LINUX VOICE
TUTORIAL

ROBOCODE: ARTIFICIAL INTELLIGENCE WARFARE

BEN EVERARD

A computer game that's played by computers? Is this an open invitation to Skynet or the future (or both)?

WHY DO THIS?

- Satisfy your power cravings by programming a robot army.
- Learn Java in a well documented environment.
- Give yourself a chance to win a Linux Voice T-shirt.

The idea of creating robotic warriors is as old as science fiction. However, how hard would they be to make? Let's not worry about the physical things – that's a problem for hardware engineers – and take a look at the software side of things. *Robocode* is an environment for tank-based autonomous warfare. That means that rather than controlling the actions of your tank via a joypad or keyboard during the fight as you may in a regular computer game, you have to program it to behave in the way you want it to, then set it running. Before we look at how to program the tanks, let's first take a look at the *Robocode* environment.

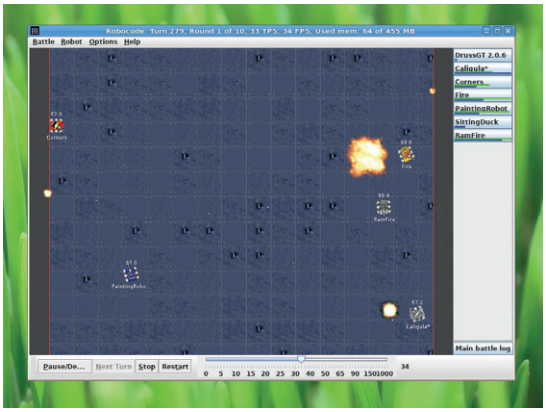
The main website is **http://robocode.sourceforge.net**, where you'll find loads of useful information that will come in handy if you decide to take on the challenge. There's also a download link that will take you to a SourceForge page where you can grab the JAR file. To use this, you'll need to have Java installed (we used OpenJDK 7, but other versions may also work).

Once it's downloaded, run:

```
java -jar robocode-1.9.2.3-setup.jar
```

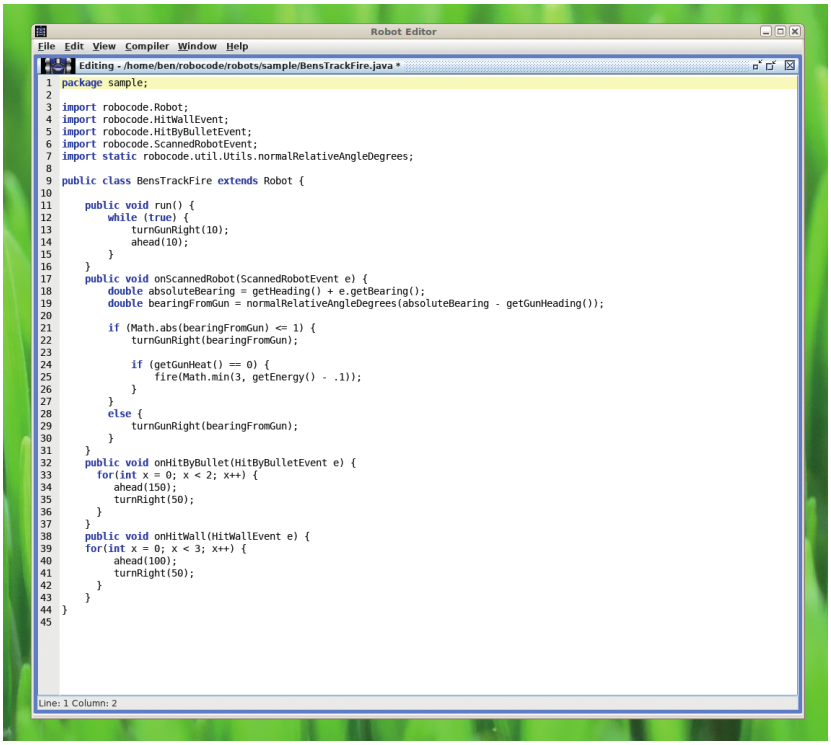
This will pull down all the files you need, and create a directory for them to live in. Once it's finished, you can **cd** into this directory (by default it will be **~/robocode**), and start the software with:

```
./robocode.sh
```



Let battle commence! Watching battles at normal speed helps you understand how the battles are fought, but cranking up the speed lets you test your robots quickly.

Robocode comes with a range of sample robots, so the first thing is to run a few battles to see how it works, and what tactics are effective. Go to Battle > New to create a new battle. On the first tab, pick a few robots to fight it out (it doesn't really matter which ones; we find that about five is a good number of competitors). The second tab allows you to change the settings for the battle (number of rounds, size of battlefield, etc). You can leave these as the defaults. Press Start Battle to begin.




```
package sample;

import robocode.Robot;
import robocode.HitWallEvent;
import robocode.HitByBulletEvent;
import robocode.ScannedRobotEvent;
import static robocode.util.Utils.normalRelativeAngleDegrees;

public class BensTrackFire extends Robot {

    public void run() {
        while (true) {
            turnGunRight(10);
            ahead(10);
        }
    }
    public void onScannedRobot(ScannedRobotEvent e) {
        double absoluteBearing = getHeading() + e.getBearing();
        double bearingFromGun = normalRelativeAngleDegrees(absoluteBearing - getGunHeading());

        if (Math.abs(bearingFromGun) <= 1) {
            turnGunRight(bearingFromGun);

            if (getGunHeat() == 0) {
                fire(Math.min(3, getEnergy() - .1));
            }
        }
        else {
            turnGunRight(bearingFromGun);
        }
    }
    public void onHitByBullet(HitByBulletEvent e) {
      for(int x = 0; x < 2; x++) {
        ahead(150);
        turnRight(50);
      }
    }
    public void onHitWall(HitWallEvent e) {
    for(int x = 0; x < 3; x++) {
        ahead(100);
        turnRight(50);
      }
    }
}
```

The *Robocode* IDE is fine for simple robots, but if you're creating a more complicated warrior, you may prefer to use a more fully-featured IDE.

Know your robots

You should see that each robot uses different tactics to attack the others. Some (like RamFire) are really aggressive, while others (like Corners) are more defensive. Essentially, there are two parts to a robot's strategy. It needs to avoid the shells from other tanks, and it needs to hit other tanks with its shells.

Before we dive in and start writing our own robot, let's take a look at how a couple of others work. *Robocode* has its own built-in IDE, which you can use to create robots in Java. Got to Robot > Source Editor to open it, then Open > Sample > RamFire.java to take a look at how the RamFire robot works. It is (in slightly abridged form):

```
public class RamFire extends Robot {
        int turnDirection = 1;

        public void run() {
```

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


        public void onScannedRobot(ScannedRobotEvent e) {
            if (e.getBearing() >= 0) {
                turnDirection = 1;}
            else {
                turnDirection = -1;}
            turnRight(e.getBearing());
            ahead(e.getDistance() + 5);
            scan();
        }

        public void onHitRobot(HitRobotEvent e) {
            if (e.getBearing() >= 0) {
                turnDirection = 1;
            } else {
                turnDirection = -1;
            }
            turnRight(e.getBearing());
            fire(1);
            ahead(40);
}}
```

There are quite a few comments that we've removed for brevity. The first thing that surprised us was just how simple the code is. *Robocode* handles everything about the environment, so you only need to decide how your robot will react to the environment and code the AI.

Standard robots extend the **Robot** class (there are other classes such as **AdvancedRobot** you can use as well), and then override the appropriate methods. In the case of RamFire, there are only three methods. **Run** is triggered at the start of the battle and is used to define the tank's default behaviour. In this case, if nothing else happens to the robot, it will simply spin around on the spot.

OnScannedRobot is triggered when the robot's scanner pics up another robot. Control of the radar (which picks up other tanks when scanning) can be a complex topic, but in simple terms, it will just look in the general area the turret is facing. Because the tank will continually spin until directed otherwise, it will always be on the look out for other tanks. This method simply points the tank at the scanned tank, and head straight for them.

When the robot rams into its victim, **onHitRobot()** will run. This fires the cannon at them (the code here is simplified), and it will ram them again.

As you can see, controlling your robot is all about understanding the *Robocode* API. This is fully documented at **http://robocode.sourceforge.net/docs/robocode**. You don't need to dive straight in and read it all, but as you develop fighting robots, you'll probably find yourself heading there more and more frequently.

RamFire works by aggressively pursuing robots – a tactic that can work, but can also incur damage for the RamFire robot itself. Now let's take a look at a more cautious robot. TrackFire also continuously looks for enemies, but rather than ramming them, it just shoots at them:

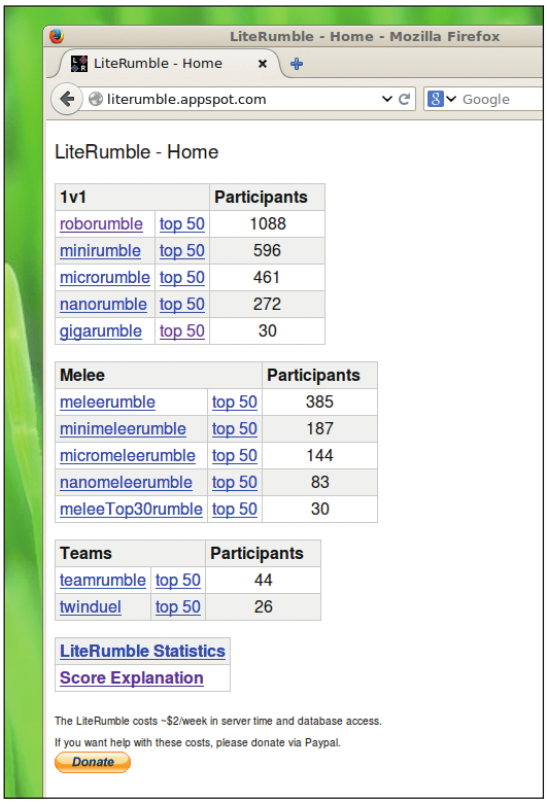
```
public class TrackFire extends Robot {
    public void run() {
        while (true) {
            turnGunRight(10);
        }
    }
    public void onScannedRobot(ScannedRobotEvent e) {

        double absoluteBearing = getHeading() + e.getBearing();
        double bearingFromGun = normalRelativeAngleDegrees(a
bsoluteBearing - getGunHeading());
```

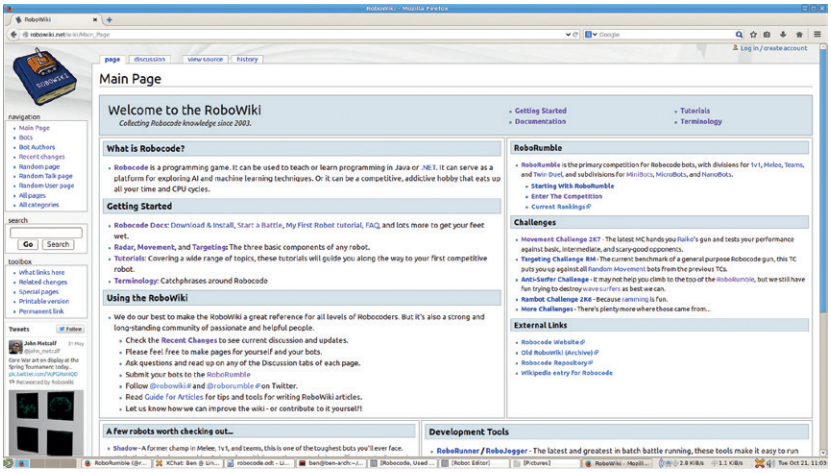
Advanced tactics

Robots vary in complexity from the simple (like the ones we're looking at) to the mind-boggling complex. Fortunately, you don't have to start out from scratch when building your robot. There's lots of information on tried and tested tactics on the *Robocode* wiki (**http://robowiki.net/wiki/Main_Page**). Here are a few of our favourite tactics:

- **Energy drop tracking** Robots can't see when bullets are fired, but they can detect the amount of energy other robots have. If that energy drops suddenly, it's usually because the other robot has fired a bullet. This can be useful to other tactics such as wave surfing and bullet shielding.
- **Wave surfing** If you see a robot fire a bullet (by energy drop tracking), you don't know what direction it's gone in, but you can predict a range of directions, and work out the rough probability of each. These areas of probability ripple outwards like waves from a stone dropped in a lake. Wave surfing is the process of getting in the place of the lowest probability on this wave. There's more details at: **http://robowiki.net/wiki/Wave_Surfing_Tutorial**
- **Bullet Shielding** The aim of bullet shielding is to protect your robot by shooting your opponents' bullets out of the air. That's quite easy to say, but much harder to achieve in practice. See **http://robowiki.net/wiki/Bullet_Shielding** for the maths.



There's an ongoing *Robocode* competition at **http://literumble.appspot.com**. If you think your creation is up to the challenge, you can set it against the rest of the world and see how it matches up.



The robot wiki is an invaluable source of information and inspiration for all things *Robocode*. **http://robowiki.net/wiki/Main_Page**

```
        if (Math.abs(bearingFromGun) <= 3) {
            turnGunRight(bearingFromGun);
            if (getGunHeat() == 0) {
                fire(Math.min(3 - Math.abs(bearingFromGun),
getEnergy() - .1));
            }
        }
        else {
            turnGunRight(bearingFromGun);
        }

        if (bearingFromGun == 0) {
            scan();
        }
    }
}
```

As you can see, this is quite simple. Rather than spinning the robot, it just turns the gun, and rather than speeding off towards its victim, it just points the cannon at them and fires.

There is a little complication with firing because you (as the programmer) can decide how much power to give the gun. The more power (the parameter to the **fire** command), the more damage it potentially does your enemy, but the more it also heats up your gun. If your gun overheats, you can't fire until it cools down.

A good general tactic is to fire with more power the more confident you are of hitting your enemy. In this example, we'll fire if the angle between where the gun is pointing, and the tank we're trying to hit is less than three degrees. However, the smaller the angle is, the more power we'll give the shot. This isn't necessarily optimal because it doesn't attempt to work out whether the target is moving.

If you watch TrackFire fight a few battles, you'll probably notice that the biggest problem it has is that it can be a sitting duck. Once it turns up in an enemy's scan, it is usually destroyed fairly quickly because it doesn't get away.

Now you've seen a few robots in operation, and seen how a couple are written, it's time to start building one. We decided to base ours on the TrackFire robot, but give it a bit more motion and the ability to run away.

To start with, then, we just need to update the parts that identify the robot, so open the TrackFire robot in the source editor if you haven't already, and change the **public class** line to:

```
public class LVTrackFire extends Robot {
```

Then go to File > Save As and save it under a new name (it should suggest **LVTrackFire.java**, which is fine). You now have a new robot. To make sure everything's gone properly, go to Compiler > Compile, then (if there are no errors), go to the main *Robocode* window and create a new battle. You should find your robot under Sample.

Added intelligence

Of course, at this stage, the AI for LVTrackFire is exactly the same as TrackFire, so let's now go back and add some more intelligence to it.

The biggest problem TrackFire has is that it stays still. We'll add a bit of movement logic to our robot in a few ways:

- Under normal operation, it will move forward slowly. This will help it avoid fire from robots far away.
- If it's hit, it will quickly get out of the way. This will hopefully move it out of the scan of the robot that's just hit it.
- If it hits a wall, it will get away. This stops it being pinned in by RamFire.

We can add this logic in stages. Firstly, we'll make it move forward under normal operation. To do this, just add one line to the **run** method as follows:

```
    public void run() {
        while (true) {
            turnGunRight(10);
            ahead(10);
        }
```

Famous fighters

The robots that come with *Robocode* are enough to keep you interested while you get started, but if you really want to test your skills, you may want to test yourself against some more powerful opposition. Alternatively, you may just want to watch some masterful killing machines at work. Whichever is your motive, you can find more robots online at **http://robowiki.net/wiki/Category:Bots**. A few of our favourites are:

- **DrussGT** (http://minifly.rchomepage.com/robocode/jk.mega.DrussGT_2.0.6.jar)
- **Diamond** (www.dijitari.com/void/robocode/voidious.Diamond_1.7.11.jar)
- **Demonic Rage** (http://sites.google.com/site/justinsitehere/file-cabinet/justin.DemonicRage_3.4.jar)

If you want to see some top *Robocode* action, put Demonic Rage in a ring with DrussGT and Diamond to see which comes out top. If you can build a robot that can compete with these three, you've done very, very well.

These are all open source, so you can poke around in their insides and see what makes them tick. Perhaps you'll find something to inspire your own battle bot.

```
        }
```

This keeps the gun spinning as before (and so keeps it scanning for other robots), but makes it move as well.

The next bit of code we want is to make the robot run away whenever it's hit. Again, we don't really care where it runs away to, we just want it get away from its current location. In this case, we could just move forwards a certain distance. This is what the first iteration of our robot did, but in a crowded battlefield, we found that this doesn't work very well. Our tank often got stuck when it hit other tanks and didn't run away successfully, so instead, we made it move forwards, turn, then move forwards again. This is done with the following method:

```
public void onHitByBullet(HitByBulletEvent e) {
    for(int x = 0; x < 2; x++) {
      ahead(150);
      turnRight(50);
    }
}
```

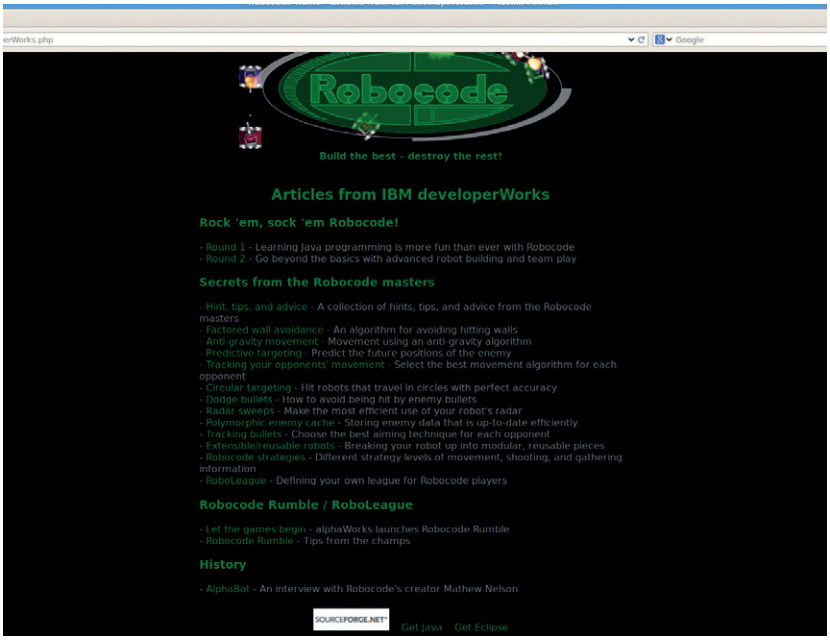
The final bit of movement (to keep the robot away from walls) is done with:

```
public void onHitWall(HitWallEvent e) {
   for(int x = 0; x < 3; x++) {
    ahead(100);
    turnRight(50);
   }
}
```

This is the mechanics robot complete. We were happy with TrackFire's firing, so we didn't change this. You can compile it and pit it against other robots in battle and see how it fares.

Although it's all done, there are still some bits you can tweak to make the robot more effective.

The speed at which our robot moves is governed by the number in **ahead(10)** in the **run** method. By increasing or decreasing this number, you can make it faster or slower. You can also adjust the accuracy of the aiming in the line:



IBM's Developer Works has a series on learning Java with *Robocode* at **http://robocode.sourceforge.net/developerWorks.php**. It's a little dated now, but should still work.

```
if (Math.abs(bearingFromGun) <= 3) {
```

Reducing the value **3** will mean the gun will wait until it has pointed more accurately before it fires. This will reduce the rate of fire, but increase the number of bullets that reach the target (since the target could be moving, it's not easy to aim accurately).

You can also adjust the firepower. None of these change the fundamental operation of the robot, but by tuning the parameters, you can make your killing machine more effective.

Our simple robot isn't the most devastating opponent, but hopefully it will have whetted your appetite to build your own AI tank. It should also prove a fairly easy starting point to surpass as you add more features, you should quickly be able to superseded this primitive AI. LV

Ben Everard is the best-selling author of *Learning Python With Raspberry Pi*. He hacks robots for fun.

COMPETITION

We're going to run a digital Battle Royale.

There are lots of open source robots available, so we're going to make our challenge a bit different to most *Robocode* competitions to discourage the use of these open source bots.

Firstly, your tank must extend the **Robot** class, not **AdvancedRobot** or any of the others available in *Robocode*. Second, the whole robot must come in at under 70 lines of Java. A line (for the purposes of this challenge) is a statement ended by a semicolon (not including the separate parts of a **for** loop), or a statement that starts a new code block (such as **if**, **else**, **while** etc). Parentheses may be put at the beginning or end of any line and don't need a separate line. Therefore, the shortest the following code can be is three lines:

```
for(int x = 0; x < 3; x++) {
  ahead(100);
  turnRight(50);}
```

Obviously, the best tactics will depend on how the battles are structured. In our war, the battle will take place in a series of heats. Each heat will have six robots (if the number of robots isn't divisible by six, sample robots will be included to bulk up the numbers). Each heat will be 100 rounds, and the top three will progress to the next round. We'll go through as many rounds as necessary to whittle the field down to six robots for the grand final. This will take place over 200 rounds and the winner of this will be crowned Linux Voice Robocode champion. The winner will, of course, get an exclusive Linux Voice winner's T-shirt. All the battles will take place on a 1000x800 battlefield with a gun cooling rate of 0.1, an inactivity time of 450 and a sentry size of 450.

All robots must be licensed under a OSI-approved open source licence (we recommend the GPL v3). The robot doesn't have to be entirely your own work: borrowing chunks of code from other openly licensed robots is allowed, as is using existing robots as a starting point for your development, although you obviously must adhere to the terms of the licence of any code you borrow.

Please email your robot's source code to **ben@linuxvoice.com** by 20 December 2014.