Technische Universität Braunschweig

ISF

Capture the flag

Team 2

Software-Entwicklungspraktikum (SEP)
Sommersemester 2015

Technischer Entwurf

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

Betreuer: Benjamin Mensing

Auftragnehmer:

Name	E-Mail-Adresse
Patrik Brendel	p.brendel@tu-bs.de
Anika Christmann	a.christmann@tu-bs.de
Carsten Hoppe	c.hoppe@tu-bs.de
Björn Kottutz	b.kottutz@tu-bs.de
Sebastian Lübbe	s.luebbe@tu-bs.de
Marina Porrmann	marina.porrmann@tu-bs.de
Vanessa Wolters	vanessa.wolters@tu-bs.de

Braunschweig, 1. Juli 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Der Technische Entwurf bildet die Grundlage für die Implementierung des Software Projekts. Anhand diesen Dokuments wird das Produkt entwickelt. Außerdem werden klassische Entwurfsentscheidungen wie zum Beispiel die Verwendung bestimmter Bibliotheken dokumentiert.

Das Spiel *Capture the Flag* mit Lego EV3 Mindstorm Robotern braucht einen festen Spielablauf. Dieser Ablauf beginnt, indem die Roboter auf das Spielfeld gestellt und eingeschaltet werden. Zudem müssen die Systeme, GUI und Server, gestartet sein. Beendet wird das Spiel, wenn die Flagge des gegnerischen Teams im eigenen Lager ist. Dies bildet einen Rahmen, der einen reibungslosen Spielablauf gewährleistet. In Abbildung 1.1 wird das System durch ein Aktivitätsdiagramm grob beschrieben. Detaillierter wird auf die einzelnen Aktivitäten des Spieles im darauffolgenden Abschnitt 1.1 eingegangen.

Bevor ein neues Spiel gestartet werden kann, muss der Benutzer die Roboter, die GUI und den Server hochgefahren haben. Danach hat er die Möglichkeit eine Karte zu erstellen oder das Spiel direkt zu starten. Wenn eine Karte erstellt wird, wird diese in der Benutzeroberfläche ausgegeben. Danach kann das Spiel begonnen werden. Die zuvor erstellte Karte wird dann in der GUI als Spielfeld dargestellt. Falls der Benutzer das Spiel direkt startet, ohne eine Karte zu erstellen, wird die Default-Karte als Spielfeld geladen. Anschließend wird eine Verbindung zwischen GUI, Server und den Robotern aufgebaut. Der Server ließt außerdem die Karte ein und initialisiert das Spiel. Die Spielkoordination wird vom Server durchgeführt und das laufende Spiel in der Benutzeroberfläche angezeigt. Bei Beendigung des Spieles, wird in der GUI Sieg oder Niederlage des Teams verkündet und der Benutzer kann das Programm abschalten.

1.1 Projektdetails

In diesen Abschnitt wird detaillierter auf das Aktivitätsdiagramm in Abbildung 1.1 eingegangen und einzelne Funktionen genauer beschrieben.

1.1.1 Starten des Spiels

Zunächst wird wie in Abbildung 1.2 dargestellt die Roboter positioniert, dabei wird im jeden Diagramm nur ein Roboter dargestellt, da der zweite genauso behandelt wird. Danach kann der Benutzer die GUI, den Server und die Roboter starten. Anschließend hat er die Möglichkeit eine Karte zu erstellen, die er in der Benutzeroberfläche bauen kann. Der Server lädt das erstellte Spielfeld oder die Default-Karte, falls der Benutzer keine Karte ausgewählt hat. Zudem hat er dann die Option die Steuerung zu wählen, ob die Roboter manuell gesteuert werden oder durch die KI. Zum Schluss kann er das Spiel starten.

1.1.2 Steuerung durch KI

Die Abbildung 1.3 stellt die Steuerung des Roboters durch die KI dar. Demnach sendet der Server ein Informationspaket an den Roboter. Dieses Informationspaket enthält den Status des Spiels, den Status des Roboters, sowie dessen Position auf dem Spielfeld. Wenn das Spiel noch nicht beendet ist, berechnet die KI den nächsten Zug und lässt den Roboter die Aktion ausführen. Anschließend schickt die KI dem Server eine Bestätigung, dass die Aktion ausgeführt wurde. Der Server aktualisiert den Spielstand und schickt dann wieder ein neues Informationspaket. Wenn das Spiel zu Ende ist, fährt sich der Roboter herunter.

1.1.3 Steuerung durch Benutzer

In Abbildung 1.4 wird die Steuerung des Roboters durch den Benutzer dargestellt. Nachdem der Server das Signal zum Spiel Start bekommen hat sendet er der GUI die Daten des laufenden Spieles, welche diese dann visualisiert. Außerdem erwartet der Server eine Eingabe des Benutzer bezüglich der vom Roboter auszuführenden Aktionen, zum Beispiel die Richtung in die er fahren soll. Sobald der Server diese Informationen erhalten hat beginnt er sie an den Roboter weiterzuleiten, woraufhin der Roboter die vom Benutzer gewünschte Aktion ausführt. Sollte nach dem Ausführen der Aktion das Spiel beendet sein, so registriert dies der Server, beendet das Spiel und die GUI gibt den Spielsieger bekannt. Andernfalls wartet der Server erneut auf eine Eingabe des Benutzers.

1.1.4 Verbindungsaufbau

Abbildung 1.5 beschreibt den Verbindungsaufbau zwischen Server, Benutzeroberfläche und Roboter. Zunächst wird vom Server ein DataOutputStream zur GUI geöffnet und eine WLAN Verbindung zum Roboter hergestellt. Danach wird auf GUI Seite ein dazu passender DataInputStream zum Server und ein von ihr ausgehender DataOutputStream geöffnet. Gleichzeitig stellt der Roboter über WLAN die Verbindung zum Server her und sendet seinen Status an diesen. Anschließend muss der Server noch einen DataInputStream zur GUI öffnen, damit sie gegenseitig Daten senden und empfangen können. Zum Schluss sendet der Server noch seine Informationen über den Roboter an die GUI.

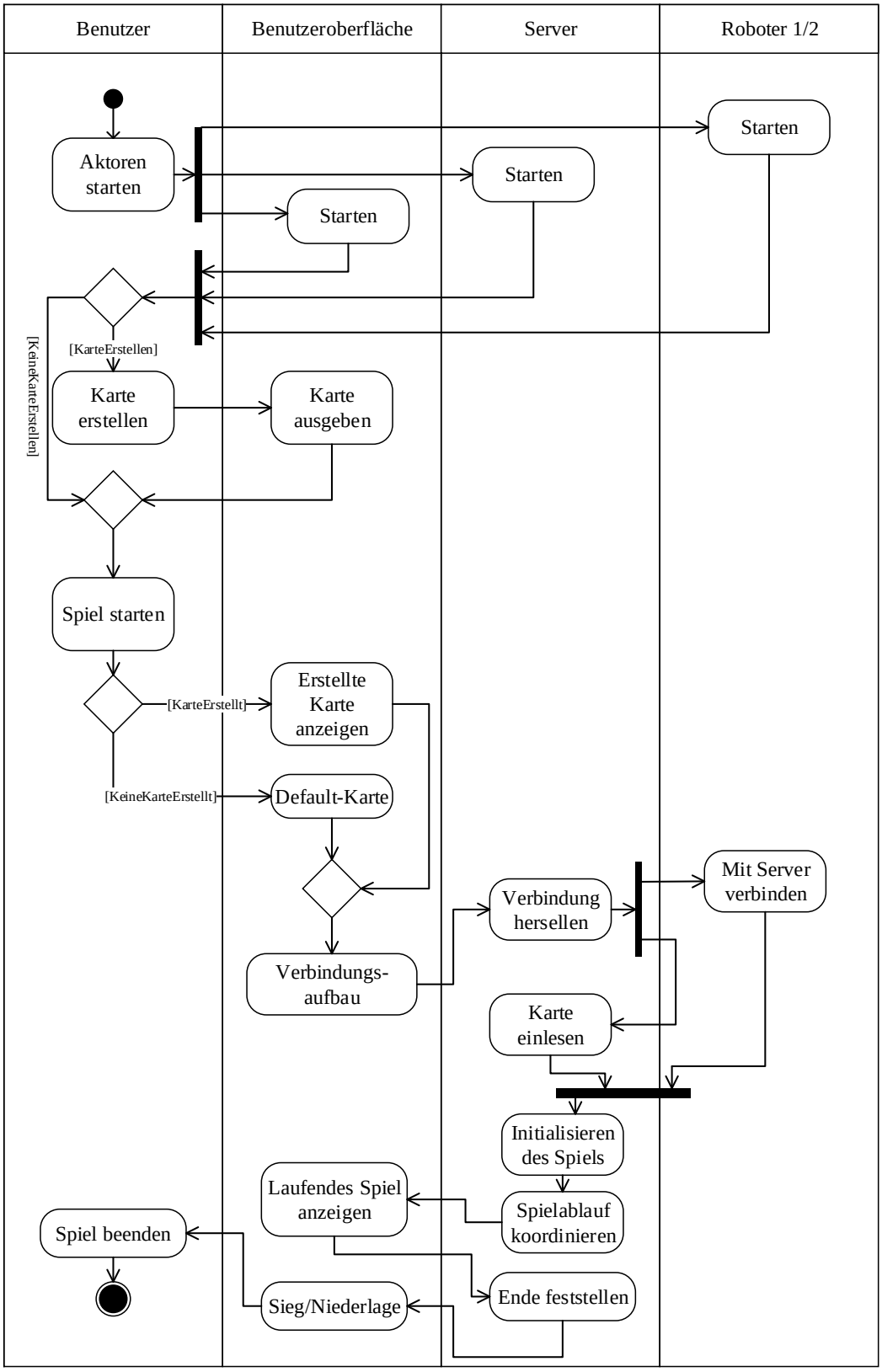

Abbildung 1.1: Aktivitätsdiagramm des Spielablaufs

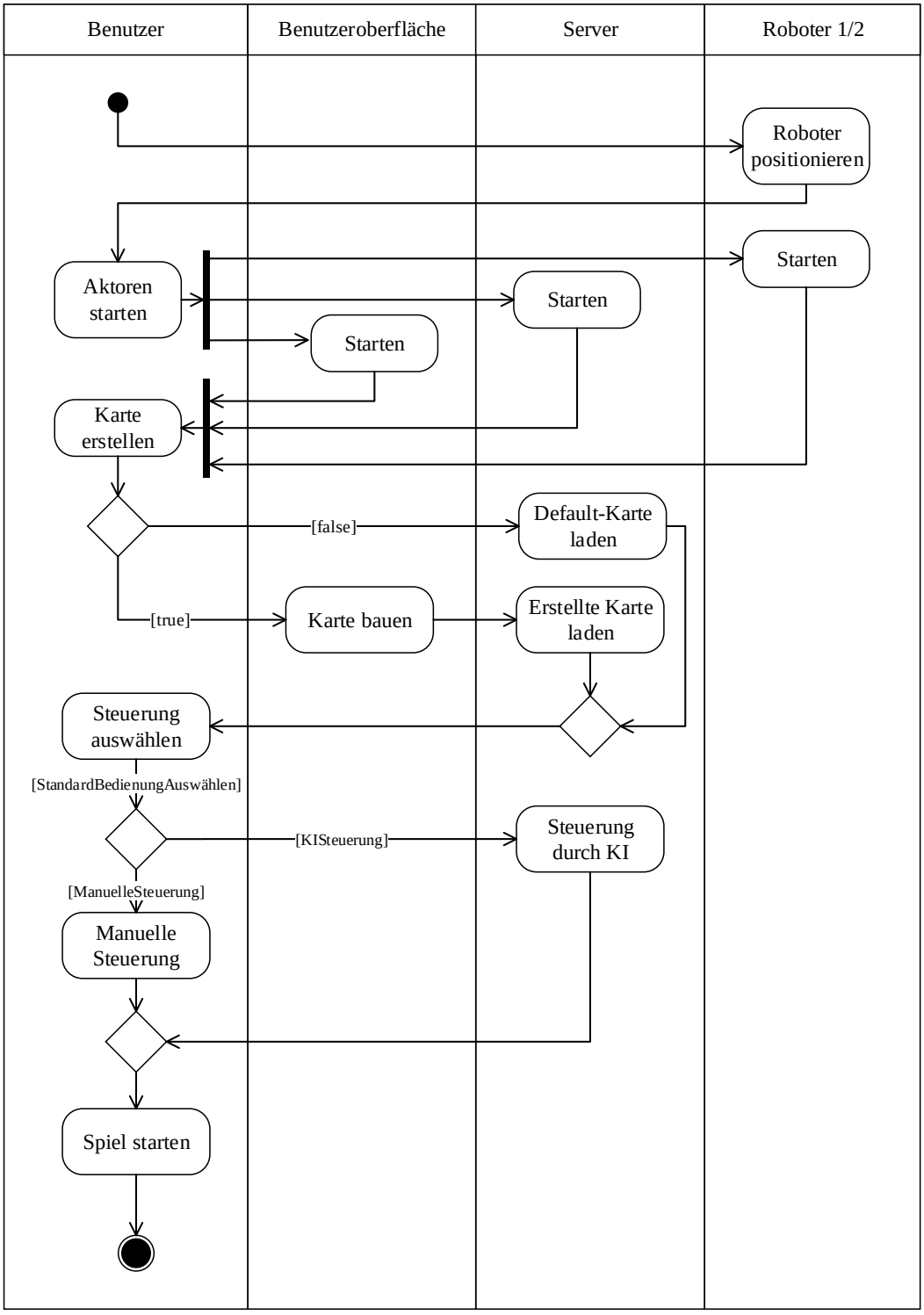

Abbildung 1.2: Aktivitätsdiagramm zum Starten des Spiels

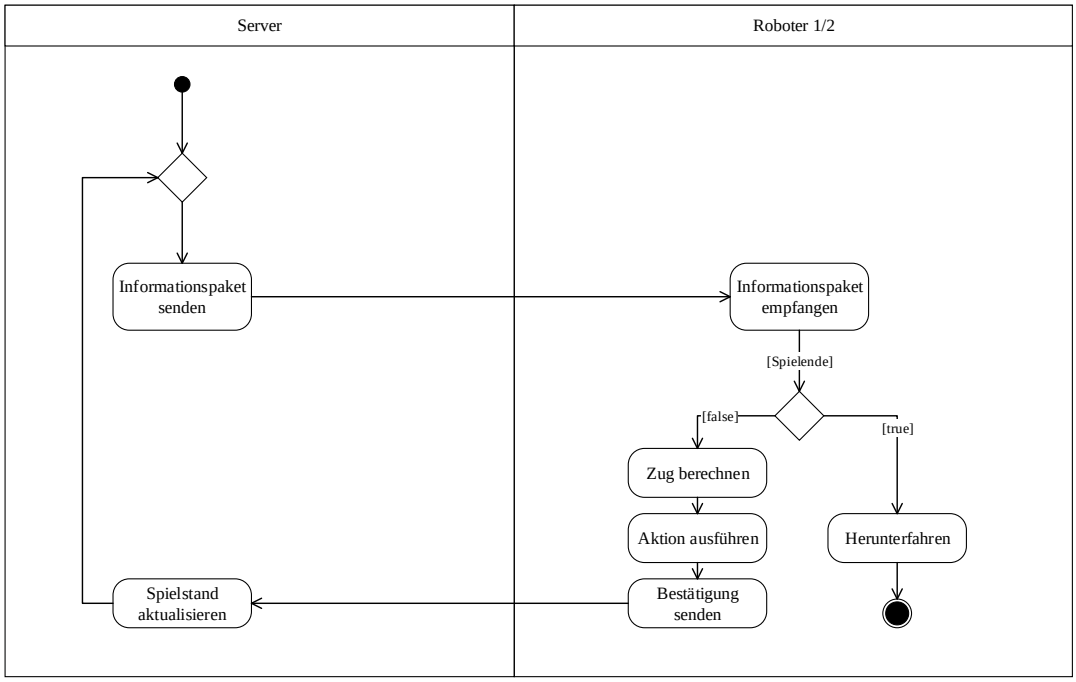

Abbildung 1.3: Steuerung durch KI

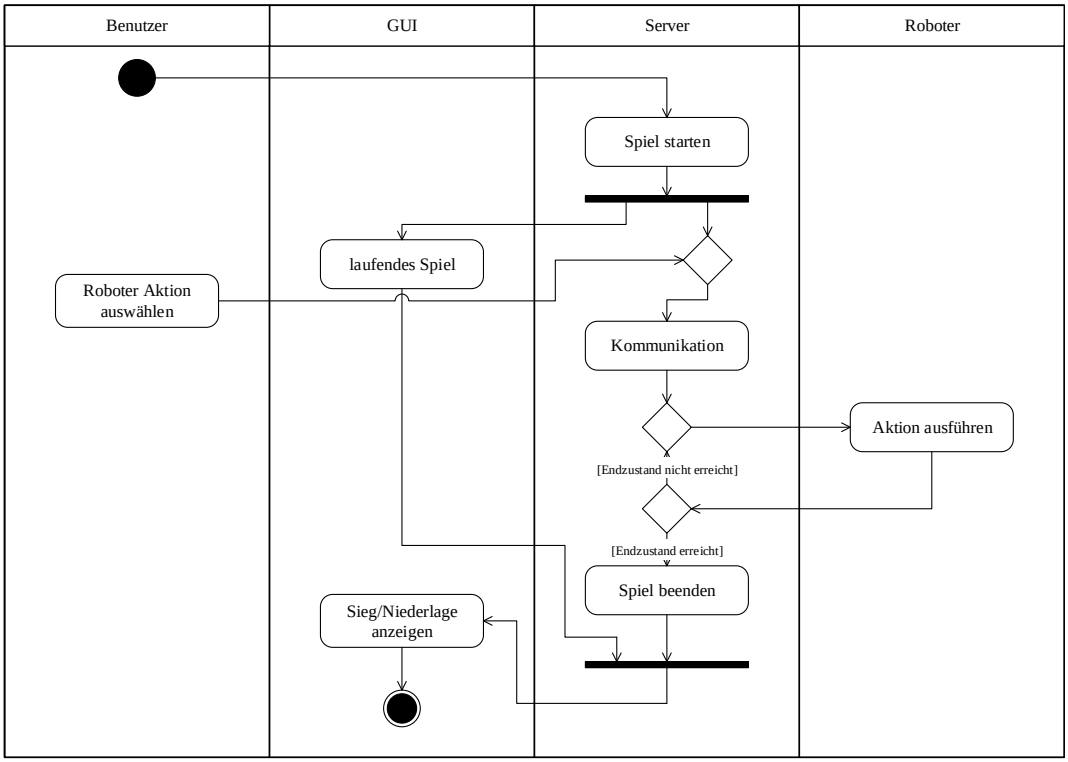

Abbildung 1.4: Steuerung durch Benutzer

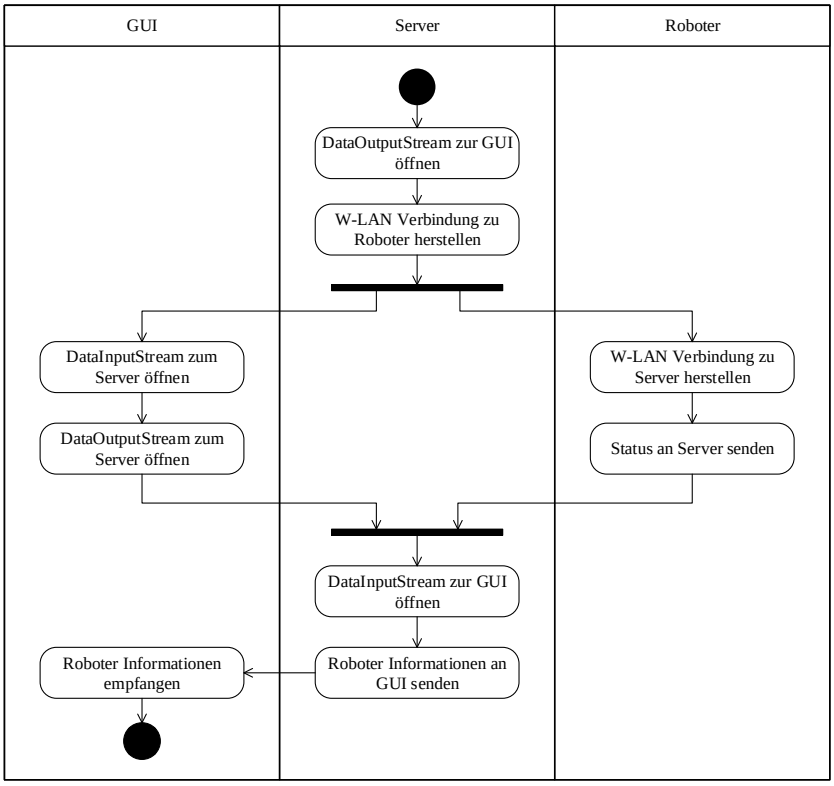

Abbildung 1.5: Verbindungsaufbau

2 Analyse der Produktfunktionen

Dieser Abschnitt beschäftigt sich mit der Analyse der einzelnen Produktfunktionen, die im Pflichtenheft aufgeführt sind.

2.1 Analyse von Funktionalität F10: Bewegen

Die Funktion **F10** beschreibt den Ablauf der Roboterbewegung. Nachdem die KI, oder der Benutzer bei manueller Steuerung, entschieden haben, dass sich der Roboter an eine neue Position bewegen soll, sendet der Server den entsprechenden Befehl über die WLAN Verbindung an den Roboter. Dieser führt daraufhin den Befehl aus und fährt. Hat der Roboter sein Ziel erreicht, sendet er eine Bestätigung an den Server zurück.

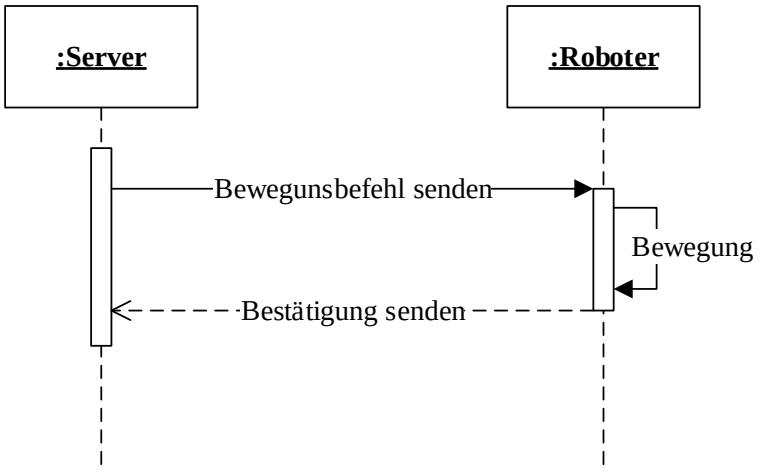

Abbildung 2.1: Sequenzdiagramm zu **F10**

2.2 Analyse von Funktionalität F20: Angriff

Die Funktion **F20** beschreibt den Vorgang des Angreifens. Soll ein Roboter einen gegnerischen Roboter angreifen, so sendet der Server an den angreifenden Roboter den Befehl seinen Angriff durch Blinken zu visualisieren. Danach sendet der Server die Informationen des Angriffs an den Server des gegnerischen Teams. Dieser kann nun den neuen Lebenspunktestand an seinen Roboter weitergeben.

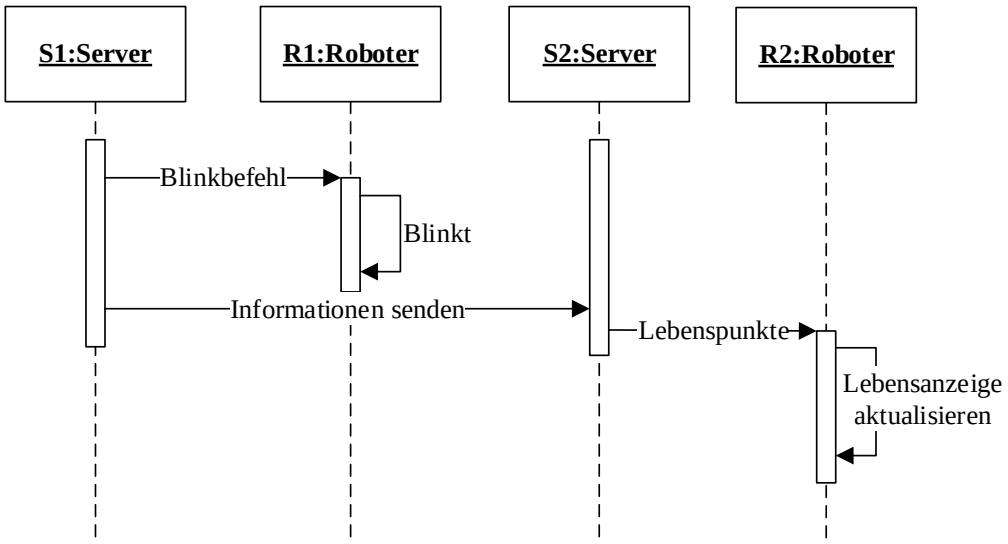

Abbildung 2.2: Sequenzdiagramm zu **F20**

2.3 Analyse von Funktionalität F30: Gittererkennung

Die Funktion **F30** beschreibt die Erkennung des Gitternetzes, welches das Spielfeld repräsentiert, durch den Roboter. Sobald der Roboter eingeschaltet und initialisiert wurde, erkennen die Sensoren des Roboters, ob sie sich über den dunklen Streifen des Spielfeldes befinden, oder über einem hellen Untergrund. Anhand dieser Sensordaten kann der Roboter erkennen, ob er sich auf den Linien des Spielfeldes befindet.

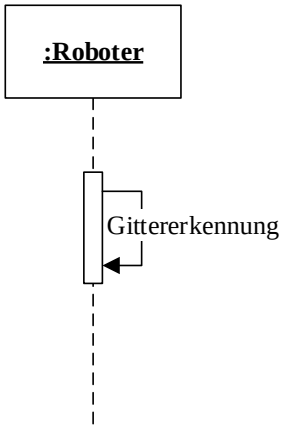

Abbildung 2.3: Sequenzdiagramm zu **F30**

2.4 Analyse von Funktionalität F40: Blinken

Die Funktion **F40** beschreibt das Verhalten des Roboters, nachdem er angegriffen wurde. Erhält der Server die Nachricht, dass ein Roboter angegriffen wurde, sendet er den Befehl zu Blinken an den entsprechenden Roboter weiter. Dieser wird dann für einige Sekunden das beleuchtbare Interface in einer Farbe blinken lassen.

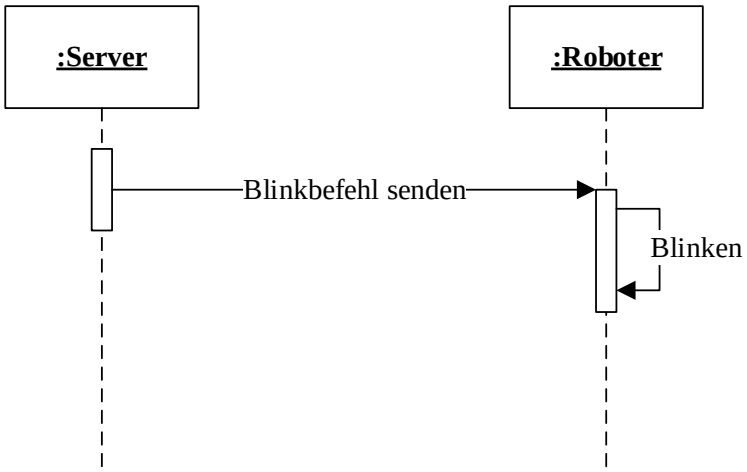

Abbildung 2.4: Sequenzdiagramm zu **F40**

2.5 Analyse von Funktionalität F50: Flaggenaufnahme

Die Funktion **F50** beschreibt das Verhalten des Roboters, nachdem er die Flagge aufgenommen hat. Bewegt sich ein Roboter auf ein Feld, auf welchem sich die gegnerische Flagge befindet, nimmt er diese auf. Registriert der Server, dass eine Flagge aufgenommen wurde, so gibt er dem entsprechenden Roboter den Befehl den Flaggenbesitz anzuzeigen. Dies geschieht mithilfe des beleuchtbaren Interface des Roboter.

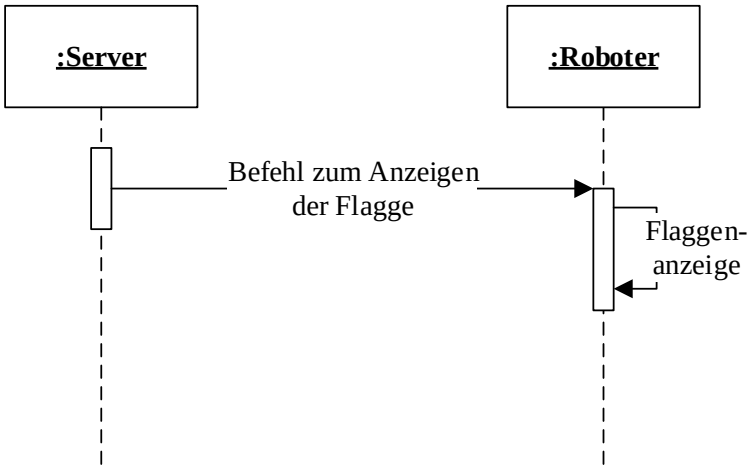

Abbildung 2.5: Sequenzdiagramm zu **F50**

2.6 Analyse von Funktionalität F60: WLAN-Datenverbindung

Die Funktion **F60** beschreibt die WLAN-Datenverbindung zwischen den Roboter und den Server. Zur Kommunikation werden Datenpakete gesendet, diese enthalten Bewegungsbefehle und Position des Roboters. Nach jedem Erhalt sendet der Empfänger eine Bestätigung an den Sender.

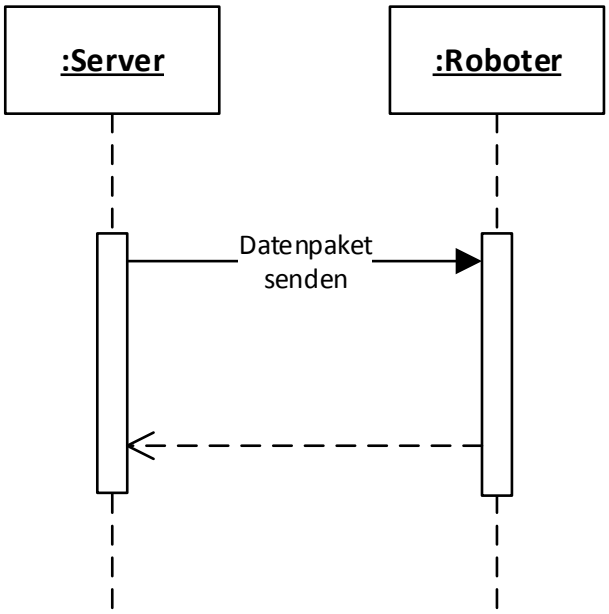

Abbildung 2.6: Sequenzdiagramm zu **F60**

2.7 Analyse von Funktionalität F70: Spielzug berechnen

Die Funktion **F70** beschreibt die Berechnung eines Spielzuges. Zu Beginn eines Spielzuges wird der KI die aktuelle Spielsituation (Positionen und Lebenspunkte der Roboter) mitgeteilt. Die KI wertet diese Informationen aus und berechnet die nächste Aktion für die Roboter.

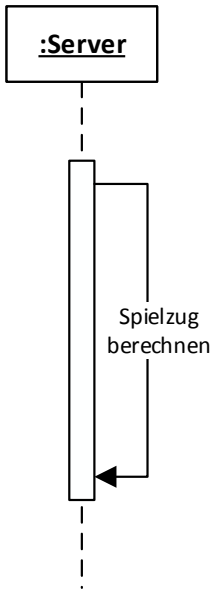

Abbildung 2.7: Sequenzdiagramm zu **F70**

2.8 Analyse von Funktionalität F80: Spielstart

Die Funktion **F80** beschreibt den Spielstart. Der Benutzer hat im Hauptmenü die Möglichkeit über „Einstellungen“ das Spiel einzustellen (Steuerungswahl, Kartenauswahl). Nachdem der Benutzer seine Einstellungen bestätigt und im Hauptmenü „Spiel starten“ wählt, wird der Server initialisiert. Über die GUI kann der Benutzer dann das Spielgeschehen verfolgen.

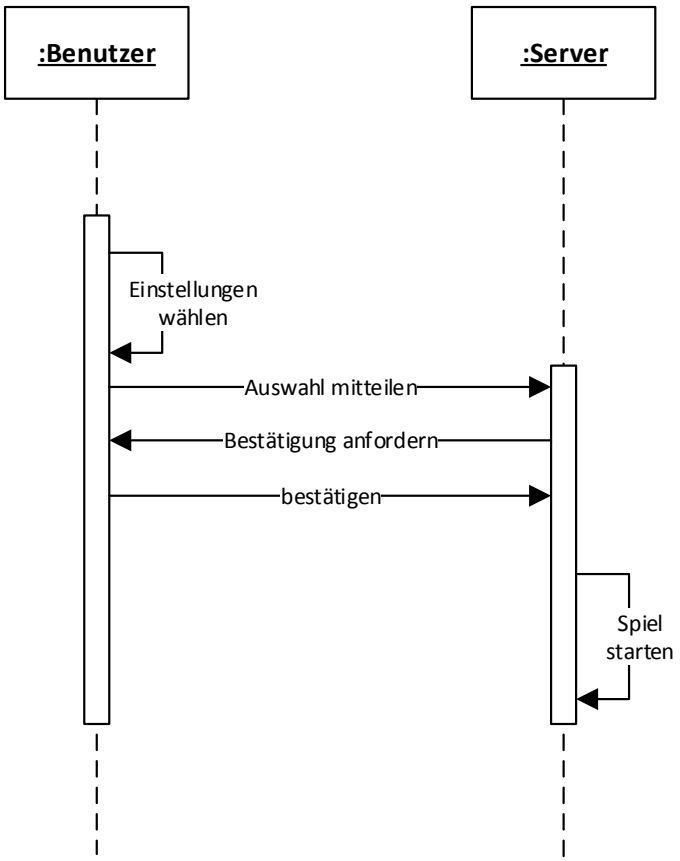

Abbildung 2.8: Sequenzdiagramm zu **F80**

2.9 Analyse von Funktionalität F90: Spielabbruch

Die Funktion **F90** beschreibt einen Spielabbruch. Nachdem ein Spiel gestartet wurde und noch läuft, hat der Benutzer die Möglichkeit das Spiel über das Userinterface abzubrechen. Bei Spielabbruch fordert der Server eine Bestätigung an, nach Bestätigung des Benutzers beendet der Server das Spiel und schaltet die Roboter aus.

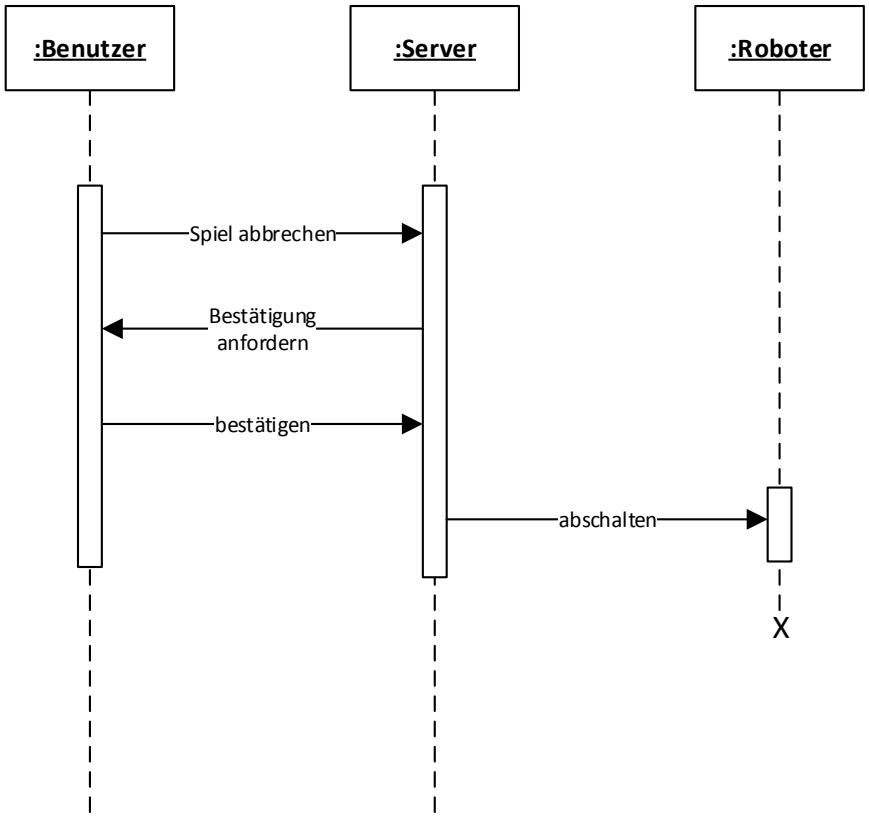

Abbildung 2.9: Sequenzdiagramm zu **F90**

2.10 Analyse von Funktionalität F100: Spielende

Die Funktion **F100** beschreibt das Spielende. Der Server überwacht ein laufendes Spiel, d.h. er überwacht u.a. Spielstände, Lebenspunkte der Roboter sowie Positionen. Das Spiel läuft solange weiter bis der Server feststellt, dass ein Team zwei Runden gewonnen hat und das Spiel beendet. Nachdem das Spielende festgestellt wurde, erhält der Benutzer eine Mitteilung über das Spielergebnis. Der Server beendet das Spiel und schaltet die Roboter aus.

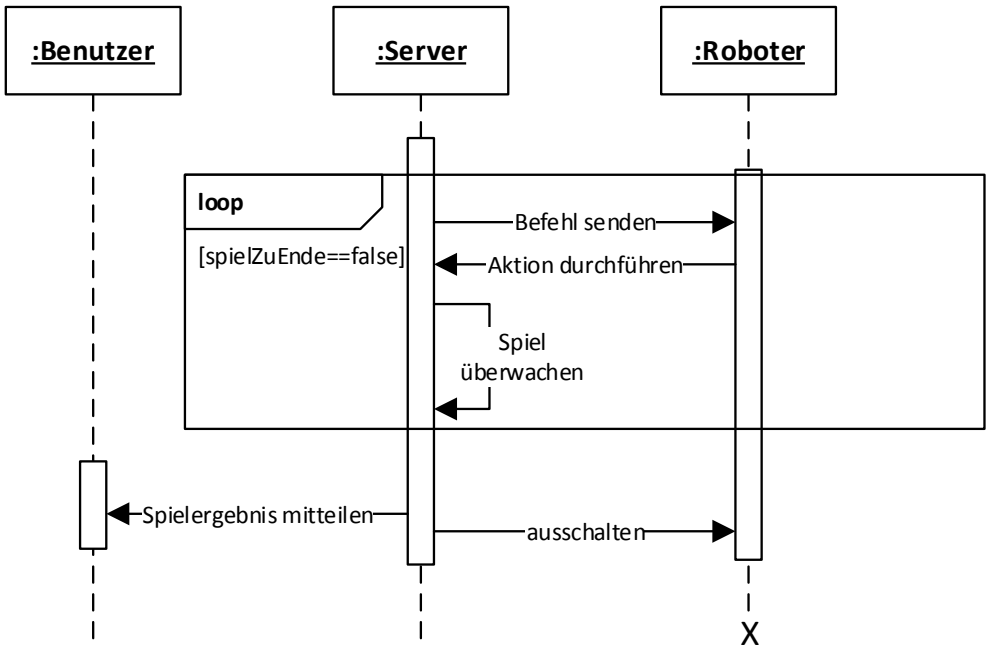

Abbildung 2.10: Sequenzdiagramm zu **F100**

2.11 Analyse von Funktionalität F110: Spieldarstellung

Die Spieldarstellung, Funktion **F110**, beschreibt die Funktionalität der Ausgabe der Spieldaten in der GUI. Der Benutzer startet das Spiel, daraufhin initialisiert der Server das Programm und die Standardelemente der Benutzeroberfläche werden geladen. Ausgehend vom Startbildschirm hat der Nutzer die Möglichkeit, wiederholt mit dem Programm zu interagieren. Durch die Interaktion werden Daten neu berechnet, die eine Aktualisierung der Benutzeroberfläche nach sich ziehen. Innerhalb der Aktionsschleife hat der Benutzer auch die Möglichkeit das Programm zu beenden. In dem Fall wird die Schleife beendet und der Server schließt die Benutzeroberfläche.

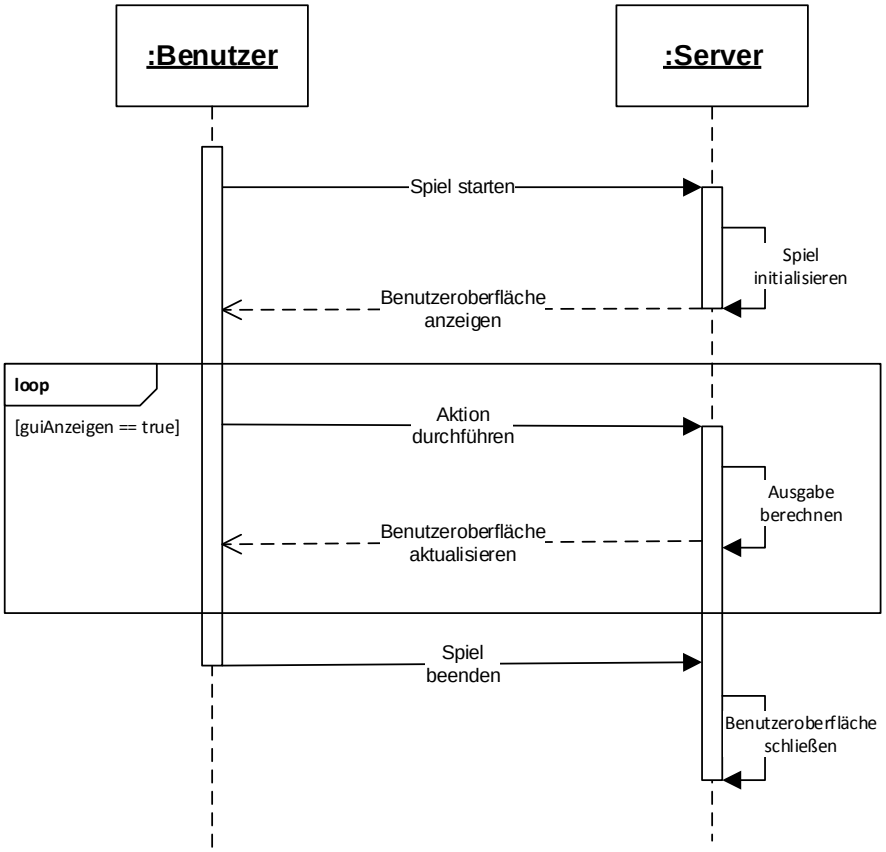

Abbildung 2.11: Sequenzdiagramm zu **F110**

2.12 Analyse von Funktionalität F120: Karte erstellen

Funktion **F120** beschreibt den Ablauf zum Erstellen oder Verändern einer Spielkarte. Ausgehend vom Hauptmenü hat der Benutzer die Möglichkeit eigene Karten zu erstellen. Dazu wählt der Benutzer den Menüpunkt Karteneditor, woraufhin der Karteneditor angezeigt wird. Der Benutzer muss auswählen, ob er eine neue, leere Karte erstellen möchte, oder eine existierende Karte aus einer Datei laden will. Entsprechend wird die geladene Karte oder eine leere Karte angezeigt. Der Nutzer hat jetzt die Möglichkeit mehrere Objekte zu platzieren oder zu löschen. Daraufhin zeigt der Server die aktualisierte Karte an und speichert die Werte zwischen.
Ist der Bearbeitungsvorgang abgeschlossen, kann die Karte in einer Datei langfristig gespeichert werden. Beendet der Benutzer den Karteneditor, wird wieder das Hauptmenü angezeigt.

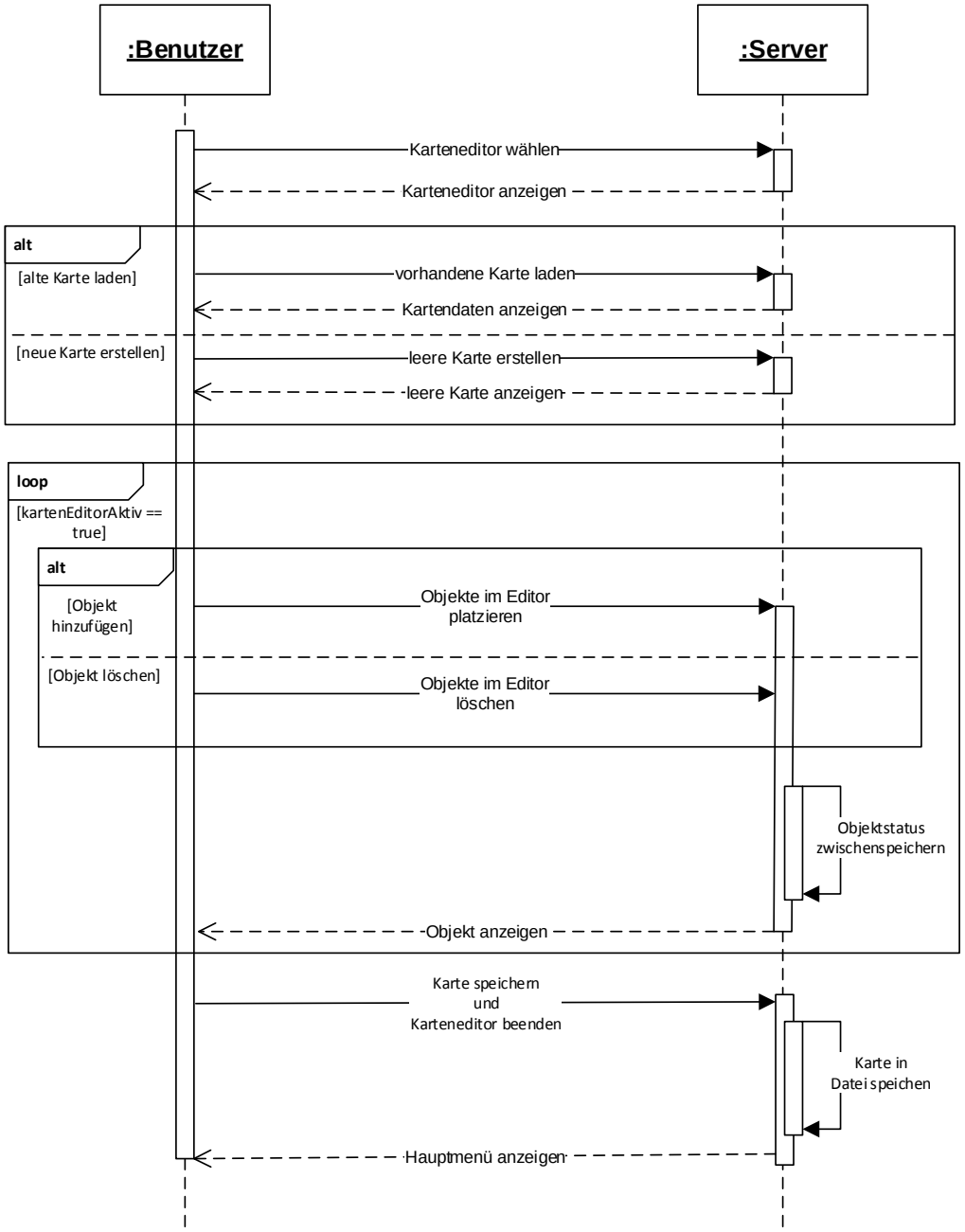

Abbildung 2.12: Sequenzdiagramm zu **F120**

2.13 Analyse von Funktionalität F130: Karte wählen

Die Funktion **F130** beschreibt die Möglichkeit des Nutzers, für das folgende Spiel eine Karte zu laden.
Ausgehend vom Hauptmenü wählt der Benutzer den Punkt „Einstellungen“ aus. Daraufhin wird die Oberfläche für das Optionsmenü dargestellt. Der Benutzer muss anschließend eine Karte aus einer Datei auswählen und die Auswahl bestätigen. Der Server lädt die Karte in das Programm und gibt eine Meldung aus.

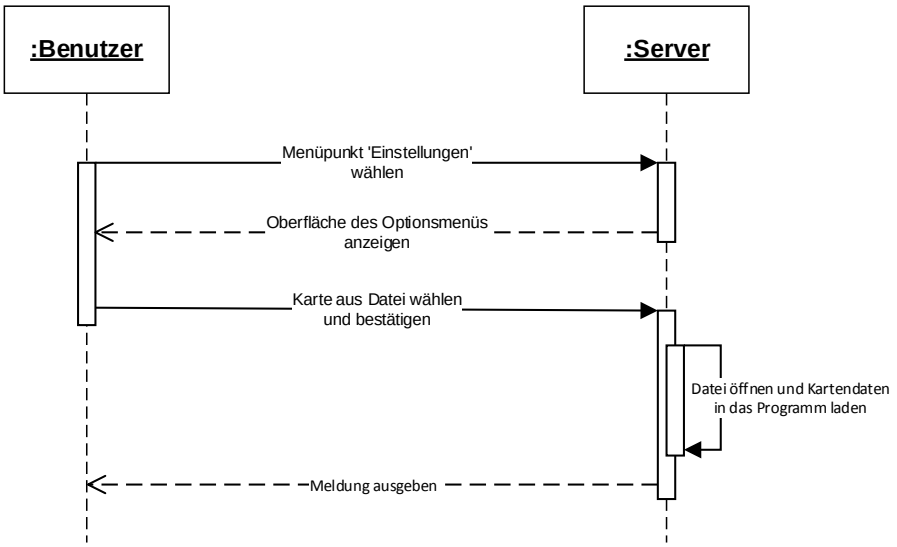

Abbildung 2.13: Sequenzdiagramm zu **F130**

2.14 Analyse von Funktionalität F140: Spielsteuerung wählen

Die Funktion **F140** beschreibt den Auswahlvorgang für die Spielsteuerung.
Ausgehend vom Hauptmenü wählt der Benutzer den Punkt „Einstellungen“ aus. Daraufhin wird die Oberfläche für das Optionsmenü dargestellt. Der Benutzer kann anschließend den Punkt „KI-Steuerung“ oder den Punkt „Manuelle Steuerung“ auswählen. Wählt der Benutzer einen der beiden Punkte an, wird der Server programmintern die Steuerung umstellen und die entsprechende Auswahl auf der Grafischen Oberfläche darstellen.

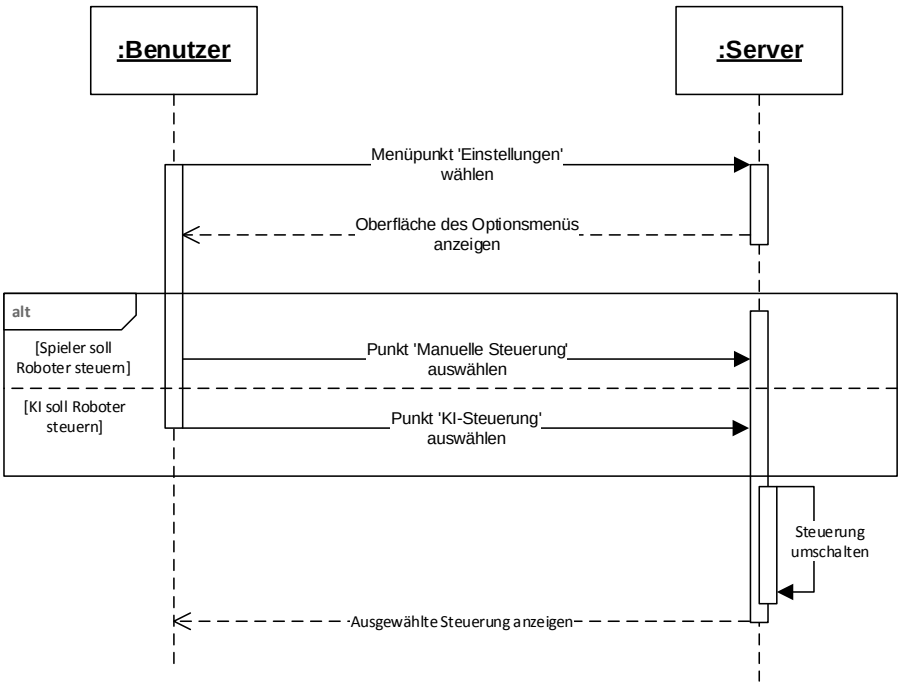

Abbildung 2.14: Sequenzdiagramm zu **F140**

2.15 Analyse von Funktionalität F150: Manuelle Steuerung

Die **F150** beschreibt die manuelle Steuerung der Roboter im laufenden Spiel.
Als Vorbedingung befindet sich das Programm in einem laufenden Spiel. Davon ausgehend wird dem Benutzer von der GUI die Spieloberfläche angezeigt. Im Menü „Einstellungen“ muss die manuelle Steuerung aktiviert worden sein.
Der Benutzer hat nun die Möglichkeit mit dem Programm zu interagieren. Ist der Benutzer an der Reihe, wählt er den zu steuernden Roboter aus. Die Auswahl wird vom Server visuell quittiert und ein Steuerkreuz auf der Benutzeroberfläche freigeschaltet. Der Benutzer kann anschließend eine Bewegungsrichtung anwählen. Der Server verarbeitet die Eingabe, der ausgewählte Roboter bewegt sich auf dem Spielfeld und die Benutzeroberfläche wird mit den neuen Daten aktualisiert.

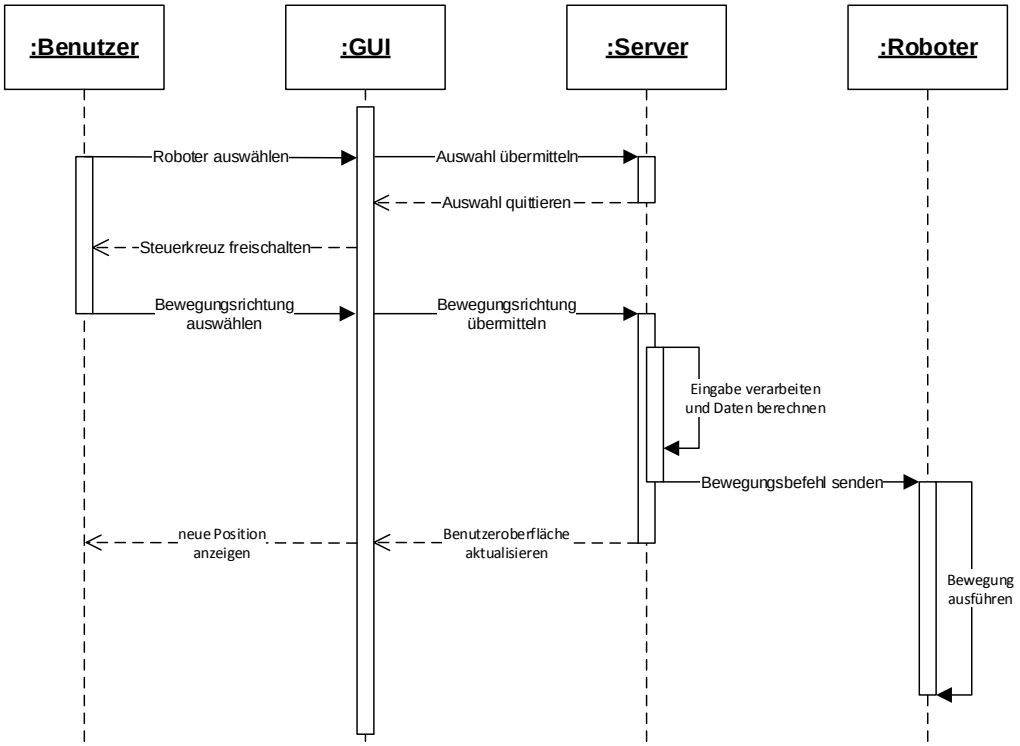

Abbildung 2.15: Sequenzdiagramm zu **F150**

3 Resultierende Softwarearchitektur

In diesem Abschnitt werden die zu entwickelnden Komponenten und Subsysteme dargestellt und beschrieben. Im Folgenden ergibt sich ein Überblick über die resultierende Softwarearchitektur des Spiels Capture the Flag.

3.1 Komponentenspezifikation

Dieses Kapitel stellt die aus der Analyse der Produktfunktionen (Kapitel 2) resultierende Struktur der Komponenten dar. Die Komponentenstruktur wird durch ein Komponentendiagramm beschrieben. Anschließend werden die einzelnen Komponenten aufgelistet und kurz eingeführt.

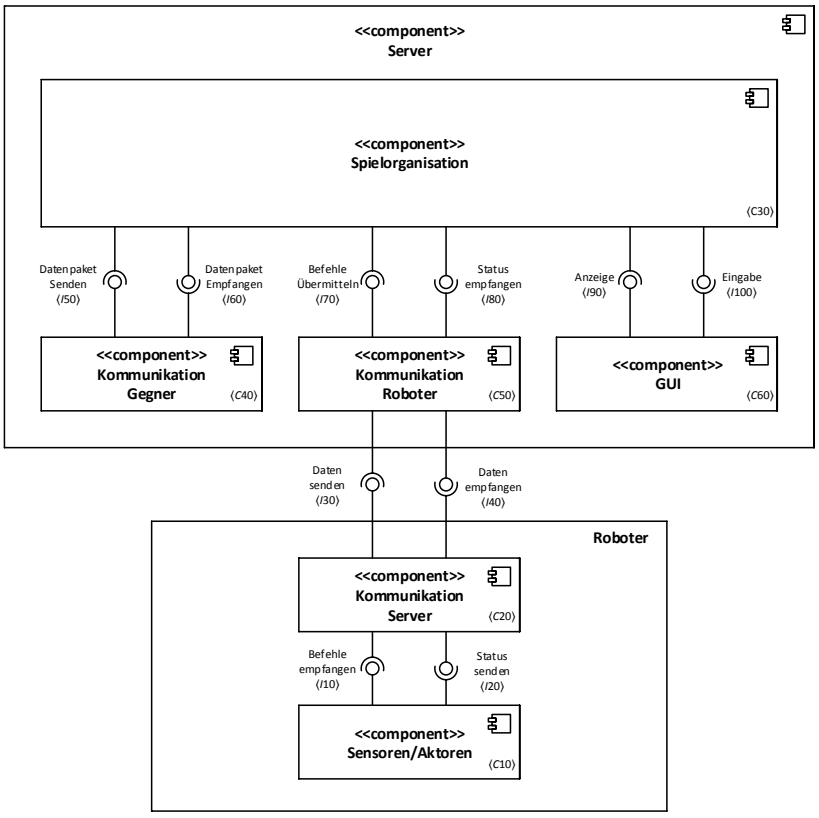

Abbildung 3.1: Komponentendiagramm

Beschreibung der Komponenten:

Komponente $\langle C10 \rangle$: <Sensoren/Aktoren (Roboter)>
Der Roboter ist als Spielfigur dafür zuständig, die vom Server übermittelten Befehle zur Bewegung und Anzeige von Parametern auszuführen und das Spiel somit physisch zu visualisieren. Die Komponente *Sensoren/Aktoren* steuert die dazu nötigen Sensoren und Aktoren des Roboters.

Komponente $\langle C20 \rangle$: <Kommunikation Server (Roboter)>
Die Komponente zur *Kommunikation mit dem Server* dient zum Verbindungsaufbau und zur Übertragung von Steuerungs- und Anzeigebefehlen zwischen Server und Roboter-Client per WLAN.

Komponente $\langle C30 \rangle$: <Spielorganisation (Server)>
Die Komponente *Spielorganisation* des Serverprogramms hat die zentrale Aufgabe, die Spielmechanik insgesamt zu verwalten. Der korrekte Ablauf der Spielzüge und die Koordination von Datenströmen zwischen den Komponenten werden von der *Spielorganisation* überwacht und durchgeführt. Zusätzlich ist die *Spielorganisation* für die Auswertung von Karten- und Spielinformationen zuständig, um Handlungsanweisungen für die Roboter und GUI zu errechnen.

Komponente $\langle C40 \rangle$: <Kommunikation Gegner (Server)>
Das gegnerische Team entwickelt das Spiel unabhängig mit eigenem Server und eigener Datenverarbeitung. Im Wettkampfmodus wird das Spiel von zwei Programmen auf zwei Servern verarbeitet. Die Komponente *Server-Server-Kommunikation* sorgt für die Übertragung und Synchronisierung von Spieldaten zwischen den beiden Servern und gibt diese zur Weiterverarbeitung an die *Spielorganisation* weiter.

Komponente $\langle C50 \rangle$: <Kommunikation Roboter (Server)>
Die Komponente zur Kommunikation mit dem Roboter stellt eine Verbindung mit dem Kommunikationsmodul des Roboters her und übermittelt Befehle zur Bewegung und Visualisierung von Informationen.

Komponente $\langle C60 \rangle$: <GUI (Server)>
Die Komponente *GUI* ist die Schnittstelle zwischen Benutzer und Programm. Die Komponente ist für die Ausgabe von Informationen auf einem grafischen Ausgabegerät zuständig. Zusätzlich stellt die Komponente Eingabemöglichkeiten zur Veränderung von Spielparametern bereit. Ändert der Benutzer Parameter, werden diese an die *Spielorganisation* übertragen.

3.2 Schnittstellenspezifikation

Auf Basis der Komponentenstruktur ergeben sich mehrere Schnittstellen zwischen den Komponenten. Im Folgenden werden die einzelnen Schnittstellen dargestellt und die nötigen Operatio-

nen zum Datenaustausch beschrieben.

Schnittstelle $\langle I10 \rangle$: <Befehle empfangen>

Operation	**Beschreibung**
run()	Verbindung zwischen Roboter und Server wird aufgebaut und Befehle werden empfangen und ausgeführt.

Schnittstelle $\langle I20 \rangle$: <Status senden>

Operation	**Beschreibung**
sendMessage(msg: String)	Sendet eine Nachricht zum Server.

Schnittstelle $\langle I30 \rangle$: <Daten senden>

Operation	**Beschreibung**
sendMessage(msg: String)	Sendet eine Nachricht zum Server.

Schnittstelle $\langle I40 \rangle$: <Daten empfangen>

Operation	**Beschreibung**
run()	Verbindung zwischen Roboter und Server wird aufgebaut und Befehle werden empfangen und ausgeführt.

Schnittstelle $\langle I50 \rangle$: <Datenpaket senden>

Operation	**Beschreibung**
connect()	Baut eine Verbindung auf.
send()	Sendet ein Datenpaket.

Schnittstelle $\langle I60 \rangle$: <Datenpaket empfangen>

Operation	**Beschreibung**
connect()	Baut eine Verbindung auf.
receive()	Empfängt ein Datenpaket.

Schnittstelle $\langle I70 \rangle$: <Befehle übermitteln>

Operation	Beschreibung
run()	Verbindung zum Roboter wird aufgebaut.
sendMessage(msg: String)	Befehl wird übermittelt.

Schnittstelle $\langle I80 \rangle$: <Status empfangen>

Operation	Beschreibung
sendMessage(msg: String)	Statusnachricht wird gesendet und empfangen.

Schnittstelle $\langle I90 \rangle$: <Anzeige>

Operation	Beschreibung
open()	Öffnet eine Oberfläche und zeigt die Inhalte an.

Schnittstelle $\langle I100 \rangle$: <Eingabe>

Operation	Beschreibung
input()	Verarbeitet die Eingegebenen Werte.

3.3 Protokolle für die Benutzung der Komponenten

Im Folgenden wird die Verwendung der einzelnen Komponenten mit Protokoll-State-Charts beschrieben. Die wesentlichen Komponenten sind projektspezifisch und benötigen eine unverhältnismäßig große Anpassung zur Weiterverwendung in anderen Projekten.

Anders als die spielspezifischen Komponenten kann die Server-Roboter-Kommunikation auch für andere Projekte verwendet werden, in denen der LEGO Mindstorm EV3 Roboter Befehle von einem Serverprogramm erhalten und ausführen soll. Aufgrund der allgemein gehaltenen Programmierung der Roboter kann das Roboterprogramm vollständig für Projekte, in denen Bewegungsbefehle auf einem aufgeklebten Linienmuster ausgeführt werden sollen, verwendet werden. Die gesendeten Befehle für die Visualisierung von Informationen, zum Beispiel Visualisierungen per Roboterdisplay, können nach Belieben angepasst werden. Die folgenden Komponenten eignen sich für eine Wiederverwendung.

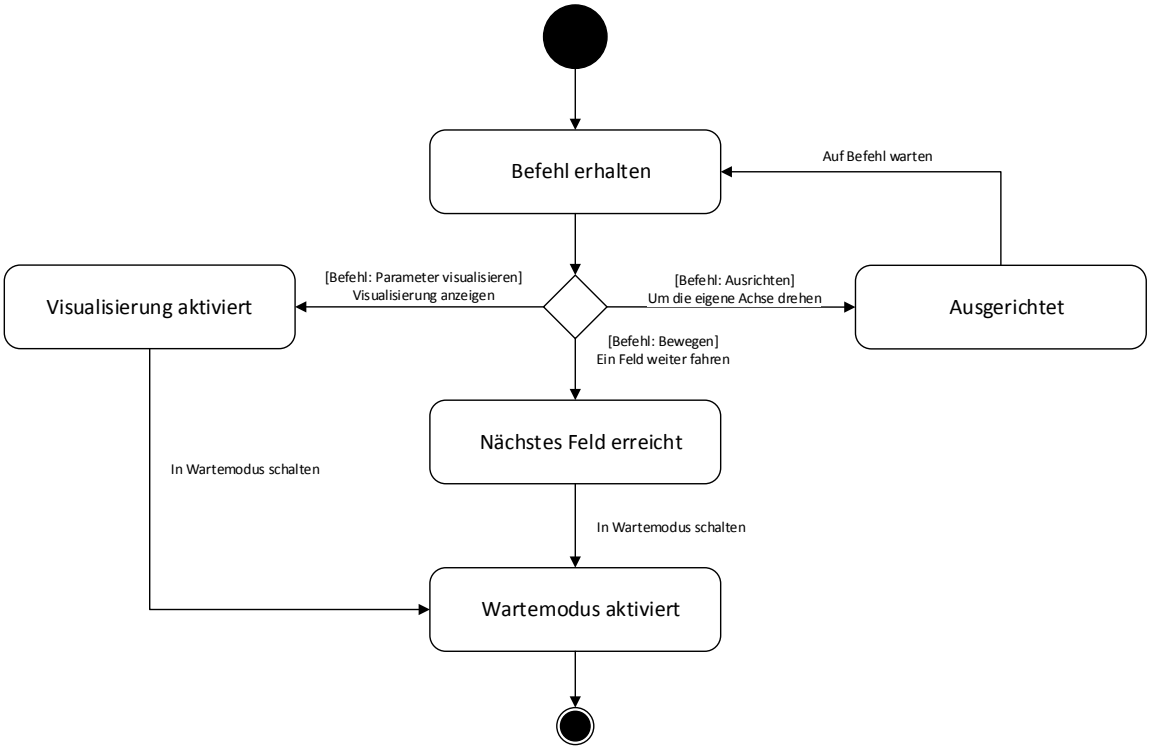

Abbildung 3.2: State-Chart zu Komponente C10: Sensoren/Aktoren (Roboter).

Die Abbildung zu Komponente C10 zeigt die möglichen Zustände bei der Verarbeitung von Befehlen, die der Roboter ausführen soll. Dabei wird je nach Art des Befehls eine gewünschte Visualisierung ausgegeben oder ein Bewegungsbefehl ausgeführt. Steht der Roboter nicht in Fahrtrichtung zur gewünschten Position, muss der Roboter sich zunächst ausrichten. Anschließend kann der Roboter auf dem Linienmuster zur nächsten Position vorrücken.

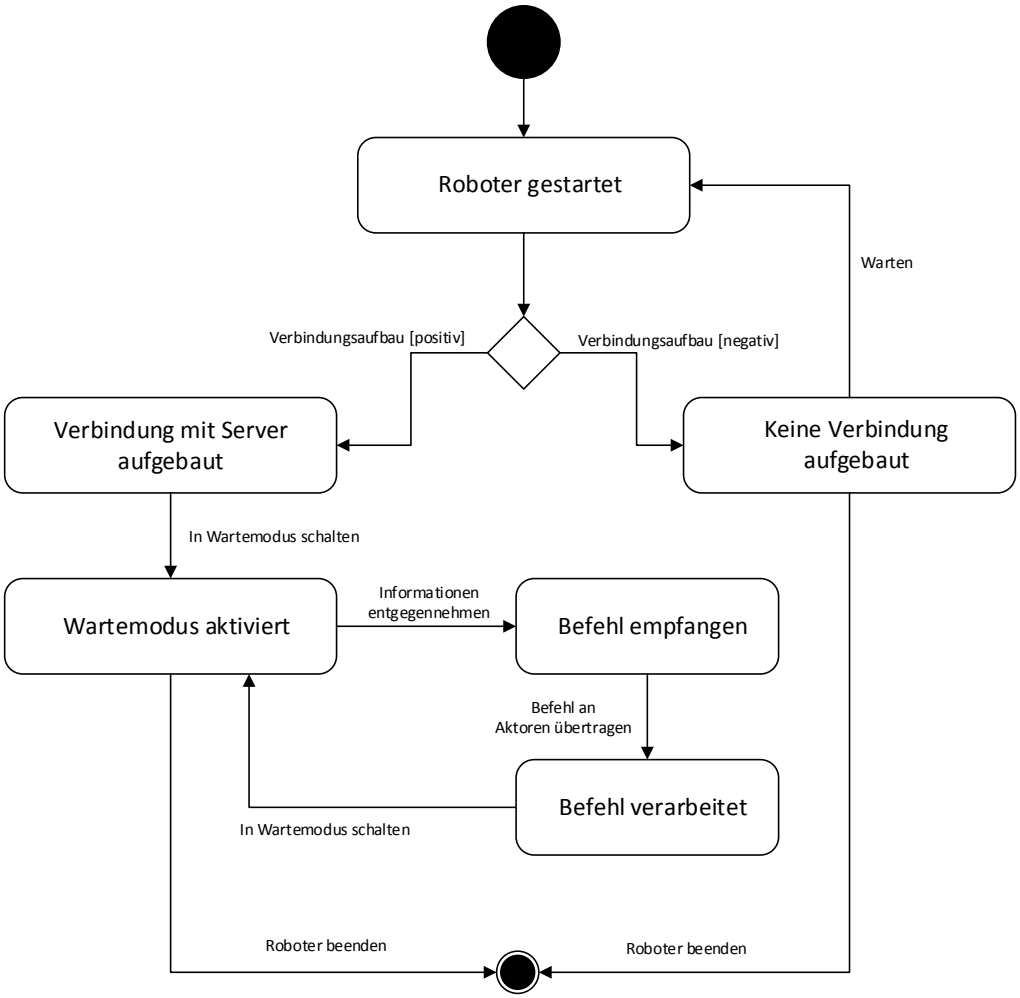

Abbildung 3.3: State-Chart zu Komponente C20: Kommunikation mit Server (Roboter).

Die Abbildung zu C20 zeigt die möglichen Zustände, wenn die Kommunikationskomponente des Roboters Befehle entgegen nimmt. Zunächst wird versucht eine Verbindung mit dem Server herzustellen. Ist die Verbindung erfolgreich aufgebaut worden, befindet sich das Kommunikationsmodul in einem Wartezustand, bis der Server Befehle sendet. Diese werden empfangen und an die Komponente C10 übergeben.

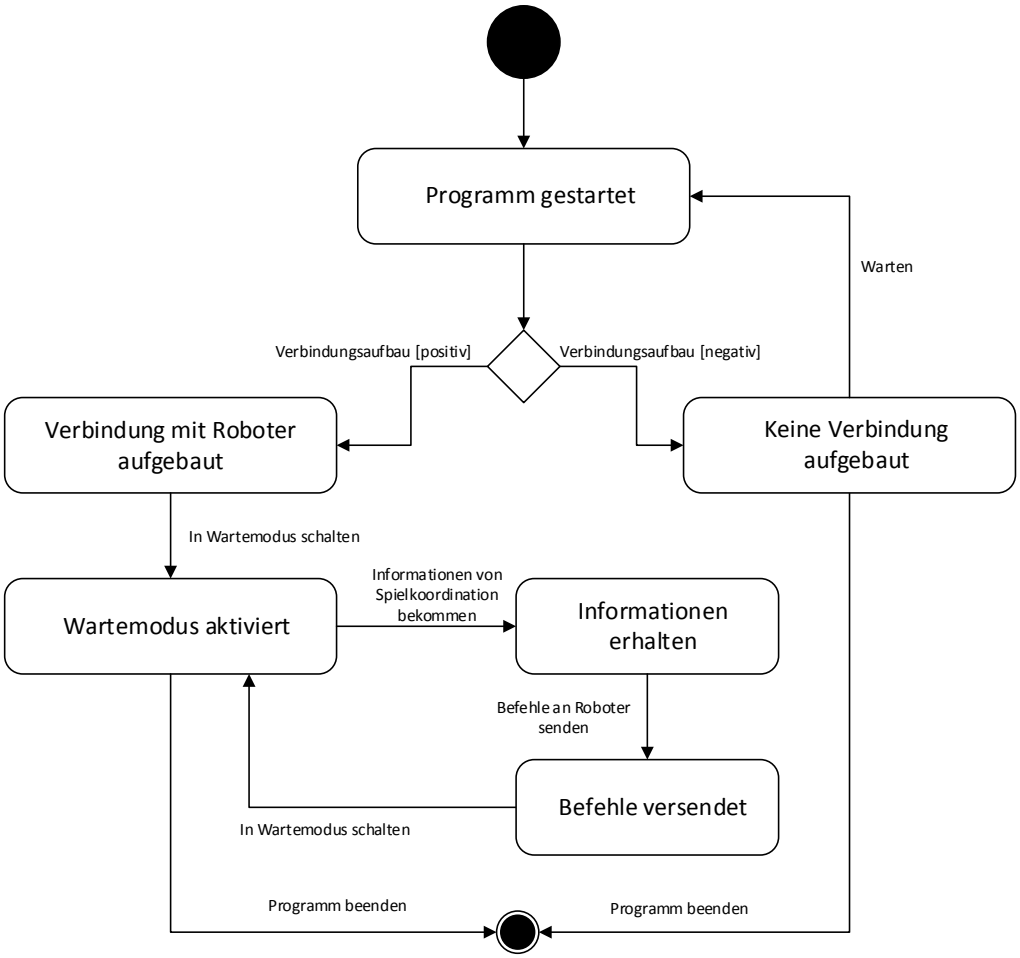

Abbildung 3.4: State-Chart zu Komponente C50: Kommunikation mit Roboter (Server).

Die Abbildung zu C50 zeigt die möglichen Zustände der Komponente *Server-Roboter-Kommunikation* beim Senden von Informationen an den Roboter. Wie auch der Roboter, versucht der Server zunächst eine Verbindung mit dem Roboter herzustellen. Ist der Verbindungsaufbau erfolgreich, wartet die Kommunikationskomponente, bis die Komponente zur Spielkoordination Daten zur Übertragung übergibt. Die Daten werden anschließend an den Roboter gesendet. Danach befindet sich das Kommunikationsmodul wieder im Wartemodus.

Einige Komponenten eignen sich nicht zur Verwendung in anderen Projekten. In der Programmierung der Komponente Spielkoordination sind die spielspezifischen Regeln und Ablaufreihenfolgen enthalten. Zudem sorgt die Spielkoordination für die Berechnung der Spielzüge, sodass sich diese Komponente nicht auf ein anderes Spiel übertragen lässt, ohne weite Teile neu schreiben zu müssen. Die Benutzeroberfläche ist auf das Spiel zugeschnitten. Eine weitere Verwendung ist nicht vorgesehen. Das Server-Server-Kommunikationsmodul basiert im Wesentlichen auf Standardoperationen in Java und sendet spezifische Spielinformationen. Eine Verwendung in anderen Programmen ist nicht sinnvoll.

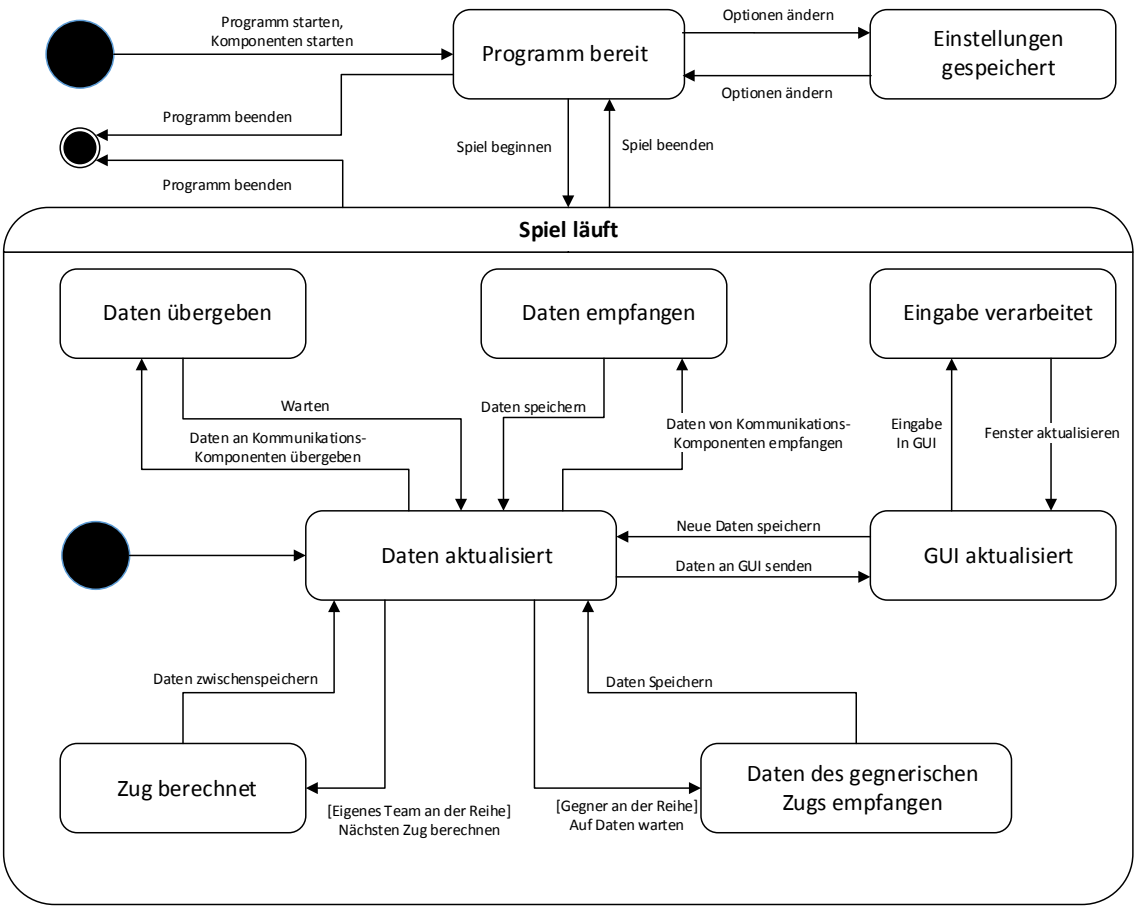

Abbildung 3.5: State-Chart zu Komponente C30: Spielkoordination (Server).

Die Spielkoordination kümmert sich um den reibungslosen Ablauf des Spiels. Wenn das Programm und die dazugehörigen Komponenten gestartet wurden, gibt es zunächst die Möglichkeit Parameter in den Optionen anzupassen. Das Programm kann jederzeit beendet werden. Nachdem das Spiel gestartet wurde, werden je nach Spielsituation verschiedene Aktionen ausgeführt. Falls sich an der grafischen Oberfläche etwas ändern muss, wird diese aktualisiert. Bewegungs- und Visualisierungsbefehle sowie Datenpakete für den gegnerischen Server werden bei Bedarf an die Kommunikationsmodule übergeben. Wenn das gegnerische Team an der Reihe ist, wird auf die aktualisierten Daten des anderen Teams gewartet. Ist das eigene Team an der Reihe, werden aus den vorhandenen Informationen die Spielzüge berechnet und die entsprechenden Befehle an die GUI und Kommunikationskomponenten gesendet.

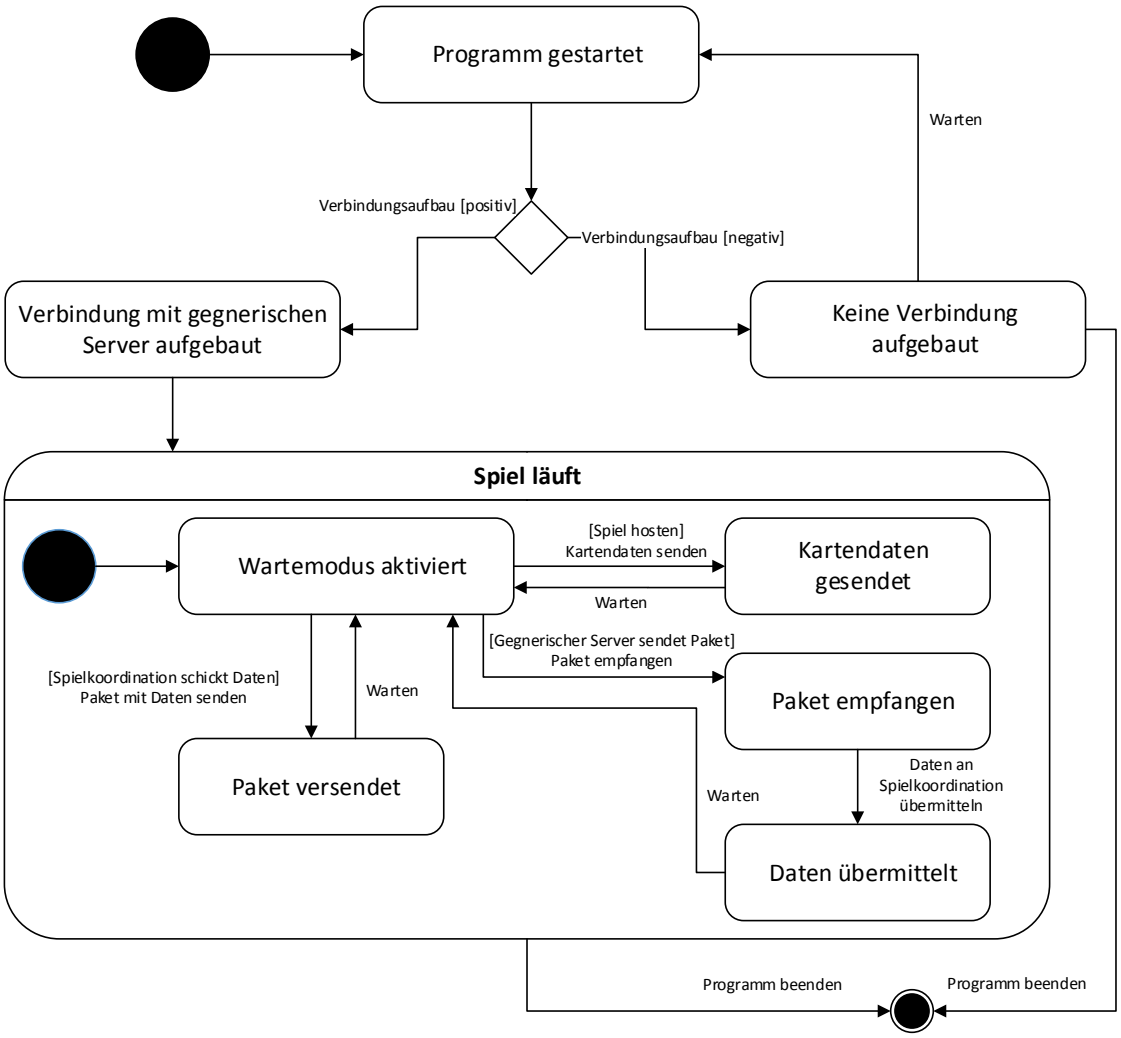

Abbildung 3.6: State-Chart zu Komponente C40: Kommunikation mit Gegner (Server).

Die Abbildung zu Komponente C40 zeigt den State-Chart der Server-Server-Kommunikation mit dem gegnerischen Team. Nach dem Start des Programms wird zunächst versucht eine funktionierende Verbindung mit dem anderen Server herzustellen. Wurde die Verbindung erfolgreich aufgebaut, befindet sich das Kommunikationsmodul zunächst im Wartemodus. Im Fall, dass unser Team ein Spiel erstellt, müssen die Kartendaten an den gegnerischen Server übermittelt werden. Während des Spiels werden fortlaufend, je nach Spielphase, Pakete gesendet und empfangen. Dabei werden definierte Informationen von und an die Spielkoordination übergeben.

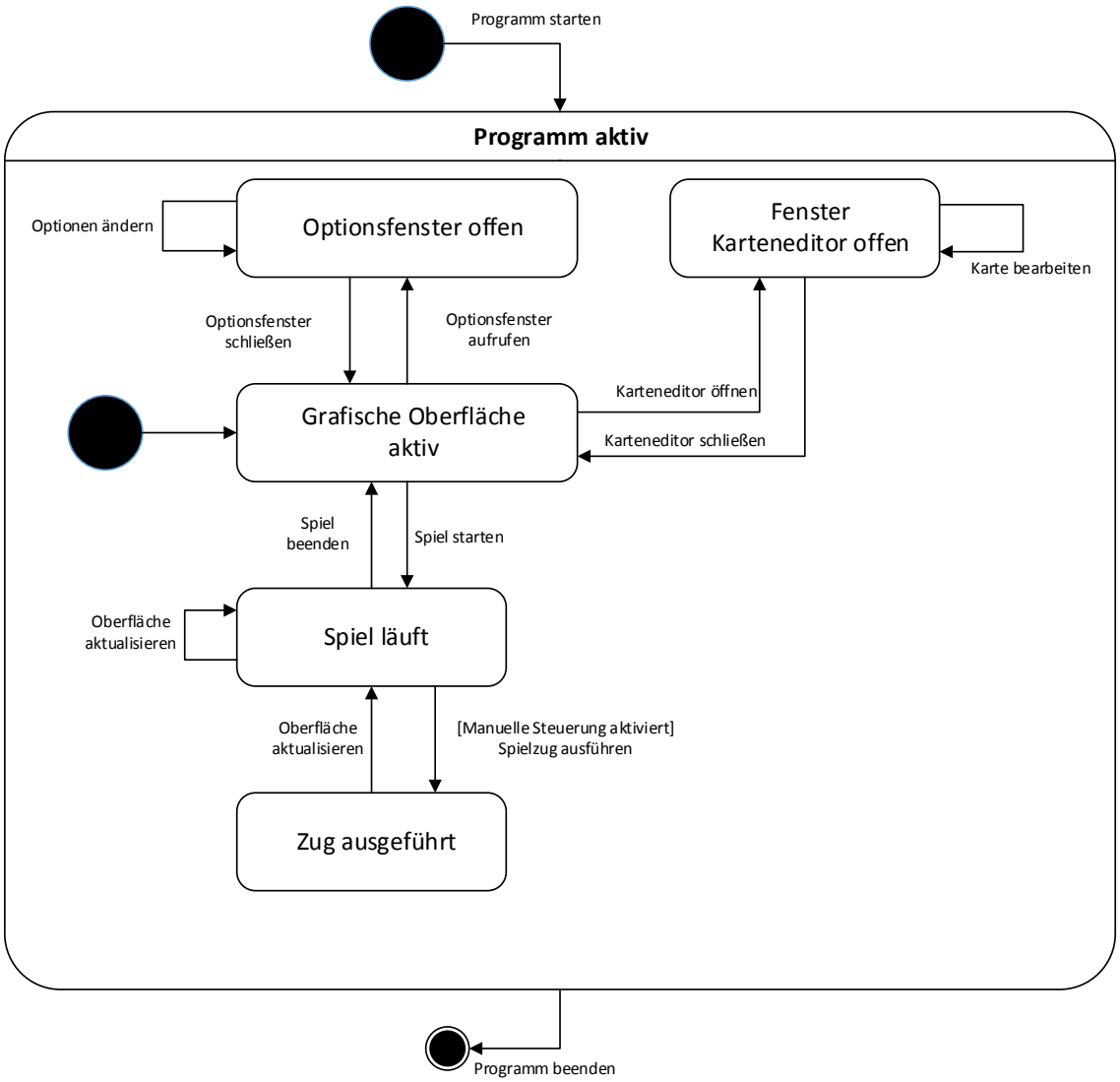

Abbildung 3.7: State-Chart zu Komponente C60: GUI (Server).

Die Abbildung zu Komponente C60 zeigt die verschieden Zustände, in denen sich die grafische Oberfläche befinden kann. Nach dem Start des Programms wird die grafische Oberfläche geladen und angezeigt. Es gibt die Möglichkeit das Optionsfenster oder einen Karteneditor zu öffnen, um dort Einstellungen vorzunehmen. Während des Spiels wird die Oberfläche fortlaufend aktualisiert. Die GUI ist aktiv, bis das Spiel beendet wird.

4 Verteilungsentwurf

In Abbildung 4.1 wird die Verteilung der Komponenten in einem Verteilungsdiagramm beschrieben. Der Server kommuniziert über WLAN mit dem LEGO Mindstorm EV3 Roboter und übergibt ihm so, welche Aktion er ausführen soll. Außerdem kommuniziert der Server über TCP/IP mit dem Server des gegnerischen Team um relevante Daten, wie Position des Gegners, von ihm gesandt zu bekommen. Damit eine Visualisierung in Form einer GUI möglich ist, muss der Server auch mit dieser über TCP/IP in Verbindung treten.

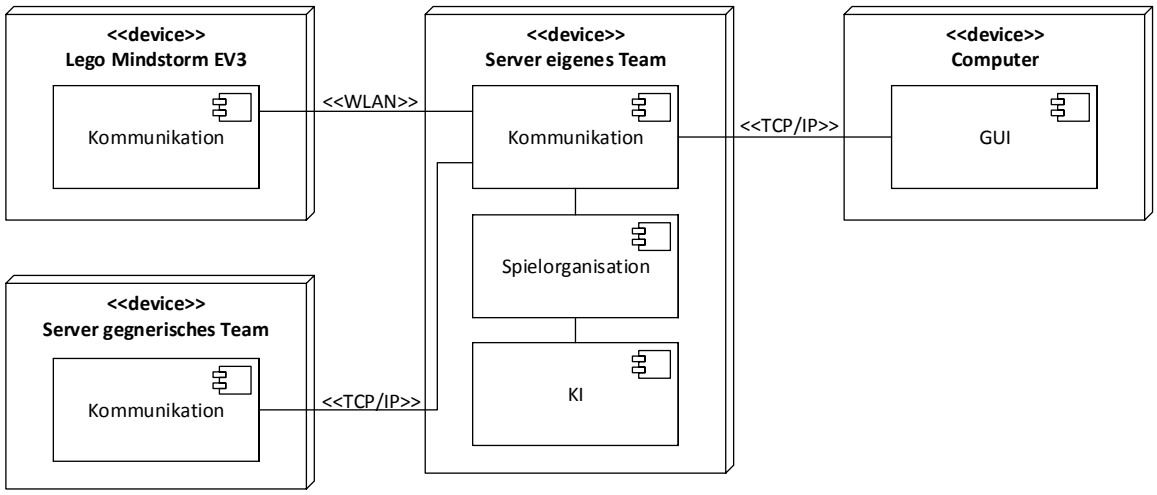

Abbildung 4.1: Verteilungsdiagramm

5 Implementierungsentwurf

Dieser Abschnitt hat die Aufgabe, alle verwendeten Klassen und Bibliotheken zu dokumentieren. Dabei wird jede Komponente aus Kapitel 3 gesondert betrachtet. Für Entwurfsentscheidungen, die mehr als eine Komponente betreffen, wird mit Verweisen zwischen den Dokumentationen der Komponente gearbeitet.

5.1 Implementierung von Komponente 10 <Sensoren/Aktoren (Roboter)>

Im folgenden Abschnitt wird die Implementierung der Komponente Sensoren/Aktoren erläutert. Die Aufgabe dieser Komponente ist es, das Spielfeld mit Hilfe von Sensoren zu erkennen, sowie die Bewegungen des Roboters zu steuern.

5.1.1 Paket-/Klassendiagramm

Das folgende Paketdiagramm gibt einen Überblick über die Struktur der Komponente Sensoren/Aktoren:

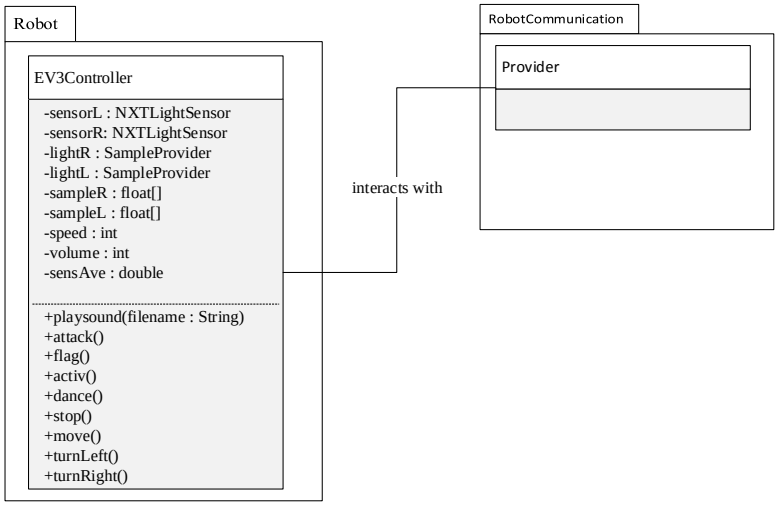

Abbildung 5.1: Paketdiagramm zu Sensoren/Aktoren

5.1.2 Erläuterung

EV3Controller$\langle CL10 \rangle$

Aufgabe

Steuerung und Verwaltung von Sensoren und Aktoren des Roboters

Attribute

`NXTLightSensor sensorL`: Der linke Lichtsensor des Roboters.

`NXTLightSensor sensorR`: Der rechte Lichtsensor des Roboters.

`SampleProvider lightR`: SampleProvider des rechten Lichtsensors. Der SampleProvider ruft die Daten des Sensors ab.

`SampleProvider lightL`: SampleProvider des linken Lichtsensors. Der SampleProvider ruft die Daten des Sensors ab.

`float sampleR[]`: Vom SampleProvider abgerufene Daten gespeichert als float Array.

`float sampleL[]`: Vom SampleProvider abgerufene Daten gespeichert als float Array.

`int speed`: Geschwindigkeit des Roboters.

`int volume`: Prozentuale Angabe der Lautstärke des Roboterlautsprechers.

`double sensAve`: Lichtsensormittelwert. Überschreitet der Lichtsensor diesen Wert erkennt der Roboter, dass sich eine Linie des Spielfeldes unter seinem Sensor befindet, bzw. nicht mehr befindet.

Operationen

`playsound(filename: String)`: Spielt einen Sound über den Lautsprecher des Roboters ab.

`attack()`: Visualisiert einen Angriff des Roboters.

`flag()`: Lässt den Roboter anzeigen, dass er gerade die Flagge hat.

`activ()`: Lässt den Roboter anzeigen, dass er gerade aktiv ist.

`dance()`: Lässt den Roboter einen Tanz aufführen.

`stop()`: Stopt die Motoren des Roboters.

`move()`: Lässt den Roboter ein Feld nach vorne fahren.

`turnLeft()`: Dreht den Roboter um 90 Grad nach links.

`turnRight()`: Dreht den Roboter um 90 Grad nach rechts.

Kommunikationspartner

-

5.2 Implementierung von Komponente 20 <Kommunikation Server (Roboter)>

Im folgenden Abschnitt wird die Implementierung der Komponente Kommunikation mit Server (Roboter) erläutert.

5.2.1 Paket-/Klassendiagramm

Das folgende Paketdiagramm gibt einen Überblick über die Struktur der Komponente Kommunikation mit Server (Roboter):

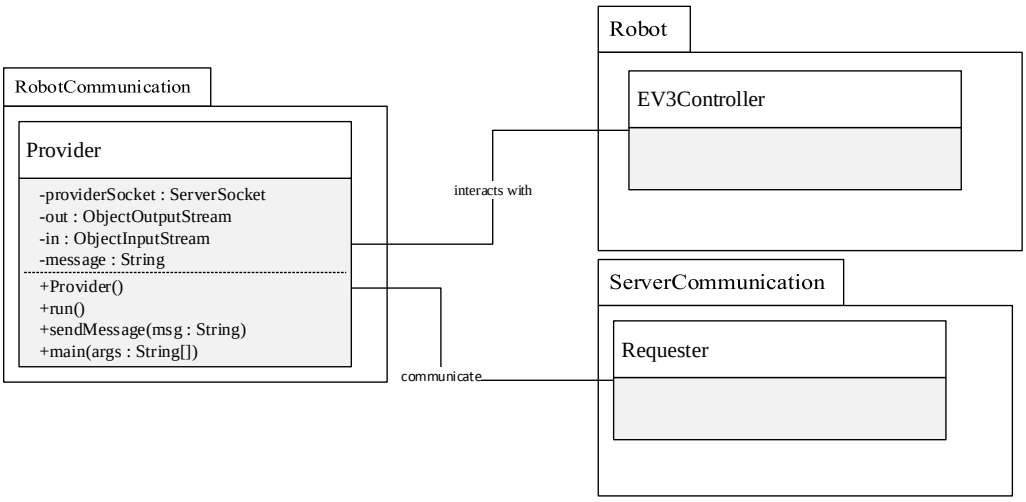

Abbildung 5.2: Paketdiagramm zu Sensoren/Aktoren

5.2.2 Erläuterung

Provider⟨$CL10$⟩

Aufgabe

Roboter Kommunikation ermöglichen

Attribute

`ServerSocket providerSocket`: Das ServerSocket wartet auf eingehende Anfragen über das Netzwerk.

`ObjectOutputStream out`: out ist ein DataOutputStream vom Roboter zum Server.

`ObjectInputStream in`: ist ein DataInputStream vom Server zum Roboter.

`String message`: Der String enthält die gesendeten und empfangenen Nachrichten.

Operationen

`Provider()`: Konstruktormethode der Klasse Provider.

`run()`: Diese Methode erzeugt über Sockets eine Verbindung zwischen Roboter und Server. Ist diese Verbindung hergestellt, wartet die Methode auf den Eingang von Nachrichten. Eingehende Nachrichten werden interpretiert und es werden entsprechende Methoden der EV3Controller Klasse aufgerufen, um die empfangenen Befehle umzusetzen.

`sendMessage(msg: String)`: Diese Methode sendet die übergebene Nachricht über den ObjectOutputStream zum Server.

`main(args: String[])`: Die main-Methode erzeugt eine Instanz von Provider und ruft die run() Methode dieser auf.

Kommunikationspartner

Server

5.3 Implementierung von Komponente 30 <Spielorganisation (Server)>

Im folgenden Abschnitt wird die Implementierung der Komponente Spielkoordination erläutert. Die Aufgabe dieser Komponente ist es, das Spiel zu koordinieren und eine künstliche Intelligenz (KI) zur Verfügung zustellen.

5.3.1 Paket-/Klassendiagramm

Das folgende Paketdiagramm gibt einen Überblick über die Struktur der Komponente Spielkoordination:

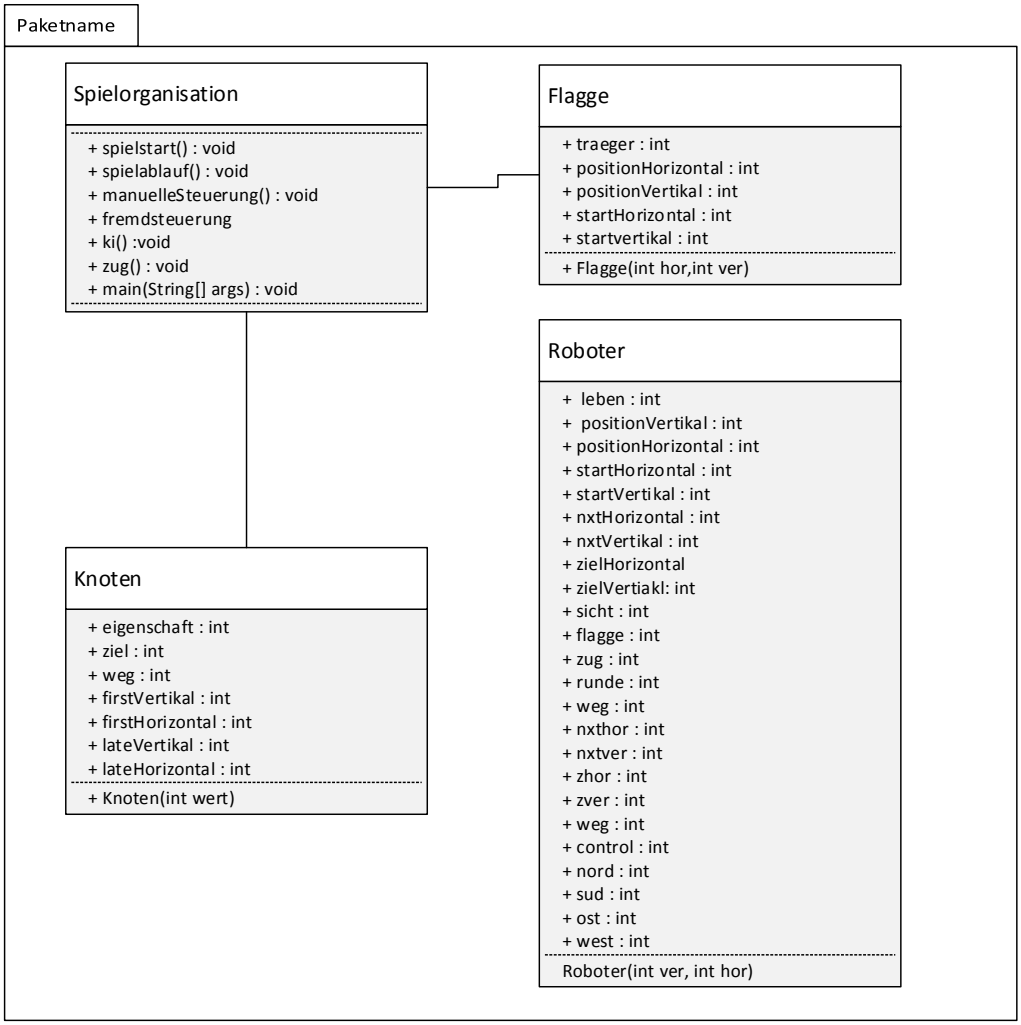

Abbildung 5.3: Spielkoordination (Server)

5.3.2 Erläuterung

Flagge$\langle CL10 \rangle$

Aufgabe

Speichert die Werte der Flaggen

Attribute

`int traeger`: Speichert den Träger der Flagge

`int positionHorizontal/Vertikal`: Speichert die Position der Flagge

`int startHorizontal/Vertikal`: Speichert die Startposition der Flagge

Operationen

`Flagge(int hor, int ver)`: Ist der Konstruktor der Klasse

Kommunikationspartner

Spielorganisation

Knoten⟨*CL*20⟩

Aufgabe

Speichert die Werten der Knoten

Attribute

`int eigenschaft`: Speichert die Eigenschaft der Knoten

`int ziel`: Makiert den Zielknoten für die KI

`int weg`: Speichert die Weite des Weges für die KI

`int firstHorizontal/Vertikal`: Speichert den Vorgängerknoten für die KI

`int lateHorizontal/Vertikal`: Speichert den Nachfolgeknoten für die KI

Operationen

`Knoten(int eigenschaft)`: Ist der Konstruktor der Klasse

Kommunikationspartner

Spielorganisation

Roboter⟨*CL*30⟩

Aufgabe

Speichert die Werten der Roboter.

Attribute

`int Leben`: Speichert das Leben des Roboters .

`int positionHorizontal/Vertikal`: Speichert die Position des Roboters.

`int startHorizontal/Vertikal`: Speichert die Startposition des Roboters.

`int nxtHorizontal/Vertikal`: Speichert die nächste Position des Roboters für di KI.

`int zielHorizonta/Vertiakl`: Speichert die Zielposition des Roboters für die KI.

`int sicht`: Zeigt, dass der Roboter vom gegnerischen Team gesehen wird.

`int flagge`: Zeigt, dass der Roboter die Flagge trägt.

`int zug`: Zeigt die aktuellen Zug des Roboters.

`int runde`: Zeigt die aktuelle Runde des Roboters.

`int weg`: Speichert die Weite des kürzesten Weges fü die KI.

`int control`: Zeigt die Art der steuerung des Roboters.

`int nord`: Zeigt die Bewegungsmöglichkeit des Roboters nach oben.

`int sud`: Zeigt die Bewegungsmöglichkeit des Roboters nach unten.

`int ost`: Zeigt die Bewegungsmöglichkeit des Roboters nach rechts.

`int west`: Zeigt die Bewegungsmöglichkeit des Roboters nach links.

Operationen

`Roboter(int eigenschaft)`: Ist der Konstruktor der Klasse

Kommunikationspartner

Spielorganisation

Spielorganisation$\langle CL40 \rangle$

Aufgabe

Steuert den Spielablauf, die Mechanik der Spielregeln und die KI.

Attribute

`Roboter roboter[][]`: Erstellt Objekte der Klasse Roboter.

`Flagge flagge[]`: Erstellt Objekte der Klasse Flagge.

`Knoten knoten[][]`: Erstellt Objekte der Klasse Knoten.

Operationen

`void Spielstart()`: Steuert den Spielbeginn und die Starteinstellungen.

`void Spielablauf()`: Steuert die Reihenfolge der Roboter.

`void ManuelleSteuerung()`: Steuert die Übergabe der Daten für die manuelle Steuerung.

`void Fremdsteuerung()`: Steuert die Übergabe der Daten zwischen den Servern.

`void KI()`: Errechnet auf Grundlage der Spielsituation den nächsten Zug des Roboters.

`void Zug()`: Berechnet die Veränderung auf Grund der Bewegungen der Roboter.

`void main(String[] args)`: Die main-Methode

Kommunikationspartner

Flagge

Knoten

Roboter

5.4 Implementierung von Komponente 40 <Kommunikation Gegner (Server)>

5.4.1 Paket-/Klassendiagramm

Das folgende Klassendiagramm gibt einen Überblick über die Struktur der Komponente Kommunikation mit dem Gegner (Server):

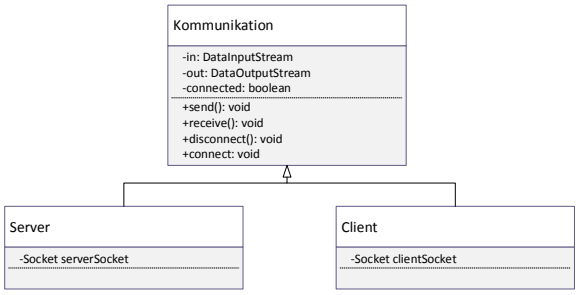

Abbildung 5.4: Kommunikation Gegner (Server)

5.4.2 Erläuterung

Kommunikation$\langle CL10 \rangle$

Aufgabe

Stellt die Attribute und Operationen für Server und Client bereit.

Attribute

`DataOutputStream out`: out ist ein DataOutputStream vom Server zum Client.

`DatatInputStream in`: in ist ein DataInputStream vom Client zum Server.

`connected`: Wird auf *true* gesetzt, wenn eine Verbindung aufgebaut ist.

Operationen

`send()`: Sendet ein Datenpaket.

`receive()`: Empfängt ein Datenpaket

`disconnect()`: Beendet eine Verbindung.

`connect()`: Baut eine Verbindung zwischen Server und Client auf.

Kommunikationspartner

-

Server$\langle CL20 \rangle$

Aufgabe

Kommunikation mit gegnerischem Client ermöglichen.

Attribute

`Socket serverSocket`: Dient als Endpunkt für die Kommunikation von Server und Client.

Operationen

Erbt Methoden der Klasse Kommunikation.

Kommunikationspartner

-

Client$\langle CL10 \rangle$

Aufgabe

Kommunikation mit gegenerischem Server ermöglichen.

Attribute

`Socket clientSocket`: Dient als Endpunkt für die Kommunikation von Server und Client.

Operationen

Erbt Methoden der Klasse Kommunikation.

Kommunikationspartner

-

5.5 Implementierung von Komponente 50 <Kommunikation Roboter (Server)>

5.5.1 Paket-/Klassendiagramm

Das folgende Paketdiagramm gibt einen Überblick über die Struktur der Komponente Kommunikation mit Roboter (Server):

5.5.2 Erläuterung

Requester$\langle CL10 \rangle$

Aufgabe

Kommunikation zwischen Server und Roboter ermöglichen

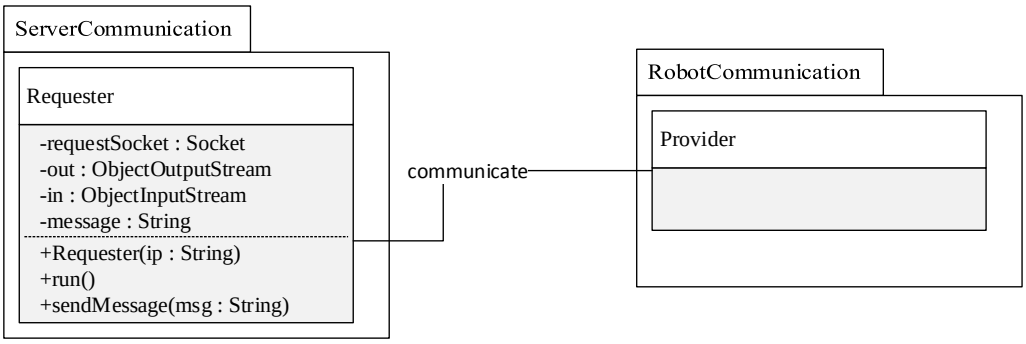

Abbildung 5.5: Paketdiagramm zu Kommunikation mit Roboter (Server)

Attribute

`Socket requestSocket`: Das Socket dient als Endpunkt für die Kommunikation zwischen Server und Roboter.

`ObjectOutputStream out`: out ist ein DataOutputStream vom Server zum Roboter.

`ObjectInputStream in`: in ist ein DataInputStream vom Roboter zum Server.

`String message`: Der String enthält die gesendeten und empfangenen Nachrichten.

Operationen

`Requester(ip: String)`: Konstruktormethode der Klasse Requester.

`run()`: Diese Methode erzeugt über Sockets eine Verbindung zwischen Roboter und Server. Ist diese Verbindung hergestellt, können Befehle an den Roboter gesendet werden.

`sendMessage(msg: String)`: Diese Methode sendet die übergebene Nachricht über den ObjectOutputStream zum Roboter.

Kommunikationspartner

Roboter

5.6 Implementierung von Komponente 60 <GUI (Server)>

Im folgenden Abschnitt wird die Implementierung der Komponente GUI erläutert. Die Aufgabe dieser Komponente ist es, das Spiel zu visualisieren und zu steuern. Auch gehören das Erstellen, Speichern und Laden von Karten sowie die Manuelle Steuerung zu dieser Komponente.

5.6.1 Paket-/Klassendiagramm

5.6.2 Erläuterung

Das folgende Paketdiagramm gibt einen Überblick über die Struktur der Komponente GUI:

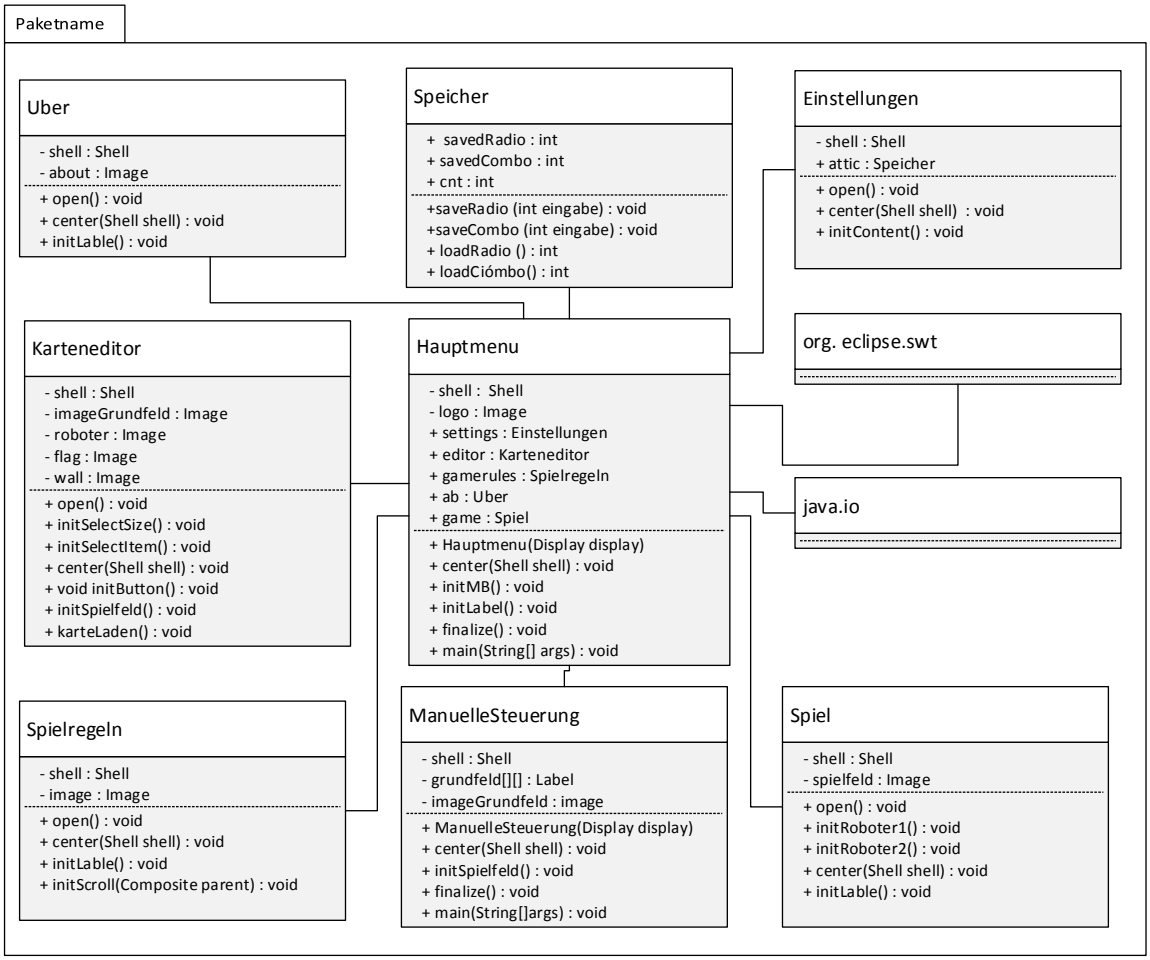

Abbildung 5.6: GUI (Server)

Uber$\langle CL10 \rangle$

Aufgabe

Dem Benutzer eine Übersicht über zusätzliche Informationen geben

Attribute

`Shell shell`: Aufrufen des Anzeigefensters

`Image about`: Aufrufen des Hintergrundbildes

Operationen

`open()`: Öffnen der Oberfläche

`center(Shell shell)`: Zentriertes Anzeigen des neuen Fensters

`initLable()`: Anzeigen der Spielversion sowie der Entwicklernamen

Kommunikationspartner

-

Speicher⟨$CL20$⟩

Aufgabe

Zwischenspeichern der Informationen über die vom Benutzer ausgewählte Steuerung und Karte

Attribute

`savedRadio`: Variable zum Speichern der vom Benutzer ausgewählten Steuerung, entweder die manuelle oder die KI Steuerung

`savedCombo`: Variable zum Speichern der Kartenauswahl des Benutzers

`cnt`: Zählervariable

Operationen

`saveRadio(int Eingabe)`: Speichern der ausgewählten Steuerung

`saveCombo(int Eingabe)`: Speichern der ausgewählten Karte

`loadRadio()`: Laden einer ausgewählten Steuerungseinstellung

`loadCombo()`: Laden einer ausgewählten Karte

Kommunikationspartner

-

Einstellungen⟨$CL30$⟩

Aufgabe

Dem Benutzer ermöglichen, eigene Einstellungen bezüglich der Steuerung und der Karten vorzunehmen

Attribute

`Shell shell`: Aufrufen des Anzeigefensters

`Speicher attic`: Erzeugen der Variable Speicher zur temporären Speicherung der Einstellungen für die Laufzeit des Programms

Operationen

`open()`: Öffnen der Oberfläche

`center(Shell shell)`: Zentriertes Anzeigen des Fensters

`initContent()`: Initialisierung der Buttons

Kommunikationspartner

-

Karteneditor⟨$CL40$⟩

Aufgabe

Dem Benutzer eine Visualisierung, Änderung und Erstellung der Karte ermöglichen

Attribute

`Shell shell`: Aufrufen des Anzeigefensters

`Image imageGrundfeld`: Anzeigen des Spielfeldes als einfache, unbearbeitete Variante

`Image roboter`: Icon, durch den die Roboter dargestellt werden

`Image flag`: Icon, durch den die Flaggen dargestellt werden

`Image wall`: Icon, durch den die Hindernisse (Wände) dargestellt werden

Operationen

`open()`: Öffnen der Oberfläche

`initSelectSize()`: Öffnen des Menüs zur Festlegung der Spielfeldgröße

`initSelectItem()`: Öffnen des Menüs zur Festlegung der Attribute des ausgewählten Feldes, also zur Platzierung von Robotern, Flaggen und Hindernissen

`center(Shell shell)`: Zentriertes Anzeigen des neuen Fensters

`initButton()`: Anlegen des Buttons, mit dem ein Dialog mittels karteLaden() zum Laden der Karte angezeigt werden kann

`initSpielfeld()`: Erstellen des Spielfeldes mittels Einsetzen von Bildern

`karteLaden()`: Darstellung eines Dialogs zum Laden einer zuvor gespeicherten Karte

Kommunikationspartner

-

org.eclipse.swt$\langle CL50 \rangle$

Aufgabe

Bereitstellen von Komponenten aus der swt-Bibliothek zur Implementierung der Graphischen Oberfläche

Attribute

-

Operationen

-

Kommunikationspartner

-

java.io$\langle CL60 \rangle$

Aufgabe

Bereitstellen von Komponenten aus der java-Bibliothek zur Implementierung der Graphischen Oberfläche

Attribute

-

Operationen

-

Kommunikationspartner

-

Spielregeln⟨*CL*70⟩

Aufgabe

Dem Benutzer eine Erläuterung der Spielregeln bereitstellen

Attribute

`Shell shell`: Aufrufen des Reiters „Spielregeln“ in der Oberfläche

`Image image`: Anzeige des Hintergrunds

Operationen

`open()`: Öffnen des Reiters

`center(Shell shell)`: Zentriertes Anzeigen des Fensters

`initLable()`: Anzeigen des Bildes, das die Spielregeln enthält

`initScroll(Composite parent)`: Initialisieren der Scrollbar

Kommunikationspartner

-

Manuelle Steuerung⟨*CL*80⟩

Aufgabe

Oberfläche zur manuellen Steuerung durch den Nutzer

Attribute

`Shell shell`: Aufrufen des Anzeigefensters

`grundfeld[][]`: Feld zur Anzeige der Positionsveränderung der Roboter

`Image imageGrundfeld`: Anzeige des Spielfeldes

Operationen

`Manuelle Steuerung (Display display)`: Anzeigen der vereinfachten Benutzeroberfläche für die manuelle Steuerung

`center(Shell shell)`: Zentriertes Anzeigen der Oberfläche

`initSpielfeld()`: Erstellen des Spielfeldes mittels Einsetzen von Bildern

`finalize()`: Schließen der Bildanzeige

`main(String[]args)`: Main-Methode

Kommunikationspartner

-

Spiel⟨$CL90$⟩

Aufgabe

Visualisierung des Spielablaufs für den Benutzer

Attribute

`Shell shell`: Aufrufen des Anzeigefensters für das laufende Spiel

`Image Spielfeld`: Anzeige des Spielfeldes des laufenden Spiels

Operationen

`open()`: Öffnen des Anzeigefensters

`initRoboter1()`: Anzeigen des Icons für den Roboter mit der ID 1 unseres Teams

`initRoboter2()`: Anzeigen des Icons für den Roboter mit der ID 2 unseres Teams

`center(Shell shell)`: Zentriertes Anzeigen der Oberfläche

`initLable()`: Anzeige des Spielfeldes

Kommunikationspartner

-

Hauptmenu⟨$CL100$⟩

Aufgabe

Anzeige des Hauptmenüs des Spieles, Übersicht über die einzelnen Aktionen und Informationen

Attribute

`Shell shell`: Aufrufen des Anzeigefensters

`Image logo`: Erzeugung einer Variable zur Anzeige des Logos des Spiels

`Einstellungen settings`: Erzeugung einer Variable zur Anzeige der Einstellungen

`Karteneditor editor`: Erzeugung einer Variable zur Anzeige des Karteneditors

`Spielregeln gamerules`: Erzeugung einer Variable zur Anzeige der Spielregeln

`Uber ab`: Erzeugung einer Variable zur Anzeige der Entwickler

`Spiel game`: Erzeugung einer Variable zur Anzeige des Spiels

Operationen

`Hauptmenu(Display display)`: Konstruktor, Erstellung des Hauptfensters und Aufrufen der einzelnen Methoden

`center(Shell shell)`: Zentriertes Anzeigen des Fensters

`initMB()`: Anzeigen der Menüleiste

`initLable()`: Anzeigen des Hintergrundbildes

`finalize()`: Schließen des angezeigten Bildes

`main(String[]args)`: Main-Methode

Kommunikationspartner

-

6 Datenmodell

Dieses Kapitel dient dazu, die langfristig gespeicherten Daten darzulegen. Hierbei handelt es sich um die Daten, die auch über das Schließen des Programmes hinaus gespeichert werden.

6.1 Diagramm

Abbildung 6.1: Diagramm der Klasse Kartendaten

6.2 Erläuterung

Die Daten der Karten können über das eigentliche Spielende hinaus in einer TXT-Datei gespeichert werden, um nicht verloren zu gehen, was besonders bei eigenen erstellten Karten für den Benutzer ausgesprochen ärgerlich sein könnte. Dabei wird jedes Kartenfeld mit einer Variable versehen, welche die leeren Felder, die Startpositionen der Roboter, die Flaggenfelder und die Hindernisse kennzeichnet, sodass ein String aus Variablen für das gesamte Spielfeld entsteht. Dieser kann beim Einlesen wieder in eine grafisch leicht zu überblickende Karte umgewandelt werden. Dies ist sowohl auf dem eigenen Server, als auch auf dem gegnerischen Server möglich, damit bei gemeinsamem Spiel ein reibungsloser Ablauf gewährleistet ist. Ein Benutzer kann beliebig viele Karten erstellen und speichern.

7 Konfiguration

Bei dem Server dieses Projekts handelt es sich nicht um einen Server im klassischen Sinne. Vielmehr existiert ein serverähnliches System, welches das Spiel koordiniert und für die Spielmechanik zuständig ist.

Auf diesem System wird das Programm auf einer Java Virtual Machine ausgeführt.

Für die Konfiguration wird eine TXT-Datei benötigt, in der ersten TXT-Datei befindet sich der Aufbau der Karte und die Startpositionen der Roboter und der Flaggen. In den folgenden Textdateien sind die Positionen und Lebenspunkte der Roboter und die Positionen der Flaggen enthalten. Diese Informationen werden als Integerwerte in den jeweiligen Objekten gespeichert.

8 Änderungen gegenüber Fachentwurf

Gegenüber dem Fachentwurf sind keine Änderungen aufgetreten.

9 Erfüllung der Kriterien

In diesem Kapitel werden nochmals die Muss-, Soll- und Kannkriterien aufgelistet und auf die Komponenten in Kapitel drei verwiesen.

9.1 Musskriterien

In diesem Abschnitt werden die Kriterien aufgelistet, die das Produkt erfüllen muss.

RM1 *Der Roboter muss sich zu einem vorgegebenen Punkt bewegen können.*
Der Server kennt das gesamte Spielfeld und schickt dem Roboter über W-LAN ein Paket, in welchem der Befehl für den nächsten Punkt steht, zu welchem sich der Roboter bewegen muss, siehe **C50**. Den Punkt findet der Roboter durch Lichtsensoren, die die Helligkeit messen (**C10**). Hat der Roboter sein Ziel erreicht, sendet er eine Bestätigung an den Server, siehe **C20**.

RM2 *Der Roboter muss sich auf dem Gitternetz mittels Helligkeitssensoren orientieren.*
Das Spielfeld, auf welchem sich der Roboter bewegt, wird aus schwarzem und hellem Klebeband auf dem Boden befestigt. Durch Lichtsensoren, die die Helligkeit messen, kann der Roboter sich anhand des erstellten Gitternetzes orientieren. Der weiße Untergrund wird als „0" und der Dunkle als einen Wert „>0" gemessen. Dies wird durch die Komponente **C10** umgesetzt.

RM3 *Der Roboter muss über Pakete und ein festgelegtes Protokoll mit dem Server kommunizieren.*
Die Kommunikation zwischen Roboter und Server wird via W-LAN hergestellt, siehe **C20** und **C50**. Das festgelegte Protokoll besteht aus Bewegungsbefehlen und Position des Roboters. Nach jedem Erhalt sendet der Empfänger eine Bestätigung an den Sender.

RM4 *Das serverseitig erstellte Spielfeld muss gleich groß oder kleiner als das aufgeklebte Gitternetz sein.*
In der Benutzeroberfläche hat man nur die Auswahl ein 8x8 großes Spielfeld zu erzeugen, siehe **C60**.

RM5 *Das Spielfeld muss durch Koordinaten beschreibbar sein.*
Das Spielfeld wird als 8x8 großes Feld erzeugt. Durch eine for-Schleife wird jedem Feld eine Koordinate zugewiesen, die einem Knoten auf dem Gitternetz entspricht (**C60**).

RM6 *Der Server muss die Karte verwalten.*
Die Kartendaten werden als .txt-Datei gespeichert und werden vom Server importiert (**C60**). Durch Methoden werden alle Kartendetails ausgelesen und durch die jeweiligen Attribute ersetzt.

RM7 *Der Server muss den Spielablauf verwalten.*
Die Spielorganisation,welche auf dem Server liegt, verwaltet den Spielablauf. Dabei bestimmt die Spielorganisation, welcher Roboter an der Reihe ist, ob ein gegnerischer Roboter abgeschossen oder eine Flagge aufgenommen werden kann. Dies passiert mittels W-LAN Verbindung und dem festgelegten Protokoll. Dies entspricht der Komponente **C30**.

RM8 *Der Server muss den Robotern Bewegungsbefehle senden.*
Der Server kommuniziert mittels Paketen mit dem Roboter, in welchem die Bewegungsbefehle stehen, siehe **C50**.

RM9 *Der Server muss den Aufenthaltsort der Roboter verwalten.*
Der Roboter sendet mit jeder Beendigung eines Zuges seinen Aufenthaltsort dem Server (**C20**). Dadurch weiß der Server, dass der Roboter seinen Zug beendet hat und auf welcher Position er sich befindet.

RM10 *Der Server muss den Status jedes Spielteilnehmers verwalten.*
Der Server erhält von allen Robotern, die sich auf dem Spielfeld befinden, immer Position und Lebenspunkte, siehe **C20**.

RM11 *Der Server muss die eigene KI der Spielteilnehmer berechnen.*
Dies wird über die Spielorganisation geregelt. Diese bestimmt die kürzeste Route zur Flagge, ob ein Roboter abgeschossen werden kann oder nicht und ob der Roboter sich bewegen soll oder nicht. Dies wird durch die Komponente **C30** dargestellt.

RM12 *Es muss ein Karteneditor zur Erstellung neuer Karten vorhanden sein.*
Die Karten werden mittels .txt-Datei erstellt. Dabei werden Hindernisse, freies Feld, Roboter und Flaggen mit einem Zeichen belegt, die in einer Methode ausgelesen und grafisch umgesetzt werden (**C60**).

RM13 *Der Karteneditor muss eine zweidimensional grafische Ausgabe besitzen.*
Die Karte wird durch ein 8x8-großes Feld erzeugt. Die .txt-Datei wird ausgelesen und mittels einer for-Schleife wird über das Feld iteriert und das jeweilige passende Bild eingefügt (**C60**).

RM14 *Es muss eine manuelle Steuerung der Roboter möglich sein.*
Die Manuelle Steuerung erfolgt unter Eingabe der Position der Roboter. Dabei wird die Position des Roboters durch Koordinaten eingegeben. Dies wird an den Server weitergeleitet. Die Spielorganisation berechnet den Weg und sendet dann über den Server einen Befehl an den Roboter (**C60**).

RM15 *Es muss eine definierte Paketverbindung zwischen den Servern der beiden Teams existieren.*
Zwei Teams lassen ihre Roboter gegeneinander antreten. Damit die jeweilige KI weiß, ob ein Roboter sich im Sichtfeld des anderen findet, muss der Aufenthaltsort jedes Roboters bekannt sein. Dies wird mit einer Server-Server-Kommunikation **C40** und einer Paketverbindung umgesetzt.

9.2 Sollkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

RS1 *Für alle Spielteilnehmer sollen die gleichen Spielbedingungen und Regeln vorherrschen.*
Der Server verwaltet die Spielregeln. Da jeder Zug von der Spielorganisation berechnet wird, wird auch automatisch überprüft ob der nächste Zug gültig ist oder die Regeln bricht. Züge, die gegen die Regel sind werden deshalb nicht ausgeführt (**C30**).

RS2 *Der Roboter soll eine vorgegebene Anzahl an Lebenspunkten haben.*
In der Benutzeroberfläche soll die Anzahl der Lebenspunkte des Roboters eingegeben und an den Server weitergegeben werden (**C60**).

RS3 *Der Roboter soll zum Startpunkt fahren, wenn die Lebenspunkte kleiner oder gleich Null sind.*
Der Server kennt den Status jedes Spielteilnehmers und wenn die Lebenspunkte des Roboters kleiner oder gleich Null sind, wird ein Paket an den Roboter gesendet (**C50**), wo der kürzeste Weg zum Startpunkt gegeben ist. Der kürzeste Weg wird durch eine Methode berechnet.

RS4 *Der Roboter soll seinen Status physisch anzeigen können.*
Auf dem Display des Roboters sollen die Lebenspunkte ausgegeben werden (**C10**).

RS5 *Im Karteneditor sollen Knoten für die Bewegung der Roboter gesperrt werden können.*
Gesperrte Knoten werden im Karteneditor als Wände dargestellt. Der Server wertet dies aus und sperrt dann die Knoten für den Roboter (**C60**).

9.3 Kannkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

RC1 *Das Spiel könnte um Bonusfelder erweitert werden.*
Neben Hindernissen, Flaggen und Roboter können weitere Zeichen als Bonusfelder erzeugt werden. Diese Bonusfelder könnten aus extra Lebenspunkte bestehen. Wenn ein Roboter auf einem Bonusfeld stehen würde und seine Position dem Server übergibt, weiß der Server vom Bonusfeld und zählt die Extraleben zu den derzeitigen Lebenspunkten des Roboters dazu.

RC2 *Es kann ein Aktionspunkte-System eingeführt werden, dass mehr Taktiken ermöglichen würde.*
Das Aktionspunkte-System baut auf die Bonusfelder auf. Dabei hat man eine feste Anzahl an Aktionspunkte, die langsam sinken, wenn man Aktionen wie Schießen ausführt. Dabei kommt mehr Taktik ins Spiel, da die KI auch noch berechnen muss, ob es sinnvoll ist, denn gegnerischen Roboter abzuschießen oder nicht.

RC3 *Aufbauend auf dem Aktionspunkte-System, würden verschiedene Aktionen unterschiedlich viele Aktionspunkte verbrauchen.*
Es könnte eine feste Anzahl an Aktionspunkte geben. Dabei kostet dann jede Aktion Punkte und wenn es keine Punkte mehr gibt, können Aktionen wie Schießen nicht ausgeführt werden.

RC4 *Die Positionen der Flaggen können vom Spieler verändert werden.*
Statt einer festen Flaggenstartposition, kann diese in der Benutzeroberfläche auch verändert werden.

RC5 *Es könnte eine Siegessequenz geben.*
Wenn der Roboter vom Server die Bestätigung erhält, dass das Spiel gewonnen wurde, kann der Roboter eine studierte Route abfahren und dabei Musik abspielen.

RC6 *Die Karten und Startpositionen könnten vom Benutzer gespeichert werden.*
Wenn eine Karte erstellt wurde, kann man diese als .txt-Datei abspeichern und beim nächsten Spiel wieder laden.

RC7 *Es könnte einen Generator für das Spielfeld geben.*
Anstatt selber ein Spielfeld zu erzeugen, könnt es einen Generator geben, der das perfekte Spielfeld aufbaut.

10 Glossar

Nachfolgend werden die im Technischen Entwurf verwendeten Fachbegriffe kurz erklärt.

Lego Mindstorm EV3:
Roboter der Produktserie Mindstorms von Lego, die mittels eines programmierbaren Legosteins (genannt *Brick*), Elektromotoren sowie Sensoren autonom Aufgaben erfüllen können.

GUI:
Graphical user interface - Die grafische Benutzerschnittstelle zwischen Programm und Benutzer zur Ein- und Ausgabe von Informationen.