Robocode – evolving your first robot

1 Before doing this tutorial

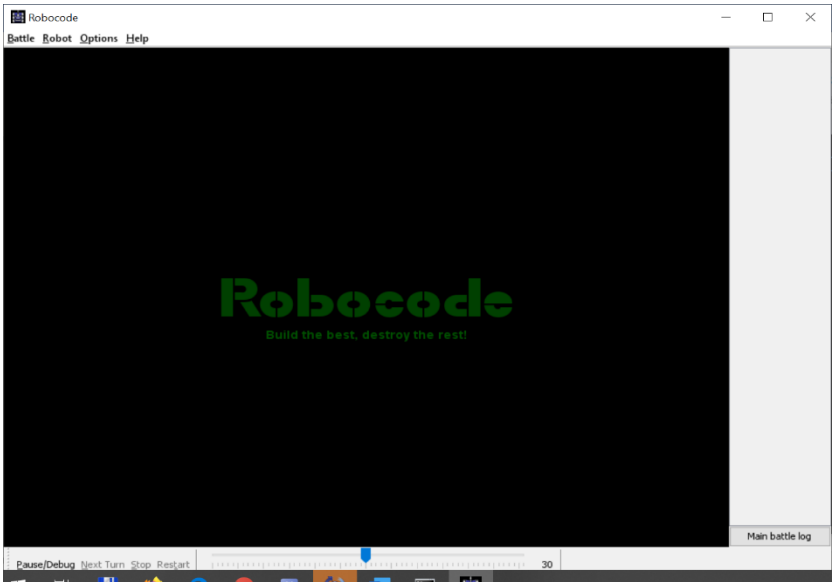
- Make sure you have installed Java and Robocode
- Make sure you have completed **Getting started with Robocode**

2 What is the plan for today?

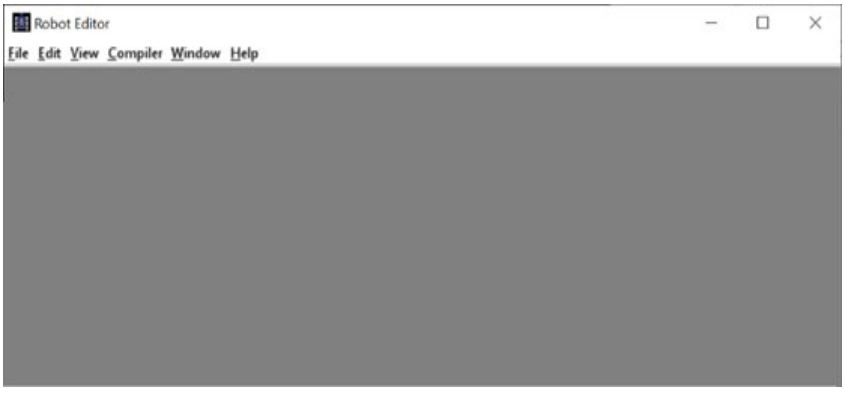
In our first session we made our first robot that can easily defeat a Target robot. Well, that wasn't hard was it? The target robot wasn't even fighting back. Today we will evolve our first robot and create a second robot. Our second robot will be able to easily defeat our first robot. No, wait… Our Day2 robot will easily be able to defeat 4 of our Day1 robots simultaneously!

3 Start Robocode and enter the code editor

- Click Start on your computer and type **Robocode**, then click on the Robocode app to start



- Now click on Robot > Source Editor to start the Java Editor



4 Create your Day2 Robot

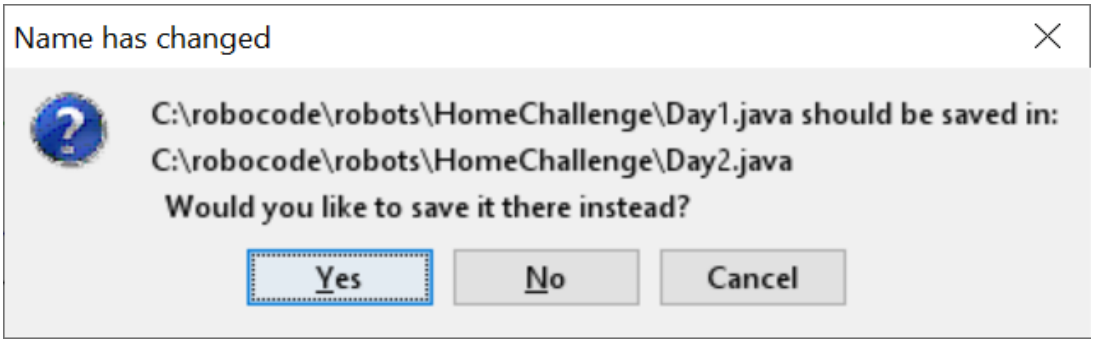
- Click **`File > Open`**
- Open the **`HomeChallenge`** folder and then Open **`Day1.java`**
- In the editor change the line

```
public class Day1 extends JuniorRobot
```

to

```
public class Day2 extends JuniorRobot
```

- Save the file and select Yes to save the new file.



5 Aiming faster, aiming better

We'll start by making 2 changes. One change will make the gun swing around faster. Only turning 1 degree at a time is very slooooooooow. We can double the rate at which we swing by simply changing the line in the **run()** method to:

```
turnGunRight(2);
```

Now we want to fix the code that tries to keep the gun pointed at the target. I don't know if you have noticed, but when we turn the gun left every time, we scan an enemy it sometimes turns off the target for a couple of shots. That is a waste of energy. So, we now want to stop the gun turning at all when we have an enemy in our scanner and only turn the gun when an enemy is not in our scanner. To do this we need variables!

A variable is a place where we can store a value that can be read by any other part of our code. So, when we have an enemy in our scanner, we can set a value to say that the enemy is being scanned, and then the **run()** method can check this value and only turn the gun if no enemy is currently being scanned.

Since there is no method to tell us when an enemy has left our scanners (when they have moved or have been destroyed) we will have to assume at the end of every turn that the enemy is no longer there, and then rely on the **onScannedRobot()** method to tell us every turn that the enemy is still there. Sounds difficult? It really isn't. Here is how we do it:

Firstly, we begin by defining a variable. In Java, a good place to define your variables is right under the opening { of your class, and before the first method begins. See the underlined code below and add it to your class:

```
public class Day2 extends JuniorRobot
{
	private boolean bScannedEnemy = false;

	public void run() {
```

Here is a quick explanation what everything means:

- **`private`** – only our robot can see this variable.
- **`boolean`** – this tells the computer what type of variable we want to store. For our purposes a Boolean is great because a Boolean can be one of two values: true or false.
- **`bScannedEnemy`** – this is the name of our variable. It can really be anything we want it to be. It is a good idea to start Boolean variable names with a b to remind us that this variable is a Boolean.
- = false; - we give the variable a default value of false. This makes sense because when the game starts, we have not scanned the enemy yet and we want the gun to turn.

Next we want to modify both the **run()** and the **onScannedRobot()** methods to use the new variable. I have added comments (beginning with a **`//`**) to the code to explain what I am doing. Make sure your **run()** and **onScannedRobot()** methods look like this too:

```
public void run() {
	// ! means NOT in Java.
	// If we have NOT scanned our enemy, turn the gun.
	if (!bScannedEnemy) {
		turnGunRight(2);
	}
	else {
		// assume we can no longer see our enemy.
		bScannedEnemy = false;
	}
}

public void onScannedRobot() {
	fire(1);
	// set the variable to true. This will stop the gun from
	                                                    turning.
	bScannedEnemy = true;
}
```

Every turn, while the enemy is in our scanners we will set the **bScannedEnemy** variable to true. At the end of every turn, the **run()** method will set it to false again. This will keep on going until the enemy is no longer being scanned, and the variable will stay **`false`** and the gun will start turning again.

Time to test our robot. Save your code and compile it. If there are no compile errors, you can go to the Robocode main screen and start a new battle. This time make sure there is a Day1 robot and a Day2 robot in the Selected Robots lists and run the battle a couple of times. You should notice:

- Our gun is turning faster, finding the enemy quicker.

- It also stays on target better, meaning we win more battles.
- However, if we start fighting against more than one robot we will lose because we are not moving. Time to….

6 Run away!!

When we get hit by an enemy bullet, we will need to run away to avoid more damage. The strategy we will adopt is to run forward until we hit a wall. Then if we get hit again, we will run backwards until we hit a wall. We will keep on doing this over and over again and hope that our faster and more accurate aiming and our movement will make us even better. Time for a new variable:

```
private boolean bScannedEnemy = false;
private boolean bGoForward = true;
```

Add the underlined code to your Robot. This is another Boolean that will tell our Robot to go forwards or backwards. We start by setting it to true, which tells the robot to run forwards.

Next we will change our **`onHitByBullet()`** method to look like this:

```
public void onHitByBullet() {
if (bGoForward) {
            ahead(1000);
      }
      else {
            back(1000);
      }
}
```

This is a pretty simple method. If our **bGoForward** variable is true, move forwards 1000 pixels. Else move backwards 1000 pixels. Simple. One more change, and this if for the **onHitWall()** method:

```
public void onHitWall() {
      bGoForward = !bGoForward;
}
```

This code is pretty simple too. Whenever we hit the wall, we set the **bGoForward** variable to the opposite value of what it was. If it was true, then NOT true is false. If it was false, then NOT false is true. (Remember: **!**, the exclamation point, means NOT in Java.)

Time to do another battle or two. Save and compile your code again, fixing any errors you may have made. Go back to the Robocode main window and start another battle pitting a Day1 robot against your Day2 robot. Run the battle a couple of times.

As you can see, the **`Day2`** robot does a pretty good job at avoiding the **`Day1`** robot's attack. There is just one more thing that would really make our **`Day2`** robot even more deadly and allow us to take on multiple **`Day1`** robots at the same time…

7 Know your enemy's coordinates

Warning! The next part is tricky and requires some tricky mathematics. You normally only learn this mathematics in 8th grade, so your parents may have to explain this to you. Or, you will just have to take our word for it.

Every time we scan a robot, we know it's there. Also, we know that the **`Day1`** robots do not move. It seems like such a waste of time to rotate the gun around after we have moved. So, wouldn't it be great if we can point our gun directly at the enemy's last position after we have stopped moving? That is exactly what we are going to do now. To do this we need to know how to do two things. Work out where on the map our enemy is (also called the enemy's coordinates) and know how to turn the gun to exactly where the enemy is. Let's do it!

X and Y Coordinates: Your position on the battle map is measured in X and Y pixels. The bottom left of the map is where both X and Y is 0, in other words the coordinates for the bottom left is (0,0). As you move to the right, your X values increases, so if you move 100 pixels to the right, your coordinates will be (100,0). If you move to the top of the map, your Y coordinates increase, so if you move 50 pixels towards the top your new coordinates will be (100,50).

JuniorRobot variables: Time to learn about some variables that we can use. These variables are part of the **JuniorRobot** code, and our robot extends the **JuniorRobot**. This means we have access to all the methods and variables that the **JuniorRobot** has access to. The ones we are going to use are:

- **`heading`** – the direction (in degrees with north being 0) that our tank is pointing
- **`scannedBearing`** – the angle of the robot that was scanned relative to the body of our robot
- **`scannedDistance`** – how far away (in pixels) the scanned robot is to us
- **`robotX`** – our current X coordinate
- **`robotY`** – our current Y coordinate

Now we know about these variables, we have all the ingredients to improve our robot.

First, we will import some code into our robot. The nice thing about java is that you don't have to write all the code yourself. Someone has probably written some useful code that you can use. We will import some code called **`Point2D`** that will help us do things like work out distances between two things. At the top of your code add the following underlined code:

```
package HomeChallenge;
import robocode.*;
import java.awt.geom.Point2D;
```

Now we will define two more variables. We will use these variables to store the X and the Y coordinate for our enemy. We will set both of them to -1 which will tell our code that we haven't scanned an enemy yet. We will use a new type of variable called an **`int`**. An **`int`** (short for integer) is used to store a number that is not a fraction, like 0, or 100 or -27. We will start the variable name with an i to remind ourselves that it is an int. Add the following underline code to our variables:

```
	private boolean bScannedEnemy = false;
	private boolean bGoForward = true;
	private int iEnemyX = -1;
	private int iEnemyY = -1;
```

We start by working out the X and Y coordinate of the enemy tank we have scanned. And we have to do this in our **onScannedRobot()** method:

```
	public void onScannedRobot() {
		// work out the X and Y coordinates for the enemy
		double angle = Math.toRadians((heading + scannedBearing)
								% 360);
		iEnemyX = (int)(robotX + Math.sin(angle) *
							scannedDistance);
		iEnemyY = (int)(robotY + Math.cos(angle) *
							scannedDistance);

		fire(1);
		bScannedEnemy = true;
	}
```

Every time we scan an enemy, we will calculate its X and Y coordinate and store it in the **`iEnemyX`** and **`iEnemyY`** variables!

Before we write the code to turn the gun, we need to write a whole new method. This method is used to calculate the angle (in degrees) between two coordinates. We will use it to calculate the angle for the gun based on the coordinates of our robot and the enemy's coordinates. Copy the next method and paste it in **right before the last }** in your code at the end of the file.

```
// computes the absolute bearing between two points
double absoluteBearing(double x1, double y1, double x2, double y2) {
	double xo = x2-x1;
	double yo = y2-y1;
	double hyp = Point2D.distance(x1, y1, x2, y2);
	double arcSin = Math.toDegrees(Math.asin(xo / hyp));
	double bearing = 0;

	if (xo > 0 && yo > 0) { // both pos: lower-Left
		bearing = arcSin;
	} else if (xo < 0 && yo > 0) { // x neg, y pos: lower-right
		bearing = 360 + arcSin; // arcsin is negative here, actuall 360 - ang
	} else if (xo > 0 && yo < 0) { // x pos, y neg: upper-left
		bearing = 180 - arcSin;
	} else if (xo < 0 && yo < 0) { // both neg: upper-right
		bearing = 180 - arcSin; // arcsin is negative here, actually 180 + ang
	}

	return bearing;
}
// the next AND ONLY character should be the last } in your code!
```

Make sure you copy all the code above and paste it in right before the final } in the bottom of your code.

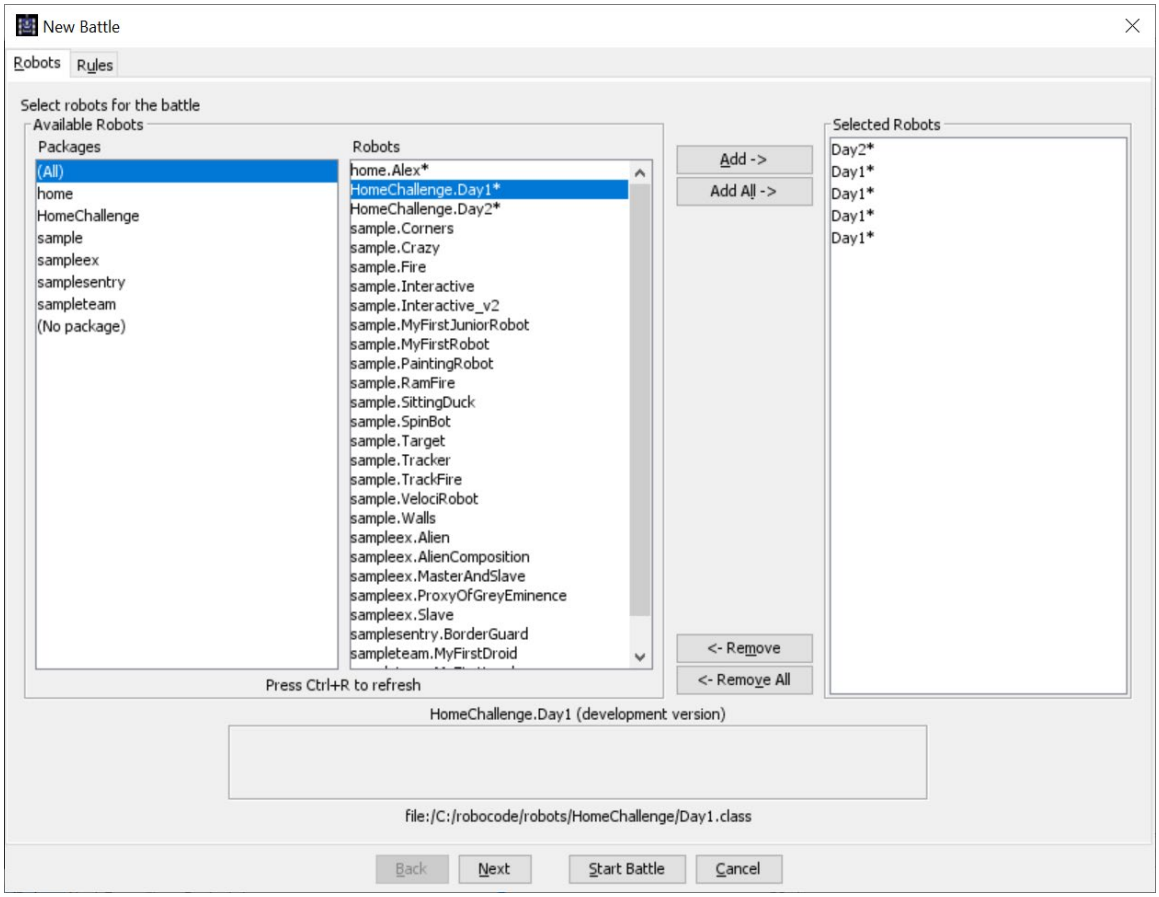
Now we will use this new method we have created to turn the gun. We will turn the gun as soon as we have hit a wall, so in the **`onHitWall()`** method. Add the following underlined lines to your code:

```
public void onHitWall() {
	bGoForward = !bGoForward;
	// if we have a known location for the enemy, point our gun at it.
	if (iEnemyX >= 0) {
		turnGunTo((int)absoluteBearing(robotX, robotY,
						iEnemyX, iEnemyY));
	}
}
```

Save and compile your code. After you have fixed any errors it is time to test the new robot. Start a new battle between a **Day1** robot and a **Day2** robot. Run it a couple of times. Can you see how efficient our new robot is?

8 4 against 1 – What an unfair battle!

Now we will let our Day2 robot battle 4 Day1 robots. Start a new battle, but add 1 x Day2 robot and 4 x Day1 robots like this:



Start the battle. Run it a couple of times. How much better is our new Day2 robot?!

Can we make a robot that can beat our Day2 robot? We will find out next time.

9 More resources

Here is a link to the Robocode documentation, specifically all the methods and variables you have at your disposal for the JuniorRobot class:

https://robocode.sourceforge.io/docs/robocode/index.html?robocode/JuniorRobot.html