Institut für Informatik
Lehrstuhl für Programmierung und Softwaretechnik

LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Minecraft in MicroPsi

Eine populäre Videospielwelt als Simulationsumgebung für eine kognitive künstliche Intelligenz

Jonas Kemper

Medieninformatik Bachelor

Aufgabensteller:	Prof. Dr. Martin Wirsing
Betreuer:	Joscha Bach PhD, Annabelle Klarl
Abgabetermin:	19. September 2013

Ich versichere hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 19. September 2013

. . . . . .

(Unterschrift des Kandidaten)

Zusammenfassung

Ziel der Arbeit ist die Entwicklung und das Testen einer Simulationsumgebung für einen kognitiven Agenten auf Basis des populären Videospiels Minecraft.

Minecraft bietet sich als Grundlage für eine Simulationsumgebung aufgrund der kompositionalen Semantik der Spielwelt besonders an, da das Agentensystem durch das Erforschen seiner Umwelt Wissen über die Spielwelt aufbauen und ähnliche Strukturen wieder erkennen kann. Objekte der Spielwelt sind in Minecraft keine bloßen Hindernisse sondern werden mit variablen Eigenschaften prozedural erzeugt und ähneln so eher einer realen Umgebung als andere virtuelle Welten.

Da Minecraft von Anfang an mit einem Mehrspielermodus ausgestattet wurde, lässt es sich zudem für Multiagenten-Umgebungen und so für kollaborative Agenten verwenden. Des Weiteren sind Minecraft-Lizenzen günstig zu erwerben, erhältlich für viele Plattformen und es gibt eine äußerst große und aktive Community für selbsterstellte Spiel-Inhalte und -Modifikationen.

Angestrebt wird, dass die kognitive Architektur MicroPsi2 sich an einen Minecraft-Server anmelden kann, ihre Umgebung wenigstens in Ansätzen wahrnimmt (Objekte, Terraintypen), sich fortbewegen kann und einfache Interaktionsmöglichkeiten besitzt (z.B. Objekt aufnehmen und ablegen).

Daneben soll eine Visualisierung der Umgebung aus Sicht des Agenten entstehen, z.B. als zweidimensionale Übersicht, in deren Zentrum der Agent steht. Diese Visualisierung soll live im Browser angezeigt werden (wahlweise mit WebGL oder Canvas/D3).

Zum Schluss der Arbeit soll zum Testen der Funktionalität ein virtuelles Braitenbergvehikel in die Simulationsumgebung gesetzt werden, welches sich daraufhin auf die nächstgelegene Lichtquelle zubewegen soll. Dieses Experiment wird als Teil der Arbeit umfangreich dokumentiert.

Abstract

What I cannot create, I do not understand Richard P. Feynman, 1988

Danksagung

Ich danke meinen Betreuern Joscha Bach und Annabelle Klarl sowie Dominik und Professor Wirsing.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objective . . . 2
- 1.3 Approach . . . 2
- 1.4 Outcome . . . 2
- 1.5 Outline . . . 2

2 Background **3**
- 2.1 A! Artificial Intelligence / Examples for A.I. applications . . . 3
  - 2.1.1 weak AI . . . 3
  - 2.1.2 strong AI . . . 3
- 2.2 A! Cognitive AI / cognitive AI . . . 3
  - 2.2.1 Concepts . . . 4
- 2.3 A! Psi Theory . . . 4
  - 2.3.1 Joschas Contribution: MicroPsi . . . 4
- 2.4 A? Summary . . . 4

3 Foundations **5**
- 3.1 Psi Implementations . . . 5
- 3.2 MicroPsi 2 Module Overview . . . 6
  - 3.2.1 Agents . . . 7
  - 3.2.2 Environment . . . 7
- 3.3 A! Minecraft . . . 7
  - 3.3.1 What is a Minecraft world? . . . 8
  - 3.3.2 The Client Server Protocol . . . 9
  - 3.3.3 Suitability of Minecraft as a simulation environment . . . 10
- 3.4 A! Minecraft Bots . . . 11
  - 3.4.1 A! Overview / What has been there so far? . . . 11
  - 3.4.2 Spockbot von Nickelpro . . . 11
  - 3.4.3 Protocol Implementation in spock . . . 11
- 3.5 Python Multimedia Libraries and Minecraft Clones . . . 12
  - 3.5.1 "Minecraft" by Michael Fogleman . . . 12

4 A! Approach / Minecraft as a Simulation Environment for MicroPsi 2 **15**
- 4.1 A! Architecture / Building the interface in between Minecraft and the simulation environment . . . 16
  - 4.1.1 Implementing the Bot in MicroPsi . . . 16
  - 4.1.2 The MicroPsi side . . . 16

4.1.3 necessary Modifications and Additions in core/worldrunner . . . 16
4.1.4 necessary Modifications and Additions in server/control and monitoring interface . . . 16
4.1.5 The Visualization . . . 16
4.1.5.1 Requirements and necessity of a visualization . . . 16
4.1.5.2 Implementation . . . 17
4.1.5.3 3D Visualisierung mit Pyglet . . . 17
4.1.5.4 Earlier attempts using JavaScript / AJAX . . . 18
4.2 A! Implementation . . . 18
4.3 A! Case Study . . . 19
4.3.1 Experiment . . . 19
4.3.1.1 Braitenberg Vehicle . . . 19

5 Conclusion **21**
5.1 What's next . . . 21

Appendix A Auszug aus dem Buch **23**

Appendix B Implementierungen **25**

Abbildungsverzeichnis **27**

Tabellenverzeichnis **29**

Inhalt der beigelegten CD **31**

Literaturverzeichnis **33**

Chapter 1

Introduction

The hunt for artificial intelligence started many years ago. Dividing the subject into the strong and weak part means separating its goals into useful applications and those that try to learn about the nature of intelligence itself. The ultimate task of strong A.I. — recreating human intelligence — admittedly still seems to be science fiction, though.

But then again: a new generation of cognitive scientists, psychologists and computer scientists strives to implement new Ideas for simulated cognition — building cognitive architectures. Many of them do so by simulating, in one way or the other, what could be called "neural networks(/nodenets)".

One of these architectures is MicroPsi 2. Developed by Joscha Bach, it is based on the Psi Theory by Dietrich Dörner, who is a german Professor for theoretical psychology. It aims at providing a solid and complete implementation of that theory of cognition and at being easily accessible, understandable and modifiable for the research and applications to come.

To test the functionality of such cognitive architectures and to figure out their capabilities and potential, we need to research their behavior inside defined environments. As implementing AI into the physical world (as robots for examples) requires building apropriate hardware and patience, computer-simulated environments therefore play an important role.

MicroPsi 2 is both: a cognitive architecture but also a set of and an interface to simulation environments.

1.1 Motivation

Video games are natural applications of artificial intelligence. The quality of a games A.I. can make all the difference in between a great title and an uninspired demo of computer graphics technology.

Building an interface between the cognitive architecture MicroPsi 2 and a video game world is what this Thesis is about.

1.2 Objective

1.3 Approach

1.4 Outcome

1.5 Outline

...

Chapter 2

Background

A widely accepted definition of the field of A.I. is that it is "the study and design of intelligent agents",[1] where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success.[2] (TODO find original sources)

2.1 A! Artificial Intelligence / Examples for A.I. applications

It took many years for A.I. research to evolve from the early ideas of thinking machines over Deep Blue, the computer that could beat mankind's best chess players to Watson, the A.I. that beats the champions of Jeopardy, the game show, that is about asking the appropriate question to a given answer. There are many other applications of A.I. out there. Self-driving cars and online-shopping recommendation systems to name a few.

... related work ...

2.1.1 weak AI

The mentioned examples for A.I. have one thing in common. They are applications of technology that serve an immediate, or at least foreseeable purpose. For A.I. in scenarios of this kind. The term "weak A.I." (or "applied A.I." has been coined.

2.1.2 strong AI

"Strong A.I.", in contrast, is about researching the nature of intelligence and consciousness themselves. An actual (hypothetical) implementation of a Strong A.I. would mean building a machine, that is capable of acting like a human being. Not just in a specialized and defined problem field, but in all of them.

2.2 A! Cognitive AI / cognitive AI

Cognitive A.I. can be thought of as architectures that implement findings and theories in cognitive science and psychology as well as the neuro-sciences for the sake of proving, if the theories hold against what they promise.

... Examples for cognitive architectures are ...

2.2.1 Concepts

Many cognitive architectures share characteristics with or implement artificial neural nodenets.

... Approaches to cognitive AI ... ... neural node nets ... ... related work ...

2.3 A! Psi Theory

The Psi theory in its foundations was described by german psychologist Dietrich Dörner in his book "Bauplan für eine Seele" and "Die Mechanik des Seelenwagens" from 1998 and 2002. It's main ideas are that it thinks of cognition as a graph-like structure (e.g. node-net) of relationships that strives to maintain homeostatic balance.[Bac09]

Basic components of the theory are Representation, Memory, Perception, Drives, Cognitive modulation and emotion, Motivation, Learning and Problem solving as well as Language and consciousness. (wikipedia)

"The PSI theory is a theory of human action regulation by psychologist Dietrich Doerner [1, 2]. It is an attempt to create a body-mind link for virtual agents. It aims at the integration of cognitive processes, emotions and motivation. This theory is unusual in cognitive psychology in that emotions are not explicitly defined but emerge from modulation of perception, action-selection, planning and memory access. Emotions become apparent when the agents interact with the environment and display expressive behavior, resulting in a configuration that resembles emotional episodes in biological agents. The agents react to the environment by forming memories, expectations and immediate evaluations. This short presentation is a good overview.

PSI agents possess a number of modulators and built-in motivators that lie within a range of intensities. These parameters combined to produce complex behavior that can be interpreted as being emotional. Additionally, by regulating these parameters, the agent is able to adapt itself to the different circumstances in the environment.This theory has been applied to different virtual agent simulations in different types of environments [3, 4, 5, 6, 7, 8] and has proven to be a promising theory for creation of biologically plausible agents." (wikipedia)

... Basics of Psi Theorie of Dörner ...

2.3.1 Joschas Contribution: MicroPsi

Joscha Bach adapted that theory to bring it in a contemporary from with his own slight modifications.

... explanation of Joschas Dissertation ...

2.4 A? Summary

Even though building a conscious machine that thinks and acts like we do is still mehre science-fiction, it is this kind of foundational research, that leads no new ways of thinking of the world, that give us our most import leaps.

... still a lot to do in AI ...

Chapter 3

Foundations

This project is fundamentally about combining existing technologies. In particular, the two most important ones are MicroPsi 2, the most ambitious framework aiming to implement the ideas of the Psi theory, and Minecraft, the super-popular sandbox-videogame.

To understand how and why they were chosen, a brief history of their creation as well as explanations of their basic ideas and relevant insights to their architecture are what this chapter is about.

3.1 Psi Implementations

Psi has been implemented by different groups at different times. The first implementations are by Dörner and his associates themselves. They used Pascal and developed it for windows environments. This implementation can still be downloaded and runs on Windows 7 installations, for example.

... psi screenshot ...

The work on Dörner's team's implementation has not been continued, so Joscha Bach and his associates built new implementations of Psi.

From 2003 to 2009 they built an implementation in Java as a set of plugins for the Eclipse IDE called MicroPsi. It consisted of a graphical editor and a 3D simulation-environment. Aiming at better understandability and to maintain platform independence, MicroPsi has been built ground up again in 2012 — using more lightweight Python code. What is remarkable about the new implementation called MicroPsi 2 (in the following MicroPsi), is that the graphical interface is completely rendered inside a webbrowser — using state-of-the-art internet- and webapplication-technologies. [Bac12]

The MicroPsi user interface is rendered completely inside a web browser and the simulation is deployed as a web application. The webinterface is based upon HTML/-Javascript and communication in between the browser and the simulation is managed via JSON remote procedure calls. Many GUI components of Twitter's Bootstrap library and the JavaScript graphics library PaperJS are used. [Bac12]

Matching the concepts of the Psi Theory, MicroPsi simulates agents as neuro-symbolic spreading activation networks. Agents can be placed and researched in simulation environments or physically embodied as robots. [Bac12]

Even though there have been more complex simulation environments (e.g. 3D-worlds) for previous implementations of Psi-architectures, the relatively new MicroPsi has only two fairly simple ones: a 2D-Island and a map of the public transportation

system of Berlin. Instead of building another 3D-world, with this project we set out for something more experimental.

3.2 MicroPsi 2 Module Overview

MicroPsi modular structure is fairly easy to understand (see figure 3.1). At first, one can differentiate between the server component (or the web-interface) and the actual simulation code (called "core").

In a minimal setup MicroPsi runs three threads. One thread for the webserver, one for the world simulation and one for every world-adapter (or agent). If more then one agent or more than one world are launched, they are instantiated as additional threads.

Furthermore, the core consists of a runtime component, a user and configuration manager. The runtime works independently of the server and can also by deployed for commandline interaction or other GUIs. It manages the simulations worlds as well as the agents (node net embodiments) and the world-adapters. [Bac12]

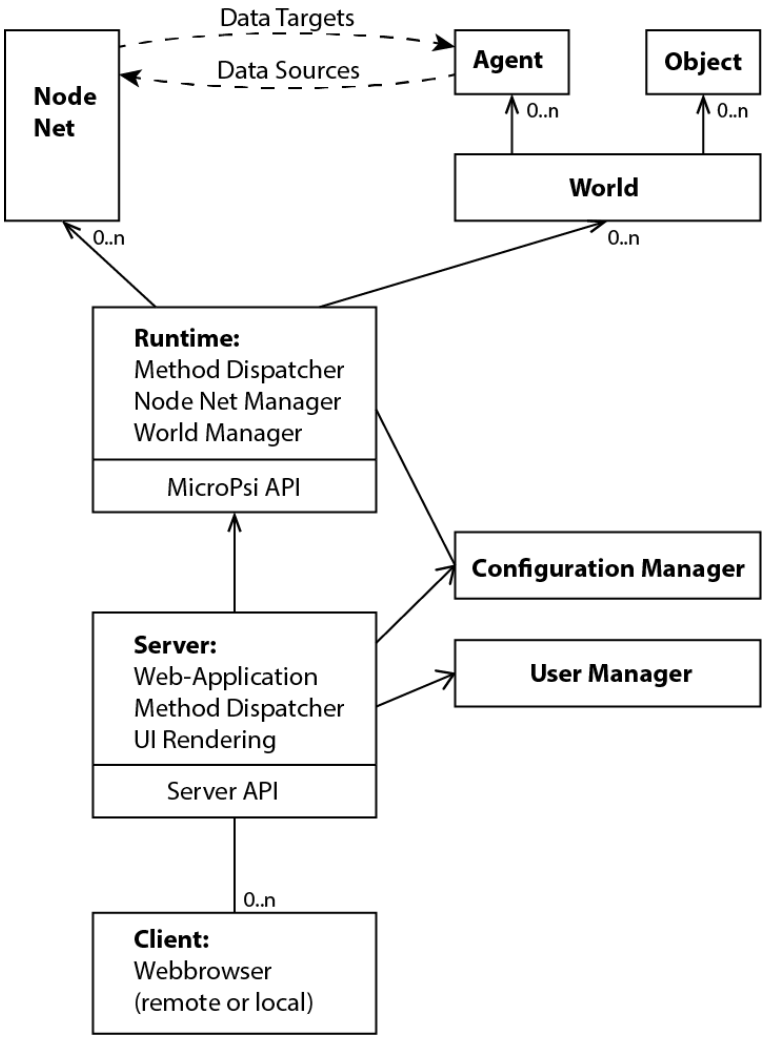

Figure 3.1: *The MicroPsi 2 architecture [Bac12]*

3.2.1 Agents

MicroPsi defines agents as node nets or, to be more specific, hierarchical spreading activation networks. They are an "abstraction of the information processing provided by brains". The assigned world-adapter provides data sources and data targets to manage the communication in between the node net and the simulation environment. They represent the agent's sensory input and motoric output. Sophisticated interconnection of those enables interaction with the environment. [Bac12]

3.2.2 Environment

The simulations worlds are the environments in which we can study our agent's behavior. Worlds need to provide a worldadapter — the interface in between the node net and the simulation. Data sources and data targets have to be defined, to get a functional and meaningful experiment going. Node nets and environments may be updated asynchronously. [Bac12]

The kind of data the world adapter interfaces, is not specified any further, which gives developers the opportunity to experiment with classic simulation worlds as well as exotic applications (eg. stock data). [Bac12]

At the time of the development of the original development of the framework, the priorized application was building a framework for knowledge representation. [Bac12]

At the time of the writing of this Thesis, MicroPsi comes with two simulation environments: an 2D-Island world and a simulation of the public transportation system of Berlin.

... screenshot of Island and Berlin ...

3.3 A! Minecraft

The story of Minecraft has many interesting aspects, but first and foremost it is a story of immense, unexpected success.

When Markus ("Notch") Persson built and released the first public version of Minecraft, it soon became clear that his creation resonated with many people. The simple concept of a world entirely build of standard sized building blocks which the player can to create, destroy and relocate one-by-one, let many gamers employ their creativity, explore the Minecraft world and test out it's possibilities and boundaries.

The game is attracted great attention ever since its first release in 2009. Since copies of the game can be obtained commercially for the first time in 2011, different versions of the game sold more than 26 million times — with the PC version priced at about 20 Euros, for example. It should be noted, that Minecraft's development studio Mojang is a so called "indie game developer", that is not associated with any classical game publisher, but distributes copies of their game exclusively via their own website.

Although the game can be downloaded and played as a single packet of software, many scenarios of playing the game consist of running a Minecraft server software, as well as one copy of the client software for each player. It is possible to mimic the official client by implementing the reverse-engineered Client-Server-Protocol and build artificial players that way.

Minecraft is a complex yet easily accessible virtual world. It is constantly developed and new features are added regularly. It has a massive fanbase and a huge community of game-modifications.

Another interesting aspect about Minecraft is the procedural semantic the game world is generated with and. Trees in Minecraft, for example, may share a similar structure that consists of a trunk and branches and leaves spreading out as fractals, but the particular characteristics of each tree are generated randomly. This makes a Minecraft world somewhat more realistic than most other videogames.

3.3.1 What is a Minecraft world?

In this section basic concepts of the game are described (in regards to our A.I. interface). Minecraft worlds are build out of blocks (see figure 3.2). Blocks are cubes. There are different types (materials) of blocks and they share a single size, which converts to the basic distance unit of Minecraft. One unit can be thought of as roughly equaling one meter.

Figure 3.2: *A "grass" Minecraft Block*

A chunk (see figure 3.3) is a segment of the Minecraft world that is 16 blocks long, 16 blocks wide and 256 blocks high (or deep) and therefore consists of up to 65,536 blocks.. [mcw13]

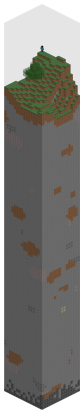

Figure 3.3: *A chunk*

"The player" (see figure 3.4) is what the playable game-character in Minecraft is called. It is usually displayed as humanoid.

Figure 3.4: *"the player" (CC-BY-3.0 Mojang AB) [ima13]*

Also, a Minecraft world has a day-night-cycle with 24 hours converting to 14 minutes by default

The game itself has no predefined goals. Players can walk around, discover the generated world (see figure 3.5 1) and collect resources by “destroying” blocks , with

the generated resources equaling the block type. They can combine different resources to “craft” items. For example, a player can destroy the blocks that represent a tree (2). The gained “wood” ressource could then be used to craft a wooden pickaxe, which could then be used to dig into the ground more effectively to “mine” more rare resources, like iron or gold.

Figure 3.5: *Minecraft Basic Mechanics (CC-BY-3.0 Mojang AB) [ima13]*

There exist different game modes. The original ”survival” mode adds monsters that attack the player at night. What solutions to survive the player comes up with (eg. building shelter or fighting the monsters) is left up to him or her.

In ”creative” mode, the player is not beeing attacked by monsters, has the ability to fly and instant access to unlimited ressources.

The mode of a Minecraft world does not effect the functionality of this project.

3.3.2 The Client Server Protocol

Minecrafts Client-Server-Protocol is not publicly documented by the developers themselves. However, the modding-community gathered full knowledge and understanding of its structure (probably by using reverse engineering techniques). The protocol is based on packets.

Packets are either ”server to client”, ”client to server” or ”Two-Way” and begin with a ”Packet ID” byte. The structure of the packet’s payload depends on it’s Packet ID.

To give an example of one of the easier packets, the ”Client Position”-Packet is fairly straight forward (see figure 3.1). It is exclusively send form clients to servers and starts with it’s Packet ID (as every packet does), followed by the X- and Y-coordinates as doubles, the stance value as a double, which is used to to modify the players bounding box, another double for the Z-coordinate and eventually a boolean that describes if the player is on the ground or not. [pro13]

Field Name	Field Type	Notes
Packet ID	Byte	0x0B
X	double	Absolute position
Y	double	Absolute position
Stance	double	Used to modify the players bounding box
Z	double	Absolute position
On Ground	boolean	Derived from packet 0x0A

Table 3.1: *Structure of packet Player Position (0x0B) [pro13]*

Knowledge of this datastructure is already sufficient to move around in the Minecraft world. To go forward, one has to figure out the players current position, calculate the absolute coordinates of the destination of the movement in regard of it and send a "Client Position" packet with these coordinates to the server. If the destination is not more than 100 blocks away from the origin, the server accepts the packet. In the official Minecraft client, a players movement from one point to the other is rendered with a smooth transition.

Other than movement, there are defined packets(-structures) for every aspect of the game. May it be the initial handshake, the creation or destruction of blocks or activity of other player- or non-player-characters

... the language an external client needs to speak, to take place in a Minecraft world ...

3.3.3 Suitability of Minecraft as a simulation environment

There are a number of reasons why using Minecraft as a simulation environment could be useful and lead to interesting results.

First, the game itself is easily accessible. It is developed using Java (for both the client and the server software) and therefore somewhat platform independent. The "desktop computer version" is sold for Windows, Mac OS X and Linux devices. There are official ports to Android, iOS, Xbox 360, the Raspberry Pi an a version for the upcoming "Xbox One" is announced. The desktop versions are priced at 19.95 euros which makes it affordable to a large audience.

The game itself already has an enormous audience. It is (like most videogames) especially popular with teenagers (or upcoming scientists). Minecraft being loved by so many people could benefit this project as for giving it more attention.

The game's developer has proven many times (proof?) that it acts very friendly to other developers when it comes to building game modifications and creating content that uses and/or changes original Minecraft content. In other words: They are not at all restrictive, when it comes to user people doing all kinds of things with their creations. This led to the availability of a fairly complete community-sourced documentation and explanation of virtually every aspect of the game — including it's software architecture, data structure and protocols. This is useful for this project, as chances are low that they will have anything against this project in the foreseeable future. (In fact, the game's A.I. creator Jon Kagström seems to be fond of this project)

The Minecraft world with it's logic, semantic and functions offers possibilities for an A.I. to proof being able to interact with the environment both on very simple and primitive ways (e.g. moving around) as well as increasingly complex tasks like building, collecting resources, craft items from blocks and interacting appropriately with both

well-disposed and hostile other incarnations.

The semantics of the gameworld share characteristics with the real world. Moving through the world one quickly realizes that it is build up out of different biomes (eg. forest or tundra). Also trees, rivers, mountains and ore veins are not hardcoded but generated procedurally and their structure appears to be (somewhat) fractal.

... cheap licenses ... ... developer friendly community and game-studio ... ... sandbox game with many possibilities but no pre-defined goals ... ... procedural semantic ...

Using Minecraft as a simulation environment will give the Psi agent possibilities to show off what kind of sophisticated behavior it is capable of. ... a more complex simulation environment could be fun ...

3.4 A! Minecraft Bots

... other popular Bot projects and game modifications ...

3.4.1 A! Overview / What has been there so far?

There exist many projects , that could be considered minecraft “bots”. One has to differentiate in between two types. On the one hand there are those, that mimic an entire client software and facilitate communication with the server on the default client software’s behalf. On the other hand there are bots which are modifications of the original client (or server) software and usually add non player characters — like animals and other non-human creatures — to the game. The code is usually injected through one of the popular ”modloaders” (eg. Minecraft Forge).

One example (and probably the most advanced one) for an entire bot framework that replaces the client is Mineflayer. [git13a] It has a high-level abstraction of the environment (eg. entity knowledge and tracking) and is written in JavaScript using node.js . However, it has not been used for this project, because a python implementation was aimed for.

Opposed to Mineflayer, an example for a game modification bot are the ”Cubebots” — fan-made cubes that aim to help Minecraft players with mundane tasks.[**?**]

... Minecraft Bots with simple as well as sophisticated AI ... ... MicroPsi 2 with Island and Berlin world ...

3.4.2 Spockbot von Nickelpro

Developed by Nick Gamberini, spock is an open-source (MIT license) bot framework (and therefore a Minecraft client) written in Python. It has been chosen as an essential part of this project for two reasons: Being written in Python it painlessly integrates in existing MicroPsi code and the absence of dependencies (with on exception) leave the code understandable and easy to deploy.

3.4.3 Protocol Implementation in spock

Inside the bot-framework spock, the protocol is implemented as follows:

There are datastructures, that describe the Minecraft protocol.

These are used, to parse packets as follows:

```
blocks = {
    0x00: "Air",
    0x01: "Stone",
    0x02: "Grass Block",
    ...
```

Figure 3.6: *data structure for block types*

```
names = {
    0x00: "Keep Alive",
    0x01: "Login Request",
    0x02: "Handshake",
    0x03: "Chat Message",
    ...
```

Figure 3.7: *data structure for packet types*

3.5 Python Multimedia Libraries and Minecraft Clones

Minecraft's success inspired many other projects - including a number of Clones (or Minecraft like games) in a wide variety of programming languages and environments.

Interesting projects include Skycraft [sky13], a Minecraft-like browser game based on WebGL.

3.5.1 "Minecraft" by Michael Fogleman

A particular project, that has the same name as the original game, that inspired it, is "Minecraft" by Michael Fogleman. It is a simple Minecraft clone in under 600 lines of Python and gained some popularity on reddit [fog13a] and Hacker News. [fog13b]

It is comparably easy to understand and modify. It is based on the Python multimedia library pyglet (http://www.pyglet.org/) .

```
structs = {
   #Keep-alive
   0x00: ("int", "value"),
   #Login request
   0x01: (
         ("int", "entity_id"),
         ("string", "level_type"),
         ("byte", "game_mode"),
         ("byte", "dimension"),
         ("byte", "difficulty"),
         ("byte", "not_used"),
         ("ubyte", "max_players")),
```

Figure 3.8: *data structure for the packets structures*

```
   def decode(self, bbuff):
      #Ident
      self.ident = datautils.unpack(bbuff, 'ubyte')

      #print hex(self.ident)

      #Payload
      for dtype, name in mcdata.structs[self.ident][self.direction]:
         self.data[name] = datautils.unpack(bbuff, dtype)
```

Figure 3.9: *function for decoding packets*

Chapter 4

A! Approach / Minecraft as a Simulation Environment for MicroPsi 2

The objective of this thesis is to build and test an interface in between MicroPsi and Minecraft, so that a Minecraft world (e.g. server) can be used as a simulation environment for the MicroPsi 2 Framework, which will act as an artificial player.

The modular architecture of MicroPsi 2 allows it to add new simulation environments (or worlds, as they are called in MicroPsi) fairly easily. A world needs interfaces to Data Sources and Data Targets and to a step-function that evolves the world and is being called frequently by the MicroPsi core. These interfaces are provided by the so called world adapter.

On the other hand, communication with a Minecraft Server typically requires a constant flow of data packets going in and out. Most third party clients, including Bots, facilitate own event loops.

For this project, the event loop of the bot framework had to be dissolved and rebuilt as the step function of the MicroPsi world. It should be noted, that the frequency in which the framework steps the bot has to be at least chosen so, that it is able to send keep-alive-signals to Server, to not get kicked from the server.

That being said, a big part of the project is about visualization. Inside the MicroPsi Core Application, a 3D-visualization of the Minecraft world and the agent within is generated. There are two main reasons for this. The first reason is, that the agents behavior within the simulation environment is supposed to be visually monitored from the MicroPsi webinterface. The second reason is, that the image data is supposed to be processed by the agent as one of it's "senses".

Do obtain these goals, the Minecraft Client-Server-Protocol had to be researched, learned and imitated.

Then, artificial Minecraft players (written in Python) had to be generated. Fortunately the code-base of an open-source bot-framework could be used for many parts of the low-level protocol implementation.

Eventually, the bot-framework had to be integrated into the MicroPsi framework.

4.1 A! Architecture / Building the interface in between Minecraft and the simulation environment

... result: a Minecraft Bot that implements MicroPsi AI and is controlled and monitored via the MicroPsi webinterface ... ... the Webinterface holds its own visualization of the Agents worldview ...

4.1.1 Implementing the Bot in MicroPsi

To make the bot work as a simulation world, two main challenges had to be overcome. First, the event loop and handling of spock had to be modified to work with MicroPsi. Therefore, the functions that were usually called from within the event-loop now have to be called from within the step function of the world-adapter. The same holds for event handling.

Second, a system for communicating "sensory data" and commands in between the bot (or Minecraft world) and the world adapter had to be implemented. Up to a certain degree, spocks plug-in system could be facilitated (well ...). In most/ cases, though, sending commands and receiving data (...) had to be be implemented on packet level.

Sending packets in spock is fairly easy. (see figure 4.3

```
1                'x': (client.position['x'] + 1)  / 1,
2                   'y': client.position['y'] / 1,
3                'z': client.position['z'] / 1,
4                'on_ground': False,
5                'stance': client.position['y'] + 0.11
6                }))
```

Figure 4.1: *Sending a packet that moves the agent on block in x direction*

... spocks plugin system ...

4.1.2 The MicroPsi side

The worldadapter spins-up and steps a spockbot as follows:

Inside spock this looks like this:

4.1.3 necessary Modifications and Additions in core/worldrunner

Data Targets and sources

4.1.4 necessary Modifications and Additions in server/control and monitoring interface

4.1.5 The Visualization

Early ideation made it clear, that this project should contain a visualization component.

4.1.5.1 Requirements and necessity of a visualization

The visualization is supposed to serve two causes. First, it should make it effective and pleasurable to monitor the bot's behavior as well as the environment it is behaving in.

```
    def step(self):
        if self.first_step: #TODO probably not too smart
            # launch minecraft bot
            plugins = [DebugPlugin.DebugPlugin, ChatMessage.ChatMessagePlugin, Chu
            self.client = Client(plugins=plugins)
            self.client.start()
            vis.commence_vis(self.client)
            self.first_step = False

        self.chat_ping_counter += 1
        if self.chat_ping_counter % 100 == 0: #TODO find other way to send "keepal
            self.client.push(Packet(ident = 0x03, data = {
                  'text': "I'm alive! ping %s" % (self.chat_ping_counter) }))
        World.step(self)
        self.client.step()
        vis.step_vis()
```

Figure 4.2: *The MicroPsi world-adapter spins up and steps a spock bot*

Figure 4.3: *The MicroPsi world-adapter spins up and steps a spock bot*

Second, it is supposed to serve the agent as a datasource itself. This means, that from pure world data (what block sits where) a 3D-visualization has to be generated from within the MicroPsi Python Code. It should contain a perspective that gives a good overview over the bots environment to forward to the webinterface, as well as a first person perspective of the agent, to function as its eyes.

As the open-source game, which has been used to implement the visualization, uses the graphics and game library pyglet (that basically encapsulates PyOpenGL), which brings it's own event-loop and -handling, the event-loop had to by broken apart and rebuilt as a part of the world adapter (that advances the visualization with every step) .

monitoring the bot from the webinterface ... make it aesthetically appealing as well as easily accessible ...

using visualization output as a Datasource / as the bots eyes

4.1.5.2 Implementation

It has been implemented as follows:

used Data ... required Data for the visualization ... ... and how to obtain it (first attempts: telnets/then sockets) ...

4.1.5.3 3D Visualisierung mit Pyglet

Fortunately, a (Minecraft inspired) open source (MIT license) project has been found, that implements a (very) simple Minecraft clone in (under 500 lines of) Python.[git13b]

The code of this project could be facilitated to serve as a visualization. ... additional 3D models, blocks an graphics had to be added ...

4.1.5.4 Earlier attempts using JavaScript / AJAX

... worked well but a little slow ...

4.2 A! Implementation

Viewing the resulting software as a whole, several modules have been added to the architecture: the Minecraft world adapter, the spock bot (communicating with a Minecraft Server) and the (pyglet) Visualization. (see figure 4.4)

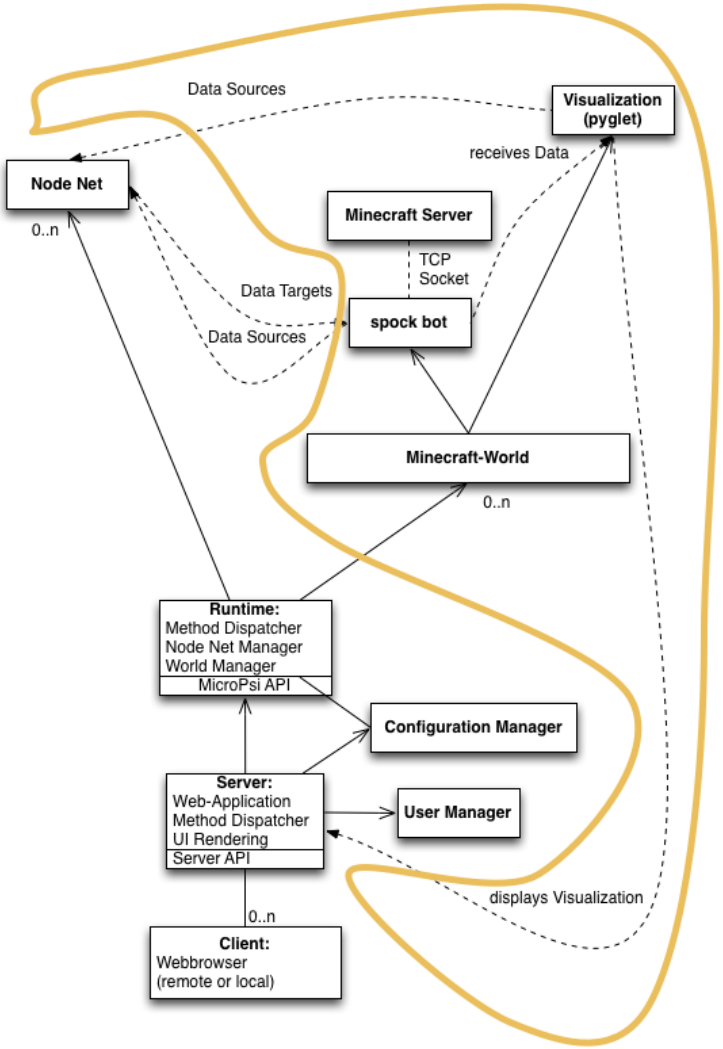

Figure 4.4: *The new architecture of MicroPsi with the Minecraft interface. New modules are framed orange.*

... graphic/UML of the project as a whole ... ... illustration of the event loop ...

4.3 A! Case Study

The functionality is to be tested with a simple Braitenberg-vehicle experiment.

4.3.1 Experiment

... experiment to test functionality of the system ... ... scope: only a simple test for time reasons ...

4.3.1.1 Braitenberg Vehicle

... simplest proof of concept of a microPsi agent ...

Chapter 5

Conclusion

After several iterations and trying out different approaches and technologies the Interface is now functional.

The experiment with the simulated Braitenbergvehikel resulted in proofing that Minecraft is usable as a simulation environment.

5.1 What's next

In the future, multiple agents shall interact with the same environment and collaborate with each other. ... what has been learned ... ... what can be done with the new environment ... ... what can be improved? ... ... what other simulation environments could be of interest? ...

Appendix A

Auszug aus dem Buch

Beispielhaft wird hier gezeigt, wie Block-Zitate eingeführt werden können. Hier der Beginn des Buchs "'Per Anhalter durch die Galaxis"' von Douglas Adams [Ada98]:

> "'Das Haus stand auf einer kleinen Anhöhe genau am Rand des Ortes. Es stand alleine da und überblickte das weite Ackerland im Westen. Absolut kein bemerkenswertes Haus - es war ungefährt dreißig Jahre alt, plump, viereckig, aus Ziegelsteinen erbaut und hatte vier Fenster an der Vorderseite, der es nach Größe und Proportion mehr oder weniger mißlang, das Auge zu erfreuen."'

Appendix B

Implementierungen

Beispielhaft wird hier gezeigt, wie Code-Beispiele in den Text eingefügt werden können. Die Pseudocode-Umgebung wird von *macros.tex* bereitgestellt und kann dort entsprechend angepasst werden.

```
1 public static Object answeringMachine() {
2    Thread.sleep(1000);
3    return 42;
4 }
```

Figure B.1: *Implementierung einer Maschine zur Beantwortung der Fragen aller Fragen.*

List of Figures

3.1 The MicroPsi 2 architecture [Bac12] 6
3.2 A "grass" Minecraft Block 8
3.3 A chunk 8
3.4 "the player" (CC-BY-3.0 Mojang AB) [ima13] 8
3.5 Minecraft Basic Mechanics (CC-BY-3.0 Mojang AB) [ima13] 9
3.6 data structure for block types 12
3.7 data structure for packet types 12
3.8 data structure for the packets structures 13
3.9 function for decoding packets 13

4.1 Sending a packet that moves the agent on block in x direction 16
4.2 The MicroPsi world-adapter spins up and steps a spock bot 17
4.3 The MicroPsi world-adapter spins up and steps a spock bot 17
4.4 The new architecture of MicroPsi with the Minecraft interface. New modules are framed orange. 18

B.1 Implementierung einer Maschine zur Beantwortung der Fragen aller Fragen. 25

List of Tables

3.1 Structure of packet Player Position (0x0B) [pro13] 10

Inhalt der beigelegten CD

Die beigelegte CD enthält folgenden Inhalt:

- diese Masterarbeit in PDF Format,
- Videos mit Interview von Fans,
- den Source-Code der Implementierung einer Maschine zur Beantwortung der Fragen aller Fragen. Der Source-Code ist im Ordner *src* zu finden.

Bibliography

[Ada98] In: Adams, Douglas: *Per Anhalter durch die Galaxis: Roman*. Heyne Verlag, 1998, S. 1

[Bac09] Bach, Joscha: *Principles of Synthetic Intelligence PSI: An Architecture of Motivated Cognition*. 1st. New York, NY, USA : Oxford University Press, Inc., 2009. – ISBN 0195370678, 9780195370676

[Bac12] Bach, Joscha: MicroPsi 2: The Next Generation of the MicroPsi Framework. In: Bach, Joscha (Hrsg.) ; Goertzel, Ben (Hrsg.) ; Iklé, Matthew (Hrsg.): *AGI* Bd. 7716, Springer, 2012 (Lecture Notes in Computer Science). – ISBN 978–3–642–35505–9, 11-20

[fog13a] `http://www.reddit.com/r/programming/comments/1b8a6z/simple_minecraft_clone_in_580_lines_of_python/`

[fog13b] `https://news.ycombinator.com/item?id=5458986`

[git13a] `https://github.com/superjoe30/mineflayer`

[git13b] `https://github.com/fogleman/Minecraft`

[ima13] `http://www.minecraftwiki.net/wiki/File:Mob1.png`

[mcw13] `http://www.minecraftwiki.net/wiki/Chunks`

[pro13] `http://wiki.vg/Protocol`

[sky13] `http://skycraft.io/`