Dokumentation

Roboter-Fangen

Maschinenbauinformatik 3. Semester

Michael Mertens,
Jonah Vennemann,
Sven Stegemann,
Eugen Zwetzich

15. Februar 2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

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.

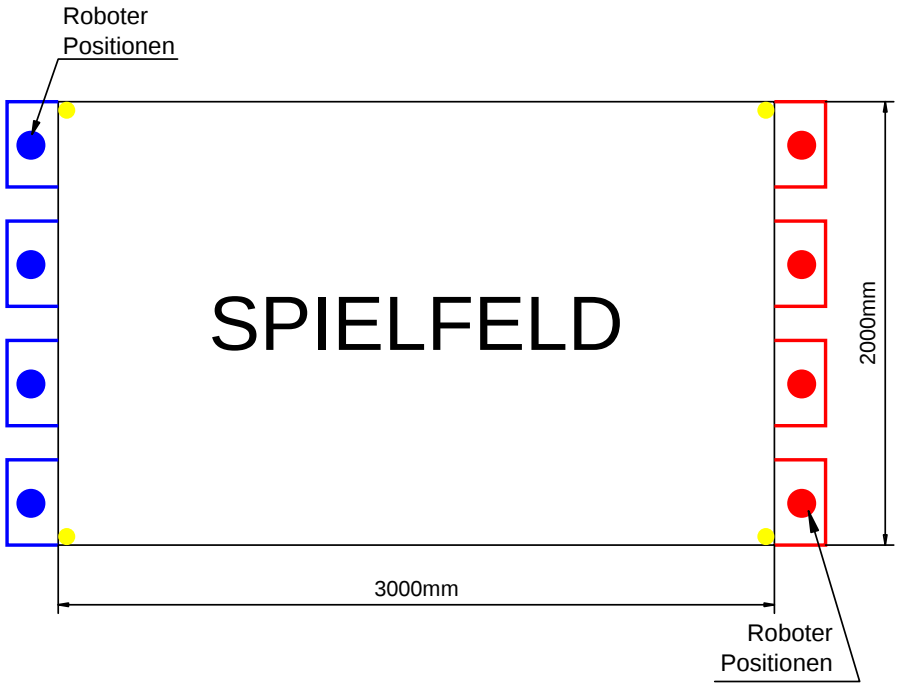
Dabei werden die Positionsdaten aller Roboter von einem Schiedsrichter-Server mit Hilfe einer Kamera berechnet und an die Steuerungssoftware der beiden Teams geschickt. Hauptbestandteile der Steuerungssoftware:

- 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 um seine Achse 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 Instrument des Projektmanagements, das die zeitliche Abfolge von Aktivitäten grafisch in Form von Balken auf einer Zeitachse darstellt.

In den unteren Abbildungen sieht man einerseits die einzelnen Aktivitäten, Ressourcen und die Mitarbeiter die wir für unser Projekt „Roboter-Fangen“ eingeplant haben, einmal als Liste und in einer grafischen Darstellung.

Außerdem kann man in der Liste erkennen, welche Aktivität welchen Vorgänger bzw. Nachfolger hat. Wo von es abhängig ist.

Durch die grafische Darstellung ist es möglich frühzeitig Engpässe zu erkennen und das Projekt nötigen falls um zu strukturieren.

Einige Aktivitäten haben wir der Übersichtlichkeit in Gruppen eingeteilt:

Gruppen:

- Planung
- Programmierung-Teil 1
  - Simple KI
- Programmierung-Teil 2
- Dokumentation

Meilensteine:

- M0: Start
- M1: Planung abgeschlossen
- M2: Erste Implementierung
- M3: Kurzpräsentation
- M4: Programmierung abgeschlossen
- M5: Endpräsentation
- M6: Dokumentation abgeschlossen

		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 Zwetzich;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 Zwetzich
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 Zwetzich
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 Zwetzich
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 Zwetzich;Com...
34		M6: Dokumentation abge...	0 tage	15.02.16 17:00	15.02.16 17:00	32	

Roboter fangen Roboter

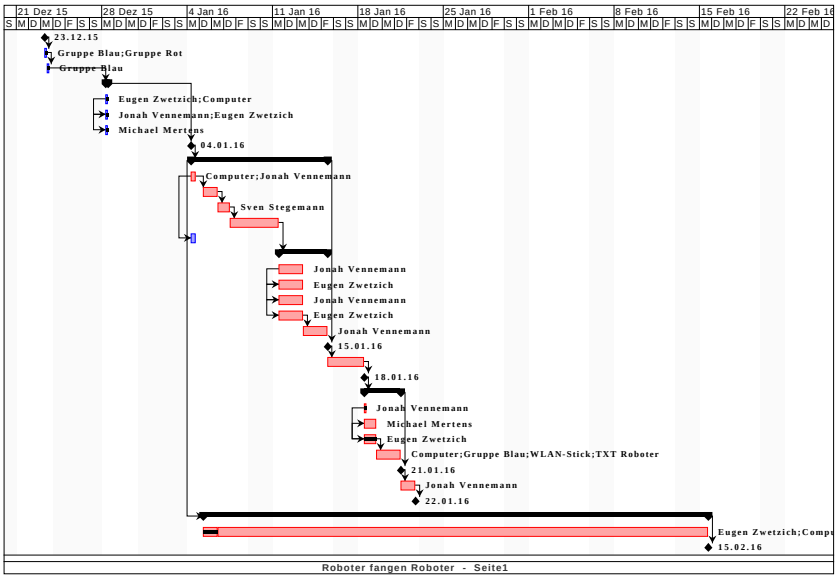


Abbildung 1: Gantt-Chart v1.0

Aufgrund von Komplikationen mit den Roboter Klassen, mit dem Voranstreiten der Programmierung des Servers und der Komplexität der Programmierung der KI mussten wir die Planung des Projektes umstrukturieren.

Wir haben die Aktivitäten für die Benutzeroberfläche, das Ereignisprotokoll und die Vektorberechnung in den Programmierung-Teil 1 vorgezogen.

Erst im 2. Teil der Programmierung fingen wir mit der Programmierung der KI und der Kommunikation mit dem Server an.

Gruppen:

- Planung
- Programmierung-Teil 1
- Programmierung-Teil 2
  - Simple KI
- Server-Team

Meilensteine:

M1: Planung abgeschlossen Es muss die Projektbeschreibung, eine Skizze für die KI und ein Spielablauf fertig sein.

M2: Kurzpräsentation Eine Präsentation über die zeitliche Planung und über die Aufwandsschätzung

M3: Programmierung abgeschlossen Es muss die ganze Programmierung(Vektorberechnung, Künstliche Intelligenz, Benutzeroberfläche) der Software abgeschlossen sein.

M4: Endpräsentation Eine Abschlusspräsentation vom Projekt mit einer Vorführung des Spiels

M4: Dokumentation abgeschlossen Abgabe der Dokumentation in ausgedruckter und digitaler Form

		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	Alle
3	✔	Aufgabenaufteilung	0,5 tage	23.12.15 12:00	23.12.15 17:00	2	Sven Stegemann;Eug...
4	✔	**Planung**	**1 tag**	**28.12.15 08:00**	**28.12.15 17:00**	**3**	
5	✔	Projektbeschreibung	1 tag	28.12.15 08:00	28.12.15 17:00		Michael Mertens
6	✔	Spielablauf	0,5 tage	28.12.15 08:00	28.12.15 13:00	5AA	Eugen Zwetzich
7	✔	Skizze KI	0,5 tage	28.12.15 08:00	28.12.15 13:00	6AA	Eugen Zwetzich;Sven ...
8		M1: Planung abgesc...	0 tage	28.12.15 17:00	28.12.15 17:00	4	
9	✔	**Programmierung-...**	**5 tage**	**07.01.16 08:00**	**13.01.16 17:00**	**8**	
10	✔	Benutzeroberfläche	3 tage	07.01.16 08:00	11.01.16 17:00		Jonah Vennemann
11	✔	Ereignisprotokoll	2 tage	12.01.16 08:00	13.01.16 17:00	10	Jonah Vennemann
12	✔	Vektorberechnung	2 tage	07.01.16 08:00	08.01.16 17:00	10AA	Eugen Zwetzich
13		M2: Kurzpräsentation	0 tage	13.01.16 17:00	13.01.16 17:00	9	
14	✔	**Programmierung-...**	**19 tage**	**14.01.16 08:00**	**09.02.16 17:00**	**13**	
15	✔	Priorität festlegen	2 tage	14.01.16 08:00	15.01.16 17:00		Eugen Zwetzich
16	✔	**Simple KI**	**11 tage**	**18.01.16 08:00**	**01.02.16 17:00**	**15**	
17	✔	Im Spielfeld bleiben	3 tage	18.01.16 08:00	20.01.16 17:00		Michael Mertens
18	✔	Spielrand auswei...	4 tage	21.01.16 08:00	26.01.16 17:00	17	Michael Mertens
19	✔	Fangen	2 tage	18.01.16 08:00	19.01.16 17:00	17AA	Jonah Vennemann
20	✔	Fliehen	2 tage	18.01.16 08:00	19.01.16 17:00	19AA	Michael Mertens
21	✔	Roboter ausweic...	4 tage	27.01.16 08:00	01.02.16 17:00	18	Michael Mertens
22	✔	Positionsdaten em...	3 tage	02.02.16 08:00	04.02.16 17:00	16	Jonah Vennemann
23	✔	Gefangener Robot...	3 tage	05.02.16 08:00	09.02.16 17:00	22	Michael Mertens
24		M4: Programmierun...	0 tage	11.02.16 17:00	11.02.16 17:00	14	
25		Tests	0,75 tage	12.02.16 08:00	16.02.16 17:00	24	Eugen Zwetzich;Sven ...
26		M5: Endpräsentation	0 tage	17.02.16 17:00	17.02.16 17:00	25;29	
27		Dokumentation	11 tage	01.02.16 08:00	15.02.16 17:00	8	Eugen Zwetzich
28		M6: Dokumentation ...	0 tage	16.02.16 17:00	16.02.16 17:00	27	
29		**Server-Team**	**32 tage**	**23.12.15 08:00**	**10.02.16 17:00**		
30	✔	Roboter Klassen	10 tage	23.12.15 08:00	05.01.16 17:00		Sven Stegemann
31		Server Programmie...	26 tage	06.01.16 08:00	10.02.16 17:00	30	Sven Stegemann

Roboter-Fangen

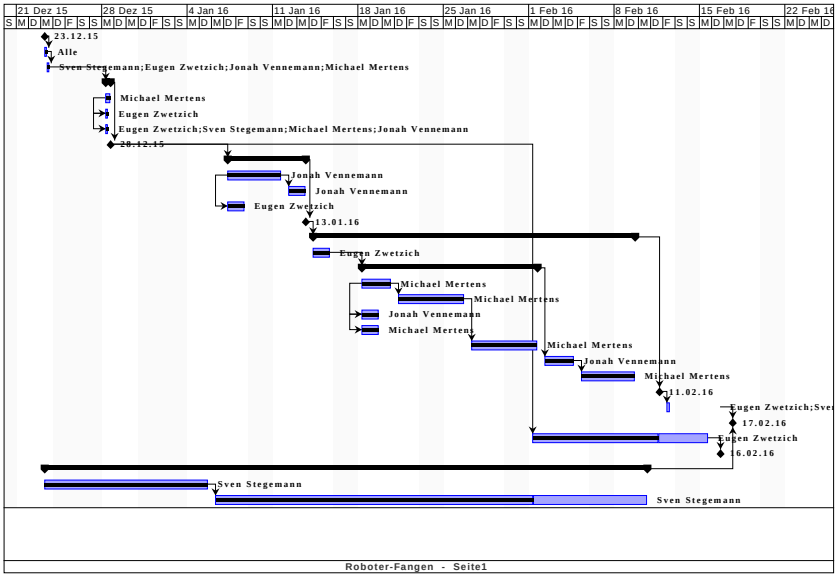


Abbildung 2: Gantt-Chart v2.0

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

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

5. Schritt

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

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

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

$$\begin{aligned}\text{3 Personen} &= 224{,}256\text{h pro Person} \\ \text{4 Personen} &= 168{,}192\text{h pro Person}\end{aligned}$$

$\Rightarrow$ Bei einem 4 Mann starken Team benötigen wir ca. 170h pro Person.

3 GitHub

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

Diesen Online-Dienst haben wir dafür benötigt, um von verschiedenen Standtorten und mit verschiedenen Mitarbeitern an dem Projekt arbeiten zu können.

Außerdem kann man sich anzeigen, welcher Mitarbeiter, wie lange an dem Projekt gearbeitet hat.

[1] TFP=Total Function Points
[2] Personenmonate(PM) = 20 Arbeitstage

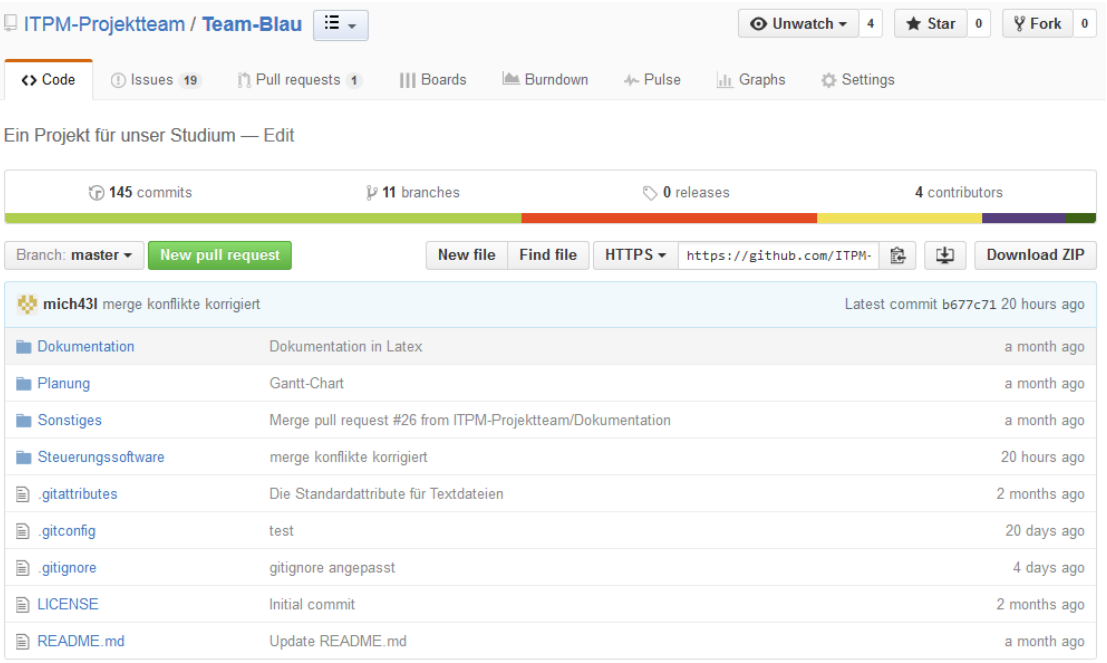


Abbildung 3: GitHub

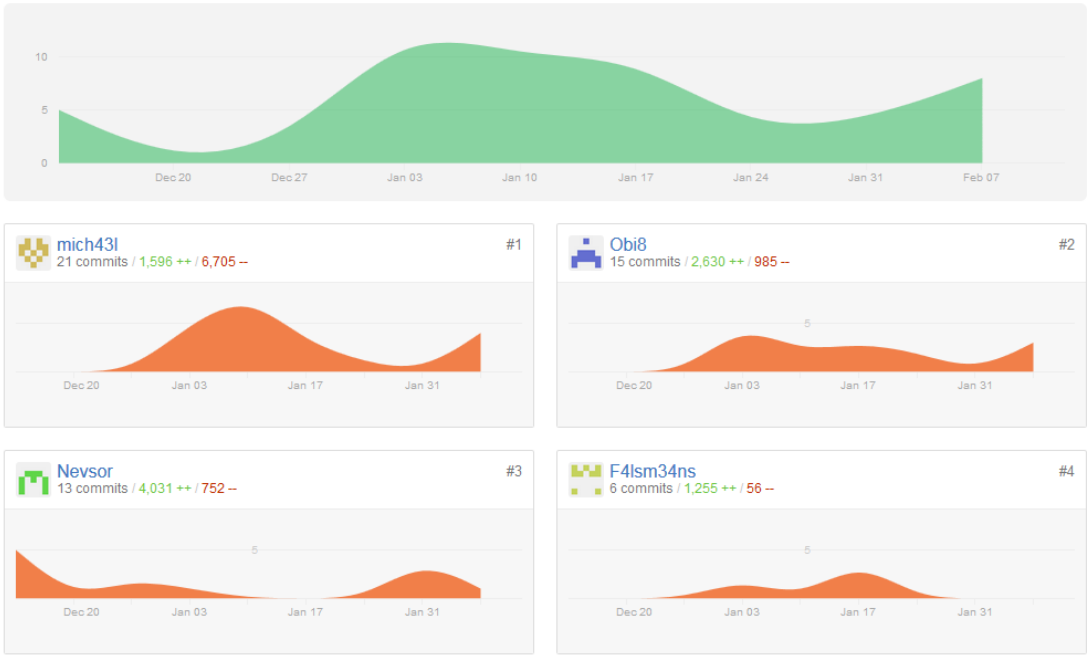


Abbildung 4: GitHub Mitarbeiterzeit

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(Eingefrorene Punkte)
- Backlog(Arbeitsrückstand)
- To Do(Muss gemacht werden)
- In Progress(In Bearbeitung)
- Done/Review(Fertig/Überprüfung)

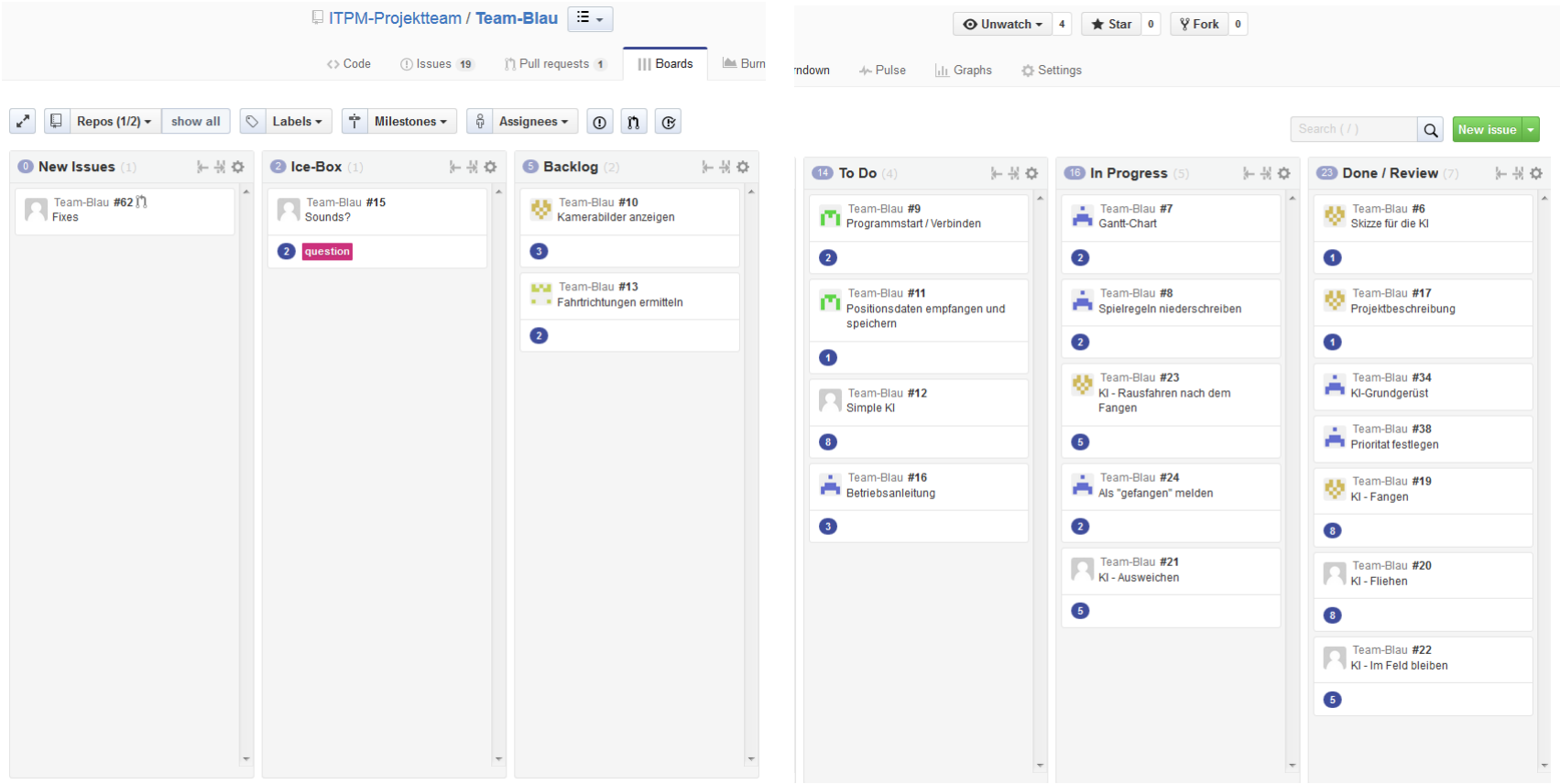


Abbildung 5: Auflistung der Punkte in ZenHub

4 Bedienungsanleitung

Vorbereitung:
Für das Spiel benötigt man 3 Computer, 8 Roboter und eine USB-Kamera. Außerdem werden in ausreichender Zahl Netzwerkadapter gebraucht, um die Roboter mit den Computern zu verbinden.

Auf 2 Computern wird jeweils die Team-KI ausgeführt. Auf dem verbleibendem Computer wird das Server Programm gestartet. Unsere Roboter haben, durch einen Eingriff über die PuTTy und die Bash eine feste IP-Adresse bekommen, diese ist am hinteren Ende eines jeden Roboters angebracht.

Alle IP-Adressen unserer Roboter müssen in der Datei „Porjektordner/Steuerungssoftware/Win32/Debug/ip_config.txt nachdem Eintrag „Roboteradressen“ eingefügt werden. Nachdem dem Eintrag „Serveradresse“ in der ip_config.txt Datei wird die IP-Adresse des Servers, sowie dessen Port in der Form „IP Port“ eingetragen.

Nun müssen sich alle Roboter mittels einer WLAN-Verbindung und dem TCP/IP an dem jeweiligen Computer anmelden, der die Team-KI steuert. Die Roboter erstellen selbstständig einen Access Point, sodass ein einfaches Verbinden unter der Windows Oberfläche möglich ist.

Die Ecken des Spielfeldes werden durch gelbe Kreise dargestellt. Die Roboter der beiden Teams werden gegenüber in Stellung gebracht.

Spielstart:
Wenn alle Vorbereitungen abgeschlossen sind und das Serverprogramm ausgeführt wurde, kann die Steuerungssoftware der beiden Teams gestartet werden. Die .exe Datei befindet sich im gleichnamigen Ordner wie die ip_config.txt.

Mit der Betätigung des Schaltknopfes „Verbinden“ in der Benutzeroberfläche(GUI), wird die Verbindung zum Server initialisiert. Erst durch das anhacken des Kontrollkästchens „Bereit“ melden sich die Roboter beim Server an und teilen den gleichzeitig mit, dass diese Bereit sind zum Spielstart.

Beim erfolgreichen Anmelden erscheint ein Hinweis im Ereignisprotokollbereich, sowie beim Misserfolg des Anmeldens.

Während des Spiels:
Sobald das Spiel läuft, sind von dem Benutzer keine Eingaben erforderlich. Der Spielverlauf und die Bewegungen der Roboter, können im Log-Bereich, sowie in der grafischen Ausgabe rechts neben dem Log-Bereich verfolgt werden. Im linken Teil der Anwendung ist es möglich sich das Geschehen aus Sicht der Roboter zu verfolgen.

Spielende:
Das Spiel wird beendet, wenn alle Roboter eines Teams gefangen wurden oder die Spielzeit(30min) verstrichen ist. In beiden Fällen schickt der Server eine Nachricht an beide Teams und die KI wird beendet. Die Roboter fahren **nicht** selbstständig in die Startaufstellungen und müssen für ein neues Spiel manuell auf die Startpositionen platziert werden.

5 Systemaufbau

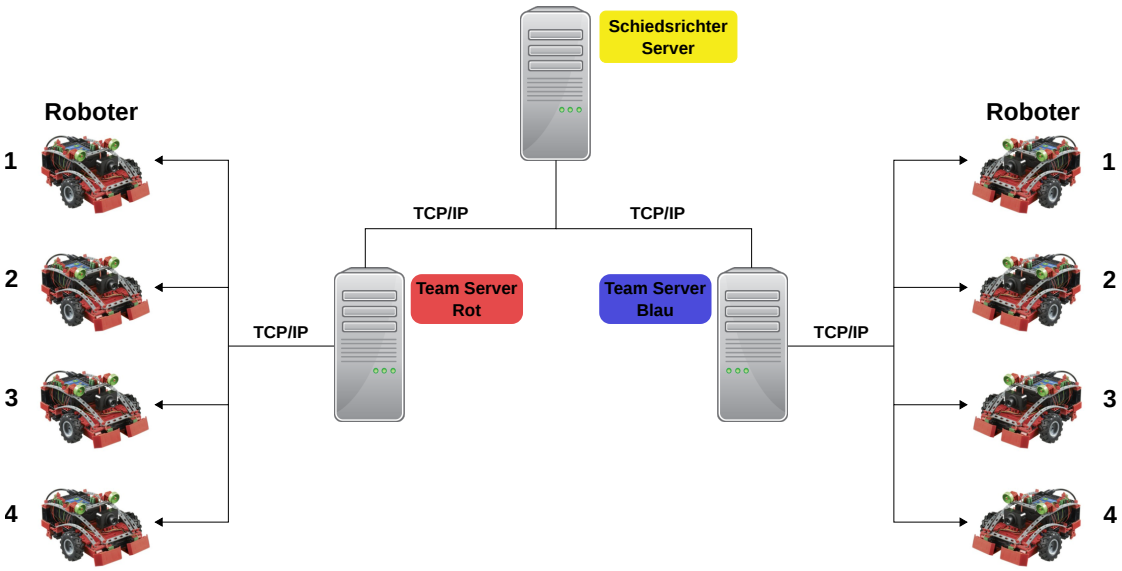


Abbildung 6: Systemaufbau

Der Systemaufbau des Projekts stellt sich wie folgt dar:

Ein zentraler Schiedsrichter-Server überwacht das Spielfeld der Roboter mithilfe einer Kamera. Anhand des Kamerabildes 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 WLAN-Verbindung und des TCP/IP-Protokolls 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.

6 Programmierung

6.1 Programmablaufplan

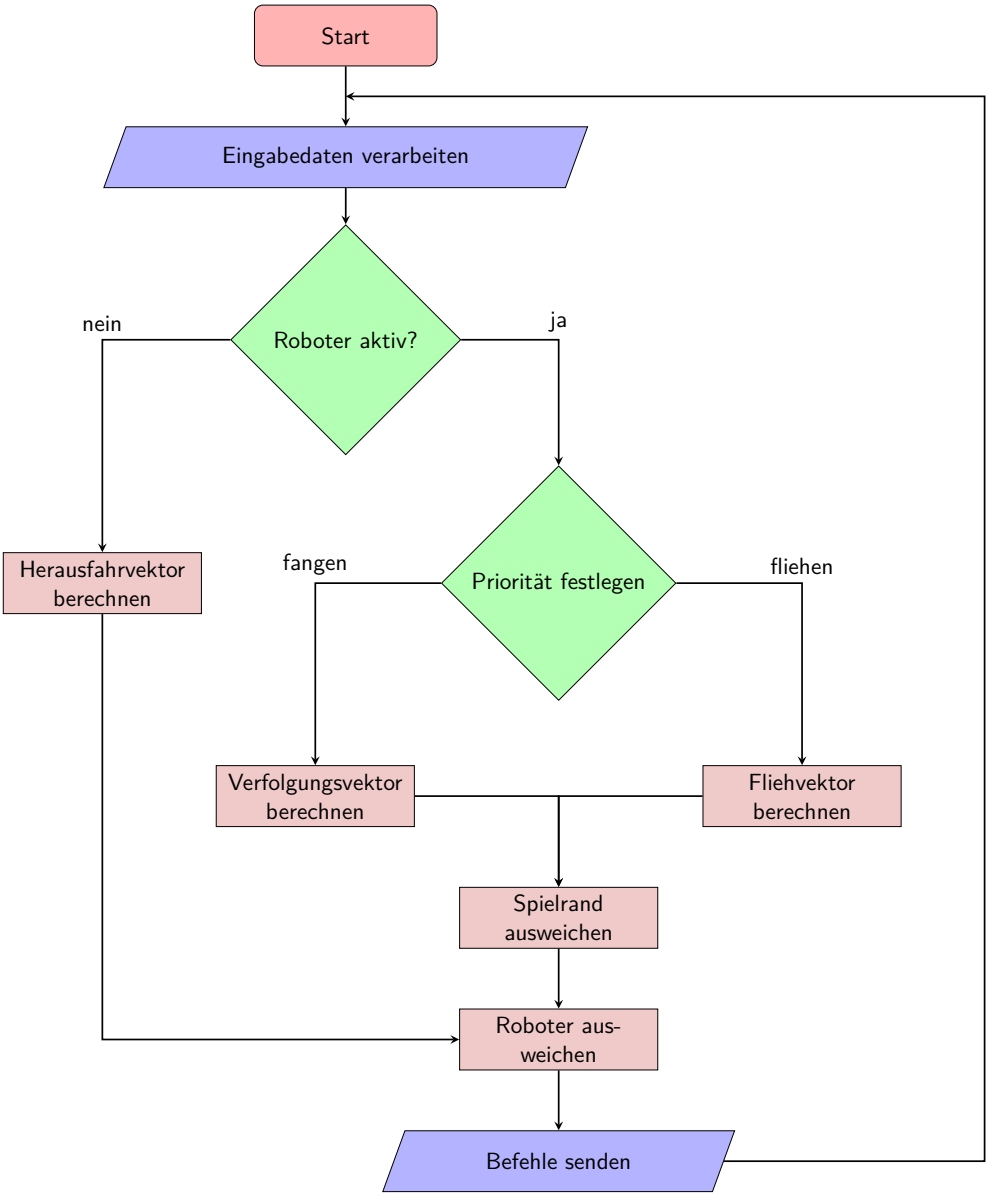
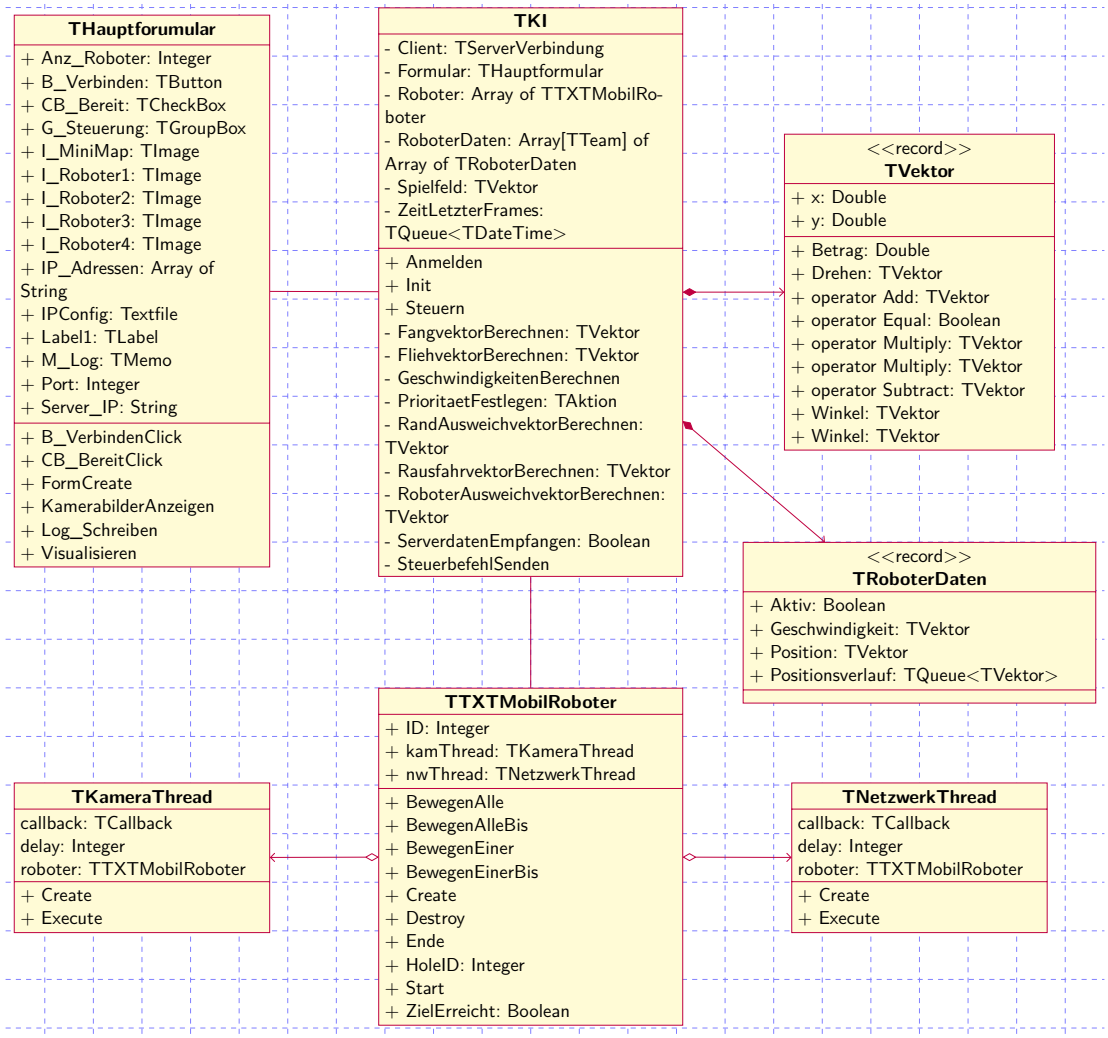
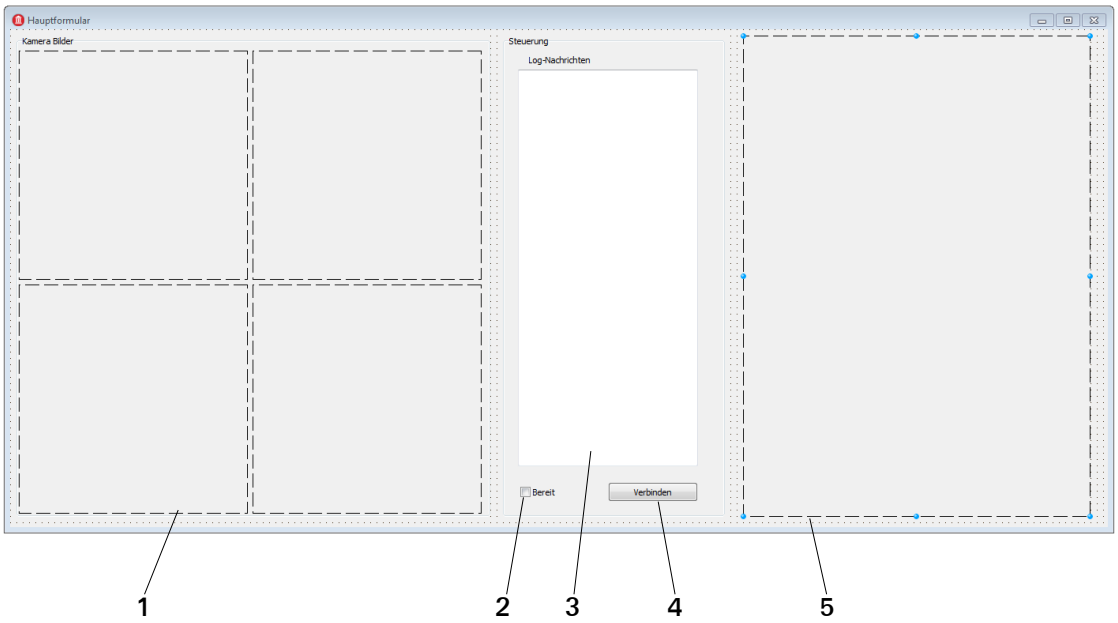


Abbildung 7: Programmablaufplan KI

6.2 Klassendiagramm



6.3 Hauptformular



1. Anzeige der Bilder, von den USB-Kameras der Roboter
2. Kontrollkästchen zum Signalisieren, dass die Roboter bereit sind
3. Anzeige von Fehler-, Hinweis- und Warnmeldungen
4. Schaltknopf zum Verbinden der Roboter mit dem Server
5. Grafische Anzeige der Roboterbewegungen

Prozeduren:

Ereignisprotokoll Hinweis-, Fehler- und Warnmeldungen werden in einer Ereignisprotokolldatei abgespeichert, sowie in einem Listenfeld **“3“** angezeigt.

KamerabilderAnzeigen Es werden die Bilder der USB-Kameras der Roboter in den Feldern **“1“** angezeigt.

Visualisierung In dem Feld **“5“** werden die Bewegungen unserer Roboter grafisch dargestellt.

FormCreate 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, dass für die spätere Verbindung zum Server benötigt wird.

6.4 Klassen

Für die Berechnungen und Logik haben wir eigene Klassen geschrieben.

Diese unterteilen sich in:

- mVektor
- mTKI
- mKonstanten
- mRoboterDaten

6.4.1 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.

Substract 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.

6.4.2 mTKI

Die Klasse mTKI hat einen Datentypen TAktion mit den Werten Fliehen und Fangen und eine abgeleitete Klasse TKI.
Die abgeleitete Klasse TKI besteht aus foglenden Funktionen, Prozeduren und Variablen:

Funktionen:

PrioritätFestlegen 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.

FangvektorBerechnen Es wird der Vektor zum nächsten gegnerischen Roboter, der gefangen werden soll, berechnet. Als Rückgabewert erhält man einen neuen Vektor.

FliehvektorBerechnen Es wird ein Vektor, mit Bezug auf den gegnerischen Roboter von dem geflohen werden soll, berechnet. Als Rückgabewert erhält man einen neuen Vektor.

RandAusweichvektorBerechnen Es wird überprüft, ob sich der Roboter im Spielfeld befindet, ist dieser außerhalb, so fährt dieser sofort ins 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.

RoboterAusweichvektorBerechnen 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.

RausfahrvektorBerechnen 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.

ServerdatenEmpfangen So bald sich der Client mit dem Server verbunden hat, werden die Variablen des Roboters mit Daten vom Server gefüllt.

Anmelden Ist eine Funktion um sich mit dem Server zu verbinden.

Prozeduren:

SteuerbefehlSenden 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.

GeschwindigkeitBerechnen Es wird von dem Vektor Geschwindigkeit, die Geschwindigkeit in $\frac{m}{s}$ berechnet.

Initialisierung 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.

Steuern Ist eine Funktion um sich mit dem Server zu verbinden.

Variablen:

- ZeitLetzterFrames
- RoboterDaten
- Roboter
- Spielfeld
- Client

6.4.3 mKonstanten

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

Variablen:

- Mindestabstand
- Nullvektor
- Rand
- Ausweichwinkel
- LängeFliehvektor

6.4.4 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

7 Arbeitspakete

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Projekt-beschreibung	1.1.1	Michael Mertens	

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Spielablauf	1.2.1	Eugen Zwetzich	

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Gantt-Chart	2.1.1	Eugen Zwetzich	

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Aufwands-schätzung	2.2.1	Eugen Zwetzich	Michael Mertens, Jonah Vennemann, Sven Stegemann

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Bedienungs-anleitung	4.1.1	Jonah Vennemann	Eugen Zwetzich

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Systemaufbau	5.1.1	Eugen Zwetzich	

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Dokumentation	.1.1	Eugen Zwetzich	

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Hauptformular:

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Ereignis-protokolldatei	6.3.1	Jonah Vennemann	Sven Stegemann

Ergebniss:
-

Aufgabenstellung:
Es werden Hinweis-, Fehler- und Warnmeldungen in einer Ereignisprotokolldatei abgespeichert und in einem Listenfeld **"3"** in der Grafischen Benutzeroberfläche(GUI) angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Kamerabilder Anzeigen	6.3.2	Sven Stegemann	Michael Mertens, Eugen Zwetzich

Ergebniss:
-

Aufgabenstellung:
Es werden die Bilder der USB-Kameras der Roboter in den Feldern **"1"** angezeigt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Visualisierung	6.3.3	Jonah Vennemann	

Ergebniss:
-

Aufgabenstellung:
In dem Feld **"5"**, werden die Bewegungen unserer Roboter grafisch dargestellt.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
FormCreate	6.3.4	Jonah Vennemann	

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, das für die spätere Verbindung mit dem Server benötigt wird.

mTKI:

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Prioritat Festlegen	6.4.1	Eugen Zwetzich	Sven Stegemann

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	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Fangvektor Berechnen	6.4.2	Michael Mertens	Eugen Zwetzich

Ergebniss:
-Vektor

Aufgabenstellung:
Es wird der Vektor zum nächsten gegnerischen Roboter, der Gefangen werden soll, berechnet. Als Rückgabewert erhält man einen neuen Vektor.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Fliehvektor Berechnen	6.4.3	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	6.4.4	Michael Mertens	Sven Stegemann

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 Ausweich-vektor Berechnen	6.4.5	Michael Mertens	Sven Stegemann

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	6.4.6	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	6.4.7	Jonah Vennemann	

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	6.4.8	Jonah Vennemann	

Ergebniss:
-

Aufgabenstellung:
Ist eine Funktion um sich mit dem Server zu verbinden.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Steuerbefehl Senden	6.4.9	Jonah Vennemann	

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	6.4.10	Michael Mertens	Sven Stegemann, Eugen Zwetzich

Ergebniss:
-

Aufgabenstellung:
Es wird von dem Vektor Geschwindigkeit, die Geschwindigkeit in $\frac{m}{s}$ berechnet.

Arbeitspaket	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Initialisierung	6.4.11	Jonah Vennemann	Eugen Zwetzich

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	**AP-Nr.:**	**Arbeitspaketverantwortlicher**	**Beteiligte Personen**
Steuern	6.4.12	Jonah Vennemann	

Ergebniss:
-

Aufgabenstellung:
Ist eine Funktion um sich mit dem Server zu verbinden.

8 Fazit

In diesem Projekt konnten wir einen guten Einblick in die Planung und die Programmierung eines IT-Projektes erhalten.

Durch die anfängliche Zeitverzögerung der Steuerklassen konnten wir nicht wie geplant mit dem Projekt starten, sondern mussten diesen um eine Woche verschieben.

Unter anderem unterschätzten wir am Anfang den Aufwand für die Programmierung der Künstlichen Intelligenz(KI). Durch diese 2 Probleme mussten wir die anfängliche Planung und das Gantt-Diagramm umstrukturieren.

Der Lernerfolg, den wir durch dieses Projekt erhielten. war sehr groß. Dadurch konnten wir einen praktischen Bezug mit den Vorlesungen in ITPM, Programmiersprachen II und der Mathematik verknüpfen.

Für die Programmierung der Bewegungen bzw. Fahrtrichtungen der Roboter haben wir uns der Vektorrechnung bedient. Diese haben wir in der Mathematik Vorlesung ausführlich kennengelernt und konnten diese für unser Projekt anwenden.

Außerdem konnten wir sehr vieles aus den Vorlesungen bzw. Praktika ITPM und Programmiersprachen II einsetzen. Z.B. die Kommunikation zwischen den Clients und dem Server mittels dem TCP/IP-Protokoll, Aufwandsschätzung anhand des Function-Point-Verfahrens, sowie die Planung mit Hilfe des Gantt-Charts uvm.

Listing 1: Klasse mVektor

```
1  //****************************************************************************
2  {*
3   * @file Vorlage_Unit.pas
4   * @author Eugen, Sven
5   * @date 05.01.2016
6   * @brief Dieses Modul stellt einen 2D-Vektortyp fuer
7   *  verschiedene Berechnungen zur Verfuegung
8   * @copyright Copyright 2016 Obi8, Nevsor
9   *
10  * @license
11  *
12  *  Diese Datei ist Teil von Roboter-Fangen.
13  *
14  *  Roboter-Fangen ist Freie Software: Sie koennen es unter den Bedingungen
15  *  der GNU General Public License, wie von der Free Software Foundation,
16  *  Version 3 der Lizenz oder (nach Ihrer Wahl) jeder spaeteren
17  *  veroeffentlichten Version, weiterverbreiten und/oder modifizieren.
18  *
19  *  Roboter-Fangen wird in der Hoffnung, dass es nuetzlich sein wird, aber
20  *  OHNE JEDE GEWAEHRLEISTUNG, bereitgestellt; sogar ohne die implizite
21  *  Gewaehrleistung der MARKTFAEHIGKEIT oder EIGNUNG FUER EINEN BESTIMMTEN ZWECK.
22  *  Siehe die GNU General Public License fuer weitere Details.
23  *
24  *  Sie sollten eine Kopie der GNU General Public License zusammen mit diesem
25  *  Programm erhalten haben. Wenn nicht, siehe <http://www.gnu.org/licenses/>.
26  *}
27 //****************************************************************************
28
29 unit mVektor;
30
31 interface
32
33 uses SysUtils, Math;
34
35 type TVektor = record
36     /// Die Komponenten des Vektors
37     x,y: Double;
38
39     /// Komponentenweise Addition zweier Vektoren
40     class operator Add(const Summand1, Summand2: TVektor): TVektor;
41
42     /// Komponentenweise Subtraktion zweier Vektoren
43     class operator Subtract(const Subtrahend, Minuend: TVektor): TVektor;
44
45     /// Komponentenweise Multiplikation eines Vektors mit einem Skalar
46     class operator Multiply(const Skalar:Double;
47       const Vektor:TVektor): TVektor; overload;
48     /// Komponentenweise Multiplikation eines Skalars mit einem Vektor
49     class operator Multiply(const Vektor:TVektor;
50       const Skalar:Double): TVektor; overload;
51     class operator Equal(const Vektor1, Vektor2: TVektor): Boolean;
52
53     /// Gibt den Winkel des Vektors zurueck bezogen auf die x-Achse
54     /// @return Winkel in Bogenmass im halboffenen Intervall [0;2*Pi)
55     /// @exception Es wird eine Exception ausgeloest,
56     /// wenn der Vektor gleich dem Nullvektor ist
57     function Winkel: Double; overload;
58     /// Gibt den Winkel des Vektors zurueck bezogen auf den Bezugsvektor
59     /// @return Winkel in Bogenmass im halboffenen Intervall [0;2*Pi)
60     /// @exception Es wird eine Exception ausgeloest, wenn der Vektor oder der
61     /// Bezugsvektor gleich dem Nullvektor ist
```

```
62      function Winkel(Bezugsvektor: TVektor): Double; overload;
63
64      /// Gibt die Laenge des Vektors zurueck (Euklidische Norm)
65      function Betrag: Double;
66
67      // Gibt den Vektor nach einer Drehung um einen Winkel zurueck
68      function Drehen(Winkel: Double): TVektor;
69
70  end;
71
72  implementation
73
74  { TVektor }
75
76  class operator TVektor.Add(const Summand1, Summand2: TVektor): TVektor;
77  begin
78    //Es werden die Komponenten der jeweiligen Vektoren addiert
79    //und anschliessend zurueckgegeben.
80    Result.x := Summand1.x + Summand2.x;
81    Result.y := Summand1.y + Summand2.y;
82  end;
83
84  class operator TVektor.Subtract(const Subtrahend, Minuend: TVektor): TVektor;
85  begin
86    //Es werden die Komponenten der jeweiligen Vektoren subtrahiert
87    //und anschliessend zurueckgegeben.
88    Result.x := Subtrahend.x - Minuend.x;
89    Result.y := Subtrahend.y - Minuend.y;
90  end;
91
92  class operator TVektor.Multiply(const Skalar: Double;
93    const Vektor: TVektor): TVektor;
94  begin
95    //Es werden die einzelnen Komponenten des Vektors mit einem Skalar
96    //multipliziert und zurueckgegeben
97    Result.x := Skalar * Vektor.x;
98    Result.y := Skalar * Vektor.y;
99  end;
100
101 function TVektor.Betrag: Double;
102 begin
103   Result := Sqrt(Sqr(x) + Sqr(y));
104 end;
105
106 function TVektor.Drehen(Winkel: Double): TVektor;
107 begin
108   //Mit der Drehmatrix wird ein neuer Vektor berechnet, der um einen als
109   //Parameter uebergebenen Winkel nach links(positiv) bzw.
110   //rechts(negativ) gedreht ist.
111   result.x :=  cos(Winkel)*self.x - sin(Winkel)*self.y;
112   result.y :=  sin(Winkel)*self.x + cos(Winkel)*self.y;
113 end;
114
115 class operator TVektor.Equal(const Vektor1, Vektor2: TVektor): Boolean;
116 begin
117   Result := (Vektor1.x = Vektor2.x) and (Vektor1.y = Vektor2.y);
118 end;
119
120 class operator TVektor.Multiply(const Vektor: TVektor;
121   const Skalar: Double): TVektor;
122 begin
123   //Es werden die einzelnen Komponenten des Vektors mit einem Skalar
```

```
124   //multipliziert und zurueckgegeben
125   Result.x := Skalar * Vektor.x;
126   Result.y := Skalar * Vektor.y;
127 end;
128
129 function TVektor.Winkel: Double;
130 begin
131   //Es wird der Winkel zwischen dem Vektor und der X-Achse berechnet und
132   //anschliessend zurueckgegeben.
133   if Self.x = 0.0 then // Wenn x = 0, kann arctan(Self.y/Self.x)
134                        // nicht berechnet werden.
135   begin
136     if Self.y > 0.0 then
137       Result := Pi * 0.5
138     else if y < 0.0 then
139       Result := Pi * 1.5
140     else
141       raise EMathError.Create('Winkel des Nullvektors'+
142       ' kann nicht berechnet werden.');
143   end
144   else
145   begin
146     Result := arctan(Self.y/Self.x); // Passt nur fuer den ersten Quadranten
147     if Self.x < 0 then
148       Result := Result + Pi // Fuer den zweiten und dritten Quadranten
149     else if Self.y < 0 then
150       Result := Result + Pi * 2; // Fuer den vierten Quadranten
151   end;
152 end;
153
154 function TVektor.Winkel(Bezugsvektor: TVektor): Double;
155 begin
156   Result := self.Winkel - Bezugsvektor.Winkel;
157   if Result < 0 then
158     Result := Result + 2*Pi;
159 end;
160
161 end.
```

Listing 2: Klasse mTKI

```
1  unit mTKI;
2
3  interface
4
5  uses mVektor,mTXTMobilRoboter,Client,ClientUndServer,DateUtils,mHauptformular,
6       mKonstanten, Math, Generics.Collections, mRoboterDaten, SysUtils, IdTCPClient;
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 TTXTMobilRoboter;
16     class var Spielfeld: TVektor;
17     class var Client: TServerVerbindung;
18
19     class function PrioritaetFestlegen(index: Integer;
20       out Ziel: Integer): TAktion;
21     class function FangvektorBerechnen(index, Ziel: Integer): TVektor;
```

```
22      class function FliehvektorBerechnen(index, Ziel: Integer): TVektor;
23      class function RandAusweichvektorBerechnen(index: Integer;
24        Zielvektor: TVektor): TVektor;
25      class function RoboterAusweichvektorBerechnen(index: Integer;
26        Zielvektor: TVektor): TVektor;
27      class function RausfahrvektorBerechnen(index: Integer): TVektor;
28      class procedure SteuerbefehlSenden(index: Integer; Zielvektor: TVektor);
29      class procedure GeschwindigkeitenBerechnen(zeit: TDateTime);
30      class function ServerdatenEmpfangen: Boolean;
31
32    public
33      class procedure Init(IP_Adressen: Array of String;
34        Server_Adresse: String; Port: Integer; Formular: THauptformular);
35      class procedure Steuern(Spielende: TDateTime);
36      class procedure Anmelden(Teamwahl: TTeam);
37  end;
38
39  implementation
40
41  { TKuenstlicheIntelligenz }
42
43  class procedure TKI.Anmelden(Teamwahl: TTeam);
44  begin
45    if Client.anmelden(Teamwahl) then
46      Formular.Log_Schreiben('Anmelden erfolgreich', Hinweis)
47    else
48      Formular.Log_Schreiben('Anmelden nicht erfolgreich', Fehler);
49
50  <<<<<<< HEAD
51  =======
52    while not Server.WarteAufSpielstart do
53    begin
54      if Server.WarteAufSpielstart then
55      begin
56        TKI.Steuern(Now); // TODO: Richtige Zeit einfuegen
    //!!!!!!!!!
57        break;
58      end;
59    end;
60
61  >>>>>>> refs/remotes/origin/master
62
63   if Client.WarteAufSpielstart then
64    TKI.Steuern(Now); // TODO: Richtige Zeit einfuegen
    //!!!!!!!!!
65  end;
66
67  class function TKI.RandAusweichvektorBerechnen(index: Integer;
68    Zielvektor: TVektor): TVektor;
69  var
70    ZielPosition, aktPos, Geschwindigkeit: TVektor;
71
72  begin
73    ZielPosition := RoboterDaten[TEAM_BLAU,index].Position + Zielvektor;
74    aktPos := RoboterDaten[TEAM_BLAU,index].Position;
75    Geschwindigkeit := RoboterDaten[TEAM_BLAU,index].Geschwindigkeit;
76
77    if Zielvektor = NULLVEKTOR then exit;
78
79    //Roboter befindet sich in der Naehe des Spielfeldrandes
80    //und darf nur in eine Richtung ablenken
81    if aktPos.x > Spielfeld.x-RAND then begin
```

```
82      if (Geschwindigkeit.x > 0) and (Zielvektor.x < 0) then
83        result := Geschwindigkeit.Drehen(-DegToRad(179))
84      else if (Geschwindigkeit.x < 0) and (Zielvektor.x < 0) then
85        result := Geschwindigkeit.Drehen(DegToRad(179));
86    end
87    else if aktPos.x < RAND then begin
88      if (Geschwindigkeit.x > 0) and (Zielvektor.x < 0) then
89        result := Geschwindigkeit.Drehen(-DegToRad(179))
90      else if (Geschwindigkeit.x < 0) and (Zielvektor.x < 0) then
91        result := Geschwindigkeit.Drehen(DegToRad(179));
92    end
93    else if aktPos.y > Spielfeld.y-RAND then begin
94      if (Geschwindigkeit.y > 0) and (Zielvektor.y < 0) then
95        result := Geschwindigkeit.Drehen(-DegToRad(179))
96      else if (Geschwindigkeit.y < 0) and (Zielvektor.y < 0) then
97        result := Geschwindigkeit.Drehen(DegToRad(179));
98    end
99    else if aktPos.y < RAND then begin
100     if (Geschwindigkeit.y > 0) and (Zielvektor.y < 0) then
101       result := Geschwindigkeit.Drehen(DegToRad(179))
102     else if (Geschwindigkeit.y < 0) and (Zielvektor.y < 0) then
103       result := Geschwindigkeit.Drehen(-DegToRad(179));
104   end;
105
106   //Roboter befindet sich ausserhalb des Spielfeldes
107   if (aktPos.x>Spielfeld.x) or (aktPos.x<0) or (aktPos.y<0) or
108     (aktPos.y>Spielfeld.y) then
109     result := Spielfeld*0.5 - aktPos
110   //Aus Ecke herausfahren
111   else if (Zielposition.x>Spielfeld.x) and (Zielposition.y>Spielfeld.y) or
112           (Zielposition.x>Spielfeld.x) and (Zielposition.y<0) or
113           (Zielposition.y>Spielfeld.y) and (Zielposition.x<0) or
114           (Zielposition.x<0) and (Zielposition.y<0) then
115   begin
116     result.x := -(Zielvektor.x);
117     result.y := -(Zielvektor.y);
118   end
119   //Oberen Spielfeldrand nicht ueberfahren
120   else if (Zielposition.y > Spielfeld.y) then begin
121     result.y := Spielfeld.y-aktPos.y;
122     result.x := Sign(Zielvektor.x) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.y));
123   end
124   //Unteren Spielfeldrand nicht ueberfahren
125   else if Zielposition.y < 0 then begin
126     result.y := -aktPos.y;
127     result.x := Sign(Zielvektor.x) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.y));
128   end
129   //Rechten Spielfeldrand nicht ueberfahren
130   else if (Zielposition.x > Spielfeld.x) then begin
131     result.x := Spielfeld.x-aktPos.x;
132     result.y := Sign(Zielvektor.y) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.x));
133   end
134   //Linken Spielfeldrand nicht ueberfahren
135   else if (Zielposition.x < 0) then begin
136     result.x := -aktPos.x;
137     result.y := Sign(Zielvektor.y) * Sqrt(Sqr(LAENGE_FLIEHVEKTOR)-Sqr(result.x));
138   end;
139
140 end;
141
142 class function TKI.RoboterAusweichvektorBerechnen(index: Integer;
143   Zielvektor: TVektor): TVektor;
```

```
144 var
145   t: Double;
146   i: Integer;
147   deltaP, deltaV: TVektor;
148   deltaWinkel: Double;
149   aktPos, Geschwindigkeit: TVektor;
150 begin
151   aktPos := RoboterDaten[TEAM_BLAU,index].Position;
152   Geschwindigkeit := RoboterDaten[TEAM_BLAU,index].Geschwindigkeit;
153   //Kollisionen mit TeamRobotern vermeiden
154   if index = High(RoboterDaten[TEAM_BLAU]) then Exit;
155 <<<<<<< HEAD
156   if Geschwindigkeit.Winkel(Zielvektor) > AUSWEICHWINKEL then Exit;
157 =======
158   if Geschwindigkeit.Winkel(vektor) > AUSWEICHWINKEL then Exit;
159
160 >>>>>>> refs/remotes/origin/master
161
162   for i := index+1 to High(RoboterDaten[TEAM_BLAU]) do begin
163     deltaP := aktPos - RoboterDaten[TEAM_BLAU,i].Position;
164     deltaV := Geschwindigkeit - RoboterDaten[TEAM_BLAU,i].Geschwindigkeit;
165     try
166       t := (deltaP.x*deltaV.x+deltaP.y*deltaV.y)/Power(deltaV.Betrag,2);
167     except
168       on EDivByZero do Continue; // Zu kleines deltaV => Roboter fahren parallel
169                                  // => kein Ausweichen noetig
170     end;
171
172     if (t>=0) and (t<5) then
173       if ((aktPos+t*Geschwindigkeit) -
174          (RoboterDaten[TEAM_BLAU,i].Position+t*
175          RoboterDaten[TEAM_BLAU,i].Geschwindigkeit)).Betrag< MINDESTABSTAND then
176       begin
177         deltaWinkel := RoboterDaten[TEAM_BLAU,i].Geschwindigkeit.winkel -
178         Geschwindigkeit.winkel;
179         if deltaWinkel < 0 then
180           deltaWinkel := deltaWinkel + 2*pi;
181         if deltaWinkel < Pi then begin
182           // Weiche nach rechts aus
183           result := Zielvektor.Drehen(-AUSWEICHWINKEL);
184         end
185         else begin
186           // Weiche nach links aus
187           result := Zielvektor.Drehen(AUSWEICHWINKEL);
188         end;
189       end;
190   end;
191 end;
192
193 class function TKI.FangvektorBerechnen(index,ziel: Integer): TVektor;
194 begin
195   result := RoboterDaten[TEAM_Rot,ziel].Position-
196   RoboterDaten[TEAM_BLAU,index].Position;
197 end;
198
199 class function TKI.FliehvektorBerechnen(index,ziel: Integer): TVektor;
200 begin
201   result := RoboterDaten[TEAM_BLAU,index].Position-
202   RoboterDaten[TEAM_rot,ziel].Position;
203   result := (LAENGE_FLIEHVEKTOR/result.Betrag)*result;
204 end;
205
```

```
206 class procedure TKI.GeschwindigkeitenBerechnen(zeit: TDateTime);
207 var
208   einRoboter: TRoboterDaten;
209   team: TTeam;
210   i: Integer;
211 begin
212   ZeitLetzterFrames.Enqueue(zeit);
213
214   for team in [TEAM_ROT, TEAM_BLAU] do
215   begin
216     for i := Low(RoboterDaten[team]) to High(RoboterDaten[team]) do
217     begin
218       einRoboter.Geschwindigkeit := (RoboterDaten[team,i].Position -
219       RoboterDaten[team,i].Positionsverlauf.Dequeue)*
220       (1/SecondSpan(zeit, ZeitLetzterFrames.Dequeue));
221     end;
222   end;
223 end;
224
225
226 class procedure TKI.Init(IP_Adressen: Array of String;
227   Server_Adresse: String; Port: Integer; Formular: THauptformular);
228 var i: Integer;
229 begin
230   Client:= TServerVerbindung.create(Server_Adresse, Port);
231
232   self.Formular:=Formular;
233
234   setlength(Roboter, Length(IP_Adressen));
235   for i:= Low(Roboter) to High(Roboter) do
236   begin
237     try
238         Roboter[i]:=TTXTMobilRoboter.Create(Ip_Adressen[i], i);
239         Roboter[i].Start;
240         Formular.Log_Schreiben('Verbindung zum Server Erfolgreich', Hinweis);
241     except
242         formular.Log_Schreiben('Verbindung nicht moeglich', Fehler);
243     end;
244   end;
245 end;
246
247 class function TKI.PrioritaetFestlegen(index: Integer;
248   out ziel: Integer): TAktion;
249 var DeltaVektor: TRoboterDaten;
250     KleinsterAbstand,Abstand: Double;
251     i,NaechsterRoboter: Integer;
252 begin
253   NaechsterRoboter := 0;
254   KleinsterAbstand := (RoboterDaten[TEAM_BLAU,index].Position -
255                        RoboterDaten[TEAM_ROT,0].Position).Betrag;
256
257   //Pruefung welcher Roboter vom Team Rot am
258   //naehesten am Roboter vom Team Blau ist
259   for i := Low(RoboterDaten[TEAM_ROT])+1 to High(RoboterDaten[TEAM_ROT]) do
260   begin
261     if RoboterDaten[TEAM_ROT,i].Aktiv then
262     begin
263       Abstand := (RoboterDaten[TEAM_BLAU,index].Position -
264                   RoboterDaten[TEAM_ROT,i].Position).Betrag;
265       if Abstand < KleinsterAbstand then
266          begin
267            KleinsterAbstand := Abstand;
```

```
268             NaechsterRoboter := i;
269           end;
270       end;
271     end;
272
273     ziel := NaechsterRoboter;
274
275     //Pruefung ob der Roboter von Team Rot sich vor oder hinter dem Roboter von
276     //Team Blau befindet
277     Try
278     if InRange((RoboterDaten[TEAM_BLAU,index].Position -
279       RoboterDaten[TEAM_ROT,NaechsterRoboter].Position)
280       .Winkel(RoboterDaten[TEAM_BLAU,index].Geschwindigkeit), pi*0.5, pi*1.5) then
281       Result := FLIEHEN
282     else
283       Result := FANGEN;
284     Except on EMathError do
285       Formular.Log_Schreiben('Zwei Roboter haben die gleiche '+
286       'Position oder ein eigener Roboter steht still.', Warnung);
287     End;
288   end;
289
290
291   class function TKI.RausfahrvektorBerechnen(
292     index: Integer): TVektor;
293   var Position: TVektor;
294       Abstand: Array[0..3] of Double;
295       KAbstand: Double;
296   begin
297     Position := RoboterDaten[TEAM_BLAU,index].Position;
298     //Berechnung der Seitenabstaende mit der Annahame,
299     //dass sich unten links der Koordinatenursprung befindet.
300     Abstand[0] := Position.x;                //links
301     Abstand[1] := Spielfeld.x-Position.x;    //rechts
302     Abstand[2] := Spielfeld.y-Position.y;    //oben
303     Abstand[3] := Position.y;                //unten
304     KAbstand := MinValue(Abstand);
305     //in x- oder y-Richtung herausfahren
306     if (KAbstand=Abstand[0]) or (KAbstand=Abstand[1]) then begin
307       result.x := -KAbstand;
308       result.y := 0;
309     end
310     else begin
311       result.x := 0;
312       result.y := -KAbstand;
313     end
314   end;
315
316   class function TKI.ServerdatenEmpfangen: Boolean;
317   var
318     i: Integer;
319     Team: TTeam;
320     Serverdaten: TSpielstatus;
321   begin
322     Serverdaten:= Client.StatusEmpfangen;
323     for Team in [Team_Blau,Team_Rot] do
324     begin
325       for i := low(Roboterdaten[Team]) to High(Roboterdaten[Team]) do
326       begin
327         // Roboterdaten des Servers werden in eingene Arrays usw. verpackt
328         Roboterdaten[Team,i].Position.x:=Serverdaten.Roboterpositionen[Team,i].x;
329         Roboterdaten[Team,i].Position.y:=Serverdaten.Roboterpositionen[Team,i].y;
```

```
330         Roboterdaten[Team,i].Aktiv:=Serverdaten.RoboterIstAktiv[Team,i];
331
332         Roboterdaten[Team,i].Positionsverlauf.
333         Enqueue(Roboterdaten[Team,i].Position);
334
335         GeschwindigkeitenBerechnen(Serverdaten.Zeit);
336       end;
337     end;
338   end;
339
340   class procedure TKI.SteuerbefehlSenden(index: Integer; Zielvektor: TVektor);
341   var
342    Roboter_Blau: TTXTMobilRoboter;
343    akt_Vektor: tVektor;
344
345   const
346     Geschwindigkeit= 512;   // Standartgeschwindigkeit der Roboter
347     c_Radius = 2;          //Konstante zum dehen, auf Grad bezogen
348
349   begin
350     Roboter_Blau:= Roboter[Index];
351     akt_Vektor:=Roboterdaten[Team_Blau,Index].Geschwindigkeit;
352
353     if not((akt_Vektor=NULLVEKTOR) or (Zielvektor=NULLVEKTOR)) then
354     begin
355       // Liegt der neue Vekter links oder rechts des Roboters
356       if Zielvektor.Winkel(akt_Vektor)<pi then
357       Roboter_Blau.Bewegenalle(Geschwindigkeit,
358       Geschwindigkeit- round(c_Radius*RadToDeg(Zielvektor.Winkel(akt_Vektor))))
359       // Der Kurvenradius haengt vom Winkel der 2 Vektoren ab, je groesser der
360       // Winkel desto groesser der Radius
361       else
362       Roboter_Blau.Bewegenalle(Geschwindigkeit-
363       round(c_Radius*RadToDeg(Zielvektor.Winkel(akt_Vektor))),Geschwindigkeit)
364     end;
365   end;
366
367
368   class procedure TKI.Steuern(spielende: TDateTime);
369   var einRoboter: integer;
370   Ziel_R: Integer;
371   FahrVektor: TVektor;
372   Aktion: TAktion;
373   begin
374     // Startaufstellung einnehmen
375     // Queues fuellen
376
377     while True do // Andere Bedingung
378     begin
379       ServerdatenEmpfangen;
380       // Gewindigkeiten werden in Serverdaten Empfangen Berechnet
381
382       //Fuer jeden Roboter wird ein Vektor festergelegt, den er entlang fahren soll
383       for einRoboter := low(Roboter)to High(Roboter) do
384       begin
385
386
387         if Roboterdaten[TEAM_BLAU,einRoboter].Aktiv then
388         begin
389           // soll der Roboter fliehen oder fangen
390           Aktion:=PrioritaetFestlegen(einRoboter,Ziel_R);
391           // Abfrage ob der Roboter gefangen wurden
```

```
392           if Roboter[einRoboter].LiesDigital(5) or Roboter[einRoboter].LiesDigital(6) then
393           begin
394             Client.GefangenMelden(einRoboter);
395             Fahrvektor:=RausfahrvektorBerechnen(einRoboter);
396             Formular.Log_Schreiben('Roboter: '+ Inttostr(einRoboter) + ' wurde gefangen', Warnung);
397           end
398           //Wenn der Roboter nicht gefangen wurde
399           else
400           begin
401             // flieht er
402             if Aktion=FLIEHEN then
403             begin
404               FahrVektor:= FliehvektorBerechnen(einRoboter,Ziel_R);
405               FahrVektor:= RandAusweichVektorBerechnen(einRoboter,FahrVektor
406               );
407               Formular.Log_Schreiben('Roboter: '+ Inttostr(einRoboter) + ' flieht', Hinweis)
408             end
409             // oder faengt er
410             else if Aktion=Fangen then
411             begin
412               FahrVektor:= FangvektorBerechnen(einRoboter,Ziel_R);
413               FahrVektor:= RandAusweichVektorBerechnen(einRoboter,FahrVektor);
414               Formular.Log_Schreiben('Roboter: '+ Inttostr(einRoboter) + ' verfolgt', Hinweis)
415             end;
416           end;
417         end
418         else
419         begin
420           FahrVektor:=RausFahrvektorBerechnen(einRoboter);
421         end;
422         // Ueberpruefungsmechanismus auf Kreuzverkehr oder Spielfeldgrenzen
423         FahrVektor:= RoboterAusweichVektorBerechnen(einRoboter, FahrVektor);
424         // Befehele werden an dern Roboter gesendet
425         SteuerbefehlSenden(einRoboter ,FahrVektor);
426         Formular.Visualisieren(RoboterDaten, Fahrvektor);
427       end;
428     end;
429   end;
430
431   end.
```

Listing 3: Klasse mKonstanten

```
unit mKonstanten;

interface

uses mVektor, Math;

const MINDESTABSTAND = 0.2;
      AUSWEICHWINKEL = 20*pi/180;
      LAENGE_FLIEHVEKTOR = 0.3;
      RAND = 0.3;
      NULLVEKTOR: TVektor = (x: 0; y:0);

implementation

end.
```

Listing 4: Klasse mRoboterDaten

```
unit mRoboterDaten;

interface

  uses mVektor, Generics.Collections;

  type
    TRoboterDaten = record
      Position: TVektor;
      Geschwindigkeit: TVektor;
      Positionsverlauf: TQueue<TVektor>;
      Aktiv: Boolean;
    end;

implementation

end.
```