Technische Universität Braunschweig

# RoboSoccer - Road to France...

## LeGoal

Software-Entwicklungspraktikum (SEP)
Sommersemester 2016

## Technischer Entwurf


Auftraggeber
Technische Universität Braunschweig
Institut für Programmierung und Reaktive Systeme
Prof. Dr. Ursula Goltz
Mühlenpfordtstrasse 23
38106 Braunschweig

Betreuer: Sascha Lity, Ken Rieke

Auftragnehmer:

| Name | E-Mail-Adresse |
|---|---|
| Sören Christmann | s.christmann@tu-braunschweig.de |
| Alexander Joost | a.joost@tu-braunschweig.de |
| Mohamad Karaki | mo.karaki@tu-braunschweig.de |
| Bengt Kensy | b.kensy@tu-braunschweig.de |
| Anna Lörke | a.loerke@tu-braunschweig.de |


Braunschweig, 29. Juni 2016

# Bearbeiterübersicht

| **Kapitel** | **Autoren** |
|---|---|
| 1 | Mohamad Karaki |
| 2 | Anna Lörke<br>Alexander Joost |
| 3 | Alexander Joost |
| 3.1 | Alexander Joost |
| 3.2 | Anna Lörke |
| 3.3 | Sören Christmann<br>Bengt Kensy |
| 4 | Sören Christmann<br>Bengt Kensy |
| 5 | Mohamad Karaki |
| 6 | Sören Christmann |
| 7 | Bengt Kensy |
| 8 | Anna Lörke |
| 9 | Alexander Joost |
| 10 | alle |

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

Der Technische Entwurf dient als Fundament für die Implementierung unserer Software „LeGoal“. Die Entwicklung richtet sich daher nach den Inhalten dieses Dokumentes. In den folgenden Kapiteln werden die klassischen Entwurfsentscheidungen genauer erläutert.

Das Ziel unseres Projekts ist die Entwicklung einer künstlichen Intelligenz für die „Lego Mindstorms EV3 - Roboter“, deren Aufgabe es ist, ein Fußballspiel gegen ein anderes Team zu gestalten. Dabei sollen die Roboter miteinander über Raspberry Pis kommunizieren und dadurch gemeinsam Entscheidungen für eine Spielstrategie anhand der Spielsituation treffen. Diese wird anhand der Positionsdaten der Roboter und des Balls erkannt. Diese Positionsdaten werden an die Raspberry Pis mittels einer Kamera, welche sowohl die Roboter als auch den Ball erkennt, übermittelt. Zusätzlich steht eine graphische Benutzeroberfläche (GUI) zur Verfügung, welche dem Benutzer zum einen eine Visualisierung des Spielfeldes, zum anderen auch die Steuerung des Spiels ermöglicht. Daneben bietet die GUI die Möglichkeit das Spiel manuell zu steuern.

Die Roboter agieren dabei nach im Vorfeld festgelegten Regeln. Diese Regeln werden im ersten Abschnitt dieses Kapitels erläutert.

Zusätzlich zu der Funktionsarchitektur unserer Software wird in diesem Dokument die Komponentenarchitektur der Software beschrieben. Diese wird im dritten Kapitel beschrieben und durch ein Komponentendiagramm visualisiert. Daneben wird die Kommunikation zwischen diesen Komponenten erläutert. Außerdem wird die korrekte Verwendung der Komponenten durch Verhaltensdiagramme genauer dargestellt. Letztlich wird für den Grobentwurf ein Verteilungsdiagramm im vierten Kapitel zur Verfügung gestellt. Dieser Grobentwurf lässt sich im Feinentwurf darstellen. Der Feinentwurf wird im fünften Kapitel erläutert und durch Paket-/Klassendiagramme dargestellt. Im weiteren Verlauf werden die Änderungen gegenüber des Fachentwurfs erörtert. Abschließend wird erneut auf die Muss-, Soll- und Kannkriterien zurückgegriffen und diese mit den Komponenten des dritten Kapitels in Verbindung gesetzt.

Zuallererst ist es am wichtigsten, dem Leser eine Übersicht des Systems zu geben. Dies ermöglicht das folgende Aktivitätsdiagramm:

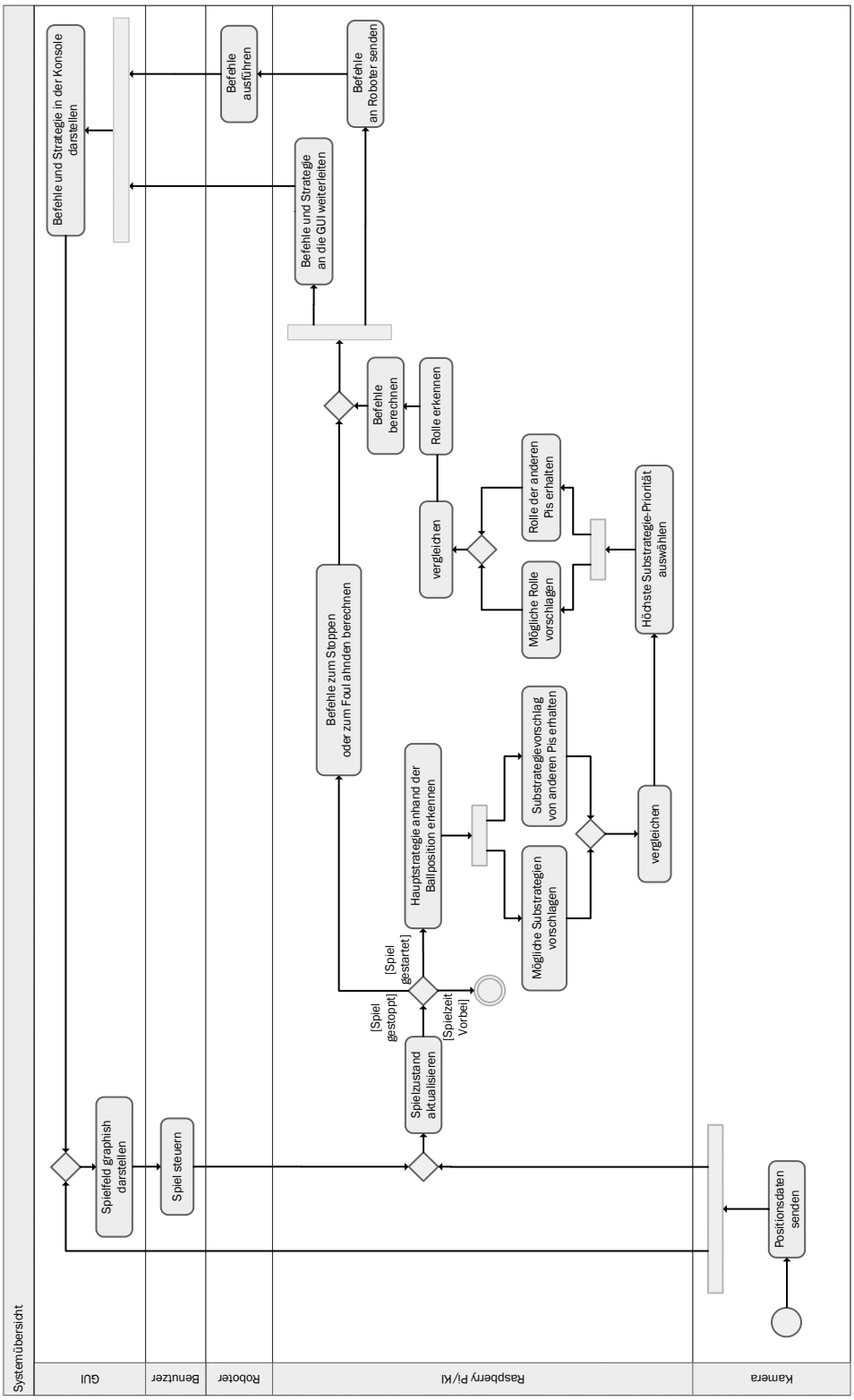


Abbildung 1.1: Aktivitätsdiagramm: *Systemübersicht*

**Beschreibung zum Aktivitätsdiagramm *Systemübersicht*:**

Dieses Aktivitätsdiagramm zeigt eine Übersicht der Funktionen des Systems, die während einer Runde durchgeführt werden. Zum Start schickt die Kamera die Positionsdaten an die GUI und an den Raspberry Pis. In der GUI wird das Spielfeld visualisiert. Danach kann der Benutzer das Spiel steuern. Bei der Steuerung des Spiels können mehrere Befehle an die Raspberry Pis gesendet werden. Diese sind „Starten“, „Stoppen“, „Foul ahnden“ und „Spiel neu starten“, welche genauer im Aktivitätsdiagramm *LeGoal Benutzeroberfläche* im Pflichtenheft erklärt wurden. Auf alle Befehle reagieren die Pis und sammeln dazu die Kameradaten. Anhand dieser Daten wird der Spielzustand aktualisiert. Wenn das Spiel gestartet ist, entscheidet sich die KI anhand der Ballposition für eine der drei Hauptstrategien, die wir in Abschnitt „Die künstliche Intelligenz“ erläutern werden. Danach schlägt der Raspberry Pi eine Substrategie vor und bekommt gleichzeitig Substrategievorschläge von den anderen Pis. Darüber hinaus werden Substrategievorschläge verglichen und sich für die Substrategie mit der höchsten Priorität entschieden. Nach einer Substrategieentscheidung müssen die Rollen der Roboter verteilt werden. Hierfür schlägt auch der Raspberry Pi eine Rolle vor und erhält dazu die Rollenvorschläge von den anderen Pis. Diese Rollenvorschläge werden verglichen, danach erkennt der Pi seine Rolle. Anhand dieser Analyse werden die Befehle berechnet und anschließend an den Roboter gesendet. Nachdem der Roboter die Befehle erhält, führt er diese aus. Die berechneten Befehle werden an die GUI für eine Darstellung der ausgeführten Befehle weitergeleitet. Anschließend werden die Befehle an die Roboter gesendet und von den Robotern ausgeführt. Nach der Ausführung und der Darstellung der Befehle startet eine neue Runde.

## 1.1 Projektdetails

Im folgenden Abschnitt werden die 16 mit dem Gegnerteam vereinbarten Spielregeln näher erläutert. Sie sollen dafür sorgen, dass beide Teams am Ende ein gemeinsames Spiel abhalten können.

### 1.1.1 Regeln

1. Das Spiel besteht aus zwei Halbzeiten. Jede Halbzeit dauert fünf Minuten.
2. Ein Zufallsgenerator entscheidet über die Startseiten der beiden Teams.
3. Die Roboter beider Teams stehen auf den im Vorfeld abgesprochenen, einheitlichen Startpositionen.
4. Für jeden Anstoß gibt es ein Vorrangteam: Hat ein Team Vorrang, so darf dieses sich mit einem Roboter innerhalb der ersten fünf Sekunden nach Spielbeginn frei bewegen. Alle anderen Roboter bleiben auf ihren jeweiligen Startpositionen. Nach Ablauf der fünf Sekunden dürfen sich alle Roboter frei bewegen. Die Sekundenzahl kann vor jedem Spiel in Absprache mit dem Gegnerteam verändert werden.
5. Zum Start entscheidet ein Zufallsgenerator welches der beiden Teams Vorrang hat.
6. Nach einem erfolgreichen Torschuss eines Teams wird dieser vom Schiedsrichter gewertet. Die Roboter setzen das Spiel an ihren jeweiligen Startpositionen fort, wobei das entsprechend andere Team Vorrang erhält.
7. Nach der ersten Halbzeit werden die Seiten gewechselt.
8. Ein Austausch der Roboter während des Spiels ist untersagt.
9. Alle Roboter müssen zeitgleich starten. Ausnahme ist hierbei ein Roboter des Vorrangteams, welcher bereits fünf Sekunden vor allen anderen startet.
10. Das Team, welches am Ende des Spiels mehr Tore erzielt hat, gewinnt.
11. Sollte am Ende des Spiels ein Gleichstand der Punkte herrschen, so ist dies ein gültiges Ergebnis.
12. Ein Foul kann nur dann entstehen, wenn sich zwei gegnerische Roboter rammen.
13. Sollte ein Roboter ein Foul ausüben, so sorgt der im Vorfeld von beiden Teams aus festgelegte Schiedsrichter für einen erneuten Spielstart. Hierbei erhält das gefoulte Team Vorrang.
14. Sollte der Foul-Verursacher nicht eindeutig zu erkennen sein, so wird das Vorrangteam für den erneuten Spielstart durch einen Zufallsgenerator bestimmt.

15. Gelangt der Ball durch die Aktionen eines Roboters in einen nicht erreichbaren Bereich, so müssen alle Roboter an ihre Startposition zurückkehren, sodass ein erneuter Anstoß- durchgeführt werden kann. Vorrang hat hierbei das entsprechend andere Team.

16. Eine Manipulation des Spielfeldes, des Balls, der Roboter oder sonstigen zum Spiel gehörigen Dingen ist untersagt.

## 1.2 Vorgänge zum Starten des Spiels

In diesem Abschnitt werden die Vorgänge zum Starten des Spiels erläutert.

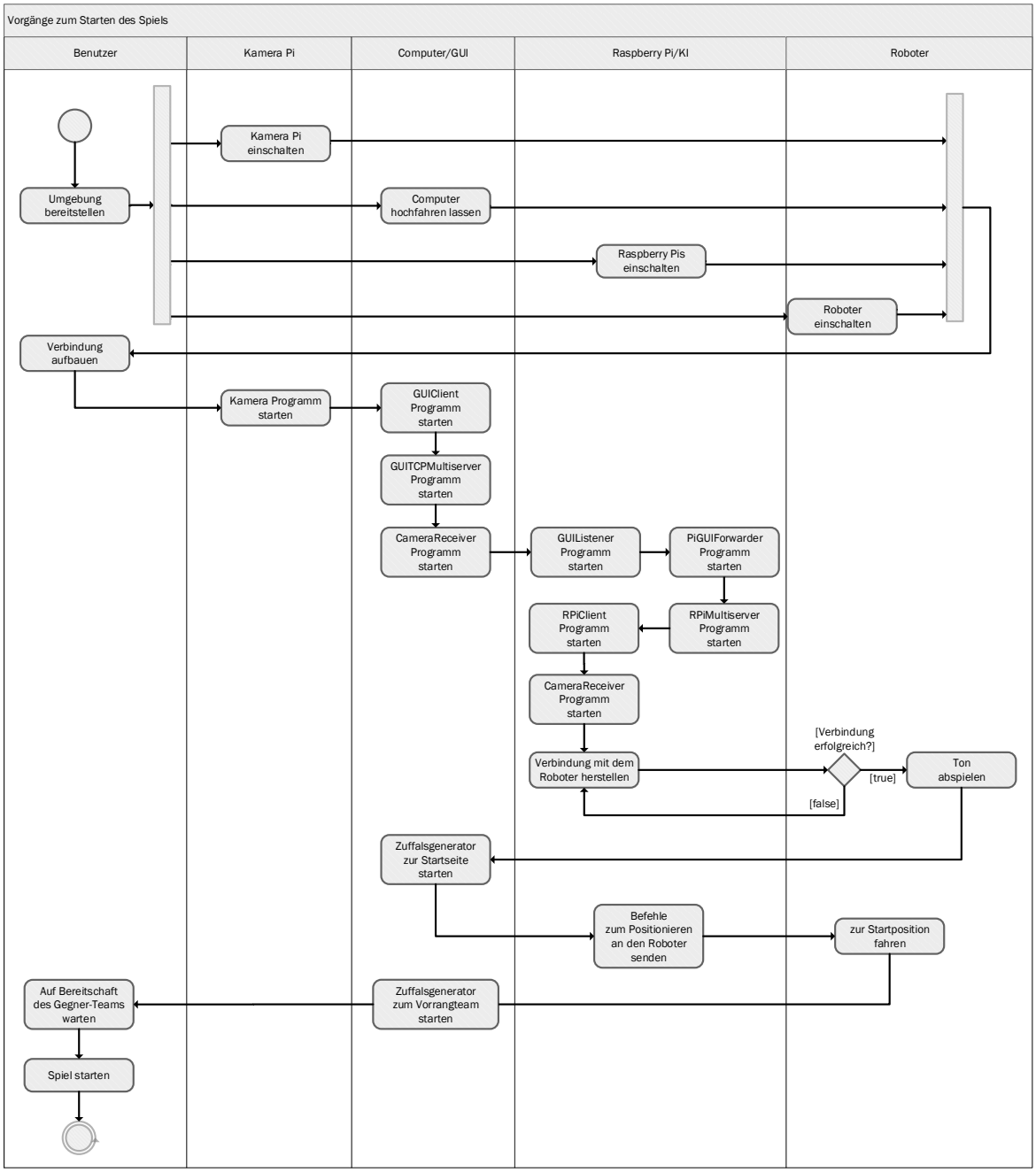


Abbildung 1.2: Aktivitätsdiagramm: *Vorgänge zum Starten des Spiels*

**Beschreibung zum Aktivitätsdiagramm *Vorgänge zum Starten des Spiels*:**

Dieses Aktivitätsdiagramm zeigt die Vorgänge, die vom Benutzer vor dem Spielbeginn durchgeführt werden müssen. In diesem Diagramm gibt es fünf Funktionen. Jede Funktion ist ein Teil des Systems. Um das Spiel zu starten, muss der Benutzer die Spielumgebung bereitstellen. Hierfür muss er den Kamera Pi, den Computer (GUI) und die Raspberry Pis sukzessive hochfahren lassen. Zuletzt müssen alle drei Roboter, die zu unserem Team gehören, gestartet werden. Nachdem alle Systemkomponenten gestartet sind, muss der Benutzer die zugehörige Software ausführen. Um die Kameradaten verteilen zu können, muss im Raspberry Pi der Kamera das Kameraprogramm gestartet werden. Daneben muss im Computer die GUI gestartet werden und die Verbindung mit den Raspberry Pis aufgebaut werden. Die Verbindung erfolgt dadurch, dass das PCClient Programm und das CameraReceiver Programm, deren Funktionen im Kapitel 7 erklärt sind, gestartet werden. Danach müssen in den Raspberry Pis die Programme RaspPiClient, RaspPiServer und CameraReceiver, deren Funktionen auch in Kapitel 7 erklärt sind, gestartet werden. Darüber hinaus soll die Verbindung zwischen den Pis und den zugehörigen Robotern durch ein Programm aus der Lejos-EV3-Bibliothek aufgebaut werden. Bei einer erfolgreichen Verbindung spielt der Roboter einen Ton ab. Zuletzt hat der Benutzer die Möglichkeit, das Spiel zu starten, aber er muss dafür auf die Bereitschaft des Gegner-Teams warten. Nach dem Spielstart können die Raspberry Pis durch die integrierte KI Befehle berechnen, welche an die Roboter gesendet und von den Robotern ausgeführt werden.

## 1.3 Vorgänge zum manuellen Steuern des Spiels

In diesem Abschnitt werden die Vorgänge zum manuellen Steuern des Spiels erläutert.

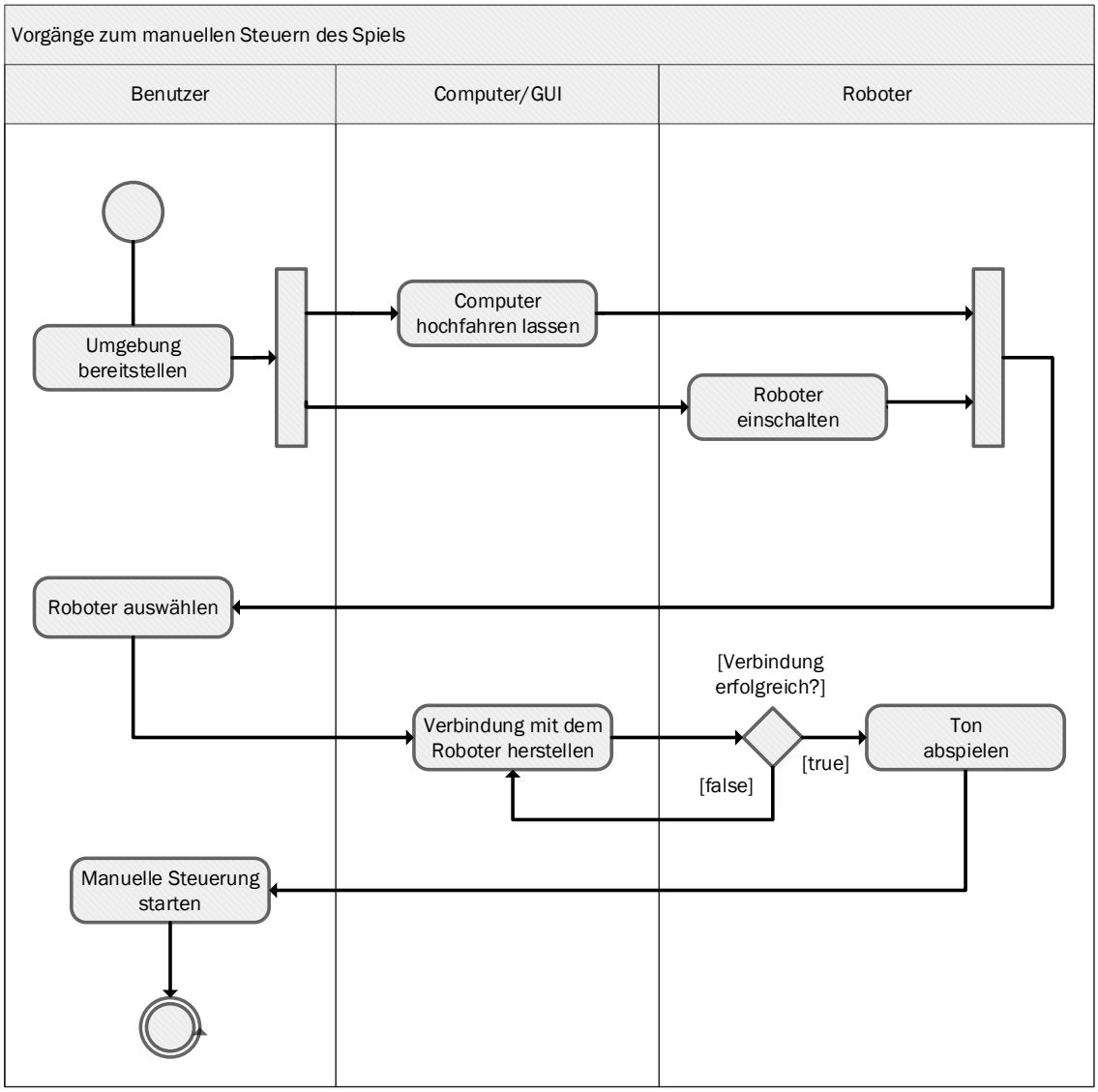


Abbildung 1.3: Aktivitätsdiagramm: *Vorgänge zum manuellen Steuern des Spiels*

**Beschreibung zum Diagramm *Vorgänge zum manuellen Steuern des Spiels*:**

Dieses Aktivitätsdiagramm zeigt die Vorgänge, die vom Benutzer vor dem Start der manuellen Steuerung durchgeführt werden müssen. In diesem Diagramm gibt es drei Funktionen. Jede Funktion ist ein Teil des Systems. Um die manuelle Steuerung zu starten, muss der Benutzer die Spielumgebung bereitstellen. Hierfür muss er den Computer und die Roboter hochfahren lassen. Nachdem der Computer und die Roboter gestartet sind, kann der Benutzer genau einen Roboter gleichzeitig steuern. Hierfür muss er einen Roboter auswählen. Außerdem soll die Verbindung zwischen dem Computer und dem Roboter durch ein Programm aus der Lejos-EV3-Bibliothek aufgebaut werden. Bei einer erfolgreichen Verbindung spielt der Roboter einen Ton ab. Zuletzt kann der Benutzer den Roboter manuell steuern und mithilfe der Tastaturtasten Befehle senden.

## 1.4 Die künstliche Intelligenz

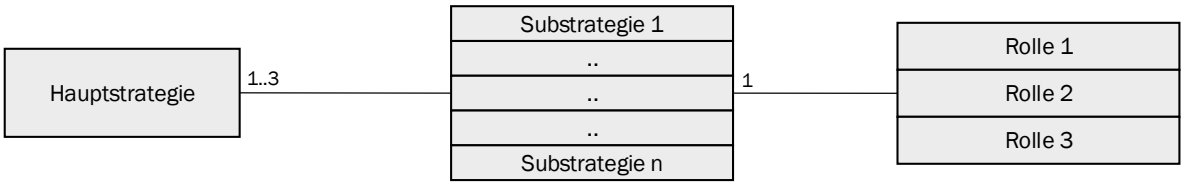


Abbildung 1.4: *Spielstrategie*

Dieses Diagramm zeigt, wie sich die künstliche Intelligenz der Raspberry Pis für eine Strategie entscheidet. Um die Künstliche Intelligenz zu entwickeln, haben wir uns für eine effiziente Methode zum Wechseln der Strategien anhand der Spielsituation entschieden. Es wird in unserem Spiel drei Hauptstrategien geben. Diese sind „ZUM BALL GEHEN“, „ANGRIFF“ und „ABWEHR“. Diese Strategien sind lediglich Hauptstrategien. Es wird sich anhand der Ballposition für eine von diesen entschieden. Die Funktionen der Hauptstrategien sind im Aktivitätsdiagramm *Roboter KI* visualisiert. Für jede Hauptstrategie wird es mehrere Substrategien geben. Die Substrategien haben Prioritäten. Es wird sich gemeinsam im Team anhand der Roboterposition für eine Substrategie mit der höchsten Priorität entschieden. Folgendes Szenario beschreibt wie alle drei Roboter ihre möglichen Substrategien vorschlagen und sich für die Substrategie mit der höchsten Priorität entscheiden. S3 hat in diesem Fall die höchste Priorität.

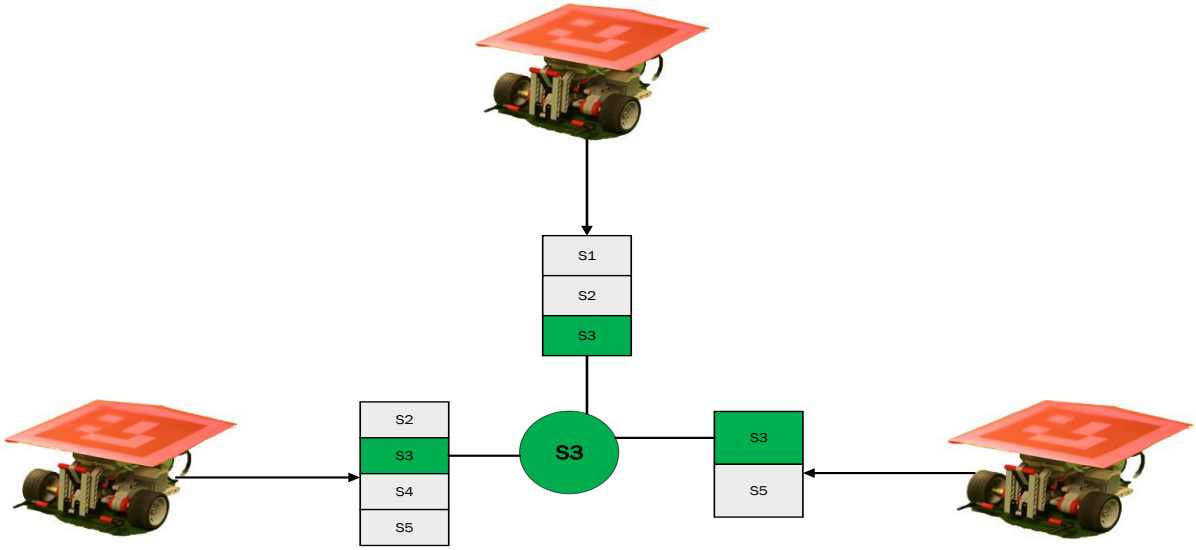


Abbildung 1.5: *Beispielszenario*

Die Prioritäten der Substrategien werden von unserer Gruppe festgelegt und dementsprechend implementiert. Außerdem hat jeder Roboter eine Rolle in der Substrategie. Die Rolle eines Roboters wird auch im Team bestimmt.

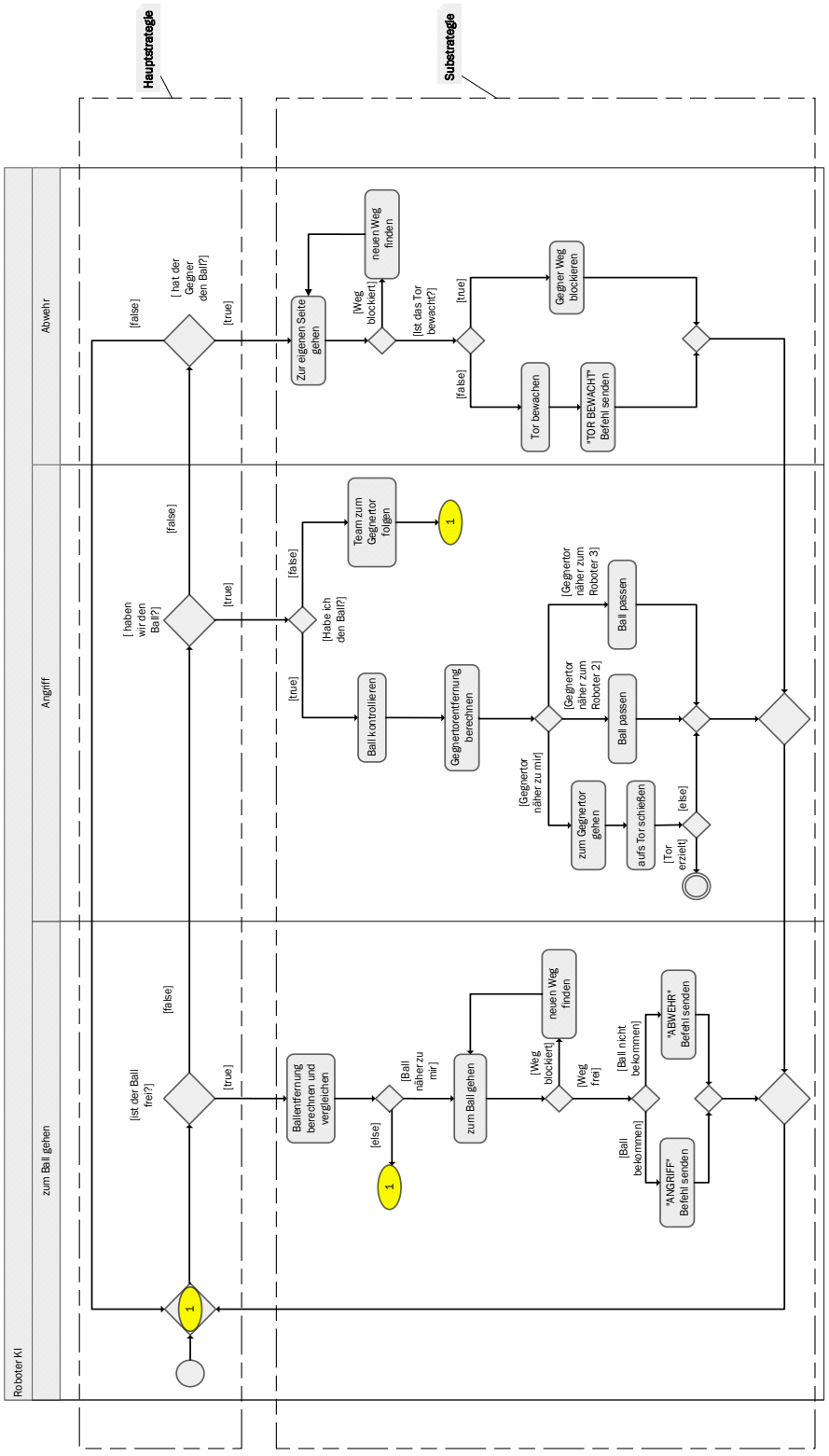


Abbildung 1.6: Aktivitätsdiagramm: *Roboter KI*

**Die 1 Markierung hat die Aufgabe, die letzte Aktivität mit dem markierten Mergenode zu verbinden.**

**Beschreibung zum Aktivitätsdiagramm *Roboter KI*:**

Dieses Aktivitätsdiagramm beschreibt die drei Hauptstrategien. Alle Strategien werden anhand der Ballposition im Spielfeld durchgeführt. Außerdem wird bei jeder Hauptstrategie eine Substrategie beschrieben. Bei einem freien Ball im Spielfeld liegt der Fokus der Roboter daran, den Ball zu kriegen. Deswegen müssen die Roboter zum Ball fahren. Allerdings wird sich nur ein Roboter gleichzeitig zum Ball bewegen, um Konflikte zu vermeiden. Beim Erlangen des Balls, sendet der Roboter den „ANGRIFF“ Befehl an die anderen Roboter. Beim missglückten Versuch den Ball zu erlangen, sendet der Roboter den „ABWEHR“ Befehl an die anderen Roboter. Beim Angriff, muss ein Roboter von unserem Team den Ball haben. Hierfür dürfen alle Roboter gleichzeitig in den gegnerischen Bereich des Spielfeldes fahren. Beim Verlieren des Balls oder wenn der Ball vom anderen Team besetzt ist, müssen die Roboter zum eigenen Bereich zurückfahren und im Abwehrzustand stehen. Daneben muss genau ein Roboter im Torhalbkreis stehen und das Tor bewachen.

# 2 Analyse der Produktfunktionen

In den nachfolgenden Abschnitten werden die Produktfunktionen des Pflichtenheftes näher ausgeführt und analysiert. Mittels Sequenzdiagrammen werden diese graphisch veranschaulicht.

Die folgende Skizze soll den allgemeinen, technischen Aufbau zwischen Kamera, den Raspberry Pis und dem Computer bzw. der GUI näher erläutern. Die Kamera sendet in kurzen Abständen Bilder an ihren Raspberry Pi (Kamera-Pi). Dieser nutzt die Bilder um Positionsdaten zu berechnen. Anschließend werden diese Daten an die Raspberry Pis der Roboter und an den Computer gesendet. Den Pis ist es möglich, untereinander zu kommunizieren, um eine schwarmbasierte, künstliche Intelligenz zu gewährleisten. Der Computer und die einzelnen Pis tauschen bezüglich der GUI, insbesondere der Darstellung der Befehle, Daten aus.

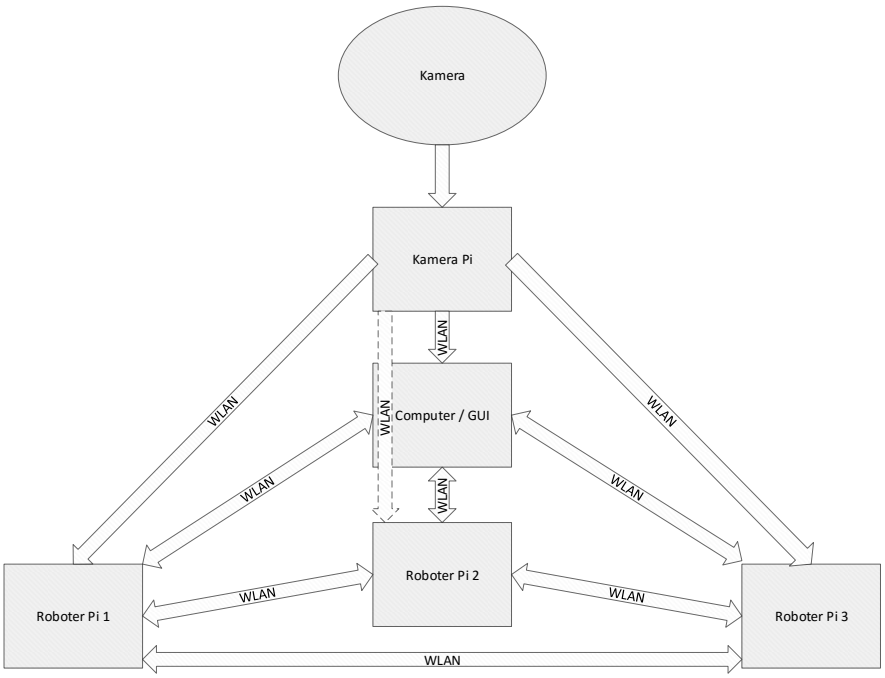


Abbildung 2.1: Skizze: *Überblick des Aufbaus*

## 2.1 Analyse von Funktionalität F10: Spiel starten

Die Funktion F10 gibt dem Nutzer die Möglichkeit das RoboSoccer-Spiel zu beginnen. Dafür wird der „Spiel starten“- Button auf der GUI gedrückt und diese zeigt dem Nutzer dann alle wichtigen Spieldaten an. Bevor die Daten jedoch angezeigt werden, wird intern eine Start-Nachricht von der GUI (dem Computer) an jeden der drei Raspberry Pis verschickt. Daraufhin wird die Information zur Startvorbereitung an den jeweils zugeordneten Roboter weitergeben und anschließend eine Bestätigung an die GUI zurückgegeben. Die Roboter selber sind nicht in der Lage Bestätigungen zu senden, führen jedoch die empfangenen Befehle zum Spielstart aus.

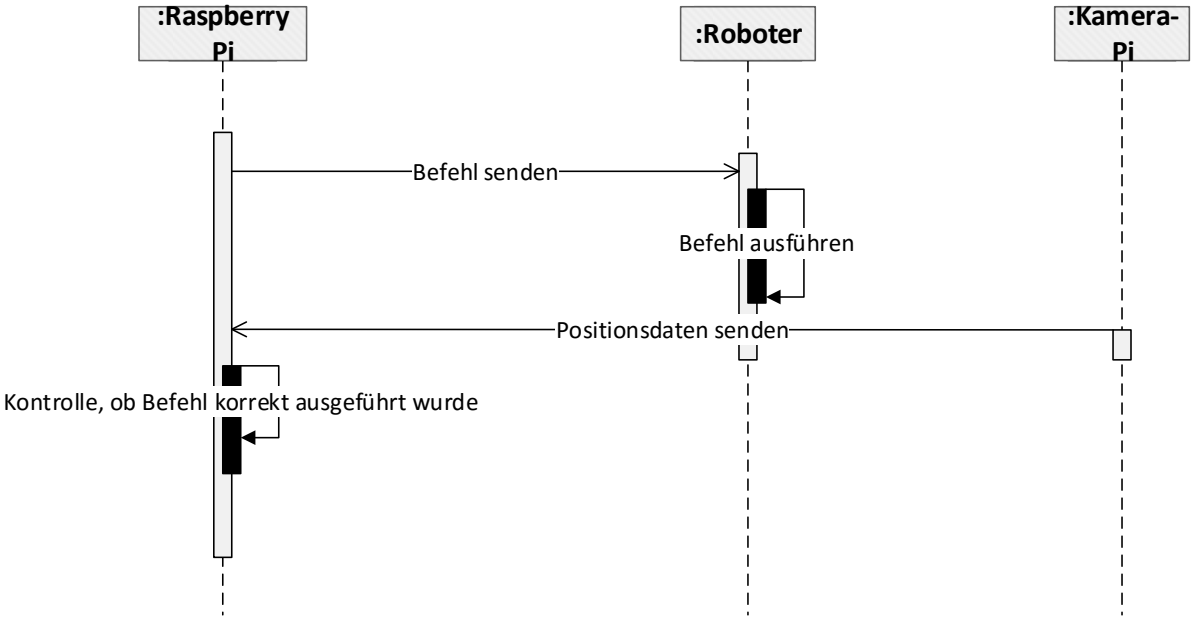


Abbildung 2.2: Sequenzdiagramm: *F10 - Spiel starten*

## 2.2 Analyse von Funktionalität F20: Spiel überwachen / Schiedsrichterfunktionen ausführen

Die Funktion F20 ermöglicht dem Spieler die Parameter des RoboSoccer-Spiels auf der GUI zu verfolgen und eventuell als Schiedsrichter einzugreifen. Mögliche Interaktionen sind dabei „Pausieren“, „Fortsetzen“, „Stoppen“, „Neustart“, „Foul“ und „Tor“ mit jeweiliger Teamauswahl. Dabei betreffen alle Aktionen jeweils sämtliche am Spiel teilnehmende Roboter und nicht nur die des LeGoal Teams, wobei die Datenverarbeitung im Raspberry Pi und die Kommunikation mit dem jeweiligen Roboter unterschiedlich gestaltet sein kann.

### 2.2.1 Analyse von Funktionalität F20a

Das folgende Sequenzdiagramm beruht auf der Annahme, dass das Spiel zuvor schon gestartet wurde. Drückt der Nutzer nun den „Pausieren“-Button in der GUI, so sendet der ausführende Computer eine Nachricht, dass das Pausieren ausgeführt werden soll, an die Raspberry Pis. Jeder Pi mit der darauf installierten KI empfängt diese Nachricht und sendet einen Befehl zum Pausieren an den jeweils zugehörigen Roboter. Nachdem der Roboter den Befehl ausgeführt hat, empfängt der Pi Positionsdaten vom Kamera-Pi und überprüft damit, ob wirklich alle Roboter stehen geblieben sind. Zu beachten ist dabei, dass der Kamera-Pi nahezu sekündlich Positionsdaten versendet und diese von den Raspberry Pis der Roboter und dem GUI-Computer entsprechend empfangen und ausgewertet werden. In sämtlichen Sequenzdiagrammen wurde aber, um die Übersichtlichkeit zu wahren, diese Verbindung nur dargestellt, wenn sie von Bedeutung für die auszuführende Funktion ist. Wenn alle Roboter stehen, sendet der Pi eine Bestätigung an die GUI zurück, die die aktualisierten Daten für den Nutzer anzeigt. Danach soll das Spiel fortgesetzt werden. Nach dem Auslösen des entsprechenden Buttons durch den User, passiert intern wieder nahezu das Gleiche wie schon beim Pausieren, mit dem Unterschied, dass ein „Fortsetzen“-Befehl im System weitergegeben wird und die Überprüfung der Positionsdaten ergeben sollte, dass die Roboter sich wieder bewegen. Die Schiedsrichterfunktion „Stoppen“, welche im Diagramm aus Gründen der Übersichtlichkeit nicht dargestellt ist, funktioniert analog zum „Pausieren“ und ist Voraussetzung für die „Foul“-Funktion, auf die im Folgenden noch eingegangen wird.

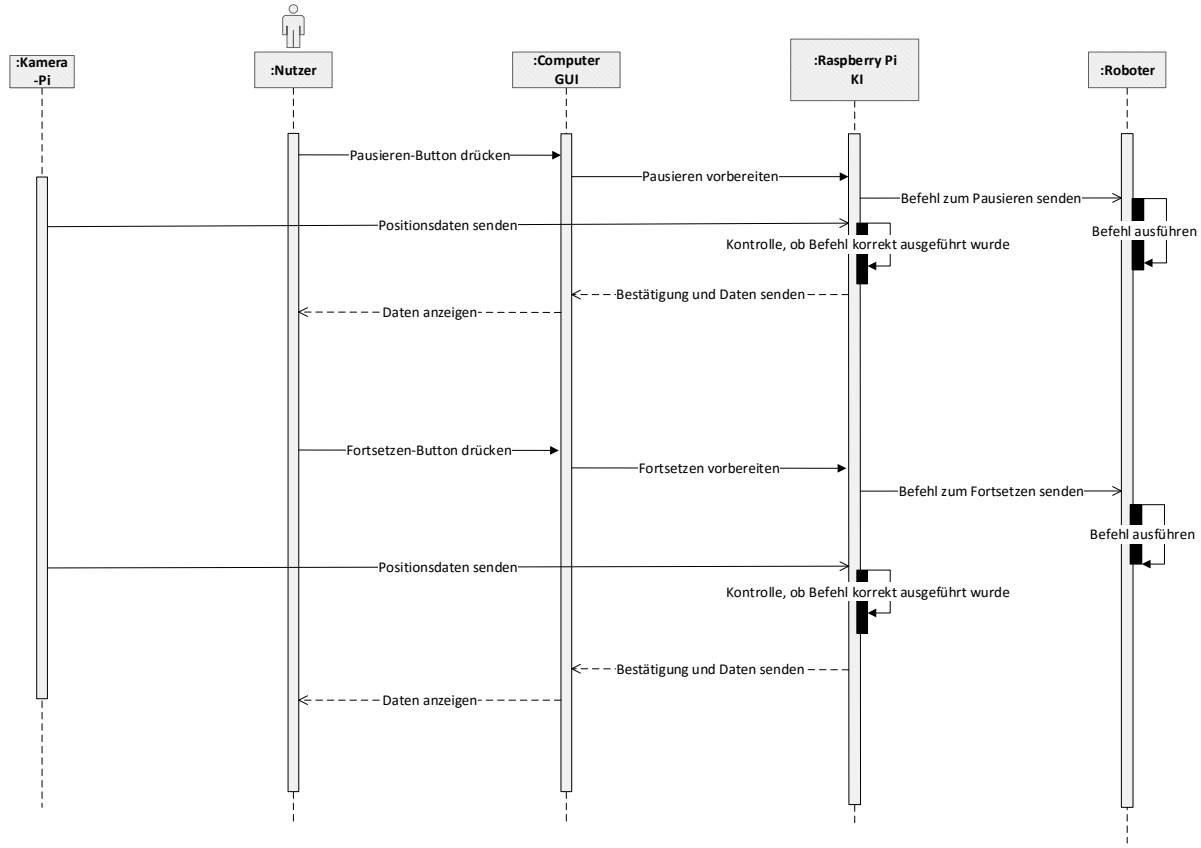


Abbildung 2.3: Sequenzdiagramm: *F20a - Spiel überwachen / Schiedsrichterfunktionen ausführen*

### 2.2.2 Analyse von Funktionalität F20b

Im weiteren Sequenzdiagramm wird das Neustarten des RoboSoccer-Spiels dargestellt. Nachdem der Nutzer den „Neustart“- Button in der GUI gedrückt hat, sendet der ausführende Computer eine entsprechende Nachricht an alle Raspberry Pis. Diese geben den „Startposition“-Befehl weiter an die Roboter, welche dann zur gespeicherten Startposition zurückfahren. Der Roboter-Pi erhält nach kurzer Zeit neue Positionsdaten vom Kamera-Pi und wertet diese aus, ebenso der Computer mit der GUI. Ergibt die Analyse der Positionen, dass alle Roboter auf ihre Startposition zurückgekehrt sind, wird vom Roboter-Pi eine Nachricht an die GUI versendet und diese zeigt die neuen Positionen dem Nutzer an. Ferner gibt der jeweilige Raspberry-Pi den Befehl zum Neustart, der aussagt, dass das Spiel von neuem beginnt, jedoch sonst keine Auswirkungen auf die Technik hat. Der Roboter empfängt den entsprechenden Befehl und führt ihn aus. Der GUI wird erneut eine Bestätigung übermittelt, die an den Nutzer weitergegeben wird.

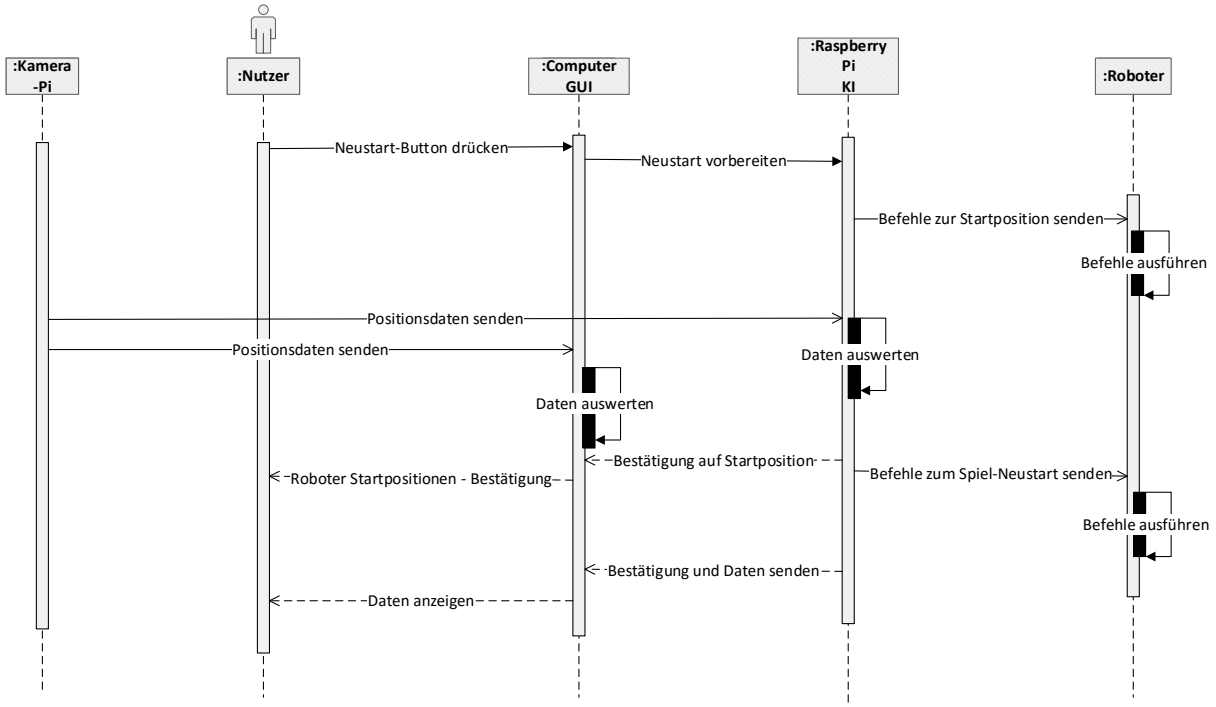


Abbildung 2.4: Sequenzdiagramm: *F20b - Spiel überwachen / Schiedsrichterfunktionen ausführen*

### 2.2.3 Analyse von Funktionalität F20c

Folgendes Sequenzdiagramm zeigt die Situation, wenn das gegnerische Team ein Tor geschossen hat. Zuerst drückt der Nutzer den „+1 Tor - Gegner “-Button in der LeGoal-GUI. Danach verschickt der dazugehörige Computer eine Meldung zum Stoppen und Anstoß vorbereiten, welchen der Raspberry Pi mit der KI empfängt. Nun sendet der Pi zuerst den schon bekannten „Stop“-Befehl an den Roboter, welcher diesen ausführt. Der Pi empfängt zwischenzeitlich neue Positionsdaten vom Kamera-Pi und überprüft damit, ob die Roboter wirklich alle zum Stehen gekommen sind. Anschließend übermittelt er den Befehl zur Startposition zurückzukehren. Zwischenzeitlich erhält der Raspberry Pi schon neue Positionsdaten vom Kamera-Pi, welche er verarbeitet und, falls alle Roboter die Startposition erreicht haben, senden die jeweiligen Pis den Befehl zum Anstoß an das LeGoal-Team. Auch der GUI-Computer empfängt Positionsdaten der Kamera und wertet diese zur späteren Darstellung für den Nutzer aus. Kurz darauf wird eine Bestätigung an den GUI-Computer übermittelt, welcher diese an den Nutzer weitergibt. Hat das LeGoal-Team ein Tor geschossen, sieht der Ablauf sehr ähnlich aus. Die Ausnahme bildet der Anstoß, welcher vom gegnerischen Team ausgeführt wird. Auch ein Foul wird systemintern ähnlich verarbeitet. Das Spiel ist zu dem Zeitpunkt schon gestoppt und es wir nur die Information, wer gefoult, hat weitergegeben. Daraufhin fahren die Roboter zu ihren Startpositionen und das Team, welches gefoult wurde, erhält den Anstoß.

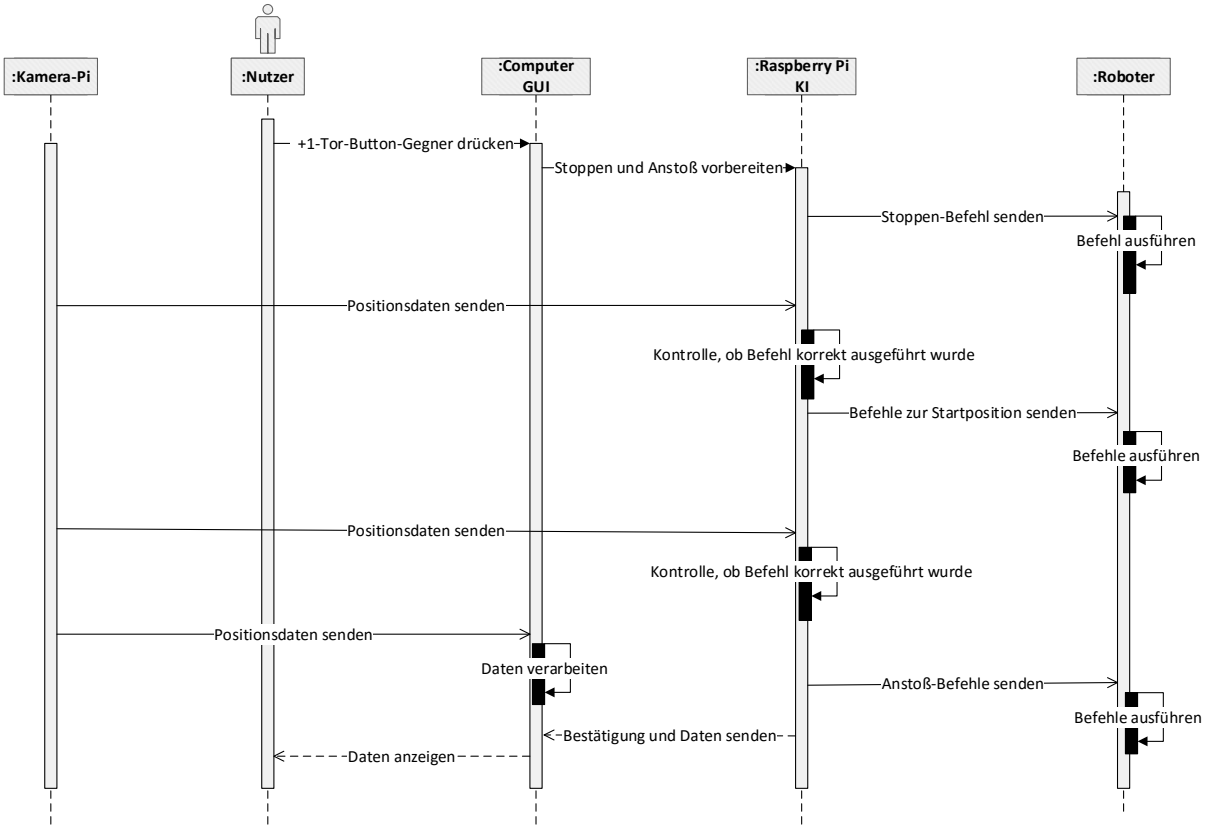


Abbildung 2.5: Sequenzdiagramm: *F20c - Spiel überwachen / Schiedsrichterfunktionen ausführen*

## 2.3 Analyse von Funktionalität F30: Spielfeld graphisch darstellen

Die Funktion F30 hat die Aufgabe das Spielfeld in der LeGoal-GUI darzustellen. Dafür werden vom Kamera-Pi über UDP die berechneten Positionsdaten an die GUI gesendet, welche diese dann verarbeitet und graphisch veranschaulicht.

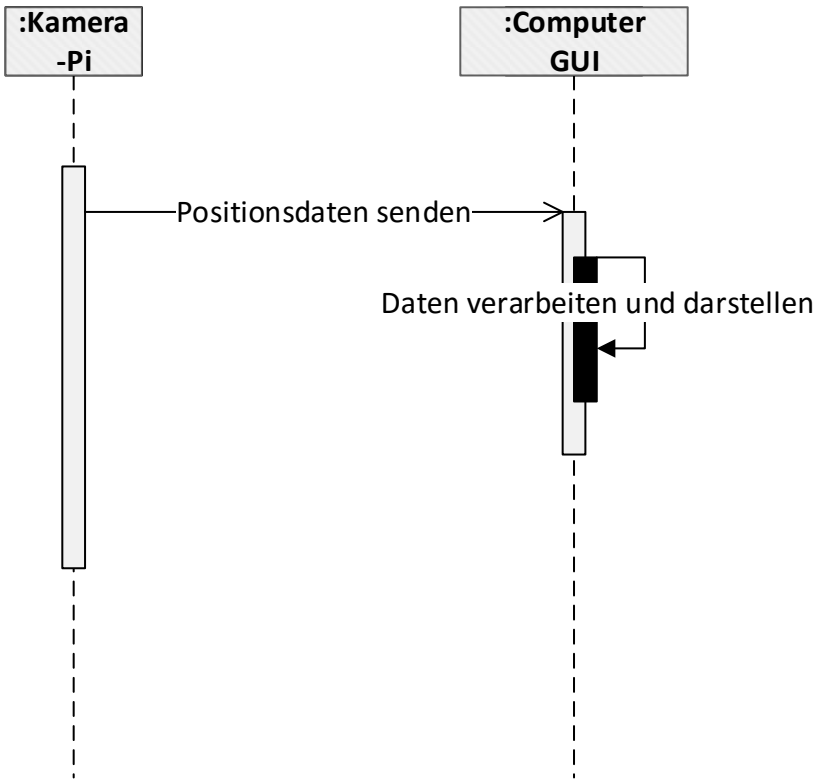


Abbildung 2.6: Sequenzdiagramm: *F30 - Spielfeld graphisch darstellen*

## 2.4 Analyse von Funktionalität F40 und F70: KI berechnet Befehle / Roboterinteraktion

Die Funktionen F40 und F70 werden gemeinsam betrachtet, da das Berechnen der Befehle in der KI eng mit der Roboterinteraktion zusammenhängt. Zuerst erhalten die Raspberry Pis der drei Roboter des LeGoal-Teams Positionsdaten vom Kamera-Pi. Diese werten sie jeweils aus. Anhand der Position des Balles entscheiden sie sich für eine Oberstrategie und jeder Roboter-Pi entscheidet sich dann aufgrund der eigenen Position für Substrategien. Diese teilt jeder Pi den anderen teameigenen Pis mit. Diese Kommunikation kann, anders als im Diagramm, auch zeitgleich geschehen. Die Substrategie mit der höchsten Priorität, welche von allen Roboter-Pis vorgeschlagen wurde, wird gewählt und die Pis senden entsprechende Bestätigungen untereinander. Danach erfolgt nach gleichem Muster eine Kommunikation über die Wahl der Rollen der einzelnen Roboter. Diese wurde aber, um die Übersichtlichkeit zu wahren, nicht dargestellt. Mithilfe der gewählten Strategie und Rolle berechnen die Pis schließlich die Befehle für die Roboter, die entsprechend übertragen werden.

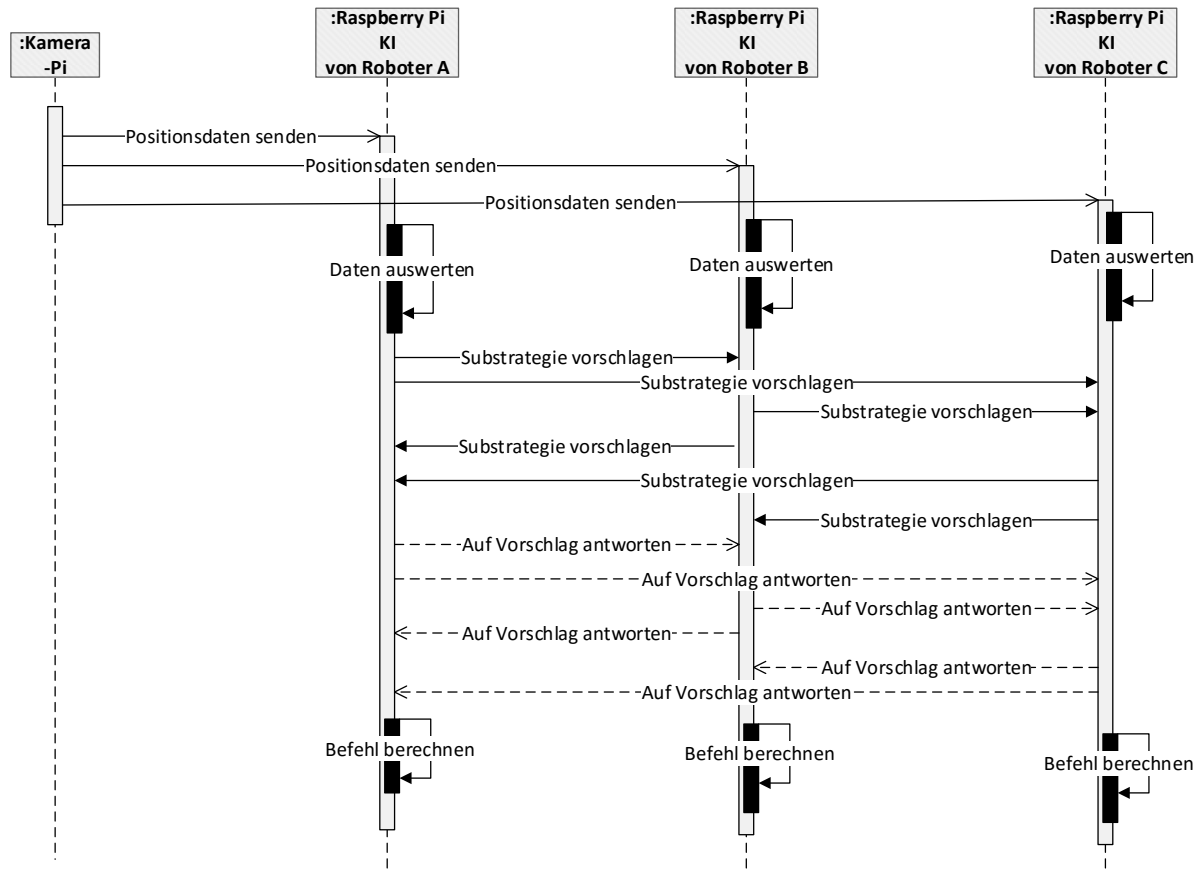


Abbildung 2.7: Sequenzdiagramm: *F40 und F70 - KI berechnet Befehle / Roboterinteraktion*

## 2.5 Analyse von Funktionalität F50 und F60: Raspberry Pis senden Befehle an Roboter / Befehle ausführen

Die Funktionen F50 und F60 werden gemeinsam betrachtet, da sie stark voneinander abhängig sind. Sobald der Raspberry Pi die von der KI berechneten Befehle (vgl. F40) versendet hat und diese vom Roboter erhalten wurden, führt der Roboter diese aus. Es wird jedoch vom Roboter keine Bestätigung übermittelt. Eine Kontrolle erfolgt über die vom Kamera-Pi versendeten Positionsdaten, die der Raspberry-Pi des Roboters empfängt und auswertet.

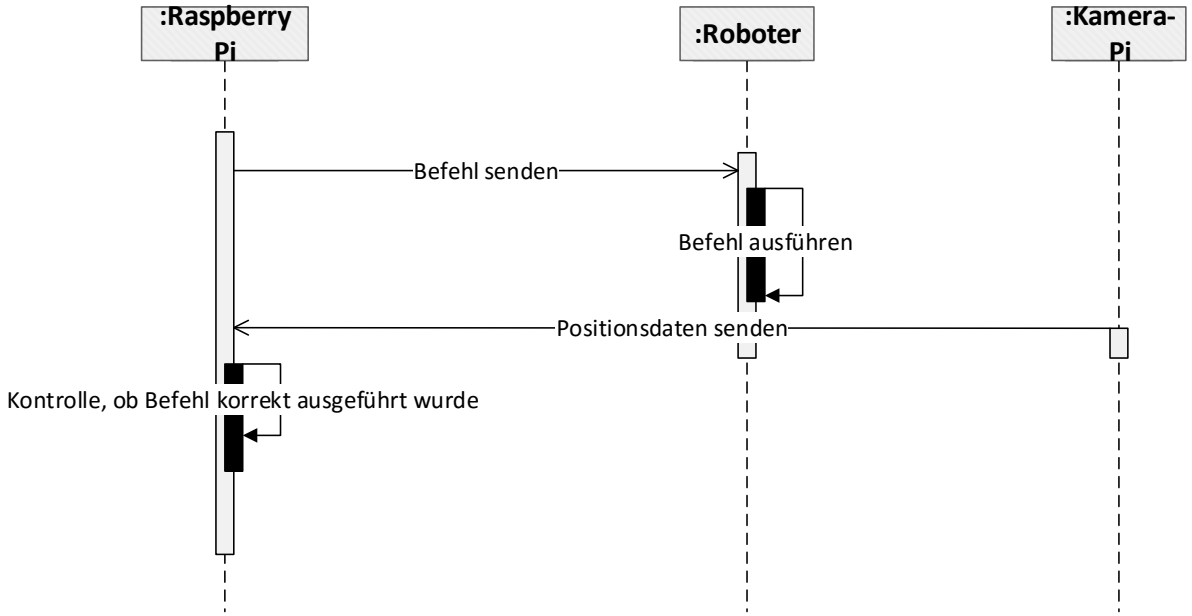


Abbildung 2.8: Sequenzdiagramm: *F50 und F60 - Raspberry Pis senden Befehle an Roboter / Befehle ausführen*

## 2.6 Analyse von Funktionalität F80: Roboter fährt zum Ball

Die Funktion F80 ermöglicht, dass jeweils ein Roboter zum Ball fährt. Dabei werden vom Kamera-Pi die Positionsdaten verschickt und der Raspberry Pi des Roboters empfängt diese. Das Sequenzdiagramm zeigt exemplarisch, wie der Roboter, der am nächsten am Ball steht, sich zum Ball bewegen soll. Dazu werden wie vormals erwähnt die Positionsdaten empfangen, daraus die Befehle zum Bewegen berechnet und diese dann an den Roboter versendet, welcher sie ausführt. Danach werden erneut aktuelle Positionsdaten vom Kamera-Pi empfangen und ausgewertet, ob die Aktion erfolgreich war, und gegebenenfalls das weitere Vorgehen berechnet.

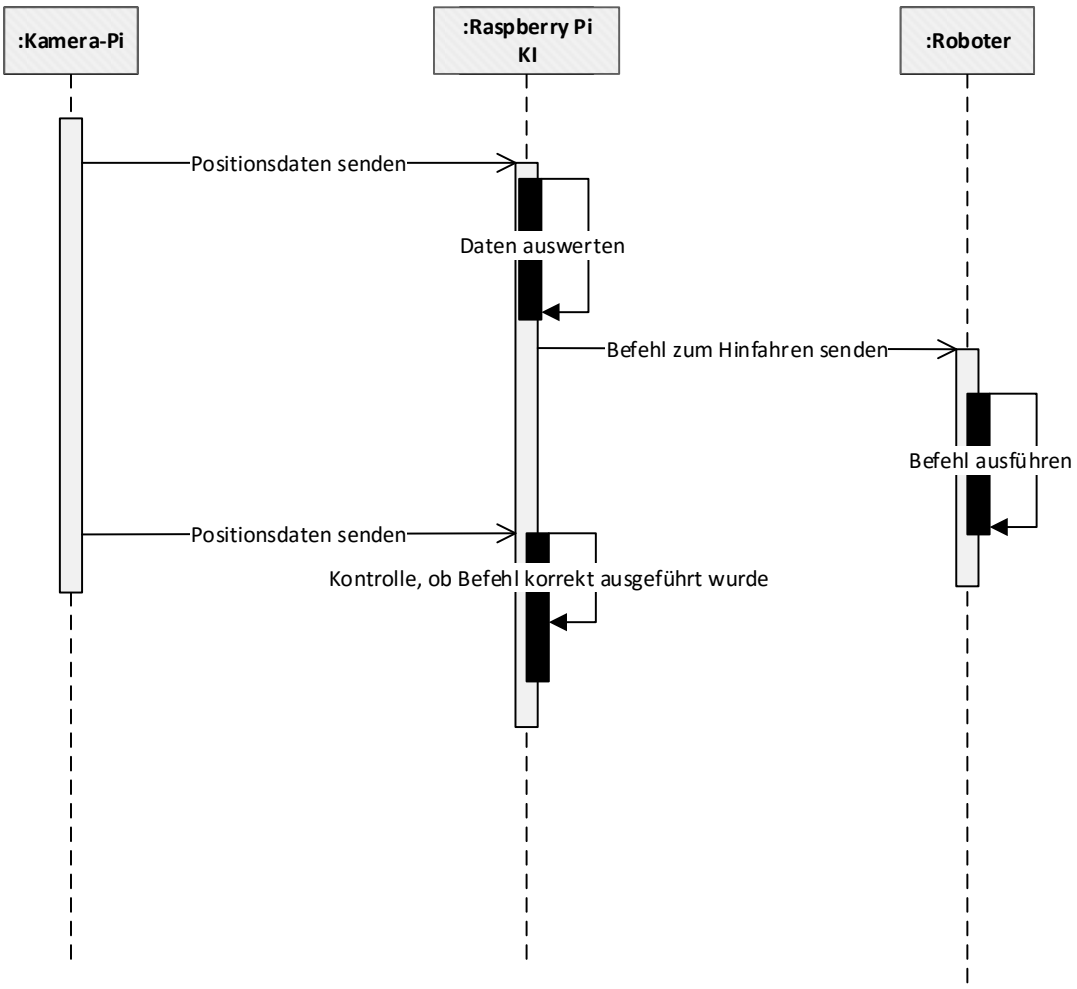


Abbildung 2.9: Sequenzdiagramm: *F80 - Roboter fährt zum Ball*

## 2.7 Analyse von Funktionalität F90: Roboter schießt den Ball

Die Funktion F90 sorgt dafür, dass die Roboter Schüsse ausführen können. Dabei wird vom jeweiligen Raspberry Pi der Befehl zum Schuss an den Roboter gesendet, der diesen dann ausführt. Danach überprüft der Raspberry Pi des Roboters anhand der aktuellen Positionsdaten vom Kamera-Pi, ob der Befehl ausgeführt wurde.

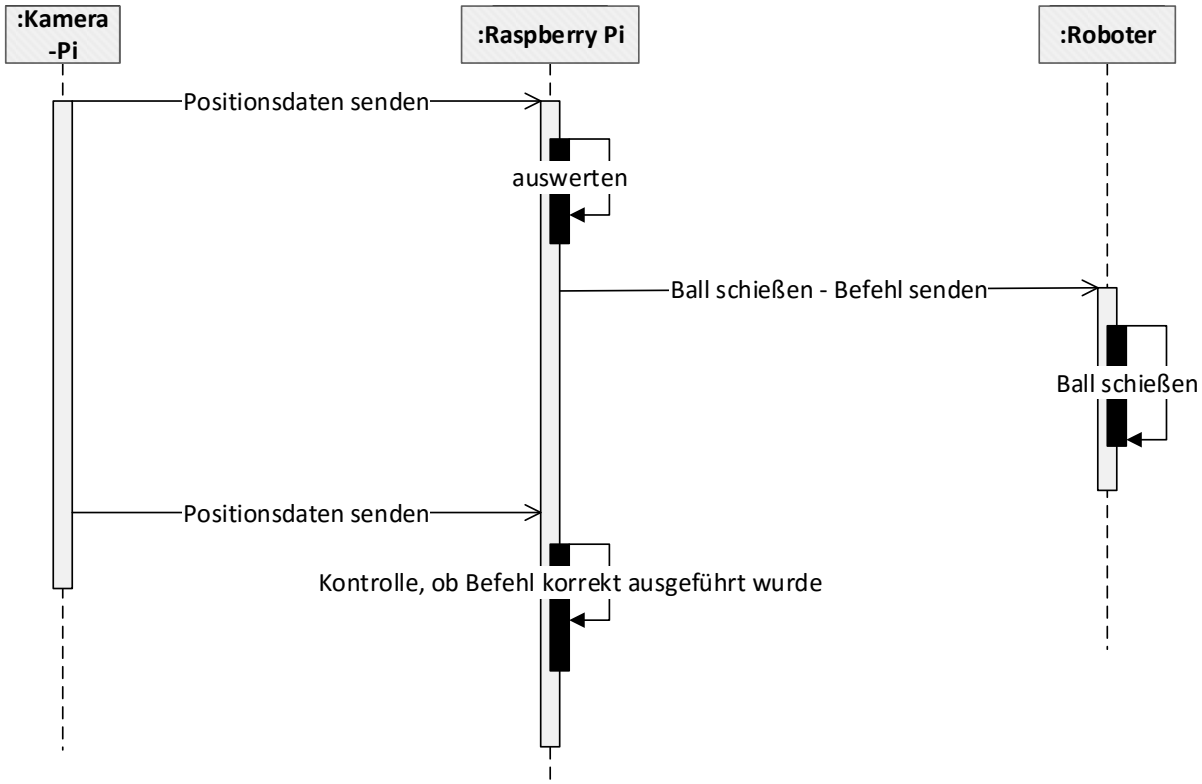


Abbildung 2.10: Sequenzdiagramm: *F90 - Roboter schießt den Ball*

## 2.8 Analyse von Funktionalität F100: Ball passen

Die Funktion F100 ermöglicht das Zuspielen des Balls von einem Roboter zum anderen. Als erstes sendet der Kamera-Pi die Positionsdaten an den Raspberry Pi des Roboters. Nun sendet der Raspberry Pi des Roboters mit Ball einen Befehl an den Roboter, damit dieser sich zum Roboter ohne Ball dreht. Äquivalent dazu erhält der andere Roboter einen Befehl zum Ausrichten und führt diesen aus. Zwischenzeitlich sendet der Raspberry Pi der Kamera wieder die Positionsdaten der Roboter und des Balls. Nun empfängt der Roboter mit Ball von seinem Raspberry Pi den Befehl zum Schießen des Balls und der Roboter ohne Ball den Befehl zum Entgegennehmen des Balls. Beide führen den entsprechenden Befehl aus. Nach jeder ausgeführten Aktion überprüft der Raspberry Pi des entsprechenden Roboters mithilfe der Positionsdaten der Kamera, ob diese ausgeführt wurde.

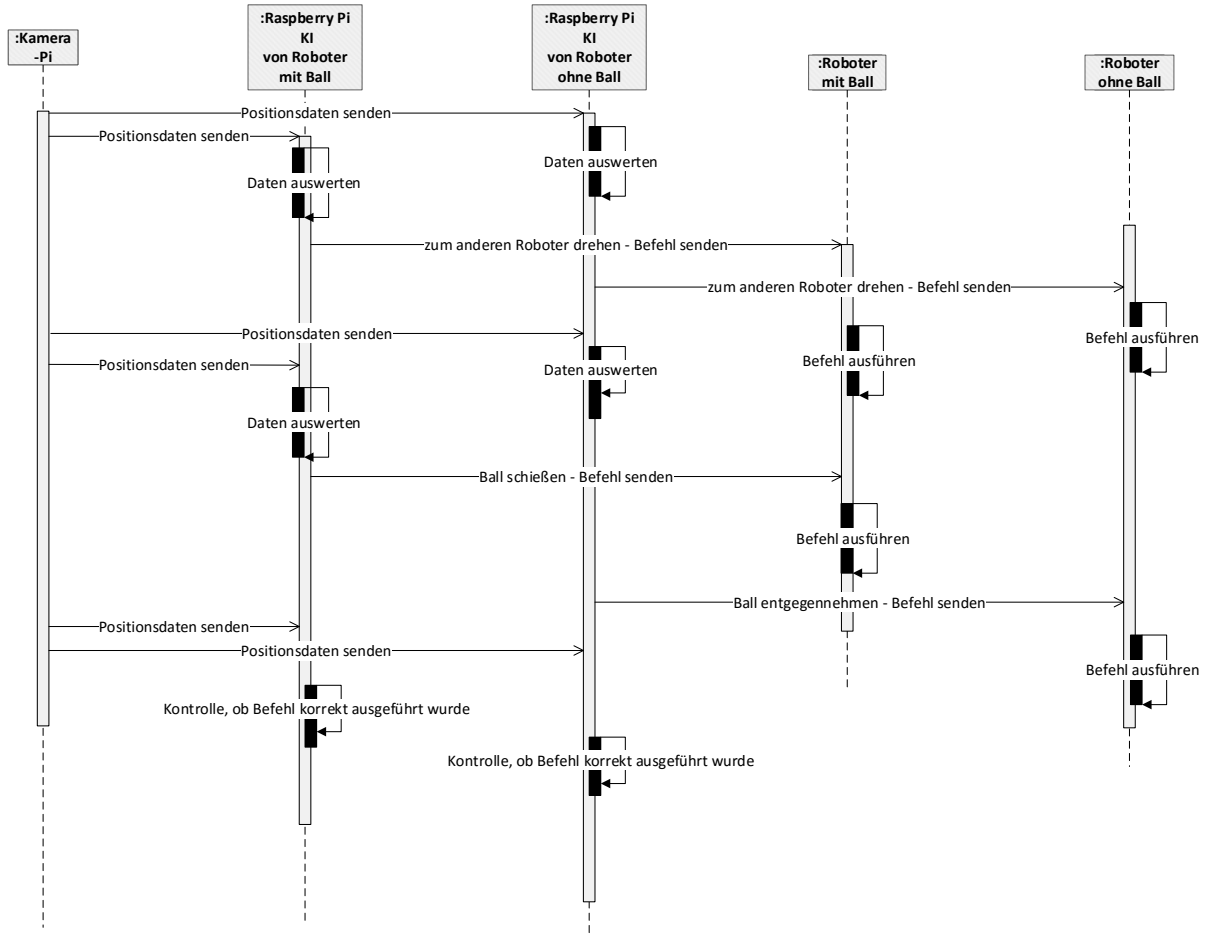


Abbildung 2.11: Sequenzdiagramm: *F100 - Ball passen*

## 2.9 Analyse von Funktionalität F110: Ball kontrollieren

Die Funktion F110 beschreibt das Kontrollieren des Balls durch einen Roboter. Der Kamera-Pi sendet zuerst die Positionsdaten. Der Raspberry Pi wertet diese Daten danach aus. Anschließend wird an den Roboter der Befehl zum Kontrollieren des Balls gesendet. Der Roboter führt diesen Befehl nun aus. Abschließend sendet der Kamera-Pi wieder aktuelle Positionsdaten, damit der Raspberry Pi überprüfen kann, ob die Aktion des Roboters erfolgreich ausgeführt wurde.

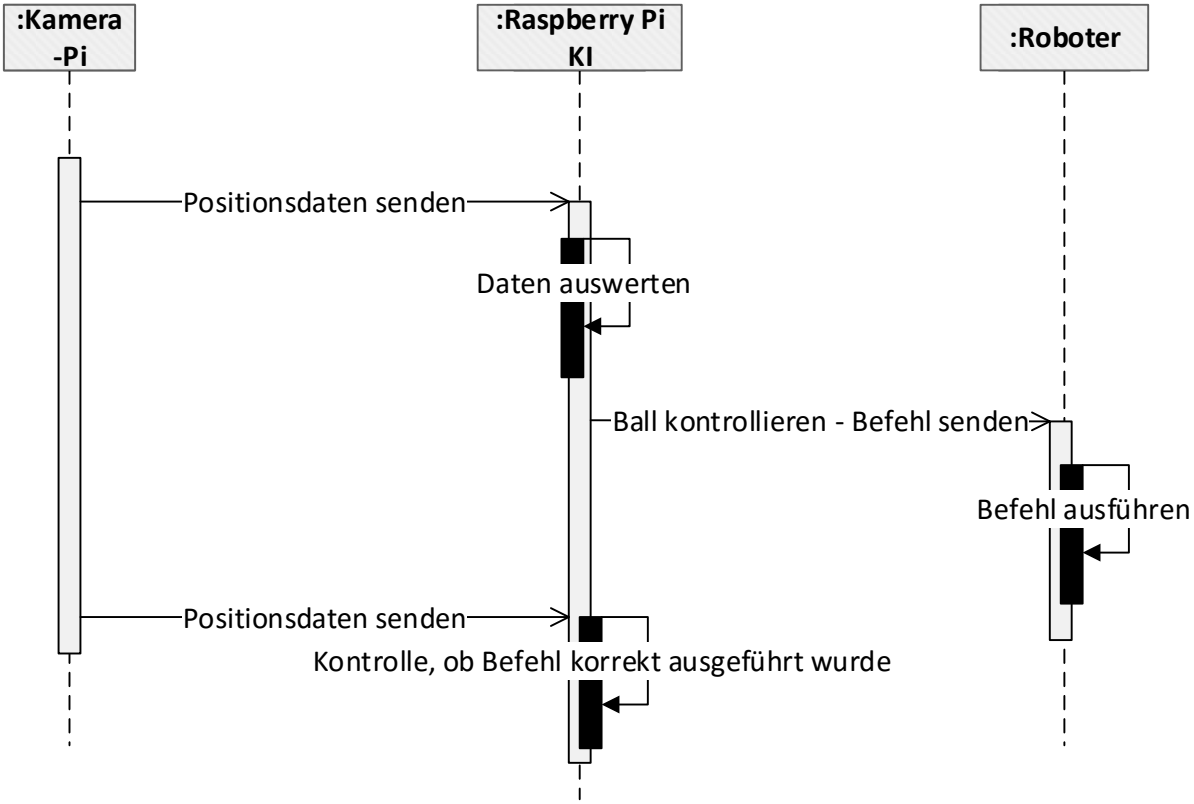


Abbildung 2.12: Sequenzdiagramm: *F110 - Ball kontrollieren*

## 2.10 Analyse von Funktionalität F120 und F130: Manuelle Steuerung

Die Funktionen F120 und F130 beschreiben jeweils die manuelle Steuerung der Roboter zum einen mit einem Controller, zum anderen mit einer Tastatur. Der Nutzer wählt als erstes einen Roboter in der GUI aus. Der Computer der GUI stellt anschließend die Verbindung zum Roboter her und zeigt dem Nutzer bei erfolgreicher Verbindung die Anleitung für die Steuerung. Danach kann der Nutzer zum Steuern eine der empfohlenen Tasten drücken. Der Computer der GUI sendet den zugehörigen Befehl dem Roboter. Dieser führt den entsprechenden Befehl aus. Dieser Vorgang wiederholt sich so lange, bis der Benutzer die manuelle Steuerung in der GUI beendet. Der Computer der GUI trennt anschließend die Verbindung zum Roboter und zeigt dem Nutzer schlussendlich wieder den Startbildschirm der GUI an. Manuell gesteuert werden kann das Spiel nur dann, wenn alle Roboter aus beiden Teams von Nutzern gesteuert werden.

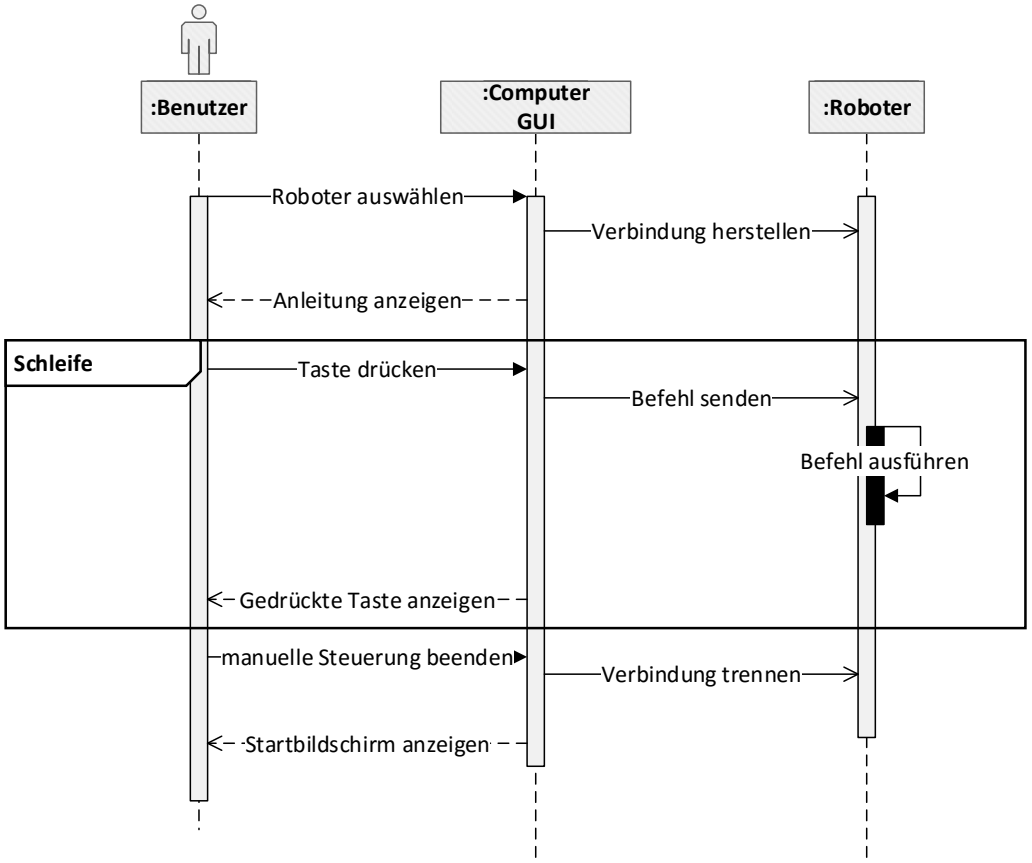


Abbildung 2.13: Sequenzdiagramm: *F120/F130 - Manuelle Steuerung*

# 3 Resultierende Softwarearchitektur

Im folgenden Kapitel werden die zu entwickelnden Komponenten genauer dargestellt und erläutert. Dies soll einen Überblick über die resultierende Softwarearchitektur des Projektes schaffen.

## 3.1 Komponentenspezifikation

Als Erstes wird die resultierende Komponentensturktur aus der Analyse der Produktfunktionen (Kapitel 2) mithilfe eines Komponentendiagramms beschrieben. Direkt danach werden sämtliche Komponenten einzeln genauer erläutert.

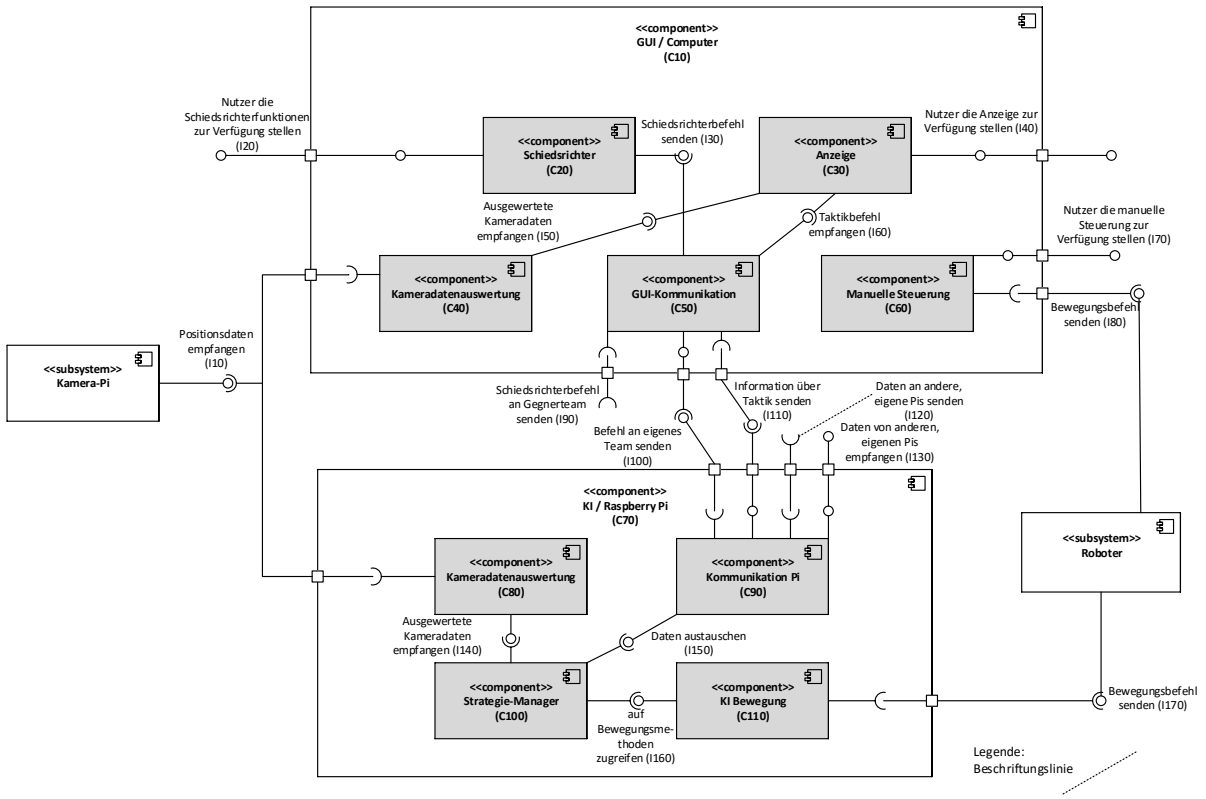


Abbildung 3.1: Komponentendiagramm

**Beschreibung der Komponenten:**

**Komponente $\langle C10 \rangle$: GUI/Computer**
Eine der zwei übergeordneten Kompontenten stellt die graphische Benutzeroberfläche (GUI) dar. Sie bietet dem Nutzer diverse Funktionen, die in den Unterkomponenten genauer ausgeführt werden.

**Komponente $\langle C20 \rangle$: Schiedsrichter**
Sollte der Nutzer mittels der Buttons der GUI Schiedsrichter-Entscheidungen getroffen haben, so muss die Komponente Schiedsrichter mit der KI unseres Team sowie den gegnerischen Robotern kommunizieren.

**Komponente $\langle C30 \rangle$: Anzeige**
Die Anzeige der GUI visualisert das Spielgeschehen für den Nutzer. Die Komponente ist somit für die Ausgabe der Kameradaten in eine graphische Oberfläche zuständig. Zusätzlich werden die gewählten Taktiken in schriftlicher Form angezeigt.

**Komponente $\langle C40 \rangle$: Kameradatenauswertung**
Diese Komponente greift auf die vom Subsystem Kamera-Pi zur Verfügung gestellten Positionsdaten zu und wertet sie für den späteren Einsatz entsprechend aus.

**Komponente $\langle C50 \rangle$: GUI-Kommunikation**
Diese Komponente kann über diverse Schnittstellen an sämtliche Roboter, auch die des Gegnerteams, die Schiedsrichter-Befehle weiterleiten. Auch erhält sie die Informationen über die Taktik von der KI.

**Komponente $\langle C60 \rangle$: Manuelle Steuerung**
Diese Komponente kommt nur zum Einsatz, wenn der Nutzer die Roboter manuell steuert, also ohne Verwendung der künstlichen Intelligenz. In diesem Fall läuft die Kommunikation mit den Robotern direkt über die GUI. Die vom Nutzer gewählten Befehle werden durch diese Komponente an den jeweiligen Roboter weitergegeben.

**Komponente $\langle C70 \rangle$: KI / Raspberry Pi**
Eine der zwei übergeordneten Komponenten stellt die künstliche Intelligenz (KI), welche auf den Raspberry Pis läuft, dar. In den verschiedenen Unterkomponenten wird genauer auf die einzelnen Funktionen eingegangen. Zu beachten ist, dass im eigentlichen Produkt drei Raspberry Pis zum Einsatz kommen, welche untereinander kommunizieren können.

**Komponente $\langle C80 \rangle$: Kameradatenauswertung**
Diese Komponente greift auf die vom Subsystem Kamera-Pi zur Verfügung gestellten Positionsdaten zu und wertet sie für den späteren Einsatz entsprechend aus.

**Komponente $\langle C90 \rangle$: Kommunikation Pi**
Diese Komponente kann über diverse Schnittstellen mit den anderen, eigenen Pis sowie der GUI

kommunizieren.

**Komponente $\langle C100 \rangle$: Strategie-Manager**
In dieser Komponente werden die Strategien für das Spiel berechnet.

**Komponente $\langle C110 \rangle$: KI Bewegung**
Diese Komponente verwaltet die verschiedenen Bewegungsbefehle und sendet diese bei Bedarf an die Roboter.

## 3.2 Schnittstellenspezifikation

Im Folgenden werden die einzelnen Schnittstellen der Komponenten aus der Komponentenspezifikation näher erläutert.

**Schnittstelle $\langle I10 \rangle$: Positionsdaten empfangen**

| Operation | Beschreibung |
|---|---|
| receivePosition() | Das Subsystem „Kamera“ sendet ständig Positionsdaten, welche von der Kameraauswertung der GUI und der KI auf den Raspberry Pis entgegen genommen werden. |

**Schnittstelle $\langle I20 \rangle$: Nutzer die Schiedsrichterfunktion zur Verfügung stellen**

| Operation | Beschreibung |
|---|---|
| RoboSoccerPrimary() | Stellt in der GUI Buttons zur Ausübung der Schiedsrichterfunktion durch den Nutzer zur Verfügung. Bei dieser Methode hat das LeGoal-Team das Tor auf der Fensterseite des Legolabors. |
| RoboSoccerSecondary() | Stellt in der GUI Buttons zur Ausübung der Schiedsrichterfunktion durch den Nutzer zur Verfügung. Bei dieser Methode hat das LeGoal-Team das Tor auf der Türseite des Legolabors. |
| actionPerformed(ActionEvent ae) | Ist ein ActionListener, der prüft, ob einer der zuvor beschriebenen Buttons gedrückt wurde. |

**Schnittstelle $\langle I30 \rangle$: Schiedsrichterbefehl senden**

| Operation | Beschreibung |
|---|---|
| setMessage(message: String) | Sendet, nachdem der Nutzer eine Schiedsrichterfunktion ausgewählt hat, einen vorgeschriebenen Schiedsrichterbefehl an die GUI-Kommunikation. |

**Schnittstelle $\langle I40 \rangle$: Nutzer die Anzeige zur Verfügung stellen**

| Operation | Beschreibung |
|---|---|
| getGUI() | Stellt die Anzeige für den Nutzer graphisch da. |

**Schnittstelle $\langle I50 \rangle$: Ausgewertete Kameradaten empfangen**

Die ausgewerteten Kameradaten werden über get-Operationen aus der Coordinate-Klasse einzeln (für jeden Roboter die X- und Y-Koordinate sowie die Rotation und für den Ball die Koordinaten), je nach Bedarf, aufgerufen. Da reine get-Operationen nicht aufgeführt werden sollten, wird hier auf eine Tabelle verzichtet. Die Schnittstelle I140 ist identisch und wird daher nicht gesondert aufgeführt.

**Schnittstelle $\langle I60 \rangle$: Taktik-Befehl empfangen**

| Operation | Beschreibung |
|---|---|
| readLine() | Die Anzeige empfängt die von der GUI-Kommunikation weitergeleitete Taktik der KI, um diese dem Nutzer darzustellen. |

**Schnittstelle $\langle I70 \rangle$: Nutzer die manuelle Steuerung zur Verfügung stellen**

| Operation | Beschreibung |
|---|---|
| new RobotPanel() | Der Nutzer kann wählen, welchen der sechs Roboter er steuern möchte. |
| new ControlPanel() | Der Nutzer steuert den gewählten Roboter über die Tastatur. |

**Schnittstelle $\langle I80 \rangle$: Bewegungsbefehl senden**

| Operation | Beschreibung |
|---|---|
| new ActionMaker(cmd: String) | Es wird ausgelesen, welche Taste der Nutzer gedrückt hat. |
| Movement.befehl() | Die manuelle Steuerung sendet Bewegungsbefehle an den Roboter. befehl() ist hierbei durch die jeweils benötigte Methode der Klasse Movement zu ersetzen. |

**Schnittstelle $\langle I90 \rangle$: Schiedsrichterbefehl an Gegnerteam senden**

| Operation | Beschreibung |
|---|---|
| out.println(message: String) | Die GUI-Kommunikation sendet Schiedsrichterbefehle per TCP an alle Raspberry Pis. |

**Schnittstelle $\langle I100 \rangle$: Befehl an eigenes Team senden**

| Operation | Beschreibung |
|---|---|
| out.println(message: String) | Die GUI-Kommunikation sendet Schiedsrichterbefehle per TCP an alle Raspberry Pis. |
| brinp.readLine() | Die Pis des eigenen Teams empfangen über ihre Kommunikation die Befehle der GUI. |

**Schnittstelle $\langle I110 \rangle$: Informationen über Taktik senden**

| Operation | Beschreibung |
|---|---|
| out.println(message: String) | Der Pi sendet über seine Kommunikation an die teameigenen anderen Pis und die GUI eine Nachricht, welche Taktik ausgeführt wird. |
| brinp.readLine() | Die GUI empfängt die Nachricht über die Taktik. |

**Schnittstelle $\langle I120 \rangle$: Daten an anderen, eigenen Pis senden**

| Operation | Beschreibung |
|---|---|
| out.println(message: String) | Der Pi sendet über seine Kommunikation an die teameigenen anderen Pis und die GUI eine Nachricht, welche Taktik ausgeführt wird. |

**Schnittstelle $\langle I130 \rangle$: Daten von anderen, eigenen Pis empfangen**

| Operation | Beschreibung |
|---|---|
| brinp.readLine() | Die Kommunikation des Pi empfängt Nachrichten der anderen teameigenen Pis. |

**Schnittstelle $\langle I150 \rangle$: Daten austauschen**

| Operation | Beschreibung |
|---|---|
| send(receiver : String, message: String) | Vorbereitung der zu sendenden Nachricht beim Client (Kommunikation Pi Komponente). |
| receive() | Zugriff auf die durch den RPIMultiServer erhaltenen Nachrichten. |
| receiveFromGUI() | Zugriff auf die durch den GUIListener erhaltenen Nachrichten. |

**Schnittstelle $\langle I160 \rangle$: Auf Bewegungsmethoden zugreifen**

| Operation | Beschreibung |
|---|---|
| new AIMovement() | Der Strategie-Manager greift auf die Bewegungsmethoden der KI zu. |

**Schnittstelle $\langle I170 \rangle$: Bewegungsbefehl senden**

| **Operation** | **Beschreibung** |
|---|---|
| AIMovement.befehl() | Die Bewegungsmethode der KI sendet Bewegungsbefehle an den Roboter. befehl() ist hierbei durch die jeweils benötigte Methode der Klasse AIMovement zu ersetzen. |

## 3.3 Protokolle für die Benutzung der Komponenten

Im Folgenden werden die einzelnen Komponenten durch Zustandsdiagramme genauer erläutert.

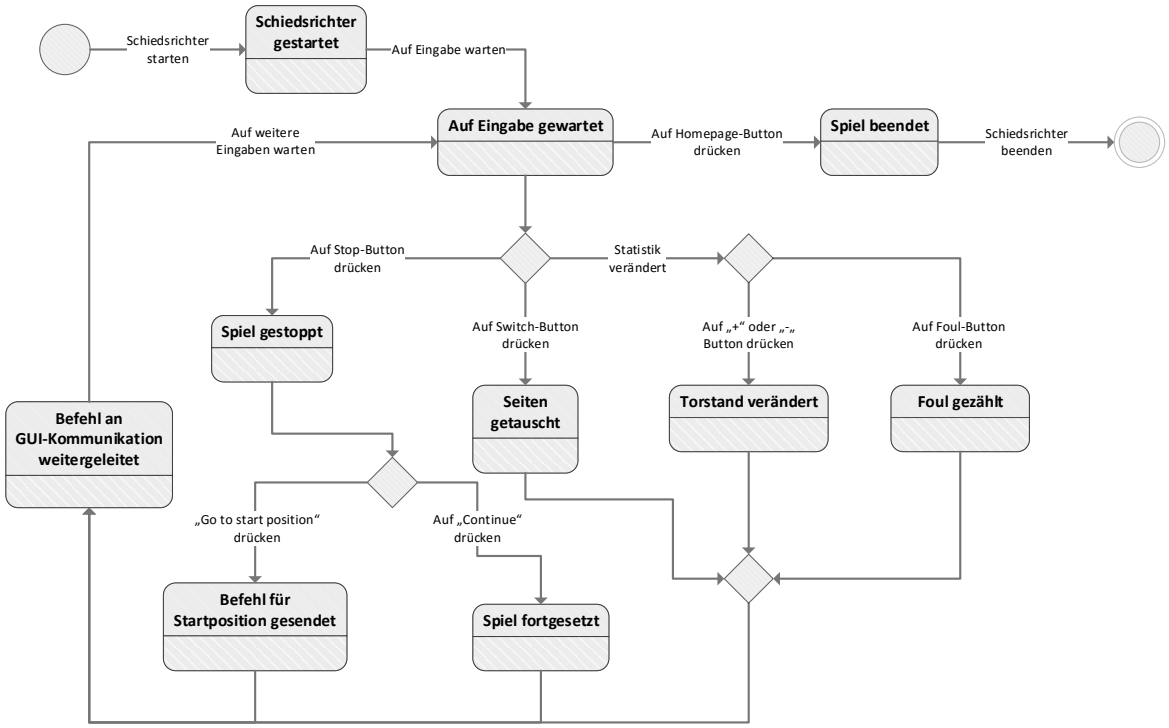


Abbildung 3.2: Zustandsdiagramm: *Schiedsrichter <C20>*

In der Komponente „Schiedsrichter“ wird die Bedienung der Steuerknöpfe in der GUI aufgezeigt. Diese Komponente leitet alle Befehle, die der Schiedsrichter im Spiel geben kann, an die Komponente „GUI-Kommunikation“ weiter. Die Befehle heißen namentlich „Spiel stoppen“, „Spiel fortsetzen“, „Alle Spieler auf Startposition“, „Foul zählen“, „Tor zählen“ und „Seiten tauschen“. Die Befehle „Spiel fortsetzen“ und „Alle Spieler auf Startposition“ können nur gegeben werden, wenn zuvor das Spiel gestoppt wurde.

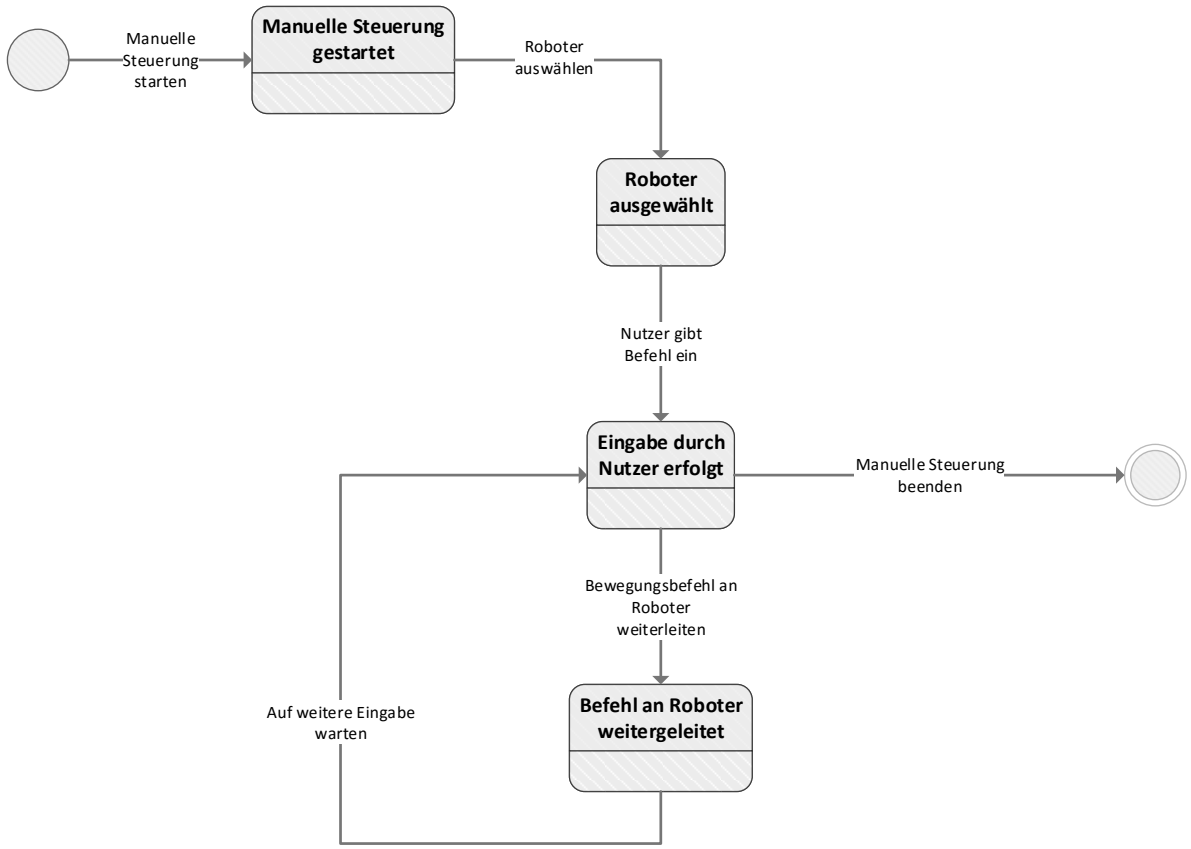


Abbildung 3.3: Zustandsdiagramm: *Manuelle Steuerung <C60>*

Der Zugang zur Funktion „manuelle Steuerung“ erfolgt über die GUI. Dort kann der Nutzer einen der Roboter auswählen. Hat dieser einen ausgewählt, verbindet sich die Komponente mit dem entsprechenden Roboter. Nun kann der Nutzer Bewegungsbefehle über die „Pfeiltasten“ eingeben und mit Hilfe der Leertaste schießen. Diese Befehle werden unverzüglich an den Roboter weitergeleitet. Die manuelle Steuerung endet, wenn der Nutzer sie beendet. Es gibt keine eingebaute Zeitbegrenzung.

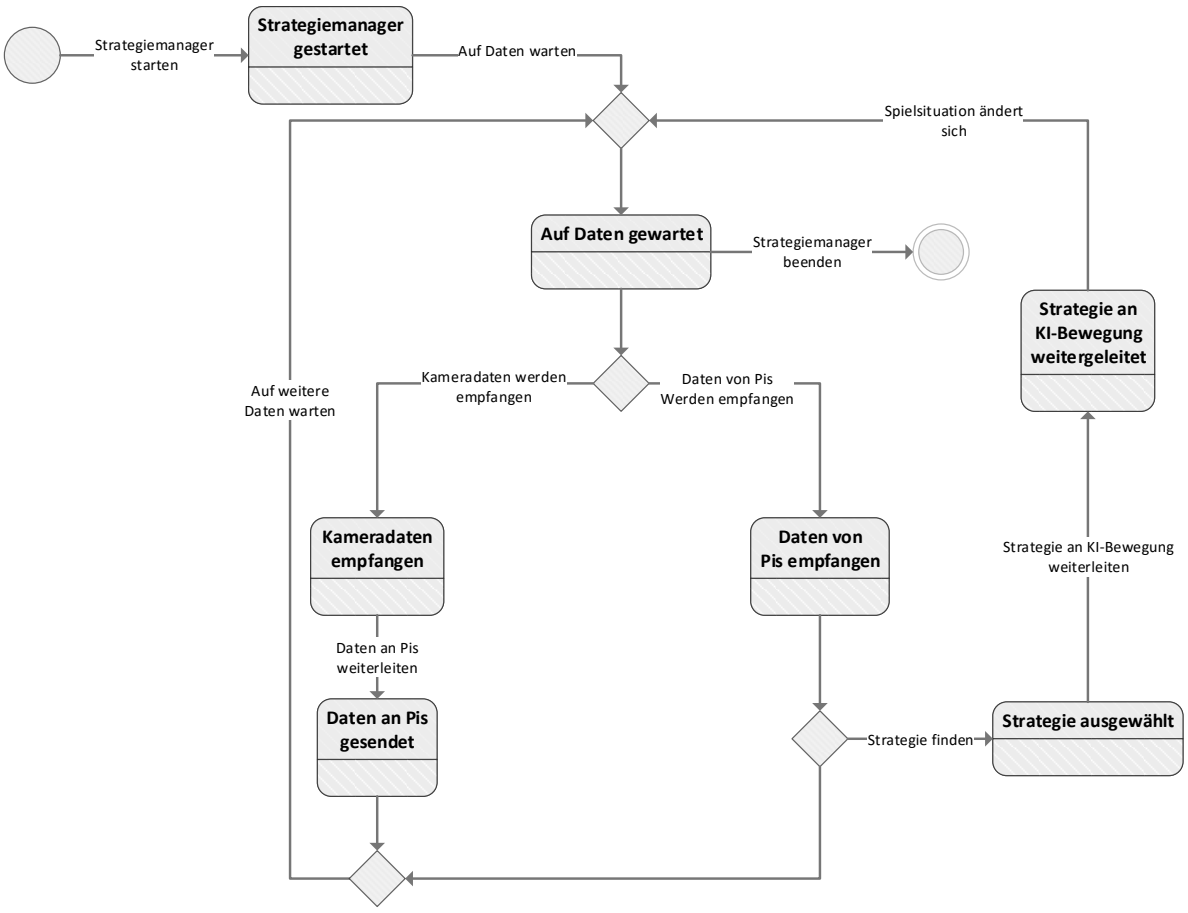


Abbildung 3.4: Zustandsdiagramm: *Strategie-Manager <C100>*

Die Komponente „Strategie-Manager“ entscheidet, welche Strategie die Roboter nehmen. Dies geschieht, indem die Roboter alle mit dem Strategie-Manager kommunizieren. Der Strategie-Manager legt zunächst eine der Hauptstrategie „Angriff“, „Abwehr“ oder „Balleroberung“ anhand des Ballbesitzes fest. Gleichzeitig leitet er alle Kameradaten, die er erhält, an die Raspberry Pis weiter. Die Pis teilen nun dem Strategie-Manager mit, welche Substrategien sie ausführen können. Der Strategie-Manager legt dann die Substrategie fest, die die meisten Roboter ausführen können und die höchste Priorität hat. Diese Strategie wird an die Komponente KI-Bewegung weitergeleitet und solange ausgeführt bis die Strategie fehlschlägt oder erfolgreich war.

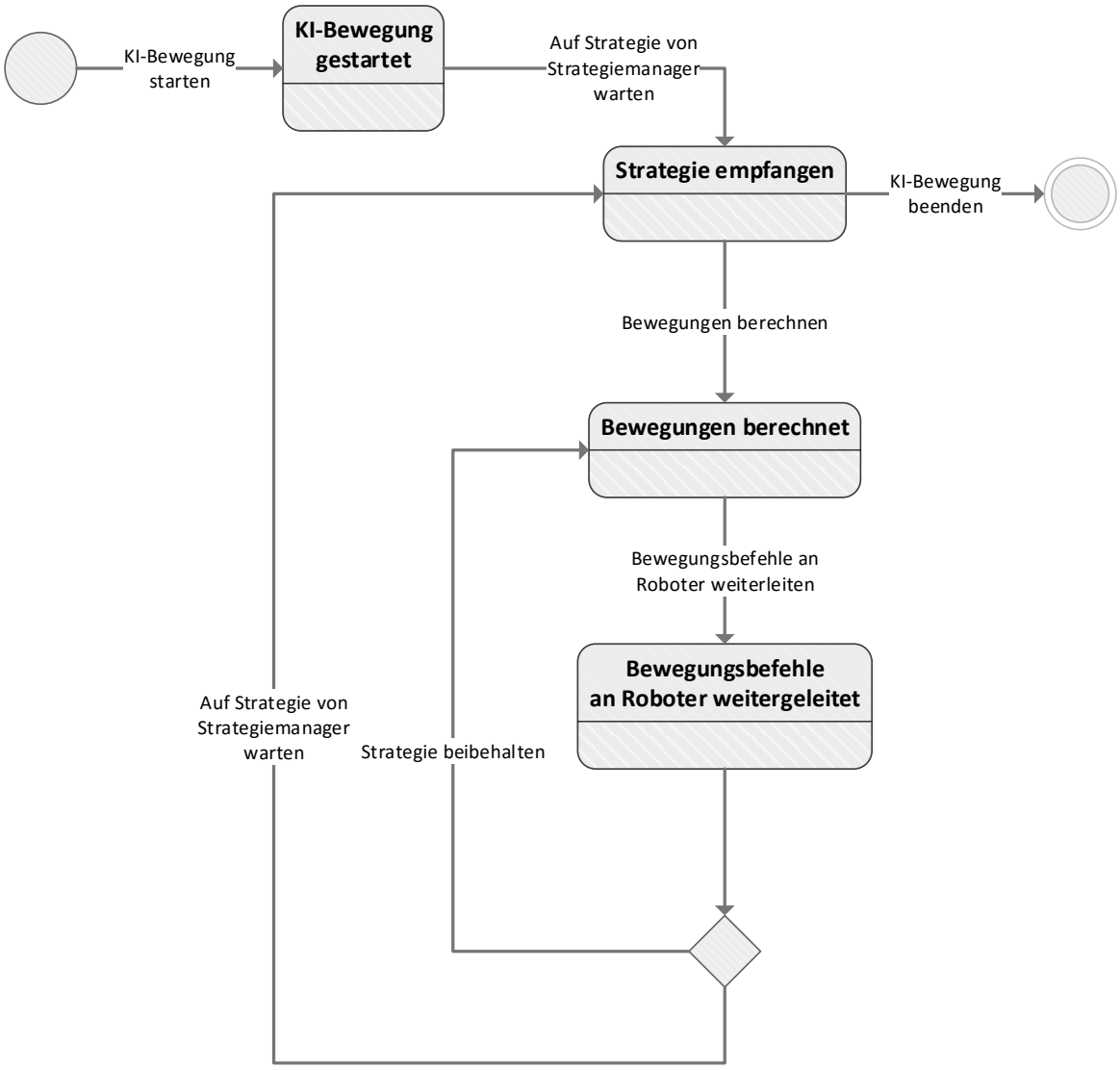


Abbildung 3.5: Zustandsdiagramm: *KI-Bewegung <C110>*

Die Komponente „KI-Bewegung“ berechnet die konkreten Bewegungsbefehle mit Hilfe der Strategie, die sie von dem Strategie-Manager erhält. Die Befehle werden dann an die einzelnen Roboter weitergeleitet. Wird die Strategie oder die Spielsituation geändert, werden neue Berechnungen ausgeführt. Die Komponente endet bei Spielende.

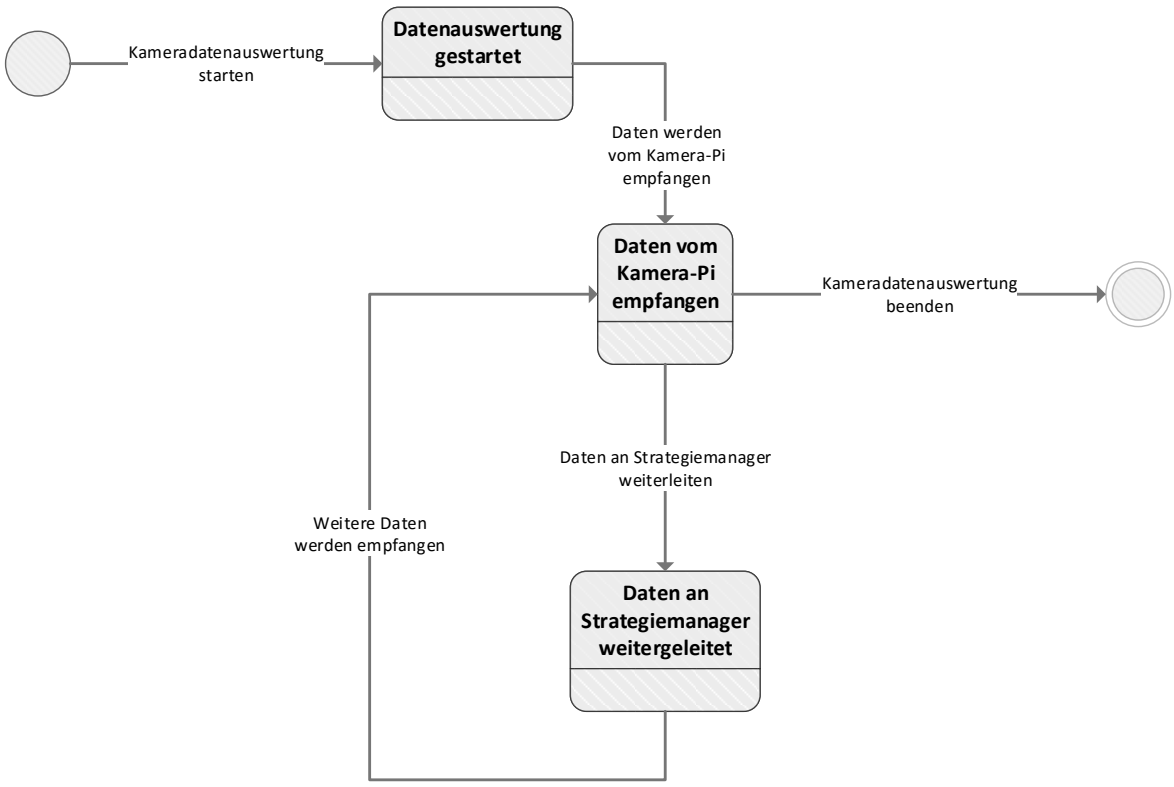


Abbildung 3.6: Zustandsdiagramm: *Kameradatenauswertung für die KI <C80>*

Die Kamera ist ein sehr wichtiges Gerät für unser Spiel. Da sie nicht von uns programmiert wurde, sondern vom Auftraggeber, können wir kein allzu genaues Zustandsdiagramm für diese Komponente entwickeln.

Zuerst nimmt die Kamera das komplette Feld auf. Danach identifiziert eine Software die einzelnen Marker, sowie den Ball. Diese erhalten von der Software dann eine Koordinate. Die Koordinaten werden per UDP an die KI weitergeleitet.

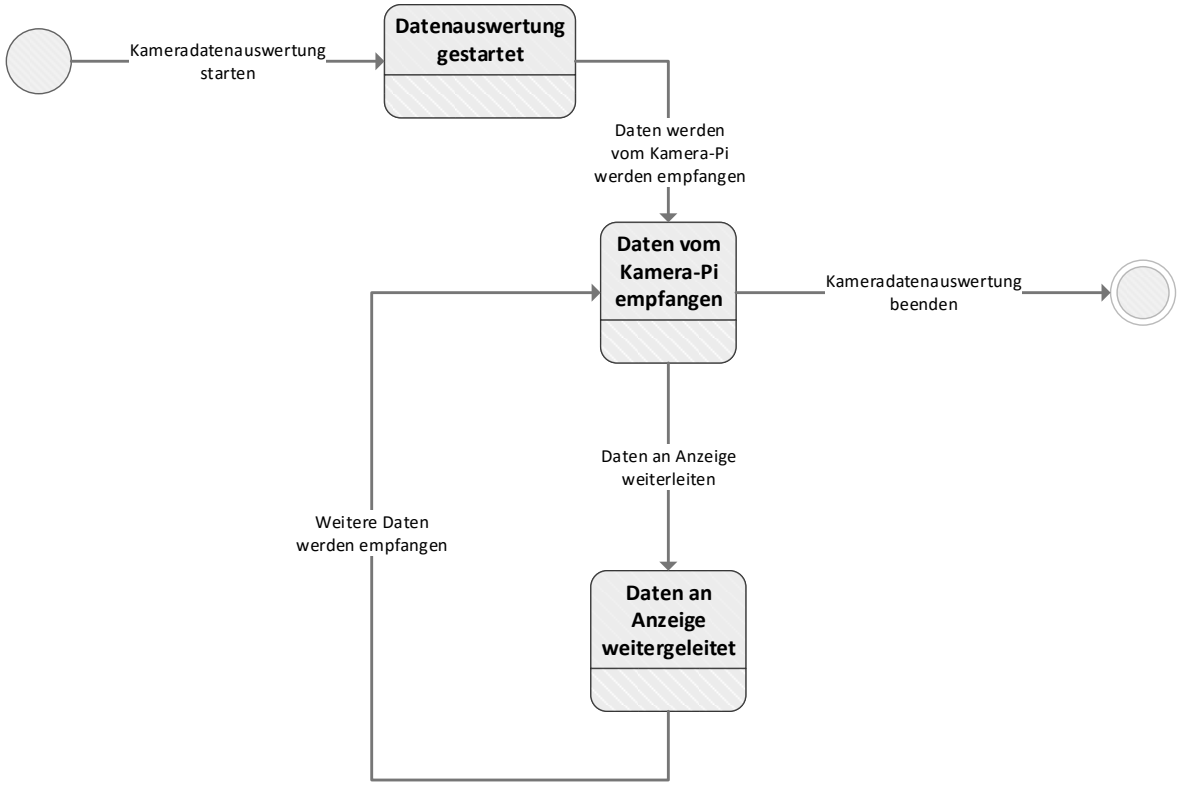


Abbildung 3.7: Zustandsdiagramm: *Kameradatenauswertung für die GUI <C40>*

Diese Komponente funktioniert genau so wie die Komponente „Kameradatenauswertung KI“. Die Koordinaten werden aber statt zur KI zur GUI weitergeleitet.

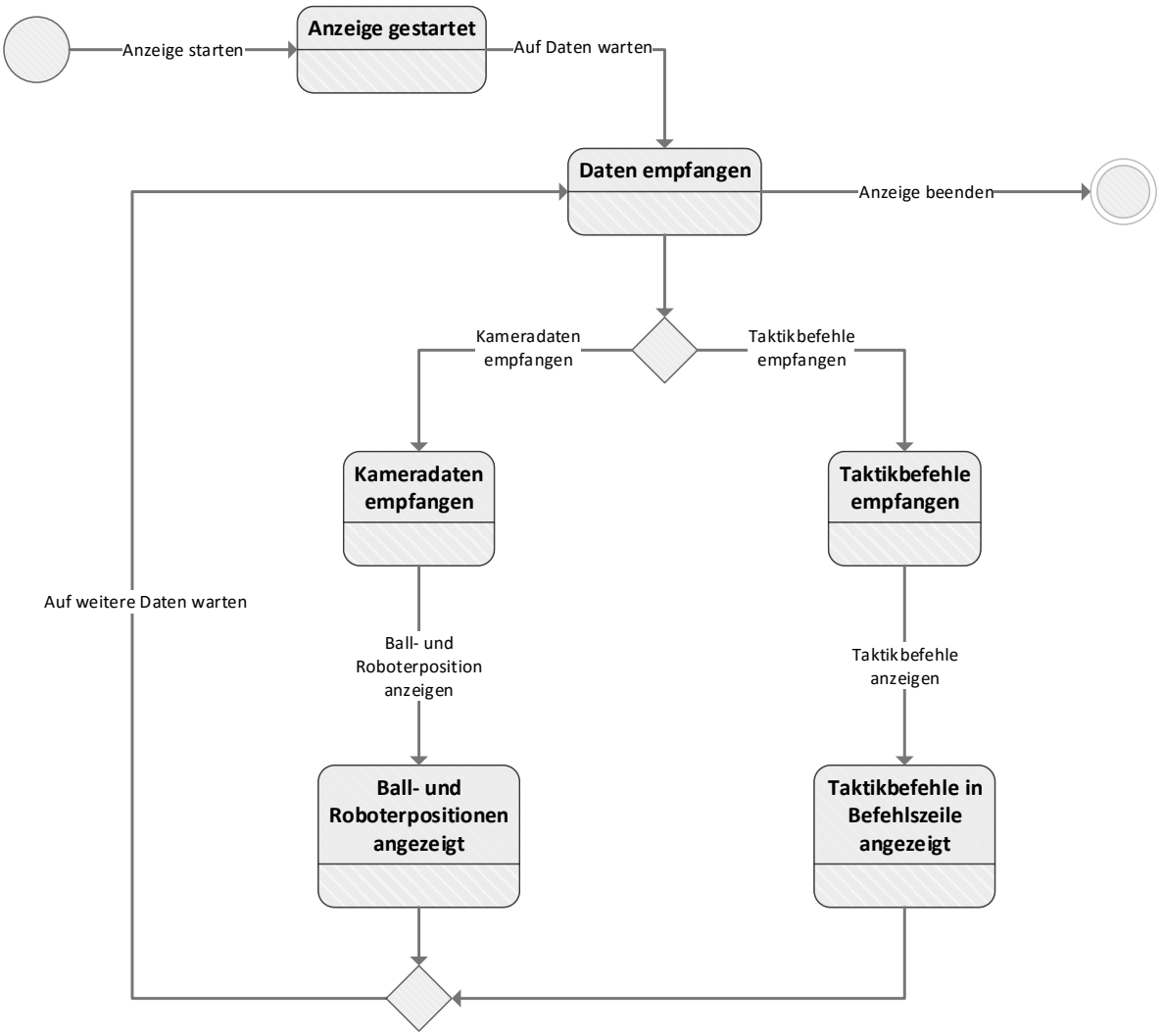


Abbildung 3.8: Zustandsdiagramm: *Anzeige in der GUI <C30>*

Die Komponente „Anzeige“ erhält Kameradaten und Taktikbefehle und stellt diese auf einer Anzeige dar, damit der Nutzer diese ablesen kann. Erhält die Komponente Kameradaten, werden die Positionen der Roboter und des Balls auf einem Spielfeld angezeigt. Die Taktikbefehle werden in einer Befehlszeile unter dem Spielfeld dargestellt und zeigen die aktuellen Bewegungen und Strategien der Roboter.

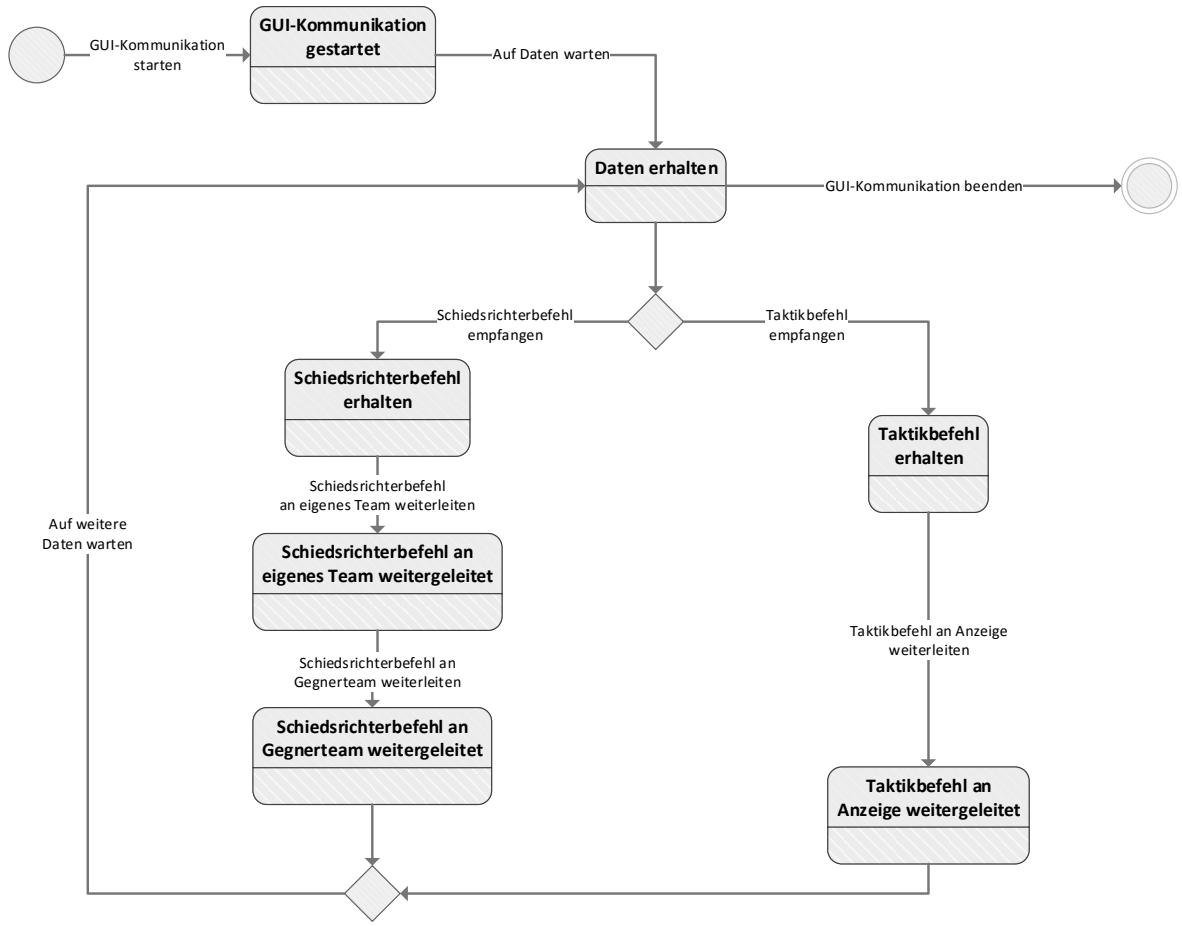


Abbildung 3.9: Zustandsdiagramm: *GUI-Kommunikation <C50>*

Diese Komponente ist die Schnittstelle zwischen GUI und KI auf Seiten der GUI. Erhält die Komponente Schiedsrichterbefehle, so werden diese an die KI weitergeleitet. Werden hingegen Taktikbefehle von der KI empfangen, werden diese an die Anzeige weitergeleitet.

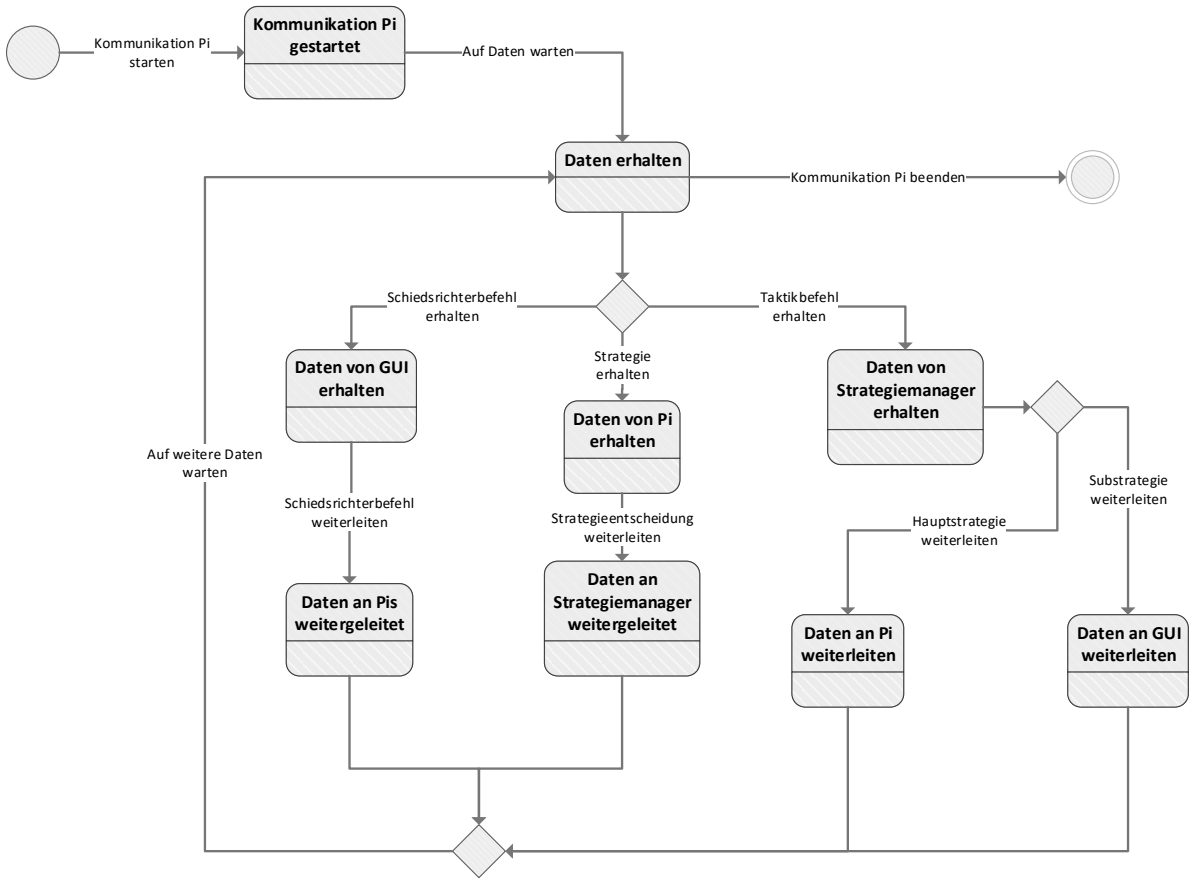


Abbildung 3.10: Zustandsdiagramm: *Kommunikation Pi <C90>*

Diese Komponente ist das Gegenstück zur GUI-Kommunikation auf der Seite der KI. Werden Schiedsrichterbefehle von der GUI erhalten, werden diese an die Pis weitergeleitet. Erhält die Komponente eine Nachricht von den Pis, ob eine Strategie ausgeführt werden kann oder nicht, wird diese an den Strategie-Manager weitergeleitet. Wird eine Hauptstrategie vom Strategie-Manager erhalten, wird diese an die Raspberry Pis weitergeleitet. Handelt sich bei der empfangenen Nachricht vom Strategie-Manager um eine Substrategie, so wird diese an die GUI weitergeleitet.

# 4 Verteilungsentwurf

In der nachfolgenden Abbildung wird die Verteilung der Komponenten und Devices über ein Verteilungsdiagramm visualisiert. Kernelement ist unser Computer. Über ihn wird die GUI ausgeführt sowie die Kommunikation zu unserem, wie auch dem Gegenerteam. Die Kommunikation von unserem Computer zu den Raspberry Pis funktioniert per TCP, so auch die Kommunikation zum Gegnerteam. Unsere Pis kommunizieren ihre Befehle per WLAN an die Lego Mindstorm EV3 Roboter. Die Pis aus unserem Team hingegen nutzen UDP um sich untereinander über Strategien auszutauschen.

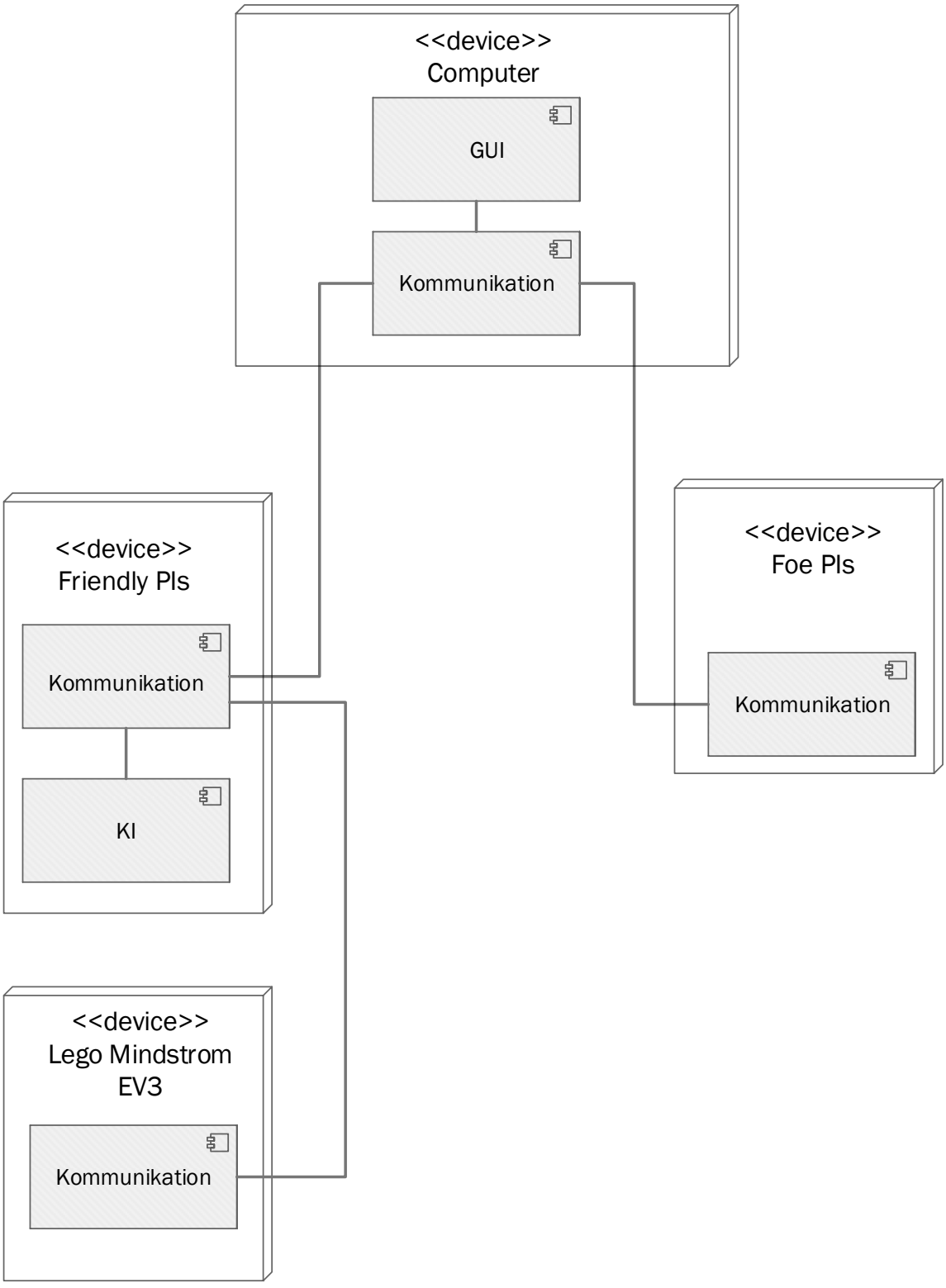


Abbildung 4.1: Verteilungsdiagramm

# 5 Implementierungsentwurf

In diesem Kapitel wird der Feinentwurf unserer Software durch Paketdiagramme und Klassendiagramme visualisiert. Die Hauptkomponente <GUI> $\langle C10 \rangle$ und <KI> $\langle C70 \rangle$ werden in den Abschnitts 5.1 und 5.7 durch Paketdiagramme visualisiert. Diese Paketdiagramme stellen die Struktur der Hauptkomponenten und die Abhängigkeiten zwischen den Paketen dar. Außerdem werden diese Pakete durch Klassendiagramme visualisiert und genauer erläutert.

## 5.1 Implementierung von Komponente $\langle C10 \rangle$: <GUI>:

Im folgenden Abschnitt wird die Implementierung der Komponente <GUI> $\langle C10 \rangle$ erläutert.

### 5.1.1 Paketdiagramm

Das folgende Paketdiagramm gibt einen Überblick über die Struktur der Komponente <GUI> und stellt die Abhängigkeiten zwischen den Paketen dar:

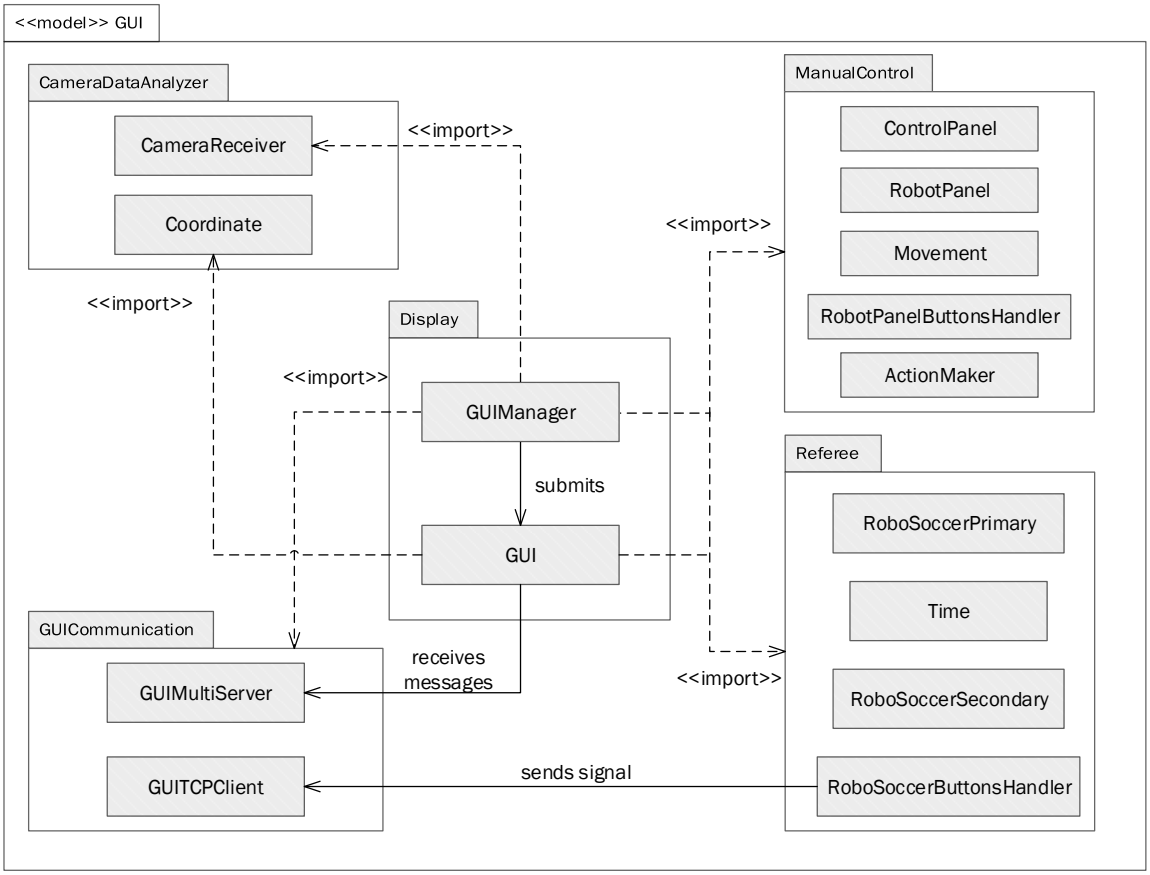


Abbildung 5.1: Paketdiagramm: *GUI* $\langle C10 \rangle$

Die Erläuterung der einzelnen Paketen und der Klassen sind in den folgenden Unterkapiteln zu finden.

## 5.2 Implementierung von Komponente $\langle C20 \rangle$: <Schiedsrichter>:

Im folgenden Abschnitt wird die Implementierung der Komponente Schiedsrichter $\langle C20 \rangle$ erläutert.

### 5.2.1 Paket-/Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <Schiedsrichter>:

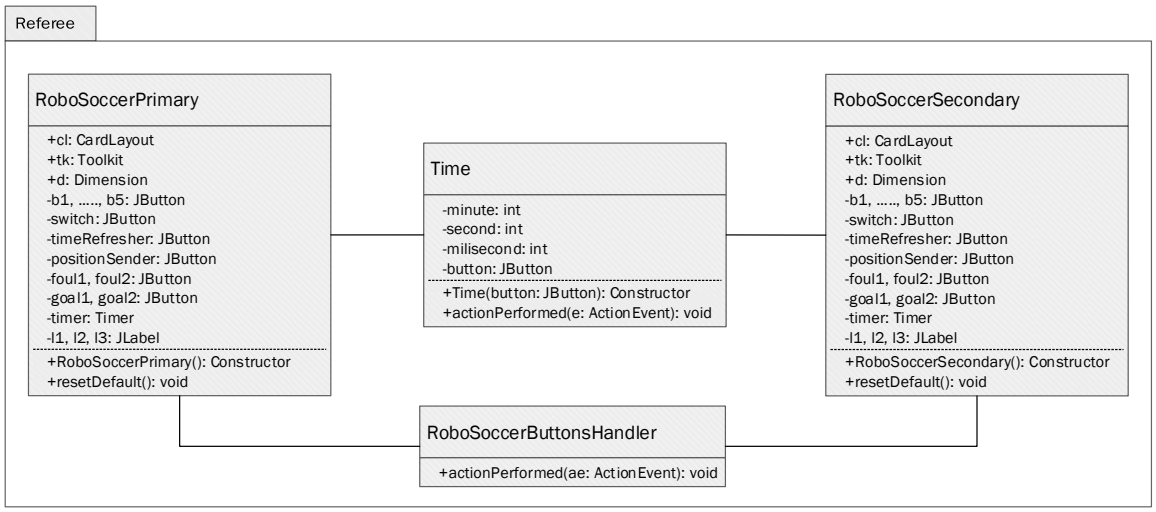


Abbildung 5.2: Klassendiagramm: *Schiedsrichter* $\langle C20 \rangle$

### 5.2.2 Erläuterung

**RoboSoccerPrimary**$\langle CL10 \rangle$

#### Aufgabe

Steuerung des Spiels ermöglichen, wenn unser Team an der linken Seite des Spielfeldes startet.

#### Attribute

CardLayout cl: Das CardLayout sorgt dafür, dass alle hintereinander liegenden Container in der gleichen Größe angezeigt werden, um ein Springen durch Redimensionierung der GUI zu verhindern.

Dimension d: Diese Variable speichert die Bildschirmgröße.

JButton b1, ....., b5: Buttons zum Ergänzen der GUI-Darstellung.

JButton switch: Button zum Umschalten des Startseite-Panel.
JButton timeRefresher: Button zum Aktualisieren der Spielzeit.
JButton positionSender: Button zum Senden der Positionsdaten.
JButton foul1, foul2: Button zum Zählen der Fouls.
JButton goal1, goal2: Button zum Zählen der Tore.
JLabel l1, l2, l3: Labels zum Ergänzen der GUI-Darstellung

**Operationen**

RoboSoccerPrimary(): Diese Operation hat die Aufgabe, das Panel zu erzeugen.
resetDefault(): Diese Operation hat die Aufgabe, das Panel wiederherzustellen.

**Kommunikationspartner**

Kein

---

**RoboSoccerSecondary**$\langle CL20 \rangle$

**Aufgabe**

Steuerung des Spiels ermöglichen, wenn unser Team an der rechten Seite des Spielfeldes startet.

**Attribute**

CardLayout cl: Das CardLayout sorgt dafür, dass alle hintereinander liegenden Container in der gleichen Größe angezeigt werden, um ein Springen durch Redimensionierung der GUI zu verhindern.
Dimension d: Diese Variable speichert die Bildschirmgröße.
JButton b1, ....., b5: Buttons zum Ergänzen der GUI-Darstellung.
JButton switch: Button zum Umschalten des Startseite-Panel.
JButton timeRefresher: Button zum Aktualisieren der Spielzeit.
JButton positionSender: Button zum Senden der Positionsdaten.
JButton foul1, foul2: Button zum Zählen der Fouls.
JButton goal1, goal2: Button zum Zählen der Tore.
JLabel l1, l2, l3: Labels zum Ergänzen der GUI-Darstellung

**Operationen**

RoboSoccerSecondary(): Diese Operation hat die Aufgabe, das Panel zu erzeugen.
resetDefault(): Diese Operation hat die Aufgabe, das Panel wiederherzustellen.

**Kommunikationspartner**

Kein

---

**Time**$\langle CL30 \rangle$

**Aufgabe**

Messung der Spielzeit.

**Attribute**

int minute: Diese Variable speichert die Minuten.

int second: Diese Variable speichert die Sekunden.

int milisecond: Diese Variable speichert die Millisekunden.

JButton button: Button zum Starten des Timers.

**Operationen**

Time(button: JButton): Diese Operation hat die Aufgabe, den Timer-Button zu erzeugen.

actionPerformed(e: ActionEvent): Diese Operation hat die Aufgabe, den Timer-Button zu starten oder zu stoppen.

**Kommunikationspartner**

Kein

---

**RoboSoccerButtonsHandler**$\langle CL40 \rangle$

**Aufgabe**

Steuerung des Spiels ermöglichen.

**Attribute**

kein

**Operationen**

actionPerformed(ae: ActionEvent): Diese Operation hat die Aufgabe, das Betätigen einer Schaltfläche zu erfassen.

**Kommunikationspartner**

Kein

## 5.3 Implementierung von Komponente $\langle C30 \rangle$: <Anzeige>:

Im folgenden Abschnitt wird die Implementierung der Komponente <Anzeige> $\langle C30 \rangle$ erläutert.

### 5.3.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <Anzeige>:

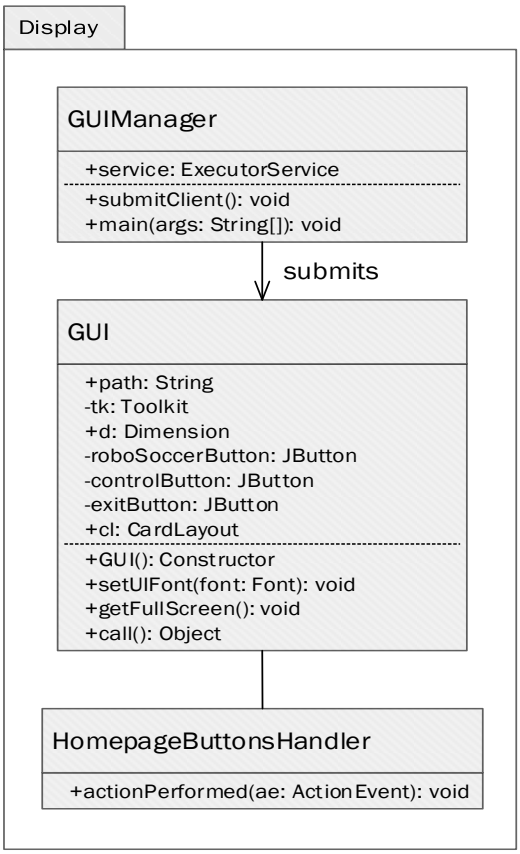


Abbildung 5.3: Klassendiagramm: *Anzeige* $\langle C30 \rangle$

### 5.3.2 Erläuterung

**GUIManager**⟨$CL50$⟩

**Aufgabe**

Ermöglichen des Multitasking innerhalb des GUI-Programms. Bestimmte Programmabschnitte können parallel ablaufen.

**Attribute**

service: ExecutorService

**Operationen**

submitClient(): Diese Operation dient dazu, dass alle IP Adressen im Lego-Labor gesucht werden und wenn ein Server mit dem Port 61002 gefunden wird, stellt dann das GUITCPClient-Programm eine Verbindung mit diesem Gerät her.

main(args: String[]): Die main-Methode hat die Aufgabe, das GUITCPMultiServer-Programm bereitzustellen und die GUI zu starten.

**Kommunikationspartner**

Kein

---

**GUI**⟨$CL60$⟩

**Aufgabe**

Darstellung der Benutzeroberfläche.

**Attribute**

path: String

tk: Toolkit

d: Dimension

JButton roboSoccerButton: Button zum Öffnen des RoboSoccer-Panels

JButton controlButton: Button zum Öffnen des manuellen Steuerung-Panels

JButton exitButton: Button zum Beenden der GUI

CardLayout cl: Das CardLayout sorgt dafür, dass alle hintereinander liegenden Container in der gleichen Größe angezeigt werden, um ein Springen durch Redimensionierung der GUI zu verhindern.

**Operationen**

GUI(): Konstruktor zum Erzeugen der GUI.

setUIFont(font: Font): Einsetzung einer Schriftart für das gesamte System.

getFullScreen(): Diese Operation ermöglicht eine Vollbildanzeige.

**Kommunikationspartner**

Kein

---

**HomepageButtonsHandler**$\langle CL70 \rangle$

### Aufgabe

Ermöglichen der Umstellung des Panels.

### Attribute

kein

### Operationen

actionPerformed(ae: ActionEvent): Diese Operation hat die Aufgabe, das Betätigen einer Schaltfläche zu erfassen.

### Kommunikationspartner

Kein

## 5.4 Implementierung von Komponente $\langle C40 \rangle$: <Kameradatenauswertung>:

Im folgenden Abschnitt wird die Implementierung der Komponente <Kameradatenauswertung> $\langle C40 \rangle$ erläutert.

### 5.4.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <Kameradatenauswertung>:

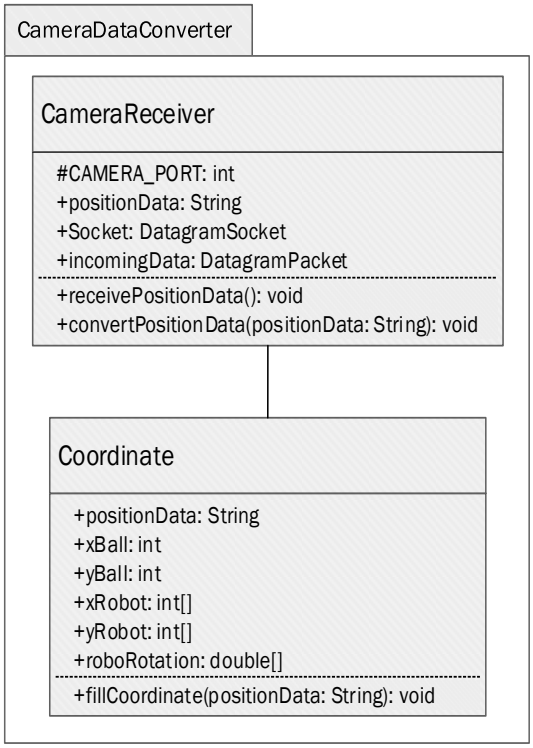


Abbildung 5.4: Klassendiagramm: *Kameradatenauswertung* $\langle C40 \rangle$

### 5.4.2 Erläuterung

**CameraReceiver**⟨$CL80$⟩

**Aufgabe**

Empfangen der Kameradaten.

**Attribute**

int CAMERAPORT: Der Port der Kamera.

String positionData: speichert das von der Kamera erhaltene String.

DatagramSocket Socket: UDP-Socket

DatagramPacket incomingData: Einlesen eingehender Datenströme.

**Operationen**

receivePositionData(): Diese Operation startet das CameraReceiver-Programm.

convertPositionData(positionData: String): Diese Operation ruft das Coordinate-Programm auf.

**Kommunikationspartner**

Kamera-Pi

---

**Coordinate**⟨$CL90$⟩

**Aufgabe**

Umwandeln der Kameradaten.

**Attribute**

String positionData: speichert das von der Kamera erhaltene String

int xBall: X-Koordinate des Balls.

int yBall: Y-Koordinate des Balls.

int xRobot[]: X-Koordinate des Roboters.

int yRobot[]: Y-Koordinate des Roboters.

double roboRotation[]: Rotation des Roboters.

**Operationen**

fillCoordinate(positionData: String): Diese Operation wandelt den String-Parameter in ein Koordinatensystem um.

**Kommunikationspartner**

Kein

## 5.5 Implementierung von Komponente $\langle C50 \rangle$: <GUI-Kommunikation>:

Im folgenden Abschnitt wird die Implementierung der Komponente <GUI-Kommunikation> $\langle C50 \rangle$ erläutert.

### 5.5.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <GUI-Kommunikation>:

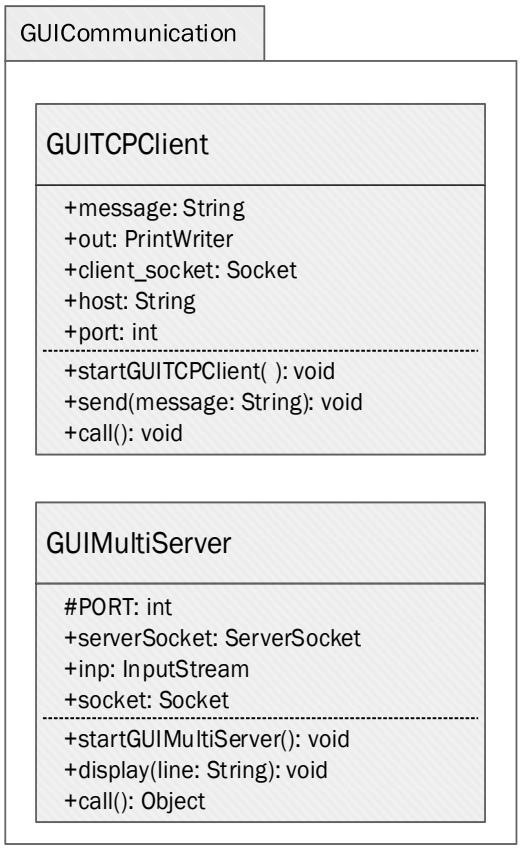


Abbildung 5.5: Paketdiagramm: *GUI-Kommunikation* $\langle C50 \rangle$

### 5.5.2 Erläuterung

**GUITCPClient**⟨$CL100$⟩

**Aufgabe**

Steuerungsbefehle an die Pis schicken.

**Attribute**

String message: speichert die Nachricht.

InputStream input: Einlesen eingehender Datenströme. PrintWriter out: zum Schreiben in das Socket.

String host: speichert die IP-Adresse der GUI.

int port: speichert das Port der GUI.

Socket clientSocket: ist das Client-Socket in der TCP-Kommunikation.

**Operationen**

startGUITCPClient( ): Diese Operation startet das Client-Programm.

send(message: String): Diese Operation dient zur Vorbereitung der zu sendenden Nachricht.

call(): Diese Operation ruft die startGUITCPClient()-Methode auf.

**Kommunikationspartner**

Pis

---

**GUIMultiServer**⟨$CL110$⟩

**Aufgabe**

Nachrichtenaustausch von den Pis empfangen.

**Attribute**

int PORT: Konstante zum Speichern des GUI-Ports.

InputStream input: Einlesen eingehender Datenströme.

ServerSocket serverSocket: serverSocket ist das bidirektionale Socket des Servers in der TCP-Kommunikation.

Socket socket: socket ist das Stream-Socket des Servers in der TCP-Kommunikation.

**Operationen**

startGUIMultiServer(): Diese Operation startet das GUIMultiServer-Programm.

display(line: String): Diese Operation stellt die empfangenen Nachrichten in der Konsole dar.

call(): Diese Operation ruft die startGUIMultiServer()-Methode auf.

**Kommunikationspartner**

Pis

## 5.6 Implementierung von Komponente $\langle C60 \rangle$: <Manuelle Steuerung>:

Im folgenden Abschnitt wird die Implementierung der Komponente <Manuelle Steuerung> $\langle C60 \rangle$ erläutert.

### 5.6.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <Manuelle Steuerung>:

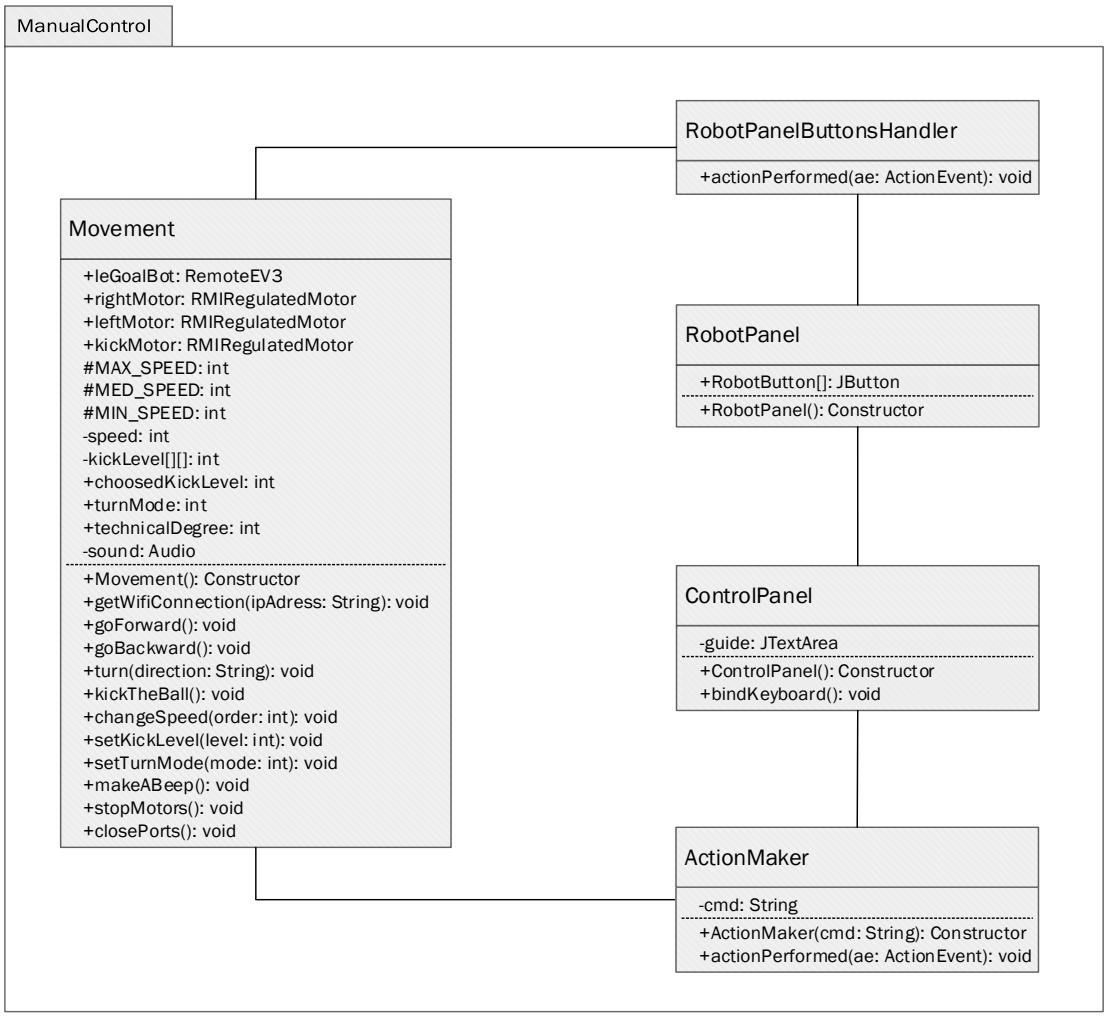


Abbildung 5.6: Paketdiagramm: *Manuelle Steuerung* $\langle C60 \rangle$

### 5.6.2 Erläuterung

**Movement**⟨$CL120$⟩

#### Aufgabe

Bestimmte befehle an die Roboter senden.

#### Attribute

RemoteEV3 leGoalBot: Roboter

RMIRegulatedMotor rightMotor: Rechter Bewegungsmotor.

RMIRegulatedMotor leftMotor: Linker Bewegungsmotor.

RMIRegulatedMotor kickMotor: Armmotor.

int MAXSPEED: Diese Konstante speichert die Maximale Geschwindigkeit.

int MEDSPEED: Diese Konstante speichert die Mittlere Geschwindigkeit.

int MINSPEED: Diese Konstante speichert die Minimale Geschwindigkeit.

int speed: Diese Variable speichert die aktuelle Geschwindigkeit.

int kickLevel[][]: Dieses Array speichert die Schießstärken.

int choosedKickLevel: Diese Variable speichert die Aktuelle Schießstärke.

int turnMode: Diese Variable speichert das Drehung-Modus.

int technicalDegree: Diese Variable speichert die Rotation.

Audio sound: Signalton.

#### Operationen

Movement(): Konstuktor

getWifiConnection(ipAdress: String): Diese Operation stellt eine Verbindung mit dem Roboter her.

goForward(): Roboter fährt vorwärts.

goBackward(): Roboter fährt rückwärts.

turn(direction: String): Roboter dreht sich nach links oder nach rechts.

kickTheBall(): Roboter schießt den Ball.

changeSpeed(order: int): Geschwindigkeit des Roboters ändern.

setKickLevel(level: int): Schießstärke ändern.

setTurnMode(mode: int): Drehung-Modus ändern.

makeABeep(): Signalton abspielen.

StopMotors(): Diese Operation stoppt die Motoren.

closePorts(): Diese Operation schließt die Ports der Roboter-Motoren.

#### Kommunikationspartner

Roboter

**RobotPanel**⟨$CL130$⟩

**Aufgabe**

Roboter Auswahl.

**Attribute**

JButton RobotButton[]: Array zum Speichern der Roboter-Buttons.

**Operationen**

RobotPanel(): Konstruktor zum Erzeugen des Roboter-Panels.

**Kommunikationspartner**

Roboter

---

**ControlPanel**⟨$CL140$⟩

**Aufgabe**

Ermöglichen der Einstellungen zum Manuellen Steuern des Spiels.

**Attribute**

JTextArea guide: guide ist eine Anleitung zum Benutzen der manuellen Steuerung.

**Operationen**

ControlPanel(): Konstruktor zum Erzeugen des Panels.

bindKeyboard(): Diese Operation hat die Aufgabe, ein Programm, mit dem eine Folge von Tastendrücken an eine kürzere Tastenkombination oder eine einzelne Taste gebunden werden, zu starten.

**Kommunikationspartner**

Kein

---

**AktionMaker**⟨$CL150$⟩

**Aufgabe**

Erkennen des Tastaturtaste-relevanten Befehls.

**Attribute**

String cmd: cmd ist die ID der Taste.

**Operationen**

AktionMaker(cmd: String): Konstruktor zum Erkennen des Tastaturtaste-relevanten Befehls.

actionPerformed(ae: ActionEvent): Diese Operation hat die Aufgabe, das Drücken einer Tastaturtaste zu erfassen.

### Kommunikationspartner

Roboter

---

**RobotButtonsHandler**⟨$CL160$⟩

### Aufgabe

Ermöglichen der Umstellung des Panel.

### Attribute

kein

### Operationen

actionPerformed(ae: ActionEvent): Diese Operation hat die Aufgabe, das Betätigen einer Schaltfläche zu erfassen.

### Kommunikationspartner

Roboter

## 5.7 Implementierung von Komponente $\langle C70 \rangle$: <KI>:

Im folgenden Abschnitt wird die Implementierung der Komponente <KI> $\langle C70 \rangle$ erläutert.

### 5.7.1 Paketdiagramm

Das folgende Paketdiagramm gibt einen Überblick über die Struktur der Komponente <KI> und stellt die Abhängigkeiten zwischen den Paketen dar:

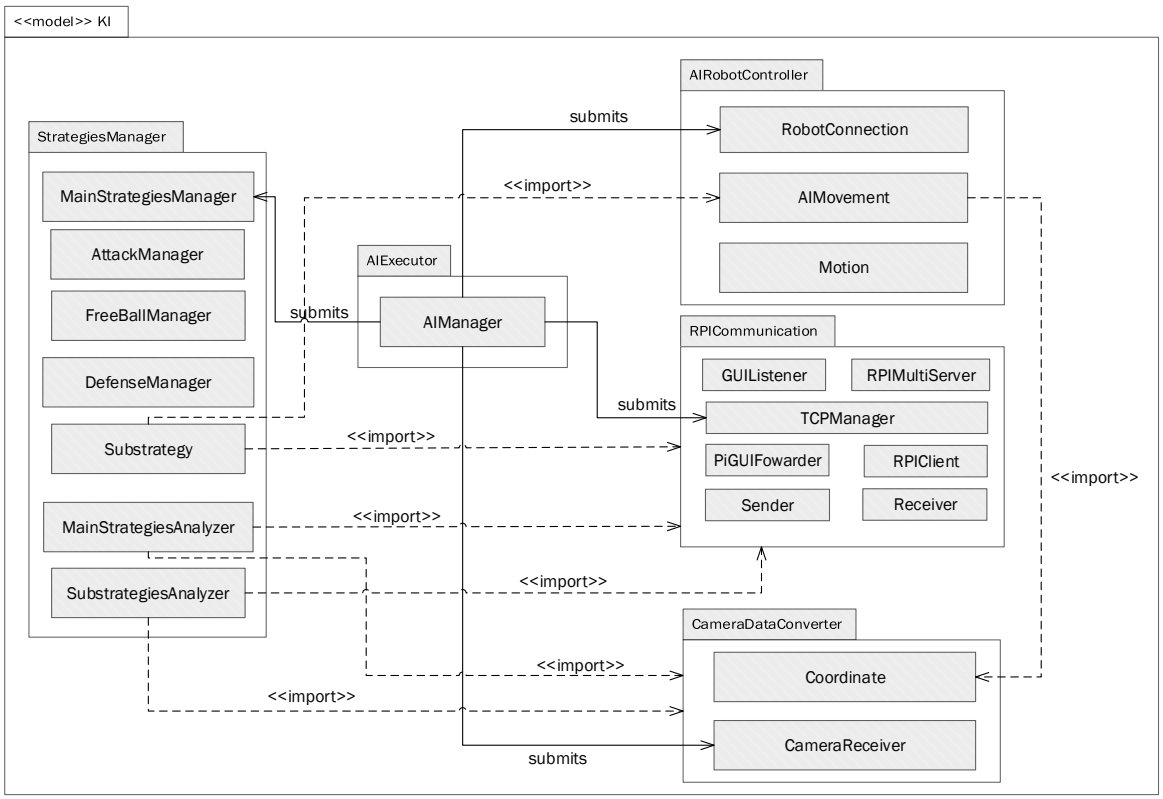


Abbildung 5.7: Paketdiagramm: *KI* $\langle C70 \rangle$

Die Erläuterung der einzelnen Paketen und der Klassen sind in den folgenden Unterkapiteln zu finden.

## 5.8 Implementierung von Komponente $\langle C80 \rangle$: <Kameradatenauswertung>:

Im folgenden Abschnitt wird die Implementierung der Komponente <Kameradatenauswertung> $\langle C80 \rangle$ erläutert.

### 5.8.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <Kameradatenauswertung>:

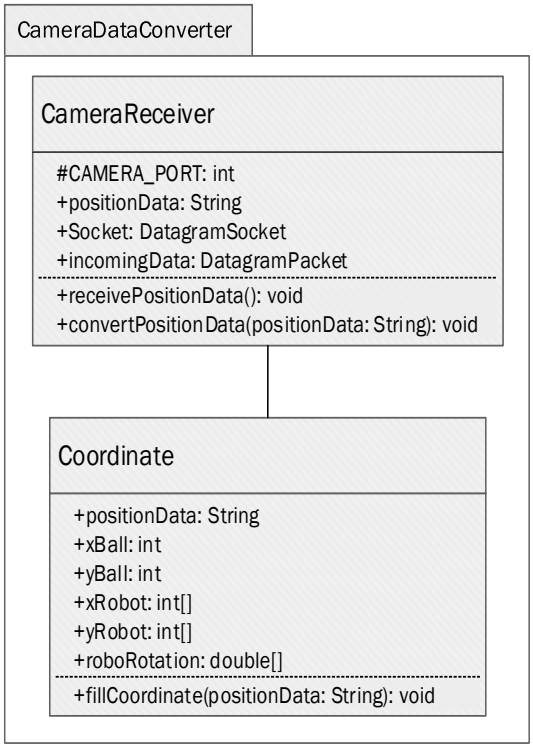


Abbildung 5.8: Klassendiagramm: *Kameradatenauswertung* $\langle C80 \rangle$

### 5.8.2 Erläuterung

**CameraReceiver**⟨$CL170$⟩

#### Aufgabe

Empfangen der Kameradaten.

#### Attribute

int CAMERAPORT: Der Port der Kamera.

String positionData: speichert das von der Kamera erhaltene String.

DatagramSocket socket: socket ist das UDP-Socket

DatagramPacket incomingData: Einlesen eingehender Datenströme.

#### Operationen

receivePositionData(): Diese Operation startet das CameraReceiver-Programm.

convertPositionData(positionData: String): Diese Operation ruft das Coordinate-Programm auf.

#### Kommunikationspartner

Kamera-Pi

---

**Coordinate**⟨$CL180$⟩

#### Aufgabe

Umwandeln der Kameradaten in ein Koordinatensystem.

#### Attribute

String positionData: speichert das von der Kamera erhaltene String

int xBall: X-Koordinate des Balls.

int yBall: Y-Koordinate des Balls.

int xRobot[]: X-Koordinate des Roboters.

int yRobot[]: Y-Koordinate des Roboters.

double roboRotation[]: Rotation des Roboters.

#### Operationen

fillCoordinate(positionData: String): Diese Operation wandelt den String-Parameter in ein Koordinatensystem um.

#### Kommunikationspartner

Kein

## 5.9 Implementierung von Komponente $\langle C90 \rangle$: <Kommunikation Pi>:

Im folgenden Abschnitt wird die Implementierung der Komponente <Kommunikation Pi> $\langle C90 \rangle$ erläutert.

### 5.9.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <Kommunikation Pi>:

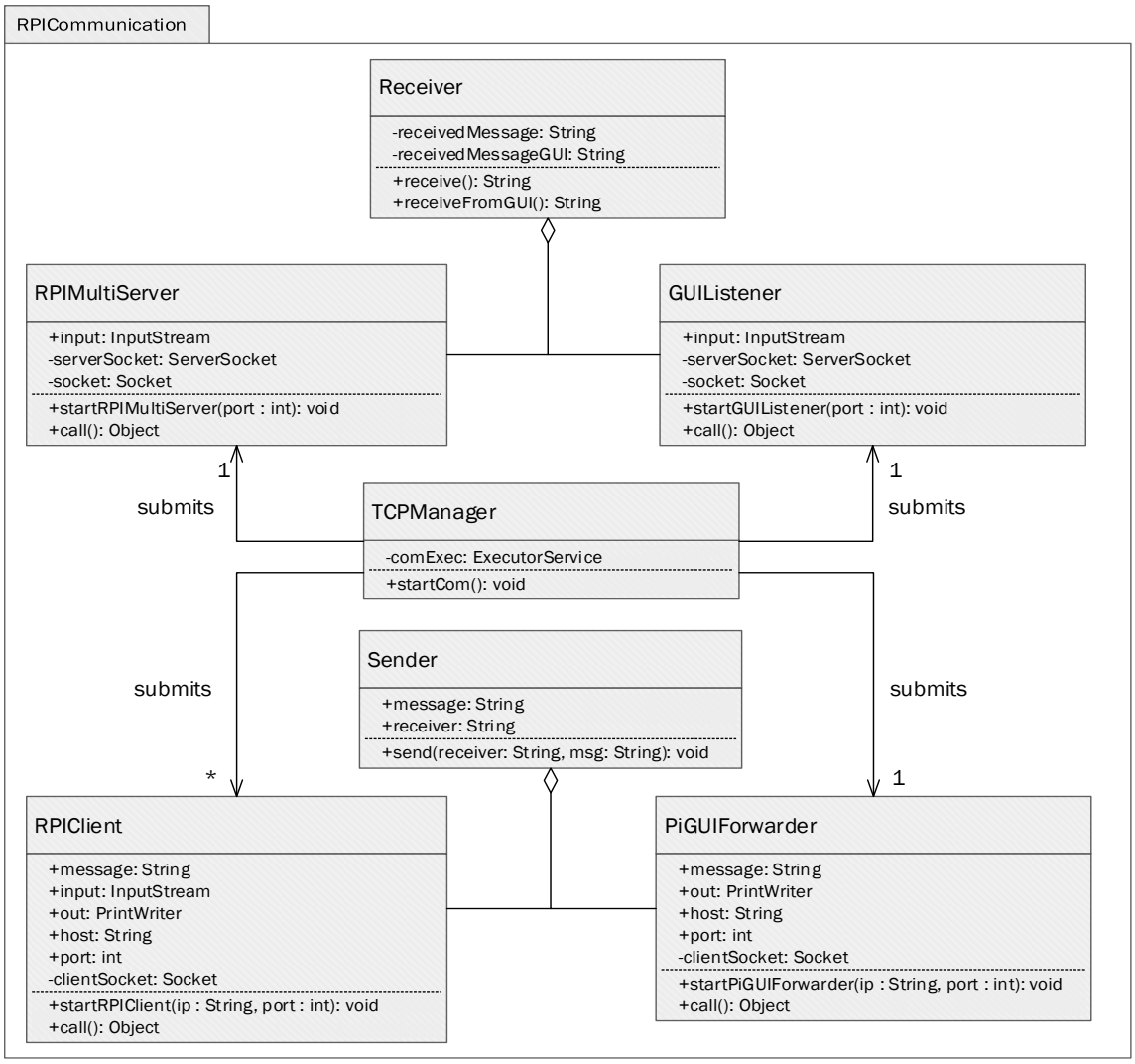


Abbildung 5.9: Klassendiagramm: *Kommunikation Pi* $\langle C90 \rangle$

### 5.9.2 Erläuterung

**TCPManager**⟨$CL190$⟩

**Aufgabe**

Verwaltung der TCP-Kommunikation.

**Attribute**

ExecutorService comExec: ExecutorService ist ein Interface zur parallelen Ausführung von Callable-Klassen.

**Operationen**

startCom(): Diese Operation startet das TCPManager-Programm.

**Kommunikationspartner**

Kein

---

**Sender**⟨$CL200$⟩

**Aufgabe**

Vorbereitung der zu sendenden Nachricht.

**Attribute**

String message: Diese Variable speichert die Nachricht.

String receiver: Diese Variable speichert die Name des Empfängers.

**Operationen**

send(receiver: String, msg: String): Diese Operation bereitet die zu sendenden Nachricht vor.

**Kommunikationspartner**

Kein

---

**RPIClient**⟨$CL210$⟩

**Aufgabe**

Senden der Befehle an die Pis.

**Attribute**

String message: Diese Variable speichert die Nachricht.

InputStream input: Einlesen eingehender Datenströme. PrintWriter out: out dient zum Schreiben in das Socket.

String host: Diese Variable speichert die IP-Adresse der GUI.

int port: Diese Variable speichert das Port der GUI.

Socket clientSocket: clientSocket ist das Client-Socket in der TCP-Kommunikation.

#### Operationen

startRPIClient(ip : String, port : int): Diese Operation startet das RPIClient-Programm.

call(): Diese Operation ruft die startRPIClient-Methode auf.

#### Kommunikationspartner

Pis

---

**PiGUIForwarder**⟨$CL220$⟩

#### Aufgabe

Weiterleiten der Befehle an die GUI.

#### Attribute

PrintWriter out: Diese Variable dient zum Schreiben in das Socket.

String host: Diese Variable speichert die IP-Adresse der GUI.

int port: Diese Variable speichert das Port der GUI.

Socket clientSocket: clientSocket ist das Client-Socket in der TCP-Kommunikation.

#### Operationen

startPiGUIForwarder(ip : String, port : int): Diese Operation startet das PiGUIForwarder-Programm.

call(): Diese Operation ruft die startPiGUIForwarder(ip : String, port : int)-Methode auf.

#### Kommunikationspartner

GUI

---

**Receiver**⟨$CL230$⟩

#### Aufgabe

Zurückgeben der empfangenen Befehle von den Pis und der GUI.

#### Attribute

String receivedMessage: Diese Variable dient zum Speichern der von den Pis empfangenen Nachrichten.

String receivedMessageGUI: Diese Variable dient zum Speichern der von der GUI empfangenen Nachrichten.

#### Operationen

receive(): Diese Operation gibt die von den Pis empfangenen Nachrichten zurück.

receiveFromGUI(): Diese Operation gibt die von der GUI empfangenen Nachrichten zurück.

#### Kommunikationspartner

Kein

---

**RPIMultiServer**⟨$CL240$⟩

**Aufgabe**

Empfangen der Befehle von den Pis.

**Attribute**

InputStream input: Einlesen eingehender Datenströme. ServerSocket serverSocket: serverSocket ist das bidirektionale Socket des Servers in der TCP-Kommunikation. Socket socket: socket ist das Stream-Socket des Servers in der TCP-Kommunikation.

**Operationen**

startRPIMultiServer(port : int): Diese Operation startet das RPIMultiServer-Programm.
call(): Diese Operation ruft die startRPIMultiServer(port : int)-Methode auf.

**Kommunikationspartner**

Pis

---

**GUIListener**⟨$CL250$⟩

**Aufgabe**

Empfangen der Befehle von der GUI.

**Attribute**

InputStream input: Einlesen eingehender Datenströme.
ServerSocket serverSocket: serverSocket ist das bidirektionale Socket des Servers in der TCP-Kommunikation.
Socket socket: socket ist das Stream-Socket des Servers in der TCP-Kommunikation.

**Operationen**

startGUIListener(port : int): Diese Operation startet das GUIListener-Programm.
call(): Diese Operation ruft die startGUIListener(port : int)-Methode auf.

**Kommunikationspartner**

GUI

## 5.10 Implementierung von Komponente $\langle C100 \rangle$: <Strategie-Manager>:

Im folgenden Abschnitt wird die Implementierung der Komponente <Strategie-Manager> $\langle C100 \rangle$ erläutert.

### 5.10.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <Strategie-Manager>:

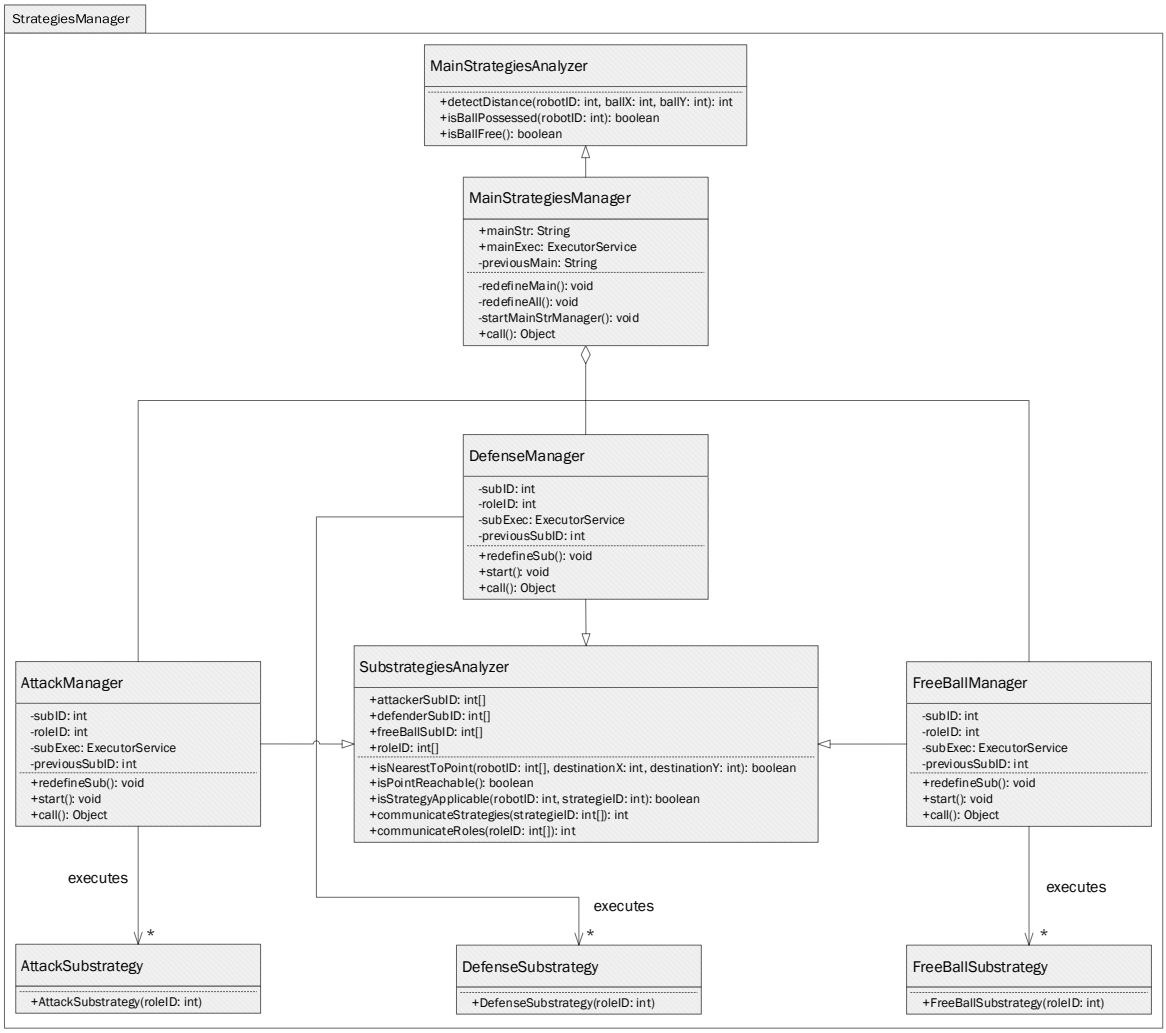


Abbildung 5.10: Klassendiagramm: *Strategie-Manager* $\langle C100 \rangle$

### 5.10.2 Erläuterung

**MainStrategiesManager**⟨$CL260$⟩

**Aufgabe**

Verwaltung der Hauptstrategien.

**Attribute**

String mainStr: Diese Variable speichert die Name der Substrategie.

ExecutorService mainExec: ExecutorService ist ein Interface zur parallelen Ausführung von Callable-Klassen.

String previousMain: Diese Variable speichert die Name der vorherigen Substrategie.

**Operationen**

redefineMain(): Diese Operation stoppt alle aktive Hauptstrategien-Threads.

redefineAll(): Diese Operation stoppt alle aktive Threads.

startMainStrManager(): Diese Operation startet das MainStrategiesManager-Programm.

call(): Diese Operation ruft die startMainStrManager()-Methode auf.

**Kommunikationspartner**

Kein

---

**MainStrategiesAnalyzer**⟨$CL270$⟩

**Aufgabe**

Analyse der ausführbaren Hauptstrategien.

**Attribute**

kein

**Operationen**

detectDistance(robotID: int, ballX: int, ballY: int): Diese Operation erkennt die Ballentfernung.

isBallPossessed(robotID: int): Diese Operation überprüft ob der Ball von unserem Roboter besitzt ist.

isBallFree(): überprüft ob der Ball frei ist.

**Kommunikationspartner**

Pis

---

**SubStrategiesAnalyzer**⟨$CL280$⟩

**Aufgabe**

Analyse der ausführbaren Substrategien.

**Attribute**

int[] attackSubID: Dieses Array speichert die IDs der Angriff-Substrategie

int[] defenseSubID: Dieses Array speichert die IDs der Abwehr-Substrategie

int[] freeBallSubID: Dieses Array speichert die IDs der FreeBall-Substrategie

int[] roleID: Dieses Array speichert die IDs der Rolle

**Operationen**

isNearestToPoint(robotID: int[], destinationX: int, destinationY: int): Diese Operation überprüft ob ein Roboter näher zum Ball ist.

isPointReachable(): Diese Operation überprüft ob ein Punkt erreichbar ist.

isStrategyApplicable(robotID: int, strategieID: int): Diese Operation überprüft ob eine Substrategie ausführbar ist.

communicateStrategies(strategieID: int[]): Diese Operation kommuniziert die Substrategie mit den anderen Pis.

communicateRoles(roleID: int[]): Diese Operation kommuniziert die Rolle mit den anderen Pis.

**Kommunikationspartner**

Pis

---

**DefenseManager**$\langle CL290 \rangle$

**Aufgabe**

Verwaltung der Abwehr-Substrategie

**Attribute**

int subID: Diese Variable speichert die ID der Substrategie.

int roleID: Diese Variable speichert die ID der Rolle.

ExecutorService subExec: ExecutorService ist ein Interface zur parallelen Ausführung von Callable-Klassen.

int previousSubID: Diese Variable speichert die ID der vorherigen Substrategie.

**Operationen**

redefineSub(): Diese Operation stoppt alle aktive Threads.

start(): Diese Operation startet das DefenseManger-Programm.

call(): Diese Operation ruft die start()-Methode auf.

**Kommunikationspartner**

Kein

**AttackManager**$\langle CL300 \rangle$

**Aufgabe**

Verwaltung der Angriff-Substrategie

### Attribute

int subID: Diese Variable speichert die ID der Substrategie.

int roleID: Diese Variable speichert die ID der Rolle.

ExecutorService subExec: ExecutorService ist ein Interface zur parallelen Ausführung von Callable-Klassen.

int previousSubID: Diese Variable speichert die ID der vorherigen Substrategie.

### Operationen

redefineSub(): Diese Operation stoppt alle aktive Threads.

start(): Diese Operation startet das AttackManager-Programm.

call(): Diese Operation ruft die start()-Methode auf.

### Kommunikationspartner

Kein

---

**FreeBallManager**⟨$CL310$⟩

### Aufgabe

Verwaltung der FreeBall-Substrategie

### Attribute

int subID: Diese Variable speichert die ID der Substrategie.

int roleID: Diese Variable speichert die ID der Rolle.

ExecutorService subExec: ExecutorService ist ein Interface zur parallelen Ausführung von Callable-Klassen.

int previousSubID: Diese Variable speichert die ID der vorherigen Substrategie.

### Operationen

redefineSub(): Diese Operation stoppt alle aktive Threads.

start(): Diese Operation startet das FreeBallManager-Programm.

call(): Diese Operation ruft die start()-Methode auf.

### Kommunikationspartner

Kein

---

**FreeBallSubstrategy**⟨$CL320$⟩

### Aufgabe

FreeBallSubstrategy durchführen.

### Attribute

kein

**Operationen**

FreeBallSubstrategy(roleID: int): Konstruktor zum Starten der FreeBall-Substrategie.

**Kommunikationspartner**

Roboter, Pis

---

**DefenseSubstrategy**⟨$CL330$⟩

**Aufgabe**

DefenseSubstratey durchführen.

**Attribute**

kein

**Operationen**

DefenseSubstrategy(roleID: int): Konstruktor zum Starten der Abwehr-Substrategie.

**Kommunikationspartner**

Roboter, Pis

---

**AttackSubstrategy**⟨$CL340$⟩

**Aufgabe**

AttackSubstrategy durchführen.

**Attribute**

kein

**Operationen**

AttackSubstrategy(roleID: int): Konstruktor zum Starten der Angriff-Substrategie.

**Kommunikationspartner**

Roboter, Pis

## 5.11 Implementierung von Komponente $\langle C110 \rangle$: <KI Bewegung>:

Im folgenden Abschnitt wird die Implementierung der Komponente <KI Bewegung> $\langle C110 \rangle$ erläutert.

### 5.11.1 Klassendiagramm

Das folgende Diagramm gibt einen Überblick über die Struktur der Komponente <KI Bewegung>:

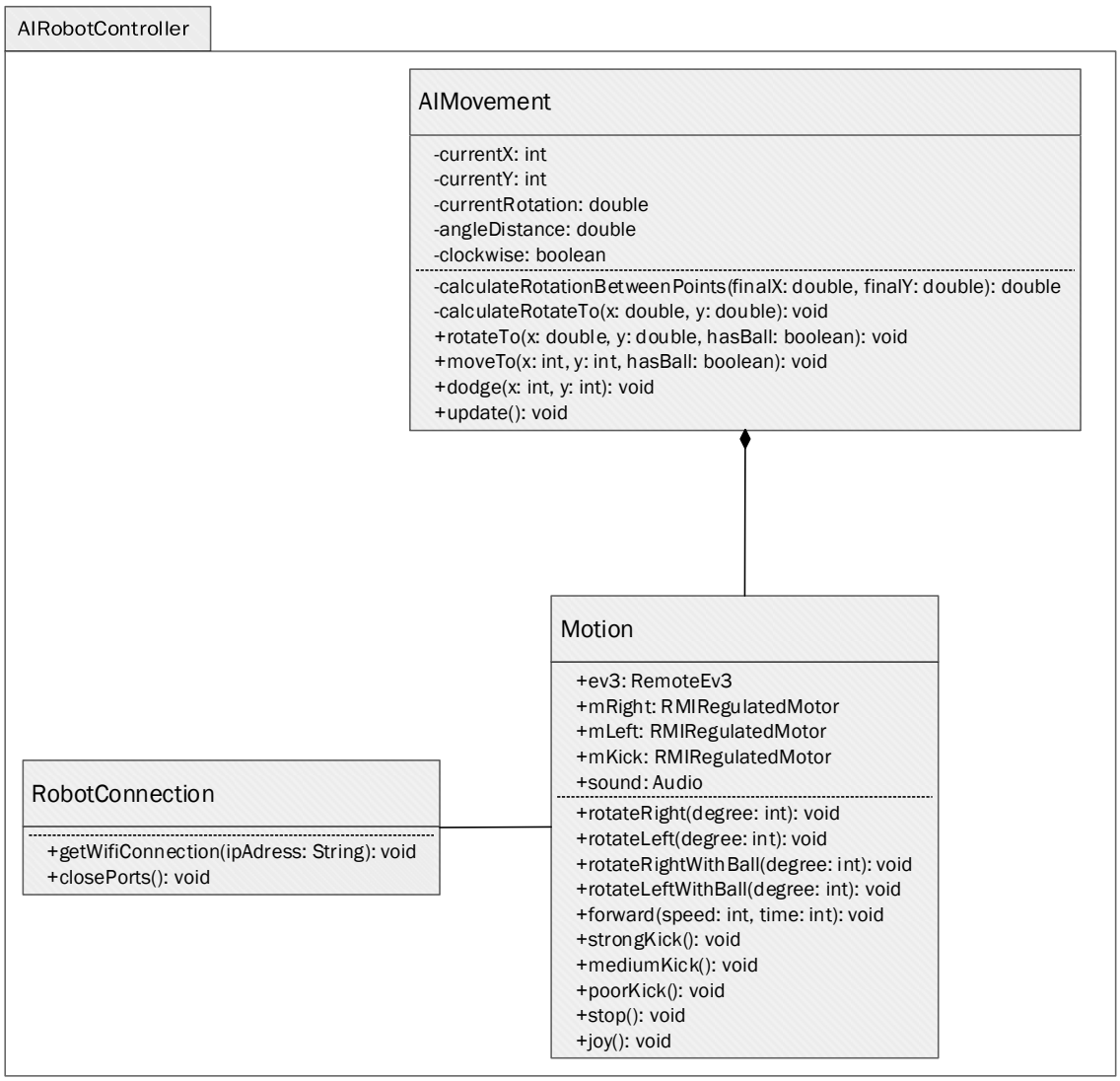


Abbildung 5.11: Klassendiagramm: *KI Bewegung* $\langle C110 \rangle$

### 5.11.2 Erläuterung

**AIMovement**⟨$CL350$⟩

#### Aufgabe

Bewegungsmethoden für die Roboter anhand der Kameradaten bereitstellen.

#### Attribute

int currentX: aktuelle X-Koordinate des Roboters.

int currentY: aktuelle Y-Koordinate des Roboters.

double currentRotation: aktuelle Rotation des Roboters.

double angleDistance: Distanz von der derzeitigen Rotation zur Rotation des Ziels.

boolean clockwise: Rotation des Ziels schneller bei Drehung im oder gegen den Uhrzeigersinn zu erreichen.

#### Operationen

calculateRotationBetweenPoints(finalX: double, finalY: double): Berechnung der Rotation vom Roboterstandort zum Ziel.

calculateRotateTo(x: double, y: double): Berechnung des Winkels, den der Roboter drehen muss mit Berechnung der Richtung (Uhrzeigersinn).

rotateTo(x: double, y: double, hasBall: boolean): Drehung des Roboters Richtung Ziel.

moveTo(x: int, y: int, hasBall: boolean): Bewegung zum Ziel.

dodge(x: int, y: int): Ausweichen anderer Roboter.

update(): Aktualisieren der Positionsdaten.

#### Kommunikationspartner

Roboter

---

**Motion**⟨$CL360$⟩

#### Aufgabe

Bewegungsmethoden für die Roboter bereitstellen.

#### Attribute

RemoteEv3 ex3: Roboter

RMIRegulatedMotor mRight: Rechter Bewegungsmotor.

RMIRegulatedMotor mLeft: Linker Bewegungsmotor.

RMIRegulatedMotor mKick: Armmotor.

Audio sound: Signalton.

#### Operationen

rotateRight(degree: int): Roboter dreht bestimmten Winkel ohne Ball im Uhrzeigersinn.

rotateLeft(degree: int): Roboter dreht bestimmten Winkel ohne Ball gegen den Uhrzeigersinn.

rotateRightWithBall(degree: int): Roboter dreht bestimmten Winkel mit Ball im Uhrzeigersinn.
rotateLeftWithBall(degree: int): Roboter dreht bestimmten Winkel mit Ball gegen den Uhrzeigersinn.
forward(speed: int, time: int): Roboter fährt in bestimmter Geschwindigkeit eine bestimmte Zeit vorwärts.
strongKick(): Roboter schießt stark.
mediumKick(): Roboter schießt normal.
poorKick(): Roboter schießt schwach.
stop(): Roboter stoppt.
joy(): Roboter freut sich (zwei Kreisdrehungen).

**Kommunikationspartner**

Roboter

---

**RobotConnection**$\langle CL370 \rangle$

**Aufgabe**

Verbindung mit dem Roboter herstellen.

**Attribute**

kein

**Operationen**

getWifiConnection(ipAdress: String): Diese Operation stellt eine Verbindung mit dem Roboter her.
closePorts(): Diese Operation schließt die Ports der Roboter-Motoren.

**Kommunikationspartner**

Roboter

# 6 Datenmodell

In diesem Kapitel wird erläutert, welche dauerhaft zu speichernden Daten unsere Anwendung benötigt. Dies wird unter Verwendung eines UML-Klassendiagramms visualisiert.

## 6.1 Diagramm


Abbildung 6.1: Klassendiagramm: *Dauerhaft zu speichernde Daten*

## 6.2 Erläuterung

Durch das kleine UML-Klassendiagramm ist leicht zu erkennen, dass es bei unserer Anwendung nicht viele dauerhaft zu speichernde Daten gibt. Da Fußball ein dynamischer Sport ist, besitzen nur wenige Parameter permanent den gleichen Wert. Eine Klasse, die dauerhaft gleich bleiben sollte, ist unser Spielfeld. Hier würden sich Veränderungen stark auf unsere Roboter sowie deren Spielverhalten auswirken. Außerdem hat unsere künstliche Intelligenz einige langfristig zu speichernde Werte, um eine intelligente Strategie entwickeln zu können.

In der Klasse „Spielfeld“ werden alle in der Realität vorhandene Daten gespeichert, also die Spielfeldbegrenzung und die speziellen Bereiche des Feldes. Zudem werden besonders wichtige Koordinaten, wie der Startpunkt auf dem Feld, gespeichert. Andere auf dem Spielfeld liegende Bereiche sind der Mittelkreis, die Torkreise und natürlich die Tore. Eine vorherige Speicherung ermöglicht einen direkten Zugriff auf die Daten.

In unsere Klasse „Roboter KI“ sollen in erster Linie die Daten dauerhaft gespeichert werden, die wichtig für eine Entscheidungsfindung sind. Diese werden dann mit den agilen Parametern kombiniert, um eine optimale Strategie auszuwählen. Die jeweilige Strategie ist durch eine ID zu identifizieren. Vorher fest definiert sind unsere Bewegungs- und Aktionsmethoden sowie Rotationsmethoden.

Die KI nutzt die Daten des Spielfeldes zur Orientierung und zur Berechnung der Strategie.

# 7 Konfiguration

In diesem Kapitel werden alle Dateien aufgeführt, die die Kommunikation der einzelnen Komponenten über das Netzwerk konfigurieren. Diese umfassen die Kommunikation des ausführenden Computers zu den Raspberry Pis, die Kommunikation zwischen den Pis, die Einbindung der Kameradaten und die Übertragung über das UDP. Die Kommunikation für die Raspberry Pis untereinander läuft über TCP.

Die folgenden Dateien müssen alle in den Unterordnern „RPIComKILI“, „RPIComFILI“ und „RPIComOIN“ liegen und dürfen nicht verschoben werden. Für die Ausführung muss die Entwicklungsumgebung Eclipse vorliegen und das leJos EV3 Plug-In in der Version „0.9.1 beta“ für selbige Entwicklungsumgebung. Das Plug-In kann über den internen Marktplatz von Eclipse abgerufen und installiert werden. Damit Eclipse nutzbar ist, muss die Java JDK und die Java JRE in der Version „1.8.0_77“ auf dem Computer vorliegen. Die erwähnte LeJos EV3-Version muss außerdem auf der SD-Karte in den Robotern vorhanden sein. Ansonsten werden keine weiteren Programme benötigt.

In der Datei **„GUITCPClient.java“** wird eine Verbindung zwischen dem ausführenden Computer und den Raspberry Pis hergestellt. Dabei werden feste IP-Adressen der Pis benötigt. Die IP-Adressen werden durch ein Programm in der GUI automatisch gesucht.

In der Datei **„CameraReceiver.java“** wird eine Verbindung zur Spielfeldkamera hergestellt. Die Datei „CameraReceiver.java“ bekommt die Daten über den Port 61001. Die Kamera liefert Positionsdaten der sechs Roboter und des Balls sowie die Rotation der Roboter. Die Positionsdaten sehen wie folgt aus: Zunächst wird die Position des Balls zurückgegeben (int X-Koordinate, int Y-Koordinate). Anschließend folgen die Daten der Roboter mit folgendem Schema: (int Roboter-ID, int X-Koordinate, int Y-Koordinate, double Rotation).

In den Dateien **„RPIClient.java“** und **„RPIMultiServer.java“** wird das Server-Client System zwischen den Raspberry Pis konfiguriert. Für die Pis werden drei Ports benötigt, die zu diesem Zeitpunkt noch nicht feststehen. Die Ports können unter der Datei „RPIClient.java“ eingegeben werden. Außerdem wird ein Port für den Pi-Server benötigt. Dieser steht ebenfalls noch nicht fest und kann in der Datei „RPIMultiServer.java“ festgelegt werden. Die IP-Adresse des Empfängers wird in der Datei „RPIClient.java“ konfiguriert.

In der Datei **„GUIListener.java“** wird die Kommunikation zwischen den Raspberry Pis und der GUI seitens der Raspberry Pis konfiguriert. Die Datei stellt einen Listener bereit, der auf Befehle der GUI hört. Dies läuft über den Port 61002.

In der Datei **„GUIMultiserver.java“** wird die Kommunikation zwischen den Raspberry Pis und der GUI seitens der GUI konfiguriert. Die Datei stellt einen Server bereit, der empfangene Befehle durch die GUI an die Raspberry Pis weitergibt. Dies läuft über den Port 11110.

# 8 Änderungen gegenüber Fachentwurf

In diesem Abschnitt werden Änderungen gegenüber dem Fachentwurf kapitelweise erläutert. Dabei beziehen sich die Kapitelnummern auf den technischen Entwurf. Allgemein wurde sämtliche Kritik, die es als Rückmeldung zum Fachentwurf gab, umgesetzt.

Ferner wurde im ersten Kapitel die Einleitung dem technischen Entwurf angepasst und die Abbildung 1.2 um benötigte Funktionen erweitert. Dementsprechend ist auch der dazugehörige beschreibende Text erweitert worden.

Im sechsten Kapitel, dem Datenmodell, wurden einige Variablen der Klassen verändert. Der Spielstand wird von der Roboter KI nicht mehr benötigt und daher entfernt. Hinzugekommen sind jedoch eine StrategieID und die Rotationsmethoden für die Roboter.

Auch in der Konfiguration, beschrieben im siebten Kapitel, ist es zu Änderungen gekommen. Diese Veränderungen haben sich im Laufe der Programmierung ergeben oder sind auf spätere Absprachen mit dem gegnerischen Team für die Schiedsrichterfunktion zurückzuführen. Neu hinzugekommen sind die Datei GUIListener.java mit Port 61002 und die Datei GUIMultiserver.java mit Port 11110. Ferner haben sich die Dateinamen RaspPiClient.java in RPIClient.java, RaspPiServer.java in RPIMultiServer.java und PCClient.java in GUITCPClient.java geändert. Des weiteren enthält der GUITCPClient nun sechs statt ursprünglich drei IP-Adressen für die Roboter, da alle, und nicht nur die eigenen, vom Schiedsrichter Nachrichten empfangen können müssen. Außerdem wurde die Ordnerstruktur nochmals geändert. Alle Dateien, bis auf GUIMultiserver.java, liegen in den Ordnern RPIComFILI, RPIComKILI und RPIComOIN. Die übrige Datei ist in LeGoalGUIRemote abzulegen.

# 9 Erfüllung der Kriterien

Im folgenden Kapitel wird erneut auf die Muss-, Soll- und Kannkriterien zurückgegriffen und diese mit den Komponenten des dritten Kapitels in Verbindung gesetzt.

## 9.1 Musskriterien

Die folgenden Kriterien sind unabdingbar und müssen durch das Produkt erfüllt werden:

**RM1:** *Die Roboter müssen sich frei auf dem Spielfeld bewegen können.*
Die künstliche Intelligenz (C70), welche auf den Raspberry Pis der Roboter ausgeführt wird, kennt das gesamte Spielfeld. Die Pis können den Robotern mittels WLAN Befehle senden (C110), damit sich diese auf einen beliebigen Punkt des Spielfeldes bewegen können.

**RM2:** *Die Roboter müssen Befehle des jeweiligen Pis empfangen und ausführen können.*
Die Roboter empfangen mittels WLAN Befehle der Raspberry Pis (C70). Diese werden mit den Motoren (für Räder und „Arm“) sowie dem Lautsprecher des Roboters ausgeführt.

**RM3:** *Die Roboter müssen schießen und passen („anspielen“) können.*
Die Roboter können den Arm bewegen. Befindet sich der Ball im Arm, kann dieser durch eine ruckartige Bewegung geschoßen bzw. gepasst werden. Dabei kann der Arm (je nach Befehl) in unterschiedlichen Gradzahlen bewegt werden. Die Bewegungsbefehle gehen von der KI (C70/C110) aus.

**RM4:** *Die Roboter müssen Bälle entgegennehmen können.*
Um Bälle entgegennehmen zu können, muss sich der Roboter zu einem anderen drehen können. Die entsprechenden Befehle enthält der Roboter von der KI (C70/C110) seines Raspberry Pis.

**RM5:** *Die Pis müssen die Positionsdaten empfangen und verarbeiten können.*
Die künstliche Intelligenz (C70), welche auf den Raspberry Pis der Roboter liegt, muss die Positionsdaten der Kamera empfangen können und entsprechend umsetzen können (C70/C80). Die Daten, die von dem Raspberry Pi der Kamera empfangen werden, sind vor dieser Verarbeitung

unbrauchbar.

**RM6:** *Die Pis müssen untereinander kommunizieren, Strategien austauschen und Positionen mitteilen können. Sie sind das schwarmbasierte Team, welches die Roboter steuert.*
Die künstliche Intelligenz läuft auf den drei Raspberry Pis der Roboter und agiert schwarmbasiert. Mit den Positionsdaten des Balls wird sich für eine Hauptstrategie entschieden. Die KIs der einzelnen Roboter überprüfen danach, welche Substrategie dieser Hauptstrategie sie durchführen könnten (C70/C100) und tauschen diese Informationen untereinander aus (C70/C90). Die Substrategie, welche alle Roboter ausführen könnten, mit der höchsten Priorität wird gewählt.

**RM7:** *Die Pis müssen Steuerungsbefehle senden können.*
Die Raspberry Pis, auf denen die künstliche Intelligenz ausgeführt wird, müssen den Robotern Befehle senden können (C70/C110).

**RM8:** *Die KI muss die Regeln beachten.*
Die künstliche Intelligenz muss nach allen im Vorfeld mit dem gegnerischen Team festgelegten Regeln handeln (C70/C100/C110). Dabei wird beispielsweise der physische Kontakt der Roboter untereinander untersagt. Genaueres ist in den kompletten Regeln in Kapitel 1.1.1 des Fachentwurfs oder Kapitel 1.1.1 dieses Dokumentes nachzulesen.

**RM9:** *Die KI muss sich für Strategien entscheiden können.*
Siehe RM6.

**RM10:** *Die KI muss sich an Änderungen der Situation anpassen können.*
Siehe RM6. In bestimmten Intervallen bzw. Zeitpunkten wird auch während der Ausführung einer Strategie durch die künstliche Intelligenz überprüft (C70/C100), inwieweit diese noch ausführbar ist. Damit kann auf die Aktionen des Gegnerteams reagiert werden.

**RM11:** *Die Erstellung und Auslieferung eines Regelwerkes.*
Zum fertigen Produkt gehören die bereits erwähnten Regeln in schriftlicher Form.

**RM12:** *Die Erstellung einer GUI mit Schiedsrichterfunktionen. Diese beinhalten: Die Ahndung von Fouls, das Erkennen von Toren und das Pausieren und Stoppen des Spiels.*
Die graphische Benutzeroberfläche eines der beiden Teams wird zur Ausführung der Schiedsrichter-Befehle verwendet (C10/C20). Um ein Foul zu ahnden, Tore zu zählen, das Spiel zu pausieren oder zu stoppen muss der Nutzer den entsprechenden Button drücken. Diese Befehle müssen

auch an das gegnerische Team gesendet werden können (C10/C50).

## 9.2 Sollkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

**RS1:** *In der GUI sollten alle für das Spiel notwendigen Daten richtig angezeigt werden.*
Die GUI soll ein virtuelles Spielfeld mit den Positionen des Balls und der Roboter darstellen können (C10/C30/C40). Diese Daten sollten möglichst „live“ sein.

## 9.3 Kannkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

**RC1:** *Robotersteuerung über Controller*
Die Roboter können über die GUI mittels WLAN mit einem Controller, welchen ein Nutzer bedient, gesteuert werden können (C10/C60). Dabei müssen jedoch alle Roboter manuell gesteuert werden, die Verwendung der künstlichen Intelligenz ist nicht möglich.

**RC2:** *Robotersteuerung über Tastatur*
Die Roboter können über die GUI mittels WLAN mit der Tastatur, welche ein Nutzer bedient, gesteuert werden können (C10/C60). Dabei müssen jedoch alle Roboter manuell gesteuert werden, die Verwendung der künstlichen Intelligenz ist nicht möglich.

**RC3:** *Alle drei Pis geben die jeweilige Strategie in der GUI mit den nächsten Aktionen, die darin enthalten sind, an.*
Die Raspberry Pis, worauf die künstliche Intelligenz läuft, senden die Befehle zusätzlich an die GUI (C70/C90), welche diese dem Nutzer in Textform anzeigen kann (C10/C30).

# 10 Glossar

**Aktivitätsdiagramm:** Ist ein Verhaltensdiagramm in UML. In einem Aktivitätsdiagramm wird der Ablauf eines konkreten Anwendungsfalls modelliert.

**Controller:** Ist ein Eingabegerät, dass hauptsächlich für die Steuerung von Computerspielen vorgesehen ist.

**Funktionsmodelle:** Ein Funktionsmodell wird dafür genutzt die Systemteile und die Kommunikation unter den Systemteilen zu modellieren.

**GUI:** Grafische Benutzeroberfläche. Hat die Aufgabe Anwendungssoftware mit Symbolen, Steuerelemente und Widgets bedienbar zu machen.

**IP-Adressen:** Eine IP-Adresse ist eine Adresse in einem Computernetz und stellt sicher, dass die einzelnen Geräte im Netzwerk erreichbar sind.

**JDK:** Das Java Delevelopment Kit ist eine Sammlung fertiger Programme in der Programmiersprache Java, um neue Programme entwickeln zu können.

**JRE:** Die Java Runtime Enviroment ist die Laufzeitumgebung von Java.

**KI:** Künstliche Intelligenz. Übertragung von menschenähnlicher Intelligenz auf Roboter, Computer und oder Maschinen.

**Klassendiagramm:** Ein Klassendiagramm ist ein Strukturdiagramm zur Modellierung von Klassen, Schnittstellen und deren Beziehungen.

**Komponentendiagramm:** Durch ein Komponentendiagramm werden die (ogischen Komponenten des Systems und deren Schnittstellen (Ports) veranschaulicht.

**LeJos:** Ist eine Programmbibliothek und auch Betriebssystem, dass eine Programmierung/Steuerung von Lego Mindstorm Robotern ermöglicht.

**Objektmodelle:** Objektmodelle werden dafür benutzt statische Strukturen von Klassen und Objekten, sowie deren Verhalten zu modellieren.

**Plug-In:** Eine optionale Softwarekomponente für ein bestehendes Programm.

**Port:** Ermöglichen den Zugriff von Client zu Server, sowohl Server als auch Client haben bei einer bestehenden Verbindung mindestens einen Port. Der Server besitzt einen festen Port und der Client einen ausgehandelten unbenutzten Port.

**Raspberry Pi:** Ein Minicomputer mit einem ARM-Mikroprozessor, auf dem als Betriebssystem hauptsächlich Linux eingesetzt wird. Er kann für verschiedene Zwecke eingesetzt werden.

**Roboter**: Sind eine technische Apparatur, die dazu dient menschliche Aufgaben zu übernehmen, sie werden meist über Computerprogramme gesteuert. In diesem Projekt hingegen wird als Roboter lediglich der „Lego Mindstorm EV3 – Roboter“ verstanden.

**RoboSoccer:** Fußballspiel zwischen Robotern mit angepassten Regeln.

**Schwarmbasiert:** Eine schwarmbasierte KI arbeitet ohne direkten Anführung oder Befehlsgeber, sondern als kollektive Intelligenz.

**Sequenzdiagramm:** Ein Sequenzdiagramm ist ein Verhaltensdiagramm in UML. In Sequenzdiagrammen wird der Austausch von Nachrichten modelliert.

**Systemkomponenten:** Einzelne Elemente eines Gesamtsystems.

**UDP:** User Datagram Protocol ist ein Netzwerkprotokoll, bei dem nicht sichergestellt wird, dass gesendete Pakete auch wirklich beim Empfänger ankommen und kein Verbindungsaufbau stattfindet. Dadurch wird der Overhead reduziert, aber auch kein Schutz der Daten gewährleistet.

**UML:** Die Unified Modelling Language ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen.

**Verteilungsdiagramm:** Ein Verteilungsdiagramm stellt die Verteilung des Systems auf vorhandener Hardware und vorhandener Netzwerktopologie dar.