Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

Entwicklung einer Java Simulationsumgebung für Lego Mindstorms NXT Roboter

Masterarbeit zur Erlangung des akademischen Grades Master of Education
im Studiengang Lehramt an Gymnasien M.Ed.

Pamina Maria Berg

Hamburg, den 7. Februar 2016

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

ABBILDUNGSVERZEICHNIS

INHALTSVERZEICHNIS

KAPITEL 1

EINLEITUNG

Informatik in der Schule ist eines der bildungspolitisch umfassend diskutierten Themen der letzten Jahre. Gerade die Fragen, ob es sinnvoll sei, ein Unterrichtsfach *Programmieren* ab der Grundschule einzuführen, oder aber ob es reicht, dass die Schülerinnen und Schüler dazu ausgebildet werden, einen Computer bedienen zu können, nehmen dabei eine zentrale Rolle ein. Dabei klaffen 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.

Zumindest in der 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, zu erreicht werden könnte, wenn dies als Spiel angeboten würde. Hierdurch entstand das bis heute bekannte Konzept der *Turtle* [Nie99, S.365].
Heutzutage geschieht dies im Zusammenhang mit LEGO MINDSTORMS Robotern zum einen mit Drag-and-Drop Softwareangeboten wie das standardmäßig mit ausgelieferte *NXT Mindstorms Tool* (s. **2.3.1**), oder auch *Enchanting* (s. **2.3.2**). 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. **2.3.3**) arbeitet.

Doch immer wieder stoßen Lehrkräfte in den Schulen auf Hardwareprobleme jeglicher Art, wie zum Beispiel das Fehlen von einer ausreichenden Anzahl an Robotern im Unterricht oder auch fehlende Firmwareupdates oder defekte Sensoren.
Auch ist das Zusammen- und wieder Auseinanderbauen der Roboter ein Aufwand, der für den alltäglichen Unterricht nicht geeignet ist. Um kleine Aufgaben mit Java zu lösen, wie zum Beispiel das Fahren einer Kurve oder das Anhalten auf einem bestimmten Punkt, muss bisher immer erst der Roboter gestartet, der Code in BlueJ verfasst, auf den Roboter übertragen und dieser dann zu einem geeigenete Parcours gebracht werden.

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.

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 [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 [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 auch, 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 [Lego].

2.1.1. Hardware

Zentraler Bestandteil der LEGO MINDSTORMS NXT Roboter ist der sogenannte NXT-Stein. 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 [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 [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 jeweils mithilfe des Programmcodes kontrolliert werden kann [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 kann auch, wie bei [Wag05] 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 an-

> deren, 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.

Gymnasium Sekundarstufe I (7 – 10) Informatik Wahlpflicht

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 Sekundarstufe I (7 –11) Informatik

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*

unterrichtet werden [HH14].

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 und das 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 so genannten *Use Cases*. Hierbei handelt es sich um aus der Realwelt gegriffene Umgebungen, die in unserem Fall mithilfe des Roboters erkundet werden sollen. Ein klassischer Use Case 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 Use Case 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.

2.3. 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.

2.3.1. LEGO Mindstorms NXT

Das LEGO MINDSTORMS NXT-Programm ist eine von der Firma LEGO zur Verfügung gestellte Programmieroberfläche für NXT-Roboter.
Die Abbildung 2.1 zeigt den Startbildschirm der Entwicklungsumgebung. Dieser stellt

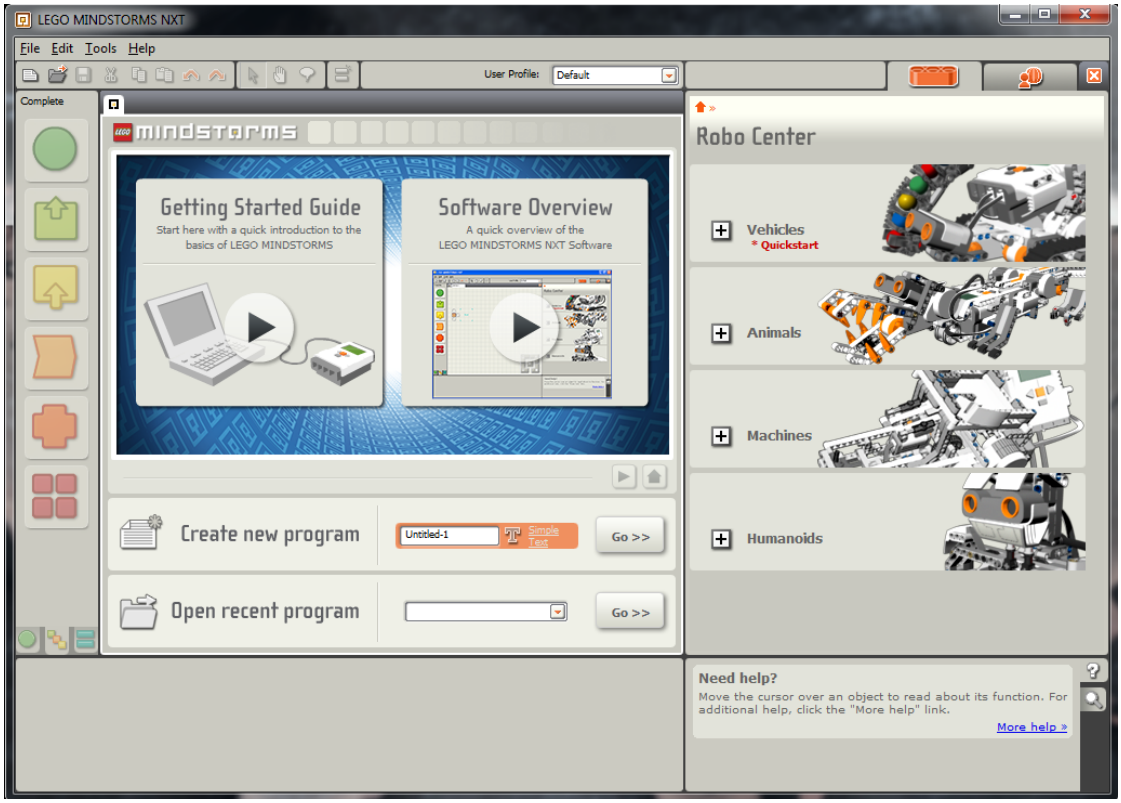


Abbildung 2.1: Der Startbildschirm des LEGO MINDSTORMS NXT-Programms

neben zwei Videoanleitungen als Hilfe für Anfänger auch das *Robo Center* zur Verfü-

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

Sobald eine neue oder bestehende Programmdatei geöffnet wird, können die SuS mithilfe von Blöcken, die unter anderem der Bewegung und Sensorik dienen und dabei einfache Steuerungsmechanismen sowie Elemente der Regelungstechnik zur Verfügung stellen, ihren Roboter programmieren. Hierbei 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 (Vlg. Abb.2.2).

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.

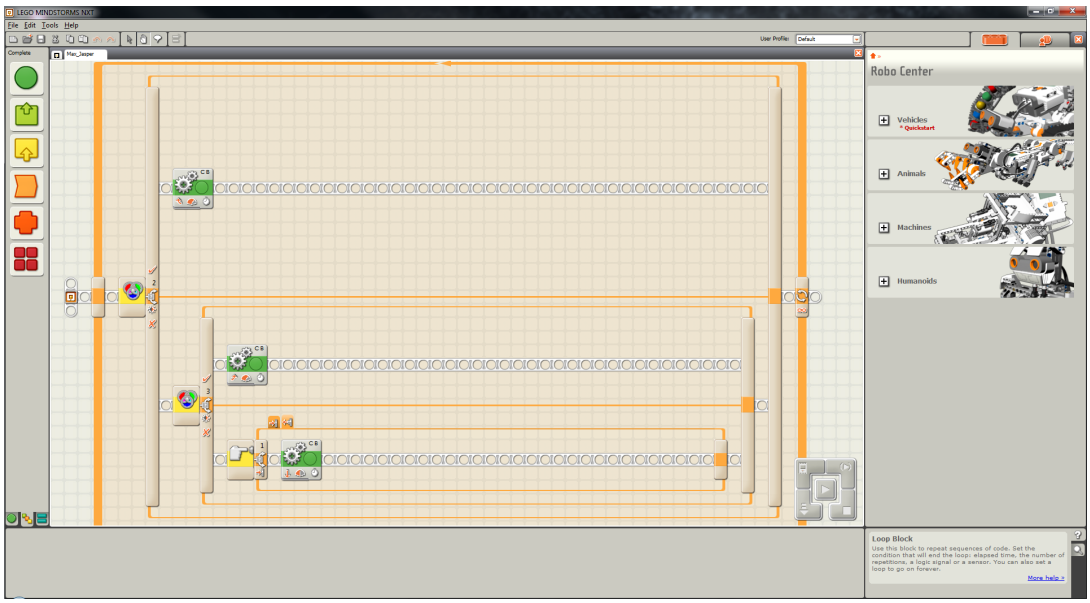


Abbildung 2.2: Fahren eines Roboters auf einer schwarzen Linie

2.3.2. Enchanting

Enchanting ist eine an das Einsteigertool für Objektorientierte Programmierung *Scratch* anknüpfende Entwicklungsumgebung. Wie das in **2.3.1** 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. 2.3.3) ersetzt. Dies sorgt dafür, dass der NXT nun auf die Programmierung mit Java reagiert.

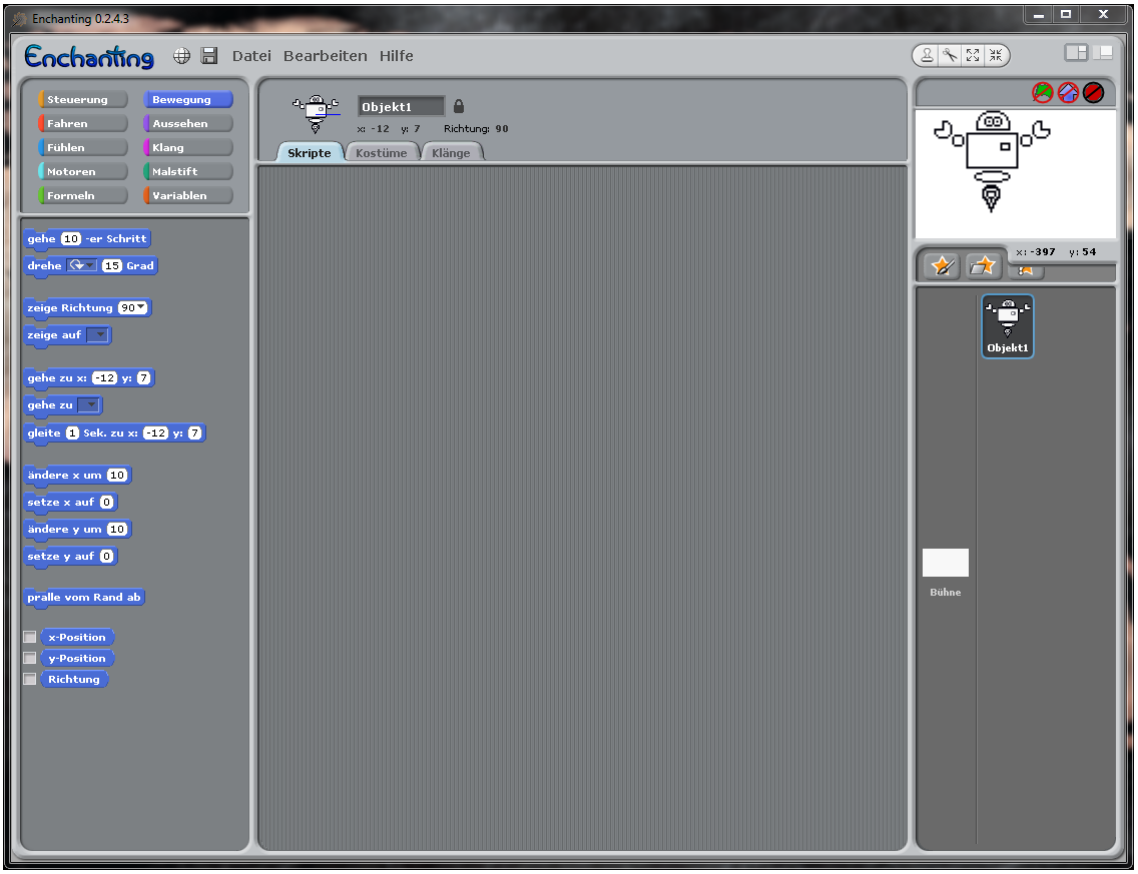


Abbildung 2.3: Der Startbildschirm von Enchanting

Wird Enchanting von den SuS geöffnet, erscheint zunächst der Startbildschirm wie in Abb. 2.3. Nun kann entweder ein neues Programm erstellt oder ein bestehendes geöff-

net 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 ganz einfach, indem z.B. man 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 if-Abfragen und anderen Steuerungselementen mithilfe des Namens auf den Motor oder Sensor zugegriffen werden.

In Abbildung 2.4 ist nun ein simples Programm zum Fahren entlang einer schwarzen

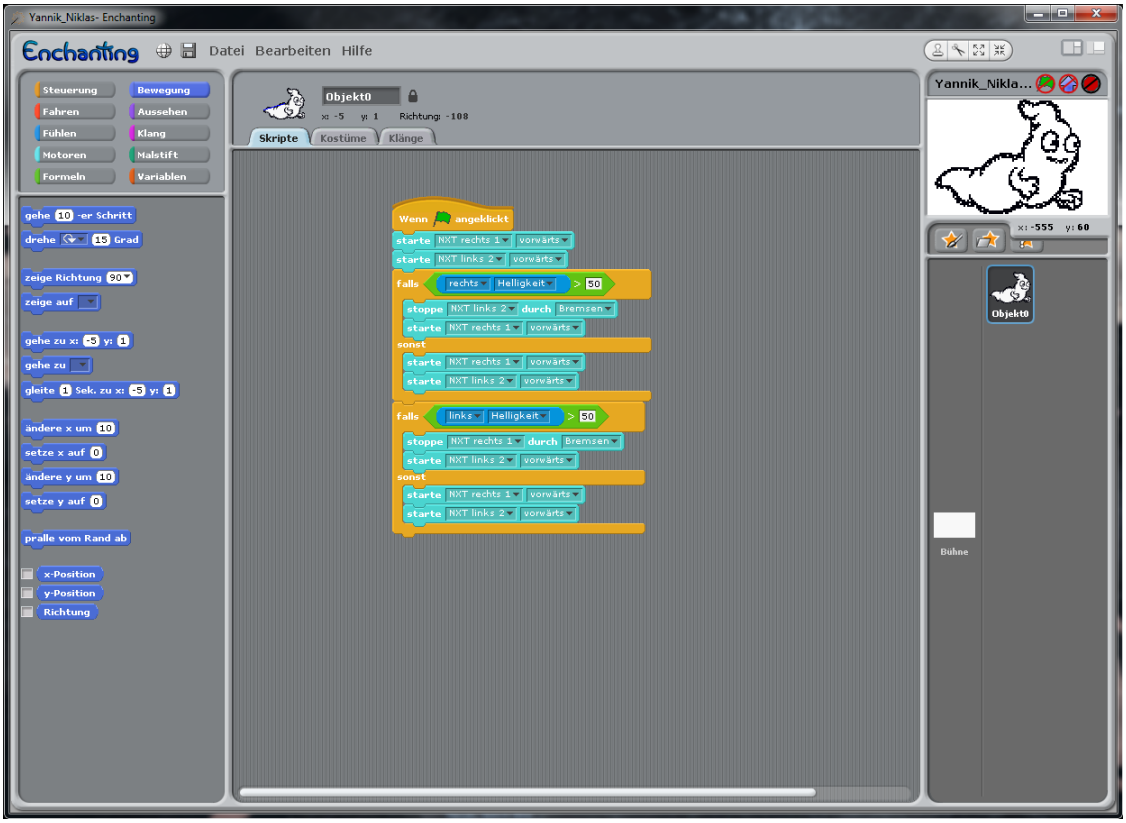


Abbildung 2.4: Beispiel eines Enchanting Programms

Linie mithilfe von zwei Lichtsensoren dargestellt. Hierbei ist zu erkennen, dass die Sensoren in den `falls...sonst...`-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.

2.3.3. Schriftliche Programmierung von NXT-Robotern

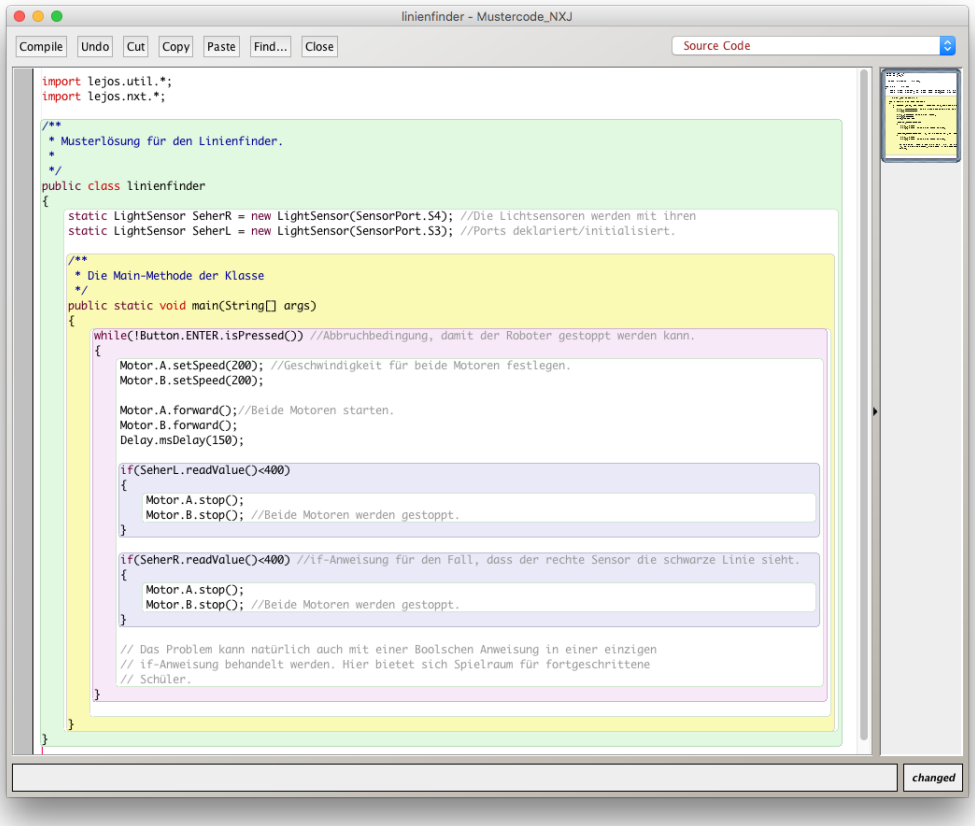
Neben den grafischen Entwicklungsumgebungen in **2.3.1** und **2.3.2** stehen natürlich auch Werkzeuge zur schriftlichen Programmierung der NXT-Roboter zur Verfügung. Nachfolgend wird eine der geläufigen Kombinationen aus Entwicklungsumgebung und virtueller Maschine für LEGO MINDSTORMS NXT vorgestellt.

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 [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.

noch etwas unvollständig




```
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 2.5: BlueJ-Beispiel zum Finden einer Linie

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 [leJOS].

In der Kombination mit BlueJ ergibt sich somit für die SuS eine besonders einfache Softwarelösung. Durch eine eigens für BlueJ geschriebene sogenannte *Extension* und den Import der NXJ API stehen den SuS nicht nur die Klassen zur Steuerung ihres Roboters zur Verfügung, sondern auch eine Möglichkeit, über eine Erweiterung des Menüs, was allen in BlueJ geschriebenen Klassen zur Verfügung steht, um spezielle Befehle, die

Quelle

dazu dienen, das in BlueJ geschriebene Programm auf den NXT-Stein zu übertragen.

hier könnte noch eine Grafik eingefügt werden - evt. anstatt einer der anderen Grafiken

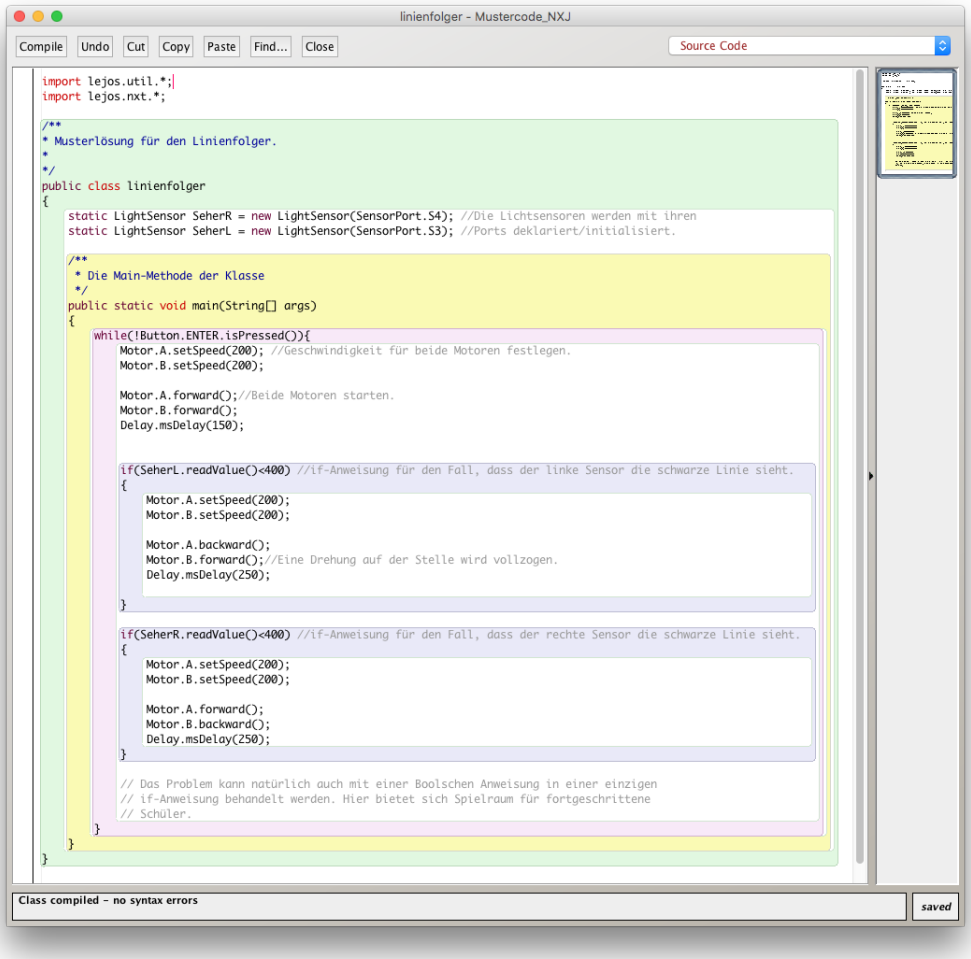


Abbildung 2.6: Der Roboter soll einer schwarzen Linie folgen

Mithilfe von BlueJ (oder einer anderen Entwicklungsumgebung) können die SuS insbesondere Aufgaben lösen, wie zum Beispiel das Anhalten auf einer Linie (Vgl. Abb. 2.5) oder auch den Roboter einer schwarzen Linie folgen zu lassen, was eine der Grundaufgaben im RoboCup Junior Rescue Wettbewerb darstellt (Vgl. Abb. 2.6).
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.

2.3.4. Simulationsumgebungen

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

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 [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, so dass diese inzwischen auch direkt von Roberta als Softwarelösung angeboten wird [Roberta].

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

KAPITEL 3

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]

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 und der Lehrer dargestellt, sowie Parallelen und Unterschiede zu den oben dargestellten herausgearbeitet.

3.1. 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 ET AL., 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 den Roboter selbst benutzen können sollten.

Ein weiterer Aspekt, den HERTZBERG ET AL. beschreiben sind die Parcours selbst. 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.

3.2. 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. Hierzu 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 S-Kurve eine Teilstrecke des realen Labyrinths darstellt, bestehen zu lassen.
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.

3.3. Erweiterbarkeit

Die Software sollte ausreichend kommentiert werden, damit auch nach Fertigstellung dieser Masterarbeit weitere Module hinzugefügt werden können. Dies betrifft sowohl die Fähigkeiten der Sensoren, sowie Änderungen, die im Rahmen von Veränderungen der leJOS NXJ API stattfinden, als auch die Möglichkeit, die geschriebene API an die Technologie der LEGO MINDSTORMS EV3, den Nachfolge-Robotern der NXT, anzupassen.
Des weiteren sollte die Möglichkeit gegeben sein, dass Lehrkräfte unter bestimmten Voraussetzungen eigene, an ihren Unterricht angepasste Parcours in die Software einbinden können. Hierzu wird ein einheitlicher Startpunkt des Roboters im Parcours von der Software festgelegt und in einer Kurzanleitung niedergeschrieben.

KAPITEL 4

ENTWICKELTE SOFTWARE

4.1. Prozesse während der Implementation

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

4.1.1. Mock-Ups

Um zunächst einen visuellen Grundgedanken festzuhalten und das Ziel der Implementation der vollständigen Simulationssoftware zu erfassen, wurden interaktive Mock-Ups erstellt. Hierzu wurde die Software *Axure RP Pro* benutzt, mit der sich sowohl einfache als auch komplex verschachtelte Mock-Ups erstellen lassen (s. Abb.4.1).

Da die zu schreibende API auch mehrere Sensoren umfasst, sollten die in den Mock-Ups beschriebenen Szenarien den für den Schulkontext erforderlichen Funktionsumfang repräsentieren. Hierzu gehörte das Anhalten auf einer schwarzen Linie wie in Abbildung 4.2, das Fahren entlang einer schwarzen Linie (s. Abb. 4.3) und das Finden und Umfahren eines Hindernisses auf der Fahrbahn (s. Abb. 4.4).

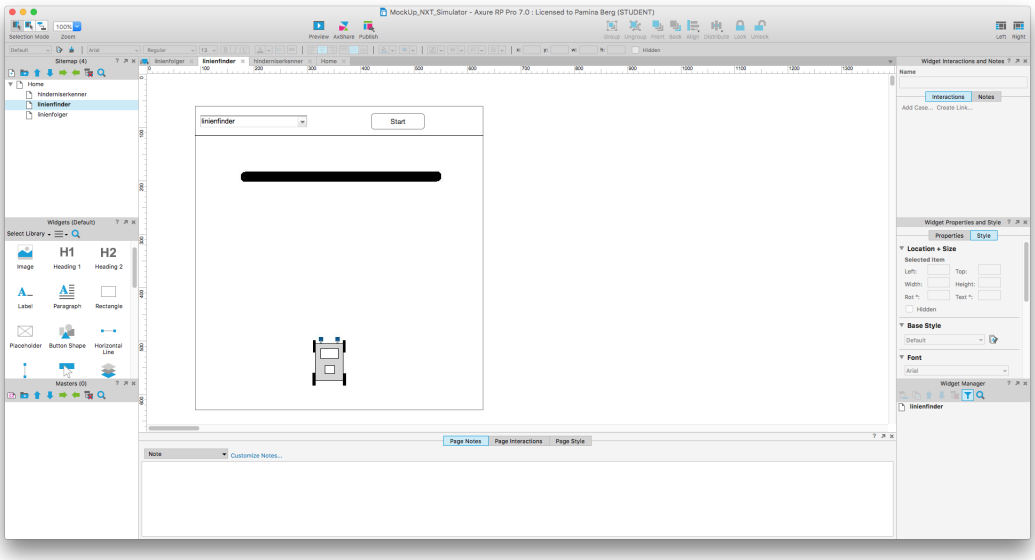


Abbildung 4.1: Die Bedieneroberfläche von Axure RP Pro

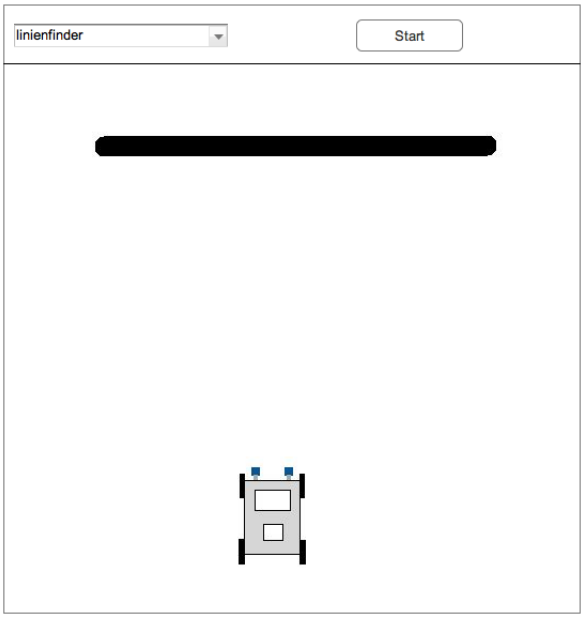


Abbildung 4.2: Anhalten auf einer schwarzen Linie

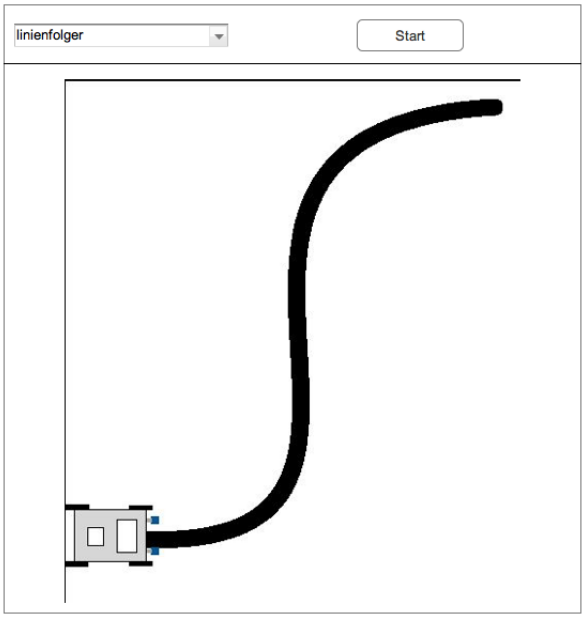


Abbildung 4.3: Fahren einer S-Kurve

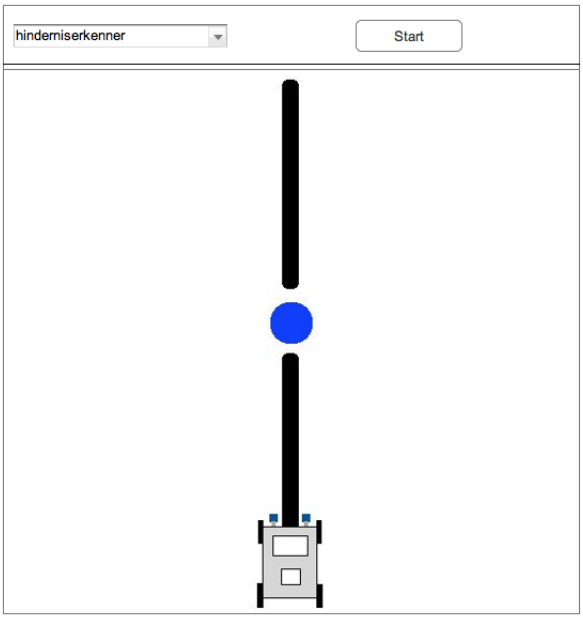


Abbildung 4.4: Erkennen eines Hindernisses auf der Fahrbahn

4.1.2. Schritte während der Implementation

Nach dem Erstellen der Mock-Ups musste nun überlegt werden, inwiefern die Überlegungen umgesetzt werden konnten. Die erste Abstraktion, die während den ersten Implementationsschritten stattgefunden hat, war die Darstellung des Roboters in der Umgebung, so dass 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 **4.3.1** beschriebenen Klassen realitätsgetreu miteinander zu verzahnen und die korrekten Beziehungen untereinander zu implementieren.

4.1.3. 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 sie, neben der Import-Anweisung für die leJOS-Bibliothek einen weiteren Import 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.

Nun kann man sich die Frage stellen, wieso nun trotzdem mit BlueJ gearbeitet werden soll, da es doch ausreichend andere Entwicklungsumgebungen gibt, in die solche Bibliotheken auch ganz simpel importiert werden können.
Der Vorteil an BlueJ besteht jedoch klar darin, dass diese Umgebung, wie in **2.3.3** beschrieben, eine klar strukturierte und übersichtliche Benutzeroberfläche bietet. Die SuS müssen sich nicht in für ihre Ansprüche überdimensional umfangreiche Programmierumgebungen einarbeiten und können, gerade weil speziell für BlueJ eine NXT-

Extension entwickelt wurde (Vgl. [Bow12]), die eine Erweiterung des Menüs, das an jeder Klasse durch Rechtsklick aufrufbar ist, um die Übertragung des Codes per USB-Kabel auf den Roboter zu starten, zur Verfügung stellt.

4.2. Beschreibung der Software

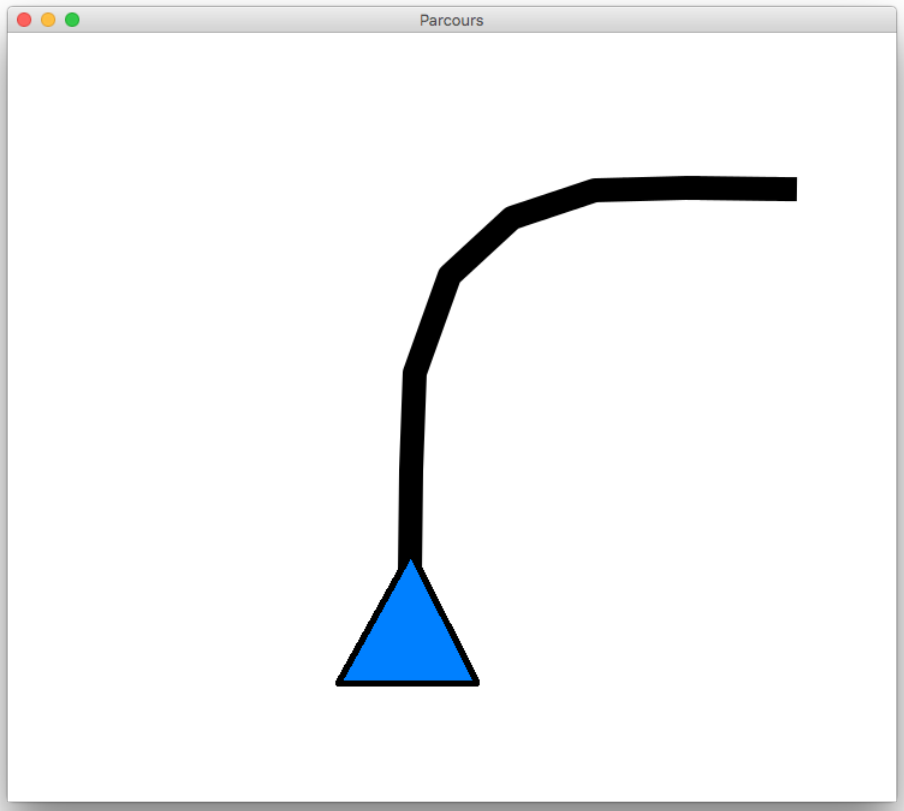


Abbildung 4.5: Eine beispielhafte Darstellung von Parcours und Roboter

4.3. Softwarearchitektur

Die entwickelte Software besteht im derzeitigen Zustand als Prototyp aus insgesamt dreizehn Klassen. Diese lassen sich in zwei Kategorien aufteilen. 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.

4.3.1. Klassen der API

In diesem Abschnitt werden nun die von leJOS vorgegebenen und an 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.

Motoren

Zunächst wurde die Klasse `Motor` implementiert. Diese besteht lediglich aus den Feldern `A, B` und `C`, die Exemplare von der Klasse `NXTRegulatedMotor` darstellen. Ein NXTRegulatedMotor ist ein an einen Motorport angeschlossener NXT-Motor.Ein an einen Motorport angeschlossener Motor wird über den Befehl `Motor.[port].[Methode]` angesteuert. Die Klasse `MotorPort` ist in diesem Projekt eine verkleinerte Version der gleichnamigen Klasse aus der leJOS API.

NXTRegulatedMotor

In der Klasse `NXTRegulatedMotor` befindet sich die Implementation der aufzurufenden Bewegungen des Roboter-Objekts der Simulation.

Jeder NXTRegulatedMotor ist an einen Port 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.
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 signalisiert.

Die SuS werden ihrem geschriebenen Code meist damit beginnen, den beiden Motoren jeweils eine Geschwindigkeit zuzuweisen. Dies geschieht mithilfe der Methode `setSpeed(int speed)`.
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 (*C* entspricht dem linken Motor, *B* dem rechten) angesteuert wird, ruft die Klasse NXTRegulatedMotor die Methode `aendereBewegung` 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 Herausforderung in diesem Prototyp bestand nun darin, dass die Motoren, auch wenn sie sich schon bewegen, eine neue Geschwindigkeit an den Motor weiterzugeben und die zuvor bestimmte Bewegung mit der neuen Geschwindigkeit auszuführen. 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 Motor am angeschlossenen Port 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 im Abstand von fünf Pixeln links und rechts von der Spitze des Roboter-Dreiecks angebracht. Da es sich bei der Implementation um einen Prototyp handelt, gibt es 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: `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 diesem steht die Methode `gibHelligkeitswert(int x, int y)` zu Verfügung, die auf das Bilddaten-Array zugreift.

Die Implementation der Farbsensoren ist im Simulator code-identisch, daher kann in dieser Ausarbeitung auf eine explizite Beschreibung 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 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.

Für die Implementation des Ultraschallsensors wird eine Implementation des Hindernisses als Objekt auf dem Parcours benötigt. Dies ist nicht essentiell für die Implementation eine ersten Prototypen des Simulators, 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.

4.3.2. Die Simulationsumgebung

Nun zu einer Beschreibung der Struktur der Simulationsumgebung selbst. Hierzu gehört die grafische Repräsentation des Parcours, des Roboters, der Auswahldialog für den Parcours, sowie die Leinwand, auf der der simulierte Roboter fahren soll.

Simulator

Der Simulator erzeugt beim Ausführen zunächst einen `BildEinleser`. Dieser lässt den Benutzer in einem Dialogfenster eine .gif-Datei als Parcours auswählen. Das ausgewählte Bild wird dann „eingelesen“ und die einzelnen Bildpunkte in einem short-Array gespeichert.

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 API-Klassen, da sie dafür zuständig ist, durch den Aufruf von `liesBilddaten()` am BildEinleser den Dialog für die Auswahl der Parcours-Grafikdatei zu erzeugen, die Bilddaten an die an dieser Stelle erzeugten Leinwand zu übergeben und Informationen über den Parcours an die Sensoren weiterzuleiten.

Insbesondere die Abfrage des Helligkeitswerts eines Pixels aus dem Array ist in dieser Klasse implementiert, sowie die Ausgabe der realitätsgetreueren Werte für die Lichtsensoren.

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.

Roboter

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

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

Neben dem `update()` muss der Roboter natürlich noch eine Methode zur Verfügung stellen, die es den Motoren ermöglicht, die Bewegungssteuerung zu übernehmen. 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.

Abschließend enthält die Klasse `Roboter` eine Vielzahl von sondierenden Methoden, die Positionskoordinaten des Roboters, sowie der Lichtsensoren und des Berührungs-

sensors 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 kann, gearbeitet wurde, besteht die Möglichkeit, relativ flexibel die Bilddatei des Roboters auszuwechseln.

KAPITEL 5

ZUSAMMENFASSUNG UND AUSBLICK

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

[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

[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

[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, 7. Februar 2016 .....

Pamina Maria Berg