PREDATOR TEAM

L'enunciat ens demana fer un equip de 5 robots iguals que tinguin el següent comportament:

- Escanejar els robots enemics i determinar el més proper a l'equip.
- Dirigir-se a l'objectiu disparant.
- Un cop arribada a una determinada distància, orbitar al seu voltant, canviant de sentit cada 2 segons.
- Llançar-se contra l'objectiu quan li queda poca vida.

Aquests són els atributs que hem necessitat per implementar el robot Predator:

```
private int fase;
private int enemics;
private int teamMates;
private int direccioOrbita;
private double enemicX;
private double enemicY;
private double distanciaEnemic;
private double enemicBearing;
private double ultimBearing;
private double vidaEnemic;
private double tempsOrbita;
private String objectiu;
private List<String> robotsEnemics = new ArrayList<>();
private List<String> robotsEnemicsLocal = new ArrayList<>();
private List<Double> distanciesEnemics = new ArrayList<>();
private List<String> robotsMorts = new ArrayList<>();
```

- **fase** és un integer que determina la fase en la que ens trobem, i pren el número de la fase, és a dir 0 per a fase 0, 1 per a fase 1…
- **enemics** és el número d'enemics vius al camp de batalla, ens serveix per determinar si ja hem escanejat tots els robots a la fase 0.
- **teamMates** és el número de robots del nostre equip que estan vius.
- **direccioOrbita** s'encarrega de determinar cap a quin sentit orbitem.
- **enemicX**, **enemicY**, **distanciaEnemic**, **enemicBearing** i **vidaEnemic** són atributs que guarden informació sobre el enemic objectiu, recollits pel radar.
- **tempsOrbita** és l'ultim temps en el que hem fet canvi de sentit en la fase d'òrbita, i l'usem per determinar quan han passat 2 segons des de l'última vegada.
- **objectiu** és el nom del robot objectiu, l'utilitzem per a que el radar sàpiga de quin robot ha de recollir les dades.
- **robotsEnemics** i **robotsEnemicsLocal** són llistes de noms dels robots. En el primer cas, es tracta dels robots ja escanejats per tot l'equip durant la fase 0, mentre que el segon són els robots escanejats per un robot en concret.
- **distanciesEnemics** són les distàncies de cada robot enemic respecte l'equip. Sabem a quin robot corresponen perquè es troben en el mateix ordre que **robotsEnemics,** ja que s'afegeixen al mateix temps.
- Per últim, comptem quins enemics han mort en **robotsMorts**, ens serveix a l'última fase per detectar quan ha mort el nostre objectiu.

Estratègia:

```
@Override
public void run(){
    setAdjustRadarForRobotTurn(independent: false);
    fase = 0;
    enemics = getOthers() - getTeammates().length;
    teamMates = getTeammates().length + 1;
    enemicX = enemicY = -1;
    enemicBearing = -1;
    direccioOrbita = 1;
    tempsOrbita = 0;
    while(true){
        gestionarMoviment();
        actualitzarFase();
    }

}
```

Per a que el run() no quedi massa gran i sigui més difícil d'entendre, l'únic que hi ha és la inicialització d'alguns atributs i les funciones gestionarMoviment() i actualitzarFase() dins del while(true) que moven el robot segons la fase en la que estem i comproven les condicions per veure si s'ha de canviar de fase.

```
default:
    if (e.getName().equals(anObject:this.objectiu)){
        double angle = this.getHeading() + e.getBearing();
        distanciaEnemic = e.getDistance();
        enemicX = this.getX() + e.getDistance() * Math.sin(a: Math.toRadians(angdeg: angle));
        enemicY = this.getY() + e.getDistance() * Math.cos(a: Math.toRadians(angdeg: angle));
        enemicBearing = e.getBearing();
        vidaEnemic = e.getLife();
    }
```

En el cas del onScannedRobot, tenim un switch amb dos casos, el primer és per a la fase 0 i l'altre el default (per a tota la resta de fases) que és el que es veu a la imatge. Bàsicament actualitzem, per al robot objectiu, les seves propietats en els atributs.

El gestionarMoviment() té el següent aspecte. En fase 0 només girem el radar per buscar enemics, en les altres hi ha funcions amb el comportament corresponent.

```
private void gestionarMoviment(){
    switch (fase){
        case 0:
            turnRadarRight(degrees: 360);
            break;
        case 1:
            turnRadarRight(degrees: 360);
            if (enemicX != -1 && enemicY != -1) dirigirseARobot();
            break;
        case 2:
            setTurnRadarRight(degrees: 360);
            orbitarRobot();
            gestionarCanviDireccio();
            break;
        case 3:
            turnRadarRight(degrees: 360);
            dirigirseARobot();
            break;
    }
}
```

Estratègia per fer el Handsahe

Com hem mencionat abans, la funció onScannedRobot és diferent per a la fase 0, primer comprovem que el robot escanejat no és del nostre equip, un cop tinguem un enemic escanejat, mirem si ja l'ha escanejat el nostre robot (si es troba a la llista robosEnemicsLocal). En cas que sigui el primer cop que el trobem, l'afegim i mirem si un altre membre de l'equip ja l'ha trobat abans. En cas afirmatiu, només canviem la distància i la comuniquem si és major a la distància que havia abans, en cas negatiu, afegim la distància i la comuniquem.

```
@Override
public void onScannedRobot (ScannedRobotEvent e){
    switch (fase){
        case 0:
            if (!isTeammate(name:e.getName())){
                boolean enviarMissatge = false;
                if (!robotsEnemicsLocal.contains(o: e.getName())){
                    robotsEnemicsLocal.add(e: e.getName());
                    if (!robotsEnemics.contains(o: e.getName())){
                        robotsEnemics.add(e: e.getName());
                        distanciesEnemics.add(e: e.getDistance());
                        enviarMissatge = true;
                    } else {
                        int i = this.robotsEnemics.indexOf(o: e.getName());
                        if (e.getDistance() > distanciesEnemics.get(i)) {
                            distanciesEnemics.set(i, e: e.getDistance());
                            enviarMissatge = true;
                        }
                        }
                    }
                    if (enviarMissatge){
                        String msg = "EnemicDetectat," + e.getName() + "," + e.getDistance();
                        try {
                            this.broadcastMessage(message: msg);
                        } catch (IOException ex) {
                        }
                    }

                }

            break;
```

El missatge d'enemic detectat té la següent sintàxis: EnemicDetectat, nomEnemic, distànciaEnemic. Quan un robot rebi el missatge, farà un split separant per comes, el primer element li indicarà el tipus de missatge, i amb els dos següents podrà afegir el nom i distàncies de robot a les llistes.

```
@Override
public void onMessageReceived(MessageEvent e){
    String missatge = (String)e.getMessage();
    String[] parts = missatge.split(regex: ",");
    String tipus = parts[0];
    if (tipus.equals(anObject:"EnemicDetectat")){
        String nom = parts[1];
        double distancia = Double.parseDouble(parts[2]);
        if (!robotsEnemics.contains(o: nom)){
            robotsEnemics.add(e: nom);
            distanciesEnemics.add(e: distancia);
        }
        else if (distanciesEnemics.get(i: robotsEnemics.indexOf(o: nom)) < distancia){
            distanciesEnemics.set(i: robotsEnemics.indexOf(o: nom), e: distancia);
        }
    }
```

D'aquesta manera les distàncies de distanciesEnemics seran sempre la distància de l'enemic respecte el membre més llunyà del nostre equip.
La fase s'acaba un cop hagi passat un segon, i seguidament guardem en l'atribut objectiu el nom del enemic més proper, amb la funció obtenirObjectiu.

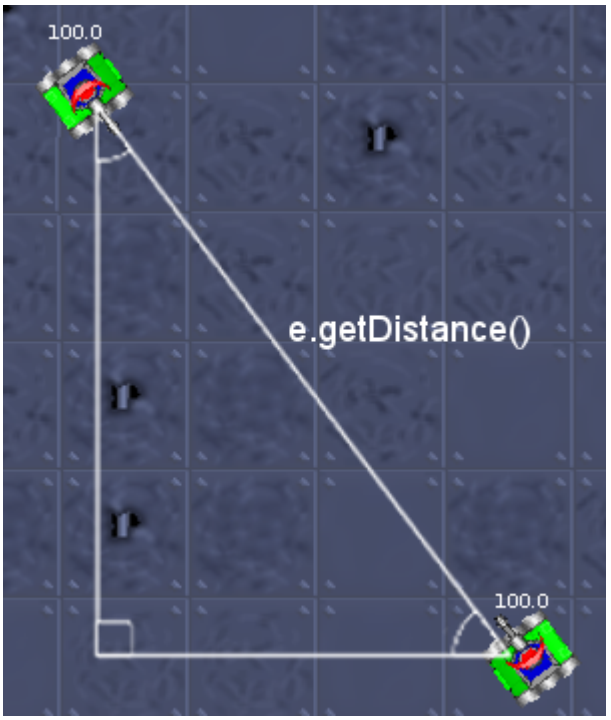
```
private void actualitzarFase(){
    switch (fase){
        case 0:
            if (this.getTime() >= 60){
                fase = 1;
                objectiu = obtenirObjectiu();
            }
            break;
```

```
private String obtenirObjectiu() {
    double min = this.distanciesEnemics.get(i: 0);
    for (int i = 1; i < this.distanciesEnemics.size(); ++i){
        if (min > this.distanciesEnemics.get(i)) min = this.distanciesEnemics.get(i);
    }
    return this.robotsEnemics.get(i: this.distanciesEnemics.indexOf(o: min));
}
```

Estratègia per a Aproximació:

Si enemicX i enemicY són -1, vol dir que el radar encara no ha obtingut les dades de l'objectiu, i s'ha d'esperar a que el radar ho fagi per entrar en la funció dirigirseARobot().

<u>Càlcul de X i Y del robot enemic:</u>

- Sabem la X i la Y del nostre robot, i sabem la distància que separa l'enemic del nostre robot. Amb aquestes dades podem formar el següent triangle en el que els robots són dos dels vèrtex.
- 

- El costat inferior equival a la diferència entre la nostra X i la X enemiga, i el costat amb el que forma un angle de 90º la diferència entre la nostra Y i la Y enemiga.
- Necessitem calcular la mida dels costats del triangle, que sumats a les nostres X i Y, ens donaran la posició de l'enemic. Per fer-ho necessitem l'angle que es forma, que s'obté amb la suma del heading del nostre robot i el bearing enemic. En la imatge del pdf de l'enunciat és veu clarament com la suma d'aquests dos angles formen l'angle que busquem.
- Un cop calculat l'angle, calculem la mida dels costats amb trigonometria, fem servir cosinus pel costat contigu (Y) i sinus pel costat oposat (X). Sumem aquest valor a les nostres X i Y i ja tenim la posició de l'enemic.

Dirigir-se a l'enemic:

```
private void dirigirseARobot(){
    double costat_x = enemicX-this.getX();
    double costat_y = enemicY-this.getY();
    double angle = Math.toDegrees(angrad: Math.atan2(y: costat_x, x: costat_y));
    double gir = angle - this.getHeading();
    if (gir > 180) gir = gir - 360;
    if (gir < -180) gir = 360 + gir;
    if (fase == 1) setFire(power: 1);
    setTurnRight(degrees: gir);
    if (fase == 1) setFire(power: 2);
    setAhead(distance: recorrerDistancia());
    if (fase == 1) setFire(power: 3);
    execute();
}
```

La clau està en saber com girar-se. Bàsicament el nostre robot ha de girar-se el mateix angle que hem calculat en l'escaneig, així que ho recalculem a partir de la x i la y utiltizant arctangent. Posteriorment, determinem si és més ràpid girar-se per la dreta o per l'esquerra (mirant si supera els 180 graus o és menys de -180, que voldria dir girar més de 180 cap a l'esquerra). A la vegada anem disparant amb cada cop més potència, ja que cada cop som més a prop i és més probable impactar a l'enemic. Mesurem quant ens apropem amb recorrerDistancia():

```
private double recorrerDistancia(){
    double distanciaMax = this.getBattleFieldWidth()*0.3;
    double recorregut = 0;
    if (fase == 1){
        if (distanciaEnemic > distanciaMax){
            if ((distanciaEnemic - 100) <= distanciaMax) recorregut = 100;
            else recorregut = distanciaEnemic - distanciaMax;
        }
    } else recorregut = distanciaEnemic;
    return recorregut;
}
```

Aquesta funció ens assegura que no ens aproparem massa, i estarem a una distància d'un 30% de l'amplada del camp per orbitar.

Final de fase:

```
case 1:
    if (distanciaEnemic <= this.getBattleFieldWidth()*0.3){
        setAdjustGunForRobotTurn(independent: false);
        turnGunRight(degrees: 90);
        turnLeft(degrees: 90);
        if (tempsOrbita == 0){
            double time = this.getTime();
            tempsOrbita = time;
            String msg = "RobotOrbita," + time;
            try {
                this.broadcastMessage(message: msg);
            } catch (IOException ex) {
            }
        }
        fase = 2;
    }
    break;
```

Quan la distància és igual o menor al 30% de l'amplada del camp, llavors acabem la fase d'aproximació. Col·loquem la pistola i rotem per apuntar cap a l'interior (el robot enemic) i posicionar-nos bé per girar. El primer robot en arribar, (el que compleix que tempsOrbita = 0) agafarà el valor del temps actual i el compartirà amb la resta de l'equip. Això ens servirà més endavant per sincronitzar els canvis de sentit. Com veiem, tots els robots tindran el mateix valor de tempsOrbita.

```
@Override
public void onMessageReceived(MessageEvent e){
    String missatge = (String)e.getMessage();
    String[] parts = missatge.split(regex: ",");
    String tipus = parts[0];
    if (tipus.equals(anObject:"EnemicDetectat")){
        String nom = parts[1];
        double distancia = Double.parseDouble(parts[2]);
        if (!robotsEnemics.contains(o: nom)){
            robotsEnemics.add(e: nom);
            distanciesEnemics.add(e: distancia);
        }
        else if (distanciesEnemics.get(i: robotsEnemics.indexOf(o: nom)) < distancia){
            distanciesEnemics.set(i: robotsEnemics.indexOf(o: nom), e: distancia);
        }
    }
    else if (tipus.equals(anObject:"RobotOrbita")){
        tempsOrbita = Double.parseDouble(parts[1]);
    }
```

Estratègia per a òrbita

El moviment d'òrbita està implementat de la següent manera:

```
private void orbitarRobot(){
    double angle = 0;
    angle = enemicBearing - 90;
    if (angle > 180) angle = angle - 360;
    else if (angle < -180) angle = 360 - angle;
    setTurnRight(degrees: angle);
    setFire(power: 1);
    setAhead(25 * direccioOrbita);
    execute();
    fire(power: 4);
}
```

Hem de mantenir un bearing enemic de 90 graus, això es calcula fàcilment restant 90 al bearing de l'enemic, i un altre cop busquem el gir més ràpid possible. L'atribut direccioOrbita val 1 (sentit horari) o -1 (sentit antihorari). Disparem amb poca potència mentre girem i disparem més fort un cop fet el gir, doncs és el moment que més probablement impactarem a l'enemic.

- Canvis de sentit: Com s'ha mencionat anteriorment, tots els robots tenen el mateix valor per a tempsOrbita, per tant tots tindran el mateix temps i detectaran que han passar 2 minuts al mateix temps, a més, es comunicarà el canvi a tots els robots juntament amb el temps, per si hi ha un robot encara en fase d'aproximació, es sincronitzi quan es reincorpori:

```
private void gestionarCanviDireccio(){
    double time = this.getTime();
    if ((time - tempsOrbita) >= 120){
        tempsOrbita = time;
        direccioOrbita *= -1;
        System.out.println("Cambio en " + time);
        String msg = "CanviDeSentit," + time;
        try {
            this.broadcastMessage(message: msg);
        } catch (IOException ex) {
        }
    }
}
```

```
else if (tipus.equals(anObject:"CanviDeSentit")){
    direccioOrbita *= -1;
    tempsOrbita = Double.parseDouble(parts[1]);
}
```

També tenim en compte les parets, si un robot xoca, tots canvien el sentit de gir, per evitar rebre més dany. Des d'aquest moment es torna a comptar i no es canvia fins passat 2 segons.

```
@Override
public void onHitWall(HitWallEvent e){
    if (fase == 2){
        double temps = this.getTime();
        String msg = "CanviDeSentit," + temps;
        try {
            this.broadcastMessage(message: msg);
        } catch (IOException ex) {
        }
        direccioOrbita *= -1;
        tempsOrbita = temps;
    }
}
```

S'avança de fase quan a l'enemic li queda 50 de vida o menys. Hem col·locat aquest valor que pot semblar alt però quan detecta que l'enemic té 50 de vida, encara hi ha bales que falten per impactar, en la fase d'òrbita les bales de potència 4 impacten quasi sempre i baixa la vida ràpidament. Col·loquem la pistola recte i girem per llançar-nos sobre el robot enemic.

```
case 2:
    if (vidaEnemic <= 50){
        turnGunLeft(degrees: 90);
        turnRight(degrees: 90);
        fase = 3;
    }
    break;
```

Estratègia ramming

Bàsicament el mateix que la fase 1, però aquest cop no dispararà i la distància que retorni la funció recorrerDistancia serà la distància a la que es troba l'enemic. Un cop detectem que hagi mort, tornem a la fase 0, reinicialitzant els atributs necessaris, i movent els robots cap enrere per a que no estiguin tots apilotonats.

```
case 3:
    if (robotsMorts.contains(o: objectiu)){
        fase = 0;
        this.distanciesEnemics.clear();
        this.robotsEnemics.clear();
        this.robotsEnemicsLocal.clear();
        enemicX = -1;
        enemicY = -1;
        enemicBearing = -1;
        tempsOrbita = 0;
        back(distance:200);
    }
    break;
```