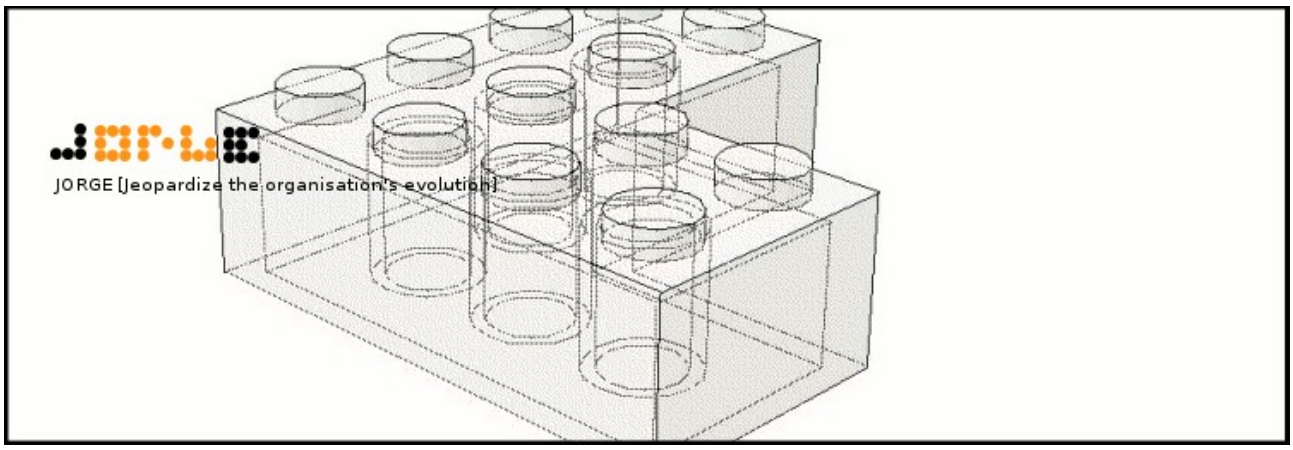
Berner Fachhochschule
Hochschule für Technik und Informatik HTI



Emulator Evaluation

[Jorge - Lego MindStorms]

Autoren	Nik Lutz [I3STW, lutzn@hti.bfh.ch]
	Stefan Feissli [I3STW, feiss@hti.bfh.ch]
	Christof Seiler [I3STW, seilc@hti.bfh.ch]
Version	0.5
Datum	15.05.2005
Status	In Arbeit
Webseite	jorge.hta-bi.bfh.ch
Projektauftraggeber	Claude Fuhrer
PM-Coaching	Frank Helbling
Projektarbeit	Sommersemester 2005, HTI Biel, Abt. Informatik

Inhaltsverzeichnis

1 Zweck des Dokuments

Damit ein Benutzer das Verhalten seiner Programme (mit Jorge) simulieren kann, muss ein Weg gefunden werden, um die vom Benutzer gschriebenen Programme auszuführen oder zu interpretieren.

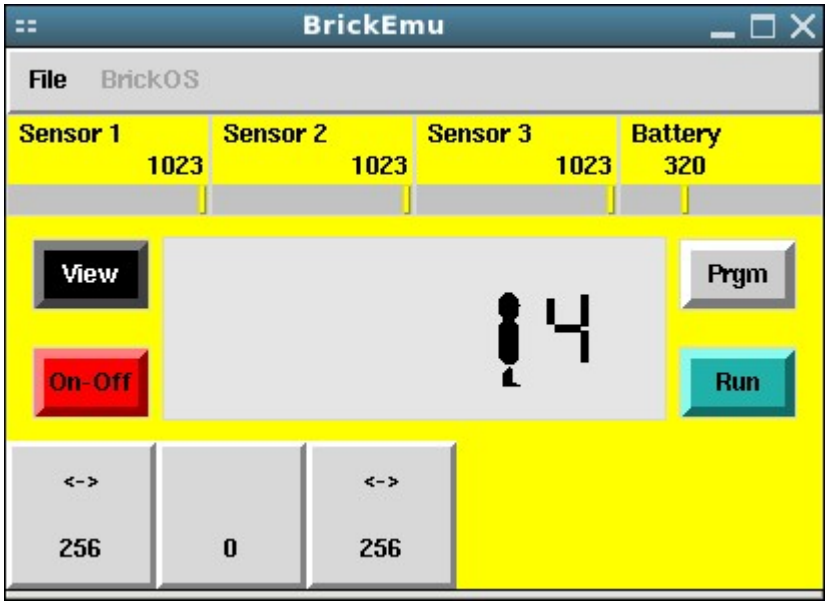
2 Kriterien

- **LeJos:** Es muss möglich sein, Programme die auf dem LeJos-API aufbauen auszuführen oder zu interpretieren.
- **Einfache Integration:** Wird bestehender Code verwendet, sollte dieser in C oder C++ geschrieben worden sein .
- **Platform:** Die gewählte Lösung sollte im minimum unter Windows, wenn möglich aber auch unter Linux und MacOsX compilierbar sein.
-

3 Evaluation

3.1 BrickEmu

BrickEmu (`http://csd.informatik.uni-oldenburg.de/~hoenicke/rcx/`) ist ein Emulator, der direckt RCX-Code ausführen kann.



3.1.1 Vorteile

- BrickEmu ist ein Emulator, dies hat den Vorteil, dass verschiedenste RCX-Betriebssysteme eingesetzt werden können.
- BrickEmu ist gut Dokumentiert.
- BrickEmu läuft auf Linux und Windows (Cygwin)
- BrickEmu kann Infrarot-Üebertragungen simulieren.
- BrickEmu hat Debugging support.

3.1.2 Nachteile

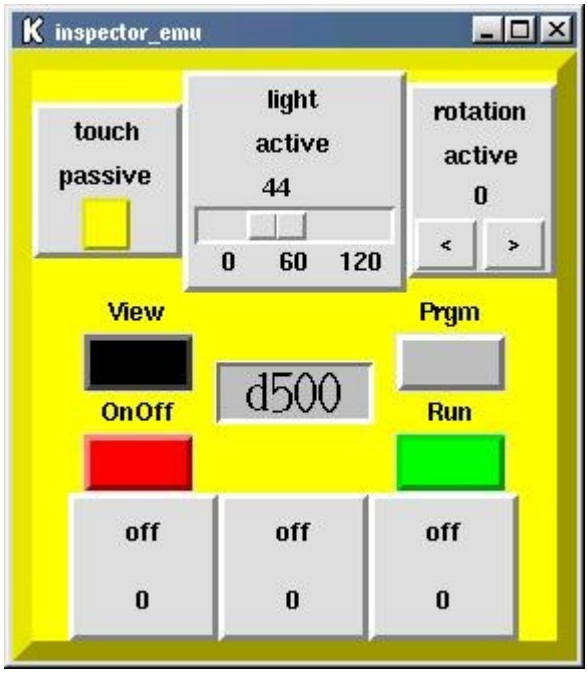
- Code: BrickEmu ist zum Teil in C, aber auch in Perl und in Tcl/Tk implementiert.
- Zum Ausführen des Codes ist ein ROM-Image erforderlich, das aus Lizenz-Gründen nicht weiterverteilt werden darf. Um ein ROM-Image selber zu generieren, muss der Benutzer einen echten RCX besitzen, und ein Program ausführen: Leider ist dieses Programme nur für die serielle Infrarot-Schnittstelle geschrieben worden.
- Die Anleitung auf der Homepage von BrickEmu zeigt wie man BrickEmu mit BrickOs benutzt, leider konnten wir kein LeJos-Programm mit BrickEmu zum laufen bringen (Diverse nicht nachvollziehbare fehler).

3.1.3 Fazit

Da wir es nicht schafften BrickEmu zur Ausführung von LeJos-Programmen zu bewegen, und sich die Fehler-Suche als ziemlich schwierig herausstellte, schied BrickEmu trotz seiner Vorteile aus.
Könnte der Fehler, der BrickEmu an der Ausführung von LeJos-Programmen hinderte, gelöst werden, müsste eine Lösung für die in Perl geschriebenen Methoden von BrickEmu gefunden werden: Neu schreiben in C/C++.
Im weiteren müsste das Program zum generieren des ROM-Images angepasst werden, und zwar für Windows, Linux und MacOsX. Da dieses Program die Infrarot-Hardware (LegoTower) anspricht, hätte für jede Platform eine eigene Lösung gefunden werden müssen, was mit einem nicht einschätzbaren Zeitaufwand verbunden ist.

3.2 EmuLegOS

EmuLegOS (`http://emulegos.sourceforge.net/`) emuliert legOS Programme.



3.2.1 Vorteile

- In C++ geschrieben (Gui in Tcl/Tk).
- Läuft unter Windows (Cygwin) und Linux

3.2.2 Nachteile

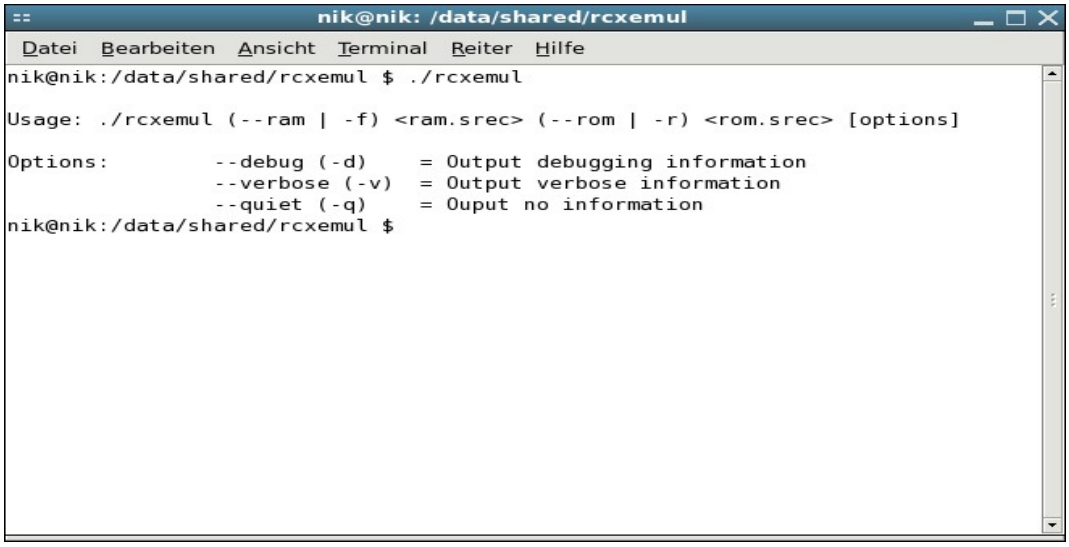
- LeJos-Programme können nicht ausgeführt werden.

3.2.3 Fazit

Da keine LeJos-Programme ausgeführt werden können, kann EmuLegOS für unser Projekt nicht eingesetzt werden. Zudem funktioniert das mitgelieferte Makefile nicht, so dass wir EmuLegOS selber gar nie starten konnten(Das Bild oben stammt von der EmuLegOS Homepage).

3.3 RCXEmu

RCXEmu (http://sourceforge.net/projects/rcxemul) emuliert den RCX-Brick, d.h. Wie BrickEmu können verschiedenste RCX-Betriebssysteme auf dem Emulator installiert werden, welche dann die gewünschten Programme ausführen.



3.3.1 Vorteile

- Komplett in C geschrieben, wenig Code

3.3.2 Nachteile

- Es gibt kein offizielles Release des Codes auf Sourceforge(Zip-Datei), der Code konnte nur vom CVS heruntergeladen werden.
- Es existiert keine Dokumentation und keine Homepage.
- Der Code kompilierte unter Linux problemlos, aber es war nicht nachvollziehbar wie LeJos-Programme (oder auch andere) ausgeführt werden können, rsp. Wie dann Motor oder Sensor-Aktivität hätte ausgelesen werden können.
- Wie BrickEmu braucht auch RCXEmu ein ROM-Image.
- Das im Source-Code enthaltene Java-Gui ist sehr Rudimentär und kann nicht benutzt werden(mangels Dokumentation).

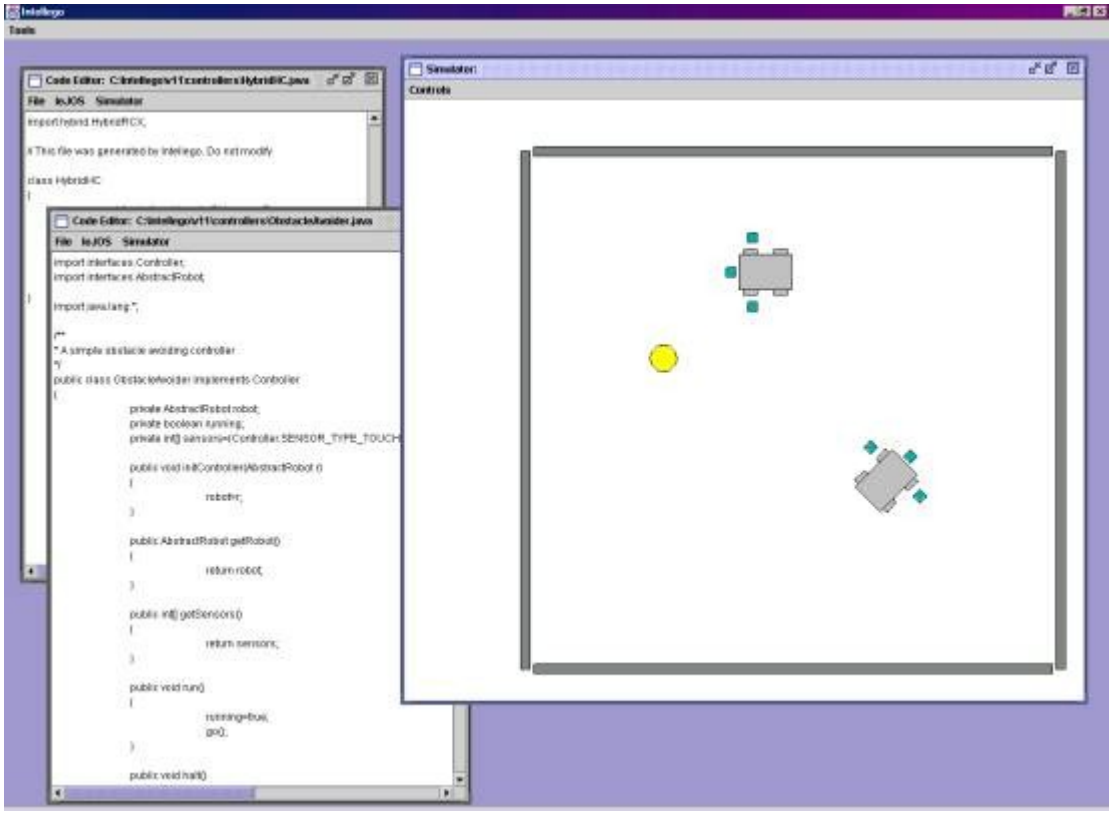
- Häntg ab vom GNU Compiler H8300, ob der Code auch auf Windows zum laufen gebracht werden kann, konnte nicht abgeschätzt werden.

3.3.3 Fazit

Die Einarbeitung in den Source-Code würde sehr viel Zeit in anspruch nehmen, ausserdem ist nicht geklärt welche Funktionen dieser Emulator unterstützt und welche man selber implementieren müsste.

3.4 IntelLego

IntelLego (`http://homepages.inf.ed.ac.uk/s0237680/intellego/intellego.html`) ist ein Simulator für LegoProgramme die auf dem LeJos-API aufbauen. IntelLego ist komplett in Java geschrieben. Das Bild stammt von der Intellego Homepage.



3.4.1 Vorteile

- Gut Dokumentiert
- Läuft unter Windows und Linux (da Java)
- Es existiert ein GUI zum testen der Implementation/Features
- Es existiert bereits ein Simulator(IntelLego selbst) mit dieser Technik.

3.4.2 Nachteile

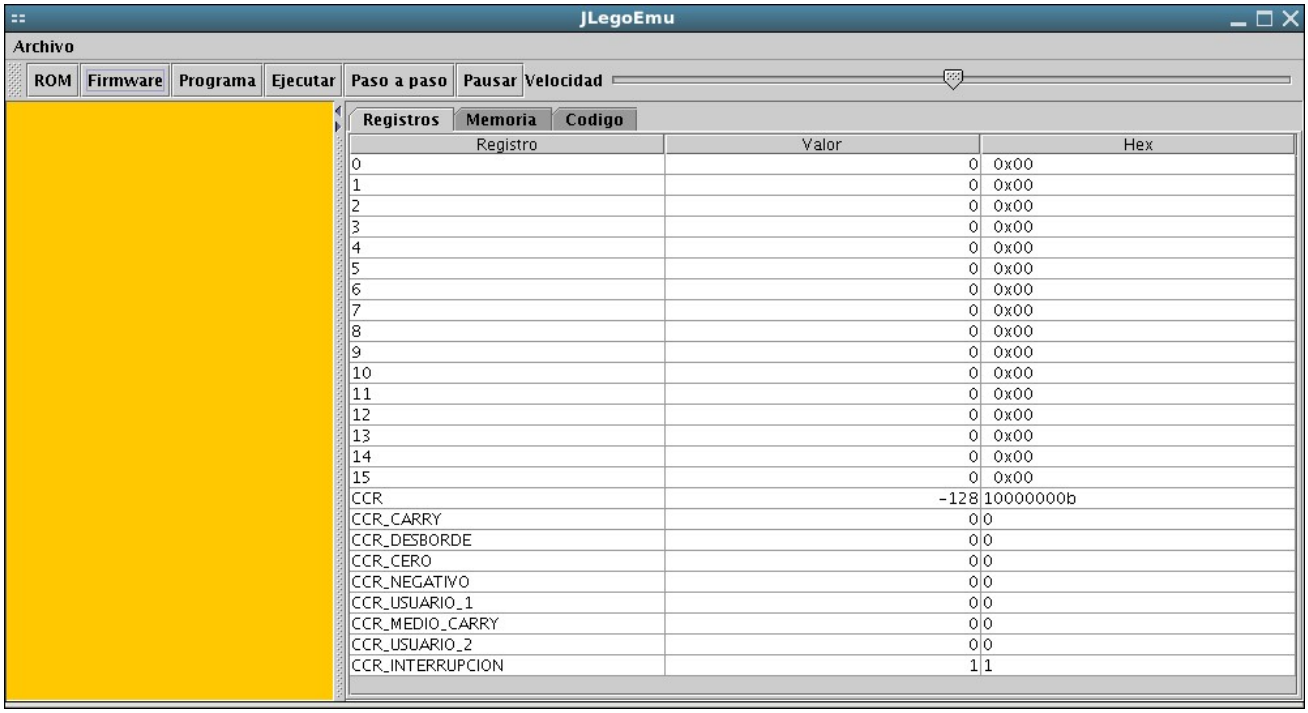
- In Java geschrieben, schwierige Integration in C++

3.4.3 Fazit

Den Source-Code von IntellLego in unser C++ Projekt zu integrieren macht nicht viel Sinn. Womöglich könnte aber die Art und Weise wie IntellLego mit LeJos-Programmen umgeht, als Vorlage/Beispiel für eine eigene Implementation dienen.

3.5 jLegoEmu

jLegoEmu (`http://sourceforge.net/projects/jlegoemu`) ist ein Simulator für LegoProgramme. Es ist nicht bekannt ob jLegoEmu LeJos-Programmen ausführen kann.



Registro	Valor	Hex
0	0	0x00
1	0	0x00
2	0	0x00
3	0	0x00
4	0	0x00
5	0	0x00
6	0	0x00
7	0	0x00
8	0	0x00
9	0	0x00
10	0	0x00
11	0	0x00
12	0	0x00
13	0	0x00
14	0	0x00
15	0	0x00
CCR	-128	10000000b
CCR_CARRY	0	0
CCR_DESBORDE	0	0
CCR_CERO	0	0
CCR_NEGATIVO	0	0
CCR_USUARIO_1	0	0
CCR_MEDIO_CARRY	0	0
CCR_USUARIO_2	0	0
CCR_INTERRUPCION	1	1

3.5.1 Vorteile

- Es existiert bereits ein Simulator(jLegoEmu selbst) mit dieser Technik.
- Es gibt ein GUI zum testen der Implementation (Register und Speicherinhalt Output)
- Läuft unter Windows und Linux (da Java)

3.5.2 Nachteile

- In Java geschrieben, schwierige Integration in C++
- Schlecht Dokumentiert, keine Homepage
- Code nur im CVS verfügbar
- Braucht ein ROM-Image

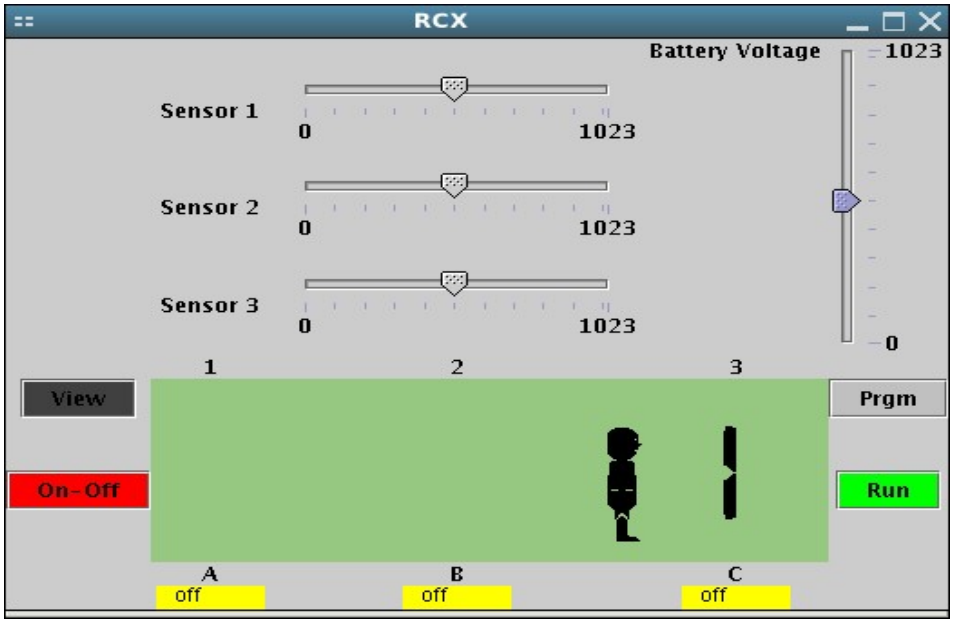
- Kommentare im Code sind Spanisch

3.5.3 Fazit

Da kein Dokumentation zu finden ist, musste der Source-Code analysiert werden, um den Funktionsumfang von jLegoEmu abzuklären. Test-Programme konnten zwar mit jLegoEmu ausgeführt werden, doch war die Ausgabe nicht sehr aufschlussreich, d.h. Es konnte kaum entschieden werden, ob ein Fehler aufgetreten war, oder ob die Ausgabe einer normalen Motor/Sensor aktivität entsprach.
Dieses Projekt ist unbrauchbar für die Integration in Jorge, möglicherweise könnte aber die Art und Weise wie jLegoEmu mit LeJos-Programmen umgeht, als Vorlage/Beispiel für eine eigene Implementation dienen.

3.6 RCXSimulator

RcxSimulator (`http://www4.informatik.uni-erlangen.de/~felser/RCXSimulator/`) ist ein komplett (?) in Java geschriebener Emulator.



3.6.1 Vorteile

- Gute User- Dokumentiert
- Aktuell, letztes Release stammt vom 9.5 2005
- Läuft auf allen Platformen (da Java)
- Implementiert vollen Funktionsumfang des RCX

3.6.2 Nachteile

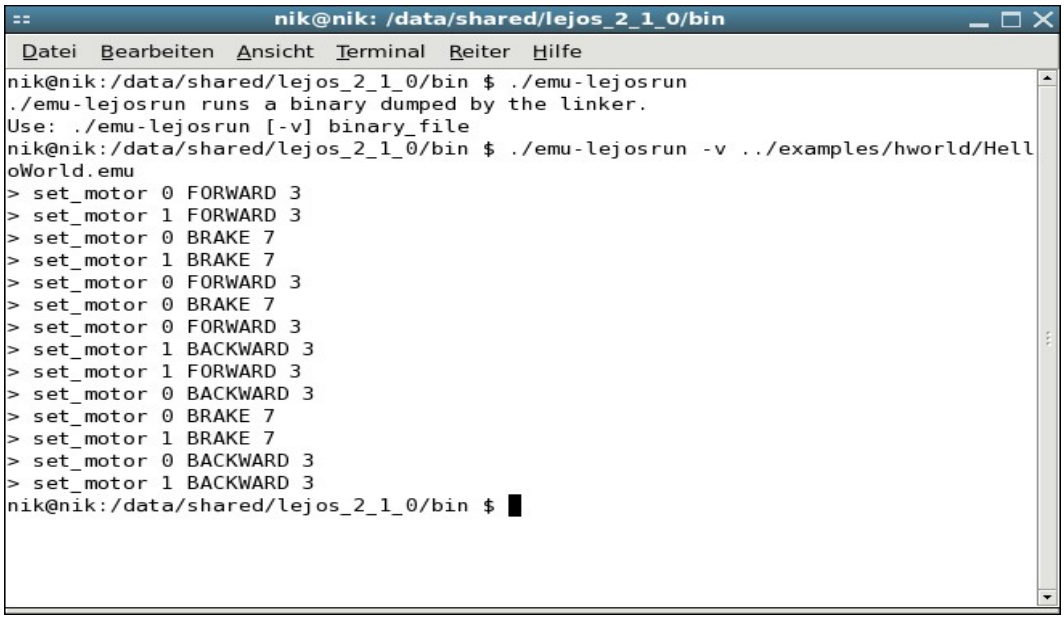
- SourceCode nicht erhältlich
- Braucht ein ROM-Image
- In Java geschrieben

3.6.3 Fazit

Leider ist der RCXSimulator in Java geschrieben, so dass sich Probleme bei der Integration in unser C++ Projekt ergeben würden. Zudem ist der Source-Code auf der Homepage nicht erhältlich, da dieses Projekt aber an einer Universität entwickelt wurde, könnte man vielleicht mit dem Entwickler eine Vereinbarung für die Benutzung des Codes zu treffen.

3.7 Emu-LeJosrun

LeJos (http://lejos.sourceforge.net/) bietet neben den Compilier- und Download-Tools auch einen Emulator zum Ausführen der LeJos-Binary's. Emulejos-run ist kein Emulator, es können daher nur Lejos-Programme ausgeführt werden. Genauer gesagt ist emulejos-run eine Java Virtual Machine mit beschränktem Funktionsumfang (stammt von Tiny VM ab), die für LeJos erweitert wurde.



3.7.1 Vorteile

- Implementiert vollen Funktionsumfang des LeJos-API (und wird auch eingesetzt)
- In C geschrieben
- Übersichtlicher, gut dokumentierter Code
- Compilierbar und Ausführbar unter Windows, Linux und MacOsX
- Einfach in die Arbeit mit LeJos integrierbar. D.h. Es braucht keine Zusätzlichen Tools wie z.B. Compiler

3.7.2 Nachteile

- Für Windows muss emu-lejosrun unter Cygwin kompiliert werden
- Ist kein RCX-Emulator: Kann nur LeJos-Programme ausführen

3.7.3 Fazit

Emulejos-run hat mit allen getesteten Programen einwandfrei funktioniert.

3.8 Eigene Implementation: Java Natural Interface (JNI)

Im Team wurde diskutiert anstelle eines bestehenden Emulators eine eigene Lösung basierend auf dem Java Natural Interface (JNI) zu implementieren. JNI bietet (unter anderem) die Möglichket Java Funktionen aus C/C++ anzusprechen. Mit Sun's Java Developement Kit kann man C/C++ Header Dateien aus Java-Klassen generieren. Werden diese Header Dateien von einem C/C++ Program eingebunden, können zur Laufzeit die in Java geschriebenen Funktionen aufgerufen werden (der Java-Code wird dabei von der Java-VM ausgeführt, JNI ist dabei die Schnittstelle zum nativen Program).
Um den Virtuellen Roboter zu Steuern brauchen wir Informationen über den Status der Motoren und die Möglichkeit Sensor-Ereignisse auszulösen. Man könnte also eine JNI-Schnittstelle für diese Aktionen definieren und die entsprechenden Java-Klassen des LeJos-Api's (Motor und Sensor) so verändern, dass sie mit der JNI-Schnittstelle arbeiten.
Damit der Benutzer des Simulators sein Program im Simulator starten kann, müsste er sein Program unter Verwendung des veränderten LeJos-Api compilieren. Die so entstandenen Klassen könnte dann in die Simulation importiert und ausgeführt werden.

3.8.1 Vorteile

- Eigene Implementation, einfache Fehlersuche

3.8.2 Nachteile

- Umständlich für den Benutzer der Simulation (compilieren mit dem veränderten LeJos-Api)
- Machbarkeit
- Zeitlicher Aufwand
- Unklare Kompatibilität und Portabilität (auch Zusammenarbeit mit verschiedenen Java-VM Versionen)

3.8.3 Fazit

Ob der beschriebene Weg wirklich machbar ist, konnten wir nicht eindeutig klären. Es existieren verschiedenste Beispiele für den Einsatz von JNI, aber uns ist kein grösseres Projekt bekannt, das JNI wirklich einsetzt. Da der Teufel bekanntlich im Detail liegt, können wir nicht abschätzen wiviel Zeit diese Lösung in anspruch nehmen könnte.

4 Auswertung

Das Resultat der Evaluation ist ernüchternd, von den 7 getesteten Projekten, kommen nur zwei in die engere Auswahl: BrickEmu und Emulejos-run.
IntelLego, jLegoEmu und RCXSimulator können nicht eingesetzt werden, da sie in Java geschrieben sind. Mit EmuLegOs können keine LeJos Programme ausgeführt werden und RcxEmu scheitern am Crossplatform Anspruch.
Die “Eigene Implementation“ beschreibt nur eine Lösungs-Idee, da wir kein Prototyp dazu erstellt haben, wissen wir nicht ob dieser Ansatz überhaubt funktioniert. Da wir aber zwei andere funktionierende Kandidaten haben, wollen wir nichts riskieren (bezüglich Zeitaufwand und Funktionalität) und verwerfen die eigene Implementation.

Somit verbleiben BrickEmu und Emulejos-run. BrickEmu ist der interessantere Ansatz, da es sich um einen vollständigen Emulator handelt, der verschiedenste Lego-Betriebssystem laden und ausführen kann. Emulejos-run ist eine abgespeckte Virtuelle Maschine die nur LeJos-Programme ausführen kann. Gegen BrickEmu spricht, dass dort 3 verschiedene Programmiersprachen(C, Perl und Tcl/TK fürs GUI) eingesetzt werden, was unter Windows Probleme verursachen könnte. Ausserdem benötigt man zum Ausführen von BrickEmu ein ROM-Image, womit wir

sehr grosse Schwierigkeiten hatten, und es bezüglich der Lizenz rechtliche Probleme gibt. Für Emulejos-run spricht, dass er für LeJos Programme optimiert wurde und auch in der LeJos Distribution enthalten ist.
Im Rahmen der Semesterarbeit ist emulejos-run die bessere Lösung, bezüglich Aufwand und Risiko. Da beide unter Windows Cygwin voraussetzen, können wir mit emulejos-run erste Erfahrungen mit dem Zusammenspiel von Cygwin und Visual Studio sammeln, ohne das wir uns mit Perl oder Tcl/Tk herumschlagen müssen. Mit diesen Erfahrungen können wir bei der Planung der Semesterarbeit nochmals über diese Entscheidung diskutieren.

4.1 Weiteres Vorgehen

Unter Linux und verwandten Systemen werden wir keine Probleme beim Kompilieren haben. Anders sieht es mit Windows aus. Es muss deshalb zuerst abgeklärt werden, ob es möglich ist, emulejos-run mit Visual Studio zu kompilieren. Falls dies möglich ist, können wir uns sofort um den Datenaustausch (Sensor- und Motordaten) zwischen emulejos-run und der virtuellen Welt (Ogre) kümmern. Falls wir emulejos-run nicht mit Visual Studio kompilieren können, müssen wir entweder emulejos-run portieren oder einen Weg finden, emulejos-run unter Cygwin zu kompilieren, und dann irgendwie in Visual Studio einzubinden. Falls wir keinen Weg finden JORGE (Ogre Teil) und emulejos-run in ein Program zu verschmelzen (linken), verbleibt die Möglichkeit den Datenaustausch über Pipes oder über eine Netzwerk-Schnittstelle zu realisieren.