Welcome to the World of Robocode:

Build the best - destroy the rest!

Robocode is a game where you can stage battles between tanks which have different behaviours and see which behaviour wins. Not only that, but you can actually (and easily) create your own tank, with its own behaviour and see how it does in battle.

Note here you specify (program) the behaviour of your tank in advance, so you have to envisage all the situations it can get into (and figure good ways out of them)

To add the icing on the cake, you can download the entire game from the internet, meaning that, if you like what you see today, you can continue working on your tank (or 'bot' in Robocode terms) at home and possibly even enter it in competitions.

So here is what we are going to do:

Download Robocode from the Internet

Start a battle using the inbuilt robots provided

Create a robot of our own, save it, and include it in a battle

Look at your robot's code and match it to its resultant behaviour in battle

Do some minimal changes to your robot, and see how it behaves in battle as a result of those changes

Describe some of the facilities available to you, to modify your robot further, and let you try them to see their worth in battle.

From there, the world is your oyster: Try to make your robot even better.

Maybe at the end we could reserve 5 minutes for having battles between your robots in a fight off!

Let the games begin....

A: Download Robocode (trialled with Window 7, service pack 1, Robocode version 1.7.3.5)

1] Your PC should be turned on.
2] Start internet explorer and go to Robocode home: http://robocode.sourceforge.net/
3] Here you will find a wealth of Robocode material, making up the majority of this tutorial in fact!
4] Select the 'Download Robocode' option and then select 'Download robocode-X.X.X.X-setup.jar(X.X MB)' in the middle of the page.
5] You will go to what looks like an advertisement page (indeed it is, but above the adverts, you will see that the computer is telling you it will download robocode shortly, or in a number of seconds.
6] When it counts down to zero, you will see a pop up window at the bottom of your browser asking if you want to save or open robocode. Select 'Open'.
7] Your computer will probably do a security scan first, but in a few seconds a window will come up in the middle of the screen. It will ask you if you wish to install robocode in C:\robocode. Select 'Yes'.
8] Click OK to the next message.
9] Then a message will ask you if you want to create file associations in something called the "Windows Registry". It doesn't matter what you select here, so lets say 'No'

Thats it, you're done!

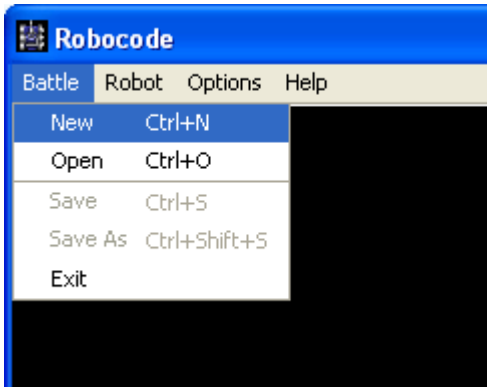
Congratulations. You're now ready to enter the weird and wonderful world of Robocode...

Notes:
If anything did go wrong for you along the "install" process, just use the back button to retreat a few internet pages and try again!

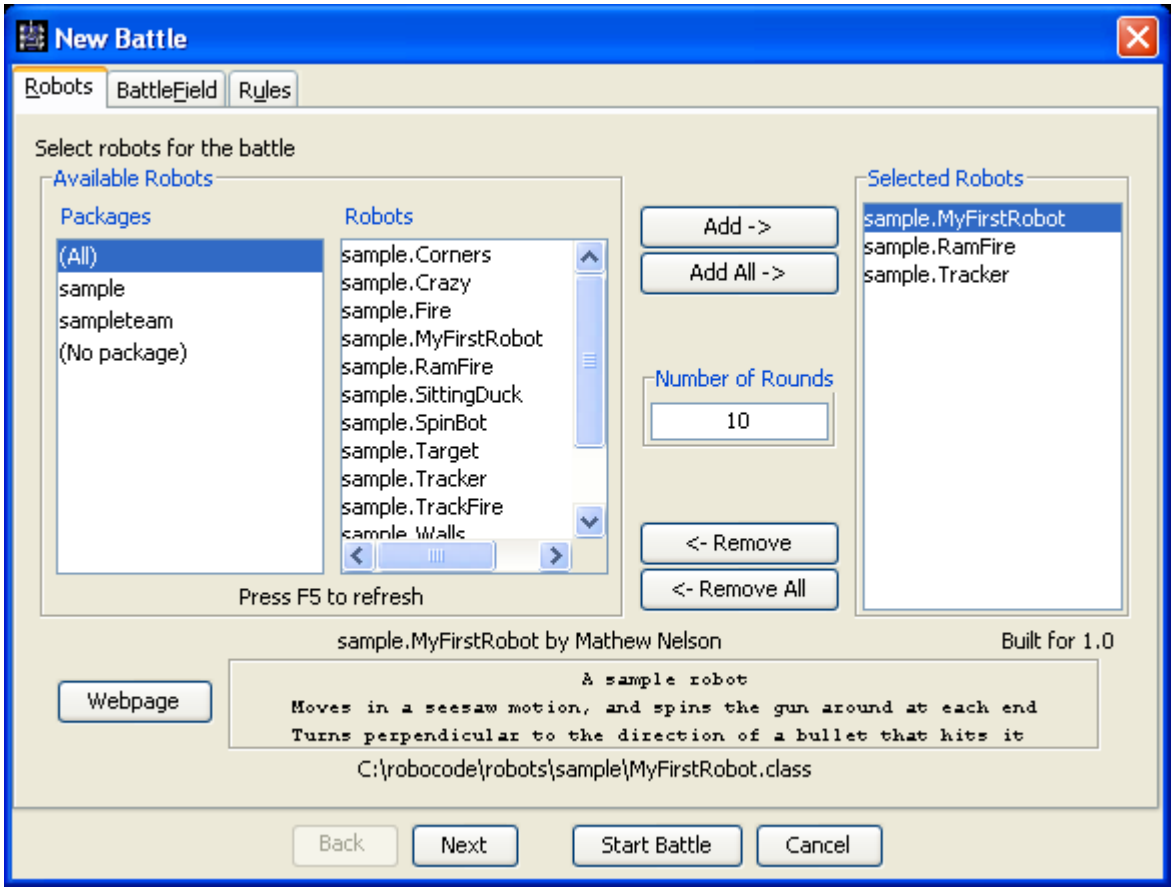
Alternatively, another download site is http://sourceforge.net/projects/robocode/, where you need to click on the download button in the middle of the page.

Aside:
Incidentally, you might have noticed that we downloaded Robocode from a site called www.sourceforge.net. Sourceforge is a place where you can get lots of software absolutely free! So instead of buying (say) Microsoft Office, you could go to Sourceforge and get "Open Office", a very similar offering that will provide you with most of the word-processing, spreadsheet and slide-preparation functionality you would ever need. In fact, no matter what your software needs are, its likely that sourceforge has a package for you. This happens because lots of hobbyist programmers give their services free to develop these systems, for the sheer enjoyment of it. They are the ones who should have done our courses!

B: Starting a battle with the robots already there:
After installation you should see a 'tank'/Robocode icon on the main screen. Double click on it and let's run a battle to see what the game looks like. Simply click the **Battle** menu, then select **New**, as shown on the picture here:



You'll be presented with the **New Battle** screen, where you pick the robots and options for a battle. For this battle, pick any 3 robots (including Tracker) from the list of robots. Add them in by double-clicking on their names (or selecting each one and clicking **Add**) The screen should now look something like this:



See the **Number of Rounds** box in the middle? In Robocode, each battle consists of a number of rounds, as you'll see soon. For now, lets make that 3.

Finally, click on the **Start Battle** button to begin! Watch for Tracker's little dance if he wins the rounds!

C: So lets get involved and make a robot for ourselves:
In this section, we'll use the Robot Editor to create your very own, brand new robot. Robot Editor is just like Word, except that, while Word allows you create documents, Robot Editor allows you make Robots (in a programming language called Java)

The first step is to open up the Robot Editor. From the main Robocode screen, click on the **Robot** menu, then select **Editor**.

When the editor window comes up, click on the **File** menu, then select **New Robot**.

In the dialog box that follows, type in a name for your robot. Then, in the next dialog box enter your initials as the "package name" - this is just another phrase for "the directory where we will save your Robot". Voila! You now see the (Java) code for your own robot. This is ROUGHLY what you should be looking at:

```
 package <<WHATEVER PACKAGE NAME YOU SPECIFIED ABOVE>>;
 import robocode.*;

//import java.awt.Color;

//API help : http://robocode.sourceforge....
/**
 * <<YOUR ROBOT NAME - a robot by (your name here)
 */

public class <<YOURROBOTNAME>> extends Robot {

    /**
     * run: YOUROBOTSNAME default behaviour
     */

          public void run() {

         // Initialization of your robot should be put here

         // After trying out your robot....
         // and the next line

         //setColors(Color.red,Color.blue,Color.green);

         // Robot main loop
          while (true) {
                    ahead(100);
                   turnGunRight(360);
                   back(100);
                   turnGunRight(360);
         }
     }

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

How do you think it will behave in battle based on this code? Don't be overawed – it's actually very similar to what the code above suggests. Have a think about it and

then see how it behaves in battle. First, save your robot by selecting the **Save** option in the **File** menu. Follow the prompts to save your robot.

But this is not enough in itself: The Java code you are looking at is quite english-y, in nature. But computers don't understand english-y type languages. So the english-y version you have in front of you needs to be converted into a form that the computer can understand. Luckily you don't have to do this manually. A kind individual has written a piece of software that takes in Java code and pumps out a computer-friendly version that the computer can execute. This piece of software is often called a COMPILER and you will now use this compiler to transform your Robot into something the computer can understand and execute.

So, compile it by selecting **Compile** in the **Compiler** menu. You should ideally get a message that the conversion (compilation) was successful. If not, there is an error in your java code that you will have to fix - but that's unlikely given that you didn't change anything.

Note that every time you make a change to your Robot code you will have to re-save it AND re-compile it, for the changes to take effect in battle.

If your robot compiles without any errors, you can start a new battle with your robot. Start a new battle by selecting **New** in the **Battle** menu. If you cannot see your robot, you might have to refresh the list of robots by **pressing F5**. Add your robot to the battle together with at least one other robot, and press the **Start Battle** button to let the games begin!

Code Inspection

So lets have a look at the code. At the moment we are concerned with the bit in **Bold** above: the bit starting 'public void run() {'

When a battle starts in Robocode, it identifies all the Robots that have been included in the battle and searches for this 'run' line. It then carries out any instructions after this line, up until the matching closing curly bracket on line 31.

In this piece of code, you will see there are some lines in green. Notice that these lines either start with '//' or are several lines of code long and are surrounded by '/*' and '*/'. These are special characters in Java that tell the compiler that what follows is to be ignored: it is for human reading purposes only. We call these lines comments, as they tend to comment on what pieces of code do, for other programmers.

The real instructions start at the line 'while (true) {'. That line of code tells the computer to keep executing the code (up to the matching closing curly bracket '}' on line 30, 'while (true)'. As 'true is always 'true', the Robot will do the instructions enclosed in these curly brackets forever (or at least until it dies a glorious death in battle!).

So what do these enclosed instructions do? Well, its fairly intuitive: they command the robot to go forward 100 “somethings”, turn its gun around 360 degrees, go back 100 somethings, and turn its gun around 360 degrees again. In this case the "100 somethings" are "pixels", a unit of screen resolution.

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

Run the battle again and notice the behaviour of your tank. True to form, it does go forward 100, turn the gun around, go back 100, and turn the gun around. So it seems to be consistent with the instructions it has been given.

Is that all it does? - Think about this for a minute before answering....

Event Driven Programming

No: It also fires a bullet. In fact, it also seems to back away when it hits a side wall or when it is hit by a bullet. In doing this, the tank is reacting to events in the battle, just like you react to events in your environment: If it starts raining you put up your umbrella. if it drys off, you take it down.

This is a certain style of programming called "event driven programming", where your Robot is informed of some event in its environment and it reacts to this event. Look at the lines of code that follow the 'run()' method:

```
public void onScannedRobot(ScannedRobotEvent e) {
     //Replace the next line with any behavior you would like
     fire(1);
 }
```

Here the environment (Robocode) has informed your Robot of a onscannedRobot Event and your Robot is waiting for that event here: The Robocode environment is telling your Robot that another Robot is in front of your gun. This onScannedRobot method ("public void onScannedRobot(ScannedRobotEvent e) {") contains the reaction to this event, and you can change it: The instruction contained in this method directs our Robot to fire a bullet of strength 1 as a reaction to this occurrence. The “}” ends the response

Note that your Robot also gets informed when it has been hit by a bullet and when it has hit the side wall of the terrain. What does it do in each of these situations?

D: Some small amendments:

Here we will change the color of the robot, the size of the shell it shoots when it scans a robot and its movement pattern

D1: First lets change the shell size. Go back to the **onScannedRobot** method and, instead of having it fire a shell of power '1', have it fire a shell of power '3'. Yes, its as easy as it seems! Note that bigger shells travel more slowly to their target and consume more of your power, but if they hit, then they cause more devastation.

Dont forget to save AND compile, and restart the battle to observe this new behaviour.

D2: Now you will change the colour of your tank. Notice the **//import java.awt.Color;** at the start of your robot. This line imports coloring facilities that are available in the Java language, for you to use in your Robots code, but at the moment the "//" is telling the compiler to ignore it. Take out these // and we will be able to change the color of our tank.

Likewise, in the run method, there is a commented out line //**setColors(Color.red….** Again, take out the //, to signify that we don't want the computer to ignore it any more

Now save the robot again, compile it again and notice the different appearance of your Robot in battle. If you are feeling really adventurous, change the color mix to something else and try again, but note that the coloring facilities do not include all colors!!

D3: You could also change the amount your robot goes ahead, the amount the gun turns right, or the amount it goes back if you like? Don't forget to save and compile after the changes!

D4: Now I want you to change the behavior in line with my description. Note: I am not going to tell you how to do this, just give you a description. This is what happens in the real world as a programmer: people give you descriptions of behaviors they want and you are supposed to make it happen.

I want you to make your robot go in triangles: ahead, turn right, go ahead, turn right, go ahead and turn right, so that it is exactly where it started off. You might consider the following facilities available to you from Robocode (available because you "**import (facilities from) robocode.***"):

- turnRight("However many degrees you like");
- turnLeft("However many degrees you like");

If you have succeeded, and are enjoying yourself, chances are you will enjoy programming and enjoy Software Engineering/Computer Science/Games

Development type courses. But always remember to check that your Robot is behaving as you would expect for the changes you make! That’s called testing: an important part of any programmer’s life!

E: Using the Facilities to empower your tank:

There are 2 helpful ways you can expand your robot's prowess.

1] You can look at other, more expert robots and hack what they do

Its easy to look at the other robots provided with Robocode. In the editor, select File and Open. Go up to the directory (or package) 'robots' and then to 'sample', and pick any of the files contained within, to show you the source code of the tank it refers to.

Maybe you could use some of this source code in your tank? Sometimes a simple cut and paste will work – but sometimes it's a bit more involved, as the code you take might refer to other stuff that isn't contained in your robot.

Lets take an example: Open the tracker robot in the sample directory. It has its onScannedRobot(ScannedRobotEvent **e**) method on line number 77. Note that this is much more elaborate than your robot. So let's steal some of it and put it into our Robot!

Look at the code from line 106 to line 112:

```
if (e.getDistance() < 100) {
    if (e.getBearing() > -90 && e.getBearing() <= 90) {
        back(40);
    } else {
        ahead(40);
        }
    }
```

Lets just talk through this code before we steal it - its important to understand this code because otherwise we may end up inserting undesirable behaviour into our tank!

The Robocode battlefield has just sent your tank an event (**e**) notifying you that there is a robot in your sights. This event contains lots of information about the tank you are targeting (for example its power reserves, its distance from you). This code basically looks at how distant the tank is from you and moves your tank away from it.

So it first checks if **e** is less than 100 pixels distant from your tank. If so, it finds the (compass) bearing of **e**: Specifically the line: **if (e.getBearing() > -90 && e.getBearing() <= 90)** checks if e is in front of you (has a bearing of > -90 degrees and < 90 degrees). If so, it moves your tank **back** 40 pixels. Otherwise (**else**) the tank is behind you and your tank should move 40 pixels **ahead** to put distance between your tank and this robot.

Lets copy and paste this code into our Robot, specifically into the OnScannedRobot method. Personally I would choose to put it in after we fire off the shell, but I'll leave that decision up to you. Now my onScannedRobot(ScannedRobotEvent e) looks like this:

```
public void onScannedRobot(ScannedRobotEvent e) {

        // Replace the next line with any behavior you would like

        fire(1);

        if (e.getDistance() < 100) {

                if (e.getBearing() > -90 && e.getBearing() <= 90) {

                        back(40);

                } else {

                        ahead(40);

                }

        }

}
```

Save and compile it and then run a battle to see how it changes the behaviour of your robot

Why don't you try to create a behaviour that finds out if a scanned robot is near? If it is fire a heavy (weight 3) bullet and get out of there. Else (if the tank is far) fire a light bullet, as there is less likelihood of a hit and maintain a normal pattern.

2] The 2nd alternative is is to look at http://robocode.sourceforge.net/docs/robocode/ and select 'robocode' to see all the facilities that are available to you as a robot builder. Its quite overwhelming. Lets look at one in specific to see how you might use one of these facilities.

Scroll down the left bottom pane until you find 'HitRobotEvent'. Obviously, when you hit a robot you want to back away. Luckily Robocode sends you this event to warn you that you have collided with another, and you can create a response to this collision in the method **onHitRobot()**

So lets create this new method in our tank. After the current lines:

```
public void onHitByBullet(HitByBulletEvent e) {

        // Replace the next line with any behavior you would like

        back(10);

}
```

Insert the lines

```
public void onHitRobot(HitRobotEvent e) {

        back(100);

}
```

This means that your robot will back away from the tank after he hits it.

Save and compile and see if you notice a difference in behavior when your robot collides. Notice that this behaviour is not perfect though. What happens if you collide back into a wall when you retreat? …..Maybe you could work on that?

So, at the end of this tutorial you have

Compiled Java code into an executable
Written Java code
Implementing behavior specified by someone else (me!)
Tested the resultant code
Learned about event driven programming
Learnt a bit about hacking other people's code
Learnt how to use Robocode documentation

Congrats:
If you have enjoyed it, then maybe you should consider computing as a future career.

If not then that's good too: you have cancelled it off you list and saved yourself an unhappy 4 years in University and 40 years afterwards as a career!

Tutors notes:

Sequence selection, iteration

And the Science!:

Note compilation;

Note that you are using functions: functions that someone else has created that your robot can sit on - lots of game development is like this.

Notes on inheritance: "extends robot;". Includes a brief review of the 'import robocode' a directory that has the original thing/type you are extending

Note event driven programming. Note the way the battlefield calls each (included) robot to run, and how it fires off relevant events to which the robot can respond in its individual way (onscannedrobot). Also good to dissect the event itself to get bearing, distance, health of scanned robot.

Again, getting them to implement a behaviour here would be great. If the tank is near, fire a 3 strength and get the flock out of there. If the tank is far, file a1 strength and maintain normal pattern.

Show how walls can screw this behaviour up.