Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

ENTWICKLUNG EINER JAVA SIMULATIONSUMGEBUNG FÜR LEGO MINDSTORMS NXT ROBOTER

Masterarbeit im Fach Informatik zur Erlangung des akademischen Grades

Master of Education

im Studiengang Lehramt an Gymnasien M.Ed.

Pamina Maria Berg
Matr.Nr. 6087438

Hamburg, den 27. Februar 2016

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

ABSTRACT

Hier der Text des Abstracts

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-AGs 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 über die Benutzung von LEGO MINDSTORMS Robotern zu optimieren, soll im Rahmen dieser Masterarbeit der Protoyp einer Simulationsumgebung für die Arbeit mit LEGO MINDSTORMS NXT Roboter entwickelt werden.

Hierzu 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 Neuplemenation gestellten Anforderungen vorgestellt, die dann im letzten Kapitel – nach der Vorstellung der Software – mit der tatsächlichen Funktionalität des entwickelten Prototyps

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

KAPITEL 2

AUSGANGSSITUATION

2.1. Das LEGO Mindstorms NXT System

Die ersten computergesteuerten LEGO Produkte wurden bereits 1986 veröffentlicht. In einer Zusammenarbeit von LEGO Education und dem Massachussettes 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 und ihre Roboter auf die Probe stellen wollen (s. [Lego]).

2.1.1. Hardware

Zentraler Bestandteil der LEGO MINDSTORMS NXT Roboter ist der sogenannte NXT-Stein. Dieser besteht aus einem 32 Bit RAM-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]).
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. In diese Motoren sind Rotationssensoren und Regler eingebaut, die dafür sorgen, dass sowohl die Drehgeschwindigkeit als auch die Umdrehungszahl mithilfe des Programmcodes kontrolliert werden kann (vgl. [Ber10, S.45–47]).
Für die NXT Roboter sind eine Vielzahl von Sensoren verfügbar, damit die SuS ihren Roboter sein Umfeld erkunden lassen können. 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“ [Ber10, S.2] wird.

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 unterschiedliche gut geeignet sind, und durch die 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 Unterricht 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 stelle 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 als sowohl didaktisch sinnvoll gesehen wird, also auch die Motivation der SuS in diesem Zusammenhang einen starken Einfluss auf den Lernerfolg und die Selbstständigkeit haben können.

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.

2.2.1. 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 Hansestadt Hamburg gelten. Des Weiteren 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 Vorstufe der Gymnasialen Oberstufe auftaucht.

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. Hier heißt es

Modul 2

- Abläufe analysieren und umgangssprachlich beschreiben, zu 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]

STS Jahrgangsstufen 7 –11 Informatik Wahlpflichtfach

Ähnlich ist es im Bildungsplan für die Stadtteilschulen. Hierbei werden allerdings 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.

Im Modul 3 sollen im zweiten Halbjahr die verbindlichen Inhalte

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

unterrichtet werden.

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 [HH09].

Gymnasiale Oberstufe – Studienstufe

In der Studienstufe gibt es den verbindlichen Inhalt *Objektorientierte Modellierung*, der die folgenden Punkte umfasst:

- 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]

Genereller 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 natürlich 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 komplexerer Beispiele 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.

KAPITEL 3

BISHER VERFÜGBARE SOFTWARELÖSUNGEN

In diesem Kapitel werden nun 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 *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

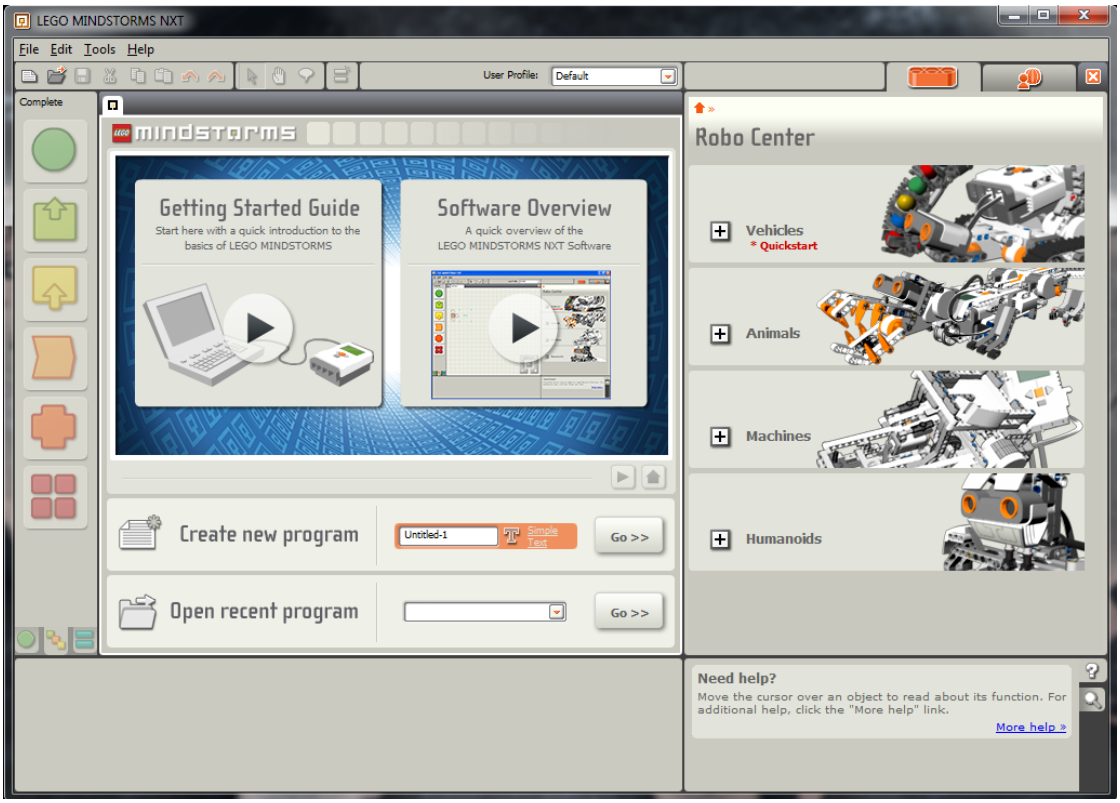

Abbildung 3.1: Der Startbildschirm des LEGO MINDSTORMS NXT-Programms

ineinander geschachtelte Schleifen erstellt werden sowie if-Abfragen mit zwei Fallunterscheidungen. Nahezu beliebig viele Bausteine lassen sich so miteinander kombinieren (Vlg. 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-Baustein technisch zur Verfügung stehen. Zudem muss es keine Veränderungen an der Firmware des NXT-Bausteins geben, was eine Zeitersparnis bei der Vorarbeit der SuS mit sich bringt.

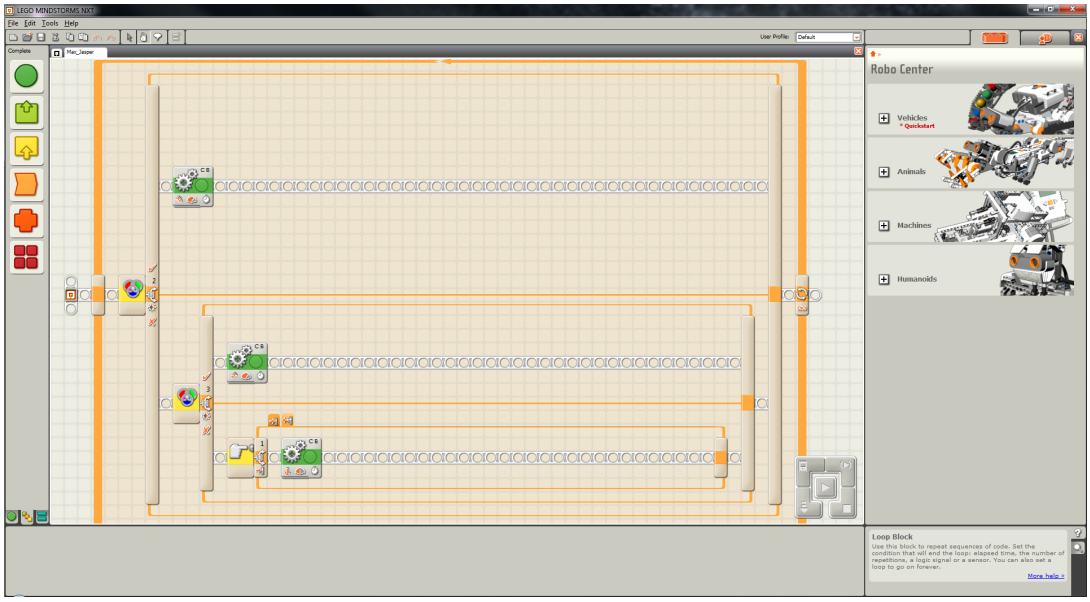

Abbildung 3.2: Fahren eines Roboters auf einer schwarzen Linie

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.

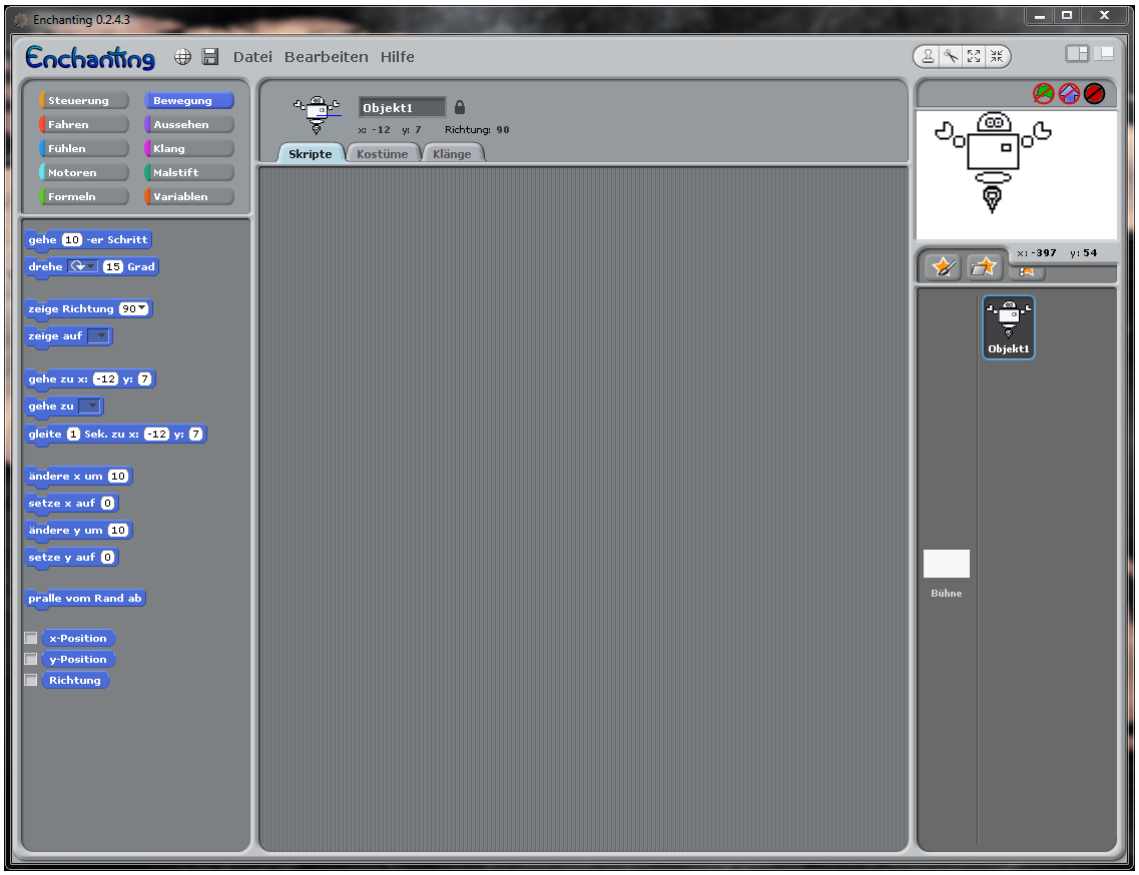

Abbildung 3.3: Der Startbildschirm von Enchanting

Wird Enchanting von den SuS geöffnet, erscheint zunächst der Startbildschirm wie in Abb. 3.3. Nun kann entweder ein neues Programm erstellt oder ein bestehendes geöffnet werden. Die verschiedenen Programmierelemente werden nach Kategorien in den oberen linken Ecke sortiert aufgeführt. Essentiell dabei sind die Kategorien *Steuerung*, *Fühlen*, *Motoren* und *Formeln*. 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 nun 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 gesagt werden, welcher Sensor angesprochen wird, und welcher Motor an welchem Port angeschlossen ist.

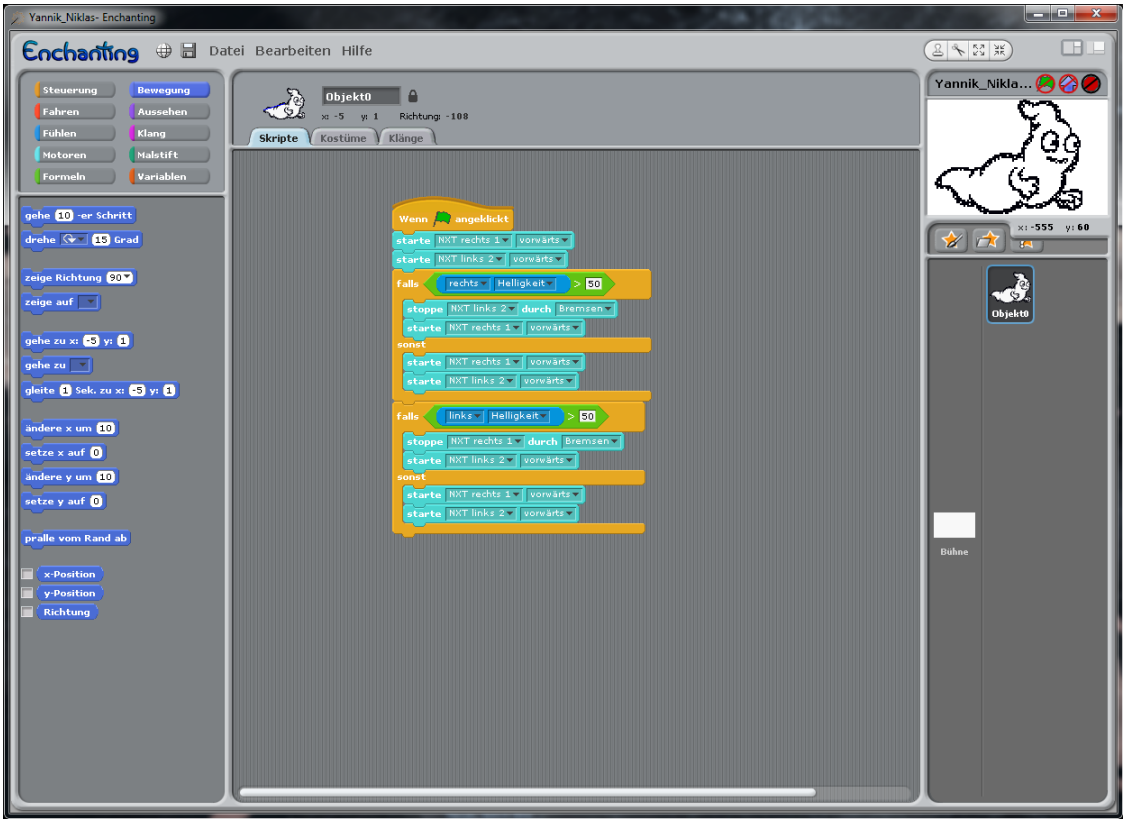

Abbildung 3.4: Beispiel eines Enchanting Programms

3.3. Schriftliche Programmierung von NXT-Robotern

Neben den grafischen Entwicklungsumgebungen in **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]).
Jedoch handelt es sich 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) und auch als Compiler und virtuelle Maschine (JVM) wird Software der Firma Oracle (bis 2010 Sun Microsystems) verwendet (vgl. [Bar03, S.15]).

Nicht nur in den Universitäten wird BlueJ inzwischen 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 anderen Entwicklungsumgebungen – im Gegensatz zu den meisten anderen Entwicklungsumgebungen – in BlueJ „schnell einige Objekte erzeugt und sofort untersucht werden“ [Ehm09, S.6] können. 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.

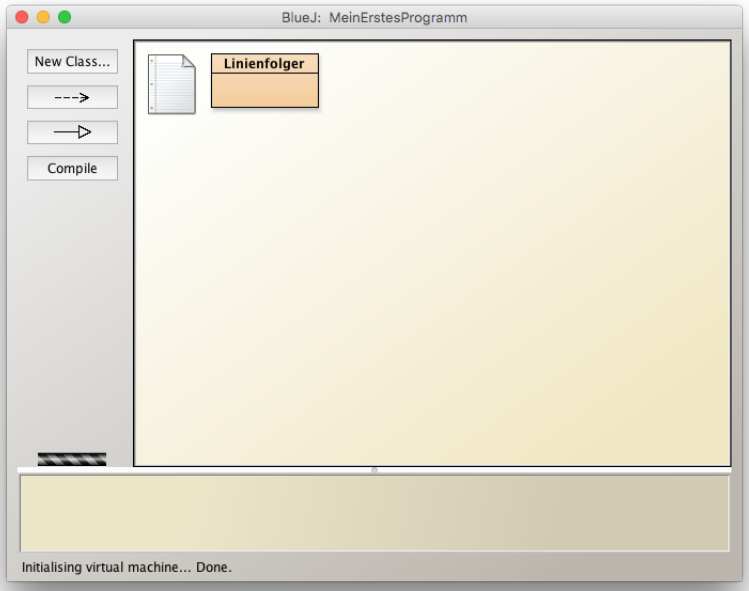

Abbildung 3.5: Die Benutzeroberfläche von BlueJ

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 der NXJ API stehen den SuS nicht nur die Klassen zur Steuerung ihres Roboters zur Verfügung. Darüber hinaus besteht Möglichkeit, über eine Erweiterung des Kontextmenüs, um spezielle Befehle, die dazu dienen, 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.

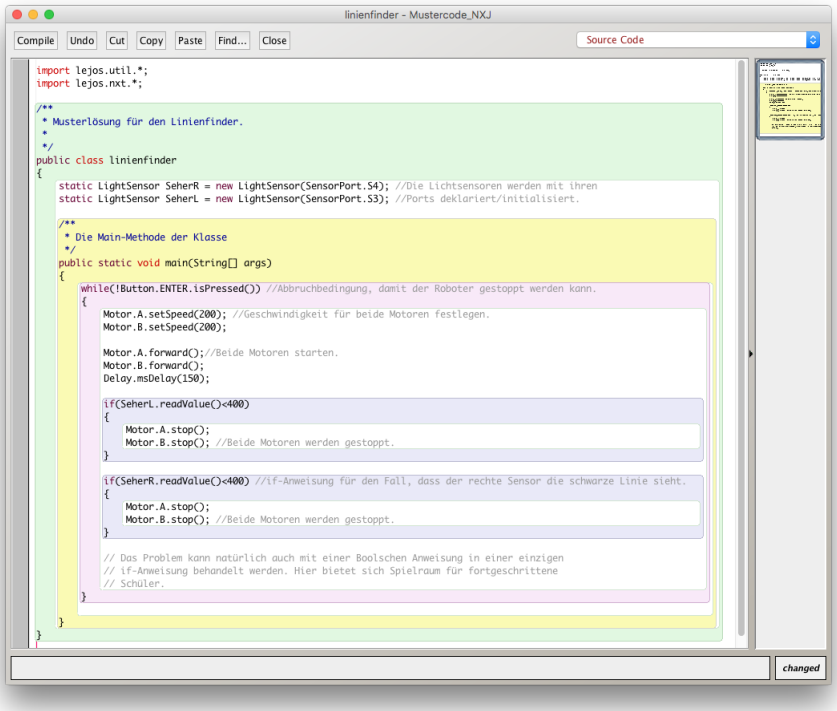

Abbildung 3.6: BlueJ-Beispiel zum Finden einer Linie

Mithilfe von BlueJ (oder einer anderen Entwicklungsumgebung) können die SuS unter anderem kontextorientierte Aufgaben lösen. z Hierzu gehören zum Beispiel das Anhalten auf einer Linie (vgl. Abb. 3.6) oder auch den Roboter einer schwarzen Linie folgen zu lassen, was eine der Grundaufgaben im RoboCup Junior Rescue Wettbewerb darstellt (vgl. Abb. 3.8).

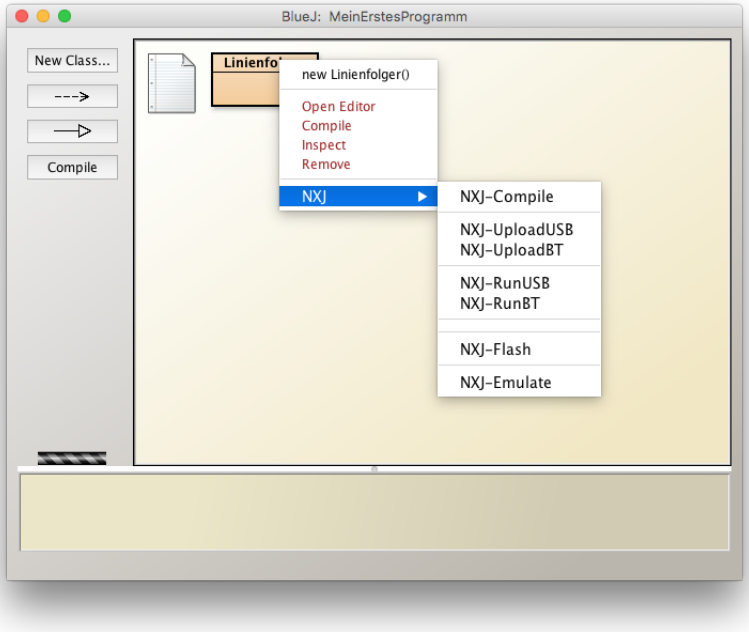

Abbildung 3.7: Das Kontextmenü mit Auswahlmöglichkeiten der NXJ-Extension

Der Vorteil an der Arbeit mit leJOS besteht darin, dass die SuS direkt mit reinem 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 des Klasse `Delay` benutzen.

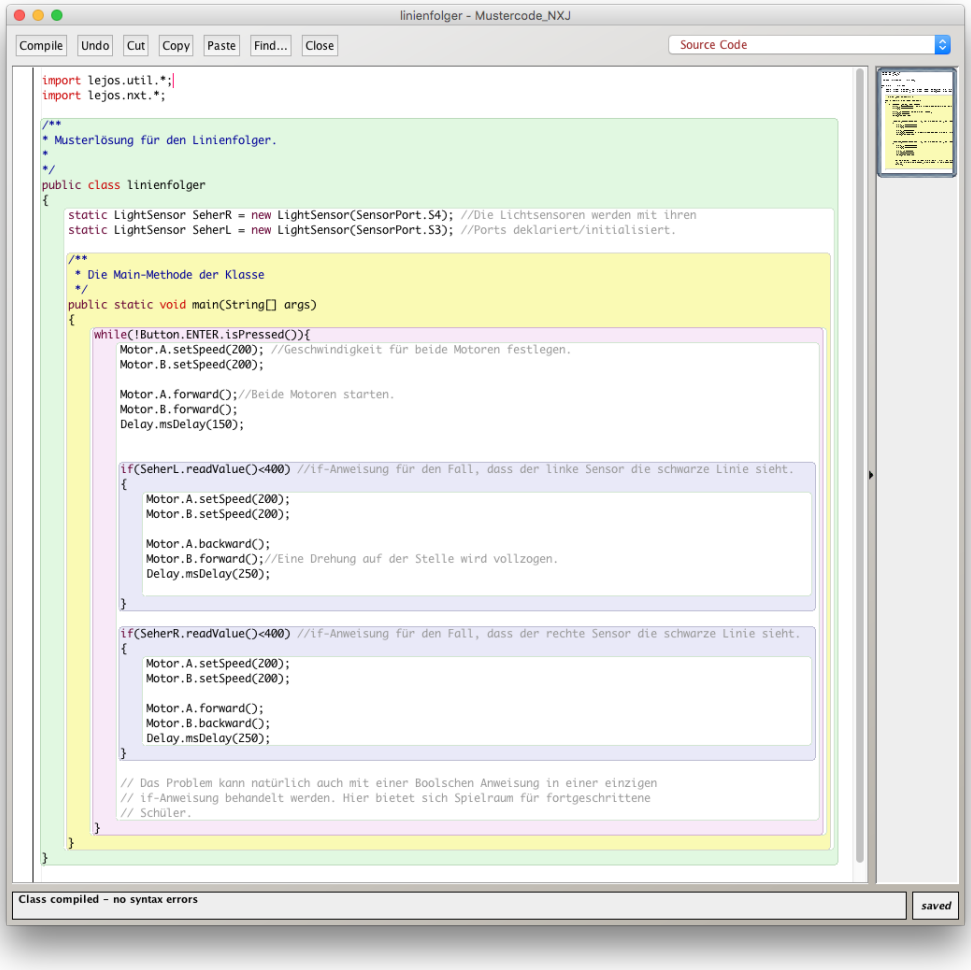

Abbildung 3.8: Der Roboter soll einer schwarzen Linie folgen

3.4. Simulationsumgebungen

Für den virtuellen Umgang mit LEGO MINDSTORMS Roboter 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 einen Simulator, *nxcSimulator* beinhaltet. Ähnlich wie BricxCC nutzt auch der nxcEditor die Programmiersprache NXC (s. [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 wird (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

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

Im Hinblick auf die Entwicklungs einer Simulationsumgebung für die Schule sind mehrere Perspektiven von Belang. In den folgenden beiden Unterkapiteln werden nun die Anforderungen an die Neuimplementierung des Simulator-Prototyps 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.

4.1. Schülerperspektive

Da es sich bei der Simulationsumgebung um einen Prototypen 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.

Des weiteren 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. Des weiteren wird hiermit auch der Aspekt der code-identischen Roboterkontrollsoftware

erfüllt, da die SuS das selbe Programm für den Simulator, wie auch für die Steuerung des Roboters selbst benutzen können sollten.

Ein weiterer Aspekt, den HERTZBERG/LINGEMANN/NÜCHTER beschreiben sind die Parcours. Diese sollten möglichst realitätsgetreu umgesetzt werden. Das heißt, dass Ausschnitte eines bestehenden realen Wettbewerbsparcours genutzt werden könnten, um für die SuS eine möglichst realitätsnahe Testumgebung anzubieten. Der Vorteil darin besteht auch in der Vergleichbarkeit der Ergebnisse. Die SuS lernen gleich zu Beginn einen wichtigen Aspekt bei der Entwicklung von Code 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.

4.2. 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, die entweder senkrecht in der Nähe des Roboters platziert ist, oder als Kurve eine Teilstrecke des realen Labyrinths darstellt, bestehen zu lassen.
In diesem Zusammenhang sollte die Möglichkeit gegeben sein, dass Lehrkräfte unter bestimmten Voraussetzungen eigene, an ihren Unterricht angepasste Parcours in die Software einbinden können.

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

4.3. Erweiterbarkeit

Nun zu dem Aspekt der Erweiterbarkeit, der für die Entwicklung dieser Simulationsumgebung – insbesondere, da diese für den Einsatz in Schulen konzipiert ist – zentral 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 Änderungen, die im Rahmen von Veränderungen der leJOS NXJ API stattfinden, müssen berücksichtigt werden können.

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. Prozesse während der Implementation

Die Entwicklung des Prototyps der Simulationsumgebung umfasste mehrere Schritte, sowie einige Vorarbeit. Diese werden im folgenden Abschnitt nun kurz zusammengefasst.

5.1.1. Mock-Ups

Um einen visuellen Grundgedanken festzuhalten und das Ziel der Implementation der vollständigen 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 wurden eine spezielle Art von Interaktionsprototypen 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).

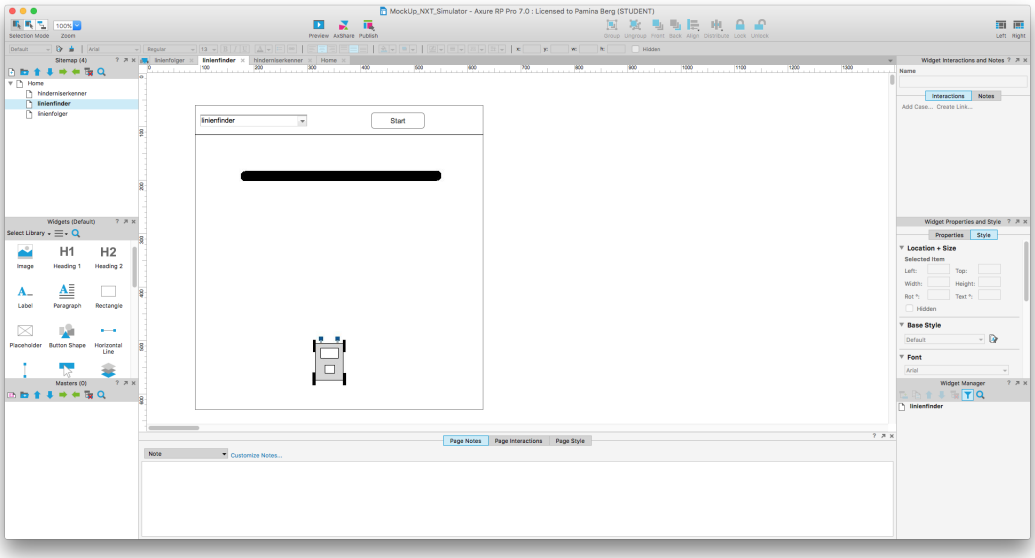

Abbildung 5.1: Die Bedieneroberfläche von Axure RP Pro

Da die zu schreibende API auch mehrere Sensoren umfasst, sollten die in den Mock-Ups dargestellten Szenarien den für den Schulkontext erforderlichen Funktionsumfang repräsentieren. Hierzu gehörte das Anhalten auf einer schwarzen Linie wie in Abbildung 5.2, das Fahren entlang einer schwarzen Linie (s. Abb. 5.3) und das Finden und Umfahren eines Hindernisses auf der Fahrbahn (s. Abb. 5.4).

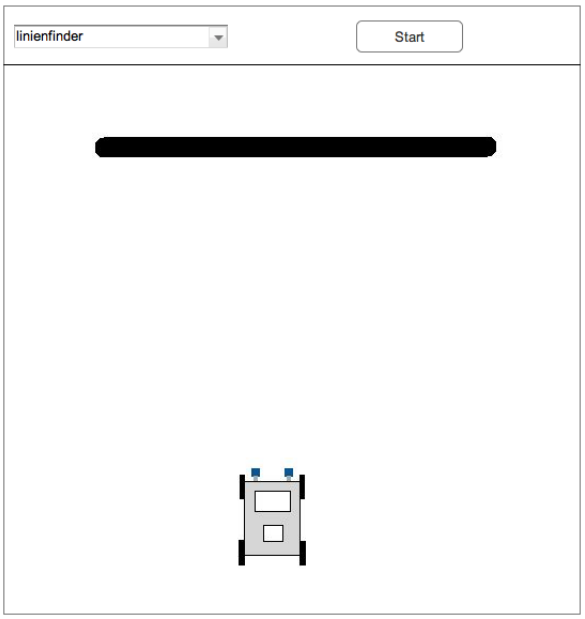

Abbildung 5.2: Anhalten auf einer schwarzen Linie

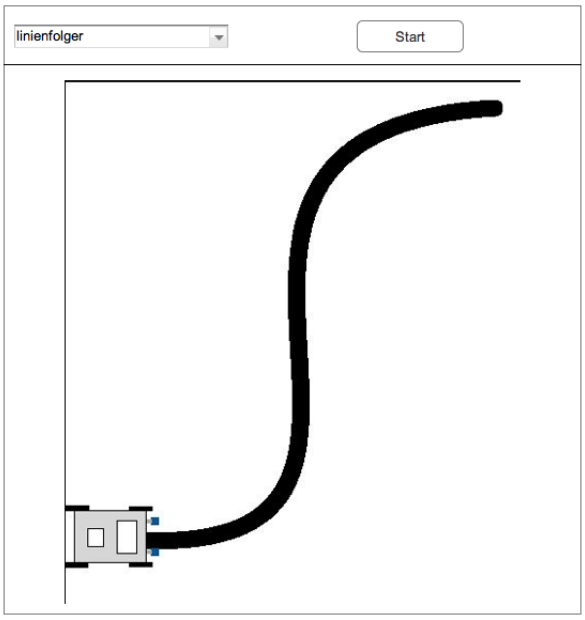

Abbildung 5.3: Fahren einer S-Kurve

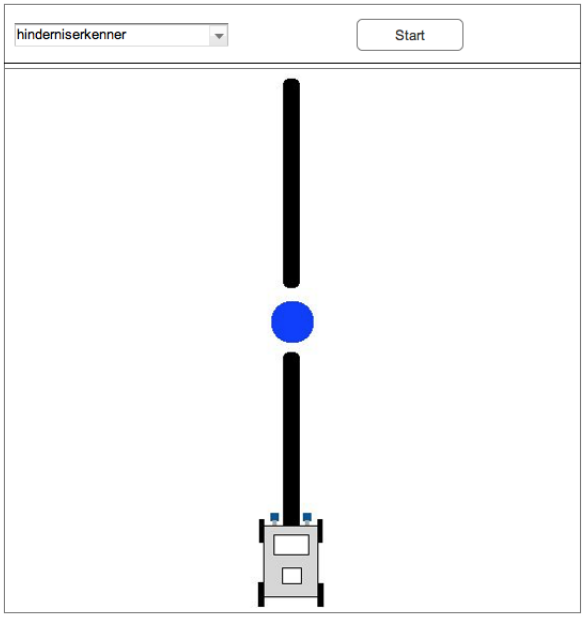

Abbildung 5.4: Erkennen eines Hindernisses auf der Fahrbahn

5.1.2. Schritte während der Implementation

Nach dem Erstellen der Mock-Ups musste nun überlegt werden, inwiefern die vorherigen Überlegungen umgesetzt werden konnten. Die erste Abstraktion, die während den ersten Implementationsschritten stattgefunden hat, war die Darstellung des Roboters in der Umgebung, sodass dieser einfach als Dreieck implementiert wurde.

Außerdem 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 verzahnen und die korrekten Beziehungen untereinander zu implementieren.

5.1.3. Integration in BlueJ

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. Da dies aber ein überschaubarer Aufwand für SuS und Lehrkraft ist, der auch als „Schablone“ für die ersten Versuche von der Lehrkraft zur Verfügung gestellt werden kann, sehe ich diese Lösung als geeignet an.

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. Dies wird von einer speziell für BlueJ entwickelten NXT-Extension (vgl. [Bow12]), die eine Erweiterung des Kontextmenüs jeder Klasse zur Verfügung stellt, realisiert.

5.2. Beschreibung der Software

Nun zu einer kurzen Beschreibung des Prototyps, in der zudem Einzelheiten zur Benutzung der Software dargestellt werden.

Beginnt man mit dem Programmieren einer Klasse, die den NXT Roboter beispielsweise einer Kurve auf dem Boden folgen lassen soll, so müssen neben des Imports für die leJOS API zwei weitere Import-Anweisungen für den Simulator eingebunden werden. Diese Klasse sei 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.

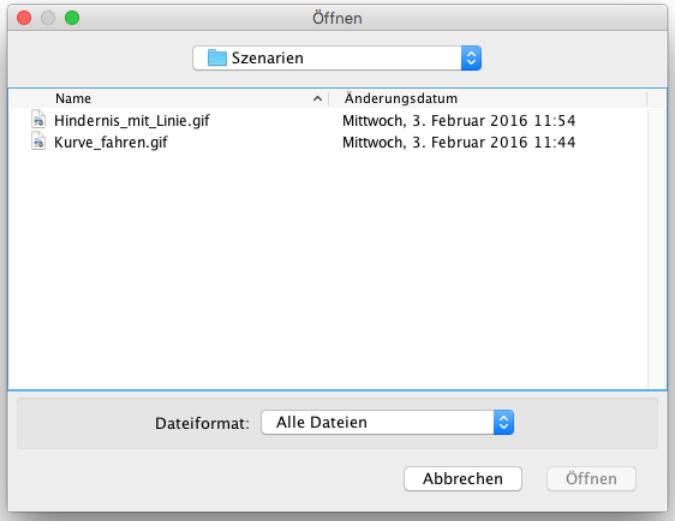

Abbildung 5.5: Auswahldialog für den Parcours

Um nun mit dem Simulator zu arbeiten, erzeugt man innerhalb der main-Methode der Klasse `linienfolger` ein neues Exemplar der Klasse `Simulator`. Nachdem man die Klasse `linienfolger` fehlerfrei kompiliert hat, kann man in BlueJ über einen Rechtsklick ein neues Exemplar dieser Klasse aufrufen, und an diesem die main-Methode ausführen lassen. Dies führt dazu, 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 von ihnen ausgewählten Parcours fährt (Vgl. Abb.5.6).

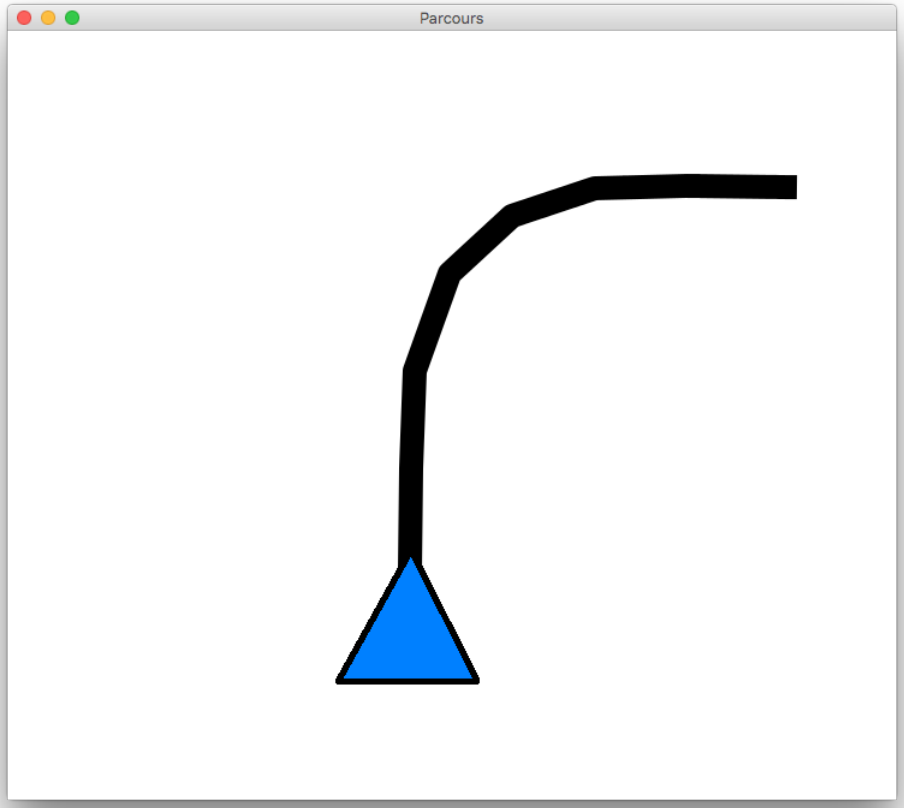

Abbildung 5.6: Eine beispielhafte Darstellung von Parcours und Roboter

5.3. Softwarearchitektur

Die entwickelte Software besteht im derzeitigen Zustand als Prototyp aus insgesamt fünfzehn Klassen. Diese sind zur Übersichtlichkeit in Pakete unterteilt. Diese unterscheiden sich in zwei Kategorien. Zum einen die Klassen, die die API für den Simulator zur Verfügung stellen und die zentralen von leJOS angebotenen Klassen für die NXT-Roboter umfassen, zum anderen die Klassen, die für die Repräsentation der Simulation auf dem Bildschirm zuständig sind. Um eine Konsistenz bei den Imports für die SuS zu bewahren, sind die Klassen, die 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 nun eine kurze Zusammenfassung der in dieser Implementation verwendeten Projektstruktur.

Pakete

Pakete in Java, auch *packages* genannt, 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 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 Darstellungoptionen des jeweiligen Betriebssystems gekoppelt, wodurch eine Eingrenzung 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`-Package

In diesem Abschnitt werden nun die von 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 den Prototyp angepasst wurden. Die interne Struktur zwischen den einzelnen Klassen ist durch die leJOS API vorgegeben und musste somit realitätsgetreu nachgebildet werden (Vgl. Abbildung 5.8).

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, die 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 haben hierbei Bewegungen der beiden Motoren über die Methoden `forward()` und `backward()`, sowie die 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 im

Prototyp durch eine Feldvariable `_port` gelöst, über die dann abgefragt werden kann, an welchem Motor gerade eine Methode aufgerufen wird. Folglich ist jedes NXT-RegulatedMotor-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 der Motor sich bewegt (vorwärts oder rückwärts), oder ob er gerade in Ruheposition ist. Dabei entspricht „vorwärts“ der Zahl 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 des Prototyps wird nun 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)` am Roboter, in Abhängigkeit zur Geschwindigkeit und eines Richtungsmultiplikators, 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 und es werden in der Konsequenz alle relevanten Werte in den Methoden gleichzeitig und gleichmäßig verändert.

Die Herausforderung bei diesem Prototyp bestand nun darin, dass die Motoren, auch wenn sie sich schon bewegen, eine neue Geschwindigkeit weitergegeben bekommen und die zuvor bestimmte Bewegung mit der neuen Geschwindigkeit ausführen. Außer-

dem 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 Prototyp der Simulation 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 Berechnung 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 $A = (x_A|y_A)$ und $B = (x_B|y_B)$.

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

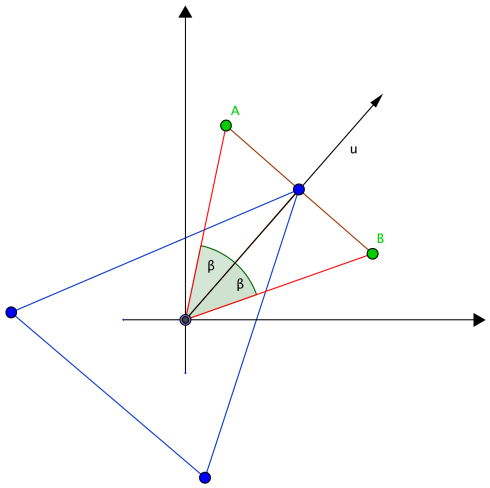

Abbildung 5.7: Die Abstraktion des Roboters zur Berechnung der Sensorpositionen

Die Klasse `Roboter` enthält die Methoden zur Abfrage der Positionen der Sensoren im Bezug auf die x- und y-Achse: `gibXLichtRechts()`, `gibYLichtRechts()`, `gibXLichtLinks()` und `gibYLichtsLinks()`. Diese werden benötigt, um das Auslesen des Farbwerts des Pixels 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. Da die realen Sensoren auch nicht nur einen einzelnen Punkt auf dem Boden, sondern einen Bereich abtasten, berechnet die Methode den Mittelwert 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.

Da sich die Farbsensoren in der Realität ähnlich verhalten wie die Lichtsensoren, ist die Implementation der Farbsensoren im Simulator code-identisch. Daher wird in dieser Ausarbeitung auf eine explizite Beschreibung verzichtet.

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 diesem Prototyp zunächst mit einer Farbunterscheidung gearbeitet. Dies funktioniert wiederum über den Helligkeitswert der Pixel, die sich gerade unter der Spitze des Roboter-Objekts befindet. Die Position des Sensors ergibt sich dabei in Anhängigkeit von 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

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

Für die Implementation des Ultraschallsensors wird eine Implementation des Hindernisses als Objekt auf dem Parcours benötigt. Dies ist für die Implementation eine ersten Prototypen des Simulators 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 genannten Sensoren ist an einen so genannten Sensorport angeschlossen. Die Klasse `SensorPort` enthält Felder, die die einzelnen Ports darstellen. Ein Exemplar einer Sensoren-Klasse wird bei der Initialisierung stets an einen Sensorport gekoppelt.

5.3.3. Die Klassen des `sim`-Pakets

Nun zu einer Beschreibung der Struktur der Klassen, die für die Simulation zuständig sind. Hierzu gehört die grafische Repräsentation des Parcours, des Roboters, der Auswahldialog für den Parcours, sowie die Leinwand, auf der der simulierte Roboter fahren soll. Eine Gesamtübersicht der wichtigsten Klassen ist in 5.9 abgebildet.

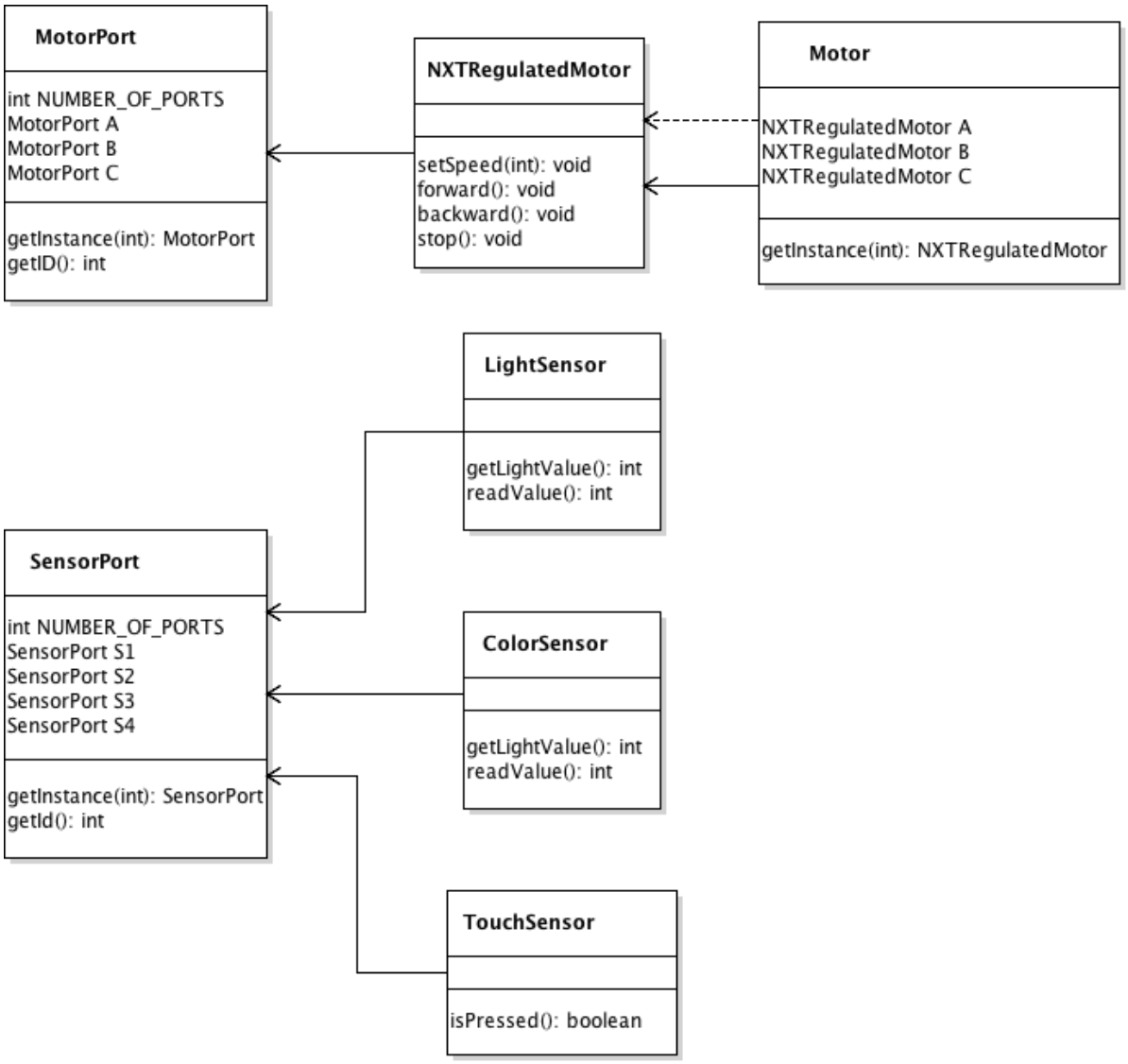

Abbildung 5.8: UML-Diagramm der wichtigsten von leJOS NXJ vorgegebenen Klassen

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()` eines Bildeinlesers wird ein JFileChooser erzeugt, mithilfe dessen der Benutzer eine Bilddatei auswählen kann, auf der der Roboter simuliert werden soll. Danach wird das Bild so umgewandelt, dass ein Array aus Bildpunkten entsteht. Dies ist hilfreich für die spätere Arbeit mit Bewegungen und dem Einsatz von Sensoren auf dem Parcours, 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 und die

Bilddaten an die an dieser Stelle erzeugten Leinwand zu übergeben. 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.

Des Weiteren 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 eben beschrieben, wird die Leinwand von dem Parcours erzeugt. Die Klasse Leinwand sorgt nun dafür, dass die als Array übergebenen Bildpunkte in einem JFrame, also vereinfacht einem neuen Fenster, angezeigt werden.
Des weiteren bietet die Leinwand eine Methode `warte()` an, die dafür genutzt wird, dass sich die Objekte auf der Leinwand tatsächlich 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` – und 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 anzumerken ist, dass

die Bilddatei einen imaginären Rahmen besitzt, so dass zwei Feldvariablen zur Positionskorrektur existieren, die für das Zeichnen auf der Leinwand benutzt werden. So ist sichergestellt, dass der Roboter auch tatsächlich an der übergebenen Position gezeichnet wird. Beim Konstruktoraufruf übergibt der Benutzer demnach 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, also 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 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 Geschwindigkeit. 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, können wir direkt darauf zugreifen. Der in der Variablen `_winkelVeränderung` gespeicherte Wert wird nun zunächst auf die Feldvariable `_ausrichtung` aufaddiert. Sei nun α der Wert der Variablen `_ausrichtung` und v die in der Variable `_geschwindigkeit` gespeicherte Geschwindigkeit. Des Weiteren 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 lauten $(x|y)$.

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.

Des Weiteren enthält die Klasse `Roboter` eine Vielzahl von sondierenden Methoden, die 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, relativ flexibel die Bilddatei des Roboters auszuwechseln (vgl. **4.3**).

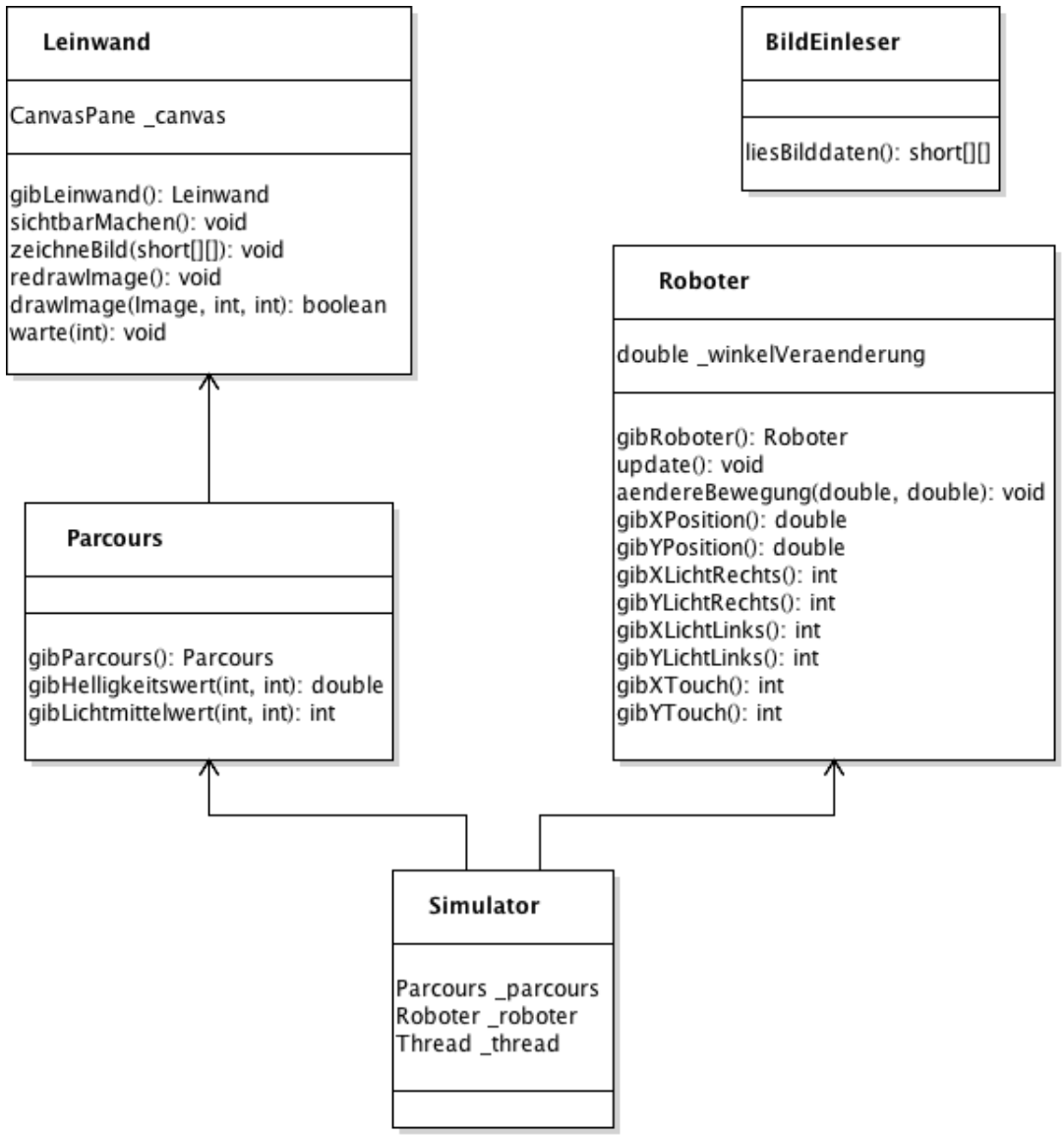

Abbildung 5.9: UML-Diagramm der essentiellen Klassen für die Simulationsumgebung

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 zur Steuerung des Roboters an die Hand zu geben. Dieser Prototyp der Simulationsumgebung umfasst alle für die Programmierung erster Software mit NXT Robotern essentiellen Klassen und 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 einfach nur zwei 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 der Größe 740 x 640 Pixel entwerfen und den SuS zur Verfügung stellen. Es muss lediglich darauf geachtet werden, dass der Startpunkt der Roboters am Punkt (335 | 450) liegt und ggf. angepasst werden muss. Auch diese Informationen können in der Kurzanleitung nachgelesen werden. 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 hinsichtlich der Schüler, Lehrer sowie der Erweiterbarkeit konnten demnach in einem guten Maße umgesetzt werden.

6.2. Verbesserungen und Erweiterungen

Da die entwickelte Simulationsumgebung bisher nur ein Prototyp ist, sollte sie folglich als nächstes in der Praxis getestet werden. Hierzu kann sie verschiedenen Lehrkräften zur Verfügung gestellt werden, die dann in unterschiedlichen Jahrgangsstufen damit arbeiten können. Dieser Schritt bewirkt, dass das Verhalten der Sensoren verfeinert und etwaige Fehler, die erst beim Testen mit von SuS geschriebenem Code auftreten, im Code lokalisiert und behoben werden können.
Damit einher geht auch der Umstand, dass das geforderte Verhalten der Sensoren bisher nur in einem beschränkten Maße getestet wurde. Dies bedeutet, dass eine hinreichend gute Approximation des Verhaltens der Sensoren beim Testen mit einem Mustercode erreicht wurde, dieser Mustercode aber nicht unbedingt den Realbedingungen entspricht.

Des weiteren kann die Roboter-Grafik im Simulator, die bisher erst einmal zu Testzwecken aus einem Dreieck besteht, durch eine beliebige andere Grafik ausgetauscht werden, um den SuS das Gefühl, dass „ihr“ Roboter im Simulator gerade die ersten Bewegungen vollführt, zu vermitteln.

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

Etwas umfassender, dafür aber eine schönere Integration in BlueJ mit sich bringend, wäre noch das Schreiben einer eigenen BlueJ Extension. Hierdurch würde die Möglichkeit entstehen, den Simulator gegebenenfalls direkt am Kontextmenü aufzurufen.

6.3. Rückmeldung von Lehrkräften nach Vorstellung des Prototyps

Im Rahmen dieser Arbeit wurde der entwickelte Prototyp der Simulationsumgebung 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 gegebenenfalls mit Lösungsansätzen für den Prototyp angereichert.

anderes Wort

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 Roboers stattfinden.

Des Weiteren wurden Anmerkungen zur Robotergrafik gegeben. Zunächst sollte 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.

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 auf 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.

6.4. Zusammenfassung

Der in dieser Masterarbeit entwickelte Prototyp einer 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 Prototyp eine Basis für Erweiterungen bezüglich UI-spezifischer Anpassungen oder auch für ein Upgrade auf die nächste Generation von LEGO MINDSTORMS Robotern – den EV3 – dar und lässt sich durch den objektorientierten Ansatz in der Programmierung und die Nutzung von Paketen, die in sinnvolle Kategorien unterteilt wurden, relativ einfach umgestalten und erweitern.

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

[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

[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

"Hiermit versichere ich, 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, 27. Februar 2016 .....

Pamina Maria Berg