

Dokumentation

Roboter-Fangen

Maschinenbauinformatik 3. & 5. Semester

Michael Mertens, Jonah Vennemann, Sven Stegemann, Eugen Zwetzig

11. Februar 2016

Inhaltsverzeichnis

1	Vorgaben	3
1.1	Projektbeschreibung	3
1.2	Spielablauf	4
2	Zeitablauf	5
2.1	Gantt-Chart	5
2.2	Aufwandsschätzung	7
3	GitHub	9
3.1	ZenHub	10
4	Systemaufbau	12
5	Programmierung	13
5.1	Programmablaufplan	13
5.2	Hauptformular	14
5.3	Klassen	15
5.3.1	mVektor	15
5.3.2	mTKI	16
5.3.3	mKonstanten	20
5.3.4	mRoboterDaten	20
6	Fazit	21

Abbildungsverzeichnis

1	Gantt-Diagramm	6
---	--------------------------	---

Quellcodeverzeichnis

1	Klasse mTKI	22
---	-----------------------	----

1 Vorgaben

1.1 Projektbeschreibung

Bei dem Projekt „Roboter-Fangen“ für das Modul IT-Projektmanagement besteht unsere Aufgabe als eines von zwei Teams in der Programmierung einer Steuerungssoftware für das Fischertechnik ROBOTICS TXT Discovery Set.

Das Gemeinziel ist ein lauffähiges Fangen-Spiel zu erstellen bei dem vier Roboter pro Team von der jeweiligen Software gesteuert werden.

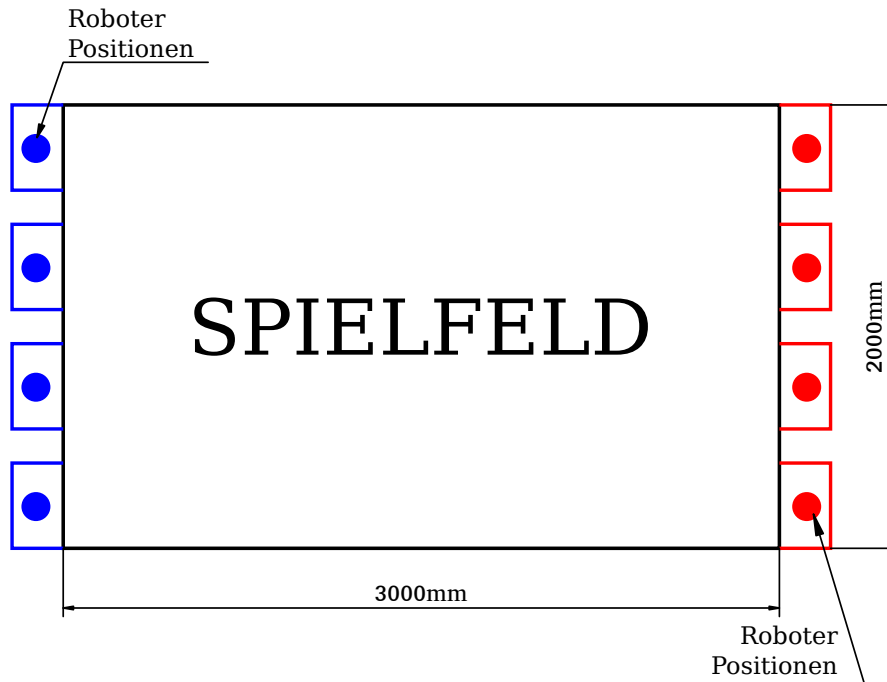
Dabei werden die Positionsdaten aller Roboter von einem Schiedsrichter-Server mit Hilfe einer Kamera berechnet und an die Steuerungssoftware der beiden Teams geschickt. Hauptbestandteile der Steuerungssoftware:

- Benutzeroberfläche:
 - Kamerabilder
 - Eingabefelder zum Verbinden
 - zusätzliche Informationen
- Positionsdatenverarbeitung über eine Vektorklasse:
 - Attribute: x,y als Typ Double
 - Methoden: Addieren, Subtrahieren, Skalar multiplizieren, Winkel berechnen
- Elemente der KI:
 - Fangen
 - Fliehen
 - Ausweichen
 - im Feld bleiben
 - Rausfahren nachdem Gefangenwerden

Neben der Programmierung gehören dabei auch die Planung, die Dokumentation des Codes sowie die Darstellung des Projekts dazu.

- Quelltextkommentare
- Präsentation
- Zeiterfassung
- Betriebsanleitung
- Spielregeln

1.2 Spielablauf



Es werden pro Gruppe 4 Roboter auf dem Spielfeld an ihre Startpositionen platziert. Die Größe des Spielfeldes ist festgelegt.

Alle Roboter, von beiden Teams, verbinden sich mit dem Server und übergeben diesem, dass Sie bereit sind. Sobald der Server das Startsignal gibt, fahren alle Roboter los.

Dann versuchen sich die Roboter, der beiden Teams, gegenseitig zu fangen. Dazu muss einer der beiden Taster, welche am hinteren Ende des Roboters angebracht sind, betätigt werden.

Wenn ein Roboter gefangen wurde, sendet dieser ein Signal an den Server und wird von diesem auf nicht Aktiv gesetzt. Ist ein Roboter nicht Aktiv, so fährt dieser aus dem Spielfeld und verbleibt dort eine gewisse Zeit, bis er in's Spiel zurück kehrt.

Folgende Regeln wurden getroffen:

- Ein Roboter darf erst losfahren, wenn der Server das Startsignal gegeben hat
- Ein Roboter darf sich nicht auf der Stelle drehen
- Ein Roboter darf nicht mit dem Rücken zur Wand stehen, da es so nicht möglich ist diesen zu fangen

2 Zeitablauf

2.1 Gantt-Chart

Ein Gantt-Diagramm oder auch Balkenplan ist ein nach dem Unternehmensberater Henry L. Gantt benanntes Instrument des Projektmanagements, das die zeitliche Abfolge von Aktivitäten grafisch in Form von Balken auf einer Zeitachse darstellt.

In der Abbildung 1 sieht man die einzelnen Aktivitäten, die wir für unser Projekt Roboter-Fangen eingeplant haben. Außerdem sieht

Einige Aktivitäten haben wir in Gruppen eingeteilt, um:

- Planung
- Programmierung - Teil 1
- Programmierung - Teil 2
- Dokumentation

		Name	Dauer	Start	Ende	Vorgänger	Ressourcen
1	✓	M0: Start	0 tage	23.12.15 08:00	23.12.15 08:00		
2	✓	Analyse	0,5 tage	23.12.15 08:00	23.12.15 13:00	1	Gruppe Blau;Gruppe ...
3	✓	Aufgabenaufteilung	0,5 tage	23.12.15 12:00	23.12.15 17:00	2	Gruppe Blau
4	✓	Planung	0,5 tage	28.12.15 08:00	28.12.15 13:00	3	
5	✓	Projektbeschreibung	0,5 tage	28.12.15 08:00	28.12.15 13:00		Eugen Zwetzig;Com...
6	✓	Spielregeln	0,5 tage	28.12.15 08:00	28.12.15 13:00	5AA	Jonah Vennemann;Eu...
7	✓	Skizze PAP KI	0,5 tage	28.12.15 08:00	28.12.15 13:00	6AA	Michael Mertens
8	📅	M1: Planung abgeschlossen	0 tage	04.01.16 08:00	04.01.16 08:00	4	
9		Programmierung - Teil I	9,5 tage	04.01.16 08:00	15.01.16 13:00	8	
10		GUI-Design	1 tag	04.01.16 08:00	04.01.16 17:00		Computer;Jonah Venn...
11		Programmstart/verbinden	1,5 tage	05.01.16 08:00	06.01.16 13:00	10	
12		Positionsdaten empfang...	1 tag	06.01.16 13:00	07.01.16 13:00	11	Sven Stegemann
13		Fahrtrichtungen ermitteln	2 tage	07.01.16 13:00	11.01.16 13:00	12	
14		Als "gefangen" melden	1 tag	04.01.16 08:00	04.01.16 17:00	10AA	
15		Simple KI	4 tage	11.01.16 13:00	15.01.16 13:00	13	
16		KI - Fliehen	2 tage	11.01.16 13:00	13.01.16 13:00		Jonah Vennemann
17		KI - Ausweichen	2 tage	11.01.16 13:00	13.01.16 13:00	16AA	Eugen Zwetzig
18		KI - Im Feld bleiben	2 tage	11.01.16 13:00	13.01.16 13:00	17AA	Jonah Vennemann
19		KI - Fangen	2 tage	11.01.16 13:00	13.01.16 13:00	18AA	Eugen Zwetzig
20		KI - Rausfahren nachde...	2 tage	13.01.16 13:00	15.01.16 13:00	19	Jonah Vennemann
21	📅	M2: Erste Implementierung	0 tage	15.01.16 13:00	15.01.16 13:00	9	
22		Vorabpräsentation erstell...	1 tag	15.01.16 13:00	18.01.16 13:00	21	
23	📅	M3: Kurzpräsentation	0 tage	18.01.16 13:00	18.01.16 13:00	22	
24		Programmierung - Teil II	3 tage	18.01.16 12:00	21.01.16 13:00	23	
25	✓	Log-Funktion	0,5 tage	18.01.16 12:00	18.01.16 17:00		Jonah Vennemann
26		Kamerabilder anzeigen	1 tag	18.01.16 13:00	19.01.16 13:00	27AA	Michael Mertens
27	✓	Klasse zur Vektorberech...	1 tag	18.01.16 12:00	19.01.16 13:00	25AA	Eugen Zwetzig
28	📅	Tests	2 tage	19.01.16 13:00	21.01.16 13:00	27	Computer;Gruppe Bla...
29	📅	M4: Programmieren abge...	0 tage	21.01.16 13:00	21.01.16 13:00	24	
30		Präsentation abschließen	1,5 tage	21.01.16 13:00	22.01.16 17:00	29	Jonah Vennemann
31	📅	M5: Endpräsentation	0 tage	22.01.16 17:00	22.01.16 17:00	30	
32	📅	Dokumentation	30 tage	05.01.16 08:00	15.02.16 17:00	9AA	
33	📅	Betriebsanleitung	30 tage	05.01.16 08:00	15.02.16 17:00		Eugen Zwetzig;Com...
34	📅	M6: Dokumentation abge...	0 tage	15.02.16 17:00	15.02.16 17:00	32	

Roboter fangen Roboter

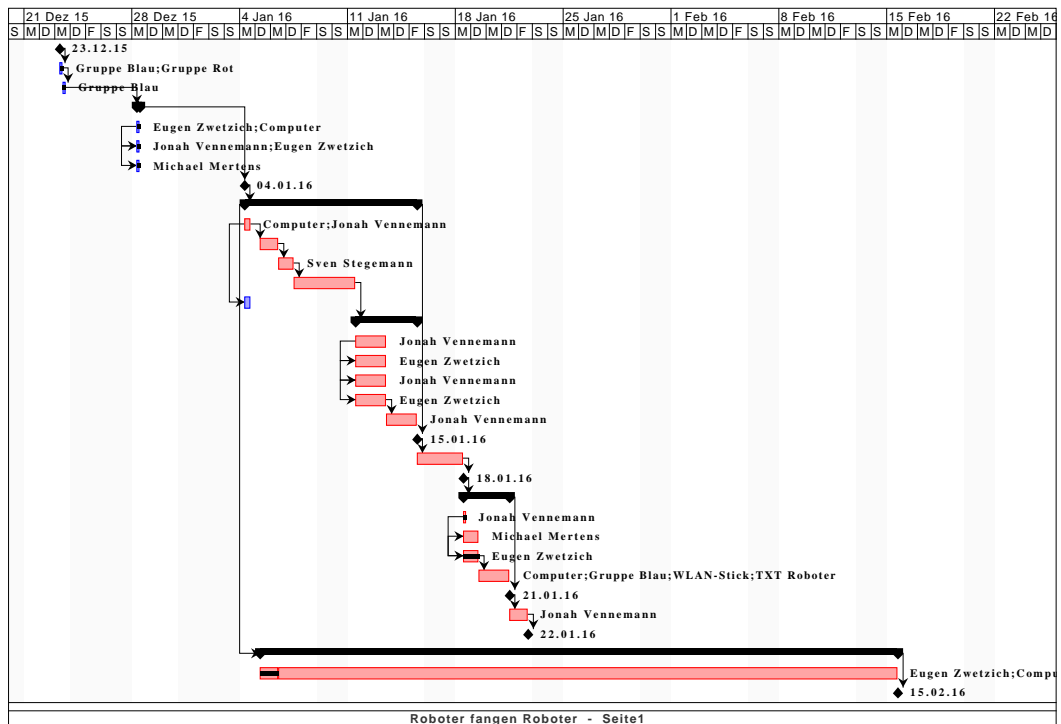


Abbildung 1: Gantt-Diagramm

2.2 Aufwandsschätzung

Um den Aufwand unseres IT-Projektes abschätzen zu können, haben wir die Methode Function-Point benutzt.

Das Function-Point-Verfahren(auch -Analyse oder -Methode, kurz: FPA) dient der Bewertung des fachlich-funktionalen Umfangs eines Informationstechnischen Systems.

Die Durchführung des Verfahrens verläuft in 5 Schritten:

1. Analyse der Komponenten und Kategorisierung ihrer Funktionalitäten
2. Bewertung der verschiedenen Funktionskategorien
3. Einbeziehung besonderer Einflussfaktoren
4. Ermittlung der sog. Total Function Points(TFP)
5. Ableitung des zu erwartenden Entwicklungsaufwandes

1. Schritt

- Eingabedaten
 - GUI
 - Programmstart
- Ausgabedaten
 - Ereignisprotokolldatei
 - Kamerabild
 - Steuerbefehle senden
- projektbez. Datenbestände
 - Fahrtrichtung
 - Fangen
 - Fliehen
 - Ausweichen
 - Im Feld bleiben
 - Rausfahren nach dem Fangen
 - Vektorberechnung
- externe Datenbestände
 - Positionsdaten
 - Mitteilung gefangen
 - Roboter aktiv?

2. Schritt

Funktionskategorie	Anzahl der Funktionen			Faktoren der Funktionen			Funktionspunkte		
	Einfach	Mittel	Komplex	Einfach	Mittel	Komplex	Einfach	Mittel	Komplex
Eingabedaten	1	1	0	3	4	6	3	4	0
Ausgabedaten	1	2	0	4	5	7	4	10	0
Projektbez. Datenbestände	1	3	3	7	10	15	7	30	45
Externe Datenbestände	3	0	0	5	7	10	15	0	0

Summe S1:	118
------------------	-----

3. Schritt

Nr	Einflussfaktoren	Gewichte
1	Schwierigkeit und Komplexität der Rechenoperatoren (Faktor 2)	2
2	Schwierigkeit und Komplexität der Ablauflogik	5
3	Umfang der Ausnahmeregelung (Faktor 2)	6
4	Verflechtungen mit anderen IT-Systemen	3
5	dezentrale Verarbeitung und Datenhaltung	0
6	erforderliche Maßnahmen der IT Sicherheit	0
7	angestrebte Rechengeschwindigkeit	1
8	Konvertierung der Datenbeständen	0
9	Benutzer- und Änderungsfreundlichkeit	1
10	Wiederverwendbarkeit von Komponenten (bspw. Klassen)	1

Summe S2:	19
------------------	----

4. Schritt

$$\begin{aligned} \text{TFP}^1 &= S1 \cdot S3 \\ &= S1 \cdot \left(0,7 + \frac{S2}{100}\right) \\ &= 118 \cdot \left(0,7 + \frac{19}{100}\right) \\ \text{TFP} &= 105,02 \end{aligned}$$

5. Schritt

$$\text{PM}^2 = 0,08 \cdot \text{TFP} - 7 \leq 1000 \text{TFP} > \text{PM} = 0,08 \cdot \text{TFP} - 108$$

$$\begin{aligned} \text{PM} &= 0,08 \cdot \text{TFP} - 7 \\ &= 0,08 \cdot 105,02 - 7 \\ \text{PM} &= 1,4016 \end{aligned}$$

$$\text{PM} = 672,77\text{h}$$

$$3 \text{ Personen} = 224,256\text{h pro Person}$$

$$4 \text{ Personen} = 168,192\text{h pro Person}$$

⇒ Bei einem 4 Mann starken Team benötigen wir ca. 170h pro Person.

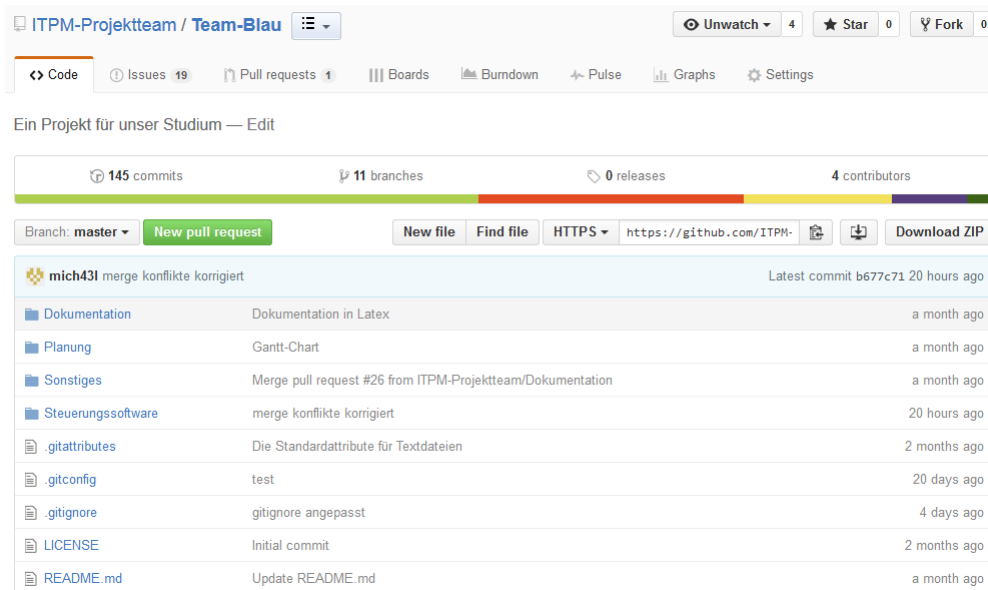
3 GitHub

GitHub ist ein webbasierter Online-Dienst, für die Versionsverwaltungssoftware Git.

In GitHub haben wir unser Projekt

¹TFP=Total Function Points

²Personenmonate(PM) = 20 Arbeitstage



3.1 ZenHub

ZenHub ist eine Projektmanagement-Erweiterung für GitHub.

ZenHub haben wir dafür benutzt, um die Programmierung des Projektes in Hauptkategorien auf zu teilen. Diesen Kategorien haben wir Mitarbeiter zu geteilt, sowie die eventuell benötigte Dauer, für die Erstellung der Punkte vergeben.

ZenHub unterteilt sich in:

- New Issues(Neue Hauptpunkte)
- Ice-Box()
- Backlog()
- To Do(Muss gemacht werden)
- In Progress(In Bearbeitung)
- Done/Review(Fertig/Überprüfung)

0

New Issues

(1)

Team-Blau #62

Fixes

2

Ice-Box

(1)

Team-Blau #15

Sounds?

question

5

Backlog

(2)

Team-Blau #10

Kamerabilder anzeigen

Team-Blau #13

Fahrtbuchungen ermitteln

Search (/)

New issue

14

To Do

(4)

Team-Blau #9

Programmstart / Verbinden

Team-Blau #11

Positionsdaten empfangen und speichern

Team-Blau #12

Simple KI

Team-Blau #16

Betriebsanleitung

16

In Progress

(5)

Team-Blau #7

Gantt-Chart

Team-Blau #8

Spielregeln niederschreiben

Team-Blau #23

KI - Rausfahren nach dem Fangen

Team-Blau #24

Als "gefangen" melden

Team-Blau #21

KI - Ausweichen

23

Done / Review

(7)

Team-Blau #6

Skizze für die KI

Team-Blau #17

Projektbeschreibung

Team-Blau #34

KI-Grundgerüst

Team-Blau #38

Priorität festlegen

Team-Blau #19

KI - Fangen

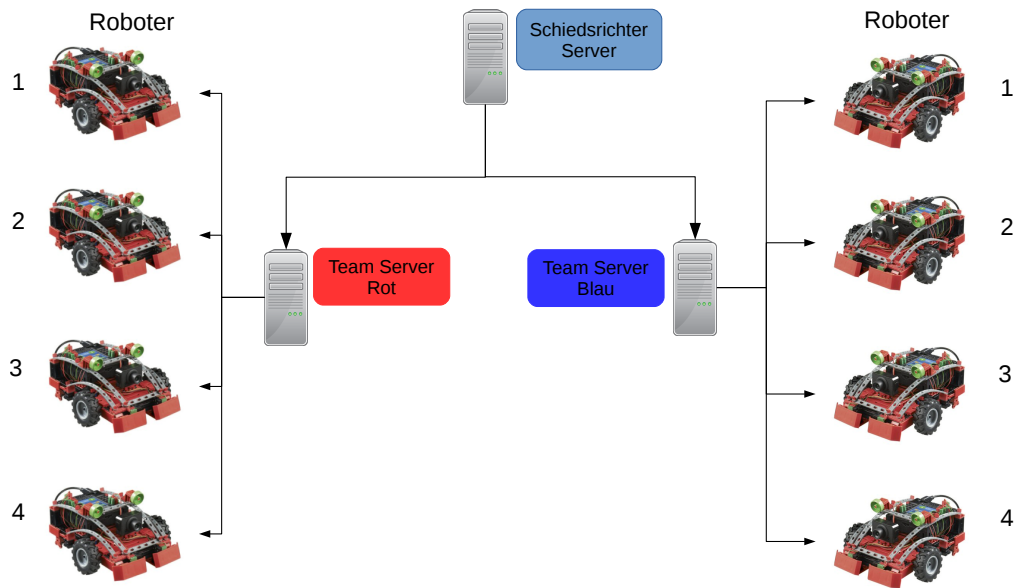
Team-Blau #20

KI - Fliehen

Team-Blau #22

KI - Im Feld bleiben

4 Systemaufbau



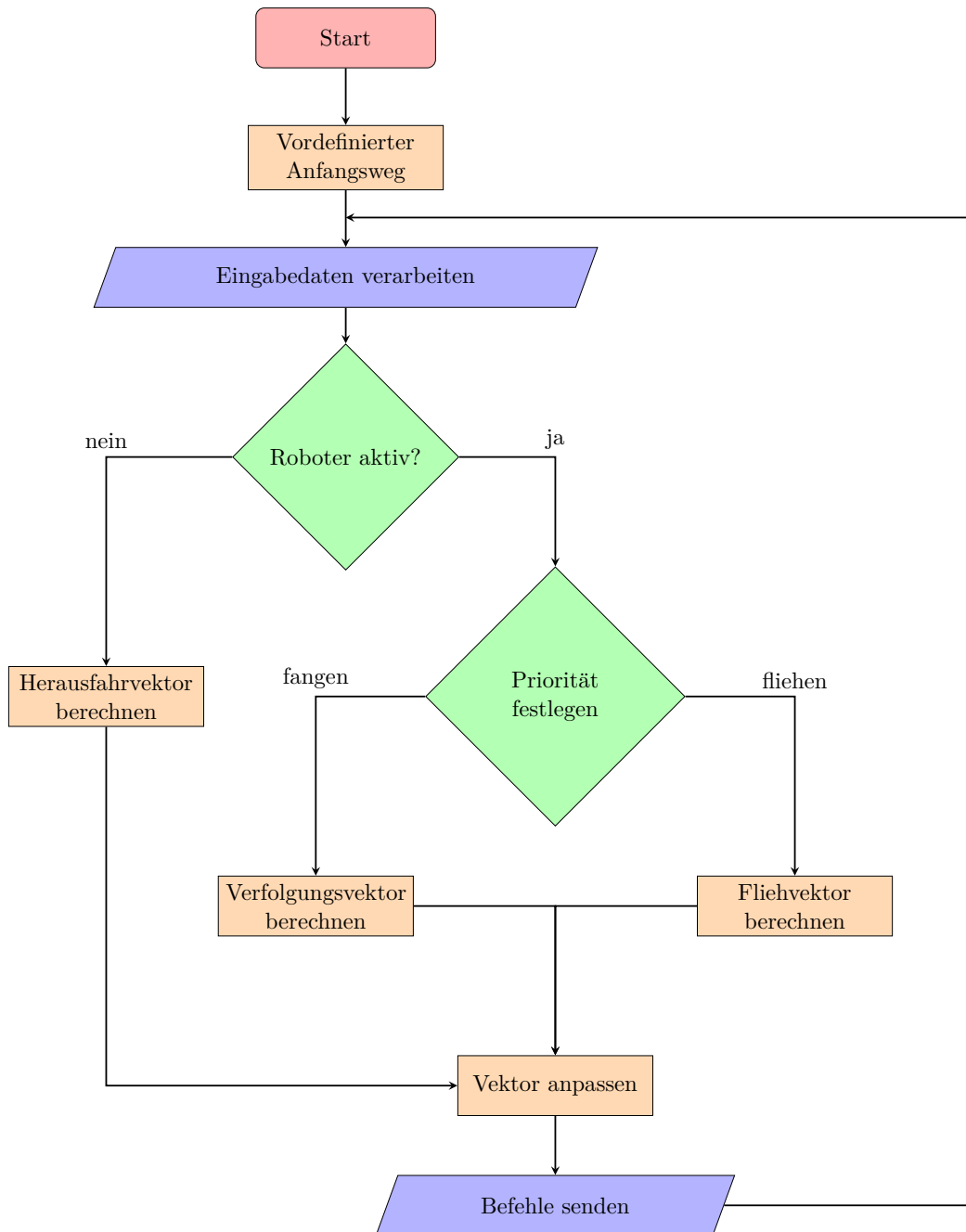
Der Systemaufbau des Projekts stellt sich wie folgt dar:

Ein zentraler Schiedsrichter-Server überwacht das Spielfeld der Roboter mithilfe einer Kamera. Anhand der Kamerabilder werden Positionsdaten der Roboter bestimmt. Des weiteren überwacht der Server den Status der Roboter, d.h. es wird überprüft ob diese gefangen wurden.

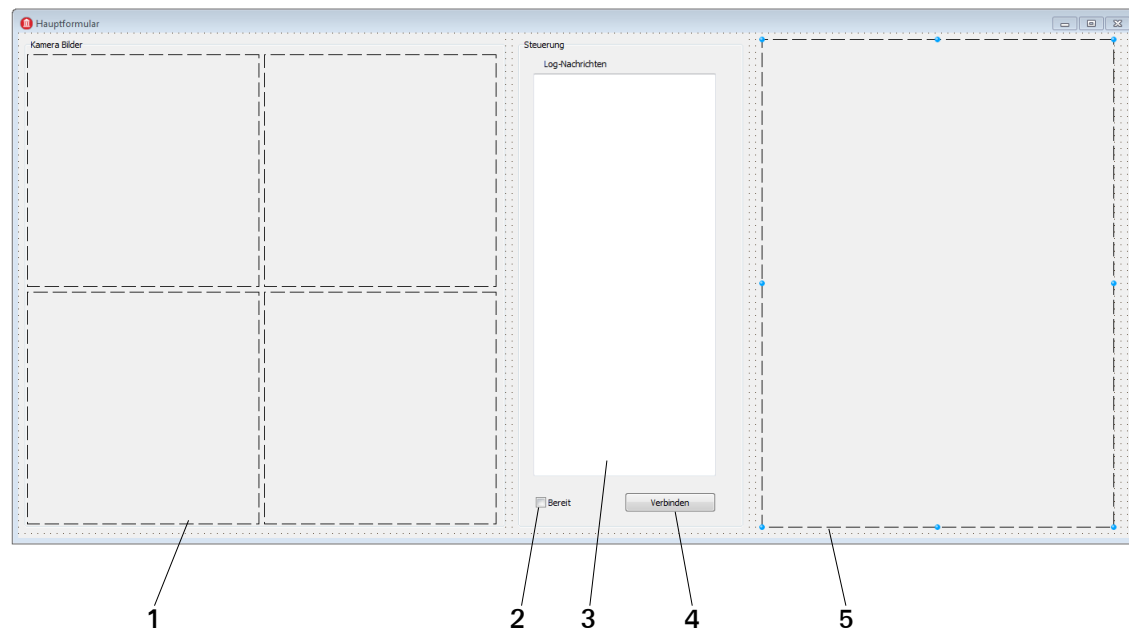
Die Positionsdaten werden mittels einer TCP/IP-Verbindung an die jeweiligen Team-Server gesendet. Diese werten die Daten aus und ermitteln die Fahrtrichtungen für jeden Roboter eines Teams. Die Befehle werden wieder mithilfe einer TCP/IP-Verbindung an die Roboter übermittelt.

5 Programmierung

5.1 Programmablaufplan



5.2 Hauptformular



1. Anzeige der Bilder, von den USB-Kameras der Roboter
2. Kontrollkästchen, das der Roboter bereit ist
3. Anzeige von Fehler-, Hinweis- und Warnmeldungen
4. Schaltknopf zum Verbinden der Roboter mit dem Server
5. Grafische Anzeige der Fahrtrichtungen von den einzelnen Robotern

Prozeduren:

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Ereignisprotokolldatei	1.1.1	Sven Stegemann	Eugen Zwetlich
Ergebniss: -			
Aufgabenstellung: Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listefeld “3” in der Grafischen Benutzeroberfläche(GUI) angezeigt.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Kamerabilder Anzeigen	1.1.2	Sven Stegemann	Eugen Zwetlich
Ergebniss: -			
Aufgabenstellung: Es werden die Bilder der USB-Kameras der Roboter in den Feldern “1“ angezeigt.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Visualisierung	1.1.3	Sven Stegemann	Eugen Zwetlich
Ergebniss: -			
Aufgabenstellung: In dem Feld “5“, werden die Bewegungen unserer Roboter grafisch dargestellt.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
FormCreate	1.1.4	Sven Stegemann	Eugen Zwetlich
Ergebniss: -			
Aufgabenstellung: In der FormCreate, wird das Fenster für das Programm erstellt. Außerdem werden in der FormCreate, die IP-Adressen aus einer IP-Config.txt Datei eingelesen und in einem Array abgelegt.			

5.3 Klassen

Für die Berechnungen und Logik haben wir eigene Klassen geschrieben.

Diese unterteilen sich in:

- mVektor
- mTKI
- mKonstanten
- mRoboterDaten

5.3.1 mVektor

Die Klasse mVektor besteht aus dem Record TVektor.

Dieser hat folgende Funktionen, überladende Operatoren und Variablen:

- Funktionen

Winkel(überladen) Berechnet den Winkel zwischen dem Vektor und der X-Achse. Als Rückgabewert erhält man einen Wert im Bogenmaß im Intervall von $[0;2\pi)$.

Winkel(überladen) Berechnet den Winkel zwischen zwei Vektoren. Als Rückgabewert erhält man einen Wert im Bogenmaß im Intervall von $[0;2\pi)$.

Betrag Es wird die Länge des Vektors(euklidische Norm: 2-Norm) berechnet.

Drehen Mit der Drehmatrix wird ein neuer Vektor berechnet, der um einen als Parameter übergebenen Winkel nach links(positiv) bzw. nach rechts(negativ) gedreht ist.

- Operatoren

Add Es werden die Komponenten der jeweiligen Vektoren addiert und anschließend ein neuer Vektor zurück gegeben.

Subtract Es werden die Komponenten der jeweiligen Vektoren subtrahiert und anschließend ein neuer Vektor zurück gegeben.

Multiply(überladen) Es werden die einzelnen Komponenten des Vektors mit einem Skalar multipliziert und ein neuer Vektor zurück gegeben.

Multiply(überladen) Es wird ein Skalar mit den Komponenten eines Vektors multipliziert und ein Skalar zurück gegeben.

Equal Es werden die einzelnen Komponenten zweier Vektoren auf Gleichheit überprüft.

- Variablen

x,y sind die Komponenten eines Vektors.

5.3.2 mTKI

Die Klasse mTKI hat einen Datentypen TAction mit den Werten Fliehen und Fangen und eine abgeleitete Klasse TKI von TObject.

Die abgeleitete Klasse TKI besteht aus folgenden Funktionen, Prozeduren und Variablen:

Funktionen:

Arbeitspaket Priorität Festlegen	AP-Nr.: 2.1.1	Arbeitspaketverantwortlicher Sven Stegemann	Beteiligte Personen Eugen Zwetlich
Ergebniss: -FLIEHEN -FANGEN Aufgabenstellung: Anhand der Positionsdaten der gegnerischen Roboter wird überprüft, welcher sich am nächsten an unserem Roboter befindet. Anschließend wird über die Winkel Funktion von der Klasse mVektor ermittelt, ob sich dieser Roboter vor oder hinter unserem befindet. Danach wird die Priorität auf FLIEHEN bzw. auf FANGEN gesetzt.			

Arbeitspaket Fangvektor Berechnen	AP-Nr.: 2.1.2	Arbeitspaketverantwortlicher Sven Stegemann	Beteiligte Personen Jonah Vennemann
Ergebniss: -Vektor Aufgabenstellung: Es wird der Vektor zum nächsten gegnerischen Roboter, der Gefangen werden soll, berechnet. Als Rückgabewert erhält man einen neuen Vektor.			

Arbeitspaket Fliehvektor Berechnen	AP-Nr.: 2.1.3	Arbeitspaketverantwortlicher Sven Stegemann	Beteiligte Personen Michael Mertens
Ergebniss: -Vektor Aufgabenstellung: Es wird ein Vektor, mit Bezug auf den gegnerischen Roboter von dem Geflohen werden soll, berechnet. Als Rückgabewert erhält man einen neuen Vektor.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Rand Ausweichvektor Berechnen	2.1.4	Sven Stegemann	Michael Mertens
Ergebniss: -Vektor			
Aufgabenstellung: Es wird überprüft ob sich der Roboter im Spielfeld befindet ist dieser außerhalb, so fährt er sofort in's Spielfeld rein. Danach wird überprüft ob sich der Roboter in der Nähe des Spielfeldrandes befindet. Ist dieser zu Nah am Spielfeldrand, wird der Roboter nach links bzw. nach rechts gedreht.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Roboter Ausweichvektor Berechnen	2.1.5	Sven Stegemann	Michael Mertens
Ergebniss: -Vektor			
Aufgabenstellung: Es wird geprüft welche Roboter aus unserem Team untereinander kollidieren würden. Wurde ein Roboter ermittelt, so wird dieser um die Konstante AUSWEICHWINKEL gedreht. Als Rückgabewert erhält man einen neuen Vektor.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Rausfahrvektor Berechnen	2.1.6	Sven Stegemann	Michael Mertens
Ergebniss: -Vektor			
Aufgabenstellung: Sobald ein Roboter als „Gefangen“ gemeldet ist, wird anhand seiner Position und der Spielfeldgröße ein Vektor zum Herausfahren berechnet. Als Rückgabewert erhält man einen neuen Vektor.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Serverdaten Empfangen	2.1.7	Sven Stegemann	Michael Mertens
Ergebniss: -			
Aufgabenstellung: So bald sich der Client mit dem Server verbunden hat, werden die Variablen des Roboters mit Daten vom Server gefüllt.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Anmelden	2.1.8	Sven Stegemann	Michael Mertens
Ergebniss: -			
Aufgabenstellung: Ist eine Funktion um sich mit dem Server zu verbinden.			

Prozeduren:

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Steuerbefehl Senden	2.2.1	Sven Stegemann	Eugen Zwetich
Ergebniss: -			
Aufgabenstellung: Als erstes wird überprüft ob der aktuelle Vektor oder der Zielvektor ein Nullvektor ist. Danach wird ermittelt ob der aktuelle Vektor sich links bzw. rechts vom Roboter befindet. Zum Schluss wird dem Roboter eine Standardgeschwindigkeit übergeben.			

Arbeitspaket	AP-Nr.:	Arbeitspaketverantwortlicher	Beteiligte Personen
Geschwindigkeit Berechnen	2.2.2	Sven Stegemann	Michael Mertens
Ergebniss: -			
Aufgabenstellung: Es wird von dem Vektor Geschwindigkeit, die Geschwindigkeit in $\frac{m}{s}$ berechnet.			

Arbeitspaket Initialisierung	AP-Nr.: 2.2.3	Arbeitspaketverantwortlicher Sven Stegemann	Beteiligte Personen Michael Mertens
Ergebniss: -			
Aufgabenstellung: Anhand der IP-Adressen, wird jeweils ein Roboter von der Klasse TTXTMobilRoboter erstellt. Ist keine Verbindung möglich, so wird ein Fehler in die Log-Datei geschrieben.			

Arbeitspaket Steuern	AP-Nr.: 2.2.4	Arbeitspaketverantwortlicher Sven Stegemann	Beteiligte Personen Michael Mertens
Ergebniss: -			
Aufgabenstellung: Ist eine Funktion um sich mit dem Server zu verbinden.			

5.3.3 mKonstanten

Da wir an verschiedenen Stellen die gleichen Werte benötigten, erstellten wir eine eigene Klasse für Konstanten.

- Variablen
 - Mindestabstand
 - Nullvektor
 - Rand
 - Ausweichwinkel
 - LängeFliehvektor

5.3.4 mRoboterDaten

Um den Zugriff auf die Daten eines Roboters zu vereinfachen, haben wir diese in einer eigenen Klasse mRoboterDaten untergebracht.

Diese besteht aus einem Record TRoboterDaten mit folgenden Variablen:

- Variablen
 - Position
 - Geschwindigkeit
 - Positionsverlauf
 - Aktiv

6 Fazit

Listing 1: Klasse mTKI

```

1  unit mTKI;
2
3  interface
4
5  uses mVektor, mTXTMobilRoboter, Client, ClientUndServer, DateUtils,
6      mHauptformular, mKonstanten, Math, Generics.Collections, mRoboterDaten, SysUtils;
7
8  type TAktion = (FANGEN, FLIEHEN);
9
10 type TKI = class(TObject)
11     strict private
12         class var Formular: THauptformular;
13         class var ZeitLetzterFrames: TQueue<TDateTime>;
14         class var RoboterDaten: Array[TTeam] of Array of TRoboterDaten;
15         class var Roboter: Array of mTXTMobilRoboter;
16         class var Spielfeld: TVektor;
17         class var Server: TServerVerbindung;
18
19         class function PrioritaetFestlegen(index: Integer; out Ziel: Integer): TAktion;
20         class function FangvektorBerechnen(index, Ziel: Integer): TVektor;
21         class function FliehvektorBerechnen(index, Ziel: Integer): TVektor;
22         class function AusweichvektorBerechnen(index: Integer; Vektor: TVektor): TVektor;
23         class function RausfahrvektorBerechnen(index: Integer): TVektor;
24         class procedure SteuerbefehlSenden(index: Integer; Vektor: TVektor);
25         class procedure GeschwindigkeitenBerechnen(zeit: TDateTime);
26         class function ServerdatenEmpfangen: Boolean;
27
28     public
29         class procedure Init(IP_Adressen: Array of String; Server_Adresse: String; Port: Integer);
30         class procedure Steuern(Spielende: TDateTime);
31         class function Anmelden(Teamwahl: TTeam): Boolean;
32 end;
33
34 implementation
35
36 { TKuenstlicheIntelligenz }
37
38 class function TKI.Anmelden(Teamwahl: TTeam): Boolean;
39 begin
40     Server.anmelden(Teamwahl);
41     if Server.anmelden then
42         Formular.Log_Schreiben('Anmelden erfolgreich', Hinweis)
43     else
44         Formular.Log_Schreiben('Anmelden nicht erfolgreich', Fehler);
45 end;
46
47 class function TKI.AusweichvektorBerechnen(index: Integer; vektor: TVektor): TVektor;
48 var
49     ZielPosition, aktPos: TVektor;
50     VWinkel, t: Double;
51     i: integer;
52     deltaP, deltaV: TVektor;
53     deltaWinkel: Double;
54 begin
55     ZielPosition := RoboterDaten[TEAM_BLAU,index].Position + vektor;
56     aktPos := RoboterDaten[TEAM_BLAU,index].Position;
57     VWinkel := 0;
58
59     if vektor = NULLVEKTOR then exit;
60     //Roboter befindet sich ausserhalb des Spielfeldes

```

```

62   if (aktPos.x>Spielfeld.x) or (aktPos.x<0) or (aktPos.y<0) or (aktPos.y>Spielfeld.y) then
63       result := Spielfeld*0.5 - aktPos
64   //Aus Ecke herausfahren
65   else if (Zielposition.x>Spielfeld.x) and (Zielposition.y>Spielfeld.y) or
66       (Zielposition.x>Spielfeld.x) and (Zielposition.y<0) or
67       (Zielposition.y>Spielfeld.y) and (Zielposition.x<0) or
68       (Zielposition.x<0) and (Zielposition.y<0) then
69       begin
70           result.x := -(vektor.x);
71           result.y := -(vektor.y);
72       end
73   //Oberen Spielfeldrand nicht ueberfahren
74   else if (Zielposition.y > Spielfeld.y) then begin
75       result.y := Spielfeld.y-aktPos.y;
76       result.x := Sign(vektor.x) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.y));
77   end
78   //Unteren Spielfeldrand nicht ueberfahren
79   else if Zielposition.y < 0 then begin
80       result.y := -aktPos.y;
81       result.x := Sign(vektor.x) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.y));
82   end
83   //Rechten Spielfeldrand nicht ueberfahren
84   else if (Zielposition.x > Spielfeld.x) then begin
85       result.x := Spielfeld.x-aktPos.x;
86       result.y := Sign(vektor.y) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.x));
87   end
88   //Linken Spielfeldrand nicht ueberfahren
89   else if (Zielposition.x < 0) then begin
90       result.x := -aktPos.x;
91       result.y := Sign(vektor.y) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.x));
92   end;
93
94   //Kollisionen mit TeamRobotern vermeiden
95   if index = High(RoboterDaten[TEAM_BLAU]) then Exit;
96   if RoboterDaten[TEAM_BLAU,index].Geschwindigkeit.Winkel(vektor.winkel) > AUSWEICHWINKEL then Exit;
97
98   for i := index+1 to High(RoboterDaten[TEAM_BLAU]) do begin
99       deltaP := RoboterDaten[TEAM_BLAU,index].Position - RoboterDaten[TEAM_BLAU,i].Position;
100      deltaV := RoboterDaten[TEAM_BLAU,index].Geschwindigkeit - RoboterDaten[TEAM_BLAU,i].Geschwindigkeit;
101      try
102          t := (deltaP.x*deltaV.x+deltaP.y*deltaV.y)/Power(deltaV.Betrag,2);
103      except
104          on EDivByZero do Continue; // Zu kleines deltaV => Roboter fahren parallel => kein Ausweichen
105      end;
106
107      if (t>=0) and (t<5) then
108          if ((RoboterDaten[TEAM_BLAU,index].Position+t*RoboterDaten[TEAM_BLAU,index].Geschwindigkeit)
109              (RoboterDaten[TEAM_BLAU,i].Position+t*RoboterDaten[TEAM_BLAU,i].Geschwindigkeit)).Betrag
110              begin
111                  deltaWinkel := RoboterDaten[TEAM_BLAU,i].Geschwindigkeit.winkel - RoboterDaten[TEAM_BLAU,index].Geschwindigkeit.winkel;
112                  if deltaWinkel < 0 then
113                      deltaWinkel := deltaWinkel + 2*pi;
114                  if deltaWinkel < Pi then begin
115                      // Weiche nach rechts aus
116                      result.x := cos(AUSWEICHWINKEL)*vektor.x + sin(AUSWEICHWINKEL)*vektor.y;
117                      result.y := -sin(AUSWEICHWINKEL)*vektor.x + cos(AUSWEICHWINKEL)*vektor.y;
118                  end
119                  else begin
120                      // Weiche nach links aus
121                      result.x := cos(-AUSWEICHWINKEL)*vektor.x + sin(-AUSWEICHWINKEL)*vektor.y;
122                      result.y := -sin(-AUSWEICHWINKEL)*vektor.x + cos(-AUSWEICHWINKEL)*vektor.y;
123                  end;
124              end;
125          end;

```



```

124     end;
125     end;
126 end;
127
128 class function TKI.FangvektorBerechnen(index,ziel: Integer): TVektor;
129 begin
130     result := RoboterDaten[TEAM_Rot,ziel].Position-RoboterDaten[TEAM_BLAU,index].Position;
131 end;
132
133 class function TKI.FliehvektorBerechnen(index,ziel: Integer): TVektor;
134 begin
135     result := RoboterDaten[TEAM_BLAU,index].Position-RoboterDaten[TEAM_rot,ziel].Position;
136     result := (LAENGE_FLIEHVEKTOR/result.Betrag)*result;
137 end;
138
139 class procedure TKI.GeschwindigkeitenBerechnen(zeit: TDateTime);
140 var
141     einRoboter: TRoboterDaten;
142     team: TTeam;
143     i: Integer;
144 begin
145     ZeitLetzterFrames.Enqueue(zeit);
146
147     for team in [TEAM_ROT, TEAM_BLAU] do
148     begin
149         for i := Low(RoboterDaten[team]) to High(RoboterDaten[team]) do
150         begin
151             einRoboter.Geschwindigkeit := (RoboterDaten[team,i].Position -
152             RoboterDaten[team,i].Positionsverlauf.Dequeue)*(1/SecondSpan(zeit, ZeitLetzterFrames.Dequeue))
153         end;
154     end;
155 end;
156
157 class procedure TKI.Init(IP_Adressen: Array of String; Server_Adresse: String; Port: Integer);
158 var i: Integer;
159 begin
160     Server.Create(Server_Adresse, Port);
161
162     setlength(Roboter, Length(IP_Adressen));
163     for i:= Low(Roboter) to High(Roboter) do
164     begin
165         try
166             Roboter[i]:=TTXTMobilRoboter.Create(Ip_Adressen[i]);
167             Roboter[i].Start;
168         except
169             Hauptformular.Log_Schreiben('Verbindung nicht moeglich', Fehler);
170         end;
171     end;
172 end;
173
174 class function TKI.PrioritaetFestlegen(index: Integer; out ziel: Integer): TAction;
175 var DeltaVektor: TRoboterDaten;
176     KleinsterAbstand,Abstand: Double;
177     i,NaechsterRoboter: Integer;
178 begin
179     NaechsterRoboter := 0;
180     KleinsterAbstand := (RoboterDaten[TEAM_BLAU,index].Position -
181     RoboterDaten[TEAM_ROT,0].Position).Betrag;
182
183     //Pruefung welcher Roboter vom Team Rot am naechsten am Roboter vom Team Blau ist
184     for i := Low(RoboterDaten[TEAM_ROT])+1 to High(RoboterDaten[TEAM_ROT]) do
185     begin

```

```

186     if RoboterDaten[TEAM_ROT,i].Aktiv then
187     begin
188         Abstand := (RoboterDaten[TEAM_BLAU,index].Position -
189                     RoboterDaten[TEAM_ROT,i].Position).Betrag;
190         if Abstand < KleinsterAbstand then
191         begin
192             KleinsterAbstand := Abstand;
193             NaechsterRoboter := i;
194         end;
195     end;
196 end;
197
198 ziel := NaechsterRoboter;
199
200 //Pruefung ob der Roboter von Team Rot sich vor oder hinter dem Roboter von
201 //Team Blau befindet
202 Try
203 if InRange(
204     (RoboterDaten[TEAM_BLAU,index].Position - RoboterDaten[TEAM_ROT,NaechsterRoboter].Position)
205     .Winkel(RoboterDaten[TEAM_BLAU,index].Geschwindigkeit), pi*0.5, pi*1.5) then
206     Result := FLIEHEN;
207 else
208     Result := FANGEN;
209 Except
210 on EMathError do Formular.Log_Schreiben('Zwei Roboter haben die gleiche Position oder ein eigene
211 End;
212 end;
213
214 class function TKI.RausfahrvektorBerechnen(
215     index: Integer): TVektor;
216 var Position: TVektor;
217     Abstand: Array[0..3] of Double;
218     KAbstand: Double;
219 begin
220     Position := RoboterDaten[TEAM_BLAU,index].Position;
221     //Berechnung der Seitenabstaende mit der Annahme,
222     //dass sich unten links der Koordinatenursprung befindet.
223     Abstand[0] := Position.x; //links
224     Abstand[1] := Spielfeld.x-Position.x; //rechts
225     Abstand[2] := Spielfeld.y-Position.y; //oben
226     Abstand[3] := Position.y; //unten
227     KAbstand := MinValue(Abstand);
228     //in x- oder y-Richtung herausfahren
229     if (KAbstand=Abstand[0]) or (KAbstand=Abstand[1]) then begin
230         result.x := -KAbstand;
231         result.y := 0;
232     end
233     else begin
234         result.x := 0;
235         result.y := -KAbstand;
236     end
237 end;
238
239 class function TKI.ServerdatenEmpfangen: Boolean;
240 var
241     i: Integer;
242     Team: TTeam;
243     Serverdaten: TSpielstatus;
244 begin
245     Serverdaten:= Server.StatusEmpfangen;
246     for Team in [Team_Blau,Team_Rot] do
247     begin

```

```

248     for i := low(Roboterdaten[Team]) to High(Roboterdaten[Team]) do
249     begin
250         Roboterdaten[Team,i].Position.x:=Serverdaten.Roboterpositionen[Team,i].x;
251         Roboterdaten[Team,i].Position.y:=Serverdaten.Roboterpositionen[Team,i].y;
252         Roboterdaten[Team,i].Aktiv:=Serverdaten.RoboterIstAktiv[Team,i];
253
254         Roboterdaten[Team,i].Positionsverlauf.Enqueue(Roboterdaten[Team,i].Position);
255
256         GeschwindigkeitenBerechnen(Serverdaten.Zeit);
257     end;
258 end;
259 end;
260
261 class procedure TKI.SteuerbefehlSenden(index: Integer; vektor: TVektor);
262 var
263     Roboter_Blau: TTXTMobilRoboter;
264     Daten: TRoboterDaten;
265     akt_Vektor: tVektor;
266
267 const
268     Geschwindigkeit= 512;
269     c_Radius = 2;           //Konstante zum drehen, auf Grad bezogen
270
271 begin
272     Roboter_Blau:= Roboter[Index];
273     akt_Vektor:=Roboterdaten[Team_Blau,Index].Geschwindigkeit;
274
275     if not((akt_Vektor=NULLVEKTOR) or (vektor=NULLVEKTOR)) then
276     begin
277         if Vektor.Winkel(akt_Vektor)<pi then
278             Roboter_Blau.Bewegenalle(Geschwindigkeit,
279                                     Geschwindigkeit- round(c_Radius*RadToDeg(Vektor.Winkel(akt_Vektor))),
280
281             else
282             Roboter_Blau.Bewegenalle(Geschwindigkeit- round(c_Radius*RadToDeg(Vektor.Winkel(akt_Vektor))),
283                                     Geschwindigkeit)
284         end;
285     end;
286
287 class procedure TKI.Steuern(spielende: TDateTime);
288 var einRoboter: TTXTMobilRoboter;
289 begin
290     // Startaufstellung einnehmen
291     // Queues fuellen
292
293     while True do // Andere Bedingung
294     begin
295         ServerdatenEmpfangen;
296         GeschwindigkeitenBerechnen;
297         for einRoboter in Roboter do
298         begin
299             {if einRoboter then
300                 if einRoboter.LiesDigital() then
301                 }
302             end;
303
304         end;
305     end;
306
307 end.

```