universität
wien

BACHELORARBEIT

Titel der Bachelorarbeit

Einsatz von Midi und NetLogo im Informatikunterricht

bzw. die Erweiterung von NetLogo um agentenbasierte Midi-Befehle

Verfasser:	Martin Dobiasch
Matrikel-Nummer:	0828302
Studienrichtung:	Informatikmanagement 522
Betreuer:	Ao. Univ.-Prof. Dr. Erich Neuwirth

Wien, am 18. 11. 2010

Diese Arbeit beschreibt die Arbeit rund um die Erweiterung von NetLogo [6] um Midi-Befehle für die einzelnen Akteure (Turtles und Patches). Sie enthlt Beispiele die im Informatik-Unterricht eingesetzt werden können um Schülerinnen und Schülern einen technischen Zugang zu Problemen aufzuzeigen

Inhaltsverzeichnis

1 Einleitung **3**
- 1.1 Anreger & Betreuer . . . 3
- 1.2 Ziel der Arbeit . . . 3
- 1.3 Verwendete Tools . . . 3
- 1.4 Ausgangslage für NetLogo . . . 3
- 1.5 Installation der Erweiterung für NetLogo . . . 4
- 1.6 Gliederung der Arbeit . . . 4
- 1.7 Test der Erweiterung . . . 5
- 1.8 Wünschenswertes für die NetLogo-Erweiterung . . . 5

2 Erweiterung **5**
- 2.1 Grundsätzliches . . . 5
- 2.2 Unterlagen . . . 6
- 2.3 Struktur . . . 6
- 2.4 Neue Befehle . . . 8

3 Der Dirigent in Midi-NetLogo **8**
- 3.1 Konzept . . . 8
- 3.2 Technische Realisierung . . . 8

4 Befehle der NetLogo-Extension **9**
- 4.1 Midi-Befehle . . . 10
- 4.2 Befehle für Turtles/Agenten . . . 10
- 4.3 Befehle für den Condcutor . . . 10

5 NetLogo Projekte **13**
- 5.1 Rotierende Trommler . . . 13
- 5.2 Rettungsauto . . . 18

Literaturverzeichnis **22**

1 Einleitung

1.1 Anreger & Betreuer

Auftraggeber und Betreuer dieser Arbeit war AO. Univ. Prof. Dr. Erich Neuwirth. Ich möchte ihm hier an dieser Stelle bei ihm für die ausgezeichnete Betreuung und Zusammenarbeit bedanken.

1.2 Ziel der Arbeit

Die vorliegende Arbeit versteht sich als Arbeitsunterlage für Lehrpersonen. Nicht jedoch aber als Handout für die Lernenden. Lehrende sollen durch dieses Dokument einen Einblick in die verwendeten Tools bekommen und aufbereitete Beispiele vorfinden, welche sie für ihren Unterricht verwenden können.

Der Einsatz von Musik im Informatikunterricht mag auf den ersten Blick etwas seltsam, wenn nicht sogar verstörend wirken. Er bringt jedoch einen großen Vorteil mit sich: Fehler im Ansatz oder in der Ausarbeitung sind sofort hörbar. Ein falsch transponierter Dreiklang kann auch von einem musikalisch nicht begabten Lernenden erkannt werden. Zu dem kann die Musik als mächtiges didaktisches Werkzeug gesehen werden, wenn es um die Vermittlung von parallelen Verarbeitungsmodellen geht. Die Metapher eines Orchesters kann sehr gut heran gezogen werden, da in ein Orchester aus mehreren MusikerInnen besteht. Diese agieren von einander im wesentlichen unabhängig, müssen jedoch bis zu einem gewissen Grad von etwas bzw. jemand gesteuert werden: Dem Dirigenten, auf englisch dem Condcutor. Auch die Metapher des Dirigenten/Conductor wird in meiner Arbeit stark zum Einsatz kommen.

Wichtig für einen Einsatz im Unterricht, ist der Hinweis, dass die Lernenden Kopfhörer mitbringen sollten. Spielen mehrere ohne Kopfhörer ist ein Unterricht kaum mehr möglich und es herrscht Chaos.

Beispiele dafür, was mit dem Toolkit möglich ist, finden sich im letzten Abschnitt. Es ist beispielsweise möglich, 2 Rettungsautos aus verschiedenen Richtungen fahren zu lassen und die Signaltöne der Autos akustisch zu lokalisieren.

1.3 Verwendete Tools

Im wesentlichen wird die NetLogo-Extension 'midi' verwendet. Die Midi-Extension für NetLogo enthält einige der Fähigkeiten die das MidiCSD-Toolkit[3] für OpenOffice bzw. Excel ebenfalls beherrscht. Der nächste Abschnitt beschreibt die Notwendigkeit der NetLogo-Erweiterung bzw. zeigt die Anleitung in 'Installation der Erweiterung für NetLogo' wie simpel sie zu installieren ist und somit einfach im Unterricht eingesetzt werden kann. Es sei an dieser Stelle explizit auf MidiCSD hingewiesen. Gerade für Experimente mit Musik eignet es sich hervorragend um im Informatik-Unterricht angewendet zu werden.

1.4 Ausgangslage für NetLogo

Bisher störte die Tatsache, dass es nicht möglich war in NetLogo Midi-Kanäle direkt anzusprechen. In der Version 4.1.1 (aktuell zum Erstellungszeitpunkt des Projektes Juli, 2010) konnten lediglich die Befehle sound:play−drum drum velocity und sound:play−note instrument keynumber velocity

duration verwendet werden um Modelle mit Tönen zu versehen. Als Konsequenz daraus konnte man den einzelnen Akteuren keine musikalischen Aktionen zuordnen. So ist es zum Beispiel nicht möglich für das Beispiel "Rettungsauto" (siehe 5.2) den Dopplereffekt zu erzeugen, geschweige denn überhaupt den Ton des Autos lauter werden zu lassen im Falle des sich annähernden Autos und wieder leiser werden zu lassen, im Fall des sich entfernenden Autos.

NetLogo hat eine (fast) unbeschränkte Anzahl an Akteuren, den Turtles. Bisher war es aber nicht möglich Musik, oder einzelne Töne mit diesen zu verbinden. Sieht man sich die Beispielsammlung der NetLogo-Distribution an, findet man im Bereich Musik nur wenige Beispiele. Auch kann in diesen die volle Macht von NetLogo nicht ausgenutzt werden. So wird z.B. im Beatbox Beispiel die Musik von einer zentralen Prozedur aus angewandt. Alle 'Trommler' spielen jedoch auf dem selben Midi-Kanal. Das wäre also so als würde doch nur ein Trommler spielen und mehrere Percussioninstrumente zur gleichen Zeit bedienen.

Im vorigen Absatz wird erwähnt, dass nur auf einen Midi-Kanal zugegriffen wird. Das ist wie schon eingangs erwähnt ein großes Manko. Es ist also nicht möglich Effekte nur für einzelne Instrumente zu steuern. Beispiel: Der Trompeter soll leise spielen weil gerade die Flöte ein Solo hat (Funktioniert im echten Leben leider auch nicht immer).

1.5 Installation der Erweiterung für NetLogo

Es gibt zwei Varianten um die Erweiterung zu installieren

- die Datei "midi.jar" in einen Unterordner "midi" neben das Model legen, welches die Erweiterung verwenden möchte.
- die Datei "midi.jar" in einen Ordner "midi" in das Verzeichnis "extensions" der NetLogo-Installation legen.

Ebenso erfolgt die Installation für als Applet im Internet hinterlegte Modelle. Auch hier muss die "midi.jar" Datei ein einem "midi" Unterordner liegen.

1.6 Gliederung der Arbeit

Ich werde zuerst kurz beschreiben wie ich NetLogo um zusätzliche Befehle erweitert habe. Dann werde ich die Konzepte hinter der Erweiterung erläutern und Beispiele bringen wie die Befehle eingesetzt werden können. Diese Beispiele sind schon so vorbereitet um in einem etwaigen Unterricht mit Schülern eingesetzt werden zu können.

Es werden einige Details hinter der Arbeit sehr ausführlich erläutert. Dies wird gemacht um den Lesern bzw. Anwendern dieser Arbeit einen tieferen Einblick in die Interna der Erweiterung zu geben. Ein tieferes Verständnis der Arbeit kann einen verbesserten Einsatz im Unterricht ermöglichen, da so jeder Lehrende individuell Stärken und Schwächen für den persönlichen Einsatz im Unterricht ausmachen kann. Weiters tauchen in der Praxis der Informatik immer wieder Probleme auf die nur durch ein detailliertes Wissen der verwendeten Mittel gelöst werden können. Auch rüstet ein gutes Hintergrundwissen über die Erweiterung die Lehrenden für etwaige Fragen von interessierten Lernenden, welche zB die im Unterricht erläuterten Beispiele noch erweitern wollen oder bei der Erweiterung auf Probleme gestoßen sind.

1.7 Test der Erweiterung

Da ein Teil meiner Arbeit war NetLogo[6] Midi tauglich zu machen, musste dies auch getestet werden. Die Arbeit wurde unter anderem mittels der weiter hinten in der Arbeit beschriebenen Beispiele getestet. Es ist also gewährleistet, dass diese Beispiele in der erläuterten Form funktionieren und keine Seiteneffekte hervorrufen. Getestet wurde sowohl auf Windows PCs so wie auf Apple Computern. Verwendet wurde die zum Zeitpunkt der Arbeit aktuelle NetLogo Version 4.1.1.

Es wurden bei den Tests keine Unterschiede zwischen Windows und Mac OS entdeckt.

1.8 Wünschenswertes für die NetLogo-Erweiterung

Die Erweiterung wurde mit der Version 4.1.1 von NetLogo entwickelt. In geraumer Zeit sollte die Version 4.1.2 fertig gestellt werden. Mit dieser Version kann ein Problem meiner Erweiterung eventuell gelöst werden und das hinzufügen von Befehlen zu Notenblättern direkt als Befehl möglich sein. Es seien hier einige Aspekte aufgeführt die von mir nicht in die Erweiterung aufgenommen wurden, manch Lehrendem aber als fehlende Aspekte auffallen. Diese Liste erhebt aber keinen Anspruch auf Vollständigkeit. Feedback von Lehrenden ist erwünscht!

Die aktuelle Implementierung könnte noch in folgenden Punkten erweitert werden:

- *Midi-Rendern* Es können zur zeit nur direkt Befehle ausgegeben werden. Eine mögliche Erweiterung ist, dass anstatt die Befehle auszugeben, die akustische Ausgabe des Modells in ein Midi-File zu schreiben.

- *Midi-Compiler* Wie später hinter in der Arbeit angemerkt wird, hat die Implementierung Befehlsabarbeitung noch Potential zur Optimierung. Es könnte angedacht werden, einzelne Blätter als Midi-Blätter zu deklarieren. Zu diesen können dann nur Midi-Befehle hinzugefügt werden, welche dann direkt als javainterne Midi-Commands kompiliert werden.

- *Protokollierung* Das Projekt könnte zusätzlich um die Funktionalität von Protokollausgaben der Befehle erweitert werden können. Anstatt von Midi-Noten werden könnten dann normale Notennamen ausgegeben werden können.

2 Erweiterung

2.1 Grundsätzliches

Eine Extentsion für NetLogo ist nichts anderes als ein Jar-Archiv. Die Erweiterung muss in einem Unterverzeichnis des Models, welches die Erweiterung verwenden will, oder in einem Unterverzeichnis des NetLogo-Programmes liegen. Jede Extension benötigt einen ClassManager. Dieser teilt NetLogo mit welche Befehle die Erweiterung mit sich bringen wird. Jeder dieser Befehle ist von org.nlogo.api.Primitive abgeleitet. Die Erweiterung muss zusätzlich ein Manifest mit den folgenden Befehlen enthalten (am Beispiel meiner Erweiterung):

```
Manifest-Version: 1.0
Extension-Name: midi
Class-Manager: at.univie.csd.MidiManager
NetLogo-Extension-API-Version: 4.1
```

2.2 Unterlagen

Für die Erstellung der Erweiterung wurden im wesentlich folgende Quellen zu Theorie zu Midi zur Rate gezogen: [1], [2] und [4] verwendet.

2.3 Struktur

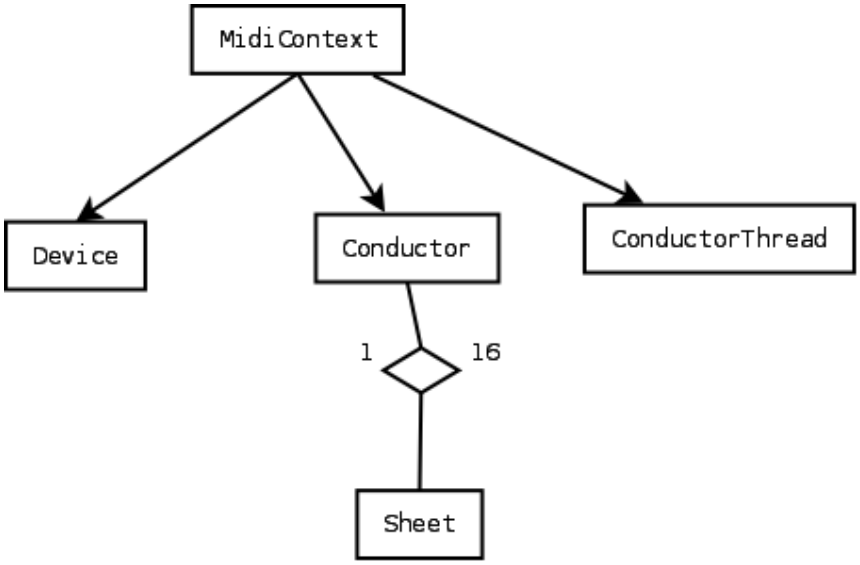

Abbildung 1: Struktur

Die Erweiterung besitzt eine relativ einfache Struktur. Beim laden der Erweiterung wird eine MidiContext Instanz erzeugt. Um eine Synchronisation der Zugriffe auf die Objekte zu ermöglichen habe ich in Anlehnung an [5] einfache Semaphore implementiert. Anstatt von Integerzählern habe ich einfach boolesche Variablen verwendet. Bei den Tests hat dies zu keinen Problemen geführt. Zu finden sind die Semaphore in der Klasse MidiContext in der Datei "MidiContext.java".

Die Implementierung als Semaphor mit Zähler würde wie folgt aussehen:

```
...
private int m_free = 1;
...
public synchronized MidiDevice getDevice()
{
  while( m_free <= 0 )
  {
    try
    {
      wait();
    }
    catch(InterruptedException e)
    {
    }
  }
  m_free--;
}
...
public synchronized void releaseDevice()
{
```

```
    m_free++;
    notify();
  }
```

Analog die Impelentierung für den Conductor-Semaphor.

2.3.1 MidiContext

Der MidiContext soll sich um die Synchronisation der Threads kümmern. Da nur ein Device zum abspielen der MidiBefehle geöffnet wird muss der Zugriff synchronisiert werden. Um einen Befehl schreiben zu können muss zuerst der Zugriff reserviert werden. Dies geschieht über den Befehl MidiManager.getMidiContext().getDevice();. Natürlich muss auch noch überprüft werden ob das Device auch offen ist. Um das Device wieder für andere Threads wieder frei zu geben muss der Befehl MidiManager.getMidiContext().releaseDevice(); verwendet werden.

2.3.2 Conductor

Weiters beinhaltet der MidiContext eine Instanz des Conductors. Diese Klasse soll helfen einen virtuellen Dirigenten zu erstellen. Dazu hat die Klasse bis zu 16 Notenblätter. Die Anzahl ist Aufgrund der Midi-Spezifikation von 16 Kanälen festgesetzt worden. Auch hier muss wieder um die Synchronität zugewährleisten mit den Befehlen MidiManager.getMidiContext().getConductor(); und MidiManager.getMidiContext().releaseConductor(); gearbeitet werden.

2.3.3 Sheet

Die Klasse Sheet hilft bei der Verwaltung von Kommandos, die timecode gesteuert ausgeführt werden sollen. Sheet ist das Pendant zu Streams in MidiCSD [3] Jedes Notenblatt verfügt über eine Liste von Kommandos. Zu Begin der Arbeit war die Anzahl dieser Commandos auf die des MidiCSD-Toolkits beschränkt und jeder dieser Befehle wurde intern zu den entsprechenden Midi-Befehlen kompiliert. Dies wurde jedoch in Rücksprache und auf Wunsch des Betreuers so geändert, dass nun jedes beliebige Kommando hinzugefügt werden kann. Während des Programmablaufes wird also das Kommando nur gespeichert und dann NetLogo zur Ausführung übergeben. Dies führt zu kleinen Laufzeiteinbusen, erhöht jedoch aber die Flexibilität und erweitert die Möglichkeiten für den Einsatz der Erweiterung. So ist es zum Beispiel möglich den virtuellen Musikern nicht nur Noten sondern auch Bewegungen beizubringen. Der "alte" Code wurde als Kommentar im Source-Code hinterlassen.

2.3.4 ConductorThread

Damit der Dirigent im Hintergrund die Notenblätter abarbeiten kann, wurde der CondutorThread erstellt. Dieser, wenn gestartet, arbeitet Stück für Stück die Notenblätter ab. Dies macht er, in dem er sich in jedem Schritt einen Zugriff auf den Conductor reserviert um sich die anstehenden Kommandos zu besorgen. Diese werden dann sequentiell ausgeführt. Danach legt sich der ConductorThread für 100ms schlafen um den anderen Threads nicht im Weg zu stehen.

2.4 Neue Befehle

Um einfach neue Midi-Befehle implementieren zu können habe ich eine Klasse MidiCommand geschrieben. Diese besitzt im wesentlichen zwei Methoden: preAction und postAction. Jeder neue Midi-Befehl, wenn von dieser Klasse abgeleitet, kann sich preAction Zugriff auf das Midi-Device holen und diesen dann wieder mit postAction freigeben. So kann sicher gestellt werden, dass durch neue Befehle kein Wirrwarr von offenen Devices entsteht. Weiters erhöht diese Vorgangsweise die Modularität. Soll zum Beispiel etwas an der Art wie das Device geöffnet wird geändert werden, muss dies nur an einer Stelle angepasst werden.

3 Der Dirigent in Midi-NetLogo

Ein weiterer Punkt den ich in die Aufgabenstellung inkludiert habe, war die Möglichkeit Sequenzen zu definieren, welche dann unabhängig abspielbar sein sollen. Dieser Punkt wurde dann erweitert, dass es möglich sein soll, alle in NetLogo verfügbaren Befehle in so eine Sequenz zu packen.

Das kann nun dafür verwendet werden um im Informatik-Unterricht Threads zu vermitteln. Ein Thread ist damit nichts anderes als ein Sheet.

3.1 Konzept

Wie der Name schon andeutet, ist dieser Aspekt mit der Metapher eines Orchesters gelöst worden. Ein Dirigent hat die Kontrolle über mehrere Musiker die Aktionen ausführen sollen. Er bestimmt wann das Orchester zu spielen beginnt und wann es wieder aufhört. Weiters kann er natürlich auch wieder von Vorne beginnen lassen.

Jedes Programm braucht auch Mechanismen zur Steuerung und Synchronisation seiner Threads. Der Dirigent übernimmt diese Aufgabe und ermöglicht so eine vereinfachte Form der Vermittlung von Thread-Synchronisation.

3.2 Technische Realisierung

Wie schon im Konzept erwähnt wird NetLogo durch die Midi-Extension eine Zentrale Instanz 'Conductor' beigebracht. Diese verfügt über die folgenden Fähigkeiten:

- Alle Notenblätter löschen
- Zu einem Notenblatt etwas hinzufügen
- Zu einem Notenblatt einen mit Timecode versehenen Befehl hinzufügen
- Alles auf Anfang setzen
- Dirigieren
- Endlos spielen lassen, normal spielen lassen

3.2.1 Sheets - Kanäle

Der Dirigent hält eine fixe Anzahl von 16 Sheets. Die Zahl ist so festgelegt, da Midi 16 Kanäle besitzt. Die Midi-Kanäle besitzen die Nummern 1 - 16, die Sheets jedoch die Indizes/Nummern 0 - 15. Über eine Variable für Agents kann eine Zuordnung von Agenten zu einem Kanal gemacht werden.

```
turtles-own [channel]
...
to init
  den Turtles channels zu ordnen
end

...

to some.procedure
  ask turtles[
    midi:instrument channel 60
  ]
end
```

Eine sehr einfach Variante den Turtles Kanäle zuzuornen ist:

```
ask turtles[
  set channel who + 1
]
```

3.2.2 Events

Die zentrale Aufgabe der Sheets ist es Befehle in chronologischer Reihenfolge auszuführen. Es gibt zwei Befehle um einem Notenblatt Befehle hinzu zufügen:

- conductor.add.to.sheet
- conductor.add.to.sheet.list

Beide Befehle machen im wesentlichen das Gleiche, jedoch werden dem Zweiten eine Liste von mit einem Timecode versehenen Befehlen übergeben, ersterer fügt einen einzelnen Befehl versehen mit einem Timecode zu einem Notenblatt hinzu. Eine genauere Beschreibung ist im Kapitel über die Befehle zu finden.

4 Befehle der NetLogo-Extension

Alle Befehle beginnen mit einem vorangestellten midi:. Ich möchte hier kurz die Befehle beschreiben um der Lehrperson einen tieferen Einblick zu gewähren und damit diese den Lernenden bei etwaigen Problemen kompetent helfen können.

Die Dokumentation der Befehle erfolgt immer in der Form Befehl Parameter was dann in einem NetLogo-Programm als midi:Befehl Parameter geschrieben wird. Die Befehlsnamen sind nicht(!) casesensitive.

4.1 Midi-Befehle

4.1.1 Standard Befehle

Auf eine Dokumentation der Standard-Midi-Befehle möchte ich hier verzichten. Es sei auf das Studium einschlägiger Fachliteratur verwiesen. Folgende Befehle sind in der Erweiterung verfügbar: noteon, noteoff, instrument (Für eine Liste an Instrumenten siehe [1]), pitch.bend, controller, key.pressure, channel.pressure, volume, expression, modulation, pan, sustain, reverb, chorus, portamento.time, portamento, portamento.from, rpn, nrpn, reset.controllers, all.notes.off, pitch.sens channel, mastertune.coarse, mastertune.fine, panic

4.1.2 Der Note Befehle

note channel note volume duration

*A*chtung: Dieser Befehl ist ein Blocking-Call. Er spielt eine Noten in der Dauer von duration (in ms angegeben ab). Erst dann wird das Programm fortgesetzt. Er setzt zuerst die Note und legt den Thread dann schlafen, nach Abwarten des Notenwertes löscht er die Note vom entsprechenden Midi-Kanal. Dieses Verhalten kann zu Problemen führen wenn sie in Programmen in Kombination mit den Conductor-Utilities verwendet werden.

4.2 Befehle für Turtles/Agenten

updatepostion channel

Dieser Befehl setzt eine eine Position im akustischen Raum für die aktuelle Turtle, kann also nur in einem Turtle-Context angewandt werden. Die Position wird für die Turtle die den Befehl aufruft errechnet und dann als virtuelle Position auf den übergebenen Channel gelegt (siehe auch 3.2.1. Verändert werden die Parameter Pan und Expression. Ausgangspunkt für die Berechnung ist der Koordinaten-Ursprung des NetLogo-Models. Als maximaler Wert für die Skalierung wird auch jeweils der maximale Wert des Koordinatensystems verwendet wird. Das heißt wenn das Model zB. auf der positiven X-Achse mehr Werte zulässt als auf der negativen. Ist die Abstufung auf der positiven Seite feiner.

4.3 Befehle für den Condcutor

4.3.1 clear.sheets

Syntax: conductor.clear.sheets

Löscht alle dem Conductor bekannten Notenblätter.

4.3.2 add.to.sheet

Syntax: conductor.add.to.sheet sheet time.distance command

Dieser Befehl fügt dem angegebenen Notenblatt am Ende ein Kommando hinzu. Das Kommando wird nach Ablauf der durch time.distance angegebenen Zeit (in ms) ausgeführt. Zusätzlich ist das der Zeitpunkt von welchem aus der Zeitpunkt für ein mögliches nachfolgendes Kommando berechnet wird. Beispiel:

```
midi:conductor.clear.sheets
midi:conductor.add.to.sheet 1 10 "midi:noteon 2 60 1"
midi:conductor.add.to.sheet 1 2000 "midi:noteoff 2 60"
midi:conductor.start
```

Nach Ablauf von zehn Milisekunden wird vom Notenblatt Eins auf dem Midi-Kanal Zwei die Note mit dem Wert 60 gespielt. Nach zwei Sekunden, also nach insgesamt 2010 Milisekunden verstummt die Note wieder.

4.3.3 add.to.sheet.list

Syntax: conductor.add.to.sheet. list sheet command.list

Dieses Kommando fügt dem angegebenen Notenblatt am Ende die Kommandos aus der Liste ein. Die Kommando-Liste hat eine einfache Syntax: [time.code command] Die Time-Codes funktionieren analog zu denen des add.to.sheet Befehls. Das Kommando muss aber als String angegeben werden! Das kommt daher, dass NetLogo zum Entwicklungszeitpunkt (Juli-Oktober 2010) nicht ausreichend mit dem Syntax-Typ "Command" umgehen konnte. Die Befehle sollten also getestet werden bevor sie auf die Notenblätter geschrieben werden. Der Vorteil dieses Befehls ist, dass er, mit geringen Modifikationen, die Ausgabe des "to Logo" des MidiCSD-Toolkits [3] für Office Dr. Erich Neuwirth verwendet kann. Das untenstehende Beispiel demonstriert, wie der Befehl eingesetzt werden kann.

```
to trommler
  clear-all
  midi:conductor.clear.sheets

  midi:conductor.add.to.sheet.list 1 [
    [0 "midi:note 10 45 1 200"]
    [200 "midi:note 10 45 0.7 200"]
    [200 "midi:note 10 45 0.7 200"]
    [200 "midi:note 10 45 0.7 200"]
    [200 "midi:note 10 45 1 200"]
  ]
  midi:conductor.add.to.sheet.list 2 [
    [200 "midi:pan 10 -0.75"]
    [200 "midi:pan 10 -0.5"]
    [200 "midi:pan 10 -0.25"]
    [200 "midi:pan 10 0"]
  ]

  midi:conductor.add.to.sheet.list 3 [
    [5 "midi:expression 10 0.75"]
    [200 "midi:expression 10 0.69"]
```

```
    [200 "midi:expression 10 0.63"]
    [200 "midi:expression 10 0.56"]
    [200 "midi:expression 10 0.5"]
  ]

  midi:conductor.setplaymode.endless

  midi:conductor.start
end
```

4.3.4 restart

Syntax: conductor.restart

Die Position auf den Notenblättern wird wieder auf Null gesetzt. Alle Notenblätter werden also wieder von Vorne abgearbeitet.

4.3.5 conduct

Syntax: conductor.conduct

Der Dirigent wird angewiesen die aktuell anfallenden Befehle auf den Notenblättern abzuarbeiten.

4.3.6 setplaymode.endless

Syntax: conductor.setplaymode.endless

Erreicht ein "Musiker" auf seinem Notenblatt das Ende, soll er wieder von Vorne beginnen. Anders gesagt: Die Notenblätter sind nun nicht mehr linear sondern Kreise. Auf unterschiedliche Längen/Dauer der Notenblätter wird aber nicht geachtet.

4.3.7 setplaymode.normal

Syntax: conductor.setplaymode.normal

Anders als beim Endlos-Playmode wird der Dirigent angewiesen, wenn ein Notenblatt zu Ende ist, nicht wieder von Vorne zu beginnen sondern aufzuhören.

4.3.8 start

Syntax: conductor.start

Dieser Befehl weißt den Dirigenten an im Hintergrund zu dirigieren. Das Programm läuft weiter während im Hintergrund die Notenblätter abgearbeitet werden. (Siehe auch conductor.stop)

4.3.9 stop

Syntax: conductor.stop

Dieser Befehl weist den Dirigenten an seine Arbeit zu beenden. Die Abarbeitung der Notenblätter im Hintergrund wird beendet.

5 NetLogo Projekte

Die Dokumentation der zwei Projekte erfolgt in folgender Form: Zuerst wird die zugrundeliegende Idee erläutert, dann werden einige Hinweise gegeben, wie das Beispiel im Unterricht vermittelt werden kann und die Teilkomponenten einer Beispiellösung erläutert. Im Ausblick werden Anregungen erläutert wie sie z.B. eifrigen SchülerInnen vorgestellt werden können oder Zusätze die in einem etwaigen Begabten-Unterricht als eigenständige Arbeit von den SchülerInnen gelöst werden kann.

Es wird dabei davon ausgegangen, dass die Lernenden schon Erfahrung mit NetLogo und ein gewisses Grundverständnis von Musik bzw. Akustik haben.

5.1 Rotierende Trommler

5.1.1 Idee

In diesem Beispiel sollen mehrere Trommler trommelnd im Kreis marschieren. Dazu soll unabhängig von der Musik, die sie spielen die Position verändert werden. Diese Positionsänderung soll natürlich auch hörbar sein. Es sollen also die Trommler die sich am linken Rand des Feldes aufhalten auch stärker auf der linken Seite der Lautsprecher zu hören sein sollen.

5.1.2 Didaktische Aufbereitung

Das Projekt soll Schritt für Schritt mit den Schülern erarbeitet werden. Von der Grundidee eines Trommlers über einen kreisenden Agenten bzw. kreise drehende Turtle hinzu mehreren im Kreis marschierenden Trommlern.

Der Trommler : Der Trommler soll einen 4/4 Takt schlagen. Also wirklich nur simpel auf jeden Takt-Schlag einmal auf seine Trommel hauen. Zusätzlich soll er immer die Eins in jedem Takt betonen. Es kann überlegt werden, diesen Schritt mit MidiCSD[3] in Excel/OpenOffice durchzuführen. Die Lernenden können sich die Tonhöhe selbst aussuchen. Die Länge des Taktes bestimmt die Dauer der einzelnen Noten. Es muss nur ein Takt von den Lernenden modelliert werden, da der Dirigent diesen dann endlos abspielen lassen kann. Wird mit Excel gearbeitet, kann über die Phraselanguage der Takt mehrmals kopiert und hinten angefügt werden um ihn so öfter abzuspielen. Gut eignen sich drei Takte. Die Lernenden sollten wissen, dass bei einem 4/4 Takt alle Noten gleich lang sein sollen. In der Beispiellösung sind alle Schläge 200ms lang.

Im Kreis gehende Turtle In Midi beginnen die Kanäle für Trommler bei Zehn. Deshalb sollen im Modell die Turtles auch erst bei Zehn beginnen. Also bis 13 erstellen und die nicht benötigten

wieder zerstören. Das nebeneinander Aufstellen ist sehr einfach: Die Turtles sollen vom Ursprung aus nach Rechts gehen und sich dann in ihre Richtung drehen. Dazu sollen sie zuerst ins rechte - obere Eck schauen und sich dann um 45 Grad nach rechts drehen. Nach Möglichkeit sollen die Lernenden auf diesen Schritt selbst kommen oder eine Alternativlösung bringen. Um die Turtles am Kreis zu postieren müssen sie dann nur um einen von ihrer Nummer abhängigen Betrag vorwärts gehen. Auch das soll gemeinsam mit den Lernenden entwickelt werden. Jede Turtle besitzt eine aufsteigende Nummer die in der Variable who gespeichert ist. Diese kann dann auch gleich verwendet werden um, wenn die Turtles gegen-gleich rotieren sollen, die Ausrichtung vorzunehmen (left-Befehl -1^{who}). Anhand des Umfanges muss nun berechnet werden wie weit sich jede Turtle nach vorne bewegen muss. Den Lernenden sollte klar sein, dass eine Turtle, die weiter aussen steht, einen weiteren Weg zurück legen muss. Auch dass sich die Turtle kontinuierlich drehen muss sollte für die Lernenden kein Problem darstellen. Je nach Niveau der Lernenden kann an dieser Stelle dann noch ein Regler für die Geschwindigkeit eingebaut werden.

Akustische-Effekte Einige Lernende werden vermutlich gleich bemerken, dass wenn nun die Trommler eingebaut werden, das ganze nicht sehr realistisch klingt. Dafür besitzt NetLogo nun den UpdatePosition Befehl. Die Lernenden sollen darauf hingewiesen werden, dass wenn keine Variable für den Kanal verwendet wird mit $(who + 1)$ zu arbeiten ist. Das eignet sich sehr gut um die Problematik der Null und Eins basierenden Indizes zu erläutern.

5.1.3 Benutzer-Oberfläche

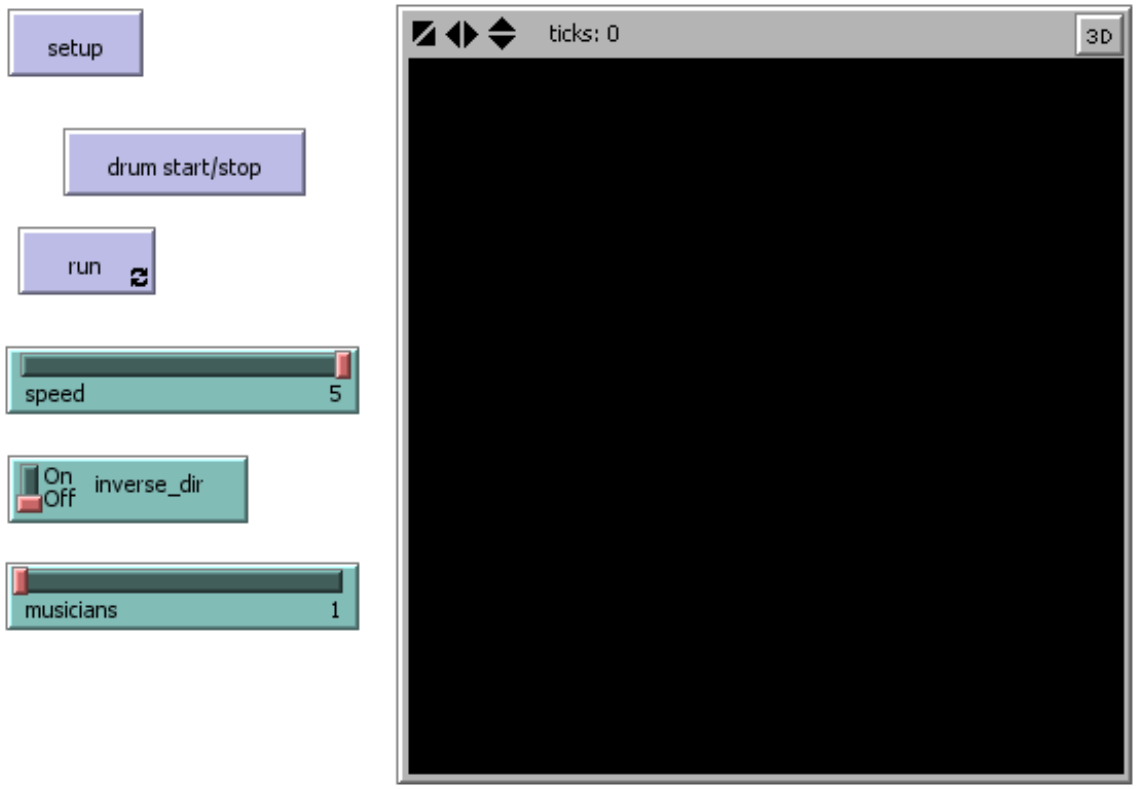

Abbildung 2: Trommler - Interface

Um das Programm zu starten ist es notwendig zuerst "Setup" und dann in beliebiger Reihenfolge

"drum start/stop" und "run" zu drücken. Der Knopf "drum start/stop" bewirkt dass der/die Trommler beginnen zu trommeln , bzw. wieder aufhören zu trommeln.

Ein Druck auf "run" lässt die Trommler im Kreis marschieren, ein weiterer Druck bewirkt, dass sie stehen bleiben.

Über den Regler mit der Beschriftung speed kann die Geschwindigkeit der Trommler reguliert werden. Das kann auch passieren während die Trommler schon im Kreis marschieren.

Der Schalter "inverse_dir" Entscheidet ob die Trommler alternierend gegengleich marschieren oder nicht. Wir er auf "On" gesetzt, marschiert der erste Trommler im Uhrzeigersinn, der Zweite gegen den Uhrzeigersinn und der Dritte wieder im Uhrzeigersinn. Wird der Schalter umgelegt während die Trommler schon marschieren verlassen diese ihre Bahnen und beginnen von ihrem aktuellen Punkt aus wieder in Kreisen zu marschieren, was an den Effekten der Akustik aber nichts ändert.

Über den Regler "musicians" wird die Anzahl der Trommler festgelegt. Nach einer Änderung der Anzahl muss erneut auf "setup" gedrückt werden.

5.1.4 Funktionsweise

Aufgrund der Möglichkeiten, die die Erweiterung mit sich bringt ist der längste Teil des Programm, die Setup-Prozedur. Sonst besteht das Programm noch aus zwei weiteren Prozeduren: der Bewegung der Trommler und das Instruieren des Conductor.

Das Marschieren der Trommler ist simpel gehalten: Es wird lediglich berechnet, wie weit jeder einzelne Trommler vorwärts gehen muss, dann wird noch für jeden Trommler abhängig davon ob sie alternierend gegengleich marschieren berechnet wie weit er sich drehen muss. Dann wird jeder Trommler verschoben und ein midi:updateposition durchgeführt.

Das Instruieren des Conductor ist sehr einfach. Die drum-Prozedur wird, von einem Schalter am Interface aus aufgerufen. Wird noch nicht getrommelt, dh. die Variable drumming ist **false**, wird über ein midi:conductor.start der Conductor angewiesen mit dem Musikstück zu beginnen und die Variable drumming auf **true** gesetzt. Im anderen Fall, also wenn schon getrommelt wird, dh. die Variable drumming ist **true**, wird der Conductor angewiesen das Musikstück zu beenden. Das passiert über den Befehl midi:conductor.stop. Anschliesend wird noch drumming auf **false** gesetzt.

Das Setup des Models ist im Grunde auch einfach: Für drei Trommler werden die Noten und die Instrumente definiert bzw. dem Conductor mitgeteilt (Befehle midi:conductor.add.to.sheet und midi:instrument)

5.1.5 Der Source Code im Detail

Die Setup-Prozedur erstellt die Turtles die dann die Trommler darstellen und richtet das Midi-Interface sowie den Conductor ein. Es müssen die Instrumente gesetzt werden, da Kanal elf und zwölf nicht automatisch Percussion-Instrumente verwenden. In diesem Beispiel habe ich die Variante gewählt, dass der Conductor angewiesen wird zu dirigieren, bzw. angewiesen wird mit dem Dirigieren aufzuhören. Das wird in der Prozedur drum erledigt. Der Dirigent arbeitet, nach dem er gestartet wurde im Hintergrund. (Achtung: Es sei an dieser Stelle nochmal auf die Problematik mit Blocking-Calls hingewiesen. Siehe auch 4.1.2). dosmg für Do-Something, lässt die Turtles wandern.

```
extensions [midi]

globals [drumming doit]

to setup
  clear-all

  ;; sicher ist sicher
  midi:all.notes.off 10
  midi:all.notes.off 11
  midi:all.notes.off 12

  midi:conductor.clear.sheets
  midi:conductor.setplaymode.endless

  set drumming false
  set doit false

  midi:conductor.add.to.sheet 10 200 "midi:noteon 10 45 1"
  midi:conductor.add.to.sheet 10 200 "midi:noteon 10 45 0.7"
  midi:conductor.add.to.sheet 10 200 "midi:noteon 10 45 0.7"
  midi:conductor.add.to.sheet 10 200 "midi:noteon 10 45 0.7"
  ;midi:conductor.add.to.sheet 10 200 "midi:noteoff 10 45"

  if musicians > 1 [
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  1"
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  0.7"
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  0.7"
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  0.7"
  ]

  if musicians > 2 [
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 1"
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 0.7"
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 0.7"
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 0.7"
  ]

  ;setup the instruments
  midi:instrument 11 116
  midi:instrument 12 119

  ;create the turtles
  create-turtles (9 + musicians) [
    home
    facexy max-pxcor max-pycor
    rt 45
    fd who
    ifelse inverse_dir [lt (-1 ^ who) * 90 ][lt 90]
    set size 3
    ;; thicker line is easier to see
    set pen-size 3
    ;; leave a trail
    pen-down
  ]

  ; let the unused turtles die
  ask turtle 0 [die]
  ask turtle 1 [die]
  ask turtle 2 [die]
```

```
  ask turtle 3 [die]
  ask turtle 4 [die]
  ask turtle 5 [die]
  ask turtle 6 [die]
  ask turtle 7 [die]
  ask turtle 8 [die]
end

to dosmg
  ask turtles [
    let umf who * 2 * pi
    let turn 0.4 * speed

    fd 0.001 * speed * umf

    ifelse inverse_dir
     [lt (-1 ^ who) * turn]
     [lt turn]

    ; update turtle position
    midi:updateposition (who + 1)
  ]

  tick
end

to drum
  ifelse drumming = false[
    print "starting"
    midi:conductor.start
    set drumming true
  ]
  [
    print "stopping"
    midi:conductor.stop
    set drumming false
    set doit false
  ]
end
```

5.1.6 Ausblick

Ziel des Models ist es die Funktion "midi:updateposition" und den Conductor auf seine "start/stop"-Fähigkeit zu testen. Aus diesem Grund ist das Model sehr einfach gehalten. Ein paar Varianten um das Model zu erweitern sind:

- *Orchester* Im aktuellen Model marschieren nur ein bis drei Trommler im Kreis. Eine Erweiterung ist weitere Musiker mitspielen zu lassen. Das kann über verschiedene Wege passieren:
  - *komplett-statisch* Im Programm-Code wird festgelegt was ein einzelner Musiker, wenn ausgewählt, spielt. Das Interface wird um Switches für die einzelnen Musiker erweitert.
  - *teil-statisch* Das Interface wird erweitert, um die Möglichkeit für eine festgelegt Anzahl an Musikern, Noten und deren Werte einzugeben. Der Programm-Code muss diese Werte dann in die Notenblätter schreiben und dem Conductor mitteilen. Achtung: Die Erweiterung des Interfaces um die Eingabe ist bei weitem nicht einfach, da eine vernünftige Struktur gefunden werden muss, wie die Noten und ihre Werte eingeben werden

können. Auch im Programm-Code muss dann einiges an Abfragen passieren ob einzelne Felder gesetzt sind

  - *dynamisch* Die Musiker und ihre Noten werden über Dateien definiert. Das Programm wird so erweitert, dass es in einem Ordner nach Dateien mit einem normierten Dateinamen sucht und aus diesen die Noten und ihre Noten-Werte ausliest. Als Format für die Dateien ist ein einfaches CSV-Format ausreichend.

- *Marsch-Formationen* Anstatt die Trommler nur in Kreisbahnen gehen zu lassen, kann das Model um die Funktionalität erweitert werden, dass sich die Trommler auf selbst definierten Funktions-Bahnen bewegen. So kann das Interface um Eingabefelder erweitert werden wo für jeden Trommler die Grenzen für die Funktion und die Funktion, welche abhänig von einem Zeitparameter sein sollte, eingegeben werden kann. Als programmiertechnische Lösung kann zB. in der Setuproutine eine Liste mit Positionsänderungen angelegt werden, diese wird durch schrittweise Berechnung, der einzelnen vom Benutzer eingegebenen Funktionen berechnet.

5.2 Rettungsauto

5.2.1 Idee

Simulierte Rettungsautos sollen über den Bildschirm fahren und dabei das Martinshorn erklingen lassen. Es soll aber ein nahezu realistisches Model werden, was bedeutet, dass der Dopplereffekt ebenfalls simuliert werden soll. Der Einfachheit halber gehen wir von einer kugelförmigen Welt aus, die jedoch auf eine zweidimensionale Oberfläche abgebildet wird. Das bedeutet nichts anderes, als das die normale NetLogo-Oberfläche verwendet werden kann, die Töne sollen im zweidimensionalen Raum erscheinen und ein Auto, welches die Welt auf der einen Seite verlässt, soll auf der Anderen wieder hineinfahren. Für den Dopplereffekt soll davon ausgegangen werden, dass der Zuhörer in der Mitte, also auf der Koordinate (0/0) steht.

5.2.2 Didaktische Aufbereitung

Auch dieses Projekt sollte zuerst in eine Teilschritte zerlegt werden.

- Martins Horn
- Dopplereffekt
- Fahrendes Auto
- Mehrere Autos

Wie im vorherigen Beispiel eignet sich auch hier der Schritt zuerst mit MidiCSD die Töne zu entwickeln und dann erst mit NetLogo zu implementieren.

Martins Horn Es soll ein österreichisches Rettungsauto modelliert werden. Das Ziel-Instrument ist also eine Trompete. Die Lernenden sollen aber zuerst selber raten und ausprobieren welches Instrument sich am Besten eignet. Ist das Instrument gefunden soll vom Ton mit dem Wert 60 aus der zweite Ton gefunden werden. Der Zugang kann entweder über das Experimentieren also das Ausprobieren welcher Ton der Richtige ist, oder die Musik, also welches das richtige Intervall ist und wie dieses auf eine Midi-Tonhöhe umgerechnet wird gewählt werden.

Dopplereffekt Der Einfachheit halber sollte ein linearer Dopplereffekt betrachtet werden. Sollten die Lernenden den Dopplereffekt nicht schon ausführlich in Physik behandelt haben sollte eine Einführung in den Dopplereffekt und dessen richtige Berechnung gegeben werden. Über Tabellen in Excel in denen mit der Formel experimentiert wird, soll dann eine Formel für die Berechnung entwickelt werden. In der Beispiellösung wird die skalierte Position des Autos mit 0.7 multipliziert. Zu Beachten ist, dass in beiden Fällen mit -1 multipliziert wird, da ein ansteigender Ton einen positiven Wert und ein abfallender Ton einen negativen Wert erfordert. Für das verschieben des Tones gibt es den Befehl `pitch.bend` Dieser bekommt einen Wert zwischen -1 und $+1$. Pitchbending bedeutet nichts anderes als das 'Beugen' der Tonhöhe.

Fahrendes Auto Das fahrende Auto ist in diesem Fall sehr einfach. Es muss lediglich eine Turtle vorwärts bewegt werden. Um das Model noch etwas realistischer zu machen sollte zusätzlich noch der UpdatePosition Befehl verwendet werden, damit die Position auf der Ebene auch hörbar gemacht wird. Das ganze kann über einen Schieberegler der die Geschwindigkeit regeln soll und über den `every` -Befehl sehr schön und einfach gelöst werden.

Mehrere Autos In diesem Schritt sollte am Besten ein Schieberegler für die Anzahl der Autos eingebaut werden. Im Setup ändert sich dann das statt einer Turtle mehrere erstellt werden. Damit diese nicht alle parallel zu einander fahren und auch nicht vom gleichen Punkt aus starten eignet sich ein wiederholtes zurückgreifen auf Random. Im Code für das Fahren ändert sich nichts. Für das erstellen der Martinshörner ist jedoch ein kleiner Trick erforderlich: Werden alle Autos mit den gleichen Werten für die Notendauer versehen, klingt das nicht sehr gut. Anstatt jedes Auto bei 0 oder 1 beginnen zu lassen kann aber einfach mit `who` begonnen werden. Das bedeutet zwar, dass sich für das zweite oder dritte Auto die Töne etwas verschieben ist aber kaum hörbar. Um das ganze echt Zeitversetzt zu machen, wären aber einige Kunstgriffe notwendig. (Eine einfache Lösung ist aber mit einem nicht hörbaren Ton zu beginnen und dann in einer Schleife die Töne für das Martinshorn hinzuzufügen und 'Playmode normal' auszuwählen.)

5.2.3 Benutzer-Oberfläche

Die Abbildung 3 zeigt die Oberfläche für die Anwender des Modells. Links oben befindet sich der Knopf um das Modell zu initialisieren. Um die das Rettungsautofahren zulassen müssen die Knöpfe "run" und "drive" gedrückt werden. Drive lässt das Auto über die Fläche fahren, während "run" das Martinshorn aktiviert. Weiter unten befindet sich ein Regler um die Geschwindigkeit des Autos zu regeln. Sowie der Regler durch den sich die Anzahl der Autos festlegen lässt. Diese Änderung wird aber nur aktiv wenn erneut auf 'Setup' gedrückt wird.

5.2.4 Funktionsweise

Das Grundprinzip des Models ist einfach: Über die zwei Knöpfe werden unabhängig von einander zwei Prozeduren wiederholt ausgeführt. Die Prozedur für das Martinshorn enthält lediglich die Anweisung `conductor.conduct` um das Abspielen der Midi-Befehle für das Martinshorn zu bewirken. Die zweite Prozedur berechnet die Positionsänderung anhand der Parameter Geschwindigkeit und vergangener Zeit. Anschließend schiebt sie die Schildkröte, welche das Rettungsauto repräsentiert nach vorne und führt ein Positions-Update für die Schildkröte durch.

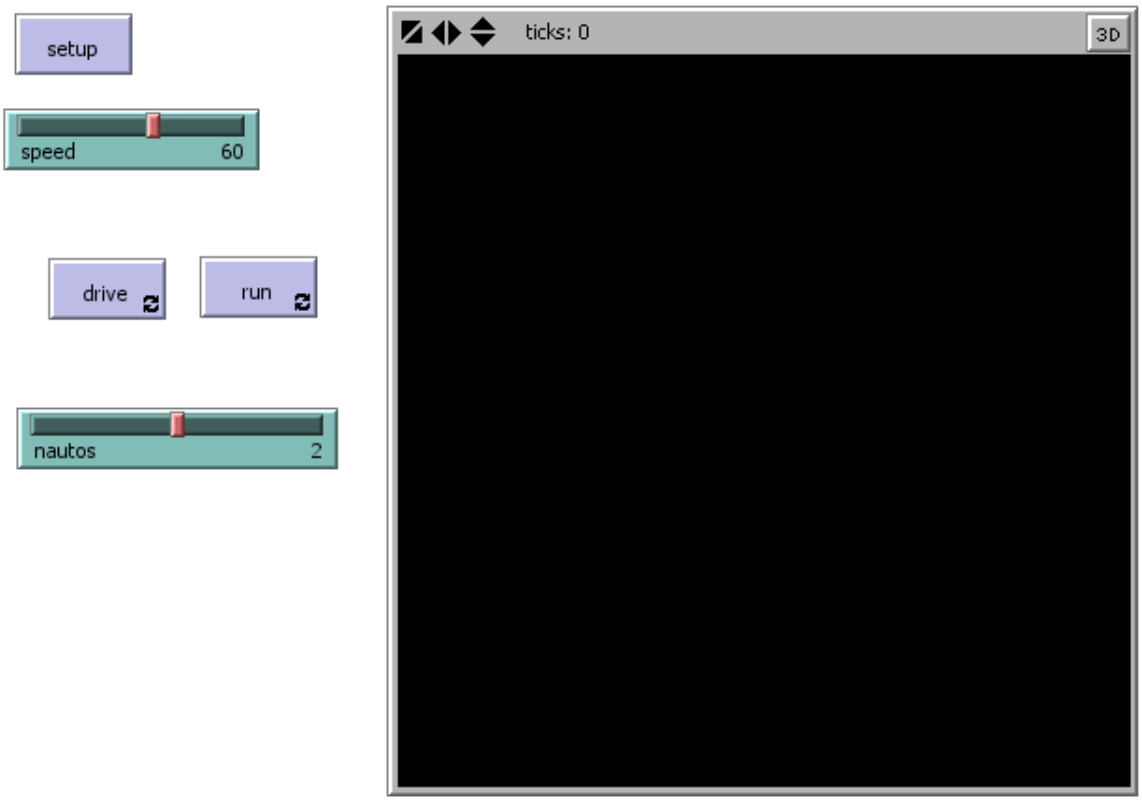

Abbildung 3: Rettungsauto - Interface

5.2.5 Der Source Code im Detail

Die Setup-Prozedur erstellt das virtuelle Rettungsauto und richtet den Conductor ein. Als Hilfsfunktion verwendet sie make.tatue um die Töne für das Rettungsauto zu erzeugen. Das Beispiel verwendet in diesem Fall, als Unterschied zum Trommler-Beispiel die Variante den Conductor in jedem Schritt einzeln aufzurufen und ihn dirigieren lassen. Damit kann das Auto unabhängig von seinem Martinshorn fahren lassen. Das Dirigieren wird in der kurzen Prozedur runit realisiert, das Fahren in drive.

```
extensions [midi]

; globals [lasttick velocity lasttime]

turtles-own[channel]

to setup
  clear-all
  random-seed new-seed

  midi:conductor.clear.sheets
  midi:all.notes.off 1

  create-turtles nautos[
    set size 10
```

```
    setxy ((random 200) - 100) ((random 200) - 100)
    facexy ((random 100) - 50) ((random 50) - 25)
    set channel who + 1
    midi:updateposition channel
  ]

  make.tatue

  reset-timer
  ; set lasttime 0
end

to make.tatue
  ask turtles[
    midi:conductor.add.to.sheet channel 0 + who "midi:noteon 1 60 1"
    midi:conductor.add.to.sheet channel 250 "midi:noteoff 1 60"
    midi:conductor.add.to.sheet channel 10 "midi:noteon 1 65 1"
    midi:conductor.add.to.sheet channel 250 "midi:noteoff 1 65"

    midi:instrument channel 57
  ]

  midi:conductor.setplaymode.endless
end

to runit
  midi:conductor.conduct
end

to drive
  every 0.25 [
  ask turtles [
    fd speed * 0.1

    every 1 [ midi:updateposition channel]

    if xcor < 0 [
      midi:pitch.bend channel ( 0.7 / min-pxcor * xcor * -1 )
    ]
    if xcor = 0 [ midi:pitch.bend channel 0]
    if xcor > 0 [
      midi:pitch.bend channel ( 0.7 / max-pxcor * xcor * -1 )
    ]
  ]

  ]

  tick
end
```

5.2.6 Ausblick

Das Modell zeigt zur Zeit nur elementare Fähigkeiten der Midi-Extension, kann aber natürlich beliebig erweitert werden.

- *Mehr Fahrzeuge:* Das Modell kann natürlich um weitere Autos erweitert werden. So ist es möglich über den Dirigenten Notenblätter für die verschiedenen Fahrzeuge anzulegen. So

kann das Model sehr einfach um Feuerwehr, Polizei, oder anderes erweitert werden. Soll sich auch die Art der Fortbewegung ändern muss die drive-Prozedur verändert werden.

- *Hindernisse:* Im aktuellen Model fahren die Autos ungehindert durch die Welt. Durch einfache Änderungen am Modell könnten breeds als Hindernisse deklariert werden. Fährt ein Auto auf eines von diesen auf, soll eine Aktion ausgeführt werden (zB das Auto soll wenden, also eine 180 Grad Drehung machen)

- *Fahrtverläufe:* In der aktuellen Implementierung kann lediglich der Startpunkt des Autos gesetzt werden, nicht jedoch der Verlauf der Fahrt. Ein mögliche Erweiterung ist, dass der Benutzer aus mehreren Funktionen auswählen oder diese auch selber eingeben kann, welche den Verlauf der Fahrt berechnen.

- *Genauerer Dopplereffekt:* Der Dopplereffekt wird im vorliegenden Modell nur approximiert. Durch Literatur-Recherche und geringfügige Änderungen am Code, kann sehr einfach eine bessere Annäherung erreicht werden.

Literatur

[1] Midi Manufacturer Assosiation. Gm 1 soundset. `http://www.midi.org/techspecs/gm1sound.php`. last seen: 18.11.2010.

[2] A Luedeke. Java-sound, midi. `http://www-lehre.informatik.uni-osnabrueck.de/~aluedeke/java/javaSound/midi.html`. last seen: 18.11.2010.

[3] E Neuwirth. Midicsd. `http://sunsite.univie.ac.at/musicfun/MidiCSD/`, 2008.

[4] Bomers F Pfisterer M. Java sound rescources: Faq: Midi programming. `http://www.jsresources.org/faq_midi.html`, 2000, 2005. last seen: 18.11.2010.

[5] U Schmidt. Oop mit java: Synchronisation von threads. `http://www.fh-wedel.de/~si/vorlesungen/java/OOPMitJava/Multithreading/Synchronisation.html`. last seen: 29.9.2010.

[6] U Wilensky. Netlogo. `http://ccl.northwestern.edu/netlogo/`, 1996.