Robison Cris Brito
Handrey Emanuel Galon

INTRODUÇÃO AOS AMBIENTES DE PROGRAMAÇÃO NXT-G E leJOS PARA O LEGO MINDSTORMS

Editora UTFPR

INTRODUÇÃO AOS AMBIENTES DE PROGRAMAÇÃO NXT-G E leJOS PARA O LEGO MINDSTORMS

Robison Cris Brito
Handrey Emanuel Galon

INTRODUÇÃO AOS AMBIENTES DE PROGRAMAÇÃO NXT-G E leJOS PARA O LEGO MINDSTORMS

Curitiba
UTFPR Editora
2016

Disponível também em: <http://repositorio.utfpr.edu.br/jspui/>.

Dados Internacionais de Catalogação na Publicação

B862 Brito, Robison Cris

Introdução aos ambientes de programação NXT-G e leJOS para o Lego Mindstorms. / Robison Cris Brito, Handrey Emanuel Galon. – Curitiba: Ed. UTFPR, 2016.

204 p. : il. color.

ISBN: 978-85-7014-169-9

1. Robótica. 2. LEGO (Brinquedo) – Inovações tecnológicas. 3. Dispositivos de lógica programável. 4. Programação lógica (Computação). 5. Inovações educacionais. 6. Tecnologia educacional. 7. Java (Linguagem de programação de computador). I. Galon, Handrey Emanuel. II. Título.

CDD (23. ed.) 629.892

Bibliotecária: Maria Emília Pecktor de Oliveira CRB-9/1510

Coordenação editorial
Camila Lopes Ferreira
Emanuelle Torino

Projeto gráfico, capa e editoração eletrônica
Vanessa Constance Ambrosio

Normalização
Camila Lopes Ferreira

Revisão gramatical e ortográfica
Adão Araújo

UTFPR Editora
Av. Sete de Setembro, 3165
80.230-901 - Curitiba – PR
www.utfpr.edu.br

DEDICATÓRIA

Dedicamos esta obra a um grande grupo de pessoas, em especial, nossos pais, irmãos, amigos, que sempre estiveram ao nosso lado, apoiando e incentivando o nosso nobre ofício de aprender para ensinar.

Dedicamos também aos amantes de novas tecnologias, em especial aos que desejam iniciar no mundo da robótica, mundo fascinante e que certamente os prenderá por muitos anos.

AGRADECIMENTOS

Queremos agradecer a um conjunto de pessoas que permitiram a transformação deste sonho em realidade.

Um obrigado muito especial à Universidade Tecnológica Federal do Paraná (UTFPR), na qual eu (Robison) estou envolvido diretamente desde 1996, quando entrei no curso Técnico em Eletrônica. Foram muitos trabalhos e desafios para me tornar professor desta instituição. E eu (Handrey), por me preparar para o mercado de trabalho no curso de Engenharia da Computação, e dar oportunidades, seja nos concursos em que participei como acadêmico, seja nos congressos que participei como autor de artigos.

Um agradecimento ao egresso do curso de tecnologia em Análise e Desenvolvimento de Sistemas da UTFPR, Iuri Menin, por acreditar no desenvolvimento de aplicativos para a plataforma LEGO Mindstorms, comprando com recursos próprios o primeiro robô e fazendo uma contribuição importantíssima para esta obra.

Agradecemos também a um grupo de professores e alunos da universidade. À professora Beatriz Borsoi e ao professor Fábio Favarim, por não medirem esforços na compra de novos kits de LEGO Mindstorms para a universidade, assim como à professora Beatriz por promover vários grupos de alunos para estudar e multiplicar o conhecimento referente à plataforma, envolvendo alunos do ensino básico, de ensino médio e de graduação.

SUMÁRIO

1 INTRODUÇÃO À ROBÓTICA

O termo robô tem origem em uma peça teatral do autor tcheco Karel Carpek, e data do início da década de 1920, de nome *Os robôs universais de Rossum* em inglês *Rossum's universal robots* (Figura 1). Robô (em tcheco, *robota*) significa **trabalhador forçado**, e esta ficção de Karel Carpek se refere aos robôs do cientista Rossum, criados para, obedientemente, servir à humanidade. O drama acontece quando estas **criaturas** passam a não mais gostar do papel de subserviência e se rebelam contra seus criadores. O autor da peça faz uso da imaginação para satirizar a forma de progresso técnico implantados na Europa pelos norte-americanos (PONTES, 2010).

Figura 1 – Representação de Rossum's universal robots

Fonte: Richardson (2013).

As primeiras referências de que se tem notícia sobre robótica datam de 350 a.C., do matemático grego Arquitas de Tarento. Amigo de Platão, Tarento teria criado um pássaro mecânico a que chamou **O Pombo** e que teria a capacidade de voar usando jato de ar comprimido. Contudo, foi Leonardo da Vinci quem elaborou o primeiro projeto de um robô humanoide, por volta do ano de 1495. Suas anotações, descobertas em 1950, continham desenhos detalha-

dos de um cavaleiro que aparentemente podia movimentar pernas e braços. O trabalho foi baseado em sua pesquisa da anatomia humana conhecida como homem vitruviano (PONTES, 2010), como se vê na Figura 2.

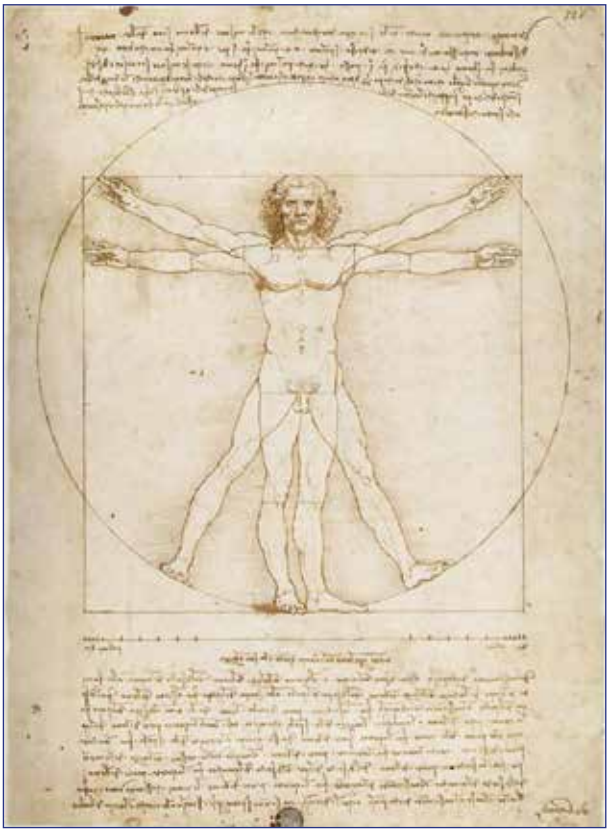


Figura 2 – Homem vitruviano, de Leonardo da Vinci

Fonte: World Mysteries (2011).

No século XVII, trabalhadores japoneses criaram um robô autônomo chamado **Karakuri** (Figura 3), que era capaz de servir chá (PONTES, 2010).

Outro exemplo de criatura mecânica é o pato de Jacques de Vaucanson (século XVIII) (Figura 4). Esse artefato ficou conhecido pela articulação realista de partes de seu corpo, por comer, digerir e defecar automaticamente. Vaucanson construiu ainda três outras criaturas humanoides: um tocador de mandolim que batia o pé, um pianista que simulava a respiração e movia a cabeça e um flautista (PONTES, 2010).

Figura 3 – Robô Karakuri, utilizado para servir chá

Fonte: Sentosa (2011).

Figura 4 – Pato de Jacques de Vaucanson

Fonte: Outer Places (2014).

Não obstante sua complexidade mecânica e a precisão com que desempenhavam suas tarefas, esses robôs, obviamente, não apresentavam capacidades cognitivas. O modelo de robôs que se conhece hoje só começou a ser desenvolvido com a chegada da computação e da inteligência artificial (PONTES, 2010).

Os primeiros passos nessa direção aconteceram em 1950, quando Alan Turing propõe, no artigo *Computing machinery and intelligence* (TURING, 1950), uma definição operacional de pensamento. Seu experimento, *imitation game*, sugere que, em vez de perguntar se uma máquina pode pensar, deve-se verificar se ela é capaz de passar em um teste de inteligência. Nesse teste, uma máquina é considerada inteligente se sua resposta for igual a resposta de um humano (PONTES, 2010).

O desafio para construir máquinas capazes de simular o comportamento cognitivo humano foi estudado por John McCarthy e Marvin Minsky, ainda na mesma década. No final dos anos 50, esses cientistas fundaram o *Artificial Intelligence Laboratory* do *Massachusetts Institute of Technology* (MIT), o primeiro laboratório dedicado à construção de robôs e ao estudo da inteligência humana (PONTES, 2010).

O objetivo principal de todos estes estudos envolvendo robôs inteligentes é permitir que estes desenvolvam tarefas feitas até então exclusivamente por humanos, já que existe uma série de vantagens ao utilizar robôs em tarefas específicas, como o aumento da precisão em tarefas manufaturadas, robustez, rapidez, uniformidade e suporte a ambientes hostis e ainda o incremento dos índices de qualidade e minimização de peças rejeitadas.

Pode-se ainda citar vantagens de fatores econômicos como a utilização eficiente de unidades de produção intensiva, o aumento de produtividade e a redução do tempo de preparação da fabricação. Também se tem benefícios no fator sociológico, que são a redução do número de acidentes, o afastamento do ser humano de locais perigosos para a saúde, redução de horários de trabalho, e ainda o aumento do poder de compra (diminui o custo dos produtos por eles construídos).

Ao contrário dos primeiros robôs, que eram mecânicos e muito limitados, os robôs atuais podem utilizar várias tecnologias presentes no dia a dia das pessoas, uma vez que estas tecnologias estão cada vez mais acessíveis do ponto de vista econômico, e mais fáceis de utilizar, do ponto de vista técnico.

Um exemplo é o uso da visão computacional nos robôs, sendo que estes podem capturar imagens do ambiente a partir de uma câmera, processar, e tomar decisões com base nestas imagens.

Outro recurso muito importante é o uso do *global positioning system* (GPS) ou sistema global de posicionamento, que permite ao robô identificar sua posição no globo terrestre e possibilitar sua navegação de forma autônoma. O GPS tradicional utiliza uma rede de satélite artificial, o que pode torná-lo impreciso, em especial, se o robô estiver em um ambiente coberto ou imerso, como por exemplo no oceano. Nesta situação, alternativas como o uso das antenas das operadoras de celulares ou ainda redes de boias submarinas podem ser adotadas.

Tecnologias como as apresentadas anteriormente permitem o desenvolvimento das chamadas *simultaneous localization and mapping* (SLAM) (que significa localização e mapeamento simultâneo). Esta técnica permite que robôs consigam mapear ambientes enquanto o utilizam.

Do ponto de vista do desenvolvimento de aplicativos para robótica, várias ferramentas para esta finalidade podem ser citadas, como a *Robot Operating System* (ROS, 2016), *Microsoft Robotics Developer Studio* (MICROSOFT, 2016), *Carnegie Mellon Robot Navigation Toolkit* (CARMEN, 2016) e *PlayerStage* (PLAYER, 2014). Todos são ambientes e/ou ferramentas que auxiliam no desenvolvimento de aplicativos para robôs ou redes de robôs, que utilizam, na maioria, a linguagem de programação C para o desenvolvimento.

Embora muito completos, esses ambientes não são indicados para iniciantes, ainda mais com pouco ou nenhum conhecimento em programação. Neste cenário, ambientes de programação *Drag and Drop* (ou seja, arrastar e soltar)) se destacam, sendo utilizados componentes com funções específicas prontas, minimizando (e muito) a tarefa de programar.

Com a chegada das últimas gerações de computadores, onde recebe destaque LEGO Mindstorms (Figura 5), já se percebe a evolução no seu ambiente de desenvolvimento nativo, chamado de NXT-G, que é toda baseada no arrastar e soltar, fazendo com que algumas dificuldades antes encontradas como a complexidade para desenvolvimento de software sejam

facilitadas, sendo este um ambiente comumente utilizado para o ensino de robótica e linguagem de programação para crianças, jovens e adultos.

Assim, crianças a partir de 10 anos (como divulgado nas embalagens de LEGO Mindstorms), estudantes, acadêmicos e pesquisadores podem usufruir das vantagens desta plataforma, aprendendo conceitos complexos, como robótica, projeto de máquinas e veículos, assim como a programação.

Figura 5 – LEGO Mindstorms NXT 2.0

Fonte: Adami (2016).

LEGO MINDSTORMS

LEGO Mindstorms é uma linha do brinquedo LEGO lançada comercialmente em 1998, voltada para a educação tecnológica. Este ambiente é resultado de uma parceria entre o Media Laboratory do MIT e o LEGO Group. O LEGO Mindstorms é constituído por um conjunto de peças da linha tradicional e da linha LEGO *Technic* (motores, eixos, engrenagens, polias e correntes), acrescido de sensores de toque, de intensidade luminosa, de temperatura, distância, entre outros. Todos os sensores são controlados pelo módulo *Robotic Command Explorer* (RCX) (Figura 6). Este módulo contém um microcontrolador Hitachi H8/300 de 8-bits da Renesas, que inclui 32 kbyte de *Random Access Memory* (RAM) para armazenar seu *firmware* e programas de utilizador.

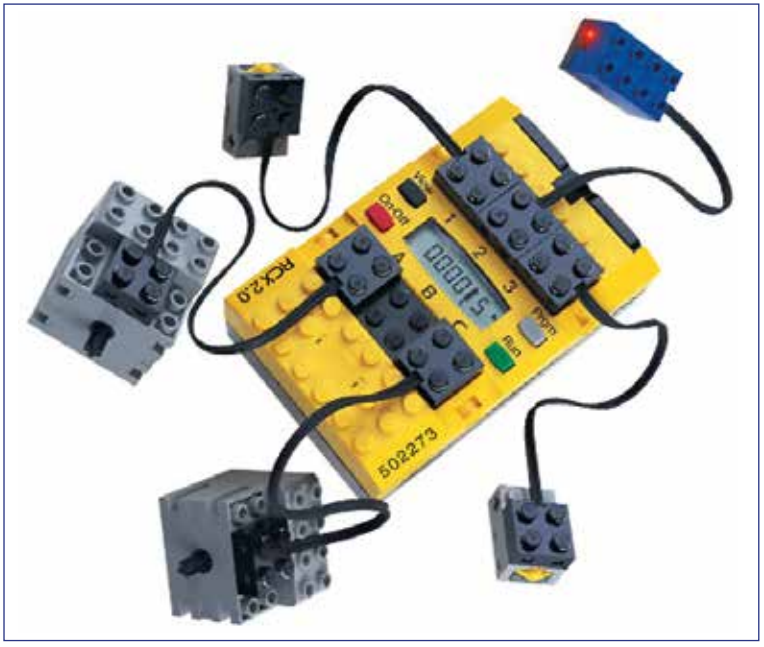


Figura 6 – Bloco programável RCX

Fonte: Cyber Toy (2011).

RCX foi o primeiro módulo programável dos produtos da linha LEGO Mindstorms, da LEGO, contendo um conjunto de setecentas peças especiais que permite a construção de robôs com diversas funções, graças a motores e a sensores de toque e de luz.

A base do conjunto é o módulo RCX, que executa as funções de comando do robô – na realidade um pequeno computador encapsulado num bloco LEGO.

Em 2006 o modelo RCX foi substituído, com o lançamento do NXT 2.0, este último representado pela Figura 7, com suas características (MINDSTORMS, 2016c).

Figura 7 – NXT 2.0

Fonte: Mindstorms (2016a).

Características LEGO Mindstorms NXT

- Processador Atmel 32-bit ARM;
- Três portas de saída digital;
- Quatro portas de entrada;
- *Display* tipo matriz;
- Alto-falante;
- Alimentado por pilhas de lítio, padrão AA;
- *Bluetooth*;
- Porta de comunicação universal serial bus (USB) 2.0;
- Conjunto de três servo motores interativos (com encoder acoplado);
- Sensores de ultrassom, som, luz, cor, contato, entre outros;
- Programa de computador intuitivo com uma versão LEGO do LabVIEW;
- Compatível com PCs tradicionais com sistema operacional Windows e com MACs.

Este conjunto é utilizado com função didática em instituições de ensino tecnológico, abordando a teoria e a prática de conteúdos direcionados para a introdução à robótica, permitindo o desenvolvimento de projetos de pequeno e médio porte, estimulando a criatividade e a solução de problemas do cotidiano por parte dos alunos.

Nota 1. Dica

A Olimpíada Brasileira de Robótica (OBR) é apoiada pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). Com a temática da robótica – tradicionalmente de grande aceitação entre os jovens, visa estimulá-los às carreiras científico-tecnológicas, identificar novos talentos e promover debates e atualizações no processo de ensino-aprendizagem brasileiro. Destinada a todos os alunos de qualquer escola pública ou privada do ensino fundamental, médio ou técnico em todo o território nacional, a OBR é uma iniciativa pública, gratuita e sem fins lucrativos. Nesse contexto destaca-se a ampla utilização do kit LEGO Mindstorms NXT 2.0. Mais informações sobre a OBR, consultar: <http://www.obr.org.br/>.

O sistema LEGO Mindstorms para escolas consiste em três partes:

a) conjuntos de construção;

b) software educativo RoboLAB;

c) esquemas (diagramas) de trabalho.

Cada parte é comercializada separadamente, permitindo a cada escola/educador selecionar uma solução apropriada para as suas necessidades.

Para o desenvolvimento, é possível utilizar o ambiente nativo que acompanha o LEGO Mindstorms NXT 2.0, chamado de linguagem NXT-G, ou ainda programar utilizando outras linguagens como o Java (leJOS), Python (NXT-Python), C (NXC), ADA (GNAT GPL), *Forth* (pbFORTH), Lua (pbLua) e *Visual basic* (usando recursos COM+), porém, para isso é necessário substituir o *firmware* original existente nos blocos programáveis do LEGO Mindstorms, pelo *firmware* da plataforma desejada.

2.1 KITS DE DESENVOLVIMENTO LEGO MINDSTORMS

Os principais kits de robô LEGO Mindstorms disponíveis no mercado são:

a) LEGO Robô 8547 Mindstorms NXT 2.0;

b) LEGO Robô Mindstorms NXT 2.0 *Education Set Base* 9797;

c) LEGO 31313 LEGO Mindstorms EV3.

A seguir são explicadas as diferenças entre estes kits.

2.1.1 LEGO Robô 8547 Mindstorms NXT 2.0

Este kit é comercializado em vários países, em especial nos Estados Unidos, sendo classificado como um brinquedo (no site da Amazon, por exemplo, ele é encontrado na categoria *Toys*). No Brasil, algumas lojas online também vendem este kit, porém, como se trata de produto importado, o preço fica bem acima do praticado nos Estados Unidos.

O tradicional humanoide pode ser montado com o kit (MINDSTORMS, 2016a):

LEGO robô 8547 Mindstorms NXT 2.0

O conteúdo da embalagem do LEGO robô 8547 Mindstorms NXT 2.0 (MINDSTORMS, 2016a) apresenta os seguintes detalhes técnicos:

Conteúdo da embalagem do LEGO robô 8547 Mindstorms NXT 2.0

- 612 peças (elementos de construção LEGO *Technic* e engrenagens, rodas, trilhos, pneus, entre outros);
- 1 processador NXT *Brick*;
- 2 sensores de toque;
- 1 sensor ultrassônico;
- 1 sensor de cor;
- 3 servo motores com sensores internos de rotação;
- 7 cabos conectores para ligar os motores e sensores para o LEGO NXT;
- 1 cabo USB;
- guia do usuário, com as instruções para a construção de alguns robôs e uma introdução ao hardware e software do equipamento;
- *compact disc* (CD) com o software NXT-G.

O ponto negativo deste modelo é ausência de bateria, assim, para o funcionamento do processador NXT *Brick,* são necessárias 6 pilhas AA de 1,5V – as pilhas recarregáveis que se encontram no mercado costumam ter 1,2V, assim, é comum que a bateria recarregável necessite ser trocada a cada 2 ou 3 horas de uso (muitas vezes, muito antes disso). Desta forma, o ideal é a alimentação com pilhas alcalinas.

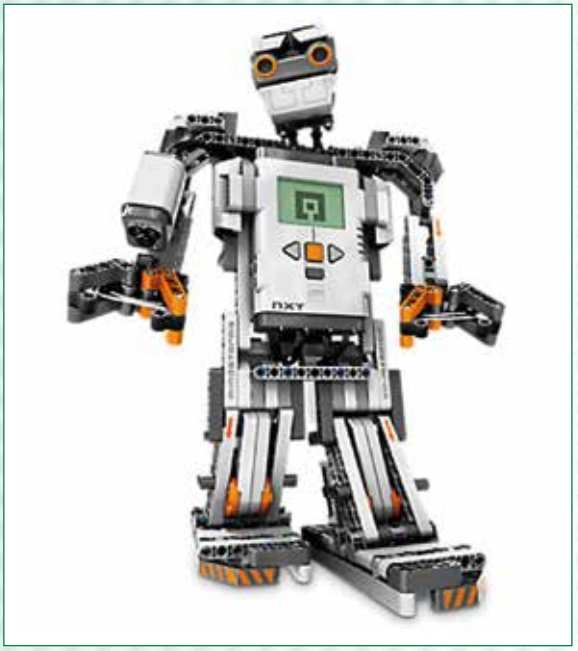
O LEGO Mindstorms NXT 8547 (MINDSTORMS, 2016a) vem com instruções para construir quatro modelos de robôs, que variam em complexidade e funcionamento:

Principais modelos que podem ser montados

Shooterbot: é um robô veículo que pode vigiar um ambiente fechado e atirar bolas.

Color sorter: é um robô que pode classificar diferentes objetos coloridos.

Alpha rex: é um robô com múltiplas funções, pode andar, evitar obstáculos, pegar e distinguir cores entre diferentes objetos coloridos.



Robogator: é um robô animal, que se move como um jacaré.

2.1.2 LEGO Robô Mindstorms NXT 2.0 *Education Set Base* 9797

Este kit é vendido pela LEGO, em especial, para escolas, faculdades e universidades, sendo que o objetivo é um kit simples (possui menos peças que o kit apresentado anteriormente), porém, apresenta a vantagem de possuir mais sensores e já vir com uma bateria recarregável.

A imagem de divulgação deste kit é apresentada na Figura 8.

Figura 8 – LEGO robô Mindstorms NXT 2.0 *Education Set Base* 9797

Fonte: Laurens (2012).

O kit da embalagem do LEGO robô Mindstorms NXT 2.0 *Education Set Base* 9797 (LAURENS, 2012) é formado por:

Conteúdo da embalagem do LEGO robô Mindstorms NXT 2.0 Education Set Base 9797

- 431 peças (elementos de construção LEGO *Technic* e engrenagens, rodas, trilhos, pneus, entre outros);
- 1 processador NXT *Brick*;
- 1 bateria de lítio recarregável e um carregador DC;
- 2 sensores de toque;
- 1 sensor ultrassônico;
- 1 sensor de luz;
- 1 sensor de som;
- 3 servo motores com sensores internos de rotação;
- 7 cabos conectores para ligar os motores e sensores para o LEGO NXT;
- 1 cabo USB.

Este modelo do LEGO Mindstorms NXT não permite realizar as montagens dos robôs apresentados para o LEGO Robô 8547 Mindstorms NXT 2.0. O menor número de peças contidas neste kit só possibilita desenvolver robôs mais simples, como o apresentado na Figura 8, cuja montagem é demonstrada passo a passo em manual que acompanha o kit.

Outra diferença deste kit educacional em relação ao kit 8547 é a ausência do CD com o ambiente de desenvolvimento visual LEGO Mindstorms NXT-G, sendo que nesta situação, para usar o software, este deve ser baixado do site da LEGO.

2.1.3 LEGO 31313 LEGO Mindstorms EV3

Em 2013, a LEGO lançou a terceira versão do LEGO Mindstorms, chamada 31313 LEGO Mindstorms EV3. Essa nova versão aumenta a capacidade de memória e processamento do robô, possibilitando configurações mais avançadas. O EV3 se comunica também com dispositivos móveis Android e iOS.

O processador do EV3 é o poderoso ARM9, 10 vezes mais rápido do que o modelo anterior, que permite que o robô seja capas de fazer mais operações com os sensores e motores ao mesmo tempo. O novo sistema conta também com memória RAM de 64 MB e memória de armazenamento interno de 16 MB. Possui porta USB para WiFi, leitor de cartão micro SD, botões retro iluminados e quatro portas de motor.

Os motores deste kit são mais velozes do que os do modelo anterior. O kit vem acompanhado de três motores: os dois maiores servem especificamente para mover o robô e o menor executa funções mais específicas, como, por exemplo, disparar munição.

O bloco de construção principal, com o processador do brinquedo, vem equipado com *Bluetooth* para conectar o robô a dispositivos móveis. Há aplicativos para iOS e Android para controlar e programar o brinquedo remotamente. Além disso, é possível comunicar o equipamento através de uma rede WiFi.

A proposta da empresa é entregar um brinquedo mais acessível e ao mesmo tempo mais sofisticado para atender as expectativas da geração infantil atual. A LEGO acredita que a linha Mindstorms permite que jovens e crianças exercitem suas habilidades e coloquem a sua criatividade para trabalhar.

O conteúdo da embalagem do 31313 LEGO Mindstorms EV3 (MINDSTORMS, 2016a) engloba:

Conteúdo da embalagem do 31313 LEGO Mindstorms EV3

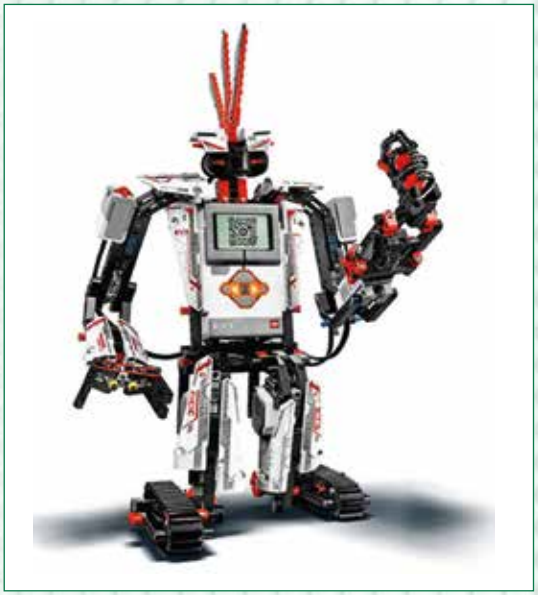
- 550 peças LEGO Technic;
- 1 EV3 Brick;
- 3 servo motores;
- 1 sensor de toque;
- 1 sensor de cor;
- 1 sensor infravermelho;
- 1 comando remoto;
- 1 conjunto de cabos;
- 1 cabo USB.

O 31313 LEGO Mindstorms EV3 inclui 17 modelos diferentes de robôs para construção. Com este kit, é possível montar, por exemplo, o *Everstorm*, robô em formato de humanoide, o *Spiker*, em estilo escorpião e o *Reptar*, uma cobra robótica que chacoalha e ataca o comando. Além disso, a linha de peças da LEGO *Technic* pode ser adicionada para modificar as funcionalidades.

Os principais modelos que podem ser montados com as peças disponíveis no kit 31313 LEGO Mindstorms EV3 (MINDSTORMS, 2016b) são:

Principais modelos que podem ser montados com as peças disponíveis no Kit 31313 LEGO Mindstorms EV3

Everstorm:



Spiker:

Reptar:

Tracker:

Gripper:

2.2 APRESENTANDO OS SENSORES DA PLATAFORMA

Os sensores são equipamentos eletrônicos capazes de responder a estímulos de natureza física, tais como: temperatura, luminosidade, distância de algum objeto, entre outros.

O kit do LEGO Mindstorms contempla os sensores de: luz, toque, som, ultrassônico e cor. Outros sensores também podem ser adquiridos, como sensor de bússola, acelerômetro, temperatura, entre outros, porém, estes costumam ser sensores de terceiros e, por esse motivo, o ambiente de desenvolvimento deve ser adaptado para suportá-los (com a instalação de *plugins*, por exemplo).

A seguir serão apresentados apenas os sensores padrões da plataforma LEGO Mindstorms.

2.2.1 Sensor de Luz (*Light Sensor*)

O sensor de luz é um dos sensores que dá visão ao NXT, permitindo-lhe distinguir o claro do escuro, ler a intensidade luminosa de uma sala e até mesmo medir a intensidade de luz em superfícies coloridas (como por exemplo, identificar se uma linha é clara ou escura).

Este sensor também pode ser utilizado como lâmpada, permitindo ao NXT acender ou desligar um *light emitting diode* (LED), que nada mais é do que um diodo emissor de luz, que se encontra nele mesmo (neste caso, o sensor funciona como um dispositivo de entrada (lendo a luminosidade do ambiente) ou saída de dados (acendendo o LED).

A Figura 9 apresenta a imagem de um sensor de luz.

Figura 9 – Sensor de luz
Fonte: LEGO Engineering (2013).

2.2.2 Sensor de Toque (*Touch Sensor*)

O sensor de toque simula o sentido do tato no NXT, detectando quando o botão do sensor está pressionado, que costuma acontecer quando o robô colide com alguma superfície.

A Figura 10 apresenta a imagem do sensor de toque.

Figura 10 – Sensor de toque
Fonte: LEGO Engineering (2013).

2.2.3 Sensor Ultrassônico (*Ultrasonic Sensor*)

Permite ao NXT detectar a que distância os objetos estão dele. Pode ser utilizado para evitar os obstáculos, detectar e medir distâncias e ainda detectar movimentos.

O sensor de ultrassom pode medir distâncias, tanto em cm quanto em polegadas. É capaz de medir distâncias entre 0 e 127 cm com uma precisão de aproximadamente 3 cm.

Objetos maiores e concretos retornam leituras mais precisas, enquanto objetos macios ou que possuam curvas (como bolas), podem retornar leituras imprecisas. A Figura 11 apresenta a imagem deste sensor.

Figura 11 – Sensor ultrassônico
Fonte: LEGO Engineering (2013).

2.2.4 Sensor de Som (*Sound Sensor*)

O sensor de som faz com que o NXT **ouça**. Este sensor detecta os decibéis (dB) do ambiente (medida da intensidade do som), podendo medir até 90 dB. Como trabalhar com decibéis pode causar alguma confusão, as medidas do sensor são apresentadas em porcentagem. A Figura 12 apresenta a imagem deste sensor.

Figura 12 – Sensor de som
Fonte: LEGO Engineering (2013).

2.2.5 Sensor de Cor (*Color Sensor*)

O sensor de cor trabalha com o padrão *red, green, blue* (RGB), emitindo uma luz e retornando o valor de cada cor, que varia em uma faixa de 1 a 6. O Quadro 1 apresenta as cores e seus respectivos valores numéricos.

Número	Cor
1	Preto
2	Azul
3	Verde
4	Amarelo
5	Vermelho
6	Branco

Quadro 1 – Valores numéricos das cores
Fonte: Help LEGO Mindstorms NXT-G (2016).

Assim como o sensor de luz, o sensor de cor também permite acender LEDs, o que o transforma em um dispositivo de entrada (lendo as cores) ou saída de dados (acendendo LEDs). A Figura 13 apresenta o sensor de cor.

Figura 13 – Sensor de cor
Fonte: Laurens (2012).

2.2.6 Servo Motores

Os servo motores possuem sensor de rotação embutido, permitindo que se verifique em quantos graus o servo motor girou. O sensor de rotação mede as rotações do motor em graus,

ou em voltas completas (com precisão de aproximandamnte 1 grau), lembrando que uma volta equivale a 360 graus. A Figura 14 apresenta um servo motor.

Figura 14 – Servo motor

Fonte: Lego Engineering (2013).

2.2.7 Outros Sensores

É possível adquirir outros sensores para a plataforma. Estes podem ser importados de sites americanos ou adquiridos em lojas no Brasil. Compatíveis com a plataforma LEGO Mindstorms, é preciso apenas instalar os *plugins* correspondentes (que podem ser baixados dos sites das fabricantes) ao ambiente de desenvolvimento (por exemplo, o LEGO Mindstorms NXT-G).

A Figura 15 apresenta o sensor de infravermelho apanhador (*Infrared Seeker Sensor)*, com o qual é possível tornar os robôs capazes de detectar fontes de luz infravermelha e determinar a sua direção e a intensidade relativa, graças a cinco detectores infravermelhos dispostos em intervalos de 60º.

Figura 15 – Sensor infravermelho apanhador

Fonte: Mindstorms (2016c).

A Figura 16 apresenta o sensor infravermelho de ligação (*Infrared Link Sensor*), com o qual é possível a comunicação dos robôs com outros dispositivos, incluindo o Mindstorms RCX.

Figura 16 – Sensor infravermelho de ligação
Fonte: Hitechnic (2016).

2.3 PROCESSADOR NXT *BRICK*

Este é o **cérebro** do LEGO Mindstorms (detalhes na Seção 2.1.3). Mesmo criando uma estrutura complexa de peças e engrenagens, é esta peça que vai ler os dados dos sensores, processar e realizar atividades com base nestes.

É neste processador que os sensores e servo motores devem ser conectados via cabos de dados (padrão proprietário da LEGO, ambas as extremidades com *plugs* RJ-12). O processador também possui um *display* monocromático e botões para seleção de opções.

Além das características físicas citadas, é no processar NXT *Brick* que estão os softwares que vêm de fábrica com o kit e que permitem:

a) ligar/desligar recursos como *Bluetooth* e som do Mindstorms;
b) navegar nos arquivos e programas instalados;
c) testar/ler dados dos sensores;
d) realizar a programação (ambiente bem limitado), usando o próprio processador, neste caso, não é necessário um computador conectado ao Mindstorms via USB, entretanto, o ambiente é muito limitado.

2.3.1 Portas de Comunicação

O NXT possui três portas de saída (*output*), onde se conectam os servo motores. Estas portas se encontram acima do visor (portas A, B e C). Abaixo dos botões se encontram quatro portas de entrada (*input*), onde se conectam os sensores (portas 1, 2, 3 e 4). A Figura 17 apresenta estas portas. É importante atentar em qual porta se encontra cada sensor/servo motor, para evitar erros na programação.

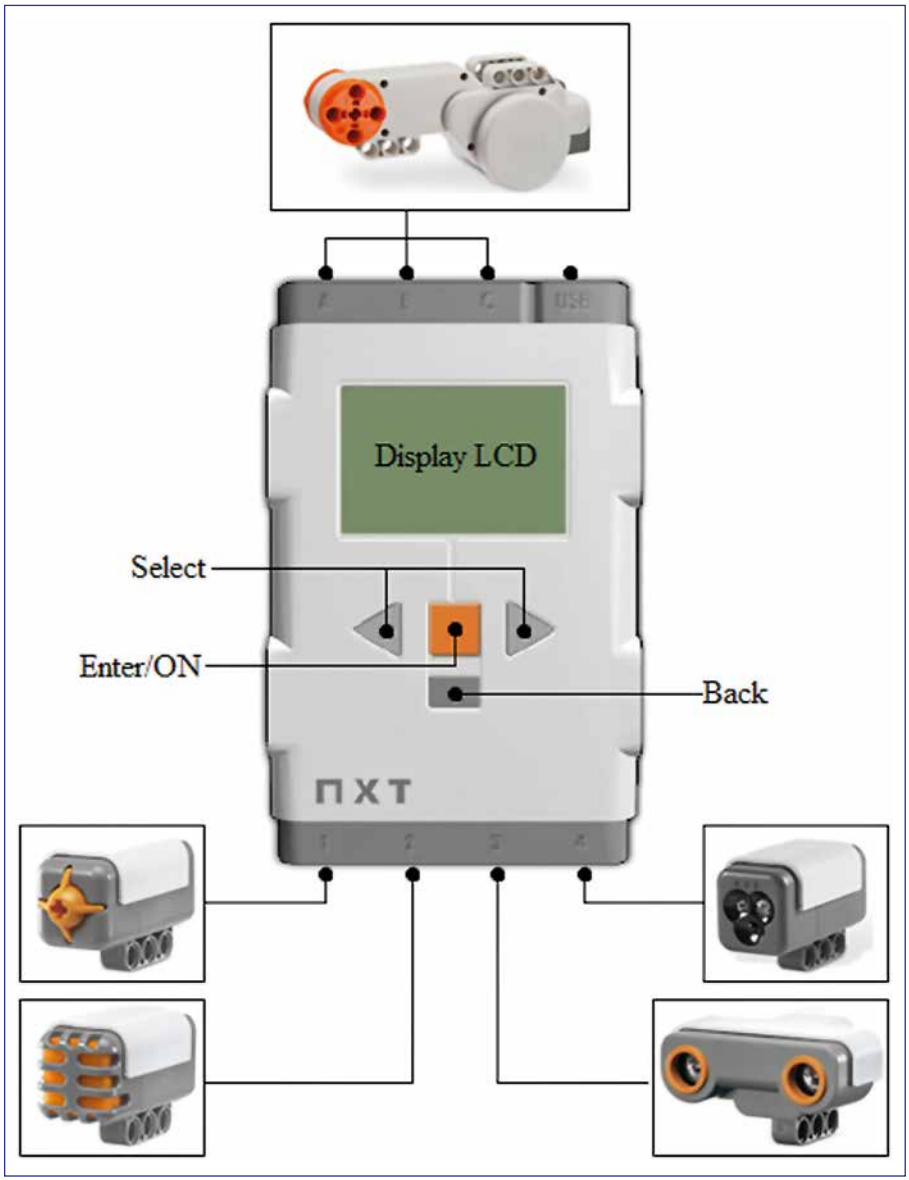


Figura 17 – Representação das portas e comandos do NXT

Fonte: Adaptado de National Instruments (2009).

Existe um padrão adotado para qual sensor será utilizado em cada porta, este segue no Quadro 2.

Sensor	Figura	Porta
Sensor de toque		1
Sensor de som		2
Sensor de cor		3
Sensor ultrassônico		4

Quadro 2 – Padrão adotado nas ligações dos sensores às portas de entrada do NXT
Fonte: Autoria própria (2015).

Como se pode observar na Figura 17, existe também no canto superior direito do *Brick* uma interface USB para comunicação com o computador. Através dela, pode-se realizar o download dos programas do computador para o NXT e também coletar dados do NXT para o computador.

Se for necessário, esta comunicação pode ser substituída pelo *Bluetooth*, o que dá maior mobilidade ao NXT, já que ele não ficará limitado ao tamanho do cabo USB. Entretanto, o uso do *Bluetooth* reduz consideravelmente a duração da bateria, não sendo aconselhável aos kits que estão sendo alimentadas por pilha AA.

2.3.2 O Visor

A Figura 18 apresenta o *liquid crystal display* (LCD) do NXT. Na parte superior esquerda do visor, há ícones que mostram o status da conexão USB ou *Bluetooth*. Já no canto superior direito há um ícone informando sobre a carga da bateria do mesmo. Ao centro, o menu para navegação.

Figura 18 – *Display* do NXT
Fonte: Adaptado de National Instruments (2009).

Para navegar nos menus do LEGO Mindstorms, utilizam-se os botões de navegação (Figura 19).

Figura 19 – Botões de navegação do NXT
Fonte: Adaptado de National Instruments (2009).

O botão central (em laranja) é a tecla de *Enter* do NXT, utilizado para entrar nos menus, assim como para ligar o mesmo. As setas direcionais são utilizadas para a navegação nos menus. O botão inferior (cinza mais escuro) serve para sair dos menus e voltar aos menus anteriores, assim como para desligar o NXT quando pressionado por algum tempo.

2.4 AMBIENTES DE DESENVOLVIMENTO

Existem várias formas de programar para o LEGO Mindstorms. É possível utilizar o ambiente de desenvolvimento presente no próprio processador (este ambiente, embora limitado, permite o desenvolvimento de programas de forma simples e rápida). Para isso, basta ligá-lo

e acessar o menu *NXT Program*, o NXT irá apresentar as configurações para as conexões dos sensores no *Brick,* a partir disso, poderá ser feita a programação utilizando cinco operações (estas disponibilizadas pelo próprio NXT), ou ambientes externos, instalados em computadores *Desktop*. Neste caso, os aplicativos devem ser instalados no LEGO Mindstorms via cabo USB ou comunicação *Bluetooth*.

Dos ambientes externos, destacam-se o LEGO Mindstorms NXT-G, este um ambiente visual nativo para a plataforma, e a API leJOS, este permite desenvolver software usando a linguagem Java. Outras linguagens também são suportadas pelo Mindstorms, mas não serão detalhadas nesta obra.

2.4.1 NXT-G

É o ambiente de desenvolvimento para o LEGO Mindstorms padrão da plataforma. Este é todo baseado em abas e no estilo *Drag and Drop*, o que torna muito fácil a aprendizagem e o desenvolvimento de programas (a LEGO informa na embalagem do Mindstorms NXT 8547 que este ambiente pode ser usado por crianças a partir de 10 anos, devido à facilidade de uso).

A Figura 20 apresenta a tela principal do ambiente de desenvolvimento NXT-G. À esquerda é possível ver os grupos de componentes e ao meio, o ambiente de desenvolvimento.

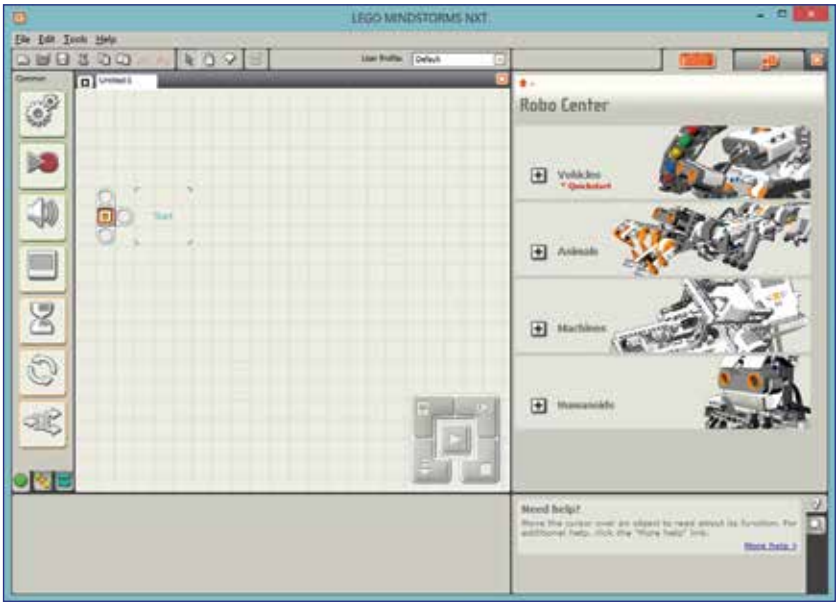


Figura 20 – Ambiente NXT-G

Fonte: Autoria própria (2016).

É importante ressaltar que este ambiente vem em um CD para usuários do LEGO Mindstorms NXT 8547. Aos usuários do kit educacional 9797 este deve ser baixado do site da LEGO. O arquivo é um ISO (imagem de um CD) e está disponível em: <http://www.lego.com/en-us/mindstorms/downloads/nxt-software-download>. Essa página também apresenta os passos para a instalação do ambiente.

2.4.2 leJOS

O *framework* leJOS NXJ, também chamado de leJOS, é um ambiente de programação Java para o LEGO Mindstorms NXT. O leJOS é composto por:

a) *firmware* substituto para o NXT, onde está inclusa uma máquina virtual Java (este deve ser instalado no processador do LEGO Mindstorms, substituindo o NXT que vem com o processador);

b) uma biblioteca de classes Java (classes.jar) que implementam a leJOS *Aplication NXJ Programming Interface* (API);

c) um *linker* para ligar classes de usuários Java com classes.jar para formar um arquivo binário que possa ser carregado e executado no NXT;

d) ferramentas para instalar o *firmware*, carregar programas, realizar depuração (*debug*) entre outras;

e) a API PC para que programas feitos no computador possam se comunicar com programas leJOS NXJ usando fluxos Java através de *Bluetooth* ou USB, ou utilizando o LEGO *Communications Protocol* (LCP), um protocolo específico para comunicação.

Como o leJOS NXJ é uma substituição do *firmware* original do NXT, após a instalação deste não é mais possível programar utilizando a linguagem NXT-G, sendo todos os arquivos armazenados no antigo *firmware* perdido. É possível também restaurar o *firmware* padrão da LEGO Mindstorms, usando o próprio ambiente de desenvolvimento do LEGO Mindstorms, a partir do menu *Tools* – *Update* NXT *Firmware*.

O leJOS é um projeto *Open Source*, e seu website oficial está hospedado na *sourceforge*. Originalmente criado a partir do projeto TinyVM, foi adaptado para implementar uma JavaVM para o sistema LEGO Mindstorms RCX (a primeira versão do LEGO Mindstorms lançada em 1998), posteriormente atualizada para a versão NXT, em 2006.

Dentre as vantagens em utilizar o *framework* leJOS para o desenvolvimento de aplicativos para o LEGO Mindstorms, destacam-se:

a) usa a sintaxe da linguagem Java;

b) suporta programação orientada a objeto;

c) é um projeto *Open Source* e tem vários colaboradores;

d) permite a escolha do ambiente de desenvolvimento, incluindo *integrated development environment* (IDEs) (ambiente de desenvolvimento integrado) Netbeans e Eclipse, estas já conhecidas pelos desenvolvedores Java;

e) *plugins* para ambos (Netbeans e Eclipse);

f) suporte completo para *Bluetooth*, USB, I2C e protocolos RS485;

g) suporta as últimas características da linguagem Java 1.6;

h) suporte multiplataforma;

i) suporta *multithreading*;

j) sistema de arquivo flash acessado pelas classes do pacote *java.io* padrão do Java;

k) suporta o registro de dados e captura remota dos logs;

l) suporte de som, incluindo a reprodução de arquivos WAV (formato padrão de arquivos de áudio da Microsoft e *International Business Machines* – IBM) de 8 bits;

m) oferece dezenas de programas de exemplo na internet.

AMBIENTE NXT-G

NXT-G é o ambiente de desenvolvimento para o LEGO Mindstorms, é todo baseado em abas e no estilo *Drag and Drop*, que torna fácil para aprendizagem e desenvolvimento de programas. A Figura 21 apresenta a tela principal do ambiente de desenvolvimento NXT-G.

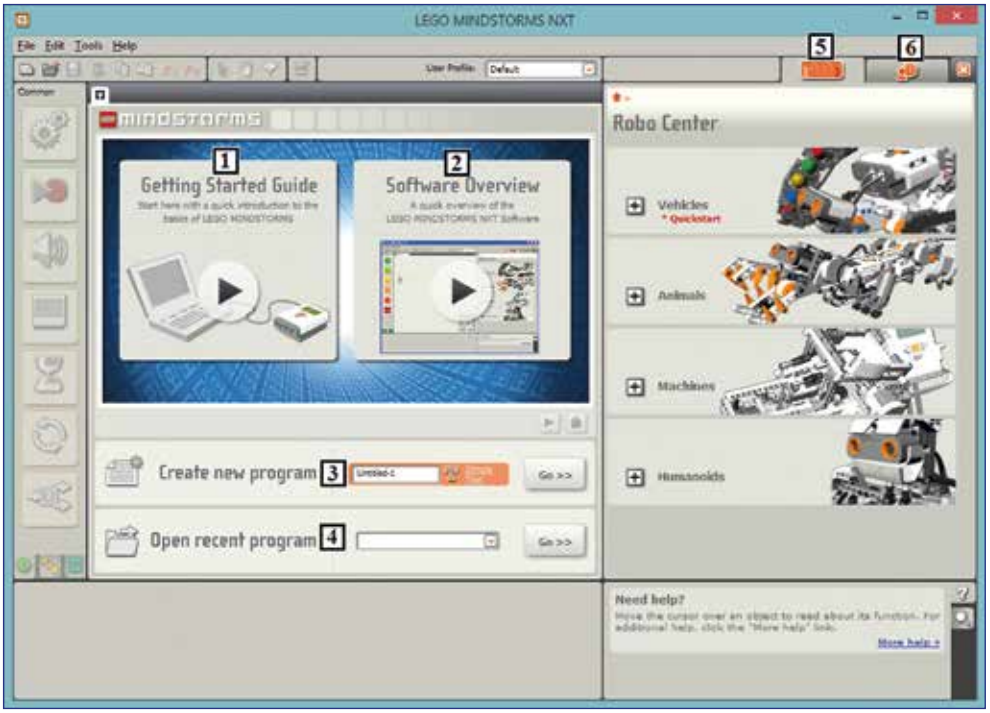


Figura 21 – Tela inicial do NXT-G

Fonte: Autoria própria (2016).

As partes da tela de desenvolvimento são apresentadas no Quadro 3.

	Nome	Descrição
1	*Getting Started Guide*	Pequeno tutorial sobre ações básicas sobre o robô.
2	*software overview*	Pequeno tutorial sobre as funcionalidades do software.
3	*Create New Program*	Opção para criação de um novo programa.
4	*Open Recente Program*	Opção para abrir um programa já salvo no computador.
5	*Robo Center*	É onde se aprende a montar e a programar o robô. Há 16 programas de exemplos que introduzem cada modelo e novos blocos.
6	*My Portal*	Acesso ao download de novas ferramentas.

Quadro 3 – Divisão da tela do ambiente de desenvolvimento

Fonte: Autoria própria (2016).

Para criar um aplicativo novo há dois métodos:

a) Método 1: informar o nome do projeto na área representada pela Figura 22 e após isso clicando no botão Go >>;

b) Método 2: através do menu *File* > *New* ou pelo atalho Ctrl + N, representado pela Figura 23.

Figura 22 – Criação de um novo programa através do Método 1

Fonte: Autoria própria (2016).

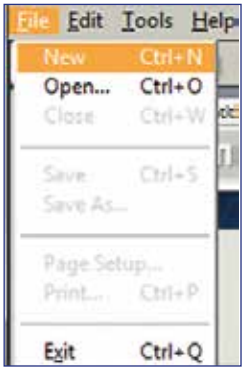


Figura 23 – Criação de um novo programa através do Método 2

Fonte: Autoria própria (2016).

Após isso, a tela de desenvolvimento é apresenta na Figura 24.

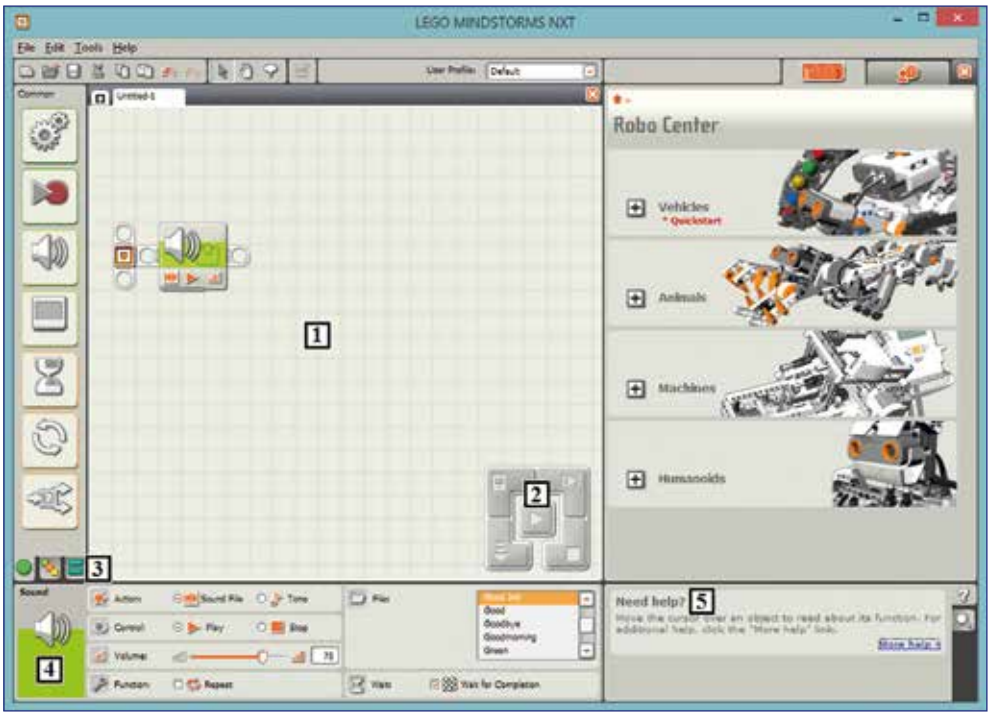


Figura 24 – Tela de desenvolvimento do NXT-G

Fonte: Autoria própria (2016).

Algumas características desta tela são apresentadas no Quadro 4.

Nome		Descrição
1	*Workspace*	Onde serão colocados os componentes referentes ao programa.
2	*The Controller*	Permite fazer o download de programas do computador para o robô. Fornece também informações sobre o robô que estiver conectado.
3	*Palettes Programming*	Contém todos os blocos que podem ser utilizados para realizar a programação do robô.
4	*Configuration Panel*	É onde se pode ajustar as configurações de um bloco de programação.
5	*Need Help?*	Abre um manual online de todos os blocos do LEGO Mindstorms (em inglês)

Quadro 4 – Características da tela do ambiente de desenvolvimento

Fonte: Autoria própria (2016).

Para o desenvolvimento dos aplicativos presentes nesta obra, será considerada a utilização de um robô veículo, sendo possível o desenvolvimento deste em todos os kits Mindstorms. Podemos encontrar modelos como o LEGO Mindstorms NXT 2.0 9797 (WS KITS EDUCACIONAL, 2016) e o LEGO Mindstorms NXT 2.0 8547 (MINDSTORMS, 2016a).

Veículos da plataforma Mindstorms

LEGO Mindstorms NXT 2.0 9797:

LEGO Mindstorms NXT 2.0 8547:

Para o robô veículo é possível desenvolver e testar um aplicativo simples (Figura 25). Os componentes utilizados foram arrastados da paleta da esquerda para o centro da tela, sequencialmente. Como o processamento é sequencial, um comando só será executado se o anterior for concluído.

Figura 25 – Exemplo simples de um programa na plataforma NXT-G

Fonte: Autoria própria (2016).

Ao ser executado, o aplicativo modificará o conteúdo apresentado pelo *display* LCD do NXT (componente 1 da Figura 25), seguido da execução de um áudio (componente 2 da Figura 25) e, por fim, são acionados os motores ligados às saídas B e C do NXT (componente 3 da Figura 25), fazendo com este se movimente para frente. Após estes comandos, o aplicativo será encerrado.

Inicialmente, não foram modificadas as propriedades dos componentes, sendo que estes executaram tarefas padrão. O que cada componente executa pode ser visualizado graficamente no ícone deste na tela.

Observe-se que o componente 1 possui uma pequena careta (*smile*) no canto inferior direito, isso significa que o componente *display* apresentará uma imagem no *display* do dispositivo.

Em relação ao componente 2, o primeiro ícone significa que será executado um arquivo de som (*Sound File*), o segundo ícone indica que o som será executado e, por fim, o terceiro ícone indica o volume do som. Finalmente, o componente 3 possui uma indicação de C e B, sendo estes os motores que serão acionados, assim com um indicador de direção e potência na parte inferior do componente.

Entretanto, além de adicionar os componentes no ambiente de desenvolvimento, é possível mudar também suas propriedades. As propriedades se referem às características dos componentes, como eles serão executados. Para o exemplo, as propriedades foram modificadas conforme a Figura 26.

Nas configurações dos componentes, é solicitada a apresentação de um *smile* no *display* do LEGO Mindstorms, seguido de um som de *Good Job*, e, por fim, os motores ligados ás portas B e C do NXT giram para frente uma rotação completa com a potência de 75%, conforme observado nas propriedades do *Display* (parte superior da Figura), som (parte central) e motor (parte inferior).

Figura 26 – Configuração dos componentes utilizados no exemplo citado

Fonte: Autoria própria (2016).

Após o desenvolvimento do aplicativo, deve-se conectar o computador ao LEGO Mindstorms, via cabo USB (que acompanha o kit) ou estabelecer uma conexão via *Bluetooth*, para na sequência compilar, instalar e executar o aplicativo no NXT, processo este que pode ser simplificado clicando no botão *Download and Run* apresentado na Figura 27.

Após a instalação e execução do aplicativo, as ações poderão ser observadas no NXT.

Para um primeiro aplicativo, alguns cuidados especiais devem ser tomados para o bom desempenho e para minimizar os erros. Primeiramente conferir a ligação dos motores/sensores ao processador NXT *Brick*. É comum tentar acionar motores ligados a porta B e C, que são os padrões do componente, porém, não funcionar porque na verdade estas estão ligadas na porta A e B, por exemplo.

Outro cuidado importante é com os comandos utilizados; como o processamento é sequencial, um comando só será executado se o anterior for concluído. Desse modo, se o comando anterior desenha uma figura no *display*, e é seguido de outro comando que desenha outra figura, a primeira figura será substituída pela segunda.

Outra dica é que ao final do processamento o *display* é limpo, desta forma, se a lógica do aplicativo consiste em apresentar uma única imagem na tela do LEGO Mindstorms, provavelmente o usuário não terá tempo de vê-la, já que a mesma será apresentada rapidamente e, ao ser finalizado, o sistema limpará o conteúdo do *display*. O mesmo acontece se duas figuras forem apresentadas sequencialmente, uma após a outra.

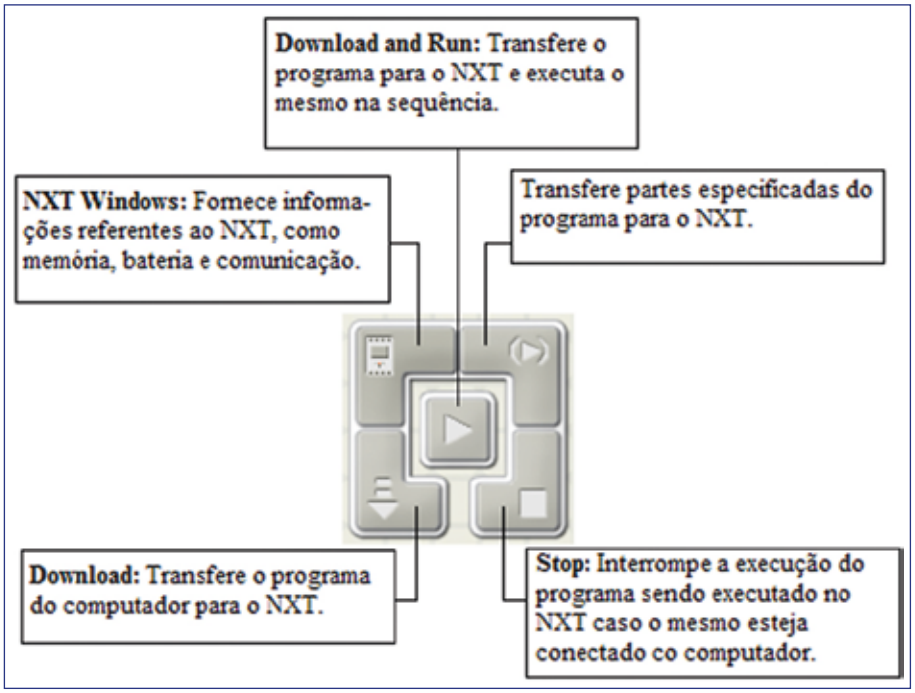


Figura 27 – Comando para compilação, instalação e execução do aplicativo

Fonte: Autoria própria (2016).

Em qualquer momento do processo de desenvolvimento, o aplicativo poderá ser salvo para ser utilizado posteriormente. Para salvar o aplicativo, escolhe-se a opção *File* > *Save*, sendo armazenado um arquivo com a extensão *.rbt* no disco.

Como existe uma grande quantidade de componentes, estes são categorizados em paletas e apresentados na sequência.

4 PALETA DE COMPONENTES PADRÃO

As paletas são separadas em três categorias, sendo elas: *Common Pallete*: contém os blocos de programação mais utilizados (Figura 28 (a)); *Complete Pallete*: contém todos os blocos de programação (Figura 28 (b)); e, *Custom Pallete*: contém os blocos de programação que foram criados ou feito o download pelo usuário (Figura 28 (c)).

Figura 28 – Paletas de programação do NXT-G

Fonte: Autoria própria (2016).

4.1 PRINCIPAIS COMPONENTES DA PALETA PADRÃO (*COMMON PALLETE*)

A paleta padrão contém os componentes visuais mais utilizados na plataforma, apresentados em detalhe no Quadro 5.

Símbolo	Nome	Descrição
	*Move*	Configura os movimentos que os motores podem realizar, individualmente ou em conjunto.
	*Record/Play*	Pode gravar a ação dos motores e executá-los posteriormente.
	*Sound*	Emite sons diversos ou tons musicais pré-gravados no NXT.
	*Display*	Exibe informações na tela no NXT, podendo estas serem imagens ou dados obtidos pelos sensores.
	*Wait*	Produz uma pausa no programa. Esta pausa pode ser um determinado tempo ou em função de algum evento dos sensores.
	*Loop*	Usado para determinar a repetição de uma sequência de ações que foram programadas.
	*Switch*	Permite ao programa avaliar uma condição e, dependendo da resposta, executar determinada ação.

Quadro 5 – Componentes visuais da paleta padrão (*Common*)

Fonte: Autoria própria (2016).

4.1.1 Comando *Move*

O comando *Move* é representado pela Figura 29. Como pode ser observado, o componente possui algumas representações visuais que refletem em suas propriedades.

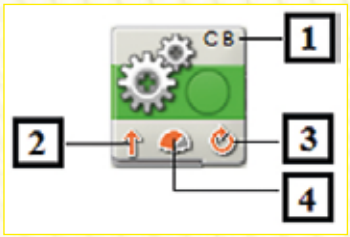


Figura 29 – Representação do ícone do componente *Move*

Fonte: Autoria própria (2016).

Estas propriedades são:

1. A quais portas de saída estão conectados os servo motores.
2. A direção que o NXT irá se movimentar.
3. A potência dos servo motores.
4. A maneira como foram configurados os movimentos dos servo motores.

Para mudar estas propriedades, basta acessar a paleta na parte inferior da tela (Figura 30).

Figura 30 – Painel de configuração do componente *Move*

Fonte: Autoria própria (2016).

Suas propriedades são:

1. *Port*: seleciona os motores que serão controlados. Caso sejam escolhidos dois motores, estes serão sincronizados, trabalhando na mesma direção e potência.
2. *Direction*: controla em qual direção os motores irão se mover (para frente, para trás ou parado).

3. *Steering*: quando mais de um motor é selecionado é possível fazer com que o robô faça curvas, indicando neste campo o sentido da mesma.
4. *Power*: nível de potência dos motores (0% a 100%).
5. *Duration*: a duração dos movimentos dos motores pode ser fornecida em:
   a) *Rotations*: equivalente a uma volta completa do eixo;
   b) *Degrees*: equivalente a quantos graus o motor irá girar, por exemplo, uma volta completa no eixo equivale a 360 graus, meia volta 180 graus;
   c) *Seconds*: o motor ficará ativo pelo tempo predeterminado;
   d) *Unlimited*: o robô irá se movimentar ininterruptamente ou até encontrar um critério de parada realizado na programação.
6. *Next action*: define a próxima ação a ser executada pelos motores, sendo elas:
   a) *Break*: fará com que o robô pare instantaneamente ao realizar a quantidade de movimentos predeterminados;
   b) *Coast*: desligará os motores, porém o robô continuará se movimentando porinércia, fazendo com que tenha uma parada mais suave.

4.1.2 Comando *Sound*

Esta função é utilizada para executar um som, ou uma sequência de sons e ainda reproduzir notas musicais durante a execução de um programa.

A Figura 31 apresenta o painel de configuração quando selecionada a opção *Sound File*.

Figura 31 – Painel de configuração do componente *Sound*

Fonte: Autoria própria (2016).

Com a opção *Sound File* selecionada, é possível selecionar um arquivo de áudio no dispositivo, mudando com isso algumas características:

1. *Action*: a opção *Sound File* significa que serão utilizados arquivos de som já gravados no NXT.

2. *Control*: inicia (*Play*) ou o interrompe (*Stop*) a execução de um som.
3. *Volume*: estabelece a intensidade do som.
4. *Function*: ao selecionar *Repeat*, o som se repete indefinidamente, caso contrário, o som é executado somente uma vez.
5. *File*: arquivo de som a ser reproduzido.
6. *Wait*: ao marcar a opção *Wait for Completion* o tom escolhido no compo *File* irá executar até o fim antes de permitir que o programa continue. Caso a opção esteja desmarcada, o tom será reproduzido enquanto o próximo bloco de programação é executado.

O painel de configuração do som, quando selecionada a opção *Tone,* é apresentado na Figura 32. Como se pode observar, somente a opção 7 é alterada ao selecionar a opção *Tone* na propriedade *Action*, a qual é descrita a seguir.

Figura 32 – Painel de configuração do componente *Sound* na opção *Tone*

Fonte: Autoria própria (2016).

7. *Note:* pode-se escolher uma nota para ser executada, através da sua simbologia ou através do teclado. Nesta opção também é possível definir a duração do tom em segundos.

4.1.3 Comando *Display*

Exibe informações na tela do NXT, desde ícones até dados obtidos pelos sensores.

Para exibir informações no *display* do NXT, podem-se selecionar várias maneiras através da opção *Action* (no painel de configuração do componente). As Figuras 33, 34 e 35 demonstram estas opções.

Figura 33 – Painel de configuração do componente *Display* do tipo Imagem

Fonte: Autoria própria (2016).

A seguir, a apresentação de cada propriedade do componente:

1. *Action*: escolha do conteúdo a ser exibido: Imagem, Texto, Desenho ou *Reset* para exibir o ícone padrão.
2. *Display*: sobrescrever o conteúdo que já existia no *display* ou apagá-lo (*Clean*).
3. *Files*: imagens disponíveis para exibição. Ao clicar em cada arquivo será exibido uma pré-visualização da imagem.
4. *Position*: exibe a imagem selecionada na opção *Files*, e a sua posição no *display* (X – Horizontal / Y – Vertical).

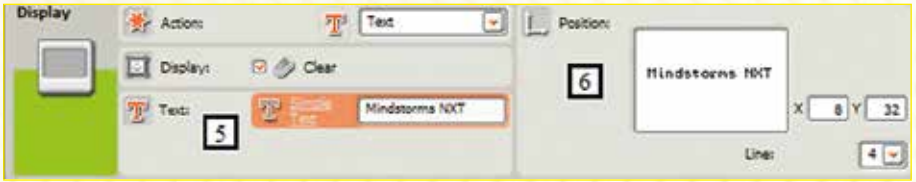


Figura 34 – Painel de configuração do componente *Display* do tipo Texto

Fonte: Autoria própria (2016).

5. *Text*: Caso esta opção seja setada em *Action*, é possível digitar um texto para ser exibido no *display* do NXT.
6. *Position*: o *display* do NXT possui oito linhas e é possível escolher a linha em que o texto irá aparece.

Figura 35 – Painel de configuração do componente *Display* do tipo Desenho

Fonte: Autoria própria (2016).

7. *Type*: tipo do desenho que será apresentado no *display* do NXT (*Point, Circle, Line*). No caso do círculo, define-se o seu raio e no caso da linha, estabelece sua posição de início e fim.
8. *Position*: posição do objeto no *display* do NXT.

4.1.4 Comando *Loop*

Este comando condiciona a repetição de uma sequência de código de programação de diversas maneiras. Equivalente a comandos como *for*, *while* ou *repeat* de linguagens de programação tradicionais.

O controle da execução deste componente pode ser feito pelo campo *Control*. Para exemplificar, a Figura 36 apresenta esta propriedade valorizada com *Forever*. Quando selecionada a opção *Forever* no campo *Control,* as ações que compõem este *Loop* acontecem **eternamente**, até o programa ser finalizado ou até acabar a bateria do NXT.

Figura 36 – Painel de configuração do componente *Loop* (*Forever*)

Fonte: Autoria própria (2016).

1. *Show*: caso a opção *Counter* esteja marcada, o programa registra quantas vezes o *Loop* foi efetuado;

Quando o campo *Control* é valorizado com a opção *Time,* o qual permite executar o *Loop* por um determinado tempo, as propriedades do componente seguem conforme Figura 37.

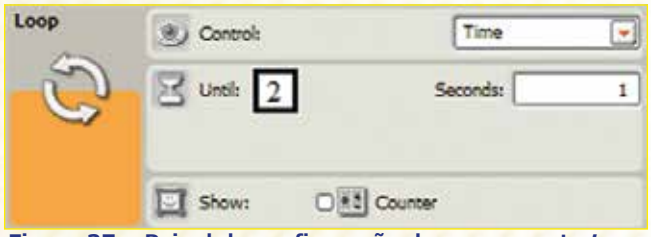


Figura 37 – Painel de configuração do componente *Loop* (*Time*)

Fonte: Autoria própria (2016).

2. *Until*: campo para informar quantos segundos o *Loop* irá executar.

Quando for selecionada a opção *Count* no campo *Control* localizado no painel de configurações do componente, é apresentada a janela demonstrada na Figura 38.

Figura 38 – Painel de configuração do componente *Loop* (*Count*)

Fonte: Autoria própria (2016).

3. *Until*: campo para informar quantas vezes as ações que se encontram dentro do *Loop* irão se repetir. Por padrão este componente executará apenas uma vez, mas o valor pode ser alterado pelo usuário.

Ao ser selecionada a opção *Logic* no campo *Control*, o *Loop* repetirá até alcançar uma condição lógica, esta fornecida por outros componentes (conectados via *Data Hub*, mais detalhes na seção 5.1). A tela para configuração das propriedades é apresentada na Figura 39.

Figura 39 – Painel de configuração do componente *Loop* (*Logic*)

Fonte: Autoria própria (2016).

4. *Until*: é verificado o valor da condição (*True/False*) para encerrar as ações que se encontram dentro do *Loop*.

4.1.5 Comando *Switch*

Este componente permite que o programa tome decisões de acordo com o que foi predeterminado. É equivalente aos comandos *if* ou *switch* das linguagens de programação tradicionais. Este componente possibilita duas opções de teste: *Value* e *Sensor*.

Quando for selecionado a opção *Value* no campo *Control* localizado no painel de configurações do componente é apresentado a janela demonstrada na Figura 40.

Figura 40 – Painel de configurações do componente *Switch* valorizado com tipo *Logic*

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

1. *Type*: define o tipo do valor: *Logic* – Lógico, *Number* – Numérico ou *Text* – Texto.
2. *Display*: como o componente será exibido. Em *Flat View*, há duas linhas separadas que permitem visualizar a programação de cada condição; caso contrário, o bloco de comutação usará interface com abas para mostrar as sequências alternativas dos blocos de programação. Ao clicar em uma guia, é possível ver e editar os blocos e ver o que a condição fará.
3. *Conditions*: permite estabelecer as condições. Caso a opção de *Number* ou *Text* for selecionada, poderão ser adicionadas novas condições.
4. Permite definir a condição padrão a ser tomada.

Se a opção *Flat View* for desmarcada (permitindo a interface com abas, como demonstra a Figura 41), o usuário poderá adicionar linhas para a tabela que irá controlar mais sequências de blocos como na Figura 42.

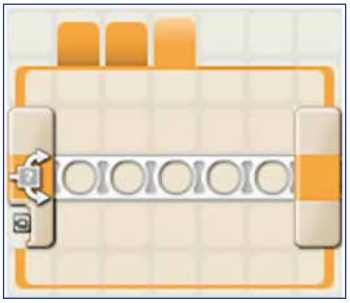


Figura 41 – *Switch* com abas

Fonte: Autoria própria (2016).

Figura 42 – Painel de configurações do componente *Switch* valorizado com tipo *Number*

Fonte: Autoria própria (2016).

4. Quando o *Switch* está setado para receber uma entrada numérica, usa-se este campo para ativar uma determinada opção de comutação.
5. Estes botões permitem adicionar ou excluir opções de comutação (ativos somente quando a opção *Flat View* está desmarcada).
6. Este botão define a opção padrão na lista. Quando o *Switch* estiver ajustado para o tipo numérico e chega um sinal que não corresponde a nenhuma das opções da lista definidas pelos números, o bloco será executado conforme a opção padrão.

Caso o usuário selecione a opção *Sensor* no campo *Control* localizado no painel de configurações do componente, poderá configurar os sensores como desejar.

Se for escolhido trabalhar com o sensor de toque como condição para o *Switch*, o painel de configurações é demonstrado pela Figura 43.

Figura 43 – Painel de configurações do componente *Switch* com sensor de toque

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

1. *Port*: porta na qual o sensor de toque é conectado, e que por padrão é definida como porta 1.
2. *Action*: nos botões de rádio, o usuário poderá definir a condição sobre o sensor de toque – *Pressed* (pressionado), *Released* (liberado) e *Bumped* (pressionado e em seguida liberado) – que fará o *Switch* executar os blocos correspondentes em determinado caso.

Caso o usuário optar por trabalhar com o sensor de som, o painel de configurações é demonstrado na Figura 44.

Figura 44 – Painel de configurações do componente *Switch* com sensor de som

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

3. *Port*: porta na qual o sensor de som é conectado, por padrão é definida como porta 2.
4. *Compare*: campo para definir o valor no qual o NXT irá realizar uma ação ao detectar (executando os blocos da parte superior do *Switch*). Por padrão é definido como sendo mais do que 50% da escala.

Se usar o sensor de luz, o painel de configurações é demonstrado na Figura 45.

Figura 45 – Painel de configurações do componente *Switch* com sensor de luz

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

5. *Port*: porta na qual o sensor de luz é conectado, por padrão definida como porta 3.
6. *Compare*: campo para definir o valor no qual o NXT irá realizar uma ação ao detectar (executando os blocos da parte superior do *Switch*). Por padrão é definido como sendo mais do que 50% da escala.
7. *Function*: se esta opção é selecionada, o sensor emite sua própria luz (acendendo um LED) e detecta se esta luz é refletida de volta para ele e em que intensidade.
8. Exibe o valor da intensidade luminosa da leitura atual (0-100%).

Caso o sensor de distância seja utilizado, o painel de configurações é demonstrado na Figura 46.

Figura 46 – Painel de configurações do componente *Switch* com sensor de distância

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

9. *Port*: porta na qual o sensor de distância é conectado, por padrão definida como porta 4.
10. *Compare*: campo para definir o valor no qual o NXT irá realizar uma ação ao detectar (executando os blocos da parte superior do *Switch*). Por padrão é definido como sendo menor que 50 polegadas ou 127 cm.

11. *Show*: opção para ler os valores em polegadas (*Inches*) ou cm (*Centimeters*).
12. Exibe o valor da distância da leitura atual.

Se utilizar os botões do NXT, o painel de configurações é demonstrado na Figura 47.

Figura 47 – Painel de configurações do componente *Switch* utilizando os botões do NXT

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

13. *Button*: seleciona qual botão do NXT vai enviar um sinal de *true* quando for ativado (encerrando o *Loop*).
14. *Action*: escolhe o comportamento do botão para executar determinada ação – *Pressed* (pressionado), *Released* (liberado) e *Bumped* (pressionado e em seguida liberado).

Se utilizar o sensor de rotação, o painel de configurações é demonstrado na Figura 48.

Figura 48 – Painel de configurações do componente *Switch* com o sensor de rotação

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

15. *Port*: escolhe a porta que deseja monitorar (ligada a um dos servo motores).
16. *Action*: escolhida a opção *Reset*, o *Switch* lê o valor corrente no sensor de rotação e, depois, irá setá-lo para zero. Escolhida a opção *Read*, o sensor de rotação lê o valor corrente sem ressetá-lo.
17. *Compare*: escolhe a direção do servo motor – para frente ou para trás.
18. Campo para definir o valor (em graus) no qual o NXT irá realizar uma ação ao detectar (executando os blocos da parte superior do *Switch*), definida por padrão em 360 graus.
19. Conta quantas rotações ou graus o sensor executou.
20. Mostra o valor atual lido do sensor de rotação.

Se utilizar o componente *Timer*, o painel de configurações é representado pela Figura 49.

Figura 49 – Painel de configurações do *Switch* para *Timer*

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

21. *Timer*: escolhe qual dos três *Timers* do NXT deseja-se monitorar.
22. *Action*: escolhida a opção *Reset*, o *Switch* lê o valor corrente *Timer* escolhido e depois irá zerar o mesmo. Escolhida a opção *Read*, o *Timer* lê o valor corrente sem zerá-lo.
23. *Compare*: campo para definir o ponto em que o programa irá executar. O valor padrão é de cinco segundos, assim, o programa irá executar os blocos de programação da parte superior do *Switch* se o tempo atual for superior a cinco segundos e executar os blocos da parte inferior se o tempo for menor do que cinco segundos.

Se utilizar o componente *Receive Message*, o painel de configurações é demonstrado na Figura 50.

Figura 50 – Painel de configurações do *Switch* para *Receive Message*

Fonte: Autoria própria (2016).

Os detalhes da configuração deste componente são:

24. *Message*: Este campo tem como função escolher o tipo de mensagem que o NXT espera receber – *Logic* – Lógico, *Number* – Numérico ou *Text* – Texto.
25. *Compare to*: Para comparar a mensagem recebida com uma mensagem para teste (se for escolhida anteriormente a opção texto ou numérico como formato).
26. *Mailbox*: Define por qual *Mailbox* a mensagem será recebida.

4.2 EXEMPLO DE PROGRAMAS QUE UTILIZAM OS PRINCIPAIS COMPONENTES

A seguir serão apresentados alguns aplicativos que utilizam os principais componentes da paleta padrão. Cada estudo de caso possui um objetivo específico, sendo este codificado usando os componentes da plataforma, clicando e arrastando-os na tela, além da mudança das propriedades destes componentes.

4.2.1 *Keep Forward*

Este programa faz o NXT sempre se movimentar para a frente, até encontrar um obstáculo, situação em que o aplicativo é finalizado. Este aplicativo é útil e é aconselhável como primeiro aplicativo, pois sua simplicidade permite desenvolver e focar mais no ambiente de desenvolvimento, como iniciar o projeto, instalar no NXT, executá-lo, entre outros conceitos simples, mas necessários.

Para o programa, será utilizado o sensor ultrassônico (para medir a distância do NXT em relação ao obstáculo), o componente *Move* (para fazer a movimentação para a frente) e o *Loop* (para fazer o mesmo permanecer em movimento enquanto não encontrar obstáculo).

A Figura 51 apresenta o aplicativo desenvolvido. Observe-se que foi utilizado basicamente o componente *Loop*, com condição de parada dos dados lidos do sensor de ultrassom, e o componente *Move*, que faz o NXT se movimentar para a frente.

Figura 51 – Representação do aplicativo *Keep Forward*

Fonte: Autoria própria (2016).

Este programa é formado pelo componente de *Loop*, que tem uma condição de parada de acordo com a leitura do sensor ultrassônico, ou seja, quando o NXT encontra um obstáculo a menos de 30 cm de distância, o programa é encerrado.

A Figura 52 apresenta o painel de configuração para o *Loop* para este programa.

Figura 52 – Painel de configuração do componente *Loop*

Fonte: Autoria própria (2016).

O segundo componente utilizado pelo programa é o *Move,* inserido dentro do bloco *Loop*. Sua função é movimentar ininterruptamente o NXT para a frente, até este encontrar um obstáculo, condição definida pelo componente *Loop.*

A Figura 53 apresenta o painel de configuração do componente *Move* para este programa.

Figura 53 – Painel de configuração do componente *Move*

Fonte: Autoria própria (2016).

Observe-se que a propriedade *Duration* foi definida como *Unlimited.* Dessa forma, o NXT se movimenta sem a parada entre uma volta e outra das rodas do veículo, já que, se escolhermos a duração 360 graus, por exemplo, a roda dará uma volta completa, irá parar, fazer o teste do componente *Loop* e, se não houver obstáculo, movimentará mais 360 graus.

Com isso o programa está pronto para ser instalado e executado no dispositivo, bastando pressionar o botão *Download and Run*, este já apresentado na Figura 27.

4.2.2 *Forward and Backward*

Pode-se aumentar a complexidade do programa, adicionando novos componentes visuais. Se adicionado um segundo *Loop* (Figura 54), o programa fará o NXT se movimentar para a frente até encontrar um obstáculo a menos de 30 cm de distância (mesma lógica do programa anterior). Encontrando o obstáculo, sairá do primeiro *Loop* e andará no sentido contrário (para trás) até encostar o sensor de toque em algum objeto, encerrando com isso o programa.

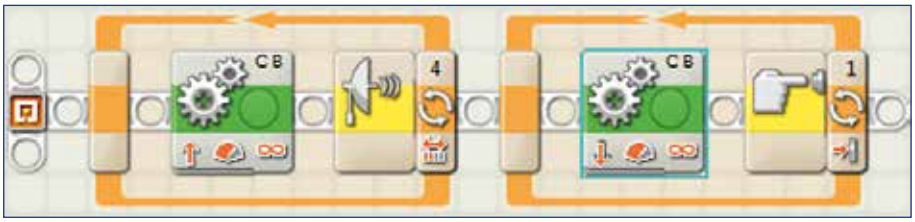


Figura 54 – Representação do aplicativo adicionando o segundo *Loop*

Fonte: Autoria própria (2016).

A Figura 55 apresenta o painel de configuração do segundo *Loop*, onde este é encerrado quando o sensor de toque é pressionado.

Figura 55 – Painel de configuração do segundo *Loop*

Fonte: Autoria própria (2016).

Já a configuração do componente *Move* referente ao segundo *Loop* é apresentada na Figura 56, diferenciando apenas pelo sentido do movimento.

Figura 56 – Painel de configuração do componente *Move* dentro do segundo *Loop*

Fonte: Autoria própria (2016).

4.2.3 *Forward and Backward Forever*

Um conceito muito comum na programação de robôs é o desenvolvimento de aplicativos que executem eternamente, ou seja, sempre recuperem dados de sensores e com base nestes tomem decisões.

Para apresentar este conceito, será aprimorado pela última vez o programa desenvolvido até então. Para tal mudança, basta colocar os dois componentes *Loop* definidos anteriormente dentro de um terceiro *Loop*, este do tipo *Forever*, conforme apresentado na Figura 57.

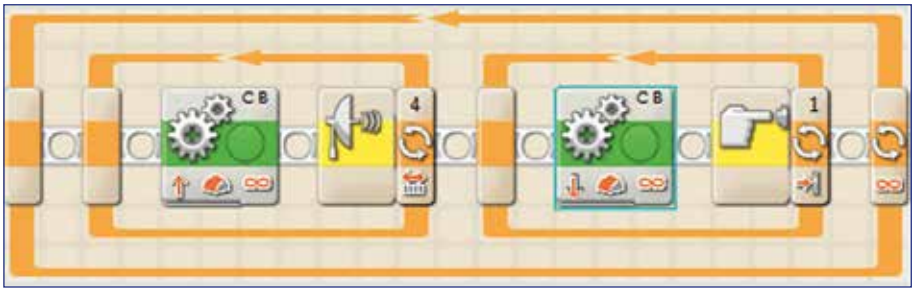


Figura 57 – Representação para que o aplicativo execute ininterruptamente

Fonte: Autoria própria (2016).

As configurações deste novo componente *Loop* adicionado são apresentadas na Figura 58.

Figura 58 – Painel de configuração do *Loop* externo

Fonte: Autoria própria (2016).

4.2.4 *Avoid Stuff*

Este programa faz o NXT sempre se movimentar para a frente e, ao encontrar um obstáculo, o mesmo faz um giro para seu lado esquerdo, até não encontrar mais obstáculo, seguindo em frente novamente.

Para o programa, será utilizado o sensor ultrassônico para medir a distância do obstáculo, o componente *Move*, para fazer a movimentação para frente ou lado, o *Loop*, para fazer o NXT permanecer em movimento enquanto não encontrar obstáculo ou manter-se virando para a esquerda enquanto um obstáculo estiver próximo e, por fim, um componente *Switch*, que identificará um obstáculo próximo ou não.

A Figura 59 apresenta o aplicativo desenvolvido.

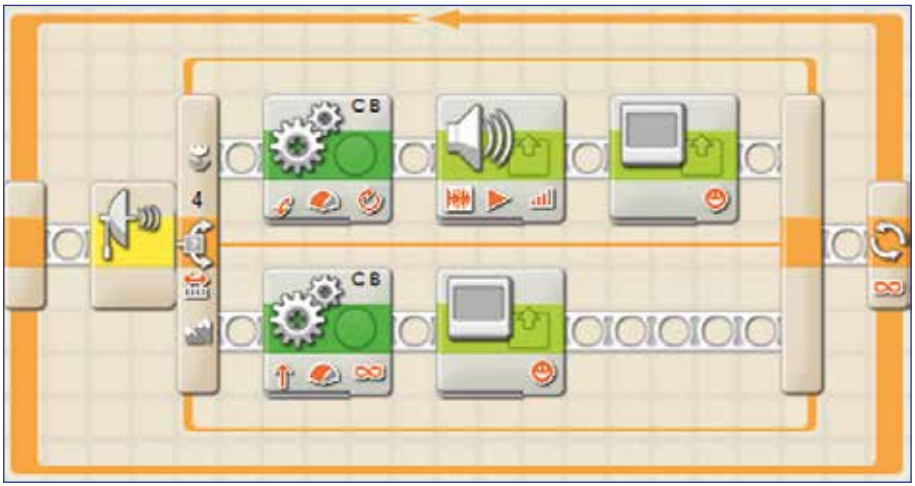


Figura 59 – Representação do aplicativo *Avoid Stuff*

Fonte: Autoria própria (2016).

A Figura 60 apresenta o painel de configuração do componente *Switch*, onde será possível verificar as condições de proximidades do sensor ultrassônico. O teste é feito se existe um obstáculo em uma distância inferior a 30 cm.

Figura 60 – Painel de configuração do componente *Switch*

Fonte: Autoria própria (2016).

Caso haja um obstáculo a menos de 30 cm do sensor ultrassônico, a parte superior do *Switch* da Figura 59 será executada. O componente *Move* presente nele tem as características de realizar o giro do NXT para a esquerda (Figura 61).

Figura 61 – Painel de configuração do componente *Move*

Fonte: Autoria própria (2016).

Em conjunto com o movimento da esquerda, um áudio (Figura 62) e uma imagem (Figura 63) são apresentados para o usuário.

Figura 62 – Painel de configuração do componente *Sound*

Fonte: Autoria própria (2016).

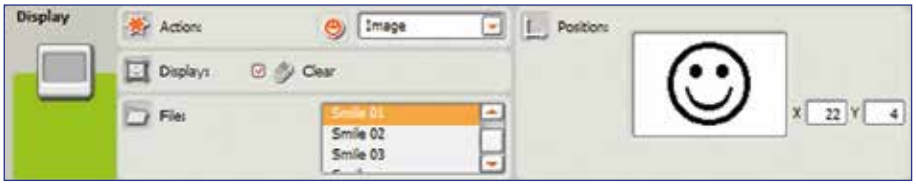


Figura 63 – Painel de configuração do componente *Display*

Fonte: Autoria própria (2016).

Não havendo um obstáculo a menos de 30 cm, a condição de falso do *Switch* é executada (parte inferior do *Switch* da Figura 59), o qual possui um componente *Move* que se movimenta para frente (Figura 64), apresentando nesta situação também uma imagem para o usuário (Figura 65).

Figura 64 – Painel de configuração do componente *Move*

Fonte: Autoria própria (2016).

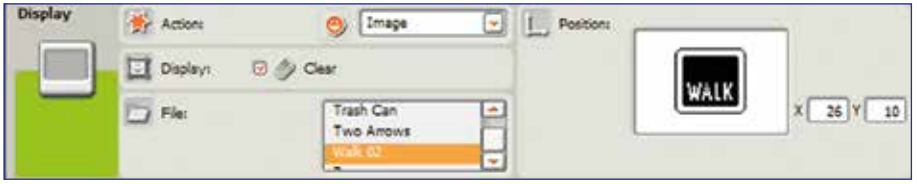


Figura 65 – Painel de configuração do componente *Display*

Fonte: Autoria própria (2016).

4.2.5 *Keep Distance*

Este programa faz o NXT seguir outro objeto, procurando manter sempre uma distância do objeto a sua frente, assim, se o objeto se afastar, o NXT segue em frente, se o objeto se aproximar, o NXT recua.

Como não é possível (ou prático) definir uma distância específica (50 cm), pois existe variação nos dados lidos, e também, pela inércia do movimento, ao ler a distância de 50 cm, quando o NXT para, esta distância já é maior ou menor, deve-se trabalhar com um intervalo tolerável de repouso. Para o exemplo, o NXT ficará em repouso caso a distância seja maior do que 50 cm e menor do que 60 cm.

Para o programa, será utilizado o sensor ultrassônico para medir a distância do obstáculo, o componente *Move*, para fazer a movimentação para frente e para trás, o *Loop* fazendo com que o NXT permaneça se movimentando enquanto não encontrar obstáculo, e um componente *Switch*, o qual identificará a distância do obstáculo.

A Figura 66 apresenta o programa desenvolvido.

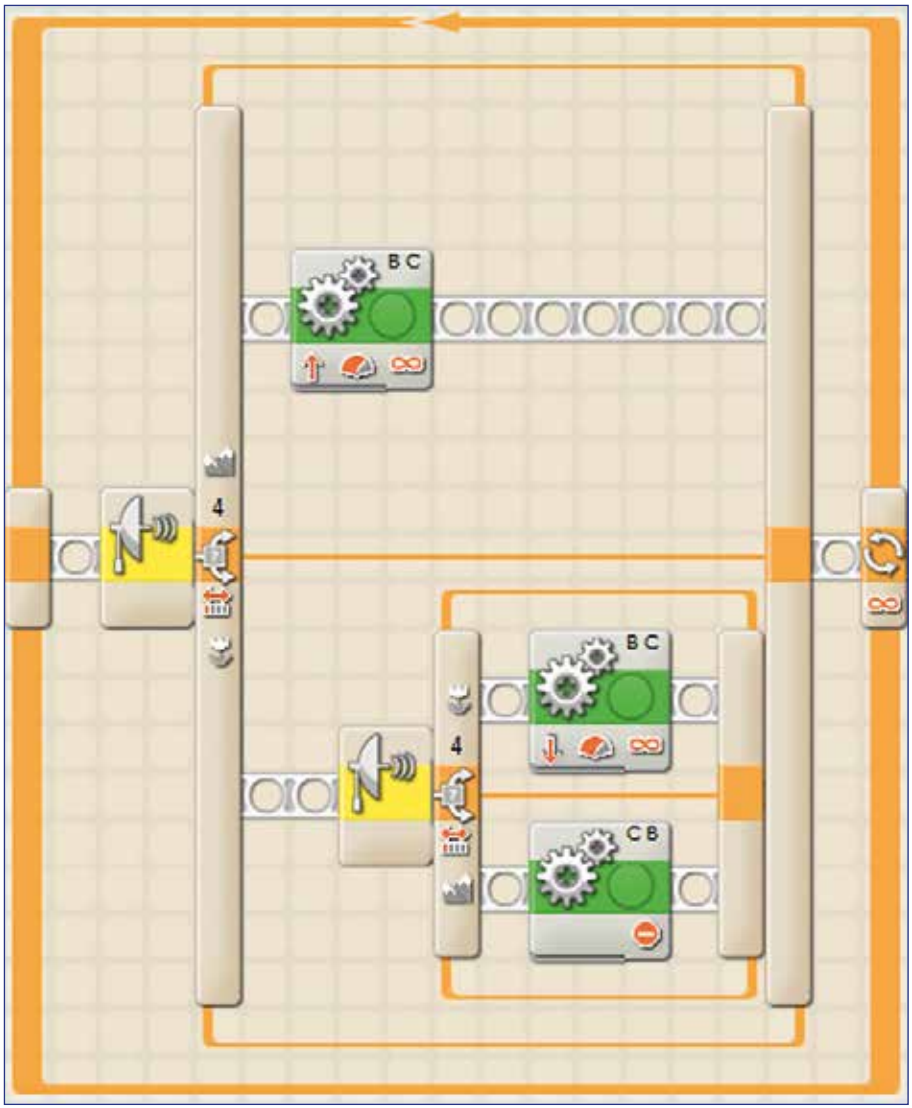


Figura 66 – Representação do aplicativo de *Keep Distance*

Fonte: Autoria própria (2016).

A Figura 67 apresenta o painel de configuração do componente *Switch* externo, o qual verifica se o NXT está a uma distância superior a 60 cm do obstáculo, caso em que o mesmo se movimenta para a frente.

Figura 67 – Painel de configuração do *Switch*

Fonte: Autoria própria (2016).

Não havendo um obstáculo a menos de 60 cm, será executada a parte superior do *Switch* da Figura 66, onde há um componente *Move*, que fará o NXT continuar se movendo para a frente (Figura 68).

Figura 68 – Painel de configuração do componente *Move*

Fonte: Autoria própria (2016).

Havendo um obstáculo a uma distância menor que 60 cm, será executada a parte inferior do *Switch* da Figura 66, onde há um *Switch* interno que verificará se há um objeto a menos de 50 cm do sensor ultrassônico, se positivo, o NXT se movimentará para trás. A Figura 69 apresenta o painel de configuração do *Switch* interno.

Figura 69 – Painel de configuração do *Switch* interno

Fonte: Autoria própria (2016).

A movimentação do NXT para trás é configurada na Figura 70, que define o sentido do movimento para trás.

Figura 70 – Painel de configuração do componente *Move* para trás

Fonte: Autoria própria (2016).

Se a distância não for superior a 60 cm (*switch* externo), nem inferior a 50 cm (*switch* interno), o NXT deve permanecer parado, e esta opção é definida na parte inferior do *switch* interno, conforme configuração do componente *Move* apresentado na Figura 71.

Figura 71 – Configuração do componente *Move* para parado

Fonte: Autoria própria (2016).

4.2.6 *Line Follow*

Este programa faz o NXT seguir uma linha preta. A ideia é que este circuito seja oval, e o NXT se movimentará em sentido horário.

Para o programa, será utilizado o sensor de cor para verificar se a cor detectada é preta, o componente *Move*, para fazer a movimentação necessária para seguir a linha e um *Loop* fazendo com que o NXT permaneça se movimentando eternamente.

A Figura 72 apresenta o programa desenvolvido:

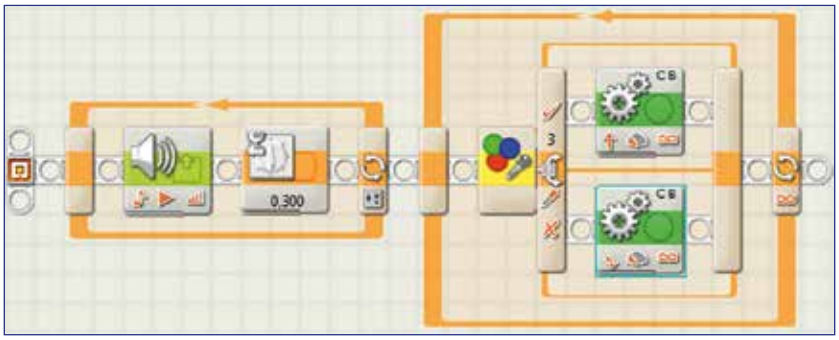


Figura 72 – Representação do aplicativo *Line Follow*

Fonte: Autoria própria (2016).

A esquerda do programa, o primeiro *Loop* representa a execução de três bips, antes do início da lógica do mesmo.

O segundo *Loop* é configurado para executar **eternamente** e o componente de *Switch* interno (Figura 73) é configurado para verificar se a cor detectada é preta, através da propriedade *Compare*.

Figura 73 – Painel de configuração do componente *Switch*

Fonte: Autoria própria (2016).

Sendo preta a cor detectada pelo sensor de cor, o componente *Move* é configurado para andar para frente, como na Figura 74.

Figura 74 – Painel de configuração do componente *Move*

Fonte: Autoria própria (2016).

Caso contrário, o NXT deverá realizar uma curva, a fim de tentar encontrar novamente a linha preta. Para que o NXT realize uma curva, o componente *Move* é configurado de acordo com a Figura 75.

Figura 75 – Painel de configuração do componente *Move*

Fonte: Autoria própria (2016).

4.3 DEMAIS COMPONENTES DA PALETA PADRÃO

Outros dois componentes também recebem destaque na paleta de componentes padrão, sendo eles o componente *Record/Play* e *Wait*, apresentados com detalhe na sequência.

4.3.1 Comando *Record/Play*

Estes comandos são utilizados para gravar uma sequência de ações que os motores realizam, lembrando que os servo motores, além de serem acionados, também geram eventos, como informações referentes a sua movimentação.

Quando adicionado a tela o componente *Record/Play*, a opção *Record* (gravar), pode ser selecionada, conforme demonstrado na Figura 76. Este componente permite armazenar a movimentação do NXT por determinado tempo.

Figura 76 – Painel de configuração do componente *Record/Play*

Fonte: Autoria própria (2016).

As propriedades do componente são:

1. *Name*: nome da sequência de movimentos a ser gravada.
2. *Recording*: motores que terão seus movimentos gravados.
3. *Time*: quanto tempo irá durar a gravação da sequência de movimentos.

Escolhida a opção *Play* (reproduzir), aparece a tela demonstrada na Figura 77, onde se poderá escolher em uma lista a ação que se deseja executar.

Figura 77 – Painel de configuração do componente *Record/Play*

Fonte: Autoria própria (2016).

No campo *Name* deve-se informar o nome da ação gravada para ser reproduzida e, caso haja arquivos já salvos no NXT, aparecerá, no campo *File*, em ordem alfabética, uma lista que permitirá selecionar o arquivo que se deseja executar.

Ressalte-se que, se o usuário gravar e reproduzir uma ação em um mesmo programa, deve lembrar-se de digitar o mesmo nome de arquivo em ambos os blocos (o nome do arquivo não irá aparecer na lista de ações previamente salvas).

Para um exemplo de programa que utiliza este componente verifique Apêndice B.

4.3.2 Comando *Wait*

O componente *Wait* insere uma espera no programa, condicionando a continuidade da programação a um tempo ou a um evento de algum sensor.

Sendo selecionada a opção *Time* no campo *Control* no painel de configurações deste componente, aparece a janela apresentada na Figura 78.

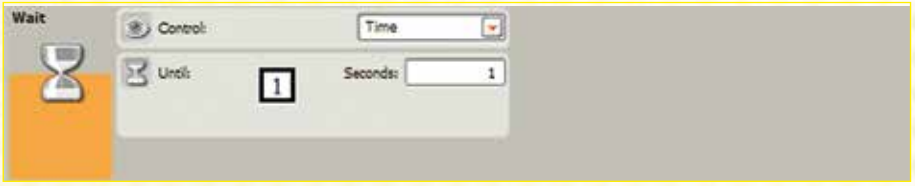


Figura 78 – Painel de configuração do componente *Wait* (*Time*)

Fonte: Autoria própria (2016).

1. *Until*: determina o tempo, em segundos e décimos de segundos, necessário para a execução de uma próxima ação (tanto a vírgula quanto o ponto são aceitos para a separação das casas decimais).

Quando a opção Sensor é selecionada no campo *Control* no painel de configurações deste componente, é possível selecionar o tipo de sensor cuja ação o componente irá esperar. A configuração destes sensores é algo parecido com o apresentado na seção 4.1.5 componente *Switch*.

PALETA COMPLETA [*COMPLETE PALLETE*]

Esta paleta traz funções específicas plataforma Mindstorms NXT, conforme se vê no Quadro 6.

Componente	Nome	Descrição
	*Common*	Reúne as funções básicas do software. Os componentes aqui presentes são os mesmos da paleta *Common*.
	*Action*	Ações a serem realizadas pelo programa. Refere-se a saída de dados/ações.
	*Sensor*	Funções e leituras dos dados obtidos pelos sensores.
	*Flow*	Funções de fluxo do programa.
	*Data*	Funções referentes ao tipo de dados e o tratamento destes.
	*Advanced*	Funções avançadas para o tratamento de dados.

Quadro 6 – Especificação da paleta completa

Fonte: Autoria própria (2016).

O Quadro 6 apresenta os componentes da paleta *Common* (seção 4.1) e componentes específicos de entrada de dados (componente *Sensor*), saída de dados (componente *Action*), componentes de lógica de programação (componente *Flow*), tratamento de dados (componente *Data*) e algumas funções mais avançadas (componente *Advanced*).

Ao longo deste capítulo, serão apresentadas em detalhes as opções de cada componente, com exceção do *Common* (já apresentado na seção 4.1).

A grande diferença entre os componentes da paleta *Common* dos componentes da paleta *Complete* é que, na *Common*, por exemplo, os componentes realizavam mais de uma função, a exemplo do componente *Switch*, que capturava o resultado de um sensor e tomava a decisão com base no valor recuperado, isso com apenas um componente. Já na paleta *Complete*, os componentes possuem funções específicas, o que permite uma maior liberdade ao programador, entretanto, com componentes mais específicos, é normal que um programa possua mais componentes, assim como ligação entre eles.

Desta forma, é muito comum nesta paleta a utilização de *Data Hub*, que nada mais é do que a troca de informações entre componentes.

Para exemplos de aplicativos utilizando os componentes da paleta completa consulte o Apêndice B.

5.1 *DATA HUB*

Ao utilizar qualquer componente do NXT-G, clicando na parte inferior deste, o usuário poderá recuperar ou setar informações específicas do componente. Isso acontece através de linhas (Figura 79).

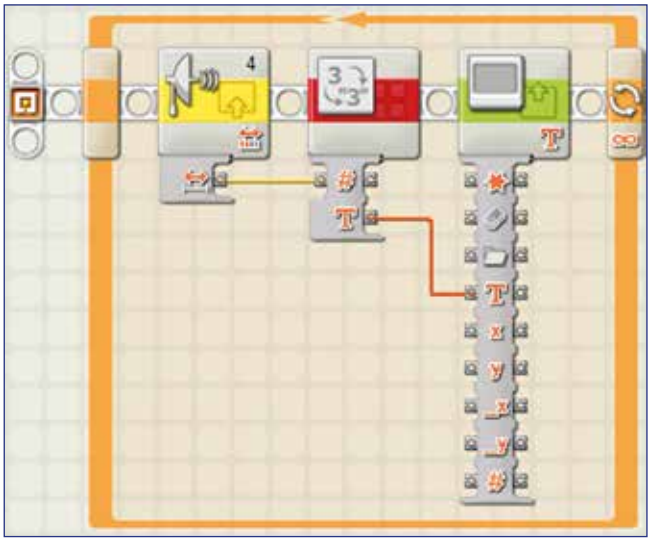


Figura 79 – Representação do *Data Hub*

Fonte: Autoria própria (2016).

Como se observa na figura, dentro de um componente de *Loop* é utilizado um componente do sensor ultrassônico (*Complete Palette*, bloco *Sensor*, componente *Ultrasonic Sensor*), um componente de conversão de informação (*Complete Palette*, bloco *Advanced*, componente *Number to Text*) e um componente *display* (*Complete Palette*, bloco *Action*, componente *Display*).

Isoladamente, tais componentes não possuem funções significativas no programa, porém, ligados pelo *Data Hub*, é possível recuperar o valor inteiro referente à leitura do sensor de distância, convertê-lo em texto e apresentá-lo no *display* do dispositivo.

Assim como nas linguagens de programação C, Pascal e Java, por exemplo, no ambiente NXT-G os valores possuem tipos específicos, como inteiro, texto, booleano, entre outros, e um local que está preparado para receber um texto não pode receber inteiro, nem vice-versa, assim, o componente de conversão de tipo é utilizado em frequência em muitas situações.

Basicamente, os componentes de entrada (sensores) recuperam informações, via *Data Hub*, e alimentam componentes de processamento, presentes na paleta *Flow*, ou componentes de tratamento e comparação de informações, como os componentes da paleta *Data* e *Advanced*. Por fim, estes dados processados são apresentados para o usuário, ou levam o robô a realizar alguma atividade específica. Isso é possível com os componentes de *Action*.

Graças ao *Data Hub*, é possível alimentar o robô com informações dinâmicas, recuperadas em tempo e execução, permitindo mudar a direção do NXT e sua velocidade de movimentação, entre outros.

Com relação aos *Data Hubs*, as saídas de dados são os plugues da direita, enquanto a entrada de dados são os plugues da esquerda (Figura 79).

Outra característica dos *Data Hubs* são as cores das linhas, que representam o tipo de informação que é trocada entre os componentes (Quadro 7).

Formato do dado	Cor da linha de conexão
Numérico	Amarelo
Texto	Laranja
Lógico	Verde
Incompatível	Cinza

Quadro 7 – Cores das linhas que representam o tipo de dado do *Data Hub*

Fonte: Help LEGO Mindstorms (2016).

Por fim, cada ícone presente nos *Data Hubs* também representa um tipo de informação.

5.1.1 Ícones do *Data Hub* da Categoria *Common*

Os ícones de *Data Hub* presentes nos blocos *Common* (Capítulo 4) vêm a seguir, apresentando a representação visual do componente e seu *Data Hub*, assim como uma tabela detalhando cada ligação.

5.1.1.1 *Data Hub* componente *Move*

A Figura 80 apresenta a representação gráfica dos *Data Hubs* do componente e, o Quadro 8, os detalhes de cada propriedade.

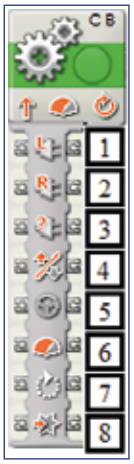


Figura 80 – Representação do *Data Hub* do componente *Move*

Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Left Motor* (motor esquerdo)	Numérico	1 – 3	1 = A, 2 = B, 3 = C
2	*Right Motor* (motor direito)	Numérico	1 – 3	1 = A, 2 = B, 3 = C
3	*Other Motor* (outro motor)	Numérico	1 – 3	1 = A, 2 = B, 3 = C
4	*Direction* (sentido)	Lógico	*True/False*	*True* = Para frente *False* = Para trás

	Plug	Tipo de dado	Variação	Especificação
5	*Steering* (direção)	Numérico	-100 – 100	< 0 = Vira motor para esquerda > 0 = Vira motor para direita
6	*Power* (potência)	Numérico	0 – 100	
7	*Duration* (duração)	Numérico	0 – 2147483647	Depende do tipo de duração: *Degrees/ Rotation*/Seconds. Obs: Este *plug* é desativado no modo de duração *Unlimited*.
8	*Next Action* (próxima ação)	Lógico	*True/False*	*True = Break* *False = Coast*

Quadro 8 – Especificação das funções do *Data Hub* do componente *Move*

Fonte: Help LEGO Mindstorms (2016).

5.1.1.2 *Data Hub* componente *Record/Play*

A Figura 81 apresenta a representação gráfica dos *Data Hubs* do componente e, o Quadro 9, os detalhes de cada propriedade.

Figura 81 – Representação do *Data Hub* do componente *Record/Play*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Action*	Numérico	0 – 1	0 = *Record* (gravar) 1 = *Play* (executar)
2	*Filename*	Texto	Máximo 15 caracteres	Nome do arquivo para gravar e executar
3	*Record A*	Lógico	*True/False*	*True* = Gravar ações do motor A *False* = Não gravar ações do motor A
4	*Record B*	Lógico	*True/False*	*True* = Gravar ações do motor B *False* = Não gravar ações do motor B
5	*Record C*	Lógico	*True/False*	*True* = Gravar ações do motor C *False* = Não gravar ações do motor C
6	*Samples per Second*	Numérico	0 – 255	Determina com que frequência os motores serão amostrados durante a gravação
7	*Total Time*	Numérico	0 – 2147483647	Tempo gravado em milissegundos

Quadro 9 – Especificação das funções do *Data Hub* do componente *Record/Play*
Fonte: Help LEGO Mindstorms (2016).

5.1.1.3 *Data Hub* componente *Sound*

A Figura 82 apresenta a representação gráfica dos *Data Hubs* do componente e, o Quadro 10, os detalhes de cada propriedade.

Figura 82 – Representação do *Data Hub* do componente *Sound*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Action*	Numérico	0 – 1	0 = Arquivo de som 1 = Tom musical
2	*Filename*	Texto	Máximo 15 caracteres	Nome do arquivo para executar
3	*Tone frequency*	Numérico	0 – 65535	Frequência do tom em Hertz. Obs: O alto-falante interno dos NXTs pode reproduzir frequências entre aproximadamente 264 e 4000 Hertz
4	*Control*	Numérico	0 – 1	0 = Executa 1 = Para
5	*Volume*	Numérico	0 – 100	Volume do som. Obs: Há cinco níveis de volume (0, 25, 50, 75 e 100), números entre estes valores são arredondados
6	*Duration*	Numérico	0 – 65535	Duração do tom musical em milissegundos

Quadro 10 – Especificação das funções do *Data Hub* do componente *Sound*

Fonte: Help LEGO Mindstorms (2016).

5.1.1.4 *Data Hub* componente *Display*

A Figura 83 apresenta a representação gráfica dos *Data Hubs* do componente e, o Quadro 11, os detalhes de cada propriedade.

Figura 83 – Representação do *Data Hub* do componente *Display*

Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Action*	Numérico	0 – 5	0 = Imagem 1 = Texto 2 = Ponto 3 = Linha 4 = Círculo 5 = Restaura a tela
2	*Clear*	Lógico	*True/False*	*True* = Limpa a tela *False* = Não limpa a tela
3	*Filename*	Texto	Máximo 15 caracteres	Nome do arquivo de imagem
4	*Text*	Texto		Texto
5	*X*	Numérico	0 – 99	Coordenada X
6	*Y*	Numérico	0 – 63	Coordenada Y
7	*End Point X*	Numérico	0 – 99	Fim da coordenada X Obs: apenas em linha

	Plug	Tipo de dado	Variação	Especificação
8	*End Point Y*	Numérico	0 – 63	Fim da coordenada Y Obs: apenas em linha
9	*Radius*	Numérico	0 – 120	Raio Obs: apenas em circulo

Quadro 11 – Especificação das funções do *Data Hub* do componente *Display*
Fonte: Help LEGO Mindstorms (2016).

Agora que foi visto o conceito de *Data Hub*, será apresentado na seção 5.1.2 o desenvolvimento de um aplicativo simples que faz uso de alguns componentes da paleta *Complete* e dos *Data Hubs*.

5.1.2 Exemplo de um Aplicativo com *Data Hub*

Será apresentado, para fins didáticos, um exemplo de programa que faz uso dos componentes da paleta *Complete*, bem como de *Data Hubs*.

Este programa faz o NXT se movimentar aleatoriamente, parando ao encontrar um obstáculo. Para realizar este programa, utilizam-se dois geradores de números randômicos (*Random*), que alimentam dinamicamente a potência de dois motores (neste exemplo, é usado um motor para cada roda – lado direito e esquerdo).

Para minimizar o efeito da inércia, será utilizado um *timer* entre um sorteio e outro, permitindo que o motor se mova na potência sorteada por um pequeno intervalo de tempo, antes do próximo sorteio.

Para o programa, serão utilizados dois componentes *Motor*, representando cada roda do NXT, e um componente *Loop*, que faz o NXT continuar se movimentando enquanto não encontrar obstáculo (Figura 84).

Figura 84 – Representação do aplicativo de *Random Move*
Fonte: Autoria própria (2016).

À esquerda, o primeiro *Loop* realiza a execução de três bips.

À direita, um *Loop* **eterno** com a lógica do aplicativo, onde dois componentes *Random* alimentam a propriedade *power* do *Motor* via *Data Hub*. Não houve mudança nas propriedades do componente *Random*, que varia de 0 a 100 por padrão.

A Figura 85 demonstra o painel de configuração do primeiro componente de *Motor,* sendo valorizada a propriedade *Port* com B.

Figura 85 – Painel de configuração do primeiro componente *Move*

Fonte: Autoria própria (2016).

A única diferença entre o primeiro componente de *Motor* e o segundo é que, neste último, a propriedade *Port* é alterada para C, que representa o outro motor.

Por fim, um componente *Wait* é inserido, fazendo com que cada motor execute seu movimento por 0,2 segundos, esperando assim um novo número aleatório para a potência. As propriedades do componente *Wait* são apresentadas na Figura 86.

Figura 86 – Painel de configuração do primeiro componente *Wait*

Fonte: Autoria própria (2016).

5.2 *ACTION*

É um componente usado para a execução de tarefas, ou seja, agrupa os comandos de saída de dados do programa. Blocos do *Action* podem ser usados para exibir uma mensagem no *display* do NXT, aguardar um evento, um tempo ou informações obtidas através dos sensores e bloco para acionar motores. O *Action* possui os blocos *Move*, *Sound* e *Display* (capítulo 4) e também *Send Message* e *Lamp*, apresentados na sequência.

5.2.1 Enviar Mensagem (*Send Message)*

Este bloco serve para enviar mensagens para outros NXTs por meio da comunicação *Bluetooth*. Para utilizar este tipo de comunicação, é necessário que, no ambiente Mindstorms dos NXTs, estes tenham sido pareados (um exemplo de aplicativo com comunicação *Bluetooth* é apresentado na seção 5.6).

A Figura 87 apresenta o painel de configurações deste comando.

Figura 87 – Painel de configuração do componente *Send Message*

Fonte: Autoria própria (2016).

Dentre suas propriedades, destacam-se:

1. *Connection*: indica o número de pareamento, realizado no próprio LEGO Mindstorms, para o qual a mensagem será enviada.
2. *Message*: permite escolher o tipo de mensagem (*Text*, *Number* ou *Logic*).
3. Campo para informar a mensagem que será enviada. Esta informação deve ser compatível com o tipo de Mensagem, que é definida no item anterior.
4. *Mailbox*: permite escolher qual o número da caixa de correio que será a mensagem de destino. Como o dispositivo pode receber mais de uma mensagem ao mesmo tempo, apenas a mensagem do *Mailbox* escolhido será processada.

A Figura 88 apresenta a representação gráfica dos *Data Hubs* do componente, e no Quadro 12 os detalhes de cada propriedade.

Figura 88 – Representação do *Data Hub* do componente *Send Message*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Connection*	Numérico	0 – 3	Número de conexões para enviar mensagens
2	*Mailbox*	Numérico	1 – 10	Caixa de entrada para as mensagens
3	*Text*	Texto	Máximo 58 caracteres	Mensagem para enviar
4	*Number*	Numérico	-2147483648 – 2147483647	Mensagem para enviar
5	*Logic*	Lógico	*True/False*	Mensagem para enviar

Quadro 12 – Especificação das funções do *Data Hub* do componente *Send Message*

Fonte: Help LEGO Mindstorms (2016).

5.2.2 Lâmpada (*Lamp)*

Este componente é utilizado para controlar a lâmpada do sensor de cor que acompanha o NXT, desta forma, o sensor se comportará como um componente de saída de dados (uma lanterna).

A Figura 89 demonstra o painel de configurações deste comando:

Figura 89 – Painel de configuração do componente *Color Lamp*

Fonte: Autoria própria (2016).

1. *Port*: deve-se selecionar a porta em que o sensor de cor está conectado ao NXT.
2. *Action*: permite escolher se a lâmpada do sensor de cor será ligada ou desligada.
3. *Color*: permite selecionar a cor da luz que será emitida pelo sensor.

A Figura 90 apresenta a representação gráfica dos *Data Hubs* do componente, e no Quadro 13 os detalhes de cada propriedade.

Figura 90 – Representação do *Data Hub* do componente *Lamp*
Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Port*	Numérico	1 – 4	1 = Porta 1 2 = Porta 2 3 = Porta 3 4 = Porta 4
2	*Action*	Lógico	*On/Off*	*True = On* *False = Off*
3	*Lamp Color*	Numérico	0 – 2	0 = Vermelho 1 = Verde 2 = Azul

Quadro 13 – Especificação das funções do *Data Hub* do componente *Lamp*
Fonte: Help LEGO Mindstorms (2016).

5.3 SENSOR

Este componente é considerado o componente para entrada de dados dentro do LEGO Mindstorms. Em geral, a entrada de dados se dá a partir de sensores, razão do nome do componente (Quadro 14).

Ícone	Nome	Descrição
	*Touch Sensor*	Este bloco verifica a condição de u m sensor de toque em um ponto específico no programa. Pode enviar sua constatação como um sinal lógico (*True/False*). Acionado o sensor, o bloco envia um sinal de **verdadeiro** e, se não tiver sido acionado, o bloco envia um sinal de **falso**.
	*Sound Sensor*	Esse bloco é um detector de som. Pode enviar o valor do nível do som capturado (intensidade de 0 a 100, sendo 0 sem som e 100 som muito alto).
	*Light Sensor*	Este sensor detecta a luz ambiente. Pode enviar o valor de luz corrente e um sinal lógico (*True/ False*), conforme o valor de luz atual se situe acima ou abaixo do valor predeterminado.
	*Color Sensor*	Este bloco possui dois modos: um para a detecção de cores diferentes e o outro para medir a intensidade da luz.
	*Ultrasonic Sensor*	Este bloco pode detectar objetos a uma distância máxima de cerca de 50 polegadas (127 cm). Pode enviar a leitura da distância atual.
	*NXT Buttons*	Este bloco emite um sinal de **verdadeiro** quando um dos botões NXT é ativado. É preciso selecionar o botão que irá enviar este sinal.
	*Rotation Sensor*	Este bloco conta o número de graus (uma volta completa = 360 graus) ou rotações completas, do servo motor presente no LEGO Mindstorms. Pode enviar o número atual de graus ou rotações.
	*Timer*	Este bloco permite controlar o tempo. Quando o programa começa, os três *timers* embutidos no NXT começam automaticamente uma contagem.
	*Receive Message*	Componente utilizado para receber mensagens através da comunicação *Bluetooth*.

Quadro 14 – Especificação do componente *Sensor*

Fonte: Autoria própria (2016).

A explicação de todos estes sensores consta na seção 2.2, onde se fala do componente *Switch*, de modo que se optou por apresentar, neste ponto, apenas os *Data Hubs* destes componentes.

A seguir, as representações gráficas dos *Data Hubs* dos componentes de sensor, assim como os quadros com suas características detalhadas.

5.3.1 *Touch Sensor*

A Figura 91 apresenta a representação gráfica dos *Data Hubs* do componente *Touch Sensor*, e no Quadro 15 os detalhes de cada propriedade.

Figura 91 – Representação do *Data Hub* do componente *Touch Sensor*
Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Port*	Numérico	1 – 4	1 = Porta 1 2 = Porta 2 3 = Porta 3 4 = Porta 4
2	*Action*	Numérico	0 – 2	0 = *Pressed* 1 = *Released* 2 = *Bumped*
3	*Yes/No*	Lógico	*True/False*	Resultado da comparação
4	*Raw Value*	Numérico	0 – 1024	Valor lido do sensor
5	*Logical Number*	Numérico	0 – 1	0 = *Released* 1 = *Pressed*

Quadro 15 – Especificação das funções do *Data Hub* do componente *Touch Sensor*
Fonte: Help LEGO Mindstorms (2016).

5.3.2 *Sound Sensor*

A Figura 92 apresenta a representação gráfica dos *Data Hubs* do componente *Sound Sensor*, e no Quadro 16 os detalhes de cada propriedade.

Figura 92 – Representação do *Data Hub* do componente *Sound Sensor*

Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Port*	Numérico	1 – 4	1 = Porta 1 2 = Porta 2 3 = Porta 3 4 = Porta 4
2	*Trigger Point*	Numérico	0 – 100	Valor para comparar
3	*Greater/Less*	Lógico	True/False	*True* = *Greater* *False* = *Less*
4	*dBA*	Lógico	*True/False*	*True* = Modo dBA *False* = Modo dB
5	*Yes/No*	Lógico	*True/False*	Resultado da comparação
6	*Sound Level*	Numérico	0 – 100	Valor lido do sensor
7	*Raw Value*	Numérico	0 – 1024	Valor não dimensionado lido do sensor

Quadro 16 – Especificação das funções do *Data Hub* do componente *Sound Sensor*

Fonte: Help LEGO Mindstorms (2016).

5.3.3 *Ligth Sensor*

A Figura 93 apresenta a representação gráfica dos *Data Hubs* do componente *Light Sensor*, e no Quadro 17 os detalhes de cada propriedade.

Figura 93 – Representação do *Data Hub* do componente *Light Sensor*
Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Port*	Numérico	1 – 4	1 = Porta 1 2 = Porta 2 3 = Porta 3 4 = Porta 4
2	*Trigger Point*	Numérico	0 – 100	Valor para comparar
3	*Greater/Less*	Lógico	*True/False*	*True = Greater* *False = Less*
4	*Generate Light*	Lógico	*True/False*	Determina se o Led do sensor está ligado ou não
5	*Yes/No*	Lógico	*True/False*	Resultado da comparação
6	*Intensity*	Numérico	0 – 100	Valor lido do sensor
7	*Raw Value*	Numérico	0 – 1024	Valor não dimensionado lido do sensor

Quadro 17 – Especificação das funções do *Data Hub* do componente *Light Sensor*
Fonte: Help LEGO Mindstorms (2016).

5.3.4 *Color Sensor*

A Figura 94 apresenta a representação gráfica dos *Data Hubs* do componente *Color Sensor*, e no Quadro 18 os detalhes de cada propriedade.

Figura 94 – Representação do *Data Hub* do componente *Color Sensor*

Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Port*	Numérico	1 – 4	1 = Porta 1 2 = Porta 2 3 = Porta 3 4 = Porta 4
2	*Range*	Lógico	*True/False*	*True* = Dentro da faixa *False* = Fora da faixa
3	*Color Range A*	Numérico	0 – 6	0 = Próximo do preto 1 = Entre preto e azul 2 = Entre azul e verde 3 = Entre verde e amarelo 4 = Entre amarelo e vermelho 5 = Entre vermelho e branco 6 = Próximo do branco

	Plug	Tipo de dado	Variação	Especificação
4	*Color Range B*	Numérico	0 – 6	0 = Próximo do preto 1 = Entre preto e azul 2 = Entre azul e verde 3 = Entre verde e amarelo 4 = Entre amarelo e vermelho 5 = Entre vermelho e branco 6 = Próximo do branco
5	*Greater/Less*	Lógico	*True/False*	*True = Greater* *False = Less*
6	*Trigger Point*	Numérico	*0 – 100*	Valor para comparar
7	*Generate Light*	Lógico	*True/False*	Determina se o Led do sensor está ligado ou não
8	*Lamp Color*	Numérico	*0 – 2*	0 = *Red* (vermelho) 1 = *Green* (verde) 2 = *Blue* (Azul)
9	*Yes/No*	Lógico	*True/False*	Resultado da comparação
10	*Detected Color*	Numérico	1 – 6	1 = *Black* (preto) 2 = *Blue* (azul) 3 = *Green* (verde) 4 = *Yellow* (amarelo) 5 = *Red* (vermelho) 6 = *White* (branco)

Quadro 18 – Especificação das funções do *Data Hub* do componente *Color Sensor*

Fonte: Help LEGO Mindstorms (2016).

5.3.5 *Ultrasonic Sensor*

A Figura 95 apresenta a representação gráfica dos *Data Hubs* do componente *Ultrasonic Sensor*, e no Quadro 19 os detalhes de cada propriedade.

Figura 95 – Representação do *Data Hub* do componente *Ultrasonic Sensor*
Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Port*	Numérico	1 – 4	1 = Porta 1 2 = Porta 2 3 = Porta 3 4 = Porta 4
2	*Trigger Point*	Numérico	0 – 255 (cm) 0 – 100 (in)	Valor para comparar
3	*Greater/Less*	Lógico	*True/False*	*True = Greater* *False = Less*
4	*Yes/No*	Lógico	*True/False*	Resultado da comparação
5	*Distance*	Numérico	0 – 255 (cm) 0 – 100 (in)	Valor lido do sensor

Quadro 19 – Especificação das funções do *Data Hub* do componente *Ultrasonic Sensor*
Fonte: Help LEGO Mindstorms (2016).

5.3.6 NXT *Buttons*

A Figura 96 apresenta a representação gráfica dos *Data Hubs* do componente *NXT Buttons*, e no Quadro 20 os detalhes de cada propriedade.

Figura 96 – Representação do *Data Hub* do componente *NXT Buttons*
Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Button*	Numérico	1 – 3	1 = *Right* (direito) 2 = *Left* (esquerdo) 3 = *Select* (principal)
2	*Action*	Numérico	0 – 2	0 = Pressed 1 = Released 2 = Bumped
3	*Yes/No*	Lógico	*True/False*	Resultado da comparação

Quadro 20 – Especificação das funções do *Data Hub* do componente *NXT Buttons*
Fonte: Help LEGO Mindstorms (2016).

5.3.7 *Rotation Sensor*

A Figura 97 apresenta a representação gráfica dos *Data Hubs* do componente *Rotation Sensor*, e no Quadro 21 os detalhes de cada propriedade.

Figura 97 – Representação do *Data Hub* do componente *Rotation Sensor*

Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Port*	Numérico	1 – 3	1 = A 2 = B 3 = C
2	*Trigger Point*	Numérico	0 – 2147483647	Valor para comparar
3	*Trigger Point Direction*	Lógico	*True/False*	*True* = Para frente *False* = Para trás
4	*Greater/Less*	Lógico	*True/False*	*True = Greater* *False = Less*
5	*Reset*	Lógico	*True/False*	*True = Reset* *False = Read*
6	*Yes/No*	Lógico	*True/False*	Resultado da comparação
7	*Direction*	Lógico	*True/False*	*True* = Para frente *False* = Para trás
8	*Degrees*	Numérico	0 – 2147483647	Valor lido do sensor

Quadro 21 – Especificação das funções do *Data Hub* do componente *Rotation Sensor*

Fonte: Help LEGO Mindstorms (2016).

5.3.8 Componente *Timer*

A Figura 98 apresenta a representação gráfica dos *Data Hubs* do componente *Timer*, e no Quadro 22 os detalhes de cada propriedade.

Figura 98 – Representação do *Data Hub* do componente *Timer*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Timer*	Numérico	1 – 3	1 = Timer 1 2 = Timer 2 3 = Timer 3
2	*Trigger Point*	Numérico	0 – 100	Valor para comparar
3	*Greater/Less*	Lógico	*True/False*	*True = Greater* *False = Less*
4	*Reset*	Lógico	*True/False*	*True = Reset Timer* *False = Read Timer*
5	*Yes/No*	Lógico	*True/False*	Resultado da comparação
6	*Timer Value*	Numérico	0-4294967296	Valor do Timer em milissegundos

Quadro 22 – Especificação das funções do *Data Hub* do componente *Timer*

Fonte: Help LEGO Mindstorms (2016).

5.3.9 *Receive Message*

A Figura 99 apresenta a representação gráfica dos *Data Hubs* do componente *Receive Message*, e no Quadro 23 os detalhes de cada propriedade.

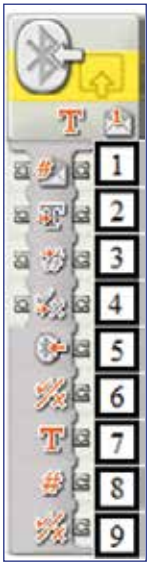


Figura 99 – Representação do *Data Hub* do componente *Receive Message*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Mailbox*	Numérico	1 – 10	Ler *mailbox*
2	*Text in*	Texto	Máximo 58 caracteres	Valor para comparar
3	*Number in*	Numérico	-2147483648 – 2147483647	Valor para comparar
4	*Logic in*	Lógico	*True/False*	Valor para comparar
5	*Message Received*	Lógico	*True/False*	*True* = Mensagem recebida
6	*Yes/No*	Lógico	*True/False*	Resultado da comparação
7	*Text out*	Texto	Máximo 58 caracteres	Dado da mensagem
8	*Number out*	Numérico	-2147483648 – 2147483647	Dado da mensagem
9	*Logic out*	Lógico	True/False	Dado da mensagem

Quadro 23 – Especificação das funções do *Data Hub* do componente *Receive Message*

Fonte: Help LEGO Mindstorms (2016).

5.4 *FLOW*

Este componente realiza as funções referentes ao fluxo do programa, definindo neste os comandos da lógica de programação, tais como: repetições, espera, tomadas de decisões e paradas. Possui os blocos de *Wait*, *Loop*, *Switch*, já apresentados, além do item *Stop* apresentado no Quadro 24.

Ícone	Nome	Descrição
	Stop	Este bloco para o programa e qualquer execução dos motores, lâmpadas ou sons. Este componente não possui propriedades.

Quadro 24 – Especificação do componente *Stop*
Fonte: Autoria própria (2016).

5.4.1 *Stop*

A Figura 100 apresenta a representação gráfica dos *Data Hubs* do componente *Receive Message*, e no Quadro 25 os detalhes de cada propriedade.

Figura 100 – Representação do *Data Hub* do componente *Stop*
Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Stop*	Lógico	*True/False*	*True* = abortar o programa *False* = não abortar o programa

Quadro 25 – Especificação das funções do *Data Hub* do componente *Stop*
Fonte: Help LEGO Mindstorms (2016).

5.5 *DATA*

Realiza o tratamento dos dados obtidos pelo programa. Está neste componente o tratamento das operações lógicas, matemáticos, relacionais, assim como o tratamento de variáveis. O Quadro 26 apresenta os blocos de programação que pertencem a este componente.

Ícone	Nome	Descrição
	*Logic*	Este bloco executa operações lógicas.
	*Math*	Este bloco executa operações aritméticas simples como adição, subtração, multiplicação e divisão, e também pode ser utilizado para operações matemáticas mais complexas, como por exemplo raiz quadrada.
	*Compare*	Este bloco permite executar operações lógicas, comparando valores com os operadores maior que (>), menor que (<) e operador de igualdade (=).
	*Range*	Este bloco permite as operações de intervalo, verificando se um dado valor está dentro ou fora de um intervalo.
	*Random*	Este bloco gera um número aleatório.
	*Variable*	Permite criar variáveis. Uma variável é um local de memória utilizado para armazenar um valor temporariamente.
	Constant	Permite a criação de constantes. Uma constante é como um lugar para armazenar um valor estático na memória do NXT, ou seja, este valor não é alterado ao longo do programa.

Quadro 26 – Especificação dos componentes *Data*
Fonte: Autoria própria (2016).

Para maiores informações sobre operações lógicas, verificar Apêndice A.

5.5.1 Componente *Logic*

Com base nos dados que são recuperados via *Data Hub*, o componente *Logic* permite realizar operações lógicas. Basicamente, as entradas deste componente devem ser dois valores lógicos, assim como sua saída será também um valor lógico.

A Figura 101 demonstra o painel de configuração deste componente.

Figura 101 – Painel de configuração do componente *Logic*

Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Operation*: define o tipo de operação lógica: *And*, *Or*, *Xor* ou *Not*.
2. Define um valor booleano estático para a entrada de dados; quando se utiliza este valor, não é necessário ligar dados em seu respectivo *Data Hub*.

A Figura 102 apresenta a representação gráfica dos *Data Hubs* do componente *Logic*, e no Quadro 27 os detalhes de cada propriedade.

Figura 102 – Representação do *Data Hub* do componente *Logic*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*A*	Lógico	*True/False*	Operando esquerdo
2	*B*	Lógico	*True/False*	Operando direito
3	*Result*	Lógico	True/False	Resultado da operação

Quadro 27 – Especificação das funções do *Data Hub* do componente *Logic*

Fonte: Help LEGO Mindstorms (2016).

5.5.2 Componente *Math*

Este componente possui duas entradas numéricas, valores com os quais ele permite a execução de operações matemáticas, resultando como saída uma terceira informação numérica.

A Figura 103 demonstra o painel de configuração deste componente.

Figura 103 – Painel de configuração do componente *Math*

Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Operation*: define o tipo de operação, que pode ser *Addition* (adição), *Subtraction* (subtração), *Multiplication* (multiplicação), *Division* (divisão), *Absolute Value* (valor absoluto) ou *Square Root* (raiz quadrada), sendo que para os dois últimos, só uma entrada será permitida.
2. Campos para definir os valores de entrada para a operação estabelecida. Estes campos podem receber dados dinamicamente, via *Data Hub*, ou estaticamente, valorizando a propriedade desejada.

A Figura 104 apresenta a representação gráfica dos *Data Hubs* do componente *Math*, e no Quadro 28 os detalhes de cada propriedade.

Figura 104 – Representação do *Data Hub* do componente *Math*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*A*	Numérico	-2147483648 – 2147483647	Operando esquerdo
2	*B*	Numérico	-2147483648 – 2147483647	Operando direito
3	*Result*	Numérico	-2147483648 – 2147483647	Resultado da operação

Quadro 28 – Especificação das funções do *Data Hub* do componente *Math*

Fonte: Help LEGO Mindstorms (2016).

5.5.3 Componente *Compare*

Este componente recebe, como entrada de dados, duas informações numéricas, realizando nestas operações relacionais. A saída deste componente será uma informação booleana.

A Figura 105 demonstra o painel de configuração deste componente.

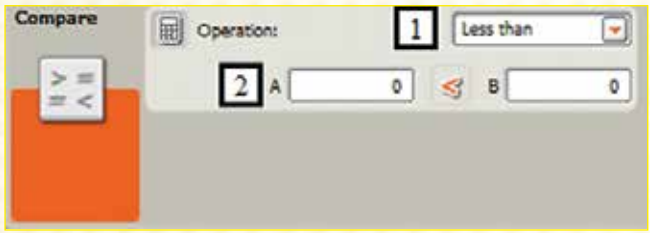


Figura 105 – Painel de configuração do componente *Compare*

Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Operation*: define o tipo de comparação: *Less than* (menor que **<**), *Greater than* (maior que **>**) ou *Equals* (igual **=**).
2. Campos para definir os valores de entrada para ser realizada a comparação. Os dados podem ser estáticos, digitados na janela de propriedades, ou dinâmicos, vindo de *Data Hubs*.

A Figura 106 apresenta a representação gráfica dos *Data Hubs* do componente *Compare*, e no Quadro 29 os detalhes de cada propriedade.

Figura 106 – Representação do *Data Hub* do componente *Compare*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*A*	Numérico	-2147483648 – 2147483647	Operando esquerdo
2	*B*	Numérico	-2147483648 – 2147483647	Operando direito
3	*Result*	Lógico	*True/False*	Resultado da operação

Quadro 29 – Especificação das funções do *Data Hub* do componente *Compare*

Fonte: Help LEGO Mindstorms (2016).

5.5.4 Componente *Range*

Este componente permite trabalhar com intervalores, verificando se uma entrada numérica está dentro ou fora de um intervalo, cujos limites inferior e superior também podem ser definidos dinamicamente, via *Data Hub*. A saída deste, por sua vez, é uma variável booleana.

A Figura 107 demonstra o painel de configuração deste componente.

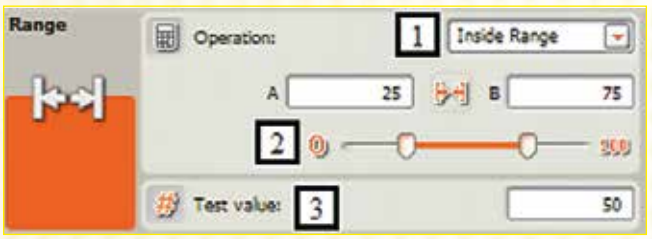


Figura 107 – Painel de configuração do componente *Range*

Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Operation*: define se os valores informados nos campos A e B estão dentro (*Inside*) ou fora (*Outside*) do intervalo determinado.
2. Pode ser usado para definir o início e o fim do intervalo. Este valor pode ser estático (digitado pelo programador) ou dinâmico (recebido via *Data Hub*).
3. *Test value*: valor que será testado, assim como a propriedade anterior, pode ser estático ou dinâmico.

A Figura 108 apresenta a representação gráfica dos *Data Hubs* do componente *Range*, e no Quadro 30 os detalhes de cada propriedade.

Figura 108 – Representação do *Data Hub* do componente *Range*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*A*	Numérico	-2147483648 – 2147483647	Limite inferior
2	*B*	Numérico	-2147483648 – 2147483647	Limite superior
3	*Test Value*	Numérico	-2147483648 – 2147483647	Valor para testar (dentro ou fora da faixa especificada)
4	*Yes/No*	Lógico	*True/False*	Resultado do teste

Quadro 30 – Especificação das funções do *Data Hub* do componente *Range*

Fonte: Help LEGO Mindstorms (2016).

5.5.5 Componente *Random*

Componente que produz um valor numérico aleatório, podendo este alimentar outros componentes.

A Figura 109 demonstra o painel de configuração deste componente.

Figura 109 – Painel de configuração do componente *Random*

Fonte: Autoria própria (2016).

Range: define o limite mínimo e máximo para sortear um valor. Esta propriedade pode ser definida de forma estática (informada pelo programador) ou dinâmica (valor recuperado via *Data Hub*).

A Figura 110 apresenta a representação gráfica dos *Data Hubs* do componente *Random*, e no Quadro 31 os detalhes de cada propriedade.

Figura 110 – Representação do *Data Hub* do componente *Random*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*A*	Numérico	0 – 32767	Limite inferior
2	*B*	Numérico	0 – 32767	Limite superior
3	*Number*	Numérico	Limite inferior – Limite superior	Valor randômico entre o limite inferior e o limite superior

Quadro 31 – Especificação das funções do *Data Hub* do componente *Random*
Fonte: Help LEGO Mindstorms (2016).

5.5.6 Componente *Variable*

Este componente permite criar um ou mais espaços em memória, chamados de variáveis, que receberão valores (os tipos podem ser numérico, texto ou booleano, por exemplo; entretanto, cada variável só poderá receber um dos tipos de valores).

As variáveis ficarão à disposição do programa enquanto este for executado. As operações possíveis em uma variável são a escrita, onde se altera o valor da variável, e a leitura, onde o valor é recuperado.

A Figura 111 demonstra o painel de configuração deste componente.

Figura 111 – Painel de configuração do componente *Variable*
Fonte: Autoria própria (2016).

Dentre suas propriedades, destacam-se:

1. *List*: define o nome e o tipo da variável: *Logic* (lógica), *Number* (numérica) ou *Text* (texto).
2. *Action*: define as ações sobre a variável, sendo possível ler o valor atual da variável ou sobrescrever novas nela.
3. *Value*: se a opção *Write* for selecionada em *Action*, é possível atribuir um valor numérico, um texto ou ainda, dependendo da variável, uma condição de verdadeiro ou falso de forma estática, não considerando a entrada dinâmica, esta via *Data Hub*.

A Figura 112 apresenta a representação gráfica dos *Data Hubs* do componente *Variable*, e no Quadro 32 os detalhes de cada propriedade.

Figura 112 – Representação do *Data Hub* do componente *Variable*
Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Especificação
1	*Value*	Todo tipo	Valor para ler ou escrever

Quadro 32 – Especificação das funções do *Data Hub* do componente *Variable*
Fonte: Help LEGO Mindstorms (2016).

5.5.7 Componente *Constant*

Constante é um espaço de memória que armazena um valor, assim como uma variável, entretanto, seu conteúdo não pode ser alterado com o tempo.

Para criar uma nova constante ou editar uma já existente, é preciso primeiro defini-la através do menu *Edit,* clicando em *Define Constant*.

A Figura 113 demonstra a janela para criar ou editar uma constante.

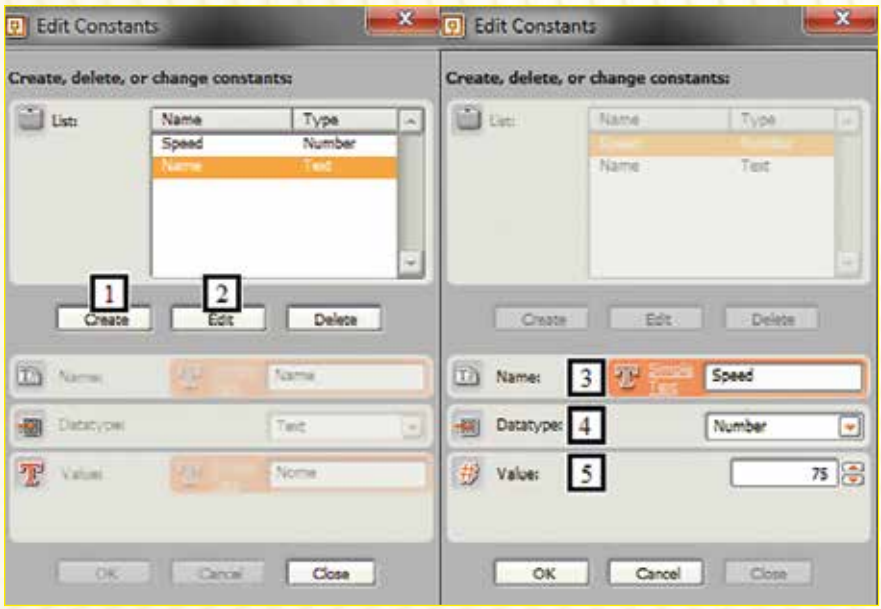


Figura 113 – Janela para a criação ou edição de uma constante

Fonte: Autoria própria (2016).

Dentre suas propriedades, destacam-se:

1. *Create*: para criar uma nova constante.
2. *Edit*: para editar ou excluir uma constante já existente.
3. *Name*: campo para fornecer um nome curto para a constante.
4. *Datatype*: define o tipo de dado da constante (*Logic*, *Number* ou *Text*).
5. *Value*: é possível atribuir um valor numérico, um texto ou ainda, dependendo da variável, uma condição de verdadeiro ou falso.

As Figuras 114 e 115 mostram o painel de configuração deste componente.

Figura 114 – Painel de configuração do componente *Constant* (*Choose from list*)

Fonte: Autoria própria (2016).

1. *Action*: permite escolher uma constante já existente (*Choose from list*) ou criar uma nova (*Custom*).
2. *List*: lista todas as constantes já existentes.

Figura 115 – Painel de configuração do componente *Constant* (*Custom*)

Fonte: Autoria própria (2016).

3. *Data type*: define o tipo de dado da constante (*Logic*, *Number* ou *Text*).
4. *Value*: é possível atribuir um valor numérico, um texto ou ainda, dependendo da variável, uma condição de verdadeiro ou falso.
5. *Name*: campo para fornecer um nome curto para a constante.

Este componente é um dos poucos do LEGO Mindstorms que não possui *Data Hubs* para valorização dinâmica de valores.

5.6 *ADVANCED*

Os componentes avançados permitem aumentar as possiblidades de um programa na plataforma Mindstorms. O Quadro 33 apresenta os blocos deste componente.

Ícone	Nome	Descrição
	*Number to Text*	Tem a função de converter um número em texto.
	*Text*	Pode concatenar até três sequências de texto.
	*Keep Alive*	Mantém o NXT em modo de espera, seja para efetuar uma ação ou recuperar dados sem consumir excessivamente a bateria.
	*File Access*	Possibilita salvar os dados fisicamente no NXT, no formato de arquivo. Estes dados podem ser recuperados posteriormente.
	*Calibrate*	Permite calibrar os sensores de luz e de som com os valores do ambiente, permitindo uma melhor interação do aplicativo.
	*Reset Motor*	Os servo motores interativos têm um mecanismo de correção automática de erros que ajuda o NXT a se mover com muita precisão. Este bloco pode reajustar o movimento dos motores.
	*Bluetooth Connection*	Permite conectar o NXT a outro dispositivo *Bluetooth*, como outros NXT, telefones celulares ou computadores.

Quadro 33 – Especificação dos componentes *Advanced*

Fonte: Autoria própria (2016).

5.6.1 *Number to Text*

Este componente irá apanhar um número (lido de um sensor, por exemplo) e transformá-lo em texto, que poderá ser mostrado no *display* do NXT. A Figura 116 demonstra o painel de configurações do componente.

Figura 116 – Painel de configuração do componente *Number to Text*

Fonte: Autoria própria (2016).

Sua propriedade significa: o número de entrada pode ser digitado ou fornecido dinamicamente utilizando *data hub.*

A Figura 117 apresenta a representação gráfica dos *Data Hubs* do componente *Number to Text*, e no Quadro 34 os detalhes de cada propriedade.

Figura 117 – Representação do *Data Hub* do componente *Number to Text*

Fonte: Autoria própria (2016).

	Plug	**Tipo de dado**	**Variação**	**Especificação**
1	*Number*	Numérico	-2147483648 – 2147483647	Converter número em texto
2	*Text*	Texto	-	Texto representando o número

Quadro 34 – Especificação das funções do *Data Hub* do componente *Number to Text*

Fonte: Help LEGO Mindstorms (2016).

5.6.2 *Text*

Este componente pode adicionar grupos de caracteres e formar um texto (concatenação de caracteres). Os textos podem conter números e até mesmo caracteres especiais. Os textos são importantes porque eles podem ser mostrados no *display* do NXT. A Figura 118 demonstra o painel de configurações do componente.

Figura 118 – Painel de configuração do componente *Text*

Fonte: Autoria própria (2016).

A Figura 119 apresenta a representação gráfica dos *Data Hubs* do componente *Text*, e no Quadro 35 os detalhes de cada propriedade.

Figura 119 – Representação do *Data Hub* do componente *Text*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Especificação
1	*A*	Text	Texto
2	*B*	Text	Texto
3	*C*	Text	Texto
4	*Combined Text*	Text	Combina os textos em A, B e C

Quadro 35 – Especificação das funções do *Data Hub* do componente *Text*

Fonte: Help LEGO Mindstorms (2016).

5.6.3 *Keep Alive*

Este componente tem como função manter o NXT no estado de dormência, ou seja, se um programa requer que o mesmo espere por determinado período de tempo, basta utilizar este bloco. Este componente não tem parâmetros para setar, por isso, seu painel de configuração é vazio, como demonstrado pela Figura 120.

Figura 120 – Painel de configuração do componente *Keep Alive*

Fonte: Autoria própria (2016).

A Figura 121 apresenta a representação gráfica dos *Data Hubs* do componente *Keep Alive*, e no Quadro 36 os detalhes de cada propriedade.

Figura 121 – Representação do *Data Hub* do componente *Keep Alive*

Fonte: Autoria própria (2016).

Plug		Tipo de dado	Variação	Especificação
1	*Time Until Sleep*	Numérico	0-4294967296	Tempo em milissegundos até que o NXT entre em suspensão

Quadro 36 – Especificação das funções do *Data Hub* do componente *Keep Alive*

Fonte: Help LEGO Mindstorms (2016).

5.6.4 *File Access*

Este componente permite a manipulação de arquivo dentro da plataforma Mindstorms. A maioria das suas características podem ser dinâmicas (definidos via *Data Hub*), como nome do arquivo e o conteúdo que será gravado. A Figura 122 demonstra o painel de configurações do componente.

Figura 122 – Painel de configuração do componente *File Access*

Fonte: Autoria própria (2016).

Dentre suas propriedades, destacam-se:

1. *Action*: define se este arquivo será usado para leitura (*Read*), gravação (*Write*), fechamento (*Close*) ou exclusão (*Delete*).
2. *Name*: define o nome do arquivo em que a operação de *Action* afetará.
3. *File*: especifica um arquivo já existente, permitindo sua seleção em um menu.
4. *Type*: campo para escolher se o arquivo armazenará texto ou número.
5. *Text*: permite gravar um valor estático no arquivo texto.

A Figura 123 apresenta a representação gráfica dos *Data Hubs* do componente *File Access*, e no Quadro 37 os detalhes de cada propriedade.

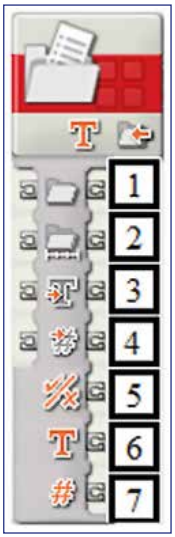


Figura 123 – Representação do *Data Hub* do componente *File Access*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Filename*	Texto	Máximo 15 caracteres	Nome do arquivo
2	*Initial File Size*	Numérico	0 – 32767	Válido somente quando o primeiro arquivo é criado, determinando o tamanho do arquivo.
3	*Text*	Texto		Texto para escrever
4	*Number*	Numérico	-2147483648 – 2147483647	Número para escrever
5	*Error*	Lógico	*True/False*	*True* se ocorrer um erro no bloco, como por exemplo, escrever em um arquivo cheio.
6	*Text out*	Texto		Texto que foi lido
7	*Number out*	Numérico	-2147483648 – 2147483647	Número que foi lido

Quadro 37 – Especificação das funções do *Data Hub* do componente *File Access*
Fonte: Help LEGO Mindstorms (2016).

5.6.5 *Calibrate*

Componente que ajusta alguns sensores às condições do ambiente do local onde o NXT se encontra.

A Figura 124 demonstra o painel de configurações do componente.

Figura 124 – Painel de configuração do componente *Calibrate*
Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Port*: deve-se selecionar a porta em que o sensor está conectado ao NXT.
2. *Sensor*: escolhe o tipo de sensor que será calibrado, podendo ser sensor de luz (*Light Sensor*) ou sensor de som (*Sound Sensor*).
3. *Action*: determina se é para calibrar o sensor com um novo valor ou excluir um valor previamente definido (voltando à configuração padrão).
4. *Value*: determina o valor mínimo e máximo a ser calibrado no sensor.

A Figura 125 apresenta a representação gráfica dos *Data Hubs* do componente, e no Quadro 38 os detalhes de cada propriedade.

Figura 125 – Representação do *Data Hub* do componente *Calibrate*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Port*	Numérico	1 – 4	1 = Porta 1 2 = Porta 2 3 = Porta 3 4 = Porta 4
2	*Max/Min*	Lógico	*True/False*	*True* = Calibrar no valor máximo *False* = Calibrar no valor mínimo
3	*Delete*	Lógico	*True/False*	*True* = Delete *False* = Calibrar

Quadro 38 – Especificação das funções do *Data Hub* do componente *Calibrate*

Fonte: Help LEGO Mindstorms (2016).

5.6.6 *Reset Sensor*

Quando se utiliza o componente *Move*, o NXT faz um ajuste automaticamente entre os motores configurados; no entanto, quando se usa o parâmetro de configuração *Coast*, o mesmo pode perder a precisão nos movimentos, e o componente *Reset Sensor* serve para reajustar a duração do movimento.

A Figura 126 demonstra o painel de configurações do componente.

Figura 126 – Painel de configuração do componente *Reset Motor*

Fonte: Autoria própria (2016).

Seleciona qual porta (em que um motor esteja conectado) será resetada.

A Figura 127 apresenta a representação gráfica dos *Data Hubs* do componente *Reset Motor*, e no Quadro 39 os detalhes de cada propriedade.

Figura 127 – Representação do *Data Hub* do componente *Reset Motor*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Reset A*	Lógico	*True/False*	O motor A será resetado
2	*Reset B*	Lógico	*True/False*	O motor B será resetado
3	*Reset C*	Lógico	*True/False*	O motor C será resetado

Quadro 39 – Especificação das funções do *Data Hub* do componente *Reset Motor*

Fonte: Help LEGO Mindstorms (2016).

5.6.7 *Bluetooth Connection*

Bluetooth é o nome dado a tecnologia de comunicação sem fio que permite a transmissão de dados e arquivos de maneira rápida e segura entre aparelhos de telefone celular, notebooks, câmeras, entre outros equipamentos (CIRIACO, 2008). Esta tecnologia está presente no LEGO Mindstorms. Para usar, basta ativar a opção no NXT *Brick* e utilizar o bloco de programação explicado nesta seção. Este componente permite trabalhar com algumas características da comunicação *Bluetooth*. A Figura 128 demonstra as opções de configuração deste componente.

Figura 128 – Painel de configuração do componente *Bluetooth*

Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Turn on*: ativa o *Bluetooth* do NXT.
2. *Turn off*: desativa o *Bluetooth* do NXT.

A Figura 129 demonstra as configurações quando escolhida a opção para iniciar uma conexão *Bluetooth* (*Initiate Connection*) com outro dispositivo.

Figura 129 – Painel de configuração do componente *Bluetooth* (*Initiate Connection*)

Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Contacts*: quando o NXT está conectado a um dispositivo e ligado, a lista de contatos irá preencher automaticamente com os dispositivos encontrados pelo NXT. Se o nome não estiver na lista de contatos, o nome deste pode ser digitado na opção Connect To (conectar em).
2. *Connection*: define um número de conexão de 1 a 3 (canal 0 é reservado para o mestre do bloco NXT). Esta conexão já deve ser feita anteriormente, no ambiente do próprio Mindstorms.

A Figura 130 demonstra as configurações quando escolhida a opção para encerrar uma conexão *Bluetooth* (*Close Connetion*) com outro dispositivo.

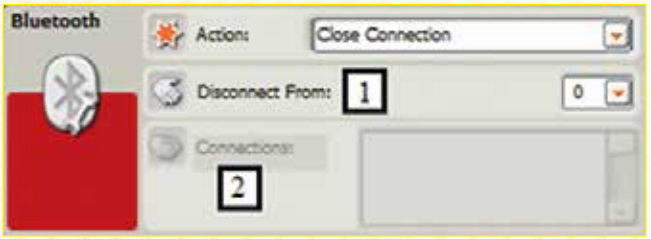


Figura 130 – Painel de configuração do componente *Bluetooth* (*Close Connection*)

Fonte: Autoria própria (2016).

Entre suas propriedades, estão:

1. *Disconnect from:* possibilita escolher o número da conexão para desconectar do dispositivo mestre.
2. *Connections*: possibilita selecionar o nome do dispositivo *Bluetooth* na lista para ser desconectado do dispositivo mestre.

A Figura 131 apresenta a representação gráfica dos *Data Hubs* do componente *Bluetooth Connection*, e no Quadro 40 os detalhes de cada propriedade.

Figura 131 – Representação do *Data Hub* do componente *Bluetooth Connection*

Fonte: Autoria própria (2016).

	Plug	Tipo de dado	Variação	Especificação
1	*Action*	Numérico	0 – 3	0 = Ligar 1 = Desligar 2 = Iniciar conexão 3 = Encerrar conexão
2	*Connect to*	Texto	Todo tipo	Nome do dispositivo *Bluetooth* a conectar
3	*Connection Number*	Numérico	1 – 4	O número da conexão *Bluetooth* quando conectar a outro dispositivo Bluetooth
4	*Disconnect From*	Numérico	0 – 3	O número da conexão *Bluetooth* para encerrar

Quadro 40 – Especificação das funções do *Data Hub* do componente *Bluetooth Connection*

Fonte: Help LEGO Mindstorms (2016).

PALETA PERSONALIZADA (*CUSTOM PALLETE*)

Esta paleta contempla apenas dois componentes, os quais são apresentados no Quadro 41.

Ícone	Nome	Descrição
	*My Blocks*	O *My Block Builder* permite que o usuário use uma série de blocos selecionados na área de trabalho do ambiente de desenvolvimento e agrupe-os em seu próprio *My Block,* funcionando como um ícone personalizado.
	*Web Downloads*	Reúne os blocos que foram baixados da internet.

Quadro 41 – Especificação dos componentes da paleta personalizada
Fonte: Autoria própria (2016).

6.1 *MY BLOCKS*

Para se ter um bloco personalizado, é preciso criá-lo antes. Para criar um bloco são necessários quatro passos:

a) selecionar um número de blocos que irão se tornar um único (Figura 132);

b) com os blocos selecionados, vá ao menu *Edit* e clique em *Make a New Block*. Isso irá abrir o primeiro *My Block Builder*. A tela do assistente é demonstrada na Figura 133;

c) dê ao seu novo *My Block* um nome e escreva uma breve descrição do que ele faz. Por exemplo: **Este bloco executa duas vezes um som e pode ser usado para avisar o usuário que o programa será executado no NXT**. e clique em *Next* para passar para a próxima etapa;

d) por fim, basta escolher um ícone para representar o novo *My Block* e clicar em *Finish*. Com isso o novo bloco aparecerá na área de trabalho do software e poderá ser utilizado na programação.

Figura 132 – Representação para selecionar blocos

Fonte: Autoria própria (2016).

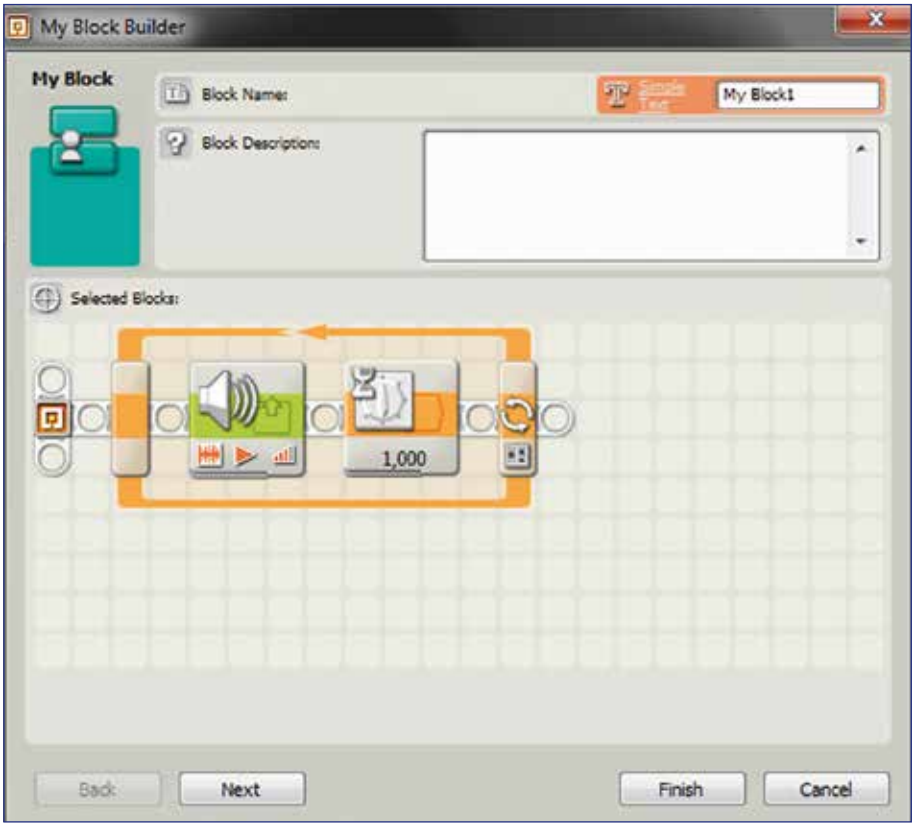


Figura 133 – *My block builder*

Fonte: Autoria própria (2016).

Um exemplo da utilização do novo bloco criado pode ser visualizado na Figura 134, na qual foi acrescentado o mesmo no início do programa *Forward and Backward Forever,* visto na seção 4.2.3.

Figura 134 – Modificação do programa *Forward and Backward Forever* da seção 4.2.3 acrescentando o ícone criado

Fonte: Autoria própria (2016).

leJOS

Um novo modelo de desenvolvimento no ambiente LEGO Mindstorms é utilizando outras linguagens de programação, como por exemplo o Java. A utilização deste tipo de linguagem permite que o desenvolvedor tenha mais opções no desenvolvimento, diminuindo inclusive a complexidade, já que, dependendo do programa, se codificado no ambiente Visual NXT-G, exigiria dezenas de componentes visuais, assim como um grande número de ligação entre eles.

Para o uso da linguagem Java, são necessários o download e a instalação do ambiente leJOS NXJ.

7.1 INSTALAÇÃO DO leJOS

O arquivo leJOS_NXJ_0.9.1beta-3_win32_setup (versão para Windows) pode ser baixado diretamente na URL <http://sourceforge.net/projects/lejos/files/lejos-NXJ/>, entretanto, é interessante conhecer a página do projeto: <www.lejos.org>, o qual possui uma documentação atualizada do leJOS, alguns tutoriais, arquivos para download, entre outras informações.

Ao baixar os binários (versão multiplataforma), basta descompactar a pasta e o ambiente está pronto para ser utilizado. Se optado pela versão Windows, é baixado um executável, que ao ser executado instala o ambiente no computador de desenvolvimento. A Figura 135 apresenta a tela inicial da instalação.

Figura 135 – *Setup* leJOS (início)

Fonte: Autoria própria (2016).

Pressiona-se **Next >** e na tela seguinte deve-se informar onde está instalado o *Java Development Kit* (JDK), que obrigatoriamente precisa ser a versão de 32 bits (Figura 136).

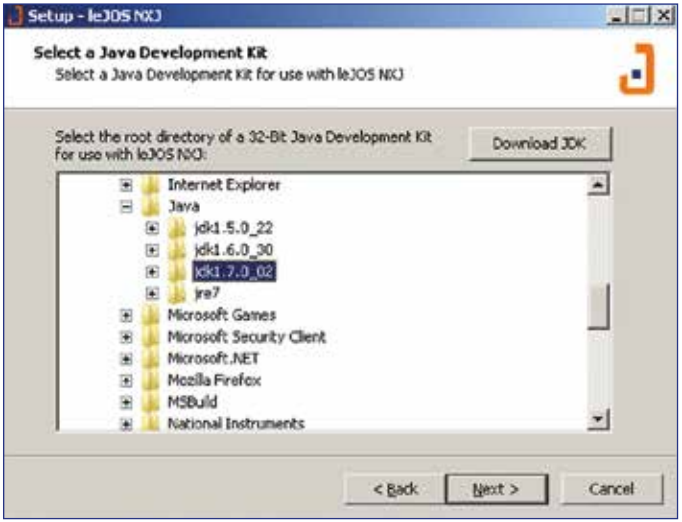


Figura 136 – Setup leJOS (selecionar *Java Development Kit* – JDK)

Fonte: Autoria própria (2016).

Na tela seguinte escolhe-se o local onde será instalado o leJOS (Figura 137).

Figura 137 – *Setup* leJOS (selecionar diretório a ser instalado)

Fonte: Autoria própria (2016).

Segundo, são apresentados alguns componentes, solicitando sua instalação. Para funcionar, é necessário ao menos o leJOS *Development Kit*, podendo ainda ser instalada a documentação e alguns exemplos de código (Figura 138).

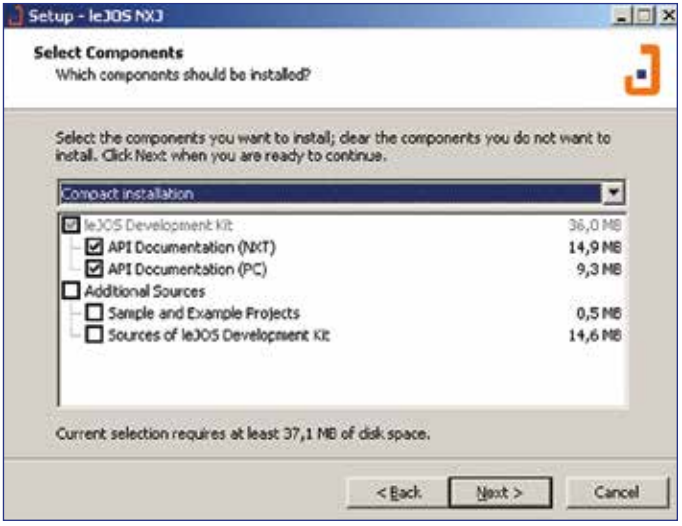


Figura 138 – *Setup* leJOS (selecionar componentes a serem instalados)

Fonte: Autoria própria (2016).

A tela seguinte solicita a pasta para instalação no menu de programas e, enfim, a programação é iniciada (Figura 139).

Figura 139 – *Setup* leJOS (iniciar processo de instalação)

Fonte: Autoria própria (2016).

Ao final, pergunta-se ao usuário deseja iniciar o ambiente de atualização do *firmware*, já que o sistema nativo – o Mindstorms NXT-G será apagado, sendo este substituído pelo leJOS, assim, a opção *Launch NXJ Flash Utility* deve ser marcada, sendo finalizada a janela (Figura 140).

Figura 140 – *Setup* leJOS (instalação completa)

Fonte: Autoria própria (2016).

Antes de clicar no botão *Flash* leJOS *firmware* (Figura 141), é necessário conectar o LEGO Mindstorms, via cabo USB, ao computador.

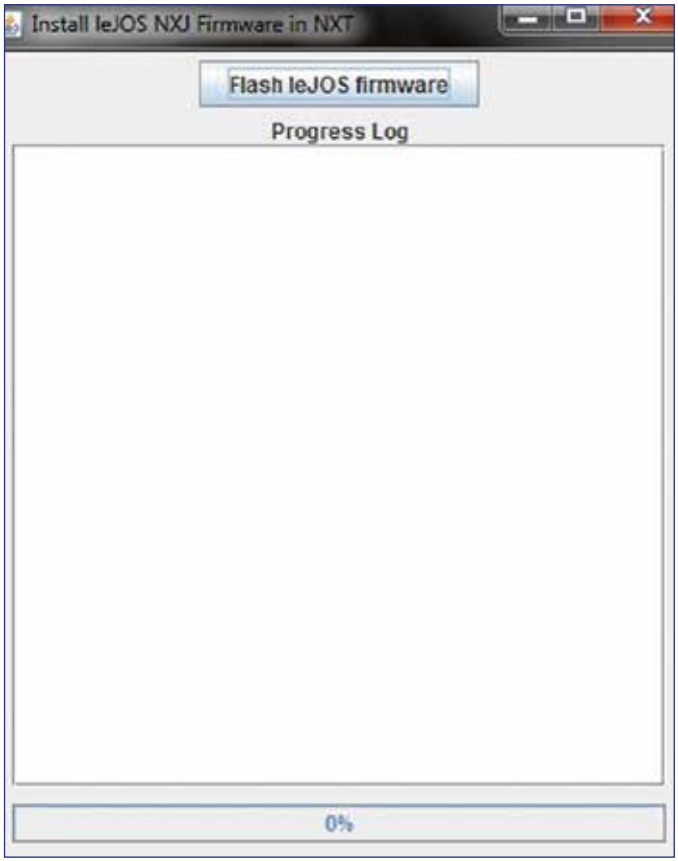


Figura 141 – Programa leJOS *Update* (instalar *Firmware* leJOS NXJ no NXT)

Fonte: Autoria própria (2016).

Deve-se observar que o NXT deve estar ligado. Após clicar no botão, uma mensagem de confirmação é apresentada (Figura 142).

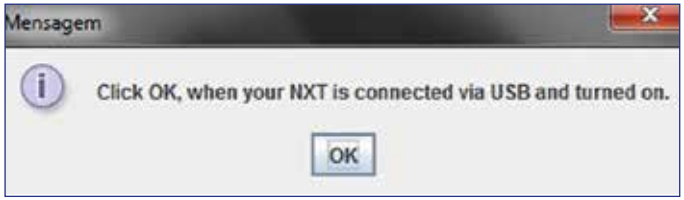


Figura 142 – Mensagem OK leJOS

Fonte: Autoria própria (2016).

Após a confirmação, uma mensagem informa que todos os dados do LEGO Mindstorms serão perdidos (Figura 143). Deve-se confirmar esta tela para a atualização do *firmware* do LEGO Mindstorms.

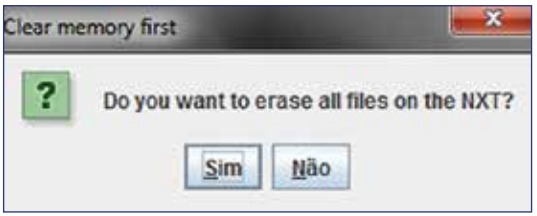


Figura 143 – Mensagem aviso leJOS
Fonte: Autoria própria (2016).

Durante o processo de atualização do *firmware*, um log com o andamento é apresentado na tela (Figura 144). Durante este processo, o LEGO Mindstorms se reiniciará, fazendo alguns bips e deixando a tela sem imagem. Este processo é normal.

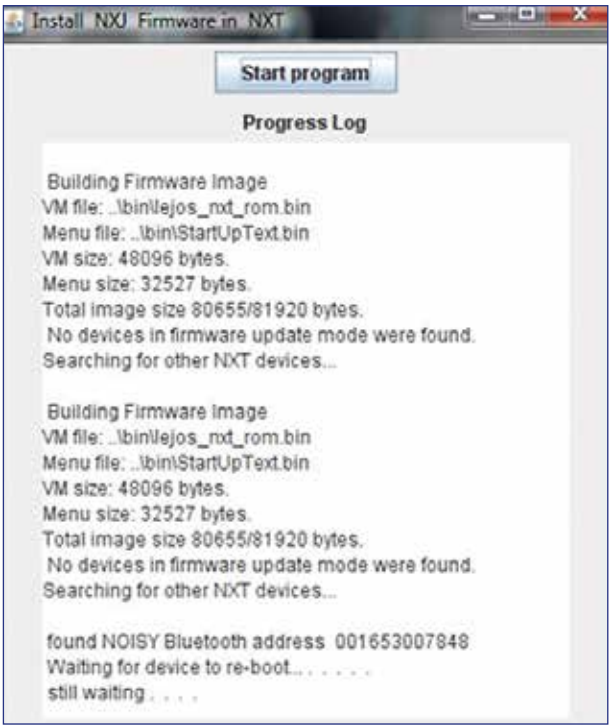


Figura 144 – LEJOS *Update*
Fonte: Autoria própria (2016).

Caso ocorra algum travamento no LEGO Mindstorms durante este processo **–** o que não é normal, mas pode acontecer **–**, o mesmo deve ser resetado, apertando com uma haste o botão, conforme apresentado na Figura 145.

Figura 145 – Resetando o LEGO Mindstorms
Fonte: Kohler (2011).

Após o término da atualização do *firmware*, uma mensagem será apresentada, perguntando ser deseja fazer atualização em outros LEGO Mindstorms, por padrão seleciona-se *No* (Figura 146).

Figura 146 – Mensagem *Updates* leJOS
Fonte: Autoria própria (2016).

Com este processo, o LEGO Mindstorms está pronto para receber os aplicativos desenvolvidos com a API da leJOS, não aceitando mais os tradicionais programas desenvolvidos, mostrados até então, desenvolvidos no ambiente visual do LEGO Mindstorms NXT-G.

7.2 INSTALAÇÃO DO *PLUGIN* leJOS NXJ PARA O ECLIPSE

Inicialmente, é preciso fazer o download da IDE Eclipse através do link <http://www.eclipse.org/downloads/> e instalá-la. Logo na sequência, com a IDE Eclipse aberta, clique na opção Help, na aba de ferramentas e em seguida em *Install New Software*. A Figura 147 demonstra este processo.

Figura 147 – Instalando *plugin* leJOS NXJ na IDE Eclipse (início)

Fonte: Autoria própria (2016).

A próxima ação é representada pela Figura 148, nesta tela, no campo *Work with*, deverá ser digitado o link: <http://www.lejos.org/tools/eclipse/plugin/nxj/> e pressionado *Enter*, após isso, o Eclipse irá verificar o site e apresentar uma lista dos componentes disponíveis para instalação. Para finalizar, basta clicar em *Next*.

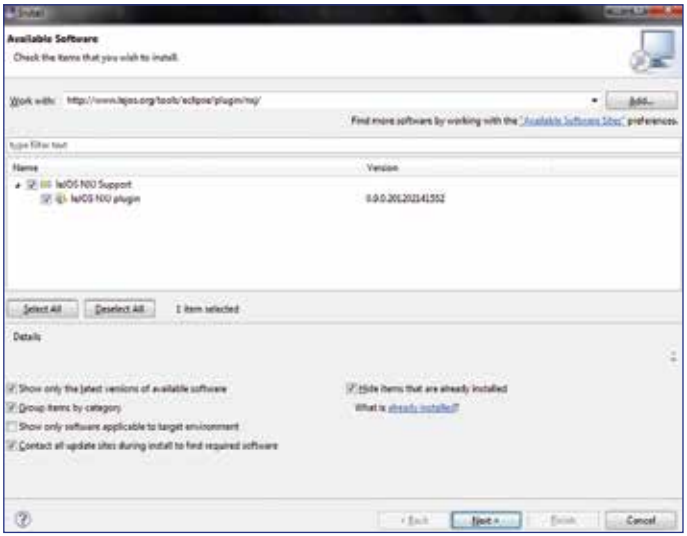


Figura 148 – Instalando *plugin* leJOS NXJ na IDE Eclipse (escolha do componente a ser instalado)

Fonte: Autoria própria (2016).

É preciso concordar com os termos da licença (Figura 149) e finalizar o processo clicando em *Finish*.

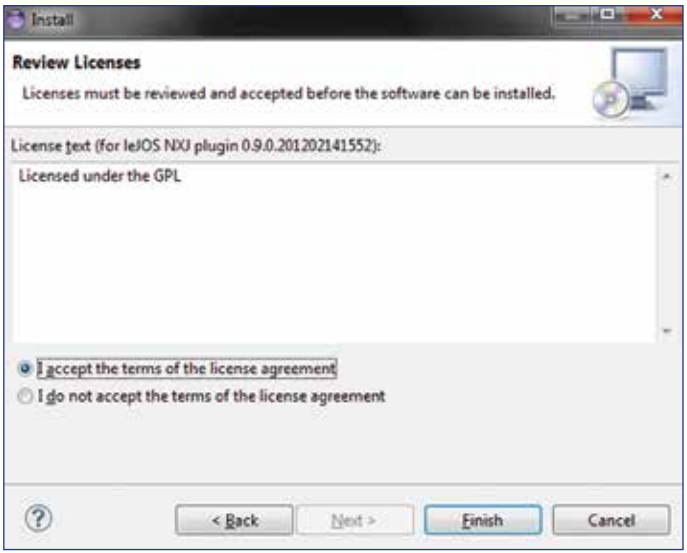


Figura 149 – Instalando *plugin* leJOS NXJ na IDE Eclipse (licença para uso do *plugin*)

Fonte: Autoria própria (2016).

Como o *plugin* do leJOS NXJ não é assinado, um alerta de segurança poderá aparecer, o mesmo é representado pela Figura 150, caso apareça, basta clicar em OK.

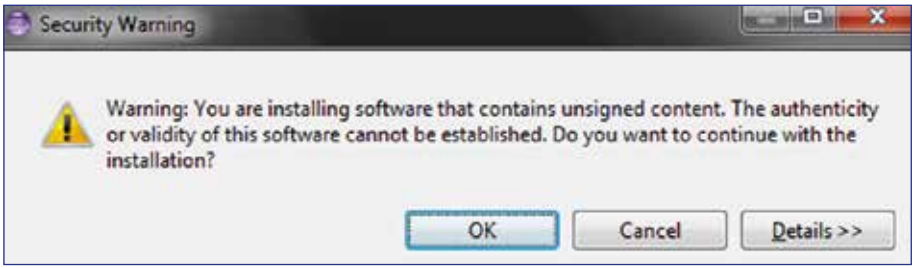


Figura 150 – Instalando *plugin* leJOS NXJ na IDE Eclipse (mensagem de aviso)

Fonte: Autoria própria (2016).

Após esse processo, será solicitado reinicializar o Eclipse (Figura 151), o que deve ser feito clicando em *Yes*.

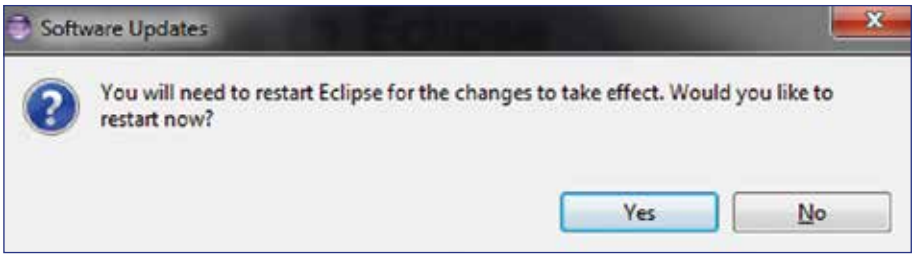


Figura 151 – Instalando *plugin* leJOS NXJ na IDE Eclipse (mensagem para reiniciar o Eclipse)

Fonte: Autoria própria (2016).

O usuário poderá atualizar o leJOS NXJ *firmware* em seu NXT *Brick* através deste *plugin* instalado. Para tanto, basta clicar em leJOS NXJ como demonstrado na Figura 152, e na sequência em *Upload Firmware*.

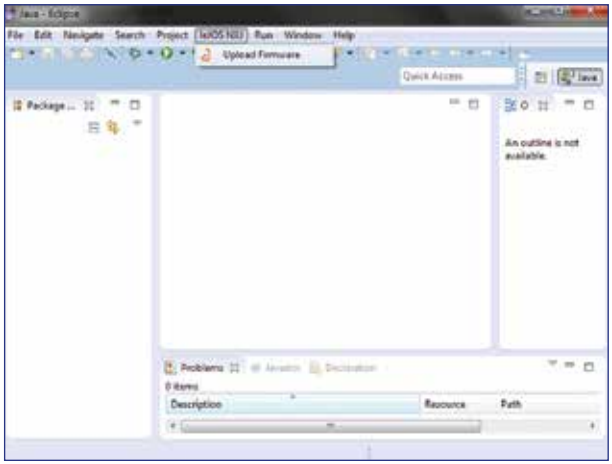


Figura 152 – Instalando *plugin* leJOS NXJ na IDE Eclipse (fazer o *Upload* do *Firmware* pelo Eclipse)

Fonte: Autoria própria (2016).

Para configurar o *plugin* instalado no Eclipse, é preciso clicar no menu *Window* > *Preferences* > *leJOS NXJ* e no campo NXJ_HOME; caso não seja detectado automaticamente, deve-se informar o local onde foi instalado o leJOS NXJ no computador (Figura 153).

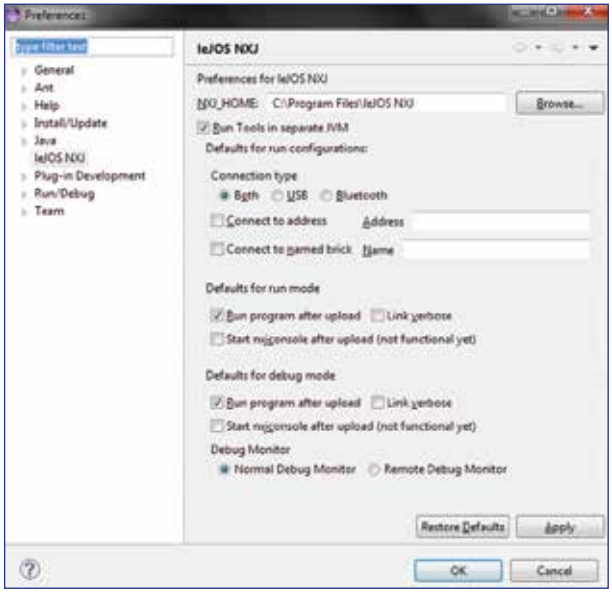


Figura 153 – Instalando *plugin* leJOS NXJ na IDE Eclipse (configurando o *plugin* no Eclipse)

Fonte: Autoria própria (2016).

7.3 CRIANDO UM PROJETO leJOS NO ECLIPSE

Para criar um novo projeto para o leJOS NXJ, basta acessar o menu *File* > *New* > *Other* (Figura 154), e na sequência selecionar a opção leJOS NXT *Project* (Figura 155).

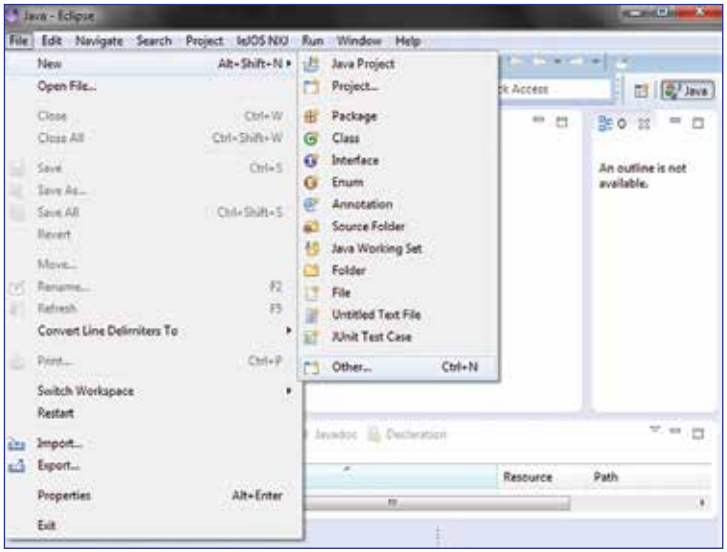


Figura 154 – Criando um projeto leJOS NXJ na IDE Eclipse (criando projeto que usa o *firmware* leJOS NXJ)

Fonte: Autoria própria (2016).

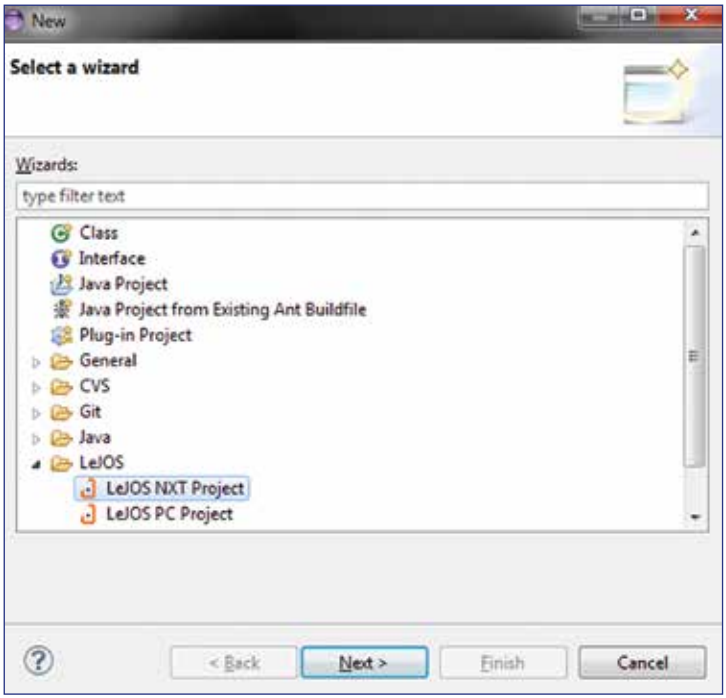


Figura 155 – Criando um projeto leJOS NXJ na IDE Eclipse (selecionar o tipo do projeto a ser criado)

Fonte: Autoria própria (2016).

Após isso, surgirá uma janela (Figura 156) na qual o usuário deverá dar um nome ao seu projeto (no caso, *HelloWorld* leJOS) e clicar em *Finish*.

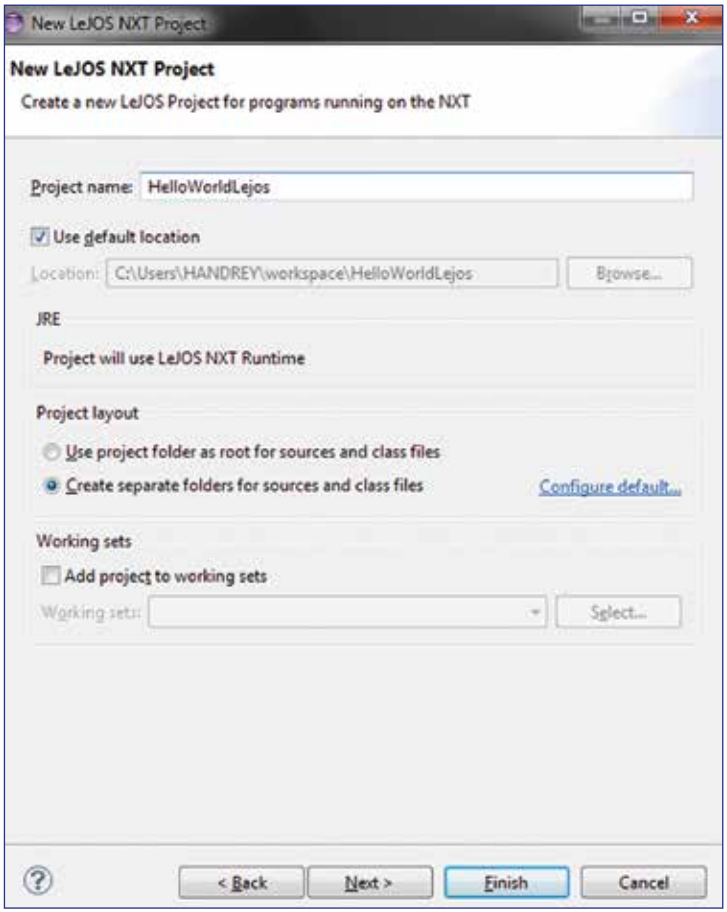


Figura 156 – Criando um projeto leJOS NXJ na IDE Eclipse (novo projeto leJOS NXJ)

Fonte: Autoria própria (2016).

Agora é preciso clicar com o botão direito do mouse na pasta *src* e navegar através de *New* > *Class*, como na Figura 157.

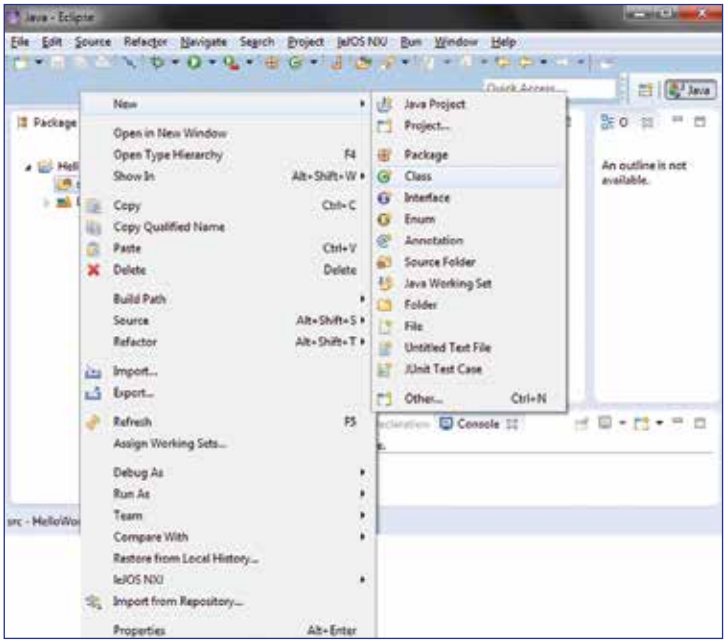


Figura 157 – Criando uma classe Java no projeto leJOS NXJ na IDE Eclipse

Fonte: Autoria própria (2016).

Na criação da Classe Java pode-se seguir as especificações de acordo com a Figura 158. Vale ressaltar que o nome do pacote é arbitrário, bem como o nome do projeto.

Figura 158 – Nova classe no projeto leJOS NXJ na IDE Eclipse

Fonte: Autoria própria (2016).

O código do programa *HelloWorld* segue na Listagem 1.

Listagem 1: Código do aplicativo *HelloWorld*

```
01. package br.edu.utfpr;
02. import lejos.nxt.Button;
03.
04. public class HelloWorld
05. {
06.   public static void main(String[] args)
07.             {
08.           System.out.println("Hello World!");
09.           Button.waitForAnyPress();
10.    }
11. }
```

Este programa mostrará a mensagem *Hello World* no *display* no NXT e esperar que qualquer botão seja pressionado.

7.4 INSTALANDO E EXECUTANDO UM APLICATIVO leJOS NO LEGO MINDSTORMS

Para fazer o upload do código para o NXT *Brick*, basta clicar com o botão direito do mouse no projeto e selecionar a opção *Run As* > leJOS NXT *Program*, como representado pela Figura 159.

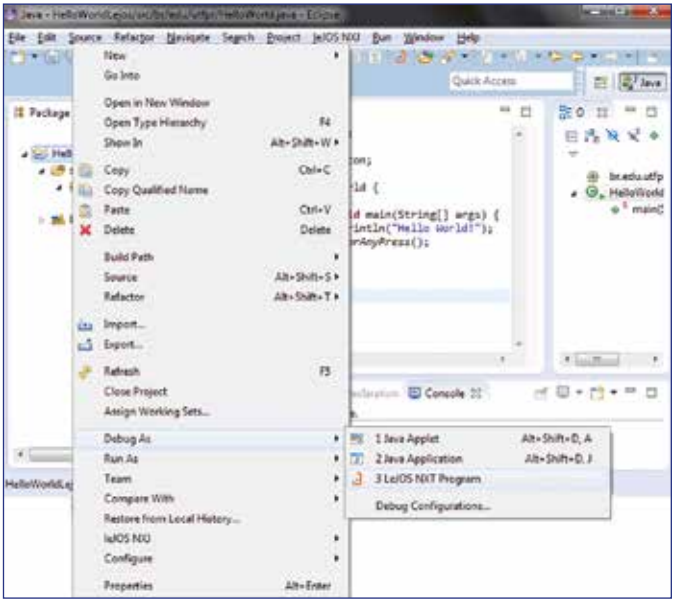


Figura 159 – Fazendo o *upload* do código para o NXT *Brick*

Fonte: Autoria própria (2016).

7.5 CLASSES leJOS

No ambiente de desenvolvimento baseado em Java (leJOS), praticamente todo componente é uma classe, por exemplo, motores e sensores. Desta forma, o modelo de desenvolvimento é um pouco diferente do visto até então. Para usuários leigos, desenvolver em leJOS costuma ser mais difícil do que no ambiente visual NXT-G, visto até então, entretanto, para usuários experientes é muito melhor desenvolver em leJOS.

Com leJOS, o programador tem suporte a recursos como *Multithread*, acesso a bibliotecas nativas, além da facilidade em fazer programas complexos, que no NXT-G ocupariam várias páginas de componentes, além de inúmeras ligações.

A seguir são apresentadas algumas das principais classes do leJOS.

7.5.1 *Class Motor*

Classe utilizada para valorizar a velocidade dos motores através do método *setSpeed* e movimentar através do método *forward*, para mover para frente e *backward* para mover para trás. Dentre os principais métodos desta classe, destacam-se:

a) setSpeed(): define a velocidade de movimentação do motor;

b) setAceleration(): define a aceleração de movimentação do motor;

c) forward(): define que o motor deve rodar para frente;

d) backward(): define que o motor deve rodar para trás.

7.5.2 *Class SensorPort*

Utilizada para identificar a porta em que o sensor em questão será utilizado.

7.5.3 *Class Ultrasonic Sensor*

Classe utilizada para ler os valores obtidos através do sensor ultrassônico através do método *getDistance* que retorna um valor em cm.

Listagem 2: Classe *Ultrasonic Sensor* e *Sensor Port*

```
01. UltrasonicSensor us = new UltrasonicSensor(SensorPort.S4);
02. int distancia = us.getDistance();
```

7.5.4 *Class Color Sensor*

Classe utilizada para ler os valores obtidos através do sensor de cor através do método *getColorID* que retorna um número referente a cor. Dentre os principais métodos desta classe, destacam-se:

a) getColor() – Retorna um *Color Object* lido do sensor;

b) getColorID() – Retorna um inteiro que representa a cor lida pelo sensor.

Listagem 3: *Classe Color Sensor*

```
01. ColorSensor cs = new ColorSensor(SensorPort.S3);
02. int corID = us.getColorID();
```

7.6 EXEMPLOS DE PROGRAMAS USANDO leJOS

Nesta seção serão apresentados alguns exemplos de programas utilizando leJOS.

7.6.1 Andar até o Obstáculo

Este primeiro aplicativo tem um objetivo muito simples: movimentar o veículo Mindstorms NXT para frente, até que este se aproxime de um obstáculo, ao chegar próximo, o veículo para e o processamento é encerrado.

Para a leitura de distância é utilizada a classe *Ultrasonic Sensor*, para o acionamento dos motores a classe Motor. O código completo do aplicativo é apresentado na Listagem 4.

Listagem 4: Código do aplicativo que movimenta o veículo até encontrar obstáculo

```
01.  import lejos.nxt.Motor;
02.  import lejos.nxt.SensorPort;
03.  import lejos.nxt.UltrasonicSensor;
04.
05.
06.
07.  public class AndarAteObstaculo {
08.
09.   public static void main(String[] args) throws InterruptedException{
10.
11.          UltrasonicSensor us = new UltrasonicSensor(SensorPort.S4);
12.
13.          boolean estaPerto = false;
14.
15.          int distancia = 0;
16.
17.          while( estaPerto == false ){
18.                  distancia = us.getDistance();
19.
20.                  Thread.sleep( 200 );
21.
22.                  Motor.B.setSpeed( 500 );
23.                  Motor.C.setSpeed( 500 );
24.
25.                  Motor.B.forward();
26.                  Motor.C.forward();
27.
28.                  if( distancia <= 30 ){
29.                          Motor.B.stop();
30.                          Motor.C.stop();
```

```
31.
32.                             estaPerto = true;
33.                     }
34.             }
35.    }
36.  }
```

Inicialmente, das linhas 01 a 03 são importadas as classes do pacote lejos.nxt, estes responsáveis pela entrada e saída de dados do aplicativo. Na linha 07 acontece a declaração da classe AndarAteObstaculo, assim como a declaração do método *public static void main()* na linha 09.

A linha 11 apresenta um objeto do tipo *Ultrasonic Sensor*, para a leitura dos dados referentes a distância. Por padrão, este sensor é conectado à porta 4 do *Brick*, por este motivo o parâmetro SensorPort.S4.

Para o controle da lógica, duas variáveis auxiliares foram necessárias: a variável booleana *estarPerto* (linha 13) e a variável inteira distância (linha 15). Após, um *Loop* (linha 17) será executado, até que a variável booleana *estaPerto* esteja valorizada com *true*.

Assim, a distância é lida através do sensor ultrassônico (linha 18), sendo realizada uma pausa de 200 milissegundos para não sobrecarregar o processador do LEGO Mindstorms. Na sequência, é atribuída uma velocidade para os motores B e C (linhas 22 e 23), assim como realizado um comando para que estes se movimentem para a frente (linhas 25 e 26).

O teste lógico para o término do processamento acontece na linha 28, onde é verificada se a distância é inferior a 30 cm, caso positivo, os motores B e C são parados (linhas29 e 30), assim como a variável de controle *estaPerto* é valorizada com *true* (linha 33), fazendo com que o programa pare de ser executado.

7.6.2 Andar Aleatoriamente (*Random Move*)

Este aplicativo tem a função de fazer o veículo se movimentar de forma aleatória durante 10 segundos. A estratégia para isso é fazer com que a potência dos motores seja valorizada com valores distintos, ambos aleatórios; assim, será impossível prever a direção do próximo movimento do veículo.

Para o desenvolvimento, o método *Math.random()* foi utilizado, conforme apresentado na Listagem 5.

Listagem 5: Código do aplicativo que movimenta o veículo de forma aleatória

```
01.  import lejos.nxt.Motor;
02.
03.  public class RandomMove {
04.
05.   public static void main(String[] args) throws InterruptedException{
06.
07.          int cont = 0;
08.          double move1 = 0, move2 = 0;
09.
10.          while( cont <= 10 ){
11.                  move1 = Math.random();
12.                  move2 = Math.random();
13.
14.                  Motor.B.setSpeed( (int)(move1*1000) );
15.                  Motor.C.setSpeed( (int)(move2*1000) );
16.
17.                  Motor.B.forward();
18.                  Motor.C.forward();
19.
20.                  Thread.sleep(1000);
21.
22.                  cont++;
23.          }
24.   }
25.  }
```

O programa desenvolvido faz a importação da classe Motor na linha 01, declara a classe *RandomMove* na linha 03 e declara o método *public static void main()* na linha 05.

A lógica do aplicativo começa nas linhas 07 e 08, onde são declaradas três variáveis de controle (*cont* inteiro, move1 e move2 do tipo *double*).

Na linha 10 acontece o *Loop* principal do programa, sendo este limitado a 10 interações (a variável *cont* é incrementada a cada interação, e cada interação dura 1 segundo), fazendo com que o programa seja executado por 10 segundos.

As linhas 11 e 12 são responsáveis pela recuperação de dois números aleatórios, sendo estes valorizados nas velocidades dos motores nas linhas 14 e 15. Por fim, as linhas 17 e 18 acionam os motores para frente, fazendo com que o veículo se movimente. Ao final da interação, é realizada uma pausa no processamento por 1.000 milissegundos, o que equivale a 1 segundo (linha 20), assim como incrementada a variável de controle *cont* (linha 22). Como a linha 20 pode gerar uma exceção, a mesma é lançada na declaração do método *main* (linha 05), com o *throws Interrupted Exception*.

7.6.3 Manter Distância

O aplicativo mantém distância visa procurar manter sempre o NXT a uma mesma distância do obstáculo à sua frente, durante um período de 50 segundos. Esta distância pode variar de 40 a 50 cm. Assim, se o obstáculo se aproximar do veículo, o mesmo se desloca para trás. Se o obstáculo se afastar, o veículo se move para frente. Para o processamento, é utilizado basicamente o sensor ultrassônico, os motores e o comando condicional *if*.

A lógica deste aplicativo é apresentado na Listagem 6.

Listagem 6: Código do aplicativo que mantém o veículo a uma mesma distância do obstáculo

```
01.  import lejos.nxt.Motor;
02.  import lejos.nxt.SensorPort;
03.  import lejos.nxt.UltrasonicSensor;
04.
05.  public class ManterDistancia {
06.
07.   public static void main(String[] args) throws InterruptedException{
08.
09.           UltrasonicSensor us = new UltrasonicSensor(SensorPort.S4);
10.
11.           Motor.B.setSpeed( 400 );
12.           Motor.C.setSpeed( 400 );
13.
14.           for( int i = 0; i <= 100; i++ ){
15.                   distancia = us.getDistance();
16.
17.                   System.out.println(distancia);
18.
19.                   if( distancia >= 50 ){
20.                           Motor.B.forward();
21.                           Motor.C.forward();
22.
23.                   } else if( distancia <= 40 ){
24.                           Motor.B.backward();
25.                           Motor.C.backward();
26.
27.                   } else {
28.                           Motor.B.stop();
29.                           Motor.C.stop();
30.
31.
32.                   Thread.sleep( 500 );
33.           }
34.     }
35.  }
```

Das linhas 01 a 03 acontecem a importação das classes *Motor*, *Sensor Port* e *Ultrasonic Port* utilizadas pelo programa, assim como a declaração da classe *ManterDistancia* (linha 05) e do método *public static void main()* (linha 07).

Antes de entrar no *Loop* principal do aplicativo, é instanciado o sensor de distância (linha 09), assim como valorizada uma velocidade padrão para os motores (linha 11 e 12).

O *Loop*, que acontecerá 100 vezes (linha 14) terá um intervalo de meio segundo entre cada interação (comando *Thread.sleep()* presente na linha 33.

O processamento iniciará com a leitura da distância (linha 15), esta é apresentada no *display* para o usuário (linha 17) e utilizada na comparação da linha 19, a qual verifica se a distância é superior a 50 cm, se isso acontecer, é porque o veículo está muito longe do obstáculo, então nas linhas 20 e 21 o veículo se movimentará para frente. Se a distância for inferior a 40 (linha 23), então o veículo está muito próximo do obstáculo, e deve se movimentar para trás (linhas 24 e 25). Por fim, se a distância for entre 40 e 50, está na distância ideal, e deve permanecer em repouso, fazendo com que os motores parem (linha 28 e 29).

7.6.4 Gravando Arquivo de LOG

Neste exemplo utilizam-se os recursos para manipulação de arquivos dentro da plataforma leJOS. O objetivo do programa é criar um arquivo chamado log.txt. gravando neste arquivo a tabuada do número 2. A cada comando, uma mensagem informativa é apresentada no *display* do dispositivo. O aplicativo completo é apresentado na Listagem 7.

Listagem 7: Código do aplicativo que grava a tabuada do número 2 em um arquivo texto

```
01.  import java.io.DataOutputStream;
02.  import java.io.File;
03.  import java.io.FileOutputStream;
04.  import java.io.IOException;
05.
06.  public class GravaTexto {
07.
08.   public static void main(String[] args) throws InterruptedException, IOException{
09.
10.          System.out.println( "Setando nome do arquivo" );
11.          Thread.sleep( 500 );
12.
13.          File nome = new File( "log.txt" );
14.
```

```
15.             System.out.println( "Criando o arquivo" );
16.             Thread.sleep( 500 );
17.
18.             FileOutputStream fout = new FileOutputStream( nome );
19.             DataOutputStream dout = new DataOutputStream( fout );
20.
21.             System.out.println( "Gravando Arquivo" );
22.             Thread.sleep( 500 );
23.
24.             for( int i = 1; i <= 10; i++ ){
25.                     dout.writeUTF( "2 x " + i + "= " + (2*i) );
26.             }
27.
28.             System.out.println( "Fechando o arquivo" );
29.             Thread.sleep( 500 );
30.
31.             dout.close();
32.             fout.close();
33.
34.     }
35.  }
```

As primeiras linhas são responsáveis pela importação das classes utilizadas (linhas 01 a 04), declaração da classe (linha 08) e declaração do método *public static void main()* (linha 08). Após isso, são apresentadas mensagens informativas com o comando *System.out.println()*, assim como realizada uma pausa de meio segundo após cada mensagem, isso acontece nas linhas 10 e 11.

Na linha 13, um objeto que representará o arquivo é criado, sendo nas linhas 18 e 19 criado fisicamente este arquivo e um *DataOutputStream* para fazer a gravação de dados neste.

Após, na linha 24 é realizado um *Loop*, 10 vezes, para gravação das linhas referentes à tabuada, estas no formato de texto, por esse motivo a utilização do método *writeUTF()* – linha 25.

Por fim o arquivo é fechado (linhas 31 e 32), o que o desbloqueia para a utilização por outros aplicativos, por exemplo.

7.6.5 Servidor e Cliente *Bluetooth*

Este exemplo apresenta a comunicação *Bluetooth* entre diferentes dispositivos LEGO Mindstorms. O exemplo escolhido para tal é o mesmo desenvolvido na seção 5.6, que apresenta a comunicação *Bluetooth* utilizando o ambiente de desenvolvimento visual NXT-G.

O objetivo é basicamente fazer o servidor recuperar a distância lida do sensor ultrassônico, enviando este dado para o aplicativo cliente. O aplicativo cliente recupera esta informação e apresenta na tela. O aplicativo servidor é detalhado na Listagem 8.

Listagem 8: Código do aplicativo servidor para comunicação *Bluetooth*

```
01.  import java.io.DataInputStream;
02.  import java.io.DataOutputStream;
03.  import java.io.IOException;
04.  import lejos.nxt.SensorPort;
05.  import lejos.nxt.Sound;
06.  import lejos.nxt.UltrasonicSensor;
07.  import lejos.nxt.comm.Bluetooth;
08.  import lejos.nxt.comm.NXTConnection;
09.
10.  public class BTServidor {
11.
12.   public static void main(String[] args) throws InterruptedException, IOException {
13.
14.           for( int i = 0; i < 3; i++ ){
15.                   Sound.beep();
16.                   Thread.sleep( 300 );
17.           }
18.
19.           System.out.println( "Aguardando Conexão" );
20.           NXTConnection conexao = Bluetooth.waitForConnection();
21.
22.           System.out.println( "Criando fluxos de I/O" );
23.           DataOutputStream out = conexao.openDataOutputStream();
24.           DataInputStream in = conexao.openDataInputStream();
25.
26.           System.out.println( "Criando um sensor ultrassonico" );
27.           UltrasonicSensor us = new UltrasonicSensor(SensorPort.S4);
28.
29.           System.out.println( "Entrando no Loop" );
30.
31.           while( true ) {
32.
33.                   int dist = us.getDistance();
34.
35.                   System.out.println( "Enviando dados" );
36.                   out.writeInt( dist );
37.                   Thread.sleep( 500 );
38.
39.                   System.out.println( "Aguardando retorno" );
40.
41.                   boolean retorno = in.readBoolean();
42.                   System.out.println( "Retorno: " + retorno );
43.
44.                   Thread.sleep( 500 );
45.           }
46.   }
47.  }
```

O aplicativo servidor, inicialmente, importa as classes necessárias para o desenvolvimento (linhas 01 a 08), assim como declara a classe *BTServidor* (linha 10) e o método principal (linha 12).

O primeiro comando executado pelo aplicativo é a estrutura de repetição da linha 14, sendo este responsável pela execução de três bips (linha 15), com intervalo de 300 milissegundos entre cada execução (linha 16).

Após, a linha 20 inicia a comunicação *Bluetooth*, aguardando que algum dispositivo cliente se conecte no servidor. Acontecendo isso, as linhas 23 e 24 são responsáveis pela criação dos fluxos de entrada e saída de dados, estes realizados via comunicação *Bluetooth*. Por fim, é instanciado um objeto do tipo *Ultrasonic Sensor* (linha 27), para a leitura dos dados do sensor ultrassônico.

No *Loop* principal do programa, que inicia na linha 31, é recuperada a distância através do sensor ultrassônico (linha 33), enviando esta informação para o cliente (linha 36). Após é realizada uma pequena pausa de meio segundo (linha 37), sendo aguardado um retorno do cliente, confirmando ou não o recebimento da mensagem *Bluetooth* (linha 42), esta mensagem é apresenta no *display* do dispositivo (linha 43).

Concluindo o aplicativo, é realizada uma nova pausa de meio segundo, antes da nova leitura do sensor ultrassônico.

Após o desenvolvimento e a execução do aplicativo servidor, o próximo passo é o desenvolvimento do aplicativo cliente, cujo código é apresentado na Listagem 9.

Listagem 9: Código do aplicativo cliente para comunicação *Bluetooth*

```
01.  import java.io.DataInputStream;
02.  import java.io.DataOutputStream;
03.  import java.io.IOException;
04.  import javax.bluetooth.RemoteDevice;
05.  import lejos.nxt.Sound;
06.  import lejos.nxt.comm.BTConnection;
07.  import lejos.nxt.comm.Bluetooth;
08.
09.  public class BTCliente {
10.
11.   public static void main(String[] args) throws InterruptedException, IOException {
12.
13.          for( int i = 0; i < 3; i++ ){
14.                  Sound.beep();
15.                  Thread.sleep( 700 );
```

```
16.             }
17.
18.             System.out.println( "Procurando Servidor" );
19.             RemoteDevice servidor = Bluetooth.getKnownDevice( "NXT-1" );
20.
21.             int i = 0;
22.
23.             while( servidor == null ){
24.
25.                     System.out.println( "Procurando Servidor " + i++);
26.                     Thread.sleep( 500 );
27.
28.                     servidor = Bluetooth.getKnownDevice( "NXT-1" );
29.             }
30.
31.             System.out.println( "Estabelecendo conexão" );
32.             BTConnection btc = Bluetooth.connect( servidor );
33.
34.             System.out.println( "Criando os fluxos de I/O" );
35.             DataInputStream in = btc.openDataInputStream();
36.
37.             DataOutputStream out = btc.openDataOutputStream();
38.
39.             while( true ) {
40.
41.                     int entrada = in.readInt();
42.                     System.out.println( "Leu dado: " + entrada );
43.
44.                     Thread.sleep( 500 );
45.
46.                     out.writeBoolean( true );
47.                     Thread.sleep( 500 );
48.             }
49.     }
50.  }
```

Assim como no aplicativo servidor, as primeiras linhas são responsáveis pelo *import* das classes (linhas 01 a 07), declaração da classe (linha 09) e declaração da classe principal (linha 11). O primeiro comando (linhas 13 a 16) é responsável pela execução de três bips iniciais, iniciando na sequência a lógica do programa.

A linha 19 é responsável por se conectar ao servidor. Para isso, é necessário conhecer o nome do mesmo. Se isso não for possível (como por exemplo, o servidor estar desligado), o objeto do tipo *RemoteDevice* é valorizado com *null*, caindo no *Loop* das linhas 23 a 29, onde novas tentativas de conexão com o servidor são realizadas. Este *Loop* só finda após um servidor ser encontrado.

Na sequência, na linha 32, é realizada a conexão com o servidor, o que permite a criação dos fluxos de entrada e saída de dados via *Bluetooth* (linhas 35 e 37), por fim o *Loop* principal do programa é iniciado na linha 39, onde é aguardada uma mensagem enviada do servidor (linha 41), sendo esta apresentada para o usuário na linha 42. É realizada uma pausa de meio segundo (linha 44), para então retornar a mensagem *true* para o servidor, informando que a mensagem foi recebida. Por fim, uma nova pausa de meio segundo (linha 47) para só então recomeçar o *Loop*.

RESTAURANDO *FIRMWARE* LEGO MINDSTORMS

Para voltar a usar o *firmware* LEGO NXT no Mindstorms, é necessário fazer o download do *firmware* no site da LEGO (Figura 160).

Figura 160 – LEGO *Support*
Fonte: Autoria própria (2016).

Após o download da última versão do *firmware*, será necessário colocar o Mindstorms in *Modo Update*. Para isso basta segurar o botão *Reset* (Figura 161) por mais de 5 segundos e um som será executado.

Figura 161 – Resetando o LEGO Mindstorms
Fonte: Kohler (2011).

Para ter certeza, verifica-se nos dispositivos do computador se o NXT está conectado, como demonstrado na Figura 162.

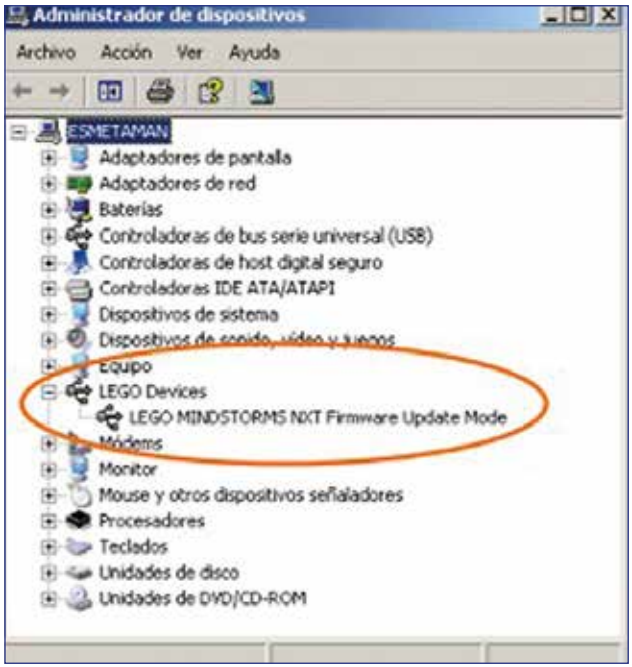


Figura 162 – LEGO *Firmware Update Mode*

Fonte: Autoria própria (2016).

Após isso, com o *Software* que acompanha o LEGO Mindstorms, basta navegar nos menus até a opção *Tools* e *Update* NXT *Firmware* (Figura 163).

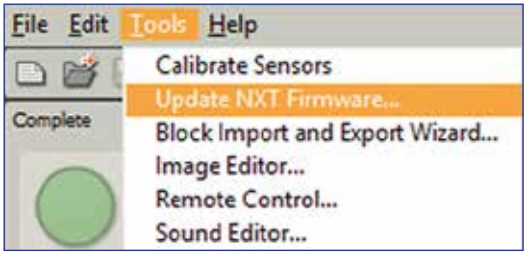


Figura 163 – LEGO *Tools*

Fonte: Autoria própria (2016).

Após clicar na opção representada na Figura 163, é exibida uma tela para indicar a pasta em que o *firmware* está localizado em seu computador (Figura 164).

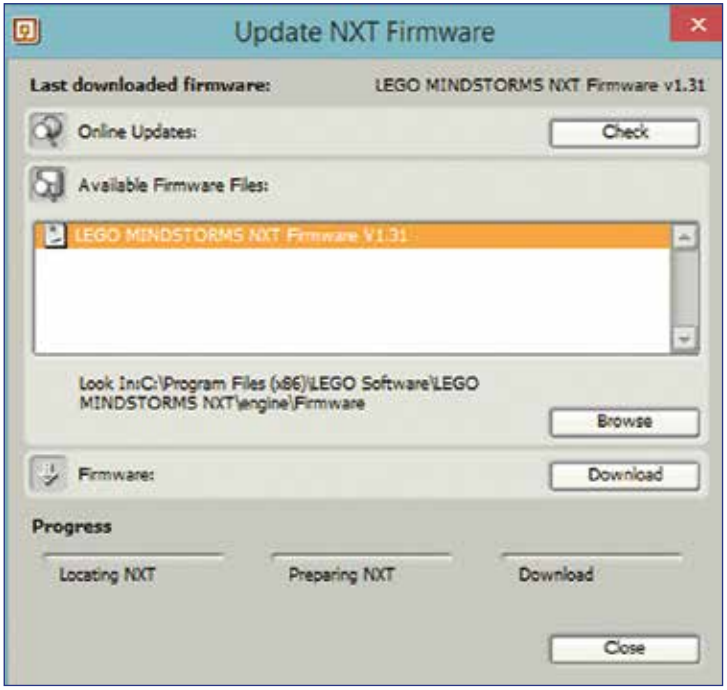


Figura 164 – NXT *Firmware Update*

Fonte: Autoria própria (2016).

Na sequência, clica-se no botão *Download*, e o *Update* do *firmware* será realizado. Ao terminar, a mensagem *Successfully Downloaded Firmware*! será mostrada (Figura 165).

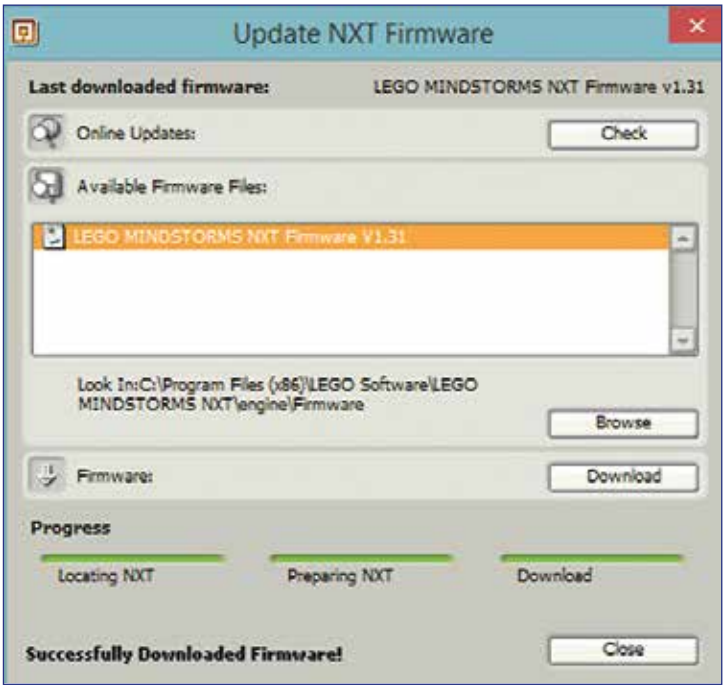


Figura 165 – NXT *Firmware Update Success*

Fonte: Autoria própria (2016).

REFERÊNCIAS

ADAMI, A. **LEGO Mindstorms**. Disponível em: <http://www.infoescola.com/tecnologia/lego-mindstorms/>. Acesso em: 10 set. 2016.

CARMEN. **Robot navigation toolkit**. Disponível em: <http://carmen.sourceforge.net>. Acesso em: 07 set. 2016.

CIRIACO, D. **O que é Bluetooth?** 2008. Disponível em: <http://www.tecmundo.com.br/bluetooth/161-o-que-e-bluetooth-.htm>. Acesso em: 08 set. 2016.

CYBER TOY. **LEGO Mindstorms 3804 Robotics Invention System 2.0. 2011**. Disponível em: <http://cybertoy.net/2011/10/19/lego-mindstorms-3804-robotics-invention-system-2-0/>. Acesso em: 10 set. 2016.

HITECHNIC. **NXT IRLink sensor**. Disponível em: <http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=nil1046>. Acesso em: 09 set. 2016.

KOHLER, D. **How to flash a Lego NXT brick on Ubuntu**. 2011. Disponível em: <http://www.damonkohler.com/2011/01/flash-nxt-ubuntu.html>. Acesso em: 11 set. 2016.

LAURENS. **Tutorial:** understanding the difference between NXT set versions. 2012. Disponível em: <http://robotsquare.com/2012/02/18/understanding-nxt-versions/>. Acesso em: 09 set. 2016.

LEGO ENGINEERING. **NXT sensors**. 2013. Disponível em: <http://www.legoengineering.com/nxt-sensors/>. Acesso em: 04 set. 2016.

LEGO GROUP. **Help and support LEGO Mindstorms NXT**. 2016.

LIMA, C. S. **Apostila de lógica**. Disponível em: <https://docente.ifrn.edu.br/cleonelima/disciplinas/fundamentos-de-programacao-2.8401.1m/fundamentos-de-logica-e-algoritmos-1.8401.1v/apostila-proposicoes-tabelas-verdade-conectivos-logicos>. Acesso em: 11 set. 2016.

MICROSOFT. **Microsoft robotics developer studio 4**. Disponível em: <http://www.microsoft.com/en-us/download/details.aspx?id=29081>. Acesso em: 08 set. 2016.

MINDSTORMS. **Build a robot**. Disponível em: <http://www.lego.com/en-us/mindstorms/build-a-robot>. Acesso em: 04 set. 2016b.

MINDSTORMS. **Infrared seeker**. Disponível em: <https://shop.lego.com/en-US/Infrared-Seeker-for-LEGO-MINDSTORMS-NXT-2852725>. Acesso em: 09 set. 2016c.

MINDSTORMS. **Shop LEGO**. Disponível em: <https://shop.lego.com/en-US/LEGO-MINDSTORMS-NXT-2-0-8547>. Acesso em: 10 set. 2016a.

NATIONAL INSTRUMENTS. **Bluetooth connection between NXT bricks**. 2009. Disponível em: <http://zone.ni.com/reference/en-XX/help/372962A-01/lvnxt/bluetoothconnectionbetweennxtbricks/>. Acesso em: 08 set. 2016.

OUTER PLACES. **The origins of the modern robot:** from da Vinci's columbine to a defecating duck. Disponível em: <https://www.outerplaces.com/science/item/4919-the-origins-of-the-modern-robot-a-wooden-bird-from-400-bc-leonardos-columbine-and-a-deficating-duck?utm_ expid=68243097-1. LutSYZjjRsmSMx2YZefKRA.0&utm_referrer=https%3A%2F%2F www.google.com.br%2F>. Acesso em: 10 set. 2016.

PLAYER. **The player project**. 2014. Disponível em: <http://playerstage.sourceforge.net>. Acesso em: 07 set. 2016.

PONTES, L. **Robótica, a origem!** 2010. Disponível em: <https://lelinopontes.wordpress.com/tag/origem-da-robotica/>. Acesso em: 07 set. 2016.

RICHARDSON, K. The truth is no stranger than fiction when it comes to robots. **The Conversation**, Aug. 2013. Disponível em: <http://robohub.org/the-truth-is-no-stranger-than-fiction-when-it-comes-to-robots/>. Acesso em: 10 set. 2016.

ROS. **Robot operating system**. Disponível em: <http://www.ros.org>. Acesso em: 08 set. 2016.

SENTOSA, S. **Karakuri:** Japanese mechanized automata doll/puppet. 2011. Disponível em: <http://myinternetcorner.com/karakuri-japanese-mechanized-automata-doll-puppet/>. Acesso em: 10 set. 2016.

TURING, A. M. Computing machinery and intelligence. **Mind**, v. LIX, n. 236, p. 435-460, 1950. Disponível em: <http://mind.oxfordjournals.org/content/LIX/236/433.full.pdf+html>. Acesso em: 07 set. 2016.

WORLD MYSTERIES. **Vitruvian Man by Leonardo da Vinci**. 2011. Disponível em: <http://blog.world-mysteries.com/science/vitruvian-man-by-leonardo-da-vinci/>. Acesso em: 10 set. 2016.

WS KITS EDUCACIONAL. **Lego Mindstorms NXT Education 9797**. Disponível em: <http://robomindstormlego.wskits.com.br/lego-9797>. Acesso em: 08 set. 2016.

APÊNDICES

APÊNDICE A – OPERAÇÕES LÓGICAS

OPERAÇÃO *AND* (CONJUNÇÃO)

Nesta operação o resultado da combinação entre dois valores lógicos de entrada somente será verdadeiro quando ambos forem verdadeiros; caso contrário, o resultado será falso. O Quadro 42 mostra a tabela verdade da operação *And*, com as entradas booleanas (verdadeiro, também representada pelo número 1, e falso, representado pelo número 0).

Entrada A	Entrada B	Saída
Falso (0)	Falso (0)	Falso (0)
Falso (0)	Verdadeiro (1)	Falso (0)
Verdadeiro (1)	Falso (0)	Falso (0)
Verdadeiro (1)	Verdadeiro (1)	Verdadeiro (1)

Quadro 42 – Tabela verdade da operação *And*

Fonte: Adaptado de Lima (2016, p. 3).

OPERAÇÃO *OR* (DISJUNÇÃO)

Nesta operação, o resultado será verdade se houver pelo menos uma entrada verdadeira. Caso contrário, se as duas entradas forem falsas, a saída também será falsa. O Quadro 43 mostra a tabela verdade da operação *Or*.

Entrada A	Entrada B	Saída
Falso (0)	Falso (0)	Falso (0)
Falso (0)	Verdadeiro (1)	Verdadeiro (1)
Verdadeiro (1)	Falso (0)	Verdadeiro (1)
Verdadeiro (1)	Verdadeiro (1)	Verdadeiro (1)

Quadro 43 – Tabela verdade da operação *Or*

Fonte: Adaptado de Lima (2016, p. 6).

OPERAÇÃO *XOR* (DISJUNÇÃO EXCLUSIVA)

Nesta operação, se as entradas possuírem valores distintos (uma verdadeira e a outra falsa), o resultado será verdadeiro. Nos demais casos, falso. O Quadro 44 apresenta a tabela verdade da operação *Xor*.

Entrada A	Entrada B	Saída
Falso (0)	Falso (0)	Falso (0)
Falso (0)	Verdadeiro (1)	Verdadeiro (1)
Verdadeiro (1)	Falso (0)	Verdadeiro (1)
Verdadeiro (1)	Verdadeiro (1)	Falso (0)

Quadro 44 – Tabela verdade da operação *Xor*

Fonte: Adaptado de Lima (2016, p. 8).

OPERAÇÃO *NOT* (NEGAÇÃO)

Está operação é unária, assim, só é aplicada a uma entrada. Sua função é inverter o valor recebido, se for verdadeiro muda-o para falso, e vice-versa. O Quadro 45 mostra a tabela verdade da operação *Not*.

Entrada A	Saída
Falso (0)	Verdadeiro (1)
Verdadeiro (1)	Falso (0)

Quadro 45 – Tabela verdade da operação *Not*

Fonte: Adaptado de Lima (2016, p. 13).

APÊNDICE B – EXEMPLO DE PROGRAMA QUE GRAVA E REPRODUZ AÇÕES

Este programa tem como finalidade copiar os movimentos realizados pelos motores do NXT por 10 segundos e executá-los na sequência.

Para o programa, será utilizado o componente de *Record/Play* para gravar os movimentos dos motores e executá-los posteriormente, além dos componentes de *Sound*, *Display* e *Wait*, de modo que, a partir destes, o usuário possa compreender melhor o funcionamento do programa.

A Figura 166 apresenta o aplicativo desenvolvido.

Figura 166 – Representação do aplicativo copia movimento

Fonte: Autoria própria (2016).

O aplicativo inicia com um componente *Display*, o qual apresenta para o usuário a mensagem **Preparando**, conforme Figura 167.

Figura 167 – Painel de configuração do primeiro componente *Display*

Fonte: Autoria própria (2016).

O primeiro *Loop*, que possui dois componentes (*Sound* e *Wait*), faz com que o aplicativo, antes de ser executado, execute três bips, informando ao usuário que o programa está prestes a iniciar.

A Figura 168 apresenta o *Loop*, que executará 3 vezes a ação.

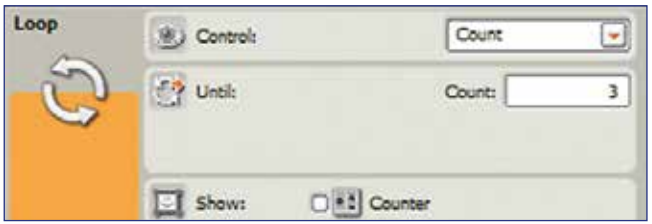


Figura 168 – Painel de configuração do primeiro *Loop*

Fonte: Autoria própria (2016).

O som executado é um tom, com duração de meio segundo, conforme Figura 169.

Figura 169 – Painel de configuração do primeiro componente *Sound*

Fonte: Autoria própria (2016).

Para permitir um tempo entre um tom e outro, o componente *Wait* foi utilizado, dando uma pause de 0,2 segundos entre os tons, conforme Figura 170.

Figura 170 – Painel de configuração do primeiro componente *Wait*

Fonte: Autoria própria (2016).

Após os bips iniciais, o aplicativo é executando, iniciando com a apresentação do texto **Gravando** no *display* do NXT (Figura 171). Este texto será apresentado durante os 10 segundos em que a gravação dos movimentos será realizada.

Figura 171 – Painel de configuração do segundo componente *Display*

Fonte: Autoria própria (2016).

Após a formatação dos dados no *display*, o componente *Record/Play* gravará os movimentos, com base nas características do componente apresentado na Figura 172.

Figura 172 – Painel de configuração do componente *Record/Play* para gravar os movimentos

Fonte: Autoria própria (2016).

Após um novo *Display*, o texto **Reproduzindo!!** é apresentado no *display* do dispositivo, conforme Figura 173.

Figura 173 – Painel de configuração do terceiro componente *Display*

Fonte: Autoria própria (2016).

Antes da reprodução, uma nova sequência de bips é realizada, da mesma forma que no *Loop* inicial do aplicativo, por fim, os movimentos são reproduzidos com a utilização de um novo componente *Record/Play* (Figura 174). Vale ressaltar que o nome inserido em *Name* deve ser o mesmo no componente ao gravar as ações e no componente que irá executar as ações.

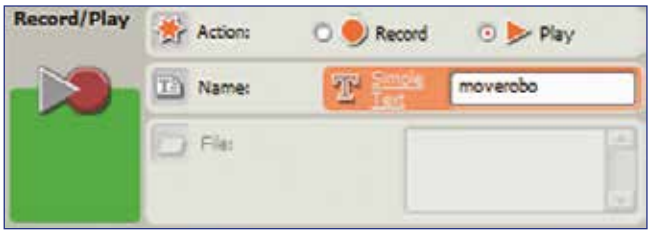


Figura 174 – Painel de configuração do componente *Record/Play* para executar os movimentos gravados anteriormente

Fonte: Autoria própria (2016).

Pode-se perceber que, ao iniciar o programa, aparecerá no *Display* do NXT a mensagem **Preparando** seguida de três bips indicando que o aplicativo foi iniciado. Em seguida, é apresentado o texto de **Gravando**, que a partir deste aviso o usuário deverá realizar os movimentos dos motores a serem gravados nos próximos 10 segundos. Após isso, é apresentado o texto de **Reproduzindo!!** e soarão outros três bips, que indicarão o início da reprodução dos movimentos dos motores gravados anteriormente.

AUMENTA VELOCIDADE

O programa que segue tem a função de fazer o NXT aumentar a velocidade gradativamente, começando com velocidade numérica igual a 0, e aumentando de 10 em 10 (através do componente *Math*), até atingir a velocidade máxima de 100. A mudança de velocidade deve acontecer a cada segundo. Ao final de 10 segundos, o programa deve ser encerrado.

A Figura 175 apresenta o programa desenvolvido.

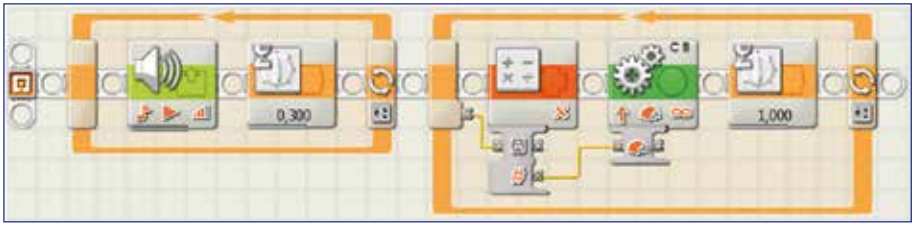


Figura 175 – Representação do aplicativo Aumenta velocidade

Fonte: Autoria própria (2016).

À esquerda do programa, o primeiro *Loop* representa a execução de três bips, antes do início da lógica do mesmo.

A Figura 176 apresenta o painel de configurações do segundo *Loop*, que está configurado para executar por dez vezes os blocos em seu interior.

Figura 176 – Propriedades do segundo *Loop*

Fonte: Autoria própria (2016).

O bloco *Math* aumenta a potência dos motores de 10 em 10 a cada segundo, este tempo é controlado através do bloco de *Wait*. O painel de configurações do componente *Math* é apresentado na Figura 177.

Figura 177 – Propriedades do componente *Math*

Fonte: Autoria própria (2016).

A Figura 178 apresenta o painel de configurações do componente *Move*.

Figura 178 – Propriedades do componente *Move*

Fonte: Autoria própria (2016).

A Figura 179 apresenta o painel de configurações do componente *Wait*.

Figura 179 – Propriedades do componente *Wait*

Fonte: Autoria própria (2016).

DIMINUI VELOCIDADE

Este programa tem a função de movimentar o NXT, sendo que a velocidade deve ser proporcional à distância do obstáculo, o que fará o NXT diminuir sua velocidade gradativamente, à medida que se aproxima de um obstáculo. Para evitar uma colisão, quando a distância for menor ou inferior a 20 cm, o programa deve encerrar o processamento, fazendo o NXT parar.

A Figura 180 representa o programa desenvolvido.

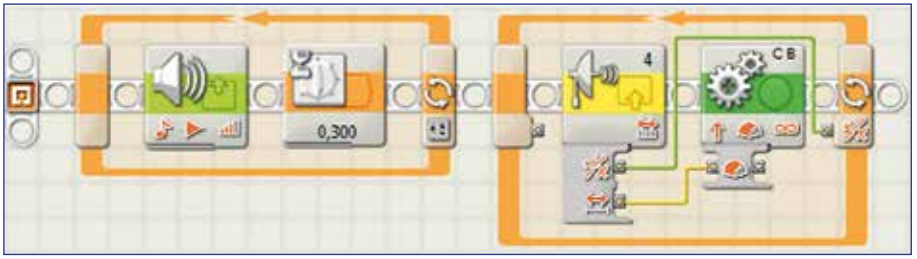


Figura 180 – Representação do aplicativo Diminui velocidade

Fonte: Autoria própria (2016).

À esquerda do programa, o primeiro *Loop* representa a execução de três bips, antes do início da lógica do mesmo, como já foi feito anteriormente.

A Figura 181 apresenta o painel de configurações do segundo *Loop*, o qual está configurado para verificar a distância que um objeto está do NXT, caso esteja próximo o mesmo irá reduzir a velocidade.

Figura 181 – Propriedades do segundo *Loop*

Fonte: Autoria própria (2016).

A Figura 182 apresenta o painel de configurações do sensor ultrassônico, cuja alteração foi mudar a forma de medida para 20 cm.

Figura 182 – Propriedades do sensor ultrassônico

Fonte: Autoria própria (2016).

A Figura 183 apresenta o painel de configurações do componente *Move*.

Figura 183 – Propriedades do componente *Move*

Fonte: Autoria própria (2016).

BRANCO E PRETO

O aplicativo seguinte é executado **eternamente**, apresentando, no *display* do NXT, a mensagem **Branco**, se o sensor de cor for colocado sobre uma superfície branca, **Preto** se colocado sobre uma superfície preta, e **Colorida**, se colocado em qualquer outra cor.

A Figura 184 representa o programa desenvolvido.

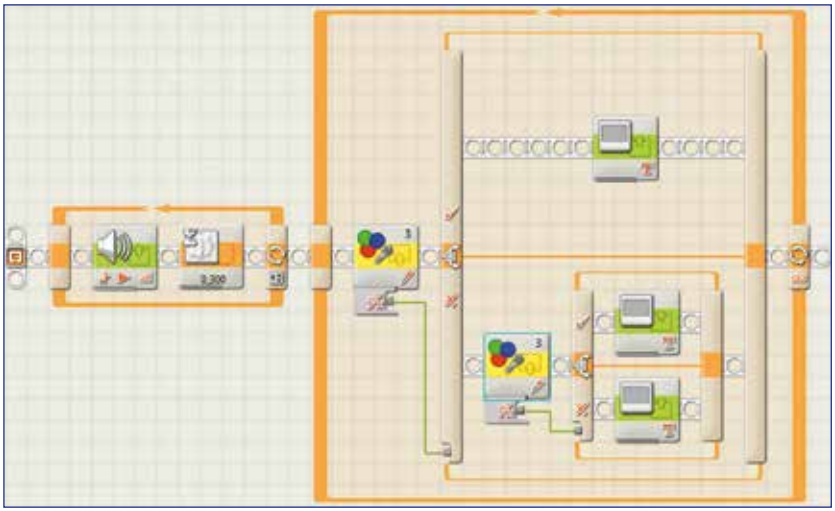


Figura 184 – Representação do aplicativo Branco e preto

Fonte: Autoria própria (2016).

À esquerda do programa, o primeiro *Loop* representa a execução de três bips, antes do início da lógica do mesmo, como já foi feito anteriormente.

O segundo *Loop* irá executar **eternamente**. O *Switch* externo está conectado ao primeiro componente *Color Sensor*, que irá fazer a verificação, se caso a cor for branca, irá aparecer o texto **Branco** no *display* do NXT, caso contrário, o segundo componente *Color Sensor*, conectado ao *switch* interno, irá verificar se a cor detectada é preta, se sim, irá apresentar o texto **Preto** no *display* do NXT, se não, irá aparecer o texto **Colorido** no *display*, representando que é qualquer outra cor.

A Figura 185 apresenta o painel de configurações do primeiro sensor de cor.

Figura 185 – Propriedades do primeiro sensor de cor

Fonte: Autoria própria (2016).

A Figura 186 apresenta o painel de configurações do *switch* externo.

Figura 186 – Propriedades do *Switch* externo

Fonte: Autoria própria (2016).

A Figura 187 apresenta o painel de configurações do componente *Display*, representando a cor branca.

Figura 187 – Propriedades do componente *Display*

Fonte: Autoria própria (2016).

A Figura 188 apresenta o painel de configurações do segundo sensor de cor.

Figura 188 – Propriedades do segundo sensor de cor

Fonte: Autoria própria (2016).

A Figura 189 apresenta o painel de configurações do *switch* interno.

Figura 189 – Propriedades do *Switch* interno

Fonte: Autoria própria (2016).

A Figura 190 apresenta o painel de configurações do componente *Display*, representando a cor preta.

Figura 190 – Propriedades do componente *Display*

Fonte: Autoria própria (2016).

A Figura 191 apresenta o painel de configurações do componente *Display*, representando outra cor.

Figura 191 – Propriedades do componente *Display*

Fonte: Autoria própria (2016).

GRAVANDO ARQUIVO DE LOG

Este aplicativo tem a função de, uma vez por segundo, durante 10 segundos, ler informações do sensor de distância, adicionando estes dados em um arquivo texto chamado log.txt. Para os dados armazenados no arquivo, deve-se utilizar um identificador único crescente, mais o caractere de ponto-e-vírgula (;), além da distância lida. Durante o processo de leitura, apresentar na tela do dispositivo a mensagem **Gravando Log**. Ao final, o usuário poderá copiar este arquivo para um computador *Desktop*, abrindo-o em um editor de texto.

A Figura 192 representa o programa desenvolvido.

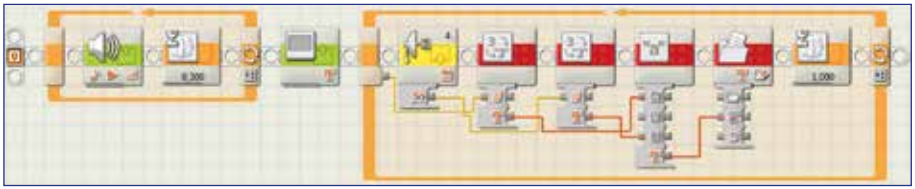


Figura 192 – Representação do aplicativo Gravando arquivo de LOG

Fonte: Autoria própria (2016).

À esquerda do programa, o primeiro *Loop* representa a execução de três bips, antes do início da lógica do mesmo, como já foi feito anteriormente.

O componente *Display* mostra uma mensagem de que será iniciada a gravação no arquivo *log.txt* (.txt é uma extensão de arquivo de textos que contém pouca formatação; por exemplo, sem negrito ou itálico). A Figura 193 apresenta o painel de configuração deste componente.

Figura 193 – Propriedades do componente *Display*

Fonte: Autoria própria (2016).

A cada leitura dos dados, o componente *Wait* fará o programa aguardar um segundo até as operações serem executadas novamente. O *Loop* irá executar dez vezes, e o valor de cada execução será armazenado no arquivo log através de um *data hub* no componente *Number to Text* (como o valor numérico é obtido através de um *data hub*, este componente não é alterado em suas propriedades). O mesmo vale para os valores lidos do sensor ultrassônico. Estes valores são então concatenados pelo componente *Text* e gravados em um log pelo componente *File Access*.

A Figura 194 apresenta o painel de configurações *Loop*.

Figura 194 – Propriedades do *Loop*

Fonte: Autoria própria (2016).

A Figura 195 apresenta o painel de configurações do componente *Text*.

Figura 195 – Propriedades do *Text*

Fonte: Autoria própria (2016).

A Figura 196 apresenta o painel de configurações do componente *File Access*.

Figura 196 – Propriedades do *File Access*

Fonte: Autoria própria (2016).

ENVIA A DISTÂNCIA LIDA (SENSOR DE DISTÂNCIA, VIA *BLUETOOTH*, PARA O OUTRO NXT)

O sistema abaixo apresenta uma introdução sobre comunicação *Bluetooth* entre diferentes dispositivos LEGO Mindstorms, desta forma, para os testes são necessários dois NXTs. Para facilitar o desenvolvimento, um deles será nomeado de servidor, e terá a função de capturar a distância, em cm, utilizando o sensor ultrassônico, e esta informação é enviada ao aplicativo cliente, o qual receberá este dado e apresentará na tela.

Como se trata de comunicação *Bluetooth*, antes da execução do aplicativo é necessário estabelecer uma conexão *Bluetooth* entre o dispositivo servidor e o cliente. Para isso, em ambos os *Brick* NXT, cliente e servidor, deve-se habilitar a comunicação *Bluetooth*, acessando o menu *Bluetooth*, e escolhendo a opção *On/Off* e, por fim, selecionando a opção *On*.

O próximo passo é tornar visível o dispositivo cliente, a fim de que o servidor o encontre para estabelecer uma comunicação. Desta forma, no *Brick* NXT do cliente, entrar novamente no menu *Bluetooth*, opção *Visibility*, e escolhendo *Visible*.

Agora o cliente está apto a ser localizado e conectado pelo aplicativo servidor, assim, no *Brick* Servidor, entrar no menu *Bluetooth*, escolhendo a opção *Search*. Aparecerão alguns pontos de interrogação (?) na tela, enquanto o dispositivo servidor localiza dispositivos *Bluetooth*. Na sequência, será apresentada uma lista de dispositivos *Bluetooth* próximos, escolha o nome do LEGO Mindstorms do dispositivo cliente (o nome do dispositivo é apresentado na parte superior, ao centro do *display*). Clicando sobre o nome do dispositivo, é solicitado um número para a conexão, podendo ser o número 1, 2 ou 3 (para este exemplo foi utilizado o número 1).

Agora, já existe uma comunicação *Bluetooth* estabelecida, na conexão 1, entre o dispositivo servidor e o dispositivo cliente, bastando para a comunicação desenvolver um software para rodar em ambos os dispositivos.

Como o objetivo é fazer o servidor ler dados do sensor ultrassônico, em cm, e enviar estes para o cliente, primeiramente será codificado o aplicativo servidor, conforme a Figura 197.

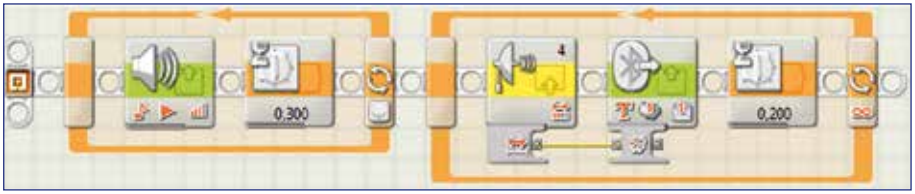


Figura 197 – Representação do aplicativo servidor

Fonte: Autoria própria (2016).

À esquerda do aplicativo, o primeiro *Loop* representa a execução de três bips, antes do início da lógica do aplicativo, como já feito em outros programas apresentados.

O segundo *Loop* será executado **eternamente**, apresentando recuperando os dados do sensor ultrassônico conforme as propriedades da Figura 198.

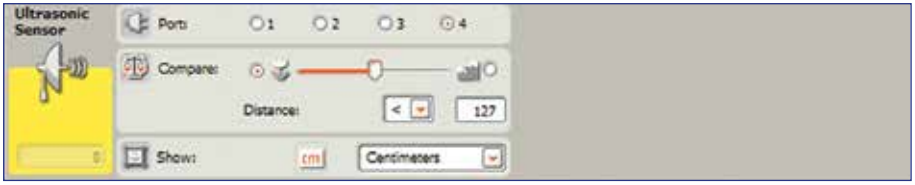


Figura 198 – Propriedades do sensor ultrassônico

Fonte: Autoria própria (2016).

No componente do sensor, a única mudança necessária está na unidade de medida, que deve ser em cm. Esta informação alimentará o componente *Send Message* via *Data Hub*, sendo a informação numérica lida enviada ao aplicativo cliente. As propriedades do *Send Message* são apresentadas na Figura 199.

Figura 199 – Propriedades do *Send Message*

Fonte: Autoria própria (2016).

Este componente estabelece a troca de informações utilizando a conexão 1, este previamente estabelecido no início deste texto, no próprio *Brick* do servidor. Como a mensagem será numérica (distância lida), é necessária a troca da propriedade *Message para Number*. A propriedade *Mailbox*, que informa qual a caixa de entrada que receberá a mensagem poderá continuar 1, sendo este o valor padrão.

Para evitar um estouro do *buffer* de saída da mensagem, dado o grande número de mensagens que podem ser enviadas em curto espaço de tempo, torna-se necessário um componente *Wait*, conforme a Figura 200.

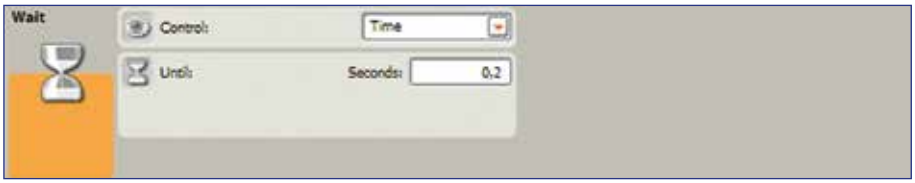


Figura 200 – Propriedades do *Wait*

Fonte: Autoria própria (2016).

Após isso, o aplicativo servidor está pronto, podendo ser executado no dispositivo servidor.

O próximo passo é o desenvolvimento do aplicativo cliente, que receberá as informações numéricas do servidor e as apresentará no *display* do cliente. Uma visão geral do aplicativo cliente é apresentada na Figura 201.

Figura 201 – Visão geral do aplicativo cliente

Fonte: Autoria própria (2016).

À esquerda, o *Loop* tradicional para emissão dos bips, este componente é aconselhável em todo o programa para LEGO Mindstorms e, no segundo *Loop* **eterno**, a lógica do aplicativo cliente. Basicamente este receberá, via componente *Receive Message*, uma mensagem *Bluetooth*, conforme Figura 202.

Figura 202 – Propriedades do *Receive Message*

Fonte: Autoria própria (2016).

A única mudança foi o tipo de mensagem, alterada para *Number*. A propriedade *Mailbox* será 1, já que foi esta a propriedade estabelecida no dispositivo servidor.

Via *Data Hub*, uma informação numérica sai do *Receive Message* e entra no componente *Text to Number*. Este não recebe mudança em suas propriedades, saindo dele uma informação textual, que alimentará o componente *Display*, cujas propriedades são apresentadas na Figura 203.

Figura 203 – Propriedades do *Display*

Fonte: Autoria própria (2016).

A única mudança realizada no *Display* foi a propriedade *Action*, mudada para *Text*, já que um texto será apresentado na tela do dispositivo.

Por fim, um componente *Wait* foi inserido para dar um *delay*, idêntico ao do servidor, entre uma mensagem e outra (Figura 204).

Figura 204 – Propriedades componente *Wait*

Fonte: Autoria própria (2016).

Capa: papel Triplex 250 gramas
Miolo: papel Couché Fosco 115 gramas
Fontes: Meiryo (textos, quadros e legendas)
e Grand National e Meiryo (títulos e subtítulos)
Tiragem: 500 exemplares
Livro impresso na Graciosa Gráfica e Editora Ltda.
Curitiba
2016
Impresso no Brasil
Printed in Brazil

A marca FSC® é a garantia de que a madeira utilizada na fabricação do papel deste livro provém de florestas que foram gerenciadas de maneira ambientalmente correta, socialmente justa e economicamente viável, além de outras fontes de origem controlada.

LEGO Mindstorms é uma linha do brinquedo LEGO lançada comercialmente em 1998, voltada para a educação tecnológica. Este ambiente é resultado da parceria entre o Media Laboratory do Massachusetts Institute of Technology (MIT) e o LEGO Group. É constituído por um conjunto de peças da linha LEGO Technic (motores, eixos, engrenagens, polias e correntes), acrescido de sensores de toque, de intensidade luminosa, de distância, de som, entre outros. Todos os sensores são controlados por um módulo programável. O LEGO Mindstorms possui um ambiente de desenvolvimento nativo, chamado de NXT-G, que é todo baseado no modo arrastar e soltar, fazendo com que algumas dificuldades antes encontradas como a complexidade para desenvolvimento de softwares sejam facilitadas, sendo este um ambiente comumente utilizado para o ensino de robótica e linguagem de programação para crianças, jovens e adultos. Este conjunto é utilizado com função didática em instituições de ensino tecnológico, abordando a teoria e a prática de conteúdos direcionados para a introdução à robótica, permitindo o desenvolvimento de projetos de pequeno e médio porte, estimulando a criatividade e a solução de problemas do cotidiano por parte dos alunos.

Agência Brasileira do ISBN
ISBN 978-85-7014-169-9
9 788570 141699