Contents

1 Software Design **4**
1.1 General design aims . . . 4
1.1.1 Naming conventions . . . 4
1.2 Component Diagramms . . . 5
1.2.1 Management Client . . . 6
1.2.2 Controlling Master . . . 6
1.2.3 Working Slave . . . 6
1.3 Class diagrams . . . 7
1.3.1 Agent model . . . 7
1.3.2 Management Client . . . 8
1.3.3 Controlling Master . . . 8
1.3.4 Working Slave . . . 8
1.3.5 World modeling . . . 8
1.4 Sequence diagrams . . . 12
1.4.1 Agent training and result evaluation - overview . . . 12

2 Implementation and prototype testing **14**
2.1 Implementation . . . 14
2.1.1 JESICA . . . 14
2.2 First Run . . . 14
2.3 Further experiments with random agent movement . . . 18
2.3.1 Experiment: more Agents . . . 21
2.3.2 Experiment: smaller step size . . . 22
2.4 Experiments with centroid agent movement . . . 23
2.4.1 Experiment: more Agents, smaller step size . . . 31
2.4.2 Experiment: smaller gradient threshold . . . 36
2.5 Experimentation Runs with changed NN topology . . . 40
2.5.1 NN testing with XOR . . . 40
2.5.2 NN testing with an agent's bag . . . 41
2.5.3 First Run with new NN-topology . . . 41

List of Figures

1.1 Component diagram: System Components 5
1.2 Class diagram: Agent Model 7
1.3 Class diagram: Manager Model 8
1.4 Class diagram: Master Model 9
1.5 Class diagram: Slave Model 10
1.6 Class diagram: World Model 11
1.7 Class diagram: Concrete implementation of a world 12
1.8 Sequence diagram: Agent training and result evaluation - overview 13

2.1 First Run: Input pattern 16
2.2 First Run: Result 16
2.3 First Run: Way of first Agent 17
2.4 First Run: Way of second Agent 17
2.5 First Run: Way of third Agent 17
2.6 First Run: Way of fourth Agent 18
2.7 Random move: Input pattern 19
2.8 Random move, Run 1: Output pattern 19
2.9 Random move, Run 1, Agent 1: Way MSE 20
2.10 Random move, Run 1, Agent 2: Way MSE 20
2.11 Random move, Run 1, Agent 3: Way MSE 21
2.12 Random move, Run 2: Output pattern 22
2.13 Random move, Run 3: Output pattern 23
2.14 Random move, Run 3, Agent 1: Way MSE 23
2.15 Random move, Run 3, Agent 1: Way 24
2.16 Random move, Run 3, Agent 2: Way MSE 24
2.17 Random move, Run 3, Agent 2: Way 25
2.18 Random move, Run 3, Agent 3: Way MSE 25
2.19 Random move, Run 3, Agent 3: Way 26
2.20 Centroid move: Input pattern 27
2.21 Centroid move, Run 1: Output pattern 27
2.22 Centroid move, Run 1, Agent 1: Way MSE 28
2.23 Centroid move, Run 1, Agent 2: Way MSE 28
2.24 Centroid move, Run 1, Agent 3: Way MSE 29
2.25 Centroid move, Run 2: Output pattern 29
2.26 Centroid move, Run 3: Output pattern 30
2.27 Centroid move, Run 4: Output pattern 32
2.28 Centroid move, Run 4, Agent 1: Way MSE 32
2.29 Centroid move, Run 4, Agent 1: Way 33

2.30 Centroid move, Run 4, Agent 2: Way MSE 33
2.31 Centroid move, Run 4, Agent 2: Way 34
2.32 Centroid move, Run 4, Agent 3: Way MSE 34
2.33 Centroid move, Run 4, Agent 3: Way 35
2.34 Centroid move, Run 5: Output pattern 35
2.35 Centroid move, Run 6: Output pattern 37
2.36 Centroid move, Run 6, Agent 1: Way MSE 37
2.37 Centroid move, Run 6, Agent 1: Way 38
2.38 Centroid move, Run 6, Agent 2: Way MSE 38
2.39 Centroid move, Run 6, Agent 2: Way 39
2.40 Centroid move, Run 6, Agent 3: Way MSE 39
2.41 Centroid move, Run 6, Agent 3: Way 40
2.42 Centroid move, Run 7: Output pattern 40
2.43 Centroid move, Run 1, new topology 42
2.44 Centroid move, Run 1, new topology, Way: Agent 1 42
2.45 Centroid move, Run 1, new topology, Way: Agent 4 43
2.46 Centroid move, Run 1, new topology, MSE: Agent 1 43
2.47 Centroid move, Run 1, new topology, MSE: Agent 4 44

List of Tables

Chapter 1

Software Design

Literature:

- General software analysis and design: [12, pages 741-822]
- UML: [13]
- Software pattern (Design pattern): [14], [15], [4]

1.1 General design aims

When designing our software to meet the requirements as described in the section for system architecture (Section **??**), we had to take care of several boundary conditions:

- Flexible implementation of agent movement control
- Use a NN-library but in a replaceable way
- Easy change of world representation
- Concentrate the agent creation process in a central module
- Information interchange between different system components should be based on calls and callbacks (avoid polling over process boundaries)
- Multi threaded agent training
- User interface should be separated from the agent controller
- Agent state extracted from implementation to reduce network traffic

1.1.1 Naming conventions

As in every project where more than one developer is working on, SatMAS needs naming conventions, too. We designed our style guide to be not too strength (because most programmers don't like to be told how to do their work). Although naming should be done by general good software engineering practices, the most common naming conventions are given here.

Interface Names

In the SatMAS design all interface classes have the prefix *I*. Example: *IWorld*.

Property methods

In addition to the classes provided by the JDK all property methods have set-, get-, is-, or update prefix followed by the name of the controlled property. (e. g. getName or setParent).

Variables

Our variable naming convention is nearly the same as the famous *hungarian notation* in the Windows programming world. There, all variables have a lower prefix with their corresponding type, which can be one of the following:

integer: Prefix: *n* (e. g. nIndex)

float: Prefix: *f* (e. g. fAverage)

String: Prefix: *str* (e. g. strFilePath)

boolean: Prefix: *b* (e. g. bResult)

If the type is none of these, you should give the variable a really speaking name. For member variable it is mandatory to put the prefix m_ to it (e. g. m_nArrayCounter). *NOTE:* This naming conventions are not mandatory for loop variables.

1.2 Component Diagramms

Components of the system and their responsibilities are shown here:

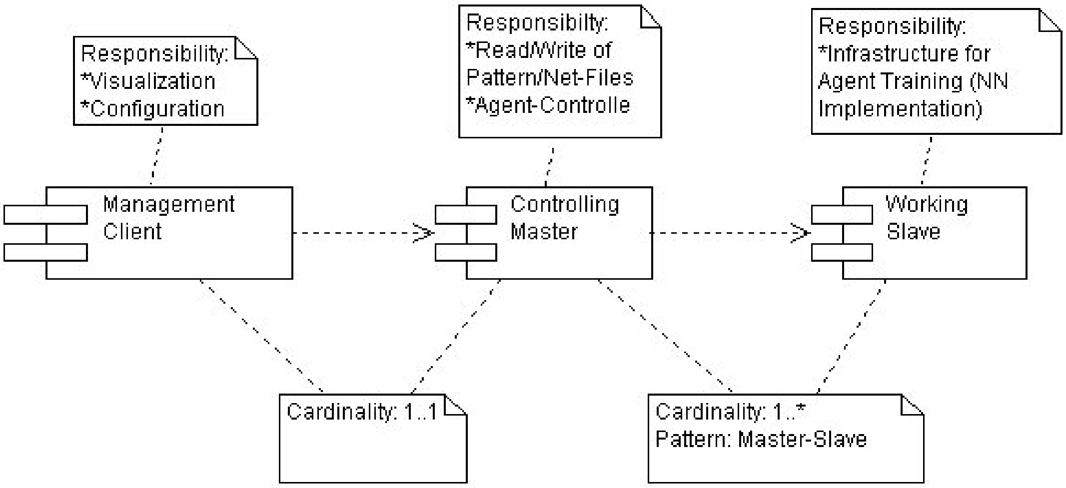

Figure 1.1: Component diagram: System Components

1.2.1 Management Client

The Management Client is the user interface of SatMAS.

Responsibility

- Visualization
- Configuration

Collaborators

- Controlling Master

1.2.2 Controlling Master

The Controlling Master is SatMAS's central intelligence component. Cooperation between Controlling Master and Working Slave is implemented as Master-Slave design pattern as described in [4, pages 245-291].

Responsibility

- Agent controller
- Disposition of agents
- Query results
- Creation of agents
- Input/Output of pattern/net files
- Notify client

Collaborators

- Management Client
- Working Slave

1.2.3 Working Slave

All time consuming operations of SatMAS are processed on a Working Slave.

Responsibility

- Provide infrastructure for agent training
- Notify master after training of an agent has finished

Collaborators

- Controlling Master

1.3 Class diagrams

1.3.1 Agent model

In order to give the Controlling Master the ability to build agents with different behaviors, the agents are modeled as seen in figure 1.2.

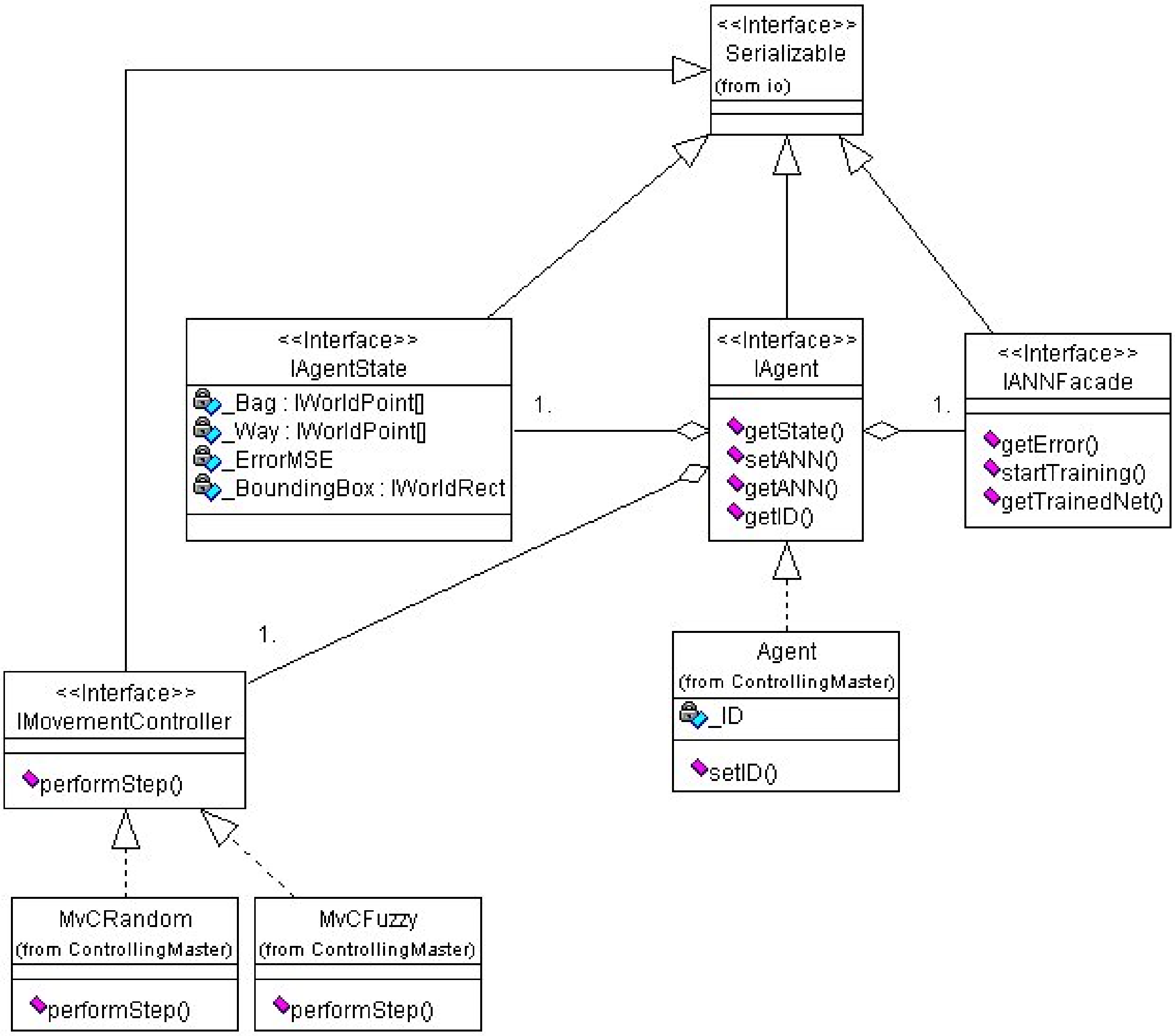

Figure 1.2: Class diagram: Agent Model

Special design decisions:

IMovementController This interface enables flexible replacement of movement behavior of an agent

IAgentState Agent state extracted from implementation to reduce network

1.3.2 Management Client

A smart and simple console interface is sufficient for this design stage (diagram 1.3).

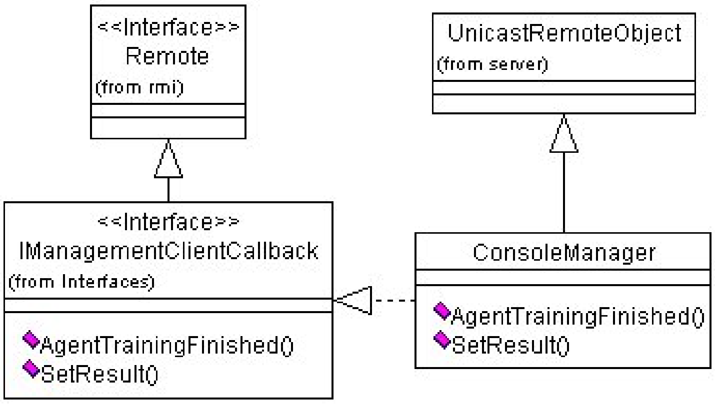

Figure 1.3: Class diagram: Manager Model

1.3.3 Controlling Master

Class modeling of the central intelligence component is in figure 1.4.

Special design decision:

IAgentFactory Separate modeling of an agent creation interface leads into a looser coupling between the agent controller and the working slave. Additionally variation of agent components is facilitated.

1.3.4 Working Slave

Figure 1.5 is a working slave's class diagram.

Special design decisions:

IANNFacade To minimize dependency on a distinct ANN-package, an abstraction level is introduced. Realized with the Facade design pattern from [15, pages 189-198].

JFroehlichANNFacade Due to lack of free available ANN-package Jörg Fröhlich wrote his own implementation, which fitted best for SatMAS at first glance.

1.3.5 World modeling

As discussed in section **??**, best selection of a concrete world model needs to be researched. Therefore an abstract model for worlds and their data is needed, as seen in diagram 1.6.

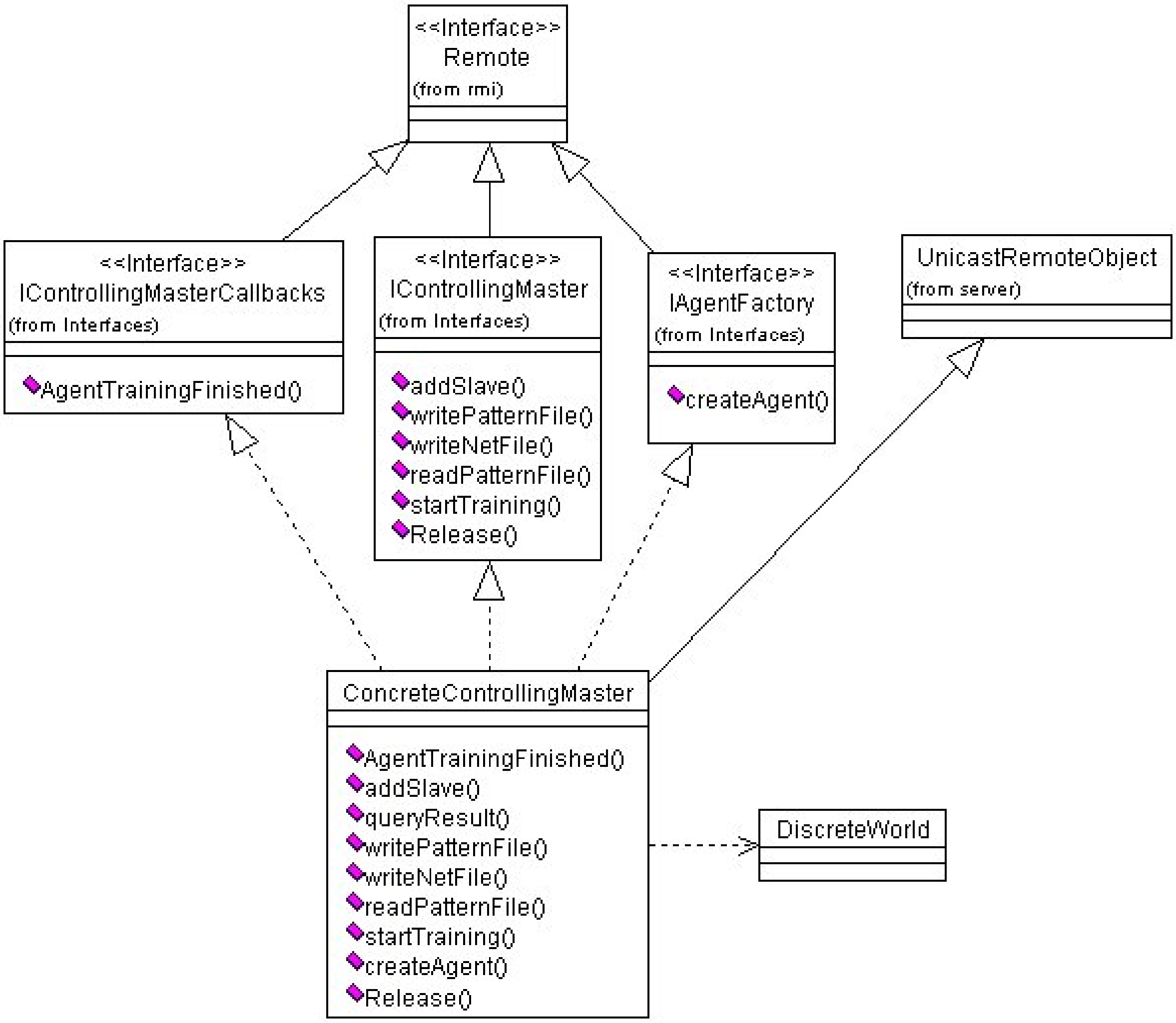

Figure 1.4: Class diagram: Master Model

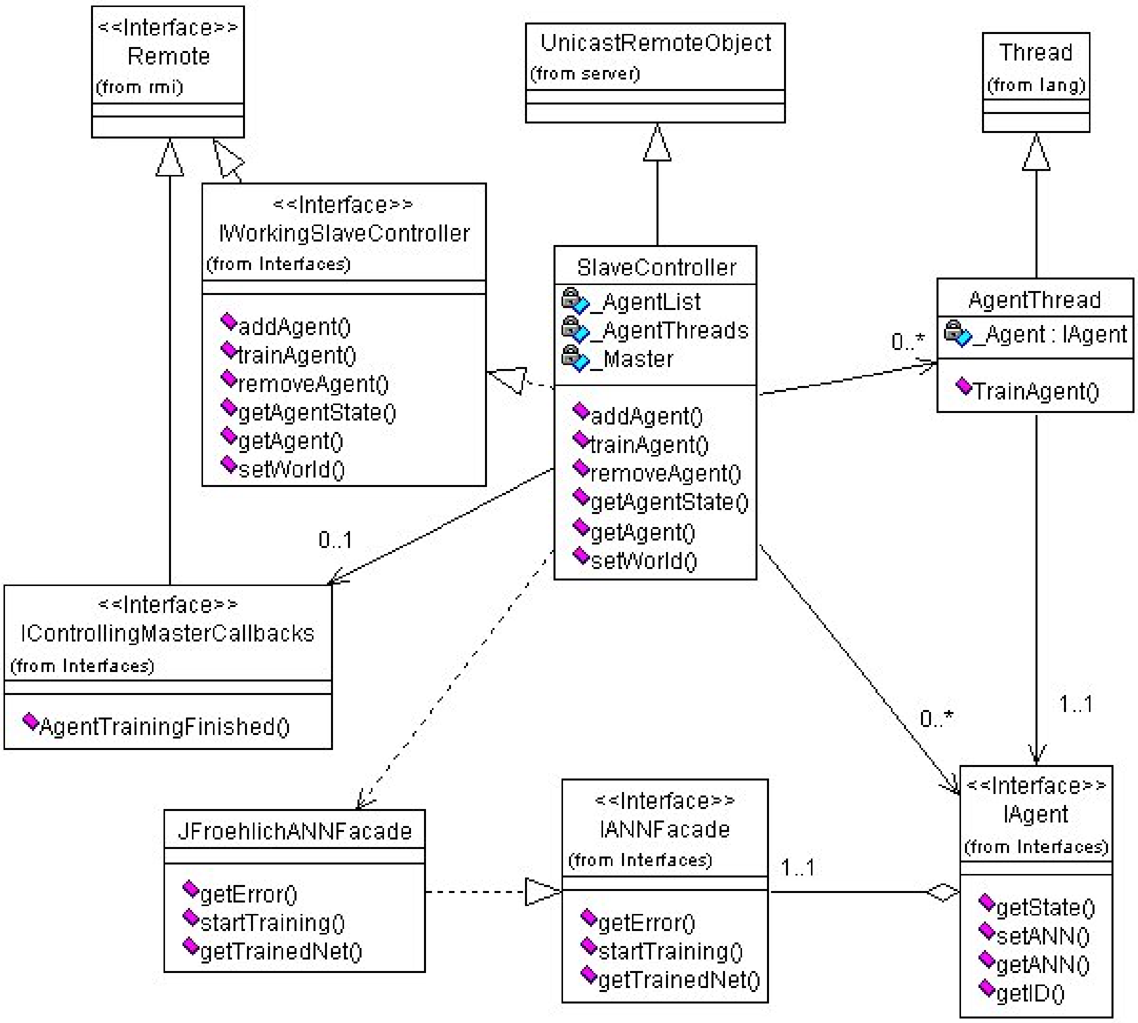

Figure 1.5: Class diagram: Slave Model

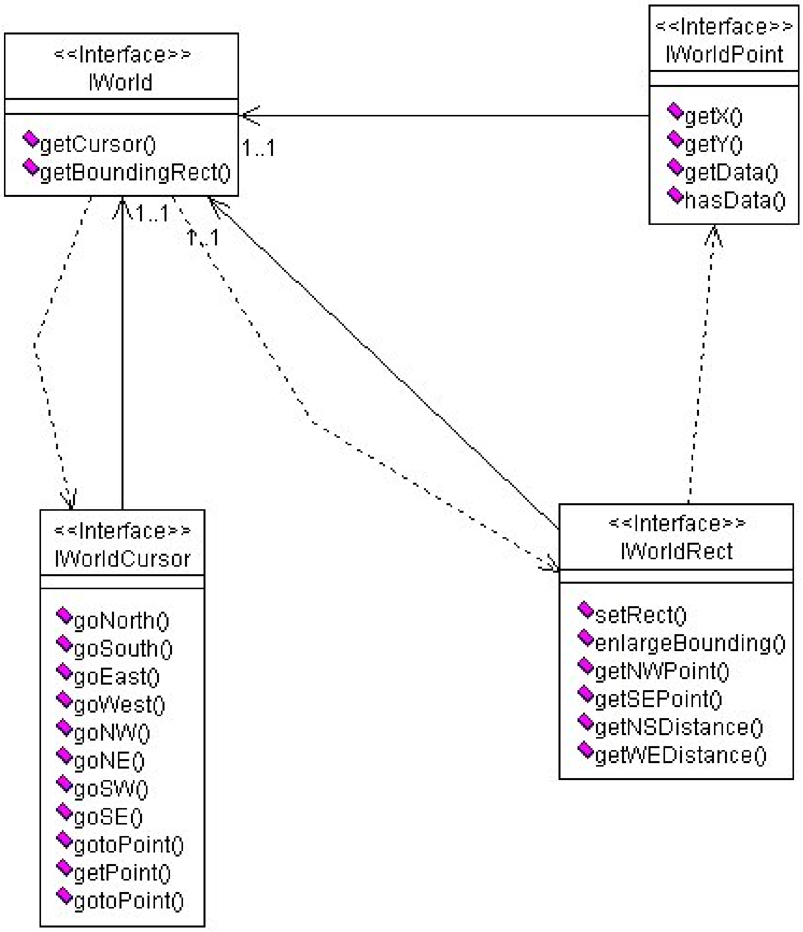

Figure 1.6: Class diagram: World Model

Special design decisions:

IWorld Abstract description of a real world model. After creation a world model is immutable.

IWorldPoint Represents a concrete position on a world model, including data access methods.

IWorldRect A rectangular region on a world model.

IWorldCursor Mobile position marker on a world model.

Concrete implementation of a world

This implementation example of a discrete world models a plane 2D area, on which each raster point is reachable.

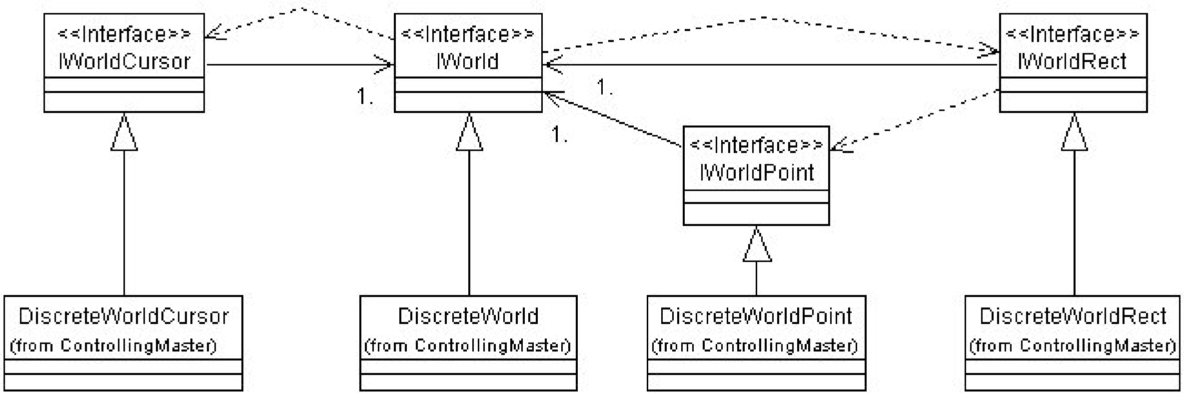

Figure 1.7: Class diagram: Concrete implementation of a world

1.4 Sequence diagrams

1.4.1 Agent training and result evaluation - overview

Diagram 1.8 gives a rough overview how agents are trained, and after finishing their learning phase the whole result is queried.

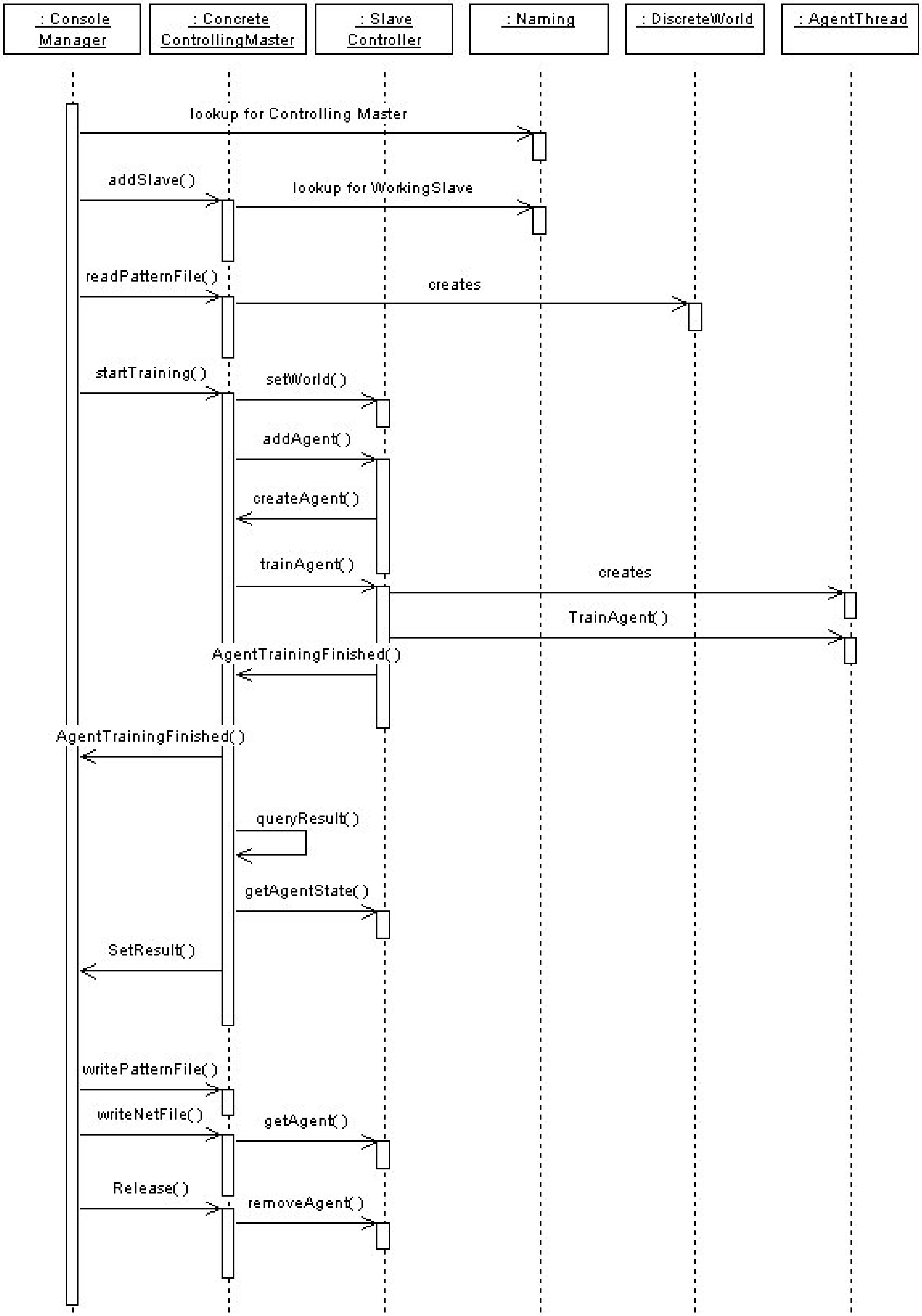

Figure 1.8: Sequence diagram: Agent training and result evaluation - overview

Chapter 2

Implementation and prototype testing

2.1 Implementation

Programming language: Java

JDK: only tested on 1.3

Neural Network package: jaNet

Pattern file library: NetJen

Unit test framework: JUnit

2.1.1 JESICA

A fancy name gives an excellent product the right touch. So we also had to find a agreeable-sounding name for this great java project. After weeks of hard, mind exhausting work, **JESICA** was born. JESICA is the short form of
J*ava* **E***xperimentation* **S***ytem for* **I***mage* **C***lassification with* **A***gents* .

2.2 First Run

Here comes the first pictures of JESICA's first working prototype.

Run parameter:

Number of Agents: 4

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 1000

NN learning rate: 0.5

NN topology 2 - 4 - 1

Gating network for classification Agent with nearest bounding rect

Run 1:

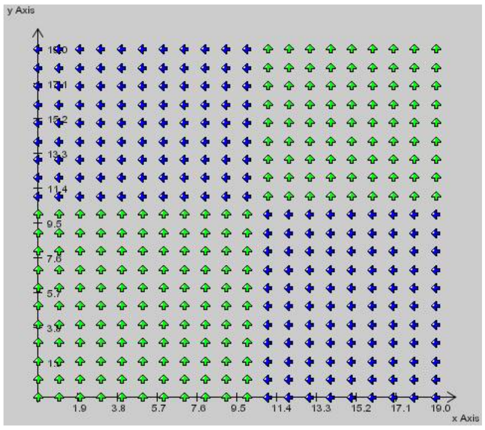

Figure 2.1: First Run: Input pattern

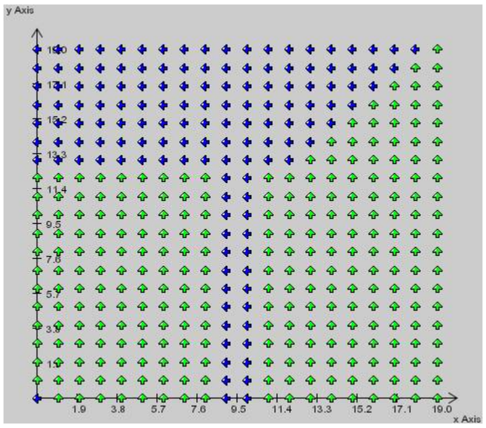

Figure 2.2: First Run: Result

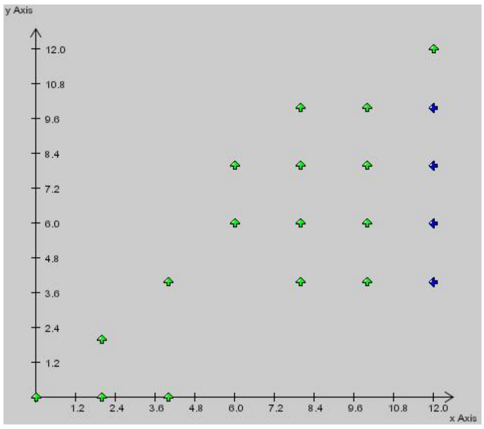

Figure 2.3: First Run: Way of first Agent

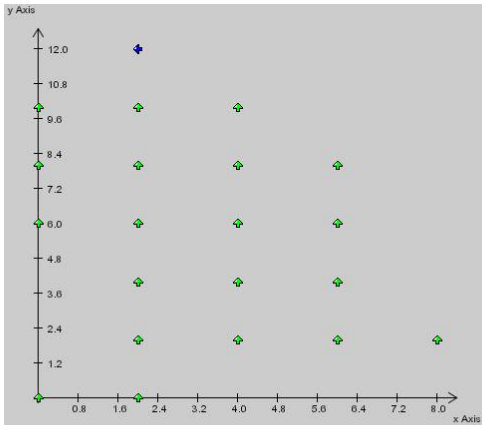

Figure 2.4: First Run: Way of second Agent

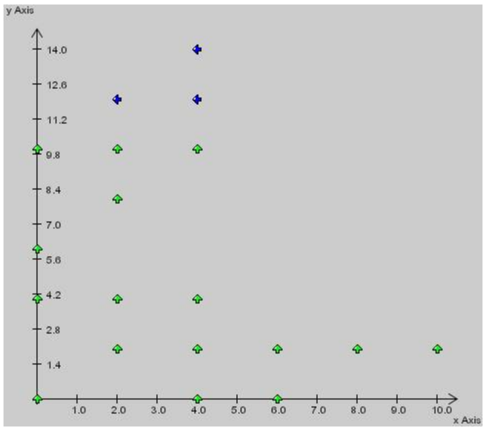

Figure 2.5: First Run: Way of third Agent

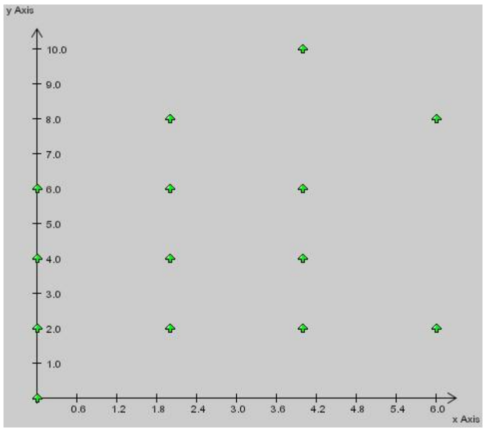

Figure 2.6: First Run: Way of fourth Agent

2.3 Further experiments with random agent movement

This agent's movement controller works on a simple, random based strategy. The individual movement steps will be decided by a randomizer, with the only constraint that no points are allowed that are already in the agent's bag.

Run parameter:

Number of Agents: 8

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random, no points that are already in bag

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

NN topology 2 - 4 - 1

Gating network for classification Agent with nearest bounding rect

Run 1:

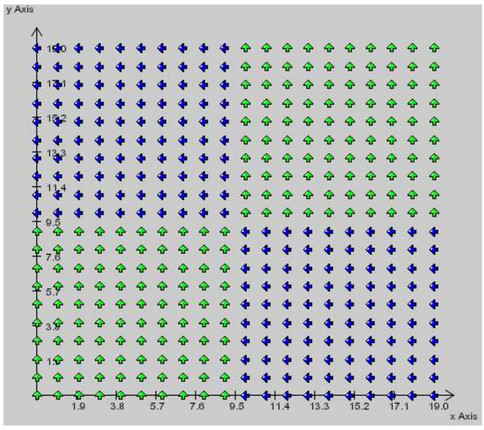

Figure 2.7: Random move: Input pattern

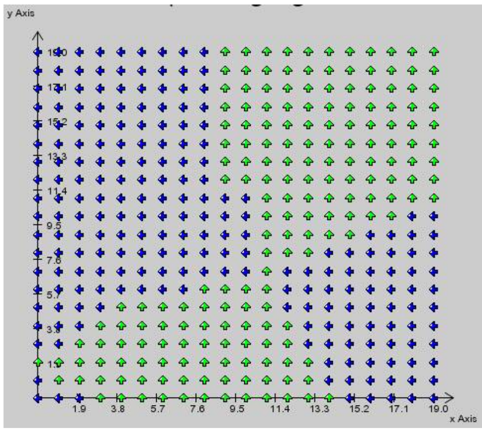

Figure 2.8: Random move, Run 1: Output pattern

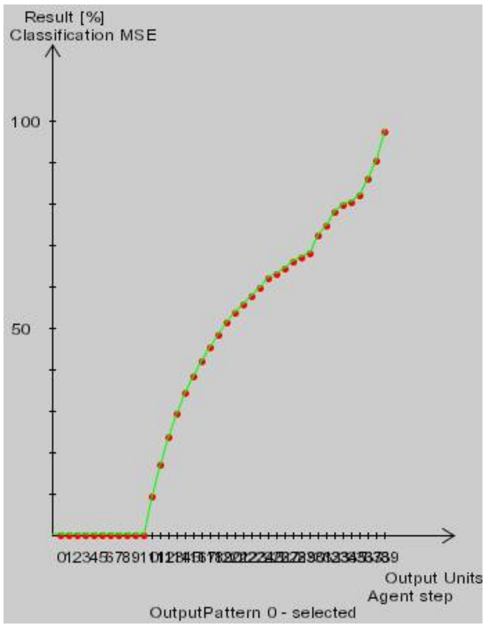

Figure 2.9: Random move, Run 1, Agent 1: Way MSE

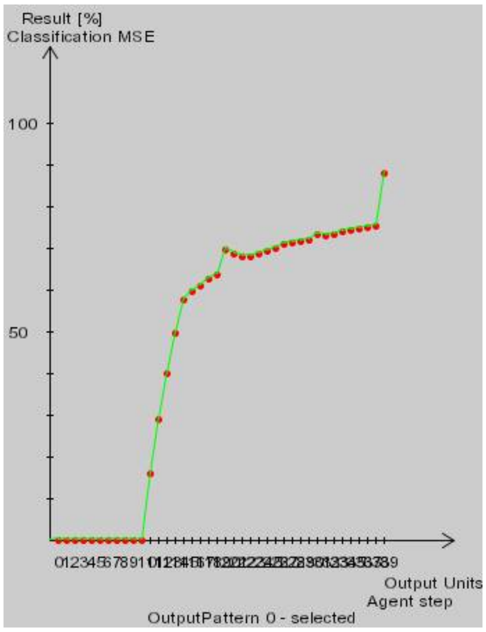

Figure 2.10: Random move, Run 1, Agent 2: Way MSE

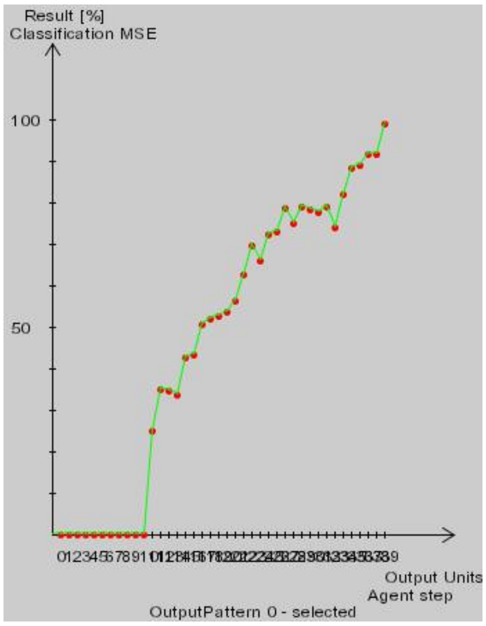

Figure 2.11: Random move, Run 1, Agent 3: Way MSE

2.3.1 Experiment: more Agents

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random, no points that are already in bag

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

NN topology 2 - 4 - 1

Gating network for classification Agent with nearest bounding rect

Run 2:

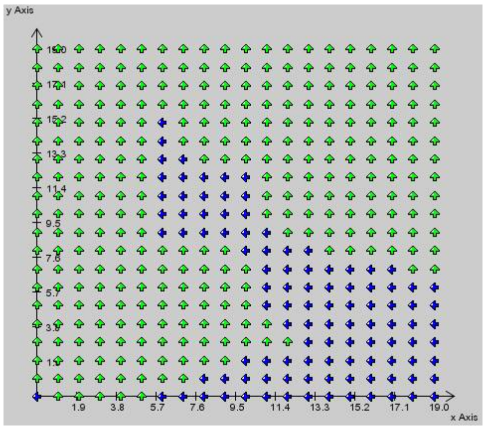

Figure 2.12: Random move, Run 2: Output pattern

2.3.2 Experiment: smaller step size

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 1

Agent movement strategy: random, no points that are already in bag

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

NN topology 2 - 4 - 1

Gating network for classification Agent with nearest bounding rect

Run 3:

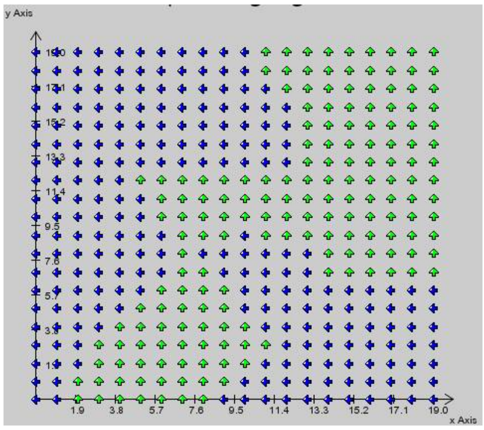

Figure 2.13: Random move, Run 3: Output pattern

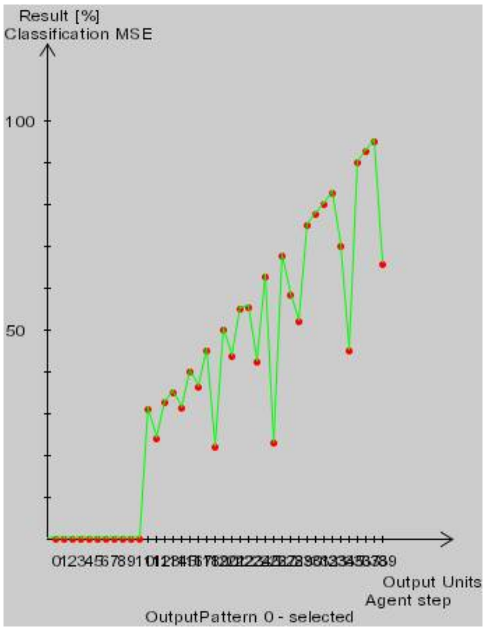

Figure 2.14: Random move, Run 3, Agent 1: Way MSE

2.4 Experiments with centroid agent movement

This agent implementation uses two different strategies for movement:

- If the gradient (calculated over a specified period) of the way MSE is less or equal null, random movement is used.
- If the gradient is positive, a centroid movement strategy will be used. Here the vectors of all point in the agent's bag will be calculated, and the resulting direction

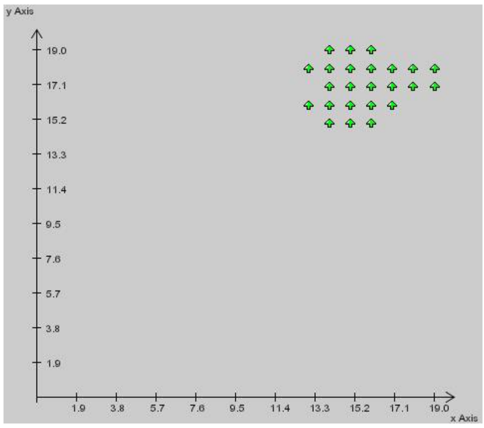

Figure 2.15: Random move, Run 3, Agent 1: Way

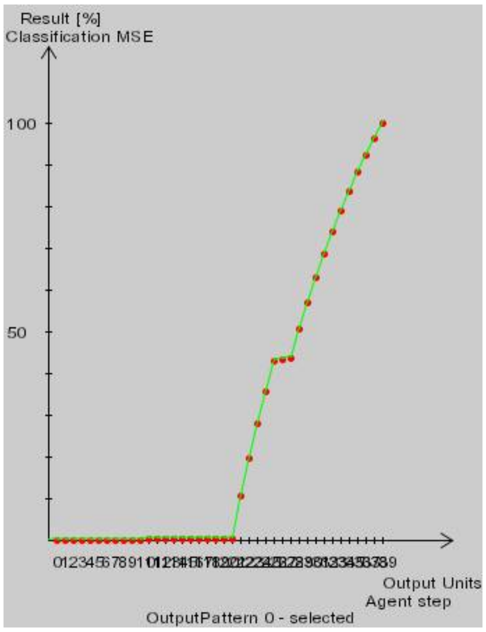

Figure 2.16: Random move, Run 3, Agent 2: Way MSE

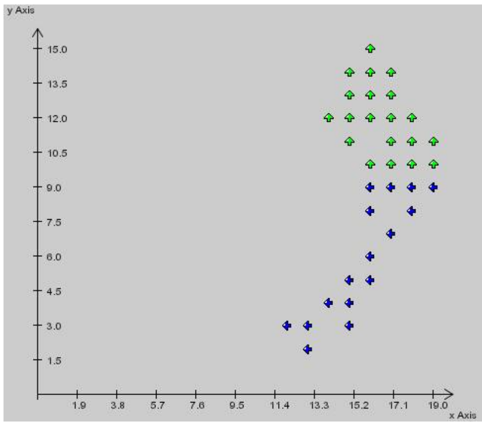

Figure 2.17: Random move, Run 3, Agent 2: Way

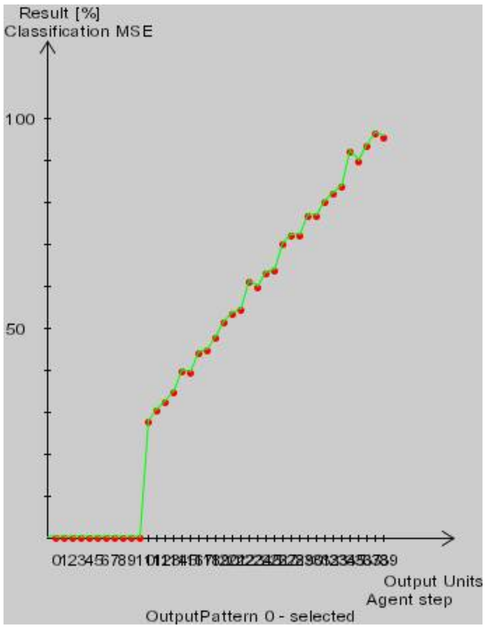

Figure 2.18: Random move, Run 3, Agent 3: Way MSE

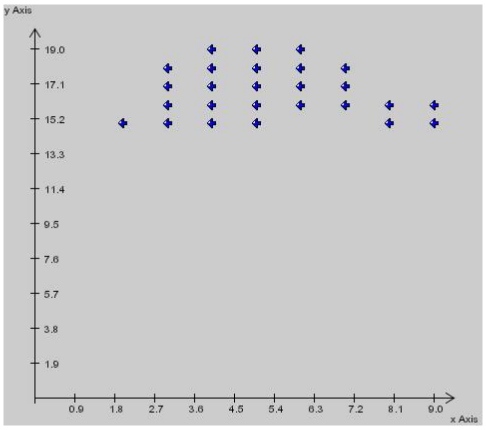

Figure 2.19: Random move, Run 3, Agent 3: Way

will be used as parameter for a "roulette-wheel-random-number generator". This generator is realized by using a modified Gaussian probability spreading.

Run parameter:

Number of Agents: 8

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 2

Agent movement strategy: random, centroid based error gradient positive

Gradient calculation threshold: 5

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

NN topology 2 - 4 - 1

Gating network for classification Agent with nearest bounding rect

Run 1

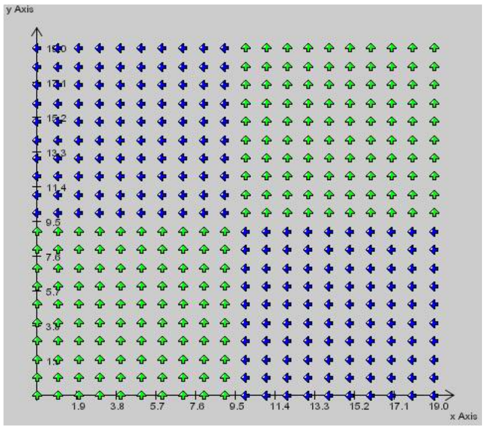

Figure 2.20: Centroid move: Input pattern

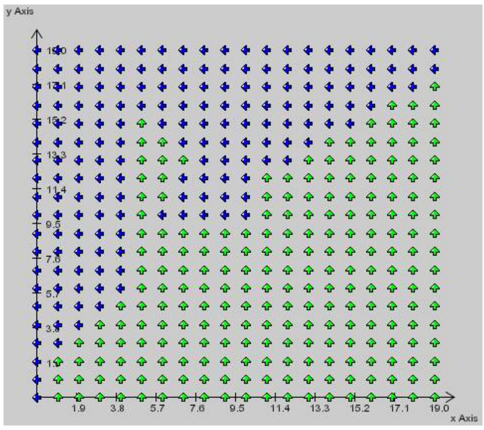

Figure 2.21: Centroid move, Run 1: Output pattern

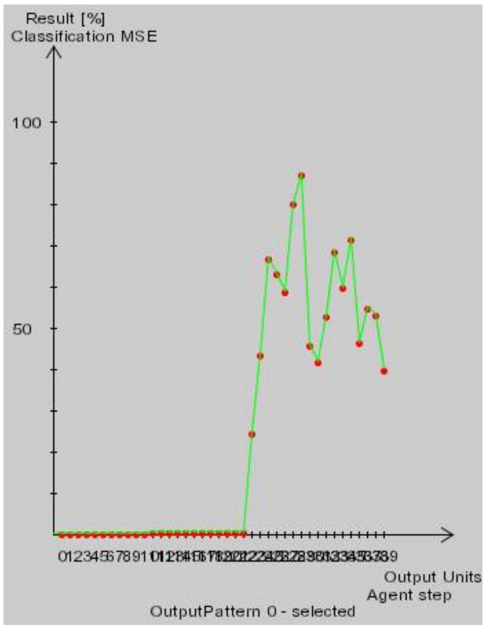

Figure 2.22: Centroid move, Run 1, Agent 1: Way MSE

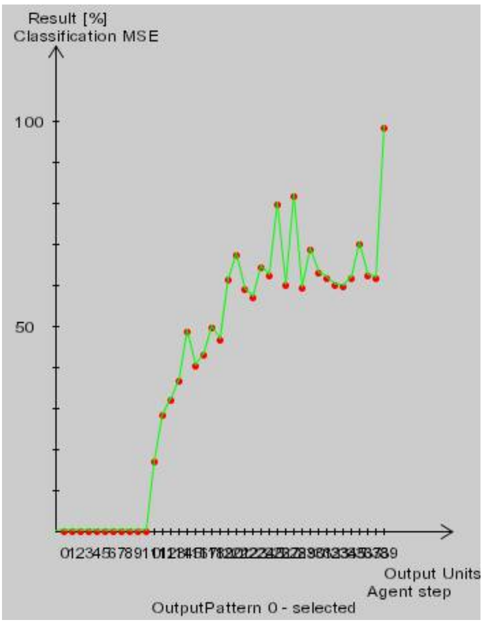

Figure 2.23: Centroid move, Run 1, Agent 2: Way MSE

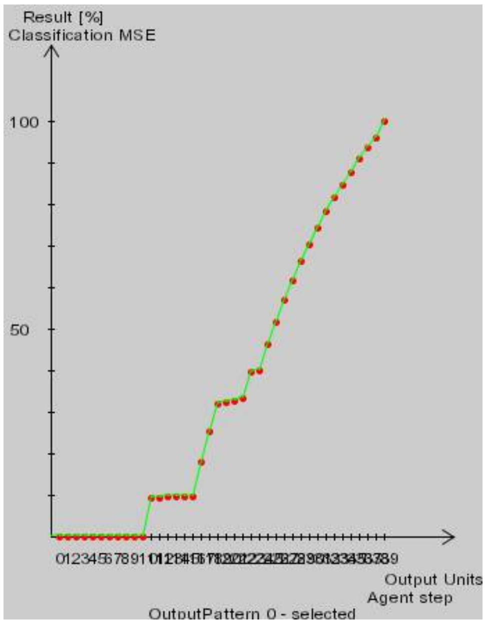

Figure 2.24: Centroid move, Run 1, Agent 3: Way MSE

Run 2

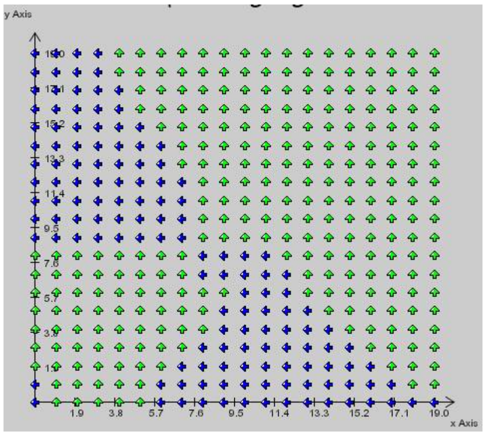

Figure 2.25: Centroid move, Run 2: Output pattern

Run 3

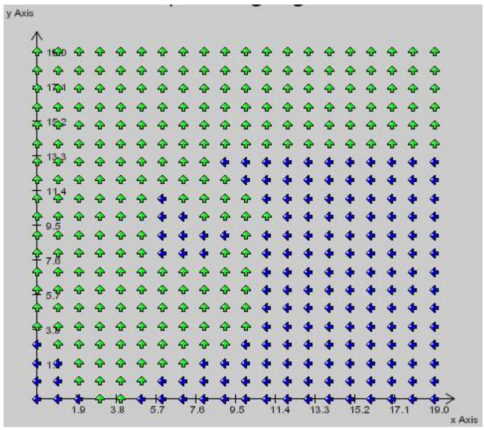

Figure 2.26: Centroid move, Run 3: Output pattern

2.4.1 Experiment: more Agents, smaller step size

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 1

Agent movement strategy: random, centroid based error gradient positive

Gradient calculation threshold: 5

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

NN topology 2 - 4 - 1

Gating network for classification Agent with nearest bounding rect

Run 4

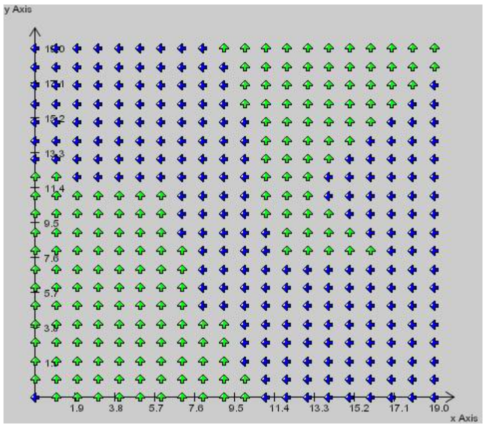

Figure 2.27: Centroid move, Run 4: Output pattern

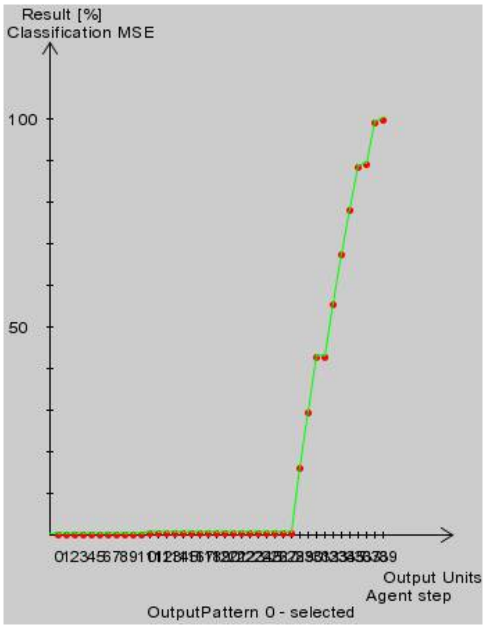

Figure 2.28: Centroid move, Run 4, Agent 1: Way MSE

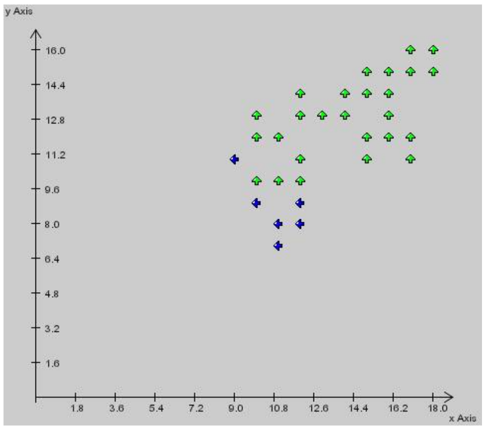

Figure 2.29: Centroid move, Run 4, Agent 1: Way

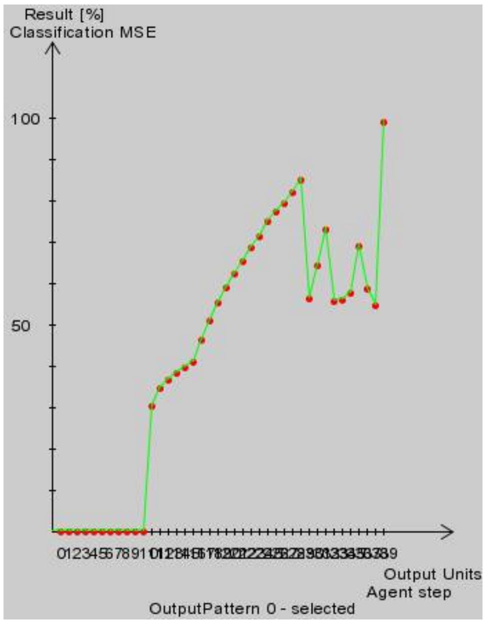

Figure 2.30: Centroid move, Run 4, Agent 2: Way MSE

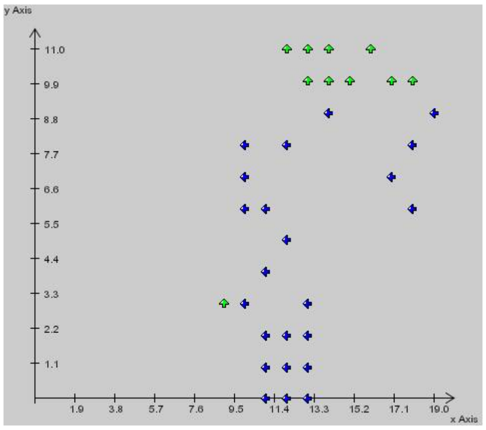

Figure 2.31: Centroid move, Run 4, Agent 2: Way

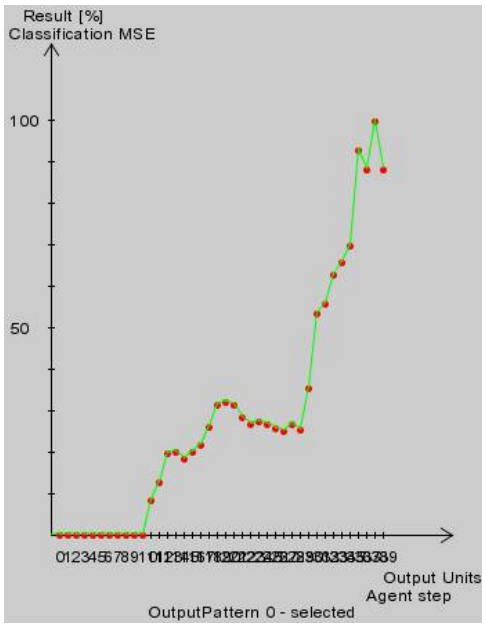

Figure 2.32: Centroid move, Run 4, Agent 3: Way MSE

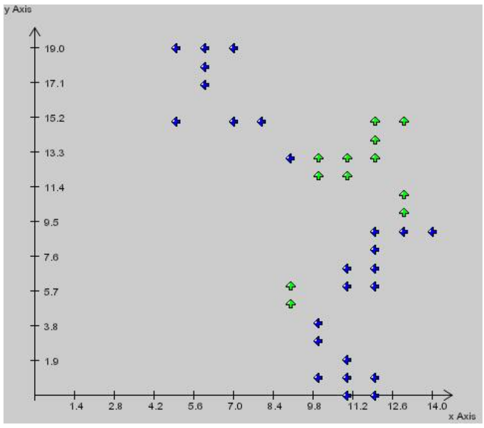

Figure 2.33: Centroid move, Run 4, Agent 3: Way

Run 5

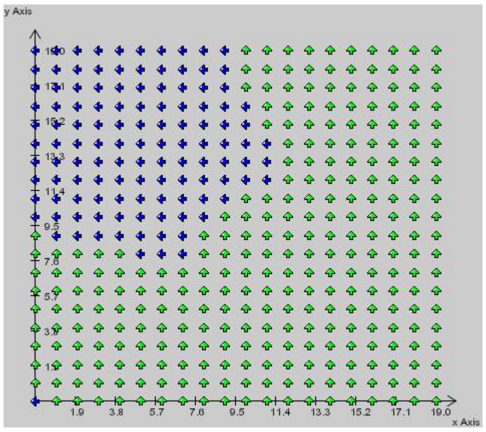

Figure 2.34: Centroid move, Run 5: Output pattern

2.4.2 Experiment: smaller gradient threshold

Run parameter:

Number of Agents: 16

Input pattern size: 400 (20x20)

Input pattern type: XOR

Agent way length: 30

Agent movement step size: 1

Agent movement strategy: random, centroid based error gradient positive

Gradient calculation threshold: 3

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

NN topology 2 - 4 - 1

Gating network for classification Agent with nearest bounding rect

Run 6

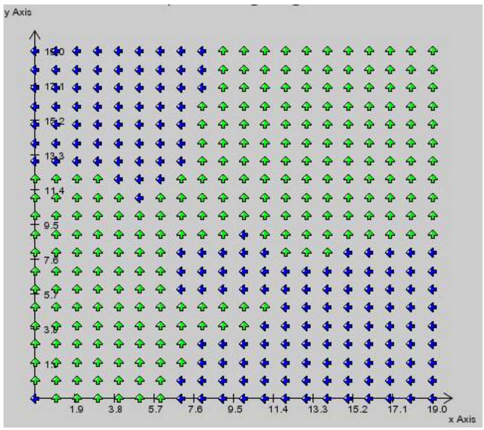

Figure 2.35: Centroid move, Run 6: Output pattern

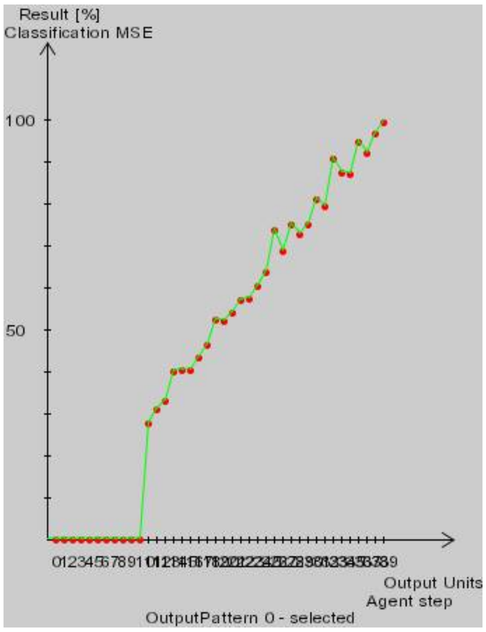

Figure 2.36: Centroid move, Run 6, Agent 1: Way MSE

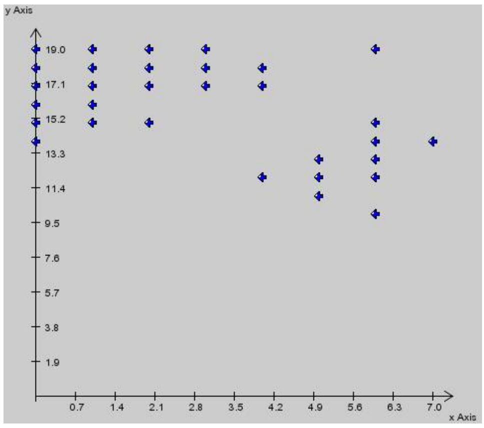

Figure 2.37: Centroid move, Run 6, Agent 1: Way

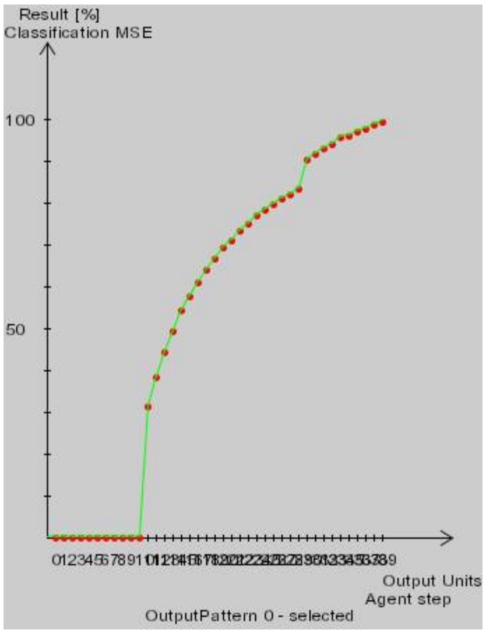

Figure 2.38: Centroid move, Run 6, Agent 2: Way MSE

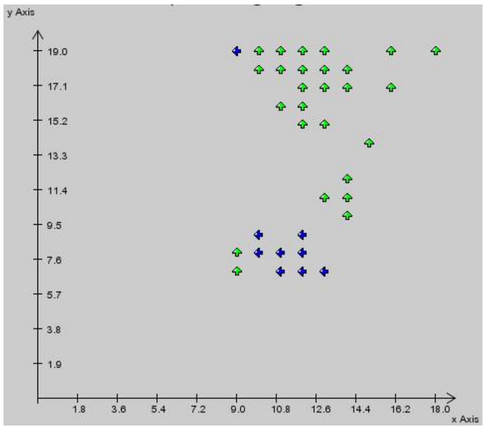

Figure 2.39: Centroid move, Run 6, Agent 2: Way

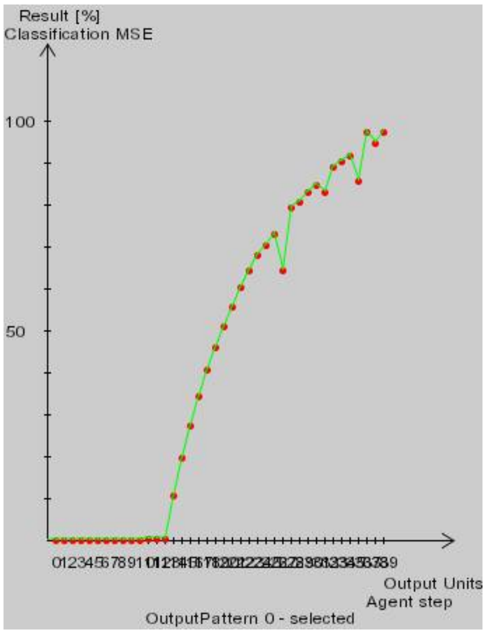

Figure 2.40: Centroid move, Run 6, Agent 3: Way MSE

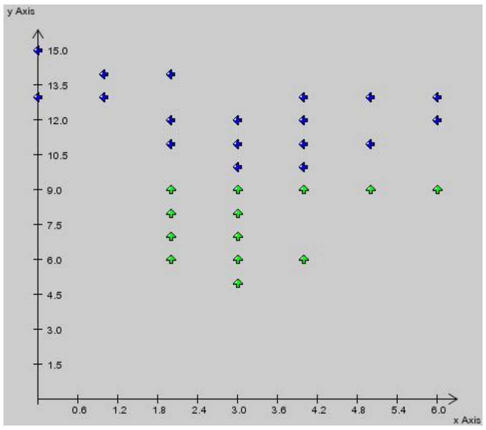

Figure 2.41: Centroid move, Run 6, Agent 3: Way

Run 7

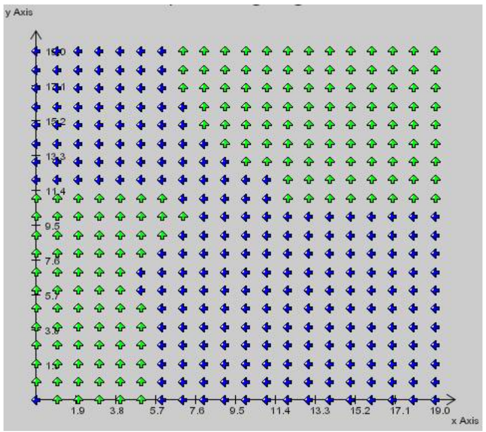

Figure 2.42: Centroid move, Run 7: Output pattern

2.5 Experimentation Runs with changed NN topology

After observation of previous results, agent ways and MSE curves, we came to the conclusion that something with the current used NN's must be wrong.

2.5.1 NN testing with XOR

Our first attempt to find the problem was by learning the net with a simple XOR problem. We used the previous NN parameters and started training of the net.

NN minimum error: 0.001

NN training cycles: 3000

NN learning rate: 0.1

NN topology 2 - 4 - 1

A few runs later with this parameters, it was clear that the NN wasn't able to learn the XOR problem, due to insufficient training cycles. After increasing the value for the learning rate, propagation of input lead to the corresponding output :-)

2.5.2 NN testing with an agent's bag

The next step in our progress to success, was by solving another problem. So we decided to use an agents bag from previous runs, with the same NN-parameters as above in the XOR-problem. After a few experiments with increased training cycles and modified learning rate, it was obvious that with the current network topology the NN didn't stand a chance to learn the bag properly.

Topology was the only parameter of the NN we haven't changed yet, that's why we started experimenting changing hidden-layer-neurons, and increasing training cycles to 50.000. After about five runs with each of the following topologies we got these results:

2 - 10 - 1 not learnable, MSE: 1.0

2 - 20 - 1 not learnable, MSE: 1.0

2 - 10 - 10 - 1 40.000 cycles, MSE: 0.3

2 - 10 - 7 - 1 50.000 cycles, MSE: 0.4

2 - 10 - 5 - 1 30.000 cycles, MSE: 0.1

2 - 7 - 10 - 1 50.000 cycles , MSE: 0.4

2 - 5 - 10 - 1 not learnable, MSE: 1.0

Even when the topology 2 - 10 - 5 - 1 was sometimes not able to learn the whole problem in less than 50.000 training steps, it still had the lowest overall MSE and the lowest classification error while validation.

2.5.3 First Run with new NN-topology

Number of Agents: 4

Input pattern size: 100 (10x10)

Input pattern type: XOR

Agent way length: 10

Agent movement step size: 2

Agent movement strategy: random, centroid based error gradient positive

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.05

NN training cycles: 30000

NN learning rate: 0.3

NN topology 2 - 10 - 5 - 1

Gating network for classification Agent with nearest bounding rect

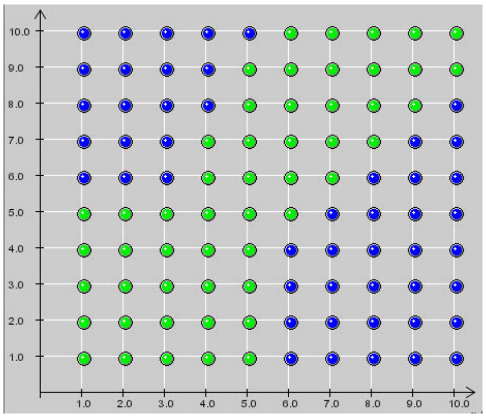

Figure 2.43: Centroid move, Run 1, new topology

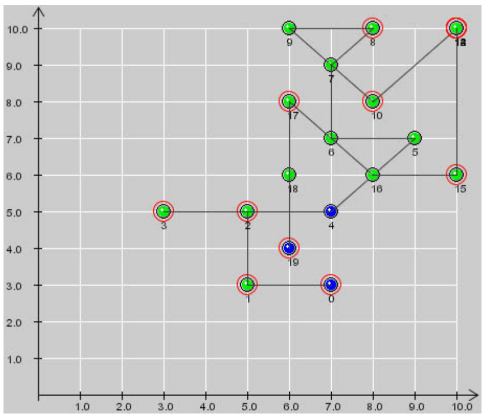

Figure 2.44: Centroid move, Run 1, new topology, Way: Agent 1

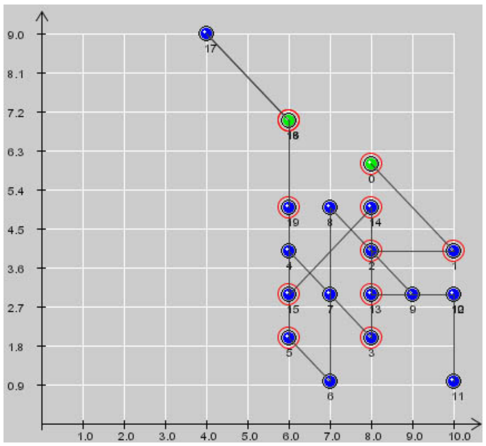

Figure 2.45: Centroid move, Run 1, new topology, Way: Agent 4

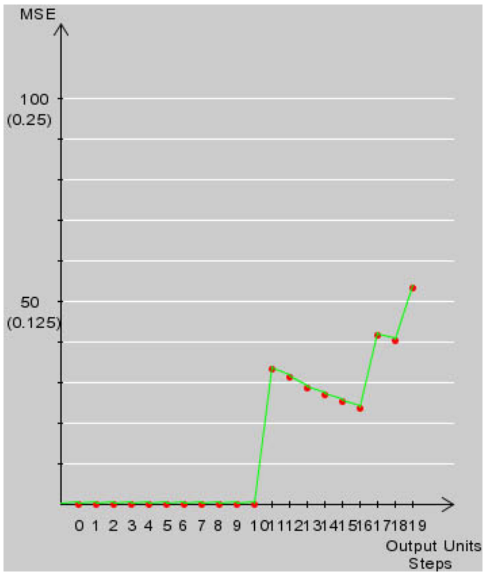

Figure 2.46: Centroid move, Run 1, new topology, MSE: Agent 1

2.6 Experimentation runs with a centroid gating network

2.6.1 Run 1

Number of Agents: 4

Input pattern size: 100 (10x10)

Input pattern type: XOR

Agent way length: 10

Agent movement step size: 2

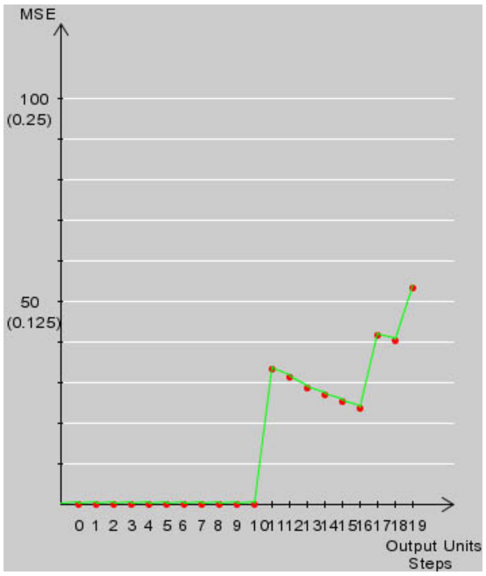

Figure 2.47: Centroid move, Run 1, new topology, MSE: Agent 4

Agent movement strategy: random, centroid based error gradient positive

Agent bag size: 10

NN validation pattern set: Agent's way

NN minimum error: 0.05

NN training cycles: 30000

NN learning rate: 0.3

NN topology 2 - 10 - 5 - 1

Gating network for classification Agent with nearest emphasis

Bibliography

[1] Doron Shalvi. An introduction to artificial neural networks, 1997.

[2] Perry Moerland. Classification using localized mixtures of experts. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN'99)*, volume 2, pages 838–843. London: IEE, 1999. (IDIAP-RR 98-14).

[3] Victor R. Lesser. Cooperative multiagent systems: A personal view of the state of the art. In *IEEE Transaction on Knowledge and Data Engineering*, volume 11, No. 1 Jan/Feb 1999, pages 133–142, 1999.

[4] Frank Buschmann; Regine Meunier; Hans Rohnert; Peter Sommerlad; Michael Stal. *Pattern-orientierte Software-Architektur. Ein Pattern-System.* Addison-Wesley, 1st edition, 1998. ISBN 3-8273-1282-5.

[5] H. Ritter M. Hasenjäger. Active learning in neural networks. Technical report, Technische Fakultät, Universität Bielefeld, Germany, 1998.

[6] Leonard G.C. Hamey Tirthankar RayChaudhuri. Accurate modelling with minimised data collection – an active learning algorithm. Technical report, School of MPCE, Macquarie University, New South Wales 2109, Australia, 1995.

[7] Axel Röbel. The dynamic pattern selection algorithm: Effective training and controlled generalization of backpropagation neural networks. Technical report, TU Berlin, Germany, 1998.

[8] Helmut Mayer Roland Schwaiger. Is evolution a good teacher? genetic algorithms create training data sets for artificial neural networks. Technical report, Department of Computer Science, University of Salzburg, A-5020 Salzburg, Austria, 1998.

[9] Kenji Nakayama Kazuyuki Hara. Training data selection method for generalization by multilayer neural networks. Technical report, The Institute of Electronics, Information and Communication Engineers, Tokyo 105, Japan, 1998.

[10] Third untited nations conference on the exploration and peaceful use of outer space, 1998. A/CONF.184/BP/3.

[11] Stephen Quirolgico; Kip Canfield; Timothy Finnin; James A. Smith. Communicating neural network knowledge between agents in a simulated aerial reconnaissance system. Technical report, Department of Information Systems, University of Maryland. Baltimore County, 2000.

[12] Bjarne Stroustrup. *Die C++ Programmiersprache.* Addison-Wesley, 3rd edition, 1998. ISBN 3-8273-1296-5.

[13] Rainer Burkhardt. *UML - Unified Modeling Language. Objektorienntierte Modellierung für die Praxis.* Addison-Wesley, 2nd edition, 1999. ISBN 3-8273-1407-0.

[14] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Design Patterns - Elements of reusable object oriented software.* Addison-Wesley, 1st edition, 1995. ISBN 0-201-63361-2.

[15] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, 1st edition, 1996. ISBN 3-89319-950-0.