Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

Masterarbeit

Entwicklung einer Java Simulationsumgebung für Lego Mindstorms NXT Roboter

Abschlussarbeit im Fach Informatik zur Erlangung des akademischen Grades

Master of Education

Hamburg, den 08. März 2016

Pamina Maria Berg

Lehramt an Gymnasien, M.Ed.
pamina_berg@freenet.de
Matr.-Nr. 6087438
Fachsemester 09

Erstgutachter	Dr. Guido Gryczan
Zweitgutachter	Prof. Dr. Maria Knobelsdorf
Betreuer	Till Aust, Fredrik Winkler

Abstract

Simulation environments are a long established part of scientific research in robotics. Before risking fragile equipment to be damaged in real life testing, a simulation based on the specific prerequisites will help approximating the expected results.

In this master thesis a Java-based simulation environment for LEGO MINDSTORMS NXT robots is developed. The environment is developed for schools and has the ability to make the basic tasks of a NXT robot visible. The requirements and restraints of such an environment will be exemplified and completed by the subsumption of the topic in the academic curriculum. Furthermore, an elaborated description of the software architecture will be presented. In conclusion, the implementation results and further demands on the simulation environment, which have been developed through interviewing computer science teachers, will be described.

Zusammenfassung

Simulationsumgebungen sind im Bereich der wissenschaftlichen Arbeit mit Robotern bereits fest etabliert. Bevor Testungen unter Realbedingungen stattfinden, in der fragiles Equipment beschädigt werden könnte, werden diese mithilfe von Simulationen so gut wie möglich approximiert.

In dieser Masterarbeit wird eine Java-basierte Simualtionsumgebung für LEGO MINDSTORMS NXT Roboter entwickelt. Diese ist für den Einsatz in Schulen konzipiert und kann die grundlegenden Aufgaben eines NXT Roboters simuliert darstellen. Es werden Anforderungen und Einschränkungen einer solchen Umgebung erläutert und es erfolgt die curriculare Einordnung des Themas in den Bildungskontext Schule. Darüber hinaus ist eine ausführliche Beschreibung der Softwarearchitektur in dieser Arbeit enthalten. Abschließend werden die Ergebnisse der Implementation sowie weitere Anforderungen dieser Simulationsumgebung, die sich aus der Befragung von Informatik-Lehrkräften ergaben, zusammengefasst dargestellt.

Abbildungsverzeichnis

Inhaltsverzeichnis

Kapitel 1

Einleitung

„Informatik in der Schule“ ist ein bildungspolitisch umfassend diskutiertes Thema der letzten Jahre. Eine zentrale Rolle nimmt dabei die Frage ein, ob es sinnvoll sei, ein Unterrichtsfach „Programmieren“ ab der Grundschule einzuführen oder ob es reicht, dass die Schülerinnen und Schüler dazu ausgebildet werden einen Computer bedienen zu können. Dabei stehen sich die Grundgedanken der beiden Extreme, nämlich die Schülerinnen und Schüler als kompetente Computer-Nutzer mit einem Verständnis der logischen Strukturen hinter der Benutzeroberfläche auf der einen und das Heranziehen einer Generation von Schülerinnen und Schülern, die auf die Verwendung der gängigen Office-Anwendungen geschult wurde, auf der anderen Seite, gegenüber.

In der gymnasialen Oberstufe ist das Erlernen einer Programmiersprache derzeit schon vorgesehen. Doch auch ab der fünften Klasse kann bereits mit dem Erlernen von objektorientierten Programmierparadigmen begonnen werden. Eine wertvolle Ergänzung des klassischen Unterrichts hat sich in

den letzten Jahren aus der Entwicklung der Ganztagsschulprogramme herauskristallisiert: Im Nachmittagsbereich wurden nun neben sportlichen und musisch-künstlerischen Angeboten sowie klassischer Nachhilfe Roboter-Arbeitsgemeinschaften gegründet, in denen Schülerinnen und Schüler an die Technik von verschiedensten programmierbaren Robotern herangeführt werden.

Auf spielerische Art und Weise sollen Schülerinnen und Schüler (im Folgenden SuS abgekürzt) mit einfachen Konstrukten der objektorientierten Programmierung umzugehen lernen. Der Grundgedanke hinter diesem Konzept reicht bis in die Siebzigerjahre des letzten Jahrhunderts: Der Mathematiker, Informatiker und Psychologe Seymour Papert hat schon damals herausgefunden, dass das Ziel, Kinder zum Programmieren zu bringen, mithilfe einer spielerischen Herangehensweise erreicht werden könnte. Hierdurch entstand das bis heute bekannte Konzept der *Turtle* (vgl. [Nie99, S.365]).

Heutzutage wird der spielerische Aspekt beim Programmieren im Zusammenhang mit LEGO MINDSTORMS Robotern zum einen mit Drag-and-Drop Softwareangeboten wie das standardmäßig mit ausgelieferte *NXT Mindstorms Tool* (s. **3.1**) oder auch *Enchanting* (s. **3.2**) realisiert. Zum anderen bietet sich ab der Mittelstufe (Klasse 7 – 10) die Arbeit mit BlueJ zur Erstellung erster selbstgeschriebener Programme an. Hierzu kann eine BlueJ Extension genutzt werden, die mit der Java Virtual Machine *leJOS NXJ* für NXT Roboter (s. **3.3.2**) arbeitet.

Immer wieder stoßen Lehrkräfte in den Schulen auf Hardwareprobleme jeglicher Art, wie zum Beispiel eine unzureichende Anzahl an Robotern im Unterricht oder auch fehlende Firmwareupdates oder defekte Sensoren.

Auch das Zusammen- und wieder Auseinanderbauen der Roboter ist ein Aufwand, der für den alltäglichen Unterricht nicht geeignet ist. Um kleine Aufgaben – wie zum Beispiel das Fahren einer Kurve oder das Anhalten auf einem bestimmten Punkt – mit Java zu lösen, muss bisher immer erst der Roboter gestartet, der Code in BlueJ verfasst und auf den Roboter übertragen werden. Schließlich muss der Roboter noch zu einem geeigneten Parcours gebracht werden, um die Lösung testen zu können.

Um diese Probleme zu umgehen und den Einstieg in die objektorientierte Programmierung mit LEGO MINDSTORMS Robotern zu optimieren, soll im Rahmen dieser Masterarbeit eine Simulationsumgebung für die Arbeit mit LEGO MINDSTORMS NXT Robotern entwickelt werden. Im Zusammenhang mit dieser Arbeit wurden die beiden folgenden Forschungsfragen herausgearbeitet, die einen gedanklichen Leitfaden und die Basis für die Verknüpfung von theoretischen Überlegungen und Implementation darstellen:

> Welche Anforderungen werden an eine solche Simulationsumgebung gestellt – aus Schülerperspektive, aus Lehrerperspektive, aus Entwicklerperspektive?
>
> Welche Integrationsmöglichkeit bietet BlueJ und wie kann diese sinnvoll genutzt werden?

Es wird zunächst die Ausgangssituation im schulischen Kontext beschrieben sowie ein Einblick in die Arbeit mit LEGO MINDSTORMS Robotern im Unterricht gegeben. Anschließend werden die bisher verfügbaren Softwarelösungen zur Programmierung von LEGO MINDSTORMS

NXT Robotern vorgestellt, insbesondere die bisherigen Versuche, Simulationsumgebungen zu realisieren.

Vor der Beschreibung der entwickelten Software werden zudem die an die Neuimplementation gestellten Anforderungen vorgestellt. Im letzten Kapitel wird – nach der Vorstellung der Software – die tatsächliche Funktionalität der entwickelten Simulationsumgebung anhand dieser Anforderungen verifiziert.

Abschließend werden weitere, von in diesem Bereich aktiven Lehrkräften gestellte Anforderungen an die Simulationsumgebung sowie Verbesserungen und Erweiterungsmöglichkeiten, die sich im Laufe der Implementation entwickelt haben, erläutert.

Kapitel 2

Ausgangssituation

2.1. Das LEGO Mindstorms NXT System

2.1.1. Geschichte der LEGO Robotik

Die ersten computergesteuerten LEGO Produkte wurden bereits 1986 veröffentlicht. In einer Zusammenarbeit von LEGO Education und dem Massachusetts Institute of Technology (MIT) wurde LEGO TC LOGO entwickelt. Dies war eine spezielle Abwandlung der Programmiersprache LOGO, mit der zusammengesetzte LEGO-Modelle gesteuert werden konnten (vgl. [Rol14]).

Die Entwicklung eines programmierbaren LEGO-Steins begann 1988 und erreichte ihren Höhepunkt mit der Vorstellung des ersten MINDSTORMS Systems im Januar 1998, bei der der LEGO MINDSTORMS RCX INTELLIGENT BRICK – ein Microcomputer und somit das Kernstück

des RCX-Systems – und das *Robotics Invention System* im Museum of Modern Art in London vorgestellt wurden.

Bereits zwei Monate nach Verkaufsstart wurde die FIRST LEGO League (FLL) gegründet – eine Zusammenarbeit zwischen LEGO und FIRST (For Inspiration and Recognition of Science and Technology), die den Grundstein für die heute noch bestehende Wettbewerbsliga legte (vgl. [Rol14]).

Die Vorstellung und der Verkaufsstart der Nachfolge-Roboter des RCX-Systems, den LEGO MINDSTORMS NXT Robotern, fand im August 2006 statt. Diese damals neu entwickelten Roboter sind dank eines Updates in 2009 fast zehn Jahre später noch in den Schulen und Universitäten zu finden.

Im April 2005 fand die erste FLL Weltmeisterschaft in Atlanta, Georgia, statt und bis heute bieten die Weltmeisterschaften einen Anlaufpunkt für Jugendliche auf der ganzen Welt, die ihr Können in Bezug auf ihre Roboter auf die Probe stellen wollen (s. [Lego]).

2.1.2. Hardware

Zentraler Bestandteil der LEGO MINDSTORMS NXT Roboter ist der sogenannte NXT-Stein (s. Abb. 2.1). Dieser besteht aus einem 32 Bit ARM-Prozessor und einem 8 Bit Co-Prozessor. Der Hauptprozessor ist für die Ausführung des Hauptprogramms zuständig, während sich der Co-Prozessor um die Auswertung etwaiger Sensordaten kümmert, die dann an den Hauptprozessor weitergeleitet werden. Für die Kommunikation mit dem NXT-Stein stehen zwei Komponenten zur Verfügung. Zum einen eine Kabelverbindung mit einer USB 2.0 Schnittstelle, zum anderen bietet

der NXT-Stein – wie schon der Vorgänger RCX – die Möglichkeit einer Kommunikation über Bluetooth an. Das Softwaremenü wird auf dem 100 x 64 Pixel großen LCD-Bildschirm angezeigt und kann über die vier Kontrolltasten auf der Oberseite des NXT-Steins bedient werden. Für eine ausreichende Stromversorgung wird entweder mittels Batterien oder eines Akkus gesorgt (vgl. [Ber10, S.42]).

Abbildung 2.1: Der NXT-Stein [Ber10, S. 42]

Der NXT-Stein verfügt standardmäßig über drei Motoranschlüsse (*Motorports*) und vier Sensoranschlüsse (*Sensorports*), die jeweils mit Buchstaben (*A, B, C* bei den Motoren) bzw. Nummern (*1, 2, 3, 4* als Sensoranschlüsse) bezeichnet sind (vgl. [Ber10, S.43]). An zwei der drei verfügbaren Motorports ist jeweils ein Elektromotor angeschlossen. Diese beiden Motoren sorgen für den Antrieb des Roboters. In den Motoren sind Rotationssensoren und Regler eingebaut, die dafür sorgen, dass sowohl die Drehgeschwindigkeit als auch die Umdrehungszahl mithilfe des Programmcodes kontrolliert werden können (vgl. [Ber10, S.45ff.]).

Für die NXT Roboter sind eine Vielzahl von Sensoren verfügbar, damit die SuS ihren Roboter sein Umfeld erkunden lassen können. Die für die vorliegende Arbeit wesentlichen Sensoren werden nachfolgend noch beschrieben werden. Eine umfassende Übersicht der einzelnen Sensoren und deren spezifischen Funktionsweisen geben BERNS und SCHMIDT (s. [Ber10, Kapitel 4.2]).

2.2. Die Arbeit mit LEGO-Robotern im Unterricht

Die Robotik als Teilgebiet der Informatik in der Schule gewinnt zunehmend an Bedeutung. Nicht nur die Programmierung steht im Vordergrund, sondern auch die Kompetenz, Probleme unter bestimmten Aufgabenstellungen zu lösen. BERNS und SCHMIDT, die Verfasser eines der Kernwerke für den Unterricht mit dem LEGO MINDSTORMS System, schreiben, dass bei der Arbeit mit Robotern im Unterricht „durch eine konkrete Aufgabenstellung, das problemspezifische Konstruieren und Programmieren eines Roboters sowie durch das letztendliche Testen der gesamte Ablauf eines Problemlösungsprozesses kennen gelernt [wird]“ [Ber10, S.2].

In diesem Zusammenhang können auch, wie von WAGNER ausführlich begründet, die Kooperationsfähigkeiten der SuS im besonderen Maße gefördert werden. Gründe hierfür bestehen unter anderem in der Vielzahl der Möglichkeiten, ein Problem mit Robotern zu lösen, die alle unterschiedlich gut geeignet sind und durch welche die SuS zur Kommunikation und dem Verständlichmachen ihrer Ideen im Team angeregt werden. Hierbei ist das Augenmerk auch auf die Kontrollinstanz zu legen – nimmt sonst die Lehrkraft diese Rolle für sich ein, so ist der Roboter bei dieser Art von Un-

terricht eine neutrale, unbestechliche und jederzeit verfügbare Möglichkeit, die geleistete Arbeit zu bewerten (vgl. [Wag05, S.6f.]).

SCHREIBER fasst in seiner Examensarbeit den Einsatz von LEGO MINDSTORMS Robotern wie folgt zusammen:

> „Als herausragende Qualität des Materials LEGO-Mindstorms erwies sich die damit erzielbare **Motivation**, welche ein hohes Maß an Selbsttätigkeit auch über einen längeren Zeitraum möglich machte. Diese Selbsttätigkeit ist die Form des idealen Lernen im Sinne der Handlungsorientierung und hat sich auch in dem durchgeführten Unterricht bewährt. [...] Die **Sozialform** der Partnerarbeit erwies sich als weit gehend produktiv, zum einen, weil inhaltlich miteinander beraten werden konnte und wurde [...], zum anderen, weil ein LEGO-Mindstorms-Roboter auch für zwei gleichzeitig tätige Personen genug Handlungsmöglichkeiten bietet“ [Sch04, S.47f.].

Ähnlich beschreibt es auch STOLT in seiner Ausarbeitung:

> „Ein großer Punkt, der für das Roboterlabor spricht, ist das starke Motivationspotential für die Teilnehmer sich mit Informatikthemenstellungen zu befassen. Die Entwicklung einer eigenen Lösung einer Aufgabe, mit anschliessendem Wettbewerb besitzt trotz – oder vielleicht gerade wegen – der möglichen Komplexität einen sehr großen Unterhaltungswert und stellt eine spannende Herausforderung dar. Pädagogisch ist das Roboterlabor durch seine teamorientierte Arbeitsweise und die Möglichkeit zum explorativen Lernen interessant. Didaktisch

> steht hier das Erlernen und Erfahren von Programmier- und Designmethoden im Vordergrund“ [Sto01, S.5f.].

Grundsätzlich ist also zu sagen, dass der Einsatz von Robotern im Unterricht sowohl didaktisch als sinnvoll gesehen wird, als auch die Motivation der SuS in diesem Zusammenhang einen starken Einfluss auf den Lernerfolg und die Selbstständigkeit haben kann.

Es erfolgt nun zunächst die curriculare Einordnung des Themas Roboter im Unterricht in die Bildungspläne aus Hamburg. Danach werden verschiedene Arbeitsmethoden im Zusammenhang mit dieser Art von Unterricht vorgestellt.

2.3. Curriculare Einordnung

Die fachlichen Inhalte, die mithilfe des Einsatzes von Robotern im Unterricht vermittelt werden können, sind sowohl im Bildungsplan der Sekundarstufe I für das Gymnasium (Klasse 7 – 10) und die Stadtteilschule (Klasse 7 – 11), als auch im Bildungs- bzw. Rahmenplan für die Gymnasiale Oberstufe, der Sekundarstufe II, verankert. Der folgende Abschnitt gibt eine Übersicht über die verschiedenen Module im Zusammenhang mit den Unterscheidungen zur Schulform und Klassenstufe.

Grundsätzlich ist vorab zu erwähnen, dass die im Folgenden vorgestellten Bildungspläne ausschließlich für die Freie und Hansestadt Hamburg gelten. Zudem existiert jeweils ein Bildungsplan für die Sekundarstufe I der weiterführenden Schulen (Stadtteilschule und Gymnasium) sowie ein einheitlicher Bildungsplan für die Gymnasiale Oberstufe, wobei der elfte Jahrgang sowohl im Bildungsplan der Stadtteilschule, als auch im Bildungsplan für die Gymnasialen Oberstufe vertreten ist.

2.3.1. Gymnasium Sekundarstufe I (7 – 10) Informatik Wahlpflichtfach

Der Einsatz von Robotern kann auf die Module 2 und 3 des Bildungsplanes für die Sekundarstufe I des Gymnasiums bezogen werden und stellt einen möglichen Inhalt zu dem Erwerb und der Förderung der festgelegten Kompetenzen dar. Im Bildungsplan werden diese durch die folgenden Module untergliedert:

Modul 2

- Abläufe analysieren und umgangssprachlich beschreiben, Algorithmen formalisieren und mit einer formalen Sprache implementieren
- Umgang mit einer einfachen Entwicklungsumgebung
- Testen, Ergebnisse interpretieren und bewerten
- Grundlagen der prozeduralen Programmierung: Sequenz, Alternative, Wiederholung, Prozedur bzw. Funktion

Modul 3 – Kontext Daten und Prozesse

- Grundlagen der prozeduralen Programmierung
- Abläufe formalisieren
- Algorithmen mit einer formalen Sprache implementieren
- Testen, Ergebnisse interpretieren und bewerten [HH11]

2.3.2. STS Jahrgangsstufen 7 –11 Informatik Wahlpflichtfach

Ähnlich ist es im Bildungsplan für die Stadtteilschulen, welcher – anders als der eben aufgeführte Bildungsplan der gymnasialen Sekundarstufe I – auch die 11. Klasse umfasst. Hierbei werden zudem innerhalb der Module verbindliche Inhalte formuliert.

So sind im Modul 2 die Inhalte *Algorithmen* und *prozedurale Programmierung: Sequenz, Alternative, Wiederholung, Funktion* zu finden, die, wie der Bildungsplan vorschlägt, in das Unterrichtsvorhaben *"Wir lassen Roboter*

arbeiten" integriert werden können (vgl. [HH14]). An dieser Stelle findet demnach ein klarer Bezug zum Einsatz von Robotern im Unterricht statt.

Im Modul 3 sollen im zweiten Halbjahr die folgenden verbindlichen Inhalte unterrichtet werden:

- Grundlagen der prozeduralen Programmierung
- Abläufe formalisieren
- Algorithmen mit einer formalen Sprache implementieren
- Testen, Ergebnisse interpretieren und bewerten [HH14]

2.3.3. Gymnasiale Oberstufe – Vorstufe

In der Vorstufe sollen sich die SuS mit den Inhalten des Moduls *Daten und Prozesse* beschäftigen. Diese beinhalten das Analysieren und umgangssprachliche Bescheiben von Abläufen, das Strukturieren von Daten sowie die Verwendung von Variablen und Parametern, das Formalisieren von Abläufen, die Grundlagen der prozeduralen Programmierung, die Implementation von Algorithmen mit einer formalen Sprache, das Testen, Interpretieren und Bewerten von Ergebnissen (vgl. [HH09]).

2.3.4. Gymnasiale Oberstufe – Studienstufe

Die Studienstufe der gymnasialen Oberstufe bezieht sich auf die letzten beiden Schuljahre sowohl des Gymnasiums als auch der Stadtteilschule.

In der Studienstufe ist im Bildungsplan unter anderem der verbindliche Inhalt *Objektorientierte Modellierung* aufgeführt. Dieser umfasst die folgenden Punkte.

- Idee des OO-Konzepts mit Objekten und ihrer Kommunikation, Vererbung und Nutzerbeziehung
- Erarbeitung der Sprachelemente der verwendeten objektorientierten Programmiersprache, Berücksichtigung von Programmierkonventionen, Nutzen von Bausteinen/Libraries
- Nutzung einer IDE mit UML-Diagrammen und Quellcode zur schrittweisen Implementierung eines Informatiksystems. [HH09]

Insgesamt ist festzustellen, dass die Arbeit mit Robotern im Informatikunterricht keine Beeinträchtigungen durch den Bildungsplan erfährt. Eher kann aufgrund der Bildungspläne der universelle Einsatz von Robotern als Hilfestellung für die SuS dienen, denn die Lehrkräfte können anhand von Praxisbeispielen leicht erkennbare Verknüpfungen zwischen den einzelnen Modulen, Themen und Anforderungsbereichen aufzeigen. Somit können in der 7. Klasse gelernte Inhalte durch Erinnern des Kontexts, in dem diese erarbeitet wurden, später leichter aufgegriffen werden.

2.3.5. Einsatz von Robotern als Hilfsmittel im Unterricht

Da laut Bildungsplan die projektorientierte Arbeit in Informatik in der Studienstufe einen bedeutenden Stellenwert hat, bietet es sich an, eine

Projektarbeit zum Thema *Einsatz von Robotern* mit Bezug auf verschiedene Ligen (schulisch und universitär) zu gestalten, bei der als Produkt eine bestimmte Aufgabe mit den Robotern erledigt werden soll. Diese projektorientierte Arbeit hat den Vorteil, dass sie an einen realen Kontext geknüpft oder in ihn eingebettet werden kann. Wie HUBWIESER in seinem Werk *Didaktik der Informatik* verdeutlicht, kann dies einen positiven Einfluss auf die innere Einstellung der SuS gegenüber diesem teilweise recht anspruchsvollen Thema haben:

> „Nach den Erfahrungen mit der Umsetzung abstrakter Konzepte [...] im Mathematikunterricht des Gymnasiums scheint eine Vermittlung der naturgemäß abstrakten informatischen Lerninhalte nur dann erfolgversprechend, wenn durch konkrete, anschauliche Problemstellungen eine erhöhte Aufnahmebereitschaft der Schüler geschaffen wird “ [Hub07, S.68].

Die einfachste Methode um SuS an die problemorientierte Arbeitsweise im Rahmen der Programmierung mit Robotern heranzuführen ist der Einsatz von Fallbeispielen. Hierbei handelt es sich um aus der Realwelt gegriffene Umgebungen, die in unserem Fall mithilfe des Roboters erkundet werden sollen. Ein klassisches Fallbeispiel für den Einstieg wäre die Situation, dass der Roboter vor einem Hindernis steht, um das er herumfahren soll. Hierbei kann man den SuS vorgeben, dass sie z.B. im Viereck um das Hindernis herum navigieren sollen.

Ein weitaus komplexeres Beispiel bietet sich im Zusammenhang mit der RoboCup Wettbewerbsliga *Rescue* an, in der ein in einem Labyrinth verstecktes Opfer gerettet werden soll. Hierbei würden sich nun die SuS damit beschäftigen, wie sie durch das Labyrinth navigieren, wie sie Hindernissen

ausweichen, und welche bautechnischen Spezifikationen ihr Roboter haben muss, um das Opfer an einen sicheren Ort zu transportieren.

Der gemeinsame Konsens aller dargestellten Möglichkeiten des Robotereinsatzes im Unterricht ist der ihnen zugrunde liegende lerntheoretische Ansatz des *Konstruktivismus*. Im Rahmen projektorientierter Arbeit kann Wissen als Rohmaterial vorbereitet werden, so dass SuS dieses individuell und in Interaktion mit dem Lerngegenstand aufbauen. Bedeutend ist hierbei, dass die SuS ihrem spezifischen Vorwissen entsprechend handeln und somit eine persönlich individuelle Verknüpfung zu dem neu konstruierten Wissen herstellen. Der Unterricht folgt als Resultat dieses Ansatzes einem Paradigma des Problemlösens, bei dem alle SuS individuell aktiv sind und und die Lehrkraft in der Rolle des Moderators oder Lernberaters zur Seite steht (vgl. [Schwa07, S.219f.]).

Kapitel 3

Bisher verfügbare Softwarelösungen

In diesem Kapitel wird eine Auswahl von verfügbaren und im Unterricht getesteten Softwarelösungen zur Programmierung der NXT Roboter sowie einige bereits verfügbare Simulationsumgebungen vorgestellt.

3.1. LEGO Mindstorms NXT

Das LEGO MINDSTORMS NXT-Programm ist eine von der Firma LEGO zur Verfügung gestellte Programmieroberfläche für NXT Roboter.

Die Abbildung 3.1 zeigt den Startbildschirm der Entwicklungsumgebung. Dieser stellt neben zwei Videoanleitungen als Hilfe für Anfänger auch das

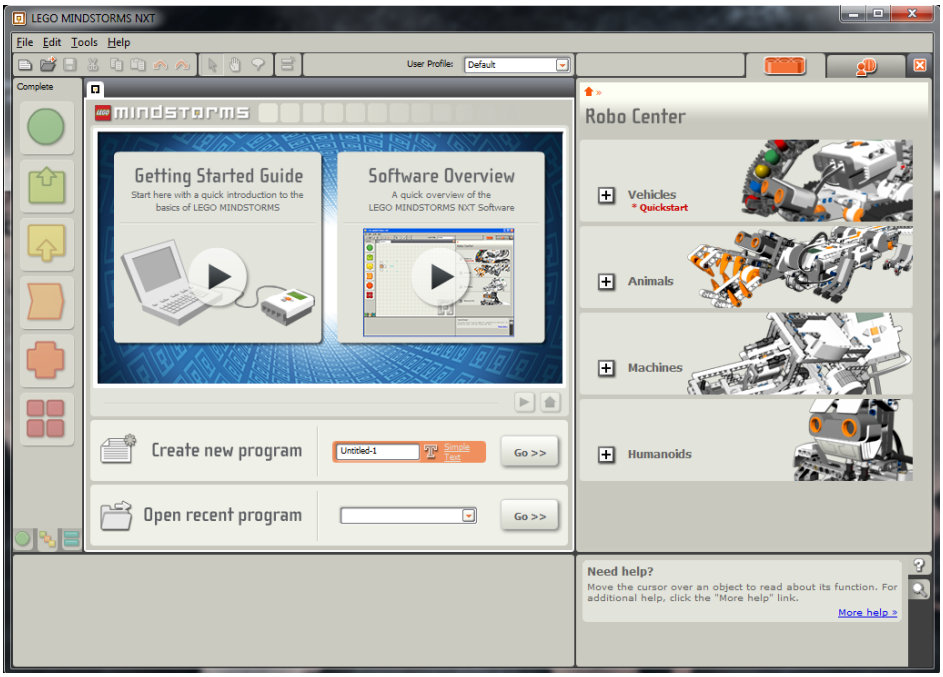


Abbildung 3.1: Der Startbildschirm des LEGO NXT-Programms

Robo Center zur Verfügung, in dem verschiedene von LEGO vorgeschlagene Roboter-Modelle vorgestellt werden, die mithilfe des Bausatzes des NXT zusammengesetzt werden können.

Sobald eine neue oder bestehende Programmdatei geöffnet wird, können die SuS ihren Roboter programmieren. Hierbei helfen ihnen Blöcke, die unter anderem der Bewegung und Sensorik dienen und dabei einfache Steuerungsmechanismen sowie Elemente der Regelungstechnik zur Verfügung stellen. Zur Programmierung werden Bausteine aus der linken Leiste per Drag-and-Drop auf die karierte Oberfläche gezogen. Es können ineinander geschachtelte Schleifen erstellt werden sowie if-Abfragen mit

zwei Fallunterscheidungen. Nahezu beliebig viele Bausteine lassen sich so miteinander kombinieren (vgl. Abb.3.2).

Der Vorteil an dieser Entwicklungsumgebung besteht darin, dass sie von LEGO zur Verfügung gestellt wurde, und somit alle Funktionen von Haus aus mitbringt, die dem NXT-Stein technisch zur Verfügung stehen. Zudem muss es keine Veränderungen an der Firmware des NXT-Steins geben, was eine Zeitersparnis bei der Vorarbeit der SuS mit sich bringt.

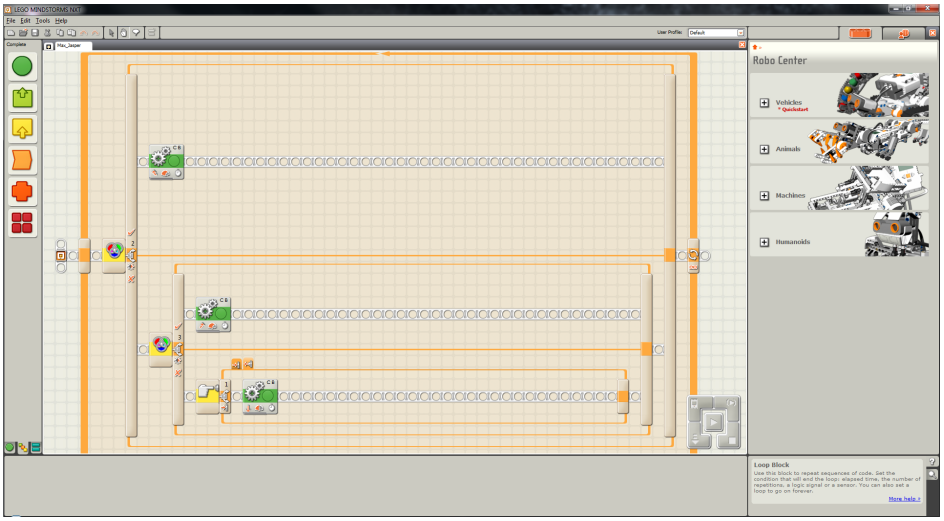


Abbildung 3.2: Fahren eines Roboters auf einer schwarzen Linie

3.2. Enchanting

Enchanting ist eine an das Einsteigertool für objektorientierte Programmierung *Scratch* anknüpfende Entwicklungsumgebung. Wie das in **3.1**

beschriebene Programm wird auch hier mithilfe von Drag-and-Drop gearbeitet. Da Enchanting keine von LEGO nativ unterstützte Entwicklungsumgebung für NXT Roboter ist, muss hier eine Änderung an der Firmware vorgenommen werden. Hierbei wird die ausgelieferte Standardfirmware mit leJOS (s. **3.3.2**) ersetzt. Dies sorgt dafür, dass der NXT nun auf die Programmierung mit Java reagiert.

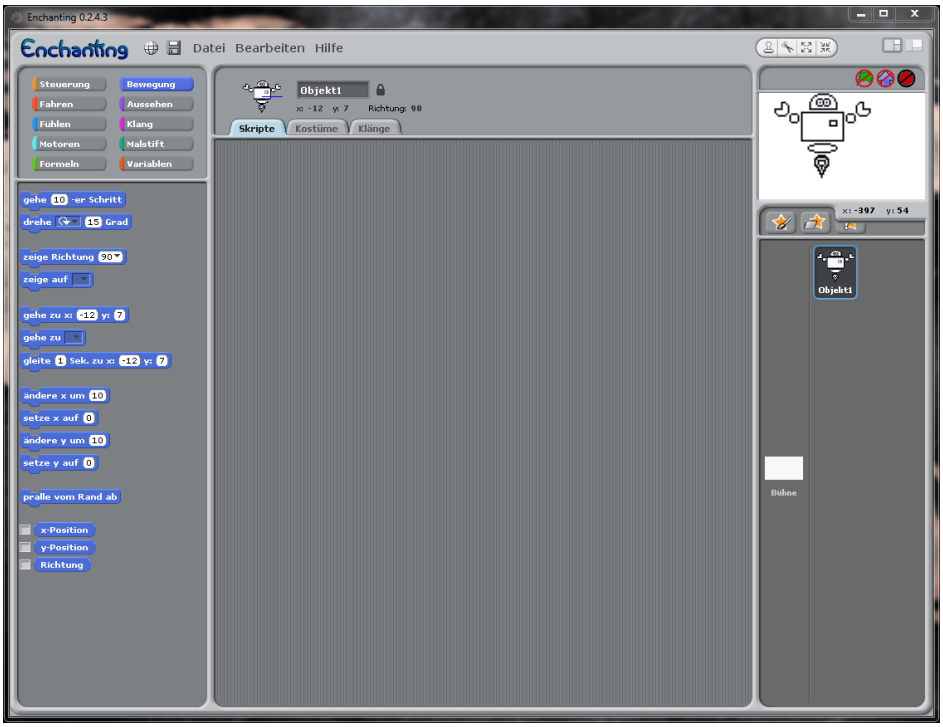


Abbildung 3.3: Der Startbildschirm von Enchanting

Wird Enchanting von den SuS geöffnet, erscheint zunächst der Startbildschirm wie in Abbildung 3.3. Nun kann entweder ein neues Programm erstellt oder ein bestehendes geöffnet werden. Die verschiedenen Program-

mierelemente werden nach Kategorien in der oberen linken Ecke sortiert aufgeführt. Für die SuS sind hierbei die Kategorien *Steuerung*, *Fühlen*, *Motoren* und *Formeln* essentiell. Die SuS lernen beim Programmieren mit Enchanting, dass sie ihre Motoren und Sensoren zunächst über die verschiedenen Ports referenzieren müssen. Dies geschieht, indem man beispielsweise einen Motor-Block an einen Port setzt und ihm einen aussagekräftigen Namen gibt. Auf diese Weise kann nun – anders als beim LEGO MINDSTORMS NXT-Programm – bei Kontrollstrukturen mithilfe des Namens auf den Motor oder Sensor zugegriffen werden.

In Abbildung 3.4 ist ein simples Programm zum Fahren entlang einer schwarzen Linie mithilfe von zwei Lichtsensoren dargestellt. Hierbei ist zu erkennen, dass die Sensoren in den Abfrage-Blöcken mit `rechts` und `links`, die Motoren in den türkisfarbenen Elementen mit `NXT rechts 1` sowie `NXT links 2` bezeichnet sind. Es kann somit von den SuS direkt aufgezeigt werden, welcher Sensor angesprochen wird und welcher Motor an welchem Port angeschlossen ist.

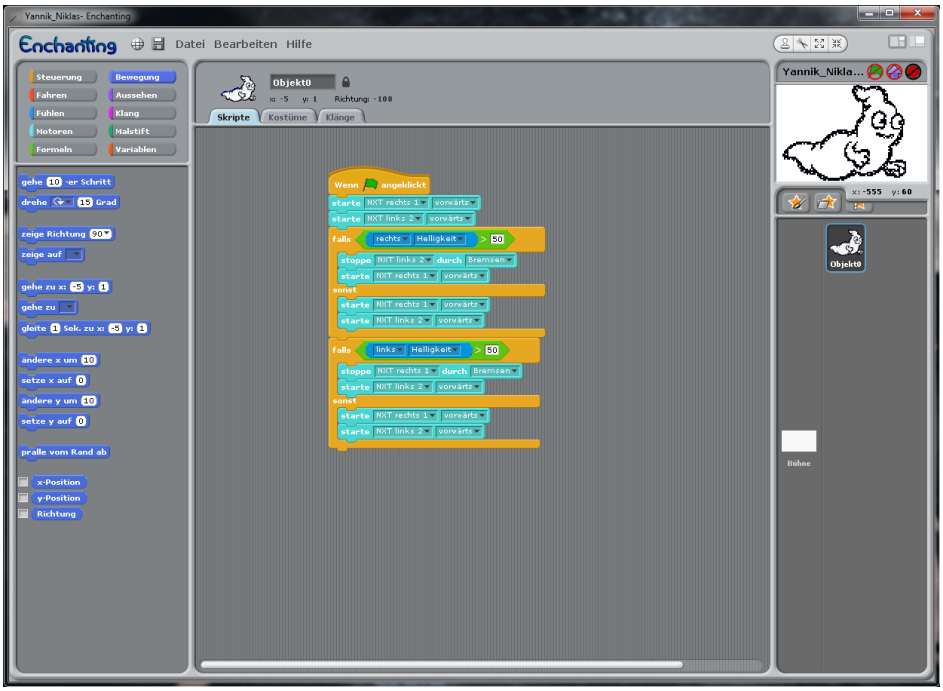


Abbildung 3.4: Beispiel eines Enchanting-Programms

3.3. Schriftliche Programmierung von NXT Robotern

Neben den grafischen Entwicklungsumgebungen aus den Abschnitten **3.1** und **3.2** stehen natürlich auch Werkzeuge zur schriftlichen Programmierung der NXT Roboter zur Verfügung. Nachfolgend wird eine der geläufigen Kombinationen aus Entwicklungsumgebung und virtueller Maschine für LEGO MINDSTORMS NXT vorgestellt.

3.3.1. BlueJ

Die Java-Entwicklungsumgebung BlueJ wurde an der Monash University in Australien entwickelt. Das Ziel von BlueJ war von Beginn an klar definiert: Es sollte eine einfache Umgebung für den Einstieg in die objektorientierte Programmierung geschaffen werden (vgl. [Bar03, S.14]).

Es handelt sich jedoch bei der Entwicklung objektorientierter Programme mit BlueJ keinesfalls um die Benutzung einer reduzierten Version von Java. BlueJ läuft, wie andere Entwicklungsumgebungen, auf dem aktuellen Java Development Kit (JDK). Auch als Compiler und virtuelle Maschine (JVM) wird Software der Firma Oracle (bis 2010 Sun Microsystems) verwendet (vgl. [Bar03, S.15]).

Inzwischen wird nicht nur in den Universitäten BlueJ als Werkzeug zur Einführung in die Programmiersprache Java und die objektorientierte Programmierung genutzt. Auch im Schulkontext hat die intuitive Bedienung und übersichtliche Gestaltung Anklang gefunden, da BlueJ „eine einfache Entwurfssicht für die Analyse gegebener Lösungen und für die Planung neuer Lösungen“ [Ehm09, S.6] zur Verfügung stellt. Eine weitere Besonderheit besteht darin, dass – im Gegensatz zu den meisten anderen Entwicklungsumgebungen – in BlueJ „schnell einige Objekte erzeugt und sofort untersucht werden [können]“ [Ehm09, S.6]. Hierzu bietet BlueJ die *Inspect*-Funktion an, die über das Kontextmenü eines erzeugten Objekts erreichbar ist und die Werte der Feldvariablen genau dieses Exemplars einer Klasse anzeigt.

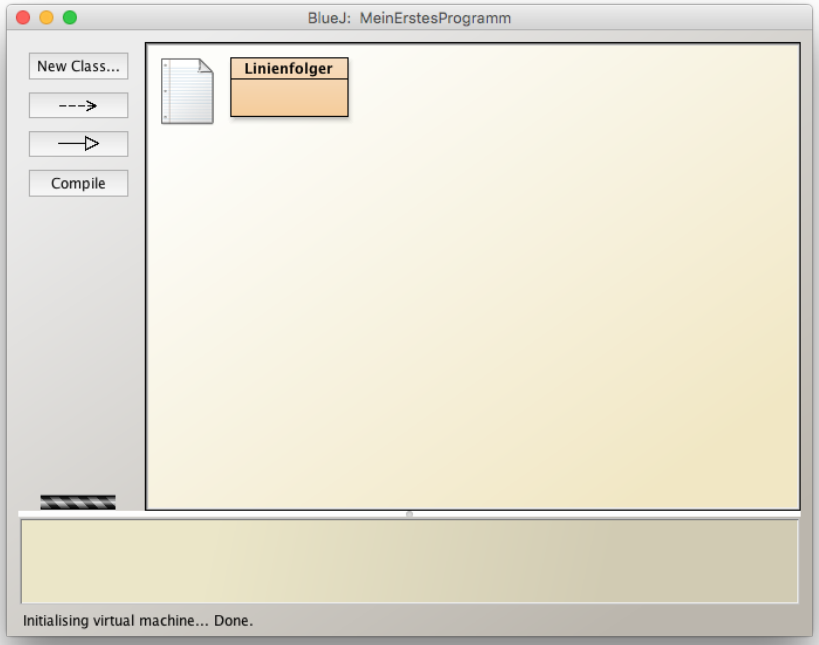
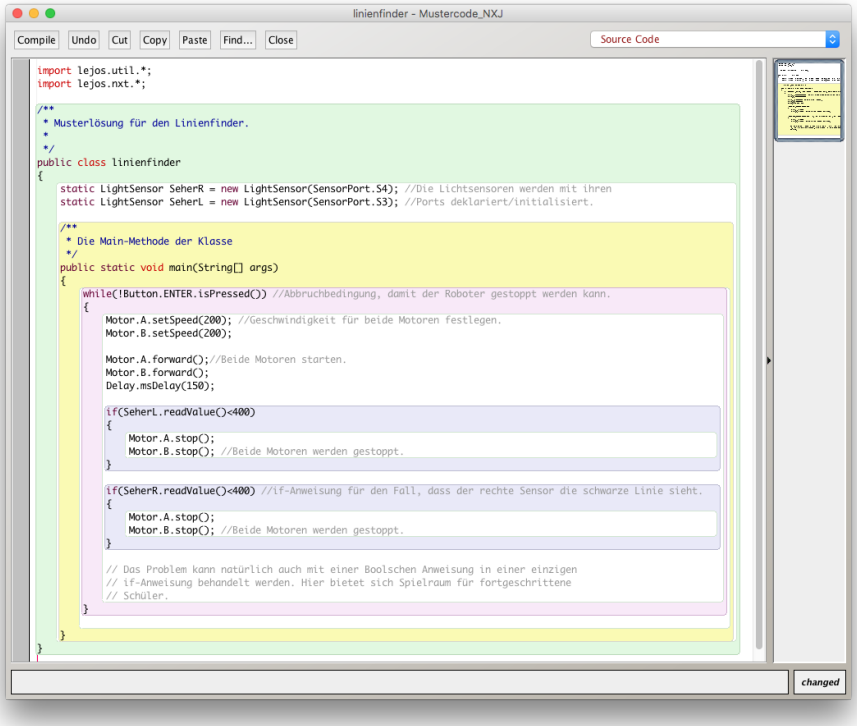


Abbildung 3.5: Die Benutzeroberfläche von BlueJ

3.3.2. leJOS

leJOS ist eine kleine Java Virtual Machine und stellt alle Klassen der NXJ API zu Verfügung, mithilfe derer LEGO MINDSTORMS NXT Roboter in Java programmiert werden können (vgl. [leJOS]).

In der Kombination mit BlueJ ergibt sich somit für die SuS eine besonders einfache Softwarelösung. Durch eine eigens für BlueJ und NXT Roboter geschriebene sogenannte *Extension* (siehe hierzu [Bow12]) und den Import




```
import lejos.util.*;
import lejos.nxt.*;

/**
 * Musterlösung für den Linienfinder.
 *
 */
public class linienfinder
{
    static LightSensor SeherR = new LightSensor(SensorPort.S4); //Die Lichtsensoren werden mit ihren
    static LightSensor SeherL = new LightSensor(SensorPort.S3); //Ports deklariert/initialisiert.

    /**
     * Die Main-Methode der Klasse
     */
    public static void main(String[] args)
    {
        while(!Button.ENTER.isPressed()) //Abbruchbedingung, damit der Roboter gestoppt werden kann.
        {
            Motor.A.setSpeed(200); //Geschwindigkeit für beide Motoren festlegen.
            Motor.B.setSpeed(200);

            Motor.A.forward();//Beide Motoren starten.
            Motor.B.forward();
            Delay.msDelay(150);

            if(SeherL.readValue()<400)
            {
                Motor.A.stop();
                Motor.B.stop(); //Beide Motoren werden gestoppt.
            }

            if(SeherR.readValue()<400) //if-Anweisung für den Fall, dass der rechte Sensor die schwarze Linie sieht.
            {
                Motor.A.stop();
                Motor.B.stop(); //Beide Motoren werden gestoppt.
            }

            // Das Problem kann natürlich auch mit einer Boolschen Anweisung in einer einzigen
            // if-Anweisung behandelt werden. Hier bietet sich Spielraum für fortgeschrittene
            // Schüler.
        }
    }
}
```

Abbildung 3.6: BlueJ-Beispiel zum Finden einer Linie

der NXJ API stehen den SuS nicht nur die Klassen zur Steuerung ihres Roboters zur Verfügung. Darüber hinaus besteht die Möglichkeit, über eine Erweiterung des Kontextmenüs um spezielle Befehle, das in BlueJ geschriebene Programm auf den NXT-Stein zu übertragen. Dies steht allen in BlueJ geschriebenen Klassen zur Verfügung, wie in Abbildung 3.7 sichtbar ist.

Mithilfe von BlueJ (oder einer anderen Entwicklungsumgebung) können die SuS unter anderem kontextorientierte Aufgaben lösen. Hierzu gehö-

ren zum Beispiel das Anhalten auf einer Linie oder auch den Roboter einer schwarzen Linie folgen zu lassen, was eine der Grundaufgaben im RoboCup Junior Rescue Wettbewerb darstellt. Einen Eindruck über den von den SuS zu schreibenden Code wird in den Abbildungen 3.6 und 3.8 verdeutlicht.

Der Vorteil an der Arbeit mit leJOS besteht darin, dass die SuS direkt mit Java-Code in Berührung kommen. Von Anfang an müssen sie auf syntaktische Korrektheit achten, damit ihr Programm überhaupt auf den NXT Roboter übertragen werden kann. Dabei stellen sich bereits mit wenigen zu lernenden Methoden schnell die ersten Erfolge ein: Für das einfache Fahren eines Vierecks müssen die SuS lediglich die Methoden `setSpeed()`, `forward()` und `backward()` an beiden Motoren sowie eine Verzögerung mithilfe der Klasse `Delay` benutzen.

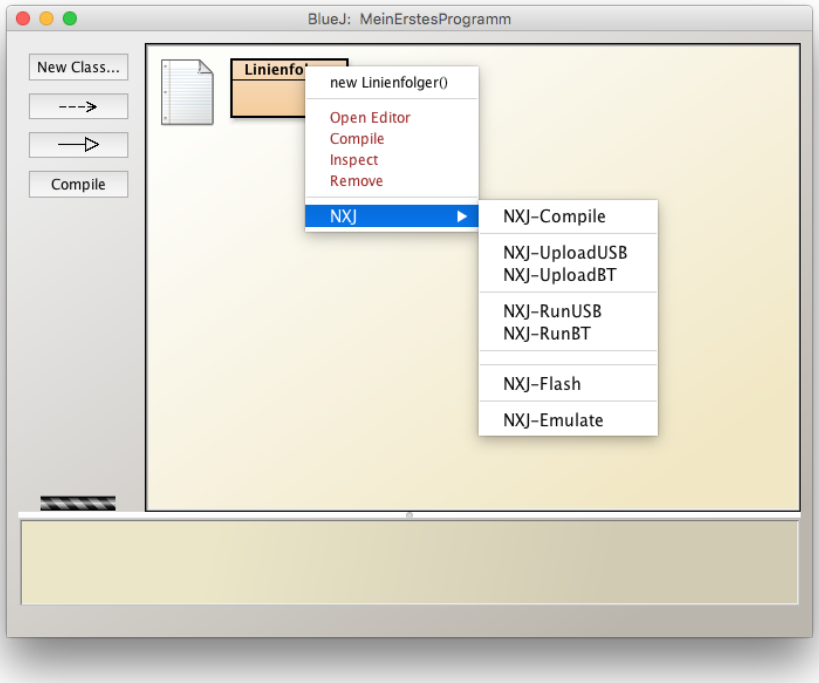


Abbildung 3.7: Die Kontextmenü-Erweiterung der NXJ-Extension

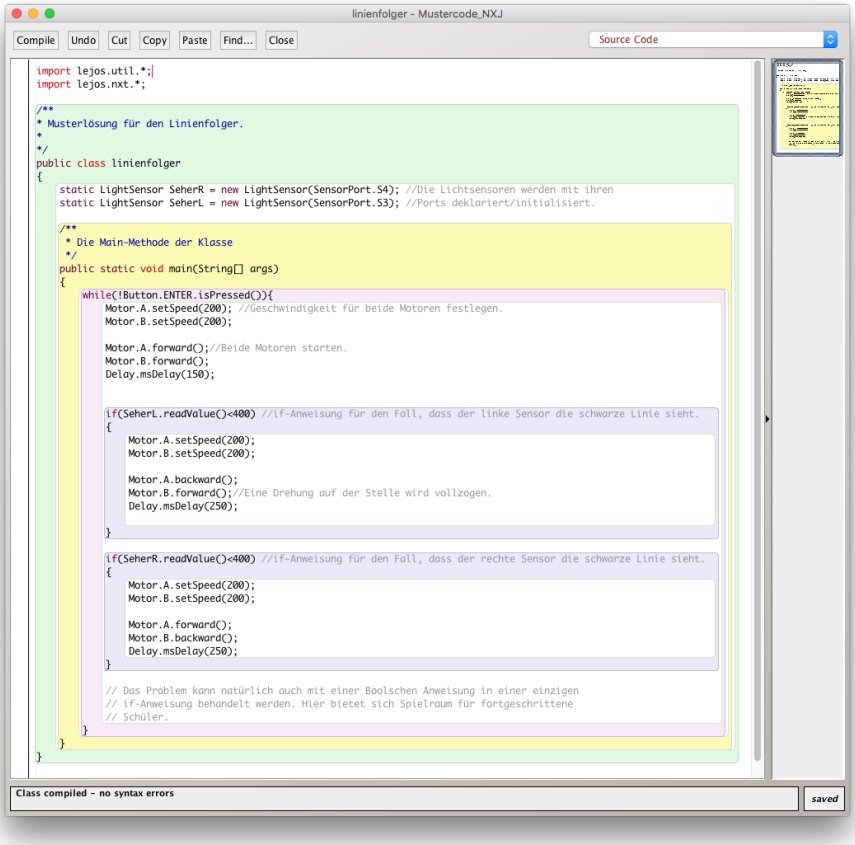


Abbildung 3.8: Der Roboter soll einer schwarzen Linie folgen

3.4. Simulationsumgebungen

Für den virtuellen Umgang mit LEGO MINDSTORMS Robotern jeder Generation gibt es bereits einige Simulatoren, die frei verfügbar sind. Zunächst sollte in diesem Zusammenhang eine an der RWTH Aachen genutzte Simulationsumgebung erwähnt werden, in der sich ein Roboter in einer virtuellen 3D-Umgebung bewegen kann (vgl. [RWTH]). Diese ist für die Benutzung der Windows-Entwicklungsumgebung *BricxCC* konzipiert, die unter anderem die an LEGO NXT Roboter angepasste Programmiersprache *Not eXactly C (NXC)* (s. [BricxCC]) unterstützt.

Des Weiteren existiert *nxcEditor*, eine sowohl mit Windows als auch mit Linux und MacOS nutzbare Entwicklungsumgebung, die den *nxcSimulator* beinhaltet. Ähnlich wie BricxCC nutzt auch der nxcEditor die Programmiersprache NXC (vgl. [nxcEditor]). Programmiert wurde die Entwicklungsumgebung von Frank Knefel und ist in Anlehnung an das vom Fraunhofer IAIS initiierte Lehr-/Lernkonzept *Roberta – Lernen mit Robotern* konzipiert. Deshalb wird die erwähnte Entwicklungsumgebung inzwischen auch direkt von Roberta als Softwarelösung angeboten (vgl. [Roberta]).

Abschließend ist noch das umfassende Softwareangebot des Virtuellen Campus Projekts der PHBern zu erwähnen. Dieses umfasst – neben der Entwicklung von Programmen für LEGO MINDSTORMS NXT Roboter mithilfe einer eigens für NXT Roboter geschriebenen Klassenbibliothek *NxtJLib* (s. [Aeg16]) – die Möglichkeit, das Programm auf verschiedene Weisen auf den NXT-Stein zu übertragen oder in einem Simulator auszuprobieren (s. [PHBern]).

Kapitel 4

Anforderungen an die Neuimplementierung

Das folgende Kapitel widmet sich der Untersuchung der in Kapitel **1** vorgestellten ersten Forschungfrage. Hierzu werden zunächst theoretische Überlegungen aus der Literatur präsentiert und diese abschließend mit der Frage nach den Anforderungen aus verschiedenen Perspektiven verknüpft.

4.1. Theoretische Verankerung

Nicht nur in der Schule, sondern insbesondere in der Wissenschaft spielt Simulation in der Robotik eine wichtige Rolle [Her12, S.13]. Die Anforderungen, die an die wissenschaftlichen Simulationsumgebungen der Robotik gestellt werden, fassen HERTZBERG, LINGEMANN und NÜCHTER wie folgt zusammen:

- „Die Umgebung muss hinreichend gut simuliert sein. Die Simulation einer Flughafenterminalhalle muss zum

Beispiel 'zufällig' umherlaufende Fluggäste mit Gepäck und Transportkarren umfassen.

- Der Roboter in seiner Funktionalität muss hinreichend gut simuliert sein. Das betrifft seine Aktionen wie auch seine Sensorik.

- Relevante Ungenauigkeit technischer Sensoren und Effektoren muss abgebildet werden. Kann zum Beispiel im realen Bild einer Kamera auf dem Roboter ein Orientierungspunkt im Gegenlicht der Fensterfront unsichtbar werden, muss die Simulation diesen Effekt reproduzieren.

- Die 'Wahrheit' im Simulator ist tabu! Natürlich ist im Simulator der Zustand jedes simulierten Objekts präzise bekannt, einschließlich der Position, Richtung und Geschwindigkeit des Roboters. Die Roboterkontrollsoftware darf hierauf nicht zugreifen, um Information über die Umgebung zu erhalten – das geht nur über die simulierten Sensoren. (Für die externe Bewertung des Roboterverhaltens ist der Vergleich zwischen der Wahrheit im Simulator und der Information in der Roboterkontrollsoftware aber erlaubt.)

- Der Simulator sollte für den simulierten Roboter die identische Schnittstelle wie der reale Roboter zwischen Roboterkontrollsoftware einerseits und Robotersensorik und -aktuatorik andererseits verwenden; die Roboterkontroll-

> software soll also code-identisch für den realen oder den simulierten Roboter verwendet werden.“ [Her12, S.14]

Im Hinblick auf die Entwicklung einer Simulationsumgebung für die Schule sind mehrere Perspektiven von Belang. In den folgenden beiden Unterkapiteln werden nun die Anforderungen an die Neuimplementierung eines Simulators für LEGO MINDSTORMS NXT Roboter aus Sicht der Schüler und der Lehrer dargestellt. Zudem wird geklärt, inwiefern die wissenschaftlichen Anforderungen aus dem universitären Bereich, wie oben dargestellt, auch auf die schulische Lehre zutreffen und wie diese gegebenenfalls angepasst werden müssen.

4.2. Schülerperspektive

Da es sich bei der Simulationsumgebung zunächst um einen prototypischen Ansatz handelt, der möglicherweise schon ab der fünften, regelmäßig aber ab der siebten Klasse eingesetzt werden soll, steht ein Augenmerk ganz besonders im Fokus: Die Einfachheit. Sowohl in der Bedienung des Simulators als auch im Aussehen sollten klare und leicht erkennbare Strukturen vorherrschen. Dieser Designaspekt steht im Zusammenhang mit dem *KISS*-Prinzip. *KISS* ist hierbei die Kurzform für *keep it short and simple* – den Leitfaden dieses Designansatzes (vgl. [Mos12, S. 144f.]).

Ferner sollte darauf geachtet werden, dass die SuS für die Programmierung des Roboters und die Benutzung des Simulators eine einheitliche Syntax verwenden können. Hierzu muss die Simulationsumgebung genau die Methoden anbieten, die auch bei dem realen Roboter die Steuerung der Motoren und den Zugriff auf Sensordaten ermöglichen. Dies entspricht

dem zweiten Aspekt nach HERTZBERG/LINGEMANN/NÜCHTER, da der simulierte Roboter alle für die Arbeit mit SuS zentralen Funktionen anbieten und sich hinreichend ähnlich wie ein Roboter in der Realwelt verhalten soll. Auch wird hiermit der Aspekt der code-identischen Roboterkontrollsoftware erfüllt, da die SuS die Möglichkeit bekommen sollten, dasselbe Programm für den Simulator, wie auch für die Steuerung des Roboters selbst zu verwenden.

Ein weiterer Aspekt, den HERTZBERG/LINGEMANN/NÜCHTER beschreiben sind die Parcours. Diese sollten möglichst realitätsgetreu umgesetzt werden. Das bedeutet, dass Ausschnitte eines bestehenden realen Wettbewerbsparcours genutzt werden könnten, um für die SuS eine möglichst realitätsnahe Testumgebung anzubieten. Der Vorteil realitätsnaher Wettbewerbsumgebungen besteht auch in der Vergleichbarkeit der Ergebnisse. Die SuS lernen gleich zu Beginn einen wichtigen Aspekt bei der Entwicklung von Programmcode für Roboter: Nur weil der Roboter im Simulator an einer schwarzen Linie entlangfahren kann, bedeutet dies nicht, dass auch der reale Roboter diese Herausforderung problemlos meistert.

4.3. Lehrkraft

Für Lehrkräfte ist es erfahrungsgemäß wichtig, dass die Simulationsumgebung alle essentiellen Bausteine der objektorientierten Programmierung im Kontext der LEGO Roboter zur Verfügung stellt. Hierzu gehört das Austesten des Fahrens mithilfe einer Schleife sowie der verschiedenen Sensoren. Dafür sollte es eine Auswahl an verschiedenen Parcours geben, damit der Fokus bei jedem Parcours auf einer einzigen Sache liegt. Sollen etwa die angebauten Lichtsensoren getestet werden, so bietet es sich an,

den Parcours einfach aus einem weißen Hintergrund und einer schwarzen Linie bestehen zu lassen, die entweder senkrecht in der Nähe des Roboters platziert ist oder als Kurve eine Teilstrecke des realen Labyrinths darstellt.

In diesem Zusammenhang sollte die Möglichkeit gegeben sein, dass Lehrkräfte unter bestimmten Voraussetzungen eigene, an ihren Unterricht angepasste Parcours in die Software einbinden können.

Vorrangig ist, dass sich Lehrerinnen und Lehrer schnell in die Simulationsumgebung einarbeiten können, um bei Fragen der SuS sofort Hilfe leisten zu können.

4.4. Erweiterbarkeit aus Entwicklerperspektive

Nun zu dem Aspekt der Erweiterbarkeit, der für die Entwicklung dieser Simulationsumgebung zentral ist – insbesondere da diese für den Einsatz in Schulen konzipiert ist.

Zunächst sollte die Software ausreichend kommentiert werden, damit auch nach Fertigstellung dieser Masterarbeit weitere Module hinzugefügt werden können. Dazu gehören insbesondere einige Arten von Sensoren, die in diesem Prototyp nicht realisiert wurden. Aber auch Modifikationen, die im Rahmen von Veränderungen der leJOS NXJ API stattfinden, müssen berücksichtigt werden können.

Zudem sollte die Software so strukturiert werden, dass gegebenenfalls auch Teile des Quellcodes für zukünftige Implementationen im Rahmen von

Erweiterungen oder Anpassungen genutzt werden können. Hierzu sollten Überlegungen zum Einsatz von Paketen und der sinnvollen Trennung zusammengehöriger Klassen getätigt werden.

Leider hat sich während der Entstehung dieser Simulationsumgebung herausgestellt, dass der Verkauf von LEGO MINDSTORMS NXT Robotern zum 31. 12. 15 eingestellt wurde. Dies bedeutet, dass im Hinblick auf die Entwicklung der Robotik in Schulen auch die Möglichkeit, die geschriebene API an die Technologie der LEGO MINDSTORMS EV3 – den Nachfolge-Robotern der NXT – anzupassen, essentiell ist.

Kapitel 5

Entwickelte Software

5.1. Implementationsablauf

Die Entwicklung des ersten Ansatzes der Simulationsumgebung umfasste mehrere Schritte sowie einige Vorarbeit. Hierzu gehörten die Erstellung von Skizzen und Mock-Ups sowie einer Übersicht über die leJOS API. Auch wurden Integrationsmöglichkeiten in BlueJ erörtert. Die einzelnen Elemente dieser Vorarbeit werden im folgenden Abschnitt kurz zusammengefasst.

5.1.1. Skizzen und Mock-Ups

Um eine grafische Struktur festzuhalten und das Ziel der Simulationssoftware zu erfassen wurden zunächst Skizzen angefertigt, die einen groben Überblick über die Benutzeroberfläche geben sollten. Diese enthielten eine

beispielhafte Darstellung eines Parcours und eines Roboters sowie eine Menüzeile am oberen Rand.

Als nächstes wurde eine spezielle Art von Interaktionsprototyp hergestellt, so genannte *interaktive Mock-Ups*, die eine Subkategorie interaktiver Wireframes darstellen. Interaktive Wireframes sind eine vereinfachte, digitale Darstellung von Benutzerschnittstellen, die einen gewissen Grad an Interaktionslogik umfassen (vgl. [Mos12, S.162ff.]).

Zur Erstellung der Mock-Ups wurde mit der Software *Axure RP Pro* gearbeitet, mit der sich sowohl einfache als auch komplex verschachtelte Mock-Ups erstellen lassen (s. Abb. 5.1).

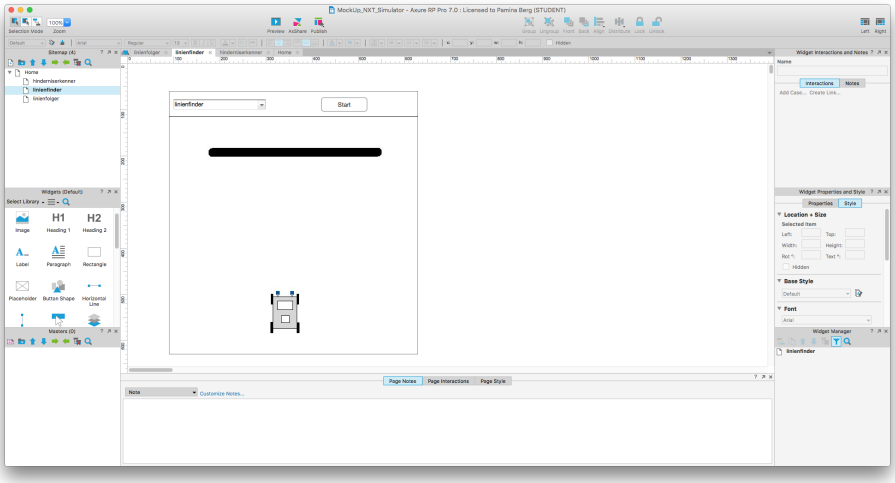


Abbildung 5.1: Die Bedieneroberfläche von Axure RP Pro

Die Mock-Ups sollten die grundsätzlichen Funktionen der Simulationsumgebung präsentieren. Dabei ist die Darstellung von Szenarien, die sich

aus dem Kontext Schule ergeben, von zentraler Bedeutung. Diese Szenarien sind an den Funktionsumfang der NXT Roboter und insbesondere an die Testung der Sensoren gekoppelt und umfassen das Anhalten auf einer schwarzen Linie wie in Abbildung 5.2, das Fahren entlang einer schwarzen Linie wie in Abbildung 5.3 und das Finden und Umfahren eines Hindernisses auf der Fahrbahn wie in Abbildung 5.4.

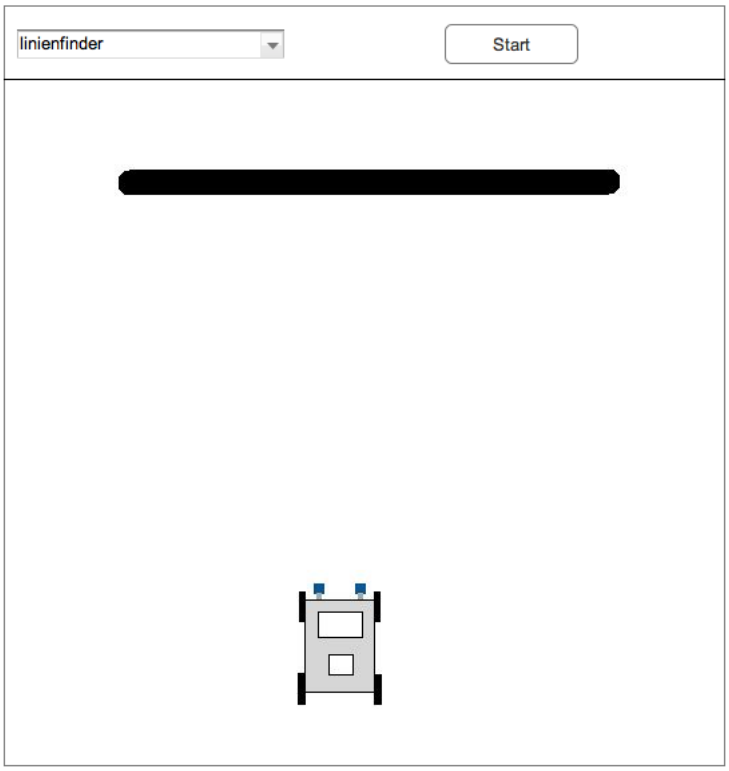


Abbildung 5.2: Anhalten auf einer schwarzen Linie

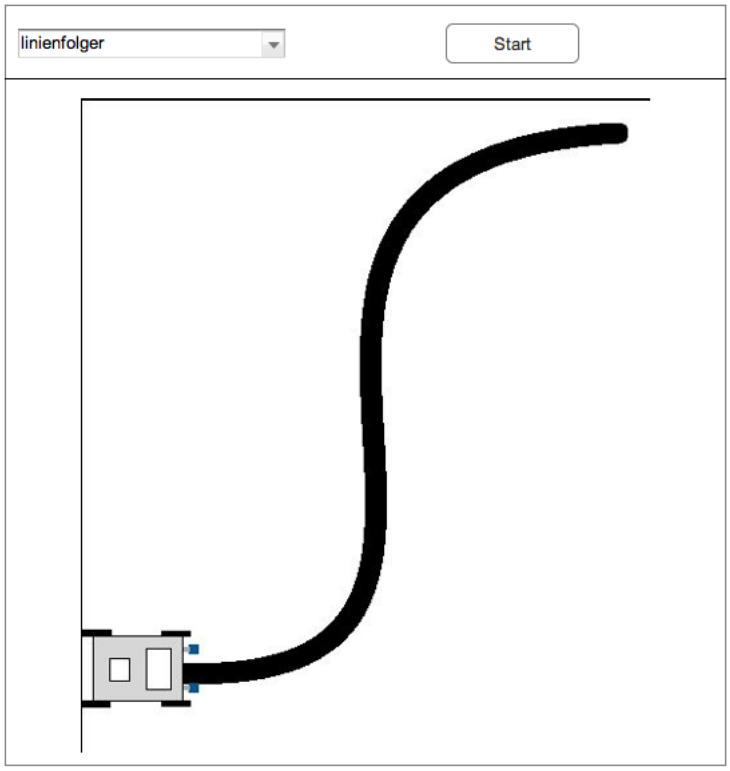


Abbildung 5.3: Fahren einer S-Kurve

5.1.2. Weitere Schritte vor Beginn der Implementation

Nach dem Erstellen der Mock-Ups wurde abgewägt, inwiefern die vorherigen Überlegungen umgesetzt werden konnten.

Zunächst konnte bestehender Code aus anderen Projekten in dieses Projekt importiert werden. So wurde die leJOS API – deren Klassen im Paket der Klassenbibliothek frei verfügbar sind – genutzt, um die Neuimplementation der in **5.3.2** beschriebenen Klassen realitätsgetreu miteinander zu verzah-

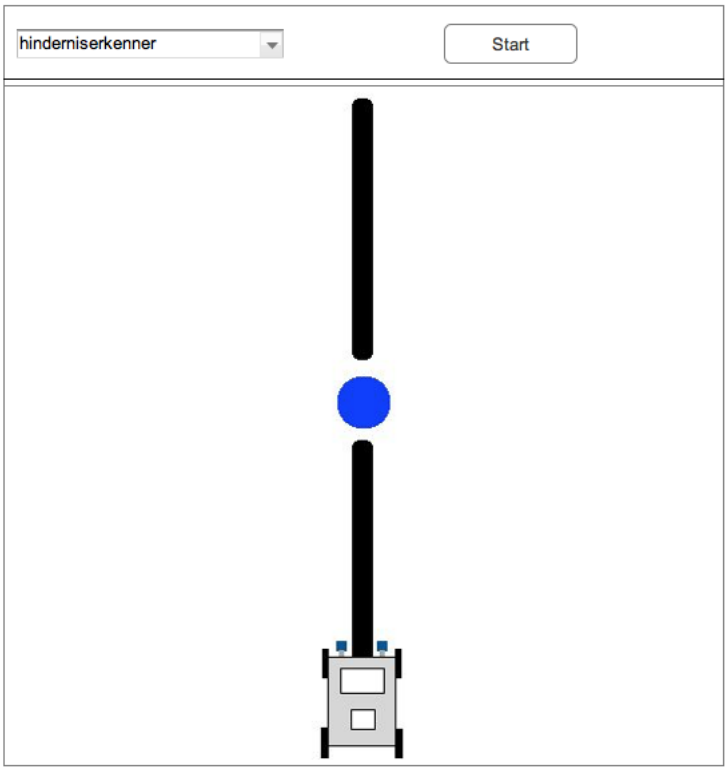


Abbildung 5.4: Erkennen eines Hindernisses auf der Fahrbahn

nen und die korrekten Beziehungen untereinander zu implementieren.

Darüber hinaus wurde die Simulationsoberfläche auf das Anzeigen des Parcours und eines als Dreieck dargestellten Roboters reduziert. Die in den Abbildungen 5.2 bis 5.4 erkennbare Menüleiste mit Startbutton am oberen Rand wurde demnach als Erweiterungsmöglichkeit in der Implementation zurückgestellt.

5.1.3. Integration in BlueJ

Nun zu der Betrachtung der zweiten Forschungsfrage und der derzeit umgesetzten Entscheidung über die Verknüpfung der Entwicklungsumgebung BlueJ mit der Simulationsumgebung.

Die Frage nach einer Integration in BlueJ kann auf vielfältige Art beantwortet werden. Die einfachste und zu diesem Zeitpunkt praktikabelste Lösung ist, die Integration der Simulationsumgebung über einen Import der Klassenbibliothek zu realisieren.

Für die SuS bedeutet dies, dass diese neben der Import-Anweisung für die leJOS-Bibliothek zwei weitere Import-Anweisungen in ihren geschriebenen Code einfügen müssen. Außerdem muss ein Exemplar der Simulator-Klasse erzeugt werden. Solche Anpassungen sind jedoch ein überschaubarer Aufwand für die Lehrkraft sowie die SuS. Diese können auch als „Schablone“ für die ersten Versuche von der Lehrkraft zur Verfügung gestellt und daher gut für den Unterricht genutzt werden.

Nun stellt sich die Frage, wieso überhaupt mit BlueJ gearbeitet werden soll, da es doch ausreichend andere Entwicklungsumgebungen gibt, in die solche Bibliotheken importiert werden können. Der Vorteil an BlueJ besteht darin, dass diese Entwicklungsumgebung, wie in **3.3** beschrieben, eine klar strukturierte und übersichtliche Benutzeroberfläche bietet. Die SuS müssen sich nicht in für ihre Ansprüche überdimensional umfangreiche Programmierumgebungen einarbeiten und können per Mausklick ihren Code per USB-Kabel auf den Roboter übertragen. Diese Übertragung wird von einer speziell für BlueJ entwickelten NXT-Extension realisiert

(vgl. [Bow12]), die eine Erweiterung des Kontextmenüs jeder Klasse zur Verfügung stellt.

5.2. Beschreibung der Software

Nun zu einer kurzen Beschreibung der Simulationsumgebung, in der zudem Einzelheiten zur Benutzung der Software dargelegt werden.

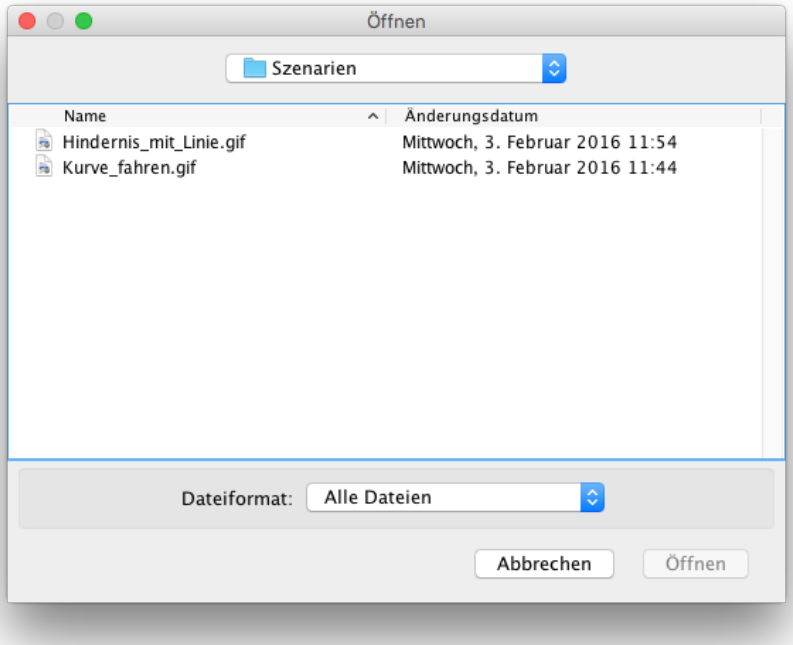


Abbildung 5.5: Auswahldialog für den Parcours

Beginnt man mit dem Programmieren einer Klasse – welche beispielsweise dafür sorgen soll, dass der NXT Roboter einer Kurve auf dem Boden folgt – so müssen neben des Imports für die leJOS API zwei weitere Import-Anweisungen für den Simulator eingebunden werden. Diese Klasse wird im Folgenden mit `linienfolger` bezeichnet. Zu beachten ist, dass immer entweder der leJOS-Import oder die Simulator-Importe auskommentiert sein müssen, da sonst weder der Simulator noch der reale NXT Roboter weiß, auf welche Bibliothek zugegriffen werden soll.

Um nun mit dem Simulator zu arbeiten, erzeugt man innerhalb der main-Methode der Klasse `linienfolger` ein neues Exemplar der Klasse `Simulator`. Nachdem man `linienfolger` fehlerfrei kompiliert hat, kann man in BlueJ über einen Rechtsklick die main-Methode ausführen lassen. Dies resultiert darin, dass mit der Erzeugung eines Simulators der Auswahldialog für den Parcours angezeigt wird, wie in Abbildung 5.5 beispielhaft dargestellt ist. Sobald der Parcours ausgewählt wurde, wird dieser in einem neuen Fenster geöffnet. Nun sehen die SuS den Roboter, der auf dem ausgewählten Parcours fährt (Vgl. Abb.5.6).

Alternativ kann die Erzeugung des Simulator-Exemplars auch in den Konstruktor ausgelagert werden. Statt einer main-Methode können die SuS in der Konsequenz eine beliebige Methode für die Ausführung ihres Quellcodes verwenden. Nach Erzeugung eines Exemplars `linienfolger`, bei der die Auswahl des Parcours nun direkt beim Aufruf stattfindet, kann danach an der Klasse die auszuführende Methode über das Kontextmenü ausgewählt werden. Die Simulationsoberfläche ist im Unterschied zum vorigen Ansatz schon vor Auswahl der zu testenden Methode sichtbar.

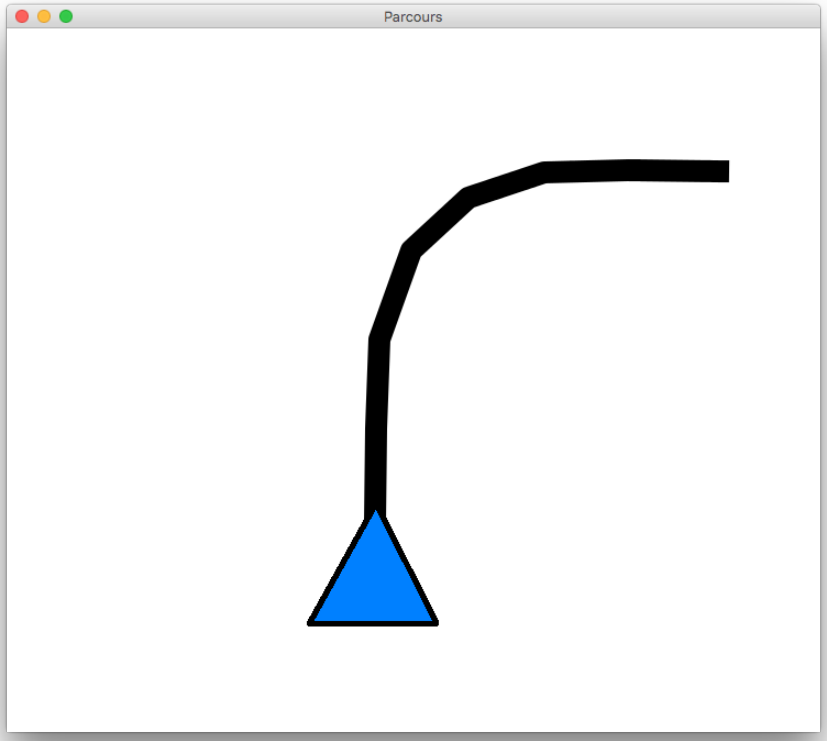


Abbildung 5.6: Eine beispielhafte Darstellung von Parcours und Roboter

5.3. Softwarearchitektur

Die entwickelte Software besteht – im derzeitigen Zustand – aus insgesamt fünfzehn Klassen. Diese sind zur Übersichtlichkeit in Pakete unterteilt, die sich in zwei Kategorien unterscheiden. Zum einen Klassen, welche die API für den Simulator zur Verfügung stellen und die zentralen von leJOS angebotenen Klassen für die NXT Roboter umfassen, zum anderen Klassen, die für die Repräsentation der Simulation auf dem Bildschirm zuständig sind. Erstere sind im Sinne des WAM-Ansatzes als *Services* zu sehen, die Klassen des Simulators als *Werkzeuge* (vgl. [Zül90, S. 61ff.], [Lil08, S. 35f.]).

Um eine Konsistenz bei den Imports für die SuS zu bewahren, sind die Klassen, welche die simulierten NXT-Teile implementieren, in einem Package mit der Bezeichnung `nxt` (ähnlich wie bei leJOS) enthalten, die Klassen des Simulators in dem Package `sim`.

Zum besseren Verständnis folgt im Weiteren eine kurze Zusammenfassung der in dieser Implementation verwendeten Projektstruktur.

Pakete

Pakete (engl.: packages) in Java fassen Gruppen von Typen zusammen. Ein Beispiel hierfür sind die Standardbibliotheken von Java, die in den Paketen `java` und `javax` zu finden sind. Diese enthalten Unterpakete, wie zum Beispiel die Bibliotheken zur Entwicklung grafischer Benutzeroberflächen *Swing* oder auch *AWT* (vgl. [Ull12, S.265]). Der Zugriff auf diese

Bibliotheks-Klassen erfolgt entweder über eine Pfadangabe bei der Variablendeklaration und -initialisierung oder über eine `import`-Anweisung. Will man beispielsweise ein Exemplar der Klasse `Point` aus der Java-Klassenbibliothek `AWT` erzeugen, so kann dies entweder durch den Befehl

```
java.awt.Point Punkt = new java.awt.Point();
```

statt der „normalen“ Deklaration und Initialisierung oder aber durch die Anweisung

```
import java.awt.Point;
```

zu Beginn des Quelltexts und dem gewohnten Erzeugen des Objekts durch

```
Point Punkt = new Point();
```

geschehen (vgl. [Ull12, S.266]). Möchte man eine ganze Reihe von Klassen eines Pakets importieren, so besteht die Möglichkeit, dies mithilfe eines * am Ende der Import-Deklaration zu tun. Im obigen Beispiel wäre dies

```
import java.awt.*;.
```

Anhand dieser Deklaration ist auch direkt zu erkennen, dass das Paket `awt` ein Unterpaket des Pakets `java` ist. Der „Paketpfad“ bzw. die hierarchische Struktur der Pakete und Unterpakete wird mithilfe von Punkten angegeben (vgl. [Ull12, S.265ff.], [Abt15, S.84ff.]).

5.3.1. Externe Frameworks

AWT

Die Klassenbibliothek AWT (*Abstract Window Toolkit*) stellt die grundlegenden Bausteine zur Erstellung grafischer Benutzeroberflächen zur Verfügung. Die Darstellung von AWT-Komponenten ist an die Darstellungsoptionen des jeweiligen Betriebssystems gekoppelt, wodurch eine Ein-

grenzung der Möglichkeiten auf diejenigen, die auf alle Betriebssysteme zutreffen, vorhanden ist. Vorteilhaft dabei ist, dass das Erscheinungsbild von Menüleisten und Schaltflächen dem Benutzeroberflächendesign des Betriebssystems entspricht (vgl. [Abt15, S.279])

Auf einem anderen Konzept bauen die Klassen des Pakets *Swing* auf.

Swing

Das Paket `javax.swing` basiert fast vollständig auf Java und ersetzt alle Grundkomponenten des AWT-Frameworks. Im Gegensatz zu AWT wird hier auf die Designkonsistenz zum Betriebssystem verzichtet. Im Gegenzug bekommt man ein einheitliches und plattformunabhängiges *Look & Feel* (vgl. [Abt15, S.279]). Dies ist darin begründet, dass die Swing-Komponenten mit „primitiven Zeichenoperationen gemalt“ werden [Ull12, S.1018].

ImageIO

Das Paket `java.imageio` bietet dem Benutzer Schnittstellen zur Einbindung von Bilddateien in Java an. Hierzu gehört unter anderem das Lesen und Schreiben von Bildern in unterschiedlichen Formaten [Ull12, S.1280]. Mithilfe der Methode `ImageIO.read(URL input)` wird ein Bild aus dem übergebenen Verzeichnis oder der URL direkt in das Projekt geladen.

5.3.2. Die Klassen des `nxt`-Pakets

In diesem Abschnitt werden nun die durch leJOS vorgegebenen und für die Implementation des Simulators angepassten Klassen vorgestellt. Hierbei handelt es sich um eine kleine Auswahl der essentiell für den Unterricht erforderlichen Klassen, die im Rahmen dieser Arbeit für die Simulationsumgebung angepasst wurden. Die interne Struktur zwischen den einzelnen Klassen ist durch die leJOS API vorgegeben und musste somit realitätsgetreu nachgebildet werden. Die Abbildung 5.8 stellt diese mithilfe eines UML-Diagramms dar.

Motoren

Zunächst wurde die Klasse `Motor` implementiert. Diese besteht lediglich aus den Feldern `A`, `B` und `C`, die Exemplare der Klasse `NXTRegulatedMotor` darstellen. Ein NXTRegulatedMotor ist ein an einen Motorport angeschlossener NXT-Motor. Ein solcher Motor wird über den Befehl

```
Motor.[port].[Methode]
```

angesteuert.

MotorPort

Die Klasse `MotorPort` ist in diesem Projekt eine verkleinerte Version der gleichnamigen Klasse aus der leJOS API. In ihr sind die statischen Variablen `A, B` und `C` enthalten, welche die zur Verfügung stehenden MotorPorts am Roboter darstellen. Intern ist jedem Port eine `id` in Form eines

Integer zugeordnet. Über die Methode `getInstance(int id)` kann über die private Variable `id` der öffentliche Port (`A, B` und `C`) abgefragt werden.

NXTRegulatedMotor

In der Klasse `NXTRegulatedMotor` befindet sich die auf die Simulationsumgebung angepasste Implementation der NXT-spezifischen Motoren. Den größten Anteil hat hierbei die Implementation der Bewegungen der beiden Motoren über die Methoden `forward()` und `backward()` sowie der Geschwindigkeitsregulierung über die Methode `setSpeed()`.

Jeder NXTRegulatedMotor ist an einen MotorPort angeschlossen, durch welchen eindeutig zuzuordnen ist, welcher Motor gerade angesprochen werden soll. Dies wird in der Implementation des Simulators durch eine Feldvariable `_port` gelöst, über die dann abgefragt werden kann, an welchem Motor gerade eine Methode aufgerufen wird. Folglich ist jedes NXTRegulatedMotor-Objekt ein Exemplar der Klasse NXTRegulatedMotor, welches vom Benutzer einen Port zugeordnet bekommt.

Zudem bekommt jeder Motor eine Feldvariable `_richtung`, die die Information enthält, ob sich der Motor bewegt (vorwärts oder rückwärts) oder ob er gerade in Ruheposition ist. Dabei entspricht „vorwärts“ dem Wert 1, „rückwärts“ dem Wert -1 und mit 0 wird der Ruhezustand bezeichnet.

Die SuS werden ihren geschriebenen Code meist damit beginnen, den beiden Motoren jeweils eine Geschwindigkeit zuzuweisen. Dies geschieht mithilfe der Methode `setSpeed(int speed)`. Da sich der Roboter auf dem Parcours pixelweise bewegt, wird an dieser Stelle die Geschwindigkeit von den für reale NXT Roboter geeigneten Geschwindigkeiten durch den

Faktor 100 geteilt, um eine angemessene Bewegungsgeschwindigkeit auf dem Bildschirm zu sichern.

Danach werden die Motoren in Bewegung versetzt. Hierzu werden die Methoden `forward()` und `backward()` benutzt. In der Implementation der Simulationsumgebung wird diese Bewegung als Veränderung der Position und der Ausrichtung der Roboter-Grafik auf der Leinwand realisiert. Je nachdem, welcher Port (*B* entspricht dem linken Motor, *C* dem rechten) angesteuert wird, ruft die Klasse NXTRegulatedMotor die Methode `aendereBewegung(double winkel, double geschwindigkeit)` – in Abhängigkeit zur Geschwindigkeit und eines Richtungsmultiplikators – am Roboter auf. Der Richtungsmultiplikator sorgt hierbei für eine möglichst realitätsnahe Approximation der tatsächlichen Veränderung der Ausrichtung des Roboters. Die Verwendung einer Variablen für diesen Faktor bringt den Vorteil mit sich, dass bei Verbesserungen innerhalb der Bewegungsimplementation der Motoren eine Konsistenz in allen involvierten Methoden sichergestellt ist. Soll beispielsweise die Vorwärtsbewegung der Motoren neu skaliert werden, so kann dies über die Veränderung des Richtungsmultiplikators geschehen. In Folge werden alle relevanten Werte in den Methoden gleichzeitig und gleichmäßig verändert.

Die Herausforderung bei dieser Simulationsumgebung bestand nun darin, dass die Motoren – auch wenn sie sich schon bewegen – eine neue Geschwindigkeit übergeben bekommen und die zuvor bestimmte Bewegung mit der neuen Geschwindigkeit ausführen. Außerdem sollte ein zweimaliges Aufrufen der Methoden `forward()` und `backward()` nicht dazu führen, dass der Roboter sich doppelt so schnell bewegt. Hierzu konnte nun die Information, die in `_richtung` enthalten ist, genutzt werden. Mit

einfachen if-Abfragen wird zu Beginn des Methodenrumpfs festgestellt, in welcher Bewegung sich der angesprochene Motor gerade befindet.

Licht- und Farbsensoren

Die Lichtsensoren sind zwei von der leJOS API vorgegebene Exemplare der Klasse `LightSensor`. Diese sind im Simulator derzeit jeweils links und rechts neben der Spitze des Roboter-Dreiecks angebracht (siehe hierzu Abb. 5.7). Die Berechnung hierzu sehen wie folgt aus:

Für die Bestimmung der Position des rechten Lichtsensors gilt:

Sei B die Breite, H die Höhe und u die Ausrichtung des Roboters. Seien mit *xPos* und *yPos* die Koordinaten des Robotermittelpunkts bezeichnet.

Dann gilt

$$\begin{aligned}
\beta &= \arctan\left(\frac{\frac{B}{4}}{\frac{H}{2}}\right) \\
\varepsilon &= 90 - (u + \beta) \\
c &= \left(\frac{B}{4}\right) \cdot \arcsin(\beta) \\
x_B &= xPos + (c \cdot \sin(\varepsilon)) \\
y_B &= yPos - (c \cdot \cos(\varepsilon)) \,.
\end{aligned}$$

Für die Berechnung der Position des linken Lichtsensors gilt analog:

Sei B die Breite, H die Höhe und u die Ausrichtung des Roboters. Seien mit *xPos* und *yPos* die Koordinaten des Robotermittelpunkts bezeichnet.

Dann gilt

$$\begin{aligned}
\beta &= \arctan\left(\frac{\frac{B}{4}}{\frac{H}{2}}\right) \\
\gamma &= u - \beta \\
c &= \left(\frac{B}{4}\right) \cdot \arcsin(\beta) \\
x_A &= xPos - (c \cdot \sin(\gamma)) \\
y_A &= yPos + (c \cdot \cos(\gamma)) \,.
\end{aligned}$$

Es folgt also für die beiden Punkte A und B in Grafik 5.7, dass $\mathsf{A} = (x_A|y_A)$ und $\mathsf{B} = (x_B|y_B)$.

Da es sich bei der Implementation um einen Prototyp handelt, existiert noch keine optische Repräsentation der beiden Sensoren am Roboter.

Die Klasse `Roboter` enthält die Methoden zur Abfrage der Positionen der Sensoren im Bezug auf die x- und y-Achse. Diese sind mit `gibXLicht-Rechts()`, `gibYLichtRechts()`, `gibXLichtLinks()` und `gibYLicht-Links()` bezeichnet und werden benötigt, um das Auslesen des Farbwerts des Pixel unter dem jeweiligen Sensor zu ermöglichen. Die eigentliche Abfrage des Helligkeitswertes passiert jedoch am Parcours-Objekt. In der zugehörigen Klasse steht die Methode `gibLichtmittelwert(int x, int y)` zur Verfügung, die auf das Bilddaten-Array zugreift (s. Abschnitt **5.3.3**).

Da die realen Sensoren auch nicht nur einen einzelnen Punkt auf dem Boden, sondern einen Bereich abtasten, berechnet die Methode den Mit-

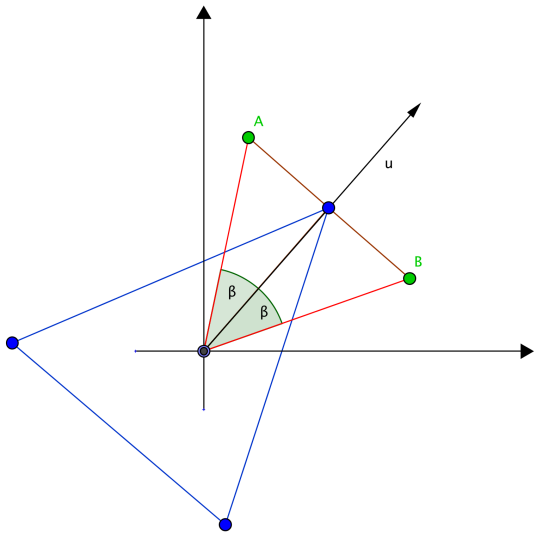


Abbildung 5.7: Die Abstraktion des Roboters und der Sensorpositionen

telwert aus neun Pixeln um die Position des Lichtsensors. Hierzu wird in der Methode `gibHelligkeitswert(int x, int y)` der Helligkeitswert jedes einzelnen der neun Pixel aus dem Bilddaten-Array abgefragt und der Mittelwert gebildet. Die Sensoren, die dann mit `gibLichtmittelwert(int x, int y)` die Helligkeit des ausgewählten Bereichs abfragen, bekommen das genormte Ergebnis (der Wert 0 entspricht schwarz, 100 entspricht weiß) des Mittelwerts ausgegeben.

Die Farbsensoren der NXT Roboter verhalten sich in der Realität ähnlich

wie die Lichtsensoren. Somit ist die Implementation der Farbsensoren im Simulator code-identisch und auf eine explizite Beschreibung kann in dieser Ausarbeitung verzichtet werden.

Ultraschall- und Berührungssensor

Der Berührungssensor, oder wie von leJOS TouchSensor genannt, ist im Simulator an der vordersten Spitze des Roboter-Objekts platziert. Da Hindernisse im Simulator schwerlich als mehrdimensionale Objekte dargestellt werden können, wird in dieser ersten Version der Simulationsumgebung zunächst mit einer Farbunterscheidung gearbeitet. Dies funktioniert wiederum über den Helligkeitswert der Pixel, die sich gerade unter der Spitze des Roboter-Objekts befinden. Die Position des Sensors ergibt sich dabei in Anhängigkeit zu der Größe des Roboters und seiner Ausrichtung. Sei B die Breite des Roboters und α der Wert aus der Variablen `_ausrichtung` des Roboters. Weiter entspreche *xPos* der x-Position und *yPos* der y-Position des Roboters.

Dann folgt für die Koordinaten $(x|y)$ des Berührungssensors

$$\begin{aligned} dx &= (\sin\alpha) \cdot \left(\frac{B}{2}\right) \\ dy &= (\cos\alpha) \cdot \left(\frac{B}{2}\right) \\ x &= xPos + dx \\ y &= yPos - dy. \end{aligned}$$

Für die Implementation des Ultraschallsensors wird eine Implementation

des Hindernisses als Objekt auf dem Parcours benötigt. Dies ist für die Implementation eines ersten Prototyps des Simulators jedoch nicht essentiell, da die für SuS relevanten Aufgaben auch mit einem Berührungssensor gelöst werden können. Daher verzichtet dieser Prototyp zunächst auf die Implementation eines Ultraschallsensors.

Sensorports

Jeder der eben aufgeführten Sensoren ist an einen so genannten Sensorport angeschlossen. Die Klasse `SensorPort` enthält Felder, welche die einzelnen Ports darstellen. Ein Exemplar einer Sensoren-Klasse wird bei der Initialisierung stets an einen Sensorport gekoppelt.

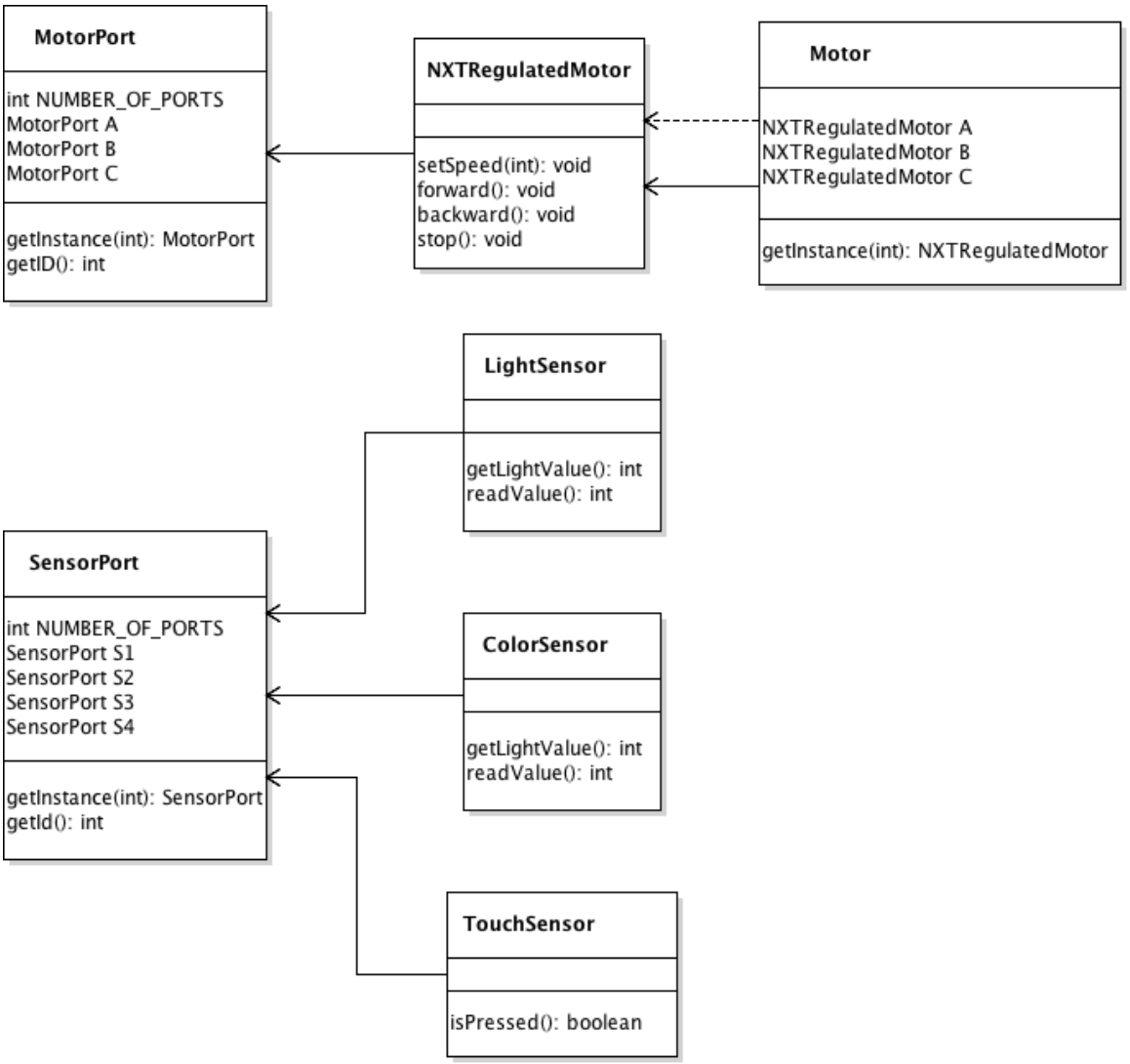


Abbildung 5.8: UML-Diagramm der wichtigsten von leJOS NXJ vorgegebenen Klassen

5.3.3. Die Klassen des `sim`-Pakets

Nun zu einer Beschreibung der Struktur und Implementation der Klassen, die für die Simulation zuständig sind. Hierzu gehören die grafische Repräsentation des Parcours und des Roboters, der Auswahldialog für den Parcours sowie die Leinwand, auf welcher der simulierte Roboter fahren soll. Im Rahmen dessen wird die Funktionalität sowie die Interaktion der einzelnen Klassen beschrieben. Eine Gesamtübersicht der wichtigsten Klassen ist in 5.9 abgebildet.

Simulator

Um die Simulationsumgebung aufzurufen, erzeugt man ein Exemplar der Klasse `Simulator`. Der Konstruktor dieser Klasse ist so konzipiert, dass zunächst ein Parcours erzeugt und in diesem Zusammenhang eine Bilddatei mit dem gewünschten Parcours ausgewählt wird.

Als nächstes wird der Roboter über den Konstruktoraufruf

```
Roboter(335, 500);
```

auf den ausgewählten Parcours projiziert.

Die Klasse `Simulator` ist nicht nur für die Erzeugung der einzelnen Elemente der Simulationsumgebung zuständig, sondern auch für die Verwaltung des Prozessablaufs. Hierzu wird mit dem Aufrufen eines Simulators ein sogenannter *Thread* erzeugt. Im Rumpf des Threads sind wiederum zwei Anweisungen innerhalb einer Endlosschleife zu finden. Zum einen der Befehl

```
_roboter.update();
```

der dafür sorgt, dass der Status des Roboter-Objekts aktualisiert wird, zum

anderen eine Verzögerung, ein *Delay*, wodurch die Anweisungen in der Schleife alle Millisekunde neu durchlaufen werden.

Bildeinleser

Die Klasse `BildEinleser` besitzt keinen Konstruktor, da diese nur als Hilfsklasse bestimmte Methoden zur Verfügung stellen muss. Durch den Aufruf der Methode `liesBilddaten()` wird ein JFileChooser erzeugt, mithilfe dessen der Benutzer eine Bilddatei auswählen kann. Die ausgewählte Bilddatei stellt den Parcours dar, auf dem der Roboter simuliert werden soll. Danach wird das Bild so umgewandelt, dass ein Array aus Bildpunkten entsteht. Dies ist für die spätere Arbeit mit Bewegungen sowie den Einsatz von Sensoren auf dem Parcours hilfreich, da auf jeden Datenpunkt des Parcours per Abruf der zugehörigen Array-Zelle zugegriffen werden kann.

Parcours

Der Parcours ist die Schnittstelle zwischen Simulator und den Klassen des `nxt`-Pakets. Sie ist unter anderem dafür zuständig, durch den Aufruf von `liesBilddaten()` am BildEinleser den Dialog für die Auswahl der Parcours-Grafikdatei zu erzeugen.

Zudem ist die Schnittstelle dafür verantwortlich, die Bilddaten an die Leinwand zu übergeben, die an dieser Stelle erzeugt wird. Hierfür sind die implementierten Methoden `aktualisiereBildgroesse(short[][]`

`bilddaten)`, `erzeugeLeinwand()` und `zeichneBild()` zuständig. Insbesondere Letztere übergibt die Bildpunkte in Form eines short-Arrays an die Leinwand.

Ferner befindet sich in dieser Klasse die Implementation der Abfrage der Helligkeitswerte des Parcours, auf dem sich der Roboter befindet. Diese wurde mithilfe der beiden Methoden `gibHelligkeitswert(int x, int y)` und `gibLichtmittelwert(int x, int y)` realisiert. Die genauere Beschreibung der Funktionalität dieser Methoden ist im vorangegangenen Abschnitt über die Licht- und Farbsensoren zu finden.

Leinwand

Wie zuvor beschrieben, wird die Leinwand von dem Parcours erzeugt. Die Klasse Leinwand sorgt nun dafür, dass die als Array übergebenen Bildpunkte in einem JFrame – also einem neuen Fenster – angezeigt werden.

Schließlich bietet die Leinwand eine Methode `warte()` an, die dafür genutzt wird, dass sich die Objekte auf der Leinwand animiert über den Hintergrund bewegen. Dies geschieht über den Aufruf `Thread.sleep(millisekunden)`, der dafür sorgt, dass der Prozess, der sich um das Zeichnen des Roboters kümmert, einige Millisekunden verzögert ausgeführt wird.

Roboter

Nun zum Kernstück der Simulationsumgebung, der Klasse `Roboter`. Diese enthält alle relevanten Informationen des Roboter-Objekts, wie die Position,

die Ausrichtung, die Geschwindigkeit – in Form von Variablen `_xPos`, `_yPos`, `_ausrichtung`, `_geschwindigkeit` – sowie das Aussehen.

Sobald ein neuer Roboter erzeugt wird (`Roboter(int x, int y)`) wird das Bild des Roboters, welches zur Zeit noch ein einfaches Dreieck ist, eingelesen, auf die Leinwand gezeichnet und auf die übergebene Position gesetzt. Hierbei ist anzumerken, dass die Bilddatei einen imaginären Rahmen besitzt, so dass zwei Feldvariablen zur Positionskorrektur existieren, die für das Zeichnen auf der Leinwand benutzt werden. Hierdurch wird sichergestellt, dass der Roboter auch tatsächlich an der übergebenen Position gezeichnet wird. Beim Konstruktoraufruf übergibt der Benutzer demnach die Koordinaten für den Mittelpunkt des Roboters.

Weiterer Bestandteil der Klasse ist die Methode `update()`. Diese ist dafür zuständig, dass der Roboter auf der Leinwand Bewegungen ausführen kann. Dies geschieht zunächst über die Veränderung der Ausrichtung des Roboters, wodurch natürlich eine Rotation der optischen Repräsentation des Roboters, und somit des Bildes, nötig ist. Die Methode `rotate(double Degrees)` erstellt in diesem Zusammenhang eine Kopie der ursprünglichen Robotergrafik, dreht diese um den angegebenen Winkel und ersetzt die bisher bestehende Grafik durch die gedrehte Kopie. Dieses Verfahren sorgt dafür, dass keine Pixelfehler und optischen Verwischungen an der Robotergrafik auf dem Parcours entstehen, da für jede Veränderung eine neue Kopie der Originalgrafik verwendet wird.

Abschließend wird nun der Roboter mithilfe der beiden Methoden `setzePosition(int x, int y)` und `zeichnen(int x, int y)` auf der neuen Position auf dem Parcours gezeichnet. In Anlehnung an die leJOS API funktioniert dies über die Veränderung des Winkels und der Geschwin-

digkeit. Dies ist damit zu begründen, dass von den SuS nur die einzelnen Motoren angesteuert werden können, was dazu führt, dass jeweils nur durch eine Veränderung der Geschwindigkeit eines Motors das Fahren von Kurven möglich ist. Die Berechnung der neuen Position findet dabei wie folgt statt:

Die Feldvariable `_winkelVeränderung` bekommt durch den Aufruf der Methode `aendereBewegung(double winkel, double geschwindigkeit)` in der Klasse NXTRegulatedMotor einen neuen Wert zugewiesen. Da dieser im Roboter-Objekt gespeichert ist, kann direkt darauf zugegriffen werden. Der in der Variablen `_winkelVeränderung` gespeicherte Wert wird nun zunächst auf die Feldvariable `_ausrichtung` aufaddiert.

Sei nun α der Wert der Variable `_ausrichtung` und v die in der Variable `_geschwindigkeit` gespeicherte Geschwindigkeit. Es entspreche *xPos* dem Wert von `_xPos` sowie *yPos* dem Wert von `_yPos`. Dann folgt

$$\begin{aligned} dx &= (\sin\alpha)\cdot v \\ dy &= (\cos\alpha)\cdot v \\ x &= xPos + dx \\ y &= yPos - dy \end{aligned}$$

und die Zielkoordinaten des Roboters entsprechen dem Punkt $(x|y)$.

Abschließend werden der Roboter und seine Position durch den Aufruf `zeichnen(int x, int y)` an die Leinwand übergeben, die dafür sorgt, dass das neue Gesamtbild im Fenster angezeigt wird.

Im Zusammenspiel mit dem Thread, der in der Klasse Simulator erzeugt

wird, führt der regelmäßige Aufruf der Methode `update()` zu der Bewegungsanimation des Roboters auf der Leinwand.

Außerdem enthält die Klasse `Roboter` eine Vielzahl von sondierenden Methoden, welche die Positionskoordinaten des Roboters sowie der Lichtsensoren und des Berührungssensors zurückgeben.

Da bei der Implementation jeweils mit der Höhe und Breite, die durch weitere sondierende Methoden (`getHeight()` und `getWidth()`) an der Robotergrafik direkt ausgelesen werden können, gearbeitet wurde, besteht die Möglichkeit, die Bilddatei des Roboters auszuwechseln.

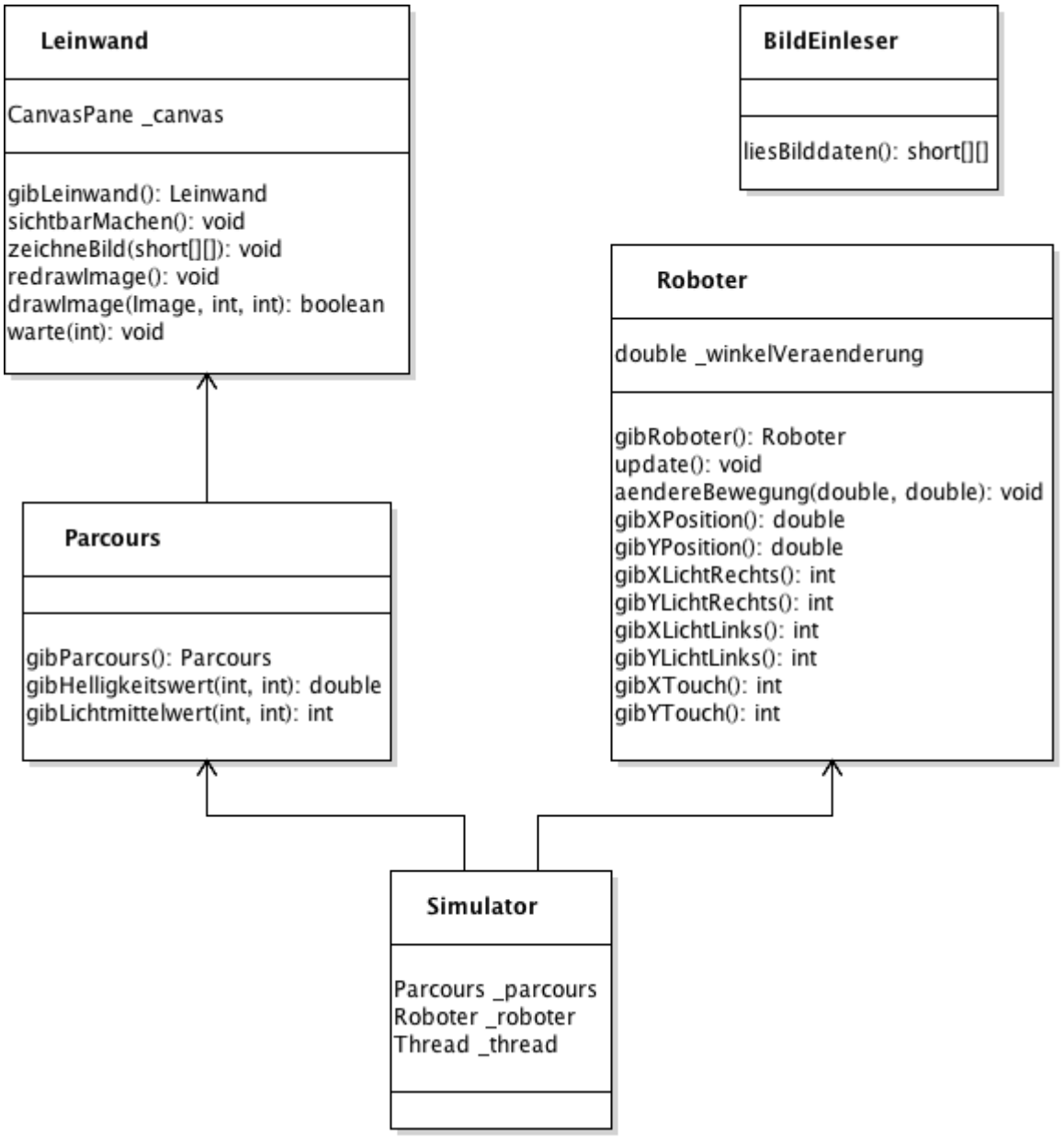


Abbildung 5.9: UML-Diagramm der essentiellen Klassen für die Simulationsumgebung

Kapitel 6

Fazit und Ausblick

In dieser Masterarbeit wurde der Protoyp einer Java-Simulationsumgebung für LEGO MINDSTORMS NXT Roboter entwickelt. Das Ergebnis dieser Entwicklung ist die Möglichkeit, SuS ein Werkzeug zur Überprüfung ihres Quelltextes für die Steuerung eines Roboters an die Hand zu geben. Die enwickelte Simulationsumgebung umfasst alle für die Programmierung erster Software mit NXT Robotern essentiellen Klassen sowie deren Methoden und stellt eine Testumgebung für Sensoren und Motorbewegungen zur Verfügung.

6.1. Umsetzung der Anforderungen

Die Simulationsumgebung bietet den SuS die Möglichkeit, mit der von ihnen kennengelernten Syntax (auf Basis der leJOS NXJ API) ihre individuellen Lösungen direkt am Computer auszuprobieren. Hierbei müssen sie sich nicht mit einer neuen, zusätzlichen Benutzeroberfläche vertraut

machen, sondern lediglich drei zusätzliche Anweisungen im Quellcode einfügen. Die Steuerung des NXT Roboters und des Roboters in der Simulation ist damit also nahezu code-identisch.

Bezüglich der Realitätstreue der Parcours lässt sich zunächst sagen, dass die beispielhaft im Projekt mit enthaltenen Parcours Ausschnitte eines realen Parcours sein könnten. Dies stellt sicher, dass die SuS ihre Roboter in Testumgebungen fahren lassen können, die einen Bezug zu den Realbedingungen haben.

Außerdem kann jede Lehrkraft mithilfe eines Grafikprogramms weitere Parcours entwerfen und den SuS zur Verfügung stellen. Auch dies gehörte zu den an die Simulationssoftware gestellten Anforderungen.

Zusätzlich zu den Kommentaren im Code, der den Lehrkräften zur Verfügung gestellt wird, ist im Anhang noch eine Kurzanleitung zu finden, die durch die Einbindung der Klassenbibliothek des Simulators in BlueJ führt sowie Hinweise zur Benutzung enthält. Hierdurch kann ein leichtes Einarbeiten der Lehrkräfte in den Umgang mit der Simulationsumgebung gewährleistet werden.

Die in **4** beschriebenen Anforderungen aus Sicht der SuS und der Lehrkräfte sowie hinsichtlich der Erweiterbarkeit konnten in vollem Umfang umgesetzt werden. Die vorgestellte prototypische Simulationsumgebung für LEGO MINDSTORMS NXT Roboter kann bereits im derzeitigen Zustand den Informatikunterricht an Schulen um eine individualisierte Lehr-Lern-Methode bereichern. Sie visualisiert Befehlsabläufe, die von den SuS entwickelt werden und stellt gleichzeitig den Bezug zwischen Quelltext und Realwelt her.

6.2. Rückmeldung von Lehrkräften zum Simulator

Im Rahmen dieser Arbeit wurde der entwickelte Prototyp der Simulationsumgebung einigen aktiv im LEGO-Roboter-Bereich tätigen Lehrkräften vorgestellt. Hierdurch ergaben sich mehrere Anmerkungen und Wünsche sowie weitere Anforderungen, die für den Einsatz unter Realbedingungen an die Simulationsumgebung gestellt werden könnten. Diese werden im Folgenden vorgestellt und um Lösungsansätze für den entwickelten Simulator erweitert.

Auf der Ebene der Benutzerfreundlichkeit wurde der Wunsch geäußert, den Simulator direkt starten zu können. Hierzu solle in einem Dialogfenster die gewünschte Parcours-Grafik sowie die auszuführende Klasse ausgewählt werden können. Da die Übertragung des von den SuS geschriebenen Programmcodes auf die NXT Roboter ähnlich stattfindet – nämlich durch die Übertragung der ausgewählten Java-Klasse, und somit der `.class`-Datei – könnte hier eine noch deutlichere Verzahnung der Abläufe von der Simulationsumgebung und des realen Roboters stattfinden.

Zusätzlich wurden Anmerkungen zur Robotergrafik gegeben. Zunächst solle die grafische Repräsentation des Roboter-Objekts auf dem Parcours die Positionen der Sensoren beinhalten. Hierzu wurde der Vorschlag unterbreitet, die drei Standardbauarten der NXT Roboter als auswählbare Grafiken mit fest positionierten Sensoren anzubieten. In der Implementation ließe sich dies realtiv gut über eine Veränderung des Konstruktors der Klasse `Roboter` lösen. Dieser würde dann, statt eine vorgegebene Image-Datei zu laden, wie die Parcours-Klasse einen Auswahldialog erzeugen, mit dem dann die gewünschte Robotergrafik ausgewählt werden könnte.

Im Hinblick auf eine weiterführende Arbeit mit der Simulationsumgebung spielen die Positionen der Sensoren und anderer Bauteile am NXT Roboter eine bedeutende Rolle. Beispielsweise können minimale Positionsänderungen darüber entscheiden, ob eine Kurve gefahren werden kann. Aufgrund dieser Feinheiten wurde der Wunsch geäußert, eine zusätzliche Bearbeitung des Roboter-Objekts im Simulator zu ermöglichen. Im Idealfall würde der Roboter dann modular aus den relevanten Einzelteilen als Objekte zusammengesetzt, so dass für jeden Motor und jeden Sensor die genaue Position vorgegeben werden könnte, die dann dem Simulator übergeben würde.

Auch die Sensoren der nächsten Generation der LEGO Roboter verhalten sich anders, als die der NXT Roboter. Somit müssten auch auf dieser Ebene in der Simulationsumgebungen Anpassungen stattfinden – insbesondere bei den Farbsensoren, da diese nun drei verschiedene Modi anbieten.

Ein weiterer programmierspezifischer Aspekt ist, dass die befragten Lehrkräfte ihren SuS beibringen, sich bestimmte Sensordaten auf dem LCD-Display des NXT Roboters anzeigen zu lassen. Die hierzu benötigten Klassen wurden bisher im Simulator nicht umgesetzt. Jedoch könnte das Display als weiteres Fenster während der Simulation hinzugefügt werden, sodass auch die Messdaten der Sensoren neben dem ausgewählten Parcours angezeigt würden.

Abschließend wurde noch das Problem der Lichtverhältnisse in unterschiedlichen Situationen angeführt. Meist stehen die realen Trainingsparcours und die Wettbewerbparcours in unterschiedlichen Ausrichtungen zu den verschiedenen räumlich bedingten Lichtquellen. So liefert ein weißer Untergrund beispielsweise bei Tageslicht andere Sensordaten als derselbe

Untergrund bei künstlichem Lichteinfall. Um auch diese Gegebenheiten möglichst realitätsgetreu simulieren zu können, gäbe es die Möglichkeit, eine Bildbearbeiter-Klasse hinzuzufügen, mit der man dem ausgewählten Parcours in der Simulationsumgebung unterschiedliche Helligkeiten, die durch verschiedene Lichtquellen hervorgerufen werden, übergeben könnte.

Insgesamt lässt sich demnach feststellen, dass die Simulationsumgebung das Potential aufweist, sich durch weitere Verfeinerungen als nützliches Softwareentwicklungstool für LEGO MINDSTORMS NXT Roboter zu etablieren.

6.3. Verbesserungen und Erweiterungen

Da die entwickelte Simulationsumgebung bisher prototypisch ist, sollte sie folglich als nächstes in der Praxis getestet werden. Hierzu kann sie verschiedenen Lehrkräften für ihren Unterricht zur Verfügung gestellt werden, die dann in unterschiedlichen Jahrgangsstufen damit arbeiten können. Dieser Schritt wird dafür sorgen, dass das Verhalten der Sensoren verfeinert und die Rubustheit des Codes verbessert werden kann.

Damit einher geht auch der Umstand, dass das geforderte Verhalten der Sensoren bisher nur in einem beschränkten Maße getestet werden konnte. Dies bedeutet, dass eine hinreichend gute Approximation des Verhaltens der Sensoren beim Testen mit einem Mustercode erreicht wurde. Eine mögliche Anpassung an die Realbedingungen ist jedoch angezeigt.

Die Roboter-Grafik im Simulator, die derzeit zu Testzwecken aus einem Dreieck besteht, kann durch eine beliebige andere Grafik ausgetauscht

werden, um den SuS das Gefühl zu vermitteln, dass „ihr“ Roboter im Simulator gerade die ersten Bewegungen ausführt.

Eine weitere Möglichkeit wäre auch, eine Steuerung der Simulation anzubieten, welche ähnlich wie bei *Greenfoot* aufgebaut ist. Konkret würde dies bedeuten, dass der JFrame, in dem der Parcours angezeigt wird, um einen Bereich mit Buttons erweitert würde. Mithilfe dieser wären die SuS dann in der Lage, die Simulation zu starten, zu pausieren und anzuhalten.

Eine umfassendere Änderung wäre das Schreiben einer eigenen BlueJ Extension, die durch Lehrkräfte und hieran interessierte SuS unter Anleitung vorgenommen werden könnte. Dies würde zwar einen hohen Aufwand bedeuten, jedoch für eine andere Art der Integration in die Entwicklungsumgebung sorgen. Auch würde hierdurch die Möglichkeit entstehen, den Simulator gegebenenfalls direkt am Kontextmenü aufzurufen.

6.4. Zusammenfassung

Die in dieser Masterarbeit entwickelte Java-Simulationsumgebung für LEGO MINDSTORMS NXT Roboter liefert ein stabiles Tool für den Einstieg in die schriftliche objektorientierte Programmierung anhand von Robotern. Der Simulator ist in der Lage, einfache, auf Basis der NXJ API von leJOS in Java geschriebene Programme auf einem digitalen Parcours auszuführen und den SuS somit eine optische Rückmeldung zu ihrem Code zu geben.

Des Weiteren stellt dieser Simulator eine Basis für Erweiterungen bezüglich UI-spezifischer Anpassungen oder auch für ein Upgrade auf die nächste

Generation von LEGO MINDSTORMS Robotern – den EV3 – dar. Die Simulationsumgebung lässt sich – wenn für den Unterricht erforderlich – durch die Lehrkräfte umgestalten und erweitern. Dies wird durch den objektorientierten Ansatz in der Programmierung und die Nutzung von Paketen, die in sinnvolle Kategorien unterteilt wurden, ermöglicht.

Die Weiterentwicklung von Robotern, die im Unterricht eingesetzt werden können, wird in Zukunft ständig zunehmend die informatische Bildung und den Informatikunterricht in Schulen beeinflussen und prägen. Insbesondere die Arbeit mit LEGO MINDSTORMS Robotern im Kontext der objektorientierten Programmierung wird sich über viele weitere Generationen von SuS fortsetzen. Der Einsatz der in dieser Masterarbeit entwickelten Simulationsumgebung für derartige Roboter bringt die SuS näher an die realen Verhältinisse wissenschaftlicher Arbeit mit Robotern. Sie eröffnet den Lehrkräften und den SuS sowohl neue Möglichkeiten als auch neue Herausforderungen in der objektorientierten Programmierung mit Robotern.

Literaturverzeichnis

[Abe01] Michael Abend. “Robotik und Sensorik. Darstellungsschwerpunkt: Selbständige Entwicklung „unscharfer“ Algorithmen zur räumlichen Orientierung (unter Verwendung des LEGO-Mindstorms-Systems)", *Schriftliche Prüfungsarbeit zur zweiten Staatsprüfung für das Amt des Studienrats*, Berlin, 2001

[Abt15] Dietmar Abts. *Grundkurs JAVA. Von Grundlagen bis zu Datenbank- und Netzwerkanwendungen*, 8., überarbeitete und erweiterte Auflage, Springer Vieweg, Wiesbaden, 2015

[Aeg16] o.V. http://www.aplu.ch/home/apluhomex.jsp?site=27, Abgerufen am 02.02.2016, Aegidius Plüss – NxtJLib, 2016

[Bar03] David J. Barnes, Michael Kölling. *Objektorientierte Programmierung mit Java. Eine praxisnahe Einführung mit BlueJ*, Übersetzt von Axel Schmolitzky, Pearson Studium, München, 2003

[Ber10] Karsten Berns, Daniel Schmidt. *Programmierung mit LEGO MINDSTORMS NXT. Robotersysteme, Entwurfsmethodik, Algorithmen*, Springer Heidelberg Dordrecht London New York, 2010

[Bow12] David Bowes. http://homepages.herts.ac.uk/~comqdhb/lego/bluej.php, Abgerufen am 07.02.2016, Herfortshire, 2012, Lejos NXJ extension for BlueJ

[BricxCC] o.V. URL: http://bricxcc.sourceforge.net/, Abgerufen am 02.02.2016

[Ehm09] Matthias Ehmann et al. *Duden Informatik - Sekundarstufe I / 9./10. Schuljahr - Objektorientierte Programmierung mit BlueJ*, Duden Schulbuchverlag Berlin Mannheim, 2009

[Her12] Joachim Hertzberg, Kai Lingemann, Andreas Nüchter. *Mobile Roboter. Eine Einführung aus Sicht der Informatik*, Springer-Verlag Berlin Heidelberg, 2012

[HH09] Behörde für Schule und Berufsbildung Hamburg (Hrsg.). *Informatik – Bildungsplan Gymnasiale Oberstufe*, Hamburg, 2009

[HH11] Behörde für Schule und Berufsbildung Hamburg (Hrsg.). *Informatik Wahlfplichtfach – Bildungsplan Gymnasium Sekundarstufe I*, Hamburg, 2011

[HH14] Behörde für Schule und Berufsbildung Hamburg (Hrsg.). *Informatik Wahlpflichtfach – Bildungsplan Stadtteilschule Jahrgangsstufen 7 – 11*, Hamburg, 2014

[Hub07] Peter Hubwieser. *Didaktik der Informatik*, 3. Auflage, Springer-Verlag Berlin Heidelberg, 2007

[Lego] o.V. URL: http://www.lego.com/en-us/mindstorms/history, Abgerufen am 06.11.2015, LEGO, 2015

[leJOS] o.V. URL: http://www.lejos.org/nxj.php, Abgerufen am 14.12.2015, leJOS Java for Lego Mindstorms, 2015

[Lil08] Carloa Lilienthal. "Komplexität von Softwarearchitekturen – Stile und Strategien –", *Dissertation im Fachbereich Informatik der Universität Hamburg*, Hamburg, 2008

[Mos12] Christian Moser. *User Experience Design: Mit erlebniszentrierter Softwareentwicklung zu Produkten, die begeistern*, Springer-Verlag Berlin Heidelberg, 2012

[Nie99] Jürg Nievergelt. *"Roboter programmieren ein Kinderspiel. Bewegt sich auch etwas in der Allgemeinbildung?*, Informatik-Spektrum 22:5, S. 364–375, 1999

[nxcEditor] o.V. URL: http://nxceditor.sourceforge.net/index_german.html, Abgerufen am 02.02.2016, nxcEditor

[PHBern] o.V. URL: http://www.java-online.ch/lego/index.php?inhalt_links=home/nav_home.inc.php&inhalt_mitte=home/home.inc.php, Abgerufen am 02.02.2016, Lego-Robotik mit Java

[Roberta] o.V. URL: http://roberta-home.de/de/aktuelles/simulator-und-editor-f%C3%BCr-nxc-linux-windows-mac-os-x, Abgerufen am 02.02.2016, Roberta – Lernen mit Robotern, Simulator und Editor für NXC (Linux, Windows, MacOS X)

[Rol14] Mark Rollins. *Beginning LEGO MINDSTORMS EV3*, Apress, Berkeley, CA, 2014

[Sch04] Rafael Schreiber. "Der Einsatz von LEGO-Mindstorms im Informatikunterricht der 11. Klasse der Leonard-Bernstein-Oberschule. Sicherung und Transfer grundlegender algorithmischer Strukturen in NQC.", *Schriftliche Prüfungsarbeit im Rahmen der zweiten Staatsprüfung für das Amt des Studienrats*, Berlin, 2004

[Schwa07] Christine Schwarzer, Petra Buchwald. "Umlernen und Dazulernen.", In: Michael Göhlich, Christoph Wulf, Jörg Zirfas (Hrsg.): *Pädagogische Theorien des Lernens*, Beltz, Weinheim und Basel, S. 213–221, 2007

[RWTH] o.V. URL:http://schuelerlabor.informatik.rwth-aachen.de/simulator, Abgerufen am 02.02.2016, Simulator für LEGO Mindstorms NXT Roboter

[Sto01] Matthias Stolt. "Roboter im Informatikunterricht", 2001

[Ull12] Christian Ullenboom. *Java ist auch eine Insel – Das umfassende Handbuch*, 10. Auflage, Galileo Press, Bonn, 2012

[Wag05] Oliver Wagner. "LEGO Roboter im Informatikunterricht. Eine Untersuchung zum Einsatz des LEGO-Mindstorms-Systems zur Steigerung des Kooperationsvermögens im Informatikunterricht eines Grundkurses (12. Jahrgang, 2. Lernjahr) der Otto-Nagel-Oberschule (Gymnasium)", *Schriftliche Prüfungsarbeit im Rahmen der zweiten Staatsprüfung für das Amt des Studienrats*, Berlin, 2005

[Zül90] Reinhard Budde, Heinz Züllighoven. *Software-Werkzeuge in einer Programmierwerkstatt. Ansätze eines hermeneutisch fundierten Werkzeug- und Maschinenbegriffs*, Oldenbourg, München [u.a.], 1990

Anhang A

Die folgende Anleitung soll einen Leitfaden für die Einbindung und Benutzung der Java Simulationsumgebung für LEGO MINDSTORMS NXT Roboter in BlueJ darstellen.

Zu beachten ist, dass die in den Abbildung sichtbaren Pfade immer abhängig von der persönlichen Ablagestruktur der Dateien ist. Deshalb wird auf eine konkrete Benennung von Pfaden im Zusammenhang mit der Klassenbibliothek sowie dem Ort der Parcours-Grafikdateien verzichtet.

Laden der Klassenbibliothek

Die Simulationsumgebung wird in Form einer `.jar`-Datei ausgeliefert (`simulator.jar`) und sollte an einem nachvollziehbaren Ort im Dateisystem des Computers abgelegt werden. Zur Verwendung muss diese in BlueJ als Klassenbibliothek importiert und geladen werden (s. Abb. 6.1).

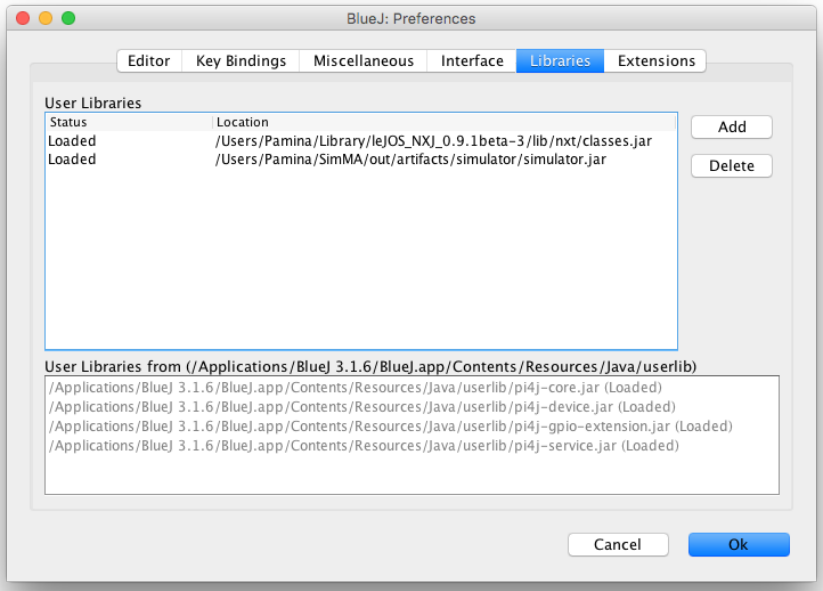


Abbildung 6.1: Die in BlueJ geladenen Klassenbibliotheken

Import innerhalb der geschriebenen Klassen

Um den Simulator nun mit dem in BlueJ geschriebenen Quellcode verwenden zu können, müssen folgende Schritte erfolgen:

Zunächst müssen zwei Import-Anweisungen im Quellcode oberhalb der Klasse eingefügt werden. Gleichzeitig müssen die Imports der leJOS-Bibliotheken auskommentiert werden, wie in Abbildung 6.2 beispielhaft dargestellt ist.

Abbildung 6.2: Die Import-Anweisungen im Quellcode

Aufrufen des Simulators

Das Starten des Simulators erfolgt über den Konstrukturaufruf

```
Simulator simulator = new Simulator();
```

welcher entweder im Konstruktor der geschriebenen Klasse oder aber innerhalb einer Methode platziert werden kann.

Nun erscheint entweder sobald ein neues Exemplar der implementierten Klasse erzeugt wird oder erst beim Aufruf einer bestimmten Methode der Auswahldialog für die Parcours-Grafik, wie Abbildung 6.3 zeigt. Wird ein Parcours ausgewählt, so startet der Roboter im Parcours-Fenster entweder direkt mit der Simulation oder wartet, bis am erzeugten Exemplar beispielsweise die Methode `start()` aufgerufen wird.

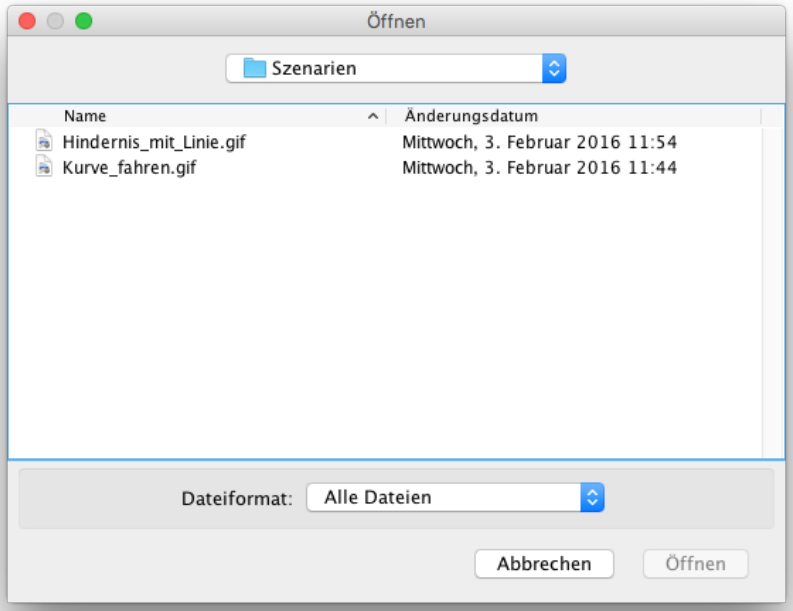


Abbildung 6.3: Auswahldialog der Szenarien

Einbindung eigener Grafiken

Zusätzlich zu den vorgegebenen Szenarien können selbstgestaltete Bilddateien als Parcours-Grafiken eingesetzt werden. Hierbei ist zu beachten, dass die Bilddateien mindestens eine Größe von 740×640 Pixel aufweisen, damit ausreichend „Bewegungsfreiheit“ für den Roboter zur Verfügung steht. Außerdem ist der Startpunkt des Roboters genormt. Dieser liegt am Punkt $(335|450)$.

Weitere Hinweise

Weiterhin ist der Umstand zu beachten, dass in der vorliegenden Implementation der Simulationsumgebung der linke Motor des Roboters am Port `B`, der rechte Motor am Port `C` angeschlossen ist.
Um unnötige Änderungen im Programmcode der SuS zu vermeiden sollte diese Portvergabe auch am NXT-Stein vorliegen.

"Hiermit versichere ich an Eides statt, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht."

Hamburg, 8. März 2016 .....

Pamina Maria Berg