

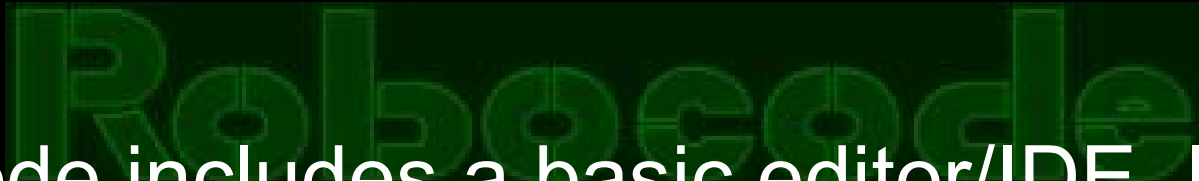


What is RoboCode?

- Java based educational game (don't worry – it's fun too...)
- Software controlled miniature battle tanks...
- ...fight to the death!
- Competition is turn based but fast paced
- Easy to learn...
- ...difficult to master!

Where do I get it?

- JDK (<http://java.sun.com/>)
- Robocode (<http://robocode.sf.net/>)



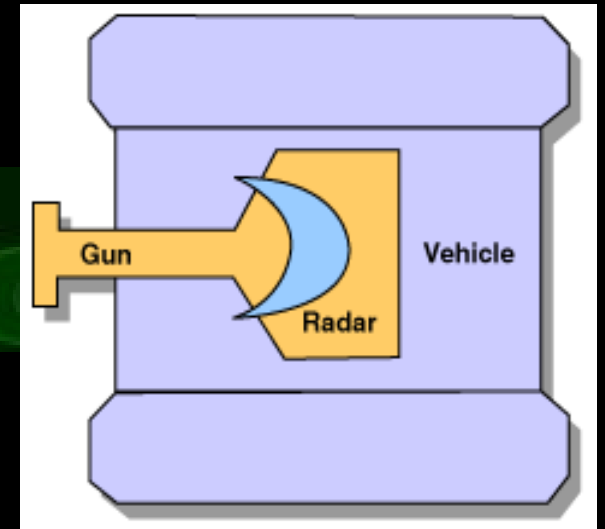
Robocode includes a basic editor/IDE. I recommend using a different one, such as:

- jEdit (<http://jedit.org/>)
- ConTEXT (<http://www.context.cx/>)

Included with robocode is the jikes Java compiler.

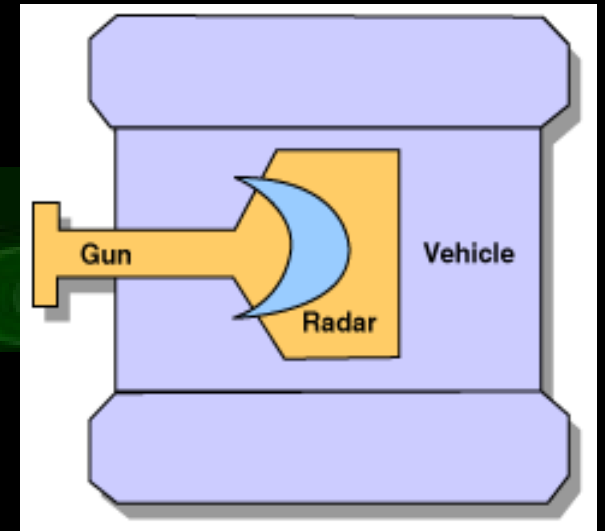
The Robot

- The robot consists of three parts – the vehicle itself, the gun and the radar
- These can be moved together or independently of each other
- The radar detects other robots as it sweeps the playing field – the robot's location is read
- The vehicle detects bullet hits and can be programmed to react to these – info on where the bullet came from can be attained



The Robot

- The radar must rotate to sweep the playing field
- The gun can fire a bullet with a power from 0.1 to 3.0. This power comes from your shields!
- High powered bullets travel slower than lower powered ones but do a lot more damage
- When a bullet hits its target your shield regains twice the power of that bullet.



Creating your first robot

- Open the Robot Editor (Robot -> Editor)
- Create a new robot (File -> New -> Robot)
- Give it a name (eg; HyperKillBot9000)
- Enter a package name
- The editor will open with a basic bot template...
change some stuff.
- When you think you have a winner save your robot and compile it (Compiler -> Compile)

Creating your first robot

- Close the Robot Editor
- Start a new battle (Battle -> New)
- Find your robot in the list and add it
- Add some others
- Click Start Battle and observe your robot's behaviour. If it didn't win it might need a tweak...
- If it did do a victory dance.

Compiling outside the RoboCode environment

- Robots are held inside subdirectories in `robocode/robots/`
- These subdirectories are the name of the robot's team or group (ie; you). If your team is called 'bob' your robot should be held in `robocode/robots/bob/`
- Robots for the `robocode.ie` competition may consist of only a single `.java` source file.

Compiling outside the RoboCode environment

It's possible to compile our robots outside the RoboCode environment by passing a classpath parameter to jikes.

This can be made even simpler by specifying a relative path to the jikes compiler from your robot's directory. You can insert this into a batch file/script.

Compiling outside the RoboCode environment

Examples:

Windows batch file:

```
..\..\jikes-1.22\bin\jikes.exe -deprecation -g -Xstdout  
+T4 -classpath "C:\Program  
Files\Java\jre1.5.0_06\lib\rt.jar";..\..\robocode.jar;..\.  
.\robots %1
```

GNU/Linux script (should also work on MacOSX):

```
#!/bin/sh  
../../jikes-1.22/bin/jikes -deprecation -g -Xstdout +T4  
-classpath /usr/lib/j2re1.5.0_06/lib/rt.jar:  
../../robocode.jar:../../robots $1
```

You can get the classpath from RoboCode's robot editor (Compiler -> Options -> Preferences)

The SpinningBanana Tutorial

NB: This tutorial does not contain the complete bot.

```
public void run() {  
  
    top = arenaHeight - (arenaHeight / 3);  
    bottom = arenaHeight / 3;  
    right = arenaWidth - (arenaWidth / 3);  
    left = arenaWidth / 3;  
  
    doomed = new Enemy(); // ;)  
  
    turnRadarLeft(Double.POSITIVE_INFINITY);  
}
```

Here we define a no-go zone, create an instance of the Enemy class for tracking and start the radar spinning. The standard method of the while(true) loop keeping the robot going is not used.

The SpinningBanana Tutorial

```
class Enemy {  
    String name;  
    public double bearing;  
    public double bearrads;  
    public double head;  
    public double headrads;  
    public double direction;  
    public double speed;  
    public double distance;  
    public double energy;  
}
```

The Enemy class is a useful way of keeping track of your opponent.

The SpinningBanana Tutorial

Inside onScannedRobot(ScannedRobotEvent e):

```
doomed.name = e.getName();
doomed.distance = e.getDistance();
if ((doomed.distance < lastDistance) ||
    (doomed.name.equals(lastName) || (getOthers() == 1))) {
    doomed.speed = e.getVelocity();
    doomed.head = e.getHeading();
    doomed.bearing = e.getBearing();
}
```

...

```
lastVelocity = doomed.speed;
lastDistance = doomed.distance;
lastName = doomed.name;
```

This is a method of targeting the closest bot and sticking to it. We don't target another until it comes closer than our current target or there are no other combatants

The SpinningBanana Tutorial

Inside `onScannedRobot(ScannedRobotEvent e)`:

```
if ((doomed.direction > 90) && (doomed.direction < 270)) {  
    bearingFromGun += doomed.speed * 1.4;  
}  
else {  
    bearingFromGun -= doomed.speed * 1.4;  
}
```

This inelegant targeting system has proved surprisingly effective despite its use of a hardcoded constant. It takes advantage of the fact that the bots in the standard robot class move at a constant speed.

It is the result of trial and error (or perhaps BFI). An alternative targeting system based on the enemy's lateral velocity also exists in SpinningBanana.

The SpinningBanana Tutorial

Inside onScannedRobot(ScannedRobotEvent e):

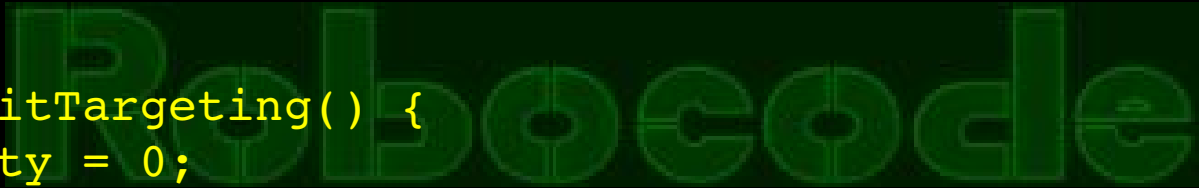
```
turnGunRight(bearingFromGun); // aim
fire(bulPower); // fire
turnLeft(90 - doomed.bearing); // strafe
if ((doomed.distance > 80) || (getOthers() > 1))
    move();
```

Since there's no move routine called in the run() method it's included in onScannedRobot as that event happens most often. We don't move in very close combat to make sure we spend more time firing.

bearingFromGun is the relative location of the target calculated earlier.

The SpinningBanana Tutorial

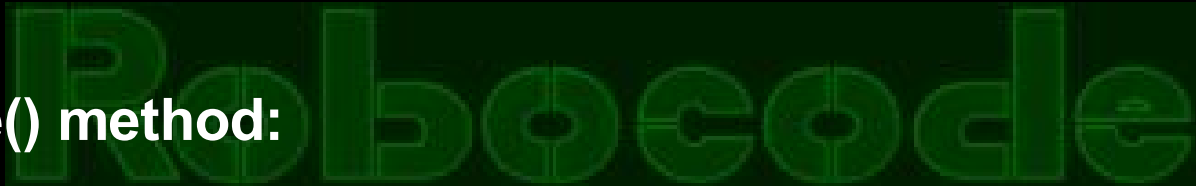
```
public void onRobotDeath(RobotDeathEvent e) {  
    if (e.getName().equals(lastName))  
        initTargeting();  
}  
  
public void initTargeting() {  
    lastVelocity = 0;  
    lastDistance = 10000;  
    lastName = "";  
}
```



When any robot dies we reinitialise our targeting system. The closest robot will be acquired as the new target.

The SpinningBanana Tutorial

```
public void onHitWall(HitWallEvent e) {  
    direction *= -1;  
    Random generator = new Random();  
    ahead(generator.nextInt(180) * direction);  
}
```



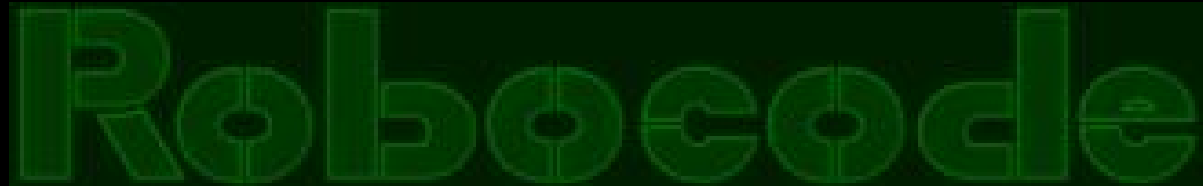
From the move() method:

```
if (Math.random() < .75) {  
    ahead(generator.nextInt(180) * direction);  
}  
else {  
    back(generator.nextInt(150) * direction);  
}
```

A global variable called direction exists. When we want to change direction we multiply it by -1 (so it is either 1 or -1). This is then multiplied by the amount we move making it a positive or negative number and dictating the direction.

Time is of the essence...

```
public void onHitByBullet(HitByBulletEvent e) {  
    // FIXME: leggit!!  
}
```

The Robocode logo is displayed in a dark green, stylized font. The letters are bold and have a slightly irregular, blocky appearance. The 'o's in 'Robocode' are particularly prominent, with the second 'o' being significantly larger than the first. The logo is centered horizontally and partially overlaps the code block above it.

It's important to keep things simple. You might not get time to implement every idea so focus on getting your best ideas to work.

General strategies

- Keep moving
- Don't shoot until target is acquired
- Avoid the centre of the arena in a melee
- Stick close in 1-on-1

Links

- <http://robocode.ie/> - Irish RoboCode competition
- <http://robocode.sf.net/> - RoboCode home
- <http://robowiki.net/> - RoboCode wiki
- <http://robocoderepository.com/> - More bots