Robocode

Manual de Instruções

Robocode

Manual de Instruções

Universidade Federal de Juiz de Fora

Helder Linhares Bertoldo dos Reis

Professores Orientadores:
Jairo Franciso de Souza
Victor Ströele de Andrade Menezes

Sumário

O que é Robocode?

Robocode é um jogo de programação, onde o objetivo é desenvolver um robô para lutar contra outros robôs.

Utilizando as linguagens de programação JAVA ou .NET (C#), você implementa as classes e objetos que definirão a inteligência artificial por trás da estratégia de batalha do seu robô.

O lema do Robocode é: "*Construir o melhor, destruir o resto!*"

Escolas e universidades estão usando Robocode como auxiliar no ensino de programação e também para estudar a inteligência artificial (AI). O conceito de Robocode é fácil de entender e uma maneira divertida de aprender a programar.

Robocode vem com seu próprio instalador, editor built-in robo e compilador Java, e só exige um Java Virtual Machine (JVM) previamente instalado. Assim, tudo que um desenvolvedor precisa para começar é fornecido com o arquivo principal Robocode distribuição (robocode-xxx-setup.jar). O Robocode também suporta o desenvolvimento de robôs utilizando IDEs externas, como por exemplo, Eclipse , IntelliJ IDEA , NetBeans e Visual Studio, que proporcionam um desenvolvimento melhor do que o editor de robô do Robocode.

O fato do Robocode rodar na plataforma Java o torna possível sua execução em qualquer sistema operacional com Java pré-instalado, o que significa que ele será capaz de rodar em Windows, Linux, Mac OS e também no UNIX e suas variantes.

ALERTA: Robocode pode ser muito divertido, mas também é muito viciante.

Robocode é grátis e está sendo desenvolvido como um projeto de reposição, em tempo, onde não há dinheiro envolvido. Ele é um projeto Open Source, o que significa que todas as fontes são abertas a todos. Além disso, Robocode é fornecido sob os termos da EPL (Eclipse Public License).

Website Oficial do Robocode: http://robocode.sourceforge.net/

História

O jogo foi originalmente criado por Matthew A. Nelson, aka Mat Nelson, como um esforço pessoal no final de 2000 e se tornou profissional quando ele o trouxe à IBM, na forma de um download AlphaWorks, em julho de 2001.

No início de 2005, Robocode foi trazido para o SourceForge como Open Source na versão 1.0.7. Neste ponto, o desenvolvimento de Robocode estava um pouco parado. No entanto, a comunidade em torno de Robocode começou a desenvolver suas próprias versões, a fim de se livrar de bugs e também para colocar novas funcionalidades.As Contribuições para o Robocode Open Source e depois no projeto RobocodeNG foram feitas por Flemming N. Larsen.

Como nada parecia acontecer com Robocode em mais de um ano, Flemming Larsen N. assumiu o projeto Robocode no SourceForge como administrador e desenvolvedor em julho de 2006. O projeto RobocodeNG foi abandonado, mas a variante Robocode 2006, que continha uma série de contribuições da comunidade, foi incorporada pelo Robocode Oficial com a versão 1.1. Desde então, os lotes de novas versões do Robocode tem sido lançados com mais e mais recursos e contribuições da comunidade.

Recentemente (a partir da versão 1.7.2.0), a plataforma .NET passou a ser suportada, atráves de um plug-in fornecido por Pavel Savara baseado em jni4net e desenvolvido por ele próprio.

A Liga Brasileira de Robocode

A Liga Brasileira é o torneio mais importante de Robocode no Brasil. No website oficial é possível acompanhar o desenrolar de cada torneio local em todo Brasil, e conhecer as equipes brasileiras campeãs de Robocode.

O funcionamento da Liga é simples: todos os torneios de universidades, escolas técnicas, escolas públicas, clubes e até bairros tem como canal de divulgação o website oficial da liga. Todos os torneios locais reportam os dados de cada rodada para a liga, e posteriormente definem as equipes campeãs. Com os campeões de cada torneio local, realiza-se a liga dos campeões brasileiros, na qual todos os campeões dos torneios locais se enfretam na arena de batalha com seus robôs e suas estratégias em busca do título.

Website Oficial da LBR: http://www.robocodebrasil.com.br/

Como Instalar?

Pré-Requisitos:

- JAVA 6 ou mais recente

Linux:

Antes de iniciar assegure-se que uma versão do JAVA 6, ou mais recente está instalado no sistema:

Acesse um **Terminal** (ctrl + alt + T) e execute o comando:

- java -version

O comando retornará algo desse tipo:

java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)

O 8 de Java version “1.8.0_91” representa a versão instalada.

*Caso o comando não dê esse retorno, antes de prosseguir, é necessário realizar a instalação de uma versão recente do JAVA.

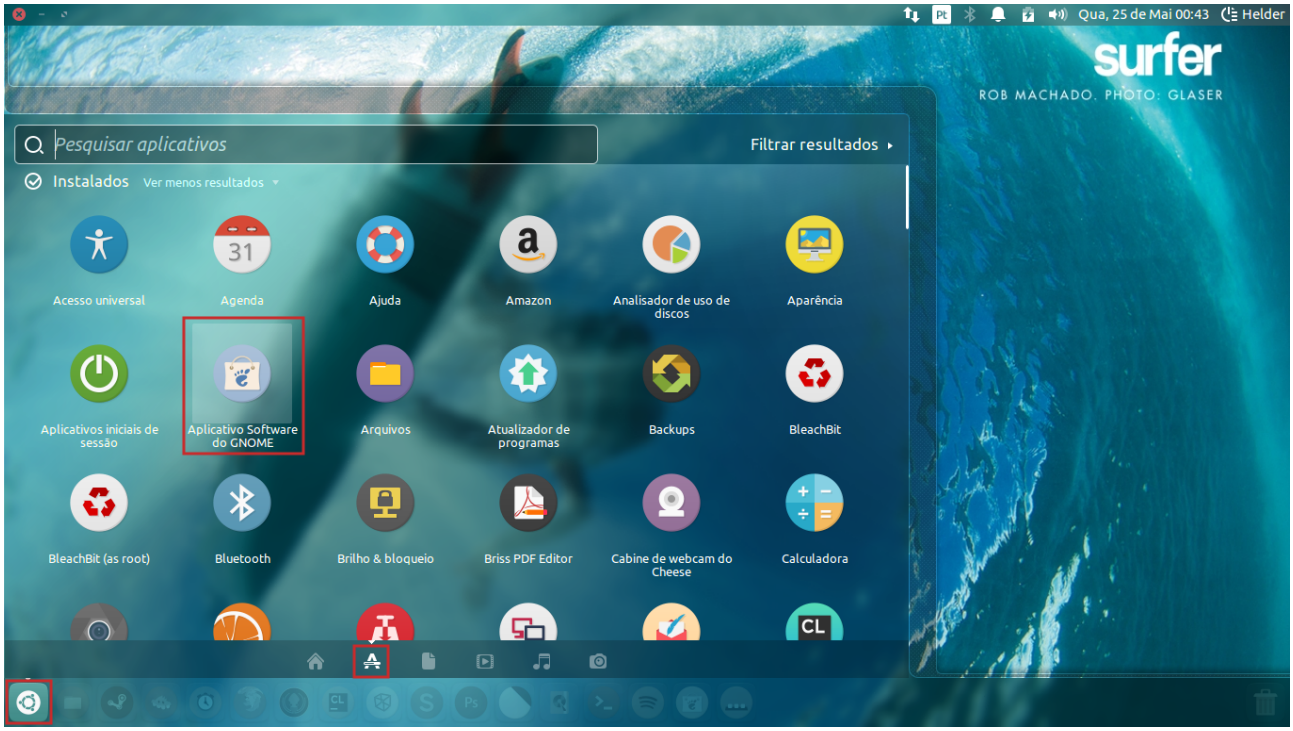
Para instalar o JAVA, ainda no Terminal execute os comandos:

- `sudo add-apt-repository ppa:webupd8team/java`
- `sudo apt-get update`
- `sudo apt-get install oracle-java8-installer`

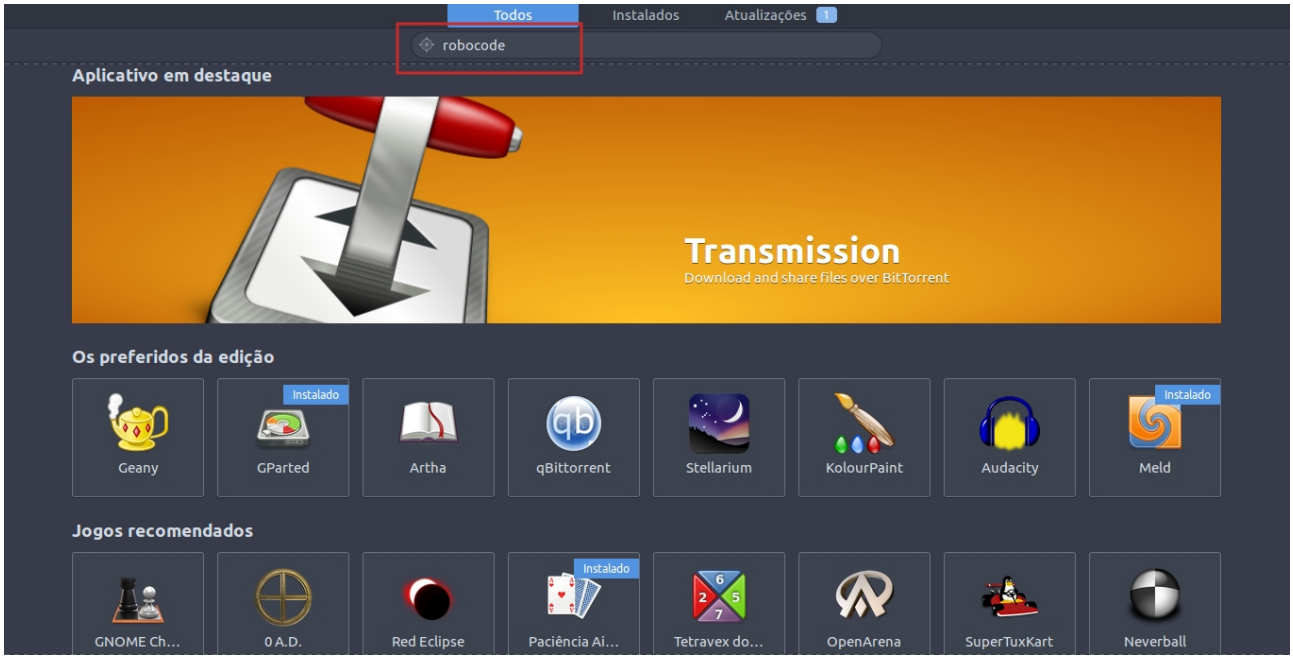
Instalando o Robocode:

1. Vá no **Menu Principal** do seu sistema e abra o aplicativo **Central de Programas** (Software Center).

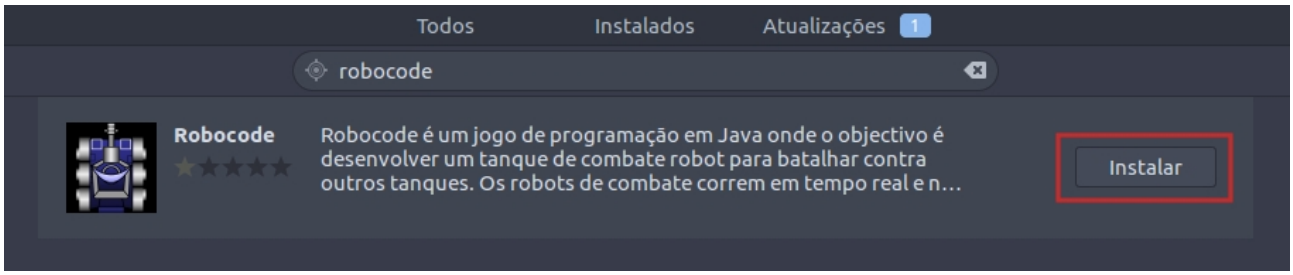
*O nome da sua Central de Programas pode variar de acordo com a sua versão.



2. Na **Central de Programas** pesquise por **Robocode**.



3. Clique na opção **Instalar** e informe a senha de administrador quando solicitado e aguarde o fim da instalação.



4. Se nenhum erro for reportado o Robocode está pronto pra usar, basta ir nos aplicativos e abri-lo.

* Caso queira executar o Robocode com gráficos melhores execute ele como administrador:

Abra um Terminal (ctrl + alt + T) e execute os comandos:

- `sudo su`
- `robocode`

Windows:

Antes de iniciar assegure-se que uma versão do JAVA 6, ou mais recente está instalado no sistema.

1. Acesse o **Menu Iniciar** / **Executar** e digite **CMD**
2. Após abrir o Prompt de Comando digite:

- `java -version`

  O comando retornará algo desse tipo:

  Java version "1.6.0_10"
  Java(TM) SE Runtime Environment (build 1.6.0_10-b33)
  Java HotSpot(TM) Client VM (build 11.0-b15, mixed mode, sharing)

O 6 de Java version “1.6.0_10” representa a versão instalada.

Caso o comando não dê esse retorno, antes de prosseguir, é necessário realizar a instalação de uma versão recente do JAVA. É possível baixar nesse link: https://www.java.com/pt_BR/

Executando o Robocode:

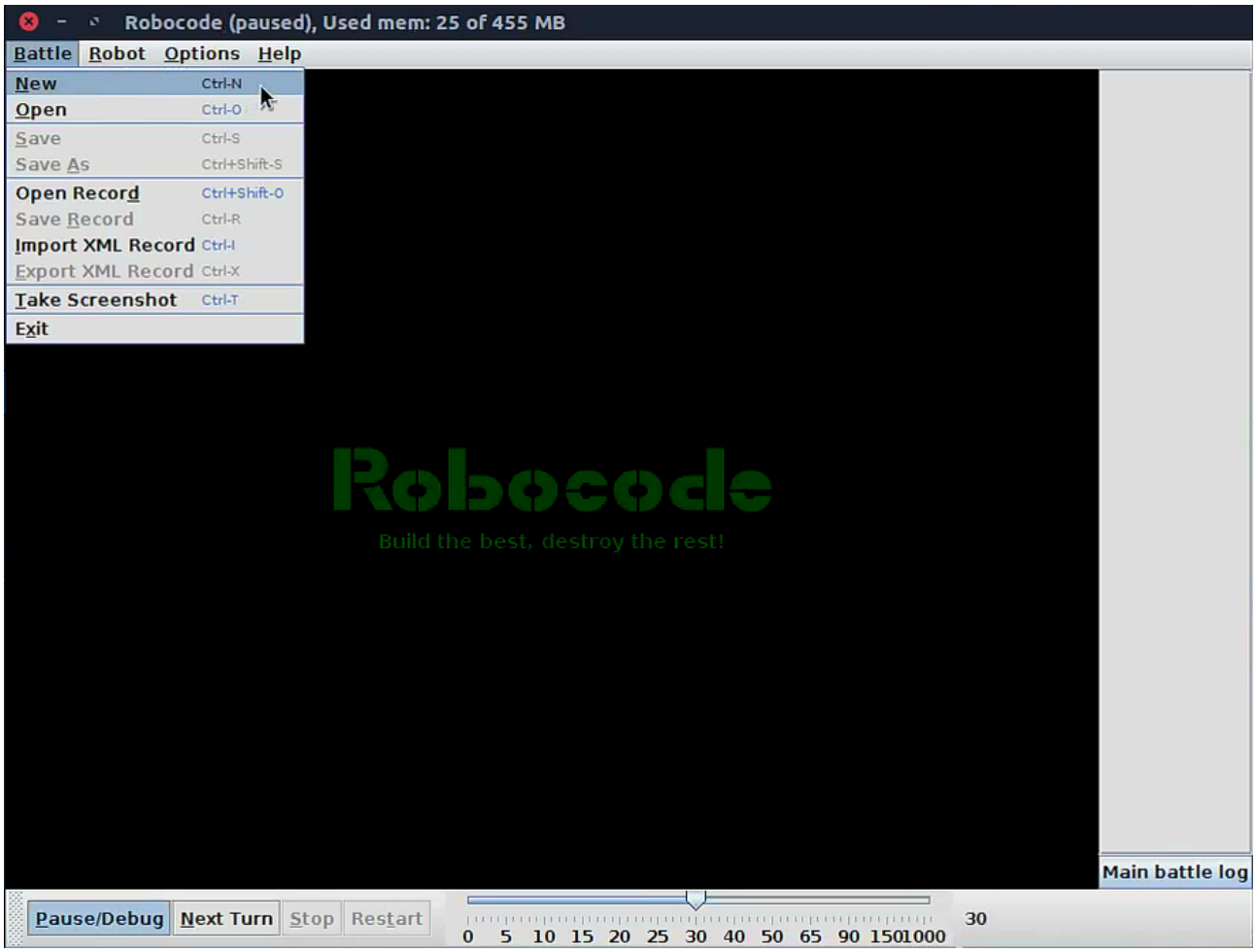
1. Acesse o link do projeto e faça o download do Robocode.

   https://sourceforge.net/projects/robocode/

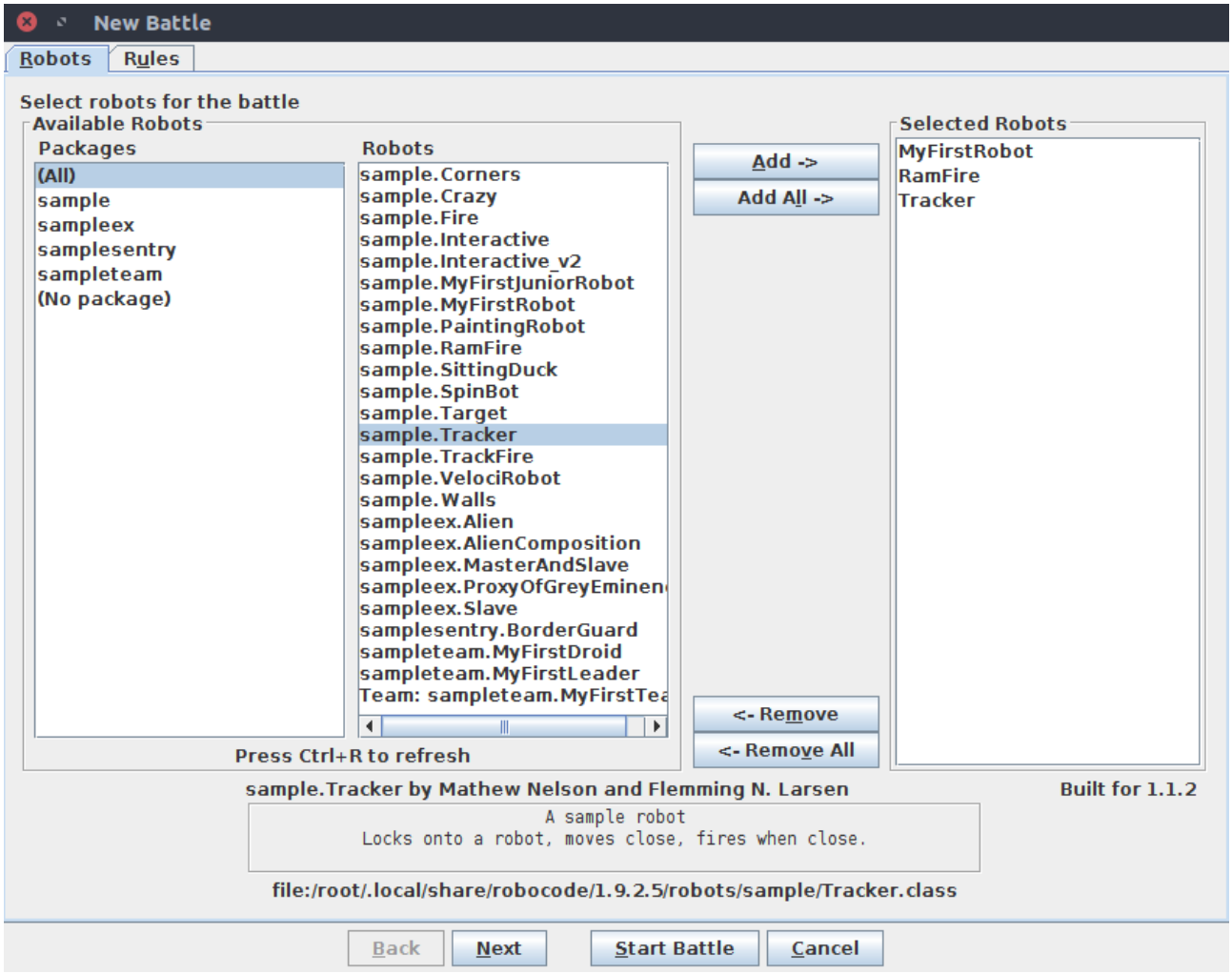
2. Basta ir até a pasta do Download e abrir o arquivo baixado que ele vai executar o programa sem precisar de instalação.

Iniciando no Robocode

- Após a instalação, primeiro, vamos executar uma batalha para ver como o jogo se parece. Basta clicar no Menu **Battle**, em seguida, selecione **New**, como mostrado na imagem abaixo:



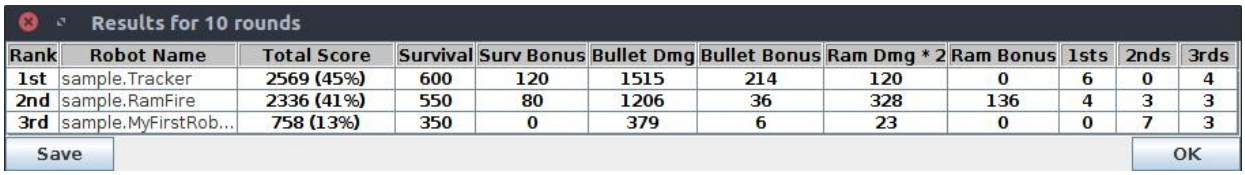
- Será mostrada a tela **New Battle**, onde você escolhe os robôs e as opções para uma batalha. Para esta batalha, vamos experimentar MyFirstRobot, RamFire e Tracker. Adicione-os, clicando duas vezes em seus nomes (ou selecionando cada uma e clicando em **Add**). A tela agora deve ser algo como isto:



- Clicar em **Next**. A primeira opção da nova tela é **Number of Rounds**.

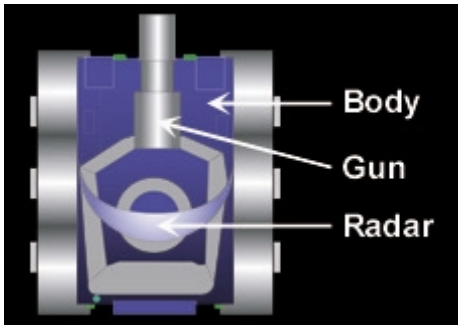
  No Robocode, cada batalha é composta por um número de rodadas, como você verá em breve. Por enquanto, vamos deixá-lo no padrão de 10.

  Há outros ajustes que podem ser feitos como o tamanho do campo de batalha, o tempo de inatividade e a taxa de resfriamento da arma, mas por enquanto não vamos modificar essas funções. Por fim, clique no botão **Start Battle** para começar! Preste atenção para a pequena dancinha do Tracker quando ele ganha rodadas.
- Após o fim da batalha é exibida as estatísticas da partida:



Anatomia de um Robô

Um robô consiste de 3 partes individuais:



- **Body (Chassi)** - Carrega a arma com o radar no topo. O corpo é utilizado para mover o robô para frente e para trás, bem como virar à esquerda ou à direita.
- **Gun (Canhão)** – Montado sobre o corpo e é utilizado para disparar balas de energia. A arma pode virar à esquerda ou à direita.
- **Radar** - Montado sobre a arma e é usado para localizar outros robôs quando movido. O radar pode virar à esquerda ou à direita. O radar gera eventos onScannedRobot quando forem detectados robôs.

Primeiro Robô

O objetivo desse tutorial é ensinar a como criar o seu primeiro robô.

Criando um Robô

Criar um robô pode ser fácil. Fazer do seu robô um vencedor não é. Você pode passar apenas alguns minutos sobre ele, ou você pode passar meses e meses. Vou avisá-lo que escrever um robô pode ser viciante! Depois que você terminar, você vai observar a sua criação, depois de passar por dificuldades no desenvolvimento, cometer erros e perder tiros críticos. Mas como você aprende, você vai ser capaz de ensinar seu robô como agir e o que fazer, onde ir, o que é para evitar, e onde disparar. Deve se esconder em um canto, ou saltar para a briga?

Primeiro Robô

Robocode vem com uma série de Robôs de exemplo para você ter uma ideia e ver como as coisas funcionam. Você pode usar o Editor de Robôs para ver todos eles.

Nesta seção, vamos usar o Editor de Robôs para criar o seu próprio robô.

O Editor de Robôs

O primeiro passo é abrir o Editor de Robôs. Na tela principal do Robocode, clique no menu **Robot**, e selecione **Source Editor**. Quando aparecer a janela do editor, clique no menu **File** e selecione **New** e depois **Robot**.

Nos diálogos que seguem, digite um nome para seu robô e em seguida, dê um nome para o pacote onde os arquivos do seu robô serão criados. Por fim o editor vai mostrar o código base do robô.

Um Novo Robô

Deve aparecer pra você um código parecido como este:

```
package Robo1;
import robocode.*;
//import java.awt.Color;

// API de Ajuda: http://robocode.sourceforge.net/docs/robocode/robocode/Robot.html

/**
 * Robo1 - criado por (seu nome aqui)
 */
public class Robo1 extends Robot
{
	/**
```

```
	 * run: Comportamento Padrão de Robo1's
	 */
	public void run() {
		// A inicialização do robô deve ser colocada aqui

		// Depois de experimentar seu robô, tente descomentar os imports no topo
		// e a próxima linha:

		// setColors(Color.red,Color.blue,Color.green); // body,gun,radar

		// Loop principal do robô
		while(true) {
			// Substitua as próximas 4 linhas com qualquer comportamento que
			// você desejar
			ahead(100);
			turnGunRight(360);
			back(100);
			turnGunRight(360);
		}
	}

	/**
	 * onScannedRobot: O que fazer quando você vê um outro robô
	 */
	public void onScannedRobot(ScannedRobotEvent e) {
		// Substitua a próxima linha com o comportamento padrão que você desejar
		fire(1);
	}

	/**
	 * onHitByBullet: O que fazer quando você for atingido por uma munição
	 */
	public void onHitByBullet(HitByBulletEvent e) {
```

```
		// Substitua a próxima linha com o comportamento padrão que você desejar
		back(10);
	}

	/**
	 * onHitWall: O que fazer quando você atingir uma parede
	 */
	public void onHitWall(HitWallEvent e) {
		// Substitua a próxima linha com o comportamento padrão que você desejar
		back(20);
	}
}
```

O que significa alguns desses códigos?

`import robocode.*;` - Esse trecho diz ao Java que você deseja usar funções da classe Robocode em seu robô.

`public class MyFirstRobot extends Robot` - Diz Java: "O objeto que eu estou descrevendo aqui é um tipo de robô, chamado MyFirstRobot".

`public void run() { }` - O jogo chama o método **run** quando a batalha começa.

`{ }` - "As chaves" (**{ }**) agrupam comandos dentro de um mesmo método ou classe. Neste caso elas estão agrupando todo o código para o robô.

Continuando a análise

Observe o funcionamento do método run:

```
while(true) {

	ahead(100);
	turnGunRight(360);
	back(100);
	turnGunRight(360);
```

```
}
```

`while(true) { }` - Significa: “Enquanto a condição true é verdadeira, faça tudo o que está entre as chaves {}”.

Uma vez que true é sempre verdadeiro, isso significa: Faça o que está entre chaves sempre.

Portanto, este robô irá:

- Avançar 100 pixels
- Virar o canhão 360º para a direita
- Recuar 100 pixels
- Virar o canhão 360º para a direita

O robô vai seguir fazendo isso sempre, até que ele seja destruido, devido à instrução while(true).

Não é tão ruim, certo?

Fogo à Vontade!

Quando nosso Radar detectar um robô, queremos fogo:

```
public void onScannedRobot(ScannedRobotEvent e) {
     fire(1);
 }
```

O jogo chama seu método **onScannedRobot** sempre que – durante uma das ações – você vê um outro robô. Essa função nos diz várias informações importantes sobre o robô - seu nome, quanto de vida ele tem, sua posição, onde está dirigindo, sua velocidade, entre outras funções.

Contudo, uma vez que este é um robô simples, não vamos olhar para essas funções agora. Vamos apenas atirar!

Compilando seu Robô

Primeiro salve seu robô selecionando a opção **Save** no menu **File**. Siga as instruções para salvar o seu robô.

Agora compile ele selecionando **Compile** no menu **Compiler**.

Se o seu robô compilar sem nenhum erro, você pode iniciar uma nova batalha com ele. Feche a janela do editor e inicie uma nova batalha, dessa vez seu robô vai aparecer na lista para ser escolhido.

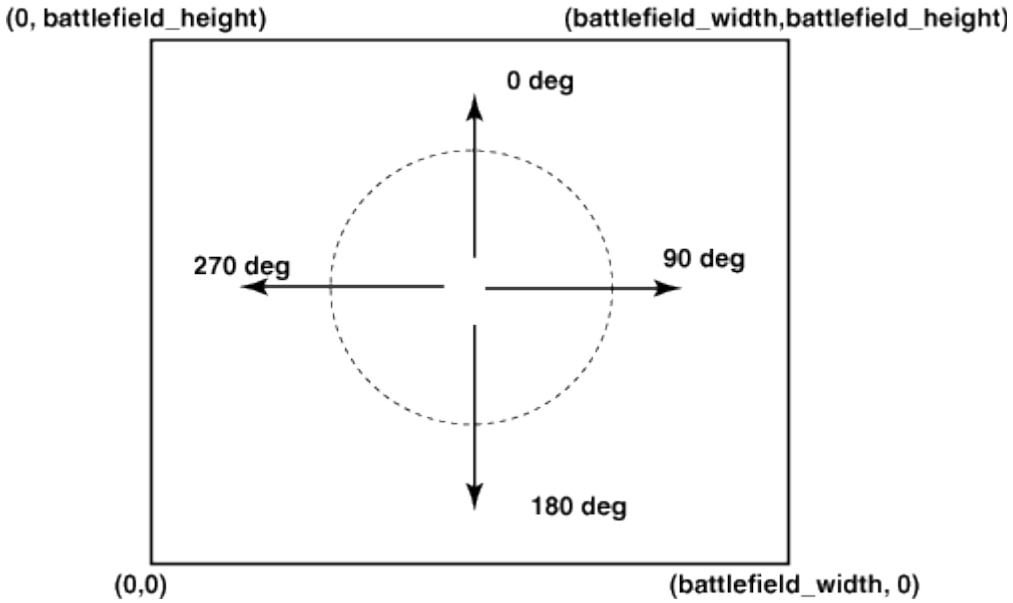
Aproveite e após a batalha volte ao editor e abra o código dos outros robôs para ver como certas coisas são feitas.

Física do Jogo

Coordenadas e Convenções de Direções

Sistema de Coordenadas: Robocode está usando o sistema de coordenadas cartesianas, o que significa que a coordenada (0, 0) está localizada no canto inferior esquerdo do campo de batalha.

Sentido Horário: Robocode está usando a convenção de sentido horário, onde 0/360 graus é no sentido "Norte", 90 graus no sentido "Leste", 180 graus no sentido "Sul", e 270 graus no sentido "Oeste".



Medições de Tempo e Distância em Robocode

Tempo (t): O tempo no Robocode é medido em “ticks”. Cada robô recebe um turno por tick.

1 tick = 1 turn

Medida de Distância: As unidades do Robocode são basicamente calculadas em pixels, com duas excessões. Primeiro, todas as distâncias são medidas com dupla precisão(*double*), então você atualmente pode mover uma fração de um pixel. Segundo, Robocode dimensiona automaticamente as batalhas para baixo pra caber na tela. Neste caso, a unidade de distância é atualmente menor do que um pixel.

A Física Por Trás dos Movimentos dos Robôs

Aceleração (a): Os robôs aceleração à taxa de 1 pixel/turno. Robôs desaceleram à taxa de 2 pixels/turno. O Robocode determina a aceleração pra você, com base na distância que você está tentando se mover.

Equação da Velocidade (v): v = a*t . A velocidade nunca pode exceder 8 pixels/turno. Note que tecnicamente, a velocidade é um vetor, mas no Robocode simplesmente assume-se a direção do vetor como sendo a frente do robô.

Equação da Distância (d): d = v*t . Isso é, a distância percorrida é proporcional a velocidade vezes o intervalo de tempo.

Robô, Canhão e Rotação do Radar

Taxa Máxima de Rotação do Robô: (10 - 0.75 * abs(velocidade)) graus / turno. Quanto mais rápido você estiver se movendo, mais devagar será para fazer curvas.

Taxa Máxima de Rotação do Canhão: 20 graus / turno. Esta é adicionada à taxa atual de rotação do robô.

Taxa Máxima de Rotação do Radar: 45 graus / turno. Esta é adicionada à taxa atual de rotação do canhão.

O Projétil

Dano: 4 * Potência do Disparo. Se a Potência do Disparo > 1, ele faz um dano adicional = 2 * (Potência – 1).

Velocidade: 20 – 3 * Potência do Disparo.

Temperatura do Canhão: 1 + Potência do Disparo / 5. Você não pode disparar se a Temperatura do Canhão > 0. Todas os canhões estão quentes no início de cada rodada.

Energia Retornada no Disparo: 3 * Potência do Disparo.

Colisões

Com outro Robô: Cada robô toma 0.6 de dano. Se um robô esta se afastando da colisão, ele não será interrompido.

Com uma Parede: AdvancedRobots tomam abs(velocidade) * 0.5 - 1; (Nunca < 0).

Robocode API

Essa seção vai tratar de algumas das funções principais usadas no jogo, para ver todas as funções disponíveis acesse a Robot API:
http://robocode.sourceforge.net/docs/robocode/

Métodos

`ahead(double distancia) : void` - Move o robô à frente, pela distância, em pixels, passada como parâmetro

`back(double distancia) : void` – Move o robô para trás, pela distância calculada em pixels

`doNothing() : void` – Não fazer nada nesse turno, significa que o robô irá ignorar sua vez

`fire(double intensidade) : void` – Dispara imediatamente um projétil. A intensidade do disparo é subtraído da energia do robô. Por isso, quanto maior a intensidade você pretende gastar com o disparo, maior também será a energia retirada do seu robô.

O disparo fará dano (4 * intensidade) se acertar outro robô. Se a energia for maior que 1, ele vai fazer um dano adicional de 2 * (intensidade - 1). Você recupera (3 * Intensidade) de energia de volta se você acertar o outro robô. Você pode chamar `Rules.getBulletDamage : double` para obter o dano que um disparo com uma intensidade específica vai fazer.

A intensidade do disparo especificado deve estar entre Rules.MIN_BULLET_POWER e Rules.MAX_BULLET_POWER.

Note-se que o canhão não pode ser disparado se estiver superaquecido, o que significa que getGunHeat () vai retornar um valor> 0.

Exemplo:

```
  // Disparar uma bala com potência máxima se a arma está pronta
if (getGunHeat() == 0) {
     fire(Rules.MAX_BULLET_POWER);
}
```

`getEnergy() : double` - Retorna o nível atual de energia do robô

`getGunCoolingRate() : double` – Retorna a taxa de resfriamento da arma, por turno

`getGunHeading() : double` – Retorna a direção que a arma do robô está apontando, em graus

`getGunHeat() : double` – Retorna a temperatura atual da arma

`getHeading() : double` – Retorna a direção que o chassi do robo está apontando, em graus

`getNumRounds() : int` - Retorna o número de rounds na batalha atual

`getNumSentries() : int` – Retorna quantos robôs sentinela estão à esquerda na rodada atual

`getOthers() : int` - Retorna o número de oponentes que estão à esquerda na rodada atual.

`getRadarHeading() : double` - Retorna o ângulo em graus que o radar está virado.

`getRoundNum() : int` - Retorna o número do round atual

`getTime() : long` - Retorna o tempo de jogo do round atual

`getVelocity() : double` – Retorna a velocidade do robô, medida em pixel/turno

`getX() : double` – Retorna a posição X do robô. (0,0) é no canto inferior esquerdo

`getY() : double` – Retorna a posição Y do robô. (0,0) é no canto inferior esquerdo

`normalRelativeAngleDegrees(double angulo) : void` - Função muito utilizada para normalizar o ângulo de um dos componentes do robô em relação à um referencial.

`scan() : void` – Analisa em busca de outros robôs

`setAllColors(Color cor) : void` – Define todas as cores do robô para a cor passada como parâmetro

`setBodyColor(Color cor) : void` – Define a cor do chassi

`setBulletColor(Color cor) : void` – Define a cor dos projéteis

`setColors(Color corChassi, Color corCanhao, Color corRadar) : void` - Define as respectivas cores no robô

`setColors(Color corChassi, Color corCanhao, Color corRadar, Color corProjetil, Color corArcoRadar) : void` – Define as respectivas cores no robô

`setGunColor(Color cor) : void` – Define a cor do canhão

`setRadarColor(Color cor) : void` – Define a cor do radar

`setScanColor(Color cor) : void` - Define a cor do arco do Scanner do robô.

`turnGunLeft (double graus) : void`
Gira o canhão para a esquerda na quantidade informada

`turnGunRight (double graus) : void` -
Gira o canhão para a esquerda na quantidade informada

`turnLeft(double graus) : void` – Gira o chassi do robô para a esquerda na quantidade informada

`turnRadarLeft(double graus) : void` – Gira o Radar do robô para a esquerda na quantidade informada

`turnRadarRight(double graus) : void` – Gira o Radar do robô para a direita na quantidade informada

`turnRight(double graus) : void` – Gira o chassi do robô para a direita na quantidade informada

Eventos

É sensato conhecer todos os eventos para ter a mente mais aberta quando for começar a programar a inteligência de seu robô.

Os eventos são chamados quando acontece algo específico no decorrer do combate. Alguns deles te enviam, por parâmetro, dados do robô adversário em questão para você trabalhar com esses valores dentro do evento. Exemplo: se você digitar "e.getBearing()" dentro de algum evento que contém na classe da variável, enviada por parâmetro, o método "getBearing()" , como os tipos ScannedRobotEvent e HitRobotEvent, retornará o ângulo do robô inimigo em questão. Sendo que "e" é o nome da variável usada como parâmetro, que pode ser qualquer outro.

`run() : void`

É executado quando o round for iniciado.

Diferente do que muitos pensam, esse evento só será chamado novamente quando iniciar outro round. Por isso é muito comum e recomendado usar um loop infinito dentro dele, para que seu robô nunca fique parado quando não tiver sendo executado outro evento.

Exemplo:

```
public void Run() {
	do {
		turnRadarLeft(360);
		setAhead(200);
		turnRight(300);
	} while(true)
}
```

onScannedRobot(ScannedRobotEvent evento) : void

Executado quando o radar do seu robô encontra um adversário.

É um dos eventos mais importantes, pois é a única forma de saber a energia, a distância, o ângulo do seus inimigos para poder atira nele. A não ser se você colidir com outro robô, que já seria um outro evento.

Metodos da classe ScannedRobotEvent:

getName() : String - Retorna o nome do robô adversário scaneado.
getBearing() : double - Retorna o ângulo do robô adversário em relação ao seu robô
getDistance() : double - Retorna a distacia do robô adversário em relação ao seu robô.
getEnergy() : double - Retorna o nível de energia do robô adversário
getHeading() : double - Retorna o ângulo em graus do adversário em relação a tela.
getVelocity() : double - Retorna a velocidade do robô scaneado.

Exemplo:

```
public void onScannedRobot(ScannedRobotEvent inimigo) {
	double angulo = inimigo.getBearing();
	double distancia = inimigo.getDistance();

	if ( distancia < 200 ) {
		turnGunRight(angulo);
		fire(2);
	}
}
```

Observação: Não confunda "getEnergy()" com "e.getEnergy()", pois o primeiro é a energia de seu robô e o outro a energia do robô scaneado.

onWin() : void

É executado quando seu robô ganha o round.

Já que aqui o round terminou, aproveite para programar uma risadinha, uma dancinha, malhando os derrotados ou para o seu robô parar de andar, evitando que bata na parede, perdendo energia.

Exemplo:

```
public void onWin(WinEvent e) {
	turnRight(36000);
```

```
}
```

onHitRobot(HitRobotEvent evento) : void

Este método é chamado quando seu robô colide com outro robô

Aproveite que você está bem perto do inimigo, vire o canhão para ele e mande um tiro de força máxima, porque dificilmente errará.

Métodos da classe HitRobotEvent

getName() : String - Retorna o nome do robô adversário colidido.
getBearing() : double - Retorna o ângulo do robô adversário em relação ao seu robô
getDistance() : double - Retorna a distacia do robô adversário em relação ao seu robô.
getEnergy() : double - Retorna o nível de energia do robô adversário
getHeading() : double - Retorna o ângulo em graus do adversário em relação a tela.
getVelocity() : double - Retorna a velocidade do robô scaneado.
isMyFault() : boolean - Retorna true se foi seu robô quem originou o evento, e false se foi o adversário que bateu em seu robô.

Exemplo:

```
public void onHitRobot(HitRobotEvent inimigo) {
	turnRight(inimigo.getBearing());
	fire(3);
}
```

onHitWall(HitWallEvent evento) : void

Este método é chamado quando seu robô colide com uma parede.

Quando seu robô bate na parede, perde energia. Então o mínimo que você deve fazer é mudar a direção dele, senão ele vai ficar de encontro à parede até perder toda a sua energia.

Métodos da classe HitWallEvent:

getBearing() : double - Retorna o ângulo da parede batida em relação ao seu robô

Exemplo:

```
public void onHitWall(HitWallEvent e) {
	turnLeft(180);
}
```

onHitByBullet(HitByBulletEvent evento) : void

Este método é chamado quando seu robô leva um tiro

Se a estratégia do seu robô é ficar parado enquanto atira, é bom utilizar esse evento para sair do local de onde acabou de levar um tiro, para você não se tornar um alvo fácil.

Métodos da classe HitByBulletlEvent:

getName() : String - Retorna o nome do robô adversário que te acertou o tiro.
getBearing() : double - Retorna o ângulo do robô adversário em relação ao seu robô
getHeading() : double - Retorna o ângulo em graus do adversário em relação a tela.
getPower() : double - Retorna a força do tiro.

Exemplo:

```
public void onHitByBullet(HitByBulletEvent e) {
	ahead(100);
}
```

onBulletHit(BulletHitEvent evento) : void

Este método é chamado quando seu tiro atinge um adversário.

Métodos da classeBulletHitEvent:

getName() : String - Retorna o nome do robô adversário atingido
getEnergy() : double - Retorna o nível de energia do robô adversário

Exemplo:

```
public void onBulletHit(BulletHitEvent e) {
	acertos++;
}
```

onBulletMissed(BulletMissedEvent evento) : void

Este método é chamado quando uma de seus tiros falha, ou seja, atinge uma parede.

Métodos da classeBulletMissedEvent

getBullet() : Bullet - Retorna o Bullet (dados do tiro), de seu robô, que bateu na parede.

Exemplo:

```
public void onBulletMissed(BulletMissedEvent e) {
	erros++;
}
```

onDeath() : Void

É executado se seu robô morrer.

Exemplo:

```
public void onDeath(DeathEvent e) {
	System.out.println(getName()+" morreu!");
	System.out.println("Quantidade de inimigos ainda vivos: "+getOthers());
}
```

Analisando Estratégias

Analisando o código de alguns dos robôs de exemplo:

Robô Fire

```
public class Fire extends Robot {
	int dist = 50; // cria uma distância padrão para se mover

	/**
	 * run:  Aqui estão as funções principais do Fire
	 */
	public void run() {
		// Seta as cores
		setBodyColor(Color.orange);
		setGunColor(Color.orange);
		setRadarColor(Color.red);
		setScanColor(Color.red);
		setBulletColor(Color.red);

		// Determina que o canhão seja girado lentamente... sempre
		while (true) {
			turnGunRight(5);
		}
	}

	/**
	 * Quando outro robô for detectado:  Atira!
	 */
	public void onScannedRobot(ScannedRobotEvent e) {
		// Se o outro robô está próximo, e ele tem bastante vida,
		// dispara intensamente!
		if (e.getDistance() < 50 && getEnergy() > 50) {
			fire(3);
		} // caso contrário, atira com intensidade 1.
		else {
			fire(1);
		}
		// Depois de atirar chama o radar novamente,
		// antes de girar o canhão
		scan();
	}

	/**
	 * Quando for atingido por um disparo:
	 * Gira perpendicularmente à bala e avança um pouco.
	 */
	public void onHitByBullet(HitByBulletEvent e) {
		turnRight(normalRelativeAngleDegrees(90 - (getHeading() -
e.getHeading())));
```

```
            ahead(dist);      //avança
            dist *= -1;
            scan();
      }

      /**
       * Quando o robô bate em outro:  Ajusta a mira pra ele e dispara
       * intensamente
       */
      public void onHitRobot(HitRobotEvent e) {

      //É criada a variável turnGumAmt que recebe o valor do calculo
      //de quanto a mira do canhão deve ser ajustada.
      //Para encontrar o valor adequado de ajuste, é chamada a função
      //para normalizar um ângulo.
      //Nesse exemplo é feito um cálculo entre o ângulo entre os robôs somado
      //com o ângulo do adversário em relação a tela menos a inclinação do
canhão

            double turnGunAmt = normalRelativeAngleDegrees(e.getBearing() +
getHeading() - getGunHeading());

            turnGunRight(turnGunAmt);
            fire(3);
      }
}
```

Robô RamFire

```
public class RamFire extends Robot {
      int turnDirection = 1; // Variável usada para determinar o giro
      /**
       * run: O robô gira ao redor de seu eixo à procura de um alvo
       */
      public void run() {
            // Seta as cores
            setBodyColor(Color.lightGray);
            setGunColor(Color.gray);
            setRadarColor(Color.darkGray);

            while (true) {
                  turnRight(5 * turnDirection);
            }
      }

      /**
       * onScannedRobot:  Quando localiza um alvo ele mira e vai atrás
       */
      public void onScannedRobot(ScannedRobotEvent e) {

            if (e.getBearing() >= 0) {    //caso o ângulo seja maior do que 0
                  turnDirection = 1;      //ele gira em sentido horário
```

```
            } else {
                  turnDirection = -1;     //caso contrário, sentido anti-horário
            }

            turnRight(e.getBearing());
            ahead(e.getDistance() + 5); //calcula a distância pro inimigo e
avança
            scan(); // usa o radar novamente para localizar os oponentes
      }

      /**
       * onHitRobot:   Quando é atingido, o robô mira o seu oponente e atira
       * com intensidade
       */
      public void onHitRobot(HitRobotEvent e) {
            if (e.getBearing() >= 0) {    // verifica o ângulo do oponente
                  turnDirection = 1;      // e toma a direção
            } else {
                  turnDirection = -1;
            }
            turnRight(e.getBearing());

            // Faz o cálculo da intensidade necessária para enfraquecer o
oponente
            if (e.getEnergy() > 16) {
                  fire(3);
            } else if (e.getEnergy() > 10) {
                  fire(2);
            } else if (e.getEnergy() > 4) {
                  fire(1);
            } else if (e.getEnergy() > 2) {
                  fire(.5);
            } else if (e.getEnergy() > .4) {
                  fire(.1);
            }
            ahead(40); // E avança para se chocar com ele
      }
}
```

Referências:

http://robocode.sourceforge.net/. Acessado em 25 de Maio de 2016

http://robowiki.net/. Acessado em 25 de Maio de 2016

https://sourceforge.net/projects/robocode/ . Acessado em 25 de Maio de 2016

http://www.robocodebrasil.com.br/ . Acessado em 25 de Maio de 2016

https://www.edivaldobrito.com.br/instalar-o-oracle-java-no-ubuntu/ . Acessado em 26 de Maio de 2016

http://compartilhartecnologia.blogspot.com.br/2011/05/verificando-versao-e-atualizando-o-java.html . Acessado em 26 de Maio de 2016

http://www.gsigma.ufsc.br/~popov/aulas/robocode/ . Acessado em 04 de Junho de 2016