Víctor López Lanuza Adrián García Campillo

Ejercicio 1

En el primer ejercicio lo primero que hicimos fue intentar entender cómo funcionaban todas las clases y métodos disponibles por parte de la biblioteca de robocode. Esto lo hicimos debido a que en clase vimos muy complicado empezar a programar desde 0 los 5 robots del ejercicio 1 sin apenas haber mirado nada de la documentación de Robocode. Así que lo primero que hicimos fue entrar a la Wikipedia oficial de Robocode, https://robowiki.net/wiki/. Una vez ahí nos pusimos a mirar todas y cada una de las clases y métodos disponibles para nuestro **PredatorRobot** con la extensión de **TeamRobot**.

Una vez vista la documentación nos pusimos a mirar lo que se nos pedía en el primer ejercicio de la práctica. Una vez ya leídas las preguntas decidimos empezar a programar nuestros **PredatorRobots.**

Lo primero que hicimos, tal y como se nos pide, es dividir el trabajo de los robots en 4 fases:

- Fase 0 → Handshake
- Fase 1 → Aproximación
- Fase 2 → Órbita
- Fase 4 → Ramming

Una vez hecha la separación de fases en el método de **run(),** lo que hicimos fue ponernos a mirar cómo podíamos la posición de cada uno de los robots y compartirla con los demás compañeros del equipo. Aquí nos encontramos con el problema de cómo podíamos enviar la posición de cada compañero a los aliados y que ellos supieran dónde se encontraban cada uno de los compañeros. Si que es verdad que teníamos un método llamado Broadcast, que nos permite mandar algún mensaje a los compañeros pero con ello no conseguimos que los mensajes nos llegaran tal y como nosotros queríamos. Aquí estuvimos un rato pensando en cómo podíamos resolver esta situación. Hasta que se nos iluminó una bombilla y pensamos en crear una nueva clase, llamada Mensaje.java, con la cual poder enviar los mensajes. Para enviar un mensaje a cualquier aliado lo que hacemos es crear un nuevo objeto de la clase mensaje en el momento del **broadcast.**
De manera inicial en la clase de Mensaje solo teníamos dos variables privadas que eran X e Y. Estas variables privadas sirven para almacenar las coordenadas del robot que creaba el mensaje.
Una vez llegado el mensaje teníamos que pensar en una manera de poder almacenar estas posiciones, y lo primero que pensamos fue en un Array, pero en este caso no nos serviría. Así que por lo que acabamos optando fue por un HashMap, a este lo hemos llamado `teammatePositions,` este HashMap es <String, Point>. En este HashMap tenemos que la clave es el nombre del aliado y el Point es un valor de coordenadas en los ejes OX y OY.
Una vez ya teníamos comunicados a los diferentes aliados, lo siguiente era conseguir la distancia de los enemigos y calcular el enemigo más cercano de todos. Aquí lo que había que hacer era calcular la distancia de cada aliado con todos los enemigos que encontramos en la partida y enviar a los compañeros cual es el más cercano de todos ellos. Una vez habíamos pasado ese mensaje, debíamos calcular cual de todos ellos era el más alejado a alguno de nuestros compañeros. Para hacer todo esto lo que debíamos hacer era empezar modificando el método de **onScannedRobot()**.

Este método se activa cuando el radar de nuestro Robot detecta a cualquier robot que haya en el mapa de juego. Tal y como lo programamos, el método lo primero que hace es comprobar si el Robot detectado es aliado o no, ya que si lo es, pasamos de él. En cambio, si no lo es lo que hacemos es encontrar el **Bearing Absoluto** del enemigo ya que este lo usaremos para calcular las coordenadas en las que se encuentra el enemigo. Y una vez obtenidas las coordenadas lo que hacemos es almacenarlas en un HashMap `enemiesPositions`.
Una vez encontrados los diferentes enemigos y sus posiciones entramos a una función llamada **calcularEnemigoMasCercano(),** encargada de decirnos para el robot que la ejecuta, cuál es el Robot enemigo que más cerca se encuentra de él.
Una vez ya sabemos cual es el Robot más cercano lo pasamos creando un nuevo objeto de la clase Mensaje. Esta vez añadimos dos nuevas variables privadas dentro del mensaje, la variable código (ayuda al código a identificar que estamos pasando en el mensaje) y la variable enemigo (indica el nombre del enemigo).
Cuando implementamos estas funcionalidades en nuestro código vimos que teníamos un problema, y esté era que todos calculaban todo y el código se quedaba pillado y no pasamos de fase, o que no todos los robots iban a por el mismo enemigo, o que no iban a por el robot correcto.
Para solucionar esto lo que se nos ocurrió fue implementar dentro del método **onMessageReceived(),** un if en el que de manera inicial solo entrará uno de los robots, de tal manera que este contara que le llegará como mínimo un mensaje, relacionado con el enemigo más cercano, de cada uno de sus compañeros para poder decidir a quién iban a atacar y por tanto cambiar de fase. Una vez se tenía calculado el enemigo al que se iba a atacar, el aliado encargado de decidirlo lo comunicaba a los demás aliados para que todos atacaran al mismo, y con la recepción del mensaje cambiar la fase.

En esta fase de Aproximación lo que hacemos es asegurarnos de que tanto el radar como el arma giren con el cuerpo ya que vamos a movernos en dirección del objetivo y así nos aseguramos que le vamos a disparar a él.
A partir de aquí accedemos al HashMap donde guardamos las posiciones de los enemigos, `enemiesPositions`, y a partir del nombre del enemigo que hemos pasado por el mensaje enviado antes de cambiar de fase 0 a 1, obtenemos las coordenadas que usaremos para calcular la distancia que tenemos al enemigo y ver si pasamos de fase, en el caso de estar cerca, o seguimos avanzando.
Aquí nos dimos cuenta de que además también necesitábamos el **Bearing** del enemigo y para obtenerlo vimos que la solución era crear otro HashMap y guardarlo en el método de **onScannedRobot().** Con este bearing enemigo vamos a rotar hacia él y a avanzar en su dirección mientras también le disparamos.

En la fase de Órbita lo que hacemos es activar que el arma se pueda girar de manera independiente al radar y al cuerpo del robot. Una vez hecho, lo que hacemos es girar el cuerpo del robot en dirección perpendicular a la del robot enemigo para poder girar a su alrededor. Mientras damos vueltas a su alrededor también giramos el arma teniendo en cuenta nuestro ángulo de **heading** (nuestra dirección del robot), el ángulo de nuestro arma (**Gun Heading**) y el **Bearing** del enemigo. Una vez girada el arma solo disparamos si el ya está fría y el ángulo de giro restante es muy cercano al objetivo. Además de todo esto tendremos que mirar cual es la vida que le queda al enemigo targeteado, ya que si le queda poca deberemos cambiar a la fase 3. Para comprobar la vida que le queda al enemigo lo

que hay que hacer es, como no hay ningún atributo que nos de su vida, fijarnos en la energía del enemigo. Esta de manera habitual será de un máximo de 100, así que en el método de onScannedRobot añadiremos un nuevo HashMap encargado de guardar las energías de los diferentes enemigos de la partida. Así que lo único que tendremos que hacer para comprobar la vida de estos enemigos será acceder al HashMap llamado **enemiesEnergies** y a partir de la variable **enemigoTargeteado,** encargada de guardar el nombre del enemigo a por el que van todos los aliados, buscar el enemigo y su energía. Si la energia que le queda al enemigo es menos que 40, entonces cambiamos a la fase 3.

En la fase de Ramming lo que hacemos es volver a girar el Robot hacia la dirección del enemigo e ir directo hasta él para chocarnos con él y quitarle toda la vida que le queda. Una vez muerto lo que hacemos es cambiar de objetivo y para ello pasamos a la fase 0 de nuevo. Para saber que el enemigo que estamos chocando es el que ha muerto, vamos al método de **onRobotDeath** y poner que compruebe si el Robot que ha muerto es enemigo o aliado. En el caso de que sea enemigo cambiamos a la fase 0.

Variables

En cuanto a las variables usadas en nuestro código tenemos las siguientes:

- 4 HashMaps:
  - private Map<String, Point> **teammatePositions** = new HashMap<String, Point>() → Guarda las posiciones de los aliados
  - private Map<String, Point> **enemiesPositions** = new HashMap<String, Point>() → Guarda las posiciones de los enemigos
  - private Map<String, Double> **enemiesBearings** = new HashMap<String, Double>() → Guarda los Bearings enemigos
  - private Map<String, Double> **enemiesEnergies** = new HashMap<String, Double>() → Guarda la Energia de los enemigos

- 4 booleanos:
  - boolean **aliado2** → Indica si el aliado2 ha pasado el enemigo más cercano.
  - boolean **aliado3** → Indica si el aliado3 ha pasado el enemigo más cercano.
  - boolean **aliado4** → Indica si el aliado4 ha pasado el enemigo más cercano.
  - boolean **aliado5** → Indica si el aliado5 ha pasado el enemigo más cercano.
  - boolean **aliado1muerto** → Indica si el aliado1 ha muerto.
  - boolean **aliado2muerto** → Indica si el aliado2 ha muerto.
  - boolean **aliado3muerto** → Indica si el aliado3 ha muerto.
  - boolean **aliado4muerto** → Indica si el aliado4 ha muerto.

- 3 integers:
  - int **fase** → Indica en que fase nos encontramos.
  - int **count** → Indica cuántos aliados han pasado el enemigo más cercano.
  - int **lado** → Indica en qué dirección nos movemos en la fase de órbita.

- Dentro de la estructura mensaje tenemos:
  - private int **codigo** → Variable encargada de decirnos que se manda en el mensaje.

    - Cod = 0 → Información de nuestra posición (Posición de un aliado)
    - Cod = 1 → Información del enemigo más cercano a nuestra posición
- private double **x** → Variable encargada de guardar la coordenada X del enemigo.
- private double **y** → Variable encargada de guardar la coordenada Y del enemigo.
- private String **Name** → Variable encargada de guardar el nombre del enemigo.

Métodos

- **handshakePhase()** → Este método tiene la función de hacer todo lo que se pide en la fase 0 del ejercicio.
- **aproximacionPhase()** → En este segundo método nos encontramos que su función es la de hacer todo lo que se pide en la fase 1 del ejercicio.
- **orbitaPhase()** → Para este tercer método lo que se hace es que el robot gire orbitando al enemigo mientras le va disparando al mismo tiempo y cada 2 segundos cambiando su posición.
- **broadcastTeammatePosition()** → En este método cogemos nuestra posición la guardamos en el HashMaps teammatePositions y se la enviamos al resto de aliados.
- **calcularEnemigoMasCercano()** → Método utilizado para calcular de todos los enemigos que hemos detectado a partir de un escaneo ver cual de ellos es el más cercano.
- **onMessageReceived(MessageEvent e)** → Método encargado de manejar los mensajes recibidos de otros aliados. Dependiendo del código mandado en el mensaje se tratará de una manera u otra.
- **onScannedRobot(ScannedRobotEvent e)** → Cuando un robot detecta que hay un robot escaneado entra en esta función y dependiendo de si es un aliado o enemigo hará cosas distintas.
- **onRobotDeath(RobotDeathEvent e)** → Este método es el que se ejecuta cuando un Robot muere y dependiendo de si es un robot aliado o enemigo el tratamiento del evento es diferente.
- **calcularDistancia(double x1, double y1, double x2, double y2)** →
- **comprobarVida()** → Método encargado de mirar cuanta vida le queda al enemigoTargeteado.
- **calcularAnguloHacia Enemigo(double x, double y)** → En esté método lo que hacemos es calcular el ángulo que tenemos hacia el enemigo dadas sus coordenadas.
- **normalizeBearing(double angle)** → Método encargado de normalizar el ángulo de Bearing