**SCHOOL OF COMPUTING**

**UNIVERSITY OF TEESSIDE**

**MIDDLESBROUGH**

**TS1 3BA**

**Artificial Evolution**

**BSc Computer Science**

**Abelether Germa**

**11 April 2016**

**Supervisor: Eudes Diemoz**

**Second Reader:**

# Abstract

# Acknowledgements

**CONTENTS**

[Abstract ii](#_Toc446612249)

[Acknowledgements iii](#_Toc446612250)

[1 Introduction 6](#_Toc446612251)

[2 Methodology 7](#_Toc446612252)

[3 Research 8](#_Toc446612253)

[3.1 Netlogo 8](#_Toc446612254)

[3.2 Java 8](#_Toc446612255)

[3.3 Clojure 8](#_Toc446612256)

[3.4 Integration 8](#_Toc446612257)

[3.4.1 Communication between Java and Clojure 8](#_Toc446612258)

[3.5 Choice of software development tools used 8](#_Toc446612259)

[3.5.1 Programing software 8](#_Toc446612260)

[3.5.2 Visualization and environment 9](#_Toc446612261)

[3.5.3 Github 9](#_Toc446612262)

[3.5.4 Pandoc 9](#_Toc446612263)

[4 Design 10](#_Toc446612264)

[4.1 Concept Design 10](#_Toc446612265)

[4.2 Netlogo 10](#_Toc446612266)

[4.3 Java 10](#_Toc446612267)

[4.4 Clojure 10](#_Toc446612268)

[4.5 Integration 10](#_Toc446612269)

[5 Implementation 11](#_Toc446612270)

[5.1 NetLogo 11](#_Toc446612271)

[5.2 Java 11](#_Toc446612272)

[5.3 Clojure 11](#_Toc446612273)

[5.4 Integration 11](#_Toc446612274)

[5.4.1 Clojure and Netlogo 11](#_Toc446612275)

[6 Testing 13](#_Toc446612276)

[6.1 NetLogo 13](#_Toc446612277)

[6.2 Java 13](#_Toc446612278)

[6.3 Clojure 13](#_Toc446612279)

[6.4 Integration 13](#_Toc446612280)

[7 Evaluation 14](#_Toc446612281)

[8 Recommendations 15](#_Toc446612282)

[9 Conclusion 16](#_Toc446612283)

[List of Figures 17](#_Toc446612284)

[References 18](#_Toc446612285)

[Appendix A 19](#_Toc446612286)

# Introduction

# Methodology

# Research

## Netlogo

Within Netlogo Agents are referred as turtles. While speaking within Netlogo the term turtle would be used to refer to a species of agents and the term agent will be used for one specific agent.

### First iteration

#### What do agents eat?

For my first draft turtles are going to go around eating grass and try to avoid roads, while getting them to just eat grass is very simple, as different type of turtles are added who doesn’t eat grass this method is not going to work anymore and need to be changed.

To avoid changing it on the second iteration, this method can be implemented right now. To simplify the first iteration a decisation was made to

## Java

## Clojure

## Integration

### Communication between Java and Clojure

Clojure run on the JVM as well as Clojure programs run in a JVM. What this mean is Clojure itself was created using java and is in fact an application that run on the JVM just like any other Java application. This also means Clojure programs gets compiled into Java bytecode which the JVM executes and sends instructions to machine as shown in the figure below. Due to this, a Clojure can be viewed just like another java class and Vis versa.

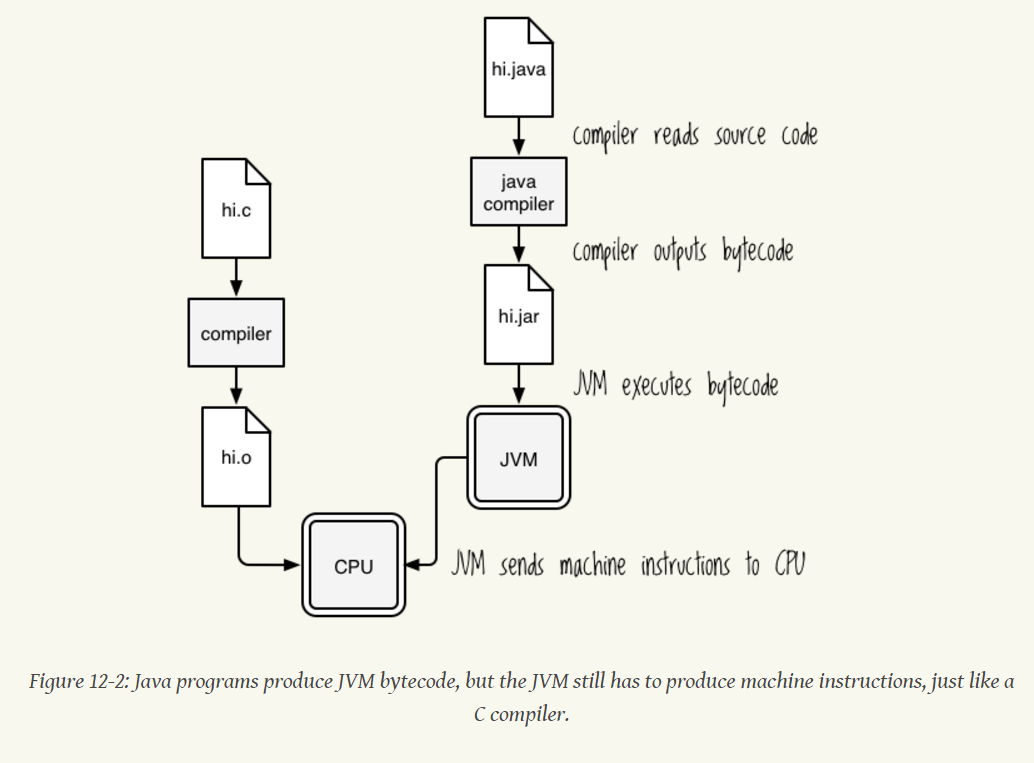


Figure 3.4.1‑2: Higginbotham, D. (2015) Clojure for the brave and true

Even though a Clojure and Java class are very alike, to access each other different syntax’s is used compared to accessing the same type of class.

For example in Clojure when referring another Clojure class, the following code is used.

(**require** [comm.wrapper :refer :all]

Require: loads the class

Refer : Adds specified item to the current name spaces.

Use : Require + User

## Choice of software development tools used

### Programing software

#### IntelliJ IDEA

Research was done on which what would be the best, integrated development environment (IDE) to use for this project. Three main and popular IDE’s were looked at, NetBeans, Intellij IDEA and Eclipse. All three IDE’s were used in past project therefore; experience did not matter on which one to use.

After looking at serval internet articles, it was clear that Intellij was a more superior IDE, mainly due to how everything can be done within the IDE including using the terminal (Command Prompt); it has a good Auto Completion support as well as support for git, which is the version control that is used for this project.

### Visualization and environment

#### Netlogo

Netlogo is an agent-based programming language and integrated modelling environment.

#### AnyLogic

### Github

For this project, I have chosen to use GitHub for Version control. I have realised that I will be using the university computers as well as my laptop home. While working on previous projects I found version control is one of the baggiest problems when working on multiple computers. I often found myself having multiple version not know in which version I implemented a feature. By using Github, not only would I be able to work on one version using multiple computers, it would also be backed up and I would have constant access to it as long as I have an internet access.

I first used git hub for my Software architecture group work and became accustomed with the advantages and disadvantages of using GitHub. For my project to efficiently use GitHub I took it upon myself to do a tutorial so I am aware of all the features it has to offer and I can use them efficiently.

### Pandoc

Pandoc is a universal document converter. I have looked into version control software for Microsoft word and found out it has a built in, version control but does not always work, as I would like them to. To make sure am not making any changes I would not want I am using Pandoc by editing the following files as shown in the figure below.



Figure 3.5.4‑1: Using Microsoft Word with git, Martin Fenner, (2014)

Using Pandoc I can now view all the changes I make before I save them, everything I have deleted and everything I have added. I did find other tools to make word version controllable using git but they were too complicated to set up and since I am the only person working on this, they were not necessary

# Design

This is the design section

## Concept Design

## Netlogo

## Java

## Clojure

## Integration

# Implementation

## NetLogo

## Java

## Clojure

## Integration

### Clojure and Netlogo

#### From Java to Clojure

Communication from Java to Clojure proved to be much more difficult than anticipated

#### From Clojure to Netlogo

To connect Clojure to Netlogo was very simple Thanks to S.Linch …

Using the wrapper was very simple and it was quickly tested using the REPL and Netlogo to confirm communication. There was some problems that occurred due to the Clojure environment.

##### Problem 1: Methods not recognised within REPL

A Clojure class called wrapper was created to handle all the communication between Clojure and Netlogo. All the necessary methods and imports were copied over to the class. In theory, this should have worked as it did on other project but when tested on the REPL, The REPL would not recognise the methods as shown in the following figure.

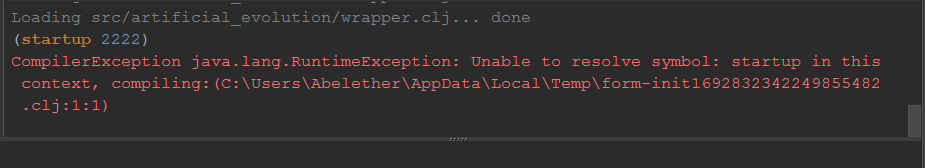


Figure 5.4.1‑1: Unable to resolve symbol Error when wrapper method was run.

Generally this would not be a problem when working on a Clojure project and can simply be solved by copying all the methods into the REPL and the REPL would recognise them but for this project it was not an option especially for it to be compiled and run as an application.

# Testing

## NetLogo

## Java

## Clojure

## Integration

### Iteration one

#### From Clojure to Netlogo

To test the communication between Clojure to Netlogo the REPL and Netlogo was used with the following steps.

1. Create a soccet with ‘(startup port\_number)’
2. Once socket is being advertised, connect to socket from Netlogo using the same port\_number. If connected, the message “Socket Connected” will appear within Netlogo command centre and “Socket Accepted” will appear within the REPL.
3. Send message to Netlogo using the following function (nlogo-send “Message”)
4. Within Netlogo click on execute REPL button and the message should appear with in the output box.

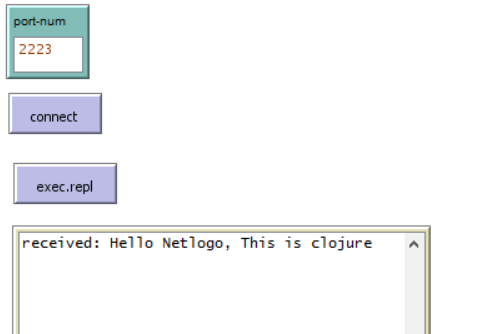
(**startup** 2223)  
(**nlogo-send** "Hello Netlogo, This is clojure")

advertising #object[java.net.ServerSocket 0x1bab8268 ServerSocket[addr=0.

0.0.0/0.0.0.0,localport=2223]]

socket accepted #object[java.net.Socket 0xa307a8c Socket[addr=/152.105.97

.113,port=1898,localport=2223]]



#### From Netlogo to Clojure

To test the communication between Clojure to Netlogo the REPL and Netlogo was used with the following steps.

1. Connect Clojure and Netlogo vie socket as shown in the previuse test ‘From Clojure to Netlogo’.
2. Within Netlogo in the command centre run the method ‘exec.repl2 “message”
3. Going back into the REPL run the following command ‘(nlogo-read)

# Evaluation

# Recommendations

# Conclusion

# List of Figures

[Figure 1 Halloween, *Zhu, Feng* (2013) 7](#_Toc446282554)

[Figure 2: Using Microsoft Word with git, Martin Fenner, (2014) 11](#_Toc446282555)

# References

# Appendix A