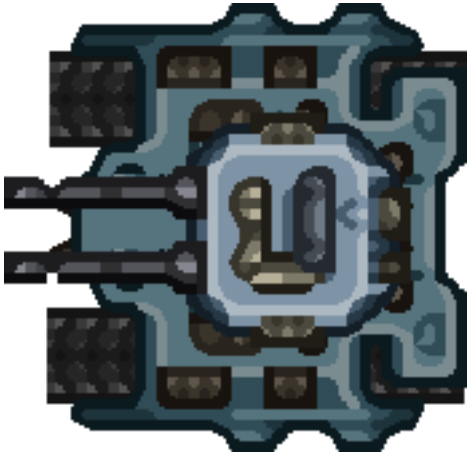
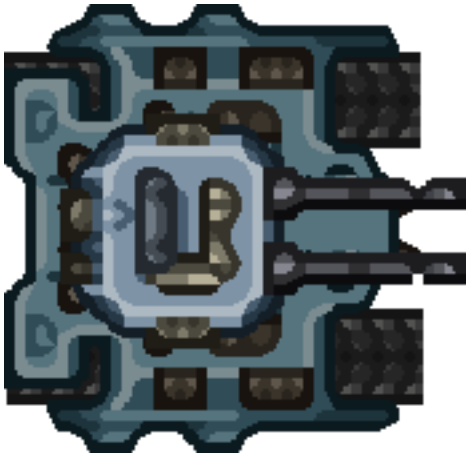


JROBOTS'2011



online Kick-off

30.11.2011
www.jrobots.de
Wolfgang Scholz

Inhalt

Vorstellung

- JROBOTS :
  - Java-Programmierwettbewerb
  - Echtzeitsimulation
- Historie:
  - Crobots (1985, ASCII-Grafik) und Robocode
  - 2002: Teilnahme an in Passau entwickeltem 'JRobots'
  - Entwicklung an JRobots seit 2005
  - Wettbewerbs-Events bisher: 2005-05, 2007-12, 2008-12, 2009-07, 2010-05, 2010-07, 2011-07

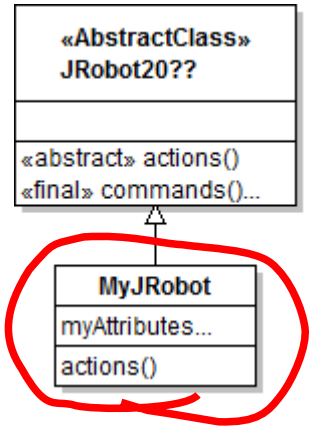
Startvoraussetzungen

Systemvoraussetzungen:

- Java-Entwicklungsumgebung (Java1.5-fähig)
- Bot-Entwicklungspaket bitte herunterladen:

http://jrobots.de/index.php/de/files

Implementiert wird ***eine*** Klasse.
Der Code eines Starter-Bots ist mitgeliefert:
OnsetBot.java



Hinweise:

- empfohlene Arbeitsweise: *Pair Programming*
- Entwicklungszeit für lokale Events: 4 h
- einige Gegner sind mitgeliefert: live-Debug nutzen!

Duellregeln

zur Ermittlung der online-Rangfolge:

jeder gegen jeden

Kampf 1 gegen 1

Gewonnen hat, wer als Letzter übrig bleibt.

Dauert ein Duell zu lang, wird der Kampf abgebrochen:
Nach Abbruch gewinnt größere Restintegrität.
Bei gleicher Integrität gewinnt kleinerer Zustand.

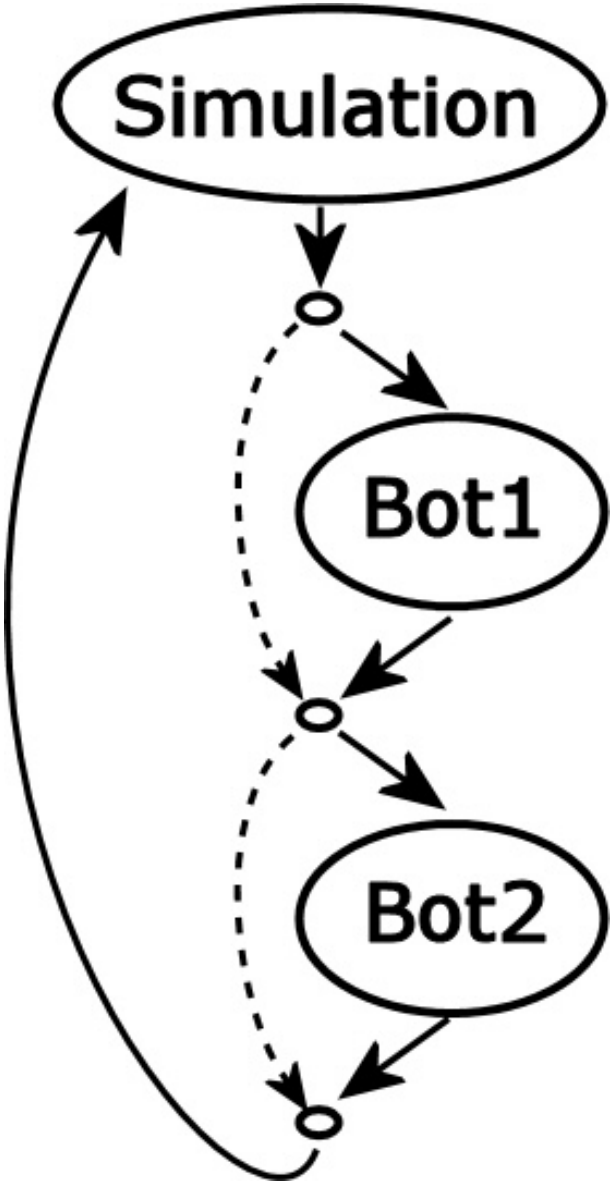
Simulationszyklus

- Simulation und Bots wechseln sich ab

- Simulation ruft die actions() - Methode der Bots auf
  (init() - Methode 1x am Anfang)

- Bot erhält Informationen zum momentanen Zustand und setzt Befehle für die nahe Zukunft ab

- Die in actions() geplanten Aktionen werden bis zum nächsten Aufruf ausgeführt



Integrität und Energie

Integrität: Wer als erster keine mehr hat, verliert

Wahl: Viele Aktionen sind verfügbar

Aktionen: Jede Aktion kostet Energie

Aktionen sind: Fahren, Gegner scannen, Schießen

Energie: Sammelt sich mit der Zeit an

keine Energie: Aktionen werden nicht ausgeführt

Nachladezeiten: Gibt es nicht (max 1 Geschoss / Frame, max 1 Gegnerscan / Frame)

Energiespeicher: Unbegrenzt, steigende Ineffizienz

Aktionen

Aktionen beeinflussen das Botverhalten

Die meisten sind gültig, bis sie von einem neuen Kommando überschrieben werden

Aktionen verbrauchen Energie
Der Energieverbrauch lässt sich ermitteln

```
addDebugArrow(Vector, Vector) : void
addDebugCrosshair(Vector) : void
addDebugLine(float, float, float, float) : void
addDebugLine(Vector, Vector) : void
setAutopilot(Angle, double) : void
setBodyColor(Color) : void
setBoost() : void
setDebugLines(Vector[]) : void
setDebugText(String) : void
setDropMineCommand(boolean) : void
setDropTankTrapCommand(boolean) : void
setLaunchProjectileCommand(Angle) : void
setLaunchRocketCommand(Angle) : void
setNameColor(Color) : void
setRocketHeading(Angle) : void
setScanAperture(Angle) : void
setScanDirection(Angle) : void
setTurretColor(Color) : void
```

Debug-Ausgaben

Fahren

Fahren

Debug-Ausgaben

Arsenal

Lenkrakete

Scanner

Sensorik

Funktionen des Bots geben Aufschluss über den momentanen Zustand

Tipps:
- Code-Vervollständigung nutzen
- JavaDoc ist zu allen Methoden vorhanden

```
getEnergyConsumptionBooster() : double
getEnergyConsumptionEngine() : double
getEnergyConsumptionMine() : double
getEnergyConsumptionProjectile() : double
getEnergyConsumptionRocket() : double
getEnergyConsumptionRocketRedirection() : double
getEnergyConsumptionScanner() : double
getEnergyConsumptionTrap() : double
getEnergyProduction() : double
getFramesPerSecond() : double
getMaxArenaDiameter() : double
getMaxBackwardVelocity() : double
getMaxForwardVelocity() : double
getMaxScanAperture() : Angle
getMaxVelocityProjectile() : double
getMaxVelocityRocket() : double
getProjectileSpeed() : double
getHealth() : double
getLastScan() : Scan
getOrientation() : Angle
getPosition() : Vector
getProximityScanDroppable() : Vector
getProximityScanLaunchable() : Vector
getRocketPosition() : Vector
getVelocity() : Vector
isScanFromNow() : boolean
memoryConsumption : int
getBodyColor() : Color
getEnergy() : double
getHealth() : double
getNameColor() : Color
getTurretColor() : Color
```

Energie

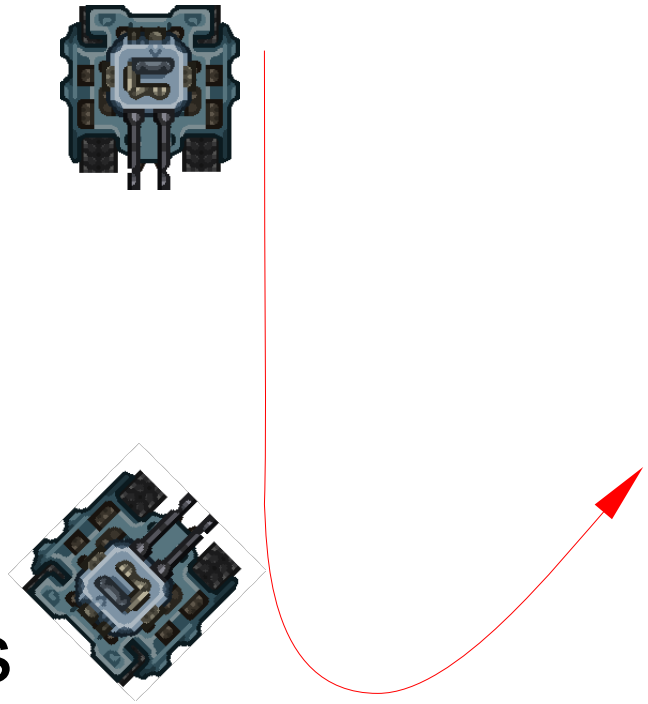
Scanner

Radar

Lenkrakete

Energie

Fahren



Kinetisches Modell

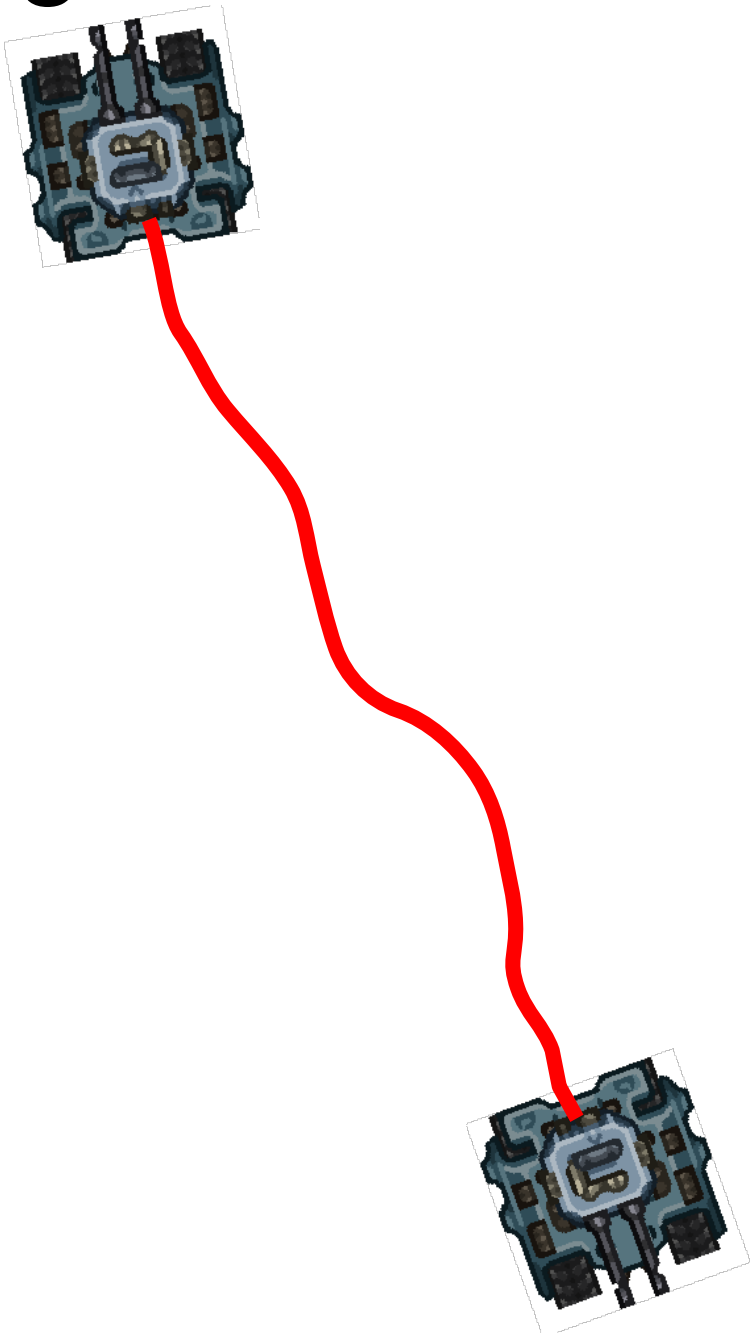
Trägheit und Beschleunigung
(auch beim Drehen)

Vorwärts schneller als rückwärts

Autopilot steuert Ketten:
nur Richtung und Geschwindigkeit anzugeben

Fahren kostet Energie

Arenagrenze



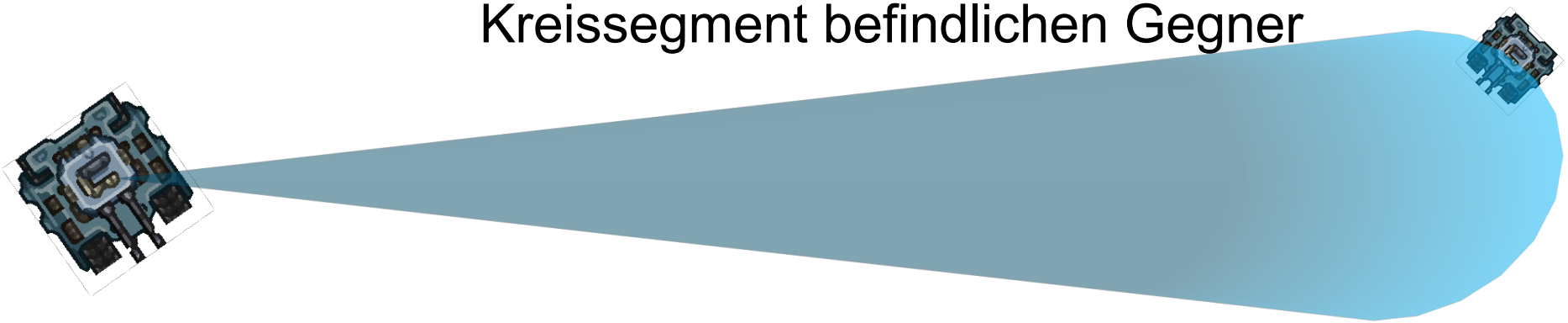
Arena ist prinzipiell unbegrenzt

Zwei Bots dürfen sich nicht unbegrenzt weit voneinander entfernen

„Gummischnur“ hindert sie daran

Scannen

Der Scanner misst die Distanz zum nächsten im gescannten Kreissegment befindlichen Gegner

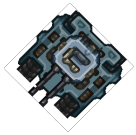


Scanbefehle speichern Scanparameter, die beim nächsten Scannen „verbraucht“ werden
Sie müssen für jeden Scan neu gesetzt werden

Es kann in jedem Zyklus ein Mal gescannt werden.
Jeder Scan benötigt Energie.

Schießen

Mit der im Panzerturm eingebauten Abschussvorrichtung ist es dem Bot möglich, in beliebige Richtungen zu schießen



Die Abschussvorrichtung muss nach jedem Feuern erneut ausgerichtet werden

Der Turm unterliegt nicht der Trägheit

Es kann in jedem Zyklus ein Mal geschossen werden. Schüsse verbrauchen aber mehr Energie, als in einem Zyklus generiert wird

Treffen

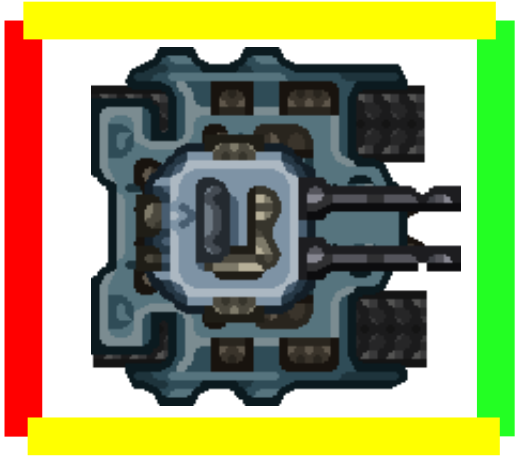
Die Projektilgeschwindigkeit ist konstant und nicht von der Geschwindigkeit des feuernden Bots abhängig

Ein Projektil richtet Schaden an,
wenn es in seiner Laufbahn einen Bot kreuzt

Fällt die Stabilität eines Panzers unter 0,
so wird er entfernt und der Bot hat verloren

Treffen

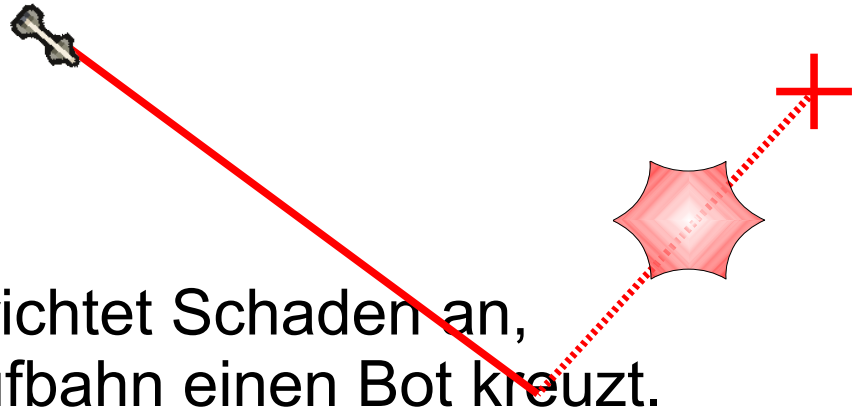
Die Seiten eines Bots sind verschieden gut abgeschirmt



Vorne am besten, hinten am wenigsten

Lenkrakete

Die Lenkrakete kann im Flug gesteuert werden



Eine Lenkrakete richtet Schaden an,
wenn es in seiner Laufbahn einen Bot kreuzt.

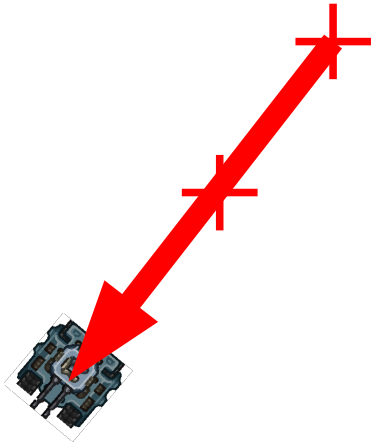
Die Zielrichtung der Rakete kann beliebig oft verändert werden, was jedes Mal Energie kostet.

Es ist immer nur eine Rakete in der Luft.

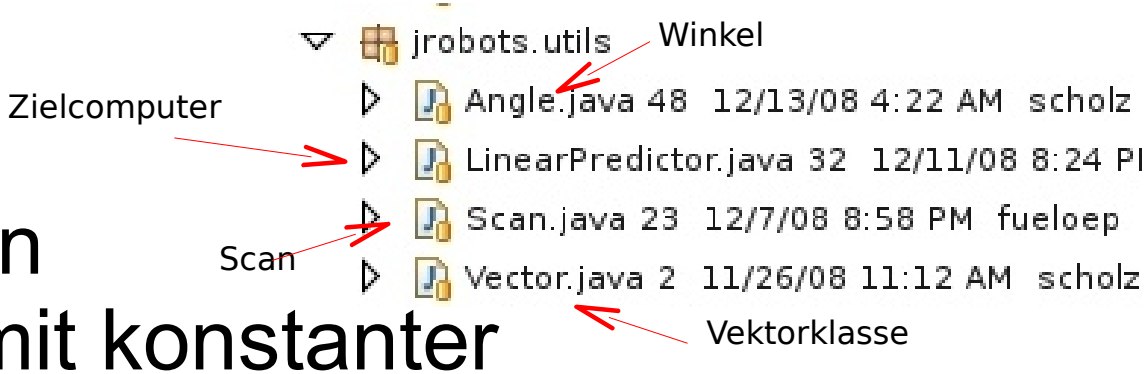
Lenkraketen sind träger als als Projektile, richten aber ebenso viel Schaden an.

Zielcomputer

Bots sind mit einem Zielcomputer ausgestattet



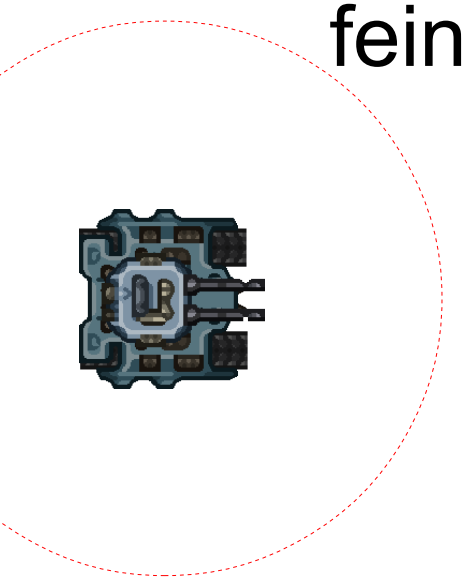
direkt mit Scanwerten füttern



extrapoliert Zielposition (Annahme: Ziel fährt mit konstanter Geschwindigkeit)

Ausweichen

Das Projektilradar erkennt das nächste anfliegende feindliche Projektil in mittlerem Umkreis

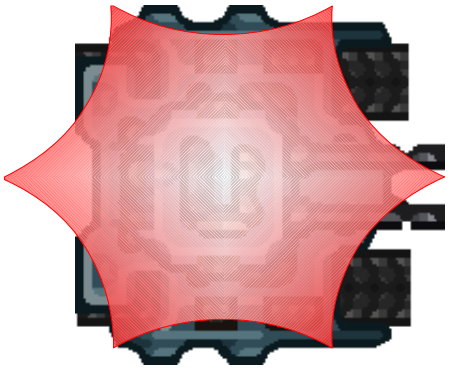


Der Minenradar erkennt die nächste Mine in kleinem Umkreis

Der Booster hilft beim Ausweichen, verbrennt aber viel Energie.

Minen

Minen helfen, Verfolger abzuwehren



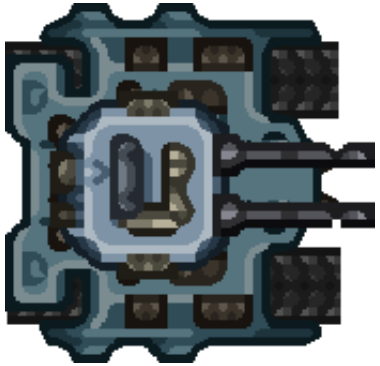
Überfahren zündet die Mine (auch eigener Bot)

Flächenschaden auf alle umliegenden Fahrzeuge

Wie beim Projektil kosten Minen Energie
Minen „zerfallen“ nach einiger Zeit

Sperren

Sperren blockieren Verfolger



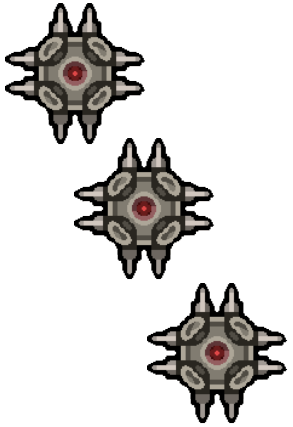
Überfahren bremst ab (auch eigenen Bot)

kein Schaden, Sperre zerbricht ~5s nach Überfahrt

Kostet weniger Energie als Mine
Sperren „zerfallen“ nach einiger Zeit

Sperren

Sperren schützen vor Geschossen

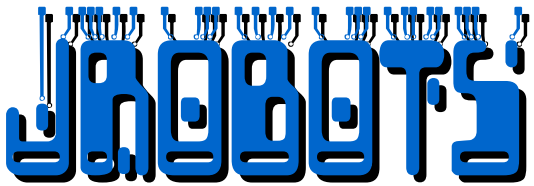


Geschosse werden aufgefangen

Sperre zerbricht ~2s nach Beschuss

Das Arsenal

- Projektile
  - Kanone
  - Lenkrakete
- Ablegbares
  - Mine
  - Sperre/Mauer
- Verteidigung
  - Fahren
  - Projektilradar
  - Booster



Säumigkeit

Bots dürfen ihr Zeitkontingent nicht überschreiten

Bei Zeitüberschreitung oder Exception
wird der Bot „ersetzt“ (neu instanziert)

Der Zustand des Bots vor der Ersetzung geht
verloren

Speicherverbrauch

In jedem Simulationsschritt wird der Systemzustand neu abgespeichert

Speicherverbrauch der Bots wird in der GUI angezeigt

Bei unverhältnismaßig vielen Zustandsdaten kann der Bot vom Wettbewerb ausgeschlossen werden

Oberfläche


JRobots
Simulate
Play
Crop
DebugLines
Save to disk!
Load from disk...
0s
10s
20s
30s
42s
ArmyBotNewMk1
ArmyBotNewMk2
Bobo
(2156) BogBot
Bot42
BotanicBot
Botty
ChameleOnDrugs
ColonelDuck
CrazyDave
DancingQueen
Desperado
Schrutz
(H:100.0%, E:0.03)
BogBot
(H:100.0%, E:0.04)
SchrutzRevolutions
E:0.17)
Minerator
1) durch Klicken
mehrere Bots
an den Start bringen

Oberfläche


JRobots
Simulate
Play
Crop
DebugLines
Save to disk!
Load from disk...
0s
10s
20s
30s
42s
ArmyBotNewMk1
ArmyBotNewMk2
Bobo
(2156) BogBot
Bot42
BotanicBot
Botty
ChameleOnDrugs
ColonelDuck
CrazyDave
DancingQueen
Desperado
1) durch Drücken auf „Simulate“ schreitet die Zeitleiste voran

Oberfläche


JRobots
Simulate
Play
Crop
DebugLines
Save to disk!
Load from disk...
0s
10s
20s
30s
42s
ArmyBotNewMk1
ArmyBotNewMk2
Bobo
(2156) BogBot
Bot42
BotanicBot
Botty
ChameleOnDrugs
ColonelDuck
CrazyDave
DancingQueen
Desperado
3) „Play“ startet die Wiedergabe
Schrutz
(H:100.0%, E:0.03)
BogBot
(H:100.0%, E:0.04)
SchrutzRevolutions
(H:100.0%, E:0.17)
Minerator
(H:100.0%, E:0.13)
50

JROBOTS

Befehlsreferenz

utils

Angle
- angle: double
- Angle()
- Angle()
- normalize()
- getValueAsDegrees()
- getValueAsRadians()
- add()
- sub()
- mult()
- getNegative()
- getOpposite()
- getNextQuadrant()
- getPreviousQuadrant()
- isPositive()
- angularDistance()
- coneAngle()
- toString()

Vector
- Vector()
- Vector()
- Vector()
- Vector()
- getX()
- getY()
- add()
- sub()
- mult()
- getNormal()
- getNegative()
- getLength()
- getAngle()
- vectorProduct()
- projection()
- rotate()
- distanceTo()

Scan
- scanDirection: Angle
- scanAperture: Angle
- distanceToTarget: double
- scannerPosition: Vector
- timeOfScan: double
- Scan()
- estimatedTargetPosition(
- getArcLength()
- isTargetLocated()

LinearPredictor
- predict()

- getEnergyConsumptionBooster() : double
- getEnergyConsumptionEngine() : double
- getEnergyConsumptionMine() : double
- getEnergyConsumptionProjectile() : double
- getEnergyConsumptionRocket() : double
- getEnergyConsumptionRocketRedirection() : double
- getEnergyConsumptionScanner() : double
- getEnergyConsumptionTrap() : double
- getEnergyProduction() : double
- getFramesPerSecond() : double
- getMaxArenaDiameter() : double
- getMaxBackwardVelocity() : double
- getMaxForwardVelocity() : double
- getMaxScanAperture() : Angle
- getMaxVelocityProjectile() : double
- getMaxVelocityRocket() : double
- getProjectileSpeed() : double
- addDebugArrow(Vector, Vector) : void
- addDebugCrosshair(Vector) : void
- addDebugLine(float, float, float, float) : void
- addDebugLine(Vector, Vector) : void
- getHealth() : double
- getLastScan() : Scan
- getOrientation() : Angle
- getPosition() : Vector
- getProximityScanDroppable() : Vector
- getProximityScanLaunchable() : Vector
- getRocketPosition() : Vector
- getVelocity() : Vector
- isScanFromNow() : boolean
- setAutopilot(Angle, double) : void
- setBodyColor(Color) : void
- setBoost() : void
- setDebugLines(Vector[]) : void
- setDebugText(String) : void
- setDropMineCommand(boolean) : void
- setDropTankTrapCommand(boolean) : void
- setLaunchProjectileCommand(Angle) : void
- setLaunchRocketCommand(Angle) : void
- setNameColor(Color) : void
- setRocketHeading(Angle) : void
- setScanAperture(Angle) : void
- setScanDirection(Angle) : void
- setTurretColor(Color) : void

Math
java.lang
- E: double
- PI: double
- abs()
- abs()
- abs()
- abs()
- min()
- min()
- min()
- min()
- max()
- max()
- max()
- max()
- sin()
- cos()
- tan()
- asin()
- acos()
- atan()
- atan2()
- exp()
- log()
- sqrt()
- pow()
- IEEEremainder()
- ceil()
- floor()
- rint()
- round()
- round()
- random()
- toRadians()
- toDegrees()
- cbrt()
- cosh()
- expm1()
- hypot()
- log10()
- log1p()
- signum()
- signum()
- sinh()
- tanh()
- ulp()
- ulp()

JROBOTS

www.jrobots.de

Nach Anmeldung kann der eigene Bot
am

Online-Wettbewerb teilnehmen