

Robocode: Programación y estrategias

9 de noviembre de 2021

1. Programación

Para crear nuestro primer tanque seleccionaremos nuestro programa ya instalado de **Robocode**, seguidamente seleccionaremos **Robot** y por último, **Source Editor** para que nos habrá el editor desde donde programar.

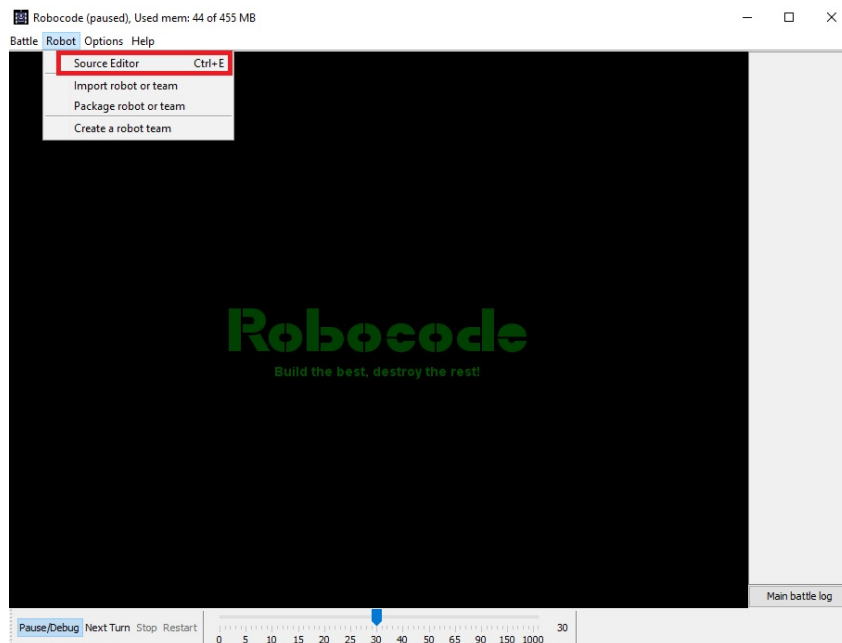
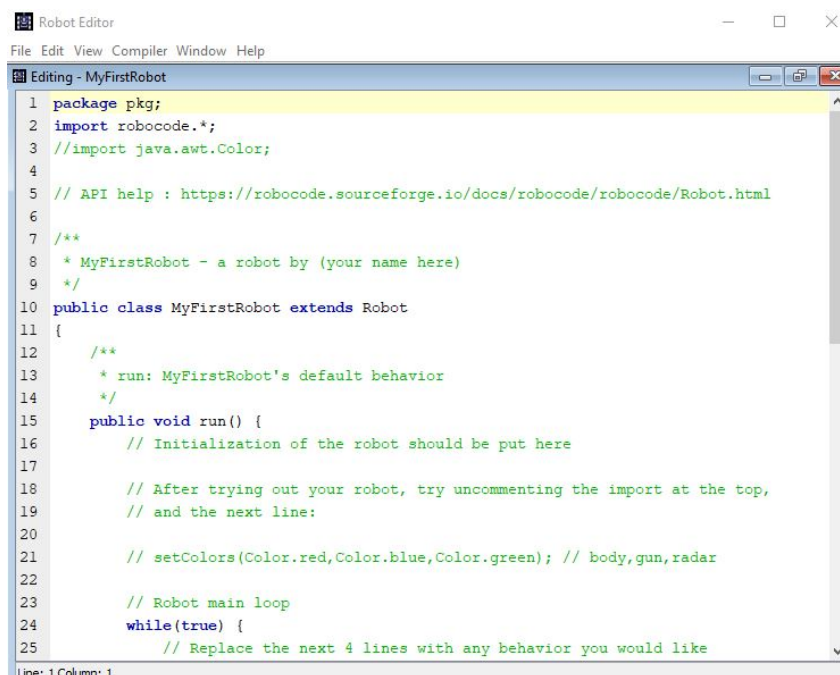


Figura 1: Selección editor

Desde el editor ya abierto seleccionaremos **File** y **New Robot** para crear nuestro primer tanque. Seguidamente, nos saltará una ventana donde nos indica que le tenemos que dar un nombre al tanque (ponemos el que queramos) y por último, nos pedirá añadir un paquete más, que este caso añadiremos el paquete **pkg**. Y el editor pasará a rellenarse de manera automática con lo siguiente:



```

1 package pkg;
2 import robocode.*;
3 //import java.awt.Color;
4
5 // API help : https://robocode.sourceforge.io/docs/robocode/robocode/Robot.html
6
7 /**
8  * MyFirstRobot - a robot by (your name here)
9  */
10 public class MyFirstRobot extends Robot
11 {
12     /**
13      * run: MyFirstRobot's default behavior
14      */
15     public void run() {
16         // Initialization of the robot should be put here
17
18         // After trying out your robot, try uncommenting the import at the top,
19         // and the next line:
20
21         // setColors(Color.red,Color.blue,Color.green); // body,gun,radar
22
23         // Robot main loop
24         while(true) {
25             // Replace the next 4 lines with any behavior you would like

```

Figura 2: Editor ya preparado

Pero antes de ver todo lo que está escrito dentro de este código vamos a ver las partes del tanque para saber cómo programarlas posteriormente en el código.

1.1. Partes del tanque

El tanque se divide en las siguientes partes mostradas en la imagen siguiente:

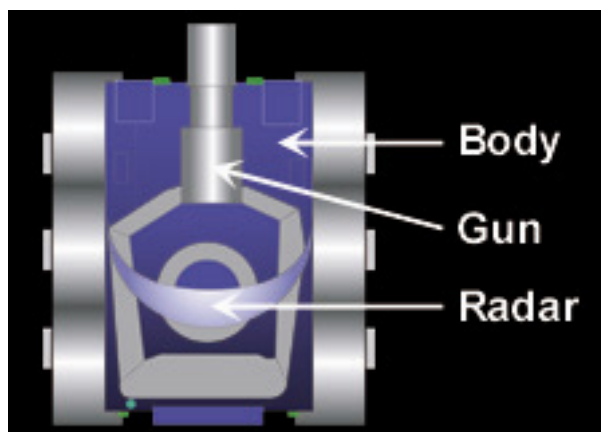


Figura 3: Partes del tanque

- **Body (Cuerpo):** Es el que lleva encima el arma y el radar. Esta parte se va a encargar de los movimientos, ya sea, de derecha, a izquierda o de arriba a abajo.
- **Gun (Arma):** Está encima del cuerpo y se encarga de disparar las balas con una energía determinada. Se podrá girar a la derecha o la izquierda. Encima de esta está el radar para detectar a los enemigos.

- **Radar:** Está encima del arma y se encarga de detectar a los otros tanques. También se podrá mover el radar y para escanear a los enemigos vamos a utilizar el comando `onScannedRobot()`.

1.2. Primeros comandos

En esta parte vamos a analizar lo que tenemos del código. Vamos a empezar con las dos primeras líneas:

Listing 1: Primeras líneas

```
1 package pkg;  
2 import robocode.*;
```

En estas dos primeras líneas se está primero importando el **pkg** necesario para una posterior importación de paquetes y para que funcionen los demás entornos y para que lo detecte como un tanque, importaremos el paquete **robocode**.

Después tendremos declarada la clase con el nombre de nuestro robot con unas acciones ya determinadas, que en este caso, se harán en todo momento:

Listing 2: Acciones en todo momento

```
1 public class MyFirstRobot extends Robot  
2 {  
3     /**  
4      * run: MyFirstRobot's default behavior  
5      */  
6     public void run() {  
7         // Initialization of the robot should be put here  
8  
9         // After trying out your robot, try uncommenting  
10        the import at the top,  
11        // and the next line:  
12  
13        // setColors(Color.red,Color.blue,Color.green); //  
14        body,gun,radar  
15  
16        // Robot main loop  
17        while(true) {  
18            // Replace the next 4 lines with any  
19            behavior you would like  
20            ahead(100);  
21            turnGunRight(360);  
22            back(100);  
23            turnGunRight(360);  
24        }  
25    }  
26 }
```

Ahora, vamos a ver qué hacen cada uno de los comandos del bucle principal (**while (true)**):

- **ahead**: Se mueve hacia delante un número determinado de píxeles que están indicados entre paréntesis. En el caso del ejemplo se mueve 100.
- **turnGunRight**: Mueve el Arma a la derecha unos grados determinados por el número que esté entre paréntesis, en este caso 360 grados.
- **back**: Se mueve hacia atrás un número determinado de píxeles que están indicados entre paréntesis. En el caso del ejemplo se mueve 100.

Seguidamente, vamos a ver qué realiza la clase declarada a continuación:

Listing 3: Código para disparar

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

Con estas líneas le estamos indicando que siempre que escanee otro tanque le dispare con una bala de fuerza 1.

La siguiente clase indica qué hacer al tanque en caso de que le haga daño una bala contraria:

Listing 4: Código en caso de daño

```
1      public void onHitByBullet(HitByBulletEvent e) {  
2          // Replace the next line with any behavior you  
3              would like  
4              back(10);  
        }
```

En este caso, si a nuestro tanque le da una bala se moverá atrás 10 píxeles.

Y por último, esta la instrucción en caso que nos demos con una pared:

Listing 5: Código en caso de dar con una pared

```
1      public void onHitWall(HitWallEvent e) {  
2          // Replace the next line with any behavior you  
3              would like  
4              back(20);  
        }
```

Para este caso, cuando se dé con una pared se moverá atrás 20 píxeles.

Dentro de las instrucciones tipo **void** podemos utilizar los siguientes comandos:

Modificador tipo void	Comando	Descripción
void	ahead(dist)	Se mueve hacia adelante según la distancia dist. en píxeles
void	back(dist)	Se mueve hacia atrás según la distancia dist. en píxeles
void	fire(pow)	Dispara una bala con la potencia indicada según pow
void	onBulletHit (BulletHitEvent event)	Dispara el evento en caso de daño por bala
void	onBulletMissed (BulletMissedEvent event)	Dispara el evento en caso que nuestra bala falle
void	onHitWall (HitWallEvent event)	Dispara el evento en caso que nuestro tanque se dé contra un muro
void	onHitRobot (HitRobotEvent event)	Dispara el evento en caso que nos demos contra otro tanque
void	onScannedRobot (ScannedRobotEvent event)	Dispara el evento en caso de que nuestro escáner detecte un tanque
void	setColors (Color bodyColor, Color gunColor, Color radarColor)	Damos color a las distintas partes del tanque
void	turnGunLeft(deg)/ turnGunRight(deg)	Gira la arma a la derecha/izquierda unos grados determinados por deg.
void	turnLeft/turnRight (deg)	Se gira a la izquierda/derecha unos grados determinados por deg.
void	turnRadarRight (deg)/turnRadarLeft(deg)	Se gira el radar a la derecha/izquierda unos determinados grados por deg.

IMPORTANTE: Para el caso de los comandos que empiezan por **on** se tiene que poner la acción entre llaves como se puede ver en los ejemplos anteriores.

Una vez que nuestro tanque esté terminado sólo tendremos que seleccionar **File**, **Save** y darle un nombre, y compilarlo con **Compiler** o con **Ctrl+B**.

2. Estrategias

En este apartado vamos a ver los distintos tipos de tanques que podemos utilizar. Estos códigos son orientativos y después se pueden modificar al gusto de cada uno e inclusive combinar varias estrategias.

2.1. Tanque pegado al muro

Este tanque irá pegado al muro y desde allí disparará a los otros tanques en el momento que los detecte:

Listing 6: Walls

```

1 package sample;
2
3
4 import robocode.DeathEvent;
5 import robocode.Robot;
6 import robocode.ScannedRobotEvent;
7 import static robocode.util.Utils.normalRelativeAngleDegrees;
8
9 import java.awt.*;
10
11
12 /**
13  * Corners -
14  * <p>
15  * This robot moves to a corner, then swings the gun back and
16  * forth.
17  * If it dies, it tries a new corner in the next round.
18  */
19 public class Corners extends Robot {
20     int others; // Number of other robots in the game

```

```
20     static int corner = 0; // Which corner we are currently
        using
21     // static so that it keeps it between rounds.
22     boolean stopWhenSeeRobot = false; // See goCorner()
23
24     /**
25      * run:  Corners' main run function.
26      */
27     public void run() {
28         // Set colors
29         setBodyColor(Color.red);
30         setGunColor(Color.black);
31         setRadarColor(Color.yellow);
32         setBulletColor(Color.green);
33         setScanColor(Color.green);
34
35         // Save # of other bots
36         others = getOthers();
37
38         // Move to a corner
39         goCorner();
40
41         // Initialize gun turn speed to 3
42         int gunIncrement = 3;
43
44         // Spin gun back and forth
45         while (true) {
46             for (int i = 0; i < 30; i++) {
47                 turnGunLeft(gunIncrement);
48             }
49             gunIncrement *= -1;
50         }
51     }
52
53     /**
54      * goCorner:  A very inefficient way to get to a corner.
55      *             Can you do better?
56      */
57     public void goCorner() {
58         // We don't want to stop when we're just turning
59         ...
60         stopWhenSeeRobot = false;
61         // turn to face the wall to the "right" of our
            desired corner.
        turnRight(normalRelativeAngleDegrees(corner -
            getHeading()));
        stopWhenSeeRobot = true;
```

```

62         // Move to that wall
63         ahead(5000);
64         // Turn to face the corner
65         turnLeft(90);
66         // Move to the corner
67         ahead(5000);
68         // Turn gun to starting point
69         turnGunLeft(90);
70     }
71
72     /**
73      * onScannedRobot: Stop and fire!
74      */
75     public void onScannedRobot(ScannedRobotEvent e) {
76         // Should we stop, or just fire?
77         if (stopWhenSeeRobot) {
78             // Stop everything! You can safely call
79             // stop multiple times.
80             stop();
81             // Call our custom firing method
82             smartFire(e.getDistance());
83             // Look for another robot.
84             // NOTE: If you call scan() inside
85             // onScannedRobot, and it sees a robot,
86             // the game will interrupt the event
87             // handler and start it over
88             scan();
89             // We won't get here if we saw another
90             // robot.
91             // Okay, we didn't see another robot...
92             // start moving or turning again.
93             resume();
94         } else {
95             smartFire(e.getDistance());
96         }
97     }
98
99     /**
100      * smartFire: Custom fire method that determines
101      * firepower based on distance.
102      *
103      * @param robotDistance the distance to the robot to fire
104      * at
105      */
106     public void smartFire(double robotDistance) {
107         if (robotDistance > 200 || getEnergy() < 15) {
108             fire(1);
109         }
110     }

```

```
102         } else if (robotDistance > 50) {
103             fire(2);
104         } else {
105             fire(3);
106         }
107     }
108
109     /**
110     * onDeath: We died. Decide whether to try a different
111       corner next game.
112     */
113     public void onDeath(DeathEvent e) {
114         // Well, others should never be 0, but better safe
115           than sorry.
116         if (others == 0) {
117             return;
118         }
119
120         // If 75% of the robots are still alive when we
121           die, we'll switch corners.
122         if ((others - getOthers()) / (double) others <
123             .75) {
124             corner += 90;
125             if (corner == 270) {
126                 corner = -90;
127             }
128             out.println("I died and did poorly...
129               switching corner to " + corner);
130         } else {
131             out.println("I died but did well. I will
132               still use corner " + corner);
133         }
134     }
135 }
```

No tenéis que copiar este código para seguir la estrategia, ya que, tenéis varias funciones para distintos casos. O podéis modificar las cifras para que su comportamiento sea distinto. Para encontrarlo, sólo tenéis que meteros en el editor y abrir el tanque **Walls**.

2.2. Movimiento aleatorio

Este tanque hará movimientos aleatorios e irá disparando según encuentre el tanque. Su código es el siguiente:

Listing 7: Crazy

```
1 package sample;
```



```

2 import robocode.*;
3 import java.awt.*;
4
5
6 /**
7  * Crazy
8  * <p>
9  * This robot moves around in a crazy pattern.
10 * /
11 public class Crazy extends AdvancedRobot {
12     boolean movingForward;
13
14     /**
15      * run: Crazy's main run function
16      */
17     public void run() {
18         // Set colors
19         setBodyColor(new Color(0, 200, 0));
20         setGunColor(new Color(0, 150, 50));
21         setRadarColor(new Color(0, 100, 100));
22         setBulletColor(new Color(255, 255, 100));
23         setScanColor(new Color(255, 200, 200));
24
25         // Loop forever
26         while (true) {
27             // Tell the game we will want to move
28             // ahead 40000 -- some large number
29             setAhead(40000);
30             movingForward = true;
31             // Tell the game we will want to turn
32             // right 90
33             setTurnRight(90);
34             // At this point, we have indicated to the
35             // game that *when we do something*,
36             // we will want to move ahead and turn
37             // right. That's what "set" means.
38             // It is important to realize we have not
39             // done anything yet!
40             // In order to actually move, we'll want
41             // to call a method that
42             // takes real time, such as waitFor.
43             // waitFor actually starts the action --
44             // we start moving and turning.
45             // It will not return until we have
46             // finished turning.
47             waitFor(new TurnCompleteCondition(this));
48             // Now we'll turn the other way...

```

```

41         setTurnLeft(180);
42         // ... and wait for the turn to finish ...
43         waitFor(new TurnCompleteCondition(this));
44         // ... then the other way ...
45         setTurnRight(180);
46         // .. and wait for that turn to finish.
47         waitFor(new TurnCompleteCondition(this));
48         // then back to the top to do it all again
49     }
50 }
51
52 /**
53  * onHitWall: Handle collision with wall.
54  */
55 public void onHitWall(HitWallEvent e) {
56     // Bounce off!
57     reverseDirection();
58 }
59
60 /**
61  * reverseDirection: Switch from ahead to back & vice versa
62  */
63 public void reverseDirection() {
64     if (movingForward) {
65         setBack(40000);
66         movingForward = false;
67     } else {
68         setAhead(40000);
69         movingForward = true;
70     }
71 }
72
73 /**
74  * onScannedRobot: Fire!
75  */
76 public void onScannedRobot(ScannedRobotEvent e) {
77     fire(1);
78 }
79
80 /**
81  * onHitRobot: Back up!
82  */
83 public void onHitRobot(HitRobotEvent e) {
84     // If we're moving the other robot, reverse!
85     if (e.isMyFault()) {
86         reverseDirection();

```

```

87         }
88     }
89 }
```

Los códigos aquí mostrados son orientativos y es para, que podáis adentraros y averiguar más sobre nuevas estrategias.

RECOMENDACIÓN: En caso de conocer qué tipo de clase es se recomienda utilizar **Help** y ver el **Robocode API** para ver cómo se estructuran.

2.3. Puntuación

Una vez se termine el combate aparecerá la siguiente tabla de puntuación:

Results for 10 rounds											
Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	sample.Tracker	2275 (46%)	650	120	1268	123	115	0	6	1	3
2nd	sample.RamFire	1820 (37%)	350	0	1023	18	308	121	0	7	3
3rd	sample.MyFirstRo...	873 (18%)	500	80	254	18	20	0	4	2	4
Save								OK			

Donde se muestran las siguientes estadísticas:

- **Total score:** Puntuación total obtenida.
- **Survival Score:** Se suman 50 puntos cada vez que se mata a otro tanque.
- **Last Survivor Bonus:** El último tanque en pie se le suman 10 puntos adicionales.
- **Bullet Damage:** Cada tanque suma 1 punto cada vez que una de sus balas da al contrario.
- **Bullet Damage Bonus:** Cuando un tanque mata a otro enemigo se le suma un 20 % a la puntuación obtenida.
- **Ram Damage:** El tanque suma 2 puntos cada vez que causa daño a más de un enemigo.
- **Ram Damage Bonus:** Cuando un tanque es destruido por daño colateral se le suma un 30 % al que le ha destruido.
- **Posiciones:** Muestra la clasificación de los tanques.