Robocode

ICO - 2025

APRESENTAÇÃO DO ROBÔ - SHIGEO KAGEYAMA

Estudantes

EDUARDO CARDOSO

TALLES SOUZA

Ideia principal

A ideia foi criar um robô que fosse muito bom em batalhas 1v1, porém que também conseguisse lidar bem com múltiplos inimigos.
Ele se tornou um robô bastante evasivo e com técnicas de combate adaptativas.

Variáveis

```
private int direction = 1;
private Map<String, EnemyData> enemies = new HashMap<>();
private String targetName = null;
```

Constantes

```
private static final double FATOR_DE_COMPENSACAO = 0.55;
private static final double MAX_FIRE_DISTANCE = 350.0;
private static final double MINIMUM_DISTANCE = 100.0;
private static final double CLOSE_RANGE_DISTANCE = 150.0;
private static final double MOVEMENT_DISTANCE = 150.0;
private static final double STRAFE_DISTANCE = 100.0;
private static final double STRAFE_OFFSET_ANGLE = 15.0;
private static final double GUN_TARGET_TOLERANCE_DEGREES = 10.0;
private static final int ESCAPE_THREAT_THRESHOLD = 2;
private static final double AIMING_TOLERANCE_DEGREES = 20.0;
private static final int TARGET_LOST_TICKS = 5;
private static final int ENEMY_DATA_TIMEOUT_TICKS = 15;
```

Loop principal

```
public void run() {
    setBodyColor(Color.black);
    setGunColor(Color.gray);
    setRadarColor(Color.gray);
    setScanColor(Color.black);
    setBulletColor(Color.gray);

    setAdjustGunForRobotTurn(true);
    setAdjustRadarForRobotTurn(true);
```

Loop principal

```
    while (true) {
        if (targetName == null) {
            scanForClosestEnemy();
        }

        if (targetName != null) {
            EnemyData target = enemies.get(targetName);
            if (target == null || getTime() - target.lastSeen > TARGET_LOST_TICKS) {
                targetName = null;
                setTurnRadarRight(360);
            } else {
                double radarTurn = getHeadingRadians() + target.lastBearingRadians - getRadarHeadingRadians();
                setTurnRadarRightRadians(Utils.normalRelativeAngle(radarTurn) * 2);
            }
        } else {
            setTurnRadarRight(360);
        }
        execute();
    }
}
```

Combate e movimento

```
public void onScannedRobot(ScannedRobotEvent e) {
    String enemyName = e.getName();
    EnemyData data = enemies.getOrDefault(enemyName, new EnemyData());
    data.update(e, this);
    enemies.put(enemyName, data);
    scanForClosestEnemy();
```

Gerenciamento de Dados e Atualização do Inimigo

Combate e movimento

```
if (targetName == null || !targetName.equals(e.getName())) {
    return;
}

int threatCount = countIncomingAimers();

if (threatCount >= ESCAPE_THREAT_THRESHOLD) {
    executeEscapeManeuver();
```

Verificação do Alvo - avaliação de ameaça e fuga

Combate e movimento

```
} else {
    double changeInEnergy = data.previousEnergy - e.getEnergy();
    if (changeInEnergy > 0 && changeInEnergy <= Rules.MAX_BULLET_POWER) {
        direction *= -1;
    }
    data.previousEnergy = e.getEnergy();

    if (e.getDistance() < MINIMUM_DISTANCE) {
        setTurnRight(normalRelativeAngleDegrees(e.getBearing() + 90));
        setBack(MOVEMENT_DISTANCE);
    } else {
        setTurnRight(normalRelativeAngleDegrees(e.getBearing() + 90 - (STRAFE_OFFSET_ANGLE * direction)));
        setAhead(STRAFE_DISTANCE * direction);
    }
}
```

Movimentação e Evasão de Tiros

Combate e movimento

```
    if (e.getDistance() <= MAX_FIRE_DISTANCE) {
        double firePower = calculateAdaptiveFirePower(e.getDistance());
        double absBearingRad = getHeadingRadians() + e.getBearingRadians();

        double compensacaoLinear = e.getVelocity() * Math.sin(e.getHeadingRadians() - absBearingRad) / Rules.getBulletSpeed(firePower);
        compensacaoLinear *= FATOR_DE_COMPENSACAO;

        if (e.getDistance() <= CLOSE_RANGE_DISTANCE) {
            compensacaoLinear *= 0.7;
        }

        double gunTurnRad = Utils.normalRelativeAngle(absBearingRad - getGunHeadingRadians() + compensacaoLinear);
        setTurnGunRightRadians(gunTurnRad);

        if (getGunHeat() == 0 && Math.abs(getGunTurnRemaining()) < GUN_TARGET_TOLERANCE_DEGREES) {
            setFire(firePower);
        }
    }
}
```

Lógica de Mira e Tiro

Análise de ameaca

```
108 private int countIncomingAimers() {
109     int aimingEnemies = 0;
110     for (EnemyData enemy : enemies.values()) {
111         if (getTime() - enemy.lastSeen > ENEMY_DATA_TIMEOUT_TICKS) continue;
112
113         double angleToMe = Math.atan2(getX() - enemy.x, getY() - enemy.y);
114         double angleDiff = Utils.normalRelativeAngle(enemy.headingRadians - angleToMe);
115
116         if (Math.toDegrees(Math.abs(angleDiff)) < AIMING_TOLERANCE_DEGREES) {
117             aimingEnemies++;
118         }
119     }
120     return aimingEnemies;
121 }
```

Fuga

```java
private void executeEscapeManeuver() {
    double avgX = 0, avgY = 0;
    int threatCount = 0;

    for (EnemyData enemy : enemies.values()) {
        if (getTime() - enemy.lastSeen < (ENEMY_DATA_TIMEOUT_TICKS + 5)) {
            avgX += enemy.x;
            avgY += enemy.y;
            threatCount++;
        }
    }

    if (threatCount > 0) {
        avgX /= threatCount;
        avgY /= threatCount;
        double escapeAngle = Math.atan2(getX() - avgX, getY() - avgY);
        setTurnRightRadians(Utils.normalRelativeAngle(escapeAngle - getHeadingRadians()));
        setAhead(MOVEMENT_DISTANCE);
        out.println("FUGINDO! Ameaças: " + threatCount);
    }
}
```

Reacões

```java
public void onHitByBullet(HitByBulletEvent e) {
    direction *= -1;
    setAhead(MOVEMENT_DISTANCE * direction);
}

public void onHitWall(HitWallEvent e) {
    direction *= -1;
    setAhead(MOVEMENT_DISTANCE);
}

public void onHitRobot(HitRobotEvent e) {
    targetName = e.getName();
    out.println("Colisão! Novo alvo: " + targetName);

    double gunTurn = Utils.normalRelativeAngle(getHeadingRadians() + e.getBearingRadians() - getGunHeadingRadians());
    setTurnGunRightRadians(gunTurn);

    setFire(Rules.MAX_BULLET_POWER);

    setBack(STRAFE_DISTANCE);
}
```

Morte de inimigo

```
167  public void onRobotDeath(RobotDeathEvent e) {
168      enemies.remove(e.getName());
169      if (e.getName().equals(targetName)) {
170          targetName = null;
171          scanForClosestEnemy();
172      }
173  }
```

Tiro adaptativo

```
private double calculateAdaptiveFirePower(double distance) {
    double power = 4.0 - (2.5 * (Math.min(distance, MAX_FIRE_DISTANCE) / MAX_FIRE_DISTANCE));
    return Math.max(0.1, Math.min(Rules.MAX_BULLET_POWER, power));
}
```

Foco em inimigos próximos

```
private void scanForClosestEnemy() {
    double closestDistance = Double.MAX_VALUE;
    String closestEnemy = null;
    for (EnemyData e : enemies.values()) {
        if (getTime() - e.lastSeen > 10) continue;
        if (e.distance < closestDistance) {
            closestDistance = e.distance;
            closestEnemy = e.name;
        }
    }
    if (closestEnemy != null) {
        targetName = closestEnemy;
    }
}

private double normalRelativeAngleDegrees(double angle) {
    return Utils.normalRelativeAngleDegrees(angle);
}
```

Memória

```java
static class EnemyData {
    String name;
    double energy, distance, headingRadians, velocity;
    double lastBearingRadians;
    long lastSeen;
    double x, y;
    double previousEnergy = 100;

    void update(ScannedRobotEvent e, AdvancedRobot robot) {
        this.name = e.getName();
        this.energy = e.getEnergy();
        this.lastBearingRadians = e.getBearingRadians();
        this.distance = e.getDistance();
        this.headingRadians = e.getHeadingRadians();
        this.velocity = e.getVelocity();
        this.lastSeen = e.getTime();

        double absBearingRad = robot.getHeadingRadians() + e.getBearingRadians();
        this.x = robot.getX() + e.getDistance() * Math.sin(absBearingRad);
        this.y = robot.getY() + e.getDistance() * Math.cos(absBearingRad);
    }
}
```

Vitória

```java
public void onWin(WinEvent e) {
    for (int i = 0; i < 50; i++) {
        setBodyColor(Color.getHSBColor((float)Math.random(), 1.0f, 1.0f));
        setGunColor(Color.getHSBColor((float)Math.random(), 1.0f, 1.0f));
        setRadarColor(Color.getHSBColor((float)Math.random(), 1.0f, 1.0f));

        turnRight(25);
        turnLeft(25);
        execute();
    }
}
```

Fontes

Para a criação deste robô foram utilizadas as seguintes ferramentas de criação e pesquisa:

- Robo Wiki;
- GitHub;
- ChatGPT;
- Gemini AI;
- Manual Robocode.

Obrigado pela atenção!

APRESENTAÇÃO DO ROBÔ - SHIGEO KAGEYAMA