NTNU
Norwegian University of
Science and Technology

Creative Behaviour in Evolving Agents

Bjørnar Walle Alvestad
Endre Larsen

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Björn Gambäck, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Computational creativity is a field of research that is primarily focused on music composition, art and linguistics. However, research pertaining to computational creative behaviour is sparse. There is no clear, established methodology when it comes to making agents behave creatively, or how to apply machine learning techniques to make them do so.

Using Evolutionary Algorithms (EAs), agents were evolved using different modular parts. The parts were several sensors and actuators that the agents used to solve tasks given to them. Their behaviour was determined by three different Artificial Intelligence (AI) methods. This thesis looks into how these three AI methods, namely Continuous Time Recurrent Neural Network (CTRNN), Fuzzy Logic, and NeuroEvolution of Augmenting Topologies (NEAT), performed when evolving behaviour in the evolving agents. In order to investigate this, a system capable of evolving and visualising agents in different environments was developed.

The results of using these methods show few signs of the evolution of creative behaviour. Instead, the agents simply evolved to solve a given task. However, some unexpected actions can arguably be considered creative.

This thesis contains the following contributions: A study of how different AI methods perform when evolving creatures to behave creatively, and an implemented system that evolves creatures in different environments using the three AI methods mentioned above.

Sammendrag

Datamaskinell kreativitet er et forskningsfelt som hovedsaklig er fokusert på komposisjon av musikk, kunst og lingvistikk. Derimot er forskning rundt kreativ oppførsel mangelfull. Det er ingen tydelig og etablert metodologi når det kommer til å få agenter til å oppføre seg kreativt, eller hvordan man skal bruke maskinlæring til å få dem til å gjøre det.

Ved å bruke evolusjonære algoritmer (EA), ble agenter utviklet med modulære deler. Delene var flere sensorer og aktuatorer som agentene brukte til å løse problemstillingen gitt dem. Oppførselen deres ble bestemt av tre forskjellige kunstig intelligens (AI) metoder. Denne masteroppgaven tar for seg hvordan disse tre metodene, nemlig CTRNN, Fuzzy Logikk og NEAT, yter når oppførsel utvikles i de utviklende agentene. For å utforske dette ble et system utviklet for å utvikle og visualisere agenter i forskjellige miljøer.

Resultatene av å bruke disse metodene viser få tegn på utvikling av kreativ oppførsel. Agentene blir istedenfor utviklet til å kun løse den gitte problemstillingen. Det finnes derimot noen uventede handlinger som kan argumenteres for å være kreative.

Denne masteroppgaven inneholder følgende bidrag: En studie av hvordan forskjellige AI metoder yter når man utvikler agenter til å oppføre seg kreative, og et implementert system som utvikler agenter i forskjellige miljøer med å bruke de tre AI metodene nevnt over.

Preface

This is the report for a master's thesis in the Computer Science study at NTNU, spring 2017. The thesis falls under the theme of computational creativity and is supervised by Björn Gambäck.

We would like to extend thanks to our supervisor Björn Gambäck for helpful feedback and guidance throughout the thesis. We also thank Marinos Koutsomichalis for useful help and feedback towards the end of the semester, and to our friend and fellow student Viktor Zoric for discussion, feedback on the report, and cooperation on developing the implementation for Artificial Neural Networks (ANNs) and Evolutionary Algorithms (EAs).

Endre Larsen

Bjørnar Walle Alvestad

Trondheim, 8th June 2017

Contents

1. Introduction **1**
1.1. Background and motivation . . . 1
1.2. Project goals . . . 2
1.3. Contributions . . . 2
1.4. Thesis structure . . . 3

2. Background Theory **5**
2.1. Evolutionary Algorithms . . . 5
2.1.1. Key principles . . . 5
2.1.2. Genotype . . . 6
2.1.3. Phenotype . . . 7
2.1.4. Selection and genetic operators . . . 7
2.1.5. Types of Evolutionary Algorithms . . . 9
2.1.6. Algorithm . . . 9
2.2. Artificial Neural Networks . . . 10
2.2.1. Neurons and connections . . . 11
2.2.2. Continuous Time Recurrent Neural Networks (CTRNNs) . . . 13
2.3. NeuroEvolution of Augmenting Topologies (NEAT) . . . 14
2.3.1. Topology in NEAT . . . 14
2.3.2. Speciation . . . 17
2.4. Fuzzy Logic . . . 17
2.4.1. Fuzzy sets . . . 17
2.4.2. Rules and fuzzy inference . . . 19
2.4.3. Mamdani-style reasoning . . . 20
2.4.4. Sugeno-style reasoning . . . 23
2.5. Evaluating computational creativity . . . 23
2.5.1. The FACE model . . . 24
2.5.2. The IDEA model . . . 24
2.5.3. SPECS . . . 24

3. Related work **27**
3.1. Creative behaviour . . . 27
3.2. Curious learning agents . . . 27
3.3. Creativity in Artificial Intelligence (AI) . . . 30
3.4. Real Time NeuroEvolution of Augmenting Topologies (rtNEAT) and NERO 31

4. Architecture **33**
4.1. Agent world . . . 33
4.1.1. Food world . . . 34
4.1.2. Poison world . . . 34
4.1.3. Arena world . . . 34
4.2. Algorithms and AI-methodologies . . . 35
4.2.1. Physical agent evolution . . . 35
4.2.2. Behaviour evolution . . . 37
ANN-module . . . 37
Fuzzy-module . . . 39
NEAT-module . . . 39
4.2.3. Run mode . . . 40
4.3. System configuration . . . 40
4.4. System limitations . . . 42

5. Experiments and Results **43**
5.1. Experimental plan . . . 43
5.2. Food world scenario . . . 45
5.2.1. Designed food world creatures . . . 45
Food creature 1 . . . 46
Food creature 2 . . . 47
Food creature 3 . . . 48
Food creature 4 . . . 49
5.2.2. Fully evolved food creatures . . . 51
CTRNN . . . 52
Fuzzy Logic . . . 53
NEAT . . . 54
5.3. Poison world scenario . . . 55
5.3.1. Designed poison world creatures . . . 55
Poison creature 1 . . . 56
Poison creature 2 . . . 57
Poison creature 3 . . . 58
Poison creature 4 . . . 59
5.3.2. Fully evolved creatures . . . 60
CTRNN . . . 61
Fuzzy Logic . . . 62
NEAT . . . 63
5.4. Arena world scenario . . . 64
5.4.1. Designed arena world creatures . . . 64
Arena creature 1 . . . 65
Arena creature 2 . . . 66
Arena creature 3 . . . 67
Arena creature 4 . . . 68

5.4.2. Fully evolved creatures 69
CTRNN 70
Fuzzy Logic 71
NEAT 72
5.5. Additional experiments 73
5.5.1. Nose navigation in a food world 73

6. Evaluation and Discussion 77
6.1. Evaluation 77
6.1.1. Food world evaluation 78
6.1.2. Poison world evaluation 78
6.1.3. Arena world evaluation 79
6.1.4. System evaluation 79
6.2. Discussion 81

7. Conclusion 85
7.1. Contributions 85
7.2. Future work 85
7.2.1. Further experiments 85
7.2.2. System improvements 86

Bibliography 89

Appendices 91
A. Static fuzzy sets 91
B. Configuration files 93
B.1. Creative Creature Behaviour (CreBe) general configuration 93
B.2. Designed experiments configuration 95
B.3. Full-evolution experiments configuration 96
B.4. Nose-navigation configuration 98
C. CreBe software guide 99

List of Figures

2.1. Discrete genotype representation 6
2.2. Real-valued genotype representation 7
2.3. Tree genotype representation 7
2.4. A simple Artificial Neural Network 11
2.5. Neuron 12
2.6. Mutation in NEAT 15
2.7. Crossover in NEAT 16
2.8. Membership function 18
2.9. Classical membership function 19
2.10. Overview of membership functions 20

3.1. Curious learning agent process model 29

4.1. Example of a creature 33
4.2. Entity overview 36

5.1. Designed creatures - Food world 45
5.2. Food world - Designed creature 1 46
5.3. Results - Food world - Creature 1 46
5.4. Food world - Designed creature 2 47
5.5. Results - Food world - Creature 2 47
5.6. Food world - Designed creature 3 48
5.7. Results - Food world - Creature 3 48
5.8. Food world - Designed creature 4 49
5.9. Results - Food world - Creature 4 49
5.10. Individual runs - CTRNN - Food world - Creature 4 50
5.11. Result - New runs CTRNN - Food world - Creature 4 50
5.13. Evolved creatures - Food world - CTRNN 52
5.14. Evolved creatures - Food world - Fuzzy Logic 53
5.15. Evolved creatures - Food world - NEAT 54
5.16. Designed creatures - Poison world 55
5.17. Poison world - Designed creature 1 56
5.18. Results - Poison world - Creature 1 56
5.19. Poison world - Designed creature 2 57
5.20. Results - Poison world - Creature 2 57
5.21. Poison world - Designed creature 3 58
5.22. Results - Poison world - Creature 3 58

5.23. Poison world - Designed creature 4 59
5.24. Results - Poison world - Creature 4 59
5.26. Evolved creatures - Poison world - CTRNN 61
5.27. Evolved creatures - Poison world - Fuzzy logic 62
5.28. Evolved creatures - Poison world - NEAT 63
5.29. Designed creatures - Arena world 64
5.30. Arena world - Designed creature 1 65
5.31. Results - Arena world - Creature 1 65
5.32. Arena world - Designed creature 2 66
5.33. Results - Arena world - Creature 2 66
5.34. Arena world - Designed creature 3 67
5.35. Results - Arena world - Creature 3 67
5.36. Arena world - Designed creature 4 68
5.37. Results - Arena world - Creature 4 68
5.39. Evolved creatures - Arena world - CTRNN 70
5.40. Evolved creatures - Arena world - Fuzzy logic 71
5.41. Evolved creatures - Arena world - NEAT 72
5.42. Designed creature used in the nose navigation experiment. 73

1. Sensor entities fuzzy sets 91
2. Actuator entities fuzzy sets 92

List of Tables

4.1. Overview of entitiy types. 36

5.1. The most relevant configuration options used in the designed creature experiments. 43
5.2. A recap of the different world scenarios and behaviour brain-modules in the system. 44
5.3. The most relevant configuration options used in the full evolution experiments. 44
5.4. System configuration used in the nose navigation experiment. 73

Acronyms

AI	Artificial Intelligence. i, 1–3, 5, 27, 29, 30, 35, 37, 39, 77, 82, 85
ANN	Artificial Neural Network. iii, 3, 10–14, 37, 39, 40, 86
COG	Center Of Gravity. 19–21
CreBe	Creative Creature Behaviour. 3, 33, 78–80, 85, 91
CTRNN	Continuous Time Recurrent Neural Network. i, ii, 13, 14, 37, 39, 40, 43, 46, 49–52, 56, 57, 59, 61, 68–70, 74–76, 78–80, 82, 83
EA	Evolutionary Algorithm. i, iii, 1, 3, 5–7, 9, 14, 35, 37, 38, 40, 42, 60, 93–95
FOV	Field Of View. 35, 36, 51, 73
GUI	Graphical User Interface. 87
HP	Hitpoints. 34, 86, 89
NEAT	NeuroEvolution of Augmenting Topologies. i, ii, 5, 14–16, 23, 31, 37, 39, 40, 43, 48, 54, 56, 57, 60, 63, 69, 72, 74, 75, 78–80, 82
NERO	NeuroEvolving Robotic Operatives. 30, 31
SPECS	Standardised Procedure for Evaluating Creative Systems. 21, 77–80
WA	Weighted Average. 20

1. Introduction

This thesis is about creative behaviour and explores some possibilities of achieving and evaluating computational creativity. In this section the motivation behind the thesis is presented and the various project goals are explained. The contributions are listed, and lastly the structure of this thesis is given.

1.1. Background and motivation

Most work on computational creativity is related to the fields of art, music composition and linguistics, such as *ThePaintingFool*[1], *DeepArt*[2], *Experiments in Musical Intelligence* (Cope, 1996), and poetry generation in Bengali (Das and Gambäck, 2014). However, the topic of computational creativity in agent behaviour is less studied. Maher et al. (2008) described how computational models based on curiosity could be used to create creative behaviour in learning agents. They investigated how a computer controlled agent (a sheep) would behave and act using curious reinforcement learning. This is one example of creative behaviour, which may have applications in artificial life, simulations, complex systems and video games. However, this thesis is focused on the comparison between different computational models and initial configurations in order to investigate differences in the resulting agent behaviour when applying Evolutionary Algorithms (EAs).

Creativity is in many ways a subjective understanding; what may be considered creative by some, may be considered unoriginal by others. There are however some mutual properties about creativity that most people agree with, such as novelty, uniqueness and usefulness. An interesting topic is how to capture these aspects in a computational model of Artificial Intelligence (AI) and display the result as the behaviour of an agent. The evaluation of creativity is also equally important and complex, and probably captures the key aspects of creativity better than the computational models themselves. Evaluating creativity in the context of agent behaviour is a complex undertaking, where human intervention and evaluation is likely to take place.

[1]http://thepaintingfool.com/
[2]https://deepart.io/

1.2. Project goals

In this section, the four goals of the thesis are defined as follows.

G1: Evolve creative behaviour in custom agents[3]
The agents will be able to develop their own attributes and thereafter maximize their behaviour to a given task. The goal here is that given enough exploration, the solutions will be diverse and somewhat novel. A more well-defined form of this goal is:
Implement a system using AI methods to evolve creatures that exhibit creative behaviour in a strictly defined environment.

G2: Implement a system to evolve and visualise creature performance
In order to test several AI methods a system needs to be implemented that can interchange between methods for evolving creatures. The system should be able to visualise all of the required scenarios and give a good indication on how the creatures are performing.

G3: Study several AI methods and their influence on evolving creative behaviour
Testing several different AI methods should give insight to the properties that are important for evolving creative behaviour. Both in terms of performance regarding the evolution, and how creative the results can be considered.

G4: Study how different environments and tasks affect a creature's ability to perform creatively
The creature's task is defined by the current scenario. Looking at the degrees of creativity achieved in different scenarios should show how a task's complexity affects the creature's ability to evolve in a novel direction.

1.3. Contributions

The contributions for this thesis are:

C1: A study of how different AI methods perform when evolving creatures to behave creatively.

C2: An implemented system that evolves creatures using different selectable AI methods and environments.

[3]Also referred to as creatures.

1.4. Thesis structure

The relevant theory for this thesis, such as EAs and Artificial Neural Networks (ANNs) is presented in Chapter 2.

Then, in Chapter 3, related work and the current state-of-the-art is looked further into. It contains work done on creativity and AI in general and work that share similarities with this thesis.

Chapter 4 describes the architecture of the implemented system, called Creative Creature Behaviour (CreBe). Here it is explained how the EA and the different behaviour-modules were implemented, and the different scenarios that the creatures will be tested in.

The experiments performed are presented in Chapter 5. This includes how they were performed, such as experimental parameters and setup. The chapter also includes the result for each experiment, with a small discussion of the results.

In chapter 6, the various results are discussed in more detail and depth. Also included in this chapter is a project evaluation, where each project goal is addressed and evaluated.

Lastly, chapter 7 contains the final conclusions with contributions and possible future work.

2. Background Theory

This chapter will explain different concepts and theories used throughout the thesis. The first part is about evolutionary algorithms, before a section about artificial neural networks. Then the NeuroEvolution of Augmenting Topologies (NEAT) method is detailed, followed by a section about fuzzy logic. Lastly, some theory on how to evaluate computational creativity is presented.

2.1. Evolutionary Algorithms

Evolutionary Algorithms (EAs) is a class of optimization algorithms, widely used in Artificial Intelligence (AI) and machine learning systems. It mimics the biological process of evolution and natural selection in order to find solutions to predefined optimization and search problems.

2.1.1. Key principles

In an EA context, a candidate solution is called an *individual* because of its residence within a *population* at a given time. Artificial evolution is based on many of the same principles as natural evolution (Floreano and Mattiussi, 2008):

- Maintenance of a population; a collection of individuals or solution candidates to our problem. The population is changed at each *generation*, i.e. each iteration of the algorithm. Based on configuration and type of EA in use, the population can be partially or completely replaced at each generation.
- Creation of diversity, in order to explore different solutions and cover the whole search space. A lack of diversity can lead to the evolution getting stuck in a suboptimal solution.
- A selection mechanism, responsible for evaluating and assigning each individual a quantitative score called the *fitness value*, which reflects how well this individual solves the given problem. The best individuals are then selected for reproduction, having their genetic code transferred to new individuals in the next generation.
- Genetic inheritance, which ensures that beneficial properties of individuals are transferred to and inherited by their offspring.

The evolutionary process in an EA is guided by the fitness function, which aims to rank an individual on how well it satisfies or solves the optimization problem. As in real world natural evolution, the fitness value of an individual describes how likely this individual is to reproduce and pass on its genetic information to the next generation. The goal is to assign a high number of offspring to the best individuals, making the next generation generally more fit to solve the optimization problem (Floreano and Mattiussi, 2008, chapter 1).

The procedure of an EA is an iterative process, where the population is first filled with randomly generated individuals. It is vital that the generated individuals are evenly spread across the search space, in order to avoid the search to get trapped in suboptimal solutions (local maxima). The selection mechanism will assign a fitness value to each individual and use a selection strategy to select individuals to reproduce. Genetic operators duplicates and combines the genetic code of the best individuals, and applies a small random mutation in order to explore variations and introduce diversity. The newly created individuals are put back in the population in place for the old ones, and the next generation starts. This procedure is repeated until a satisfying or *good enough* solution is found.

2.1.2. Genotype

In EA terms, the genetic information of an individual is referred to as the *genotype*. It is responsible for transmitting features from parent to offspring, as well as for storing the genetic code that will define all aspects of an individual. There are different types and formats of genotypes used in EA. Some of the most commonly used genotype representations are:

Discrete representation, which consists of n values drawn from a set (alphabet) with cardinality k. A binary representation has a cardinality of 2, and alphabet [0,1]. The binary representation is often appreciated for its data-density and high performance on binary computers. Other discrete representations include the integer and character representations, with alphabets [0,1,2,...] and [A,B,C,...] respectively.

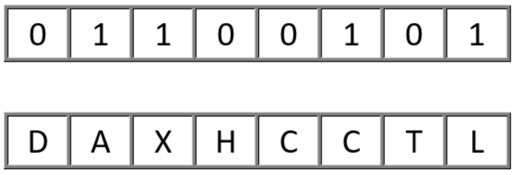


Figure 2.1.: Discrete representation, showing binary and character representation, respectively. In this example, $n = 8$.

Real-valued representation stores a series of n real-valued numbers. It is up to the EA designer to decide a legal range for each of these values, if any restrictions at all.

0.54	2.56	0.01	1.10	0.56	0.98	1.12	3.14

Figure 2.2.: Real-valued representation, with $n = 8$.

Tree representation is used to represent structures that require a hierarchical order. This genotype representation is typically used to represent mathematical expressions, a computer program, electronic circuits or other structures that require branching. The elements stored in the tree may be discrete values, real numbers or of other custom formats.

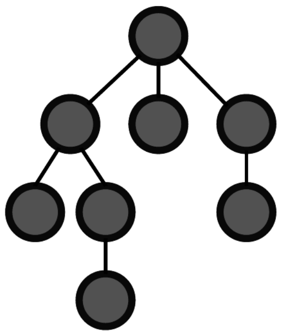


Figure 2.3.: Tree representation.

2.1.3. Phenotype

The physical manifestation of a genotype is called a *phenotype*, and represents an actual individual, or in an artificial evolutionary context, a candidate solution to the predefined problem. The fitness evaluation procedure of the EA system will operate on phenotypes explicitly, and only phenotypes are selected for reproduction. The terms phenotype and individual are often used interchangeably. They both *contain* the genotype, that is, the genetic information necessary to define an individual.

2.1.4. Selection and genetic operators

As mentioned before, the selection mechanism is used to allocate a bigger portion of the population to more fit individuals, in order to motivate for the evolution of beneficial genes and individuals. The *selection pressure* is the percentage of individuals that will create offspring for the next generation (Floreano and Mattiussi, 2008). A high selection pressure means few individuals get to reproduce, which can deteriorate the diversity of the population over time. On the other hand, a too low selection pressure may prohibit the evolution of beneficial genes because the best individuals are prevented from reproducing.

Different selection methods include:

- **Proportionate selection**. The probability of an individual reproducing and make a copy of its genetic code is the ratio between its own fitness and the total fitness of all individuals in the population:

$$p(i) = \frac{f(i)}{\sum_i^N f(i)} \tag{2.1}$$

  where $p(i)$ is the probability of individual i reproducing, $f(i)$ is the fitness value of individual i, and N is the population size (Floreano and Mattiussi, 2008).

- **Rank-based selection**. This method ranks all individuals from best to worst and then assign a reproduction probability based on this rank. This has the effect of neglecting the absolute difference in fitness values of individuals, which may negatively affect the selection pressure.

- **Truncated rank-based selection**. This method is identical to the regular rank-based method, but only the top n individuals in the ranked list is considered for reproduction.

- **Tournament selection**. For each offspring to be produced, a smaller subset of k individuals are selected randomly from the population, where the best individual in each tournament is allowed to reproduce. The k individuals are then placed back in the population and eligible for entry in new tournaments. This method offers a good compromise between selection pressure and genetic diversity (Floreano and Mattiussi, 2008), but can be somewhat computationally heavy, especially if k and the population size N are relatively big.

- **Elitism**. Not a complete method per se, but rather optional functionality in addition to the selection method. The strategy propose retaining n individuals from the previous generation, which will automatically be inserted into the new generation without modification.

Genetic operators, such as mutation and crossover, allow the algorithm to improve diversity, explore the search space and find novel solutions. Mutation is implemented as introduction of small, random changes to the genotype after reproduction. The amount of mutation to introduce, and the percentage of genomes to mutate, are often left out as a configuration option of the evolutionary system.

Crossover, or recombination, is responsible for the inheritance of features from parents during reproduction. Crossover requires a pair of genomes, which are selected randomly among the newly created offspring. There are many different ways to chose what part of the genotype to inherit from one parent and what part to inherit from the other. Some of these methods are: One-Point, Uniform and Arithmetic. The *one-point* crossover operator randomly selects a point in the two genomes, where the genetic material is swapped between the two genotypes. *Uniform* crossover selects n random points in which the genetic information is swapped. *Arithmetic* crossover only applies to real-valued genotype representation, by taking the average of the two genotypes at any given position.

2.1.5. Types of Evolutionary Algorithms

There are several types of EAs, tailored towards different optimization problems. The main difference between them is the choice of genotype representation and genetic operators. The best choice of type depends on the properties of the problem to be solved (Floreano and Mattiussi, 2008). The most relevant EA types are:

- **Genetic algorithms**, where the choice of genetic representation is binary or discrete, which makes the crossover methods simple and effective.
- **Genetic programming**, which operates on tree-based genotype representation. This method is often used on more complex optimization problems, such as computer programs and electronic circuit design.
- **Evolutionary programming**, which is used to optimize a set of problem or program parameters. A real-valued genotype representation is suited in this type of EA. Mutations are drawn from a zero-mean Gaussian distribution, making small mutations more common than large.
- **Island model**, where multiple independent populations are maintained and evolved in parallel. Every generation, some genotypes are exchanged between the populations in order to introduce variations and exploit potential synergies.

2.1.6. Algorithm

Even though multiple types of EAs exist, they share the same high level steps during execution (Floreano and Mattiussi, 2008, chapter 1).

1. The initial population is filled with N randomly generated individuals. It is important that the random population covers the whole search space, and that the generated individuals are evenly distributed in the search space. Low diversity in the initial population may lead to important features being missed in the evolution.
2. Evaluation of the population, using the fitness function. High fitness value entails that the individual is well suited to solve the optimization problem.
3. The best individuals are selected for reproduction. Many strategies exist, but they are all based on the principle that individual with higher fitness should have a higher probability of being selected for reproduction. This step is called *parent selection.*
4. If more than n individuals exists at this point, the population is reduced in order to fit in the original population. This selection is called *adult selection*, and is also based on the fitness value of the individual.

5. The selected individuals undergo crossover and mutation, which is responsible for inheritance of parent features, and of small random changes. This ensures that the search space is explored during the evolutionary process.

6. The evolution halts if the highest fitness in the population reaches a predefined threshold, or after a set number of generations. Otherwise, continue the evolution by going to step 2.

2.2. Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model and classification algorithm that tries to emulate the features of a biological nervous system. ANNs consist of nodes, called neurons, that are structured and interconnected in specific patterns using weighted connections. Most commonly, the neurons are separated into layers. In this case, a designated layer called the input layer is responsible for receiving external information and forward it to the next layer. Such networks are called *feed forward networks*, due to the way data and signals are always propagated forward in the network. Other layers are to a certain degree connected to the previous layer of the network. Figure 2.4 presents a simple feed-forward neural network, where information is forwarded from left to right. This is a fully-connected network, meaning all neurons in one layer is connected to every neuron in the next layer. (Floreano and Mattiussi, 2008, chapter 3).

Notable features of ANNs are (Floreano and Mattiussi, 2008):

- Adaptiveness. A neural network can be *trained* to recognize and react to different input patters, by running a so called training set multiple times on the network. A training set includes a high number of data-label pairs randomly sampled from a larger data domain, where the label denotes the "correct" network output for the corresponding input data. The resulting error term obtained when running a data-label example, is used to alter connection weights in the network in a way that will minimize future error terms. This procedure is known as the backpropagation algorithm (Werbos, 1974).

- Robustness. A neural network is to some degree resilient from signal noise, input degradation and malfunctioning of connections and neurons. Even if such events occur, the network responds by decreasing its output accuracy and increasing the error rates. Neural networks can also, to some degree, be trained to compensate for damages and input noise.

- Flexibility, in that neural networks can be used in a variety of domains and solve different types of problems. However, some types and configurations of neural networks are more or less applicable to certain problem types.

- Generalization. A neural network can successfully classify input patterns that has never been seen before, given that the input shares similarities to training patters.

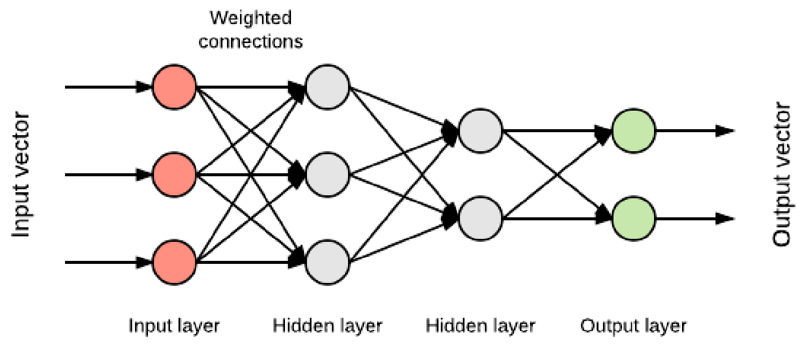


Figure 2.4.: A simple ANN with two hidden layers (gray nodes). The input layer (red) takes a vector of length 3 and feeds each of the values into the neurons of the first hidden layer. The output layer (green) outputs a vector of length 2, and is considered the output of the network. The configuration of this network is $[3, 3, 2, 2]$, which specifies the number of neurons in each layer and the total number of layers.

2.2.1. Neurons and connections

Each neuron uses a function, called an activation function, that takes in the weighted input sum and calculates the output of the neuron. This value is then forwarded to all neurons connected to it. An activation threshold can be specified so that the weighted input must be greater than this constant value in order to activate. By using different configurations of connections in the network, one can decide which neurons should be affected by other activated neurons. This often creates an emergent pattern that neurons activated together, often activate together again, showing similarities to how the human brain operates (Floreano and Mattiussi, 2008, chapter 3). Figure 2.5 shows a schematic illustration of a neuron, with internal functionality presented.

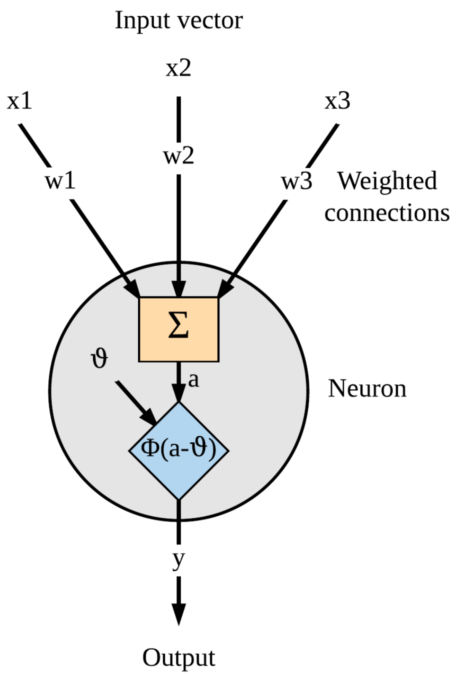


Figure 2.5.: Schematic illustration of a neuron, with input vector $\vec{x}$, connection weights $\vec{w}$, activation threshold ϑ and output y. Φ is the activation function.

Every connection in the network has a multiplier factor tied to it, usually referred to as a weight. These weights determine how important another neuron's activation should be for the current neuron activation. It is by altering these weights that ANNs exhibit learning capabilities (Floreano and Mattiussi, 2008, chapter 3).

The activation a_i of a neuron i is given by the scalar product between its input weights $\vec{w}_i$ and the input vector $\vec{x}$:

$$a_i = \vec{w}_i \cdot \vec{x} = \sum_{j=1}^{N} w_{ij} x_j \tag{2.2}$$

where N is the number of weights/inputs to the neuron.

The output signal y_i of a neuron i is computed using the activation function $\Phi(\cdot)$, with the weighted neuron input a_i. The neuron threshold ϑ_i is subtracted from the weighted input sum:

$$y_i = \Phi(a_i - \vartheta_i) \tag{2.3}$$

The activation function $\Phi(\cdot)$ can take many different forms, which will affect the performance and behaviour of the network. Some of the most common activation functions are linear functions, the step function and the 'S'-shaped sigmoid function.

2.2.2. Continuous Time Recurrent Neural Networks (CTRNNs)

A subset of ANNs are CTRNNs. The keywords here, continuous time and recurrent refer to the attributes of these specific networks.

A network can be either continuous or discrete in how its output changes in regards to time. A continuous-time model will mean that the network output changes continuously, while a discrete-time model means that the network output changes discretely (Beer, 2006).

When it comes to the architecture of a network, it is often split into feed-forward and recurrent. Feed-forward is what is most commonly associated with ANN. Here the information from the input is fed forward through layers in the network and processed in each of them, before resulting in the output. In the recurrent architecture, there can also be connections that go in the opposite direction, i.e. allows for feedback loops (Beer, 2006).

A neuron i in a CTRNN has an internal activation state y_i, which is persistent between successive activations. Equation 2.4 shows the derivative of y_i, i.e. the change in internal activation with respect to time:

$$\frac{dy_i}{dt} = \frac{1}{\tau_i}(-y_i + a_i + \vartheta) \tag{2.4}$$

where:

- y_i is the current internal activation of neuron i.
- t is time, which is represented as discrete timesteps. Each successive run of a CTRNN constitutes a timestep.
- τ_i is the time constant, where a low value entails fast change and more spontaneous reactions from input. A high value creates more memory in the system as the internal activation undergoes less "leakage" between each timestep.
- a_i is the weighted neuron input given in equation 2.2.
- ϑ_i is the activation threshold of neuron i.

The output value of a CTRNN neuron i is calculated using the sigmoid function:

$$o_i = \frac{1}{1 + e^{-g_i y_i}} \tag{2.5}$$

Notice that y_i is multiplied with the gain term g_i, which is a neuron specific coefficient for the activation.

In CTRNN, in addition to being connected to the previous layer, each neuron can also be connected to itself and other neurons in the same layer. These connections signify a connection to the previous state of said neuron. Compared to regular ANNs, neurons in a CTRNN retain some of their activation between successive runs, resembling simple forms of memory. Because of this, a CTRNN may not produce equal results for equal input (Beer, 1995).

2.3. NEAT

NeuroEvolution is the concept of evolving ANN by using genetic algorithms. Most often, this involves using a kind of EA in order to develop weights for a fully connected ANN, i.e. the topology of the network is decided beforehand. The argument for not using a fully connected network is that there will be a lot of unnecessary connections and weights, leading to more computation required.

In order to increase performance in neuroevolution, Stanley and Miikkulainen introduced the NEAT method. The increase in efficiency is due to three main factors: The crossover method for different topologies, the protection of structural innovation using speciation, and the fact that the growth always happens from a minimal network (Stanley and Miikkulainen, 2002).

2.3.1. Topology in NEAT

The third factor given by Stanley and Miikkulainen is pertaining to the superfluous nodes and connections in a fully connected network with hidden layers. Due to this, a network in NEAT always starts with the same topology: All input nodes are directly connected to the outputs, meaning no hidden nodes are used. This insures that the population of networks always evolve from minimal structures. In order to increase the complexity, the networks will have to undergo mutation during evolution.

There are two types of mutation for the topology in NEAT; adding a new node and adding a new connection. When a mutation results in a new node, it splits an existing connection with a node, resulting in two new connections. The old connection is set to be disabled, while its weight is transferred to the connection leading out of the new node. The connection weight leading into the new node is set to 1 (Stanley and Miikkulainen, 2002).

As can be seen in figure 2.6, each new connection is added to the end of a list. This list is what allows crossover between networks with different topologies. Each mutation is given a so called innovation number. This number serves as an identification for the related mutation. Whenever a mutation occurs, it is given an incremented number to be its innovation number. Within each generation, a table keeps track of all the mutations,

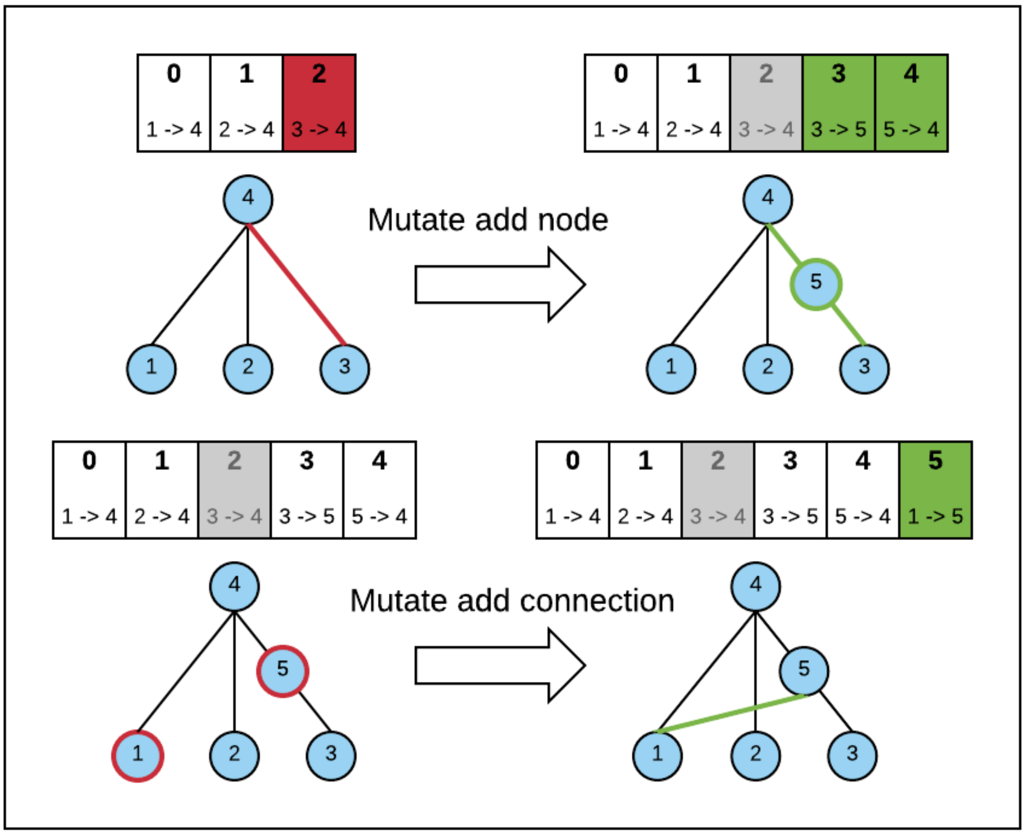


Figure 2.6.: Mutation in NEAT. Red shows the selected spot for mutation and green shows the resulting additions. The big numbers in each cell indicate the innovation number.

in order to ensure that a particular mutation is not assigned multiple innovation numbers.

For crossover, the innovation numbers are used to guide the topology inheritance. The system can now know how connections from different genotypes match up. The connections can also be called genes. Genes can be separated into three groups: Matching, disjoint and excess genes. Matching genes are genes that are in both of the parents. Disjoint and excess genes are genes that only one of the parents have. The difference between them being that disjoint genes have an innovation number between the other parents lowest or highest innovation number, while excess genes have innovation number outside of the range of lowest to highest innovation number in the other parent (Stanley and Miikkulainen, 2002).

All of the matching genes are passed down to the offspring, and the weights are randomly chosen from either parent for each gene. The disjoint and excess genes are only passed down from the most fit parent (Stanley and Miikkulainen, 2002). When both parents have the same fitness, the non-matching (disjoint and excess) genes are passed down from both, as can be seen in figure 2.7.

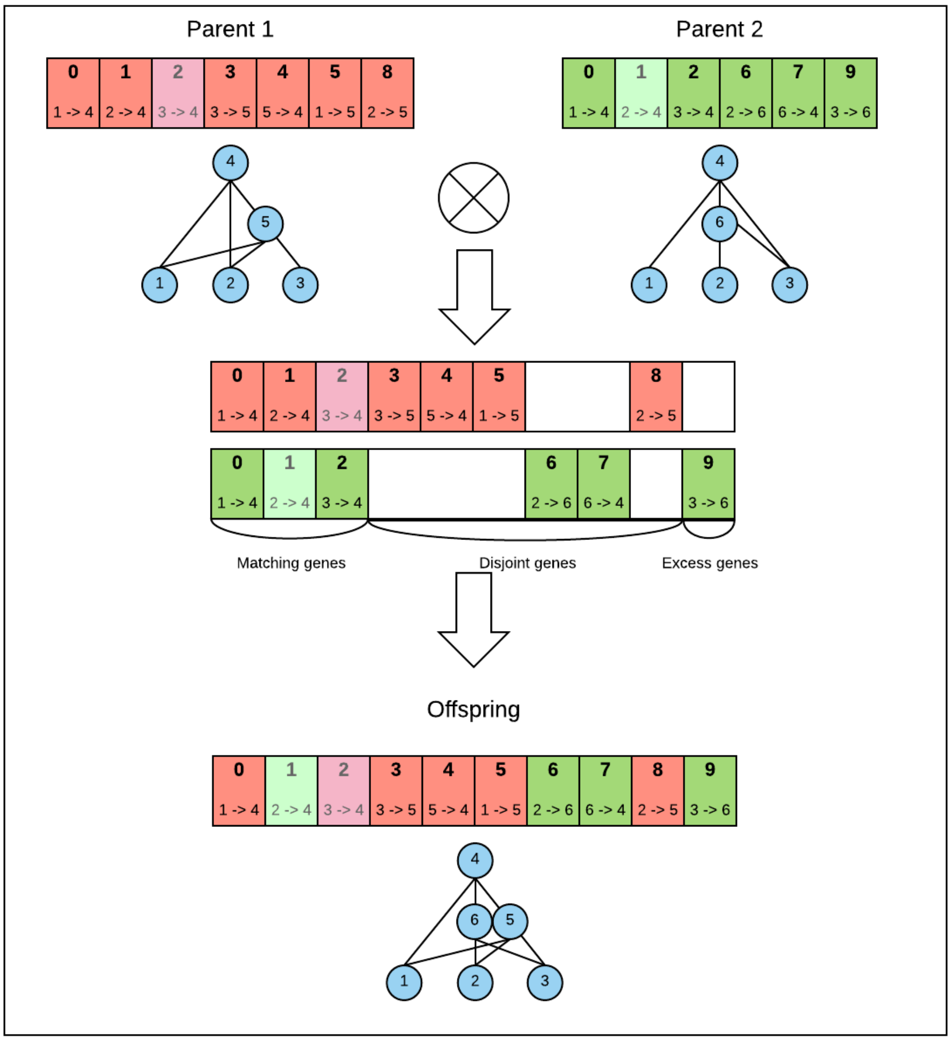


Figure 2.7.: Crossover in NEAT where both parents are equally fit. Lighter colours indicate a disabled connection. The big numbers in each cell indicate the innovation number.

2.3.2. Speciation

When a network develops a new structure, it is very likely that it will not survive before it is optimized. In order to protect structural innovation, NEAT uses what is called speciation. This is the concept that groups similar solutions together. For each species, NEAT uses explicit fitness sharing. That is, within each species, all solutions have the peak fitness. The height of the peak determines how many solutions that can exist in a species. For measuring compatibility with a species, the number of non-matching genes are used. If the compatibility is too low for every existing species, then a new species is created (Stanley and Miikkulainen, 2002).

2.4. Fuzzy Logic

Fuzzy logic can be considered a generalization of classical logic, in the way that fuzzy propositions can represent degrees of truth, in contrast to the binary truth values found in classical logic. The truth value of a proposition can take any real value in the range $[0, 1]$, where the values 0 and 1 maps to the classical logical values *false* and *true* respectively.

2.4.1. Fuzzy sets

Consider a universal, classical set $X = \{x_1, x_2, ..., x_n\}, n \in \mathbb{N}$. A subset A of X has a binary membership vector $Z(A) = \{m_1, m_2, ..., m_n\}$, where the i'th element m_i specify whether or not element x_i is in the subset A. For example, a membership vector $Z(A) = \{1, 0, 1, ..., 0\}$ tells us that the elements x_1 and x_3 are members of A, because the first and third element is set to 1. A zero element denotes that the respective element is not part of the subset. In classical set theory, only the values 0 and 1 are allowed in a membership vector, i.e. an element is present in the subset or not. In fuzzy set theory however, no such limitation is imposed. Elements have a degree of membership (fuzziness), specified as a real value between 0 and 1. A fuzzy set B may have a membership vector $Z(B) = \{0.3, 0.9, ..., 0.0\}$, which implies that element x_1 has a somewhat small membership of B (0.3), while x_2 has almost complete membership (0.9).

A special kind of function, the *membership function*, is responsible for assigning membership value to each element of a fuzzy set. In classical logic, the membership function is only allowed to assign a membership value of 1 or 0, that is, if an element is in the set or not, respectively:

$$f_A(x) = \begin{cases} 1, & \text{if x} \in \text{A} \\ 0, & \text{if x} \notin \text{A} \end{cases} \tag{2.6}$$

where A is some classical set.

In fuzzy sets, an element can have any degree of membership in the range $[0, 1]$. A fuzzy set A of the universal set X is defined by the membership function $\mu_A(x)$:

$$\mu_A(x) : X \to [0,1] \tag{2.7}$$

where $x \in X$. The membership function describes the degree of membership for an element x in the fuzzy set A, which is all pairs $(x, \mu_A(x))$ where $x \in X$. Figure 2.8 shows two examples of membership functions. The classical logic analogy is presented in figure 2.9.

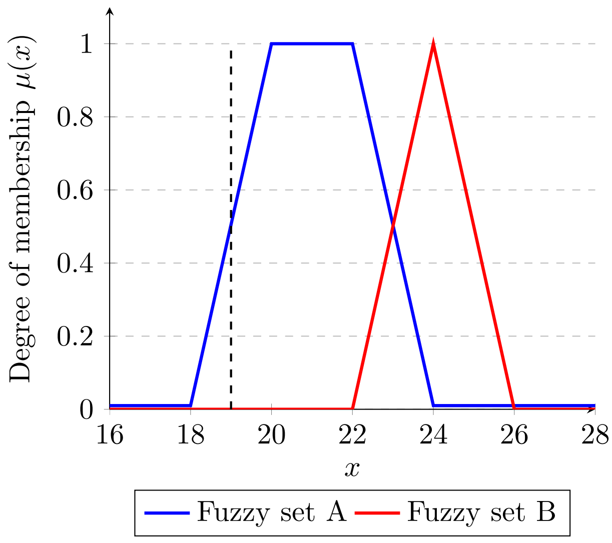


Figure 2.8.: A membership function operating on the universal set of all real-numbers $\mathbb{R}$. The dashed line at $x = 19$ indicates that the value 19 has membership 0.5 in fuzzy set A, and 0.0 in set B. The shape of these membership functions are *trapezoidal* and *triangle*, respectively. Figure 2.10 presents the most relevant membership function shapes.

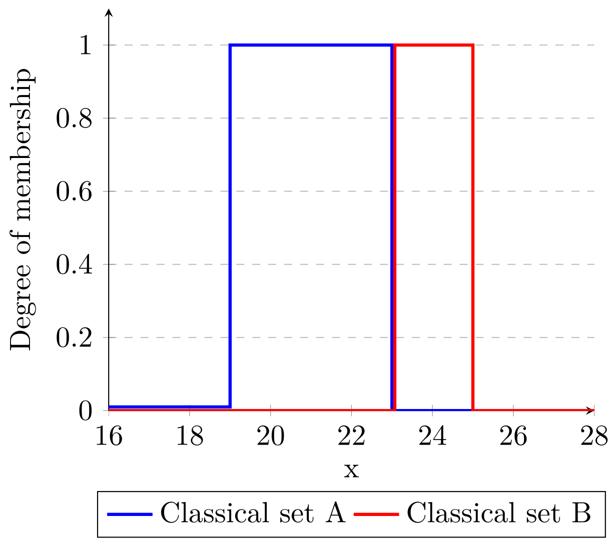


Figure 2.9.: The classical analogy to the membership functions shown in figure 2.8. Note the restriction to only horizontal and vertical lines, due to boolean logic.

2.4.2. Rules and fuzzy inference

In a fuzzy system, knowledge is captured in *rules*, specifically in the form:

IF x is X1
THEN y is Y1

Where x and y are linguistic variables, and X1 and Y1 are values defined by fuzzy sets. Consider the example of indoor temperature. Here, *temperature* is a linguistic variable, whilst the concepts *cold*, *ideal* and *warm* are values of this domain. A rule may consist of multiple propositions, using the intersection operation *AND*, and the union operation *OR*. The unary operation *NOT* is defined as the complement of a set:

IF x is X1
AND y is NOT Y3
THEN z is Z2

IF x is X1
OR y is Y3
THEN z is Z2

IF x is X1
OR (y is Y3 AND z is Z3)
THEN v is V2
(Negnevitsky, 2005)

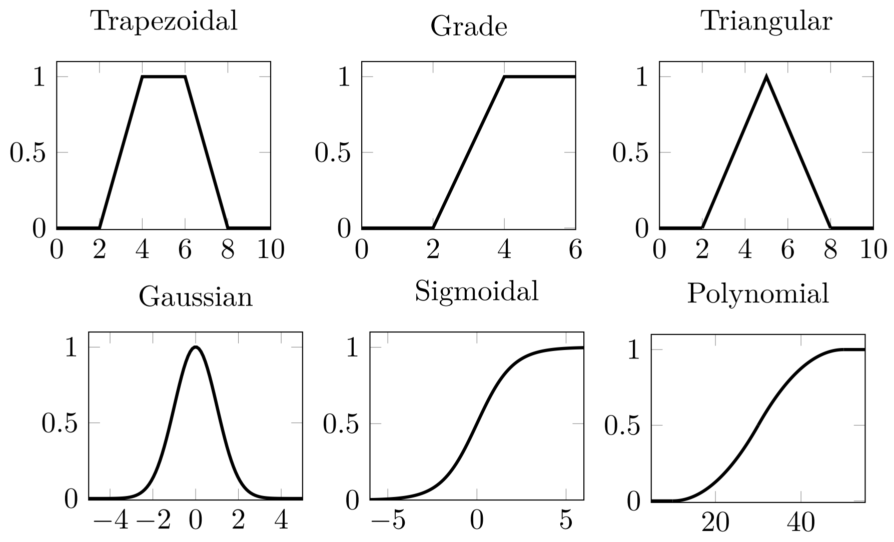


Figure 2.10.: The membership functions *trapezoidal*, *grade*, *triangular*, *gaussian*, *sigmoidal* and *S-formed polynomial*. The asymmetric examples also have reversed versions of themselves.

In practice, the rules of a fuzzy system are defined by experts in the appropriate field of work. The fact that the fuzzy variables, values and rules resemble human reasoning, makes it easier for an expert to produce efficient rules, compared to for example training a neural network (Rojas, 1996, p. 290).

2.4.3. Mamdani-style reasoning

The most commonly used fuzzy inference method is the Mamdani method, named after professor Ebrahim Mamdani who was one of the first to design a fuzzy system for a practical application (Negnevitsky, 2005; Mamdani and Assilian, 1975). This inference method consists of four steps:

1. Fuzzification of the input.
2. Rule evaluation.
3. Aggregation of the rule outputs.
4. Defuzzification.

In the first step, input variables (typically measurements or sensor data) are transformed into fuzzy membership sets using their respective membership function. Referencing figure 2.8, a measurement of $x = 19$ produces the fuzzy set $\{0.5/A, 0.0/B\}$[1].

[1]Note that the symbol "/" does not denote a fraction or division, but is merely used as a syntactical delimiter between the membership value and a fuzzy concept.

The second step is rule evaluation, where the fuzzified inputs are applied to each rule of the system. Each rule evaluates to a single real number, describing the degree of membership for a particular output value. In a rule, each clause evaluates to a real number, which is the membership for that particular value. For example, consider the fuzzy set $\{0.7/X1, 0.2/X2\}$, which is the fuzzified input of some system. The rule:

IF x is X1 THEN y is Y2

can be broken down into simple, mathematical operations. The output of the rule is the real-valued membership of Y2: $Y2 = X1 = 0.7 \to 0.7/Y2$.

When a rule consists of multiple propositions, the *max* operation is used in place for the union operator (OR), and the *min* operation is used in place for the intersection operator (AND). The rule:

IF x is NOT X1 OR x is X2 THEN y is Y2

evaluates to: $Y2 = max(1 - X1, X2) = max(1 - 0.7, 0.2) = 0.3 \to 0.3/Y2$.

In general, the different operations permitted on fuzzy sets are:

$\mu_A(x) \cap \mu_B(x) = \mu_{A \cap B}(x) = min[\mu_A(x), \mu_B(x)]$ (AND)
$\mu_A(x) \cup \mu_B(x) = \mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)]$ (OR)
$\mu_{\neg A}(x) = 1 - \mu_A(x)$ (NOT)

where $x \in X$, the universal set.

The third step of the inference process is to aggregate all rule outputs into one fuzzy set, which is then converted into a *crisp*[2] number. This is done by copying the output variable set, clipping each value at their respective membership degree. See figure 2.11. In order to obtain the crisp output value, the centroid technique is applied to the aggregated set. This method finds a point where a vertical line would split the set into two equal masses; the Center Of Gravity (COG):

$$COG = \frac{\displaystyle\int_a^b \mu_A(x) x dx}{\displaystyle\int_a^b \mu_A(x) dx} \tag{2.8}$$

where a and b are the start and end-points of the set, respectively (Negnevitsky, 2005).

[2]A single value, not fuzzified.

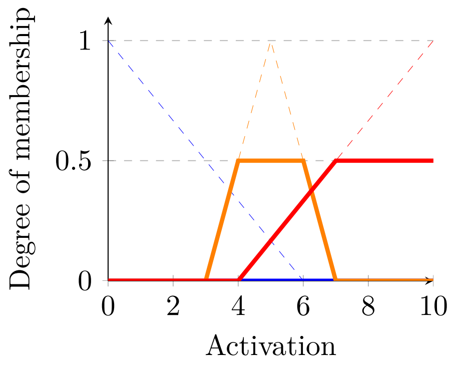


Figure 2.11.: Example plot of aggregated rule outputs, where two of the values are clipped at $y = 0.5$. The full membership shapes are shown in dashed lines.

In computer systems, the COG is estimated numerically using a fixed number of sample points, using the formula (Negnevitsky, 2005):

$$COG = \frac{\sum_{x=a}^{b} \mu_A(x)x}{\sum_{x=a}^{b} \mu_A(x)} \tag{2.9}$$

In figure 2.12, the center of gravity is calculated on an example plot using equation 2.9. The COG is shown as a dashed, vertical line in the figure, at $x = 6.75$. This is the crisp output of a fuzzy system.

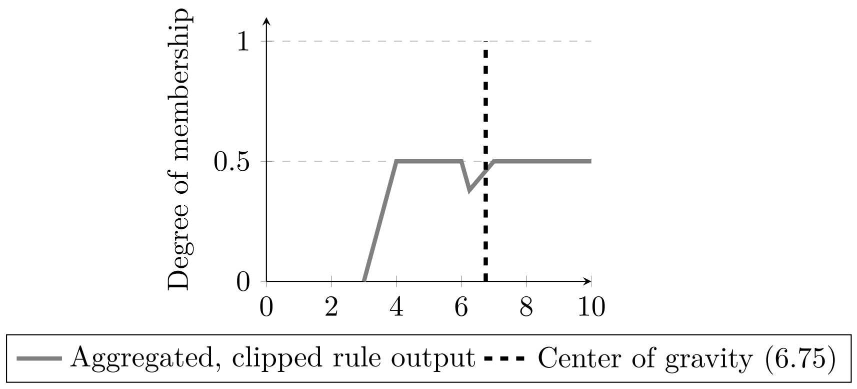


Figure 2.12.: Center of gravity calculated for an example aggregated plot. The set is the same as shown in figure 2.11.

2.4.4. Sugeno-style reasoning

Another method of rule inference is the Sugeno-style reasoning, introduced by Michio Sugeno in 1985 (Takagi and Sugeno, 1985). The main steps of the algorithm remain identical to the Mamdani-reasoner, but with some key differences in the rule format and evaluation. In Sugeno-reasoning, the consequent part of a rule, that is, the clause in the *THEN*-part, is instead defined as a function of the input parameters.

IF x is X1
AND y is Y1
THEN z is $f(x, y)$

where x and y are input variables, X1 and Y1 are fuzzy set values, z is the output variable, and $f(x, y)$ is a function that calculates the value of the output singleton set. When $f(x, y) = k$, where k is constant, the system is called a *zero-order Sugeno fuzzy model* (Negnevitsky, 2005).

The main advantage of the Sugeno-model is the computational efficiency over Mamdani-reasoning. Instead of calculating the COG using equation 2.8, the crisp output is calculated as a Weighted Average (WA) of all singleton outputs. In a zero-order Sugeno model, the WA is calculated using the equation:

$$WA = \frac{\mu(k_1)k_1 + \mu(k_2)k_2 + ... + \mu(k_n)k_n}{\mu(k_1) + \mu(k_2) + ... + \mu(k_n)} = \frac{\sum\limits_{i=0}^{n} \mu(k_i)k_i}{\sum\limits_{i=0}^{n} \mu(k_i)} \tag{2.10}$$

where n is the number of output singleton sets, k_i is the i'th singleton constant value, and $\mu(k_i)$ is the i'th rule output.

The compuational complexity of equation 2.10 scales linearly with the number of output sets, whereas the COG formula in equation 2.8 must be estimated numerically using an appropriate number of iterations. This makes the Sugeno-style inference more computational efficient in the vast majority of cases, because it generally needs less iterations to compute.

2.5. Evaluating computational creativity

In the field of measuring and evaluating computational creativity, different models have been suggested by Pease and Colton (2011). Namely the FACE model and the IDEA model. Building on this, Jordanous (2013) writes about a standardised procedure for evaluating creativity, Standardised Procedure for Evaluating Creative Systems (SPECS).

2.5.1. The FACE model

This model is about separating creativity into different generative acts. By generative acts, Pease and Colton (2011) mean a creative act. They call it generative due to the fact that if the question of creativity arises, something new has to have been created, i.e., generated. The different generative acts in the FACE model are:

- F: Framing information
- A: Aesthetic measures
- C: Concepts
- E: Expressions of a concept

As can be seen here, the FACE model splits the creative evaluation into several different parts. These four aspects each contain two generative acts: An act for the item of the aspect, and an act for a method of generating said item. Pease and Colton (2011) also point out that a single creative act, as they call it, will not necessarily fulfill each of the aspects in this model. They also specify that this model can be used both quantitatively, i.e. counting the creative acts, and qualitatively, i.e. comparing the acts against a given measure (Pease and Colton, 2011).

2.5.2. The IDEA model

IDEA is short for Iterative Development Execution Appreciation. The concept for this model is that one develops software and presents it to an audience in a cyclical fashion. The idea is that a measure of the creativity can be evaluated every cycle and be steered towards different aspects of creativity.

In order to calculate aesthetic criteria, Pease and Colton (2011) say that for an ideal audience, every individual can evaluate the effect an act had upon them. This is split into an indication of change in well-being from -1 to 1, and between 0 to 1 by how much cognitive effort they spent in order to appreciate it. These are then used in various formulas to calculate a score for different topics. Some examples of these topics are the shock-factor, disgustedness, popularity and how opinion-splitting the act was (Pease and Colton, 2011).

2.5.3. SPECS

Jordanous (2013) has also done some research on the evaluation of creative works. She came up with SPECS, the Standardised Procedure for Evaluating Creative Systems. In this, she specifies three main steps for evaluating the creativity of a system.

1. Identify the correct definition of creativity. This would be the definition that would determine if the system could be considered a creative system. After that, one should focus on the aspects of creativity that are most important for the system.

The definition of creativity used should reflect the domain of the creative work. An example of this can be of a computer creating a painting. The domain would then be the picture created. A definition of creativity for this, could be that the system creates something new and that it gets a reaction from a viewer.

2. Find each criteria from the definition found in step one, and use each of them as a standard to test the system.

3. Evaluate the system against the standards found in step two. One needs to consider each aspect of the definition while testing, and weigh the importance against each other. Notice that it is no need to combine all of the test results into one final score.

All these methods have one thing in common. Namely that they separate the evaluation into the different aspects of creativity, rather than evaluating the system as a whole (Jordanous, 2013).

3. Related work

This chapter will look into existing work and development in the field of computational creativity and computational creative behaviour. It will be focused on the current state of creative behaviour research, computational creativity evaluation research and previously used relevant Artificial Intelligence (AI) methods. There has been a modest amount of work in the field of creative behaviour evolution recently, compared to other related fields such as sentiment and linguistic analysis, art and music composition. However, a substantial amount of research has been conducted regarding the definition and measurement methods of computational creativity (Aguilar and Pérez y Pérez, 2014).

3.1. Creative behaviour

A great number of explanations and definitions have been proposed for creativity. Maher et al. (2008) define creative behaviour as behaviour that results in a product that is unique or somehow valuable to an individual or society. In a more behaviouristic context, creativity may be defined as a unique and valuable response or pattern of responses to a characteristic external input (Razik, 1976). That is, behaviour and actions that are considered unexpected and novel, but simultaneously valuable and meaningful for the acting entity. A clear definition of creative behaviour is useful in this thesis because one goal is to evolve creative behaviour for different worlds and scenarios. Amabile and Collins (1999) propose that there may be different types of creativity, namely extrinsically and intrinsically motivated creativity. Extrinsically motivated creativity revolves around external reward, recognition and direction from a supervising entity, much like reinforcement learning, in order to motivate creative behaviour. On the other hand, intrinsically motivated creativity is characterised by actions that are satisfying or in other ways interesting to the individual. The latter also motivates for a higher level of creativity, while extrinsic motivation can have a negative effect on the emergence of creativity, because it relies on external and/or human-like guidance (Amabile, 1996).

3.2. Curious learning agents

Computational creativity involves both the creation of some sort of creative product, such as art and poems, but also creative behaviour. Maher et al. (2008) focused on the latter in their work on curious learning agents. They considered two approaches, the first using reinforcement learning to train the agents, and the other supervised learning. The immediate difference is that supervised learning requires example data in order to exert learning capabilities. Such data can be hard to implement in an intrinsically motivated system, because the strategy entails high levels of intervention from an external source. This contradicts some aspects of the found definition of creativity, which states that human and other external influence should be avoided.

Maher et al. (2008) also experimented using reinforcement learning in curious agents. They used a computational model in order to detect the novelty of stimuli in real time (Saunders, 2001). The novelty figure is then used to calculate the curiosity value for this particular stimulus, and to to ultimately focus the attention of the perceiving agent. The goal is to teach the agent to explore its surroundings, and repeat interesting events that may have a positive influence on the individual. As seen in figure 3.1, the model incorporates multiple steps in order to convert a perceptual state into an action to be performed by the agent. As data is received from the environment through sensors, it is parsed by the *Sensation* module into an attribute based state representation, where each attribute represent some property of the observed world. As each state attribute take a numeric value, it allows for one state to be 'subtracted' from the previous, yielding the change between two states. This vector is referred to as an *event* by Maher et al., and is used to calculate the curiosity value for that particular event. Curiosity is used to update a policy using table-based Q-learning (Watkins, 1989). Actions are selected using a policy that maps states to actions, where the action with the highest curiosity value is selected. However, there is small chance that a random action is selected instead.

Maher et al. (2008) consider behaviour and patterns of behaviour that is new and unexpected as creative. In order to consider a particular behaviour as adopted and learned, the agents must be able to reproduce the behaviour at least five times to identical stimuli. In the validation of creative behaviour, Maher et al. consider emergent behaviour in terms of its focus, novelty and unexpectedness. Their first experiment considers the behaviour of a sheep in a computer game simulation, where the players are asked to build objects that attract the attention of the sheep. One such object is a food-machine, where food appears once a button is pressed. One observed behaviour was:

- Press the button, move to the food, eat the food, move back to the button...

This is more or less the expected behaviour of the sheep, and is not considered very creative, given the situation. Another observed behaviour was:

- Move to the food outlet, press the button, eat the food, press the button, eat the food...

In this situation, the sheep were able to repeatedly eat without having to move between the button and the food tray. Maher et al. found this behaviour particularly creative, because it was unexpected and demonstrates that the sheep was able to utilize a limitation in the food-machine that the designers were unaware of.

Another observed pattern of behaviour was that the sheep pressed the button multiple times to eject more than one food at a time. However, it was never observed that the sheep pressed the button more than three times before eating the food. This behaviour suggest that the curious learning approach has limitations in emergence of longer action sequences. If models of curiosity have the ability to support more complex behaviour, is still an open question. The findings of Maher et al. (2008) suggest that the evolution of creative behaviour is limited to short action sequences in simple microworlds. According to Russel and Norvig, Q-learning agents are seriously restricted in their ability to learn, due to their inability to look ahead and anticipate the outcome of their actions (Russel and Norvig, 2009, p. 874). This supports the conclusion of Maher et al., that the agents struggle to learn longer action-command chains. If one wishes to evolve a greater level of creativity, a more sophisticated level of curiosity is needed in order to motivate such behaviour. In other words, it is relatively easy to achieve simple creative behaviour, but a lot more complex if the goal is to evolve truly creative behaviour. This may limit the result of this thesis to simple agent behaviour, or behaviour that is subject to human intervention, which should be avoided in a computational creativity context.

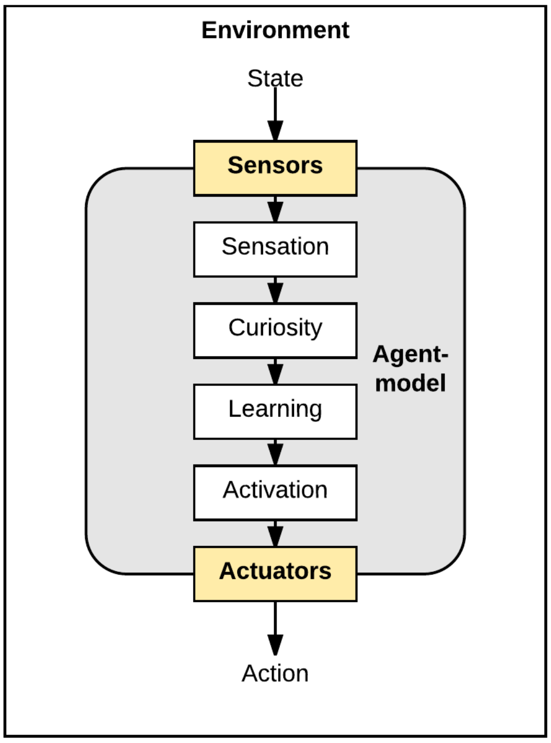


Figure 3.1.: The process model of a curious learning agent (Maher et al., 2008).

3.3. Creativity in AI

In Boden (1998), creativity is defined as something that is novel, surprising and valuable. There is also a distinction between psychological(P)-creativity and historical(H)-creativity. Here, P-creativity is something that produces novelties on a personal level, while H-creativity pertains to something that is novel for the entirety of history. It follows that AI should focus on P-creativity, as if an agent manages to model this, then it will lead to H-creativity in some cases.

Boden (1998) also separates creativity into three different types, namely combinational, exploratory and transformational creativity. The first of these being a novel combination of familiar ideas, such as poetic imagery and analogies. The two latter types of creativity separates from the first, but still have a lot of commonalities towards each other. Exploratory creativity contains novel ideas that search, or rather explore, a pre-existing structured conceptual space. The last type of creativity according to Boden (1998), transformational creativity, transforms the dimensions of the concept space itself. Looking at creativity in humans, it can be seen that a lot of human creativity falls into the second type, such as artists and musicians. However, by using transformational creativity, one can create unexpected and shocking results.

“Jape” is an AI specified in Binsted (1996) that falls into the first type of creativity defined by Boden (1998). This program is focused on generating humorous riddles using puns. “Jape” uses nine general sentences in order to generate jokes. An example of this is "What do you get when you cross X with Y?". Tests showed that people did not have too much difficulty in separating “Jape”’s jokes from human-made jokes, as the human jokes were often better. However, if the results of “Jape” were pruned to only contain the best jokes, the difference between the generated and human-made jokes began to disappear (Binsted, 1996).

When it comes to exploratory creativity, there are a lot of existing AI programs. Amongst others, (Boden, 1998) mentions the following: EMI(Experiments in Musical Intelligence), a program that creates new music in the styles of famous composers (Cope, 1991), and AARON, which produces both line-drawings and colour using specific styles (Cohen, 1995)(McCorduck, 1991).

There is not a lot of programs that can be called transformational compared to exploratory. However, some exist in the form of programs that use genetic algorithms (Boden, 1998). This leads into Boden’s conclusion, namely that there currently are two bottlenecks for further development. These are the requirement of domain-expertise and valuation of results. The latter being especially the case for transformational creativity(Boden, 1998).

3.4. Real Time NeuroEvolution of Augmenting Topologies (rtNEAT) and NERO

Stanley et al. developed the video game NeuroEvolving Robotic Operatives (NERO), a game using machine learning as its mechanics. The point in this game was to train AI agents to win against another team of trained agents. Each player had to design sequential tasks or scenarios that would gradually teach the agents to perform increasingly complex behaviour (Stanley et al., 2006).

In order to achieve these mechanics, Stanley et al. made an extension to NeuroEvolution of Augmenting Topologies (NEAT) called Real Time NEAT (rtNEAT). Due to the agents needing to evolve during training, and while the scenarios were changing, the system had to be able to make improvements on the fly. This new method, rtNEAT, continuously breeds two agents with high fitness at a specified tick rate. The offspring replaces the currently worst individual.

Using a pre-designed collection of scenarios, behaviour evolved quickly in NERO. It could be observed that agents were able to learn diverse skills and perform complex strategies that dominate simpler tactics. This makes a strong case that the speciation in NEAT promotes diversity in strategies, and does not kill off any individuals that has yet to optimize their behaviour (Stanley et al., 2006).

4. Architecture

In order to work towards the project goals, a system capable of evolving, testing and visualising agents was implemented, called *Creative Creature Behaviour (CreBe)*. This chapter contains the architectural structure and rationale for this project. First, the world in which the agent simulation takes place is described. Then, the implemented system and its algorithms are presented. Also included is a short description of the different configuration files used by the system. The system was implemented solely for this project.

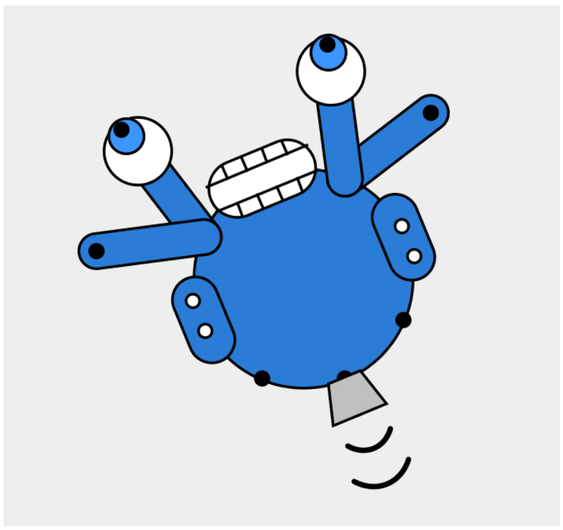


Figure 4.1.: A creature with a motor, two noses, two eyes and a mouth. Each eye is attached to a fork, which enables branching of entities. The black dots indicate empty attachment points, where other entities may be attached.

4.1. Agent world

In order to stimulate the evolution of intelligent behaviour, a virtual world is created where agents can move, roam and act freely. This world is mostly empty, with the exception of other agents and scenario specific entities, such as food and poison. One part of the research is to investigate different prerequisites for the emergence of creative and intelligent behaviour. For this, different world types with different goals can be experienced by the agents. These world types act as different scenarios when evaluating the performance of an agent. It is more likely that a multi-objective scenario will inspire the agents to behave creatively, which is why different scenarios and goals are designed. This pertains to the project goal of studying different environments and tasks in order to motivate for the emergence of creative behaviour.

This project models agents as animal-like entities, known as *creatures*. See figure 4.1. The fitness score of a creature is calculated by running a simulation of a specific world type scenario. These simulations are independent operations, meaning multiple creatures can be evaluated in parallel on multiprocessor systems.

4.1.1. Food world

This is the world that represents the most simple scenario. It contains several bits of food, and only one agent is present in the world at a time. The goal for the agent in this world is to eat as many food pieces as possible, within a predefined timeframe. Creatures in a food world can see and smell food using the eye and nose entities respectively, but needs a mouth entity in order to eat the food. The number of foods eaten is stored as the agent's fitness value. This scenario requires only very simple behaviour, and acts more as a benchmark and test scenario during the implementation of the system.

4.1.2. Poison world

The poison world scenario is very similar to the food world scenario. The only difference being that there are also bits of poison among the food scattered around. In addition to this, not all sensors can differentiate between the two. Due to this, the behaviour of the agents need to be a bit more complex. A creature sees food and poison as equal, but smells them differently. This constitutes a big advantage for creatures that evolve a nose entity, given that they are able to use this information in a useful way.

4.1.3. Arena world

In contrast to the other food and poison scenarios, this is a multi-agent environment, meaning more than one creature is present at a time. The main idea behind this world is that two different agents can fight against each other, develop strategies and incrementally improve their fighting skills. Using damage-inflicting entities such as the spike, the goal is to kill the other creature. In this environment, the energy level of a creature doubles as health, or Hitpoints (HP). A creature dies when its HP reaches zero. The opponent in an arena world can be changed to a pre-designed creature with pre-determined behaviour to train the current agent to win against certain strategies. Due to the multi-agent property of this scenario, the idea is that self-improvement will produce creatures with richer and more complex behaviour over time.

4.2. Algorithms and AI-methodologies

This section aims to describe the different algorithms used and implemented, the system architecture and overall architectural design.

4.2.1. Physical agent evolution

In order to evolve the physical appearance of agents, that is, the collection of entities[1] that makes up a creature, a custom Evolutionary Algorithm (EA) was implemented. The algorithm uses a predefined set of entities, each with different purpose, functionality and mutable properties. See figure 4.2 and table 4.1 for an overview of the different entity types available during evolution. Some entities (body, arm and fork) have attachment points where other entities of any type can attach. The evolutionary task is to try different combinations of entities together, while also mutating their properties slightly in order to cover a bigger search space. All creatures have a body as the root entity, while other body parts are attached to this. This makes up a tree of entities, making the genotype of this evolution process a tree representation. The only restriction on physical evolution is that entities may only attach to other entities with a free attachment point (see figure 4.2).

Each entity is equipped with an energy upkeep cost, which is calculated from mutable properties of the specific entity. For example, an eye with a big Field Of View (FOV) will have higher associated upkeep than an eye with normal FOV. The relation between upkeep and mutation of properties is linear, meaning an eye with double FOV and double max range, will have double entity upkeep. The upkeep system is intended to place a cost on evolving numerous entities, to limit the number of entities that a creature can sustain. A creature that is able to replenish energy quickly will be able to support more and “better” entities, which in turn enables for further performance improvements. During creature simulation, that is, when a creature is placed in a world, the total upkeep is subtracted from the creature energy level at each timestep[2]. In the food world scenario, creatures that are not able to gather enough food to support their entity upkeep will die sooner and therefore receive a lower fitness value.

[1]Body parts.
[2]Tick.

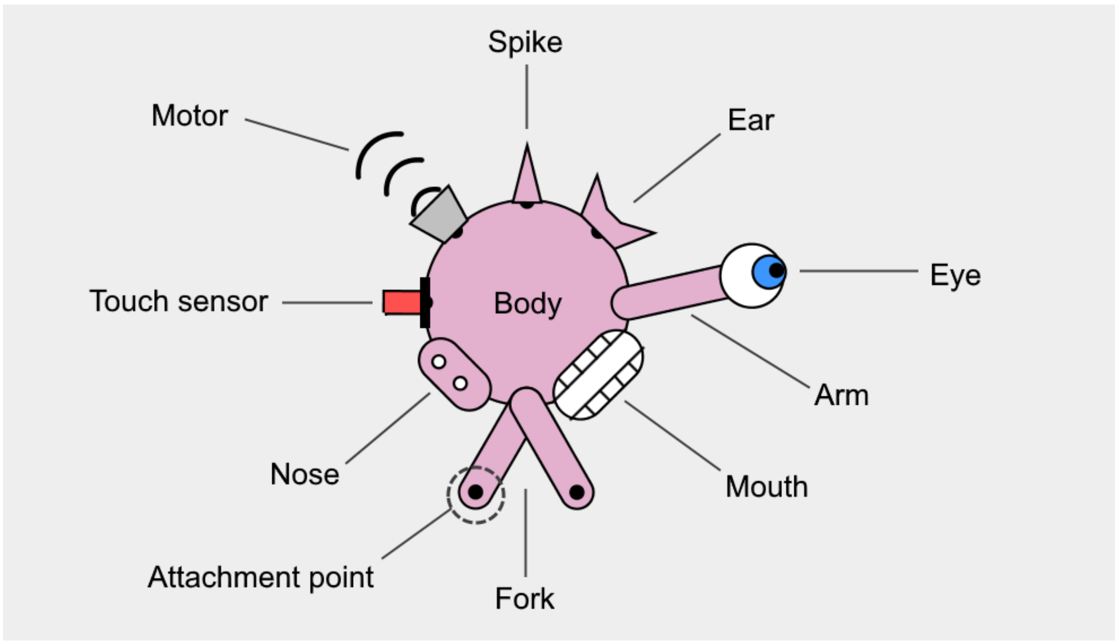


Figure 4.2.: An overview of all the different entity types available in the system. See table 4.1 for explanations. Entities attach at *attachment points*, as illustrated in the figure.

Name	**Purpose/functionality**	**Mutable properties**	**Type**
Arm	An arm with rotation. Can attach one other entity at the end.	Arm length, maximum rotation angle and rotation speed.	Actuator
Body	Acts as the base entity for the agent. Other entities may be attached to the body.	Body size and creature rotation speed.	Actuator & sensor
Ear	Can "hear" other agents within range.	Hearing range.	Sensor
Eye	Can see objects in the world, such as food and other creatures.	Visual range and FOV.	Sensor
Fork	Splits an entity attachment point to two attachment points.	Fork branch length and angle.	*None*
Motor	Creates a force to propel the agent.	Motor strength.	Actuator
Mouth	Can eat food and poison.	*None*	Sensor
Nose	Can "smell" food and poison.	Smelling range.	Sensor
Spike	Used to inflict damage based on the speed of the entity to other agents.	Spike length and damage.	*None*
Touch sensor	Sense touch from other agents.	*None*	Sensor

Table 4.1.: An overview and explanation of the different entity types available during evolution.

The fitness evaluation procedure associated with the physical agent evolution is a key research point in this project, as it is highly responsible for the evolutionary outcome. A poorly designed fitness function may not stimulate for the evolution of intelligent and creative behaviour. However, physical agent evolution is not responsible for the evolution of behaviour and action inference. This is handled by a second evolutionary system. The fitness evaluation procedure of physical evolution is therefore the process of running an entire behavioural evolution.

4.2.2. Behaviour evolution

A second evolutionary algorithm is responsible for the cognitive and behavioural evolution of an agent. This algorithm seeks to construct the action inference systems, referred to as *brain modules*. These modules can be injected into any creature whose input and output variables are compatible. That is, the number of input and output variables of both the creature and the brain module must match. Sensors are considered input variables, while actuators are considered output variables (table 4.1). Figure 4.3 illustrates the high level essentials of the system.

The system features three different types of brain modules; Artificial Neural Network (ANN), fuzzy logic and NeuroEvolution of Augmenting Topologies (NEAT).

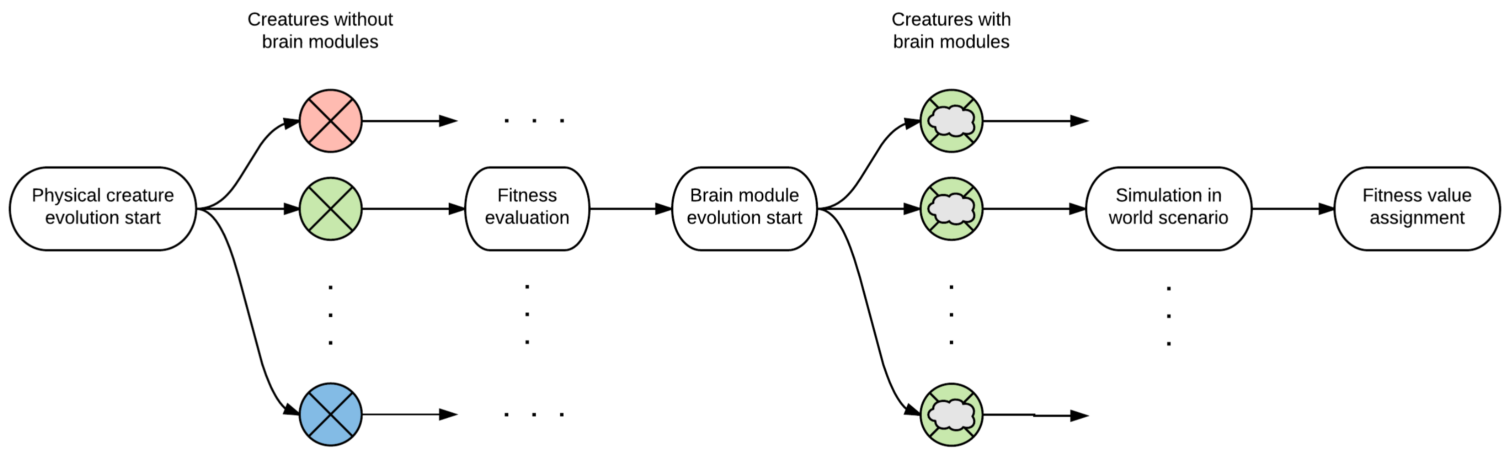
ANN-module

For the purpose of this project, a minimalistic and simple Continuous Time Recurrent Neural Network (CTRNN)(see section 2.2.2) system was implemented. The produced networks can have any layer configuration, input and output size. It also features functionality to retrieve and update connection weights, as well as taus (τ), gains (g) and bias (ϑ) terms of every neuron. This is to enable adjustments of network parameters during evolution in order to exhibit learning capabilities of the system as a whole. The interface of a CTRNN consists of a vector input and output.

The aim of the ANN brain module is to evolve weights and accompanying parameters to be used in a CTRNN. Because this EA is evolving network weights, the genotype is a real-valued representation. The layer configuration of the network is left to the human operator to decide, and is specified in the system configuration files. Other properties, such as input layer size and output layer size, are implicitly derived from the creature that this brain module is created for. Each sensor entity constitutes an input node, while every actuator represent an output node of the evolved network.

Genetic mutation of an ANN-module consists of adding to every parameter, a small, random value drawn from a zero-mean normal distribution with configurable standard deviation, usually around 0.1. This is done for every weight and parameter of the CTRNN-genotype. Crossover is implemented as drawing each parameter from either of the two genotypes in question, with equal probability.

Figure 4.3.: A high level overview of the EA workflow of the system. The figure only illustrates a single generation of the physical entity evolution. Notice that for every phenotype (creature) in the physical EA, a complete run of a brain module EA is necessary in order to assign a fitness value.


Creatures without brain modules
Creatures with brain modules
Physical creature evolution start
Fitness evaluation
Brain module evolution start
Simulation in world scenario
Fitness value assignment

Fuzzy-module

The second type of brain-module is based on fuzzy logic, using zero-order Sugeno-style inference (see section 2.4.4). Upon initialization, the fuzzy-module genotype is filled with random rules based on the statically defined action fuzzy sets[3]. For every action of every present actuator (e.g. arm rotation or body movement), 1-3 randomly generated fuzzy rules are added to the rulebase. This is to ensure that every action has at least one reference to a rule, in order for all actions to have a chance to fire. During rule generation, the sensor variables, sensor values, the number of clauses to generate as well as rule clause operation are selected randomly. Mixing of rule clause operations are not permitted within a single rule, meaning the whole rule consists of either a series of AND or OR clauses. An example of a randomly generated rule with two clauses:

IF Distance IS Close AND HP IS Low THEN Move is Forward.

Or more generally:

IF *sensor variable* IS *sensor value* AND/OR ...
THEN *actuator variable* IS *action*

Mutation of a fuzzy-module genotype involves iterating through the rulebase, bumping *sensor value* or *action* fields either up or down for every clause of the rule in question. For example, this may turn "HP IS Moderate" into "HP IS High". The probability of this mutation happening is configurable. There is also a chance rules are removed or added, by the same procedure as the initial rules were generated. The crossover operator between two fuzzy genotypes selects a rule from either the first or second genotype with equal probability.

NEAT-module

Lastly, the third brain-module uses the NEAT method (see section 2.3). This is also an ANN, however, unlike the CTRNN implementation, this cannot be structured in layers. NEAT is about evolving the topology of the network, something which creates a need for a more flexible structure. Here, each neuron contains a list of its connections to other neurons that affect its activation.

When a NEAT-module genotype is initialized, input neurons for each of the creature's sensor entities, and output neurons for each of the creature's actuators are created. Every output neuron gets a connection to each input neuron, i.e. the network becomes like a fully connected network with no hidden layers. This is the minimal structure that the NEAT implementation uses as a basis for evolution.

[3]Consult appendix A, page 91 for the definition of these sets.

In every NEAT-genotype the data for the network is stored as a list of unique neurons and another list for unique connections. During mutation a neuron or connection can be added, and the weights of existing connections can be changed. When adding a neuron, an existing connection is first selected that the new neuron will split. This creates two new connections and the old one is tagged as disabled. Adding a new connection requires more work. First all of the possible new connections are found, and then one of them is selected randomly.

All of the different connections are assigned a number from a global counter. This becomes the innovation number for the connection, and is necessary during the mating process.

4.2.3. Run mode

The system supports two different run modes; designed and full (figure 4.4). In the designed mode, the physical appearance of a creature is predefined in a configuration file specified when running the system. In this case, only the behaviour evolution procedure (either CTRNN, fuzzy or NEAT) is performed in order to produce the behaviour of the creature. In the full run mode, physical creature appearance is evolved through physical agent evolution, detailed in section 4.2.1. The designed mode allows for significantly larger population sizes and more generations for behaviour evolution, because only one EA has to be performed.

4.3. System configuration

The system can be configured by different configuration files, one for the physical entity EA, one for each of the brain-module subsystems and lastly a general configuration for the system and neural network configuration. Consult appendix B for an example of how these configuration files are structured.

- Physical entity EA configuration:
  Contains parameters that apply to the entity evolution EA; the population size, max fitness/generations, selection strategies, mutation- and crossover rates.

- Brain-module configuration:
  These configuration files apply to the brain-module subsystems, which is responsible for the evolution of behaviour. As they also are EA configuration files, they contain the same parameters as the physical entity EA configuration file.

- General system and ANN configuration:
  This configuration file contains general options for the system, including entity mutation factors, world options such as food spawn rate, the simulation length for evaluation, and ANN layer configuration.

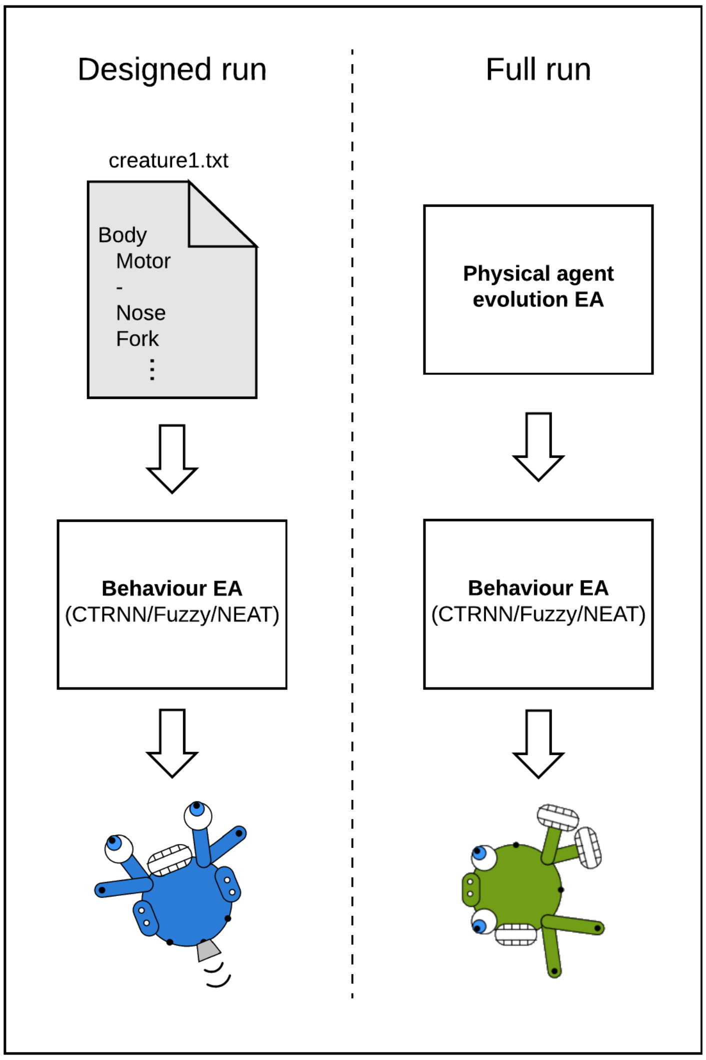

Designed run
Full run
creature1.txt
Body
Motor
-
Nose
Fork
⋮
Physical agent evolution EA
Behaviour EA (CTRNN/Fuzzy/NEAT)
Behaviour EA (CTRNN/Fuzzy/NEAT)

Figure 4.4.: Difference between a designed run and a full run. In the designed case, a .txt file is manually devised in order to define the physical entity layout, while a behavioural EA will create the appropriate brain-module. In the full-run case, physical creature layout is evolved using the physical agent EA.

4.4. System limitations

There are some clear and some more hidden limitations of the implemented system. One obvious is the computational performance limitation. The system relies on utilization of multiple processors in order to complete the simulations in a reasonable amount of time. This applies especially to the full-run computation, where two EAs are nested together to produce both creature entities and behaviour. The long execution times are particularly noticeable when running a high number of generation with a large population. This may put a practical limit on these configuration parameters, potentially limiting the behaviour of the evolved creatures.

Another limitation of the system, which is less exposed, is the fact that behaviour is not inherited and carried over to the next physical EA generation. This limitation may restrict the behavioural performance of the creatures, because the behaviour-EA is reset every time an entity is changed, added or removed. However, this limitation only applies to the full-run mode, where entities actually undergo evolution.

5. Experiments and Results

This chapter details all of the experiments that were designed and performed during the thesis. Along with experimental setup, parameters and results, a short discussion of the results it also present.

5.1. Experimental plan

In order to test the performance, evolvability and creative capabilities of the system, a number of experiments was designed using the world scenarios discussed in section 4.1, and recapped in table 5.2. For each scenario, four different creatures were manually designed before they were to be optimized for their given task. The experiments performed for each scenario in the designed runs, were as follows:

For every designed creature, 10 runs of 100 generations with a population size of 200 were performed. This procedure was repeated 3 times, one for each brain module; Continuous Time Recurrent Neural Network (CTRNN), fuzzy logic, and NeuroEvolution of Augmenting Topologies (NEAT). The most relevant configuration parameters are presented in table 5.1. For the complete configuration, consult appendix B.2 (page 95).

Table 5.1.: The most relevant configuration options used in the designed creature experiments.

Parameter	**Value**
Population size	200
Generations	100
Elitism	1 (0 for NEAT)
Crossover type	Uniform
Adult selection	Generational mixing
Parent selection	Fitness proportionate

In addition, experiments were performed in order to test the full-evolve capability of the system. In this case, no creatures were manually designed, but evolved using physical agent evolution as explained in section 4.2.1. These tests are expected to show the highest level of creativity and novelty, as both the physical creature layout and behaviour is left for the system to devise. In these full-run experiments, the system performed four runs of 50 generations with a population size of 16, as stated in table 5.3. This was again repeated for each scenario. In order to produce brain-modules for these creatures,

Table 5.2.: A recap of the different world scenarios and behaviour brain-modules in the system.

Scenarios	**Brain-modules**
Food world	CTRNN
Poison world	Fuzzy logic
Arena world	NEAT

a behaviour evolution was performed with a population size of 20 for 20 generations. Because of this nested EA-operation, the number of generations and population sizes had to be reduced in order to keep the run-times on a manageable level. Complete configuration for the full-run experiments are available in appendix B.3 (page 96).

Table 5.3.: The most relevant configuration options used in the full evolution experiments.

Parameter	**Physical evolution**	**Behaviour evolution**
Population size	16	20
Generations	50	20
Elitism	0	1 (0 for NEAT)
Crossover type	Uniform	Uniform
Adult selection	Generational mixing	Generational mixing
Parent selection	Fitness proportionate	Fitness proportionate

The following sections contain the experimental parameters and results, presented one scenario at a time. In order to get a sense of the fitness landscape of an evolution, the maximum fitness is presented as a plot for each of the brain-modules, along with standard error at each generation, which is calculated using equation 5.1.

$$SE = \frac{SD}{\sqrt{n}} \tag{5.1}$$

where SE is the standard error, SD is the standard deviation of the recorded maximum fitness values in a generation, and n is the number of runs, which is 10 for both the designed and full-run experiments.

5.2. Food world scenario

In the food world, the objective is to eat as many food pieces as possible while the creature is alive, or within a predefined maximum timeframe. Only one creature is present in the world during simulation. The fitness function for the food world is given in equation 5.2.

$$f(c) = n_{c,foods} \tag{5.2}$$

where c is the creature in question, and $n_{c,foods}$ is the total number of foods consumed during the simulation. The maximum fitness is 100, because 100 food pieces spawn in the world.

5.2.1. Designed food world creatures

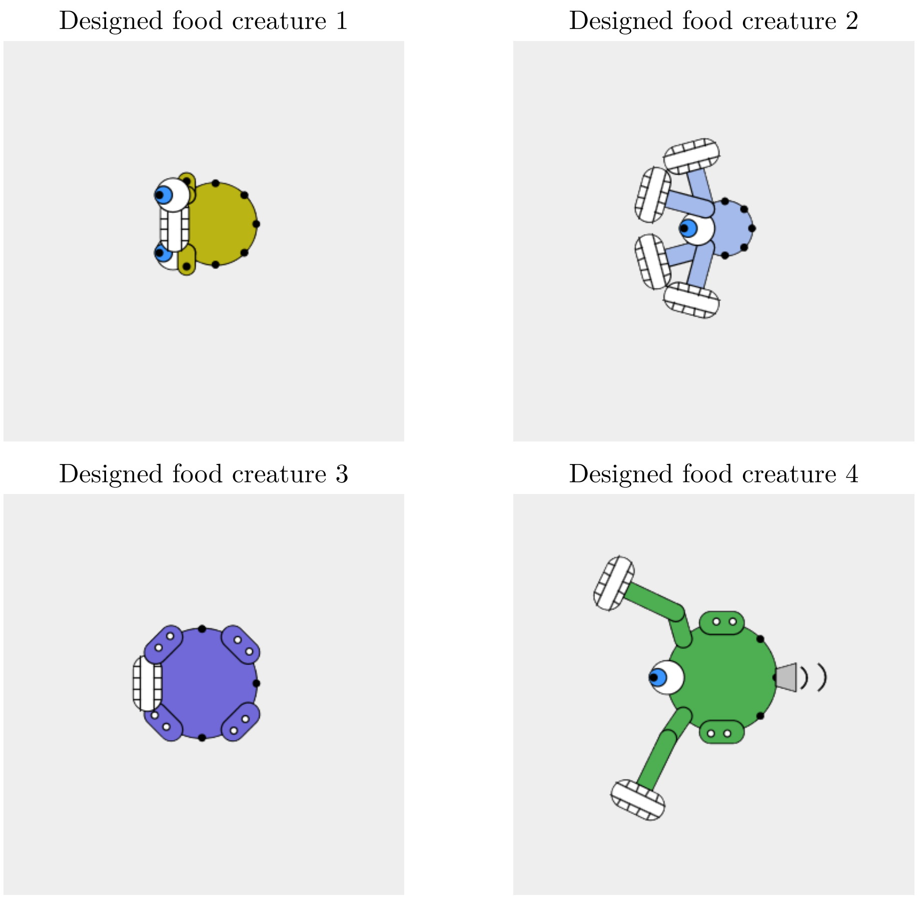


Figure 5.1.: Designed creatures used in the food world scenario.

Food creature 1

The first designed creature in the food scenario is a very simple one, equipped with two eyes and a mouth (figure 5.2).

The average maximum fitness at each generation is presented in figure 5.3, along with standard error. The plot shows that all three behaviour systems more or less stabilized their maximum fitness after 100 generations, with CTRNN marginally achieving the highest fitness and fuzzy logic by far the lowest. It also presents the best evolvability of the three, showing a steady increase throughout the run. The highest possible fitness in the food world is 100.

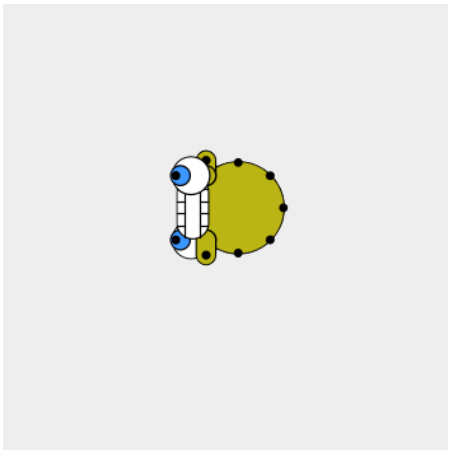


Figure 5.2.: Food world
Designed creature 1

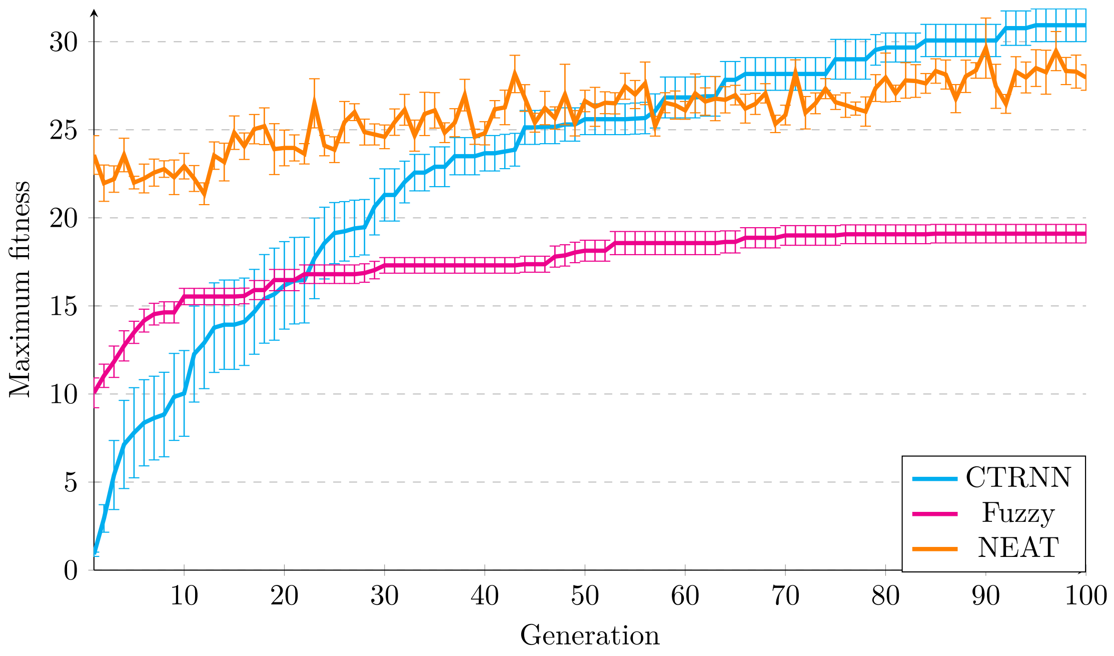


Figure 5.3.: Food world scenario results for designed creature 1.

Food creature 2

The second designed creature in the food scenario has four mouth entities, producing a larger area for collecting food. However, the creature was not able to utilize this advantage, as it usually used its central mouths to eat. All brain-modules behaved more or less equally, with fuzzy logic performing somewhat behind the two others.

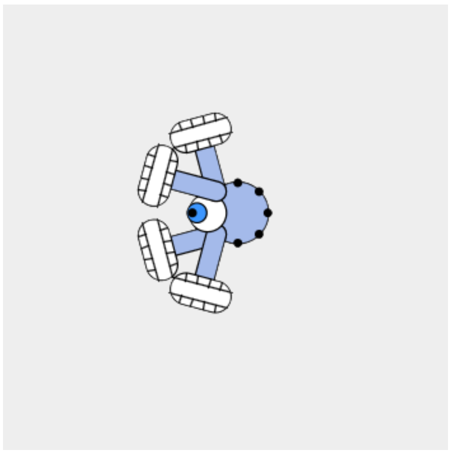


Figure 5.4.: Food world
Designed creature 2

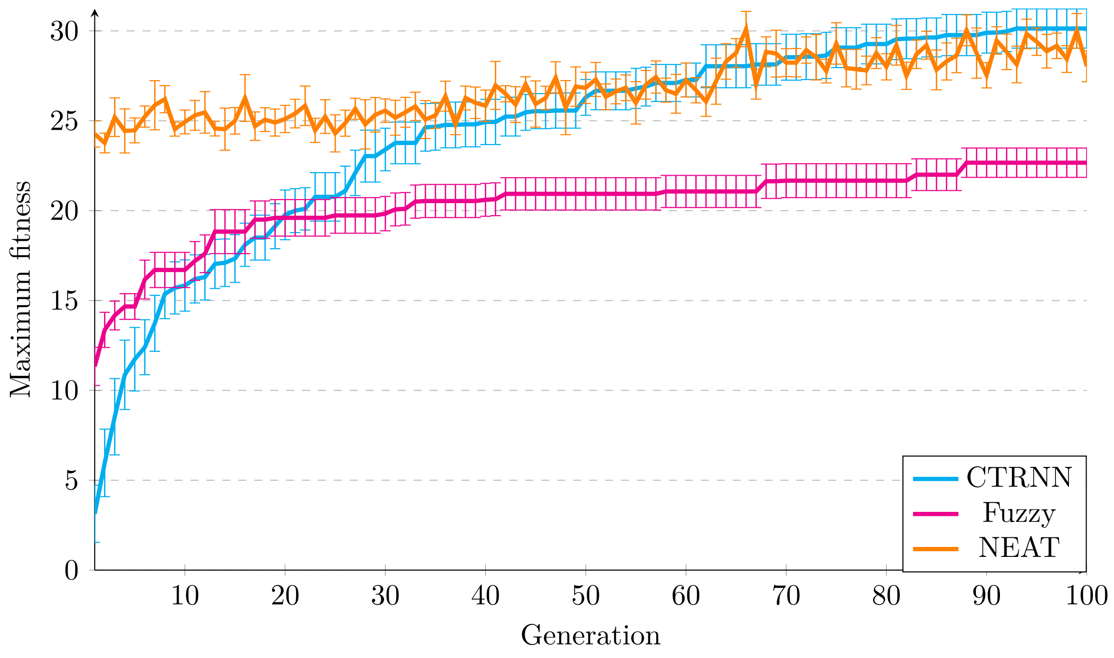


Figure 5.5.: Food world scenario results for designed creature 2.

Food creature 3

The third designed creature in the food scenario has four nose entities and no eyes, in order to investigate agent performance with limited sensory input. The creature is able to collect food using only its many noses, however, not as efficiently as the other designed creatures in the food scenario experiment. The fuzzy logic behaviour was able to improve its performance by decreasing its speed, making the turning radius smaller and more likely to intercept a food piece. The NEAT based creature was able to eat by spinning rapidly when encountering food, making it likely that the mouth would intersect it. We also see a higher than usual standard error, probably due to limitations in only using noses as sensor input.

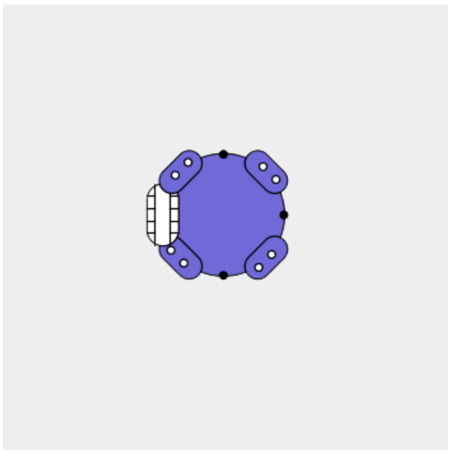


Figure 5.6.: Food world Designed creature 3

The discrete sensor value of a nose entity[1] may appeal to fuzzy logic because of the simple rule-base. With this particular designed creature, the number of sensor states the creature can perceive is relatively low, which can be effectively captured by fuzzy rules.

[1]Nose entity will only report the value 0 or 1.

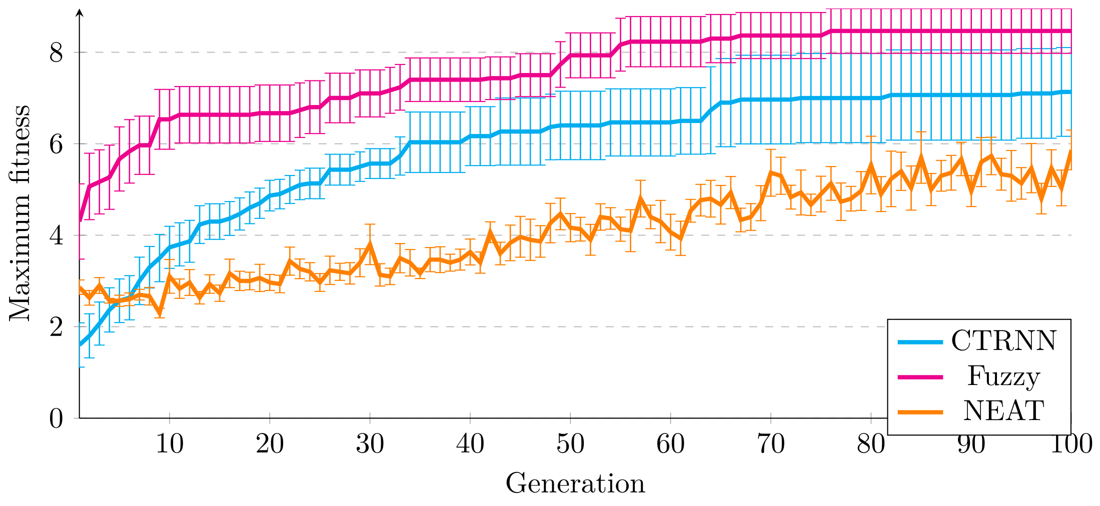


Figure 5.7.: Food world scenario results for designed creature 3.

Food creature 4

The fourth designed creature in the food scenario has mouths attached to arms, in order to investigate complex arm behaviour. Only the NEAT based creature showed remote signs of this, moving one of its arms in front of the eye when observing a food piece, allowing for easier navigation to the food.

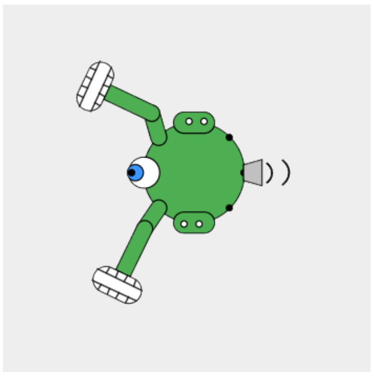


Figure 5.8.: Food world Designed creature 4

The sudden rise in maximum fitness for CTRNN at the end of the run may be explained by the evolution escaping a suboptimal solution. Figure 5.10 shows that three out of ten individual runs experience a sudden increase in fitness at generation 95, 96 and 97 respectively. The other seven runs do not experience the same increase in fitness at this particular point, which suggest that the somewhat unexpected jump in fitness is just a product of random improvements in three out of ten runs. Figure 5.11 shows three re-runs of the same experiment, which further suggest that the fitness jump is just a random coincidence. In fact, it appears the original CTRNN experiment performed poorly, and that the sudden jump is just a correction to more “normal” levels.

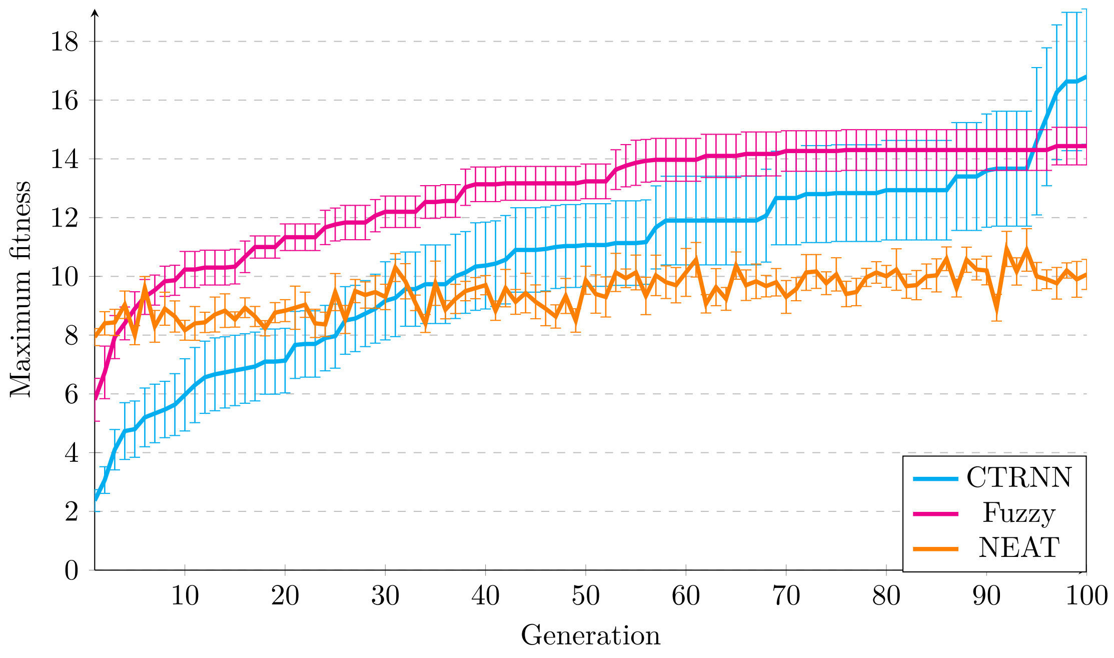


Figure 5.9.: Food world scenario results for designed creature 4.

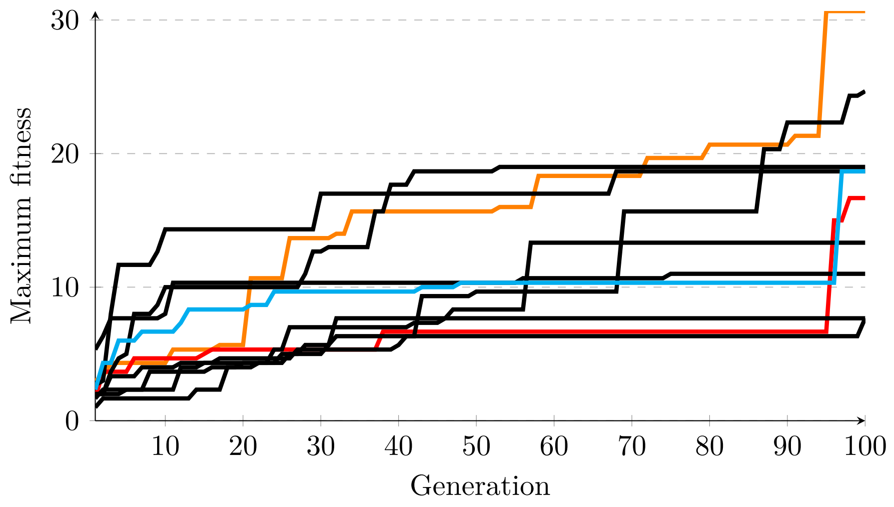


Figure 5.10.: Plot showing the maximum fitness of the individual runs of food creature 4-CTRNN experiment. Notice the coloured plots highlighting the three irregular runs. The other seven runs are painted in black.

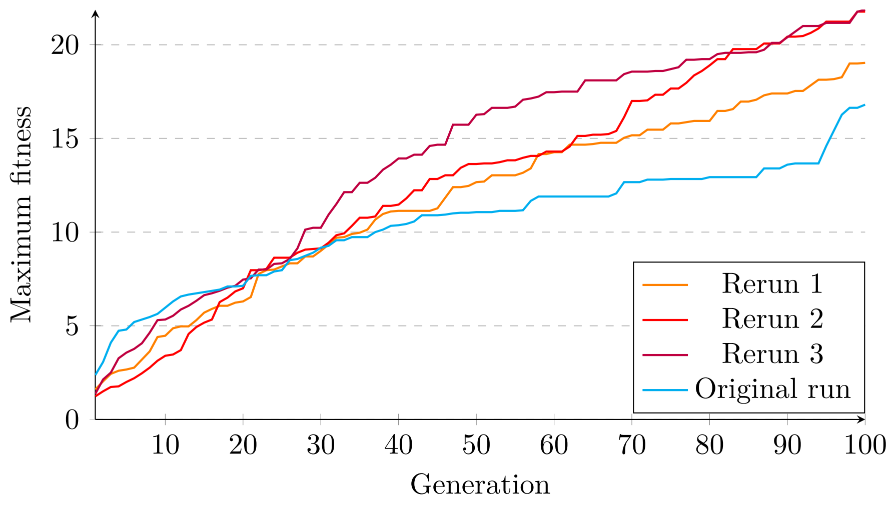


Figure 5.11.: Plot showing maximum fitness of three re-runs of the food creature 4-CTRNN experiment. The blue line shows the original CTRNN experiment, as presented in figure 5.9. The original run consistently performs lesser in the generation range 40-90, but manage to escape this mediocre fitness level towards the end of the run.

5.2.2. Fully evolved food creatures

In the full-run mode, both physical features and behaviour is subject to evolution, as explained in section 4.2.3. The fitness plots in figure 5.12 show a steady improvement in maximum fitness for all three behaviour models. Notice that elitism is disabled in this experiment, which explains why the maximum fitness may slightly decline in local areas throughout the run. All behaviour models performed acceptable in terms of number of foods eaten, but NEAT showed the greatest potential after 50 generations. However, Fuzzy logic is able to produce the most consistent results, with the lowest standard error.

The evolved creatures in figure 5.13, 5.14 and 5.15 show that multiple mouths is an important factor in the food world scenario. This will increase the collection area for food, and thereby increase the possibility for capturing a food piece. Additionally, three of the creatures evolved a bigger body, probably in order to spread out the mouth entities for an even bigger collective area. All four creatures also evolved an eye entity with increased Field Of View (FOV) and maximum range, in order to easier locate food. Some of the creatures also evolved arms with eyes attached, but were not able to utilize this combination in a particularly intelligent or creative way.

The observed behaviour is not considered very creative. The food world is the simplest of all scenarios, with all three brain-modules behaving almost identical. The exception is CTRNN, which displays some signs of exploration when food become scarce.

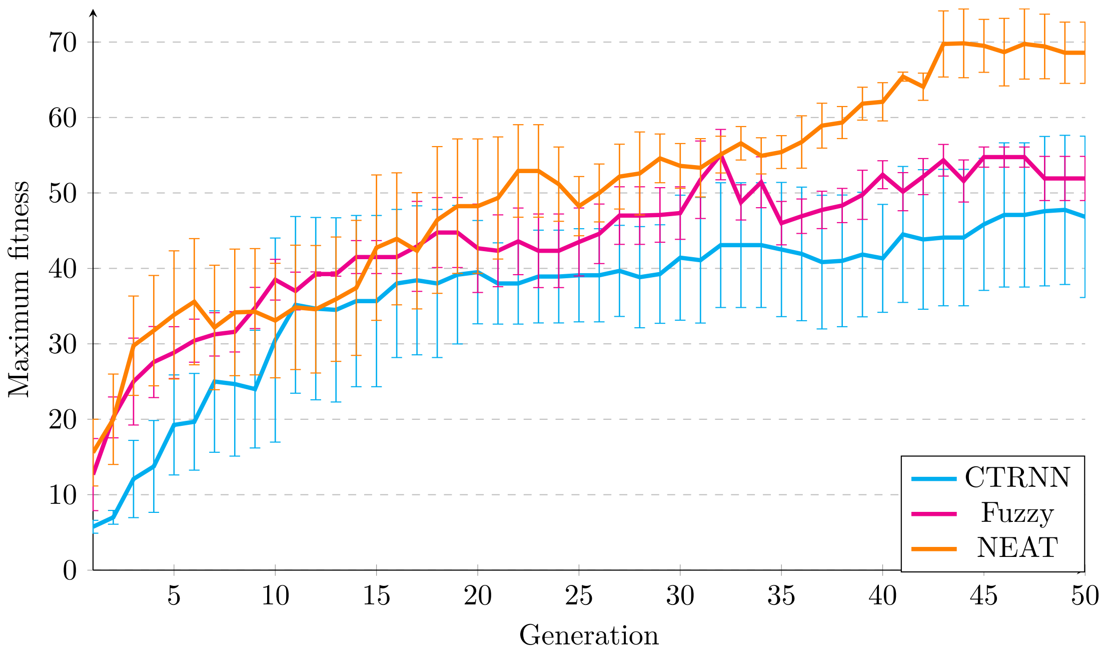


Figure 5.12.: Food world scenario for the evolved creatures

CTRNN

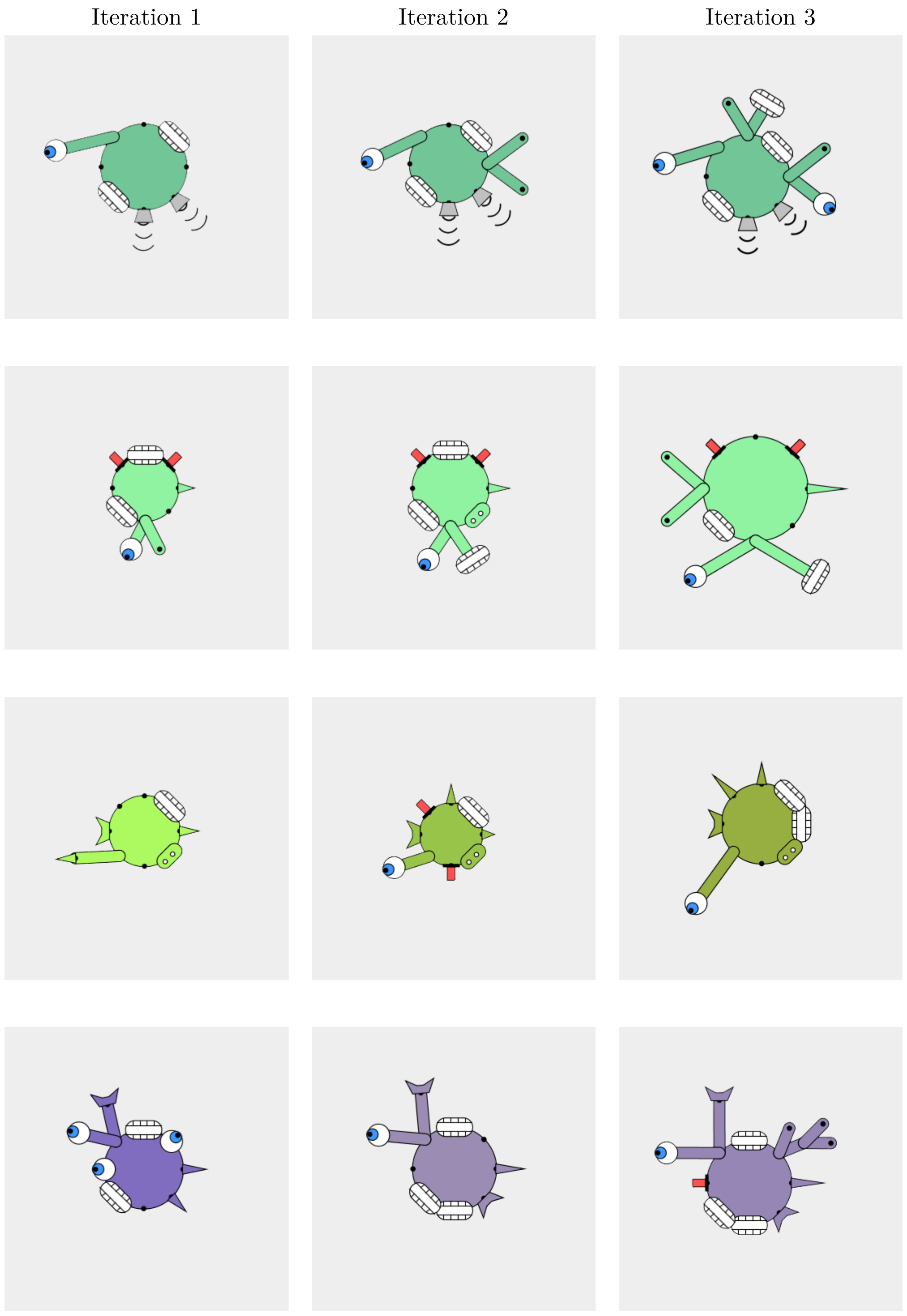


Figure 5.13.: The evolved creatures from the food world scenario runs with CTRNN as behaviour module.

Fuzzy Logic

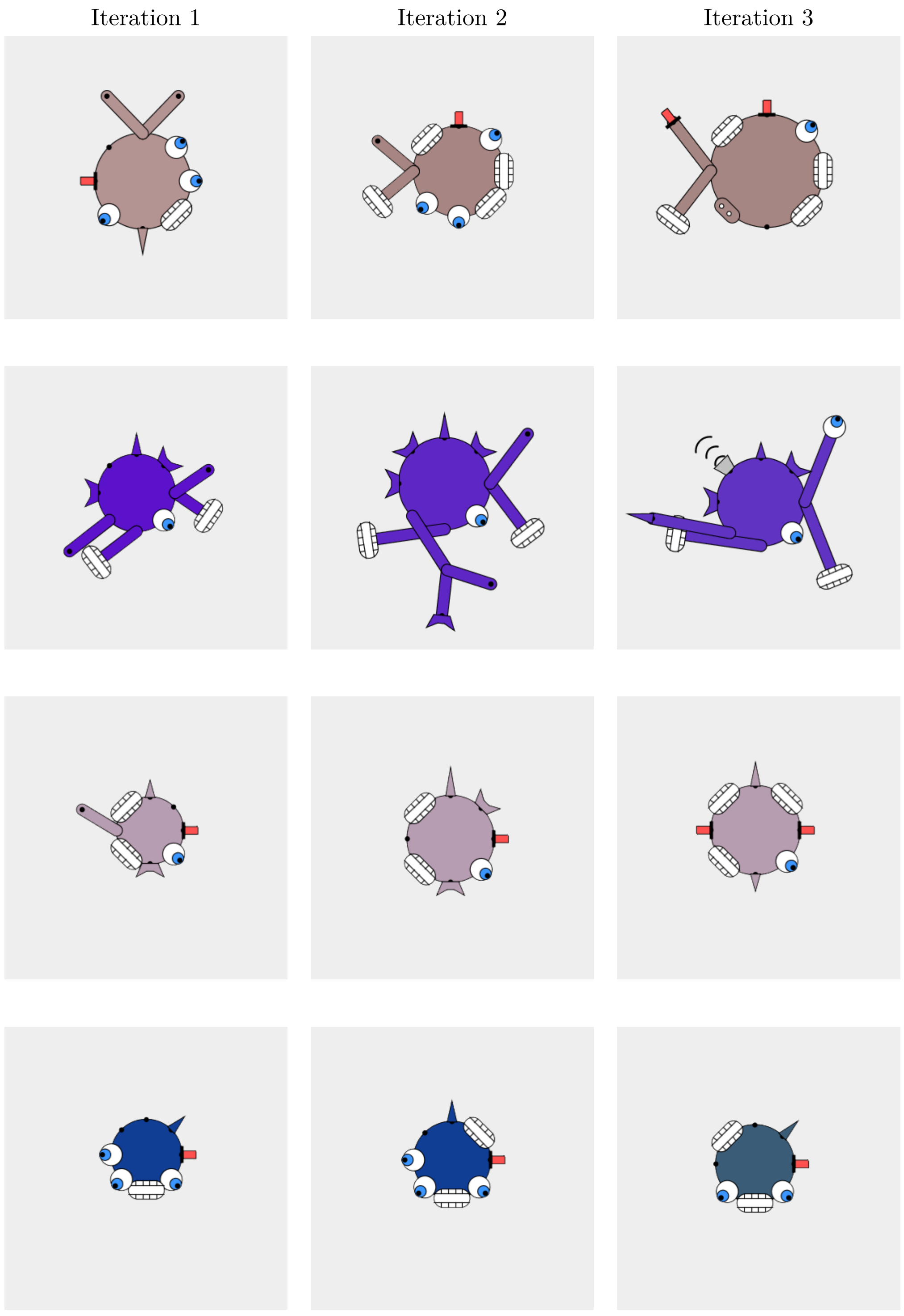


Figure 5.14.: The evolved creatures from the food world scenario runs with fuzzy logic as behaviour module.

NEAT

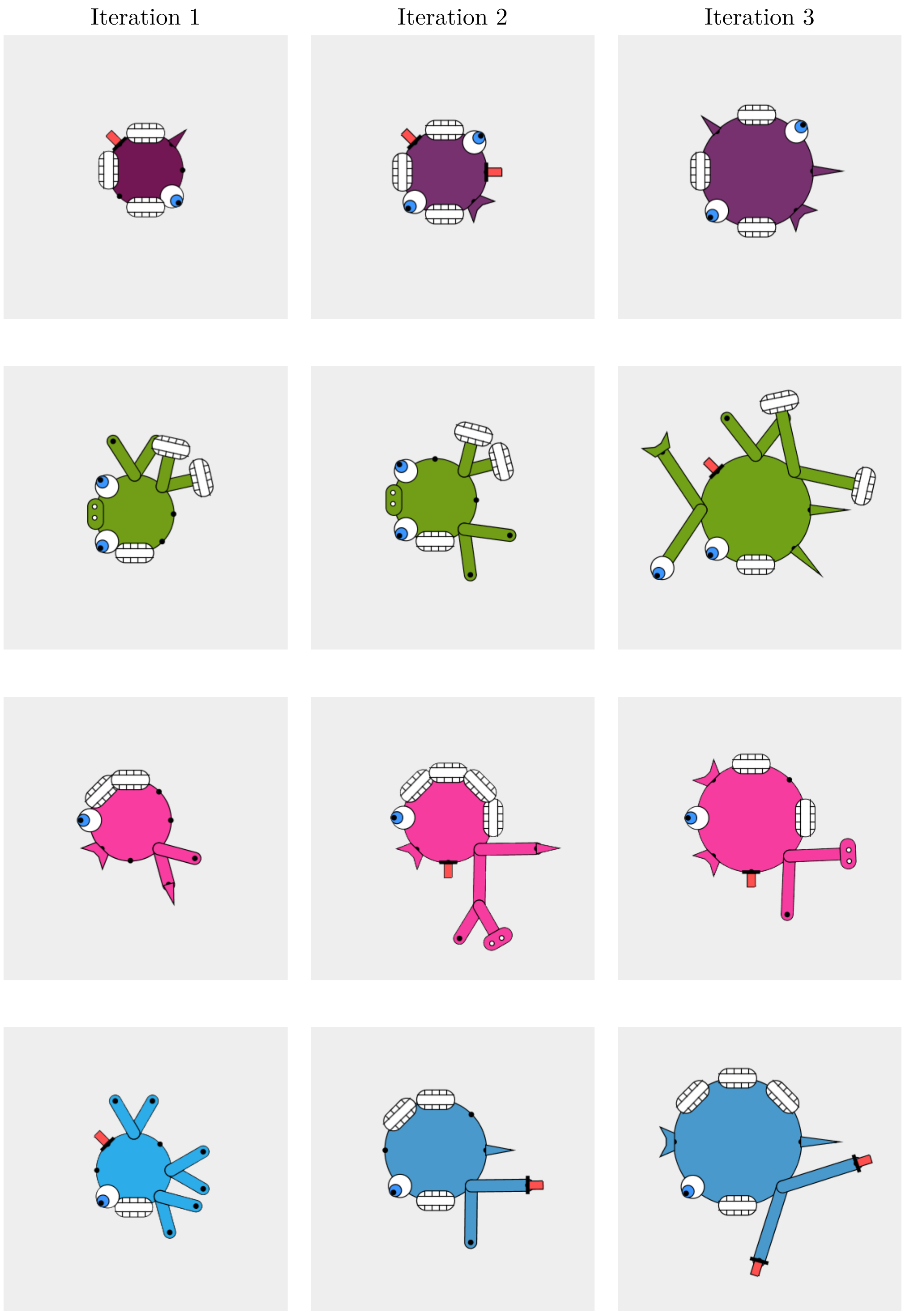


Figure 5.15.: The evolved creatures from the food world scenario runs with NEAT as behaviour module.

5.3. Poison world scenario

In the poison world, much like the food world, the objective is to eat as many food pieces as possible. However, in this scenario there also exist poisonous food, that can be identified by smell. Eating a piece of poison will subtract as much as eating a piece of food will add to the fitness of the creature. Also here, there is only one creature present in the world during simulation. The fitness function for the food world is given in equation 5.3.

$$f(c) = n_{c,foods} - n_{c,poison} \tag{5.3}$$

where c is the creature in question, $n_{c,foods}$ is the total number of foods consumed during the simulation, and $n_{c,poison}$ is the total number of poison consumed during the simulation. The maximum fitness is 50, as there are 50 pieces of food and 50 pieces of poison.

5.3.1. Designed poison world creatures

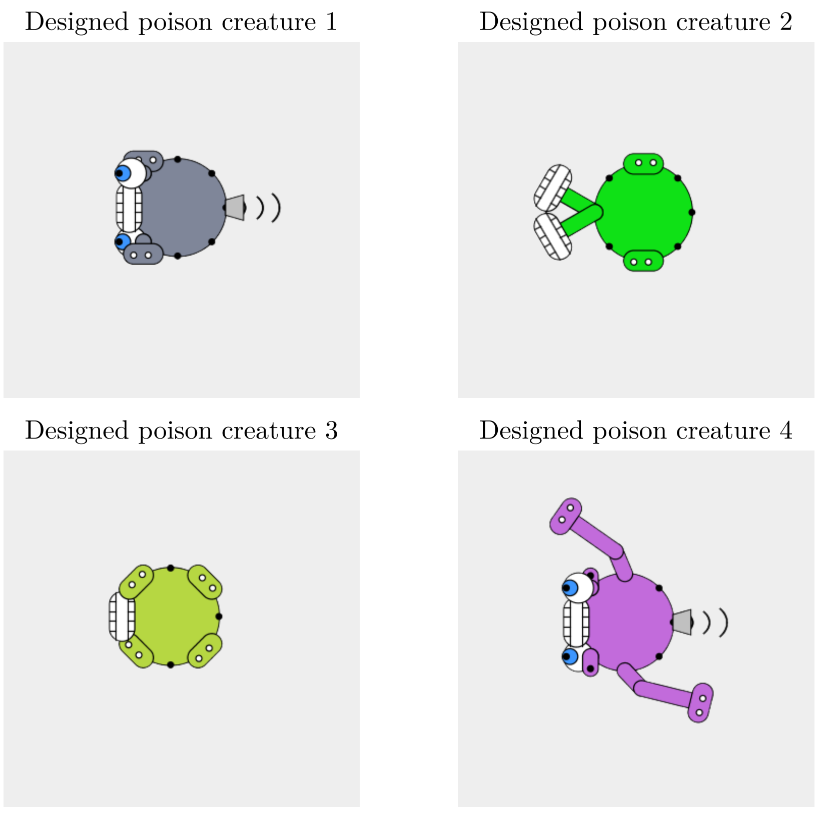

Designed poison creature 1
Designed poison creature 2
Designed poison creature 3
Designed poison creature 4

Figure 5.16.: Designed creatures used in the poison world scenario.

Poison creature 1

The first designed creature in the poison scenario is almost identical to the first food creature, with the exception of added noses on each side. This is to enable the detection of poison, because the eye entity is unable to differentiate poison from food.

This creature manages to avoid poison and steer away from it. However it sometimes steers away too late and eats the poison anyway. It also does not trust its own eyes, resulting in less food eaten. All three brain modules arrived at approximately the same behaviour.

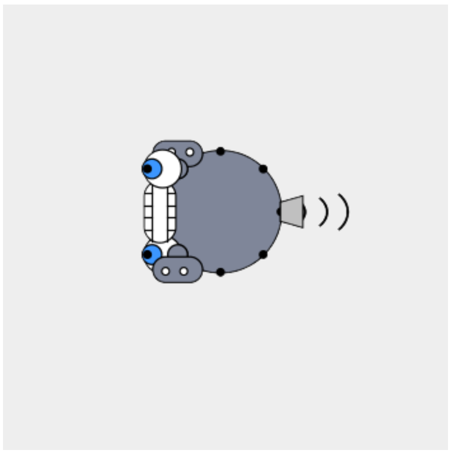


Figure 5.17.: Poison world Designed creature 1

In the results of the poison world simulations, the creature did not manage to evolve behaviour complex enough to have a good solution to the scenario. The data shows that all three brain modules performed poorly in this task. CTRNN shows the best evolvability by a small degree, just beating NEAT.

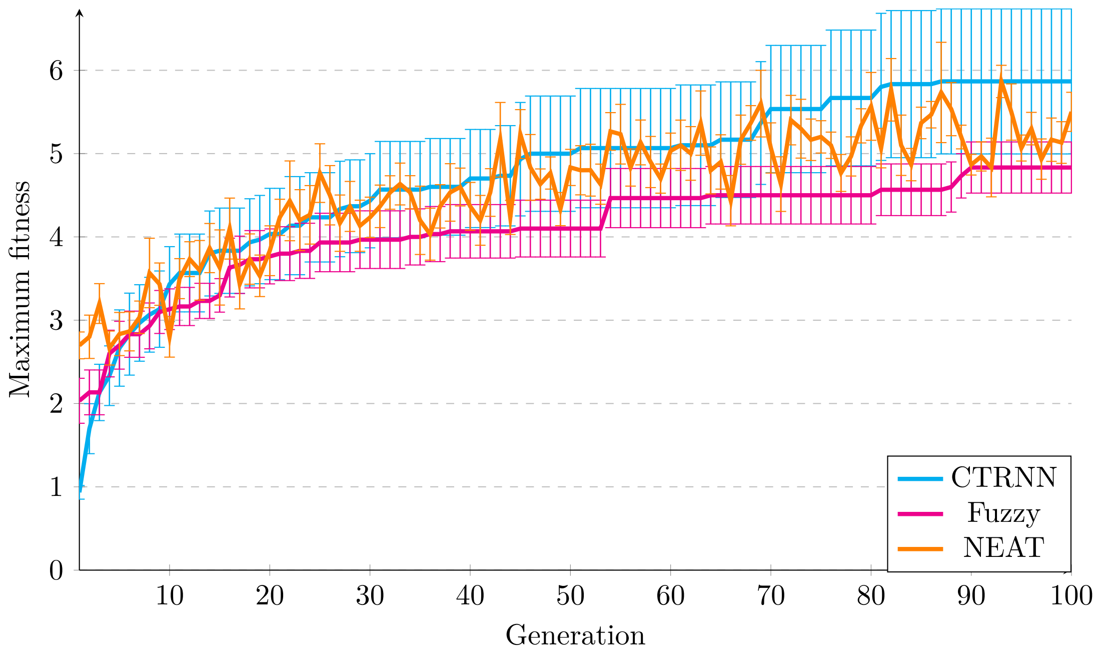


Figure 5.18.: Poison world scenario results for designed creature 1.

Poison creature 2

The second designed creature in the poison scenario is equipped with two mouths and two noses. The idea is that it should be able to use its two noses to know where food is and separate out the poison.

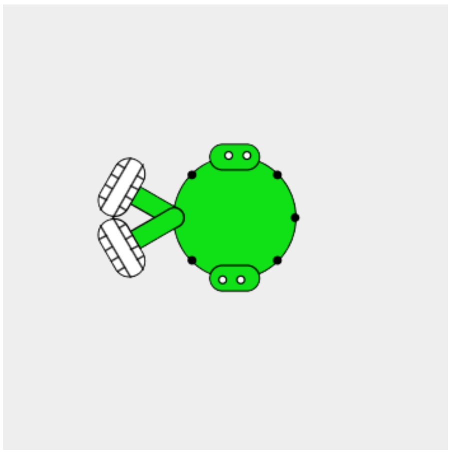


Figure 5.19.: Poison world
Designed creature 2

This creature does not evolve behaviour good enough to solve the task with any of the three brain modules. When observing the different modules, it is difficult to see any reason as to why one performs better than the others. None of them manages to use their senses well.

Also, as with the first poison creature, the plot shows poor performance. However, there is a visible difference between the brain modules. Both CTRNN and fuzzy logic show at least some evolvability, but NEAT does not manage to evolve anywhere from where it started.

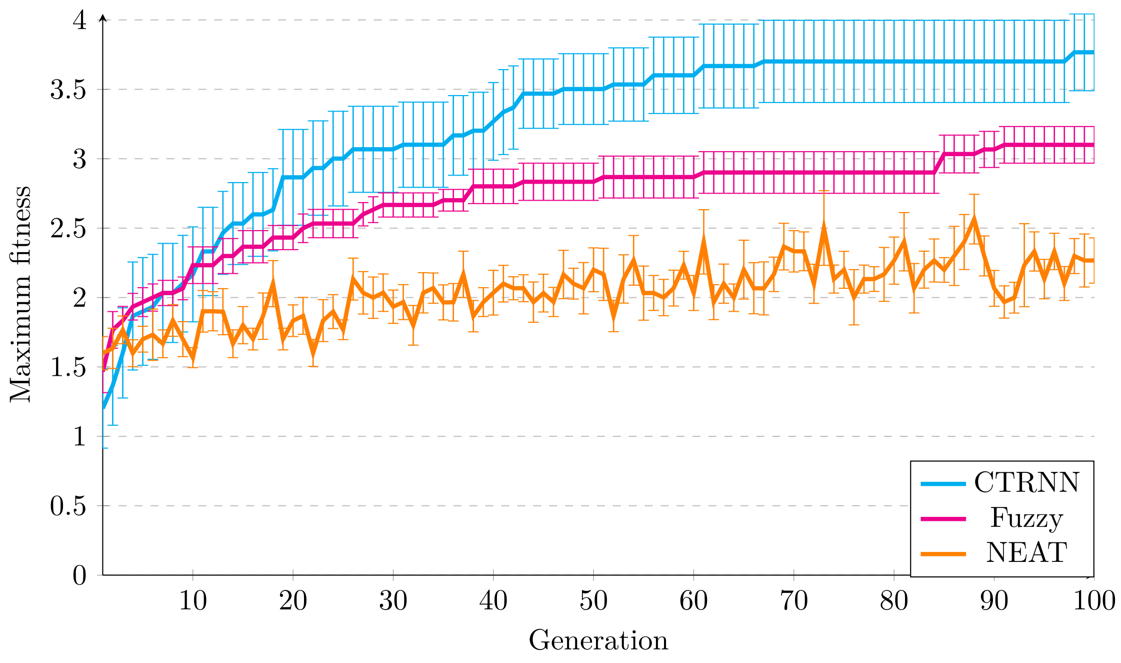


Figure 5.20.: Poison world scenario results for designed creature 2.

Poison creature 3

The third designed creature in the poison scenario is equal to the third used in the designed food scenario run, with four noses as its only sensory input. The performance of CTRNN and NEAT is very poor in this run, not able to properly differentiate between food and poison. However, the Fuzzy logic based creature is able to marginally separate the two, making it the best performer in this test. As with food creature 3, the limited sensory input space of the four nose entities may appeal to the rules of fuzzy logic.

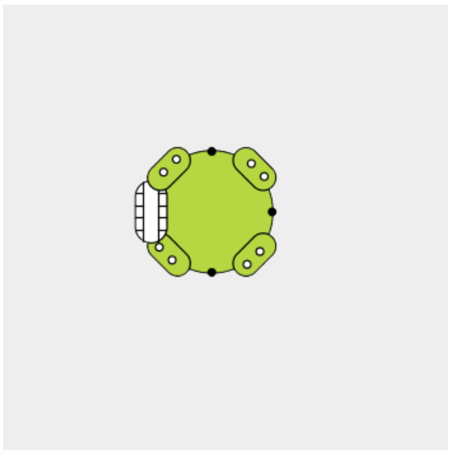


Figure 5.21.: Poison world Designed creature 3

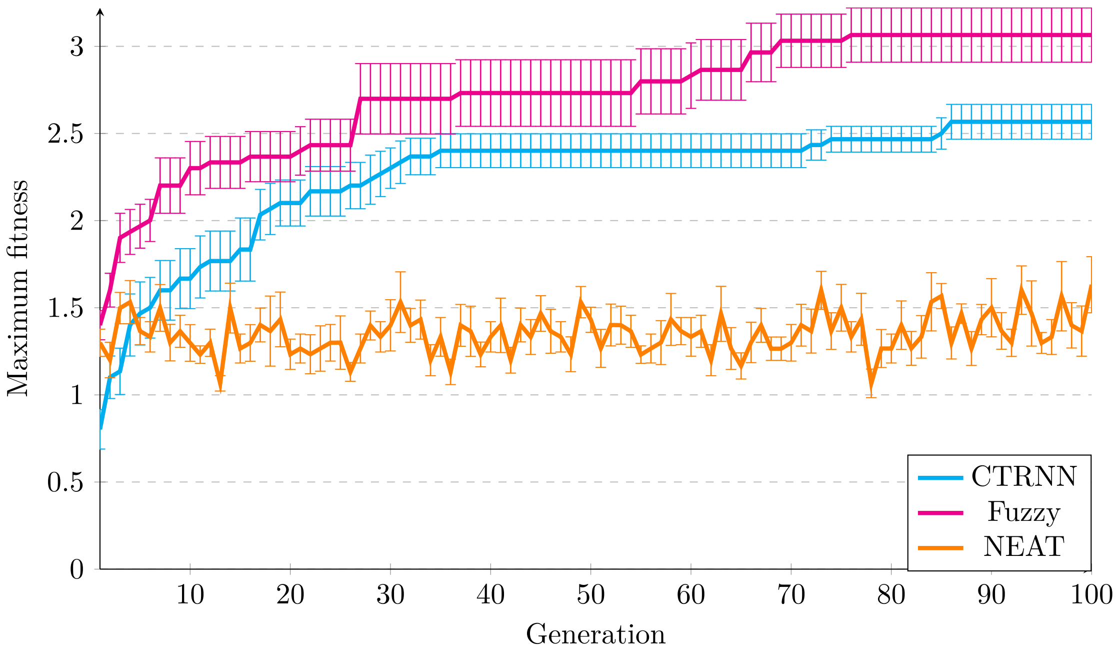


Figure 5.22.: Poison world scenario results for designed creature 3.

Poison creature 4

The fourth designed creature in the poison scenario has noses attached to arms. These may be utilized in the search for food and poison by rotating the arms to cover a bigger search space. The behaviour is very similar to food creature 4, which also has nested arm entities. However, in this test NEAT showed signs of more complex behaviour by using its arms to "look around". Unfortunately this behaviour did not impact the performance of NEAT positively, as the fitness plot shows. Out of the three behaviour modules, CTRNN produced the highest fitness, and also showed a healthy evolvability throughout the run.

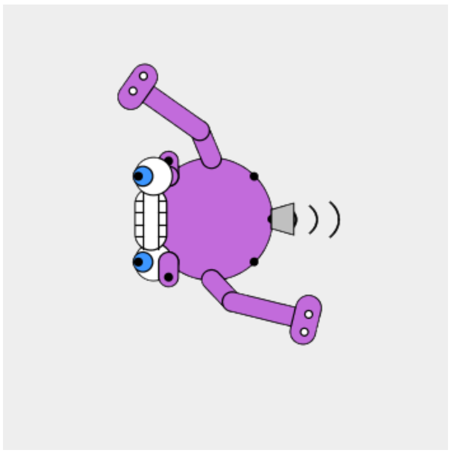


Figure 5.23.: Poison world Designed creature 4

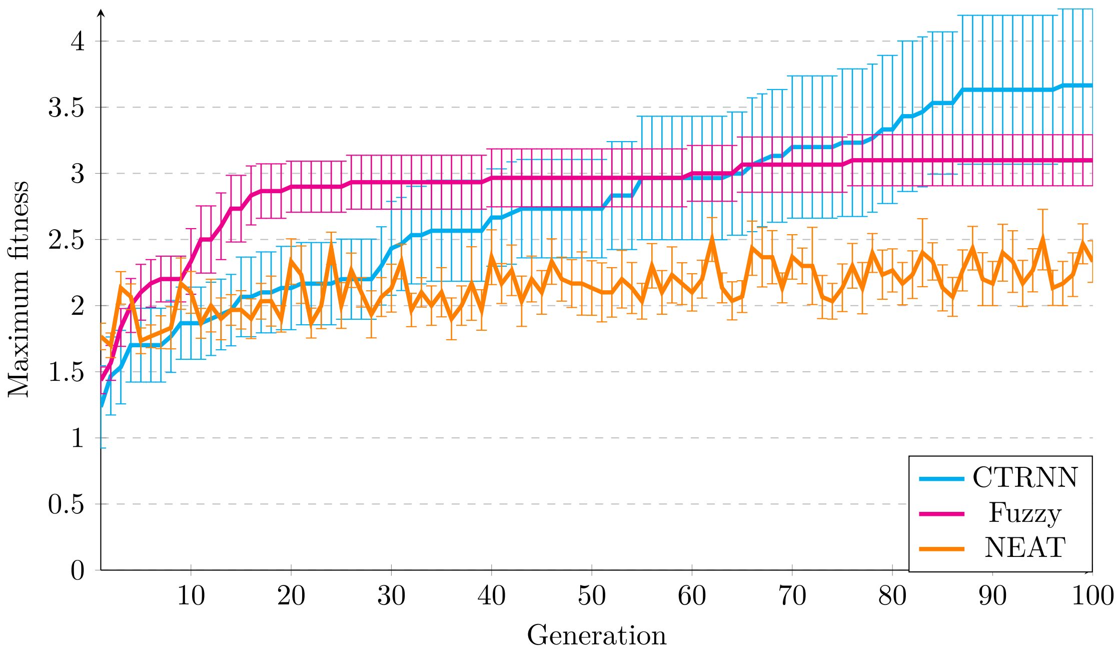


Figure 5.24.: Poison world scenario results for designed creature 4.

5.3.2. Fully evolved creatures

Here, using the fully evolved run mode, there were no evolved creatures that could solve the poison world scenario well. However, looking at the data in the graph 5.25, NEAT achieves significantly higher results than when used with the designed creatures.

One thing that sticks out in these results, is that although NEAT performs better, the other two brain modules perform at about the same level as when used with the designed creatures. It is worthy to note that these runs do multiple behavioural Evolutionary Algorithm (EA) runs that have much fewer generations than the designed creature runs. Therefore these differing results may have something to do with how quickly a brain module optimizes for a given task. Another thing to note, is that by looking at the creatures produced in this scenario (figures 5.26, 5.27 and 5.28), it can be observed that there does not exist any pattern or similarities to the evolution. This falls in line with the idea that this scenario is really difficult for the creatures to solve. The differences signify that the solution space is more explored than if the creatures all looked the same. Some creatures try to brute force their way to better fitness by having a lot of mouths, similar to the results of the food world scenario. Others try to combine eyes and noses as sensors, while some test out entities like the ear that does not really have an effect in this scenario. When it comes to the creatures' behaviour, they behave as expected with their current sensors. The creatures with only eyes tries to eat both food and poison, although not as surely as the creatures in the food world scenario, and the creatures that use both eyes and noses try to avoid the poison with varying degrees of success.

However, the standard error on NEAT is really high compared to the designed creature runs. This suggests that it is a bit reliant on luck in order to evolve quickly to fit the current task.

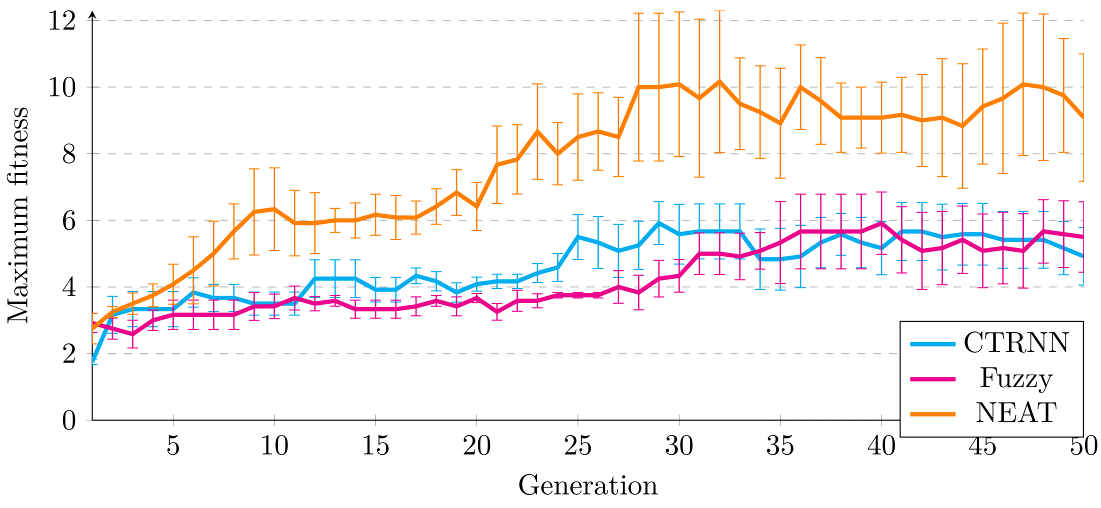


Figure 5.25.: Poison world scenario for the evolved creatures

CTRNN

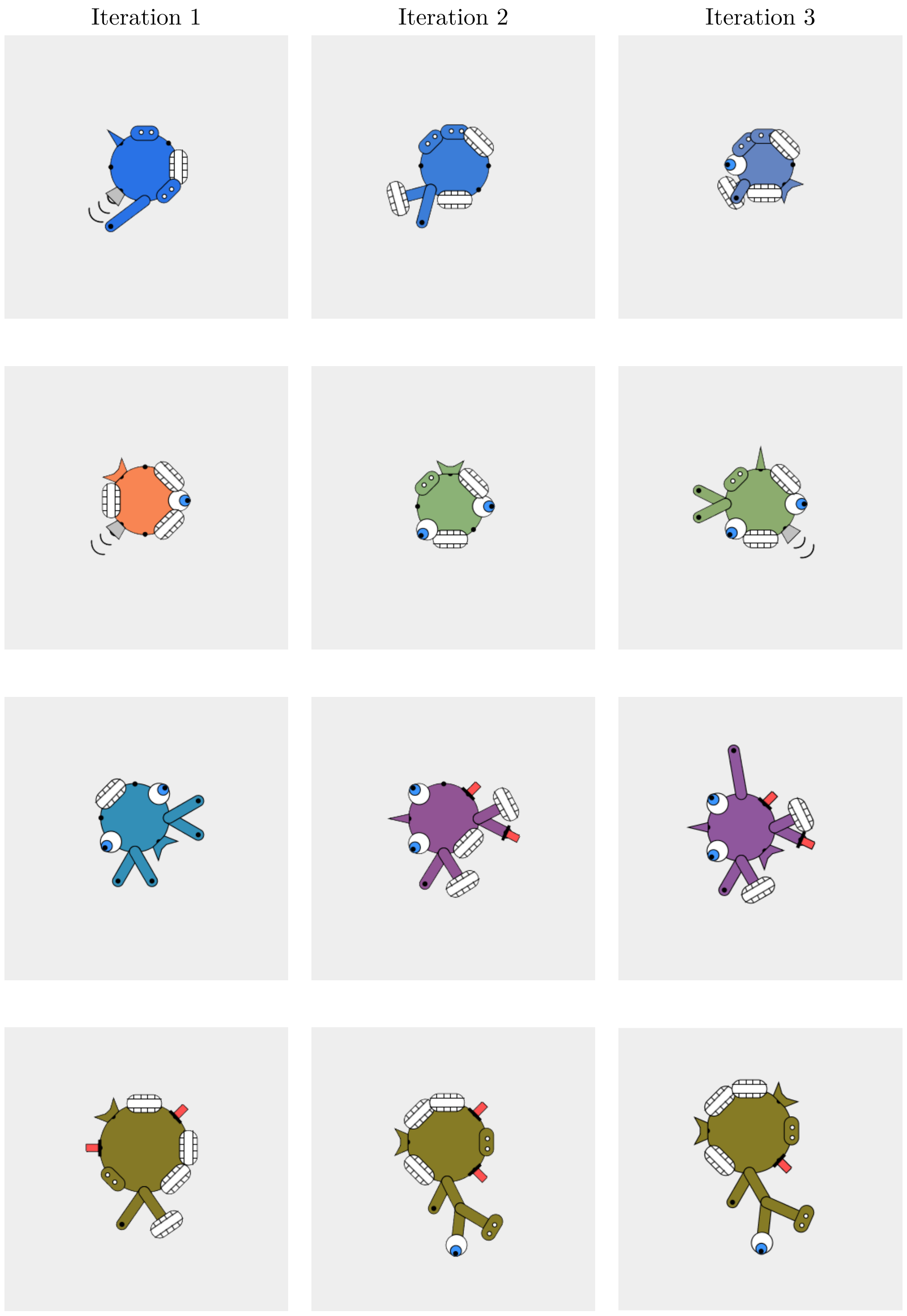


Figure 5.26.: The evolved creatures from the poison world scenario runs with CTRNN as behaviour module.

Fuzzy Logic

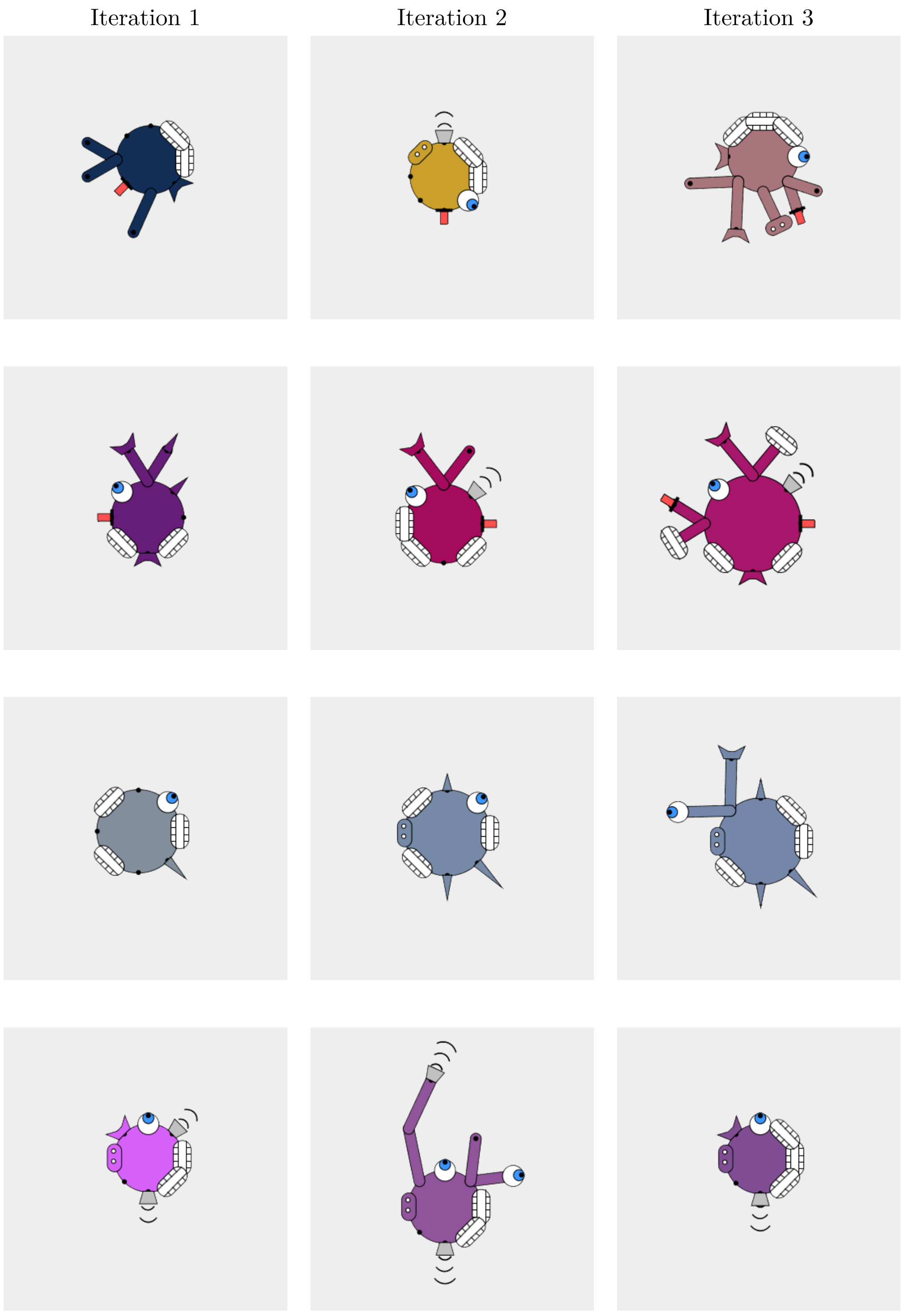


Figure 5.27.: The evolved creatures from the poison world scenario runs with fuzzy logic as behaviour module.

NEAT

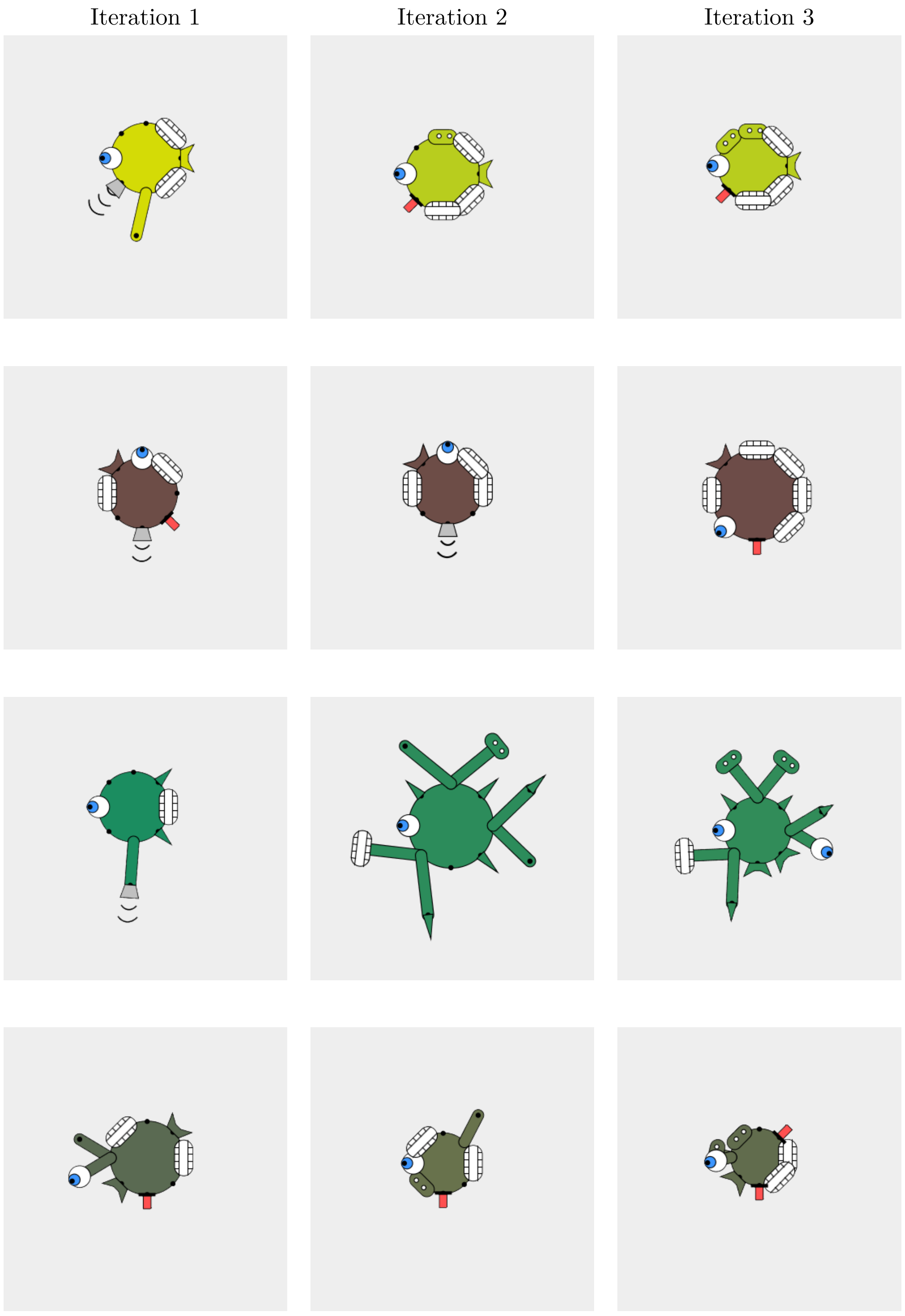


Figure 5.28.: The evolved creatures from the poison world scenario runs with NEAT as behaviour module.

5.4. Arena world scenario

In the arena world, the objective for a creature is to defeat another creature while staying as healthy as possible itself. In this world there are two creatures present during simulation. One is the creature to be tested, and the other is a predefined creature with a predetermined behaviour (from now on called benchmark creature). The benchmark creature has two eyes looking forward, a spike in the front and a motor at the back. Its behaviour is really simple. For every tick, the benchmark creature will move a bit forward and either turn a bit left or right. This is enough to give the tested creature a target that moves unpredictably and is somewhat dangerous to approach due to the spike. In the arena world, the fitness is calculated using equation 5.4.

$$f(c, b) = \frac{H_c - H_b - n_{c,ticks}}{100} \tag{5.4}$$

where c is the tested creature, b is the benchmark creature, H_c is the health of the tested creature, H_b is the health of the benchmark creature and $n_{c,ticks}$ is the number of ticks the tested creature lived. This is divided by 100 to make the fitness a more reasonable number. The maximum fitness is 20, due to the initial health of 2000. If a test creature is able to instantly defeat the benchmark creature, while not losing any health itself, it is awarded a fitness of 20.

5.4.1. Designed arena world creatures

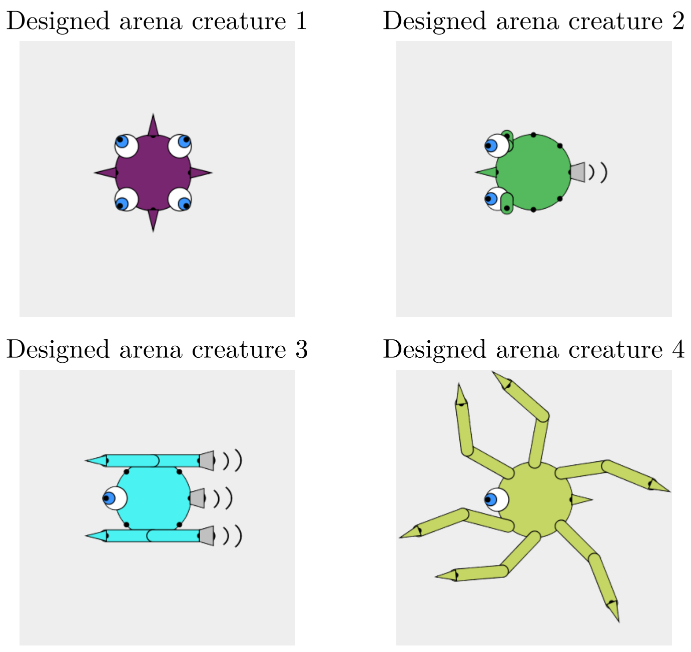

Designed arena creature 1
Designed arena creature 2
Designed arena creature 3
Designed arena creature 4

Figure 5.29.: Designed creatures used in the arena world scenario.

Arena creature 1

The first designed creature in the arena scenario has four eyes and spikes, which should give the creature the ability to locate the adversary in almost all directions. However, that is not the observed behaviour. The CTRNN-based creature simply moves in a straight line, vibrating its spikes in a hope to randomly hit its adversary. On the other hand, the fuzzy logic creature seems to be aware of the adversary by stopping its movement when observed.

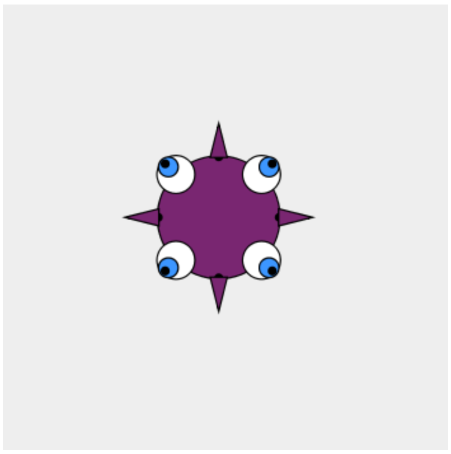


Figure 5.30.: Arena world
Designed creature 1

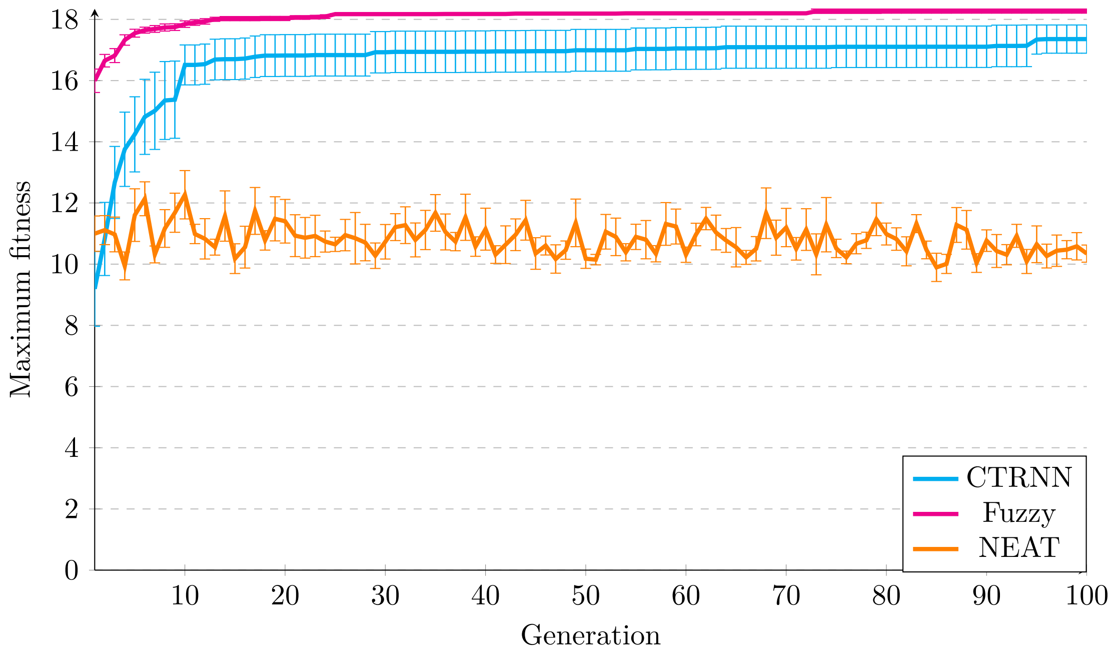


Figure 5.31.: Arena world scenario results for designed creature 1

Arena creature 2

The second designed creature in the arena scenario is very simple, with two eyes and a spike in the center to inflict damage. This design is similar to the benchmark creature used in the arena scenario tests. The behaviour in this test is very much similar to the arena creature 1, where the preferred behaviour is to simply move in a straight line. The three different behaviour models perform very similar.

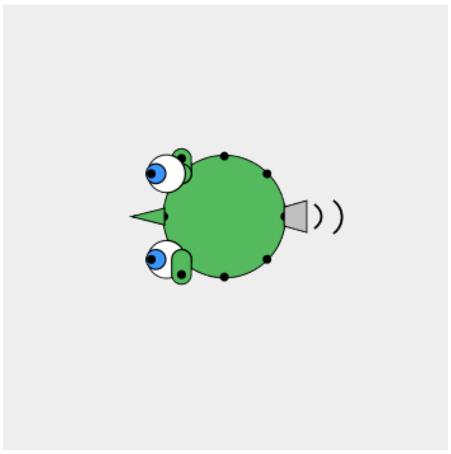


Figure 5.32.: Arena world
Designed creature 2

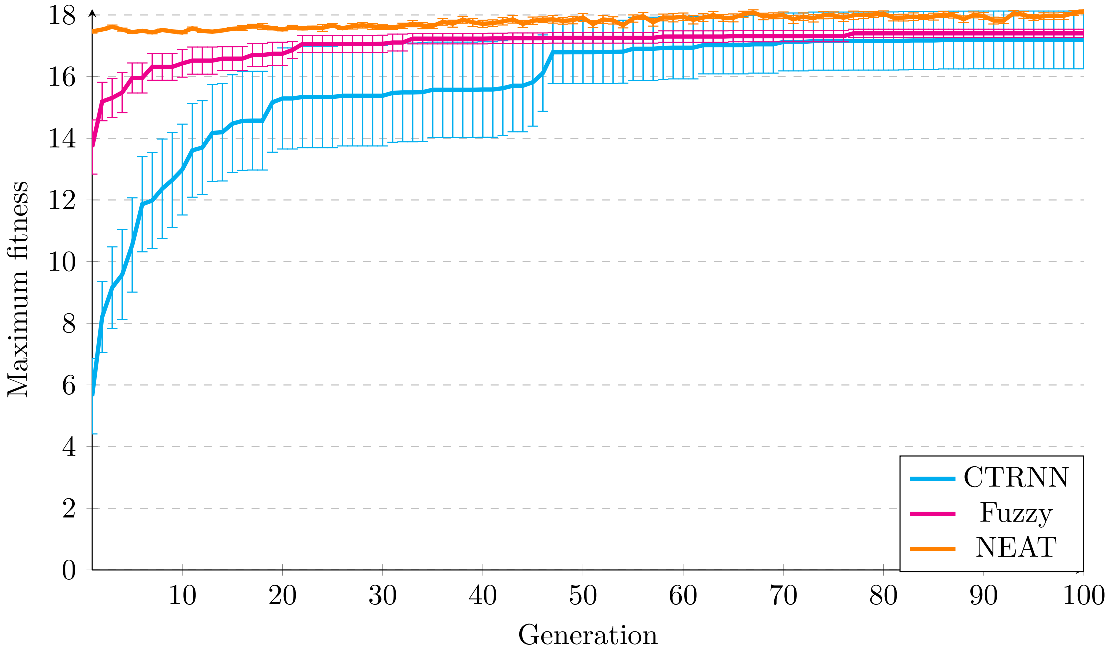


Figure 5.33.: Arena world scenario results for designed creature 2.

Arena creature 3

The third designed creature in the arena scenario has three motors to propel it forward, potentially increasing the damage output of its spikes in the front.

This creature performs as expected with every brain module. It finds the target with its eye, and then launches itself toward the other creature. As damage is based on the speed of the impact, the result is that the creature does more damage.

Looking at the results, it can be seen that all three brain modules evolve to their maximum after a few generations. In addition to this, they all have very small standard error, something that says that this creature was easy to optimize for the given task.

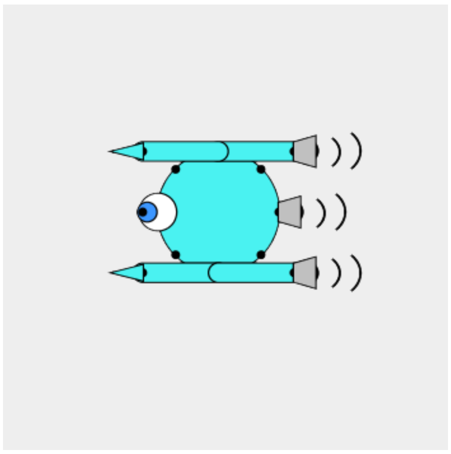


Figure 5.34.: Arena world
Designed creature 3

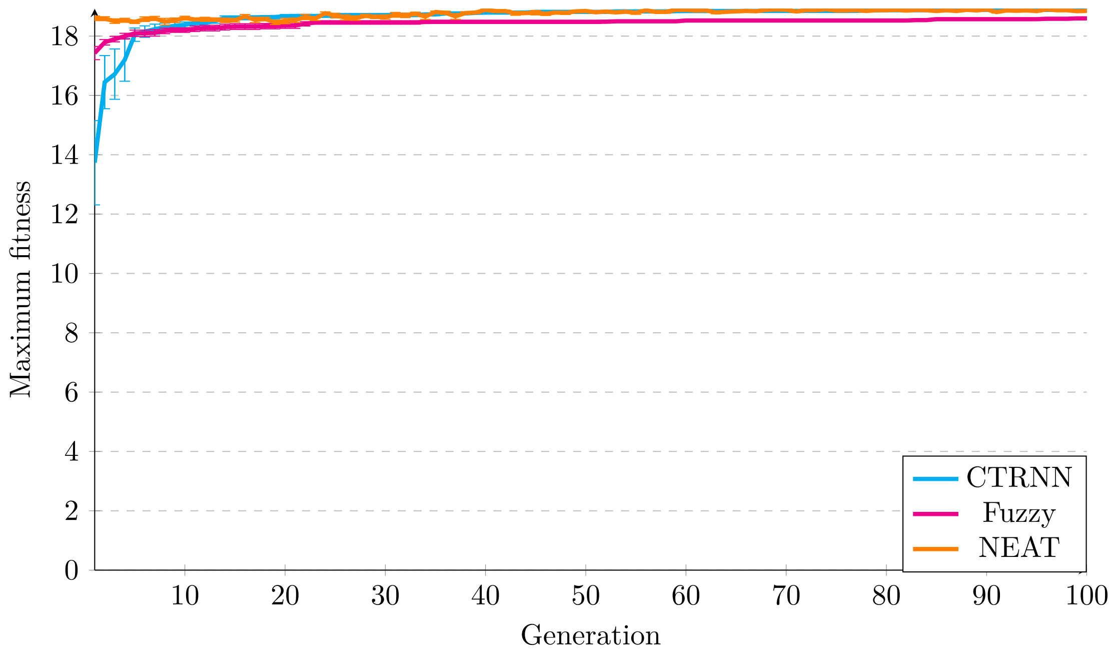


Figure 5.35.: Arena world scenario results for designed creature 3.

Arena creature 4

The fourth designed creature in the arena scenario has six arms. Each equipped with another arm, creating a joint, and a spike at the end. The behaviour between fuzzy logic and CTRNN differs greatly here. Using CTRNN as the brain module, the creature systematically goes after the target creature and defeats it with several small hits. On the other hand, using fuzzy logic as the brain module, the creature spins in place, using its arms to cover a large area in order to hit the other creature.

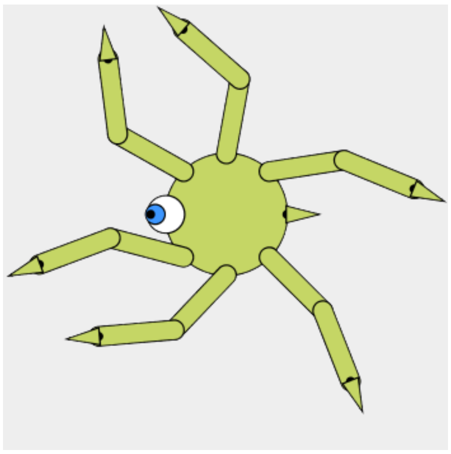


Figure 5.36.: Arena world Designed creature 4

The behaviour difference is reflected in the graph of the data. Although not by more than 1.5 points in fitness, it can be seen that the behaviour of fuzzy logic is slightly more efficient.

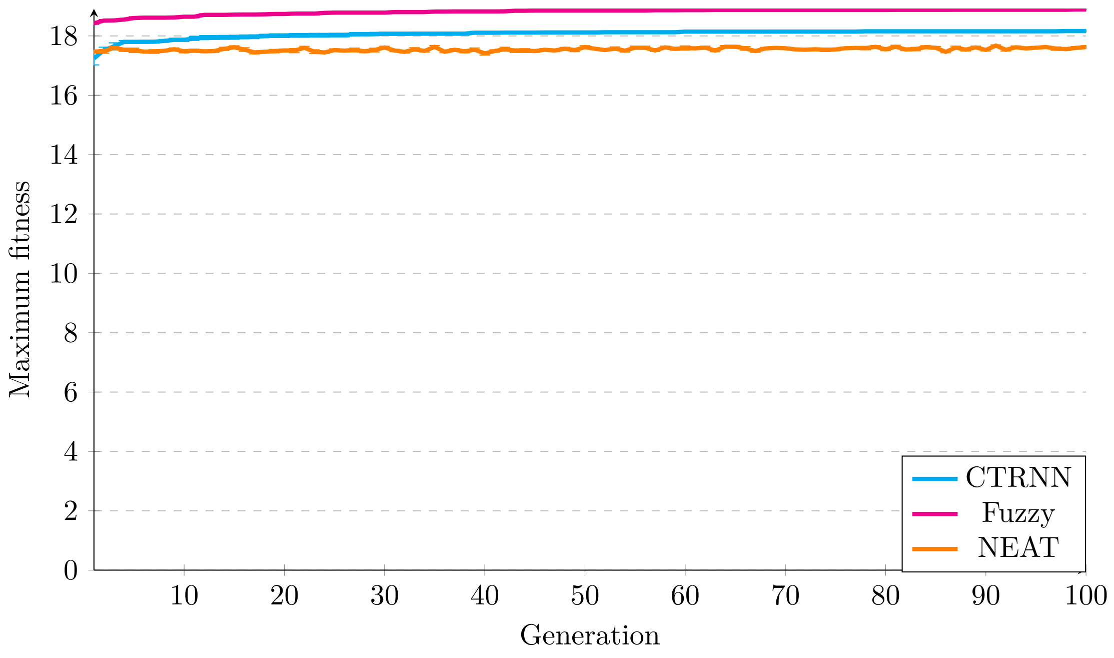


Figure 5.37.: Arena world scenario results for designed creature 4.

5.4.2. Fully evolved creatures

The arena world is probably the most complex scenario in the test suite, however, the observed behaviour of the fully evolved creatures does not coincide with the scenario complexity. The established behaviour is to move in circles, hoping for the adversary to hit one of its spikes in order to inflict damage. Two of the four evolved NEAT creatures have no means for receiving sensory data, because they lack the appropriate entities (figure 5.41). In spite of this simple, repetitive and non-intelligent behaviour, all behaviour models achieve an acceptable fitness value as shown in figure 5.38.

Even though the evolved behaviour is not considered creative, the evolution of specific entities and structural configuration may be considered creative. For example, the second fuzzy logic creature (figure 5.40) and third NEAT creature (figure 5.41) have evolved large structures of fork and spike entities in order to reach longer and gain more momentum in their attacks. Another example is the first CTRNN based creature (figure 5.39), which has adopted a vibrating arm with an attached spike. This vibration increases the net traveling speed of the spike, effectively increasing its damage output.

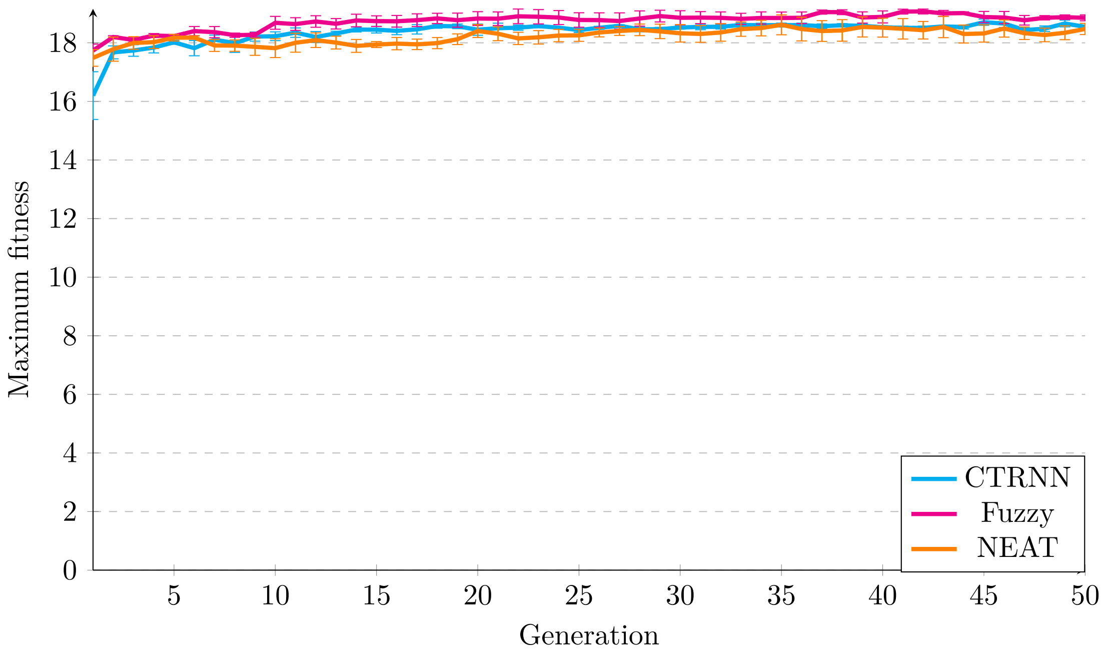


Figure 5.38.: Arena world scenario for the evolved creatures

CTRNN

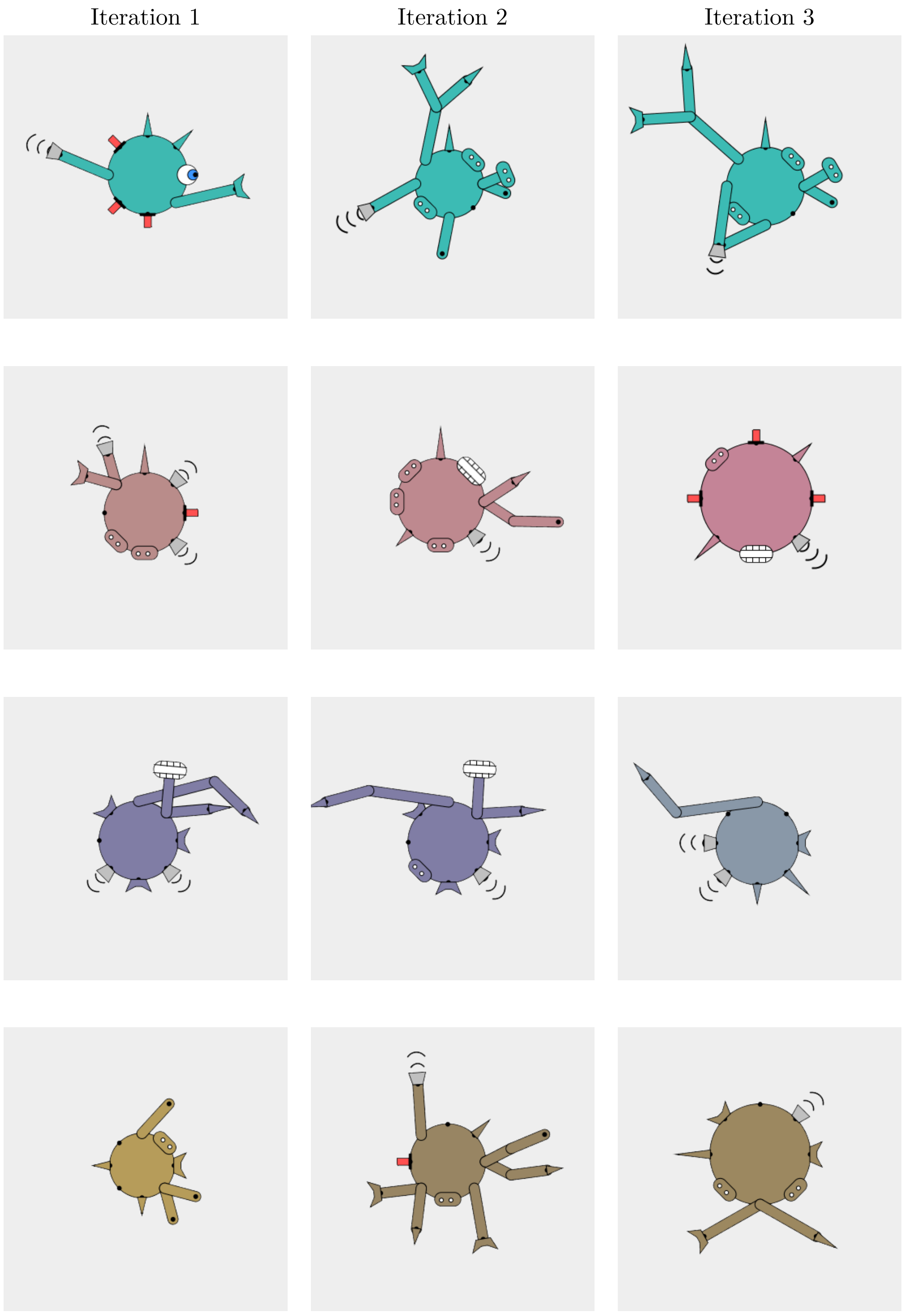


Figure 5.39.: The evolved creatures from the arena world scenario runs with CTRNN as behaviour module.

Fuzzy Logic

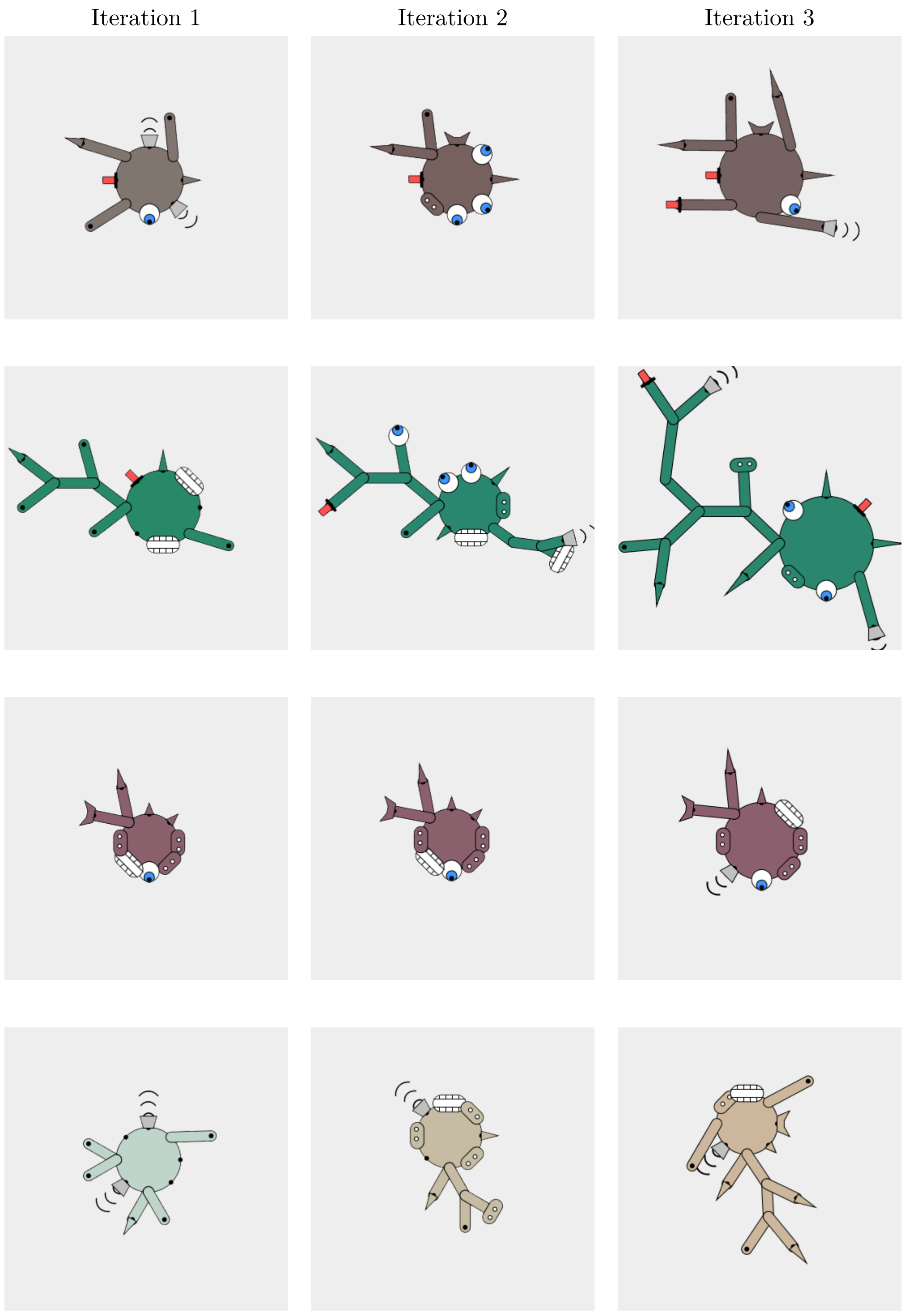


Figure 5.40.: The evolved creatures from the arena world scenario runs with fuzzy logic as behaviour module.

NEAT

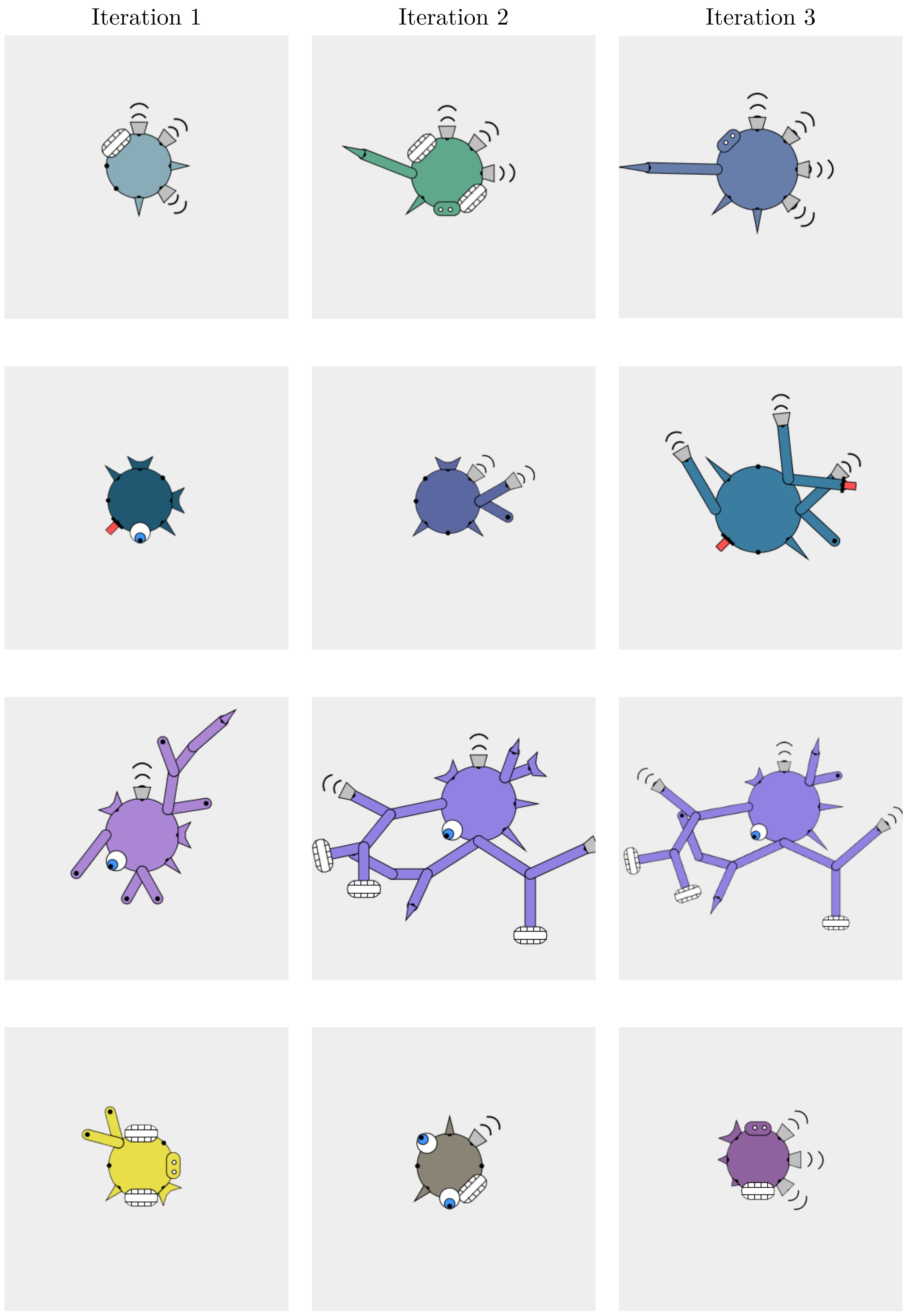


Figure 5.41.: The evolved creatures from the arena world scenario runs with NEAT as behaviour module.

5.5. Additional experiments

In addition to the systematic experiments performed in section 5.2 through 5.4, some independent experiments are presented in this section. These tests make up a selection of interesting scenarios, system configurations, run-times and initial conditions that do not naturally fit in the previous sections.

5.5.1. Nose navigation in a food world

This experiment seeks to investigate the possibility of using nose entities as a substitute for eyes. The main difference between the two entity types are sensory range, FOV, and the value of the returned sensor data. An eye entity reports both distance and direction to the nearest object in view, whereas a nose only reports a value of 1 when a food piece is inside its sensory radius. This seriously limits the amount of useful information coming from a nose, versus an eye. Figure 5.42 shows the designed creature used in this experiment. Five nose entities with different sensory ranges (shown in red, dashed lines) is used. Table 5.4 contains the most relevant configuration parameters of this experiment. See appendix section B.4 (page 98) for the full configuration file.

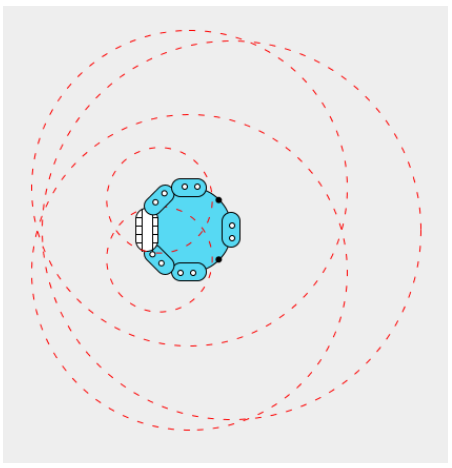


Figure 5.42.: Designed creature used in the nose navigation experiment.

Table 5.4.: System configuration used in the nose navigation experiment.

Parameter	**Value**
Scenario	Food world
Neural network configuration	Two hidden layers: 6,6
Population size	50
Generations	2000
Elitism	0
Parent selection strategy	Tournament selection

The systematic nose placement in the designed creature, along with custom sensory ranges, allows the creature to perform well in the food world. Figure 5.43 shows the maximum fitness out of ten runs, along with standard error[2]. Around generation 1880, the evolution reached a maximum average fitness of 20.2 using CTRNN, and a peak fitness of 45.0 at generation 1802 which translates into 45 eaten food pieces. This is far greater than the comparable food creature 3 experiment (page 48), which peaked at 15.0. The main difference between these experiments are the number of generations, selection strategy and the fact that the nose navigation experiment utilize nose entities with modified sensory ranges and optimal nose placement. The nose creature is able to eat most of the food it encounters, and handles multiple food pieces residing inside the sensory range, which is generally something creatures struggle to handle. However, this can only be said about the CTRNN-based creature. Using fuzzy logic, the creature performs acceptable with a maximum average fitness of 10.6. We see that CTRNN outperforms fuzzy logic, compared to the food creature 3 experiment where fuzzy behaviour outperformed CTRNN. These findings suggest that CTRNN present the greatest level of evolvability, as the other two behaviour models struggle to increase their maximum fitness in the same manner as CTRNN.

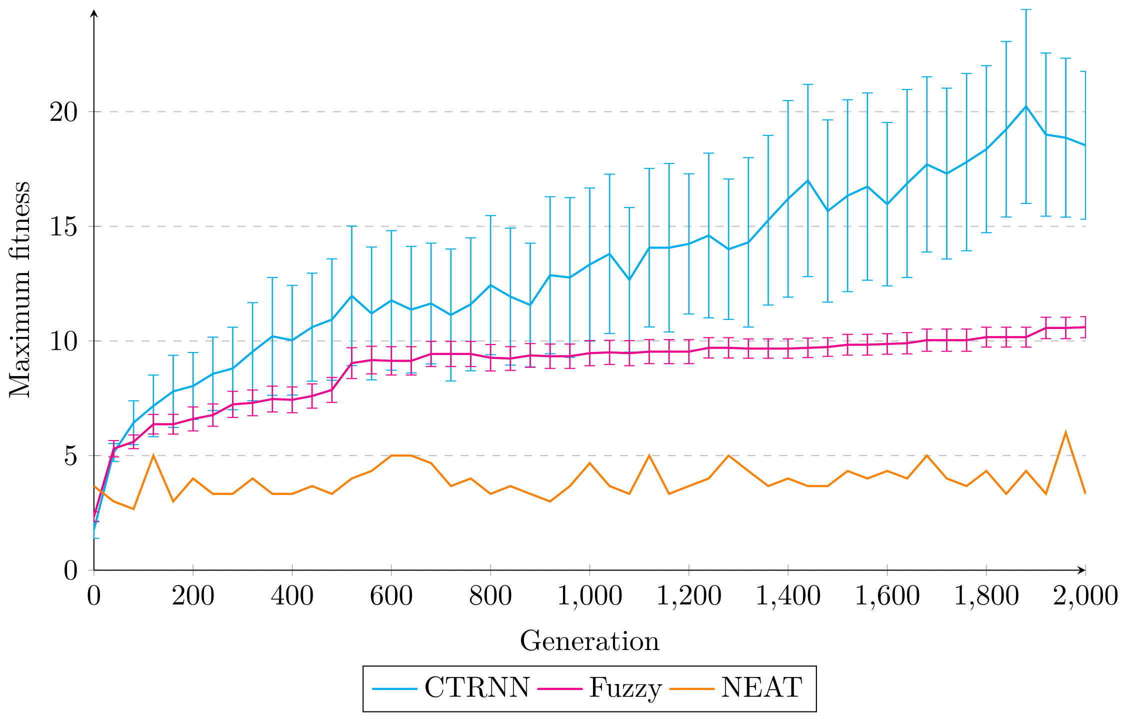


Figure 5.43.: Maximum fitness of the nose navigation experiment.

[2]The NEAT-run was only performed once due to extensive run-times, and therefore lacks error bars.

Using NEAT-based behaviour, the evolution was not able to improve its maximum fitness throughout the run. The exact reason for this is unknown. The suspicion is that this creature requires close coordination between the different noses, i.e. the network needs to be very interconnected. As NEAT starts out with no hidden nodes, it has a bigger challenge in evolution than CTRNN that already starts fully connected with hidden layers. However, this is something that should be solved with evolution. The fact that it does not manage to evolve a solution for this creature may suggest that there are hidden factors yet to be revealed, which could be worth investigating in the future.

As mentioned earlier, the nose-navigation creature shows many physical similarities with food creature 3. However, the nose-navigation creature performs better in terms of maximum achieved fitness. This may be explained by the change of parent selection strategy, from fitness proportionate to tournament selection, or the deduction of elitism. In order to investigate the change in fitness that these parameters represent, the food creature 3 experiment were re-run with tournament selection. In figure 5.44, the resulting fitness landscapes are presented, showing that tournament selection *with* elitism produces the highest fitness.

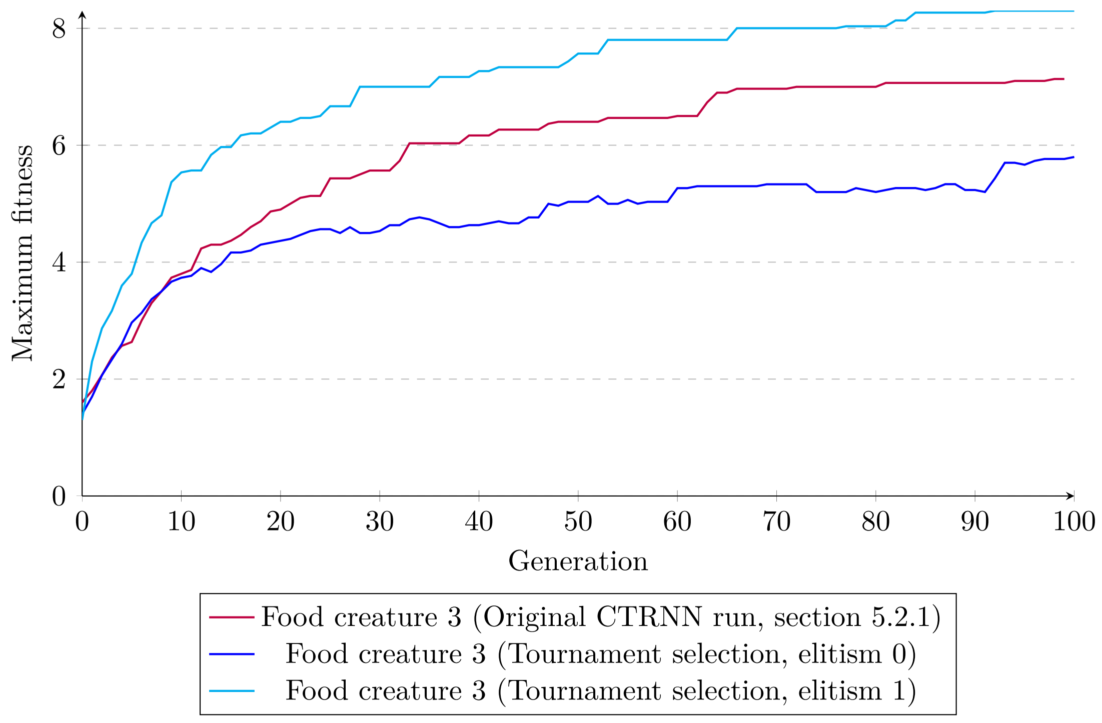


Figure 5.44.: A re-run of the food-creature 3 experiment, using tournament selection with/without elitism enabled. The plot shows average maximum fitness at each generation for ten runs.

It appears in figure 5.43 that the fitness value has not fully converged within the 2000 generations the experiment were conducted. Therefore, a new run of 10000 generations is presented in figure 5.45. This is using the same configuration parameters as stated in table 5.4, but for 10000 generations instead of 2000. For CTRNN, we see that the fitness value quickly rose to about 30 before 1000 completed generations, and that the simulation were not able to evolve beyond an average fitness of around 40. The highest recorded fitness throughout the run was 49.0 for CTRNN, and 10.3 for fuzzy logic. It is clear that the fuzzy behaviour module struggles to obtain a fitness level on a par with CTRNN. Other experiments, such as food creature 3 and poison creature 3, favours the fuzzy-based behaviour, suggesting that the scenario and creature design choices may greatly influence the evolutionary outcome.

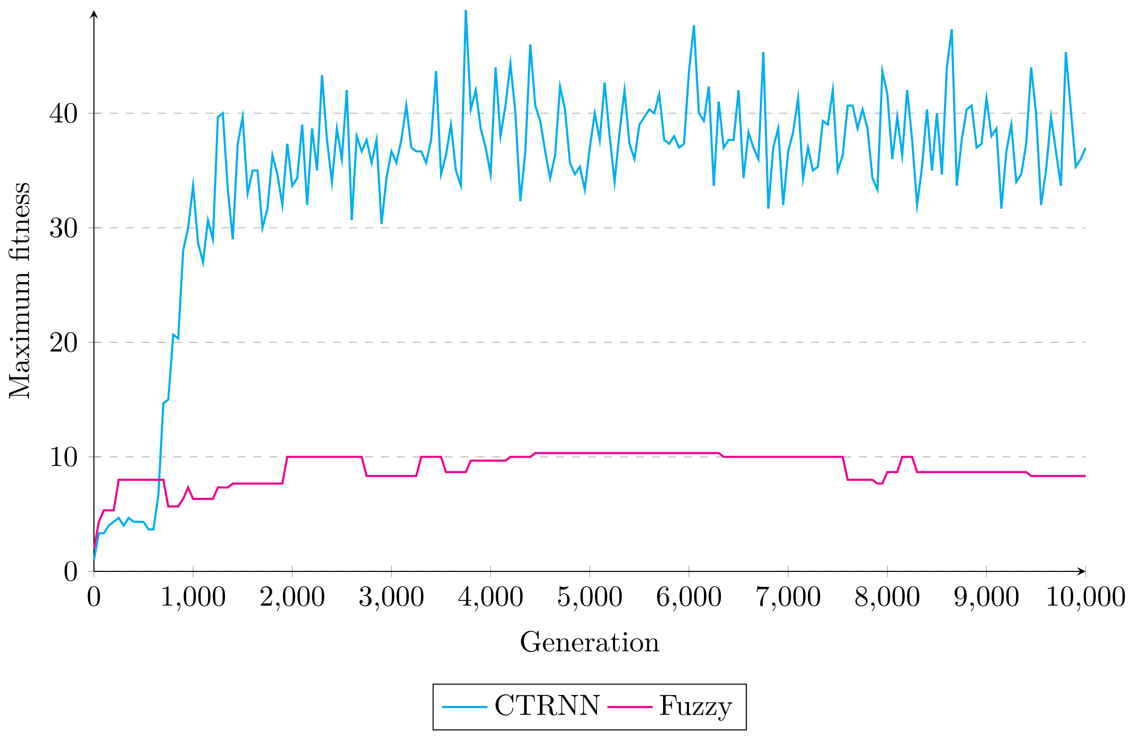


Figure 5.45.: Nose-navigation experiment executed for 10000 generations.

6. Evaluation and Discussion

In the introduction to this master's thesis, four goals were devised:

G1: Evolve creative behaviour in custom agents.

G2: Implement a system to evolve and visualise creature performance.

G3: Study several Artificial Intelligence (AI) methods and their influence on evolving creative behaviour.

G4: Study how different environments and tasks affect a creature's ability to perform creatively.

This chapter contains evaluations around the results of the experiments and a discussion on how well they lined up with the thesis goals. In addition to this, there will be a discussion on the merits and the limitations of the system.

6.1. Evaluation

In this section, the results of the experiments will be evaluated in relation to the project goals. Each world scenario of the system will be evaluated separately, using the Standardised Procedure for Evaluating Creative Systems (SPECS)(Jordanous, 2013) (see section 2.5.3). Both designed- and full-runs will be considered in the evaluation. The system as a whole will also be evaluated.

- Step 1: Identify an applicable definition of creativity:
  The definition that is going to be used, is the one that was specified in section 3.1.

  Creative behaviour is behaviour that results in a product that is unique or somehow valuable to an individual or society (Maher et al., 2008).

- Step 2: Find the standards to test from the definition in step 1.
  From the definition, it is possible to extract three different standards. These are:
  1. Uniqueness
  2. Valuable to an individual
  3. Valuable to society

- Step 3: Test the system on the given standards.
  For creatures in the three different scenarios, only the uniqueness standard will be

tested. This is because there is very little intrinsic value in the difference between a food, a poison and an arena based creature. This is instead discussed somewhat in the system evaluation later in this section. When evaluating the entire system, all three standards will be used.

6.1.1. Food world evaluation

Here, the fully evolved creatures clearly outperform the designed creatures when it comes to gathering food. Considering that most of the designed creatures performed well, this points to that the Creative Creature Behaviour (CreBe)-system is very capable of evolving and adapting creatures to solve a given task.

In the designed creature runs, the best brain module changes from creature to creature. However Continuous Time Recurrent Neural Network (CTRNN) seems to often get the upper hand given enough time. In the fully evolved runs, NeuroEvolution of Augmenting Topologies (NEAT) leads by a substantial margin. This is due to its ability to evolve behaviour in few generations.

Evaluating with SPECS:

Uniqueness:
Almost all of the creatures evolved here utilize the same strategy in order to gather food. They use one or more eyes to locate food, and evolve several mouths in strategic locations in order to sweep up as much food as possible. The creatures all display the same behaviour. When they see food, they all approach it with varying degrees of success. In addition to this, they do not differ that much from the designed creatures. I.e. they evolved in a predictable way. The argument could be made, that the creatures are unique physically, as it is something we would have never designed ourselves. However, that brings on the discussion of where randomness ends and where creativity begins.

6.1.2. Poison world evaluation

In this scenario the fully evolved creatures perform better than the pre-designed ones with every brain module. Looking at the results, it is clear that this is a very difficult task for the creatures to solve, as they need to learn to rely on multiple sensors. When the scenario is at this difficulty, the creatures' ability to evolve their physical attributes plays a bigger role. It appears that the pre-designed creatures were not good enough.

NEAT performs a lot better than the other two brain modules in the fully evolved runs than in the designed creature runs. This points to that with fewer generations to evolve behaviour, NEAT evolves faster to solve the given task.

Evaluating with SPECS:

Uniqueness:
Given that none of the creatures really managed to solve the task at hand, it is difficult to see the differences in their behaviour. The best performing creatures sometimes managed to make a quick turn if they smelled a poison. Some creatures tried a brute forcing route, where they just tried eating everything and relying on luck to eat more food than poison. There was no creature that stood out, either in physical appearance or behaviour. So for uniqueness, these creatures showed none.

6.1.3. Arena world evaluation

The results for the fully evolved creatures are consistent with the results of the designed creatures. The performance of NEAT vary a lot with each creature. It sometimes performs really well, and other times gets stuck at a bit lower fitness that the two other brain modules. CTRNN and fuzzy logic are both produce really consistent results, however on average, fuzzy logic comes out on top. This is reflected in the fully evolved creatures results. Here, fuzzy logic clearly performs the best, while NEAT and CTRNN are about the same somewhat lower in fitness. However, the standard error is a lot smaller with CTRNN as brain module. In some cases, NEAT experiences poor evolvability and struggles to gain evolutionary momentum. Maybe the creature behaviour problem domain does not fit well with NEAT, but more tests and experiments have to be conducted before extensive conclusions can be drawn on this matter.

Evaluating with SPECS:

Uniqueness:
As can be seen in the results of the fully evolved creatures (5.39, 5.40 and 5.41), they are quite different from the designed creatures (5.29). Thus the results can be called somewhat unique. However, comparing the fully evolved creatures against each other, it can be seen that they do not differ that much. They all evolve towards having spikes to cover as large an area as possible. On the other hand, some of the fully evolved creatures evolve motors in order to move faster. This was partly the idea with some of the designed creatures. From this, it can be observed that the fully evolved creatures showed some uniqueness that separated them from the designed creatures.

6.1.4. System evaluation

The system as a whole (CreBe) has shown in the experiment its capabilities to produce independent agents, with different types of behaviour modules. The architectural choices of the software enables for the implementation of multiple action-inference mechanisms with relative ease. This also applies to the different scenarios available in the system, where new world types with different rules can easily be implemented.

The world scenarios presented in this thesis are relatively simple, with no to little world dynamics. However, the type of world is alone responsible for defining the different objectives for which the agents are optimized against. Eating food and avoiding poison are the primary objectives in the food/poison world, which is of course, the observed behaviour in this kind of world. An important question is, can this scenario-behaviour relation be applied to more real-world problems? By introducing different entity types, better world physics and scenario rules, the system should be able to be tailored towards custom problem domains where emergent behaviour is a favourable property. The entity objects are in fact abstract input and output interfaces to a simulation, which can be tailored towards a range of applications where CTRNN, fuzzy logic and NEAT would be applicable.

Evaluating with SPECS:

> Uniqueness:
> When evaluating uniqueness, there are two different concepts to consider. The first is artifacts, or in the context of this system, the produced creatures. Their appearance are inspired by animals and insects, which includes most of the entity types apart from the touch sensor and motor. The different types of sensors and actuators are also heavily inspired by biological entities, along with the quality of their input and output interfaces. For example, the nose entity can not sense the direction of a smell, which is also difficult in real-life.
> The second concept of uniqueness is related to the system implementation itself, which is somewhat limited in the sense that the different algorithms used are well-known and established outside of this thesis. However, the way that different action-inference models can be inserted into and control creatures, makes the system somewhat new and unique.
>
> Valuable to an individual:
> In order for the system to be valuable to an individual, it must be able to produce some sort of product, sensation or experience that positively affects said individual. These are highly subjective topics, but also includes more objectively better aspects, for example better health care. According to the found definition of creativity, this product can pertain to any aspect of life. For example increased life expectancy, better health, amusement and art. It is clear that the CreBe system is unable to directly increase ones health, but it may have some value as amusement and in recreational use. For example as a toy for children who find the creatures cute and funny.
>
> Valuable for society:
> When it comes to evaluating if something is valuable for society, it has to be impactful in some way or another. The CreBe system evolves creatures to solve certain tasks, and given the right circumstances, the solution might have an impact. However, as the simulation of the system is somewhat simple and does not model reality very accurately, the tasks the system tries to solve do not reflect the real world. This makes it unlikely for the system to give some value to society. On the

other hand, if the system could be valuable to a lot of individuals within a society, it could be argued that the system is valuable to the society itself.

6.2. Discussion

In this section, we discuss each project goal separately, evaluating the degree to which they have been accomplished.

G1: Evolve creative behaviour in custom agents.
The first goal of this thesis applies to a very broad spectrum in the field of computational creativity. In order to work towards this goal, a more specific subgoal was casually defined, namely:

Implement a system using AI methods to evolve creatures that exhibit creative behaviour in a strictly defined environment.

One immediate limitation of the implemented system is that the various world scenarios are indirectly subject to human intervention, in the way that the world rules are static and strictly defined. This contradicts a key aspect of the found definition of computational creativity, namely to limit human intervention. It is clear from the results of this thesis that the amount of creativity exhibited by the evolved agents is in fact very limited. The food world scenario offers few opportunities in regard to creative behaviour, because eating food is the only motivated action in that world type. Introducing poison, which can be hard for creatures to separate from ordinary food, motivates for more complex behaviour. The arena world expands on this even more, by introducing a multi-agent environment. However, these complex scenarios did not affect the cognitive abilities and creativity of creatures the way we hoped. The system still struggles to produce creative behaviour in evolved agents, which suggests that this project goal is not accomplished.

The claim that the implemented system is capable of producing creative behaviour in agents is hard to justify, considering the output results. Even though it shows some capabilities, the conclusion is that the system comes up short in reaching the original project goal of producing creative behaviour.

G2: Implement a system to evolve and visualise creature performance.
The need for a system to evolve and visualise agents in real time arose early in the project period. The implemented system (called *CreBe*), fulfills these requirements and thereby accomplishes this project goal. Although not all planned features were implemented in the final version of the software, e.g. behaviour inheritance between generations, human intervention in fitness evaluation and creature vs. creature evaluation in the arena scenario (currently creature vs. benchmark). Overall, the implementation served its purpose during the experiments.

G3: Study several AI methods and their influence on evolving creative behaviour.
The three different AI techniques considered in this thesis are:

- CTRNNs.
- Fuzzy logic.
- NEAT.

That are commonly referred to as behaviour models or brain-modules in this thesis. In order to compare these methodologies, a number of experiments were conducted in order to study their influence and decide what method performed the best in the context of creative behaviour. In the designed creature experiments, CTRNN has a tendency of achieving the highest fitness of the three models, except for the designed food and poison creature 3, where fuzzy logic were the best performer. In the full-evolution experiments, NEAT really shines, and outperforms both CTRNN and fuzzy logic in the food and poison scenarios.

Because CTRNN is the only computational model that implements forms of memory (using recurrent connections), this model produces slightly different behaviour in creatures. Sometimes, without any change in sensory input, a creature may spontaneously perform an action like changing direction and/or speed. This is due to the nondeterministic nature of a CTRNN, which does not apply to the two other computational models. These will produce a predicable output for given input in all cases, resembling the cognitive level of a simple reflex agent.

G4: Study how different environments and tasks affect a creature's ability to perform creatively.
The different test environments considered in the experiments are:

- Food world scenario.
- Poison world scenario.
- Arena scenario.

These scenarios represent different complexity levels, and present different requirements in behaviour in order to perform well. The simplest scenario, the food world, was mainly implemented as a test and benchmark scenario during the development phase. Nevertheless, each behaviour model generally performed respectable in this scenario, but did not show notable levels of creativity in their behaviour. CTRNN arguably shows signs of creativity in its ability to explore the environment for food, something that NEAT and Fuzzy logic do not show. This can be partially explained by CTRNN having simple forms of memory, enabling for a richer and less reflex driven behaviour.

In the poison scenario, we see a very similar behaviour as in the food world, although with a reduction in fitness levels. This is due to the introduction of poison, reducing the amount of food pieces and effectively making the maximum achievable fitness lower. The density of food pieces is also reduced, increasing the general difficulty of the scenario. It is expected that the reduced density would further motivate for exploratory behaviour, in order to locate food, but that is not the observed case. It appears that the creatures are very careful in their navigation, probably in order to avoid poison. Further experiments with varying degrees of poison ratios may confirm this.

The arena scenario is arguably the most complex scenario of the three. However, it appears that the level of creativity evolved in the arena world is very limited. The established behaviour is simply to move in a straight or circular motion, until the adversary randomly hits one of its spikes. This behaviour may be a result of limitations in the scenario and its rules, or that the behaviour models need to be scaled up (e.g. increase the number of neurons and hidden layers in CTRNN).

Although the different scenarios produce and motivate for the evolution of different behaviour, it is hard to support the claim that the observed behaviour is creative.

7. Conclusion

In this thesis, a system was developed that is able to evolve animal-like agents called creatures using modular parts. Both action-inference and physical creature layout are subject to evolution, using a series of simulations in specific world scenarios. The observed behaviour in the experiments indicate that the capability of the system is limited in terms of creativity, and that the agents are simply solving a task rather than exploring novel and unexpected behaviour. This shows that the type of scenario used in the evolution may have a big impact on the final behaviour of the agents.

The rest of this chapter will go into the contributions of the thesis and possible future work that can expand on the findings and work of this thesis.

7.1. Contributions

The contributions of this thesis are:

C1: A study of how different AI methods perform when evolving creatures to behave creatively.

C2: An implemented system that evolves creatures using different selectable AI methods and environments.

These contributions form a basis for future work that build upon this thesis. Contribution C1 is the results achieved through the experiments, and C2 is the CreBe system itself.

7.2. Future work

In this section, possible future work on the thesis is discussed. This consists of improvements to the CreBe system and possible more experiments that can be conducted.

7.2.1. Further experiments

Due to how modular the scenarios of the CreBe system are, creating more interesting and complex ones are definitively something that could be investigated further. The original thought was to have creatures fight against each other in the arena world scenario, but due to time restraints it was not implemented. It could be interesting to see how different the creatures would evolve when their opponent also is able to evolve and adapt. Maybe different strategies and counter strategies could emerge. In addition to this there are a lot of different configurations with the existing scenarios that could be experimented with. The density and distribution of food and poison could be changed to be tied to certain areas instead of being uniformly random. Different world sizes and the ability to wrap around the world are also things that could be worth looking into.

In order to make the creatures evolve creative behaviour, more work needs to be done on fitness functions. Looking into a function that could measure some sort of novelty factor for the creatures would better be able to steer the evolution towards creativity.

The experiments conducted in chapter 5 test only a small fraction of the available system configuration and evolutionary parameters. The initial configuration of a simulation can have a great impact on the evolution and final state of the system, which is why different combinations of parameters should be tested. Example of such parameters are population size, number of generations, selection strategies, Artificial Neural Network (ANN)-topologies and the number of fuzzy rules used. This also includes new world scenarios and world rules, which may be tailored towards a specific problem domain. In this thesis, only a small portion of the available initial configuration were tested, which is why further experiments should be conducted in order to investigate the full potential of the system.

7.2.2. System improvements

In the fuzzy-logic based behaviour module, a current limitation of the system lies in the mutation and crossover aspect of the fuzzy genotype. Currently, the system is unable to apply evolutionary operators to the fuzzy sets themselves. These sets are manually defined, as seen in appendix A, page 91. With the ability of fuzzy set mutation and crossover, the system would be able to itself define and vary what constitutes 'close' distance or 'moderate' Hitpoints (HP), to name examples. It is however unknown how big impact such an implementation would have to the evolutionary performance and creature behaviour, but it could be worth investigating. The increased search-space caused by the introduction of more evolutionary variables may have a negative impact on computational performance, the number of sub-optimal solutions, and how fast the population would converge to an acceptable fitness level.

As they are now, the creatures have a fair share of entities available (see 4.1) for them to evolve and use to solve their given task. However, there are a lot of possibilities for new entities and improvements to the old ones. From the results of our experiments, it is clear that the nose entity does not perform as well as desired. The way it is implemented in the current system, is that it outputs a discrete value that coincides with the sum of good smelling objects with the bad smelling objects subtracted. The nose's functionality could be expanded to behave more like the eye entity does. Outputting more information regarding the position of the smelly object in a continuous interval.

When it comes to the evolutionary runs, the system has already been improved to be able to pause an ongoing run and resume it later. It would also be beneficial to be able to save an ongoing run in order to continue it at a later time. This would help in very long runs, as one could continue old short runs instead of starting from scratch every time. Having this functionality would improve experimentation with the system due to not having to decide on how many generations to run the evolution before it starts.

In the current system, a lot of the configuration are hidden away in separate files. Some of this has been alleviated with a Graphical User Interface (GUI), however there are still many options that can only be changed by editing the configuration files. Looking into how all of the configuration variables could be presented graphically and have them be easily editable would be something that could be done in the future.

Bibliography

Aguilar, W. and Pérez y Pérez, R. (2014). Criteria for evaluating early creative behavior in computational agents. In *5th International Conference on Computational Creativity (ICCC)*, Ljubljana, Slovenia.

Amabile, T. M. (1996). *Creativity in context: update to the social psychology of creativity.* Oxford Westview Press.

Amabile, T. M. and Collins, M. A. (1999). Motivation and creativity. In *Handbook of Creativity*, volume 297, pages 1051–1057. Cambridge University Press.

Beer, R. D. (1995). On the dynamics of small continuous-time recurrent neural networks. In *Adaptive Behavior*, volume 3, pages 469–509. Sage Publications.

Beer, R. D. (2006). Parameter space structure of continuous-time recurrent neural networks. In *Neural Computation*, volume 18, pages 3009–3051. MIT Press.

Binsted, K. (1996). *Machine humour: An implemented model of puns.* PhD thesis, University of Edinburgh.

Boden, M. (1998). Creativity and artificial intelligence. In *Artificial Intelligence*, volume 103, pages 347–356. Elsevier Science Publishing Company, Inc.

Cohen, H. (1995). The further exploits of aaron, painter. In *Stanford Humanities Review*, volume 4, pages 141–158. Stanford Humanities Review.

Cope, D. (1991). Computers and musical style. In *Computational Linguistics Volume 18 Number 4, Briefly Noted.* Oxford University Press.

Cope, D. (1996). *Experiments in Musical Intelligence.* A-R Editions, Wisconsin, USA.

Das, A. and Gambäck, B. (2014). Poetic machine: Computational creativity for automatic poetry generation in bengali. In *5th International Conference on Computational Creativity (ICCC)*, Ljubljana, Slovenia.

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence.* MIT Press, Massachusetts Institute of Technology, 1st edition.

Jordanous, A. K. (2013). *Evaluating computational creativity: a standardised procedure for evaluating creative systems and its application.* PhD thesis, University of Sussex.

Maher, M. L., Merrick, K., and Saunders, R. (2008). Achieving creative behaviour using curious learning agents. In *Spring Symposium on Creative Intelligent Systems*, Palo Alto, California. Association for the Advancement of Artificial Intelligence (AAAI).

Mamdani, E. H. and Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. In *International journal of man-machine studies*, volume 7, pages 1–13. Elsevier Science Publishing Company, Inc.

McCorduck, P. (1991). *AARON's Code: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen.* W.H. Freeman and Company, New York, United States.

Negnevitsky, M. (2005). *Artificial Intelligence - A Guide to Intelligent Systems.* Addison-Wesley, Essex, England.

Pease, A. and Colton, S. (2011). On impact and evaluation in computational creativity: A discussion of the turing test and an alternative proposal. In *Symposium on AI and Philosophy*, University of York. Society for the Study of Artificial Intelligence and the Simulation of Behaviour (AISB).

Razik, T. A. (1976). Programming creative behaviour. In *British Journal of Educational Technology*, volume 7, pages 5–21. Wiley-Blackwell.

Rojas, R. (1996). *Neural Networks - A Systematic Introduction.* Springer-Verlag, Berlin, Germany.

Russel, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach.* Pearson, Essex, England, 3rd new international edition.

Saunders, R. (2001). *Curious design agents and artificial creativity.* PhD thesis, University of Sydney.

Stanley, K. O., Bryant, B. D., Karpov, I., and Miikkulainen, R. (2006). Real-time evolution of neural networks in the nero video game. In *AAAI*, volume 6, pages 1671–1674.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. In *Evolutionary computation*, volume 10, pages 99–127. MIT Press.

Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. In *IEEE Transactions on systems, man, and cybernetics*, number 1, pages 116–132. Institute of Electrical and Electronics Engineers (IEEE).

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD thesis, King's College, Cambridge.

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University.

Appendices

A. Static fuzzy sets

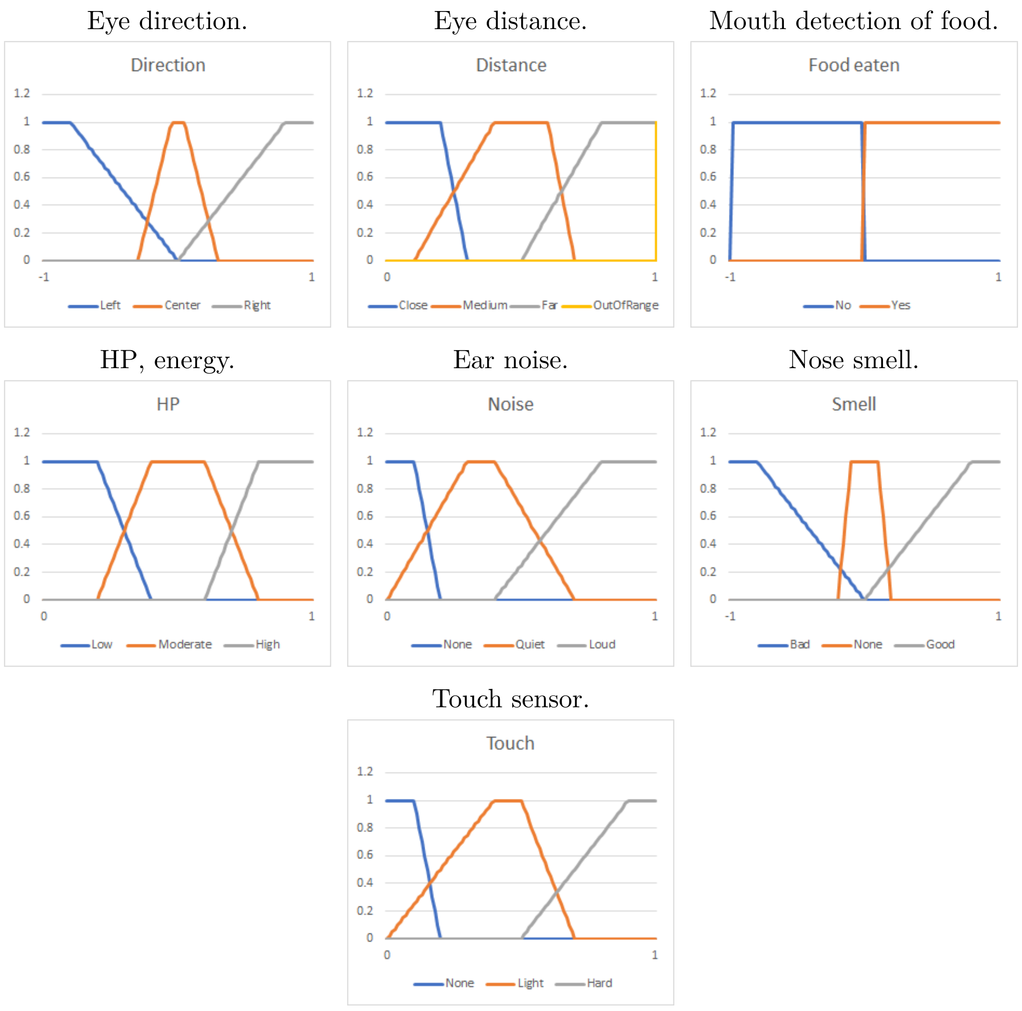


Figure 1.: Static fuzzy sets used in the fuzzy brain-modules, referencing sensor variables of the different entities. These sets are defined at compile time and are not changed during system execution.

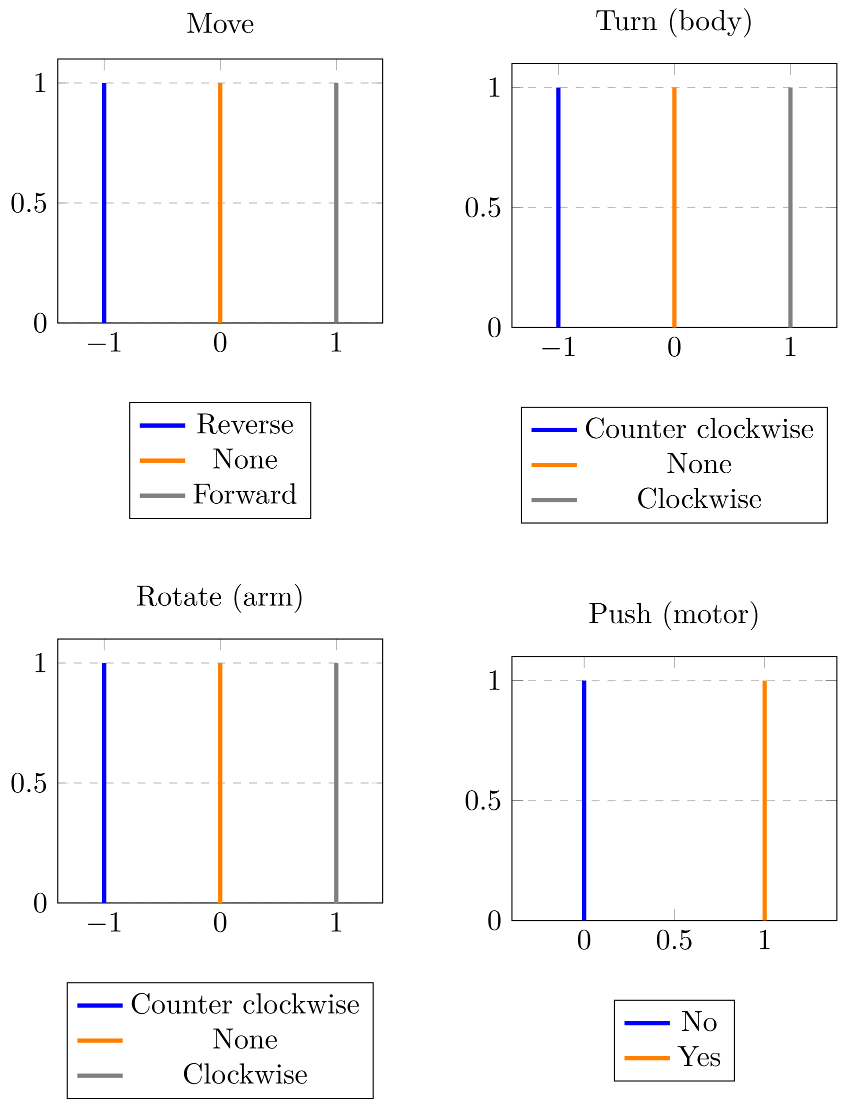


Figure 2.: Static sugeno fuzzy sets for every action of all actuator entities. These are defined at compile time and are not changed during system execution.

B. Configuration files

B.1. CreBe general configuration

Listing 1: General configuration of the CreBe system, used in all designed and full-evolution experiments.

```
# mutation factor for entity mutations
mutation_factor      = 0.8
# number of entities on creature at init (max 8)
num_start_entities   = 6
# max number of ticks during fitness eval
simulation_ticks     = 9999
# start energy of creatures
start_energy         = 2000
# amount of energy given per food
food_nourishment     = 200
# ticks between each food spawn
food_spawnrate       = 0
# number of foods spawned at startup
num_start_foods      = 100
# the fraction of num_start_foods that should be poison
poison_ratio         = 0.5
# number of simulation runs to perform per fitness eval
fitness_eval_runs    = 3
# see WorldType enum for possible values
world_type           = <different values>

# prob of removing already existing entity
ent_remove_mp        = 0.02
# prob of change an entity to something else
ent_changetype_mp    = 0.02
# prob of adding new entity in empty attach point
ent_add_mp           = 0.3
# default behaviour is to retain the entity

# neural network
ann_hidden_config    = 6,6
activation_function = mod_sigmoid

# creature export dir
export_dir           = exported

# entity mutate factors
```

```
arm_length_mf          = 10
arm_speed_mf           = 15
arm_strength_mf        = 0.05
arm_jointangle_mf      = 10
body_radius_mf         = 5
body_rot_speed_mf      = 0.02
ear_range_mf           = 50
eye_fov_mf             = 1
eye_range_mf           = 50
eye_measdist_mf        = 0.05
fork_forkangle_mf      = 12
fork_armlength_mf      = 8
motor_speed_mf         = 10
motor_strength_mf      = 0.1
mouth_canshoot_mf      = 0.05
mouth_shootrange_mf    = 20
mouth_shootdmg_mf      = 0.1
nose_range_mf          = 60
spike_length_mf        = 4
spike_damage_mf        = 0.1
```

B.2. Designed experiments configuration

Listing 2: Evolutionary Algorithm (EA) configuration file used in the designed-creature experiments.

```
# required config
population_size       = 200
overprod_factor       = 1.0
target_fitness        = 99999
max_generations       = 100
crossover_type        = UNIFORM
adult_selection       = GENERATIONAL_MIXING
parent_selection      = PROPORTIONATE
crossover_rate        = 0.5
mutation_rate         = 1.0
tournament_size       = 10
tournament_prob       = 0.8
inversefitness        = false

weight_mutation_std = 0.3

# optional config
elites                = 1
verbose               = false
quiet                 = false
threads               = 0
dynamicworkload       = true
recalculate_fitness = false
```

B.3. Full-evolution experiments configuration

Listing 3: EA configuration file used in the evolution of physical creature entities in the full-evolution experiments.

```
# required config
population_size     = 16
overprod_factor     = 1.0
target_fitness      = 99999
max_generations     = 50
crossover_type      = UNIFORM
adult_selection     = GENERATIONAL_MIXING
parent_selection    = PROPORTIONATE
crossover_rate      = 0.5
mutation_rate       = 0.5
tournament_size     = 10
tournament_prob     = 0.8
inversefitness      = false

# optional config
elites              = 0
verbose             = false
quiet               = false
threads             = 0
dynamicworkload     = true
recalculate_fitness = false
```

Listing 4: EA configuration file used in the behaviour evolution part of the full-evolution experiments.

```
# required config
population_size     = 20
overprod_factor     = 1.0
target_fitness      = 99999
max_generations     = 20
crossover_type      = UNIFORM
adult_selection     = GENERATIONAL_MIXING
parent_selection    = PROPORTIONATE
crossover_rate      = 0.5
mutation_rate       = 1.0
tournament_size     = 10
tournament_prob     = 0.8
inversefitness      = false

weight_mutation_std = 0.3

# optional config
elites              = 1
verbose             = false
quiet               = false
threads             = 0
dynamicworkload     = true
recalculate_fitness = false
```

B.4. Nose-navigation configuration

Listing 5: Configuration used in the designed nose-navigation experiment.

```
#Mon May 22 11:25:15 CEST 2017
parent_selection=TOURNAMENT
inversefitness=false
quiet=true
max_generations=2000
population_size=50
mutation_rate=1.0
crossover_rate=0.5
dynamicworkload=true
verbose=true
overprod_factor=1.0
crossover_type=UNIFORM
tournament_prob=0.8
target_fitness=99999
elites=0
adult_selection=GENERATIONAL_MIXING
tournament_size=10
recalculate_fitness=false
threads=0
weight_mutation_std=0.3
```

C. Creative Creature Behaviour (CreBe) software guide

This is a short documentation on how to setup and run the Creative Creature Behaviour (CreBe) software system. This includes installing the Java runtime environment, run the system and how to configure the various software parameters.

System requirements, setup and installation

The CreBe software is written in the Java programming language. In order to run this software, the Java Runtime Environment (JRE) or Java Development Kit (JDK) needs to be installed on the system. However, JDK is only needed if you intend to alter the source code and recompile the software. The JRE is sufficient for simply running the software.

The newest version of JRE and JDK can be downloaded from Oracle's website:

- JRE: https://java.com/en/download/
- JDK: http://www.oracle.com/technetwork/java/javase/downloads/index.html

It is recommended to always use the newest version of Java provided by Oracle.

Running the software

Once Java (either JRE or JDK) is installed on the host system, the CreBe software can be started by simply double-clicking the *CreativeBehaviour.jar* file. When run for the first time, five configuration files will be automatically generated and placed in the same folder as *CreativeBehaviour.jar*. These files contain all configuration for the software system, and are initialised to some standard configuration.

The software can also be started from command line, as

```
java -jar CreativeBehaviour.jar <params…>
```

Note that starting from command line require Java to reside in the system PATH-variable.

Possible parameters for the software are:

Parameter	Description
-debug	Start the system in debug mode (visible sensory ranges, brain-module input/output values etc.)
-headless	Start the system in headless mode, meaning no GUI will be shown. Creatures are automatically exported to disc. Suitable for servers where no graphics context is available.
-seed	Sets the seed of the random number generator. Same seed will always produce the same simulation results. Suitable for development and during debugging. **Note** that this parameter only works as intended in single thread operation.
-presentation	Presentation of one or more creatures. No simulation or evolution is performed. Specify creatures to present using **-designed <txt_files>** where txt_files are the list of manually defined creatures to present.

Figure 1 shows the main window of the application.

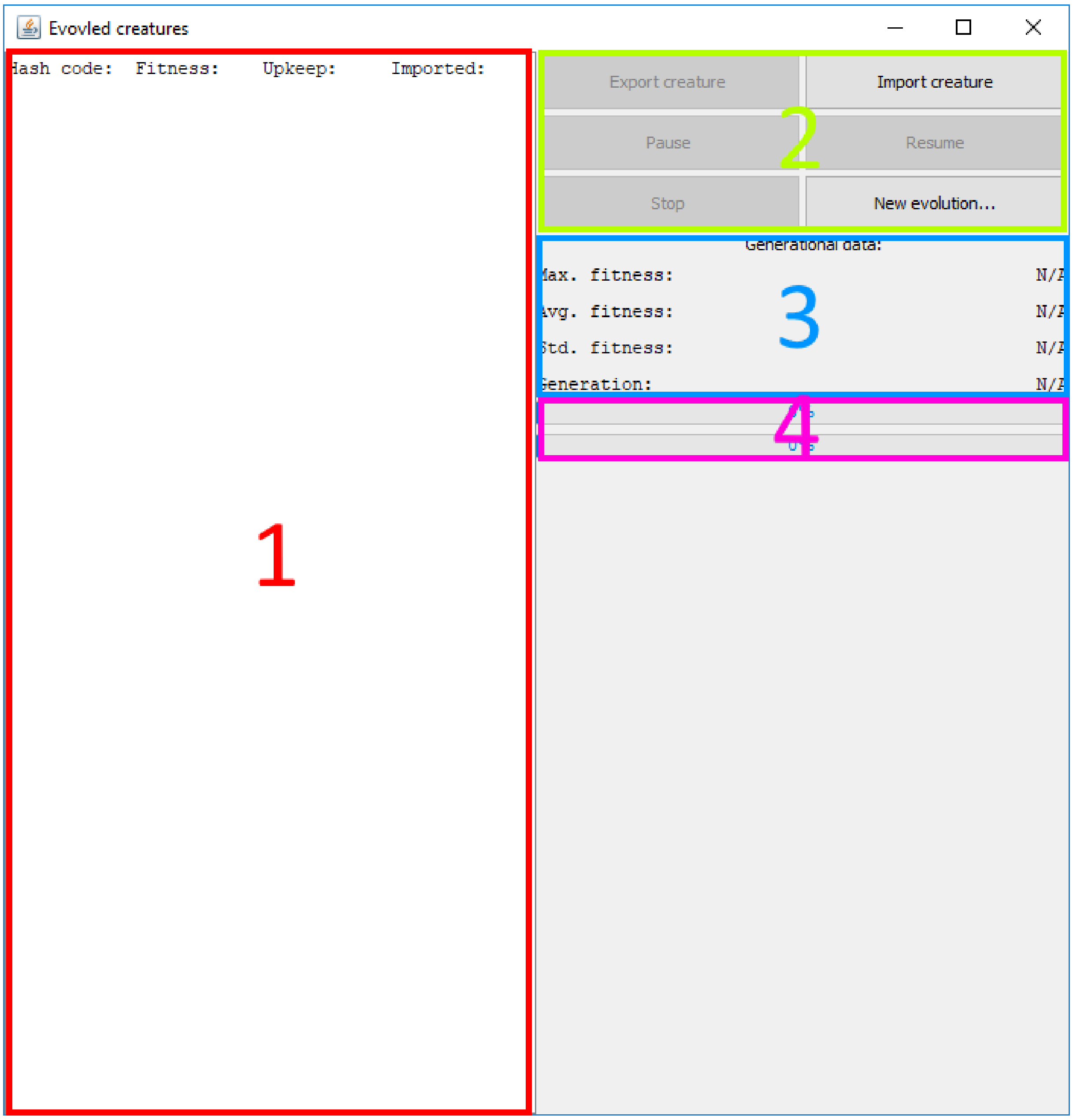
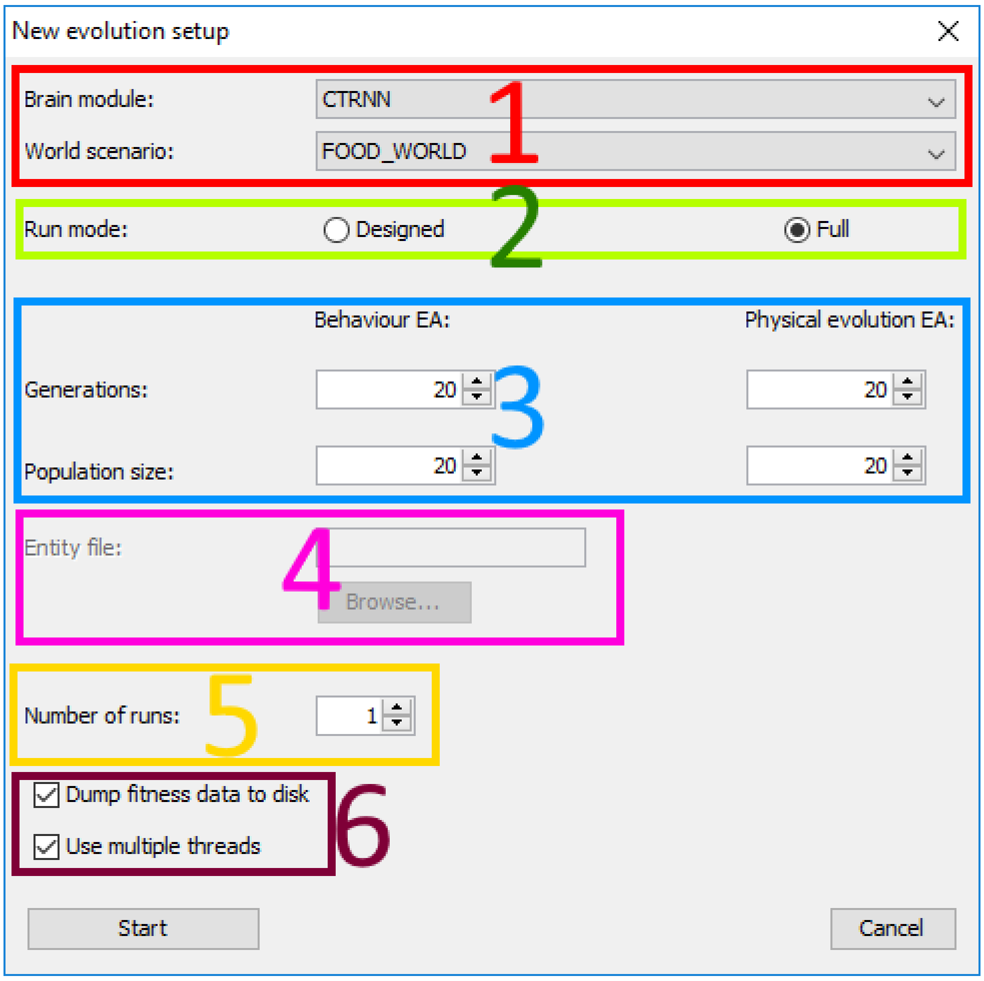


Figure 1: Screenshot of the main window. Consult the following table for a description of the various parts.

	Main window explanation
1	Evolved creatures window. When a new best creature is found, it is added to this list. Double click on an element in this list to run a simulation of that creature.
2	Control buttons. Export a creature to disk by selecting it from the creature list, then click the “Export creature” button. Start a new run by clicking “New evolution…”, which brings up the configuration window.
3	Live telemetry from the current evolution: maximum fitness, average fitness, fitness standard deviation and current generation.
4	Progress bars for the current evolution. The upper progress bar applies to the current run, the lower applies to the whole evolution.

System configuration

Clicking the "New evolution…" button brings up the configuration window, and the ability to start a new evolution. Some configuration options are not available in this user interface, and must be manually specified in the configuration files before the application is started.


New evolution setup

1

Brain module: CTRNN

World scenario: FOOD_WORLD

2

Run mode: Designed [✓ Full]

3

	Behaviour EA:	Physical evolution EA:
Generations:	20	20
Population size:	20	20

4

Entity file:

5

Number of runs: 1

6

[✓] Dump fitness data to disk

[✓] Use multiple threads

Figure 2: Configuration window.

	Configuration window explanation
1	Brain module and world scenario selection.
2	Run mode. Designed: evolve behaviour only. Use a predefined set of entities for the creature. Full: evolve both behaviour and entities.
3	Population size and number of generations. Separate configuration for behaviour and physical evolution.
4	If run mode is set to "Designed", select a text file containing the entity specification.
5	Number of runs to perform (repetitions).
6	"Dump fitness data to disk" will dump maximum, average and fitness standard deviation to log files. "Use multiple threads" will allow the system to utilize multiple processor cores.

In order to run a simulation of a creature, double click on some creature in the creature list. The simulation will begin automatically, using the world scenario that was used to evolve that particular creature. Figure 3 shows a simulation example.

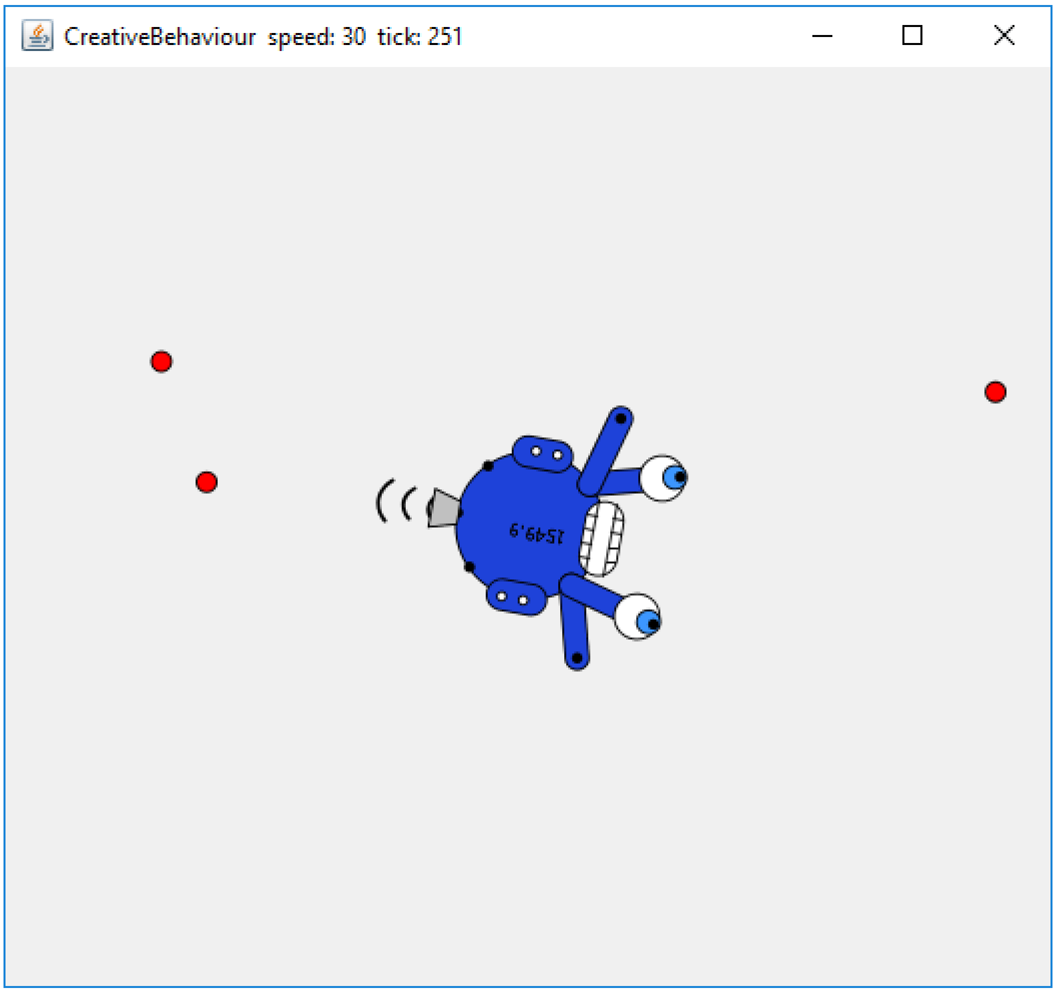


Figure 3: Simulation of a creature.

Control the simulation with the following keys:

Key	Description
D	Toggle debug information, such as sensor ranges, eye field-of-view, actuator activation values, etc.
V	Centre the camera and follow a creature. If multiple creatures are present in the world, repeatedly press to cycle through all creatures.
+	Speed up simulation.
-	Slow down simulation.
Drag with mouse	Look around the world (disabled if camera is centred on a creature).
Mouse scroll wheel	Zoom in and out.