Technische Universität Braunschweig

# NeXT Generation Transport Tycoon

## Team 0

Software-Entwicklungspraktikum (SEP)
Sommersemester 2013

# Testspezifikation

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

Betreuer: Benjamin Mensing

Auftragnehmer:


| Name | E-Mail-Adresse |
|---|---|
| Dennis Stelter | d.stelter@tu-bs.de |
| Henrik Lange | henrik.lange@tu-bs.de |
| Jochen Steiner | jochen.steiner@tu-bs.de |
| Markus-Björn Meißner | m-b.meissner@tu-bs.de |
| Patricia-Tatjana Kasulke | p.kasulke@tu-bs.de |
| Tessa Fabian | tessa.fabian@tu-bs.de |


Braunschweig, 10. Juli 2013

# Versionsübersicht

| Version | Datum | Autor | Status | Kommentar |
|---|---|---|---|---|
| 0.0 | 17.04.2013 | Dennis Stelter, Henrik Lange | i.B. | Testfälle hinzugefügt |
| 0.1 | 19.04.2013 | Jochen Steiner | i.B. | Testumgebung hinzugefügt |
| 0.2 | 19.04.2013 | Tessa Fabian | i.B. | Einleitung und zu testende Komponenten hinzugefügt |
| 0.3 | 21.04.2013 | Sebastian Eilf | i.B. | Zu testende Anforderungen hinzugefügt |
| 0.4 | 21.04.2013 | Markus-Björn Meißner | i.B. | Abnahme Einleitung und Testverfahren hinzugefügt |
| 0.5 | 21.04.2013 | Dennis Stelter, Henrik Lange | i.B. | Testfälle sortiert und überarbeitet |
| 0.6 | 21.04.2013 | Patricia-Tatjana Kasulke | i.B. | Zu testende Funktionen/ Merkmale, nicht zu testende Funktionen und Vorgehen hinzugefügt |
| 0.7 | 21.04.2013 | Dennis Stelter | i.B. | Fehlerkorrektur |
| 0.8 | 21.04.2013 | Henrik Lange | i.B. | LaTeX-Probleme behoben |
| 0.9 | 23.04.2013 | Dennis Stelter, Markus-Björn Meißner, Henrik Lange | i.B. | Testfälle überarbeitet |
| 1.0 | 24.04.2013 | Henrik Lange | abg. | Abgabe ISF |
| 1.1 | 04.07.2013 | Tessa Fabian | i.B. | Testprotokolle zu Komponente C20 |
| 1.2 | 05.07.2013 | Jochen Steiner | i.B. | Testprotokolle zu Komponente C10 und C120 |
| 1.3 | 05.07.2013 | Markus Björn Meißner | i.B. | Testprotokolle zu Komponente C40, C70, C80, C100 |
| 1.4 | 05.07.2013 | Dennis Stelter | i.B. | Testprotokolle zu Komponente C30, C60 und C90 |
| 1.5 | 05.07.2013 | Patricia-Tatjana Kasulke | i.B. | Testprotokolle zu Komponente C50 |
| 1.6 | 06.07.2013 | Henrik Lange | i.B. | Abnahmetest |
| 1.7 | 06.07.2013 | Jochen Steiner | i.B. | Unit-Tests |

| | | | | |
|---|---|---|---|---|
| 1.8 | 06.07.2013 | Dennis Stelter | i.B. | Intergrationstests |
| 1.9 | 10.07.2013 | Henrik Lange | abg. | Abgabe ISF |

**i.B**: in Bearbeitung, **abg.**: abgeschlossen

# Inhaltsverzeichnis

# 1 Einleitung

Das Testen von Software ist ein unerlässlicher Teil des gesamten Entwicklungsprozesses und soll die Herstellung eines guten Produktes, welches die im Pflichtenheft festgehaltenen Kriterien erfüllt, sichern. Daher ist das sorgfältige Planen der Tests sehr wichtig.

Im Rahmen des SEPs 2013 am Institut für Programmierung und Reaktive Systeme wird in diesem Dokument der Test des Softwareprodukts NeXT Generation Transport Tycoon geplant. Das Produkt wird im fertigen Zustand im Wesentlichen aus drei Bestandteilen bestehen: dem Server, dem Spieler-Client und der Software nebst KI, welche auf den NXT-Robotern laufen wird.

Bei einer qualitativ hochwertigen Umsetzung ist es wichtig, dass einige Dinge beachtet werden. NeXT Transport Tycoon soll vor allem stabil laufen, modifizierbar sein und darüber hinaus korrekte Spielergebnisse liefern. Damit der Spieler und der Administrator mit dem Spiel umgehen können, kommt es auf leicht zu bedienende und verständlich für sie zugedachte Oberflächen an. Gerade für die Funktionalität des gesamten Produktes kommt es bei allen Komponenten auf eine gewisse Fehlertoleranz und Zuverlässigkeit an.
Schlussendlich sind Prüfbarkeit und Interoperabilität ebenso wichtige Qualitätsmerkmale, die nicht vergessen werden sollten.

# 2 Testplan

Hier werden die zu testenden Komponenten und Funktionen vorgestellt. Dazu gehört ebenso eine Beschreibung des Abnahme- und Funktionstests und der Testumgebung, in der die Tests stattfinden sollen.

## 2.1 Zu testende Komponenten

Im Wesentlichen gibt es drei zu testende Komponenten: den Server, den Client und die KI der Roboter.

- Der Server: Das Server-Programm wird später auf einem Massenspeicher vorliegen und von dort auf einem Rechner installiert werden. Der verwendete Rechner sollte also über einen entsprechenden Anschluss für den verwendeten Datenträger besitzen. Zum Testen der Funktionen muss der Server an ein TCP/IP-Netzwerk angebunden sein, damit eine Kommunikation zwischen Client und Server möglich ist. Bei Tests, die Roboter und Server betreffen, ist eine funktionierende Bluetooth-Schnittstelle erforderlich.
- Der Client: Die Client-Software wird – wie die des Servers – auf einem Massenspeicher vorliegen und von dort installiert werden. Zum Testen des Clients muss dieser eine Verbindung zum gleichen TCP/IP-Netzwerk verfügen, an das der Server angebunden ist. Über den Client kann ein menschlicher Spieler am Spiel teilnehmen.
- Die KI bzw. der Roboter: Die KI ist das Herzstück der Roboter. Sie entscheidet neben einem menschlichen Spieler über Sieg oder Niederlage. Damit die Roboter und damit die KI getestet werden können, müssen die Akkumulatoren der Roboter vollständig aufgeladen und die Roboter selber angeschaltet sein. Zudem müssen sie via Bluetooth mit dem Server kommunizieren können. Damit eine Teilnahme der Roboter am Spiel möglich ist, wird durch die Bluetoothverbindung das Programm auf die Roboter übertragen und dort anschließend gestartet.

Alle Programme werden als kompilierbarer Java-Quellcode vorliegen. Die Tests der gelisteten Komponenten werden durch das Testen der unten aufgeführten Funktionen/Merkmale durchgeführt.

## 2.2 Zu testende Funktionen/Merkmale

Zunächst werden alle im Pflichtenheft unter Produktfunktionen (Abschnitt 4) aufgeführten Funktionen getestet.

- Bluetooth-Datenverbindung $\langle F10 \rangle$
- Datenverwaltung $\langle F20 \rangle$
- Initialisierung der Roboter $\langle F30 \rangle$
- Straßenerkennung durch Roboter $\langle F40 \rangle$
- Transportvolumen der Roboter $\langle F50 \rangle$
- Treibstoffverbrauch der Roboter $\langle F60 \rangle$
- Streckenfreigabe durch Server $\langle F70 \rangle$
- Routenberechnung durch Roboter $\langle F80 \rangle$
- Spielstart durch Benutzer $\langle F90 \rangle$
- Spieldarstellung der Spieleroberfläche $\langle F100 \rangle$
- Statistikanzeige der Spieleroberfläche $\langle F110 \rangle$
- Auktionen für Transportaufträge $\langle F120 \rangle$
- Vermeidung von Deadlocks $\langle F130 \rangle$
- Spieldarstellung über Benutzeroberfläche $\langle F140 \rangle$

Anschliessend wird überprüft, ob die Musskriterien durch alle Testfälle abgedeckt werden. Die Tests sind des Weiteren so zu wählen, dass die nichtfunktionalen Anforderungen mit diesen einfach überprüft werden können.

## 2.3 Nicht zu testende Funktionen

Keine, da alle Produktfunktionen getestet werden.

## 2.4 Vorgehen

In diesem Abschnitt wird die allgemeine Vorgehensweise für die einzelnen zu testenden Funktionen und Funktionskombinationen beschrieben. Dabei wird auf die Hauptaktivitäten eingegangen.
Zu den wichtigsten Funktionalitäten werden Verfahren genannt, die beim Testen verwendet werden. Dabei wird dokumentiert, welche Aktivitäten, Techniken und Werkzeuge für den Test benötigt werden.
Der Test wird nach den Phasen des V-Modells angewandt. Das heißt, zunächst werden die Komponenten (Module, Programme, Unterprogramme) getestet. Danach wird ein Integrationstest vorgenommen, der die Zusammenarbeit der einzelnen Komponenten testet. Darauf folgt ein Systemtest, bei dem das gesamte System gegen die gesamten Anforderungen (funktionale und nichtfunktionale Anforderungen) getestet wird. Zum Schluss folgt der Abnahmetest, bei dem das Produkt durch den Kunden bzw. Auftraggeber getestet wird. Auf die genannten Phasen des V-Modells wird in den Abschnitten a) bis d) genauer eingegangen.

a) Komponententest
Die in 2.1 genannten Komponenten werden einem Modultest unterzogen - mit dem Ziel, die technische Lauffähigkeit und korrekte fachliche Ergebnisse zu überprüfen. Dabei werden die Komponenten, so weit dies möglich ist, voneinander isoliert betrachtet um Wechselwirkungen mit den anderen Komponenten auszuschließen.
Ein weiterer Testaspekt sind die nichtfunktionalen Anforderungen wie z.B. Richtigkeit, Stabilität und Modifizierbarkeit, die ebenfalls getestet werden müssen. Dabei werden Methoden, Klassen, Funktionen und Module auf funktionaler Ebene getestet.
Verwendet werden sowohl Black- als auch White-Box-Tests. Mögliche Fehler werden noch vor dem Integrationstest behoben.
Der Komponententest wird mit Hilfe von JUnit, soweit dies möglich ist, automatisiert vorgenommen.
Als Techniken kommen die Äquivalenzklassenanalyse/Grenzwertanalyse und Erfüllung der Überdeckungskriterien in Frage.

b) Integrationstest
Es wird das Zusammenspiel einzelner Systemkomponenten überprüft, die vorher dem Modultest unterzogen worden sind. Getestet wird, ob richtige Funktionen und Parameter korrekt verwendet werden, ob die Nachrichtenreihenfolge stimmig ist und ob der gemeinsame Datenzugriff möglich ist.
Dabei wird die Bottom-up-Integration verwendet. Hierbei werden zunächst die Infrastrukturkomponenten, zum Beispiel Zugriff des Roboters auf den Server, integriert. Danach folgenden die anderen Einzelkomponenten.

Der White-Box-Test wird hierbei benutzt, um die Pfadabdeckung zu überprüfen. Auch hier wird JUnit als Testvariante eingesetzt, um die Nutzbarkeit, die Funktionalität sowie Szenarien zu testen.

c) Systemtest
Beim Systemtest wird die Vollständigkeit der funktionalen sowie der nichtfunktionalen Eigenschaften (Benutzbarkeit, Zuverlässigkeit usw.) getestet. Es wird ein realistischer Betrieb der Software simuliert. Es werden Benutzereingaben empfangen und Systemantworten geliefert. Ein sinnvolles Testverfahren ist daher der Black-Box-Test, da nicht die Implementierungsdetails sondern nur die Spezifikation verwendet werden darf.
Hierbei werden auch Performanceanforderungen getestet. Um einen vollständigen Test zu erhalten, werden der Standardablauf sowie die alternativen Abläufe aller Anwendungsfälle überprüft.

d) Abnahmetest
Der Abnahmetest findet vor der Auslieferung des Systems statt und wird in betrieblichen Abnahmetest, Vertragsabnahmetest und regulativen Abnahmetest unterteilt.
Beim Vertragsabnahmetest wird eine Testperson oder der Kunde das fertige Produkt testen. Es werden die Computer sowie der Roboter und das Straßennetz mit der installierten Software zur Verfügung gestellt. Dabei soll getestet werden, ob der Kunde mit dem Produkt zufrieden ist. Dabei schließt man mit ein, dass alle Funktionen voll funktionsfähig sind und die grafische Oberfläche intuitiv und wie erwartet vorzufinden ist. Die Abnahmekriterien erschließen sich aus dem Pflichtenheft. Sind alle Abnahmekriterien erfüllt, gilt der Vertrag als abgeschlossen.
Der betriebliche Abnahmetest stellt die Abnahme des Systems durch den Systemadministrator dar und umfasst z.B. den reibungslosen Ablauf des Programms sowie mögliche Wartungsaufgaben (z.B. Akku des Roboters bei schwachem Ladestand auszuwechseln).
Im regulativen Abnahmetest werden alle Regularien durchgeführt, denen das System entsprechen muss. Das heißt, staatliche, gesetzliche oder Sicherheitsbestimmungen. Da bei diesem Produkt keine von genannten Bestimmungen zutreffen, wird auf diesen Teil des Abnahmetests verzichtet.

## 2.5 Testumgebung

Für die Testumgebung wird ein Raum benötigt, der groß genug ist, damit das Wegenetz auf dem Boden aufgeklebt werden kann. Dies erfüllt der Raum IZ 033B. Der Raum besitzt ein aufgeklebtes Wegenetz, welches mit Klebeband realisiert wurde. Schwarzes Klebeband bildet eine 19 mm breite, tiefschwarze Linie, welche sich auf weißem Klebeband befindet und damit den Kontrast für die Sensoren liefert. Das Wegenetz muss vor Betriebsbeginn auf dem Server gespeichert sein. Außerdem werden drei Mindstorms NXT-Roboter benötigt, die jeweils über zwei Lichtsensoren

verfügen. Es müssen mehr als zwei Roboter sein, um auch die Konfliktfälle testen zu können. Auf den Robotern sollte das Betriebssystem leJOS ab der Version 0.8.5 beta installiert sein. Außerdem wird mindestens ein Computer benötigt, der als Server/Client dient. Dazu muss der Computer ein Bluetooth-Modul enthalten um mit den NXT-Robotern kommunizieren zu können. Idealerweise sollten zum Zeitpunkt des Testes keine weiteren Bluetooth- und/oder Funksysteme in Reichweite von Roboter und Server sein. Auf den Computern sollte ein Betriebssystem (Windows, Linux, etc.) installiert sein, auf dem ein Java Development Kit (JDK) ab Version 1.7 sowie ein funktionierender Bluetooth-Treiber installiert sind. Zusätzlich sollte auf dem Computer eine JUnit Testsuite installiert sein.
Falls Client und Server auf unterschiedlichen Computern laufen, müssen beide mittels einer TCP/IP-Netzwerkverbindung verbunden sein. Um dies zu realisieren, muss ein zweiter Computer vorhanden sein, der die gleichen Voraussetzungen erfüllt wie der erste, mit der Ausnahme, dass er kein Bluetooth-Modul enthalten muss.

# 3 Abnahmetest

Der Abnahmetest ist ein Test des Produkts aus Kundensicht. Für diesen aufgabenorientierten Test stehen besonders die Ergebnisse des Zusammenwirkens der Funktionen des Produkts im Vordergrund.
Dieser Vorgang wird in zwei wesentliche Punkte untergliedert. Zum einen findet eine Überprüfung auf Vollständigkeit der Spezifikationen des Produkts statt und zum anderen ein Test der geforderten Funktionen auf ihre jeweilige Funktionalität.
Eine Zweckmäßigkeit der Anwenderoberfläche sowie eine gute Dokumentation spielen ebenfalls eine Rolle. Als Maßstab für eine korrekte Arbeitsweise dienen dabei die Wünsche und Vorstellungen des Auftraggebers.

## 3.1 Zu testende Anforderungen

Die jeweiligen Anforderungen können als funktionsfähig betrachtet werden, wenn alle zugeordneten Testfälle problemlos durchlaufen.

| **Nr.** | **Anforderung** | **Testfälle** | **Kommentar** |
|---|---|---|---|
| 1 | Ist der genutzte Computer bzw. die genutzte Peripherie funktionsfähig? $\langle F20 \rangle$(erste Hälfte), $\langle F90 \rangle$ | $\langle T100 \rangle$, $\langle T200 \rangle$ | Sofern sich Spiel und Server starten lassen, müssen PC und Peripherie funktionieren. |
| 2 | Funktioniert der Antrieb des Roboters dahingehend korrekt, dass er sich wie gewünscht bewegen kann? $\langle F40 \rangle$, $\langle F80 \rangle$ | $\langle T800 \rangle$ | Wenn der Roboter das berechnete Ziel auch erreicht. |
| 3 | Sind die Bluetooth-Komponenten von Server und Roboter funktionsfähig und können miteinander kommunizieren? $\langle F10 \rangle$, $\langle F20 \rangle$(zweite Hälfte), $\langle F30 \rangle$, $\langle F110 \rangle$ | $\langle T300 \rangle$, $\langle T400 \rangle$, $\langle T1300 \rangle$, $\langle T1400 \rangle$ | |

| | | | |
|---|---|---|---|
| 4 | Setzt der Roboter die serverseitigen Anweisungen korrekt um? $\langle F50 \rangle$, $\langle F70 \rangle$, $\langle F80 \rangle$, $\langle F120 \rangle$ | $\langle T500 \rangle$, $\langle T800 \rangle$, $\langle T900 \rangle$, $\langle T1000 \rangle$ | |
| 5 | Arbeitet die KI der Roboter korrekt? Können die Roboter autonom von dieser KI gesteuert werden? $\langle F80 \rangle$, $\langle F120 \rangle$ | $\langle T500 \rangle$, $\langle T800 \rangle$ | Beide Testfälle müssen in Kombination und ohne menschliche Interaktion durchlaufen. |
| 6 | Arbeitet das Modul zur optischen Linienerkennung am Roboter korrekt? Kann der Roboter dem Straßennetz problemlos folgen? $\langle F40 \rangle$ | $\langle T600 \rangle$ | |
| 7 | Wird das Wegenetz vom Roboter korrekt erkannt und benutzt? Kann er eigene Routen finden? $\langle F80 \rangle$ | $\langle T800 \rangle$ | |
| 8 | Kann der Server das Wegenetz korrekt verarbeiten und benutzte Strecken sperren? $\langle F70 \rangle$ | $\langle T900 \rangle$ | |
| 9 | Besteht eine Verbindung zwischen Server und Client? $\langle F100 \rangle$ | $\langle T1100 \rangle$ | |
| 10 | Besitzt der Roboter Algorithmen, die dem Entstehen von Deadlocks* entgegenwirken? $\langle F130 \rangle$ | $\langle T1200 \rangle$ | |
| 11 | Sind die angegebenen Statistiken korrekt und logisch? $\langle F20 \rangle$(zweite Hälfte), $\langle F110 \rangle$ | $\langle T1300 \rangle$, $\langle T1400 \rangle$ | Die Korrektheit der angezeigten Daten muss manuell überprüft werden. |
| 12 | Werden Aufträge korrekt versteigert und können Roboter diese unter Beachtung der Auktionsregeln erhalten? $\langle F120 \rangle$ | $\langle T500 \rangle$ | Die Auftragsverwaltung muss korrekt ablaufen, weil viele Funktionen über Aufträge getestet werden. |
| 13 | Beachten Roboter ihren Tankstand bzw. scheiden sie aus, wenn ihr Tank leer ist? $\langle F60 \rangle$ | $\langle T700 \rangle$, $\langle T1600 \rangle$ | |
| 14 | Entspricht die Visualisierung auf der Benutzeroberfläche den zugrunde liegenden Statistiken? $\langle F140 \rangle$ | $\langle T1500 \rangle$ | |
| 15 | Werden Waren von den Industrien gemäß des Produktionszyklus weiterverarbeitet? $\langle F20 \rangle$ | $\langle T1700 \rangle$ | |

| | | | |
|---|---|---|---|
| 16 | Lassen sich die Roboter über die Spieleroberfläche bei ihrer Routenwahl beeinflussen? $\langle F80 \rangle$ | $\langle T1800 \rangle$ | |

Tabelle 3.1: Zu testende Anforderungen

## 3.2 Testverfahren

Für den Abnahmetest kommt als dynamischer Funktionstest das Black-Box Verfahren zum Einsatz. In möglichst praxisnaher Umgebung wird hier nur das Zusammenspiel der einzelnen Systemkomponenten des Produkts getestet. Auf eine bestimmte Eingabe über die von außen sichtbaren Schnittstellen (Eingabemasken) oder einen bestimmten über die Anwenderoberfläche gesteuerten Programmablauf muss das Produkt in entsprechender Weise reagieren und korrekte Zwischenergebnisse sowie Ausgaben erzielen. Das beobachtete Verhalten wird in zahlreichen Testfällen mit dem über die funktionalen Anforderungen festgelegten Verhalten verglichen. Testfälle bezüglich möglicher Dateneingaben der Anwender können beispielsweise mit Kombinationen von Grenzwerten durchgeführt und anhand des zu erwartetenden Ergebnisses auf ihre Korrektheit überprüft werden.
Es werden dabei keine Testskripte verwendet.

## 3.3 Testfälle

Im folgendem werden alle Testfälle beschrieben.

### Testfall $\langle T100 \rangle$ - Initialisierung des Servers

**Ziel**

Überprüfung der Funktionsfähigkeit vom Computer bzw. der genutzten Peripherie (Tabelle 3.1 Nr. 1).

**Objekte/Methoden/Funktionen**

**Objekte**: Server

**Funktionen**: Datenverwaltung $\langle F20 \rangle$ (erste Hälfte)

**Pass/Fail Kriterien**

Der Test ist erfolgreich, wenn die Einstellungen übernommen wurden.

**Vorbedingung**

Keine.

**Einzelschritte**

Eingabe:

1. Gewünschte Spieleinstellung über die Oberfläche eingeben.
2. Spiel starten.

Ausgabe:

1. Oberfläche enthält alle eingestellten Einstellungen.

**Beobachtungen / Log / Umgebung**

Betrachten der Benutzeroberfläche. Hier ist zu prüfen, ob die gewählten Einstellungen auch korrekt übertragen wurden.

**Abhängigkeiten**

Keine.

### Testfall $\langle T200 \rangle$ - Spielstart durch Benutzer

#### Ziel

Überprüfung der Funktionsfähigkeit vom Computer bzw. der genutzten Peripherie (Tabelle 3.1 Nr. 1).

#### Objekte/Methoden/Funktionen

**Objekte**: Server

**Funktionen**: Spielstart durch Benutzer$\langle F90 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn das Spiel ohne Fehler gestartet wird.

#### Vorbedingung

$\langle T100 \rangle$

#### Einzelschritte

Eingabe:

1. Programm starten (passiert durch $\langle T100 \rangle$).
2. Spiel starten.

Ausgabe:

1. Benutzeroberfläche oder Konsole wird angezeigt (je nach Einstellung).

#### Beobachtungen / Log / Umgebung

Beobachtung des Computers, auf dem das Spiel gestartet wurde. Bei korrekter Ausführung sollte die Benutzeroberfläche angezeigt werden (alternativ auch die Konsole).

#### Abhängigkeiten

Abhängig von $\langle T100 \rangle$ , da ohne Spieldaten kein Spiel gestartet werden kann.

### Testfall $\langle T300 \rangle$ - Datenübertragung zwischen Roboter und Server

**Ziel**

Überprüfung, ob die Bluetooth-Komponenten von Server und Roboter funktionsfähig sind und miteinander kommunizieren können (Tabelle 3.1 Nr. 3).

**Objekte/Methoden/Funktionen**

**Objekte**: Ein Roboter, Server

**Funktionen**: Bluetooth-Datenverbindung $\langle F10 \rangle$, Initialisierung der Roboter $\langle F30 \rangle$

**Pass/Fail Kriterien**

Der Test ist erfolgreich, wenn der Roboter auf dem Display anzeigt, dass er mit dem Server in Verbindung steht.

**Vorbedingung**

Das Programm ist in Betrieb. Darüber hinaus ist der Roboter eingeschaltet und bereit zum Verbinden.

**Einzelschritte**

Ausgabe:

1. Der Roboter zeigt an, dass er mit dem Server in Verbindung steht.

**Beobachtungen / Log / Umgebung**

Beobachtung des Displays auf dem Roboter bezüglich erwarteter Ausgabe.

**Abhängigkeiten**

Abhängig von $\langle T200 \rangle$, da das Programm gestartet sein muss, damit Roboter und Server kommunizieren.

### Testfall $\langle T400 \rangle$ - Server initialisiert Roboter

**Ziel**

Überprüfung, ob die Bluetooth-Komponenten von Server und Roboter funktionsfähig sind und miteinander kommunizieren können (Tabelle 3.1 Nr. 3).

**Objekte/Methoden/Funktionen**

**Objekte**: Server, ein Roboter

**Funktionen**: Initialisierung der Roboter $\langle F30 \rangle$

**Pass/Fail Kriterien**

Der Test ist erfolgreich, wenn der Roboter auf dem Display „ready“ anzeigt.

**Vorbedingung**

Roboter ist eingeschaltet und bereit zum Verbinden. Das Programm ist gestartet.

**Einzelschritte**

Eingabe:

1. Spiel starten ($\langle F90 \rangle$).

Ausgabe:

1. Die Anzeige auf dem Roboter wird mit der Ausgabe „ready“ aktualisiert.

**Beobachtungen / Log / Umgebung**

Display des Roboters bezüglich erwarteter Ausgabe beobachten.

**Abhängigkeiten**

Abhängig von $\langle T300 \rangle$, da zur Initialisierung des Roboters eine Bluetooth-Verbindung bestehen muss.

### Testfall $\langle T500 \rangle$ - Auktionen für Transportaufträge

#### Ziel

Überprüfung, ob der Roboter die serverseitigen Anweisungen korrekt umsetzt (Tabelle 3.1 Nr. 4). Außerdem, ob die KI der Roboter korrekt arbeitet und die Roboter von dieser KI autonom gesteuert werden (Tabelle 3.1 Nr. 5).

#### Objekte/Methoden/Funktionen

**Objekte**: Zwei Roboter, Server

**Funktionen**: Auktionen für Transportaufträge $\langle F120 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Roboter auf Aufträge bieten und der Roboter mit dem besseren Angebot den Auftrag erhält.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$).

#### Einzelschritte

Eingabe:

1. Mehrere Auftragsauktionen erstellen.
2. Mehrere Auftragsauktionen senden.

#### Beobachtungen / Log / Umgebung

Beobachtung der Roboter über die Konsolen-Ausgabe des Servers.

#### Abhängigkeiten

Die Roboter müssen initialisert worden sein $\langle T400 \rangle$, damit sie am Spiel und somit an Auktionen teilnehmen können.

### Testfall $\langle T600 \rangle$ - Straßenerkennung durch Roboter

#### Ziel

Überprüfung, ob das Modul zur optischen Linienerkennung am Roboter korrekt arbeitet und dem Wegenetz korrekt gefolgt wird (Tabelle 3.1 Nr. 6).

#### Objekte/Methoden/Funktionen

**Objekte**: Ein Roboter, Server

**Funktionen**: Straßenerkennung durch Roboter $\langle F40 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn der Roboter dem Wegenetz folgt und davon nicht abweicht. Darüber hinaus erkennt er Kreuzungen und Straßenenden.

#### Vorbedingung

Das Wegenetz ist vorhanden und das Spiel läuft ($\langle F90 \rangle$). Darüber hinaus muss der Roboter mit einer FCFS-KI* zur Abarbeitung der Aufträge initialisiert worden sein, damit sichergestellt wird, dass er den Auftrag auf jeden Fall annimmt und beendet.

#### Einzelschritte

Eingabe:

1. Einen Auftrag erstellen.
2. Diesen Auftrag senden.

#### Beobachtungen / Log / Umgebung

Beobachtung des Roboters bei seiner Fahrt auf dem Wegenetz. Es ist darauf zu achten, ob er dem Wegenetz folgt.

#### Abhängigkeiten

Testfall ist abhängig von $\langle T500 \rangle$, da der Roboter ohne einen Auftrag keinen Grund hat, eine Bewegung durchzuführen.

### Testfall $\langle T700 \rangle$ - Roboter haben einen virtuellen Benzinverbrauch

#### Ziel

Überprüfung, ob der Roboter auf seinen Tankstand achtet (Tabelle 3.1 Nr. 13).

#### Objekte/Methoden/Funktionen

**Objekte**: Ein Roboter, Server

**Funktionen**: Treibstoffverbrauch der Roboter $\langle F60 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn der Tankstand des Roboters mit fortlaufender Zeit weniger wird.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$). Darüber hinaus muss der Roboter mit einer FCFS-KI* zur Abarbeitung der Aufträge initialisiert worden sein, damit sichergestellt wird, dass er den Auftrag auf jeden Fall annimmt und beendet.

#### Einzelschritte

Eingabe:

1. Einen Auftrag erstellen.
2. Diesen Auftrag senden.

#### Beobachtungen / Log / Umgebung

Beobachtung des Roboter-Tankstandes über die Serverkonsole zu verschiedenen Zeitpunkten. Der Wert des Tankstandes muss mit fortlaufender Zeit sinken.

#### Abhängigkeiten

Mit der Teilnahme an seiner ersten Auktion ($\langle T500 \rangle$) startet der Benzinverbrauch des Roboters.

### Testfall $\langle T800 \rangle$ - Routenberechnung durch Roboter

#### Ziel

Überprüfung, ob der Antrieb des Roboters korrekt funktioniert, so dass er sich wie gewünscht bewegen kann (Tabelle 3.1 Nr. 2). Außerdem, ob der Roboter die serverseitigen Anweisungen korrekt umsetzt (Tabelle 3.1 Nr. 4). Weiterhin muss überprüft werden, ob die KI der Roboter korrekt arbeitet und die Roboter von dieser KI autonom gesteuert werden (Tabelle 3.1 Nr. 5). Schließlich muss die Überprüfung erfolgen, ob das Wegenetz vom Roboter korrekt erkannt und benutzt wird und er eine eigene Route finden kann (Tabelle 3.1 Nr. 7).

#### Objekte/Methoden/Funktionen

**Objekte**: Ein Roboter, Server

**Funktionen**: Routenberechnung durch Roboter $\langle F80 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn der Roboter eigenständig seine Route zu dem für seinen Auftrag benötigten Ziel berechnet.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$). Darüber hinaus muss der Roboter mit einer FCFS-KI* zur Abarbeitung der Aufträge initialisiert worden sein, damit sichergestellt wird, dass er den Auftrag auf jeden Fall annimmt.

#### Einzelschritte

Eingabe:

1. Auftrag erstellen, Ziel notieren.
2. Auftrag senden.

#### Beobachtungen / Log / Umgebung

Beobachtung des Roboters bei der Abwicklung seines Auftrages. Zu Untersuchen ist, ob er am vorher notierten Ziel ankommt.

#### Abhängigkeiten

Dem Straßenverlauf korrekt folgen zu können ist zwingend erforderlich $\langle T600 \rangle$.

### Testfall $\langle T900 \rangle$ - Streckenfreigabe durch Server

#### Ziel

Überprüfung, ob der Roboter die serverseitigen Anweisungen korrekt umsetzt (Tabelle 3.1 Nr. 4). Außerdem, ob der Server das Wegenetz korrekt verarbeitet und benutzte Strecken sperrt (Tabelle 3.1 Nr. 8).

#### Objekte/Methoden/Funktionen

**Objekte**: Roboter, Server

**Funktionen**: Streckenfreigabe durch Server $\langle F70 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Roboter Freigaben für Streckenabschnitte erhalten bzw. ihre Routen bei Sperrungen ändern.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$).

#### Einzelschritte

Eingabe:

1. Mehrere Auftrag erstellen.
2. Diese Aufträge senden.
3. Sperrung einer Kante, die zur Erfüllung eines Auftrags, den ein Roboter angenommen hat, befahren werden muss.

#### Beobachtungen / Log / Umgebung

Beobachtung des Roboters während seiner Fahrt. Zu beachten ist, ob er seine Route ändert.

#### Abhängigkeiten

Abhängig von $\langle T800 \rangle$, da der Roboter ohne eine zuvor berechnete Route keine Bewegung durchführt, für die er eine Freigabe des Servers benötigt.

### Testfall $\langle T1000 \rangle$ - Transportvolumen der Roboter

#### Ziel

Überprüfung, ob der Roboter die serverseitigen Anweisungen korrekt umsetzt (Tabelle 3.1 Nr. 4).

#### Objekte/Methoden/Funktionen

**Objekte**: Ein Roboter, Server

**Funktionen**: Transportvolumen der Roboter $\langle F50 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn der Roboter nur maximal so viel transportiert, wie es sein Transportvolumen zulässt oder er den Auftrag nicht annimmt.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$). Darüber hinaus muss der Roboter mit einer FCFS-KI* zur Abarbeitung der Aufträge initialisiert worden sein, damit sichergestellt wird, dass er den Auftrag auf jeden Fall annimmt.

#### Einzelschritte

Eingabe:

1. Auftrag erstellen, dessen Volumen die Kapazität des Roboters überschreitet.
2. Auftrag senden.

#### Beobachtungen / Log / Umgebung

Beobachtung der Auftragsauktion. Der Roboter darf diesen Auftrag nur annehmen, wenn er er den Transport im Nachhinein aufteilt, so dass das Volumen sein maximales Transportvolumen nicht überschreitet.

#### Abhängigkeiten

Bei Auktionsannahme prüft der Roboter sein Transportvolumen $\langle T500 \rangle$.

### Testfall $\langle T1100 \rangle$ - Spieldarstellung der Spieleroberfläche

#### Ziel

Überprüfung, ob zwischen Server und Client eine Verbindung besteht (Tabelle 3.1 Nr. 9).

#### Objekte/Methoden/Funktionen

**Objekte**: Server, Spieleroberfläche*

**Funktionen**: Spieldarstellung der Spieleroberfläche* $\langle F100 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Oberfläche angezeigt und aktualisiert wird.

#### Vorbedingung

Das Spiel darf noch nicht gestartet sein. Darüber hinaus muss ein freier Roboter zur Verfügung stehen, welcher dem Spieler zugewiesen werden kann.

#### Einzelschritte

Eingabe:

1. Als neuer Spieler anmelden und einen Auftrag ersteigern.

Ausgabe:

1. Spieleroberfläche* wird angezeigt.
2. Parallel startet das Spiel wie üblich.
3. Die Daten des Autrags können über die Spieleroberfläche mitverfolgt werden.

#### Beobachtungen / Log / Umgebung

Beobachtung des Computers, auf dem die Spieleroberfläche* gestartet wurde. Die Oberfläche muss die Visualisierung des Wegenetzes mit belegten und freien Streckenabschnitten und der Position der Roboter anzeigen und aktualisieren.

#### Abhängigkeiten

Abhängig von $\langle T900 \rangle$, da der Roboter zur Auftragsdurchführung die Freigaben des Servers benötigt.

### Testfall $\langle T1200 \rangle$ - Vermeidung von Deadlocks

#### Ziel

Überprüfung, ob die Algorithmen der Roboter einem Deadlock* entgegenwirken (Tabelle 3.1 Nr. 10).

#### Objekte/Methoden/Funktionen

**Objekte**: Roboter, Server, Spieleroberfläche*, zwei Spieler

**Funktionen**: Vermeidung von Deadlocks* $\langle F130 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Roboter einander ausweichen und sie nicht in eine Deadlock-Situation* kommen.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$), darüber hinaus befinden sich die Roboter auf für eine Simulation dieses Testfalles geeigneten Positionen.

#### Einzelschritte

Eingabe:

1. Pro beteiligten Roboter einen Auftrag so erstellen, dass eine Deadlock-Situation* entsteht.
2. Beide Aufträge senden.
3. Je einen Auftrag pro Client annehmen.

#### Beobachtungen / Log / Umgebung

Beobachtung der Roboter bei den geplanten Deadlock-Situation*.

#### Abhängigkeiten

Für jede Bewegung des Roboters auf dem Wegenetz wird eine Freigabe des Servers benötigt $\langle T900 \rangle$.

### Testfall $\langle T1300 \rangle$ - Datenverwaltung des Servers

#### Ziel

Überprüfung, ob die Bluetooth-Komponenten von Server und Roboter funktionsfähig sind und miteinander kommunizieren können (Tabelle 3.1 Nr. 3). Außerdem, ob die Statistiken korrekt und logisch sind (Tabelle 3.1 Nr. 11).

#### Objekte/Methoden/Funktionen

**Objekte**: Server, Roboter

**Funktionen**: Datenverwaltung $\langle F20 \rangle$ (zweite Hälfte)

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die unter dem Punkt Beobachtung stehenden Statistiken der Roboter sowie Kartendaten laufend aktualisiert werden.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$).

#### Einzelschritte

Eingabe:

1. Mehrere Aufträge erstellen.
2. Mehrere Aufträge senden.

Ausgabe:

1. Statistiken (siehe Beobachtung ).

#### Beobachtungen / Log / Umgebung

Beobachtung von Aufträgen und Preisbildung von Waren durch Angebot und Nachfrage, des Weiteren Gewinne und Statistiken der Roboter.

#### Abhängigkeiten

Abhängig von $\langle T900 \rangle$ und $\langle T1000 \rangle$, da sämtliche Daten für die Statistiken benötigt werden.

### Testfall $\langle T1400 \rangle$ - Statistikanzeige der Spieleroberfläche

**Ziel**

Überprüfung, ob die Bluetooth-Komponenten von Server und Roboter funktionsfähig sind und miteinander kommunizieren können (Tabelle 3.1 Nr. 3).

**Objekte/Methoden/Funktionen**

**Objekte**: Roboter, Server

**Funktionen**: Statistikanzeige der Spieleroberfläche* $\langle F110 \rangle$

**Pass/Fail Kriterien**

Der Test ist erfolgreich, wenn die Statistiken der Roboter angezeigt und aktualisiert werden.

**Vorbedingung**

Das Spiel ist in Betrieb ($\langle F90 \rangle$).

**Einzelschritte**

Eingabe:

1. Mehrere Aufträge erstellen.
2. Mehrere Aufträge senden.

**Beobachtungen / Log / Umgebung**

Beobachtung der Statistiken auf der Spieleroberfläche* (Client).

**Abhängigkeiten**

Die Spieleroberfläche bezieht ihre Daten von dem Server $\langle T1300 \rangle$.

### Testfall $\langle T1500 \rangle$ - Spieldarstellung über Benutzeroberfläche

#### Ziel

Überprüfung, ob die Visualisierung auf der zugrunde liegenden Statistik beruht (Tabelle 3.1 Nr. 14).

#### Objekte/Methoden/Funktionen

**Objekte**: Roboter, Server, Benutzeroberfläche

**Funktionen**: Spieldarstellung über Benutzeroberfläche $\langle F140 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn alle Statistiken auf der Benutzeroberfläche korrekt angezeigt werden. Darüber hinaus müssen Aufträge, die über die Oberfläche eingegeben werden, korrekt in das Auktionssystem eingegliedert werden.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$).

#### Einzelschritte

Eingabe:

1. Mehrere Aufträge erstellen.
2. Mehrere Aufträge senden.

#### Beobachtungen / Log / Umgebung

Beobachtung und Verwaltung der Aufträge, Verwaltung der Roboter, die Oberfläche muss darüber hinaus die Visualisierung des Wegenetzes mit belegten und freien Streckenabschnitten und der Position der Roboter anzeigen und aktualisieren.

#### Abhängigkeiten

Die Benutzeroberfläche bezieht ihre Daten vom Server $\langle T1300 \rangle$.

### Testfall $\langle T1600 \rangle$ - Roboter soll bei leerem Tank stehen bleiben

**Ziel**

Überprüfung, ob der Roboter auf seinen Tankstand achtet (Tabelle 3.1 Nr. 13).

**Objekte/Methoden/Funktionen**

**Objekte**: Roboter, Server, Benutzeroberfläche

**Funktionen**: Routenberechnung durch Roboter $\langle F80 \rangle$

**Pass/Fail Kriterien**

Der Test ist erfolgreich, wenn der Roboter stehen bleibt, sobald sein Tank leer ist.

**Vorbedingung**

Das Spiel ist in Betrieb ($\langle F90 \rangle$).

**Einzelschritte**

Keine Schritte notwendig, der Roboter soll „verhungern“.

**Beobachtungen / Log / Umgebung**

Beobachtung des Roboters über die Benutzeroberfläche. Sein Tankstand soll nach einer gewissen Zeit auf 0 sinken. Er darf daraufhin keine Aufträge annehmen und ist aus dem Spiel ausgeschieden.

**Abhängigkeiten**

Über die vom Server an die Benutzeroberfläche übermittelten Daten (Benzinverbrauch) $\langle T1300 \rangle$ kann der Tankstand überprüft werden.

### Testfall $\langle T1700 \rangle$ - Industrien produzieren Waren

#### Ziel

Überprüfung, ob die Industrien erhaltene Waren gemäß des Produktionszyklus zu neuen Waren weiterverarbeiten (Tabelle 3.1 Nr. 15).

#### Objekte/Methoden/Funktionen

**Objekte**: Ein Roboter, Server, Benutzeroberfläche

**Funktionen**: Datenverwaltung $\langle F20 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn die Industrien aus angelieferten Waren neue Produkte fabrizieren.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$). Darüber hinaus muss der Roboter mit einer FCFS-KI* zur Abarbeitung der Aufträge initialisiert worden sein, damit sichergestellt wird, dass er den Auftrag auf jeden Fall annimmt.

#### Einzelschritte

Eingabe:

1. Ein Auftrag erstellen, so dass der Roboter die benötigten Waren als Rohstoff an die gewünschte Industrie liefert.
2. Diesen Auftrag senden.

#### Beobachtungen / Log / Umgebung

Beobachtung der Rohstoffe über die Benutzeroberfläche; das Vorkommen der bestehenden Waren, die zur Produktion benötigt werden, muss nach Erfüllung des Auftrages ansteigen. Nach einer gewissen Zeit muss dieses Vorkommen wieder sinken und das der produzierten Ware steigen.

#### Abhängigkeiten

Anzeige der vom Server berechneten Daten ist erst durch $\langle T1500 \rangle$ möglich.

### Testfall $\langle T1800 \rangle$ - Steuerung des Roboters durch Spieler

#### Ziel

Überprüfung, ob der Roboter bei seiner Routenwahl durch den Spieler beeinflussbar ist, wenn dieser ihn über die Spieleroberfläche verwaltet (Tabelle 3.1 Nr. 13).

#### Objekte/Methoden/Funktionen

**Objekte**: Ein Roboter, Server, Spieleroberfläche*

**Funktionen**: Routenberechnung $\langle F80 \rangle$

#### Pass/Fail Kriterien

Der Test ist erfolgreich, wenn der Roboter die vom Spieler eingestellte Route übernimmt sowie die vom Spieler ausgewählten Aufträge ausführt.

#### Vorbedingung

Das Spiel ist in Betrieb ($\langle F90 \rangle$). Spieler nimmt am Spiel teil und ein Roboter wurde ihm zugeteilt.

#### Einzelschritte

Eingabe:

1. Beliebigen Auftrag annehmen.
2. Die vom Roboter vorgeschlagene Route ändern.
3. Routenänderung senden

Ausgabe:

1. Geänderte Route anzeigen.

#### Beobachtungen / Log / Umgebung

Beobachtung der Route des Roboter und Vergleich mit der Ausgabe

#### Abhängigkeiten

Spieldarstellung auf Spieleroberfläche benötigt $\langle T1400 \rangle$.

# 4 Integrationstest

Dieses Kapitel prüft die Integration der einzelnen Komponenten im Softwareprojekt.

## 4.1 Zu testende Komponenten

Die jeweiligen Komponenten können als funktionsfähig betrachtet werden, wenn alle zugeordneten Testfälle problemlos durchlaufen.

| **Nr.** | **Komponente** | **Testfälle** |
|---|---|---|
| 1 | $\langle C60 \rangle$ Kommunikation (Server), $\langle C30 \rangle$ Kommunikation (Roboter) | $\langle T1900 \rangle$ |
| 2 | $\langle C10 \rangle$ KI (Roboter), $\langle C30 \rangle$ Kommunikation (Roboter) | $\langle T2000 \rangle$ |
| 3 | $\langle C10 \rangle$ KI (Roboter), $\langle C120 \rangle$ Steuerung (Roboter) | $\langle T2100 \rangle$ |
| 4 | $\langle C40 \rangle$ Kommunikation (Spieler), $\langle C60 \rangle$ Kommunikation (Server) | $\langle T2200 \rangle$ |
| 5 | $\langle C20 \rangle$ GUI (Spieler), $\langle C40 \rangle$ Kommunikation (Spieler) | $\langle T2300 \rangle$ |
| 6 | $\langle C50 \rangle$ GUI(Server), $\langle C80 \rangle$ Spielkoordination, $\langle C70 \rangle$, $\langle C100 \rangle$ | $\langle T2400 \rangle$ |
| 7 | $\langle C50 \rangle$ GUI(Server), $\langle C80 \rangle$ Spielkoordination, $\langle C100 \rangle$ Industrieverwaltung | $\langle T2500 \rangle$ |
| 8 | $\langle C50 \rangle$ GUI(Server), $\langle C80 \rangle$ Spielkoordination, $\langle C90 \rangle$ Wegenetzverwaltung | $\langle T2600 \rangle$ |
| 9 | $\langle C50 \rangle$ GUI(Server), $\langle C80 \rangle$ Spielkoordination, $\langle C90 \rangle$ Wegenetzverwaltung, $\langle C100 \rangle$Industrieverwaltung | $\langle T2700 \rangle$ |
| 10 | $\langle C10 \rangle$ KI (Roboter, $\langle C30 \rangle$ Kommunikation (Roboter), $\langle C50 \rangle$ GUI(Server), $\langle C60 \rangle$, $\langle C80 \rangle$ Spielkoordination, $\langle C90 \rangle$ Wegenetzverwaltung, $\langle C120 \rangle$ Spielverwaltung | $\langle T2800 \rangle$ |

Tabelle 4.1: Zu testende Komponenten

## 4.2 Testverfahren

Für den Integrationstest kommt als dynamischer Funktionstest das Black-Box Verfahren zum Einsatz. In möglichst praxisnaher Umgebung wird hier nur das Zusammenspiel der einzelnen Systemkomponenten des Produkts getestet. Auf eine bestimmte Eingabe über die von außen sichtbaren Schnittstellen (Eingabemasken) oder einen bestimmten über die Anwenderoberfläche gesteuerten Programmablauf muss das Produkt in entsprechender Weise reagieren und korrekte Zwischenergebnisse sowie Ausgaben erzielen. Das beobachtete Verhalten wird in zahlreichen Testfällen mit dem über die funktionalen Anforderungen festgelegten Verhalten verglichen. Testfälle bezüglich möglicher Dateneingaben der Anwender können beispielsweise mit Kombinationen von Grenzwerten durchgeführt und anhand des zu erwartenden Ergebnisses auf ihre Korrektheit überprüft werden.
Es werden dabei keine Testskripte verwendet.

## 4.3 Testfälle

### Testfall $\langle T1900 \rangle$ - Komponenten C60 + C30

**Ziel**

Überprüfung der einwandfreien Kommunikation zwischen Server und Roboter.

**Objekte/Methoden/Funktionen**

Komponente C60: Klassen NXTManager und NXTReceiver (alle Methoden)
Komponente C30: Klassen BluetoothSender und BluetoothListener (alle Methoden)

**Pass/Fail Kriterien**

Der Testfall ist erfolgreich, wenn alle Kommunikationsmethoden ohne Fehler funktionieren und die jeweiligen Ausgaben mit den beschriebenen Erwartungen übereinstimmen.

**Vorbedingung**

Keine.

**Einzelschritte**

**Richtung C60 -> C30**

1. Instanz der Klasse **NXTManager** mit passendem Namen erstellen und die Initialisierungsmethode mit geeigneten Daten (Name, Position) für den Roboter aufrufen. Den dazugehörigen Roboter vor dem Aufruf der Methode anschalten und das Empfangsmodul ($\langle C30 \rangle$) starten.
2. Methode „sendGraph“ aufrufen, hier muss ein gültiger Graph übergeben werden (evtl. durch Verwendung von Teilen der Komponente $\langle C50 \rangle$).
3. Die Methode „sendOrder“ mit einem gültigen Auftrag aufrufen.

4. Methode „sendAuctionResult“ mit der ID, die beim Senden des Auftrags auf dem Display des Roboters angezeigt wurde, und einem „true“ zur Bestätigung der erfolgreichen Auktion aufrufen.
5. Nach der Kanten-Statusanfrage des Roboters (siehe Punkt 3 bei der entgegengesetzten Richtung) wird vom Server aus die Methode „approveEdge“ aufgerufen, die passenden Knotendaten werden gefolgt von einem „true“ für die Bestätigung der Freigabe übergeben.
6. Die Methode kill wird aufgerufen.
7. Dem Roboter wird mit der Methode „sendRoute“ eine Route übertragen.

**Richtung C30 -> C60**

1. Der Roboter wird angeschaltet und durch ein passendes Programm wird eine Instanz der Klasse **BluetoothSender** initialisiert. Passend dazu wird in einer Testumgebung (die restlichen Komponenten des Servers sind auszuschließen) die Klasse **NXTReceiver** zum Empfang der Daten instanziiert.
2. Die Methode „sendLocation“ wird mit einer Zahl für den aktuellen Knoten des Roboters aufgerufen.
3. Die Methode „requestEdgeStatus“ wird mit zwei passenden anliegenden Knoten aufgerufen.
4. Die Methode „releaseEdge“ wird mit zwei passenden anliegenden Knoten aufgerufen.
5. Die Methode „releaseNode“ wird mit einem freizugebenden Knoten und dem Knoten, der als nächstes besucht werden soll, aufgerufen.
6. Die Methode „sendBid“ wird mit der ID (möglichst die aus Punkt 3 der anderen Richtung) und einem passenden Gebot aufgerufen.
7. Die Methode „sendRoute“ wird mit einem beliebigen passenden Routen-Objekt aufgerufen.
8. Die Methode „sendOrderPickup“ wird mit einer ID, die zu einem gültigen Auftrag passt, aufgerufen.
9. Die Methode „sendOrderDeposit“ wird mit einer ID, die zu einem gültigen Auftrag passt, aufgerufen.

### Beobachtungen / Log / Umgebung

**Richtung C60 -> C30**

1. Der Roboter zeigt auf seinem Display durch den Hinweistext "Robot Data receivedän, dass er korrekt initialisiert wurde.

2. Der Roboter zeigt auf seinem Display zuerst die übertragene Soll-Größe und nach der Übertragung die tatsächliche Größe des Graphen an, diese Werte sollen gleich sein (im Falle der empfohlenen Datei jeweils 20).
3. Der Roboter bestätigt den Empfang des Auftrages mit einer entsprechenden Ausgabe auf seinem Display.
4. Der Roboter bestätigt den Empfang des Auktionsgewinns mit einer entsprechenden Ausgabe auf seinem Display.
5. Der Roboter dokumentiert den Empfang der Kantenfreigabe mit einer entsprechenden Ausgabe auf seinem Display.
6. Der Roboter empfängt den Befehl, stellt alle Aktionen ein und beendet das ausgeführte Programm.
7. Der Roboter zeigt die Route in Form der Knoten auf seinem Display an.

**Richtung C30 -> C60**

1. Der Roboter bestätigt die Verbindung mit der Anzeige „connected“ auf seinem Display.
2. Der Server gibt die empfangene Position in der Konsole aus.
3. In der Konsole des Servers wird die Anfrage mit den beiden übergebenen Knoten ausgegeben.
4. In der Konsole des Servers wird die Freigabe der Kante mit den beiden übergebenen Knoten ausgegeben.
5. In der Konsole des Servers wird die Freigabe des Knotens mit den beiden übergebenen Knoten ausgegeben.
6. In der Konsole des Servers wird das empfangene Gebot mit der zugehörigen ID ausgegeben. Die ID gleicht der des ursprünglich verschickten Auftrags.
7. In der Konsole des Servers wird die empfangene Route in Form von abzufahrenden Knoten ausgegeben. Diese stimmt mit der gesendeten überein.
8. In der Konsole des Servers wird die Aufnahme des Auftrages mit der passenden ID angezeigt.
9. In der Konsole des Servers wird die Abgabe des Auftrages mit der passenden ID angezeigt.

### Besonderheiten

Keine.

**Abhängigkeiten**

$\langle T3000 \rangle$

### Testfall $\langle T2000 \rangle$ - Komponenten C10 + C30

**Ziel**

Überprüfung der korrekten Übermittlung von Daten von der KI zur Kommunikation und umgekehrt.

**Objekte/Methoden/Funktionen**

Komponente C10: Klasse SimpleKI

Komponente C30: Klassen Bluetooth-Sender, BluetoothListener (alle Methoden)

**Pass/Fail Kriterien**

Der Testfall ist erfolgreich, wenn alle komponentenübergreifenden Methodenaufrufe ohne Fehler funktionieren.

**Vorbedingung**

Keine.

**Einzelschritte**

**Richtung C10 -> C30**

1. Methode „sendBid“ mit einem passenden Gebot aufrufen.
2. Methode „requestEdgeStatus“ mit zwei passenden Knoten aufrufen.

**Richtung C30 -> C10**

1. Methode „calculateQuotation“ mit einem passenden Order-Objekt aufrufen.
2. Methode „actionAfterAuction“ mit einer passenden Auftrags-ID und dem Wert „true“ für eine Auftragsbestätigung aufrufen.
3. Methode „deadlockAction“ mit zwei passenden Knoten und dem Wert „0“ für einen noch nicht gelösten Deadlock aufrufen.

**Beobachtungen / Log / Umgebung**

Bei Ausführung der beschriebenen Schritte sollten in den empfangenden Komponenten die gleichen Ergebnisse/Reaktionen auftreten, die ohne Einbindung der Kommunikation zwischen beiden Komponenten auch auftreten würden.

**Besonderheiten**

Da die Funktionalität der Methoden schon durch die Unit-Tests der Komponenten kontrolliert wurde, wird davon ausgegeangen, dass der erfolgreiche Test jeweils einer Methode pro Richtung die Funktionalität aller Methoden bestätigt.

**Abhängigkeiten**

Alle Unit-Tests zu den beiden angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2100 \rangle$ - Komponenten C10 + C120

**Ziel**

Überprüfung der korrekten Übermittlung von Daten zwischen KI und Steuerung.

**Objekte/Methoden/Funktionen**

Komponente C10: Klasse SimpleKI (Methoden getOrderList, moveToBillOfLading, sendOrderDeposit removeFirstOrder, getBillOfLading, removeFirstBillOfLading)
Komponente C120: Klassen Actionpoint, ActionpointListener (Methoden getDistance, getDestination, setRoute, decDistance, getApproveCount, getNextNode, setFirstCommand, setFreeEdge)

**Pass/Fail Kriterien**

Der Testfall ist erfolgreich, wenn alle komponentenübergreifenden Methodenaufrufe ohne Fehler funktionieren.

**Vorbedingung**

Keine.

**Einzelschritte**

**Richtung C10 -> C120**

1. Über die KI die angegebenen Methoden der Klasse **ActionpointListener** mit passenden Übergabewerten aufrufen (siehe Unit-Tests der Klasse).

**Richtung C30 -> C10**

1. Über die Klasse **Actionpoint** die angegebenen Methoden der Klasse **SimpleKI** mit passenden Übergabewerten aufrufen (siehe Unit-Tests der Klasse).

**Beobachtungen / Log / Umgebung**

Bei Ausführung der beschriebenen Schritte sollten in den empfangenden Komponenten die gleichen Ergebnisse/Reaktionen auftreten, die ohne Einbindung der Kommunikation zwischen beiden Komponenten auch auftreten würden.

**Besonderheiten**

Da die Funktionalität der Methoden schon durch die Unit-Tests der Komponenten kontrolliert wurde, wird davon ausgegeangen, dass der erfolgreiche Test jeweils einer Methode pro Richtung die Funktionalität aller Methoden bestätigt.

**Abhängigkeiten**

Alle Unit-Tests zu den beiden angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2200 \rangle$ - Komponenten C40 + C60

**Ziel**

Überprüfung der korrekten Übermittlung von Daten zwischen Server und Client.

**Objekte/Methoden/Funktionen**

Komponente C40: Klasse CommunicationClient (Methode sendData)

Komponente C60: Klasse CommunicationServer (Methode sendData)

**Pass/Fail Kriterien**

Der Testfall ist erfolgreich, wenn die Ausgaben der jeweils empfangenen Objekte mit den gesendeten Eingaben übereinstimmen.

**Vorbedingung**

Keine.

**Einzelschritte**

Mit den jeweiligen „sendData“-Methoden werden in beide Richtungen verschiedene Objekte gesendet, welche auf der jeweils anderen Seite empfangen und ausgegeben werden.

**Beobachtungen / Log / Umgebung**

Betrachtung der ausgegebenen Objekte und Vergleich mit den Eingaben.

**Besonderheiten**

Keine.

**Abhängigkeiten**

Alle Unit-Tests zu den beiden angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2300 \rangle$ - Komponenten C20 + C40

**Ziel**

Überprüfung der korrekten Übermittlung von Daten zwischen Server und Client.

**Objekte/Methoden/Funktionen**

Komponente C20: Klasse PlayerAdministration (Methode distributeData)

Komponente C40: Klasse CommunicationServer (Methode sendData)

**Pass/Fail Kriterien**

Der Testfall ist erfolgreich, wenn alle komponentenübergreifenden Methodenaufrufe ohne Fehler funktionieren.

**Vorbedingung**

Keine.

**Einzelschritte**

**Richtung C20 -> C40**

1. Methode „sendData“ der Klasse **CommunicationServer** von der Klasse **PlayerAdministration** aus mit passenden Test-Objekten aufrufen.
2. Empfangene Objekte in der Konsole ausgeben lassen.

**Richtung C40 -> C20**

1. Methode „distributeData“ der Klasse **PlayerAdministration** von der Klasse **CommunicationServer** aus mit passenden Test-Objekten aufrufen.
2. Empfangene Objekte in der Konsole ausgeben lassen.

**Beobachtungen / Log / Umgebung**

Betrachtung der ausgegebenen Objekte und Vergleich mit den Eingaben.

**Besonderheiten**

Keine.

**Abhängigkeiten**

Alle Unit-Tests zu den beiden angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2400 \rangle$ - Komponenten C50 + C80 + C70 + C100

**Ziel**

Automatische oder durch den Benutzer erstellte Aufträge werden von der Spielkoordination an die Auktionsverwaltung weitergegeben.

**Objekte/Methoden/Funktionen**

**Methoden**: actionperformed(), searchForOrder(), addOrder()
**Funktionen**:$\langle F20 \rangle$, $\langle F140 \rangle$

**Pass/Fail Kriterien**

Die Industrieverwaltung erstellt Aufträge mit der Methode „searchForOrder“ und leitet sie über die Spielkoordination an die Auktionsverwaltung weiter. Über die GUI(Server) kann der Benutzer einen Auftrag manuell erstellen. Der erstellte Auftrag wird kurz darauf in der Auktionsliste angezeigt.

**Vorbedingung**

Die Initialisierung wurde abgeschlossen und das Spiel gestartet. Die Industrien wurden korrekt ins Spiel eingebunden und es existiert eine Erzeuger-Verbraucher Situation zwischen zwei Industrien.

**Einzelschritte**

1. Der Benutzer wählt den Radio-Button zum manuellen Erstellen von Aufträgen.
2. Die entsprechenden Felder für Start, Ziel, Menge, Ware und Preis werden vom Benutzer korrekt ausgefüllt.

**Beobachtungen / Log / Umgebung**

Während bei aktivierter automatischer Auftragserstellung die Aufträge automatisch von der Industrieverwaltung erzeugt werden, wird bei der Auswahl zur manuellen Erstellung die automatische Auftragserstellung gestoppt. Die Eingabe des Benutzers wird mit einem

Hinweisfenster bestätigt. Die Konsole zeigt in jedem Fall den an die **GameAdministration** übermittelte Order und den von der Auktionsverwaltung folgenden Auktionsstart an.

#### Besonderheiten

Keine.

#### Abhängigkeiten

Alle Unit-Tests zu den angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2500 \rangle$ - Komponenten C50 + C80 + C100

#### Ziel

Die im Initialisierungsfenster angezeigten Industrien werden ins Spiel eingebunden.

#### Objekte/Methoden/Funktionen

**Methoden**: actionperformed(), setIndustryList()
**Funktionen**:$\langle F20 \rangle$, $\langle F140 \rangle$

#### Pass/Fail Kriterien

Nachdem die XML Datei für Industrien geladen wurde, werden Änderungen in der Industrietabelle registriert. Beim Starten des Spiels werden die Industrien an die Industrieverwaltung weitergereicht.

#### Vorbedingung

Der **IndustryParser** arbeitet korrekt. Die Industriedatei kann ausgelesen werden.

#### Einzelschritte

1. Starten des Server-Programms.
2. Laden der Kartendatei.
3. Laden der Industriedatei.
4. Hinzufügen eines beliebigen Roboters, damit das Spiel gestartet werden kann.
5. Starten des Spiels.

#### Beobachtungen / Log / Umgebung

Die Liste der Industrien wird zum einem vor Spielstart auf der Konsole ausgegeben. Zum anderen können die Industrien über Industriedetails auf der GUI(Server) abgerufen werden. Zu beobachten ist, ob die jeweils angezeigten Daten mit denen bei der Initialisierung angezeigten übereinstimmen.

#### Besonderheiten

Keine.

#### Abhängigkeiten

Alle Unit-Tests zu den angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2600 \rangle$ - Komponenten C50 + C80 + C90

**Ziel**

Die XML-Kartendatei wird geladen, der dazu passende Graph wird von der Wegenetzverwaltung erstellt und von der GUI(Server) angezeigt.

**Objekte/Methoden/Funktionen**

**Methoden**: actionperformed(), setGraph(), makeCoordinates(), alignGraphList()
**Funktionen**:$\langle F20 \rangle$, $\langle F140 \rangle$

**Pass/Fail Kriterien**

Die in der Karten-XML-Datei enthaltenen Informationen werden an die Größe des Fensters der GUI(Server) angepasst als Graph dargestellt.

**Vorbedingung**

Die XML-Kartendatei wurde korrekt erstellt und der darin enthaltene Graph ist zusammenhängend.

**Einzelschritte**

1. Starten des Server-Programms.
2. Laden der Kartendatei.
3. Laden der Industriedatei.
4. Hinzufügen eines beliebigen Roboters, damit das Spiel gestartet werden kann.
5. Starten des Spiels.

**Beobachtungen / Log / Umgebung**

Bei Erzeugung der Darstellung des Graphen durch „makeCoordinates“ werden die Knoten und Richtungen auf der Konsole ausgegeben. Der Zusammenhangtest wird von der gleichen Methode bestätigt. Nach Verschieben der Koordinaten zum Nullpunkt wird wieder der gesamte Graph auf Konsole ausgegeben. Zu beobachten ist, ob die auf Konsole angegebenen Daten mit denen der XML und der Darstellung auf dem Gamedisplay übereinstimmen. Die Darstellung auf dem Gamedisplay soll dabei korrekt sein, ohne Überschneidung von Knoten.

**Besonderheiten**

Keine.

**Abhängigkeiten**

Alle Unit-Tests zu den angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2700 \rangle$ - Komponenten C50 + C80 + C90 + C100

**Ziel**

Die Positionen der Roboter und der Industrien werden während der Initialisierung über

das Initialisierungsfenster angezeigt und können entsprechend verändert werden. Zu Spielbeginn werden die Objekte dann auf ihren jeweiligen Positionen in der - mit der Klasse GameDisplay erstellten - Kartenvisualisierung angezeigt.

**Objekte/Methoden/Funktionen**

**Methoden**: actionperformed(), addRobot(),setIndustryList() ,robotToMap(), industryToMap(), setLocation(), drawEdge()
**Funktionen**:$\langle F20 \rangle$, $\langle F140 \rangle$

**Pass/Fail Kriterien**

Die durch den Benutzer gemachten Eingaben werden korrekt ausgelesen und an die Industrieverwaltung und Wegenetzverwaltung weitergegeben. Die Methode „setLocation“ meldet dabei keine Kollision. Werden die Roboter auf ihren gewünschten Positionen angezeigt, ist der Test erfolgreich.

**Vorbedingung**

Die XML-Dateien können korrekt ausgelesen werden.

**Einzelschritte**

1. Starten des Server-Programms.
2. Laden der Kartendatei.
3. Laden der Industriedatei.
4. Hinzufügen eines beliebigen Roboters, damit das Spiel gestartet werden kann.
5. Ändern identischer Standorte bestimmter Roboter oder Industrien in den Tabellen des Initialisierungsfensters.
6. Starten des Spiels.

**Beobachtungen / Log / Umgebung**

Zu beobachten ist, ob die Positionen der Industrien bzw. Roboter innerhalb der Industrieliste bzw. Roboterliste mit den Eingaben des **InitFrame** und der Platzierung auf der GUI(Server) übereinstimmen. Die Konsole listet vor Spielstart sowohl die gespeicherten Industriedaten als auch die gespeicherten Roboterdaten. Nach Aufruf von „robotToMap“ und „industryToMap“ meldet die Methode „setLocation“ über die Konsole, ob die Platzierung erfolgreich war. Die Methode „drawEdge“ zeichnet den Roboter anhand der zuvor gespeicherten Daten auf den Graphen.

**Besonderheiten**

Keine.

**Abhängigkeiten**

Alle Unit-Tests zu den angegebenen Komponenten müssen erfolgreich verlaufen sein.

### Testfall $\langle T2800 \rangle$ - Komponenten C10 + C30+ C50 + C60+ C80 + C90 + C120

#### Ziel

Der Roboter soll seine Ware an dem Startpunkt des Auftrages aufladen und am Zielpunkt des Auftrages wieder abliefern. Falls der Roboter dieses innerhalb einer vorgegebenen Zeit erledigen konnte, wird auf seinem Konto ein Gewinn verbucht.

#### Objekte/Methoden/Funktionen

**Methoden**:sendOrder(), sendOrderPickup(), sendOrderDeposit(), sendLocation(), distributeData(), checkRobotPosition()
**Funktionen**:$\langle F20 \rangle$, $\langle F140 \rangle$

#### Pass/Fail Kriterien

Die Methode „sendOrder“ erteilt dem Roboter einen Auftrag. Die KI des Roboters berechnet die Route und sendet bei Erreichen des Startknoten den Wunsch Ware aufzuladen „sendOrderPickup“. Bei Erreichen des Zielknotens wird „sendOrderDeposit“ aufgerufen. Die Kommunikation auf Serverseite mit der Spielkoordination verläuft dabei über die Methode „distributeData“. Diese überprüft die jeweiligen Meldungen des Roboters mit „checkRobotPosition“ und teilt dem Roboter die Waren zu. Durch Vergleich mit der Zeitstempel eines Order-Objekts wird der Erfolg oder Misserfolg eines Auftrags festgestellt. Dem Roboter wird im Erfolgsfall sein Gewinn gutgeschrieben. Im Falle eines Misserfolgs wird eine Strafe erteilt.

#### Vorbedingung

Die Initialisierung des Servers und der Roboter wurde abgeschlossen. Das Spiel ist in Betrieb. Die Statistiken werden korrekt dargestellt. Aufträge werden erstellt. Der Roboter hat einen Auftrag ersteigern können.

#### Einzelschritte

1. Warten auf das Abholen der Ware durch den Roboter.
2. Warten auf das Abliefern der Ware durch den Roboter.

#### Beobachtungen / Log / Umgebung

Die Kommunikation zwischen Roboter und Server wird auf der Konsole angezeigt. Die GUI(Server) zeigt den Roboter auf seiner jeweiligen Position an. Über die Auftragsliste kann geprüft werden, welche Knoten der Roboter zum Erledigen des Auftrages anfahren muss. Zusätzlich erscheinen Hinweisfenster beim Auf-/Abladen der Ware. Das Konto des Roboters wird in der Robotertabelle angezeigt. Die Meldung beim Abladen gibt im Erfolgsfall den Gewinn mit an.

#### Besonderheiten

Keine.

### Abhängigkeiten

Alle Unit-Tests zu den angegebenen Komponenten müssen erfolgreich verlaufen sein.

# 5 Unit-Tests

Dieses Kapitel testet die Funktion der Komponenten selbst.

## 5.1 Zu testende Komponenten

Die jeweiligen Komponenten können als funktionsfähig betrachtet werden, wenn alle zugeordneten Testfälle problemlos durchlaufen.

| **Nr.** | **Komponente** | **Testfälle** |
|---|---|---|
| 1 | Komponente $\langle C20 \rangle$ GUI(Spieler) | $\langle T2900 \rangle$ |
| 2 | Komponente $\langle C50 \rangle$ GUI(Server) | $\langle T3000 \rangle$ |
| 3 | Komponente $\langle C70 \rangle$ Auktionsverwaltung | $\langle T3100 \rangle$ |
| 4 | Komponente $\langle C80 \rangle$ Spielkoordination | $\langle T3200 \rangle$ |
| 5 | Komponente $\langle C90 \rangle$ Wegenetzverwaltung | $\langle T3300 \rangle$ |
| 6 | Komponente $\langle C100 \rangle$ Industrieverwaltung | $\langle T3400 \rangle$ |

Tabelle 5.1: Zu testende Anforderungen

## 5.2 Testverfahren

Es wurden größtenteils Klassen so modifiziert, dass sie Ausgaben haben mit den man die Richtigkeit prüfen kann. Falls es sich beispielsweise um eine GUI handelt wurden neue Fenster geöffnet die das Ergebnis enthielt oder die Ergebnisse waren direkt auf der GUI sichtbar.

### 5.2.1 Testskripte

Es wurden keine Testskripte verwendet.

## 5.3 Testfälle

### Testfall $\langle T2900 \rangle$ - Klassen PlayerFrame, InfoFrame, ProfitFrame, HelpFrame, GameDisplay

#### Ziel

Funktionalität des Spielerfensters testen.

#### Objekte/Methoden/Funktionen

**Objekte der Klassen**: PlayerFrame, InfoFrame, ProfitFrame, HelpFrame, GameDisplay

#### Pass/Fail Kriterien

Der Test war erfolgreich, falls das Spielerfenster alle Funktionen erfüllt.

#### Vorbedingung

Das Spiel muss erfolgreich gestartet sein und das Spielerfenster ist offen.

#### Einzelschritte

1. Klicken auf den Industriedetailsknopf.
2. Klicken auf den Gewinnstatistikknopf.
3. Drücken auf die „F1“ Taste oder auswählen des Menüpunktes „Hilfe“ in der Menubar.
4. Klicken auf den Infoknopf in der Menubar.
5. Darstellung des Wegenetztes.
6. Gebot zu ausgewähltem Auftrag abgeben.
7. Roboter einen Auftrag zuweisen.
8. Klicken auf den Schließenknopf.

#### Beobachtungen / Log / Umgebung

Das Industriedetailsfenster wird aufgerufen und wird geschlossen. Das Gewinnstatistikfenster wird aufgerufen und zeigt Gewinnstatistiken an und wird geschlossen. Nach dem Klick auf den Infoknopf in der Menubar wird das Infofenster aufgerufen mit den erforderlichen Werten und wird geschlossen. Das Wegenetz wird im Spielerfenster dargestellt und zeigt Informationen bei Interaktion an. Es kann Route für den Roboter geändert werden. Ein Gebot zu einem ausgewählten Auftrags aus der Auftragstabelle wird durch Klicken auf den Knopf „Gebot abgeben“ und auf der Konsole wird die Auftrags-ID und das Gebot ausgegeben. Falls der Spieler eine Falscheingabe getätigt hat erscheint ein Fenster, in dem auf den Fehler hingewiesen wird und das Gebot wird nicht abgegeben. Nach Auswahl eines Auftrags in der Auftragstabelle, der Eingabe des Roboternamens und dem Klicken des Knopfs „Roboter beauftragen“ erscheint ein Fenster, indem der Roboter und der Auftrag steht, der beauftragt wird. Nach Klicken des Schließenknopfs wird das Spielerfenster geschlossen.

#### Besonderheiten

Keine.

#### Abhängigkeiten

$\langle T1100 \rangle$, $\langle T1400 \rangle$, $\langle T1800 \rangle$

### Testfall $\langle T3000 \rangle$ - Klasse InitFrame, ServerFrame, InfoFrame, AddRobotFrame, IndustrieDetailsFrame, IndustryContentHandler, GameDisplay, GrapghParser, MyFirstIndustry, Gamedisplay

#### Ziel

Funktionalität des Benutzers- und Initialsierungsfensters testen.

#### Objekte/Methoden/Funktionen

**Objekte der Klassen**: InitFrame, ServerFrame, InfoFrame, AddRobotFrame,IndustrieDetailsFrame, IndustryContentHandler, GameDisplay, GrapghParser, MyFirstIndustry, Gamedisplay

#### Pass/Fail Kriterien

Der Test war erfolgreich, falls das Initialisierungsfenster und Benutzerfenster alle gewünschten Funktionen erfüllt.

#### Vorbedingung

Das Initialisierungsfenster muss offen sein.

#### Einzelschritte

1. Auswahl der Kartendatei.
2. Auswahl der Industriedatei.
3. Anzeigen der Industriedaten.
4. Hinzufügen von Robotern.
5. Hinzufügen von Spielern.
6. Roboter einem Team zuweisen.
7. Schließen des Initialisierungsfensters.
8. Aufträge anzeigen.
9. Öffnen des Informations-Pop-Ups.
10. Sperren und Freigeben von Knoten.
11. Sperren und Freigeben von Kanten.
12. Roboter entfernen.
13. Schließen des Benutzerfensters und Spiel beenden.

#### Beobachtungen / Log / Umgebung

Es wird die Kartendatei ausgewählt und es wird mit „Ok“ bestätigt, falls sie konform ist. Die Industriedatei wird ausgewählt und mit „Ok“ bestätigt, falls sie konform ist. Zusätzlich werden die Industriedaten, die eingelesen werden in der Tabelle angezeigt. Initialisierte Roboter konnten über den Knopf „Roboter hinzufügen“ und der Eingabe, des Namens des Roboters und seines Startstandorts in dem **AddRoboterFrame** hinzugefügt werden. Diese stehen dann in der Robotertabelle. Mit dem Klick auf „Spieler verbinden“ wird ein Spieler, der in seinem Verbindungsfensters seine Daten eingegeben hat, hinzugefügt und in der Tabelle „Spieler“ eingetragen. Nach Auswahl eines Roboters in der Robotertabelle und dem darauffolgendem Klick auf „Teammitgleid hinzufügen“ wird ein Roboter einem Team zugewiesen und in die jeweilige Tabelle des Teams geschrieben. Durch Klicken auf den Schließenknopf wird das Initialisierungsfensters geschlossen. Im Benutzerfenster werden in einer Tabelle Aufträge angezeigt und es kann durch das Betätigen eines Drop-Down-Menüs eine Auswahl der anzuzeigenden Aufträge getroffen werden. Durch das Anklicken von „Industriedetails“- und „Info“-Knöpfen erschienen Pop-ups, in denen die jeweiligen Informationen enthalten sind. Knoten oder Kanten konnten durch Rechtsklick und Auswahl des Unterpunkts „Sperren“ oder „Freigeben“ gesperrt und freigegeben werden, falls sie gesperrt sind erscheint ein Sperrungssymbol und bei Freigabe wird dieses nicht mehr angezeigt. Ein Roboter wird in der Robotertabelle ausgewählt und dann wird auf den Knopf „Roboter entfernen“ geklickt und es wird ein Pop-up geöffnet in dem man aufgefordert wird, den Roboter vom Spielfeld zu nehmen. Nach Bestätigung des Entfernens wird der Roboter aus dem Spiel und der Tabelle im Benutzerfenster entfernt. Das Benutzerfenster wird nach Klick des Schließenknopfs geschlossen und das Spiel wird beendet.

#### Besonderheiten

Keine.

#### Abhängigkeiten

$\langle T100 \rangle$, $\langle T200 \rangle$, $\langle T400 \rangle$, $\langle T900 \rangle$,$\langle T1500 \rangle$

### Testfall $\langle T3100 \rangle$ - Klasse Auction

#### Ziel

Gebote werden registriert und nach Ablauf der Auktion wird das niedrigste Gebot ermittelt.

#### Objekte/Methoden/Funktionen

**Objekte der Klassen**: Auction

#### Pass/Fail Kriterien

Der Test war erfolgreich, falls auf der Konsole ausgegeben wurde das ein Gebot zu einer Auktion erhalten wurde und nach Ablauf der Auktion ein Gewinner mit der Auftrags-ID ausgegeben wurde.

**Vorbedingung**

Das Spiel muss erfolgreich gestartet sein. Die Klasse **Auction** wurde so modifiziert, dass automatisch Gebote zu einer Auktion abgegeben werden.

**Einzelschritte**

Keine Einzelschritte erforderlich, da alles automatisch funktioniert.

**Beobachtungen / Log / Umgebung**

Auf der Konsole wird ausgegeben, dass zu einer Auktion ein Gebot erhalten wurde. Nach dem Ablauf der Auktion wird auf der Konsole die Auktions-ID und der Gewinner ausgegeben.

**Besonderheiten**

Keine.

**Abhängigkeiten**

$\langle T200 \rangle$, $\langle T500 \rangle$, $\langle T900 \rangle$

### Testfall $\langle T3200 \rangle$ - Klasse GameRegistry, GameAdministration

**Ziel**

Alle Teilnehmer des Spiels werden in der **GameRegistry** in den jeweiligen Listen „playerList“, „robotList“ oder „humanList“ gespeichert. Nur die bei Spielstart im Initialisierungsfenster festgelegten Daten für Industrien und Roboter werden übernommen bei. Durch das Befahren von Strecken sollen den Robotern Kosten entstehen. Der Preis für die Benutzung einer Kante ist gleich deren Gewicht.

**Objekte/Methoden/Funktionen**

**Objekte der Klassen**: GameRegistry, GameAdministration
**Methoden**: addPlayer(), addRobot(), addHuman(), makeCoordinates(), alignGrapgList(), industryToMap(), robotsToMap(), addCash(), updateRoboTable(), distributeData()

**Pass/Fail Kriterien**

Der Test war erfolgreich, falls die bei der Initialisierung des Servers hinzugefügten Teilnehmer alle in der **GameRegistry** registriert wurden, nur die bei der Initialisierung eingegebenen Industrien im Industriefenster stehen, die Industrien auf dem **GameDisplay** richtig angezeigt werden und falls dem Roboter für das Befahren einer Kante das Gewicht dieser in Rechnung gestellt wird.

**Vorbedingung**

Ein verbindungsbereiter Spieler hat das Client-Programm bereits auf seinen PC geladen. Das Initialisierungsfenster muss auf dem Server-PC offen sein. Aufträge müssen erstellt werden. Der Roboter muss einen Auftrag ersteigert haben und sich entlang der für den Auftrag berechneten Route bewegen.

**Einzelschritte**

1. Hinzufügen eines spielbereiten Roboters und des verbindungsbereiten Spielers.

2. Laden der Kartendatei.
3. Laden der Industriedatei.
4. Zuweisen des Roboters und des Spielers zu einem Team.
5. Starten des Spiels.
6. Gegebenenfalls als Benutzer einen Auftrag manuell erstellen.

**Beobachtungen / Log / Umgebung**

Während Roboter und Spieler im **InitFrame** hinzugefügt werden, werden die jeweilige Methoden „addRobot“ bzw. „addHuman“ und nachfolgend „addPlayer“ aufgerufen. Die Eingaben bei der Initialisierung sind zu vergleichen mit der Konsolenmeldung kurz vor Spielbeginn, die die finalen Array-Listen der **GameRegistry** ausgibt. Nachdem die Karten- und Industriedatei ausgewählt sind, meldet die Konsole, welche „Information“ die Industrie hat. Nach dem Serverstart sind Industrien und Roboter an den richtigen Positionen im **GameDisplay**. Mit jeder von dem Roboter empfangenen Knotenfreigabe wird der Fall „Knotenfreigabe“ der Methode „distributeData“ ausgeführt und dem Konto des Roboters dadurch jedes Befahren einer Kante negativ angerechnet. Die Methode „updataRoboTable“ aktualisiert im Sekundentakt die Robotertabelle der GUI(Server). Zu beobachten ist der angezeigte Kontostand des Roboters während er der Route folgt.

**Besonderheiten**

Keine.

**Abhängigkeiten**

$\langle T100 \rangle$, $\langle T200 \rangle$, $\langle T400 \rangle$, $\langle T1300 \rangle$, $\langle T1500 \rangle$

### Testfall $\langle T3300 \rangle$ - Klassen MapAdministration, Graph

**Ziel**

Feststellen ob nur zusammenhängende Graphen eingebunden werden und ob Industrien und Roboter auf der Karte positioniert werden.

**Objekte/Methoden/Funktionen**

**Objekte**: MapAdministration, Graph

**Methoden**: makeCoordinates(), alignGraphList(), setLocation()

**Pass/Fail Kriterien**

Der Test war erfolgreich, falls nur zusammenhängende Graphen eingebunden werden und die dementsprechende Ausgabe auf der Konsole kommt. Außerdem werden Industrie und Roboter auf der Karte gesetzt und mit „Industrie gesetzt“ bzw. „Roboter gesetzt“ auf der Konsole bestätigt.

**Vorbedingung**

Es wird eine Instanz der Klasse **MapAdministration** initialisiert. Das Initialisierungsfenster muss offen sein.

#### Einzelschritte

1. MapAdministration bekommt nicht zusammenhängenden Graph.
2. MapAdministration bekommt zusammenhängenden Graph.
3. Industrieobjekt wird übergeben.
4. Roboterobjekt wird übergeben.

#### Beobachtungen / Log / Umgebung

In einer geeigneten Testumgebung (getrennt von anderen Komponenten) wird eine Instanz der Klasse **MapAdministration** initialisiert. Bei Eingabe eines nicht zusammenhängenden Graphen als Graphobjekt wird auf der Konsole ausgegeben „Graph nicht zulässig, da nicht zusammenhängend!“. Wenn die **MapAdministration** einen zusammenhängenden Graphen als Graphenobjekt bekommt wird dies mit der Meldung „Alles Knoten markiert“ auf der Konsole bestätigt. Der Graph, der eingegeben wurde wird auf der Konsole ausgegeben und hat keine negativen Koordinaten mehr. Nachdem das Industrieobjekt übergeben wird soll auf der Konsole eine Ausgabe kommen „Industrie gesetzt“. Nachdem das Roboterobjekt übergeben wird kommt die Ausgabe „setLocation(0): NXT_19 gesetzt auf Knoten 0“.

#### Besonderheiten

Keine.

#### Abhängigkeiten

Keine.

### Testfall $\langle T3400 \rangle$ - Klassen IndustyAdministration

#### Ziel

Die automatische Auftragserstellung soll versuchen, Aufträge mit einer Industrie als Ziel erstellen, falls diese Industrie die Produkte einer anderen Industrie verarbeiten kann. Wenn ein Auftrag nicht versteigert worden ist, soll die reservierte Ware wieder für neue Auktionen freigegeben werden.

#### Objekte/Methoden/Funktionen

**Objekte**: IndustryAdministration
**Methoden**: searchForOrder(), returnOrderToIndustry()

#### Pass/Fail Kriterien

Wenn ein Auftrag erstellt wird, ist der Test erfolgreich.

#### Vorbedingung

Die Industriedaten müssen von der **GameAdministration** nach der Initialisierung des Servers korrekt an die Industrieverwaltung übermittelt worden sein. Zumindest zwei mit der Industrie-XML-Datei geladene Industrien hängen mit Ihrer Produktion von einander

ab. Falls keine entsprechende Warenmenge am Warenausgang der liefernden Industrie gesetzt wurde, können vom Benutzer Aufträge manuell erstellt werden und über die Schnittstelle der **GameAdministration** an die Auktionsverwaltung weitergegeben werden.

**Einzelschritte**

1. Warten auf die automatische Auftragserstellung.
2. Setzen des Wertes für die maximale Anzahl von Aufträgen von Robotern auf 0.
3. Starten des Spiels.
4. Warten auf die automatische Auftragserstellung bzw. manuelles Erstellen eines Auftrages durch den Benutzer.

**Beobachtungen / Log / Umgebung**

Zu Beobachten ist die Konsolenausgabe der Klasse **IndustryAdministration**. Die Methode „searchForOrder“ meldet eine gefundene Übereinstimmung des Wareneinbzw. Warenausgangs zweier Industrien. Falls es dann zur Auftragserstellung kommt, wird dies ebenfalls über die Konsole gemeldet. Die Konsole zeigt außerdem die Attribute der Startindustrie einmal vor und einmal nach Anpassung der Warenmengen an. Zu vergleichen sind hier die Werte „resOut“ und „outAmount“.

**Besonderheiten**

Keine.

**Abhängigkeiten**

$\langle T1700 \rangle$

# Glossar

### Actionpoint

Actionpoints sind Knoten und die senkrecht auf dem Wegenetz aufgeklebten Markierungen.

### Arbitrator

Der Arbitrator entscheidet in Abhängigkeit der Reihenfolge und der definierten Methode takeControll in jedem Behavior, wann welcher Behavior aktiv wird.

### Auftrag

Der Auftrag setzt sich aus Auftragnehmer, Auftraggeber, der zu transportierenden Ware, Start- und Zielpunkt sowie dem zu erwartenden Lohn zusammen.

### Behavior

Ein Behavior definiert Aktionen, die bei bestimmten Geschehnissen passieren sollen.

### Benutzer

Menschlicher Anwender, der auf den Server zugreift. Der Benutzer (Admin) verwaltet den Server. Der Benutzer kann (muss aber nicht) Spieler sein.

### Broadcast

Die identische Nachricht an alle teilnehmenden Roboter.

### Deadlock

Beim Deadlock können die Roboter nicht mehr ihrer Aufgabe nachkommen, weil sie sich gegenseitig behindern. Ein Deadlock ist also ein zu vermeidender Spielzustand.

### FCFS-KI

Diese KI arbeitet nach dem Prinzip first-come, first-served, nimmt also immer den zuerst zur Verfügung stehenden Auftrag an.

### GUI

Die GUI ist die grafische Benutzeroberfläche für den Server und den Spielerclient.

### Kartendaten

Mit den Kartendaten sind die Informationen über das Wegenetz gemeint, welches als Graph implementiert wird.

### Konflikt

Ein Konflikt tritt auf, wenn eine Straße von einem Roboter versperrt ist. Andere Roboter können diese zur selben Zeit nicht befahren.

### Künstliche Intelligenz (KI)

Das Programm auf den Robotern, welches das Handeln des Roboters festlegt.

### NXT

NXT ist ein Roboterbausatz der Firma Lego.

### Ressourcen

Unter Ressourcen fallen Industrien, Waren und der Kraftstoff für die Roboter.

### Roboter

Die Roboter arbeiten jeweils mit einer NXT-2.0-Einheit der Firma Lego und stellen die Transportmittel für die Waren dar.

### Server

Der Server dient hier als Koordinationsschnittstelle und übernimmt somit die Aufgabe des Spielleiters.

### Spieler

Menschlicher Anwender, der auf die Spieleroberfläche (nicht den Server) zugreift.

### Spieleroberfläche

Graphische Benutzeroberfläche für den Spieler, um einerseits das Spielgeschehen zu beobachten, andererseits aktiv am Spiel teilzunehmen.

### Spielzeit

Die Spielzeit bezeichnet den Zeitraum, der nach Initialisierung des Spiels und dessen Start bis zum Erreichen des Spielziels abläuft.