# Lego hunts the Wumpus

## – Team 0 –

Software-Entwicklungspraktikum (SEP)
Sommersemester 2012

# G r o b e n t w u r f


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

Betreuer: Benjamin Mensing

Auftragnehmer:

| Name | E-Mail-Adresse |
|---|---|
| Lars Luthmann | l.luthmann@tu-bs.de |
| Marcel Mast | m.mast@tu-bs.de |
| Christian Pek | c.pek@tu-bs.de |
| Yevgen Pikus | y.pikus@tu-bs.de |
| Tobias Maximilian Vogt | t.vogt@tu-braunschweig.de |

Braunschweig, 23.05.2012

# Versionsübersicht

| Version | Datum | Autor | Status | Kommentar |
|---|---|---|---|---|
| 0.1 | 08.05.2012 | Luthmann, Mast, Pek, Pikus, Vogt | in Bearbeitung | erste Version der Analyse der Produktfunktionen |
| 0.2 | 11.05.2012 | Luthmann, Mast, Pek, Pikus, Vogt | in Bearbeitung | Analyse der Produktfunktionen überarbeitet |
| 0.3 | 12.05.2012 | Luthmann, Mast, Pek, Pikus | in Bearbeitung | Einfügen des Spielverlaufs |
| 0.4 | 12.05.2012 | Vogt | in Bearbeitung | Verteilungsentwurf eingefügt |
| 0.5 | 14.05.2012 | Luthmann, Mast, Pek, Pikus, Vogt | in Bearbeitung | diverse Fehler ausgebessert |
| 0.6 | 15.05.2012 | Pek | in Bearbeitung | Einleitung Kapitel 3 und Fertigstellung von 3.1 |
| 0.7 | 15.05.2012 | Luthmann, Pikus | in Bearbeitung | Fertigstellung von Kapitel 3.2 |
| 0.8 | 16.05.2012 | Luthmann, Mast, Pikus, Vogt | in Bearbeitung | Fertigstellung von Kapitel 3.3 |
| 1.0 | 18.05.2012 | Luthmann, Mast, Pek, Pikus, Vogt | in Bearbeitung | diverse Fehler ausgebessert; erster Release Candidate |
| 1.1 | 22.05.2012 | Luthmann, Mast, Pek, Pikus, Vogt | in Bearbeitung | weitere Fehler in allen Bereichen ausgebessert |
| 2.0 | 23.05.2012 | Luthmann, Mast, Pek, Pikus, Vogt | abgenommen | fertige Version |

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

Für die Simulation des Spiels *Hunt the Wumpus* mit Lego NXT Robotern benötigt man einen festen Spielablauf. Dieser beginnt mit dem Aufstellen und Starten der Systeme und endet mit dem Spiel an sich. Ohne diesen Ablauf ist die Simulation nicht möglich. Daher ist auf den korrekten Ablauf der Aktivitäten zu achten. Sie werden in den folgenden Aktivitätsdiagrammen grob beschrieben. Im Kapitel 1.1 werden die Abläufe detailliert dargestellt.

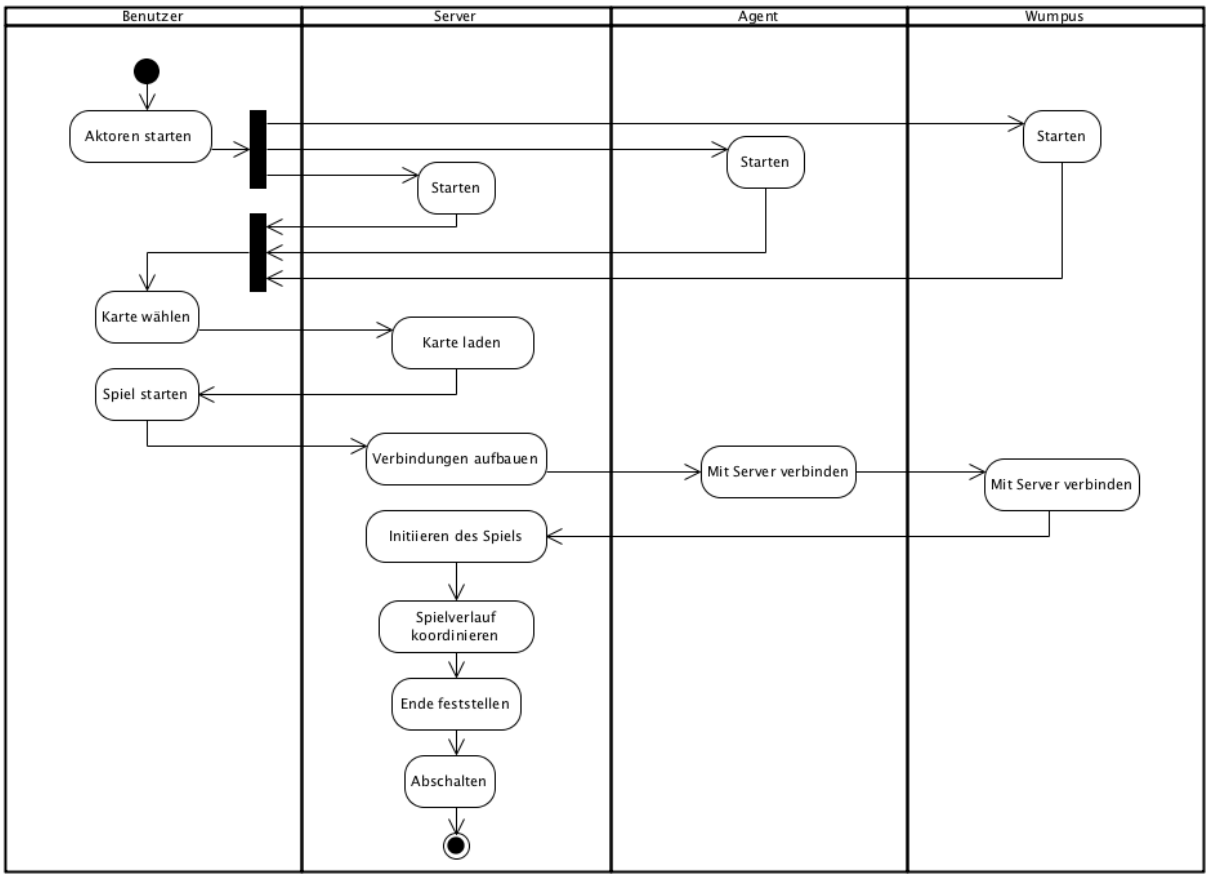


Abbildung 1.1: Aktivitätsdiagramm zum Simulationsverlauf 1

Als erstes muss der Benutzer die Simulation vorbereiten. Dazu gehört das Starten und Platzieren der Aktoren Server, Agent und Wumpus. Sobald der Server gestartet ist, lädt er eine zuvor vom Benutzer ausgewählte Karte. Damit ist die Vorbereitung abgeschlossen, sodass der Benutzer als nächstes das Spiel starten kann.

Der Server baut die Verbindung zum Agenten und Wumpus auf. Sobald die Verbindung steht, kann der Server mit dem Initiieren des Spiels beginnen und daraufhin das Spiel starten.

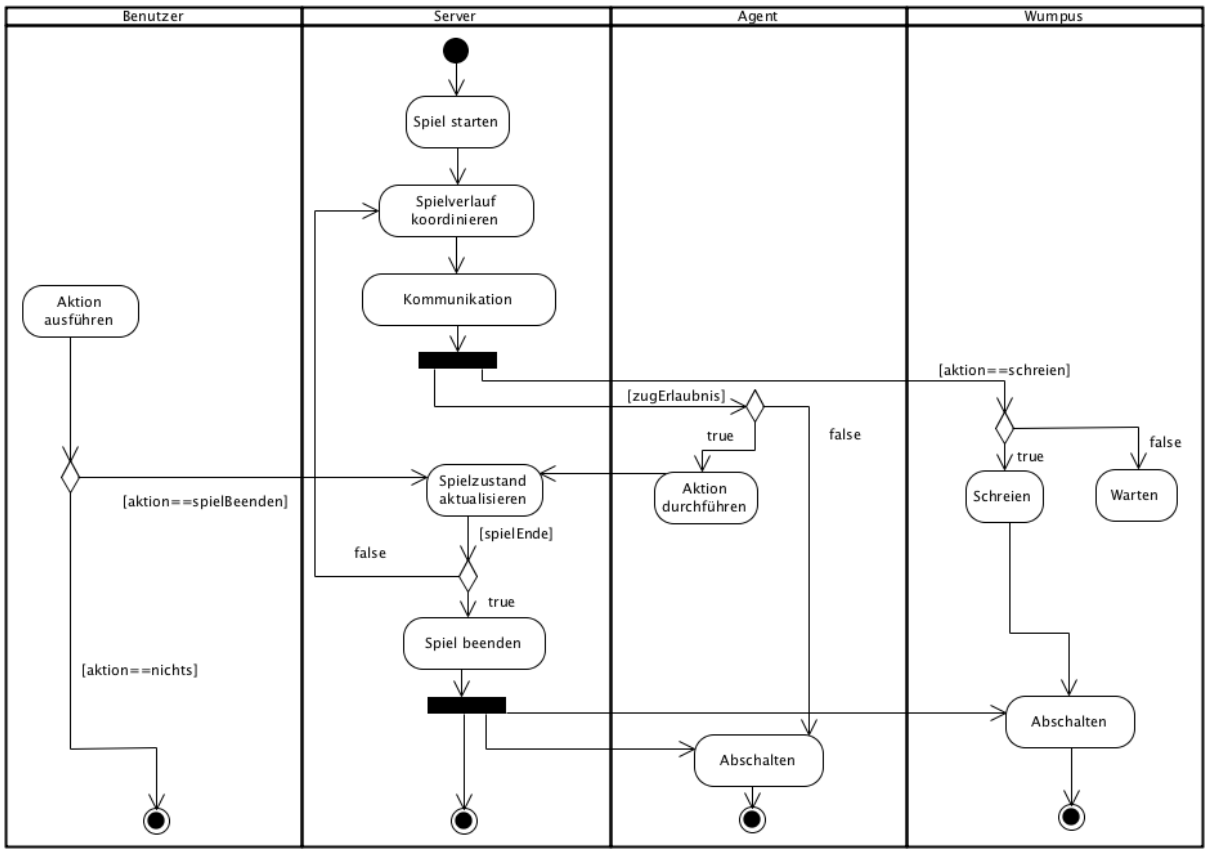


Abbildung 1.2: Aktivitätsdiagramm zum Simulationsverlauf 2

Das Spiel selbst besteht aus einer Folge von Aktionen, die im Kreis ausgeführt werden, solange das Spiel nicht beendet ist. Zu Beginn koordiniert der Server den Spielverlauf, indem er Daten und Aktionen für die Kommunikation vorbereitet. Nach der anschließenden Kommunikation mit den Aktoren, in der der Agent seinen Zug mitgeteilt hat, führt der Agent, falls erlaubt, seinen Zug aus. Gegebenenfalls führt der Wumpus seine Aktion *Schreien* aus. Zum Schluss aktualisiert der Server den Spielzustand und prüft, ob das Spiel zu Ende ist. Falls dem nicht so ist, wird das Spiel fortgesetzt. Der Benutzer hat während des gesamten Spielverlaufs die Möglichkeit das Spiel zu beenden. Beim Beenden des Spiels, werden alle Aktoren abgeschaltet.

## 1.1 Projektdetails

In diesem Abschnitt werden nun die beiden Diagramme aus der Einleitung aufgeteilt und detaillierter beschrieben.

### 1.1.1 Starten des Spiels

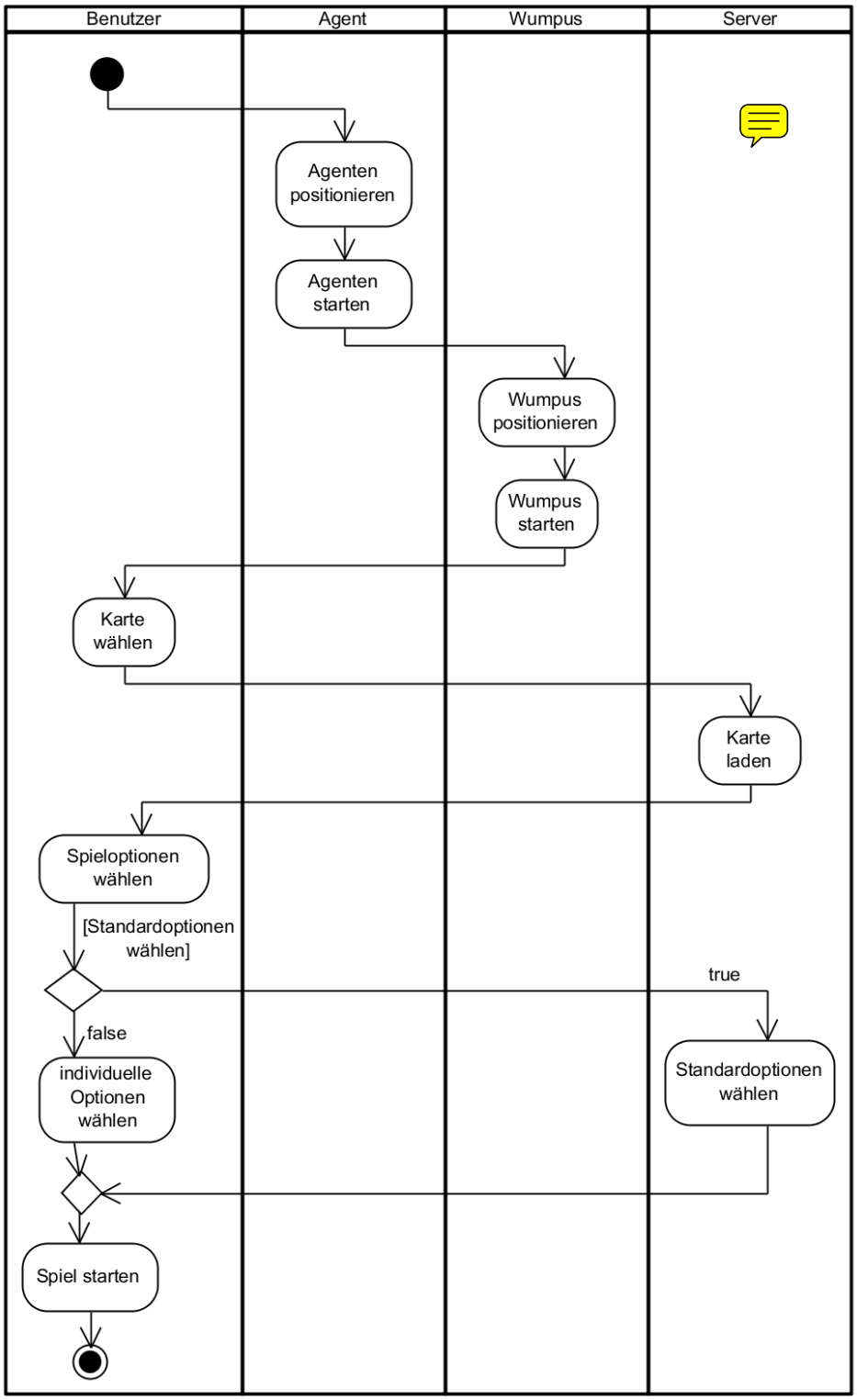


Abbildung 1.3: Aktivitätsdiagramm zum Starten des Spiels

Wie in Abbildung 1.3 zu sehen ist, wird zunächst der Agent positioniert und gestartet. Anschließend ist der Wumpus zu positionieren und zu starten. Dann hat der Benutzer eine Karte zu wählen, dabei entschließt er sich entweder dazu eine neue Karte zuerstellen oder eine bereits vorhandene Karte im Server zu laden. Ist eine Karte ausgewählt, werden die Spieloptionen vom benutzer gewählt. Dafür hat er die Wahl zwischen den Standardeinstellungen oder individuelle Einstellungen. Ist die Wahl erfolgt, so kann das Spiel vom Benutzer gestartet werden.

### 1.1.2 Verbindungsaufbau

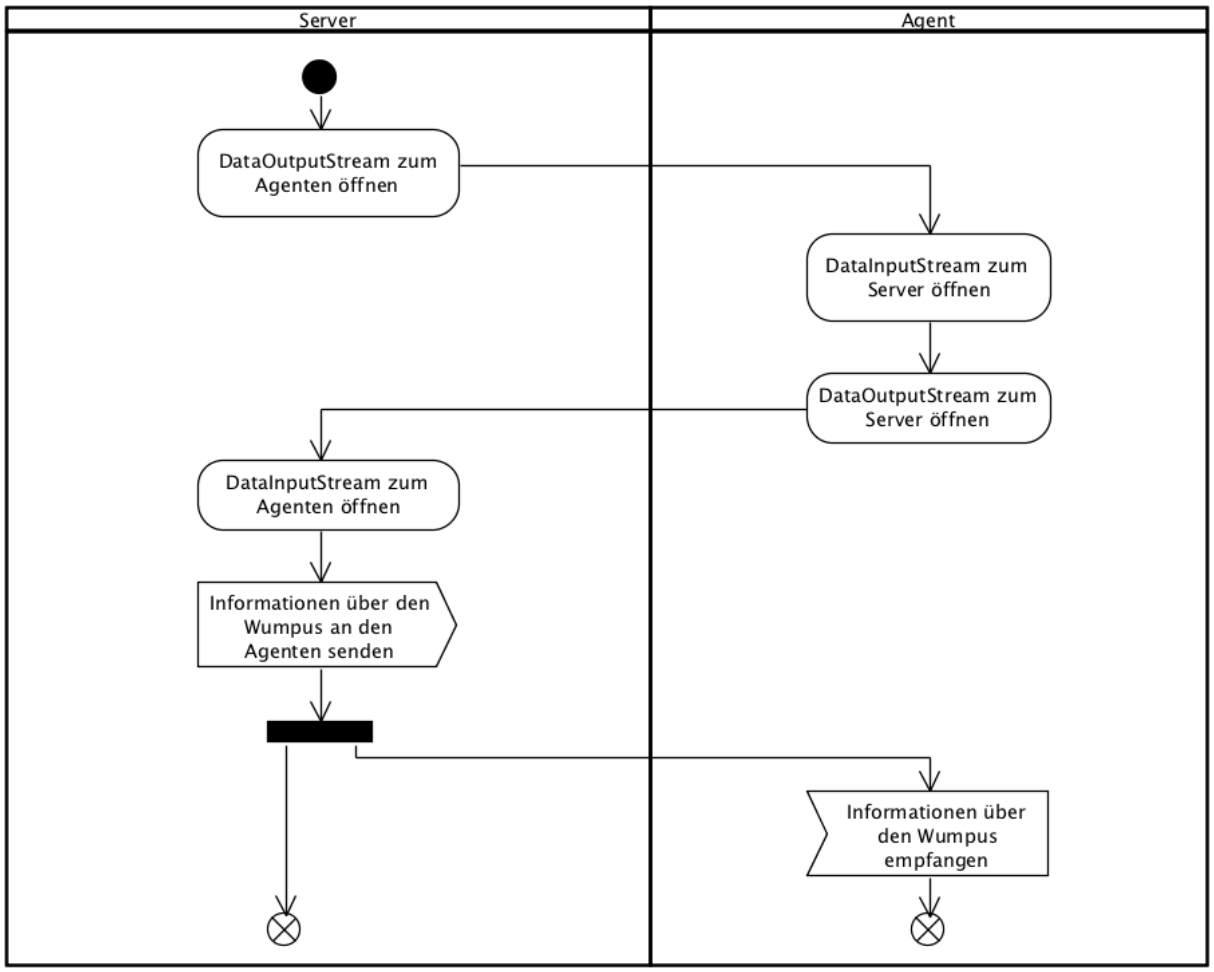


Abbildung 1.4: Aktivitätsdiagramm zum Verbindungsaufbau

Wie in Abbildung 1.4 zu sehen ist, wird beim Verbindungsaufbau zunächst der DataOutputStream vom Server zum Agenten geöffnet. Danach wird auf Agentenseite der entsprechende DataInputStream geöffnet. Nun hat der Server die Möglichkeit, Nachrichten an den Agenten zu schicken. Damit der Agent auch Nachrichten an den Server senden kann, muss danach auf Agentenseite der DataOutputStream zum Server und auf Serverseite der DataInputStream zum Agenten geöffnet werden. Zum Schluss sendet der Server an den Agenten eine Nachricht über

den Wumpus, in der steht, ob sich der Wumpus bewegen darf oder nicht. Diese wird daraufhin vom Agenten empfangen.

### 1.1.3 Kommunikation während des Spiels

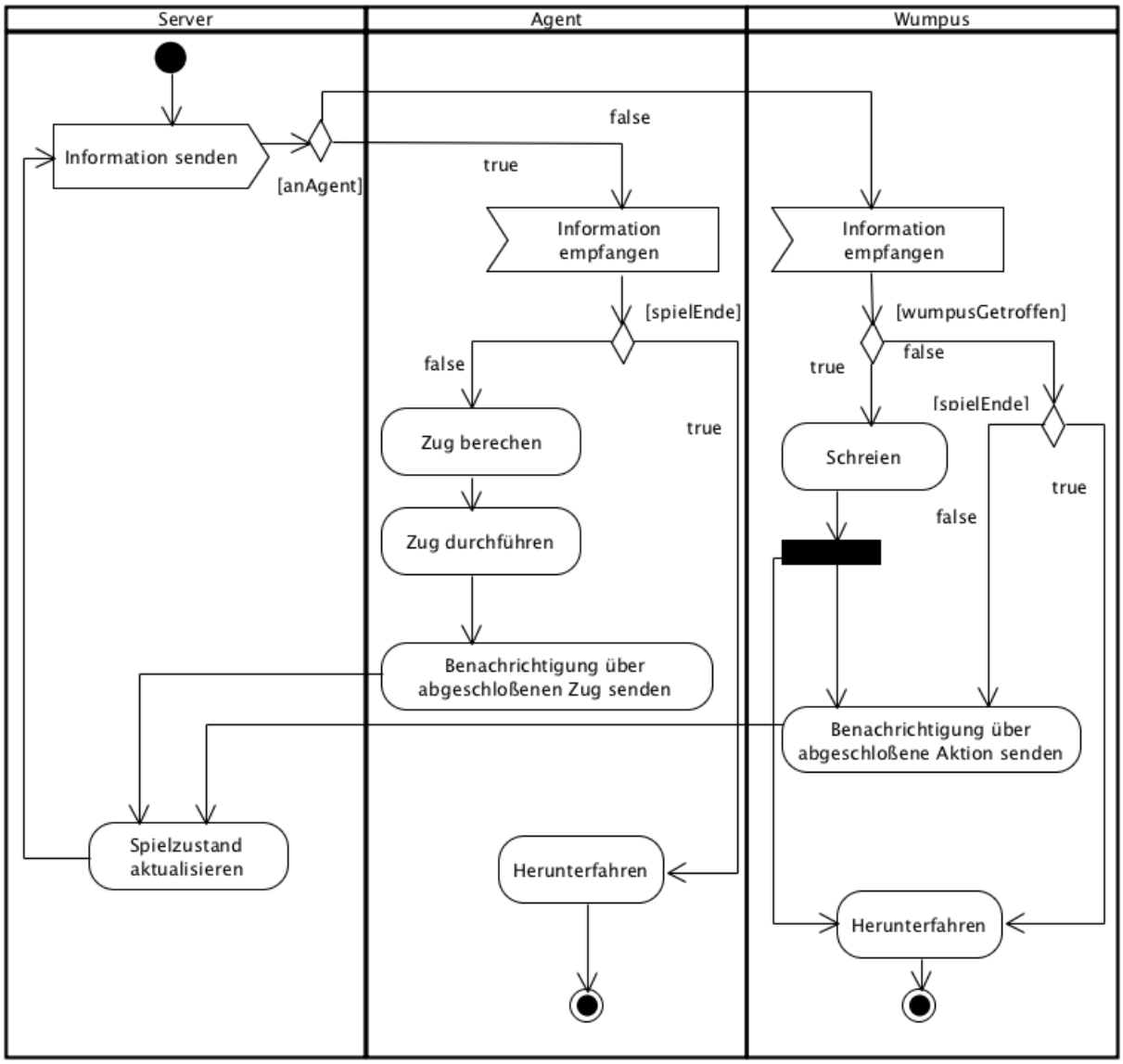


Abbildung 1.5: Aktivitätsdiagramm zur Kommunikation während des Spiels

Wie in Abbildung 1.5 zu sehen ist, besteht die Kommunikation aus einer Folge von Aktionen, die im Kreis ausgeführt werden, solange das Spiel nicht beendet ist. Der Server verschickt eine Mitteilung entweder an den Agenten oder an den Wumpus. Wenn der Wumpus feststellt, dass er getroffen wurde, dann führt er seine Aktion *Schreien* aus. Danach benachrichtigt er den Server über seine durchgeführte Aktion und der Roboter fährt herunter. Im anderen Fall wird der Server direkt benachrichtigt. Wenn der Agent feststellt, dass er den Raum mit Wumpus befuhr,

dann fährt der Roboter herunter. Im anderen Fall kann der Agent mittels seiner KI den nächsten Zug berechnen. Danach wird der Zug ausgeführt und eine Benachrichtigung über abgeschlossene Aktion an Server versandt. Anschließend aktualisiert der Server den Spielzustand und kehrt zur Aktion *Information senden* zurück. Der Server hat die Möglichkeit das Spiel zu beenden. Beim Beenden des Spiels, werden alle Aktoren heruntergefahren.

# 2 Analyse der Produktfunktionen

Im folgenden Kapitel werden die Produktfunktionen des Pflichtenhefts analysiert, um auf deren Basis eine geeignete Architektur festlegen zu können.

## 2.1 Analyse der Funktionalität /F10/: drehen

Bei der Funktion *drehen* sind die künstliche Intelligenz (im Folgenden *KI*), die Steuerung und die Sensoren/Aktoren des Agenten beteiligt. Wenn die KI entscheidet, dass eine 90°-Drehung stattfinden soll, dann meldet sie dies an die Steuerung. Diese gibt den Befehl an die Komponente *Sensoren/Aktoren* weiter, welche dann die 90°-Drehung durchführen. Durch das Zusammenfassen von Sensorik und Motorsteuerung in einer Komponente wird der Roboter beim Drehen automatisch korrrekt ausgerichtet (siehe Kapitel 3.1).

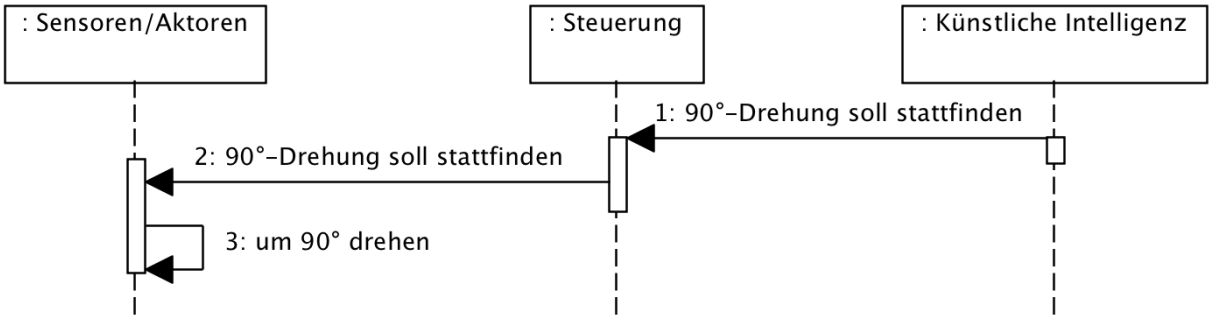


Abbildung 2.1: Sequenzdiagramm zu */F10/*

1. Die KI entscheidet, dass eine 90°-Drehung stattfinden soll und meldet dies der Steuerung.
2. Die Steuerung gibt den Befehl an die *Sensoren/Aktoren* weiter.
3. Die *Sensoren/Aktoren* drehen den Roboter um 90°.

## 2.2 Analyse der Funktionalität /F20/: geradeaus fahren

Bei der Funktion *geradeaus fahren* sind die künstliche Intelligenz (im Folgenden *KI*), die Steuerung und die Komponente *Sensoren/Aktoren* des Agenten beteiligt. Wenn die KI entscheidet, dass geradeaus gefahren werden soll, dann meldet sie dies an die Steuerung. Diese gibt den Befehl an die *Sensoren/Aktoren* weiter, welche dann ein Feld geradeaus fahren. Durch das Zusammenfassen von Sensorik und Motorsteuerung in einer Komponente wird der Roboter beim Bewegen automatisch korrrekt ausgerichtet (siehe Kapitel 3.1).

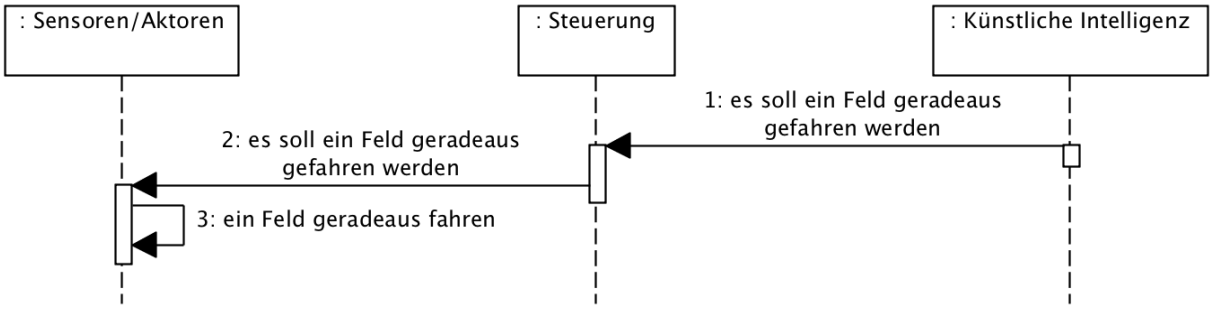


Abbildung 2.2: Sequenzdiagramm zu */F20/*

1. Die KI entscheidet, dass ein Feld geradeaus gefahren werden soll und meldet dies der Steuerung.
2. Die Steuerung gibt den Befehl an die Sensoren/Aktoren weiter.
3. Die Sensoren/Aktoren fahren ein Feld geradeaus.

## 2.3 Analyse der Funktionalität /F30/: Karte erstellen

Bei der Funktion *Karte erstellen* sind der Benutzer, ein externer Texteditor und der Server beteiligt. Der Benutzer erstellt eine Karte im XML Format in dem Texteditor und lässt diese vom Server laden. Das Laden läuft gemäß */F40/ Karte laden* ab.

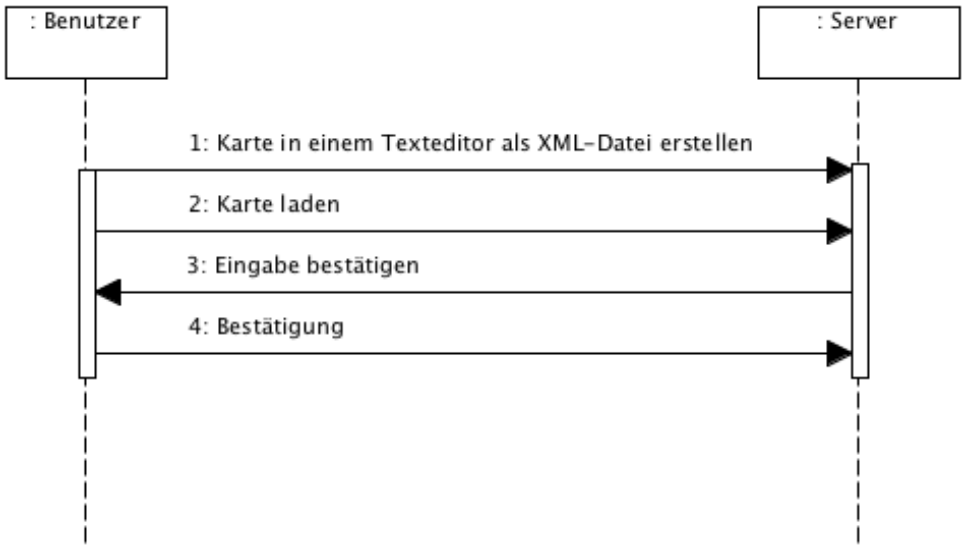


Abbildung 2.3: Sequenzdiagramm zu */F30/*

1. Der Benutzer erstellt eine Karte im XML Format in einem externen Texteditor.
2. Der Benutzer teilt dem Server mit die Karte zu laden (*/F40/*).
3. Der Server benötigt eine Bestätigung für die getätigte Eingabe des Benutzers.
4. Der Benutzer bestätigt seine Eingabe.

## 2.4 Analyse der Funktionalität /F40/: Karte laden

Bei der Funktion *Karte laden* sind der Benutzer und der Server beteiligt. Der Benutzer möchte eine Karte laden und teilt dem Server den Speicherort mit.

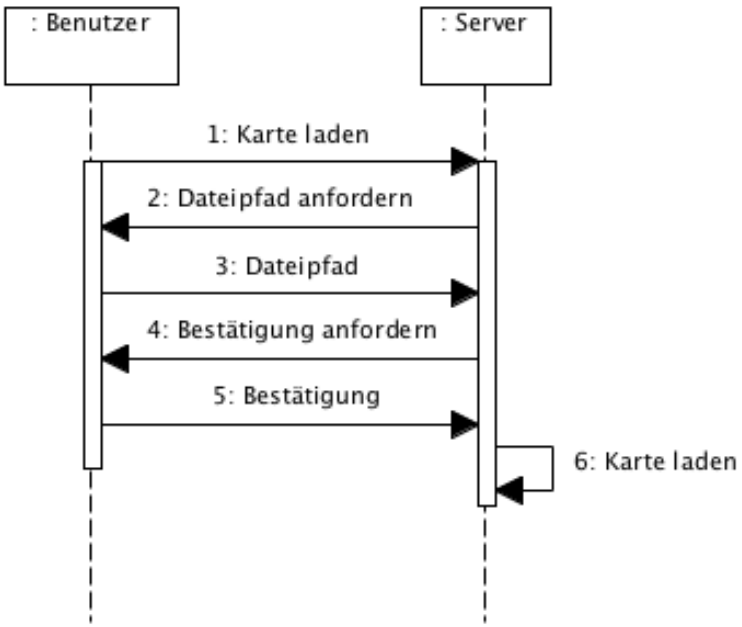


Abbildung 2.4: Sequenzdiagramm zu */F40/*

1. Der Benutzer entscheidet sich, eine Karte zu laden.
2. Der Server fragt den Benutzer nach dem Speicherort der Karte.
3. Der Benutzer teilt dem Server den Speicherort der Karte mit.
4. Der Server benötigt eine Bestätigung für die getätigte Eingabe des Benutzers.
5. Der Benutzer bestätigt seine Eingabe.
6. Der Server lädt die Karte.

## 2.5 Analyse von Funktionalität /F50/: Karte wählen

Aufgrund der hohen Redundanz mit *Karte laden* entfällt *Karte wählen*.

## 2.6 Analyse von Funktionalität /F60/: Spiel starten

Bei der Funktion *Spiel starten* sind der Benutzer und der Server beteiligt. Hat der Benutzer vor das Spiel zu starten, so wählt er zunächst die Spieloptionen aus (*/F70/*) und bestätigt anschließend gegenüber dem Server seine Spieloptionenauswahl sowie das Aufstellen der Roboter und das Spiel startet.

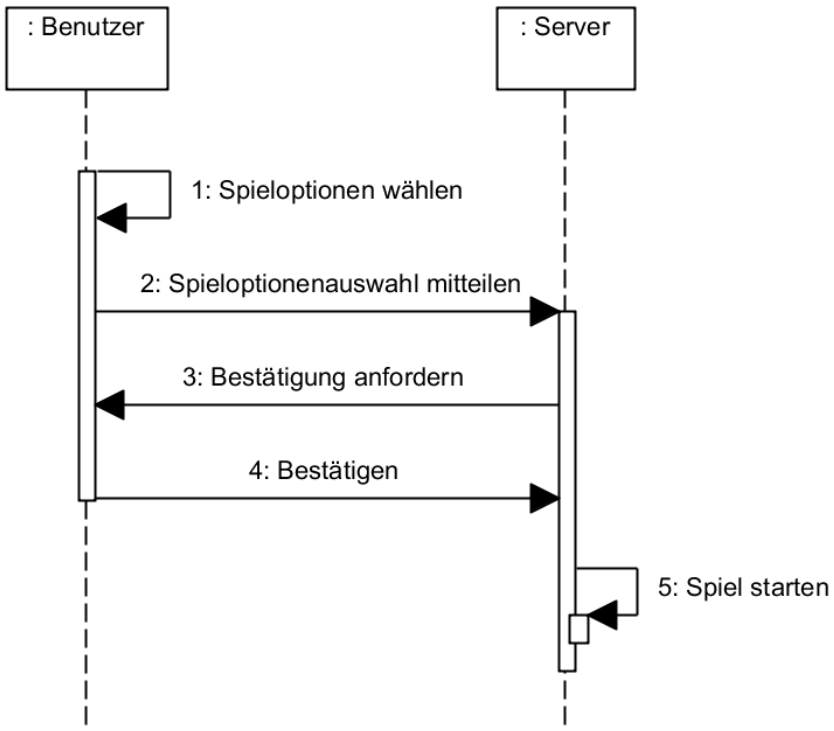


Abbildung 2.5: Sequenzdiagramm zu */F60/*

1. Der Benutzer wählt die Spieloptionen aus (*/F70/*).
2. Der Benutzer teilt die ausgewählten Spieloptionen dem Server mit.
3. Der Server fordert eine Bestätigung vom Benutzer.
4. Der Benutzer bestätigt das Starten des Spieles.
5. Der Server startet das Spiel.

## 2.7 Analyse von Funktionalität /F70/: Spieloptionen wählen

Bei der Funktion *Spieloptionen wählen* sind der Benutzer und der Server beteiligt. Ist der Benutzer dabei ein Spiel zu starten (*/F60/*), so entscheidet er sich zunächst für eine Karte. Dabei kann er entweder eine neue Karte erstellen (*/F30/*) oder eine bereits vorhandene Karte laden. Anschließend kann der Benutzer entweder die Standard-Spieleinstellungen auswählen, d.h der Wumpus bewegt sich nicht, oder er entscheidet sich dafür individuelle Einstellungen zu wählen, die die Bewegung des Wumpus erlauben (*vgl. /M140/ und /W10/*). Hat sich der Benutzer für die gewünschten Spieloptionen entschieden, bestätigt er seine Wahl gegenüber dem Server.

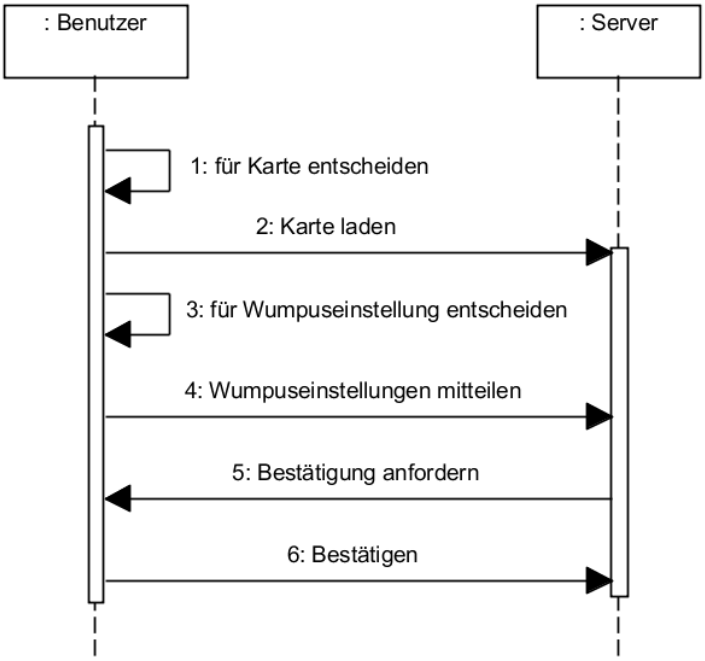


Abbildung 2.6: Sequenzdiagramm zu */F70/*

1. Der Benutzer entscheidet sich für eine Karte und erstellt evtl. eine neue Karte (*/F30/*).
2. Der Benutzer lädt seine Karte.
3. Der Benutzer entscheidet, ob Bewegungen des Wumpus erwünscht sind (*vgl. /M140/ und /W10/*).
4. Der Benutzer teil dem Server die Wumpuseinstellung mit.
5. Der Server fordert eine Bestätigung vom Benutzer.
6. Der Benutzer bestätigt die ausgewählten Spieloptionen.

## 2.8 Analyse von Funktionalität /F80/: Spiel abbrechen

Bei der Funktion *Spiel abbrechen* sind der Benutzer und der Server beteiligt. Möchte der Benutzer das Spiel vorzeitig beenden, hat er die Möglichkeit, das Spiel abzubrechen. Entscheidet sich der Benutzer dafür das Spiel abzubrechen, so wird diese Entscheidung dem Agenten und dem Wumpus mitgeteilt und diese werden abgeschaltet (*/F130/*).

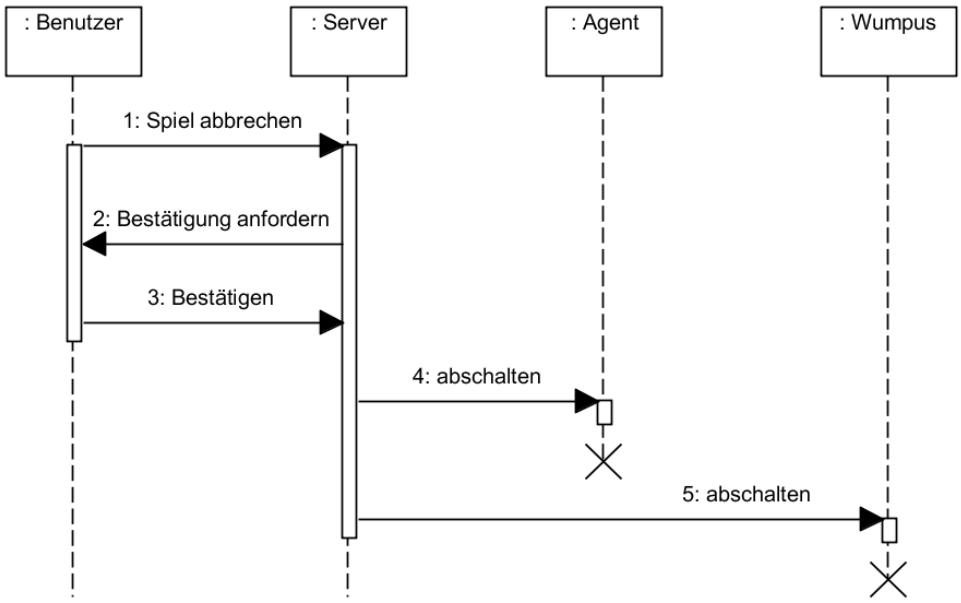


Abbildung 2.7: Sequenzdiagramm zu */F80/*

1. Der Benutzer möchte das Spiel vorzeitig beenden und bricht dieses ab.
2. Der Server fordert eine Bestätigung an.
3. Der Benutzer bestätigt seine Entscheidung gegenüber dem Server.
4. Der Server schaltet den Agenten ab.
5. Der Server schaltet den Wumpus ab.

## 2.9 Analyse von Funktionalität /F90/: Spielinformationen kommunizieren

Bei der Funktion *Spielinformationen kommunizieren* sind der Agent, der Wumpus und der Server beteiligt. Nachdem der Agent die Information über sein aktuelles Spielfeld erhalten hat, kann er über den nächsten Zug entscheiden. Nach der Entscheidung kann der Agent den Zug durchführen und den Server über den erfolgreichen Ablauf dessen benachrichtigen. Der Wumpus bekommt ebenfalls die Information vom Server. Wenn der Wumpus erfährt, dass er getroffen wurde, dann stößt er einen Schrei(*/F140/*) aus. Im anderen Fall tut er nichts. Nach der durchgeführten Aktion benachrichtigt er den Server. Die Kommunikation der Spielinformationen dient dazu, dass der Spielablauf synchron abläuft.

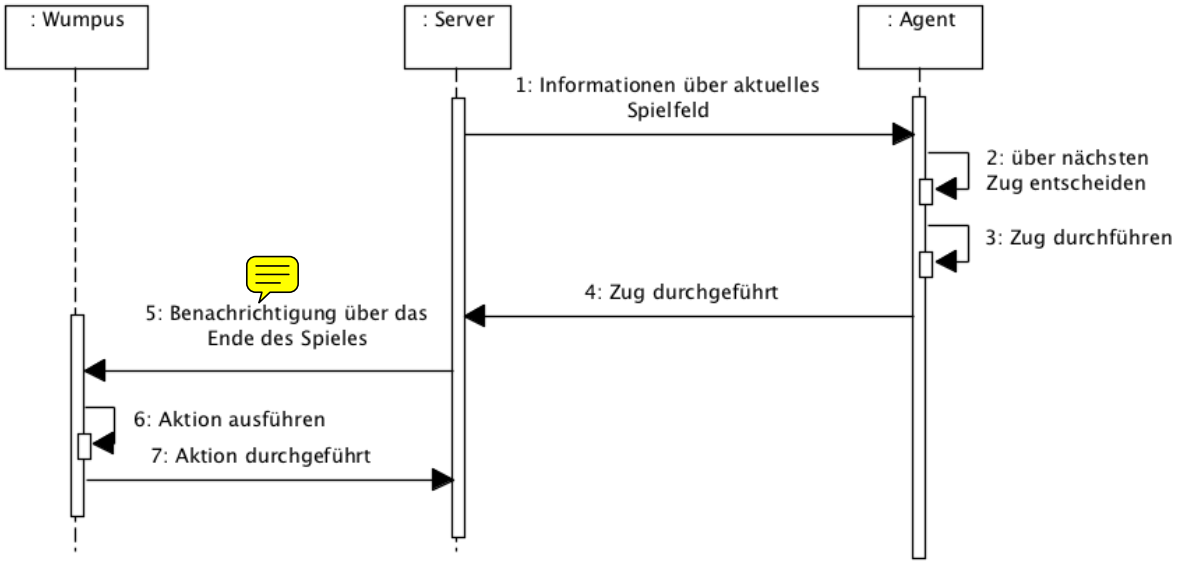


Abbildung 2.8: Sequenzdiagramm zu */F90/*

1. Der Server sendet die Information über das Spielfeld, auf dem der Agent sich befindet.
2. Der Agent entscheidet über den nächsten Zug.
3. Nun führt der Agent seinen Zug durch.
4. Nach dem erfolgreich durchgeführten Zug, teilt dies der Agent dem Server mit.
5. Der Server sendet die Information, ob das Spiel läuft oder zu Ende ist, an den Wumpus.
6. Nun macht der Wumpus seine Aktion, sofern */W10/* umgesetzt wird.
7. Nach der erfolgreich durchgeführter Aktion, teilt dies der Wumpus dem Server mit (*/M140/*).

## 2.10 Analyse von Funktionalität /F100/: Spielverlauf überwachen

Bei der Funktion *Spielverlauf überwachen* sind der Agent und der Server beteiligt. Wenn der Server die Zuginformation von Agenten erhält, dann überwacht er den Spielverlauf. Es werden die aktuelle Punktzahl und der Spielzustand aktualisiert, die sich mit dem Agentenzug verändern haben. Diese Funktionalität kann beliebig oft während des Spiels wiederholt werden.

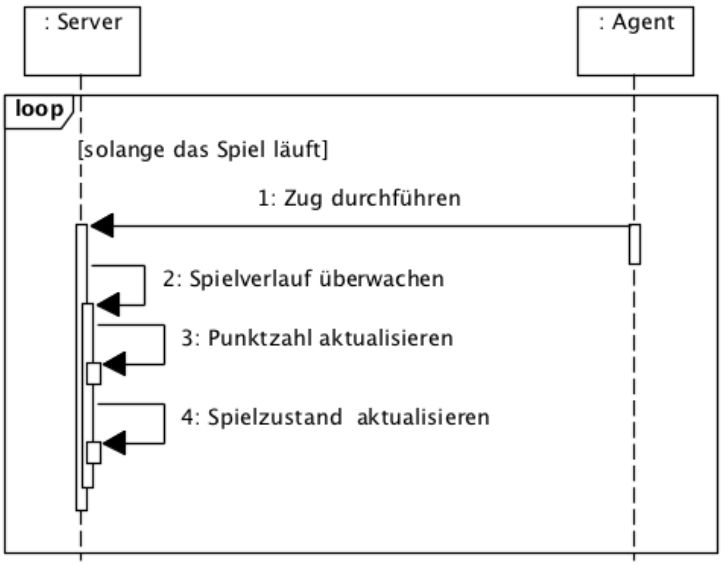


Abbildung 2.9: Sequenzdiagramm zu */F100/*

1. Nach einem erfolgreich durchgeführten Zug teilt der Agent dies dem Server mit.
2. Der Server startet die Funktion Spielverlauf überwachen.
3. Es wird die Punktzahl aktualisiert.
4. Es wird der Spielzustand aktualisiert.

## 2.11 Analyse von Funktionalität /F110/: Spiel beenden

Bei der Funktion *Spiel beenden* sind der Agent, der Wumpus und der Server beteiligt. Wenn der Server die Zuginformation von Agenten erhält, überwacht er den Spielverlauf gemäß (*/F100/*). Dies kann beliebig oft wiederholt werden bis der Server ein Sieg oder eine Niederlage des Agenten feststellt. Das Spielende wird dem Agenten und dem Wumpus mitgeteilt.

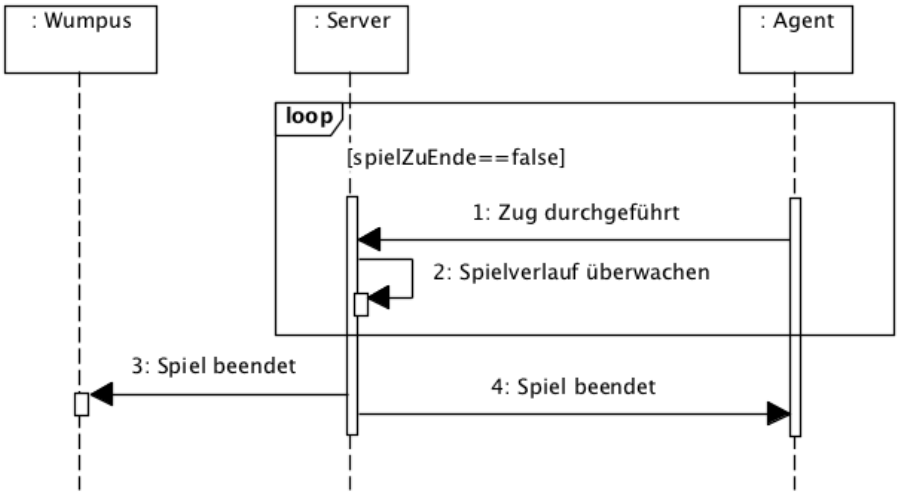


Abbildung 2.10: Sequenzdiagramm zu */F110/*

1. Nach dem erfolgreich durchgeführten Zug, teilt der Agent dies dem Server mit.
2. Der Server erkennt, dass das Spiel zu Ende ist.
3. Der Server teilt das Ende des Spieles dem Agent mit.
4. Der Server benachrichtigt den Wumpus über das Ende des Spieles.

## 2.12 Analyse von Funktionalität /F120/: Punktestand ausgeben

Bei der Funktion *Punktestand ausgeben* ist der Server beteiligt. Nachdem das Spiel beendet wurde (*/F110/*), gibt der Server den Punktestand aus.

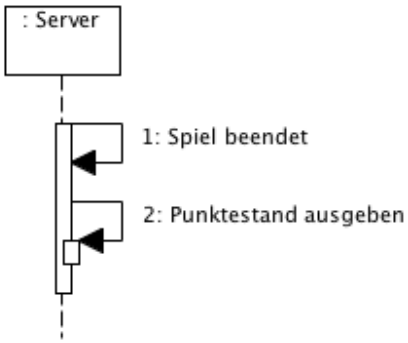


Abbildung 2.11: Sequenzdiagramm zu */F120/*

1. Der Server beendet das Spiel.
2. Der Punktestand der Spielrunde wird ausgegeben.

## 2.13 Analyse von Funktionalität /F130/: abschalten

Bei der Funktion *abschalten* sind der Agent oder der Wumpus und der Server beteiligt. Der Server hat festgestellt, dass das Spiel für einen oder beide Roboter (Agent / Wumpus) zu Ende ist. Bricht der Spieler das Spiel ab oder entkommt der Agent mit dem Gold aus der Höhle, werden beide Roboter (Agent und Wumpus) abgeschaltet. Dies geschiet ebenfalls, wenn der Wumpus den Agenten frisst. Wird der Wumpus jedoch vom Pfeil des Agenten erschossen, wird nur der Wumpus abgeschaltet.

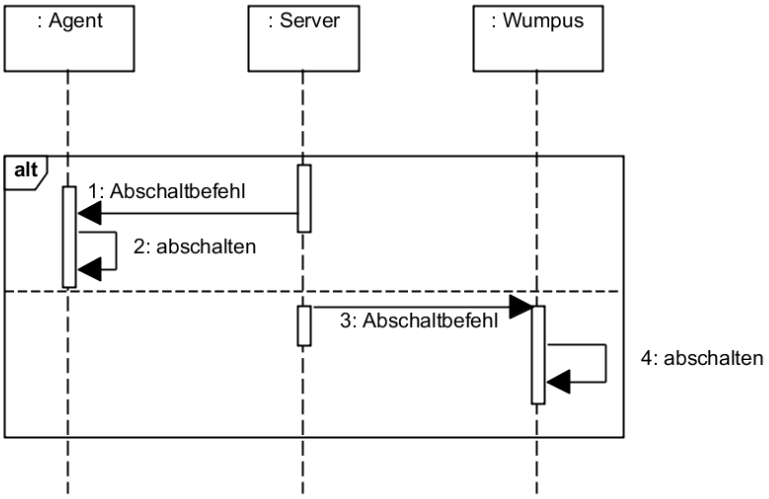


Abbildung 2.12: Sequenzdiagramm zu */F130/*

1. Der Server teilt dem dem Agenten mit, dass er sich abschalten soll.
2. Darauf schaltet sich der Agent ab.
3. Der Server teilt dem dem Wumpus mit, dass er sich abschalten soll.
4. Darauf schaltet sich der Wumpus ab.

## 2.14 Analyse von Funktionalität /F140/: schreien

Bei der Funktion *schreien* sind der Wumpus und der Server beteiligt. Der Wumpus gibt einen Ton (= Schrei) aus. Zuvor hat der Server ihm mitgeteilt, dass er vom Pfeil des Agenten getroffen wurde.

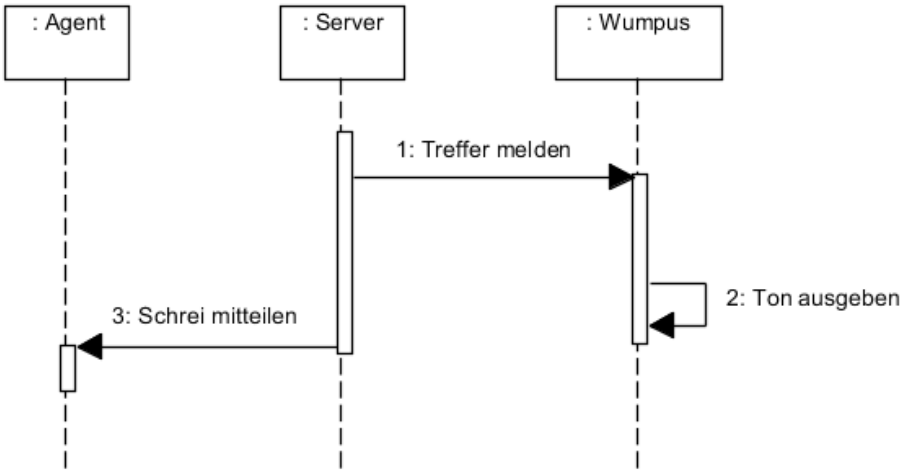


Abbildung 2.13: Sequenzdiagramm zu */F140/*

1. Der Server teilt dem Wumpus mit, dass er vom Pfeil des Agenten getroffen wurde.
2. Daraufhin gibt der Wumpus einen Ton aus.
3. Danach berichtet er dem Agenten vom Schrei des Wumpus

## 2.15 Analyse von Funktionalität /F150/: Pfeil abschießen

Bei der Funktion *Pfeil abschießen* sind der Agent und der Server beteiligt. Der Agent schießt seinen Pfeil ab und teilt dem Server mit in welche Richtung er geschossen hat. Mit Hilfe des Pfeiles ist der Agent so in der Lage den Wumpus zu töten. Dies wird vom Server überprüft. Wurde der Wumpus vom Pfeil des Agenten getroffen, berichtet der Server gemäß */F140/* vom Schrei des Wumpus. Die Benutzung des Pfeiles kostet 10 Punkte. Außerdem wird das Töten des Wumpus mit 10 Punkten belohnt.

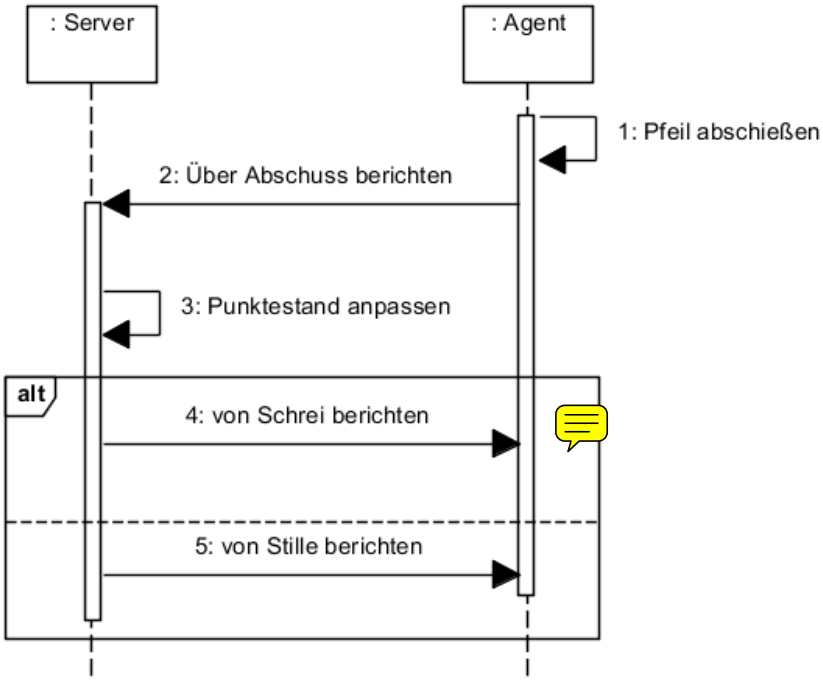


Abbildung 2.14: Sequenzdiagramm zu */F150/*

1. Der Agent schießt seinen Pfeil ab.
2. Danach teilt er dem Server mit in welche Richtung er geschoßen hat.
3. Anschließend wird der Punktestand angepasst (*/F100/*).
4. Zum Schluß berichtet der Server dem Agenten vom Schrei des Wumpus... (*/F140/*).
5. ..oder der anschließenden Stille.

## 2.16 Analyse von Funktionalität /F160/: Gold aufheben

Die Funktion *Gold aufheben* wird von dem Agenten ausgeführt. Der Agent hebt (rein virtuell) das Gold auf. Zuvor hatte er auf dem Feld ein Glitzern wahrgenommen. Das Aufheben des Goldes ist eine vollständige Aktion und dauert einen Zug. Der Spieler erhält hierfür 1000 Punkte.

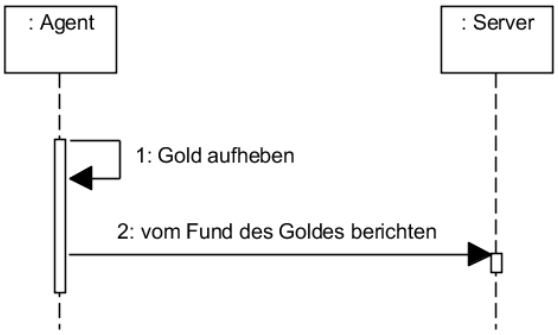


Abbildung 2.15: Sequenzdiagramm zu */F160/*

1. Der Agent hebt das Gold auf.
2. Danach teilt er dies dem Server mit.

## 2.17 Analyse von Funktionalität /F170/: Glitzern erkennen

Bei der Funktion *Glitzern erkennen* sind der Agent und der Server beteiligt. Der Agent hat einen neuen Raum betreten und fragt den Server nach Eigenschaften, die er durch seine Hardware nicht erkennen kann. Da der Agent nicht in der Lage ist, das Glitzern mit seinen Sensoren zu erkennen, teilt ihm der Server dies mit.

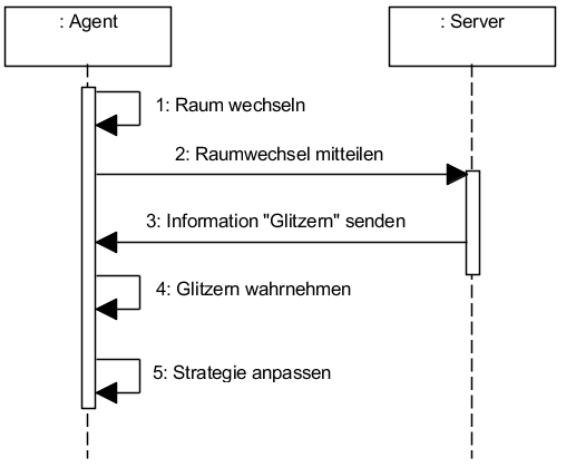


Abbildung 2.16: Sequenzdiagramm zu */F170/*

1. Zunächst wechselt der Agent den Raum.
2. Nun teilt der Agent dem Server den Raumwechsel mit.
3. Der Server sendet dem Agenten die Information, dass es in diesem Raum ein Glitzern gibt.
4. Der Agent nimmt das Glitzern wahr.
5. Der Agent passt seine Strategie an.

## 2.18 Analyse von Funktionalität /F180/: Luftzug erkennen

Bei der Funktion *Luftzug erkennen* sind der Agent und der Server beteiligt. Wenn der Agent den Raum wechselt, teilt er dies dem Server mit und bekommt daraufhin vom Server bestimmte Informationen über diesen Raum. Wenn es in dem Raum einen Luftzug gibt, teilt der Server dem Agenten dies mit. Dadurch nimmt der Agent den Luftzug wahr und kann seine Strategie anpassen. Da es keine entsprechende Sensorik gibt, kann der Luftzug nur virtuell wahrgenommen werden.

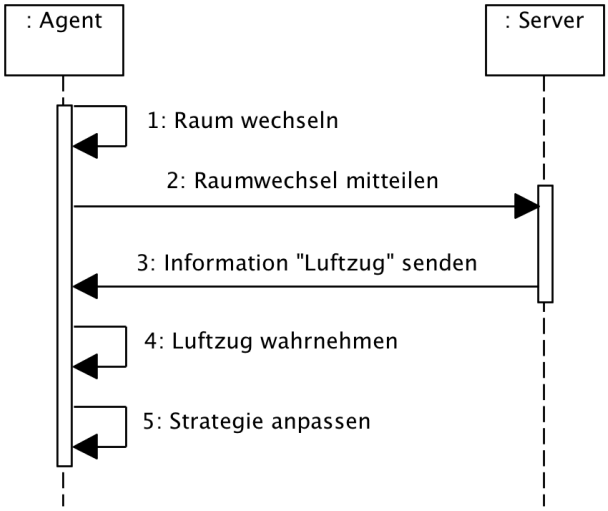


Abbildung 2.17: Sequenzdiagramm zu */F180/*

1. Zunächst wechselt der Agent den Raum.
2. Nun teilt der Agent dem Server den Raumwechsel mit.
3. Der Server sendet dem Agenten die Information, dass es in diesem Raum einen Luftzug gibt.
4. Der Agent nimmt den Luftzug wahr.
5. Der Agent passt seine Strategie an.

## 2.19 Analyse von Funktionalität /F190/: Gestank wahrnehmen

Bei der Funktion *Gestank wahrnehmen* sind der Agent und der Server beteiligt. Wenn der Agent den Raum wechselt, teilt er dies dem Server mit und bekommt daraufhin vom Server bestimmte Informationen über diesen Raum. Wenn es in dem Raum stinkt, teilt der Server dem Agenten dies mit. Dadurch nimmt der Agent den Gestank wahr und kann seine Strategie anpassen. Da es keine entsprechende Sensorik gibt, kann der Gestank nur virtuell wahrgenommen werden.

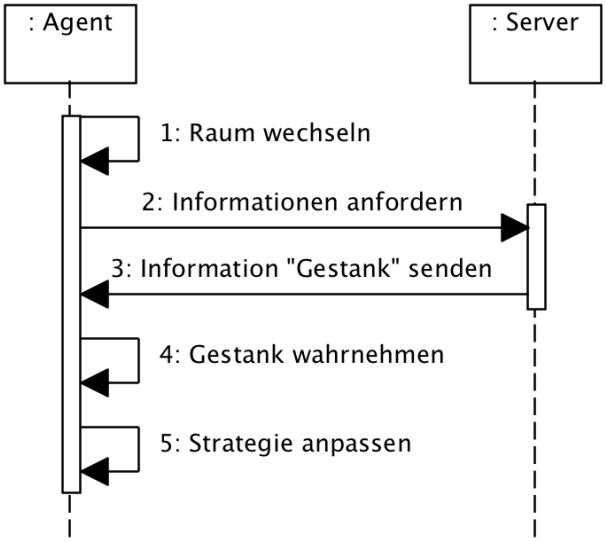


Abbildung 2.18: Sequenzdiagramm zu */F190/*

1. Zunächst wechselt der Agent den Raum.
2. Nun teilt der Agent dem Server den Raumwechsel mit.
3. Der Server sendet dem Agenten die Information, dass es in diesem Raum Gestank gibt.
4. Der Agent nimmt den Gestank wahr.
5. Der Agent passt seine Strategie an.

## 2.20 Analyse von Funktionalität /F200/: korrigieren

Bei der Funktion *korrigieren* ist nur der Agent beteiligt. Wenn der Agent sich während seines Zuges nicht mittig im Raum befindet oder nicht parallel zu den (aus seiner Sicht) seitlichen Wänden steht, erfährt er dies durch seine Sensorik. Wenn er falsch steht, bestimmt er mit Hilfe der Sensorik, welche Korrekturen er beim Fahren vornehmen muss und setzt diese um.

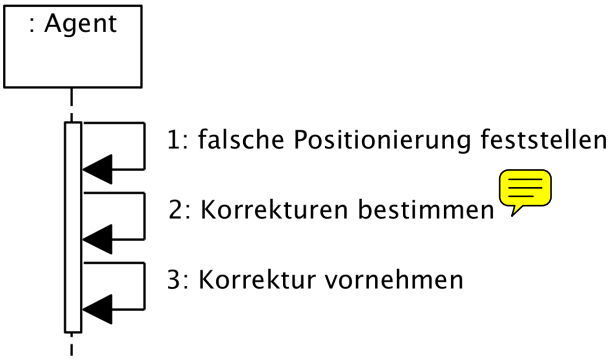


Abbildung 2.19: Sequenzdiagramm zu */F200/*

1. Zunächst stellt der Agent die falsche Positionierung fest. Das bedeutet, dass er sich entweder nicht mittig im Raum befindet oder nicht parallel zu den (aus seiner Sicht) seitlichen Wänden steht.
2. Der Agent bestimmt die nötigen Korrekturen.
3. Der Agent nimmt die Korrekturen vor.

## 2.21 Analyse von Funktionalität /F210/: fressen

Bei der Funktion *fressen* sind Agent, Wumpus und Server beteiligt. Wenn der Agent den Raum des Wumpus betritt, teilt der Server dem Wumpus dies mit. Daraufhin frisst der Wumpus den Agenten. Da der Roboter des Wumpus den Roboter des Agenten nicht wirklich fressen kann, ist dies nur ein virtueller Vorgang.

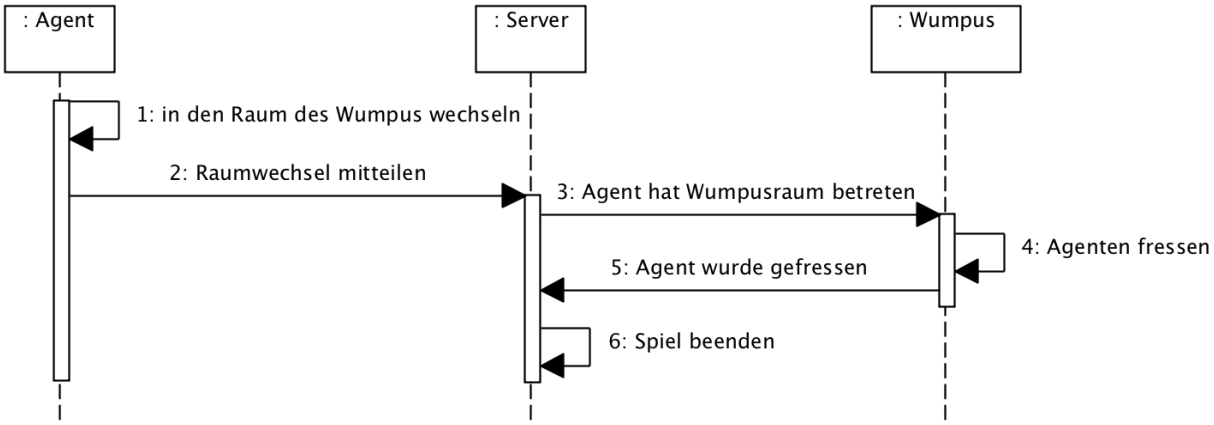


Abbildung 2.20: Sequenzdiagramm zu */F210/*

1. Zunächst wechselt der Agent in den Raum des Wumpus.
2. Danach teilt der Agent dem Server seinen Raumwechsel mit.
3. Der Server teilt dem Wumpus mit, dass der Agent seinen Raum betreten hat.
4. Der Wumpus frisst den Agenten.
5. Der Wumpus teilt dem Server mit, dass er den Agenten gefressen hat.
6. Der Server beendet das Spiel gemäß */F110/*.

# 3 Resultierende Softwarearchitektur

Die Softwarearchitektur der Simulation des Spiels *Hunt the Wumpus* ist sehr wichtig für die Umsetzung der im Pflichtenheft enthaltenden Kriterien. Sie muss zwei wichtige Eigenschaften erfüllen:
Zum einen muss sie ein geeignetes Verteilungsmuster beinhalten und zum anderen müssen die Komponenten leicht austauschbar sein.
Vor allem des letztere bedeutet für die Architektur eine hohe Kohäsion innerhalb der Komponenten und eine geringe Kopplung zwischen ihnen.
Im folgenden wird die geplante Softwarearchitektur für dieses Projekt vorgestellt.

## 3.1 Komponentenspezifikation

### 3.1.1 Client

Die Softwarearchitektur des Clients bzw. des Agenten ist auf zwei Schichten verteilt (siehe Abbildung 3.1) . Die untere Schicht besteht aus den Komponenten, die die Grundfunktionen, wie die Bewegung oder die Kommunikation mit dem Server, des Roboters anbieten. Die obere Schicht koordiniert und steuert den korrekten Ablauf der Aufgaben des Clients und nutzt dabei die Grundfunktionen in der darunter liegenden Schicht. Innerhalb der Komponenten herrscht eine hohe Kohäsion. So befinden sich in der Komponente *Kommunikation* nur Klassen bzw. Module, die für die Kommunikation des Clients mit dem Server nötig sind. Die Komponente *Sensoren/Aktoren* ist für die Bewegung des Roboters zuständig. Sie muss daher Zugriff auf die Aktoren (Motoren) und die Sensorik (für die Korrektur der Position) haben, um den Roboter korrekt durch das Labyrinth zu manövrieren.

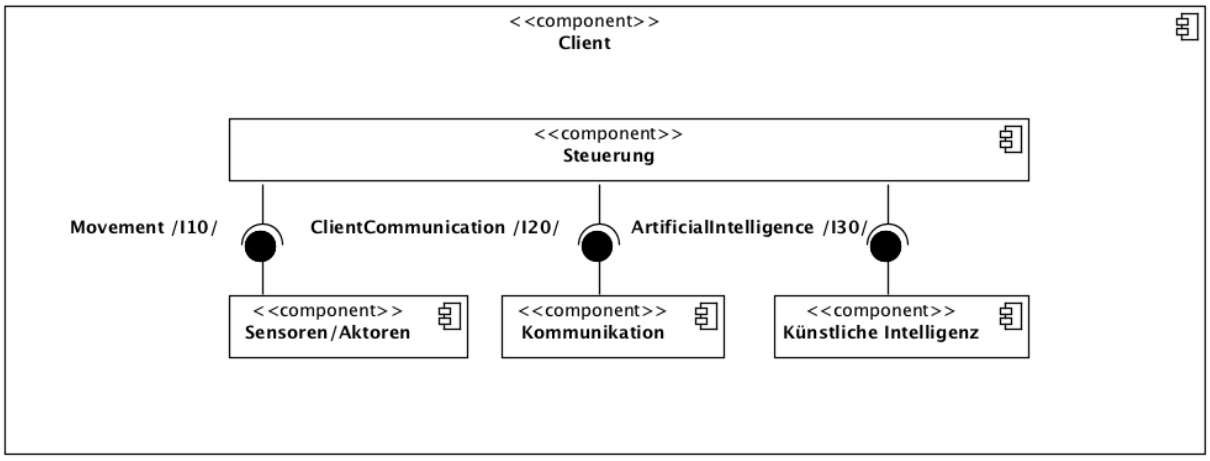


Abbildung 3.1: Komponentendiagramm des Clients (vgl. *Agent*)

Die dritte Grundkomponente (*Künstliche Intelligenz*) ist für die Berechnung des nächsten Zuges zuständig. Sie stellt den Teil mit der größten Komplexität dar, da sie das Hauptziel des Projekts ist.
Da jede dieser Komponenten austauschbar sein soll, müssen geeignete Schnittstelen zu der Komponente *Steuerung* vorhanden sein. Diese sind im Komponentendiagramm an den entsprechenden Punkten gekennzeichnet. Die Bedeutung und Spezifikation der Schnittstellen kann im Kapitel 3.2 nachgelesen werden.

Die Architektur für den Aktor *Wumpus* entspricht dem Aufbau derer des *Agenten*. Jedoch besitzt der *Wumpus* die Komponenten *Künstliche Intelligenz* und *Sensoren/Aktoren* zunächst nicht (vgl /W10/).

### 3.1.2 Server

Wie beim Client ist auch die Softwarearchitektur des Servers auf zwei Schichten verteilt (siehe Abbildung 3.2). Die untere Schicht stellt für die obere Schicht die Grundfunktionen, wie die Kommunikation zum Client oder das Laden der Karte aus einer XML-Datei. Die obere Schicht koordiniert das gesamte Spiel und kümmert sich um die Kommunikation mit den Aktoren.
Auch hier werden wieder geeignete Schnittstellen eingesetzt, um die Komponenten leicht austauschbar zu gestalten. Die Besonderheit ist jedoch die doppelte Schnittstelle zwischen den Komponenten *UserInteraction* und *Spielkoordination*. Nur so ist gewährleistet, dass sowohl die *Spielkoordination* auf die Komponente *UserInteraction* zugreifen kann als auch umgekehrt. Die Bedeutung und Spezifikation der Schnittstellen kann im Kapitel 3.2 nachgelesen werden.

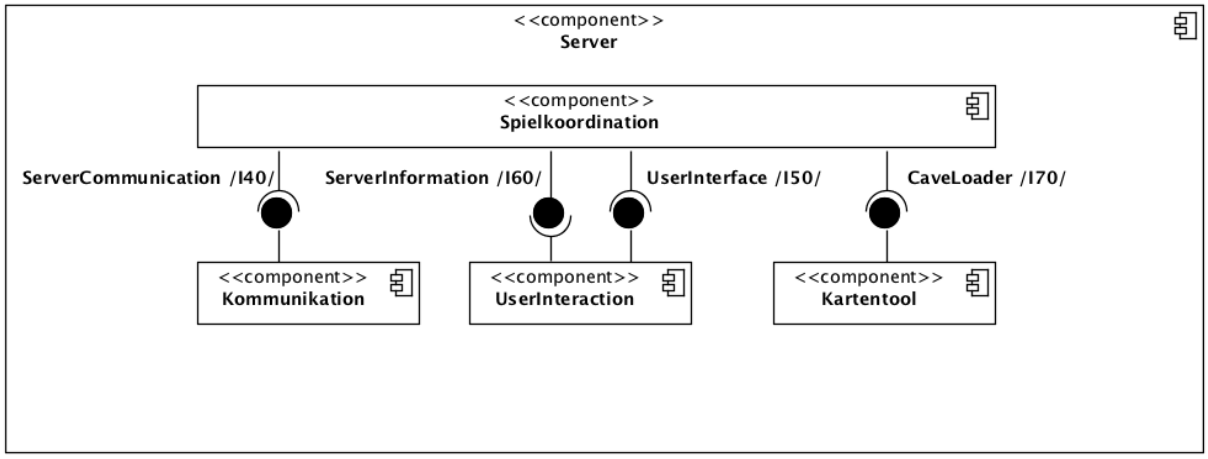


Abbildung 3.2: Komponentendiagramm des Servers

Während des Spiels sind die jeweiligen Kommunikationskomponenten des Servers und des Clients über eine Bluetooth Verbindung miteinander verbunden (siehe Abbildung 3.3).

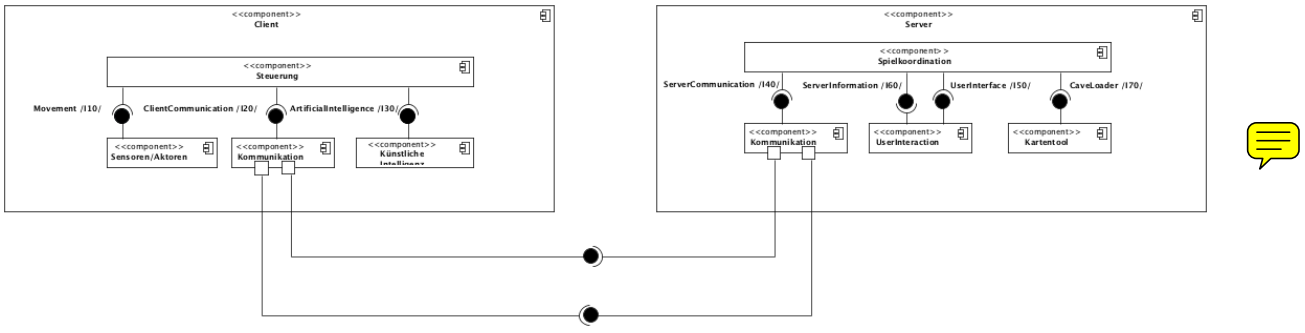


Abbildung 3.3: Komponentendiagramm der Kommunikation zwischen Server und Client

## 3.2 Schnittstellenspezifikation

Im Folgenden werden die einzelnen Schnittstellen aus den Komponentendiagrammen (Abbildungen 3.1 und 3.2) erläutert.

| **Schnitstelle** | **Aufgabenbeschreibung** | |
|---|---|---|
| **/I10/**: Movement | **Operation** | **Beschreibung** |
| | moveForward() | Der Roboter bewegt sich um ein Feld vorwärts |
| | moveLeft() | Der Roboter dreht sich um 90° nach links |
| | moveRight() | Der Roboter dreht sich um 90° nach rechts |
| **/I20/**: ClientCommunication | **Operation** | **Beschreibung** |
| | send() | Daten an den Server senden |
| | receive() | Daten des Servers empfangen |
| | connect() | mit dem Server verbinden |
| | disconnect() | Verbindung zum Server trennen |
| **/I30/**: ArtificalIntelligence | **Operation** | **Beschreibung** |
| | nextTurn() | nächste Aktion des Agenten berechnen |
| **/I40/**: ServerCommunication | **Operation** | **Beschreibung** |
| | send() | Daten an den Roboter senden |
| | receive() | Daten des Roboters empfangen |
| | connect() | mit einem Roboter verbinden |
| | disconnect() | Verbindung zum Roboter trennen |
| **/I50/**: UserInterface | **Operation** | **Beschreibung** |
| | displayScore() | Punktestand ausgeben |
| | display() | Text ausgeben |
| **/I60/**: ServerInformation | **Operation** | **Beschreibung** |
| | inform() | Nachricht an die Spielkoordination senden |
| **/I70/**: CaveLoader | **Operation** | **Beschreibung** |
| | load() | lädt eine Karte aus einer Datei |

## 3.3 Protokolle für die Benutzung der Komponenten

In diesem Abschnitt wird die korrekte Verwendung der Komponenten aus den Diagrammen 3.1 und 3.2 anhand von Statecharts dargestellt.

### 3.3.1 Steuerung

In diesem Teilabschnitt wird das Steuerungsmodul des Clients erläutert.

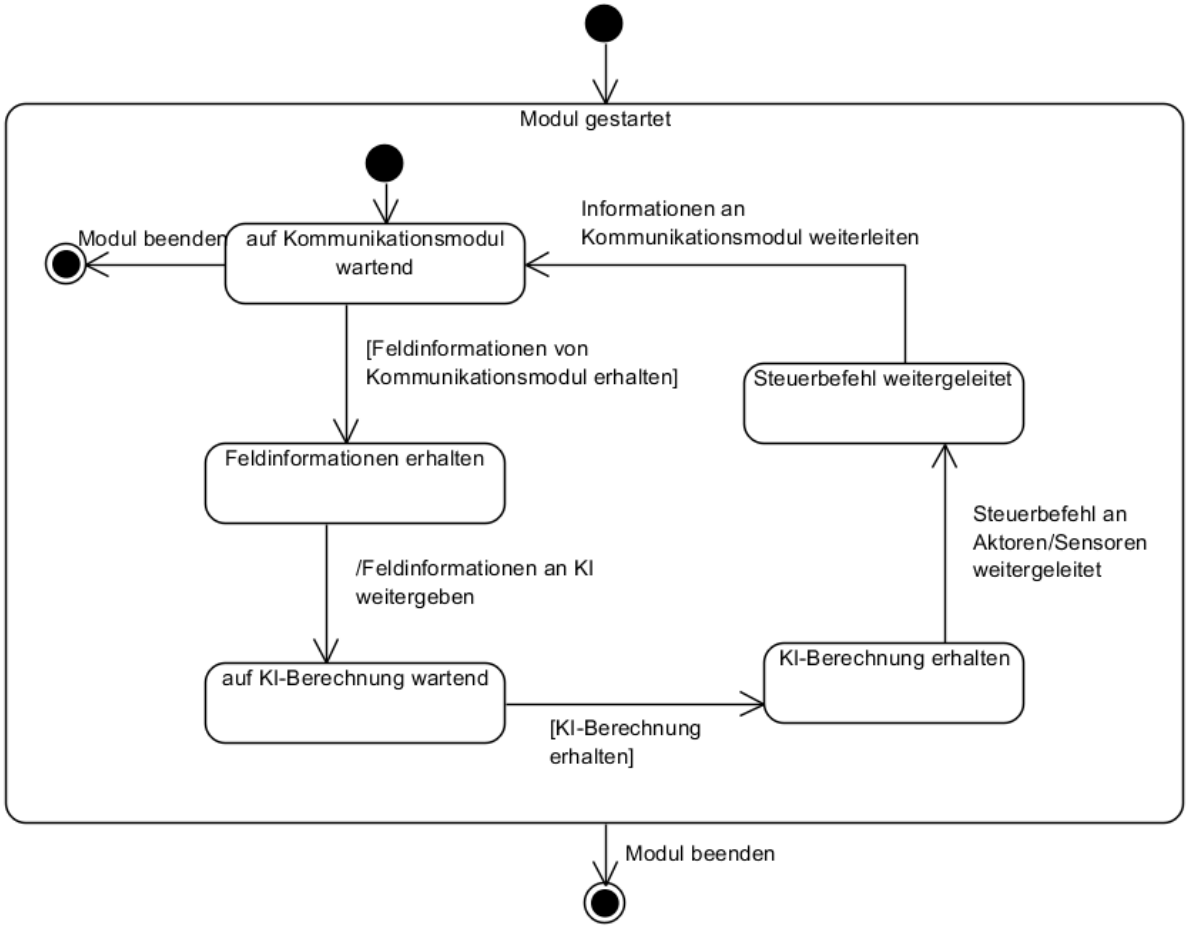


Abbildung 3.4: Statechart zum Modul Steuerung

Wie in Abbildung 3.4 zu sehen, wartet das Steuerungsmodul zunächst auf das *Kommunikationsmodul*, bis dieses eine Verbindung zum Server hergestellt hat und die ersten Feldinformationen empfangen hat. Solange das Steuerungsmodul nicht beendet wird, wird folgender Ablauf wiederholt:

- Es werden die Feldinformationen über das aktuelle Feld des Roboters von dem *Kommunikationsmodul* empfangen.

- Die Feldinformationen werden an die *Künstliche Intelligenz* zur Berechnung des nächsten Zuges weitergeleitet.
- Die Steuerbefehle der *Künstliche Intelligenz* werden empfangen und an das Modul *Sensoren/Aktoren* weitergeschickt.
- Die Informationen des Spielzuges werden an das *Kommunikationsmodul* geschickt und es wird auf neue Informationen gewartet.

### 3.3.2 Sensoren/Aktoren

In diesem Teilabschnitt wird das Sensoren/Aktoren Modul des Clients erläutert.

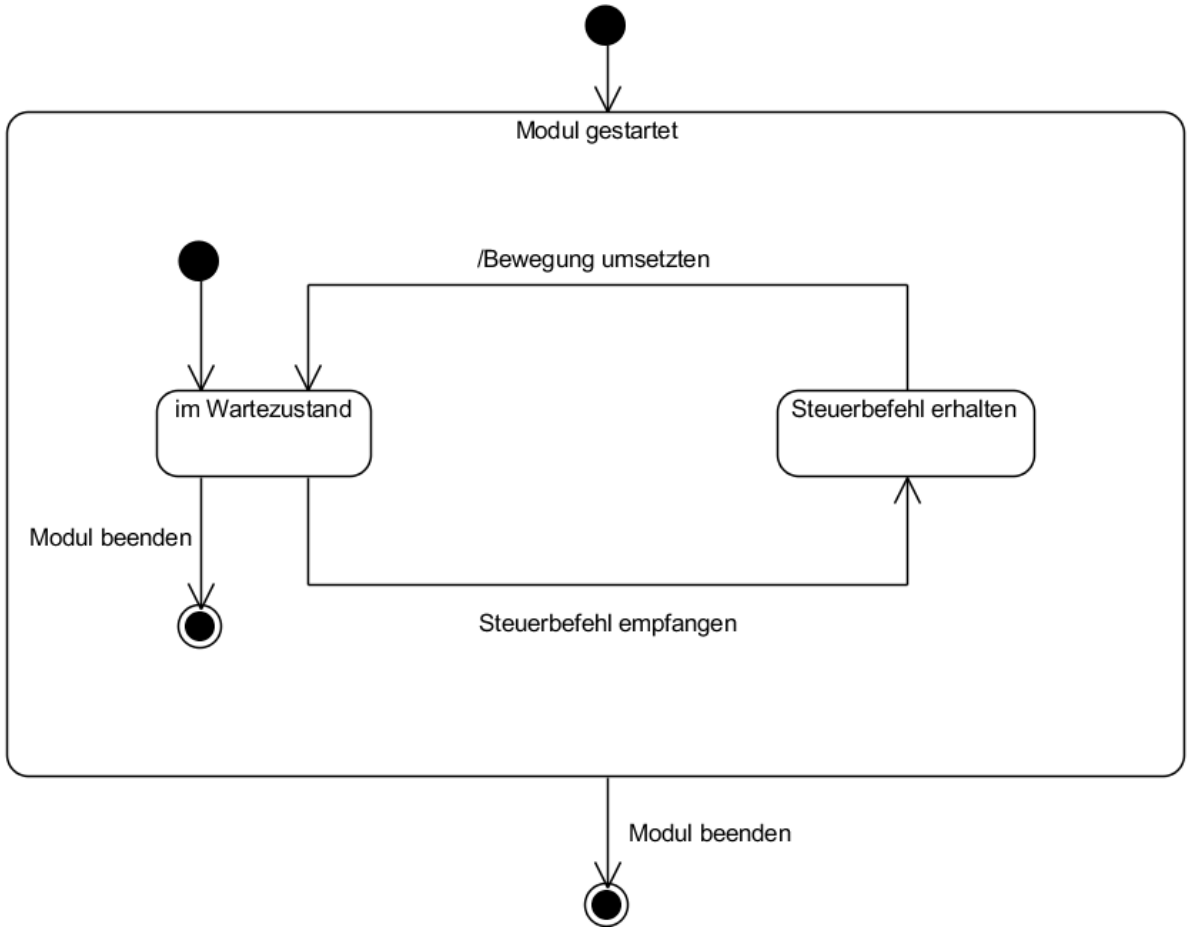


Abbildung 3.5: Statechart zum Modul Sensoren/Aktoren

Wie in Abbildung 3.5 zu sehen ist, befindet sich das Modul nach dem Start zunächst im Wartezustand. Bekommt das Modul von dem *Steuerungsmodul* entsprechende Steuerbefehle *Links-, Rechtsdrehung oder Vorwärtsbewegung* so setzt es diese um und befindet sich danach wieder im Wartezustand. Dieser Vorgang wiederholt sich bis zum Beenden des Moduls.

### 3.3.3 Kommunikation (Client)

In diesem Teilabschnitt wird das Kommunikationsmodul des Clients erläutert.

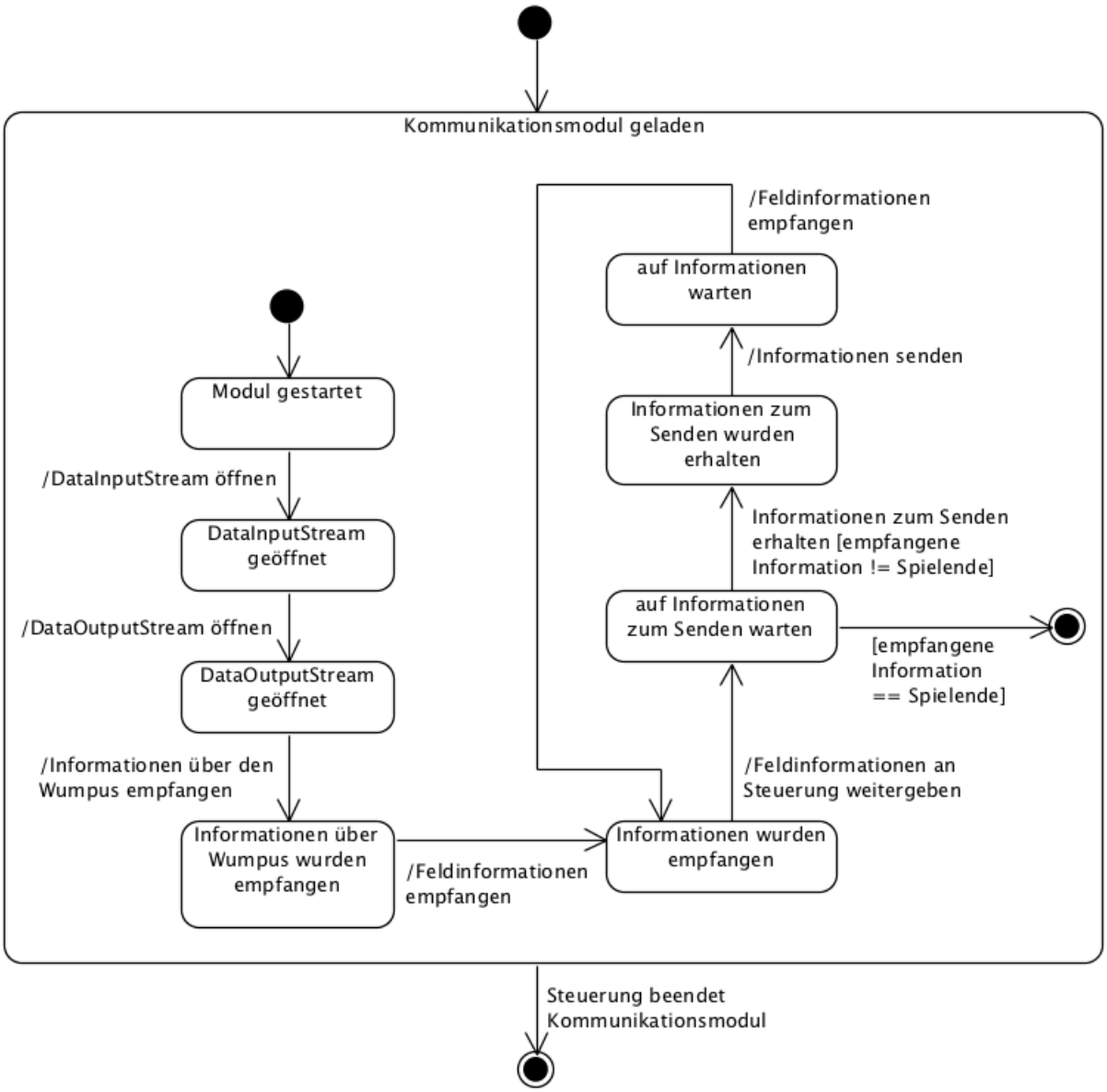


Abbildung 3.6: Statechart des Kommunikationsmoduls des Clients

Wie in Abbildung 3.6 zu sehen ist, wird das Modul Kommunikationsmodul des Clients zunächst geladen und gestartet. Daraufhin werden der DataInput- und der DataOutputStream geöffnet. Jetzt werden Informationen über den Wumpus empfangen (*Darf sich der Wumpus bewegen?*). Solange das Kommunikationsmodul nicht beendet wird, wird nun folgender Ablauf wiederholt:

- Zu Beginn jeden Zuges werden Informationen erhalten. Dies können Informationen über das aktuelle Feld sein oder auch, dass das Spiel zu Ende ist.

- Nachdem die Informationen an die Steuerung (siehe Teilabschnitt 3.3.1) weitergegeben wurden, wartet das Modul auf Informationen zum Senden.
- Wenn die empfangenen Informationen nicht *Spielende* enthalten, werden die erhaltenen Informationen an den Server gesendet. Ansonsten wird das Kommunikationsmodul beendet.
- Nun wartet das Kommunikationsmodul auf die nächsten Feldinformationen.

Die Schleife bricht ab, wenn das Modul die Information erhält, dass das Spielende erreicht ist, oder wenn das Modul von der Steuerung beendet wird.

### 3.3.4 Künstliche Intelligenz

In diesem Teilabschnitt wird das Künstliche Intelligenz Modul des Clients erläutert.

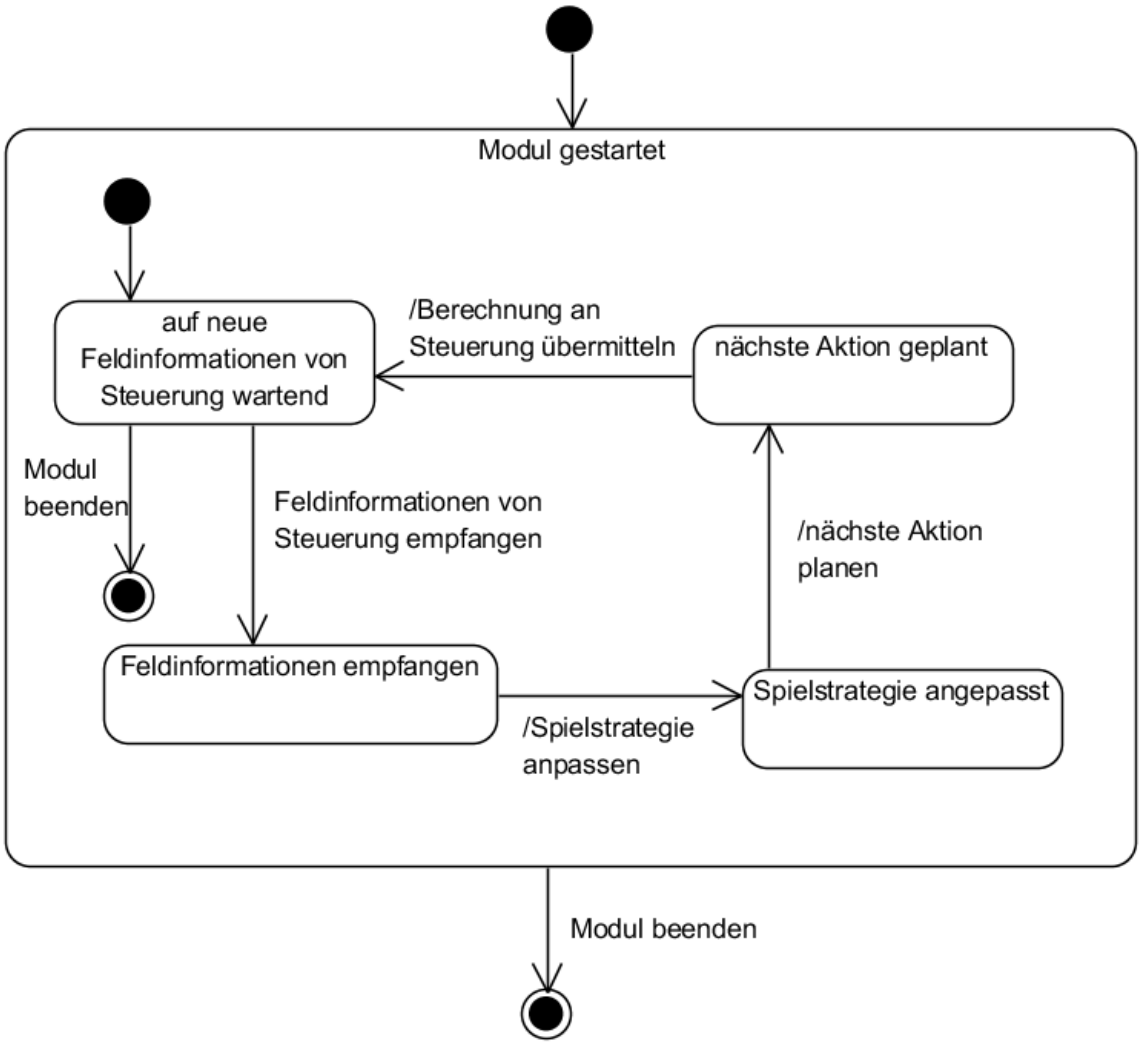


Abbildung 3.7: Statechart zum Modul KünstlicheIntelligenz

Wie in Abbildung 3.7 zu sehen ist, wartet das Modul nach dem Start zunächst auf Feldinformationen des *Steuerungsmoduls*. Erhält das Modul diese Informationen, so passt es seine Strategie an und plant den nächsten Zug des Clients. Die errechneten Steuerbefehle werden dem Steuergerät übermittelt und das Modul befindet sich wieder im Wartezustand. Dieser Vorgang wiederholt sich, bis das Modul beendet wird.

### 3.3.5 Spielkoordination

In diesem Teilabschnitt wird das Modul Spielkoordination des Servers erläutert.

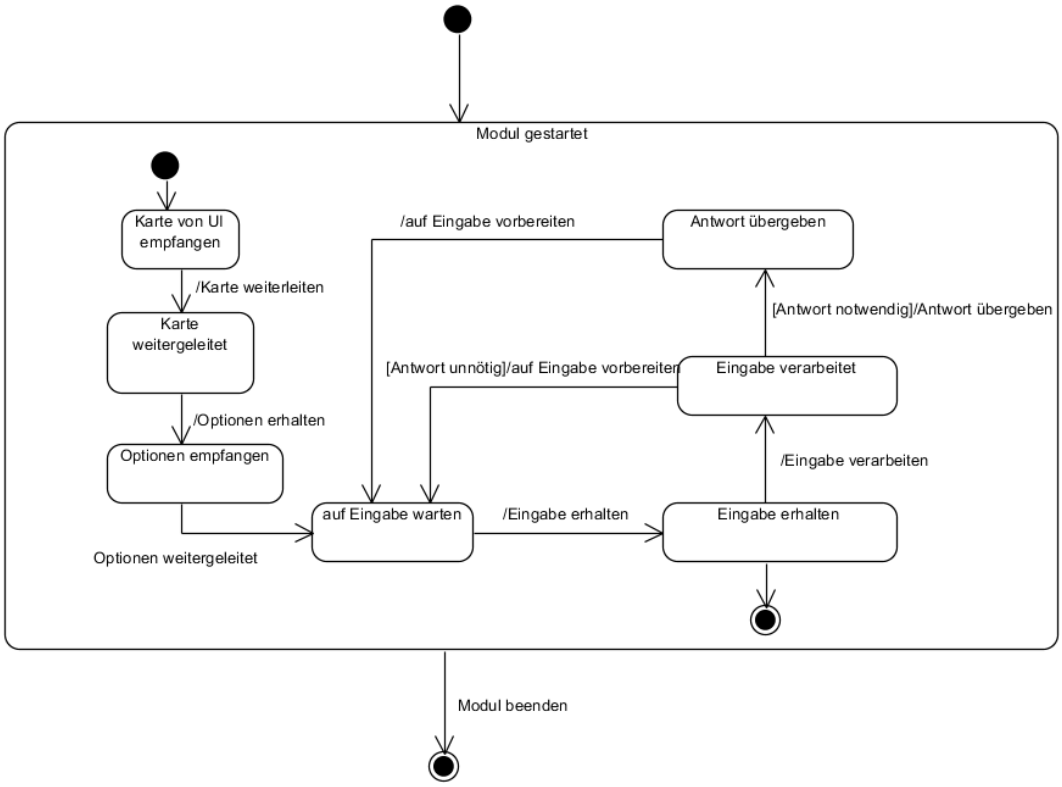


Abbildung 3.8: Statechart zum Modul Spielkoordination

Nachdem das Modul gestartet wurde, erhält es zunächst die Karte vom Modul *User Interface* und leitet diese an das Kartenmodul weiter. Anschließend empfängt es die Spieloptionen und leitet diese an das Kommunikationsmodul weiter. Danach wiederholt das Modul folgende Schritte bis das Modul beendet wird:

- Warten auf eine Nachricht vom Kommunikationsmodul

- Nach Erhalten der Nachricht wird diese verarbeitet. So werden zum Beispiel Feldinformationen vom Kartenmodul abgerufen und weitergeleitet. Außerdem wird, falls nötig, der Punktestand angepasst.
- Falls nötig wird geantwortet. Eine Antwort an das Kommunikationmodul ist nicht immer nötig (z.B. kann die Nachricht auch das Aufheben des Goldes durch den Agenten beinhalten).

### 3.3.6 Kommunikation (Server)

In diesem Teilabschnitt wird das Kommunikationsmodul des Server erläutert.

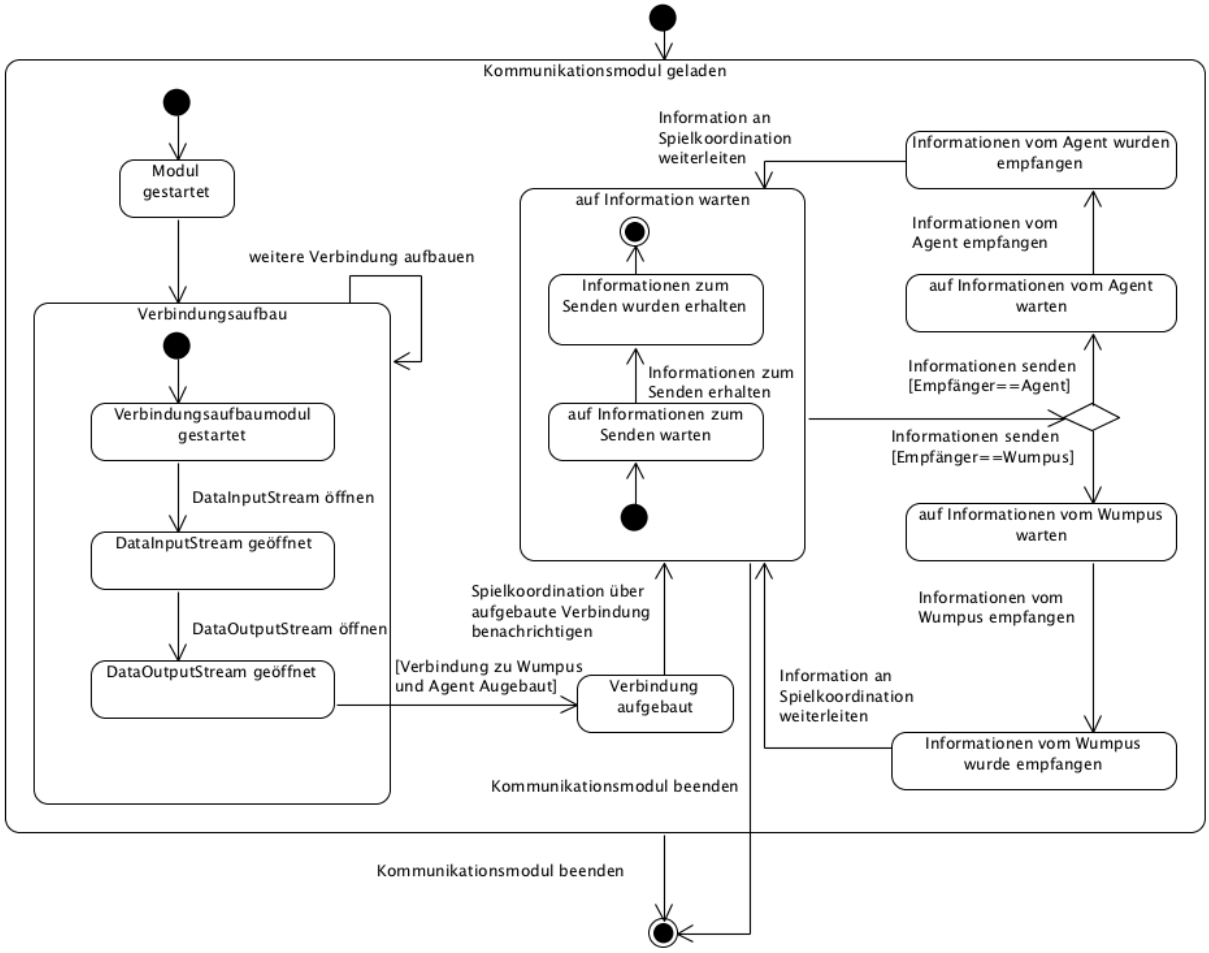


Abbildung 3.9: Statechart-Diagramm des Kommunikationsmodul des Server

Wie in Abbildung 3.9 zu sehen ist, wird das Modul Kommunikationsmodul des Server zunächst geladen und gestartet. Schließlich werden im Unterzustandsdiagramm *Verbindungsaufbau* der DataInput- und der DataOutputStream zum Agent und zum Wumpus geöffnet. Bei erfolgreichem Verbindungsaufbau wechselt man in den Zustand *Verbindung aufgebaut*. Daraufhin wird

die Spielkoordination über die Verbindungsaufbau benachrichtigt. Somit wechselt man in das Unterzustandsdiagramm *auf Information warten*. Solange das Kommunikationsmodul nicht beendet wird, wird nun folgender Ablauf wiederholt:

- Im Unterzustandsdiagramm *auf Information warten* wird gewartet bis die Spielkoordination die Informationen zum Senden bereitstellt. In diesen Informationen muss auch ein Empfänger angegeben werden.
- Nachdem die Information zum Senden bereitgestellt wurde, werden die Informationen zum Empfänger übertragen.
- Es wird auf ein Antwort gewartet.
- Nachdem das Kommunikationsmodul die Informationen empfing, werden diese an die Spielkoordination weitergegeben.

### 3.3.7 User Interface

In diesem Teilabschnitt wird das Modul User Interface erläutert.

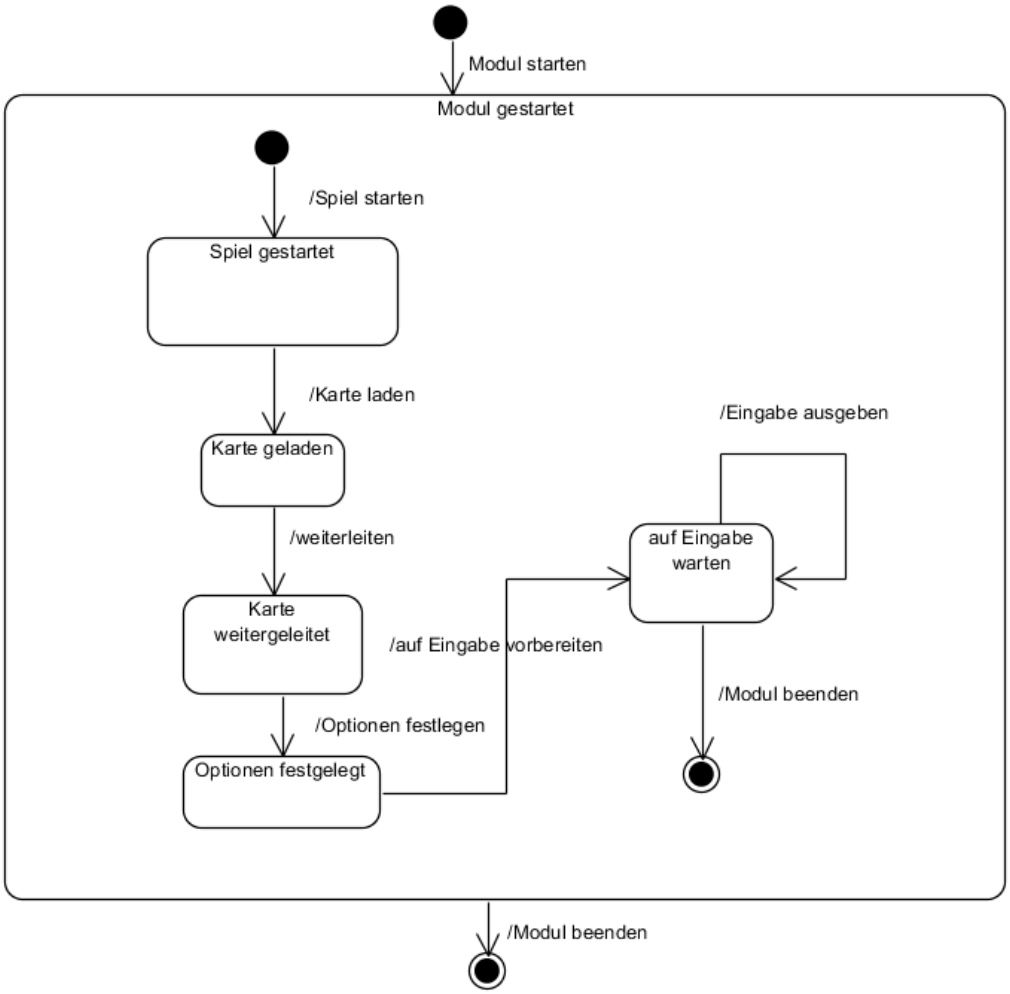


Abbildung 3.10: Statechart zum Modul User Interface

Nach dem Starten des Moduls wird zunächst das Spiel gestartet. Anschließend kann der Spieler eine Karte eingeben (Speicherort angeben). Diese wird dann an das Modul Spielkoordination weitergeleitet. Danach werden noch die Optionen vom Spieler festgelegt und ebenfalls weitergeleitet. Nachdem dies getan ist, wartet dieses Modul den Rest des Spieles auf Eingaben von der Spielkoordination und gibt diese ggf. aus (z.B.: Punktestand).

### 3.3.8 Kartentool

Dieser Teilabschnitt erläutert das Modul Kartentool.

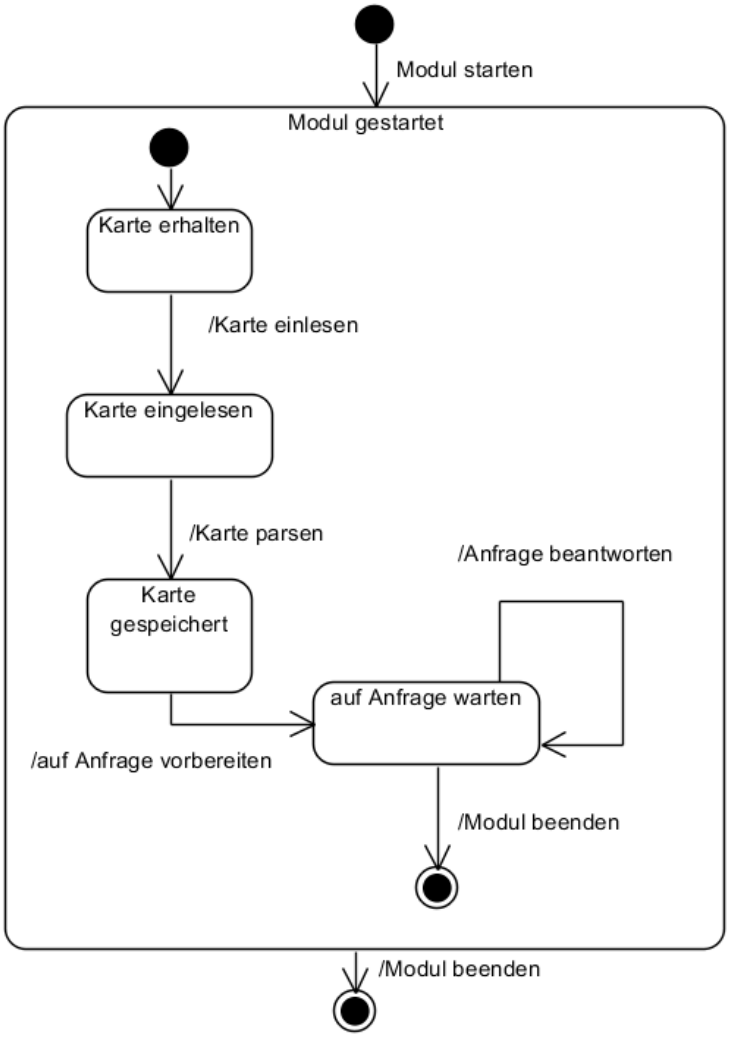


Abbildung 3.11: Statechart zum Modul Kartentool

Nach dem Start des Modules erhält es den Speicherort der Karte als XML vom Modul Spielkoordination. Diese wird dann eingelesen, geparst und intern als Graph gespeichert. Danach wartet das Modul auf Anfragen bezüglich Feldinformationen und beantwortet diese.

# 4 Verteilungsentwurf

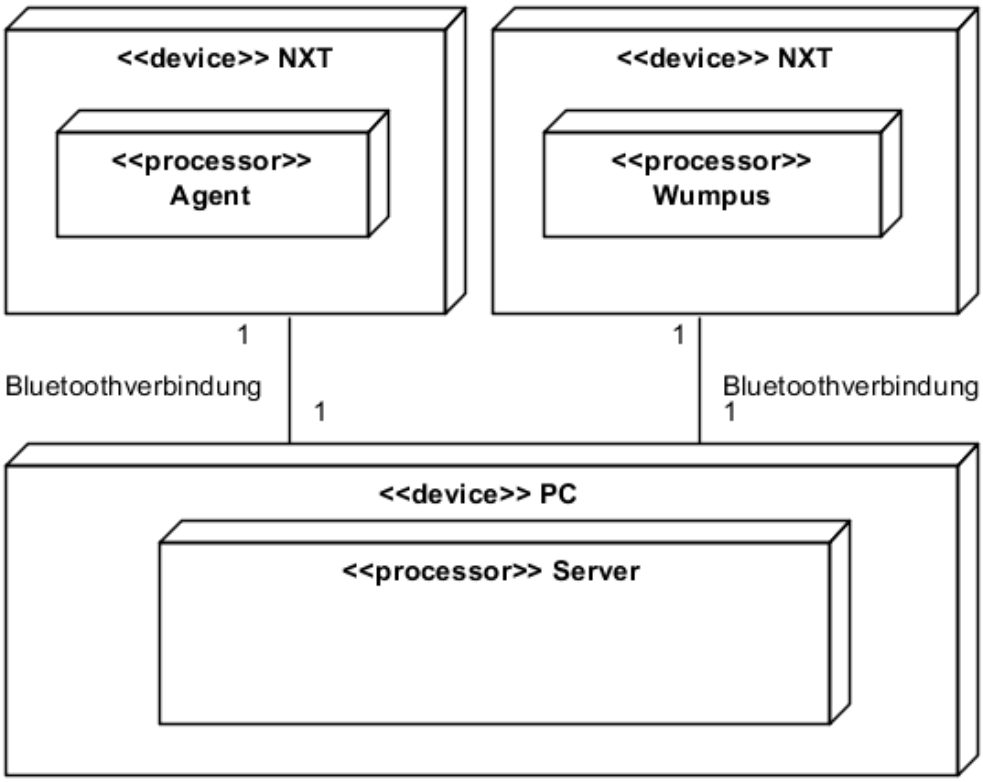


Abbildung 4.1: Verteilungsdiagramm zum Aufbau des Systems

Das System ist als Server/Client Muster angelegt. Der Agent und der Wumpus (je ein NXT Roboter) sind hierbei Clients. Der Verbindungsaufbau erfolgt durch den Server. Dies ist aus technsischen Gründen von Lejos so vorgesehen.