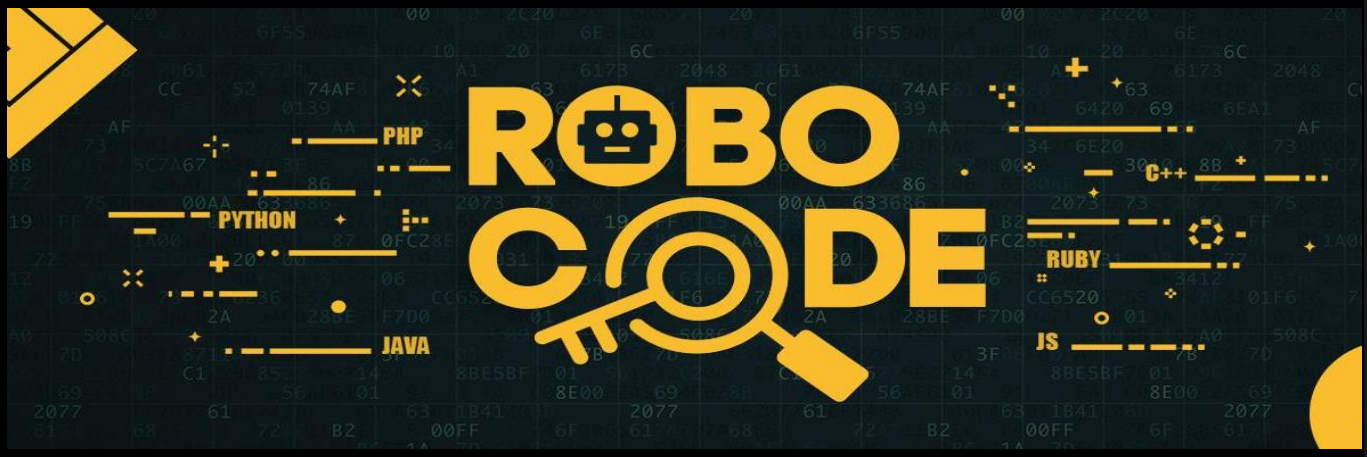
Departamento de Ingeniería de sistemas y Automática

Universidad Carlos III de Madrid



CL4P_TP

Robot orbitador de Robocode

Javier Galán Méndez
UNIVERSIDAD CALOS III DE MADRID

Informática Industrial II

ÍNDICE

1.Introducción

El funcionamiento del robot CL4P_TP es de tipo orbitador, a lo largo de la memoria se mostrarán las diferentes funciones relacionadas con el movimiento y disparo del robot y una breve explicación de cada una. En el código se encontrarán muchos métodos set (Ej: setAhead()) y execute(), esto es debido a que estos métodos permiten ejecutarse en paralelo con los métodos siguientes que se llamen, haciendo del robot muy eficiente.

2.Clase CL4P_TP y métodos

Es la función principal del robot, inicializa los colores, las funciones setAdjust son para independizar los movimiento de cada componente del robot y el while crea una barrido en el radar para buscar posibles objetivos.

El objeto condition crea una condición al robot para que en el caso de detectar una perdida de energía en el enemigo(posible disparo de bala), salte el metodo CustomRobotEvent para esquivar la bala.

```
////////////////////////////////////////////////////////////////////////
    public void run() {

        setColors(Color.yellow,Color.black,Color.red,Color.red,Color.blue);
        this.setAdjustGunForRobotTurn(true);//La partes del robot se manejan independientemente
        this.setAdjustRadarForGunTurn(true);
        this.setAdjustRadarForRobotTurn(true);

        Condition triggerHitCondition = new Condition("esquivar") {//Evento para esquivar balas
            public boolean test() {
                return (energia_actual-energia_enemigo<10);//Revisa un cambio de energía en el enemigo
            }
        };
        addCustomEvent(triggerHitCondition);

        while(true){
            turnRadarRightRadians(2);//Barrido
            scan();
        }

    }
////////////////////////////////////////////////////////////////////////
```

onScannedRobot es un evento de interrupción que rompe el ciclo de la función run para realizar la tarea del ScannedRobot cuando el radar ha detectado un enemigo.
En este evento, se llama a la función de disparo predictivo y del movimiento de la clase Orbitar
Se obtiene la energía del robot escaneado para el evento custom comentado anteriormente y según la potencia descrita en movimiento() realizará una cantidad de daño y velocidad de la bala disparada.
Tras disparar, llamas a la función scan() para volver a scanear

```
////////////////////////////////////////////////////////////////////////
    public void onScannedRobot(ScannedRobotEvent e) {

        disparo_predictivo(e);
        energia_enemigo=e.getEnergy();

        Orbitador(e);//Cambio de estrategia según los jugadores

        setFire(power);//Dispara
        scan();
    }
////////////////////////////////////////////////////////////////////////
```

onHitWall es un evento como onScannedRobot, pero se activa cuando el robot choca contra las paredes de la arena,llama al metodo hitlwall de la clase Orbitar.

```
  public void onHitWall(HitWallEvent e){//En caso de chocar contra las paredes, alejarse hacia el centro

     orbitar.hitWall(this);

  }
```

onCustomEvent() se activa cuando se cumple la condición dada en la función run, cunado este evento es activado, el sentido del movimiento cambia y la energía del enemigo se actualiza

```
    public void onCustomEvent(CustomEvent e){
        if(e.getCondition().getName().equals("esquivar")){//Evento custom para esquivar balas
            movimiento=-movimiento;//Cambio de dirección
            energia_actual=energia_enemigo;
        }
    }
```

onWinEvent() se activa cuando el robot ha ganando una batalla, ejecuta un pequeño balie moviento el cañón y el cuerpo de manera infinita hacia lados opuestos.

```
    public void onWin(WinEvent e){//Baile de la victoria
        setTurnGunRightRadians(Double.POSITIVE_INFINITY);
        setTurnLeftRadians(Double.POSITIVE_INFINITY);
        setAhead(10000);
        execute();
    }
```

3.Clase Estrategia y métodos

Estrategia es la clase padre de la clase orbitador. Inicializa su único atributo, movimiento, que permite dar el sentido del movimiento del robot, que se modifica y obtiene con sus métodos get y set.

```
public class Estrategias
{
    private int _movimiento;

    public Estrategias(int movimiento){
        this._movimiento=movimiento;
    }

    public void setMovimiento(int movimiento){//Sentido del movimiento
        this._movimiento=movimiento;
    }

    public int getMovimiento(){
        return _movimiento;
    }
```

4.Clase Orbitar y métodos

movimiento() tiene como principal función el de orbitar en torno al enemigo según la distancia al mismo, si el robot está lejos, el robot girará hacia el enemigo según la orientación que tenga y se dirige hacia el mismo en base a la distancia que quede con el enemigo.

Cundo está cerca del mismo, el cuerpo del robot quedará perpendicular al enemigo y a la misma vez avanzará y si el enemigo dispara una bala, este cambiará de sentido según la variable movimiento, declarada en onCustomEvent.

Si el robot está muy lejos, la potencia de fuego será menor, haciendo más rápida su bala; pero si está cerca aumentará su potencia de fuego, haciendo más daño

```
    public void movimiento(AdvancedRobot c,ScannedRobotEvent e){

        if(e.getDistance()>170){//Si está lejos, el robot se acercará al enemigo
            c.setTurnRightRadians(Utils.normalRelativeAngle(c.getRadarHeadingRadians()-c.getHeadingRadians()));
            //El robot debe mantenerse directo con el enemigo
            c.setAhead(e.getDistance()-170);//Controla distancia con el enemigo

        }
        else if(e.getDistance()<=170){//Cuenado está cerca del enemigo, el robot girará en torno al enemigo
            //Perpendicular al enemigo
            c.setTurnRightRadians(Utils.normalRelativeAngle(e.getBearingRadians()-Math.toRadians(90)));

            c.setAhead((e.getDistance()-170)*getMovimiento());//Controla distancia con el enemigo
            c.setMaxVelocity(8);
        }

        c.execute();
        if(e.getDistance()>100) c.setFire(2);
        else c.setFire(3);//Mayor potencia al estar más cerca del enemigo
    }
```

disparo_pedictivo() se encarga de mover el cañón y el radar, según varios parámetros trigonométricos del robot con el enemigo, el radar se moverá y, según la velocidad y ángulo del enemigo, el radar girará más o menor grados a los establecidos, previniendo el movimiento del enemigo. Al radar se plantea unas condiciones según el ángulo para girar en menos tiempo y así poder mejorar la eficiencia del disparo. Según a que distancia se encuentre con respecto al enemigo, la velocidad de predicción será mayor o menor, ya que el enemigo si se encuentra lejos, el arco del movimiento será menor que si se encuentra cerca.

Luego el cañón es girado tomando como referencia la posición del radar.

```
    public double centro_mapa(AdvancedRobot c){
        double ladoX=(c.getBattleFieldWidth()/2)-c.getX();//Distancia X del robot al centro
        double ladoY=(c.getBattleFieldHeight()/2)-c.getY();//Distancia Y del robot al centro
        double alpha = Math.toDegrees(Math.atan(ladoY/ladoX));//Angulo del triangulo equivalente

        //Según el sector del mapa, tendrá que girar un ángulo u otro
        if (ladoY < 0 && ladoX < 0)      alpha = -90 -c.getHeading()- alpha;
        else if (ladoY < 0 && ladoX > 0) alpha = -270 -c.getHeading()- alpha;
        else if (ladoY > 0 && ladoX < 0) alpha = -90 -c.getHeading()- alpha;
        else                             alpha = 90 -c.getHeading()- alpha;

        if(alpha<=180) alpha=alpha;//Busca el ángulo y sentido de giro más óptimo
        else alpha-=360;
        if(alpha>=-180) alpha=alpha;
        else alpha +=360;

        return alpha;
    }
```

centro_mapa() tiene como función devolver el valor de cuento se tiene que mover el robot con respecto a la orientación y posición con respecto al mapa y el centro de este. Las condiciones son para mejorar la eficiencia den el giro y evitar giros de más de 180 grados buscando el camino más corto para girar

```
    public void disparo_predictivo(AdvancedRobot c,ScannedRobotEvent e){
        double cantidad_velocidad;//Cantidad de velocidad segun la distancia
        double absBearing=e.getBearingRadians()+c.getHeadingRadians();//bearing absoulto del enemigo
        double velocidad=e.getVelocity() * Math.sin(e.getHeadingRadians() -absBearing);//Velocidad del enemigo

        double angulo_radar =c.getHeadingRadians() + e.getBearingRadians()-c.getRadarHeadingRadians();

        if(angulo_radar<=180) angulo_radar=angulo_radar;
        else angulo_radar-=360;//Busca el ángulo y sentido de giro más óptimo
        if(angulo_radar>=-180) angulo_radar=angulo_radar;
        else angulo_radar +=360;

        if(e.getDistance()>=200) cantidad_velocidad=velocidad/22;//Numeros obtenidos experimentalmente
        else cantidad_velocidad=velocidad/13;

        c.setTurnRadarRightRadians(Utils.normalRelativeAngle(angulo_radar));//Radar sigue al enemigo
        //Torreta a posición futura
        c.setTurnGunRightRadians(c.getRadarHeadingRadians()-c.getGunHeadingRadians()+cantidad_velocidad);
    }
```

en hitwall() el robot gira su cuerpo en torno al angulo definido con la función centro_mapa() para alejarse de las paredes dirigiendose al centro del mapa, pero solo avanza 100 metros .

```
    public void hitWall(AdvancedRobot c){
        double angulo_giro = centro_mapa(c);
            c.setMaxTurnRate(Rules.MAX_TURN_RATE);//Máxima velocidad de giro
            c.setTurnRight(angulo_giro);
            c.setMaxVelocity(8);//Se incrementa la avelocidad
            c.setAhead(100);
            c.execute();
    }
```