Documentacao Dino_Explosivo

Código e documentação por DinossauroBebado

▼ Sumário

Dino_Explosivo é um rôbo do jogo RoboCode, programado em Java, no qual se usa uma movimentação circular, um sistema de mira - focado para conseguir atingir rôbos que possuem sistema de movimentação circular - e um radar de lock razoalvelmente aberto.

Bom resultados em X1 e não tão bons no meele rumble.

Observação : o código foi formato para melhor visualização neste documento sendo que no arquivo .java está formatado para melhor leitura continua.

Overview

Funções

Nome	TIpo	Retorno	Objetivo
normalizeBearing(double angulo)	double	angulo normal a frente do tanque inimigo	Fazer com que a movimentação do seu tanque seja sempre a 90 graus do inimigo
canto()	boolean	se o Tanque esta perto das beiradas da arena	Uma função que monitora sua posição na arena
atirou(ScannedRobotEvent e)	boolean	se o inimigo atirou	Uma função que monitora se o inimigo atirou ou não
radar(ScannedRobotEvent e)	void	null	Implementação da estratégia do radar
movimento(ScannedRobotEvent e)	void	null	implementação da estratégia de movimentação circular
mirar(ScannedRobotEvent e)	void	null	Implementação da estratégia de mira
Untitled			

Variáveis globais

Nome	Tipo	Objetivo
UltEner	double	Controle para monitorar a vida do adversário
UltDist	double	Controle para monitorar se houve aproximação do adversário
canto	boolean	Iniciando a variável para monitoramento dos cantos da arena
TOLERANCIA	int	Qual a tolerância para se aproximar das beiradas da arena, esse valor é usado para multiplicar o tamanho do tanque
MOV	int	Quanto o tanque se move por evento
dir	int	Variável para mudar a direção do tanque se necessário
alvo	String	Nome do alvo a ser mirado

Código

```
package dinossauro;
import robocode.*;
import java.awt.Color;
import robocode.util.*;
import java.awt.geom.*;
```

Importando as bibliotecas:

- java.awt.geom: para cálculos em 2D
- robocode.util: para cálculos com ângulos entre tanques

Run( )

```
public class Dino_explosivo extends AdvancedRobot
{

  public static double UltEner = 100.0;

  public static double UltDist =1000;

  public static boolean canto  = false;

  public static int TOLERANCIA = 3 ;

  public static int dir = 1;

  private String alvo ;

  public void run() {

    setColors(Color.blue,Color.blue,Color.white); // body,gun,radar
    setAdjustGunForRobotTurn(true);
    setAdjustRadarForGunTurn(true);

    while(true) {
      if(getRadarTurnRemaining()==0.0){

        setTurnRadarRightRadians(Double.POSITIVE_INFINITY);
      }

      execute();

    }
  }
```

Cria a classe como rôbo avançado

Inicia as variáveis

- Função `run` é onde roda o código inicialmente análogo ao main em c.
- Configura as cores.
- Configura para o radar e a arma girar independente do corpo do tanque.

- Caso o radar não esteja girando ele gira infinitamente para direita.

- Executa as ações que estão na lista de eventos .

onScannedRobot()

```
public void onScannedRobot(ScannedRobotEvent e) {

  if (alvo == null ) {

    alvo = e.getName();

  }

  out.println(">>>>>>>>>>>"+alvo+"<<<<<<<<<<<<");

  if(e.getName().equals(alvo)){

    if(getOthers()>1){

      TOLERANCIA = 1 ;

      MOV = 100 ;
    }

    radar(e);

    movimento(e);

    mirar(e);

  }

}
```

Esse é o evento ativado quando o radar detecta um inimigo, é o mais importante do código.

- Caso não tenha um inimigo predefinido, ele coloca o primeiro inimigo que ver como alvo
- Muda variáveis de ajuste para o rumble, assim, ele anda mais para evitar áreas muito movimentadas e permite que ele chegue mais perto das beiradas, evitando, assim, o caótico centro do rumble.

Se o inimigo que ele achou foi o alvo, executa as funções para:

- Manter o radar centralizado;
- Para gerar os movimentos de desvio;
- Para mirar e atirar.

Funções mais simples

```
public void onHitWall(HitWallEvent e) {

   dir *= -1 ;

  }
```

```
public void onHitByBullet(HitByBulletEvent e) {

    alvo = e.getName();
  }
```

```
public void onHitRobot(HitRobotEvent e  ){

     alvo = e.getName();
   }
```

```
public void onRobotDeath(RobotDeathEvent e){

     if(e.getName().equals(alvo)){

       alvo = null ;
     }
   }
```

Funções principais

Radar ( )

```
public void radar(ScannedRobotEvent e){

    double anguloParaInimigo = getHeadingRadians() + e.getBearingRadians() ;

    double radarTurn = Utils.normalRelativeAngle(anguloParaInimigo - getRadarHeadingRadians());

    double extraTurn = Math.min(Math.atan(36.0 / e.getDistance()), Rules.RADAR_TURN_RATE_RADIANS);

    radarTurn += (radarTurn< 0 ? - extraTurn : extraTurn);

    setTurnRadarRightRadians(radarTurn);

}
```

- Ângulo entre os tanques;
- Quanto o radar tem que girar para localizar o inimigo, garante que esta normalizado;
- `ExtraTurn` é para ampliar o radar, ou seja, ter uma sobra e garantir que o inimigo não vai escapar, sendo que o 36 define quanto além do centro do tanque inimigo o radar vai pegar ;
- Então, implementa essa extra, indo mais pra esquerda do que deveria, ou mais para direita.

Movimento( )

```
public void movimento(ScannedRobotEvent e){


  if(canto()){
    setTurnRight(normalizeBearing(e.getBearing())+90);
  }else{
    setTurnRight(normalizeBearing(e.getBearing())-90);
  }


  // detecta tiro
  double distancia = e.getDistance();

  if (atirou(e)){
      if(distancia< UltDist){

      setAhead(3*MOV*dir);

    }else{
      setAhead(MOV*dir);
    }
  }



  UltDist = distancia;

}
```

Se chegar perto da beirada da arena, muda o lado que vai virar.

Para implementar a movimentar circular, se mantém o corpo do seu tanque sempre a 90 graus da frente do tanque inimigo, assim, facilitando o desvio dos tiros e, então, envia para que lado o tanque inimigo está virado(`e.getBearing()`) para a função normalize bearing, que faz esse movimento linearmente.

O movimento do tanque só é feito se o outro atirar. Isso ocorre para economizar energia, além de que se o tanque chegar mais perto do inimigo, ele anda mais. Se não, ele anda menos, assim, se tem uma aproximação mais cuidadosa.

E a variável `dir` que faz ele mudar de direção caso bata em uma parede, por exemplo.

canto()

```
boolean canto(){
  if(getX() > (getBattleFieldWidth()  - (getHeight()*TOLERANCIA))  || getX() < (getHeight()*TOLERANCIA)||
     getY() > (getBattleFieldHeight() - (getHeight()*TOLERANCIA))  || getY() < (getHeight()*TOLERANCIA)){
    canto = true;
  }
  else {
    canto = false;
  }
   return canto ;
}
```

Função que olha se o tanque esta dentro dos limites no eixo X e Y, simultaneamente, ao usar o tamanho da arena e o tamanho do tanque com o coeficiente TOLERANCIA para achar quais áreas são seguras.

normalizeBearing()

```
double normalizeBearing(double angulo) {
  while (angulo > 180) {

      angulo -= 360;
  }
```

```
  while (angulo < -180) {

      angulo += 360;
  }
  return angulo;
 }
```

Mirar()

Esse é uma das funções mais complexas, por envolver bastante matemática. Aqui segue ela completa e, depois, destrincho suas partes.

```
public void mirar(ScannedRobotEvent e){

  double forcaTiro = Math.min(3.0,getEnergy());

  double xDino = getX();
  double yDino = getY();

  double anguloParaInimigo = getHeadingRadians() + e.getBearingRadians();

  double inimigoX = getX() + e.getDistance() * Math.sin(anguloParaInimigo);
  double inimigoY = getY() + e.getDistance() * Math.cos(anguloParaInimigo);

  double headingInimigo = e.getHeadingRadians();
  double ultHeadingInimigo = headingInimigo;
  double inimigoHeadingDelta = headingInimigo - ultHeadingInimigo;
  double velocidadeInimigo = e.getVelocity();


  double deltaT = 0;
  double battleFieldHeight = getBattleFieldHeight(), battleFieldWidth = getBattleFieldWidth();


  double previsaoX = inimigoX, previsaoY = inimigoY;

  while((++deltaT) * (20.0 - 3.0 * forcaTiro) <  Point2D.Double.distance(xDino, yDino, previsaoX, previsaoY)){

    previsaoX += Math.sin(headingInimigo) * velocidadeInimigo;
    previsaoY += Math.cos(headingInimigo) * velocidadeInimigo;

    headingInimigo += inimigoHeadingDelta;

    if(previsaoX < 18.0  || previsaoY < 18.0 || previsaoX > battleFieldWidth - 18.0|| previsaoY > battleFieldHeight - 18.0){

      previsaoX = Math.min(Math.max(18.0, previsaoX), battleFieldWidth - 18.0);

      previsaoY = Math.min(Math.max(18.0, previsaoY), battleFieldHeight - 18.0);

      break;
    }
  }
  double theta = Utils.normalAbsoluteAngle(Math.atan2(previsaoX - getX(), previsaoY - getY()));



  setTurnGunRightRadians(Utils.normalRelativeAngle(theta - getGunHeadingRadians()));

  setFire(3);

 }
```

Agora suas partes :

```
  double forcaTiro = Math.min(3.0,getEnergy());

  double xDino = getX();
  double yDino = getY();

  double anguloParaInimigo = getHeadingRadians() + e.getBearingRadians();
```

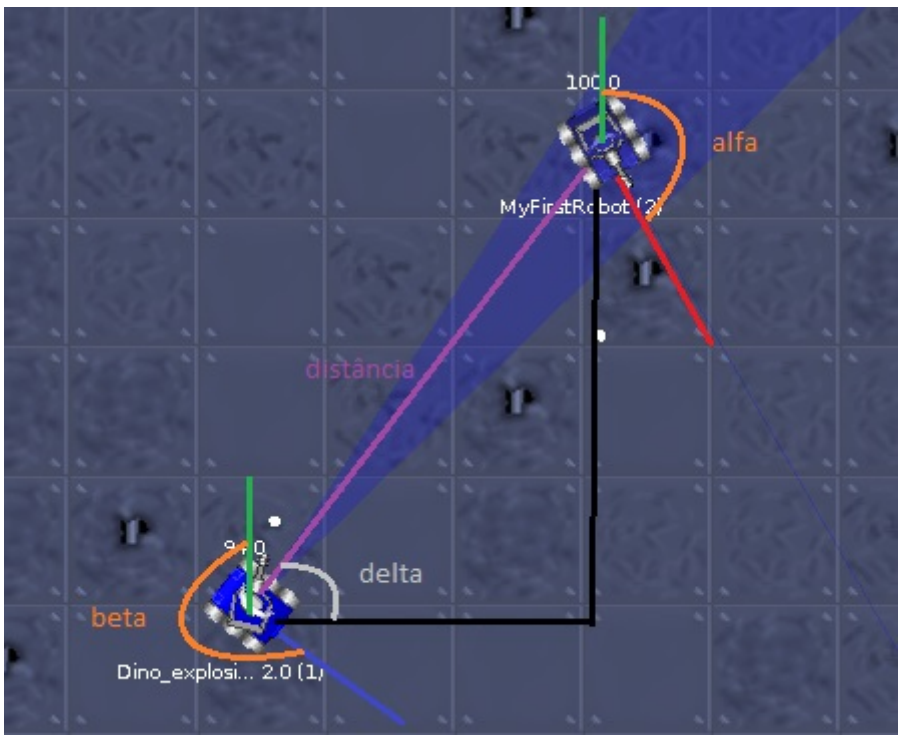
- Garante que eu não vou me matar, ao atirar mais do que eu tenho de vida;
- X e Y do meu tanque;
- Análogo ao radar, pega o ângulo entre o meu tanque e o do inimigo.

```
double inimigoX = getX() + e.getDistance() * Math.sin(anguloParaInimigo);
double inimigoY = getY() + e.getDistance() * Math.cos(anguloParaInimigo);

double headingInimigo = e.getHeadingRadians();
double ultHeadingInimigo = headingInimigo;
double inimigoHeadingDelta = headingInimigo - ultHeadingInimigo;
double velocidadeInimigo = e.getVelocity();

double deltaT = 0;
double battleFieldHeight = getBattleFieldHeight(),
 battleFieldWidth = getBattleFieldWidth();
```

- Calcula a posição do inimigo, com base no ângulo entre os tanques e posição do meu tanque (posição geral na arena);
- Variáveis de controle como:
  - ultima ângulo do inimigo;
  - quanto variou;
  - sua velocidade.



$$anguloParaInimigo = delta = beta + alfa$$

$$Xinimigo = cos(delta) * distância$$

$$Yinimigo = sen(delta) * distância$$

$$deltaAngulo = anguloAnterior - anguloAtual$$

```
double previsaoX = inimigoX, previsaoY = inimigoY;

  while((++deltaT) * (20.0 - 3.0 * forcaTiro) <
  Point2D.Double.distance(xDino, yDino, previsaoX, previsaoY)){

     previsaoX += Math.sin(headingInimigo) * velocidadeInimigo;
     previsaoY += Math.cos(headingInimigo) * velocidadeInimigo;

     headingInimigo += inimigoHeadingDelta;

     if(previsaoX < 18.0  || previsaoY < 18.0 ||
        previsaoX > battleFieldWidth - 18.0   ||
        previsaoY > battleFieldHeight - 18.0){

          previsaoX = Math.min(Math.max(18.0, previsaoX), battleFieldWidth - 18.0);

     previsaoY = Math.min(Math.max(18.0, previsaoY), battleFieldHeight - 18.0);

     break;
  }
  }
```

A maior questão aqui é prever quanto tempo o tiro vai demorar para chegar até o futuro alvo, então, deixa o tempo passar até que a bala consiga viajar até onde o alvo estará.

$$V = S/T$$

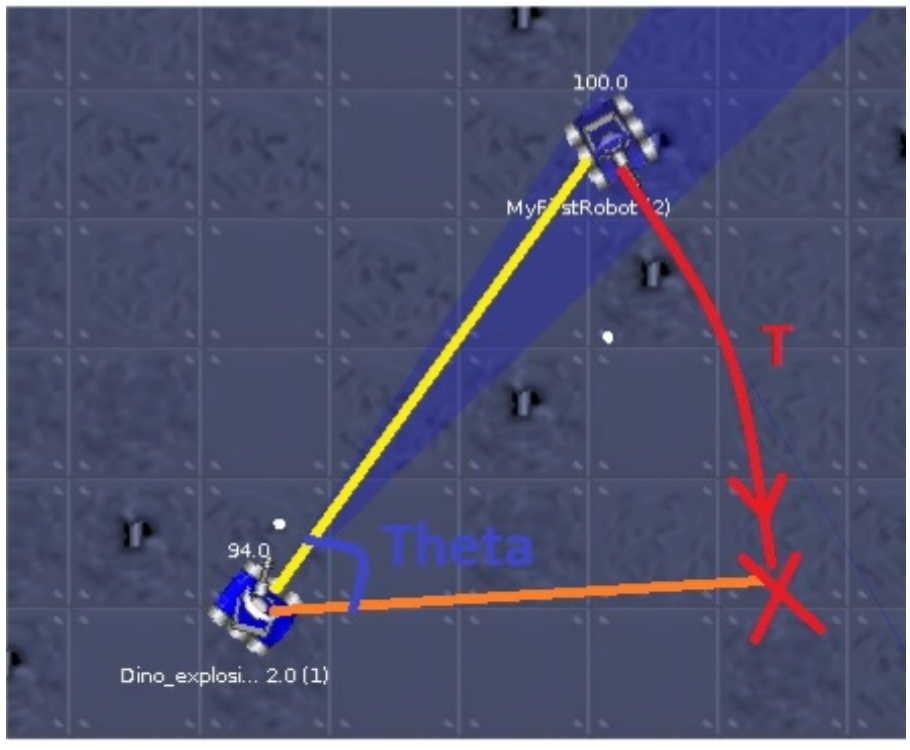
Ao saber que a velocidade da bala pode ser calculada pela fórmula

$$V = 20 - 3 * power$$

calcular a distância de viagem da bala fica fácil, e com geometria analítica, se tem a distancia entre o seu tanque

e a previsão da posição inimiga.

Se não, atualiza a previsão e verifica se ela não está fora da arena.




```
double theta = Utils.normalAbsoluteAngle(Math.atan2(previsaoX - getX(), previsaoY - getY()));



  setTurnGunRightRadians(Utils.normalRelativeAngle(theta - getGunHeadingRadians()));

  setFire(3);

}
```

Então, com uma previsão que pode ser acertada pela bala, acha-se o ângulo que tem que mirar a arma.

Mira

E atira