. É importante ressaltar que não é possível declarar duas rotinas com o mesmo nome.

Exemplo:

```
rotina1
rotina2
rotina rotina1
    rotina2
fim da rotina
rotina rotina2
    rotina1
fim da rotina
```

Código correspondente em Java:

```
private static final LejosRobot _ROBOT = new LejosRobot();

public static void main(String[] args) {
    try {
          _ROBOT.onStart();
          rotina1();
          rotina2();
          _ROBOT.onEnd();
    } catch (Exception e) {
          _ROBOT.manageException(e);
    }
}

private static void rotina1() throws Exception {
    rotina2();
}
```

```
    private static void rotina2() throws Exception {
        rotina1 ();
    }
```

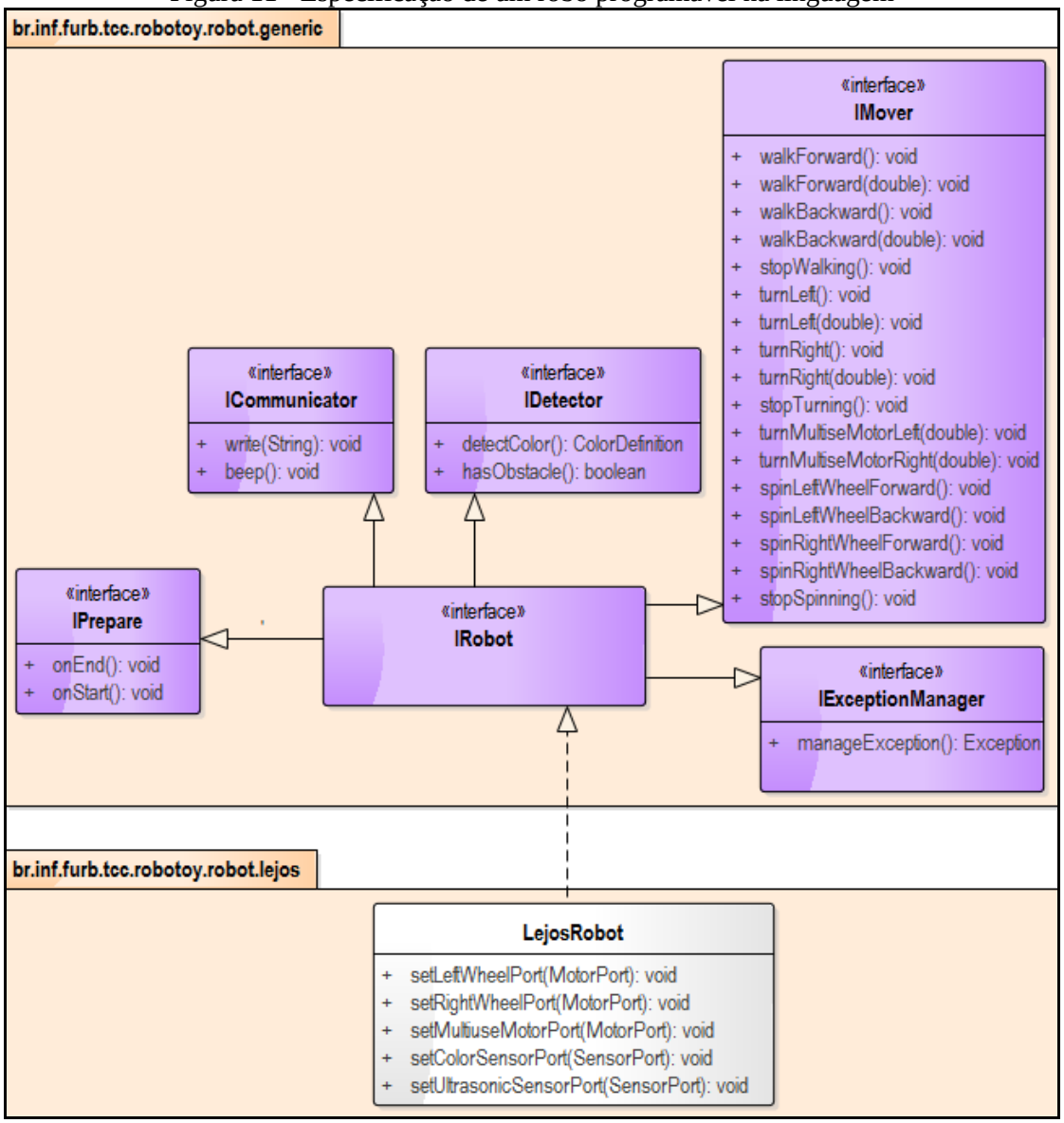
3.3 ROBÔ ROBOTOY

Esta seção apresenta as características de um robô suportado pela linguagem. Em seguida é apresentada a implementação e a montagem do robô LEGO Mindstorms NXT.

3.3.1 Especificação genérica de um robô

A linguagem foi planejada para não ser exclusiva para a plataforma LEGO Mindstorms NXT. Para isso, foi necessário definir uma interface que determina o que é um robô suportado pela linguagem, conforme ilustrado no diagrama de classes apresentado na Figura 11.

Figura 11 - Especificação de um robô programável na linguagem



Para que um robô seja suportado pela linguagem, é necessário que duas regras sejam respeitadas: o robô deve implementar a interface `IRobot` e a classe `LejosRobot` deve possuir apenas um construtor sem parâmetros. Essas restrições garantem o funcionamento do gerador de código, que depende dessas características para instanciar o objeto robô na classe gerada.

A interface `IRobot` implementa outras interfaces que são responsáveis por definir comportamentos e características do robô, onde:

a) `IPrepare`: permite que o robô realize configurações antes (`onStart`) e depois (`onEnd`) do programa definido pelo usuário ser executado. Isto é, o `onStart` é executado no início do programa, em seguida é executado o algoritmo do usuário e, por fim, é executado o `onEnd`. Na implementação do robô leJOS, por exemplo, o `onStart` foi utilizado para diminuir a velocidade de rotação dos motores além de invocar o método `Button.waitForAnyPress()` do leJOS. Esse mesmo método também foi invocado no `onEnd`. Nos dois casos, o `Button.waitForAnyPress()` garante tempo ao programador para realizar algo, como por exemplo, tempo para posicionar o robô no cenário (`onStart`) ou tempo para visualizar o *display* ao término do programa (`onEnd`);

b) `IMover`: permite que o robô se movimente dentro do cenário com o auxílio de três motores, identificados como roda direita, roda esquerda e motor multiuso;

c) `ICommunicator`: permite que o robô se comunique pela escrita ou pela emissão de sons;

d) `IDetector`: permite que o robô realize detecções. Atualmente essa interface disponibiliza duas ações de detecção, uma das ações trata a verificação de obstáculos via sensor de distância, enquanto que a outra trata a detecção de cores via sensor de cor;

e) `IExceptionManager`: permite que o robô realize o tratamento de exceções. Na implementação do robô leJOS esse método imprime a mensagem da exceção no *display* e invoca o método `Button.waitForAnyPress()` do leJOS, permitindo que o programador possa ler a mensagem de exceção antes de finalizar o programa.

3.3.2 Implementação e montagem do robô LEGO Mindstorms NXT

A construção do robô LEGO Mindstorms não tem restrições quanto aos componentes usados, sendo o único obrigatório na montagem do robô o *brick*, pois os motores e os sensores são opcionais. Entretanto, é fundamental que o usuário respeite as limitações do modelo montado. Por exemplo, se não houver sensor de cor conectado, não será possível utilizar o

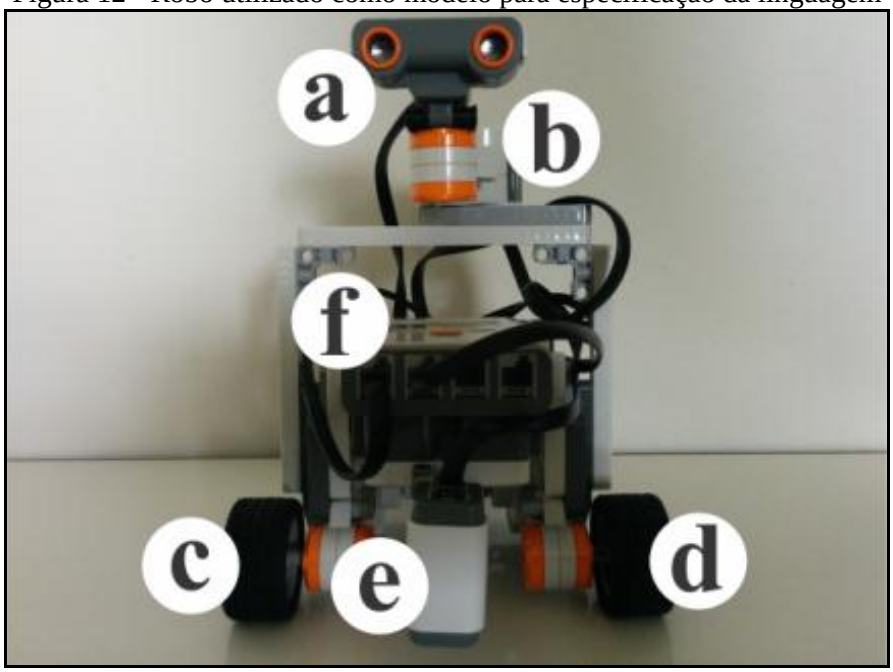
comando `cor identificada`. Mesmo assim, se o usuário insistir na utilização desse comando, será apresentada uma mensagem de erro no *display* no momento em que a instrução for executada. No Quadro 18 é apresentada a lista de comandos e os componentes dos quais dependem para a correta execução.

Quadro 18 - Lista das dependências de componentes eletrônicos para um comando

comando	componente
`escrever <texto>`	*display*
`emitir som`	caixa de som
`<negação> tem obstáculo`	sensor ultrassônico
`cor identificada`	sensor de cor
`andar para <direção> <quantidade>`	roda direita ou esquerda
`parar de andar`	
`virar para <direção> <quantidade>`	
`parar de virar`	
`girar <lado> para a <direção>`	
`parar de girar`	
`virar motor multiuso para a <direção> <quantidade>`	motor multiuso

Apesar da montagem do robô não ter restrições, a linguagem foi construída levando em consideração um modelo de robô que se assemelha a um carro. Portanto, a programação na linguagem fará mais sentido se essa questão for levada em consideração no momento de montar o robô. Na Figura 12 é apresentado o modelo de robô que auxiliou o processo de levantamento e especificação dos comandos da linguagem.

Figura 12 - Robô utilizado como modelo para especificação da linguagem



Na Figura 12 é possível observar: (a) sensor ultrassônico, para detecção de distância; (b) motor multiuso, para virar o sensor ultrassônico aumentando a mobilidade do robô; (c)

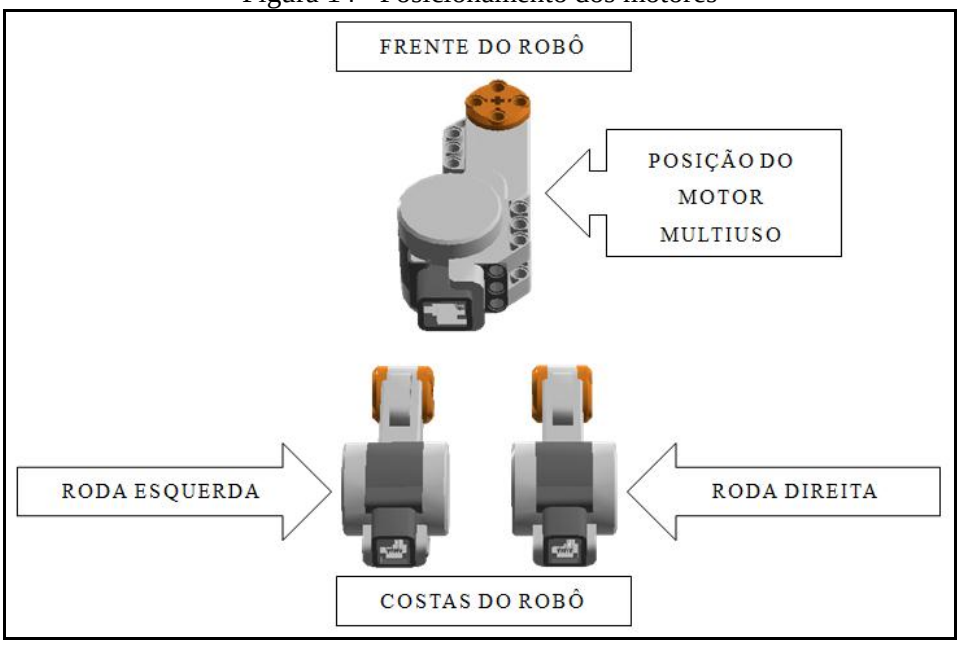
roda direita paralela à roda esquerda; (d) roda esquerda paralela à roda direita; (e) sensor de cor voltado para o chão e o (f) *brick.*

Um segundo modelo de robô é ilustrado na Figura 13 para mostrar que os modelos de robôs suportados pela linguagem podem variar. Em relação ao modelo principal, o segundo modelo é relativamente simples, possuindo apenas o *brick* e um sensor de cor conectado.

Figura 13 - Robô identificador de cor

Existem algumas restrições quanto ao posicionamento de motores, pois dependendo de como os motores são posicionados, as direções adotadas podem inverter-se, ou seja, frente vira trás, trás vira frente, direita vira esquerda e esquerda vira direita. Na Figura 14 é apresentado o posicionamento das rodas e do motor multiuso baseado no robô utilizado como modelo para especificação da linguagem. Nessa figura é como se o robô fosse visto de trás. A figura define como devem ficar posicionados os motores quando visualizados de costas.

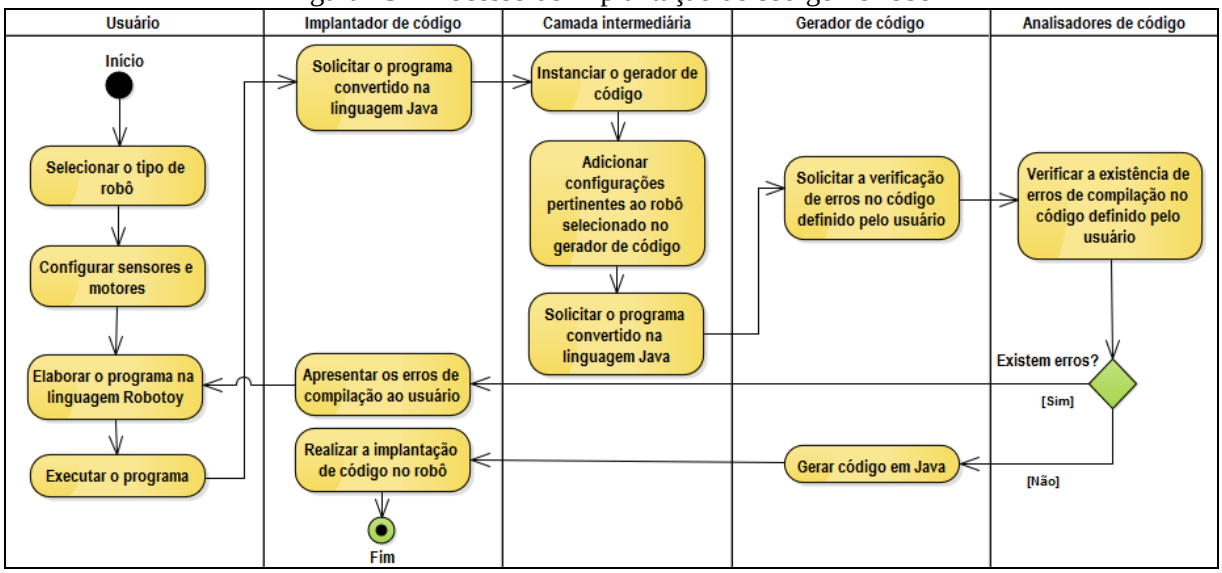
Figura 14 - Posicionamento dos motores



3.4 ESPECIFICAÇÃO DA FERRAMENTA

Para facilitar o entendimento da ferramenta como um todo é necessário explicar o funcionamento do processo de implantação de código no robô, no intuito de evidenciar as partes que compõem este processo e que são detalhadas posteriormente. Para isso, foi adotado o diagrama de atividades apresentado na Figura 15.

Figura 15 - Processo de implantação de código no robô



Primeiramente, o usuário escolhe o tipo de robô, o configura, elabora um programa e o executa, o que dá início ao processo de implantação estabelecido pelo implantador de código. O implantador de código tem a responsabilidade de distinguir o robô escolhido pelo usuário para, a partir daí, identificar o integrador – membro da camada intermediária – responsável pela geração de código daquele modelo. No momento em que o integrador é identificado, é dado início ao processo de compilação realizado pelo próprio integrador. No caso da plataforma LEGO Mindstorms, o integrador utilizado é o `LejosIntegrator`, que instancia o gerador de código e inclui informações adicionais no conteúdo da classe Java a ser gerada. No Quadro 19 é destacado o código que o `LejosIntegrator` insere na classe Java, que diz respeito ao mapeamento de sensores e motores para funcionamento do robô LEGO Mindstorms.

Quadro 19 - Destaque do código adicionado pelo `LejosIntegrator` na classe Java

```
1  import lejos.nxt.SensorPort;
2  import lejos.nxt.MotorPort;
3  import br.inf.furb.tcc.robotoy.robot.lejos.LejosRobot;
4
5  public final class Program {
6
7        private static final LejosRobot _ROBOT = new LejosRobot();
8
9        public static void main(String[] args) {
10             try {
11                   _ROBOT.setColorSensorPort(SensorPort.S3);
12                   _ROBOT.setUltrasonicSensorPort(SensorPort.S2);
13                   _ROBOT.setMultiuseMotorPort(MotorPort.C);
14                   _ROBOT.setLeftWheelPort(MotorPort.B);
15                   _ROBOT.setRightWheelPort(MotorPort.A);
16                   _ROBOT.onStart();
17                   _ROBOT.onEnd();
18             } catch (Exception e) {
19                   _ROBOT.manageException(e);
20             }
21       }
22
23 }
```

A camada intermediária existe pelo fato da linguagem não ser exclusiva da plataforma LEGO Mindstorms, o que desencadeia a especificação de um componente que adapte o conteúdo do gerador de código para garantir que a classe gerada seja compatível com o robô escolhido pelo usuário. A camada intermediária, após adaptar o conteúdo do gerador de código, solicita ao mesmo o arquivo Java a ser implantado no robô, o que dá início ao processo de geração de código. Este processo ocorre em conjunto com as análises léxica, sintática e semântica, que procuram por erros de compilação no programa. Caso algum erro seja encontrado, o processo é abortado e o erro é exibido ao usuário.

Em relação ao gerador de código, é importante dizer que o mesmo está inserido dentro do analisador semântico. No analisador semântico as sentenças são descobertas realizando-se a leitura do programa da esquerda para a direita. Quando a sentença analisada é considerada válida, o gerador de código é autorizado a gerar o código correspondente em Java, utilizando para isso o `JavaClassGenerator`. O `JavaClassGenerator` viabiliza a geração de um arquivo Java simplificado, que se adequa à realidade da linguagem Robotoy limitada ao tratamento de exceções, declaração e alteração de variáveis, declaração e invocação de métodos, manipulação de tipos primitivos, `String` e objetos construídos especialmente para a linguagem, além da utilização de cláusulas para controle de fluxo. No Quadro 20 é apresentado um programa construído na linguagem Robotoy equiparado ao código correspondente em Java.

Quadro 20 - Programa Robotoy equiparado ao código correspondente em Java

Programa na linguagem Robotoy

```
andarEnquantoNãoTemObstáculo

rotina andarEnquantoNãoTemObstáculo
       número passos <- 1
       enquanto não tem obstáculo
              andar para frente passos
       fim do enquanto
fim da rotina
```

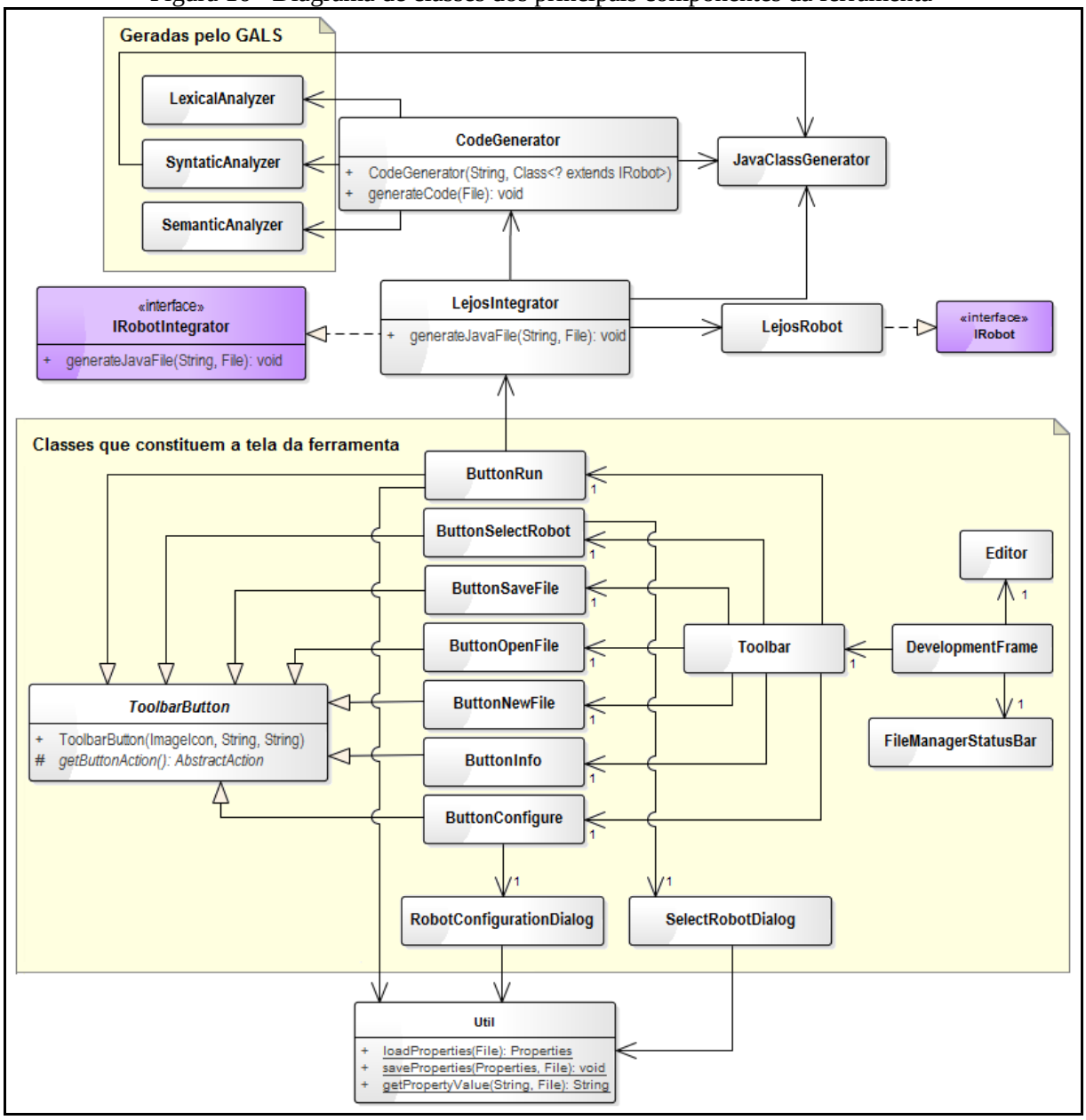
Programa correspondente na linguagem Java

```
 1  import lejos.nxt.SensorPort;
 2  import lejos.nxt.MotorPort;
 3  import br.inf.furb.tcc.robotoy.robot.lejos.LejosRobot;
 4
 5  public final class Program {
 6
 7         private static final LejosRobot _ROBOT = new LejosRobot();
 8
 9         private static double passos;
10
11         public static void main(String[] args) {
12                try {
13                       _ROBOT.setLeftWheelPort(MotorPort.B);
14                       _ROBOT.setRightWheelPort(MotorPort.A);
15                       _ROBOT.onStart();
16                       andarEnquantoNãoTemObstáculo();
17                       _ROBOT.onEnd();
18                } catch (Exception e) {
19                       _ROBOT.manageException(e);
20                }
21         }
22
23         private static void andarEnquantoNãoTemObstáculo() throws Exception {
24                passos = (double) 1;
25                while (!_ROBOT.hasObstacle()) {
26                       _ROBOT.walkForward(Math.abs(passos));
27                }
28         }
29
30  }
```

Na linha 7 é instanciado o robô que será utilizado, conforme *import* declarado na linha 3. Na linha 9 é apresentada a declaração de uma variável em forma de atributo. Essa estratégia garante que todas as variáveis declaradas no programa sejam globais. Além disso, é importante destacar que o `JavaClassGenerator` previne que o código definido pelo usuário conflite com o código da regra de negócio, pois as variáveis e os métodos de negócio utilizam *underline* em sua nomenclatura, o que na linguagem Robotoy não é permitido. Na linha 11 é declarado o método `main`, que contém as configurações do robô adicionadas pelo `LejosIntegrator`, o programa definido pelo usuário e o tratamento de exceções. Por fim, na linha 23 é apresentada uma rotina definida pelo usuário convertida em um método Java. É importante destacar que a linguagem Robotoy não suporta a passagem de parâmetros nas rotinas.

No diagrama de classes ilustrado na Figura 16 é apresentado o relacionamento entre os principais componentes da ferramenta.

Figura 16 - Diagrama de classes dos principais componentes da ferramenta

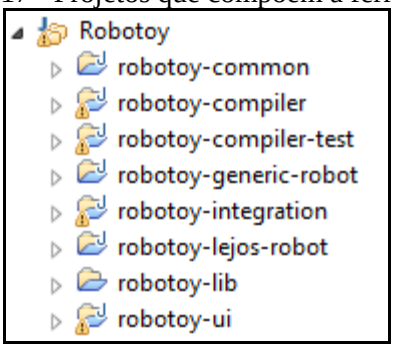


O diagrama apresenta a constituição visual da ferramenta, onde a classe que constrói a interface visual é a `DevelopmentFrame`. As classes `RobotConfigurationDialog`, `SelectRobotDialog` e `ButtonRun` se associam com a classe `Util`, que realiza a manipulação dos arquivos de propriedades. O `ButtonRun` é o componente responsável por iniciar a comunicação com os integradores, sendo que o único disponível é o `LejosIntegrator`. O `LejosIntegrator` instancia o gerador de código (`CodeGenerator`) que por sua vez instancia os analisadores léxico, sintático e semântico, a fim de identificar a existência de erros de compilação no programa definido pelo usuário. Os analisadores de código são classes geradas

pela ferramenta GALS, onde o `SemanticAnalyzer` é uma classe esqueleto gerada sem nenhuma regra de negócio, que contém apenas um método onde é possível definir as regras semânticas da linguagem. No caso da linguagem Robotoy, a geração de código é realizada após uma verificação semântica, sendo que isso acontece dentro da classe `SemanticAnalyzer` que depende de uma instância de `JavaClassGenerator` para providenciar a geração de código.

Para explicar a implantação do código gerado no robô é necessário apresentar os projetos que constituem a ferramenta, evidenciados na Figura 17.

Figura 17 - Projetos que compõem a ferramenta



Cada projeto possui uma finalidade:

a) `robotoy-ui`: provê a tela que interage com o usuário, que possibilita abrir e salvar arquivos, escrever o código, selecionar o robô, configurar sensores e motores, executar o programa, entre outras opções;

b) `robotoy-common`: centraliza as rotinas comumente utilizadas por vários projetos, como por exemplo, a de gerenciamento dos arquivos de propriedades. Para melhor explicar, é adotado o exemplo do arquivo de propriedades do robô leJOS: quando o usuário configura os sensores e motores um arquivo de propriedades é gerado, arquivo este que futuramente será consultado pelo `LejosIntegrator` para definir os sensores e motores na classe gerada. A responsabilidade de geração, armazenamento e acesso a um arquivo de propriedades é atribuída ao projeto `robotoy-common`;

c) `robotoy-generic-robot`: armazena apenas a definição do que é um robô programável na linguagem Robotoy, como a interface `IRobot`, as interfaces de comportamento que o `IRobot` estende e as classes especiais que viabilizam a definição de cor, conhecidas como `Color` e `ColorDefinition`. O `Color` é uma enumeração de cores que comporta as cores verde, azul, vermelho, amarelo, branco e preto. Por outro lado, o `ColorDefinition` é um objeto que encapsula

`Color` e uma `String` que denota a forma como o usuário declarou a cor. Assim, se o usuário declarar a cor através da sentença `cor tom <- vErDe`, `ColorDefinition` garante a exibição correta do valor atribuído à variável. Isto é, caso seja requisitada a exibição do valor da variável no *display*, será apresentado `vErDe` ao invés de `verde`;

d) `robotoy-lejos-robot`: armazena apenas a implementação do robô LEGO Mindstorms, que utiliza a API leJOS para definir os comportamentos decretados pela interface `IRobot`;

e) `robotoy-integration`: armazena a definição genérica de um integrador (`IRobotIntegrator`) e suas implementações, sendo que a única opção existente atualmente é a `LejosIntegrator`;

f) `robotoy-compiler`: armazena todos os artefatos responsáveis pela geração de código, como os analisadores léxico, sintático e semântico, o `JavaClassGenerator` e demais artefatos responsáveis pela compilação do programa;

g) `robotoy-compiler-test`: armazena todos os testes JUnit relacionados ao processo de compilação e geração de código;

h) `robotoy-lib`: armazena todas as bibliotecas de terceiros utilizadas pela ferramenta, como a `commons-io-2.4.jar`, `commons-lang-2.6.jar`, entre outras.

O processo de implantação de código no robô varia dependendo da plataforma escolhida pelo usuário. No caso do LEGO Mindstorms é utilizado o processo de implantação provido pela API leJOS, que dispõe de comandos *batch* para isso. No Quadro 21 é apresentada a sequência de comandos *batch* utilizados para realização da implantação de código na plataforma LEGO Mindstorms.

Quadro 21 - Comandos *batch* para realização da implantação de código no Mindstorms

```
nxjc.bat -cp "lib\robotoy-generic-robot.jar;lib\robotoy-lejos-robot.jar"
gen\Program.java

nxj.bat –cp "lib\robotoy-generic-robot.jar;lib\robotoy-lejos-
robot.jar;gen" -r Program
```

No primeiro comando é realizada a compilação da classe Java que foi gerada para, na sequência, ser estabelecido o *link* e *upload* no robô LEGO Mindstorms. É possível observar que a implantação só pode ser concebida se definido o *classpath*, composto pelo `.jar` dos projetos `robotoy-generic-robot` e `robotoy-lejos-robot`, que possuem apenas o conteúdo essencial para execução do algoritmo no robô.

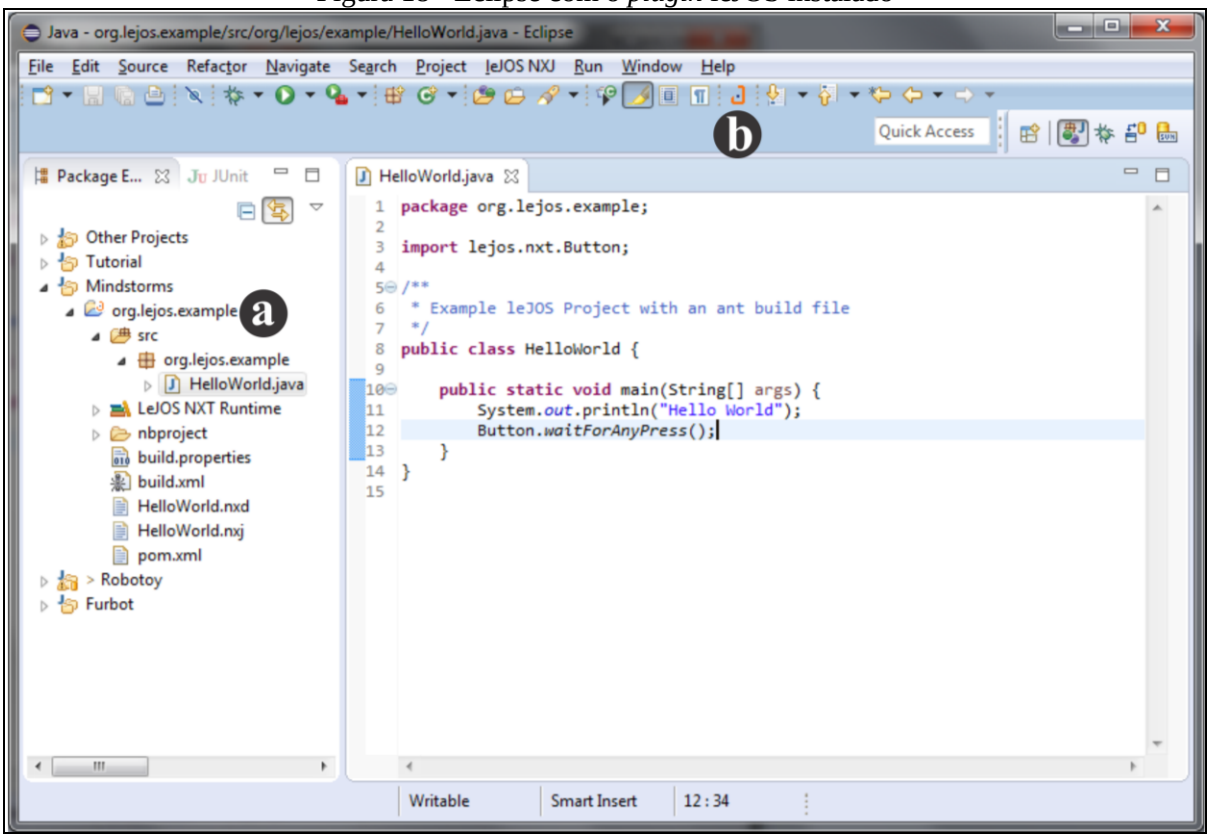
3.5 IMPLEMENTAÇÃO

Nesta seção são apresentadas as ferramentas utilizadas para o desenvolvimento da aplicação, seguida da operacionalidade da ferramenta e operacionalidade da integração.

3.5.1 Técnicas e ferramentas utilizadas

O desenvolvimento da ferramenta foi realizado no IDE Eclipse, contendo a biblioteca do JUnit para elaboração de testes automatizados e o *plugin* do leJOS instalado. O *plugin* do leJOS permite a criação de projetos com a natureza leJOS, que por sua vez podem ser executados no LEGO Mindstorms. Embora o *plugin* do leJOS não tenha sido utilizado de forma explícita na criação da ferramenta, é importante dizer que ele auxiliou na execução de testes manuais e na pesquisa sobre como funciona o processo de implantação de código no robô, provido pela biblioteca do leJOS. Na Figura 18 é apresentado o Eclipse com o *plugin* do leJOS instalado, onde: (a) projeto com natureza leJOS e (b) botão para *download* do *firmware* do LEGO Mindstorms NXT.

Figura 18 - Eclipse com o *plugin* leJOS instalado

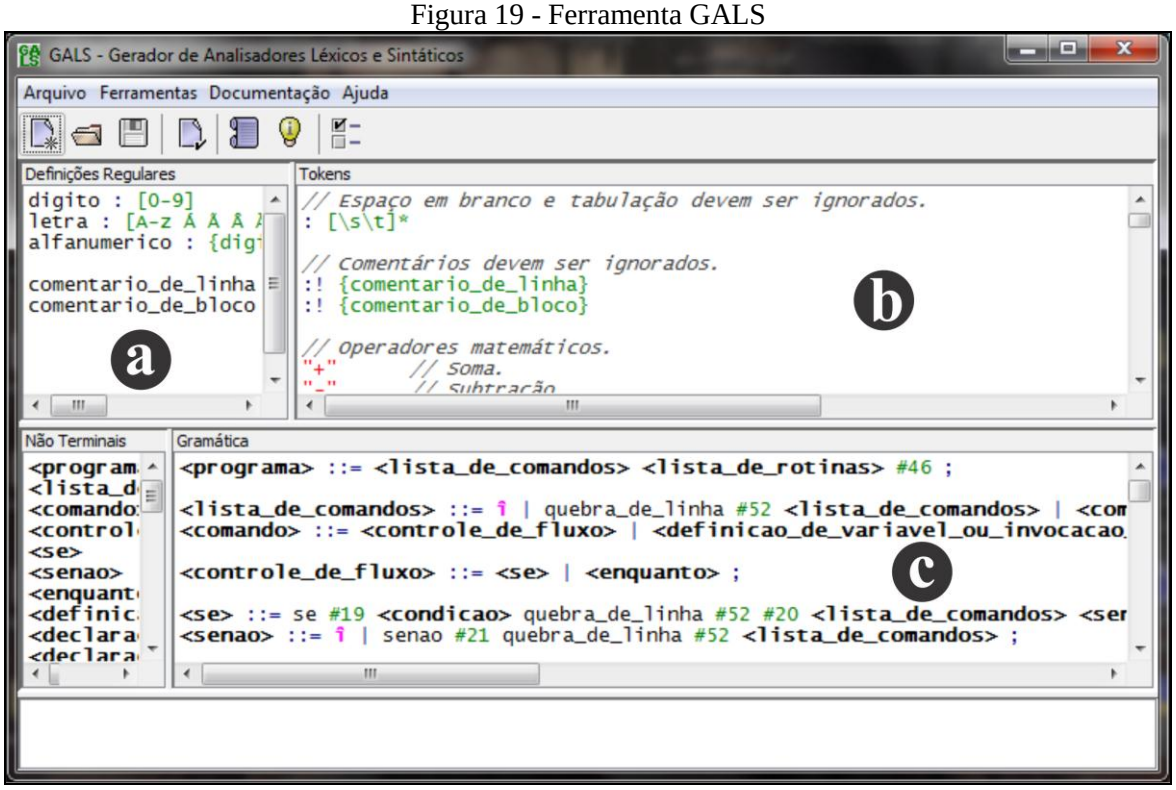


A interface do compilador foi construída com a biblioteca `Swing` do Java, onde o tema foi modificado através da biblioteca JGoodies que se assemelha ao tema do Windows 7.

Por sua vez, a linguagem foi especificada utilizando-se o GALS, uma ferramenta que gera os analisadores léxico e sintático e provê uma classe esqueleto que permite a construção

do analisador semântico. Na Figura 19 é apresentada a ferramenta GALS, onde: (a) especificação das definições regulares usadas para especificar os *tokens*; (b) definição dos *tokens* da linguagem e (c) especificação das regras sintáticas da linguagem.

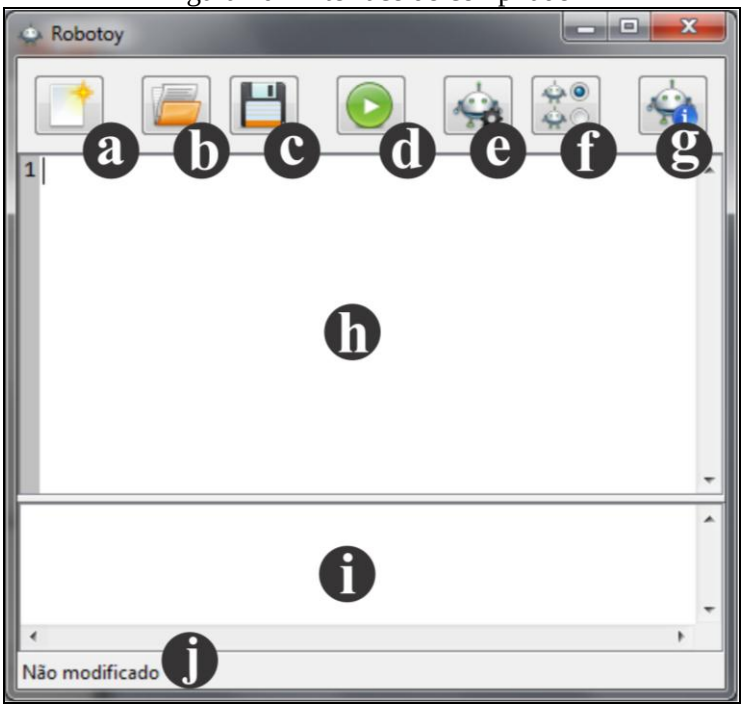
Figura 19 - Ferramenta GALS



3.5.2 Operacionalidade da ferramenta

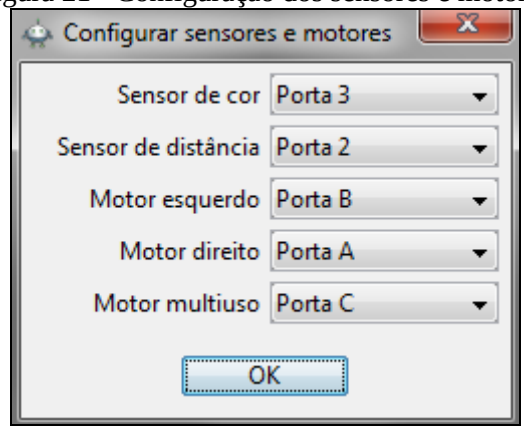
Na Figura 20 é apresentada a interface do ambiente de programação. O ambiente de programação possui: (a) botão responsável por limpar o editor de código; (b) botão para abrir um arquivo com a extensão `.toy`; (c) botão para salvar o código em edição num arquivo com a extensão `.toy`; (d) botão para executar o código; (e) botão para configurar a porta dos sensores e motores exclusivos do robô LEGO Mindstorms; (f) botão para selecionar o tipo do robô, sendo que atualmente a única opção disponível é o robô LEGO Mindstorms; (g) botão de ajuda que abre uma documentação sobre os comandos da linguagem; (h) editor de código com linhas enumeradas no intuito de facilitar o apontamento de erros encontrados durante a compilação; (i) *console*, onde são exibidos erros ao tentar compilar um código com erro de compilação e (j) *status* de modificação do programa, caso o *status* seja `Modificado` e o usuário tente fechar o ambiente de programação, será perguntado se o mesmo deseja descartar o código em edição. Todos os botões da tela possuem um *hint* com uma descrição seguida da tecla de atalho entre parênteses. O *hint* do botão (c), por exemplo, é `Salvar (Ctrl+S)`.

Figura 20 - Interface do compilador



Na Figura 21 é apresentada a tela para configuração de sensores e motores, específicos do robô LEGO Mindstorms (ação associada ao botão (e)). O sensor de cor e o sensor de distância podem ser conectados nas portas 1, 2, 3 ou 4, já os motores podem ser conectados nas portas A, B ou C. Caso o usuário informe a mesma porta para dois ou mais componentes, ao clicar no botão `OK`, será exibida uma mensagem de alerta informando que a configuração é inválida.

Figura 21 - Configuração dos sensores e motores



Para executar um programa, basta o usuário escrever um algoritmo no editor de código e clicar no botão `Executar` (d). O código será implantado no robô se: (a) não houver erro de compilação; (b) o computador estiver conectado ao robô, sendo que para o LEGO Mindstorms o computador deve estar conectado ao *brick*, seja pelo cabo USB ou por *bluetooth*. Após a implantação ser realizada com sucesso, o usuário poderá posicionar o robô no cenário e clicar no botão de *input*, iniciando a execução do algoritmo no robô.

3.5.3 Operacionalidade da integração

Nesta seção é explicado o procedimento para adicionar uma nova plataforma suportada pela linguagem. Primeiramente, o usuário deve modificar a tela de escolha do robô definida pela classe `SelectRobotDialog`, incluindo um *radio button* na tela que denota a plataforma a ser adicionada, conforme apresentado na Figura 22.

Figura 22 - Inclusão de outra plataforma na janela `SelectRobotDialog`

A geração do arquivo `robot.properties` e seu conteúdo é realizada pela classe `SelectRobotDialog`, no momento em que o usuário seleciona o tipo de robô. Quando selecionado o robô leJOS, o conteúdo do arquivo `robot.properties` condiz com o exposto no Quadro 22. Na primeira linha é informado o tipo do robô selecionado, posteriormente é informada a classe do integrador para o robô em questão, por fim é informado o caminho do arquivo de propriedades direcionado ao robô.

Quadro 22 - Conteúdo do arquivo `robot.properties` quando selecionado o robô LEGO Mindstorms

```
robot.type=LejosRobot
robot.integrator=br.inf.furb.tcc.robotoy.integrator.lejos.LejosIntegrator
robot.configuration.file=<diretório descompactado>\gen\lejos-robot.properties
```

Em relação ao arquivo de propriedades direcionado ao robô, é necessário implementar uma tela própria para gerenciar o arquivo de propriedades. No caso do robô LEGO Mindstorms foi adicionado na barra de ferramentas um botão para configuração dos sensores e motores. O mesmo pode ser feito para outro modelo. Para adicionar um botão na barra de ferramentas é necessário criar um novo botão que estenda de `ToolbarButton`, sendo que a ação do botão se resume à implementação do método abstrato `getButtonAction()`, onde a implementação é livre para atender as necessidades da nova plataforma. Após ter definido o botão, é possível adicioná-lo na barra de ferramentas pela classe `Toolbar`, cuja definição é apresentada no Quadro 23. É possível basear-se na implementação da tela de configuração de sensores e motores do robô LEGO Mindstorms, definida pela classe `ButtonConfigure`, cuja a ação é abrir a janela `RobotConfigurationDialog`.

Quadro 23 - Definição da barra de ferramentas

```
private Toolbar() {
    setLayout(new FlowLayout(FlowLayout.LEFT, 10, 10));

    ICompilerFacade facade = new CompilerFacade();
    add(new ButtonNewFile());
    add(new JSeparator(SwingConstants.VERTICAL));
    add(new ButtonOpenFile(facade));
    add(new ButtonSaveFile(facade));
    add(new JSeparator(SwingConstants.VERTICAL));
    add(new ButtonRun());
    add(new JSeparator(SwingConstants.VERTICAL));
    add(new ButtonConfigure());
    add(new ButtonSelectRobot());
    add(new JSeparator(SwingConstants.VERTICAL));
    add(new ButtonInfo());
}
```

Na sequência, o implantador de código, definido dentro do `ButtonRun`, consulta o arquivo `robot.properties` e recupera o integrador e o arquivo de propriedades direcionado ao robô em questão. Após obter esses dados, o implantador instancia o integrador via reflexão, passando por parâmetro o arquivo de propriedades a ser consultado pelo integrador, para que o mesmo possa inserir informações adicionais na classe Java a ser gerada.

O integrador deve ser criado dentro do projeto `robotoy-integration`, isolado num pacote próprio, como no caso do `LejosIntegrator` inserido dentro do pacote `br.inf.furb.tcc.robotoy.integrator.lejos`. Um integrador deve estender da classe `IRobotIntegrator`, que define o que é um integrador no contexto da linguagem. Além disso, o integrador deve conhecer o tipo de robô ao qual está diretamente relacionado, conforme evidenciado na instanciação do gerador de código exposto no Quadro 24. O gerador de código depende do programa escrito pelo usuário e de uma classe que implementa `IRobot`, representados respectivamente pelos parâmetros `program` e `LejosRobot.class`.

Quadro 24 - Instanciação do gerador de código no `LejosIntegrator`

```
CodeGenerator generator = new CodeGenerator(program, LejosRobot.class);
```

A implementação de `IRobot` para uma nova plataforma deve ser feita num projeto separado. O `LejosRobot` por exemplo, está inserido dentro do projeto `robotoy-lejos-robot`, que armazena apenas a implementação do robô leJOS. O mesmo deve ser feito para uma nova plataforma, para impedir que as dependências de um robô impactem na implantação de código de outra plataforma, o que aconteceria se os mesmos compartilhassem o mesmo projeto. Depois de criado o novo projeto é necessário referenciá-lo no *classpath* dos projetos `robotoy-compiler` e `robotoy-integration`, que devem conhecer o novo projeto para gerar a classe Java a ser implantada no robô.

Por fim, no `ButtonRun`, através da propriedade `robot.type`, é possível identificar a plataforma onde a implantação de código será realizada. A partir daí basta definir como essa implantação será feita.

3.6 RESULTADOS E DISCUSSÃO

Para validar a linguagem especificada, foi necessário construir uma superfície para a realização de testes. Essa superfície foi construída com um pedaço de lona acrílica e fita isolante, onde a fita isolante foi usada para traçar as linhas e colunas de um tabuleiro. Na Figura 23 é possível visualizar a superfície construída, divida em cinco linhas e cinco colunas, onde cada célula mede 25cm².

Figura 23- Tabuleiro construído para a execução de testes

A medida de um passo do robô foi definida levando em consideração a superfície elaborada. Desta forma, um passo do robô equivale à distância entre uma célula e outra. Atualmente, considerando as configurações que a ferramenta disponibiliza, não tem como customizar esse valor, o que de certa forma restringe a elaboração de cenários para a linguagem.

Inicialmente, para testar o funcionamento do robô foi adotado o problema do robô perdido na mina, o mesmo problema aplicado aos trabalhos correlatos. Na Figura 24 é apresentado o cenário onde o robô foi inserido, organizado com caixas de papelão que representam a mina.

Figura 24 - Réplica do exercício do robô perdido na mina

Para resolver o problema foi elaborado um programa relativamente simples, conforme exposto no Quadro 25.

Quadro 25 - Solução Robotoy para o problema do robô perdido na mina

```
enquanto tem obstáculo
       virar para a esquerda 1
fim do enquanto
andar para frente 2
```

Nesta solução o robô verifica a existência de obstáculos com auxílio do sensor ultrassônico. Caso encontre um obstáculo, o robô vira para a esquerda, para mais uma vez verificar a existência de obstáculos. No cenário elaborado o robô está de costas para a saída. Nestas condições, o robô vira duas vezes para a esquerda. Na terceira verificação, o laço do `enquanto` é encerrado, o que permite ao robô andar duas células para frente e atingir o objetivo do exercício proposto.

Para efetuar uma melhor avaliação sobre o desempenho do robô no cenário, foi necessário afastar um pouco as paredes para poder observar a movimentação do robô, pois o mesmo eventualmente colidia nas paredes no momento de virar. Após esse ajuste, foi possível constatar que a linguagem atende o que propõe, pois o robô conseguiu realizar o percurso corretamente.

No Quadro 26 é possível observar o código Java que o compilador gerou para a solução adotada.

Quadro 26 - Código gerado para a solução proposta

```
import lejos.nxt.SensorPort;
import lejos.nxt.MotorPort;
import br.inf.furb.tcc.robotoy.robot.lejos.LejosRobot;
public final class Program {

	private static final LejosRobot _ROBOT = new LejosRobot();

	public static void main(String[] args) {
		try {
			_ROBOT.setColorSensorPort(SensorPort.S3);
			_ROBOT.setUltrasonicSensorPort(SensorPort.S4);
			_ROBOT.setMultiuseMotorPort(MotorPort.B);
			_ROBOT.setLeftWheelPort(MotorPort.A);
			_ROBOT.setRightWheelPort(MotorPort.C);
			_ROBOT.onStart();
			while (_ROBOT.hasObstacle()) {
				_ROBOT.turnLeft(Math.abs(1));
			}
			_ROBOT.walkForward(Math.abs(2));
			_ROBOT.onEnd();
		} catch (Exception e) {
			_ROBOT.manageException(e);
		}
	}

}
```

Para validar a característica da ferramenta de permitir modelos diferenciados de robôs, foi elaborado um exercício com o robô identificador de cor. Neste exercício o robô deve ficar imprimindo no *display* a cor identificada, sendo que quando o robô identificar a cor branca, deve finalizar o programa, emitindo um som. O algoritmo utilizado para resolver o problema é apresentado e equiparado ao código correspondente em Java no Quadro 27.

Quadro 27 - Solução para o exercício do robô identificador de cor

Programa na linguagem Robotoy

```
1 cor tom <- cor identificada
2 enquanto tom =/ branco
3       texto tomIdentificado <- "Tom: " . tom
4       escrever tomIdentificado
5       tom <- cor identificada
6 fim do enquanto
7 emitir som
```

Programa correspondente na linguagem Java

```
import lejos.nxt.SensorPort;
import lejos.nxt.MotorPort;
import br.inf.furb.tcc.robotoy.robot.generic.ColorDefinition;
import br.inf.furb.tcc.robotoy.robot.generic.Color;
import br.inf.furb.tcc.robotoy.robot.lejos.LejosRobot;

public final class Program {

	private static final LejosRobot _ROBOT = new LejosRobot();

	private static ColorDefinition tom;
	private static String tomIdentificado;

	public static void main(String[] args) {
		try {
			_ROBOT.setColorSensorPort(SensorPort.S1);
```

```
                    _ROBOT.onStart();
                    tom = _ROBOT.detectColor();
                    while (tom.getColor() != Color.WHITE) {
                           tomIdentificado = "Tom: " + tom.getName();
                           _ROBOT.write(tomIdentificado);
                           tom = _ROBOT.detectColor();
                    }
                    _ROBOT.beep();
                    _ROBOT.onEnd();
             } catch (Exception e) {
                    _ROBOT.manageException(e);
             }
      }

}
```

Na linha 1 o robô identifica a cor. Na linha 2 é feita uma repetição verificando se a cor identificada anteriormente é diferente de branco. Se for, os comandos dentro do `enquanto` são executados, o que desencadeia a impressão da cor no *display*, para em seguida atualizar o valor da variável `tom` com uma nova cor capturada. Quando é identificada a cor branca, a repetição é encerrada, o que leva o robô a emitir um som e finalizar o programa.

Esse algoritmo foi executado várias vezes no robô, pois foi realizado num ambiente com pouca iluminação, o que eventualmente ocasionava a leitura de uma cor que não condizia com a realidade. Por outro lado, quando o robô é submetido a um ambiente com bastante iluminação dificilmente ele erra a leitura. Outro teste realizado com o robô foi submetê-lo à leitura da cor rosa. Como o rosa não é suportado pela linguagem, a rotina que identifica a cor mais próxima é disparada, o que resultou no retorno da cor vermelha, a que justamente era esperada. Diante dessas observações e das observações realizadas no cenário do robô perdido na mina, é possível perceber que a disposição do ambiente pode influenciar bastante nos resultados dos exercícios. Por isso, é necessário adequar bem o ambiente antes de trabalhar com crianças, para que o foco das atividades se direcione ao ensino da programação e não a problemas de ambiente.

Existem outros cenários que foram elaborados para validar o desempenho da ferramenta como um todo. Esses cenários acompanhados das respectivas soluções encontram-se no Apêndice B.

Durante o desenvolvimento do trabalho foram encontradas algumas dificuldades. Uma delas diz respeito às limitações que o Java embarcado do leJOS oferece. Inicialmente, a linguagem fornecia outro comando orientado a tempo para controle de fluxo, cujo comando era definido pela sintaxe exposta no Quadro 28.

Quadro 28 - Comando para controle de fluxo removido da linguagem

```
executar por <tempo> <unidade de tempo>
      <lista de comandos>
fim do executar
```

Descrição:

Este comando executa uma lista de comandos enquanto o tempo limite não é atingido, onde:

```
<tempo> ::= <valor numérico> | <variável numérica>
<unidade de tempo> ::= s | min
<lista de comandos> ::= ε | <cláusula para controle de fluxo> |
                          <chamada de rotina> | <comando do robô> |
                          <declaração de variável> | <atribuição de variável>
```

Exemplo:

```
executar por 10 min
    virar para a esquerda 1
fim do executar
```

Por não ter previsto que o leJOS trabalhava com um Java embarcado, foi utilizada a biblioteca `joda-time-2.4.jar` na geração de código, biblioteca que facilita a programação de rotinas que dependem de tempo. No entanto, ao realizar a implantação de código de um programa que utilizava esse comando, foram apresentados inúmeros problemas, o que ocasionou a descoberta dessa limitação. Para tentar manter o comando, foi necessário estudar o Java embarcado do leJOS, no intuito de encontrar alguma alternativa que pudesse substituir a biblioteca utilizada. Durante a pesquisa, foi constatado que nem o `Date` do Java embarcado poderia ser usado para resolver o problema, sendo que o que mais se adequou nesse caso foi uma classe própria do leJOS, denominada `Timer` e que realiza exatamente a ação do comando em questão. No entanto, para utilizar essa classe seria necessário adaptar mais uma parte da camada intermediária, pois essa solução se adequa apenas ao robô LEGO Mindstorms, o que contraria aquilo que a camada intermediária defende. Como demandaria muito tempo para adaptar a camada intermediária, decidiu-se remover esse comando por questão de indisponibilidade de tempo.

Outra dificuldade encontrada que vale a pena ser mencionada diz respeito à mobilidade do robô no cenário. Em muitos casos, quando o robô realizava o percurso, o ambiente ou a própria estrutura física do robô comprometia a sua movimentação. Eventualmente, era necessário alinhar o robô em relação ao cenário quando este começava a pender para um lado ou quando a roda traseira do robô travava na fita isolante. É inviável afirmar que a execução do robô dentro do cenário acontece, na maioria dos casos, sem intervenção externa, porque isso é algo que depende da montagem do robô e do meio onde ele foi inserido.

Em relação aos trabalhos correlatos, o Quadro 29 traz uma comparação entre os mesmos e a ferramenta desenvolvida, evidenciando as características de cada trabalho.

Quadro 29 - Quadro comparativo entre os trabalhos correlatos e a ferramenta desenvolvida

características	FURBOT	RoboMind FURB	NXT-G	Robotoy
permite a execução em um robô real		X	X	X
fornece um meio para inclusão de outras plataformas				X
permite ao usuário montar o robô de sua preferência			X	X
possui a linguagem inteiramente em português		X		X
tipo de de programação	textual	textual	visual	textual
possui um ambiente para simulação virtual	X	X		
possui um ambiente de programação próprio, que dispensa o uso de um IDE como o Eclipse por exemplo		X	X	X

4 CONCLUSÕES

Atualmente, existem inúmeras metodologias de ensino capazes de desenvolver as habilidades cognitivas de uma criança, onde é possível abranger o ambiente da robótica educacional, que proporciona ao indivíduo uma aprendizagem lúdica e multidisciplinar, movida pela curiosidade que o ambiente desperta. Diante dessas características, decidiu-se elaborar uma linguagem em torno desse ambiente, que pudesse contribuir com o desenvolvimento de uma criança através da programação e automação de robôs.

Diante dos experimentos realizados, é possível dizer que o trabalho proposto cumpre os objetivos estabelecidos, pois fornece uma linguagem de programação, possui o robô LEGO Mindstorms implementado com a API leJOS e realiza a implantação de código nessa plataforma. Além disso, a ferramenta traz consigo duas características adicionais, uma delas viabiliza a integração com outros robôs, enquanto que a outra permite que o usuário possa montar o robô de sua preferência levando em consideração algumas restrições. A primeira característica aumenta as possibilidades de programação na ferramenta, pois apesar do *kit* LEGO Mindstorms ser mais aderente ao meio educacional, os preços não contribuem muito para a sua adoção, o que torna interessante a possibilidade de integração com outras plataformas, como o Arduino, por exemplo. Por sua vez, a segunda característica estimula a criatividade da criança, uma vez que não restringe a etapa de montagem proporcionada pelo *kit* à replicação de um modelo pré-determinado.

Inicialmente, a ideia era avaliar o desempenho da linguagem com crianças. No entanto, por indisponibilidade de tempo, não foi possível cumprir essa etapa. Embora os testes nesse sentido não tenham sido realizados, é possível dizer que a linguagem poderia ser utilizada por crianças pelo fato de assemelhar-se ao RoboMind. Considerando que as soluções para resolver o exercício do robô perdido na mina se assemelharam nas linguagens RoboMind e Robotoy, é possível dizer que os exercícios do RoboMind destinados a crianças do ensino fundamental (BENITTI et al., 2010b) poderiam ser replicados na linguagem Robotoy. Talvez o que impacte na receptividade das crianças em relação à ferramenta seja a interface visual, comparado ao RoboMind, o Robotoy possui uma interface consideravelmente simples. A existência de um simulador e de um editor de código mais elaborado são características que chamam a atenção. Portanto, seria necessário desenvolver melhor a interface da ferramenta antes de experimentar efetivamente com crianças.

4.1 EXTENSÕES

Como sugestão de extensões para a ferramenta propõe-se:

a) elaborar a partir da linguagem da ferramenta uma linguagem gráfica, semelhante ao NXT-G, onde o usuário arrasta peças para elaborar programas, permitindo selecionar entre programação textual e programação visual;
b) adicionar suporte ao Arduino como plataforma de programação;
c) estender o vocabulário da linguagem, adicionando novos comandos como por exemplo `executar por`, explicado na seção 3.5;
d) fazer um estudo detalhado do NXT-G e estender o Robotoy com algumas de suas características, já que nesta ferramenta não há restrições quanto ao posicionamento das rodas do robô, o que torna a programação nesta ferramenta mais genérica que a programação na linguagem Robotoy;
e) adicionar propriedades de configuração extras, pois a linguagem disponibiliza um valor inalterável para a quantidade de giros do motor, fazendo com que os comandos de movimentação se tornem limitados;
f) desenvolver um simulador semelhante ao disponibilizado pelo RoboMind, onde seja possível a criação de cenários customizados;
g) melhorar o IDE com recursos de edição, como *auto complete*, *syntax highlight* e as opções para refazer (Ctrl+Y) e desfazer (Ctrl+Z).

REFERÊNCIAS

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores**: princípios, técnicas e ferramentas. Tradução Daniel de Ariosto Pinto. Rio de Janeiro: LTC, 1995. Disponível em: <http://www.cin.ufpe.br/~mgmp/arquivos.html/Compiladores+-+Principios+Tecnicas+E+Ferramentas+%2528Pt+Br%2529.pdf>. Acesso em 31 nov. 2014.

ANDRADE, Fátima J. S. **Robótica educacional**: uma metodologia educacional no estudo de funções de 7º ano. 2011. 98 f. Relatório de Estágio de Mestrado (Mestrado em Ensino de Matemática no 3º Ciclo e Secundário) – Universidade da Madeira, Madeira. Disponível em: <http://digituma.uma.pt/bitstream/10400.13/375/1/MestradoF%C3%A1timaAndrade.pdf>. Acesso em: 16 out. 2014.

BENITTI , Fabiane B. V. et al. Robótica como elemento motivacional para atração de novos alunos para cursos de computação. In: CONGRESO IBEROAMERICANO DE EDUCACIÓN SUPERIOR EN COMPUTACIÓN, 18., 2010a, San Lorenzo, Paraguay. **Proceedings...** San Lorenzo: [s.n.], 2010. Não paginado. Disponível em: <http://www.inf.furb.br/dsc/download/ciesc2010_submission_16.pdf>. Acesso em: 26 mar. 2014.

_____. **Ensino fundamental**. Blumenau, [2010b?]. Disponível em: <http://robolab.inf.furb.br/index.php?option=com_content&view=category&id=3&Itemid=14>. Acesso em: 19 nov. 2014.

BORGES NETO, Hermínio; BORGES, Suzana M. C. **O papel da informática educativa no desenvolvimento do raciocínio lógico**. [Ceará], [2007]. Disponível em: <http://www.multimeios.ufc.br/arquivos/pc/pre-print/O_papel_da_Informatica.pdf>. Acesso em: 30 out. 2014.

CASTILHO, Maria I. **Robótica na educação:** com que objetivos? [Porto Alegre], [2009?]. Disponível em: <http://www.pucrs.br/eventos/desafio/2007/mariaines.php#raclog>. Acesso em: 16 out. 2014.

CHELLA, Marco T. **Ambiente de robótica para aplicações educacionais com SuperLogo**. 2002. 186 f. Dissertação (Mestrado em Engenharia Elétrica e da Computação) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas. Disponível em: <http://www.bibliotecadigital.unicamp.br/document/?down=vtls000283839>. Acesso em: 16 out. 2014.

CREATIVE COMMONS. **Uma apresentação**. [S.l.], 2009. Disponível em: <http://projetologo.webs.com/texto1.html>. Acesso em: 30 out. 2014.

FERREIRA, Alan S. **A contribuição da robótica para o desenvolvimento das competências cognitivas superiores no contexto de projetos de trabalho**. [Rio de Janeiro], [2005]. Disponível em: <http://www.educacaopublica.rj.gov.br/biblioteca/tecnologia/0017.html>. Acesso em: 16 out. 2014.

GIOPPO, Lucas L. et al. **Robô seguidor de linha**. Curitiba, 2009. Disponível em: <http://paginapessoal.utfpr.edu.br/msergio/portuguese/ensino-de-fisica/oficina-de-integracao-ii/oficina-de-integracao-ii/Monog-09-2-Seguidor-de-linha.pdf>. Acesso em: 23 out. 2014.

GLASSEY, Roger; GRIFFITHS, Lawrie. **Compiling and running programs**. [S.l.], [2012?]. Disponível em: <http://www.lejos.org/nxt/nxj/tutorial/Preliminaries/CompileAndRun.htm#1>. Acesso em: 23 out. 2014.

GOMES, Evandro L. B.; TAVARES, Luís A. **Uma solução com Arduino para controlar e monitorar processos industriais**. [Curitiba], [2013]. Disponível em: <http://www.inatel.br/pos/index.php/downloads/doc_download/26-luis-tavares-arduino>. Acesso em: 23 out. 2014.

GRIFFITHS, Lawrie et al. **NXJ technology**. [S.l.], 2009. Disponível em: <http://www.lejos.org/nxj.php>. Acesso em: 29 mar. 2014.

HALMAF, Arvid. **RoboMind:** como criar arquivos .map. Tradução Fernando Luís Merízio. Revisão Diego Leonardo Urban. Blumenau, [2010?]. Disponível em: <http://robolab.inf.furb.br/index.php?option=com_content&view=article&id=42&Itemid=48>. Acesso em: 02 nov. 2014.

KAHN, Ken. ToonTalk: An animated programming environment for children. **The Journal of Visual Languages and Computing**, [S.l.], v. 7, n. 2, 1996. Disponível em: <http://www.toontalk.com/Papers/jvlc96.pdf>. Acesso em: 30 out. 2014.

KELLY, James F. **LEGO Mindstorms NXT-G**: programming guide. 2nd ed. New York: Apress, 2010. Disponível em: <http://www.kirp.chtf.stuba.sk/moodle/pluginfile.php/46109/mod_resource/content/1/Lego%20Mindstorms%20NXT-G%20programming%20guide.pdf >. Acesso em: 13 nov. 2014.

LEGO GROUP. **History of Lego robotics**. [S.l.], [2013a?]. Disponível em: <http://www.lego.com/en-us/mindstorms/gettingstarted/historypage/>. Acesso em: 25 mar. 2014.

_____. **Lego Mindstorms NXT 2.0**. [S.l.], 2013b. Disponível em: <http://shop.lego.com/en-US/LEGO-MINDSTORMS-NXT-2-0-8547>. Acesso em: 21 abr. 2014.

_____. **Our company**. [S.l.], 2014. Disponível em: <http://education.lego.com/en-us/about-us/lego-education-worldwide/our-company>. Acesso em: 29 mar. 2014.

LIFELONG KINDERGARTEN GROUP. **Scratch**. [S.l.], [2014?]. Disponível em: <http://wiki.scratch.mit.edu/wiki/Scratch>. Acesso em: 30 out. 2014.

MARCON, Fernando et al. Potencializando a aprendizagem da lógica com uso de ambiente de programação de alto nível. In: SEMINÁRIO NACIONAL DE INCLUSÃO DIGITAL, 1., 2012, Passo Fundo. **Anais...** Passo Fundo: IMED, 2012. Não paginado. Disponível em: <http://senid.upf.br/2012/anais/96177.pdf>. Acesso em: 25 mar. 2014.

MORGADO, Leonel; CRUZ, Maria G. Tópicos sobre programação de computadores como método educativo no jardim-de-infância. In: CONGRESSO INTERNACIONAL DE EDUCAÇÃO "O MUNDO DA CRIANÇA", 3., 2004, Vila Real, Portugal. **Anais...** [S.l.]: UTAD, 2004. Disponível em: <http://home.utad.pt/~leonelm/papers/omundodacrianca/omundodacrianca.pdf>. Acesso em: 30 out. 2014.

OLIVEIRA, Thiago H. B.; SILVA, Cherlia; ANDRADE, Mariel J. P. Scratch: utilizando programação para desenvolvimento do raciocínio lógico em crianças. In: JORNADA DE ENSINO, PESQUISA E EXTENSÃO, 13., 2013, Recife. **Anais...** Recife: UFRPE, 2013. Não paginado. Disponível em: <http://www.eventosufrpe.com.br/2013/cd/resumos/R1503-2.pdf>. Acesso em: 26 mar. 2014.

RICARTE, Ivan L. M. **Programação de sistemas:** uma introdução. Campinas, 2004. Disponível em: <http://www.dca.fee.unicamp.br/~eleri/ea876/04/cap9.pdf>. Acesso em: 13 nov. 2014.

ROBOMIND ACADEMY. **Introduction**: what is Robo/RoboMind? [S.l.], 2014. Disponível em: <http://www.robomind.net/en/introduction.htm>. Acesso em: 01 abr. 2014.

RUSTON, Jeremy. **What is it?** [S.l.], 2014. Disponível em: <http://haiku-vm.sourceforge.net/>. Acesso em: 27 out. 2014.

SCHOLZ, Matthias P. **Lesson:** What is leJOS. [S.l.], [2006]. Disponível em: <http://www.lejos.org/rcx/tutorial/getstarted/whatislejos.html>. Acesso em: 23 out. 2014.

SETZER, Valdemar W. **O computador no ensino**: nova vida ou destruição? [São Paulo], 2007. Disponível em: <http://www.ime.usp.br/~vwsetzer/computador-no-ensino.html>. Acesso em: 30 out. 2014.

SILVA, Marcelo E. **Uma proposta de ensino de física para turmas noturnas**. 2014. 145 f. Dissertação (Mestrado Profissional em Ensino de Física) - Programa de Pós-Graduação em Ensino de Física, Instituto de Física, Universidade Federal do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.if.ufrj.br/~pef/producao_academica/dissertacoes/2014_Marcelo_Elias/dissertacao_Marcelo_Elias.pdf>. Acesso em: 23 out. 2014.

SOUZA, Anderson R. et al. A placa Arduino: uma opção de baixo custo para experiências de física assistidas pelo PC. **Revista Brasileira de Ensino de Física**, [São Paulo], v. 33, n. 1, 2011. Disponível em: <http://www.sbfisica.org.br/rbef/pdf/331702.pdf>. Acesso em: 23 out. 2014.

VAHLDIK, Adilson et al. **Projeto FURBOT**: introdução. [Blumenau], 2008a. Disponível em: <http://www.inf.furb.br/poo/furbot/introducao.html>. Acesso em: 30 mar. 2014.

_____. **Projeto FURBOT**: primeiros passos. [Blumenau], 2008b. Disponível em: <http://www.inf.furb.br/poo/furbot/introducao.html>. Acesso em: 30 mar. 2014.

APÊNDICE A – Especificação sintática da linguagem

No **Erro! Fonte de referência não encontrada.** é apresentada a especificação sintática completa da linguagem.

Quadro 30 - Especificação sintática da linguagem

```
<programa>          ::= <lista_de_comandos> <lista_de_rotinas>

<lista_de_comandos> ::= ε | quebra_de_linha <lista_de_comandos> |
                        <comando>  quebra_de_linha <lista_de_comandos>
<comando>           ::= <controle_de_fluxo> |
                        <definicao_de_variavel_ou_invocacao_de_metodo> |
                        <acao>

<controle_de_fluxo> ::= <se> | <enquanto>

<se>                ::= se <condicao> quebra_de_linha
                           <lista_de_comandos>
                           <senao>
                        fim do se
<senao>             ::= ε | senao quebra_de_linha <lista_de_comandos>

<enquanto>          ::= enquanto <condicao> quebra_de_linha
                           <lista_de_comandos>
                        fim do enquanto

<definicao_de_variavel_ou_invocacao_de_metodo> ::=
                        <tipo> identificador <declaracao_obrigatoria> |
                        identificador <declaracao_nao_obrigatoria>
<declaracao_obrigatoria> ::= <- <expressao>
<declaracao_nao_obrigatoria> ::= ε | <- <expressao>
<tipo>              ::= numero | texto | cor

<expressao>         ::= <aritmetica>
<aritmetica>        ::= <operacao> <soma_subtracao>
<soma_subtracao>    ::= ε | + <operacao> <soma_subtracao> |
                          - <operacao> <soma_subtracao>
<operacao>          ::= <termo> <multiplicacao_divisao_resto>
<multiplicacao_divisao_resto> ::= ε | * <termo> <multiplicacao_divisao_resto> |
                        / <termo> <multiplicacao_divisao_resto> |
                        % <termo> <multiplicacao_divisao_resto>
<termo>             ::= + <termo> |
                        - <termo> |
                        identificador <concatenar_literal> |
                        numerico |
                        ( <aritmetica> ) |
                        literal <concatenar_literal> |
                        <cor> |
                        cor identificada
<cor>               ::= branca | branco | preta | preto | vermelha |
                        vermelho | amarela | amarelo | verde | azul
<concatenar_literal> ::= ε | . <opcoes_de_concatenacao>
<opcoes_de_concatenacao> ::= identificador <concatenar_literal> |
                        ( <aritmetica> ) <concatenar_literal> |
                        literal <concatenar_literal>

<acao>              ::=  <andar> |
                         virar <virar> |
                         <girar_roda> |
                         <parar> |
                         <escrever> |
                         <emitir_som>

<andar>             ::= andar para <frente_ou_tras> <quantidade_de_vezes>
```

```
<virar>              ::= para a <esquerda_ou_direita> <quantidade_de_vezes> |
                         motor multiuso para a <esquerda_ou_direita>
                         <quantidade_de_vezes_obrigatoria>

<girar_roda>         ::= girar roda <esquerda_ou_direita> para <frente_ou_tras>

<parar>              ::= parar de <acoes_que_param>

<escrever>           ::= escrever <expressao>

<emitir_som>         ::= emitir som

<frente_ou_tras>     ::= frente | tras
<esquerda_ou_direita>::= esquerda | direita
<quantidade_de_vezes>::= ε | <expressao>
<quantidade_de_vezes_obrigatoria> ::= <expressao>
<acoes_que_param>    ::= andar | virar | girar
<condicao>           ::= <tipo_condicao> <condicao_extra>
<tipo_condicao>      ::= <verificar_obstaculo> | <comparar_numerico_literal>
<condicao_extra>     ::= ε | e <condicao> | ou <condicao>
<verificar_obstaculo>::= <nao> tem obstáculo
<nao>                ::= ε | não
<comparar_numerico_literal> ::=
                         <expressao> <operador_relacional_numerico> <expressao>
<operador_relacional_numerico> ::= > | < | >= | <= | = | =/

<lista_de_rotinas>   ::= ε | <rotina> <lista_de_rotinas>
<rotina>             ::= rotina identificador quebra_de_linha
                           <lista_de_comandos>
                         fim da rotina quebra_de_linha
```

APÊNDICE B – Lista de exercícios

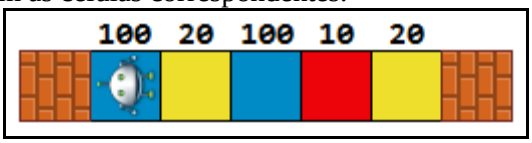
Neste apêndice são apresentados alguns exercícios elaborados para a linguagem Robotoy, expostos nos quadros a seguir.

Quadro 31 – Exercício 01: robô contador de cédulas

Pré-condições:
As pré-condições para esse exercício são:
a) possuir um sensor de cor conectado, apontado para o chão;
b) possuir um sensor de distância conectado, voltado para frente;
c) possuir dois motores conectados, que constituem as rodas do robô.

Enunciado:
O robô deve contabilizar uma quantidade de reais, que são representados por células coloridas pelas quais o robô percorre. Conforme o robô vai realizando o percurso, ele deve identificar a cor da célula, fazendo a soma de valores. Ao final do percurso, quando o robô identificar uma parede, ele deve imprimir no *display* a soma das cédulas. As cores disponíveis são:
a) vermelho, equivalente à 10 reais;
b) amarelo, equivalente à 20 reais;
c) azul, equivalente à 100 reais.

Na figura abaixo é ilustrado um exemplo de cenário do exercício, onde os valores que devem ser somados estão alinhados com as células correspondentes.



Solução:

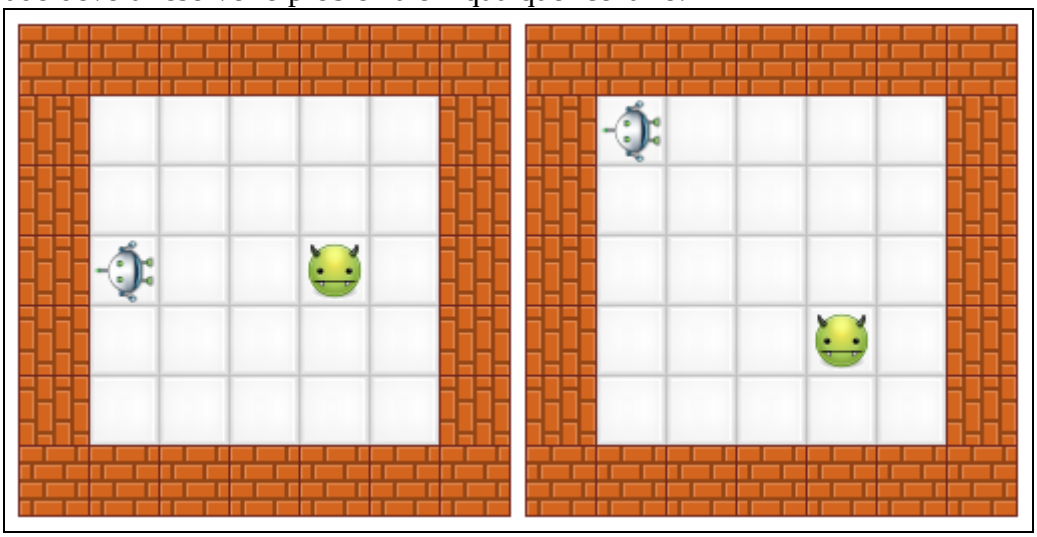
```
número valor <- 0

enquanto não tem obstáculo
    somarValor
    andar para frente 1
fim do enquanto
somarValor

texto resultado <- "Valor = " . valor
escrever resultado

rotina somarValor
    cor corCédula <- cor identificada

    se corCédula = vermelha
          valor <- valor + 10
    fim do se

    se corCédula = amarela
          valor <- valor + 20
    fim do se

    se corCédula = azul
          valor <- valor + 100
    fim do se
fim da rotina
```

Quadro 32 - Exercício 02: travessia com obstáculos

Pré-condições:
As pré-condições para esse exercício são:
a) possuir um sensor de distância conectado, voltado para frente;
b) possuir dois motores conectados, que constituem as rodas do robô.

Enunciado:
O robô começa na coluna 2 e deve atravessar a sala para chegar na coluna 6, finalizando sempre na linha onde começou. É possível que o robô encontre algum obstáculo durante a travessia. Neste caso, deverá ser feito um desvio a fim de contornar o obstáculo.

Abaixo são apresentados dois cenários para o problema em questão, sendo que o algoritmo elaborado deverá resolver o problema em qualquer cenário.



Solução:

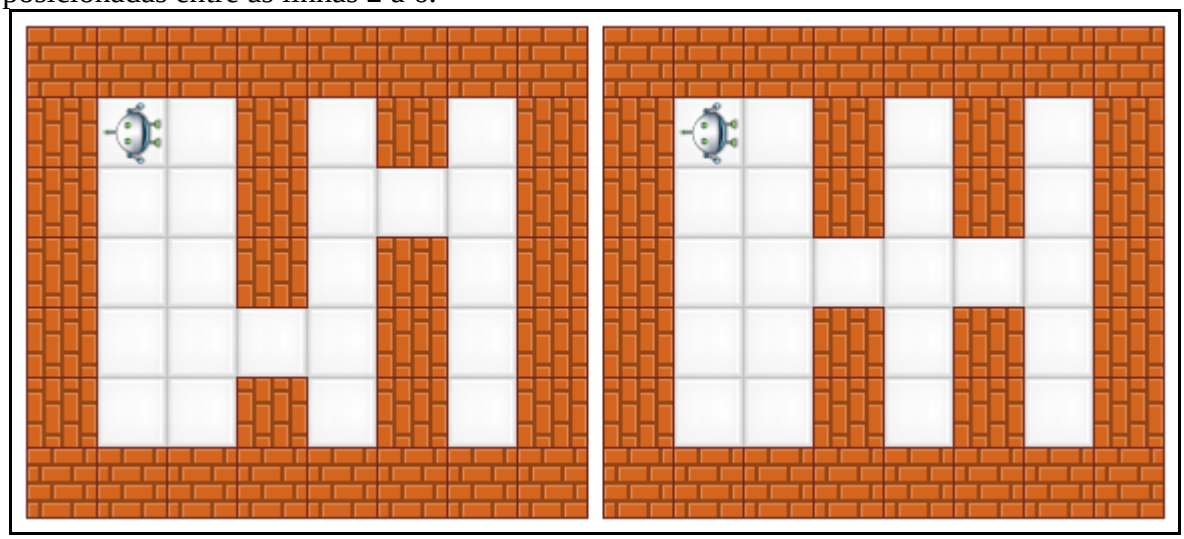
```
número células Percorridas <- 0

enquanto não tem obstáculo
    andar para frente 1
    células Percorridas <- células Percorridas + 1
fim do enquanto

se células Percorridas < 4
    virar para a direita
    andar para frente
    virar para a esquerda
    andar para frente 2
    virar para a esquerda
    andar para frente
    virar para a direita
fim do se
```

Quadro 33 - Exercício 03: atravessar duas paredes

Pré-condições:
As pré-condições para esse exercício são:
a) possuir o motor multiuso conectado com um sensor de distância acoplado;
b) possuir dois motores conectados, que constituem as rodas do robô.

Enunciado:
O robô deve procurar as passagens e atravessar o cenário, chegando na coluna 7.

Abaixo são apresentados dois cenários para esse problema, sendo que o algoritmo elaborado deverá resolver o problema em qualquer cenário. Observa-se que as passagens podem estar posicionadas entre as linhas 2 a 6.



Solução:

```
número célulasPercorridas <- 0

andarEnquantoNãoTemObstáculo

enquanto células Percorridas < 5
    procurarAtravessarPassagem
fim do enquanto

rotina procurarAtravessarPassagem
    se tem obstáculo
           posicionarParaBusca
           procurarPassagem
    fim do se
    atravessarPassagem
fim da rotina

rotina posicionarParaBusca
    virar para a esquerda 1
    enquanto não tem obstáculo
           andar para frente 1
    fim do enquanto
    virar para a direita 2
fim da rotina

rotina procurarPassagem
    virar motor multiuso para a esquerda 1
    enquanto tem obstáculo
           andar para frente 1
    fim do enquanto
    virar para a esquerda 1
    virar motor multiuso para a direita 1
```

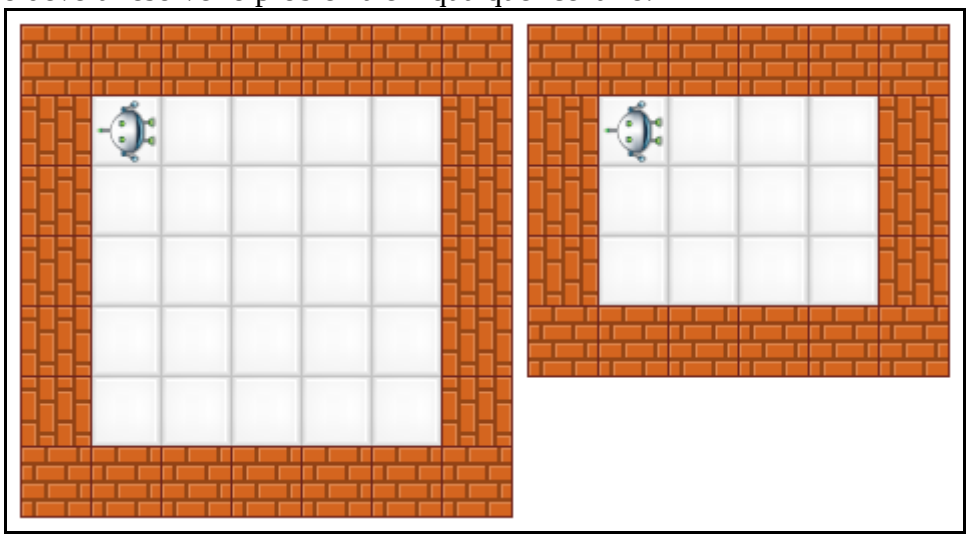
```
fim da rotina

rotina atravessarPassagem
    andar para frente 2
    célulasPercorridas <- célulasPercorridas + 2
fim da rotina

rotina andarEnquantoNãoTemObstáculo
    enquanto não tem obstáculo
           andar para frente 1
           célulasPercorridas <- célulasPercorridas + 1
    fim do enquanto
fim da rotina
```

Quadro 34 - Exercício 04: descobrir as dimensões do cenário

Pré-condições:
As pré-condições para esse exercício são:
a) possuir um sensor de distância conectado, voltado para frente;
b) possuir dois motores conectados, que constituem as rodas do robô.

Enunciado:
O robô começa na linha 2 e na coluna 2 e deve imprimir no *display* as dimensões do tabuleiro onde foi inserido, ou seja, a quantidade de linhas e colunas. Quando o valor for impresso no *display*, o robô deverá emitir um som.

Abaixo são apresentados dois cenários para o problema em questão, sendo que o algoritmo elaborado deverá resolver o problema em qualquer cenário.



Solução:

```
número linhas <- 1
número colunas <- 1

enquanto não tem obstáculo
    andar para frente 1
    colunas <- colunas + 1
fim do enquanto

virar para a direita 1

enquanto não tem obstáculo
    andar para frente 1
    linhas <- linhas + 1
fim do enquanto

texto qtdLinhas <- "Linhas: " . linhas
texto qtdColunas <- "Colunas: " . colunas
escrever qtdLinhas
escrever qtdColunas
emitir som
```