

# Projektarbeit

## Robocode



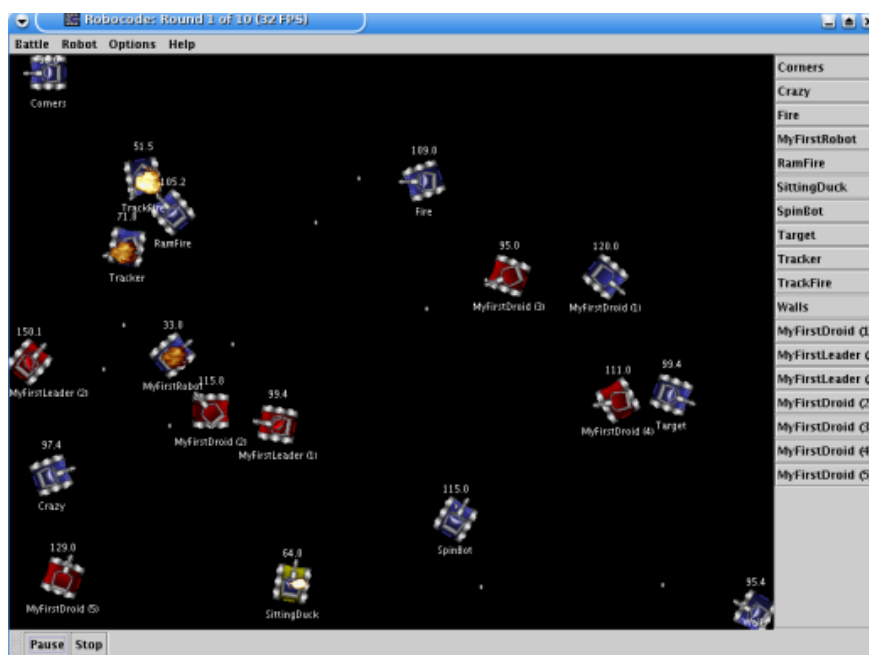
**Team: JUDOMAX**

<u>Name</u>	<u>Matrikelnummer</u>
Junior Dassen Nana	883611
Maxime Paul Mendomo	883155
Billy Max Tsane Tsafack	882814
Donald Richter Donchi	883894
Roch Dominique Kamgaing	884273
<u>Abgabedatum:</u>	<u>14.07.2022</u>

## Einführung (Junior)

Robocode ist ein Programmierspiel, bei dem virtuelle Roboter auf einem Spielfeld gegeneinander kämpfen. Es wurde von IBM unter einer OpenSource Lizenz entwickelt.

Das Spiel bietet zum einen eine praxisnahe Anwendung der bisher gelernten Programmierkenntnisse aus den Fächern „Grundlagen der Informatik“ und „Objektorientierte Programmierung“, zum anderen erlaubt es, die in Softwareentwicklung vorgestellten Vorgehensweisen anzuwenden. Dabei spielt die ausführliche Dokumentation seitens der Entwickler und die bereitgestellte Laufzeit- und Entwicklungsumgebung eine wichtige Rolle. Robocode richtet sich sowohl an Java Anfänger - die dadurch erste Erfahrungen sammeln und eine gewisse Routine erlangen können als auch an Profis, die sich mit anderen in virtuellen Wettkämpfen messen können.



Robocode basiert auf der Programmiersprache Java und ermöglicht die Programmierung eines Roboters, der in einer virtuellen Arena mit anderen Robotern konkurrieren muss. Ziel ist es, als Team oder allein, die gegnerischen Roboter zu besiegen.

Um den feindlichen Robotern Schaden zuzufügen steht eine Schuss-Funktion und eine Ramm-Funktion zur Verfügung. Die Wahrnehmungsmöglichkeiten der Umgebung des Roboters bestehen dabei aus einem Radar, der z.B. das „Sehen“ von gegnerischen Robotern ermöglicht und verschiedenen „Events“, die beim Eintreten einer bestimmten Situation eine Rückmeldung liefern. Erstellt man

ein Roboterteam, so stehen besondere Funktionen zum Nachrichtenaustausch untereinander bereit. Durch das Übermitteln von Nachrichten können so Teamstrategien, wie z.B. Fokusfeuer, umgesetzt werden.

Bei dem Start einer Runde wird jedem Roboter eine festgelegte Energiemenge zugewiesen. Durch verschiedene Aktionen oder Ereignisse, wie z.B. Feuern oder der Kollision mit einer Wand, geht Energie verloren. Soll die Energie erhöht bzw. aufgefüllt werden, so muss der Gegner gerammt oder mit abgefeuerten Projektilen getroffen werden. Erreicht die Energie, durch Bewegung oder das Abfeuern von Projektilen, eines Roboters 0 kann dieser keine Aktionen mehr ausführen und wird immobil, wodurch er ein leichtes Ziel für seine Gegner darstellt. Wird die Energie durch Schaden, z.B. durch eine Kollision mit einem anderen Roboter oder durch Projektile, auf 0 gesenkt, explodiert der Roboter. Der Sieger, meist der überlebende Roboter, wird durch eine Punktzahl ermittelt, die sich aus verschiedenen Einzelbewertungen. wie beispielsweise Überlebensbonus oder Schadensboni zusammensetzt.

## **Planung (Junior)**

### **Projektmanagement**

#### Aufgabenteilung:

<b>Verantwortlicher</b>	<b>Aufgabe</b>
Junior Dassen Nana	Planung(Projektleiter)
Donald Richter Donchi	Analyse
Billy Max Tsane Tsafack	Entwurf
Maxime Paul Mendo, Kamgaing Roch Dominique	Programmierung
alle	Tests
alle	Dokumentation

## Projektplan/Zeitplan :

<u>Wann :</u>	<u>Was :</u>	<u>Wer :</u>
Woche 1 (23.05.22 bis 29.05.22)	Aufgabenstellung analysieren. Ziel festsetzen. Aufgaben verteilen. Projektbearbeitung festlegen. Arbeitsmethoden wählen.	<u>Junior,Donald, Billy, Maxime, Dominique</u>
Woche 2 (30.05.22 bis 05.06.22)	Programmierung: Orientierungsphase, erste Blicke in Beispielroboter werfen und einen Überblick verschaffen.	<u>Junior,Donald, Billy, Maxime, Dominique</u>
Woche 3 (06.06.22 bis 12.06.22)	Einarbeitung in Robocode, Möglichkeiten ausloten, Analyse, Design und Planung.	<u>Junior,Donald, Billy, Maxime, Dominique</u>
Woche 4 (13.06.22 bis 19.06.22)	Programmierung der Roboter, Fragen klären.	<u>Junior,Donald, Billy, Maxime, Dominique</u>
Woche 5 (20.06.22 bis 26.06.22)	Programmierung der Roboter, Teamfunktionen einfügen und Roboterteam erstellen. Team Simulation mit anschließender Analyse und ggf. Änderungen vornehmen.	<u>Junior,Donald, Billy, Maxime, Dominique</u>
Woche 6 (27.06.22 bis 03.07.22)	Dokumentation, TheorieFragen	<u>Junior,Donald, Billy, Maxime, Dominique</u>
Woche 7 (04.07.22 bis 10.07.22)	Dokumentation, Theorie-Frage	<u>Junior,Donald, Billy, Maxime, Dominique</u>
Woche 8 (11.07.22 bis 14.07.22)	Letzte Besprechungen	<u>Junior,Donald, Billy, Maxime, Dominique</u>

## Projektplan: (Junior, Donald)

Woche 1: Die Aufgabenstellung wird analysiert und die ersten groben Meilensteine bzw. Ziele werden festgelegt. Eine Verteilung der Rollen wie z.B. Projektmanager, Programmierer etc. wird vorgenommen und einzelne Verantwortungsbereiche eingeteilt. Das weitere Vorgehen wie z.B. die Entwicklungsmethode und weitere Team-Meetings werden erörtert und festgehalten.

Woche 2: Der Schwerpunkt liegt auf dem Programmierteil, hier wird eine Orientierungsphase angesetzt damit sich das Team langsam an Robocode herantasten kann. Es werden die Beispielroboter näher angeschaut und generell ein Überblick verschafft.

Woche 3: Der Schwerpunkt liegt auf dem Programmierteil. Eine tiefere Einarbeitung in Robocode findet statt. In einer Planungsphase wird gleichzeitig die generelle Strategie festgelegt. Für die entsprechende Strategie wird eine Taktik erörtert und mögliche geeignete Methoden ausgelotet bzw. analysiert, indem passende Roboter angeschaut oder die zur Verfügung stehenden Möglichkeiten erkundet und auf Machbarkeit überprüft werden.

Woche 4: Die eigentliche Programmierarbeit findet statt. Nachdem bereits genaue Vorstellungen der Roboter festgelegt wurden, werden diese entsprechend zusammengebaut. Zuerst werden die Grundfunktionen wie etwa Bewegung, Feuern oder Scannen umgesetzt und simuliert. Sind die entsprechenden Simulationen positiv verlaufen, werden die bestehenden Funktionen übernommen oder bei Bedarf erweitert und verfeinert.

Woche 5: Die Endphase der Programmierung wird eingeläutet. Bei den nun funktionsfähigen Robotern werden die Team-Funktionen eingefügt und anschließend findet eine Team-Simulation statt. Treten Probleme auf, müssen diese behoben werden.

Woche 6: Das Erstellen der Dokumentation beginnt. Es müssen unter anderem Skizzen und Erklärungen zu dem verwendeten Code erstellt werden. Die Bearbeitung der Theorie-Fragen beginnt.

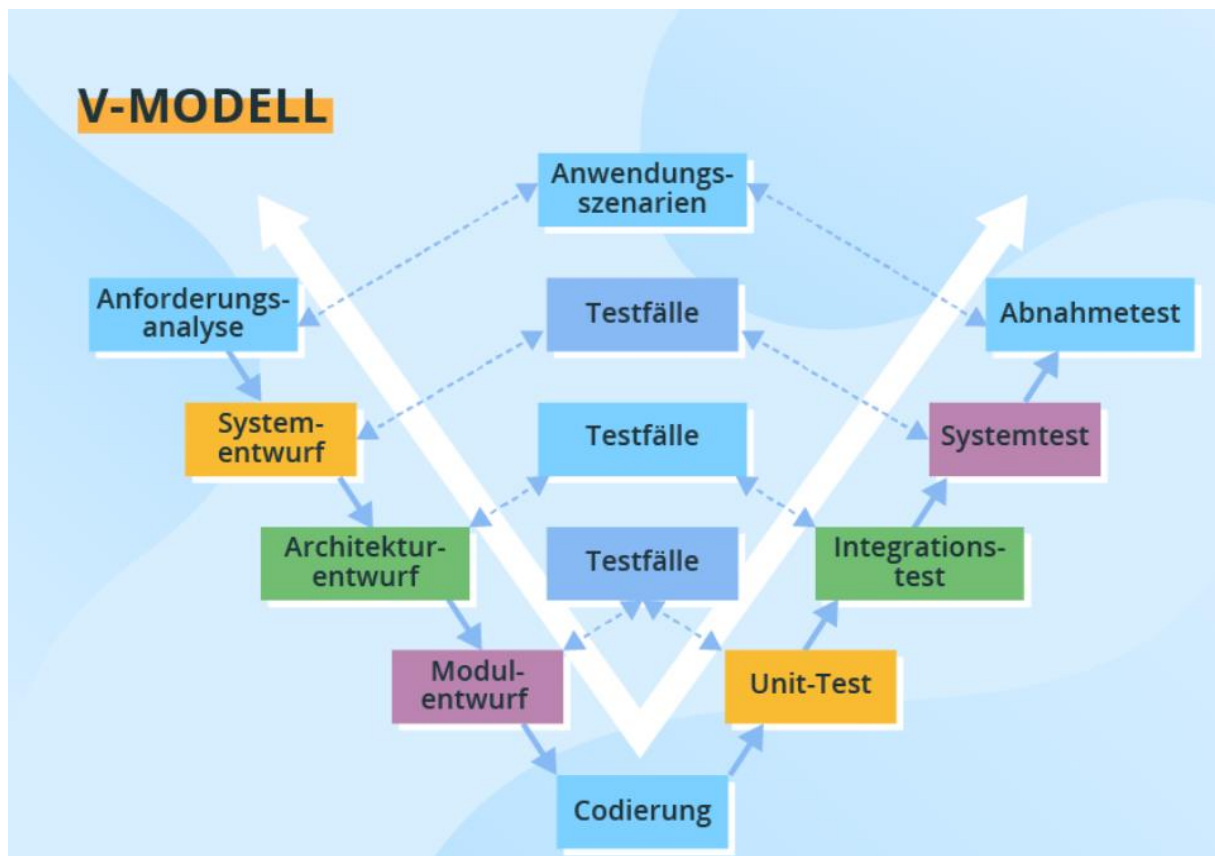
Woche 7: Abschließen der Dokumentation inklusive der Theorie-Fragen. Klärung letzter offener Fragen und Vorbereitung auf Abgabe des Projekts.

Woche 8: Letzte Besprechungen und Abgabe des Projekts.

## Vorgehensmodell (Junior,Donald):

Wir haben zum Erstellen unserer Roboter die Methode V-Modell benutzt, dabei werden alle Phasen nacheinander, also linear, durchgelaufen. Aber für jede Stufe werden entsprechende Testaktivitäten definiert.

Einerseits ermöglicht das, die Softwarequalität effizienter zu kontrollieren und damit Projektrisiken auf ein Minimum zu reduzieren. Andererseits gehört das V-Modell dadurch zu einem der kosten- und zeitaufwändigsten Modelle.



Quelle: [Vorgehensmodelle der Softwareentwicklung im Vergleich: Grundprinzipien und Anwendungsbereiche \(scnsoft.de\)](http://www.scnsoft.de)

# Lastenheft

## 1. Ausgangssituation und Zielsetzung

Mit der Ausweitung des Internetzugangs wurden Multiplayer-Spiele immer beliebter. Trotzdem spielen Einzelspieler-Spiele immer noch eine große Rolle. Da bei diesen die Gegner vom Computer gesteuert werden, haben die Entwickler die Aufgabe, diese mit einer **künstlichen Intelligenz (KI)** auszustatten. Solche Computergegner haben Parallelen zu modernen, autonom agierenden Maschinen: Beide erhalten über (reale oder virtuelle Sensoren) Informationen über ihre Umwelt und stehen vor der Aufgabe, basierend auf diesen Informationen über ihr weiteres Vorgehen zu entscheiden. Ziel des Projektes ist, diese Entscheidungsprozesse zu beleuchten. Dazu wird das Programmierspiel [Robocode](#) benutzt. Das Ziel in Robocode ist es, ein Team von zwei Robotern zu programmieren, das in einer Kampfarena gegen andere Roboter antritt.

## 2. Produkteinsatz

### 2.1. Zielgruppen

1. Schüler und Studenten, denen die Möglichkeit geboten wird, Kenntnisse in den Bereichen Programmierung und Robotik im Rahmen ihrer MINT-Kenntnisse (Wissenschaft, Technik, Ingenieurwesen und Mathematik) zu erwerben und zu entwickeln;
2. Lehrer, deren Profile durch das Programm zur beruflichen Weiterbildung weiterentwickelt und gestärkt werden und die eine Chance bekommen "Code / Robotic Trainer" zu werden;
3. Schulbehörden, Behörden und politische Entscheidungsträger

### 2.2. Betriebsbedingungen

#### 2.2.1. Umgebung des Systems

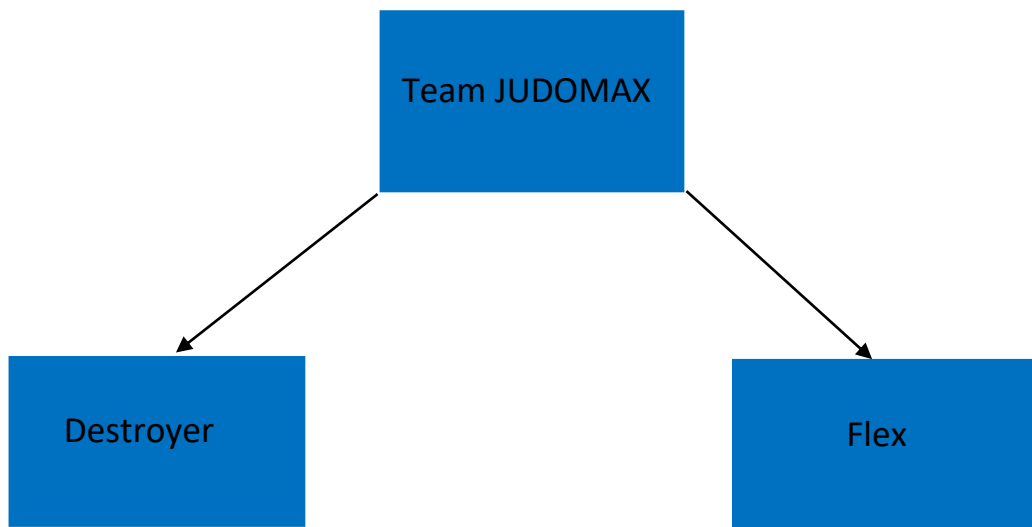
Die Software benötigt zu ihrer korrekten Funktion mindestens folgende Dienste auf einem netzmäßig erreichbaren Rechner:

- Das Robocode Programm
- JRE (Java Runtime Environment)

### 2.2.2. Tägliche Betriebszeit

Da das Produkt auf den o.g. Serverdiensten basiert, ist es bei deren Verfügbarkeit zeitlich unbegrenzt einsatzbereit.

## 3. Übersicht



Das Roboterteam JUDOMAX besteht aus den beiden Robotern Destroyer und Flex als Java Datei, einer Dokumentation im PDF-Format und dem Quellcode.

## 4. Funktionale Anforderungen

- ✓ Die Roboter müssen sich gegenseitig unterstützen
- ✓ Die Roboter sollen sich nicht gegenseitig schießen
- ✓ Der eine soll blau sein und der andere rot
- ✓ Die Roboter sollen die Kugeln der Gegner ausweichen können
- ✓ Die Roboter sollen beim Scannen eines Gegners direkt anfangen in seine Richtung zu schießen
- ✓ Die Roboter sollen sich ständig bewegen
- ✓ Die Roboter müssen die Gegner scannen und Ihnen hinterherfahren.
- ✓ Die Roboter müssen ständig schießen, auch während sie ihre Kanone bewegen.



- ✓ Wenn die Roboter einen Gegner mit einer Kugel treffen, soll die Radarbewegung eingeschränkt werden.

## 5. Nicht funktionale Anforderungen

- ✓ Wenn die Roboter gegen die Wand stoßen, sollen sie sich um 90 Grad drehen
- ✓ Die Roboter sollen mit der Schussenergie 1 schießen.
- ✓ Wenn der Abstand eines Gegners weniger als 50 Meter ist, soll der Roboter mit dem Power 2 schießen
- ✓ Wenn die Energie unseres Roboters weniger als 90 ist, soll er mit dem Power 3 schießen
- ✓ Wenn unsere Roboter sich bekämpft fühlen, sollen sie einen Schritt nach vorne gehen und sich um 90 Grad umdrehen.

## 6. Lieferumfang

Im Lieferumfang enthalten ist das Roboterteam bestehend aus den Robotern Destroyer und Flex in Form einer Java-Datei, die Dokumentation und der Quellcode.

## 7. Abnahmekriterien

Das Produkt gilt als abgenommen, wenn das Roboterteam bei der Vorführung am 14.07.2022 ohne sichtbare Mängel gegen die Roboterteams der anderen Teilnehmer spielt. Des Weiteren sollen die beiden Roboter einzeln oder im Team eine höhere Siegeschance als 50% gegen die Beispielroboter von Robocode (im speziellen gegen Tracker, Walls, Crazy) haben.

# **Pflichtenheft**

## **1. Zielbestimmung**

Ziel und Aufgabe des Projektes ist es, ein Team von genau 2 Robotern zu planen, zu implementieren und zu dokumentieren. Das Roboterteam soll mit einer gewinnenden Strategie sowohl gegen die von den anderen Kursteilnehmern programmierten Roboterteams als auch gegen die in der Robocode-Umgebung vorhandenen Beispielroboter und -teams in einer ausreichend großen Arena (meist 1000×1000) erfolgreich antreten können.

### **1.1. Muss-Kriterien**

Um die Teamstrategie und das Verhalten unserer Roboter umsetzen zu können sind folgende Funktionen zwingend notwendig.

(Teil Programmierung unten ansehen)

## **2. Produktdaten**

Robocode speichert seine Daten in einem vom Benutzer selbstgewählten Ordner im Computer

## **3. Qualitätsanforderungen**

- ✓ Wenn die Roboter gegen die Wand stoßen, sollen sie sich um 90 Grad drehen
- ✓ Die Roboter sollen mit der Schussenergie 1 schießen.
- ✓ Wenn der Abstand eines Gegners weniger als 50 Meter ist, soll der Roboter mit dem Power 2 schießen
- ✓ Wenn die Energie unseres Roboters weniger als 90 ist, soll er mit dem Power 3 schießen
- ✓ Wenn unsere Roboter sich bekämpft fühlen, sollen sie einen Schritt nach vorne gehen und sich um 90 Grad umdrehen.

## 4. Rechtliche Rahmenbedingungen, Sicherheitsanforderungen, unterstützte Plattformen

### 4.1. Lizenzen

Robocode wird ohne proprietäre Bestandteile basierend auf Open Source Komponenten entwickelt und unter den Bedingungen der EPL (Eclipse Public Licence) bereitgestellt. Der Empfänger der Software hat somit das Recht, die Software beliebig einzusetzen, zu analysieren, zu verändern, in unbegrenzter Anzahl zu kopieren, und das Original oder veränderte Versionen oder Kopien davon weiterzugeben.

### 4.2. Sicherheitsanforderungen

Sicherheit kommt mit der JVM (Java Virtual Machine). Die stellt zum Beispiel sicher, dass Klassen nur auf diejenigen Elemente einer anderen Klasse zugreifen dürfen, die dafür bestimmt sind.

### 4.3. Unterstützte Plattformen

Robocode basiert auf der Programmiersprache Java, deshalb kann es auf jedem Betriebssystem, welche die Java-Runtime-Environment ausführen kann, betrieben werden.

## 5. Technische Produktumgebung

### ○ Verwendete Programmiersprachen und Entwicklungsumgebungen

Die verwendete Programmiersprache ist hierbei JAVA und die Entwicklungsumgebung kann entweder Robocode selbst sein oder externe IDEs wie: IntelliJ IDEA, Eclipse, Netbeans oder Visual Studio Code.

# Entwurf

Beim Softwareentwurf geht es also darum, eine gemeinsame Basis für die Softwareentwicklung und wichtige technische Zusammenhänge zu identifizieren und festzulegen. Ziel ist es, **eine Grundlage zu definieren, die leicht zu verstehen, zu ändern, zu erweitern und zu warten ist.**

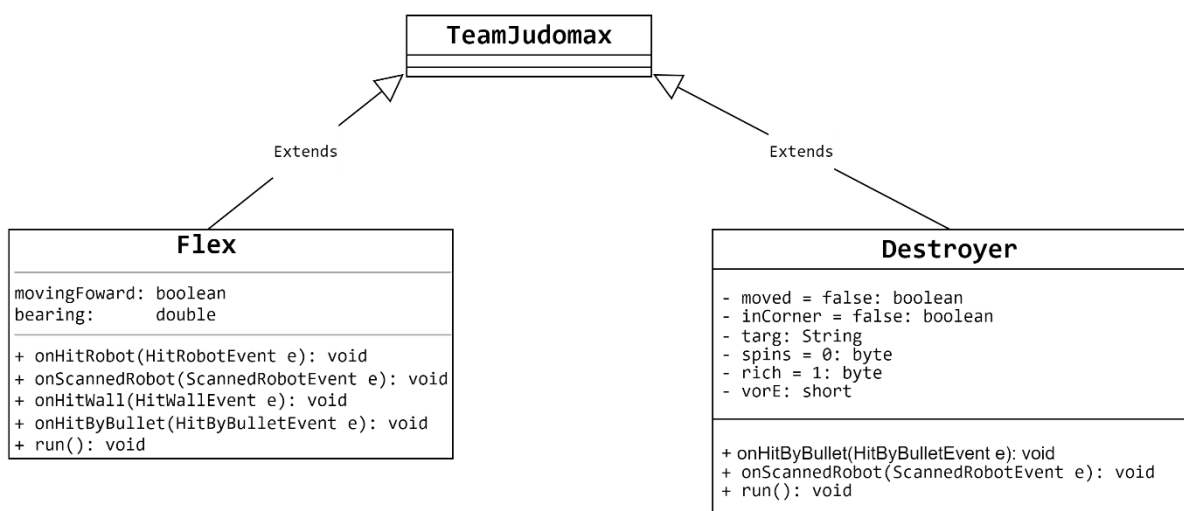
Für den Entwurf unser Projekt, wird das Methode **Objektorientiertes Design (OOD)** gebraucht werden, da die Software Robocode eine Objektorientiertes Programmierung ist. Die verwendete Sprache für diese Software ist Java.

Für das Objektorientiertes Design unser Robocode, haben wir als Tools das UML Diagramm nämlich das Klassendiagramm ausgewählt.

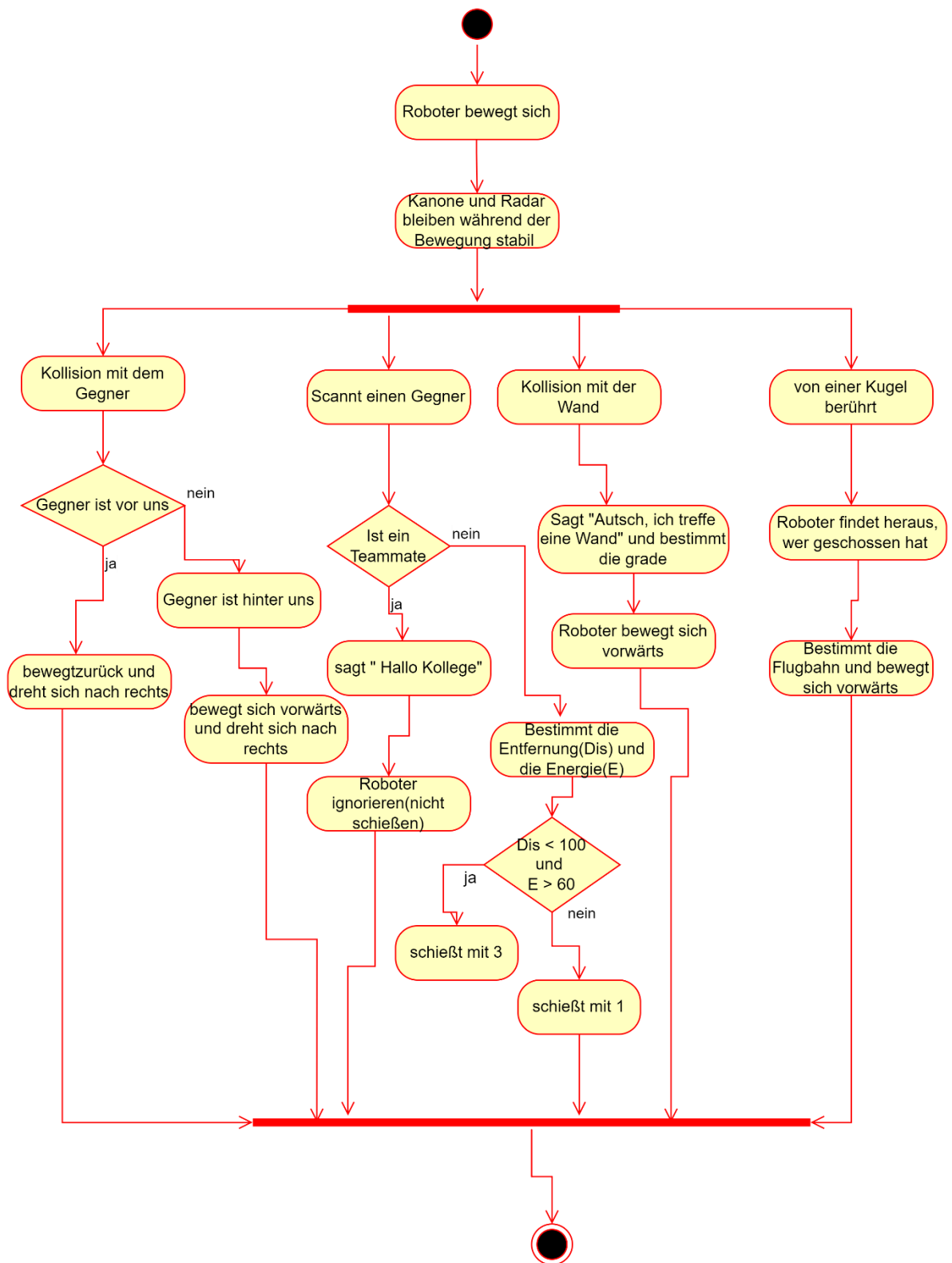
## Klassen für Robocode

- TeamJudomax
- Flex
- Destroyer

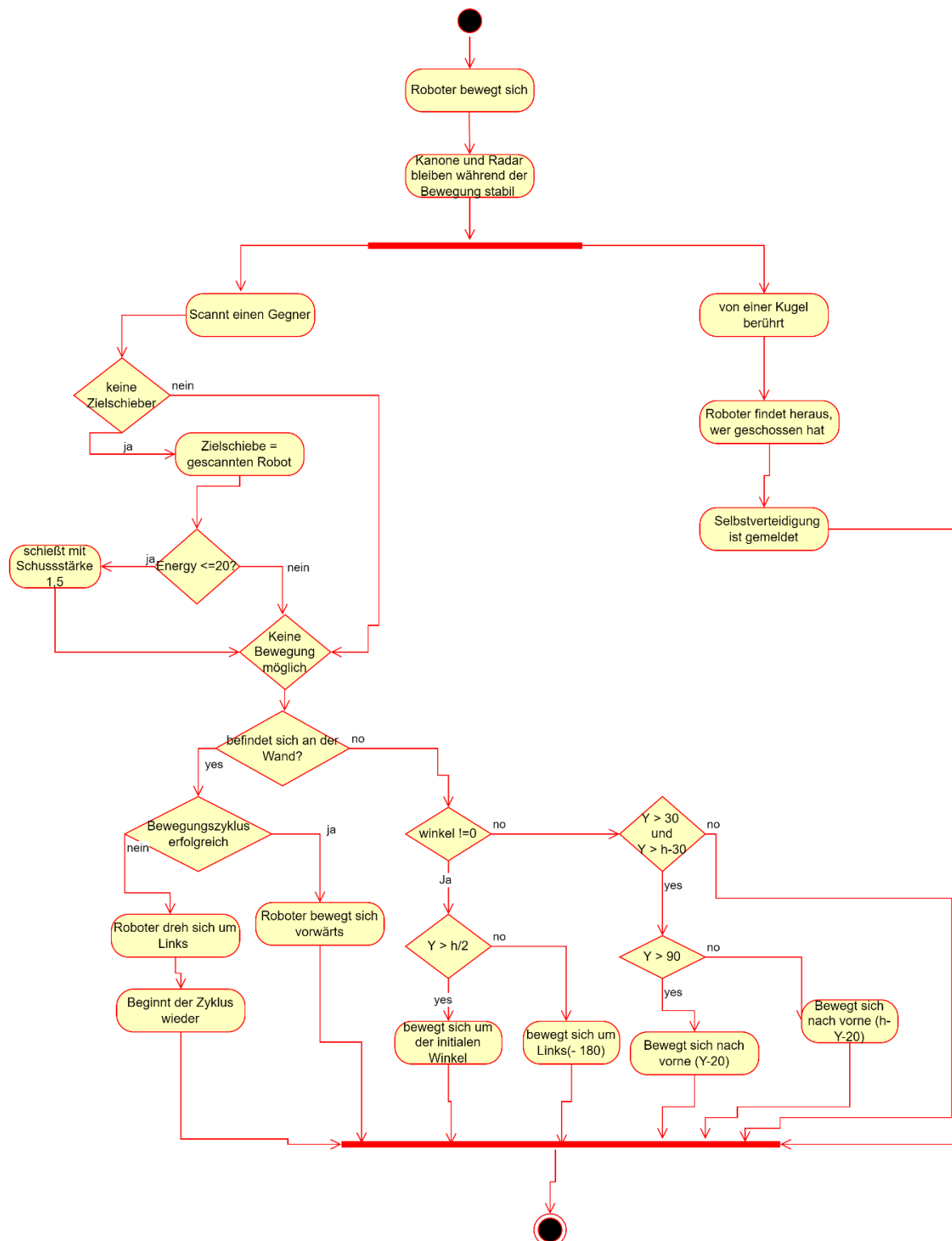
## UML(KLASSENDIAGRAMM)



# Aktivitätsdiagramm (Flex)



# Aktivitätsdiagramm (Destroyer)



## Entwurfsmuster

Wir haben in unseren Robotern kein Entwurfsmuster aus der Vorlesung verwendet.

Prinzipiell passend wäre aber das Entwurfsmuster „Beobachter“, da es Zustandsänderungen zu gewissen Zeitpunkten auswertet. Das wäre insbesondere für die Lebensenergie, Kollisionsabfrage und das Tracking von Gegner denkbar gewesen. Damit könnten die Objekte automatisch aktualisiert und Methoden durch Ereignisse aufgerufen werden.

Sinnvoll wäre außerdem das Entwurfsmuster „Einzelstück“ gewesen, um sicher zu stellen, dass Objekte nur einfach aufgerufen werden und Funktionen des Roboters dadurch nicht gegenläufig arbeiten. Allerdings war das in unserem Roboter nicht zwingend notwendig.

Das Entwurfsmuster „Dekorierer“ wäre nicht geeignet gewesen, da die Roboter statische Eigenschaften haben und nicht auf das dynamische Hinzufügen von Eigenschaften angewiesen sind.

Ein weiteres Entwurfsmuster wäre „Fabrik“, jedoch hier ebenfalls nicht passend, da wir nicht darauf angewiesen waren derart viele neue Objekte zu erstellen. Das Entwurfsmuster „Fliegengewicht“ hätte ebenfalls wenig Sinn gemacht, da es in unserem Roboter nicht vorkommt, dass so viele Instanzen eines Objektes erstellt werden müssen

# Programmierung

## Roboter Flex(Maxime)

Strategie des Roboters Flex:

Der Flex soll sich immer ständig in Forme einer Runde bewegen

Es werden hier verschiedene Package importiert:

```
import robocode.*;  
import java.awt.Color;  
import robocode.ScannedRobotEvent;  
import robocode.TeamRobot;  
import robocode.AdvancedRobot;
```

### Erklärung der Packages:

Hier benutzen wir alles aus dem Package Robocode

```
Import Robocode.*;
```

Dieses Package ist wichtig, damit wir unserem Roboter eine Farbe zuweisen können.

```
Import java.awt.Color.
```

Ein ScannedRobotEvent wird an onScannedRobot() gesendet, wenn unser Roboter einen Roboter scannt.

```
import robocode.ScannedRobotEvent;
```

Ein TeamRobot ist eine Erweiterung des AdvancedRobot und bietet Unterstützung beim Senden von Nachrichten zwischen Teammitgliedern.

```
import robocode.TeamRobot;
```

Hier erbt unseren Roboter alle Funktionen von der Oberklasse TeamRobot.

```
Public Class Flex Extends TeamRobot
```



Diese Methode wird von der Arena aufgerufen, um den Roboter zu starten; er läuft, bis das Match zu Ende ist.

`Public void run()`

Zu Beginn der Runde werden einmalig die Farben an den Flex zugewiesen.

```
setBodyColor(Color.green);  
setBulletColor(Color.green);  
setRadarColor(Color.white);  
setGunColor(Color.red);  
setScanColor(Color.yellow);
```

Für alle Ewigkeit wird sich der Flex folgendermaßen bewegen:

Der Flex wird sich immer mit der Geschwindigkeit 8 bewegen und sich um 10000 Grad nach Recht umdrehen, dann geht er 10000 Pixel vorne.

Also er bewegt sich in Form einer Runde.

```
// Für alle Ewigkeit  
while(true) {  
  
    // Robot dreht sich nach Rechts  
    setTurnRight(10000);  
  
    // Geschwindigkeit  
    setMaxVelocity(8);  
  
    // gehe nach vorne  
    ahead(10000);  
    // Wiederhole die Bewegung  
}  
}
```

Damit unsere beiden Roboter sich gegenseitig nicht vernichten, müssen sie sich erkennen können. Dies Erfolgt, indem wir zuerst die Roboter der Klasse TeamRobot unterordnet haben.

`isTeammate()` gibt einen boolean-Wert zurückgibt, der uns zeigt, ob der gescannte Robot auch zu unserem Team gehört. Die Methode wird dabei danach in `onScannedRobot()` aufgerufen und unser Robot soll das Teammitglied ignorieren und weiter nach Gegnern suchen.

Jedes Mal, wenn der Flex sein Mitspieler scannt, soll er ihn grüßen.

```

public void onScannedRobot(ScannedRobotEvent e) {
    out.println("Hallo Kollege");
    if(isTeammate(e.getName())) {
        return;
    }
}

```

Als nächstes haben wir die Methode onHitRobot () erstellt. Es wird hier erstmal überprüft, an welcher Stelle unser Gegner von uns sich befindet. Wenn der Gegner vor uns ist, wird uns als Nachricht bescheid gegeben, dass der Gegner vor uns ist, damit der Flex genau weißt, wie er dem Gegner ausweichen wird. Der Flex wird dann 100 Pixel nach hinten gehen und sich um 90 Grad nach Rechts umdrehen. Und wenn der Gegner hinter uns ist, wird der Flex 100 Pixel vorne gehen und sich um 90 Grad nach rechts umdrehen. Natürlich mit Nachricht, dass der Gegner hinter uns ist.

```

public void onHitRobot(HitRobotEvent e) {
    // wenn der Gegner vor uns ist.
    if (e.getBearing() > -90 && e.getBearing() < 90)
    {
        out.println("Der Gegner ist vor mir");
        setBack(100);
        turnRight(90);
    } // wenn er hinter uns ist.
    else {
        out.println("Der Gegner ist hinter mir");
        setAhead(100);
        turnRight(90);
    }
}
}

```

Danach haben wir die Methode onscanRobot() aufgebaut. Die Methode entscheidet, wie unser Roboter sich verhalten muss, wenn er einen Gegner scannt.

Erstmal wird die Methode überprüfen, ob der gescannte Roboter unser Team Kollege ist. Wenn es der Fall ist, wird der Flex einfach nicht auf sein Team Kollege schießen. Ansonsten schießt er auf den Gegner mit der Stärke 1. Darüberhinaus wird der Abstand gegenüber den Gegner gemessen.

Wenn der Abstand gegenüber dem Gegner weniger als 100 Pixel ist und unsere Energie größer als 60 ist, muss der Flex mit der Stärke 3 schießen. Ansonsten schießt er mit der Stärke 1.

```

public void onScannedRobot(ScannedRobotEvent e) {
    out.println("Hallo Kollege");
    if(isTeammate(e.getName())){
        return;
    }

    else if (e.getDistance()< 100 && getEnergy()>60 ) {
        fire(3);
    }
    else {
        fire(1);
    }
}

```

Danach erfolgt die Methode onHitWall(), welche angibt, was unser Robot tut, wenn er gegen den Rand des Spielfelds stößt. In dem Fall wird der Flex sich erstmal die Flugbahn merken und wird sich jeweils um (90 Grad – die Flugbahn) nach Rechts umdrehen, jenachdem in welchem Rand er sich gestoßen hat, und danach geht er 100 Pixel nach vorne.

```

public void onHitWall(HitWallEvent e) {
    out.println("Autsch, ich treffe ein Wandlager " + e.getBearing() + " Grad.");
    double bearing = e.getBearing(); //Lager Bearing merken
    setTurnRight(-bearing); // möglich roboter frei von Wand freigeben
    setAhead(100); // Der Roboter verschwindet
}

```

Am Ende haben wir die Methode onHitByBullet(). Diese gibt das Verhalten des Robots, wenn ein Gegner auf ihn schießt. Der Flex wird erstmal die Richtung merken, woher der Kugel kommt und dann versucht er nach Rechts zu flüchten

```

public void onHitByBullet(HitByBulletEvent e) {

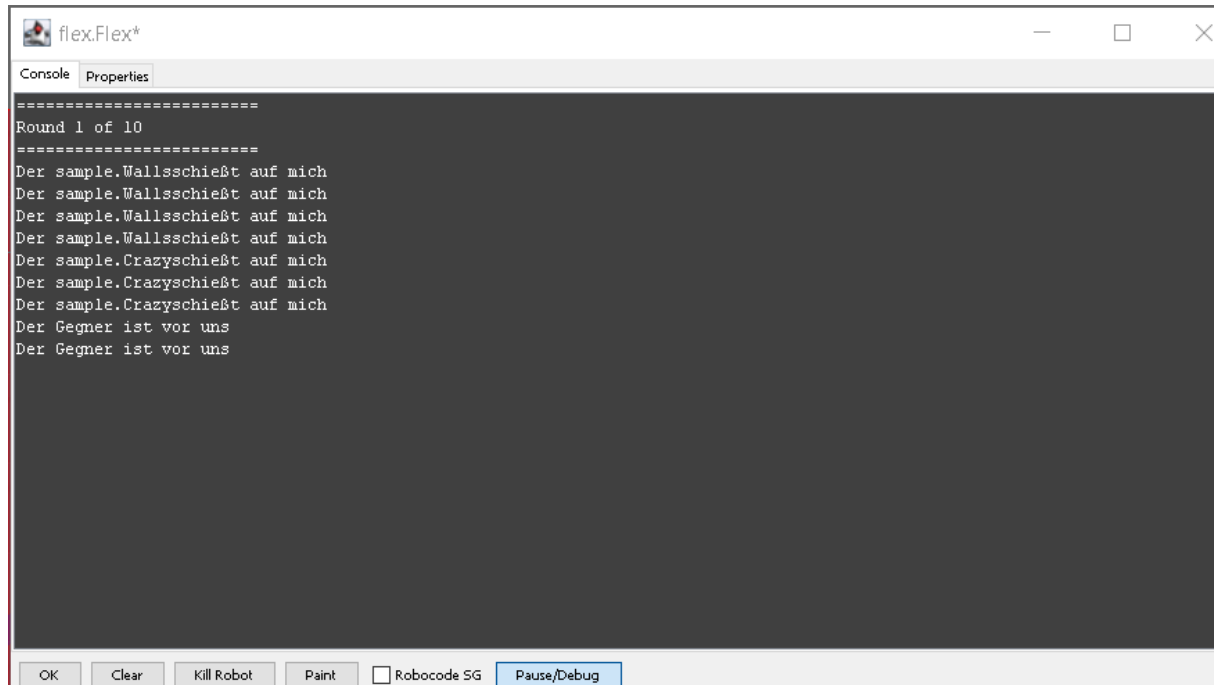
    out.println("Der " +e.getName()+" "+"schießt auf mich");
    double bearing = e.getBearing(); //merke die Richtung
    setTurnRight(-bearing); //versuch nach Rechts zu flüchten.
    setAhead(100); //gehe 100 Pixel vorne

}

```

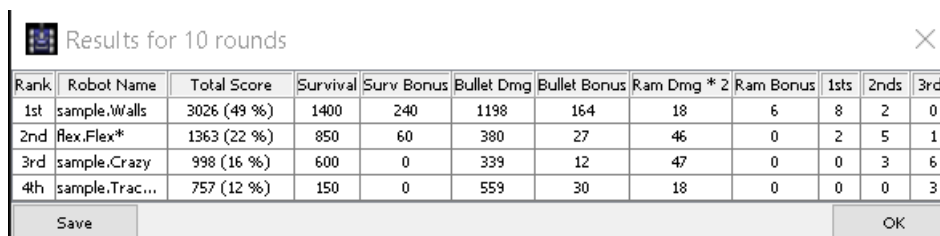
## Tests:

Das sind die Ausgaben auf der Konsole (gelten als Bestätigung, dass unsere Methoden reibungslos funktionieren)



```
=====
Round 1 of 10
=====
Der sample.Wallsschießt auf mich
Der sample.Wallsschießt auf mich
Der sample.Wallsschießt auf mich
Der sample.Wallsschießt auf mich
Der sample.Crazyschießt auf mich
Der sample.Crazyschießt auf mich
Der sample.Crazyschießt auf mich
Der Gegner ist vor uns
Der Gegner ist vor uns
```

Hier einmal die Ergebnisse von Flex nach 10 Runden gegen Walls, Crazy und Tracker



Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	sample.Walls	3026 (49 %)	1400	240	1198	164	18	6	8	2	0
2nd	flex.Flex*	1363 (22 %)	850	60	380	27	46	0	2	5	1
3rd	sample.Crazy	998 (16 %)	600	0	339	12	47	0	0	3	6
4th	sample.Trac...	757 (12 %)	150	0	559	30	18	0	0	0	3

## Roboter Destroyer (Dominique)

Hier einmal der Code von Destroyer erklärt.

Es werden hierbei die gleichen Packages und Methoden wie die vom Robot Flex verwendet, deswegen werden wir keine Redundanz bei der Erklärung machen.

```
package DestroyerJUDOMAX;
```

```
import robocode.*;
```

```
import java.awt.Color;
```

```
import robocode.AdvancedRobot;
```

```
import robocode.HitByBulletEvent;
```

```
import robocode.ScannedRobotEvent;
```

```
import robocode.util.Utils;
```

```
/*
```

```
 *Bewegung:
```

```
 Dieser Robot geht bis zur nächsten Ecke und bewegt sich dann in eine Kiste.
```

```
 Es besteht eine Wahrscheinlichkeit von 15 %,
```

```
 dass es die Richtung ändert, wenn der Feind feuert.
```

```
 * Ausrichtung:
```

```
 Zuerst zielt dieser Robot auf den ersten Roboter, den er sieht,
```

```
 aber er wechselt zu jedem Robot, der ihn schießt.
```

```
 */
```

```
public class DestroyerJUDOMAX extends TeamRobot{
```

```
    private boolean moved = false;           // wenn sich der Robot bewegen oder wenden muss
```

```
    private boolean inCorner = false;        // wenn er sich in der Ecke befindet
```

```
    private String targ;                    // Robot zu zielen (Zielscheibe)
```

```
    private byte spins = 0;                 // Anzahl der Runden
```

```
    private byte rich = 1;                  // Richtung der Bewegung
```

```
    private short vorE;                     // vorherige Energie des Roboters, auf den er zielt
```

```

public void run(){
    setBodyColor(Color.green);    //Das setzt die Farbe des Koepers vom Robot auf grün
    setGunColor(Color.red);       //Das setzt die Farbe des Gewehrlaufes auf Rot
    setRadarColor(Color.yellow);  //Das setzt die Farbe des Radars auf Gelbe
    setScanColor(Color.red);      //Das setzt die Farbe des Scanners auf Rot
    setBulletColor(Color.orange); //Das setzt die Farbe der Kugel auf Orange

    setAdjustGunForRobotTurn(true); //Wenn sich der Roboter dreht, passt sich
                                    //die Waffe der entgegengesetzten Richtung an
    setAdjustRadarForGunTurn(true); //Wenn sich die Waffe dreht, stellt sich das Radar
                                    // in die entgegengesetzte Richtung ein

    while(true){                  // für Radarsperre
        turnRadarLeftRadians(1);  // Dreh das Radar kontinuierlich nach links
    }
}

public void onHitByBullet(HitByBulletEvent e){    // Wenn der Robot von einer Kugel getroffen
wird,
    targ = e.getName();                          // zielt er auf den, der ihn gezielt hat!
    System.out.println(targ + " zielt auf mich, Selbstverteidigung aktiviert");
}

public void onScannedRobot(ScannedRobotEvent e){    // Beim Scannen eines Gegners
    if(targ == null || spins > 6){                // wenn es keine Zielscheibe gibt
        targ = e.getName();                      // schieß auf den ersten gescannten Robot
    }
    if(e.getEnergy() <= 20){                      // Wenn die Robot Energie < = 20, schieß mit 1.5 Schusstärke
        setFire(1.5);
    }
}

```

```
}
```

```
if(getDistanceRemaining() == 0 && getTurnRemaining() == 0){    //wenn keine Bewegung oder
Umdrehung möglich ist

    if(inCorner){                                              //wenn er sich in der Ecke befindet

        if(moved){                                            // wenn er im letzten Bewegungszyklus in Bewegung war,

            setTurnLeft(90);                                  // dreh diesen Zyklus um

            moved = false;                                     // und beginn den nächsten Zyklus

        }

    }

    else{                                                      // sonst, wenn er im letzten Zyklus umdreht

        setAhead(160 * rich);                                // Diesen Zyklus verschieben

        moved = true;                                         // und dreh den nächsten Zyklus um

    }

}

else{

    // wenn er nicht nach N/S geht, dann soll er nach Norden oder Süden gehen

    if((getHeading() % 90) != 0){

        setTurnLeft((getY() > (getBattleFieldHeight() / 2)) ? getHeading()

        : getHeading() - 180);

    }

    // Wenn er nicht oben oder unten ist, geht er zu dem, was näher ist

    else if(getY() > 30 && getY() < getBattleFieldHeight() - 30){

        setAhead(getHeading() > 90 ? getY() - 20 : getBattleFieldHeight() - getY()

        - 20);

    }

    // wenn er nicht nach Osten/Westen schaut, schaut er ihm zu

    else if(getHeading() != 90 && getHeading() != 270){

        if(getX() < 350){

            setTurnLeft(getY() > 300 ? 90 : -90);

        }

    }

    else{

        setTurnLeft(getY() > 300? -90 : 90);

    }

}
```

```

    }
}

// Wenn er nicht links oder rechts ist, geht er zu dem, was näher ist
else if(getX() > 30 && getX() < getBattleFieldWidth() - 30){
    setAhead(getHeading() < 180 ? getX() - 20 : getBattleFieldWidth() - getX()
        - 20);
}

// der Robot ist vor der linken Wand; drehen und losfahren
else if(getHeading() == 270){
    setTurnLeft(getY() > 200 ? 90 : 180);
    inCorner = true;
}

// der Robot ist vor der rechten Wand; drehen und losfahren
else if(getHeading() == 90){
    setTurnLeft(getY() > 200 ? 180 : 90);
    inCorner = true;
}
}

if(e.getName().equals(targ)){                // wenn der gescannte Robot unsere Zielscheibe ist
    spins = 0;                                // Radar Rundenzähler( spins ) zurücksetzen

    // wenn der Feind mit einer Chance von 15 % schießt,
    if((vorE < (vorE = (short)e.getEnergy())) && Math.random() > .85){
        rich *= -1;                            // die Richtung ändern
    }

    setTurnGunRightRadians(Utils.normalRelativeAngle((getHeadingRadians() + e
        .getBearingRadians()) - getGunHeadingRadians()));    // Kanone in die Richtung des
Gegners bewegen

    if(e.getDistance() < 200){                //wenn der Feind mehr als 200px entfernt ist

```



```

        setFire(3); //schieß mit voller Kraft (Schussstärke 3)
    }
    else{
        setFire(2.4); // sonst schieß mit Schussstärke 2.4
    }

    double radarTurn = getHeadingRadians() + e.getBearingRadians()
        - getRadarHeadingRadians();

    setTurnRadarRightRadians(2 * Utils.normalRelativeAngle(radarTurn));
    // Radar sperren, weil der Roboter in dem Moment auf einen Gegner schießt und soll die
    // Radarbewegung beschränken

        System.out.println("Radar sperren");

    }

    else if(targ != null){ // sonst wenn eine Zielscheibe getroffen wurde

        spins++; // erhöhe die Anzahl der Runden um 1

        System.out.println(spins);

    }

}

}

```

des.Destroyer\*


Console Properties

```

1
Radar sperren
1
Radar sperren
1
Radar sperren
1
Radar sperren
1
Radar sperren
1
Radar sperren
sample.Tracker zielt auf mich, Selbstverteidigung aktiviert
Radar sperren
1
Radar sperren
1
Radar sperren
1
Radar sperren
1
Radar sperren
1
Radar sperren
1
Radar sperren

```

OK Clear Kill Robot Paint ☐ Robocode SG Pause/Debug


Results for 10 rounds
✕

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	des.Destroye...	3541 (51 %)	1100	120	1933	233	106	49	4	4	2
2nd	sample.Walls	1690 (25 %)	950	150	504	46	24	15	5	1	2
3rd	sample.Crazy	875 (13 %)	650	30	168	3	24	0	1	3	4
4th	sample.Trac...	789 (11 %)	300	0	458	12	19	0	0	2	2

Save

OK