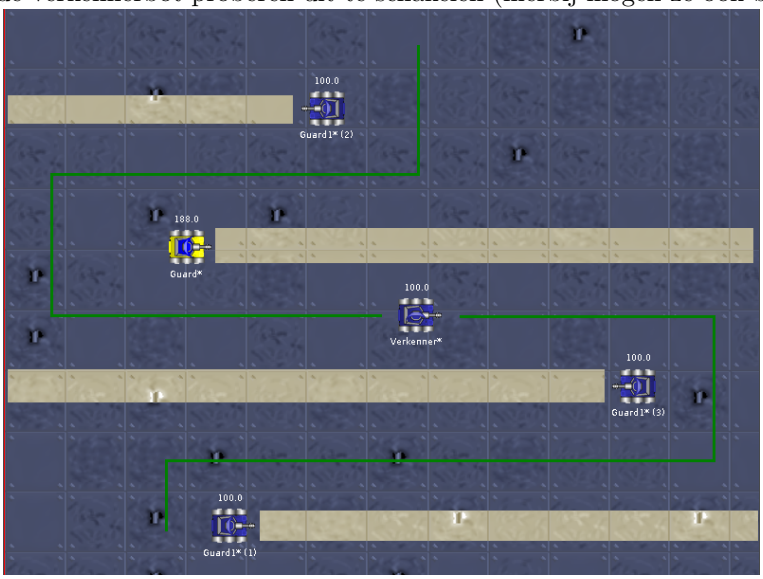
Kunstmatige Intelligentie opdracht 2

Robocode

23 februari 2011

1 Coöperatie

Het doel van de opdracht is om twee verschillende robots te programmeren: *bewakingsbots* en een *verkennerbot*. Er zijn vier bewakingsbots, die ieder een deel van het veld in de gaten moeten houden. Het doel is om de bewakingsbots te verdelen over het veld, zodat elke bot een horizontale strook van het veld bewaakt (zie onderstaande afbeelding). Tijdens het bewegen naar hun plaats mogen ze de verkennerbot niet aanvallen of zijn positie onthouden. Als de bewakingsbots op hun plek zijn aangekomen moeten ze stil blijven staan en vooruit kijken. Als ze een verkennerbot zien, of als ze worden beschoten, moeten ze de verkennerbot proberen uit te schakelen (hierbij mogen ze ook bewegen).



Aan het begin van elke ronde worden alle bots door Robocode op een willekeurige plek in het veld gezet. Gebruik berichten om de vier stroken van het veld te verdelen over de bewakingsbots. Stuur de bots vervolgens naar hun plaats. Denk aan collision detection.

Als de verkennerbot gedetecteerd wordt, start dan een gecoördineerde aanval van alle bewakingsbots op de verkennerbot.

De verkennerbot moet verticaal over het veld heen en weer lopen, en ervoor zorgen dat hij niet gezien wordt door de bewakingsbots.

2 Competitie

Maak een robot voor een toernooi met de andere studenten, met als doel als laatste over te blijven.

3 Achtergrond Robocode

3.1 Algemeen

Robocode wordt gespeeld op een rechthoekig veld. Voor deze opdracht wordt de standaardgrootte van het veld gebruikt, dit is 800 posities breed en 600 posities hoog. De linkerbenedenhoek heeft de coördinaten (0,0). Robots kunnen individueel spelen of georganiseerd in teams.

Een robot bestaat uit een *body*, een *gun* en een *radar*. Deze drie onderdelen kunnen onafhankelijk van elkaar draaien. De radar genereert een `ScannedRobot event` als een andere robot zich binnen het bereik van de radar bevindt. Het event bevat de afstand en de hoek tot de andere robot.

Robots hebben een energieniveau, dat verhoogd en verlaagd kan worden tijdens het spelen. Een robot verliest energie als hij schiet, als hij geraakt wordt of als hij tegen een muur of een andere robot aanrijdt. Een robot krijgt energie door te schieten op een andere robot. Een robot verliest de ronde als zijn energie op 0 uitkomt. Daarnaast is er *inactivity time*: alle robots snel energie als er gedurende een aantal beurten (standaard: 450) geen activiteit plaatsvindt. Activiteit is gedefinieerd als het verliezen van 10 energie of meer door minimaal 1 bot. Schieten kost de schietende robot energie, dus dit kan worden gebruikt om de *inactivity timer* weer op nul te zetten.

3.2 Ingebouwde functies

Er zijn verschillende types robots, met eigen ingebouwde functies. Voor deze opdracht kan je de *AdvancedRobot* gebruiken. Enkele veelgebruikte basisfuncties:

- `out.println(msg)` Print *msg* in het statusvenster van de robot. Dit venster wordt zichtbaar door in het hoofdvenster aan de rechterkant op de naam van de robot te klikken.
- `ahead(afstand)` Rij vooruit (bij negatieve afstand: achteruit)
- `back(afstand)` Rij achteruit (bij negatieve afstand: vooruit)
- `fire(kracht)` Schiet een kogel met impact *kracht*
- `turnRight(graden)` Draai de robot naar rechts (in graden)

- `turnRightRadians(radialen)` Draai de robot naar rechts (in radialen)
- `turnGunRight()` Draai de gun naar rechts
- `turnRadarRight()` Draai de radar naar rechts
- `getHeading()` Oriëntatie van de robot ten opzichte van de 0-gradenlijn (recht omhoog)
- `getBearing()` Oriëntatie van een object (bijvoorbeeld een andere robot) ten opzichte van de eigen robot
- `getDistance()` Afstand tot een object

Van de meeste functies met een draaihoek of oriëntatie zijn ook versies met *Left* en/of met radialen beschikbaar, zoals `turnGunLeft` of `getHeadingRadians`. Zie voor een volledig overzicht `http://robocode.sourceforge.net/docs/robocode/`.

3.3 Events

Robocode is *event-driven*, dit betekent dat je code kunt schrijven die automatisch wordt uitgevoerd als er een bepaalde gebeurtenis plaatsvindt. Een gebeurtenis is bijvoorbeeld 'tegen een muur aanrijden', dit genereert een `HitWallEvent`. Je kunt een functie `onHitWall(HitWallEvent e)` schrijven (dit heet een *event handler*) die op deze gebeurtenis reageert. Het event heeft eigenschappen die beschikbaar zijn via de variabele (`e` in dit geval). Een paar voorbeelden van veel voorkomende events:

- `onScannedRobot(ScannedRobotEvent e)` De radar heeft een andere robot gedetecteerd. Eigenschappen: `getBearing`, `getHeading`, `getDistance`, `getEnergy`, `getVelocity`, `getName`.
- `onHitRobot(HitRobotEvent e)` De robot heeft een andere robot geraakt. Eigenschappen: `getBearing`, `getEnergy`, `getName`, `isMyFault` (waar als jouw robot naar de andere robot toe aan het rijden was).
- `onHitWall(HetWallEvent e)` De robot heeft een muur geraakt. Eigenschap: `getBearing`.
- `onHitByBullet(HitByBulletEvent e)` De robot is geraakt door een kogel. Eigenschappen: `getBearing`, `getHeading`, `getName`.

3.4 Communicatie

Robots van het type `TeamRobot` kunnen met elkaar communiceren. Een nieuw team kan samengesteld worden in het hoofdscherm van Robocode onder *Robot-Create a robot team*. Declareer de robots als TeamRobot (`public class RobotNaam extends TeamRobot`) en importeer `java.io.IOException` om exceptions rond de communicatie op te vangen. Enkele functies die je kunt gebruiken voor TeamRobots:

- `getTeammates()` Array met alle teammates
- `broadcastMessage(msg)` Verstuur `msg` naar alle teammates
- `sendMessage(receiver,msg)` Verstuur `msg` naar `receiver`
- `onMessageReceived(MessageEvent)` Na het sturen van een bericht wordt een `MessageEvent` gegenereerd bij alle ontvangers. Dit event bevat de verzendende robot (`getSender()`) en het bericht (`getMessage()`).