University of St Andrews

School of Computer Science
University of St Andrews

ROBOCODE

Simone Conte
(sic2@st-andrews.ac.uk)

(with thanks to Ruth Letham, Shyam Reyal, Greg Bigwood, James Smith, Andrea Rendl, Kris Getchell, Tim Storer and Martin Bateman)

University of St Andrews

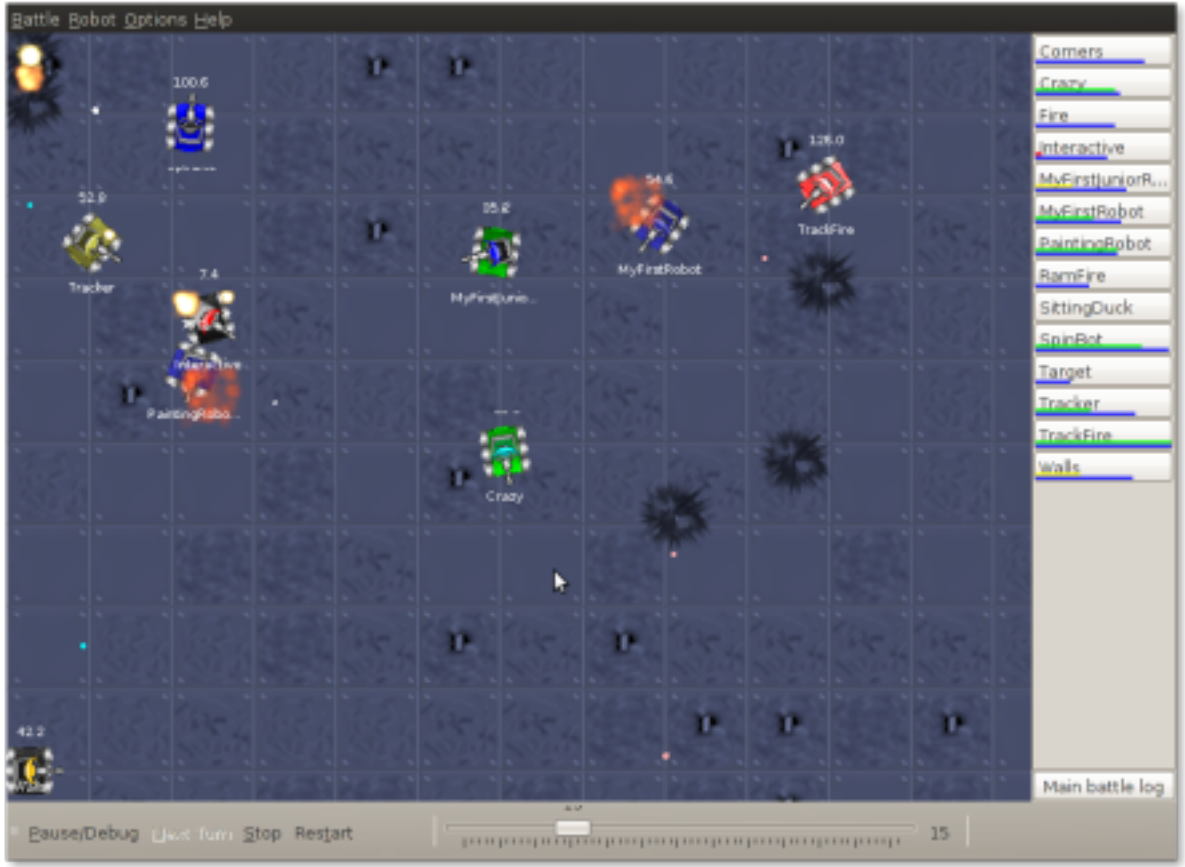
School of Computer Science
University of St Andrews

What is ROBOCODE

- Developed by IBM®
- Programming game
- You develop strategies for your robot
- Write *Java*™ to control your robot

University of St Andrews

School of Computer Science
University of St Andrews



Write code in Java

- High-level
- Object-Oriented
- Case-sensitive (Foo, foo, FOO)

- ***"Write once, run anywhere"***

University of St Andrews

School of Computer Science
University of St Andrews

Write code in Java

Java code for your robot

- package & import
- comments (/* ABC */ or // XYZ on one line)
- robot definition
  - **run**
  - **onScannedRobot** - what to do when you see a robot
  - **onHitByBullet** - what to do when you are shot
  - **onHitWall** - what to do if you bump into a wall
- Edit the program
  - change name of the author in the comment

University of St Andrews

School of Computer Science
University of St Andrews

Change the color of your robot

import java.awt.Color;

setColors(Color bodyColor, Color gunColor, Color radarColor)
setColors(Color bodyColor, Color gunColor, Color radarColor, Color bulletColor, Color scanArcColor)

Example

```
public void run() {
        setColors(Color.ORANGE, Color.YELLOW, Color.GREEN);
}
```

Moving

- **Movement** is controlled by **ahead()** and **back()**
- **Direction of travel** is controlled by **turnLeft()** and **turnRight()**
- **Direction of the gun** is controlled by **turnGunLeft()** and **turnGunRight()**

University of St Andrews

School of Computer Science
University of St Andrews

Moving

- **Movement** is controlled by **ahead()** and **back()**
- **Direction of travel** is controlled by **turnLeft()** and **turnRight()**
- **Direction of the gun** is controlled by **turnGunLeft()** and **turnGunRight()**

- Need to know how far to move (**pixels**) or turn (**degrees**)
- This code makes the robot:
  - moveforward
  - spin around clockwise
  - move backwards
  - spin its gun counter-clockwise

```
ahead(100);
turnRight(360)
back(50);
turnGunLeft(360);
```

Shooting

- Shooting is controlled by **fire()**
- When you see a robot an **event** is triggered
  - **onScannedRobot()** is executed

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

```
public void onScannedRobot(ScannedRobotEvent e) {
        fire(1.5);  // MAX is 3
}
```

University of St Andrews

School of Computer Science
University of St Andrews

Shooting

- Shooting is controlled by **fire()**
- When you see a robot an **event** is triggered
  - **onScannedRobot()** is executed

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

- Want to pack more of a punch?

```
public void onScannedRobot(ScannedRobotEvent e) {
        fire(10); // This will be equal to fire(3);
}
```

University of St Andrews

School of Computer Science
University of St Andrews

Shooting

- Shooting is controlled by **fire()**
- When you see a robot an **event** is triggered
  - **onScannedRobot()** is executed

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

- Want to pack more of a punch?

```
public void onScannedRobot(ScannedRobotEvent e) {
        fire(10);
}
```

Be Careful!
Firing costs energy

No energy = Sitting Duck!

Shooting

The more you shoot, the more power is spend

- (4 * power) damage if it hits another robot.
- If power is greater than 1, it will do an additional 2 * (power - 1) damage.
- You will get (3 * power) back if you hit the other robot.

Taking Fire

- If you are dishing it out, you are probably going to take some hits too
- When your robot gets shot, another **event** is triggered
  - **onHitByBullet()** is executed

Taking Fire

- If you are dishing it out, you are probably going to take some hits too
- When your robot gets shot, another **event** is triggered
  - **onHitByBullet()** is executed

- Try changing your direction by a fixed amount and moving away

```
public void onHItByBullet(HitByBulletEvent e) {
        turnLeft(90);
        back(100);
}
```

University of St Andrews

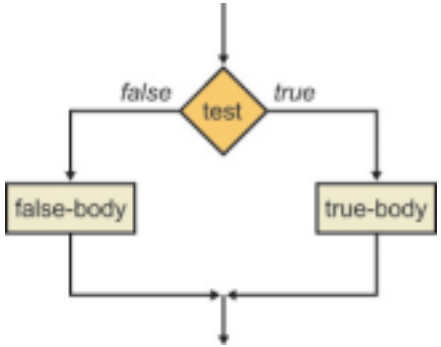
School of Computer Science
University of St Andrews

If … else Conditions

- Used to make ***Boolean*** *(true/false)* decisions in code
  - do one thing, or another thing depending on a condition
- Basic Structures

```
if (<condition>) {
        // code
}
```

```
if (<condition>) {
        // code
}
else {
        // code
}
```



University of St Andrews

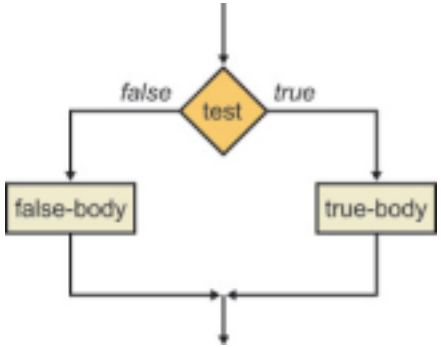
School of Computer Science
University of St Andrews

If … else Conditions

- Used to make ***Boolean*** *(true/false)* decisions in code
  - do one thing, or another thing depending on a condition
- Basic Structures

```
if (<condition>) {
        // code
}
```

```
if (<condition>) {
        // code
}
else {
        // code
}
```



- Condition is enclosed in parentheses **()**
- Action is enclosed in braces **{}**
- Use **! (*NOT*)**, **&& (*AND*)** and **|| (*OR*)** to make more complex conditions

Example

- Strategy: When I see an enemy, it is close by and I have lots of energy fire hard at it, otherwise just fire a basic shoot

Example

- Strategy: When I see an enemy, it is close by and I have lots of energy fire hard at it, otherwise just fire a basic shoot
- When **onScannedRobot** is executed
  - **e** contains information about the robot we scanned
  - Get the distance of the scanned robot with **e.getDistance()**
  - Get energy of my robot with **getEnergy()**

```
public void onScannedRobot(ScannedRobotEvent e) {
        if (e.getDistance() < 50 && getEnergy() > 50) {
                fire(3);
        } else {
                fire(1);
        }
}
```

Variables

- You can also make use of variables

```
public void onScannedRobot(ScannedRobotEvent e) {
        double myEnergy = getEnergy();
        double enemyDistance = e.getDistance();

        if (enemyDistance < 50 && myEnergy > 50) {
                fire(3);
        } else {
                fire(1);
        }

        if (myEnergy > 40) {
                turnLeft(120);
                fire(2);
        }
}
```

Example 2: Shooting

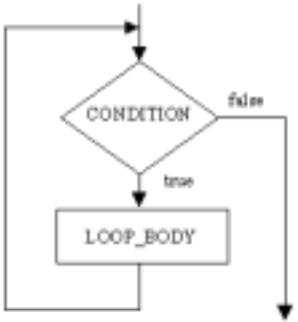
You cannot fire if your gun is overheated

```
if (getGunHeat() == 0) {
        fire(Rules.MAX_BULLET_POWER);
}
```

While Loops

- Repeat a block of code any number of times while a condition holds
- Basic Structure

```
while (<condition>) {
        // code
}
```



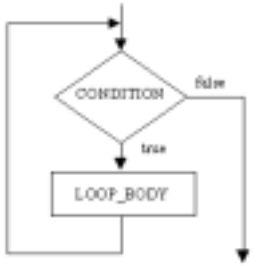
University of St Andrews

School of Computer Science
University of St Andrews

While Loops

- Repeat a block of code any number of times while a condition holds
- Basic Structure

```
while (<condition>) {
        // code
}
```



- Condition is enclosed in parentheses **()**
- Action is enclosed in braces **{}**
- Use **! (*NOT*)**, **&& (*AND*)** and **|| (*OR*)** to make more complex conditions
- Example:

```
while (getEnergy() > 90) {
        fire(1);
}
```

```
double myEnergy = getEnergy();
while (myEnergy > 90) {
        fire(1);
}
```

Java Coding

- Java is case sensitive
  - getenergy() **is not** the same as getEnergy()

Java Coding

- Java is case sensitive
  - getenergy() **is not** the same as getEnergy()
- White space does not change the meaning of your code, but can make it easier to read (i.e. **if (condition)** is the same as **if(condition)**)

Java Coding

- Java is case sensitive
  - getenergy() **is not** the same as getEnergy()
- White space does not change the meaning of your code, but can make it easier to read (i.e. if (condition) is the same as if(condition))
- Parentheses **()** and braces **{}** always come in pairs
  - An open parenthesis **(** must have a matching closing parenthesis **)**
  - An open brace **{** must have a matching closing brace **}**

University of St Andrews

School of Computer Science
University of St Andrews

Java Coding

- Java is case sensitive
  - getenergy() **is not** the same as getEnergy()
- White space does not change the meaning of your code, but can make it easier to read (i.e. if (condition) is the same as if(condition))
- Parentheses **()** and braces **{}** always come in pairs
  - An open parenthesis **(** must have a matching closing parenthesis **)**
  - An open brace **{** must have a matching closing brace **}**
- Instruction statements must end with **semicolon ;**
  - fire(1);
  - getDistance();

Java Coding

- Parentheses enclose conditions in an if-statement or while statement
  - if (<condition>)
  - while (<condition>)

Java Coding

- Parentheses enclose conditions in an if-statement or while statement
  - if (<condition>)
  - while (<condition>)
- Parentheses enclose the information needed for a method
  - fire(1)
  - getEnergy()

Java Coding

- Parentheses enclose conditions in an if-statement or while statement
  - if (<condition>)
  - while (<condition>)
- Parentheses enclose the information needed for a method
  - fire(1)
  - getEnergy()
- Braces mark the start and end of a section of code
  - public void run() { … }
  - if (<condition>) { … } else { … }
  - while (<condition>) { … }

Programming Hints

- If you do not know how to do something:
  - Dabble with the code - you might work out the solution by accident!
  - Use the notes provided for commands to the robots, scanning, shooting, turning, etc…
  - Ask the demonstrators in the Lab
  - Have a look on the internet
    - Google
    - http://robocode.sourceforge.net (good resource for code)
      - API: http://robocode.sourceforge.net/docs/robocode/
    - http://www.ibm.com/developerworks/library/j-tipstrats/
    - Boolean Algebra
      - http://law.lclark.edu/live/files/9385-boolean-basics

More resources

- Some more useful info on Robocode:
  https://sic2.host.cs.st-andrews.ac.uk/robocode-extra.html

- Some basic info on java:
  https://sic2.host.cs.st-andrews.ac.uk/basic-java.html
  https://learnxinyminutes.com/docs/java/

Design a Strategy

- Decide on your strategy and tactics
  - Choose your overall strategy
    - **Aggressive** - seek and destroy
    - **Defensive** - back to the wall/find a corner
    - **Neutral** - patrol route

Design a Strategy

- Decide on your strategy and tactics
  - Choose your overall strategy
    - **Aggressive** - seek and destroy
    - **Defensive** - back to the wall/find a corner
    - **Neutral** - patrol route
  - What events do you want to react to?
    - See an enemy, hit a wall, hit an enemy, hit by a bullet, etc…

Design a Strategy

- Decide on your strategy and tactics
  - Choose your overall strategy
    - **Aggressive** - seek and destroy
    - **Defensive** - back to the wall/find a corner
    - **Neutral** - patrol route
  - What events do you want to react to?
    - See an enemy, hit a wall, hit an enemy, hit by a bullet, etc…
  - How do you want to react to them?
    - Shoot, turn, move?

Design a Strategy

- Decide on your strategy and tactics
  - Choose your overall strategy
    - **Aggressive** - seek and destroy
    - **Defensive** - back to the wall/find a corner
    - **Neutral** - patrol route
  - What events do you want to react to?
    - See an enemy, hit a wall, hit an enemy, hit by a bullet, etc…
  - How do you want to react to them?
    - Shoot, turn, move?
- Writing your overall strategy and individual tactics down in English before you start to code will help you later

University of St Andrews

School of Computer Science
University of St Andrews

Building your Robot

- Build your robot step-by-step
- Choose one tactic to start with:
  - Write the code
  - Fix any errors displayed
  - Test your robot against the sample robots
  - Check the robot is behaving as expected
  - Make any changes or improvements

- Do this for each tactic until you have built your entire strategy

University of St Andrews

School of Computer Science
University of St Andrews

Create your new Awesome Robot!

- From the Editor
  - File → New → Robot
  - Name your robot (has to start with a capital case)
  - Package name: **django**

- Remember to write the author of the robot in the comment

```
/**
 * Franky - a robot by Simone Conte
 */
```

Starting a Battle

- From the menu bar, choose: **Battle → New**
- In the dialog box that appears:
  - click on button marked Add All
  - click on button marked Next
- Set Number of Rounds to 1
- Battlefield size using buttons on the right
  - *1000x1000* is a good size
- Click the Start Battle button to the being the **WAR!**

Q/A

- **Q**: My robot is not winning with the highest score, even though it is the only one left on the battlefield. Why is that the case?
- **A**: A robot that does not fire much, but "just" saves its energy is getting a lesser score than a robot that hits other robots with a lot of bullets.

- **Q**: How fast does a bullet travel?
- **A**: A bullet travels at a speed between 11.0 and 19.7 depending on the power. The more powerful the bullet, the slower. The formula to calculate it is velocity = 20 - (3 * power)

- **Q**: Which is the range of a bullet?
- **A**: A bullet has no range. It keeps going until it hits a robot or a wall.

University of St Andrews

School of Computer Science
University of St Andrews

Rest of Session

- For the next 30 minutes
  - Design & build your robot
  - Use the slides, handout & online resources for programming tips
  - API Documentation accessible from **Robot Editor/Help → Robocode API**
  - Handout contains
    - a list of methods you are likely to use
    - example code showing loops, conditional statements, etc.

- We will be around to help.
- If you want to run something by us, put up your hand.
- For the last 15 minutes
  - Your robots will compete against each other on the big screen!