ROBOCODE: ¿Un robot inteligente?

INTELIGENCIA EN REDES DE COMUNICACIONES

Eduado Salagaray Cepeda
Francisca Merchán Higuera
(5º Ingeniería de Telecomunicación)

Contenidos

1. INTRODUCCIÓN

RoboCode es un simulador de combates entre tanques desarrollado por IBM Alphaworks. El tanque debe recorrer el entorno para evitar ser disparado por sus oponentes y evitar también chocar contra las paredes. Además, para ganar, el tanque debe localizar a su adversario y dispararle. Como todos los simuladores, RoboCode recoge algunos aspectos de la realidad, pero obvia otros.

ASPECTOS RECOGIDOS:

- Las actuaciones llevan tiempo; mientras rotas te pueden disparar.
- Las balas tardan en llegar. Si disparar a un objetivo que se mueve, debes tener en cuenta dónde estará cuando la bala llegue.
- El sensor unidireccional debe apuntar al adversario para verlo.
- El cañón se calienta al disparar y debe enfriarse antes del próximo disparo.
- Chocar contra los obstáculos debilita

ASPECTOS NO REALISTAS:

- Los sensores y el resto de los componentes son silenciosos.
- Los sensores detectan velocidad, posición, orientación y energía restante.
- Los combates tienen lugar en un entorno plano (2D) y cerrado.

1.1 Reglas del juego

Los tanques RoboCode están escritos como programas Java. Se dirigen por eventos; hay un bucle principal, que puede ser interrumpido por un número de manejadores de eventos que incluyen acciones como:

- ver al adversario
- ser alcanzado por una bala
- chocar con un adversario
- y más...

Los tanques comienzan con una determinada cantidad de energía, que se va perdiendo por:

- recibir un disparo
- chocar contra algo
- por disparar a balas

También, si el juego dura un tiempo, todos los robots pierden energía hasta que uno de ellos muere. Esto impide que la partida dure infinitamente. Cuando dos robots colisionan, ambos pierden lo mismo, por eso, chocar puede ser una buena estrategia para un robots que está a punto de ganar.

Sólo se gana energía de una manera: disparando al adversario. Si un robot se queda sin energía debido a su tiroteo, se inhabilita. Si una bala da a su enemigo después, algo de energía se le devuelve. Pero si un robot se queda sin energía por alguna otra razón –como ser alcanzado por una bala o por chocar contra la pared- entonces, muere.

1.1.1 Acciones

Un robot puede rotar su cuerpo entero, la torreta cañon/radar, o el radar solo. Todas las rotaciones llevan su tiempo, pero rotar el cuerpo entero lleva más tiempo y rotar el radar solo es lo más rápido. Además, los robots pueden moverse hacia delante o hacia atrás, a un ritmo fijo de aceleración o con una velocidad fija.

Un robot puede disparar su cañón con distintas potencias: si usa más, necesita más energía, pero hace más daño. Como ejemplo, aquí tenemos un programita que rota el robot diez grados, luego lo mueve hacia delante 45 unidades, luego rota el cañón diez grados en otra dirección y por último dispara a su máxima potencia:

```
left(10);
forward(45);
gunRight(10);
fire(3);
```

Cada robot es implementado como un hilo y el programa puede ser interrumpido en varios puntos. Suele ser un error escribir largas secuencias de acciones, puesto que típicamente algún evento interrumpirá el hilo tarde o temprano. No obstante los robots no son entes puramente reactivos.

1.1.2. Sensores

Todos los robots están equipados con un único sensor que tiene un rango infinito pero que sólo puede escanear oponentes dentro de la anchura de un grado. El radar sensor es la única manera que tiene el robot de obtener información sobre su adversario. El radar devuelve la posición del enemigo, su orientación, su ángulo de cañón y su energía. El robot también es consciente de su posición, su orientación, su energía y de la orientación de su cañón y su radar.

1.2 Estrategias

Debido a la gran popularidad de RoboCode han surgido muchos tipos de estrategias. Muchos robots intentan seguir las pared para reducir el rango que su radar necesita escanear para detectar a su oponente. Otros robots van hacia delante y hacia atrás en un movimiento errático para confundir al sistema de apuntamiento de su adversario. Otros se centran en técnicas de reconocimiento de patrones, para intentar aprender los patrones de movimiento de sus oponentes de manera que puedan predecir su movimiento. Por último, los rastreadores persiguen a su enemigo, llegando incluso a chocar contra él para producirle un daño adicional.

2. EQUIPOS

Una de las más recientes vueltas de tuerca de RoboCode es la introducción del combate con equipos. Los equipos están formados por dos tipos de robots: los “leaders” y los “droids” (tambIén pueden estar formador por tanques normales). Los líderes están equipados con un escáner, y pueden enviar y recibir mensajes. Los droid no tienen mensajes y sólo pueden recibir mensajes. Los equipos exitosos protegen al líder que se comunica con sus droid y coordina su comportamiento.

3. ESTADO DEL ARTE

Robocode ha generado un gran interés debido a su adictividad. En Internet hemos encontrado una página “wiki” que se va enriqueciendo con las aportaciones de todos los freakies de Robocode.
Algunas áreas estructuradas en este sitio, y por tanto, representativas del fenómeno Robocode, son las siguientes:

3.1 Movimiento.

Algunas técnicas de movimiento:

- LinearMovement - Moving in a straight line. Vulnerable to LinearTargeting.

- Oscillators - Moving in a set pattern, which repeats (although the period can be hundreds of ticks long). Vulnerable to PatternMatching.
- Avoiders
- RandomMovement - Moves in a randomly determined way, often combined with other techniques to avoid moving in undesirable ways. Check RandomMovementBot for an implementation.
- Cannibals - Copy the direction of the enemy (often using MirrorMovement). Vulnerable to AntiMirrorTargeting, and to bad enemy movement.
- Antigravity - Works by imagining forces repeling your robot from the other robots, and the walls. Works well in Melee.
- PerpendicularMovement - A general term describing a lot of movements systems. They move at right angles to the enemy so their bearing changes as much as possible.
- ProvocativeMovement - Moving in a way that aids your Targeting, rather than harming your enemy's.
- ChaoticMovement - Similar to RandomMovement, in that is moves in a completely unpredictable way.
- CircleBots - Moves in circles. Vulnerable to CircularTargeting.
- WallBots - Moves along the walls. Vulnerable to LinearTargeting, except when near corners.
- RamBots - Moves to hit the enemy. Vulnerable to most targeting, but gets extra ram bonus.
- ArbitraryMovement - Moves in an arbitary, but often unpredictable way.
- NeuralReinforcementLearningMovement - Moves in a learned fashion, using a NeuralNetwork
- MinimumRiskMovement - Picks a selection of possible moves, and follows the run with least risk (ie. furthest from enemies)
- OrbitalMovement? - circles around the enemy. A type of PerpendicularMovement.
- BombShieltering - in melee, hides behind weak enemy bots to avoid being hit

Movimiento avanzado

- DynamicDistancing - selecting a distance from the enemy using past experience.
- /FlatMovementProfile - trying to move to each GuessFactor equally.
    - SandboxFlattener
- Track Movement
- WaveSurfing

FuturePosition - a code snippet to work out where your movement system will take you.

3.2_ Selección del objetivo.

1. HeadOnTargeting
2. LinearTargeting
3. CircularTargeting
4. AngularTargeting
5. AngularTargeting/Factored
6. PatternMatching
7. NeuralTargeting
8. RandomTargeting
9. GuessFactorTargeting
10. StatisticalTargeting
11. BestPSpace
12. LaserTargeting
13. AveragedBearingOffsetTargeting
14. ChaseBullets
15. EscapeEnvelope
16. ZoomTargeting

3.3 Gestión de la Inteligencia

- WhatInfoToCollect
- WhatToSaveBetweenRounds
- WhatToSaveBetweenBattles
- SavingDataHowto
- ReducingDataFileSize
- CompressedSerialization
- GeneticProgramming

3.4 Otros...

4. PROGRAMACIÓN GENÉTICA aplicada a la programación de ROBOCODE

La técnica de una programación genética en lugar de un robot programado a mano, pretende obtener buenos robots a través de mecanismos de evolución. Se comienza con una población generada aleatoriamente que, en general, no hace gran cosa. De ahí se toman lo mejores, que se evalúan en la siguiente generación. La mutación y el cruce son los responsables de que aparezcan nuevos individuos (que son fuente de mejoras). En cada generación, el comportamiento medio debe mejorar. A lo largo del tiempo, van surgiendo mejores programas. La idea de la programación genética es que, en lugar de codificar un controlador de Robocode a mano, puedes producir tanques muy eficaces de manera automática usando la evolución.

Si se quieren evolucionar tanques Robocode, hay unas cuantas cosas que hacer. Procedamos a una descripción más detallada en los siguientes apartados.

4. 1 *Decidir La Representación*

Los algoritmos genéticos pueden tratar cualquier secuencia de bits como si fuera un genoma. La complicación viene en decidir qué significa esa secuencia de bits, es decir, en cómo interpretar el genoma. La solución más simple hubiera sido tratar cada genoma como código Java e intentar compilarlo y ejecutarlo. Pero la aplastante mayoría de los programas no hubieran llegado a compilar, así que esto no hubiera funcionado.
En cambio, se puede crear un lenguaje de control propio, con su propia semántica, que asegure que las mutaciones y los cruces genéticos siguen dado cadenas de bits con significado. Para más información remitimos a nuestras referencias.

4.2 *Evaluación*

El modo más obvio para evaluar un robot es combatir contra otros robots. Sin embargo, todo este acercamiento tiene un inconveniente: el tiempo. Incluso en ordenadores rápidos, hacer toda esta tarea lleva mucho, mucho tiempo por lo que a veces no es viable hacer la mejor evaluación, puesto que tardaría mucho.

4.3 *Posiciones Iniciales*

Hemos observado que el resultado de las batallas es muy sensible a la posición inicial de los tanques. Un robot inferior puede ganar si tiene suerte y empieza apuntando a la espalda de su adversario. Si se decide permitir posiciones iniciales aleatorias, se necesitarán muchas batallas para obtener una buena evaluación –quizás 25 o más-. Pero esto supone retardar la obtención de resultados considerablemente.

Para evitar esto, se pueden intentar un par de cosas. Descompilar el corazón de Robocode y modificarlo para poder elegir las posiciones iniciales manualmente. Hacer esto no es muy difícil, y está dentro de las reglas de la licencia si no se distribuye el código modificado. Se han hecho pruebas de robots con una única posición inicial: cada robot empieza en una esquina, mirando 90º de su adversario. Esto tuvo el resultado esperado de acelerar las cosas, pero al final los robots que habían evolucionado eran incapaces de ganar en ninguna otra posición inicial.

4.4 *Adversarios*

Lo primero que se puede intentar es poner a combatir a los robots que van evolucionando entre ellos, pero esto no funciona: los robots no hacen nada. Lo que se puede hacer es intentar evaluar a los nuevos robots con los que ya vienen con Robocode y también con robots de Robocode repository (ver rereferencias).
Aquí también hay una decisión que tomar: evaluar tus robots con un solo adversario o con múltiples. Cuando se utiliza sólo un adversario, el tanque sufre el riesgo de no ser capaz de vencer a otro tipo de robots. Pero si utilizamos múltiples adversarios, el proceso se ralentiza considerablemente, especialmente si se combina con diferentes posiciones iniciales.

4.5 *Conclusiones*

En este apartado queríamos poner sobre el papel cómo se hubiera hecho una evolución genética de los robots. Sin embargo, el tiempo necesario para las simulaciones en sí, sumado al tiempo y complejidad de diseño de las mismas y de la semántica de los genes superaba con creces los propósitos de esta asignatura. Dejamos esta línea de trabajo abierta para futuros proyectos, e invitamos a consultar el trabajo realizado por Jacob Eisenstein del MIT.

5. PROCESO DE DISEÑO DE NUESTROS ROBOTS

A continuación pasaremos a relatar con más o menos detalle el proceso de creación tanto de nuestro robot como de nuestro equipo. Como cabe esperar, comenzamos pensando sobre un simple robot.

Lo primero que nos vino a la cabeza fue si conocíamos algún método o estrategia que pudiese modelar el comportamiento del robot. Debido a las asignaturas que estamos cursando en la carrera, pensamos alguna forma de predecir el comportamiento del oponente. El gran problema para la implementación de esta alternativa fue la falta de tiempo que tendríamos para obtener los suficientes datos para el procesamiento (para más información ver apartado referente a la programación evolutiva).

Una vez llegados a este punto decidimos que el modelado del robot se basara en ideas surgidas de nuestra cabeza, sin tener ninguna base ni conocimiento científico en el que apoyarnos. Fijada esta idea, pensamos y discurrimos sobre lo que haríamos si nosotros fuésemos el robot. Utilizando en su mayoría nuestra basta experiencia de videojuegos de índole parecida, encontramos varias ideas de actuación.

La primera estrategia de el movimiento del tanque fue dar vueltas alrededor del enemigo más cercano mientras le disparábamos. La idea en sí era buena pero tenía el problema que si el enemigo nos evitaba llegaba momentos en que nos encontrábamos muy lejos. Para evitar este problema hacíamos que siempre nos encontráramos dentro de un margen de distancia respecto al enemigo; la distancia siempre era mayor que un valor X y menor que un valor Y.

Una vez implementada esta opción intentamos mejorarla. Para ello pasamos de heredar de la clase Robot a la clase AdvancedRobot. Con esto conseguimos una fluidez y rapidez de movimientos notablemente mejor.

A base de competir y probar nuestro robot contra otros robots, vimos que se podía quedar encajonado en una esquina. Para evitarlo decidimos que si el tiempo transcurrido entre dos choques consecutivos entre

dos paredes era menor a una cantidad X, nuestro robot se movería imitando a un coche para salir de un aparcamiento (primero hacia atrás girando a la derecha y luego avanzando).

Otra mejora que realizamos fue el hecho de modificar la potencia de disparo. Mientras más lejos estuviese el enemigo, menos potencia tendría el disparo que realizaríamos.

Como última modificación que hicimos a este modelo fue el hecho de intentar predecir la posición futura del enemigo. Basándonos en la distancia del enemigo, predecimos con una función cuanto tiempo pasara desde que disparamos hasta que el enemigo recibiría la bala. Esta función es creciente (mientras más lejos estuviese el enemigo, dicho tiempo sería mayor) siendo implementada a base de experimentación. Una vez "conociésemos" este tiempo, sabiendo la orientación y velocidad del enemigo disparamos a su posible posición futura.

Llegados a este punto decidimos implementar otro robot para evaluar el primero. Nuestro segundo robot se mueve hacia derecha e izquierda realizando una especie de esquiva mientras disparaba hacia el enemigo. Este movimiento duraba una cantidad fija y más otra aleatoria.

Una vez pusimos a combatir a nuestros robots, vimos que ninguno era superior al otro de forma clara. Este hecho junto a que ambos poseían unos movimientos bastantes predecibles hizo que nos decidiésemos a hacer una mezcla.

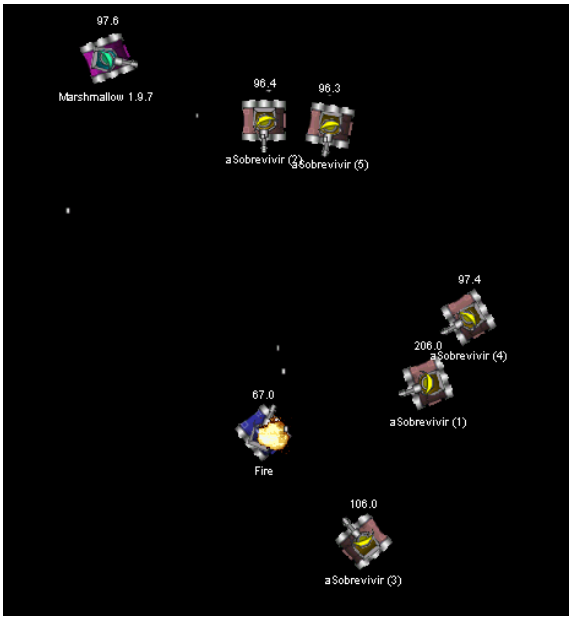
El robot "definitivo" se mueve a derecha e izquierda mientras dispara al enemigo. Si el enemigo descubre su movimiento y logra acertar 4 disparos, nuestro robot empieza a moverse en círculos durante X segundos volviendo finalmente a su movimiento de esquiva. Esta simple variación lograr que sea mucho más complicado que otro robot deduzca nuestro algoritmo de movimientos.

Una vez construido el robot base, a la hora de formar un equipo se nos ocurrió crear un grupo de cinco tanques "básicos" donde uno de ellos sería el líder que fijaría un único enemigo al que atacasen todos. Este proceso se repetirá hasta acabar con todos los enemigos. Si muriese nuestro líder, otro tomaría su papel.

Para probarlo lo hicimos luchar con el de ejemplo del robocode con resultados muy satisfactorios. A pesar de todo decidimos hacer otro equipo de prueba. En este caso eran nuestros robots que funcionaban como células independientes. Cada uno liberaba su guerra por separado.

Para nuestra sorpresa, este grupo de robots independientes era mejor que el organizado, así que optamos finalmente por él. Esto creemos que es debido a que los robots que están muy lejos del objetivo no influyen casi en la destrucción del enemigo (potencia de fuego mínima) convirtiéndose así en "simples espectadores de la acción".

Una vez decido el equipo de robot, intentamos evitar un problema; notificar a un compañero de equipo que estaba en nuestra línea de fuego. Tras varios intentos de comunicaron entre robot, vimos que era más simple y eficiente que, si nuestro robot recibía un disparo de un compañero, se moviese para evitarlo. Esto es debido a que como nuestros robots están en constante movimiento cuando que se procesaba la información y se disparaba, la posición enviada ya no era "válida".



6. *CÓDIGO*

6. 1 *Combate individual*

```
package Edu_Paqui.Un_Robot;
   import robocode.*;
   import java.awt.Color;

/**
 * Socorro - Robot diseñado por: Eduardo Salagaray Ceepda
 *                                Paqui Merchán Higuera
 * Este robot intenta esquivar disparos recorriendo pequeños arcos
 * de circunferencia. En caso de ser impactado por las balas, se
 * deplaza de posición utilizando un arco de circunferencia.
*/
   public class Socorro extends AdvancedRobot
   {
   // Variables utilizadas en la ejecucción;

   //Constante matemática utilizada para los procesos de calculos.

      final double pi = Math.PI;

   //Variable utilizada para saber la ultima vez que chocamos con una pared.
   //Aqui se almacena un instante de tiempo
      long inCorner;

   //Objeto Enemy donde se almacenara el enemigo al que atacaremos.
      Enemy target;

   //Numero de balas que han impactada en nuestro robot
      int hit = 0;
```

```
      //Entero que nos indica la direccion con la que debemos movernos
      int direccion = 1;

   //Potencia con la que disparamos
      double potencia;


   //Hilo principal de ejecución de nuestro robot
      public void run() {
      //Imponemos un toque de distinción a nuestro tanque
         setColors(Color.PINK,Color.ORANGE,Color.YELLOW);

      //Inicializamos variables
         inCorner=getTime();
         target = new Enemy();

      //Este valor es inicalizado a este valor para que sea muy grande
         target.distance = 900000000;

      //Hacemos que los movimientos del radar, cañon y robot sean
independientes.
         setAdjustGunForRobotTurn(true);
         setAdjustRadarForGunTurn(true);
         turnRadarRightRadians(2*pi);

         while(true) {

            //Calcula el proximo movimiento
            calculoMovimiento();

            //Calcula la potencia de disparo
            calculoPotencia();

            //Buscamos enemigos
            escanear();

            //apuntamos al enemigo
            apuntar();

            //Calculado ya todo, disparamos
            fire(potencia);

            //Executa todas los recxultados de los metodos anteriores.
            execute();

         }
      }


   //Calculamos movimiento a realizar
      void calculoMovimiento() {
      //Si estamos mas lejos de 300, nos acercamos con un angulo de 45 grados.
         if(target.distance <300){
         //Si ha pasado un tiempo determinado des el ultimo movimiento;
            if (getTime()%20 == 0)  {

            //Establecemos la direccion del movimiento
               if(hit<4){
               //Cambiamos de direccion;
```

```
                direccion *= -1;
            }
            else{
            //Si nos impactado 4 veces, describimos un arco de
circunferencia durante
            //un tiempo determinado
               if (getTime()%60 == 0)  {
                  hit = 0;
               }
            }
         //Una vez estableciada la direccion, avanzamos con el robot
            setAhead(direccion*(350+(int)((int)Math.random()*350)));
         }

      //Establecmos el redio de giro respecto al robot
         setTurnRightRadians(target.bearing + (pi/2));
      }
      else{
         setAhead(300);
         setTurnRightRadians(target.bearing + (pi/4));
      }

   }

//Metodo para contar el numero de impactos recibidos
   public void onHitByBullet(HitByBulletEvent event) {
      hit = hit +1;
   }

//Metodo por si golpeamos una pared
   public void onHitWall(HitWallEvent event) {

      //Obtenemos el instante de tiempo en que ha courrido todo;
      long temp = getTime();

      //Si ha pasado muy poco tiempo desde el ultimo choque
      if ((temp - inCorner) < 100){
            //Nos girmos hacia el nemigo y avanzamos hacia el
         setBack(100);
         setTurnRightRadians(target.bearing);
         execute();
         setAhead(300);
         execute();
      }
         //Actualizamos la variable para saber la utlima vez que chocamos
con la pared
      inCorner=temp;

   }


//Calculamos el movimiento del radar para apuntar al enemigo
   void escanear() {
      double radarOffset;

      if (getTime() - target.ctime > 5) {
      //Si hacemo mucho que no vamos a nadie, hacemos que mire hacia todas
partes.
         radarOffset = 360;
```

```
         }
         else {

         //Calculamos cuanto se debe mover el radar para apuntar al enemigo
            radarOffset = getRadarHeadingRadians() -
absbearing(getX(),getY(),target.x,target.y);

         //Como hasta que ejecutemos esta funcion puede var a pasar un cierto
tiempo
         //determinado, añadiremos una pequeña cantidad al offset para no
perder al enemigo
            if (radarOffset < 0)
               radarOffset -= pi/7;
            else
               radarOffset += pi/7;
         }

      //Gira el radar
         setTurnRadarLeftRadians(NormaliseBearing(radarOffset));
      }

   //Metodo para apuntar el cañon
      void apuntar() {

      //En esta variable, intentamos estimar el tiempo que pasara hasta que
nuestra bala llegara al enemigo
         long time = getTime() + (int)(target.distance/(20-
(3*(400/target.distance))));

      //Offset que debemos imponer al cañon
         double gunOffset = getGunHeadingRadians() -
absbearing(getX(),getY(),target.guessX(time),target.guessY(time));
         setTurnGunLeftRadians(NormaliseBearing(gunOffset));
      }


   //Metodo para obtener un angulo entre pi y -pi
      double NormaliseBearing(double ang) {
         if (ang > pi)
            ang -= 2*pi;
         if (ang < -pi)
            ang += 2*pi;
         return ang;
      }

   //Metodo para obtener un angulo entre 0 y 2*pi
      double NormaliseHeading(double ang) {
         if (ang > 2*pi)
            ang -= 2*pi;
         if (ang < 0)
            ang += 2*pi;
         return ang;
      }

   //Devuelve la distancia entre dos puntos
      public double distancia( double x1,double y1, double x2,double y2 )
      {
         return Math.sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) );
      }
```

```
   //Obtiene el angulo entre dos puntos
      public double absbearing( double x1,double y1, double x2,double y2 )
      {
         double xo = x2-x1;
         double yo = y2-y1;
         double h = distancia( x1,y1, x2,y2 );
         if( xo > 0 && yo > 0 )
         {
            return Math.asin( xo / h );
         }
         if( xo > 0 && yo < 0 )
         {
            return Math.PI - Math.asin( xo / h );
         }
         if( xo < 0 && yo < 0 )
         {
            return Math.PI + Math.asin( -xo / h );
         }
         if( xo < 0 && yo > 0 )
         {
            return 2.0*Math.PI - Math.asin( -xo / h );
         }
         return 0;
      }


   //Metodo para buscar nuevo enemigo y actualizar la variable target
      public void onScannedRobot(ScannedRobotEvent e) {
      //Entramos si hemos detectado a nuestro objetivo o a un enemigo
      //que se encuentre mas cerca que nosotros
         System.out.println("Escaneo a :" + e.getName());
         if ((e.getDistance() < target.distance)||(target.name ==
e.getName())) {
            System.out.println("He entrado");
         //Fijamos los paremetros del enemigo
            target.name = e.getName();
            target.bearing = e.getBearingRadians();
            target.head = e.getHeadingRadians();
            target.ctime = getTime();
            target.speed = e.getVelocity();
            target.distance = e.getDistance();
            double absbearing_rad =
(getHeadingRadians()+e.getBearingRadians())%(2*pi);
            target.x = getX()+Math.sin(absbearing_rad)*e.getDistance();
            target.y = getY()+Math.cos(absbearing_rad)*e.getDistance();
         }
      }

      //Método para calcular la potencia de disparo
      void calculoPotencia() {
      //Dependiendo la distancia al enemigo, diparo con mas o menos potencia.
      //La potencia es inversamente proporcional a la distancia.
         potencia = 500/target.distance;
      }

      //Cuadno muere un robot, hago que mi distancia al enemigo sea muy
grande.
      public void onRobotDeath(RobotDeathEvent e) {
```

```
            if (e.getName() == target.name)
                target.distance = 9000000;
        }

    //Si nos golpeamos con un robot, le atacamos
        public void onHitRobot(HitRobotEvent event) {
            if (event.getName() != target.name)
                target.distance = 9000000;
        }

    //Gira el cañon cuando gana
        public void onWin(WinEvent event) {
            while(true){
                execute();
                setTurnGunLeftRadians(pi/2);
            }
        }

    }

//Clases que contiene las caracteristicas del enemigo
    class Enemy {

        //Nombre del enemigo
        String name;
        //Almacena la orientacion
        public double bearing;
        //Almacena el apuntamiento
        public double head;
        //Tiempo en que ha sido detectado
        public long ctime;
        //Velocidad del enemigo
        public double speed;
        //Posicion del enemigo
        public double x,y;
        //Distancia del enemigo
        public double distance;

    //Métodos para obtener y modificar las propiedades
        public String getname(){
            return name;
        }

        public double getbearing(){
            return bearing;
        }

        public double gethead(){
            return head;
        }

        public long getctime(){
            return ctime;
        }

        public double getspeed(){
            return speed;
        }
```

```
      public double getx(){
         return x;
      }

      public double gety(){
         return y;
      }

      public double getdistance(){
         return distance;
      }

   //Metodos para intentar adivinar la posicion futura del tanque.
      public double guessX(long when)
      {
         long diff = when - ctime;
         return x+Math.sin(head)*speed*diff;
      }
      public double guessY(long when)
      {
         long diff = when - ctime;
         return y+Math.cos(head)*speed*diff;
      }

   }
```

6.2 Combate por equipos

```
package Edu_Paqui.Equipo;
   import robocode.*;
   import java.awt.Color;

/**
 * aSobrevivir - Robot diseñado por: Eduardo Salagaray Ceepda
 *                                                  Paqui Merchán Higuera
 * Este robot intenta esquivar disparos recorriendo pequeños arcos
 * de circunferencia. En caso de ser impactado por las balas, se
 * deplaza de posición utilizando un arco de circunferencia.
 * Actuara independientemente pero tendrá un objetivo común con otros
 * 4 robots idénticos. Entre ellos no se atacarán.
*/
   public class aSobrevivir extends TeamRobot
   {
   // Variables utilizadas en la ejecucción;

   //Constante matemática utilizada para los procesos de cálculos.

      final double pi = Math.PI;

   //Variable utilizada para saber la última vez que chocamos con una pared.
   //Aqui se almacena un instante de tiempo
      long inCorner;

   //Objeto Enemy donde se almacenara el enemigo al que atacaremos.
      Enemy target;

   //Numero de balas que han impactada en nuestro robot
```

```
      int hit = 0;

      //Entero que nos indica la direccion con la que debemos movernos
      int direccion = 1;

   //Potencia con la que disparamos
      double potencia;


   //Hilo principal de ejecución de nuestro robot
      public void run() {
      //Imponemos un toque de distinción a nuestro tanque
         setColors(Color.PINK,Color.ORANGE,Color.YELLOW);

      //Inicializamos variables
         inCorner=getTime();
         target = new Enemy();

      //Este valor es inicalizado a este valor para que sea muy grande
         target.distance = 900000000;

      //Hacemos que los movimientos del radar, cañon y robot sean
independientes.
         setAdjustGunForRobotTurn(true);
         setAdjustRadarForGunTurn(true);
         turnRadarRightRadians(2*pi);

         while(true) {

            //Calcula el próximo movimiento
            calculoMovimiento();

            //Calcula la potencia de disparo
            calculoPotencia();

            //Buscamos enemigos
            escanear();

            //apuntamos al enemigo
            apuntar();

            //Calculado ya todo, disparamos
            fire(potencia);

            //Ejecuta todas los resultados de los métodos anteriores.
            execute();

         }
      }


   //Calculamos movimiento a realizar
      void calculoMovimiento() {
      //Si la distancia es mayor a 300, nos acercamos
         if(target.distance <300){
         //Si ha pasado un tiempo determinado des el ultimo movimiento;
            if (getTime()%20 == 0)  {

            //Establecemos la dirección del movimiento
```

```
               if(hit<4){
               //Cambiamos de direccion;
                  direccion *= -1;
               }
               else{
               //Si nos impactado 4 veces, describimos un arco de
circunferencia durante
               //un tiempo determinado
                  if (getTime()%60 == 0)  {
                     hit = 0;
                  }
               }
            //Una vez estableciada la direccion, avanzamos con el robot
               setAhead(direccion*(350+(int)((int)Math.random()*350)));
            }

         //Establecmos el redio de giro respecto al robot
            setTurnRightRadians(target.bearing + (pi/2));
         }
         else{
            setAhead(300);
            setTurnRightRadians(target.bearing + (pi/4));
         }

      }

   //Método para contar el numero de impactos recibidos
      public void onHitByBullet(HitByBulletEvent event) {
         hit = hit +1;

      //Si el impacto viene de un compañero, nos movemos.
         if (isTeammate(event.getName()))
         {
            hit = 1000;
         }

      }

   //Metodo por si golpeamos una pared
      public void onHitWall(HitWallEvent event) {

         //Obtenemos el instante de tiempo en que ha ocurrido todo;
         long temp = getTime();

         //Si ha pasado muy poco tiempo desde el ultimo choque
         if ((temp - inCorner) < 100){
               //Nos girmos hacia el nemigo y avanzamos hacia él
            setBack(100);
            setTurnRightRadians(target.bearing);
            execute();
            setAhead(300);
            execute();
         }
            //Actualizamos la variable para saber la utlima vez que chocamos
con la pared
         inCorner=temp;

      }
```

```
   //Calculamos el movimiento del radar para apuntar al enemigo
      void escanear() {
         double radarOffset;

         if (getTime() - target.ctime > 5) {
         //Si hace mucho que no vemos a nadie, hacemos que mire hacia todas
partes.
            radarOffset = 360;
         }
         else {

         //Calculamos cuánto se debe mover el radar para apuntar al enemigo
            radarOffset = getRadarHeadingRadians() -
absbearing(getX(),getY(),target.x,target.y);

         //Como hasta que ejecutemos esta función pasará un cierto tiempo,
         //añadiremos una pequeña cantidad al offset para no perder al enemigo
            if (radarOffset < 0)
               radarOffset -= pi/7;
            else
               radarOffset += pi/7;
         }

      //Gira el radar
         setTurnRadarLeftRadians(NormaliseBearing(radarOffset));
      }

   //Método para apuntar el cañon
      void apuntar() {

      //En esta variable, intentamos estimar el tiempo que pasara hasta que
nuestra bala llegara al enemigo
         long time = getTime() + (int)(target.distance/(20-
(3*(400/target.distance))));

      //Offset que debemos imponer al cañon
         double gunOffset = getGunHeadingRadians() -
absbearing(getX(),getY(),target.guessX(time),target.guessY(time));
         setTurnGunLeftRadians(NormaliseBearing(gunOffset));
      }


   //Metodo para obtener un angulo entre pi y -pi
      double NormaliseBearing(double ang) {
         if (ang > pi)
            ang -= 2*pi;
         if (ang < -pi)
            ang += 2*pi;
         return ang;
      }

   //Metodo para obtener un angulo entre 0 y 2*pi
      double NormaliseHeading(double ang) {
         if (ang > 2*pi)
            ang -= 2*pi;
         if (ang < 0)
            ang += 2*pi;
         return ang;
```

```
      }

   //Devuelve la distancia entre dos puntos
      public double distancia( double x1,double y1, double x2,double y2 )
      {
         return Math.sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) );
      }

   //Obtiene el angulo entre dos puntos
      public double absbearing( double x1,double y1, double x2,double y2 )
      {
         double xo = x2-x1;
         double yo = y2-y1;
         double h = distancia( x1,y1, x2,y2 );
         if( xo > 0 && yo > 0 )
         {
            return Math.asin( xo / h );
         }
         if( xo > 0 && yo < 0 )
         {
            return Math.PI - Math.asin( xo / h );
         }
         if( xo < 0 && yo < 0 )
         {
            return Math.PI + Math.asin( -xo / h );
         }
         if( xo < 0 && yo > 0 )
         {
            return 2.0*Math.PI - Math.asin( -xo / h );
         }
         return 0;
      }


   //Método para buscar nuevo enmigo y actualizar la variable target
      public void onScannedRobot(ScannedRobotEvent e) {
      //Entramos si hemos detectado a nuestro objetivo o a un enemigo
      //que se encuentre mas cerca que nosotros

         if (isTeammate(e.getName()))
         {
            return;
         }

         System.out.println("Escaneo a :" + e.getName());
         if ((e.getDistance() < target.distance)||(target.name ==
e.getName())) {
            System.out.println("He entrado");
         //Fijamos los paremetros del enemigo
            target.name = e.getName();
            target.bearing = e.getBearingRadians();
            target.head = e.getHeadingRadians();
            target.ctime = getTime();
            target.speed = e.getVelocity();
            target.distance = e.getDistance();
            double absbearing_rad =
(getHeadingRadians()+e.getBearingRadians())%(2*pi);
            target.x = getX()+Math.sin(absbearing_rad)*e.getDistance();
            target.y = getY()+Math.cos(absbearing_rad)*e.getDistance();
```

```
         }
      }

      //Método para calcular la potencia de disparo
      void calculoPotencia() {
      //Dependiendo la distancia al enemigo, diparo con mas o menos potencia.
      //La potencia es inversamente proporcional a la distancia.
         potencia = 500/target.distance;
      }

      //Cuando muere un robot, hago que mi distancia al enemigo sea muy
grande.
      public void onRobotDeath(RobotDeathEvent e) {

         if (e.getName() == target.name)
            target.distance = 9000000;
      }

   //Si nos golpeamos con un robot:
   //-Si es compañero nos separamos
   //-Si es enemigo, le atacamos
      public void onHitRobot(HitRobotEvent event) {
         if (isTeammate(event.getName())){
            setAhead(direccion*-1*700);
         }
         else{
            if (event.getName() != target.name)
               target.distance = 9000000;

         }
      }

   //Gira el cañón en señal de victoria
      public void onWin(WinEvent event) {

         while(true){
            execute();
            setTurnGunLeftRadians(pi/2);
         }
      }
   }

//Clases que contiene las características del enemigo
   class Enemy {

      //Nombre del enemigo
      String name;
      //Almacena la orientacion
      public double bearing;
      //Almacena el apuntamiento
      public double head;
      //Tiempo en que ha sido detectado
      public long ctime;
      //Velocidad del enemigo
      public double speed;
      //Posicion del enemigo
      public double x,y;
      //Distancia del enemigo
      public double distance;
```

```
//Métodos para obtener y modificar las propiedades
   public String getname(){
      return name;
   }

   public double getbearing(){
      return bearing;
   }

   public double gethead(){
      return head;
   }

   public long getctime(){
      return ctime;
   }

   public double getspeed(){
      return speed;
   }

   public double getx(){
      return x;
   }

   public double gety(){
      return y;
   }

   public double getdistance(){
      return distance;
   }

//Metodos para intentar adivinar la posicion futura del tanque.
   public double guessX(long when)
   {
      long diff = when - ctime;
      return x+Math.sin(head)*speed*diff;
   }
   public double guessY(long when)
   {
      long diff = when - ctime;
      return y+Math.cos(head)*speed*diff;
   }

}
```

7. Conclusiones

Vemos que la aplicación de conceptos relacionados con la asignatura tales como el aprendizaje son complejos y requieren sistemas de recogida de información pertinente. Sin embargo, podemos hacer una inteligencia artificial basada en heurísticos más a nuestro alcance.
Respecto a los algoritmos genéticos, que pueden parecer una solución “fácil” porque evoluciona “sola”, encierra otro tipo de complicaciones como la elección del modelo, la elección de una semántica para los genomas y la flexibilidad del espacio de búsqueda, entre otros.

8. Referencias / fuentes de información

- http://counties.csc.calpoly.edu/team13fk/F02/part1.html
- http://robowiki.net
- http://www.robocoderepository.com.
- http://counties.csc.calpoly.edu/~team14fk/F04/AIM-2003-023.pdf, Jacob Eisenstein work about genetic programming for Robocode.

- http://robocode.alphaworks.ibm.com/home/home.html, AlphaWorks (web oficial de Robocode).
- http://www-106.ibm.com/developerworks/java/library/j-robocode/index.html, _Rock 'em, sock 'em Robocode! (tutorial).
- http://www.ug.cs.usyd.edu.au/~csled/biwf/robocode/, The Robocode Tutorials (tutorial y ejemplos)

- John R. Koza. Evolving emergent wall following robotic behaviour using the genetic programming paradigm. In ECAL'91, Paris, 1991.
- John R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, 1992.