



Robocode

An Introduction to the Assignment

..and learn Java :-)



Acknowledgement

*All notes in this chapter are based on an online article by Sing Li
(<http://www-106.ibm.com/developerworks/java/library/j-robocode/>).*

This chapter will not cover any topics associated with neural net learning. Its purpose is to acquaint the reader with the tools necessary to complete the coursework: Robocode, the java programming language and the Eclipse IDE.

Introduction to Robocode



- Robocode is a fun, easy to use battle simulator
- Robocode is from IBM (2001)
- Runs on the Java 2 platform
 - Originally meant as a Java teaching tool
 - Has quickly become a world wide game
 - ... and AI research tool!
- So this year - in 592, prepare to do battle!



September 2008

EECE 592 - Robocode



Robocode is an easy-to-use robotics battle simulator that runs on all platforms that support Java 2. Robocode is the brainchild of Mathew Nelson, a software engineer in the Advanced Technology, Internet division at IBM.

You create a robot, put it onto a battlefield and let it battle to the bitter end against opponent robots created by other developers. Each Robocode participant creates his or her own robot using elements of the Java language, enabling a range of developers, from rank beginners to advanced hackers to participate in the fun.

Assignment Summary

- Create your own robot
 - Give it intelligence
- Your 'bot must
 - use neural networks to train your 'bot
 - and reinforcement learning

The coursework this year requires each student to develop and submit their own Robocode robot. And because this course is about architectures for learning systems, it is required that you apply (and be able to demonstrate) what you learn in class, to your robot. As a minimum, you must use the error backpropagation algorithm as applied to multi-layer perceptrons **and** reinforcement learning in aspects of your robot's attack or defence strategy. Suggestions will be offered as to where and how to apply these learning methodologies.

Installing Robocode

- Go to <http://robocode.sourceforge.net/>
- Download robocode-setup.jar
- To install it, you need a Java VM
- Now type at the command line

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

September 2008

EECE 592 - Robocode

The latest version of Robocode can be found at
<http://robocode.sourceforge.net/>

The latest version of Robocode is v1.6.1.2 as of 12 September 2008. Note that this version requires JDK 1.5 on your machine.

Installation is straight forward. Just run `robocode-setup.jar` within your java VM.

Running Robocode

- Launch Robocode



- You should see this launch screen

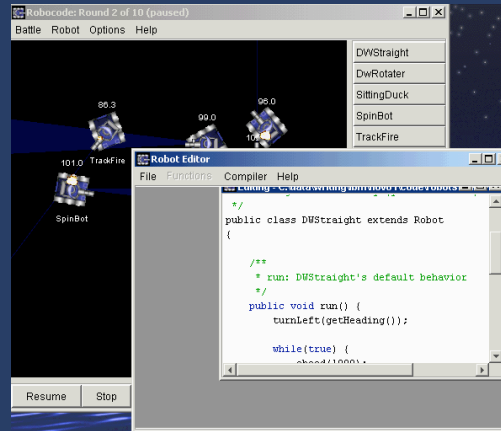
September 2008

EECE 592 - Robocode

After your installation, you can start the Robocode system from either the shell script (robocode.sh), batch file (robocode.bat), or icon on the desktop. When I tried, double clicking the icon worked fine on both Mac (OS X) and PC computers. Note that I could not get it to install on a Mac running 9.x.

The Robocode IDE

- With Robocode you get:
- A battlefield
- And an editor

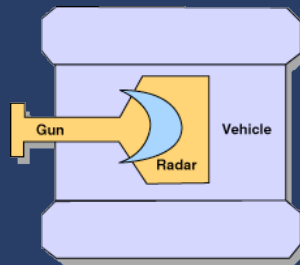


The battlefield is where the battle between the robots plays itself out. It houses the main simulation engine and allows you to create, save, and open new or existing battles. You can pause and resume the battle, terminate the battle, destroy any individual robot, or get the statistics of any robot using the controls available in the arena. Furthermore, you can activate the Robot Editor from this screen. The Robot Editor is a customized text editor for editing the Java source files that make up a robot. It integrates both the Java compiler (for compiling robot code) and the customized Robot packager in its menu. Any robot created with the Robot Editor and successfully compiled is in a ready-to-deploy location for the battlefield. A robot in Robocode consists of one or more Java classes. These classes can be archived into a JAR package. Robocode provides a "Robot Packager" that can be activated from the battlefield GUI window, for just this purpose.

Tip: Watching a battle unfold can be lengthy at times. If you would like to speed things along and are happy not to actually watch the battle, minimize the Robocode window. The battle continues but since the java vm no longer needs to generate a graphical output, you'll find it progresses much much faster! When it comes to reducing learning times, you'll find this very useful.

Robotank Features

- A Robocode Tank is
 - a mobile vehicle mounted with a
 - a gun upon which is mounted a
 - a radar



September 2008

EECE 592 - Robocode

Note that the robot has a rotating gun, and on top of the gun is a rotating radar. The robot vehicle, the gun, and the radar can all rotate independently. At any moment in time, the robot's vehicle, the gun, and radar can be turned in different directions. By default, these items are aligned, facing the direction of the vehicle movement.

You can also fix the direction of the gun turret and the radar for that matter, to always point in the direction the tank is heading. Might be useful if you're in pursuit!

The radar is able to detect any tank within a certain distance.

Robotank Features

- Battlefield size *800 x 600 pixels (default)*
- Tank Energy *100 initially*
- Tank Speed *8 pixels / turn*
- Turn Rate *20° / turn*
- Bullet Speed *20 pixels / turn (I think)*
- Gun Power *1 - 3 energy*
Note: bigger bullets are slower
- Scanning range *1200 pixels*
- Scanned data *distance, bearing, heading, name
energy, velocity*

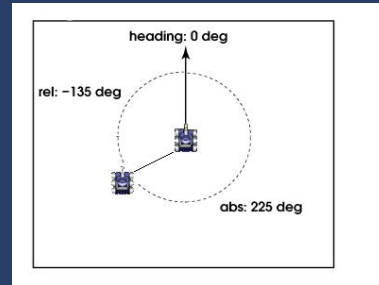
Note that with the default battlefield size of 800 x 600 pixels, the maximum distance between you and any other tank is 1000 pixels. This is within your scanning range of 1200 pixels. I.e. you can scan any tank in a battlefield of this size.

Some other useful information:

- Your gun heats up and you'll find that rapid fire attempts will be unsuccessful. You should query the heat of the gun before firing if you need to be sure of a successful shot.
- The radar is unable to detect enemy bullets.

Robocode Bearings

- Absolute bearing in Robocode are such that north is 0 degrees.
- A scan event returns a bearing relative to your heading.



The most useful piece of information you'll likely apply is the relative bearing of a scanned opponent. In Robocode, upon scanning an enemy, if the **very next** command you issue is `fire`, your tank will turn this amount first before firing. I.e. it will automatically point at the scanned tank, then fire. Handy!

Scoring

- Hit $+(3 \times \text{fire power})$
- Miss $-\text{fire power}$
- Ram $+1.2$
- Being rammed $?$
- Collision -0.6
- A robot is killed $+50$ (*you have to be alive*)
- You kill a robot $+20\%$ of damage inflicted
- Kill by ramming $+30\%$ of damage inflicted
- Win a round $+10$

A strategy where your tank tries to stay out of trouble and does not attack, may be enough to survive until the end of a round. However, since points are awarded for killing enemy tanks, you should be aware that such a strategy may not be satisfactory to win overall.

After a battle, Robocode collects the scores and generates a rankings table showing who is 1st, 2nd etc.

Starting a Battle

- Start Robocode
- Select the *Battle* menu
- Find your robot and others that you would like to enter into battle!
- If you out survive all others, you win a *round*!



- Tip: Minimizing the window makes the simulation run faster!



September 2008

EECE 592 - Robocode



Entering robots into a battle is easy. From the Robocode Battle menu, select New. From here you should be able to see your robot and others in the dialog that pops up.

Creating your own Robots

```
package sarb;
import robocode.*;

/**
 * MyFirstRobot - a robot by (sarb)
 */
public class MyFirstRobot extends Robot
{
    ... // <<Area 1>>
    /**
     * run: MyFirstRobot's default behavior
     */
    public void run() {
        ... // <<Area 2>>
        while(true) {
            ... // <<Area 3>>
        }
    }
    ... // <<Area 4>>
    public void onScannedRobot(ScannedRobotEvent e) {
        fire(1);
    }
}
```

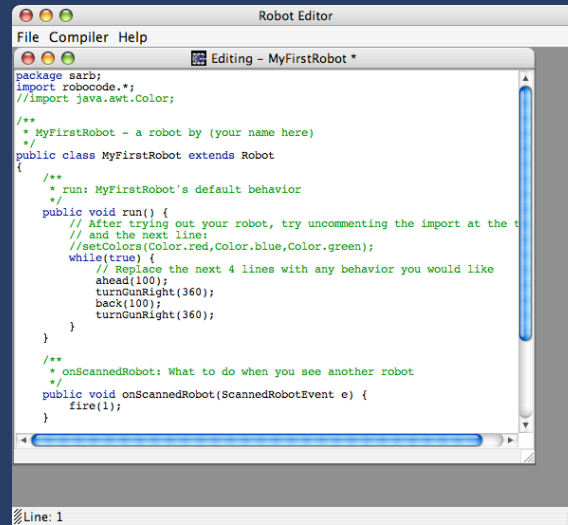
The following shows you what you get to start with when you create your first robot. <<Area1>> is typically used to declare global variables. They will be accessible from anywhere in your code. <<Area 2>> is inside the robot's "run" method which is called from within Robocode to instruct your robot to start. <<Area 2>> will typically contain code that only needs to be executed once per robot instance. E.g. to get the robot into some predetermined state. <<Area 3>> is the endless while loop that controls the normal behaviour of your robot. I.e. what your tank is doing when it's not firing or acting on a radar event. <<Area 4>> is where you implement helper methods and event handlers that determine how your robot will react to certain events. The robocode editor generates two of these event handlers, onScannedRobot and onHitByBullet, automatically for you

Example of a Simple 'bot

- The notes show the code for a fully working robot.
 - Can you figure out what it's doing?
 - And it really is this simple.
 - You do not need to be an expert in Java!

```
public class MyFirstRobot extends Robot
{
    public void run() {
        while(true) {
            // Replace the next 4 lines with any behavior you would like
            ahead(100);
            turnGunRight(360);
            back(100);
            turnGunRight(360);
        }
    }
    /**
     * onScannedRobot: What to do when you see another robot
     */
    public void onScannedRobot(ScannedRobotEvent e) {
        fire(1);
    }
    /**
     * onHitByBullet: What to do when you're hit by a bullet
     */
    public void onHitByBullet(HitByBulletEvent e) {
        turnLeft(90 - e.getBearing());
    }
}
```

Robocode Java Editor



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

/**
 * MyFirstRobot - a robot by (your name here)
 */
public class MyFirstRobot extends Robot
{
    /**
     * run: MyFirstRobot's default behavior
     */
    public void run() {
        // After trying out your robot, try uncommenting the import at the t
        // and the next line:
        //setColors(Color.red,Color.blue,Color.green);
        while(true) {
            // Replace the next 4 lines with any behavior you would like
            ahead(100);
            turnGunRight(360);
            back(100);
            turnGunRight(360);
        }
    }

    /**
     * onScannedRobot: What to do when you see another robot
     */
    public void onScannedRobot(ScannedRobotEvent e) {
        fire(1);
    }
}
```

Line: 1

The slide shows a screen show of the very simple editor that comes with Robocode.

Robocode Java Editor

- Robocode editor lets you
 - Edit
 - Compile
 - Package your robot into a java archive (`jar`)
 - ... but is quite basic.
- Recommend IDE is Eclipse for the serious robot developer. See later.

One useful feature of this “IDE” is that it allows you to package up your robots into a java archive file. This is what you would typically upload to the many online robot repositories.

However, the editor is quite basic. Instead, Eclipse is recommended.

Class Robot methods

- Robocode installation comes complete with Java docs.
- Look for `\javadoc\index.html`
- There are lots of methods e.g.
 - `void ahead (double distance)`
 - `void fire (double power)`
 - `double getEnergy ()`
 - `void turnLeft (double degrees)`
 - `void turnRadarLeft (double degrees)`

`void ahead ()` moves your robot forward the specified distance. Units of distance are pixels.

`void fire ()` fires a shot with the specified power.

`double getEnergy ()` will return how much energy you have left.

`void turnLeft()` rotates your robot the specified number of degrees anticlockwise.

`void turnRadarLeft ()` turns your robots radar the specified number of degrees to the left.

These are just a few of the methods supported by the Robot class. See the javadocs for the rest. Note that these methods are all synchronous methods. That is they will block execution until each method completes. So if your code looks like:

```
ahead (100);  
turnGunRight (30);  
fire (1);
```

Your 'bot will first travel 100 pixels. Upon getting there the gun will turn right 30 degrees. Upon completion of that your robot will fire. If you derive your robot from class AdvancedRobot, it provides all these methods plus some asynchronous methods. More on this later.

Robocode Events

- When something happens to your robot, an event is generated.
- In Robocode, this means that a virtual method that you implement is called.



There are many events, e.g.

- `void onBulletHit (BulletHitEvent event)`
- `void onHitByBullet (HitByBulletEvent event)`
- `void onScannedRobot (ScannedRobotEvent event)`



September 2008

EECE 592 - Robocode



When something happens in Robocode, e.g. you get hit, Robocode needs to know how you would like to react to, or handle this event. In software engineering, *callback* methods are a common way of implementing such event handlers and in languages such as C++ and Java, callbacks typically manifest themselves as virtual methods. The following is a sampling of some of the virtual methods that Robocode will call and expects you to implement:

`void onBulletHit (BulletHitEvent event)` is called when a bullet your robot fires, reaches a target. The event parameter passed to you by this method can provide you with some valuable information. For example, the `BulletHitEvent` class allows you to obtain the attacked robots name and energy.

`void onHitByBullet (HitByBulletEvent event)` is called when your 'bot is hit. The event parameter passed to you tells you the name of the robot that fired the bullet, the bearing to the bullet, the power of the bullet etc.

`void onScannedRobot (ScannedRobotEvent event)` is called when your robot scans another. As you may expect this event can tell you all sorts of information about who you've just spotted!

These are just a small sample of the events supported by Robocode. There is even an `onWin ()` event called when you win a round and an `onDeath ()`. Robocode provides an implementation of this last one for you so you don't have to explicitly implement this method. Unless you need to perform some cleanup (e.g. closing a file) on robot death.

Class AdvancedRobot Methods

- AdvancedRobot provides support for file access and asynchronous commands
- Some methods:
 - `void setTurnGunRight (double degrees)`
 - `void setAhead (double distance)`
 - `void setFire (double power)`
 - `void execute()`



- Class `RobocodeFileStreamOutput`
 - used for writing to a file



September 2008

EECE 592 - Robocode

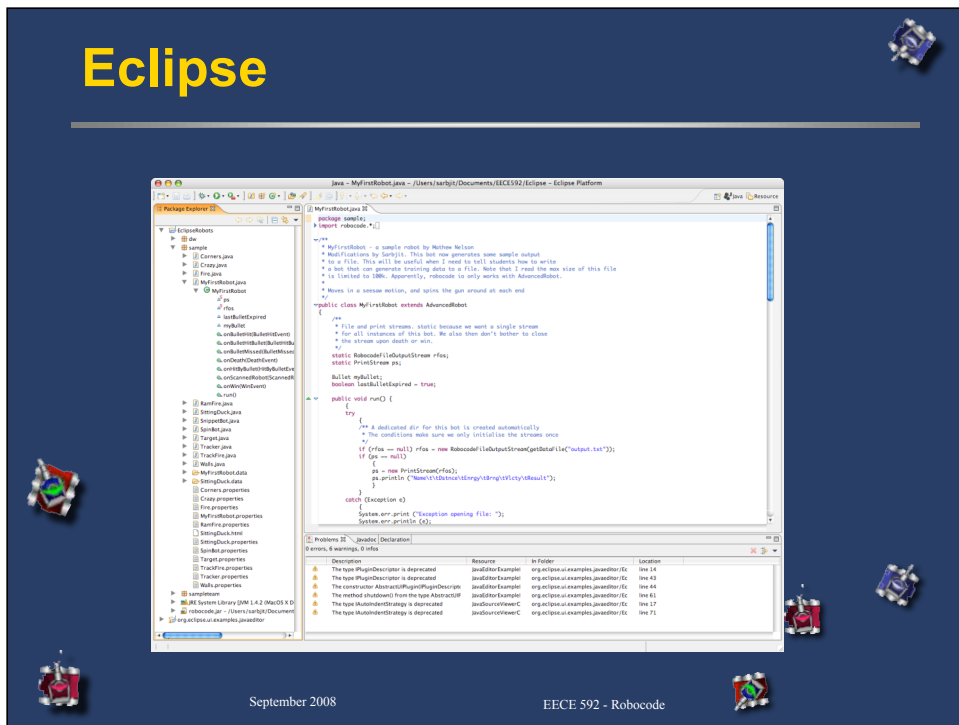
Deriving from the AdvancedRobot class provides further methods. Notably asynchronous robot commands. Consider the following example:

```
setAhead (100);  
setTurnLeft (30);  
setFire (1);  
execute();
```

In this case, each method completes immediately and your robot does nothing until you return control back to Robocode. You do this by issuing the `execute` method. When `execute` is reached, Robocode will start all requests at the same time and advance your tank as much as possible in one *tick* of the clock. This tick represents a single frame of motion. To find out how much your tank actually travels in one tick you need to look at the physics of the tank. E.g. maximum tank speed is 20 pixels per tick and turn rate is 20 degrees per tick. These commands will remain in an event queue that Robocode creates for you. The event queue is checked each tick. Once the commands are complete, the events are removed from the queue. Note that there is a `waitFor()` method that you may use to wait for a particular condition to complete. E.g. you may want to wait for your tank to complete its 30 degree turn before firing. By the way, can you see what path the first two `setAhead` & `setTurnLeft` commands above would cause the tank to take?

At times it will be necessary for you to write out information to a file. In this case you cannot use the `java.io`, `FileOutputStream` methods. Java security prohibits this. Instead, you have available to you a Robocode class for this. `RobocodeFileStreamOutput`.

Eclipse



Eclipse is a recent IDE introduced by the Eclipse Foundation. It is written in Java and admittedly, early versions were sluggish in use. However, the latest versions are slick. Of course it can be used to develop Java, but the IDE is not restricted to any one language. I would highly recommend you use this to develop your robots for this course. It is far more user friendly than the basic editor that comes with Robocode. For those new to Java, it supports class browsing and method completion, so you don't have to rummage around for the documentation all the time. Compilation errors are highlighted as you type, and Eclipse will even suggest how to fix the error! Very useful, if you are a beginner to Java.

How to Install Eclipse

- Go to <http://www.eclipse.org>
 - Current version 3.4 (as of 12 September 2008)
 - Look for “Eclipse IDE for Java Developers”
- You need a Java VM
- Launch from the icon

As with Robocode, you will need a preinstalled java virtual machine on your PC or Mac. Eclipse is being continually updated and I fully expect there to be newer versions in the future.

How to setup Eclipse for Robocode

- Create a new project
 - File->New->Project
 - In the pop wizard, select Java Project
 - In New Java Project dialog Give it a name
 - In same dialog, select create project in external location.
 - Then browse and choose the robots folder installed by robocode.

When creating a project, Eclipse will look for folders containing java source files. These folders will appear as “packages” in the browser. It’s important to make sure that the location selected for the external location is a directory above where the folder containing the actual robot folders is. Each folder will then appear as a package in Eclipse Package explorer view on the left.

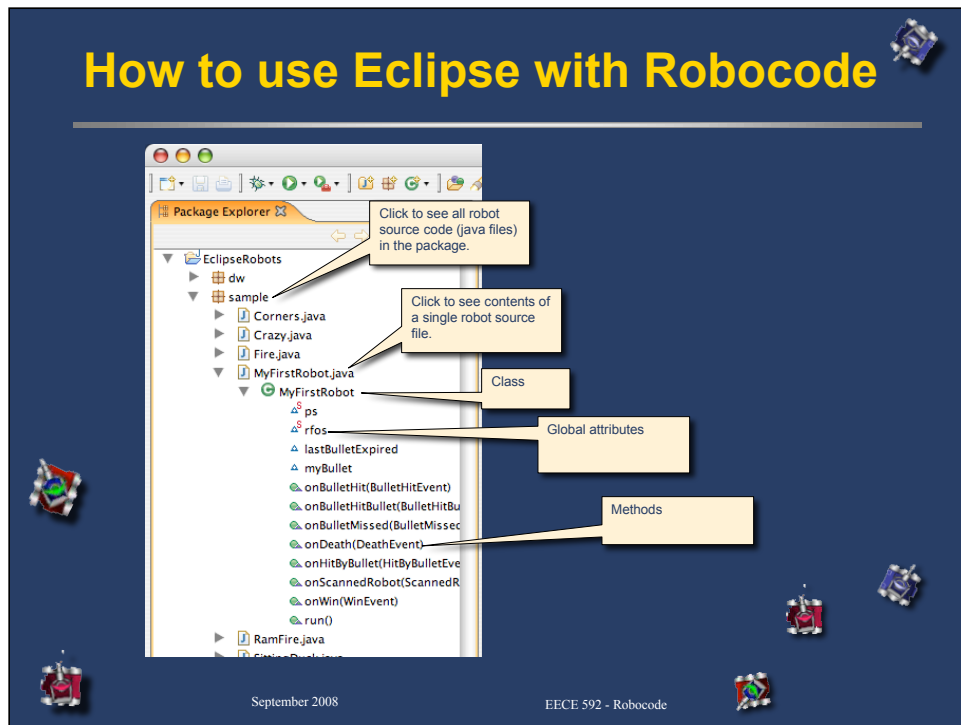
If you don’t see a ‘sample’ and ‘sample team’ package, try selecting a folder even higher, e.g. the robocode folder itself.

How to setup Eclipse for Robocode

- Next you need to add the library for robocode.
- Select the project in the Package Explorer window
 - Select menu item **File->Properties**
 - Select **Java Build Path** in Properties dialog
 - Select the **Libraries** tab
 - Then press **Add External Jar**
 - And browse to and locate **robocode.jar**
 - Done

Robocode.jar will be in the directory where robocode is installed.

How to use Eclipse with Robocode



The Package Explorer view will show you all folders installed by Robocode under 'robots'. Each folder containing java source code will appear as a package. Opening a package lists all java source files. Double clicking on one of these source files will open the file in the editor window (not shown above). Eclipse makes it easy traversing classes and methods. If you haven't correctly set up the external location, you will see warning symbols against each package and source file. At least when initially setting up Eclipse using the previous instructions, you should not see any such problems.

How to use Eclipse with Robocode

- Eclipse automatically compiles all java files.
- Eclipse can even be set to automatically build
 - I.e. create '.class' files.
- Once built, you can switch to robocode to try out your newly compiled robot!



September 2008

EECE 592 - Robocode



With the appropriate settings, Eclipse can be set to automatically compile and build java files in the background. This is done when you save the updated java source. By default, the compiled class file is created in the same directory the source file is.

After editing and saving a robot in Eclipse, your compiled robot will be ready to enter into a battle straight away!

Sample Robocode

```
import robocode.*;

/**
 * SpinBot - a sample robot by Mathew Nelson
 * Moves in a circle, firing hard when an enemy is detected
 */
public class SpinBot extends AdvancedRobot {

    /**
     * SpinBot's run method - Circle
     */
    public void run() {
        while (true) {
            // Tell the game that when we take move,
            // we'll also want to turn right... a lot.
            setTurnRight(10000);
            // Limit our speed to 5
            setMaxVelocity(5);
            // Start moving (and turning)
            ahead(10000);
            // Repeat.
        }
    }
}
```

Upon starting a battle, Robocode will call the SpinBots run method, which simply puts the tank in an endless while loop. So if no other event handler is triggered, this tank is programmed to go ahead a distance of 10000, **while at the same time** turning right. The end result is that SpinBot goes around in circles! Can you guess what would happen if the velocity was increased?

Sample Robot cont.

```
/**
 * onScannedRobot: Fire hard!
 */
public void onScannedRobot(ScannedRobotEvent e) {
    fire(3);
}

/**
 * onHitRobot: If it's our fault, we'll stop turning and moving,
 *             so we need to turn again to keep spinning.
 */
public void onHitRobot(HitRobotEvent e) {
    if (e.getBearing() > -10 && e.getBearing() < 10)
        fire(3);
    if (e.isMyFault())
        turnRight(10);
}
}
```

Here we see two event handlers. Upon scanning the action is quite simple. SpinBot will fire with a power setting of three. Note that Robocode will automatically turn the robot using the bearing returned in the scanned event prior to firing. This happens only if the first command in this event handler is fire.

The next event handler attempts to deal with the situation if SpinBot hits another robot. Robocode can even tell you if it was your fault for hitting the other robot!

Robocode web resources

- There are lots of resources. Try:
- Excellent two part article by Sing Li
- Robocode Wiki

Sing Li's article can be found at :

<http://www-106.ibm.com/developerworks/java/library/j-robocode/>

The Robocode wiki page is http://robowiki.net/cgi-bin/robowiki?Robo_Home

How to track bullets:

<http://www-128.ibm.com/developerworks/library/j-tipbullet.html>

In fact there are many resources and literature on the web that you will find useful.

All links valid at time of writing (September 2008).

Applying NN & RL to Robocode

- Assignment for EECE 592:
 - Build your own robot!
- You must demonstrate the use of neural nets *and* reinforcement learning in the robot you develop.

The assignment for this course requires you to develop your own Robocode robot. The course will cover a number of topics related to neural networks, including learning algorithms, data representation, and heuristics. Multi-layer perceptrons and reinforcement learning are two of the topics covered. To complete this course, you are expected to demonstrate the use of at least these two areas within the development of your robot. Of course, while building a strong adversary is a nice goal, it is more important to show your understanding of the techniques taught in the course and your ability to apply them in practice. It is also hoped that this will be a fun and challenging coursework assignment.

For more details visit <http://www.ece.ubc.ca/~eece592/> and click on the “*coursework*” link.

RL for Target Selection

- Can reinforcement learning (RL) be used to help your Robot win a round?
 - What are the states (*value function*)?
 - What is the reward?
 - What is the policy?
- Use a more localized goal.
 - Use RL to help pick a target you can destroy.

Reinforcement learning is based upon approximating probabilities. What RL must do is, given any particular state of a battle, it must help you determine what the the next move your robot should make in order to maximize its probability of winning a round. Unlike a game like noughts-and-crosses (Tic-Tac-Toe), the number of states here in Robocode is, for all practical purposes, infinite. You will have to decide how best to simplify the problem in order to make it tractable.

RL / Strategies

- Can RL help with other robot behaviour?
 - How to avoid walls?
 - How to avoid bullets?
 - BTW: Enemy bullets cannot be scanned!
 - How to avoid damage in general?

Consider applying reinforcement to other robot actions. For example, being rammed by another robot or hitting a wall incurs damage. As of course does being hit by an enemy bullet! However, the scanning of enemy bullets in Robocode is unfortunately, not support. Does this mean enemy bullets cannot be avoided? Perhaps not. Would it instead be possible to predict if an enemy was about to fire at you once scanned? If so, such information would be critical in directing robot motion for bullet avoidance. For example, the energy of a robot will drop by a value between 0.1 and 3 when it fires. While not unique, this figure may be sufficient to tell if an attack is imminent.