Toward tool use in species:
The emergence of tool use in different worlds

Timo Veldt

January 21, 2008

Timo Veldt
Studentnumber: 0413615
Email: T.Veldt@student.ru.nl
Supervised by Dr. I Sprinkhuizen-Kuyper
Faculty of Social Sciences
Radboud University Nijmegen

Acknowledgements

I would like to thank my supervisor, Dr. I. Sprinkhuizen-Kuyper, for helping me when neccesairy. I would also like to thank Dr. W. Haselager, for giving some useful comments on my writing. For helping me with some programming in JAVA, I would like to thank Dr. F. Grootjen. For checking my thesis, I thank Marleen Nicolai.

Abstract

In this bachelor thesis I describe my investigations of the use of tools in animal species in the simulation environment Framsticks. The choice for Framsticks is described in detail and experiences with the simulator are documented in this thesis. By conducting several simulations, I investigate if it is possible to find out in what kind of environments tool use evolves. The difference between the worlds in the experiments is the different amounts of food present. The results show that only in a world with very little food, tool use emerges.

The choice for Framsticks is described in detail and experiences with the simulator are documented in this thesis.

1 Introduction

The research described in this bachelor thesis is inspired by the paper by McGrew et al. (1979), where an extensive study is performed among chimpanzees to discover if tool using behavior occurred in a similar fashion in multiple populations. Other papers describe tool use in animal species, for instance (Alcock, 1972) and (Lefebvre et al., 2002). These papers all focus on one ore more specific species or on several groups of the same species.

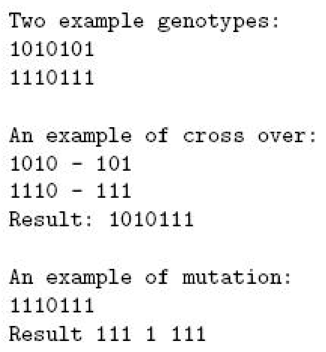
I will focus on the effect of the amount of food on learning to use a tool that will give a better foraging capability. The definition of using a tool is subject of a discussion which basically has three points of view (Alcock, 1972). Firstly, there are those who think that using any external or internal object in any way can be considered as using a tool. The second view allows only for the use of active and passive external objects, while internal objects are not considered tools. The last view only considers active external objects as tools.

An example of an internal object could be the spider's web. The material is manufactured inside the spider's body, thus is considered to be an internal object. External objects as described above, come in two classes, passively used and actively used objects. An example of passive use is a gull that drops an egg on a rock to eat the inside of the egg. Here the rock can be viewed as a passive tool. An example of active use is for instance a vulture that smashes the egg with a stone held in its beak (Lefebvre et al., 2002). In my simulations, I will use a twig that individuals can pick up and use to gather more food, which in this case belongs to the category of active, external tools.

There is a more extreme view as well, which postulates that a tool can not be any actively used external object, but that only external objects, which have been modified by the individual can be considered tools (McGrew et al., 1979). I can understand this view, but believe that the modification of the object is no prerequisite to consider the object a tool. In my opinion, a tool is used to simplify a task, which does not require the object to be modified. The vulture that breaks an egg with a rock in it's beak is using a tool, because the rock makes it easier to get inside the egg. Since the rock has not been modified, McGrew et al. (1979) amongst others would not consider this rock a tool, but the rock is used to make a task simpler, so it can be considered a tool.

Evolutionary robotics

The means I will use to test my hypotheses about tool use (described further on), are evolutionary robotics (Harvey et al., 2005). Evolutionary robotics is based on ideas from evolution theory. A genotype is defined, which encodes a certain possible solution to a problem. There are several applications which use this theory, the most important class of problems for this bachelor thesis are optimization problems. The goal is to find a solution that offers better results, amongst other solutions that also "solve" the problem, but in a less efficient way. An important aspect of evolutionary robotics is defining a genotype, which is a description of how the problem should be solved. In robotics, there are several parameters that can be included in the genotype, such as body form, brain structure and weights of a neural network. When body form is included in the genotype, it becomes possible to change the body via evolutionary operations, such as mutation and crossover (discussed later on). Using these operations, new genotypes are created or in other words, new solutions. The newly created solutions are tested and receive a score or fitness value based on their performance. The term fitness has a foundation in biological evolution and is used to determine which genotypes are used to create new genotypes. Usually, the solutions with a higher fitness also have a higher chance to reproduce (survival of the fittest). The operators make adjustments to a genotype when it is reproduced, in such a way that the result is a new solution. Crossover creates a new genotype, based on two parent genotypes. Both genotypes are cut off at a random point and two of the four resulting parts are used to form a new genotype. Mutation alters a genotype slightly and is based solely on chance. This is useful, because when a certain property is not present in the solutions thus far, it can not be created by crossover but it can appear through mutations.



```
Two example genotypes:
1010101
1110111

An example of cross over:
1010 - 101
1110 - 111
Result: 1010111

An example of mutation:
1110111
Result 111 1 111
```

Figure 1: Examples of evolutionary operations

In Figure 1 a few examples of evolutionary operation are given. The two genotypes used are defined by a string of 7 bits, which could be interpreted as follows: when the value of a certain bit is 1, the solution has the property associated with this bit, when the bit has a value 0, the genotype does not have the property associated with the bit[1]. These operations are just the basics of evolutionary robotics and there are a lot of variations possible on each rule. For

[1]If the first bit stands for "motion", it means that an individual will be able to move if the bit value is 1, else it cannot move.

a more detailed description see Eiben and Smith (2007).

Evolutionary robotics will provide the possibilities to test individuals and give them a score based on the amount of food they find. This form of robotics also allows for easy modification of genotypes, thus give the possibility to discard bad changes and keep modifications that result in a better performance in the population. Another reason to use evolutionary robotics is given by Alcock (1972). He points out that it is impossible to discover the true source of the usage of tools in all animal species that use tools at this moment. He suggests two possible sources for tool using behavior in primates, but also states that even if one of these theories proves to be correct, it will only be for a certain group of a certain species and might not be applicable to other groups of the same species, let alone to other species. However, it is still possible to form hypotheses concerning the conditions in which tool use might emerge, which will be done later on. This possibility exists, because it might not lead us to the source of tool use in primates or other species, but it might give us an idea why tool use may have emerged at all. Because the source of a trait is impossible to determine, evolutionary robotics provides a perfect way to develop creatures, since this method does not depend on any rules that need information on how the task is done or what the source of certain behavior is.

A theory from anthropology

An interesting theory Alcock (1972) provides is that there are three stages in the process of learning to use tools.

> *"(1) The initial appearance of the trait in a population of animals, (2) the initial transmission of the trait through the population, and (3) the subsequent evolution of the behavior over time."* Alcock (1972)

With these three points in mind, it seems obvious to choose evolutionary robotics. The first step can be accomplished by evolutionary learning, giving rise to new characteristics in individuals, while the second step is also achieved through evolutionary learning and learning in the population, allowing for the spreading of a characteristic and the third step allows for sophistication of the characteristic. All steps also can be argued to have a learning component in them, which can be responsible for the experience an individual gains through using a particular skill.

Research questions

In my experiment I will focus on step 1 of the theory by Alcock (1972), the initial appearance of a trait in the population and more specifically, the use of a tool to gather food more efficiently. There are two aspects which will be looked into:

1. I will investigate whether or not the amount of food present in the environment will have an influence on the appearance of the behavior altogether.

2. The influence on the time it takes to reach a reasonable skill with the tool.

My hypothesis is that in a world with food readily available, the behavior of the individuals will be less efficient than the behavior of individuals in world with less food available. Individuals evolved in a world with little food available, are less certain of finding another food source, so should leave no food unfinished.

The conditions will be defined in such a way that fitness values between these conditions are almost the same, but the way these values are achieved should differ between the conditions. In other words, the individuals in the condition with much food could be driving around almost random, using no tools, while the individuals in the condition with little food will be actively searching for a tool and finishing food sources first before moving on to the next food source only when the closest food source is depleted.

My research question can be formulated as follows: is there a stronger evolutionary pressure to use a tool, that improves the amount of food gathered, in a world with a lower food density than in a world with plenty food available? To answer this question, I first describe the preparations of the experiments in Section 2 and give more details about these experiments in Appendix B. The actual experiments are described in Section 3. The results of the experiments will be presented after this in Section 4 and a discussion of these results in Section 5. I end with my conclusions in Section 6.

2 Preparations of the experiments

For conducting experiments, I had to decide whether to use real robots or a simulator. I chose for simulations, since it works much faster than real world experiments. Simulations seemed a good choice, since I wanted to run multiple conditions with randomly placed individuals and food. The next choice I had to make, was on the simulator I was going to use. After an investigation on the web, I examined two simulators in detail, Framsticks (Adamatzky, 2000) and EvoRobot (Nolfi, 2000), since these two have been used in several other experiments (see e.g. (Nolfi, 1997) and (Jelonek and Komosinski, 2006)). These experiments have some similarities with the experiments I intend to conduct.

In the following sections I will describe the process of choosing between simulators (Section 2.1) and a pilot study of the simulator I chose (Section 2.2).

2.1 Choosing a simulator

I composed a list of criteria which a simulator should meet to be a possible candidate. The list is presented below followed by a brief description of how and where these points are present or absent in Framsticks and EvoRobot. After this description, I will explain why I chose Framsticks.

- Designing experiments: the simulator should allow for easy adjustments of experiments, since I want a few things that probably will not be present in every simulator.
- Creating individuals: The control structure should allow for a sort of memory, so a creature knows and remembers it has a twig. For instance, I could use a recurrent layer or some sort of switch which can be on when the individual has a twig and off when it does not have a twig.

- Learning individuals: It would be preferable if the individuals have the opportunity to learn in a run, but this is not a requirement for the first step of Alcocks theory.
- Multiple individuals: It would be less important for this experiment, but good for future research, when the simulator allows for multiple individuals to be present in the world at the same time.

Of course there are a few other points, such as the program being stable, but these are more general demands and although important for this experiment, the points above will have to be present so that the experiment in itself can be conducted.

Designing experiments

Framsticks allows for the development of an experiment in a much orderly fashion than EvoRobot. Framsticks is pre-compiled and has several types of files that can be added to the program by the user, including adjustments to selection mechanisms, new neuron definitions and different groups of creatures. In appendix A, I have put together a brief manual for the use of Framsticks based on some of my own experiments since I found the documentation on the Framsticks site (Komosinski and Ulatowski, 1997) a little confusing at some points. EvoRobot has its source code available, but has no documentation on these files, making it pretty hard to adjust it for a specific experiment that differs from the rules already present.

Creating individuals

EvoRobot uses a model of the Khepera robot (Mondada et al., 1993), limiting adjustments to the body to modifications of the control structure. These modifications are further restricted, since the number of hidden units is the only property that can be set. There is an option for a gripper on the robot, which allows the robot to grab an object. This seems a good option for picking up a twig and doing something with it, but the representation of this system does not allow for any subtle use. The control structure is basically a perceptron with an input, an output and a hidden layer. The Khepera robot has enough sensors, but these are hard to adjust and located in a fixed place. The sensors are all sensible to infrared vision, which makes it harder to build a good world in which it is easy to discern the difference between food and twigs.

The individuals in Framsticks are free to creation and modification. A creature is built from a genotype, which is constructed by the user in a special, easy language which allows for placement of neurons at specific places and adjusting properties where needed. This allows for very simple design of the body, while at the same time allowing development of all kinds of control structures. There is a Khepera robot genotype available on the internet as well (Virtual Life Lab, 2007). But if you want to use Khepera robots, I believe it is best to use EvoRobot, since it allows for easy transfer from data of the simulation to the real robots and vice versa. Because of the built-in freedom in control structure, I believed Framsticks was the best choice, since it allows for a recurrent network, which in turn, allows for a "memory" for the robot to know whether or not it is holding a twig.

Learning individuals

Framsticks does not support learning during the lifespan of an individual. It is possible to implement it, but the way the neurons communicate, does not make it easy to do so (Komosinski, 2000). Framsticks offers another way to make the changes in weights through evolution. Via crossover and mutations, it is possible to create the desired weights. EvoRobot has Hebbian and automatic learning built-in, which might be important in the second step of Alcocks theory, but I believe that the appearance of a trait can be achieved through evolution only. Evolution will give a high score to a genotype that has learned to use the stick, while giving a low score to a genotype that has not yet learned to use the best strategy for a certain task. An individual with a bad score will eventually be removed from the gene pool, while a good scoring individual is evolved in different ways to see if change in weights causes an improvement of the current score. This results in learning through individuals in a gene pool instead of learning within one individual.

Multiple individuals

Framsticks allows for multiple individuals to be present in the world. In EvoRobot, it might be possible to construct this mechanism, but this is difficult given the undocumented source code. There is a version of EvoRobot that allows for multiple individuals, but this version is not available for research. Framsticks has a more prominent place for this feature. Although this probably seems the least important feature for *this* experiment and it probably is, concerning the contents of Alcock's first step. Experimenting with multiple individuals becomes relevant in the second step of Alcocks theory mentioned in Section 1.

Choice for Framsticks

Given the flexibility of experiment designing and the freedom in creating bodies and control structures, I decided to use Framsticks. This leaves me with the problem that individuals can not learn during their runs, but this seems less important in this first step toward the goal, the first step out of three steps from Alcock's theory. Another positive side of Framsticks is that it also allows for multiple individuals, which, again, is not very important for *this* experiment, but I believe that building a learning component into Framsticks is easier than building multiple individuals in EvoRobot. Furthermore, Framsticks already has multiple file types which can be added and deleted in an easy fashion, making it easy to store certain settings in a new file or re-using old files in a new experiment. Another reason why I chose Framsticks is the master thesis of Van den Braak (van den Braak, 2005). She concludes that the incomplete version of EvoRobot that is available on the web, is not capable of reproducing Nolfis results from Nolfi and Floreano (2000). One of the causes is, according to van den Braak (2005), the simplicity of the Khepera robots that are simulated. In the experiments by van den Braak (2005) predators were not good at differentiating between walls and prey, since the Khepera robot has a limited sensor system.

2.2 Pilot study of Framsticks

To investigate whether Framsticks was indeed a good choice, I decided to test the simulator by doing a small experiment. In order to do this, several steps had to be taken. The first step was to make a proper genotype. I chose for a simple structure, that allows for visual identification of the “head” and “tail” (see Figure 2). There are two wheels and four sensors on the body. The neural network consists of an input layer, a hidden layer, a recurrent layer and an output layer (see Figure 3). The input of the network is provided by the four sensors on the body (two sensors on the left side and two on the right side of the head, with one food sensor and one stick sensor on each side). The output was directed to the two wheels that moved the body around. For a more detailed description of the pilot study and the genotype used, see Appendix B. The evolutionary parameters were all set within the boundaries of the program, but the experiment definition was slightly modified (see Appendix A for more information on experiment definitions).

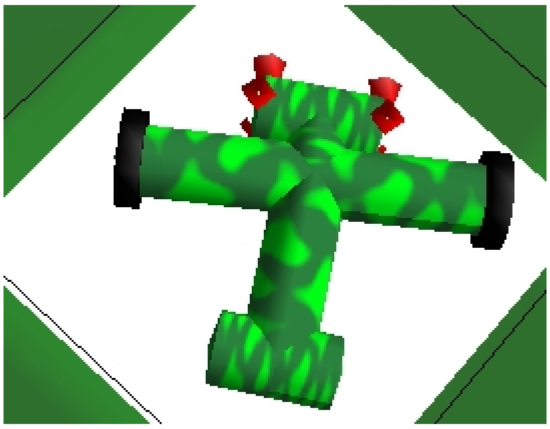


Figure 2: The genotype in 3D

Parameter settings

The worldsize was 20, the default value. I used teleport borders[2], which means that an individual crossing a border of the world is teleported to the other side of the world where it enters the world again. The option of no boundaries was ruled out, since the amount of food present in the world is not infinite. The ratio of foodballs[3] over area of the world is important for my experiments, so I wanted to create an environment which was as close to the final experiments as possible. I included two foodballs but no twigs.

The only thing allowed to change in evolution were the connection weights and the only way to change the weights was through mutation. When a certain genotype gets mutated, a new genotype is created and the “parent” genotype

[2]Also known as a Torus World, which is derived from the mathematical shape Torus (donut-like).

[3]A foodball is the term used in Framsticks literature to indicate a unit of food and I will use this term as well.

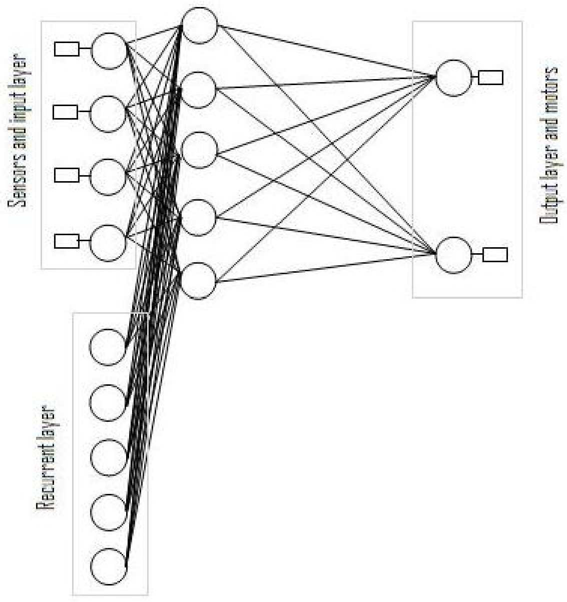


Figure 3: Schematic view of the Neural Network.
The recurrent layer is positioned under the input layer.

remains in the gene pool. The new genotype is immediately used to create the next individual. When this individual performs better than the worst individual in the population, the worst individual is deleted. If this was the only individual from its genotype, the entire genotype is also removed from the gene pool. If there are other individuals left from that genotype, it remains in the gene pool. The newly created individual is stored in the population and its genotype is stored in the gene pool, using the values of the individual for the genotype.

If there was no mutation, the selected genotype is used to create the next individual. Again, when this individual performs better than the worst individual in the population, the worst individual is replaced by the new individual. When the worst individual was the last of its genotype, the genotype is also deleted from the gene pool. The performance of the new individual will be used to calculate the mean fitness of its genotype, which consists of the mean fitness of all individuals still in the population[4].

When a genotype was selected, there was a 30% chance that it would be mutated into a new genotype. When it got mutated, the relative probability for a change in a connection weight was set to 10%. This means that each weight had a chance of 3% to mutate. How a connection actually gets mutated is unclear, but through several observation, it appears that a random value proportional to the original value is added to the connection weight[5].

Framsticks offers evolution of body, brain structure and neuron parameters. I created a fixed body, which should be able to perform the task. This speeds

[4]In Section 5 this way of calculating the mean fitness is discussed further.

[5]In these observations, one weight of the genotype would change from, for instance, 1.00 to 1.03.

up evolution considerably, because if the body is allowed to change, it could well be that the wheels get placed in such a way that the creature is unable to move. This sort of change makes the process slower than necessary, generating all kinds of genotypes that will not be able to handle the task.

Other important parameters of the simulator were:

- Population size of 100. This influences the flow of the evolutionary process through limiting the number of individuals possible. When this value is high, more individuals will be remembered and thus a genotype can stay longer in the gene pool, making it possible to select and mutate it again. This requires more harddisk space etc.
- Delete only the worst instances from the gene pool.
- Tournament Selection with 5 instances.
- When selecting an individual for creation, mutate it 30% of all selections, leave 70% of the selections unchanged.
- The fitness was set to equal the lifespan of the individuals, i.e. the longer an individual lived, the higher its fitness value. This fitness value is influenced by the amount of food an individual consumes, since it needs food to survive.
- I did not define a stop condition, but let the simulator run for about one week.

The simulator ran pretty stable for a week and when I checked the results, I noticed that there was some growth in fitness each time. I will describe these results in the next section.

2.2.1 Results of the pilot study

The experiment provided results in terms of the lifespan of an individual, so the longer the lifespan, the better the individual had performed (i.e. I used the lifespan as the fitness of an individual). This mapping is one to one, so if an individual had a lifespan of 1000, its fitness was 1000 as well. By collecting foodballs, the lifespan of the individual could be prolonged. The results proved satisfactory in that after a short period of time, the genotypes had evolved to a point were the fitness had risen considerably, but more interesting was the behavior of the individuals created. A strategy seemed to have appeared, since the individuals would not drive around randomly, but home in on food and finish the entire portion before moving on. Given the relatively short period of time this experiment took, this appeared to be promising for longer runs. See Appendix B for more details on this pilot study.

3 The experiments

In this section I will give a short summary of the most important parameters I used and changed in my conditions in Section 3.1. After this summary, a description of how the conditions were determined and why these conditions were chosen is presented in Section 3.2.

3.1 Important Parameters

The number of parameters Framsticks offers is substantial and sometimes confusing. To make things more clear, this sections gives a list of parameters that were important to this experiment. Some of these parameters may have already been discussed in previous sections and others will be described later in this section as well. The parameters are discussed in categories such as “Food” and “Twigs”, where everything that has to do with food will be listed under “Food”.

3.1.1 Evolution

- *Gene pool* This term is used to describe the pool of all genotypes that are currently in the simulation. The minimum of genotypes in all conditions is 1, since all individuals have to be created from at least one genotype. The maximum number of genotypes is equal to the population size, when all individuals have been created from a different genotype.
- *Population and population size* These terms are used with respect to individuals. The population contains the best individuals created at that moment. The population size is used to set the maximum number of individuals stored. When this number is raised, each individual can stay in the population longer before it becomes the worst individual and is deleted.
- *Individuals* An individual is an instance of its genotype. In other words, an individual is an actual creature, with the properties as described by its genotype.

3.1.2 Individuals

- *Genotype* The genotype in Framsticks contains all the information on how an individual should be constructed. Framsticks has several languages in which a genotype can be written. The most important part in Framsticks is the stick, which is the basic building block for a body. The genotype used, contains a number of sticks connected in a certain way, the number of neurons and how these neurons are connected, the sensors and actuators. For more on the construction of genotypes see the Framsticks website (Komosinski and Ulatowski, 1997).
- *Starting Energy* The individuals get a certain amount of energy when they are created. This number is determined by the number of sticks in the genotype. The starting energy defines the amount of energy units the individual for each stick.
- *Idle Metabolism* The individuals burn energy at all times. The amount of energy burned in each step is defined by the number of sticks in the genotype. For each stick, the individual looses one energy unit each step.

3.1.3 Food

- *Genotype* Foodballs have a genotype, which by default results in an apple which lies on the ground. The user can define other food sources, including moving objects.

- *Starting energy* The amount of energy units a foodball contains and thus the maximum amount of energy units that can be gained by an individual when it eats the entire foodball.
- *Automatic feeding* This makes sure that there is a constant number of foodballs in the world. When one foodball is depleted, the program adds a new foodball in a random place.
- *Ingestion Multiplier* To get energy from a foodball, an individual has to bump in to a foodball. The amount of energy gained, is determined by the number of parts of the body that hit the foodball. The ingestion multiplier is a factor that allows the user to influence the amount of energy units gained for each body part that hits the foodball.

3.1.4 Twigs

- *Number of twigs* The simulator provides a fixed number of twigs in the world. When an individual picks up a twig, that twig disappears and a new one is placed in a random position.
- *Picking up a twig* An individual can pick up a twig by bumping into it. The twig disappears and the simulator registers that the individual has picked up a twig.
- *Using the twig* When an individual has picked up a twig, it never loses the twig. An individual does not have to manipulate the twig, it is assumed that when the individual has a twig, it always uses the twig. The experiments focus thus on using a twig or not using a twig, but not on how the twig is actually manipulated to gain extra energy units.
- *Efficiency with the twig* When an individual hits a foodball, it gains an amount of energy units. When it is holding a twig, this amount of energy units gets multiplied with a predetermined number. In this experiment, the individuals gained four times as much energy.

3.2 Description of the conditions

As the results of the pilot study were satisfactory, I had to decide how I would actually test my hypothesis. I chose for three conditions, one in which a predetermined amount of food would be placed, where the amount of food would be sufficient to give the individuals the possibility to locate food in an easy fashion (how these amounts were determined is described further on). This condition will be referred to by condition 2. The other two conditions would have a different amount, one with less than the amount of the first condition (referred to by condition 1), the other with more (condition 3). This should allow to see the differences in behavior caused by the amount of food present.

To determine what a good amount of food would be for the second condition, I took the genotype and took several measurements on the performance of this genotype. I put the input of motors both at their maximum and looked how much distance it would cover in its lifespan of 1000. This measurement did not seem very reliable and I took a different approach.

I calculated that to survive indefinitely, an individual, on average, should pick up 8 energy units in each simulator step, because this is the amount it uses each step[6]. The amount of energy stored in a foodball was set to 1000 energy units. To make sure the individuals would be able to search for the next foodball (meaning that they would have enough energy to drive from one ball to the next), I estimated that 50 foodballs in a world with world size 55 would be an environment where the individuals could learn to survive indefinitely without much trouble.

The twigs are supposed to make the individual more efficient. This was accomplished by storing whether or not an individual had picked up a twig and changing the procedure for assimilating food. When an individual had a twig, it got four times the amount of food from a collision with a foodball, opposed to the default value of one when it did not have a twig in possession. The amount of energy an individual gains in a collision, is subtracted from the energy of the foodball. When more energy is subtracted, the foodball will be depleted sooner. This makes an individual more efficient, because it needs less time to consume one entire foodball. This means that it loses less energy (caused by idle metabolism) while getting more energy (i.e. a more positive energy balance). This makes it possible to search a little longer for the next foodball.

The other two conditions were set at 40 foodballs in condition 1 (-20%) and at 60 foodballs (+20%) in condition 3. As an extra test I also made an extra condition in which only 5 foodballs where present. This condition will be referred to by condition 4. There were two twigs in each environment.

Other important settings were the same as in the pilot study (see Section 2.2 for more information), but a few things changed:

- Tournament selection was set to a tournament size of 3 instead of 5.
- Let each new genotype run at least 10 times before mutating it or selecting another. This way, the genotypes get a more representative score.
- When selecting an individual for creation, create a new, mutated genotype from the original in 70% of all selections instead of 30%. This makes evolution a bit faster. Because 10 instances were created before the selection procedure is called again, there is no need to make more runs with existing genotypes. This allows a higher mutations percentage.
- The ingestion multiplier was set to 4. This was done after testing a few different values. This change makes it possible for an individual to compensate its idle metabolism and in a good collision even get some extra energy. The default value is one.
- Another population group was added: Twigs. This group would contain all the twigs that were located in the world.
- The world size was set to 55.

The rest of settings is identical to the settings of the pilot study. Again I did not define a stop condition, since I did not know how high the fitness values would become or when a maximum would be reached. I let the conditions run until all conditions showed stable results, which translates to a period of 150000

[6]Framsticks defines this as idle metabolism

instances where fitness values fluctuated no more than 500. In condition 3, this stop condition was reached in the extended run.

To summarize, I investigated four conditions, each with a different amount of foodballs.

- Condition 1: 40 foodballs, 2 sticks.
- Condition 2: 50 foodballs, 2 sticks.
- Condition 3: 60 foodballs, 2 sticks.
- Condition 4: 5 foodballs, 2 sticks.

And another run on the following point:

- An extended run of condition 3. This run was stopped after the results were stable for a longer period of time.

4 Results

The simulations ran for about a month, each condition on its own computer. After this period about 740.000 individuals had been created and tested in the condition with 40 foodballs and around 300.000 individuals in the other two conditions. The difference between these condition is probably due to the much shorter lifespan of the individuals in condition 40, making it possible to test more individuals in the same amount of time. Framsticks made a file every 1000 simulator steps with which is was possible to start the experiment anew from that moment. I wrote a JAVA-application which allowed me to extract the data I needed from these files. Per file, I extracted the following data:

- The highest instance number of each genotype.
- The lifespan of the genotype.
- The population size of each genotype at that moment.

The instance number is defined by the number the last instance of the genotype got, when it was created by the simulator. This number starts at one when the experiment begins and increases with each instance the simulator creates of any genotype. The fitness of a genotype is defined by the mean fitness of all instances still present in the gene pool. This entails that the mean of a genotype rises over time, since individuals with a bad score will be removed from the gene pool, leaving only individuals with a good score. Finally, I extracted the population size of each genotype at the moment of the save. The population size is defined by the number of instances still present in the population at the time the file was made.

I will only look at the genotypes which had at least 20 instances still in the gene pool, since the fitness of these genotypes is probably reasonably reliable. This resulted in 3 graphs, one for each of the three conditions (see Figures 4, 5 and 6).

It is clear that in all conditions the "free" lifespan[7] of 1000 is immediately surpassed. This is because the probability of hitting a foodball by accident is

[7]All individuals receive 1000 energy units for each body part it has, resulting in a starting energy of 8000 units for each instance of the predefined body.

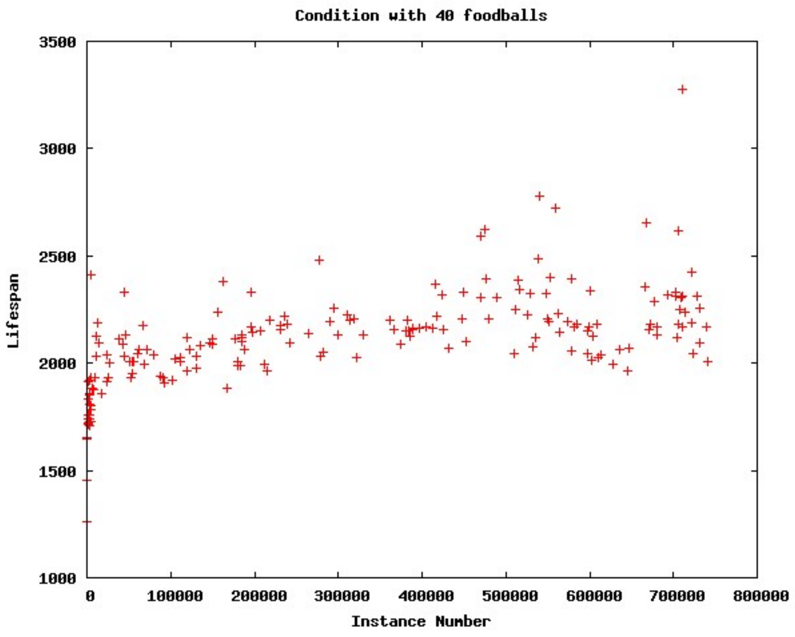


Figure 4: Lifespan with 40 foodballs

high, since the world is filled with foodballs (see Figure 7 to get an idea of the world). The individuals cover the distance from one end of the world 20 times easily.

In all conditions it seems there is a rapid increase to a lifespan of about 1.500 and a pretty stable period for about another 200.000 individuals. Only in the condition with 60 foodballs there is a major growth in the last 50.000 individuals. Some of the instance numbers never made it into a file, probably due to (relative) bad performance, which meant that the instance with that number already got deleted from the population when a new save file was made. This explains the blanks in some of the graphs. Below, I will discuss the results of each condition in more detail in Sections 4.1, 4.2, 4.3, 4.4 and 4.5 evaluating the behavior in each condition first and then talk about the graph related to the condition. In Section 4.6 I describe an extra test with the best genotypes of each condition. In this test, I placed these genotypes in another world to see what their performance would be in a new environment.

4.1 Condition 1: 40 foodballs

The behavior of the individuals in the condition with 40 foodballs seems to be directed to finding and consuming food, but the individuals tend to hit each target once, then move away from the target and turn back after a short distance, which seems longer than necessary. On the long run, this probably means that the creatures will not survive, because they can not compensate their idle metabolism. There does not appear to be even a remote tendency to move toward the twig when it is in the neighborhood. A few individuals did pick up the twig, but the behavior of these individuals did not deviate from the

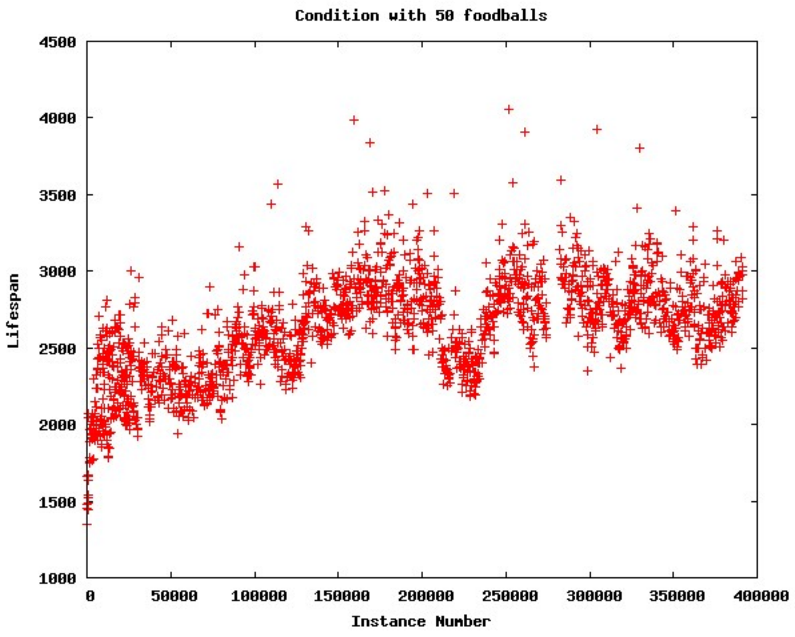


Figure 5: Lifespan with 50 foodballs

usual behavior the individuals from this condition displayed, when a stick was in the neighborhood.

The lifespan increases quickly to 1.500, which probably means the amount of foodballs is big enough to facilitate such a rise, without learning anything. It seems that with the current number of individuals tested, there is an upper bound to the fitness, which is surpassed only by a few instances, but the more reliable the fitness becomes, the more it seems to fall around this bound. When examining the data, another difference with the other conditions was found. The number of times some of the genotypes had been created, were much higher in this condition than in the other conditions, but I was not able to find a reasonable explanation for this.

4.2 Condition 2: 50 foodballs

The behavior of the individuals, created from the genotypes in the last file stored, does not appear to be directed at bumping into foodballs when they are in the neighborhood, in other words, they do not seem to deviate from their original course when they get near a foodball. It sometimes looks as if they try to position themselves in such a way that they can take a bite of several foodballs in one rush. This demonstrates itself by a sudden increase in speed compared to the usual speed. There appears to be no interest at all in obtaining a twig, even when there is twig only a short distance away. There does appear some sort of pattern the individuals will drive when there are no foodballs in the neighborhood, but I can not be sure if this a coincidence, since there only a few areas with no foodballs in them.

The lifespan in this condition quickly rises to about 2500. It appears to rise

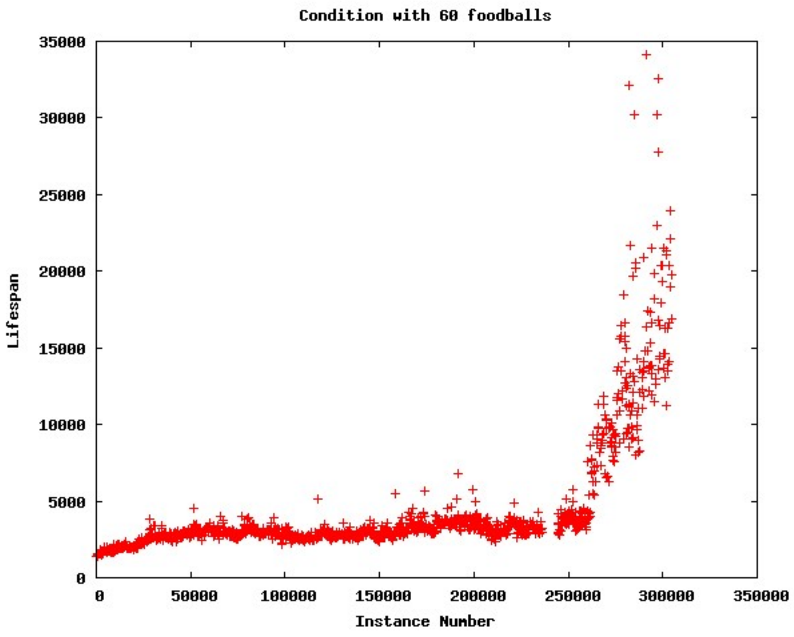


Figure 6: Lifespan with 60 foodballs

even further, to around 3000, but around 30.000 individuals there is a drop to 2500. There seems to be a slow climb to 3000 from around 30.000 to 170.000, but then it seems to stop again. In the end, the fitness seems to have stabilized somewhere between 2500 and 3000, going seldom outside these boundaries for 150000 instances. In data files from this condition the same type of peaks are visible as in the condition described above, but here they shoot up as high as 10.000.

4.3 Condition 3: 60 foodballs

This condition showed the most promising behavior of all, since the individuals actually ate whole foodballs and after one foodball was gone, went looking for a new one. The individuals in this condition only displayed a very peculiar sort of motion, since all individuals were driving backwards (see Section 2.2 and Appendix B for more details on the genotype). This is probably due to coincidence, since I only did one run with each experiment. I expect that this will not occur in all runs when multiple runs are done. Another difference with the other condition was that the individuals from this experiment appeared to jerk with their body. This could help with gaining food, because the gain in energy is calculated for each part of the body that hits the foodball. By jerking, more of the body actually hits the foodball, resulting in a higher energy gain. This behavior is not found in any of the other experiments I did, nor did I expect it.

In contrast to the other two conditions the lifespan of this condition with 60 foodballs rises to 3000 instead of the 2500 in the condition with 50 foodballs and 1500 in the condition of 40 foodballs. This is the best improvement until

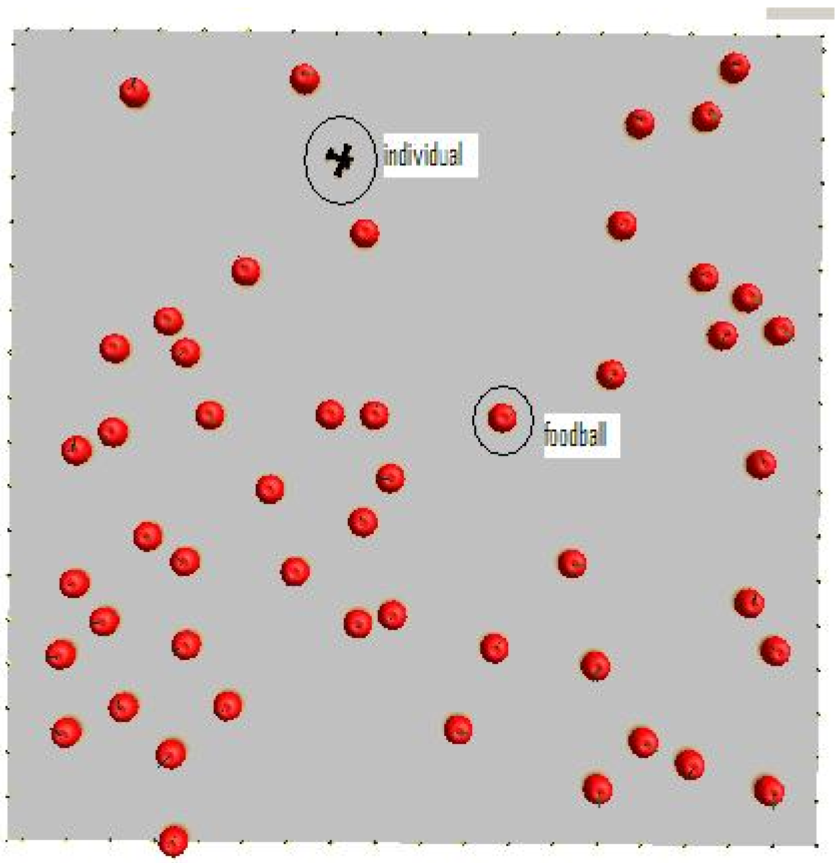


Figure 7: The world with 50 foodballs and one individual

individual 250.000. After individual 250.000 there is a sudden growth to a lifespan of 15.000 and there is no indication this is the maximum. To investigate this further, I let the simulator evolve another 200.000 individuals resulting in Figure 8.

4.4 Condition 3: extension

As can be seen in Figure 8, there was no real improvement of the results with the extension on the experiment. Only in the end there seem to be several peaks, but these are no reason to conclude there has been a significant change. The lifespan of these individuals does seem to grow a bit in the end of the graph, where it reaches a range of 20.000 to 30.000.

The behavior looked much the same, individuals almost always ate a whole foodball, although it sometimes took a few collisions with the foodball. When multiple collisions where needed, this did not look very efficient. Again the twig does not seem an object of attention for the individuals.

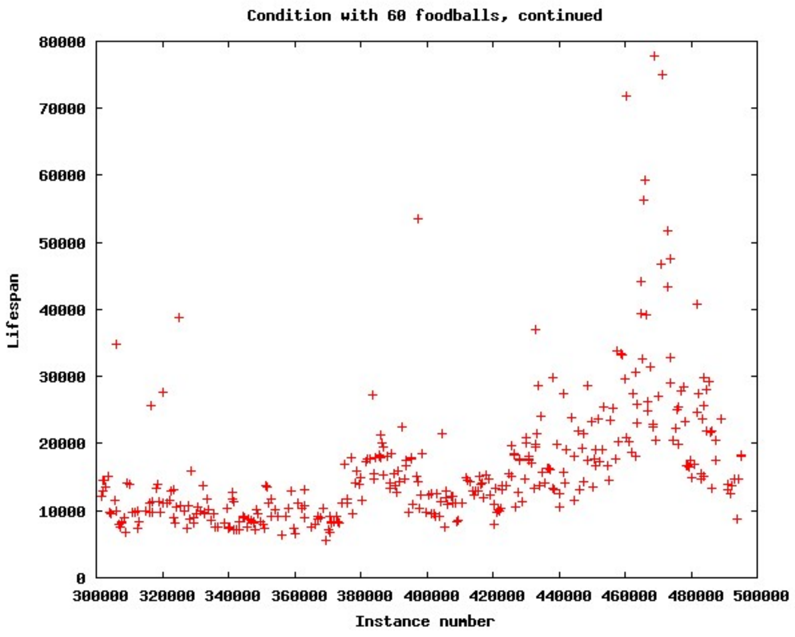


Figure 8: Lifespan with 60 foodballs, part 2

4.5 Condition 4: 5 foodballs

To be actually able to see what the achievements of the best genotypes of each condition could have looked like in the test described in Section 4.6, I ran another simulation, this time with the test settings, so only five foodballs. The fitness of these creatures does not grow beyond 1600 as can be seen in Figure 9. In this figure it is also visible that the base fitness lies a little lower than with the other conditions, around 1100 in this condition but already 1500 in the condition with 40 foodballs. This means that the maximum lifespan of this condition with five foodballs is approximately equal to the starting fitness of the condition with 40 foodballs.

The best individuals of this condition displayed pretty efficient behavior, finishing whole foodballs and picking up every twig in the neighborhood. The way in which the foodballs were completely emptied seems the most efficient I have seen so far, except for the pilot study (see Appendix B). The individuals home in on a foodball, hit it once and then turn around their own center point in such a way that the foodball is hit several times. The behavior in between the picking up of objects also seemed to make sense. The individual that was not near a food source drove around in a circle, which almost guaranteed another foodball within 10 rounds. The peculiar behavior of the individuals in the condition with 60 foodballs is repeated here, the individuals drive backwards most of the time.[8]

When placed in an environment with much more food, the individuals created from these genotypes drove around in small circles all the time, not going

[8]Again, this is only one run, so there is no guarantee this is the standard form of motion for the genotype.

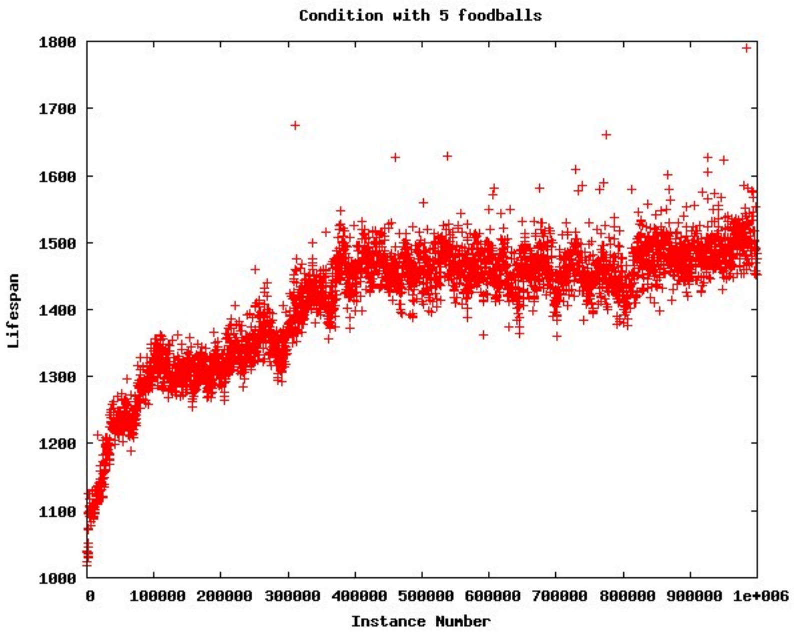


Figure 9: Lifespan with 5 foodballs

to a food source.

4.6 Another test of the genotypes

To see how these individuals performed in a world which did not contain as much foodballs as the world in which they evolved, I took the most reliable individuals from the final files of the conditions 1, 2 and 3 and placed them in the world with only 5 foodballs in it (see Section 4.5 for the original condition with 5 foodballs).

The individuals that got created from the condition with 40 foodballs, did not perform very well. They were able to get some of the food and even pick up a twig, but this behavior only occurred when the initial placement was lucky, meaning not to far from a foodball. This was visible as the individuals did not get much food when driving around, but only bumped into food when they were placed near a foodball on creation. When the individual was placed further away, it just moved around in circles, only occasionally finding something to eat. A few individuals did not even manage to find food.

The genotypes of the condition with 50 foodballs behaved similarly. In the initial phase, the individuals searched for food, but did not efficiently eat all the food. The performance got worse when a twig was picked up, the individuals appeared to switch to moving around in large circles.

Last, I tried the best genotypes from the condition with 60 foodballs. I used the genotypes from the first run, not of the extended run. The individuals from the best genotype displayed pretty random behavior, not directed to any object. The jerking bodies appear to prevent the individuals from driving in an orderly fashion, so it appears as if these individuals have some difficulties if they have to cover bigger distances to get to the next foodball. The individuals from the extended run display pretty much the same behavior when put in an environment with significantly less food, but I noted during these tests that the individuals only tried to completely eat a foodball when another foodball was in the vicinity.

When checking the genotypes from this condition, I noticed that in their own environment, sensor values where a lot higher than in the world with only five foodballs. This difference was around 0.4 in the test world opposed to 14.0 in the original environment and could well be the cause of the inefficient behavior in this extra test.

4.7 Summary of the results

In the sections above, all the results have been described, but not all of these results are equally important with the hypothesis from Section 1 in mind. The most important results will be summarized here.

The results of the conditions 1, 2 and 3 show that individuals created from the genotypes are not able to work with the twig. In condition 2, individuals even have trouble finding food. The lifespan of these individuals is not as good as I expected when designing the conditions. As stated above, the conditions were defined in such a way that individual should be able to learn to live indefinitely, but not a single individual was able to do this. The most promising behavior came from the fourth condition, where the behavior of the individuals had developed in such a way that food and twigs were used optimally.

Another, unexpected result was the shaking behavior of the individuals in the third condition. This is a strategy that makes individuals more efficient in gathering food, since more body parts hit the foodball and probably some parts hit the foodball more than once, allowing for a much higher energy gain.

One final observation can be considered important. When the genotypes from the first, second and third condition were placed in a different environment (that of condition four), the sensor values differ too much to allow the same behavior as in the world in which the genotypes were originally developed.

5 Discussion

I hypothesized that the fitness values would be pretty much the same in the first, second and third condition, but this appears to be wrong. It appears the amount of food in the conditions differed to much to have the same results. Another possibility could be that the individuals never learned the task in such a manner that they could gain the maximum score possible. The behavior over the three conditions did differ, but not in a way that it would explain the difference in lifespan as I hypothesized in Section 1. The results from condition 4 are probably the most satisfactory with the task in mind, to live so efficient as possible.

The individuals did not learn the benefit that could be gained from picking up a twig, since there never appeared any deviation from the original path when a twig was close. The best indication I got that the twig caused any different behavior was in the test with individuals from the condition with 50 foodballs. I described in Section 4.6 that these individuals appeared to move in large circles after they picked up a twig. This could imply that they learned that once they obtained a twig, there was no need to be efficient in the gathering of food. Simply driving around in circles ensures that there will be enough food to survive. The only discrepancy with this theory is that if this strategy ensures enough food, why is the lifespan still around 3000 while the individuals in the 60 foodballs condition already reached a lifespan of 20.000. It could also entail that there was not enough time to evolve the behavior after a twig was picked up to a more sophisticated level.

A success was achieved in condition 4, where the creatures did use the stick to gain a little more lifespan. In this condition there were only 5 foodballs, which required a perfectly efficient strategy to survive in the relatively huge environment. But even this perfect strategy did not appear to be enough to live indefinitely, because the lifespan did not rise above 1600. This is caused by a too low amount of food in a too big world.

The individuals in the conditions 1, 2 and 3 did not even learn the task of gathering enough food to survive without using a stick. This is against expectations, since the amount of food placed in the condition with 50 and the condition with 60 foodballs should be more than sufficient, when an individual efficiently empties each foodball.

Experiences with Framsticks

There are several possibilities to improve the results from the experiments. The save files are automatically generated by Framsticks, which is good to get the experiment back on track after a crash, but these files can contain a few gaps with respect to instance numbers. It could be a good idea to make sure the data of each genotype is stored in a more orderly fashion and it might not be a bad idea to store the data on each instance of a genotype. This will make the experiment significantly slower, since writing to a file takes time (probably a downsize of at least several hundred simulator steps per second), but it guarantees complete data. As I mentioned in Section 4, the mean fitness of a genotype rises over time, because bad individuals are removed from the gene pool, leaving only the better scoring individuals to calculate the mean fitness. This is definitely a point to take into account, since this could drive the fitness values up, even when nothing actually changes. This emphasizes the need to store all data on a genotype, for calculation of the actual fitness of a genotype.

When checking the results of the experiments, I noticed something else, which could have a great influence on the results of any experiment. I saw that the teleport boundary I used, does not always function properly. This results in individuals moving outside the world, where there is nothing for them to home in on (no food or twigs) and the teleport mechanism does not appear to correct such a mistake, so most individuals died outside the world. I did not find out what caused this error, but it left the question whether or not this was caused by some graphical problem in Windows. Unfortunately the Linux machines running the experiment, there is no visual feedback possible, so I could not check this hypothesis. If it only is a graphical error, the experiment should run fine in evolution mode[9].

[9]This mode can be used in Windows. It minimizes all output on the screen, allowing the simulator to run at higher speeds.

6 Conclusion

In Section 1 I described a theory by Alcock (1972), comprised of three steps. The first step of this theory was the appearance of a certain trait in a population, the second was the spreading of the trait through the population and the last step described the subsequent evolution of the trait. I tried to realize the first step of this theory in this thesis, but I estimate that there is more research required to create the right environment for this experiment. The individuals were not able to adapt to a new environment, as I described in Section 4.6. The sensor values probably differed too much from the usual input. The same can be said about the individuals from the extra condition with only 5 foodballs. When these individuals were placed in a world with 50 foodballs, they just moved around in small circles, probably caused by the high sensor values.

Future Research

Future research can be continued through the other two steps of Alcock, namely the spreading of the trait through the population in which it appeared and the evolution of the behavior involved with the trait. But before this can be done, a form of learning might be introduced, so the individuals can learn from others in the second step and learn to be more efficient by themselves. It could also be wise to research the right ratio of food versus worldsize. The number of twigs in a world could be varied as well.

Another subject of future research could be Framsticks itself. In Appendix A, a start is made at manual, but there are still a few gaps in the available literature and not all functions have been documented very precisely as I also explain in Section 2.2 with the mutation function. The only thing that can be set are the probabilities that a mutation will occur, but what a mutation actually entails is not clear.

References

Adamatzky, A. (2000). Software review: Framsticks. *Kybernetes*, 29(9-10):1344–1351.

Adamatzky, A. and Komosinski, M., editors (2005). *Artificial Life Models in Software*. Springer-Verlag, New York.

Alcock, J. (1972). The evolution of the use of tools by feeding animals. *Evolution*, 26(3):464–473.

Eiben, A. E. and Smith, J. E. (2007). *Introduction to Evolutionary Computing*. Springer.

Harvey, I., Di Paolo, E., Wood, R., Quinn, M., and Tuci, E. (2005). Evolutionary robotics: A new scientific tool for studying cognition. *Artificial life*, 11(1-2):79–98.

Jelonek, J. and Komosinski, M. (2006). *Biologically-Inspired Visual-Motor Coordination Model in a Navigation Problem*, pages 341–348. Springer Berlin/Heidelberg.

Komosinski, M. (2000). Framsticks experimentation center. Retrieved October 29, 2007, from news://news.alife.pl/alife.forum.

Komosinski, M. (2003). The Framsticks system: versatile simulator of 3D agents and their evolution. *Kybernetes: The International Journal of Systems & Cybernetics*, 32(1/2; Special Issue on Artificial Life Software):156–173.

Komosinski, M., Koczyk, G., and Kubiak, M. (2001). On estimating similarity of artificial and real organisms. *Theory in Biosciences*, 120(3-4):271–286.

Komosinski, M. and Ulatowski, S. (1997). Framsticks web site. Retrieved December 22, 2007 from http://www.frams.alife.pl.

Lefebvre, L., Nicolkakakis, N., and Boire, D. (2002). Tools and brains in birds. *Behavior*, 139(7):939–974.

McGrew, W. C., Tutin, C. E., and Baldwin, P. J. (1979). Chimpanzees, tools, and termites: Cross-cultural comparisons of senegal, tanzania, and rio muni. *Man, New Series*, 14(2):185–214.

Mondada, F., Franzi, E., and Ienne, P. (1993). *Mobile Robot Miniatuisation: A Tool for Investigation in Control Algorithms*, pages 501–513. Springer Verlag.

Nolfi, S. (1997). Evolving non-trivial behaviors on real robots: A garbage collectiong robot. *Robotics as Autonomous Systems*, 22:187–198.

Nolfi, S. (2000). *EvoRobot 1.1 User Manual*. National Research Council of Italy, Institute of Cognitive Sciences and Technologies.

Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics: The biology, intelligence and technology of self-organizing machines*. MIT press/Bradford.

van den Braak, S. (2005). Progress is not optimization: Co-evolutionary robotics in simulation. Master's thesis, Radboud University Nijmegen.

Virtual Life Lab (2007). Downloads. Retrieved October 29, 2007, from Virtual Life Lab: http://www.aisland.org/vll/projects/downloads.htm.

Appendix A: Framsticks Function Overview

An important note on this overview is that it is based on the Framsticks GUI for Windows, but all options and modifications that are used in genetic experiments are available to the CLI for Linux and Windows as well. The only difference is that to access them, you have to edit the files through a text editor. I got a lot of information from the Framsticks website (Komosinski and Ulatowski, 1997) and the articles made available there (e.g. (Adamatzky and Komosinski, 2005), (Komosinski et al., 2001) and (Komosinski, 2003)).

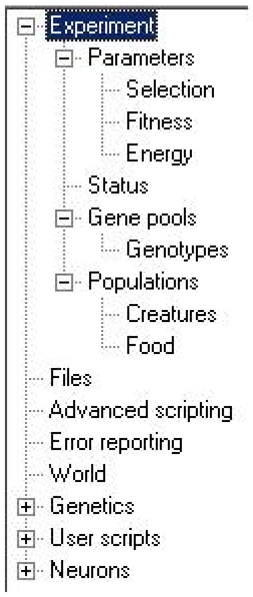


Figure 10: The menu structure in Simulation → Parameters.

Simulation Parameters

The simulation parameters accessed by Simulation → Parameters are sorted by several categories, listed on the left side of the pop-up menu (see Figure 10). I will review all important parameters for this experiment following the order of the list displayed in Figure 10.

Experiment When Experiment is opened the user can select an experiment type, which is defined in a .expdef file, located in the Framsticks\scripts folder. I will go into detail about writing such a .expdef file below. There are two buttons placed at the bottom, one to initialize the experiment applied and one to reload the definition when the file is modified while Framsticks was still running. Reloading might be useful when an irreparable error occurred, but I did not have to work with this button. When selecting a definition, press the "Apply" button first and then choose Initialize, else the simulator might initialize the original definition.

Parameters The next group of options is located under Parameters, where you can enter the initial genotype. This genotype can also be specified in the .expdef file, but I will discuss this later. The initial genotype is placed in the first gene pool when the experiment is initialized and thus will result in the first individual used for evolution.

Apart from the initial genotype (for more information and examples on building genotypes, see the Framsticks website (Komosinski and Ulatowski, 1997)), a few other parameters can be set. The gene pool capacity is used to keep the maximum number of individuals in the gene pool to an acceptable level. This influences the flow of the evolutionary process through limiting the number of individuals possible. When this value is high, more individuals will be remembered and thus different mutations are possible. Also when the simulator is set to deleting the worst genotypes when reaching this maximum, it seems intuitive that the higher this number, the longer bad individuals can stay in the population. This could slow the evolution down, because when an unchanged bad individual is selected for another run, this run is wasted if the individual is truly a bad one. The next option allows the user to set which of the genotypes should be deleted when the maximum is reached. These options explain themselves pretty much. When I wrote my own experiment definition file, I stumbled on a problem. When I loaded my own experiment definition, the options "Inverse proportionally to fitness" and "Only the worst" were placed on the same line, which makes it impossible to select one of the two. I don't know which one is used when selecting this line, but in the experiment definition file you can set either option and that will work as it is supposed to. If you really want to overcome this problem, you could save your simulator-settings, open that file in a text editor and replace the "Õ" with "∼ O". Since "∼ " has a few special functions in the Framsticks syntax, be careful when using it.

"Simulated Creatures" is used to allow for more than one creature to be simulated. The simulator will automatically place additional creatures in the world if the number of simulated creatures drops below the number set here. The next three options are all used for the placement of creatures when they are created. Placement and Orientation can all be set through drop-down menus while the height can be set as precise as desired. The button at the bottom, "Reset performance data", is used to delete the data stored on performance of the simulator. It will not clear the gene pool nor the world, it only removes the numbers stored on how many creatures there were tested etc.

Selection *Selection* is the next set of options. In this menu there are three important numbers, Unchanged, Mutated and Crossed over. These numbers are used to determine whether a selected genotype will be modified or not before it is placed in the world. The selection procedure goes as follows. The three numbers are added up and multiplied by a random number between 0 and 1. If the outcome (referred to by X) is smaller than the number entered for Unchanged, a genotype is selected and no operation is performed on this genotype. If X is bigger than Unchanged, Unchanged is subtracted from X and if this outcome X - Unchanged is smaller than Mutated, a genotype is selected and a mutation is applied on this genotype. This genotype is then placed in the simulator. If X - Unchanged is bigger than Mutated, a crossover is performed and the resulting genotype is placed in the simulator. So the selection process can be described by the pseudo code in Figure 6.

The selection process in Figure 6 is standard, but it is possible

```
X = ( Unchanged + Mutated + Crossed over ) * $R_{0,1}$
If ( X < Unchanged)
   Take unchanged individual
Else if ( ( X - Unchanged ) < Mutated )
   Take a mutated individual
Else
   Cross over and take the resulting creature
```

Figure 11: Pseudo code for selection. $R_{0,1}$ stands for a random number between 0 and 1.

to modify this in the .expdef files. “Minimal similarity” is used to determine whether or not two genotypes can be crossed over. See (Komosinski et al., 2001) for a discussion on the topic of similarity.

The next setting is used to set the “Selection rule”. If set to random, a random genotype will be selected, which pretty much explains itself. If fitness-proportionate is selected, the program will calculate the chance for each genotype to be selected. The higher the fitness of a certain genotype, the higher the chance it will be selected. See Eiben and Smith (2007) for a discussion fitness-proportionate selection and tournament selection. The other four choices are all tournament selection, but differ in the size of the tournament.

Fitness The next set of options is all about Fitness. Here weights can be set for a set of parameters to calculate the fitness of an individual. This explains itself except for the last two options, Criteria Normalization and Similarity Speciation. Criteria Normalization converts all values to a [0,1] interval before weighing, which is useful for experiments where multiple parameters are used to determine fitness[10]. Similarity Speciation is a feature that calculates the similarity of two genotypes based on the degrees between parts (as I understand the number of sticks connected to each other on a certain point), the number of neurons on a part and the total number of neurons. (This will not work in my experiment, since both the body and the number of neurons are fixed.) For more information on similarity, see (Komosinski et al., 2001).

Energy In the Energy settings screen the Starting energy for a creature can be set. “Idle Metabolism” appears to be a fully customizable value, but the only values it takes are 0 and 1, 0 for no idle metabolism, which means that a creature does not die over time when it does not get enough food. When the value is set to 1, a creature uses 1 energy unit for each body part it has in each simulator step.

The next three options are about food. “Automatic Feeding”

[10]e.g. if distance and lifespan are both fitness criteria for an individual that is very slow and lives very long.

specifies the number of food balls present in the world. This number will be maintained at all times, so if a food source is depleted, another will be placed at a random position. “Food Energy” represents the amount of energy a food source has. If set at a low value, creatures can eat a whole source in one bite, but if set on a high value creatures need multiple collisions to deplete a source. “Foods Genotype” can be left empty which results in an apple if OpenGL is used. If a movable food source is required one should create a genotype which can move (either through evolution or set it all by hand) and place that genotype here. The “Ingestion Multiplier” is set to 1 by default and is used to influence the effectiveness of a collision with a food source. If set to a lower level, a creature will get very little energy when in contact with a food source. If set to a higher level, a collision will give a lot of energy and a food source is depleted sooner. “Aging time” is used to make sure a creature dies. When set to any number above zero, this parameter causes the creature to use more energy each step, it doubles the Idle Metabolism after this number of steps.

Gene pool Under Gene pools the number of pools is displayed and each group can be accessed. In the standard experiment, there is only one group, Genotypes. The most important settings in this menu are Fitness-related. The Fitness formula displayed here is built up from the parameters set earlier on in the Fitness screen. The parameters under the formula can be used to scale the fitness, by shifting and multiplying it. Under Populations the number of population groups is displayed. Each group can be accessed separately, which then displays that groups options. There are two groups in the standard experiment, Creatures and Food. The settings here cover multiple aspects of the simulation. Energy calculation refers to the fact if the creatures from this group depend on energy. If switched off, the instances in the respected group have a constant energy level. If Death is turned on, creatures die when energy hits zero or below. Neural net simulation is used to determine when the neural net starts working. After freeze means that the creature is dropped to the ground first, then the simulation waits for it to have stopped moving completely and then starts the neural net.

Files Under Files settings for back-ups can be adjusted. This is important to do, since the standard settings does not make back-ups. Back-ups are stored in the .expt format, which includes all simulator settings, genotypes and performance numbers or .gen files, which contains the same information, although the normal .gen files only stores genotype information. *It appears that the file type depends on the last file loaded.*

World Under World are the settings for the lay-out of the world. The Size of the world can be set, which is in an unknown unit. The Boundaries are important for making either a world without boundaries, a world with a fence around it or a world that is surrounded by a teleportation system, also known as a Torus world. There are several ways to create a world yourself, with walls, water and a lot more special properties, but I will not discuss these methods here.

Genetics Under Genetics the parameters for genetic operations can be found. These are quite easy to understand, except which language (f0, f1 or f4) is used at what time. I believe the operations used are those for the language in which the individual that is being operated upon is written. The numbers for f0 are the relative mutation probabilities, which means that if, for instance, Change Connection Weight is set to 10, a mutation will change a connection weight 10 times out of 100. For instance if mutation is set at 30% and Change Connection Weight at 50%, the chance that a change in connection weights is mutated is 15%. Unfortunately this only describes the probability *if* a weight is changed, not *how* is changed. What a mutation actually does to a connection weight is unclear in the literature. The same goes for both f1 and f4. The language f4 is not much used and in some of the tutorial files described as an untested language, so it's unclear whether or not writing in this language will succeed or work as desired.

Userscripts Userscripts contains a number of buttons which allow for some other functionality added by other users. Such as displaying all genotypes in the gene pool as static creatures in the Gallery option. If there was a modification in the neuron definitions while running the program, this menu also contains a Reload Neuron definitions, that allows you to reload the modified neuron without closing the program. I haven't used any of these scripts seriously in my experiment so I don't know whether or not they actually function well in all circumstances.

Neurons The last branch of Simulation → Parameters is Neurons. Here the activity of each neuron on initialization can be set with a factor under Simulation and which neurons actually are used can be set under Active, i.e. whether or not this neuron is an allowed addition to any genotype in the simulation. With this last branch all Simulator Parameters have been described. If you made some changes to the default values, you can save these settings under File → Save Simulator Parameters as This results in a .sim file that can be loaded from the menu File → Load Simulator Parameters from Such a .sim file can be opened and edited quite easily with a text editor. The abbreviations are a bit hard to understand sometimes, but most of these can be understood when comparing the list in the .sim file with the actual menu in the program itself. Something that can come in handy is that the .sim file basically follows the structure in the menu except for the genotype and creature groups which can be found at the end of the file.

Experiment Definition Files

An experiment definitions file (.expdef) contains many settings that can be stored in the simulator parameters file (.sim) as well. Always make sure that when you load such a file (whether .expdef or .sim) that your settings are preserved. Apart from these settings an experiment definition file also contains much information on genotype and creature groups. Perhaps the most important in the experiment definitions file is that it allows for modifications in numerous procedures such as selection and fitness calculation. In this section I will explain how an experiment definition file is made and what the different functions do by taking my own experiment definition as an example. (My Experiment definitions is included in Appendix C. For more information on Experiment definitions see (Komosinski, 2003))

```
expdef:
name:BachelorThesis
info:~
Designed for Thesis by Timo Veldt

Provides:
three groups for objects (Creatures, Food, Sticks)

~
```

The code above is the first part of an experiment definition file. I copied it directly from my own file, which is not something I'll do with the entire file. The first line tells what kind of file this is, an expdef or experiment definition. The second line gives the name of the experiment definition, which is in this case BachelorThesis. On the third line more info about the experiment definition can be placed. Noteworthy is that behind info: a " ~ " can be seen. This denotes that info covers multiple lines. This allows for the setup as displayed in line 4 through 6, where the " ~ " gets closed. Everything between the tildes is considered info and will be displayed in the Description box under Experiment in the Simulation parameters window. After the information about the experiment definition follows the code that actually constitutes the definition. I will discuss the functions used and the most important things that I noticed during using and writing experiment definitions.

The first function in the Code is **OnExpDefLoad()**. This function is called when loading an experiment definition. It contains the definitions of the Genotype groups and Creature groups. After clearing the current settings, the new groups can be made and some of their parameters can be set. Noteworthy in this process is that this is the only place where the Colmask can be set. This value is used when a collision between an individual from the current group and any other group occurs. The number used here stands for a binary code, which is built as follows. The Colmask is stored as a byte, thus has 8 positions that can be either on or off. When all positions are on, it can be represented as cCsScCsS, when all positions are off, I will use the following notation ———. As seen above, there are four different fields in the Colmask: c, C, s and S which have the following meaning. C stands for custom collision handling, while c stands for inhibit custom collision handling. The same goes for S and s, where S stands for standard collision handling. Colmask is used to determine how a certain collision is handled and always requires the two Colmasks of both the colliding individual and the individual that is being hit. These two bits can then be compared and the result will be used to determine what happens. If the result is a standard collision handler, the objects will bump into each other and then bounce off. When the result is a Custom collision handler, it is possible to edit the effect of the collision in the corresponding onXCollision function, where X stands for the type (or CreatureGroup.name) of the object being hit. The way the result is determined, is by comparing both Colmasks. For example, if we have an individual with Colmask —-cC-S (13) colliding with a foodball with Colmask c–S-C–, the outcome will be C, because, looking from left to right, this is the first letter they have in common. This is desirable, because when a creature bumps into food, you want it to eat the food, so an energy transfer has to take place, which can be defined in the onFoodCollision function. The last important point about the Colmask is that if two individuals with Colmasks —-cC-S collide, the outcome will be S, because the lower case c will inhibit the upper case C, thus resulting in the standard collision handler. More information on collision handling can be found on the Framsticks website (Komosinski and Ulatowski, 1997) under the class descriptions, CreaturesGroup.

Under the creation of GenotypeGroups and CreatureGroups, the function allows for some adjustments in the simulation parameters, which is basically what I described above in the Section parameters. Important to note is that I haven't tried to set all possible parameters, I created a .sim file instead, which overwrites the option set in the experiment definition. A warning is appropriate here, because if you reload the experiment definition after you loaded the .sim file, the settings of the .sim file are discarded again, so one should make sure the files are loaded in the proper order and that the settings remain correct after loading each file. The final step in this function is putting the number of tested creatures to zero, because there are no creatures tested when an experiment definition file is loaded (in contrast with an experiment state file, but more on that later).

The next function in the file is **OnExpInit()**. This function is called when the button initialize experiment is used in the Experiment screen in the Simulation Parameters. This function clears all groups and then adds the initial genotype to the first GenotypeGroup. Again, the number of tested creatures is set to zero.

The next statement is used to include a file. There a several statements like this one and they can be found throughout the code. All these files contain more specific functions that are called in the experiment definition. These files are all of the .inc type and can be edited as desired, but I won't discuss that in this overview.

The function **OnBorn()** is used to place creatures in the proper group and give them the properties of that group. All parameters found in the class overview on the website can be set, although the most important things to set here are the variables concerning energy. This function has a jump in it, the goto placed_ok command lets the program jump forward to placed_ok. When following this jump, one can see how the program determines to which group a creature belongs. This is done by accessing the variable LiveLibrary.group and checking what its value is. This is an important way of checking and setting variables. Almost all variables are public and can be accessed at any time. This seems a bit strange at first, but after trying a few modifications one gets used to it (this is considered sloppy programming by most, but since the source code is not available for modification at this time, it cannot be altered).

The next three functions are all called in the **OnStep()** function. The OnStep function checks if several parameters are being followed each step. It checks whether or not enough individuals per CreatureGroup are present and adds creatures when the number drops below the user specified value. The functions **addFood()** and **addStick()** are called when the number of foodballs or sticks drops below the allowed minimum respectively. The function **CheckNumberOfRuns()** I added, because I wanted to get a more reliable fitness for my creatures by making sure a creature made at least 10 runs. This can be looked at in two ways, the first view says that this doesn't make the fitness of a genotype more reliable, because each instance has its own fitness and when a bad instance is removed because it was the worst at that time, the fitness shown by the program will be biased, because only the best individuals will remain. This is true, the fitness shown by the program will at all times only contain the best individuals which the genotype has had so far. The second view is that this way, each genotype has a fair chance to live up to its full potential and that, if a creature has performed below its actual abilities, its entire genotype will be deleted much sooner and so, positive information may be lost. The number of runs is also stored by using this function, which gives, together with the number of instances a pretty good idea whether or not the individuals still present are the lucky ones or that the genotype is in fact better than the rest.

Another important activity this function handles, is the regulation of the aging time. The aging time is used to make sure creatures cannot live forever by increasing the amount of energy they consume each step exponentially. This function was not used in my experiments.

The function **updatePerformanceWithPopsiz()** is called in the next function **OnKill()**. The first is responsible for keeping track of the total performance of a genotype, while the latter is used to kill individuals on command of the user or when the creature runs out of energy. The documentation on the site states that the update function assumes that the number of instances is a good measure for estimating the weight for the new data. There it also state that this function is open for your own adjustments when you don't think your experiment gets a good population size (population size is abbreviated to popsiz, which means, in this program, number of instances present in the population at this moment). The onKill then does several other important things, as limiting the gene pool and making a back up of the current state. I did not remove the backup function call, but I have not found such a backup while running my experiment. It could be that the option "save every N-th time" (see Section Parameters) overwrites this backup step, but this is pretty much a guess.

The function **SelectGenotype()** is responsible for selecting a genotype for creation, based on the selection parameters as set in the simulation parameters. For more information on this function, see Section Parameters above.

OnFoodCollision() and **OnStickCollision()** are both custom collision handlers. These functions both rely on the Collision class. By testing a few things, I found out that the creature that hits another creature is placed in Collision.Creature2, while the object being hit is placed in Collision.Creature1. This information can be used to decide if a creature gains energy when colliding as happens with food, or the object is removed and an internal value of the creature gets changed as happens with the twig in my experiment.

The **ExpParams_cleardata_call()** removes all instances from the gene pool, leaving the genotypes behind. The next two functions have something to do with the gene pool as well. **LimitGenePool()** actually does the work and **Expparams_capacity_set()** calls this function. The goal of these functions are to make sure that when the size of the gene pool is set to a new, lower value, the gene pool is cleaned up according to the deletion rules set by the user. The functions **LimitCreaturesGroup(g,n)** does the same for a creatures group (argument g). The maximum number of creatures allowed can be given via the argument n. This function is called by **ExpParams_MaxCreated_set()**, which fills in the arguments so that group 0 is selected (this is the Creatures group by default) and the limit is the simulator parameter ExpParams.maxcreated (see Section Parameters above). Last in this line of limiting functions is **ExpParams_feed_set()**, which limits the number of foodballs in the world.

Next in line are functions that have to do with the composition of the fitness function. The fitness function is stored as a string in the genotype class with which it is considered. The creation of this string is accomplished by concatenating several different substrings which in turn consist of the weight of a certain variable and the actual variable itself.

Appendix B: Pilot study of Framsticks

I will describe the pilot study I did in more detail in this section, as well as the genotype I used in all my experiments. I will start with describing the genotype in detail and finish with the settings of the pilot study.

The genotype

I chose to use a rather simple body, which made it easy to identify which side is the front. The two sensors are located on the front of the body. These sensors have been placed in such a way that given a decent control structure, the individual will be able to detect to which side food is located. I got the food sensor from the Virtual Life Lab (Virtual Life Lab, 2007). This sensor is only receptive of the creatures in a certain creatures group, more specifically group 1, which is the standard group for food. A creature group is a group in which all "living" individuals are listed. With living, I mean all objects currently present in the world, including creatures, food and later on, twigs. The food sensor goes through the food group and determines the state of the sensor by taking the energy of the creature being sensed and the location of the sensor itself. I made the twig sensor like the food sensor, which required another group to be added to the experiment, the twigs group, which is the second group. This is the "head" of the creature. The head is attached to the rest of the body by a short stick. There are two arms, with a left wheel on the left arm and a right wheel on the right arm. This positioning of the wheels should allow for easy rotation, if the wheels actually worked like I expected them to work. The wheels on the body are developed for a Braitenberg vehicle, also designed by the Virtual Life Lab. The two wheels (left and right) need to be placed on the creature in order for the mechanism to work. As said above, the wheel does not actually function like a wheel. Instead the creatures get lifted of the ground a little and uses rotation and forward motion to move the creature around. The tail was added for balance, but has no real function, since the whole body gets lifted off the ground, giving a creature all the balance it will ever need. In Figure 2 (Section 2.2), the sensors are visible as a few stubs at the top of the figure. The two wheels are located on the most left and the most right part of the genotype.

The neural network used in this experiment is visible in Figure 12, where lines depict a connection and little triangles depict neurons in the creature. The input units and sensors are located on the left and right side in the top of the picture. Note that this is the neural network used in the main experiments, in the pilot study the network had no recurrent layer and there were no twig sensors as well.

The output neurons are located in middle, while the wheels are located on the left hand side and right hand side. The hidden layer and the recurrent layer are located in the tail. The final network consists of 4 sensors, with their 4 input neurons, 5 hidden neurons, 5 neurons in the recurrent layer, 2 output neurons and their respective motor/wheel. The input neurons were necessary because a sensor has a preferred output of 1, which means that it can not be connected to the multiple neurons in the hidden layer. To solve this, a neuron was added between each sensor and the hidden layer. The same problem arose with the motors, as they had a preferred input of 1. This means that only one of the neurons in the hidden layer would be able to provide output. An extra neuron was added between the hidden layer and each motor, to circumvent this problem. The network is actually changed considerably, because these new connections between the sensor and the input unit for that sensor is also sensitive to the evolution process. This means that it is now possible to turn the whole sensor off, by simply setting the weight between the sensor and the input neuron to zero, rendering all connections from the input neuron to hidden layer useless as well. It also allows for modification of the entire signal, because some sensors can be made more important than others before the actual input reaches the hidden layer. For the wheels a similar problem arises, since one or both motors can get a connection weight of zero between the output neuron and the wheel.

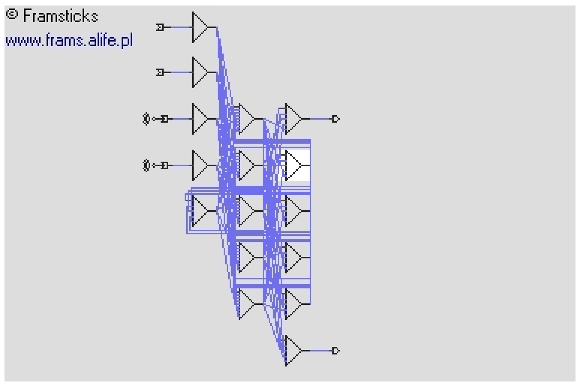


Figure 12: The Final Network by Framsticks

The network as displayed by Framsticks (Figure 12) is not really clear, because the recurrent layer is spread around the other layers. In Figure 3, this should become more clear. From left to right Figure 3 contains the sensors, input layer, hidden layer, output layer and motors. The recurrent layer is located below the input layer. The connections from the recurrent layer connect to the hidden layer and the connections from the hidden layer connect to the output layer, but also to the recurrent layer. The hidden and recurrent layers are fully connected in both directions, but the weights have no relation. So a hidden unit can have a weight of 0.5 to a recurrent unit, but the connection from this recurrent unit to the same hidden unit does not have to be 0.5 as well. The layers are fully connected, so that the creature has all the links available and evolution will decide whether or not to use the full capabilities of the network.

The evolutionary parameters were all set within the boundaries already present in the program, but the experiment definition (see appendix A for more information) was modified to accommodate the stick sensor, which required a third creatures group. Selection was fitness proportionate (see (Eiben and Smith, 2007)) to give the genotypes that did not perform well a chance to get another run. The connections were all set to one, to save the time of implementing a genotype with randomized weights, which will be done when the experiment proves satisfactory. The only form of change to genotypes was set to mutation, because the only things that can actually change are the connection weights. The body was kept intact, because it does not make sense to change the body in unpredictable ways, since the bodies of monkeys do not change all that much, while they are still capable of learning to use a new tool. The number of neurons was kept at a constant as well, since I believe the neural structure that was implemented, should be able to handle the task and this experiment was going to test that hypothesis.

The environment consisted of a world which contained two twigs and two foodballs at all times (i.e. when a foodball is fully consumed, it disappears and a new foodball is dropped at a random location). The world did not have boundaries, but once an individual exited the world on the left side, it got transported to the right side of the world and vice versa[11]. This type of world is chosen because without boundaries, the creatures don't have to learn how to behave around those boundaries. Also, this makes the distribution of food easier, since the simulator only allows for 1000 foodballs to be present in any world and I don't know yet what a reasonable amount is for the overall experiment. By using the torus world, the problem of a limited amount of food is solved, because if there is enough food in world, then there automatically is enough in all reachable places.

This choice was made, because the creature that collects the most food will live the longest as collecting food is collecting energy. When the energy reaches zero, the creature dies, therefore, more energy collected means a longer life. Energy can reach zero, because idle metabolism is set to 1, which means that the creatures created from the chosen genotype have an idle metabolism of 8. This number is calculated by taking each stick that is used to create the genotype and multiplying that number with the idle metabolism parameter. The genotypes consist of 8 sticks, resulting in an idle metabolism of 8, which means that in each step the simulator runs, the creatures will lose 8 units of energy. The program is set in such a way that the creatures are capable of living at least 1000 steps before the energy reaches zero, so the starting energy of a creature is 8000. To collect energy, the creatures had to bump into food, at which point they receive an amount of energy determined by the number of parts that actually collided with the foodball, multiplied by an ingestion parameter. The ingestion parameter does nothing else than multiplying the calculated number of energy units gained in one collision and adding that number to the energy supply of the creature.

After approximately 30 hours of running this simulation, the mean lifespan of a certain genotype had risen to 1068.11, while the first creatures had a lifespan

[11]Also known as Torus world, derived from the mathematical shape.

of 1000. This family consisted of 195 individuals and had arrived at generation 271. The average lifespan of the all creatures in the gene pool at that moment was 1067.98 and the maximum lifespan at that moment was 1115. The behavior of these creatures seemed to be directed at collecting food. After I noted these results, the simulation ran on for another 2 to 3 days, after which the hardware failed due to unknown reasons. The results at that time are thus lost, but a day before the experiment crashed the results were somewhat like this. Minimum generation was at 500 and average fitness in the gene pool at that moment was around 1085. From these results I concluded that Framsticks did offer the means necessary for running the other experiments.

7 Appendix C My Experiment Defintion

This section contains my experiment definition. If "//" is written, the text on the rest of line is considered a comment by the simulator. The initial genotype has been removed. All the weights of the genotype were set to a random number between -1 and 1, but this resulted in a very chaotic structure. All the weights have been set to one in the initial genotype.

```
expdef:
name:BachelorThesis
info:~
Designed for Thesis by Timo Veldt

Provides:
- three groups for objects (Creatures, Food, Sticks)

~
code:~
function onExpDefLoad()
{
    //define genotype and creature groups
    GenotypeLibrary.clear();
    LiveLibrary.clear();

    GenotypeGroup.name = "Genotypes";
    update_fitformula();

    CreaturesGroup.name = "Creatures";
    CreaturesGroup.nnsim = 1;
    CreaturesGroup.enableperf = 1;
    CreaturesGroup.death = 1;
    CreaturesGroup.energy = 1;
    CreaturesGroup.colmask = 13;     // ( - - - - c C - S )

    LiveLibrary.addGroup("Food");
    CreaturesGroup.nnsim = 0;
    CreaturesGroup.enableperf = 0;
    CreaturesGroup.death = 1;
    CreaturesGroup.energy = 1;
    CreaturesGroup.colmask = 148;   // ( c - - S - C - - )
```

```
LiveLibrary.addGroup("Sticks");
CreaturesGroup.nnsim = 0;
CreaturesGroup.enableperf = 0;
CreaturesGroup.death = 1;
CreaturesGroup.energy = 1;
CreaturesGroup.colmask = 148;   // ( c - - S - C - - )

//This creature has a recurrent network and 5 hidden units, 4 sensors,
//2 wheels. All weights are set to 1
//ExpParams.initialgen = "(X[N,5:1,6:1,7:1,8:1,9:1]
[N,4:1,5:1,6:1,7:1,8:1][N,3:1,4:1,5:1,6:1,7:1]
[N,2:1,3:1,4:1,5:1,6:1][N,1:1,2:1,3:1,4:1,5:1]
[N,-5:1,-4:1,-3:1,-2:1,-1:1,8:1,10:1,12:1,14:1]
[N,-6:1,-5:1,-4:1,-3:1,-2:1,7:1,9:1,11:1,13:1]
[N,-7:1,-6:1,-5:1,-4:1,-3:1,6:1,8:1,10:1,12:1]
[N,-8:1,-7:1,-6:1,-5:1,-4:1,5:1,7:1,9:1,11:1]
[N,-9:1,-8:1,-7:1,-6:1,-5:1,4:1,6:1,8:1,10:1]
(lllllllllllllllFFFlX,,lllllllllllllllllllFFFX),
CCC(,,,,,,,X[Right,ms:0.1, 1:1],,,,,,,,
 llllllllllllX[N,-6:1,-5:1,-4:1,-3:1,-2:1]
 [N,-7:1,-6:1,-5:1,-4:1,-3:1]ccc(lllllllllllX[N,1:1]
[Sf][N,1:1][St], , lllllllllllX[N,1:1][Sf]
[N,1:1][St]),,,,,,,, X[Left, ms:0.1, -9:1],,,,,,,))";


ExpParams.capacity = 100;
//0 = randomly delete creatures; 1 = Inverse-proportionally to
//fitness; 2 = Only the worst
ExpParams.delrule = 2;
ExpParams.MaxCreated = 1;

//Population Parameters:

// % No modification
ExpParams.p_nop = 30;
// % Mutations
ExpParams.p_mut = 70;
// % Crossover (NB: Crossover only possible with
//compatible genotypes!)
ExpParams.p_xov = 0;
// Selection Rule (0 = random, 1 = Fitness proportional, 2 through 5 =
// Tournament (i genotypes where i is a selection rule))
ExpParams.selrule = 3;

//Fitness Criteria:

//Constant
ExpParams.cr_c = 0;
//Lifespan
ExpParams.cr_life = 1;
//velocity
ExpParams.cr_v = 0;
// nr of bodyparts
ExpParams.cr_gl = 0;
```

```
    // nr of bodyjoints
    ExpParams.cr_joints = 0;
    // nr of neurons in brain
    ExpParams.cr_nnsiz = 0;
    // nr of connections in brain
    ExpParams.cr_nncon = 0;
    // Distance travelled
    ExpParams.cr_di = 0;
    // Vertical position (flying/swimming)
    ExpParams.cr_vpos = 0;
    // Vertical Velocity
    ExpParams.cr_vvel = 0;
    // Normalize criteria? 0 is false, 1 is true
    ExpParams.cr_norm = 0;
    // Similarity Speciation
    ExpParams.cr_simi = 0;

    // Starting energy of a creature
    ExpParams.Energy0 = 1000;
    // Idle Metabolism: nr is amount of energy required
    //for each part in the body
    ExpParams.e_meta = 1;
    // How many food-balls available in the world at all
    //times?
    ExpParams.feed = 50;
    // How much energy is contained in one food-ball
    ExpParams.feede0 = 1000;
    // How fast is the digestion of a creature (how much
    //of the food is transferred in one collision)
    ExpParams.feedtrans = 4;
    // Genotype of Food-balls ("" is the standard,
    //resulting in an Apple in OpenGL or a little yellow circle.)
    ExpParams.foodgen = "";

    // 0 = Random;1 = Central
    ExpParams.placement = 0;
    // 0 = Always 0 degrees; 1 = Randomized 180 degrees;
    //2 = Randomized 90 degrees;3 = Randomized 45 degrees;
    //4 = Random
    ExpParams.rotation = 4;
    // Create Creatures at height <x>
    ExpParams.creath = 0.1;

    // Total number of creatures evaluated in the experiment
    //(starts at 0).
    ExpState.totaltestedcr = 0;
}

function onExpInit()
{
    //Clear Population-Group 1: Creatures
    LiveLibrary.clearGroup(0);
    //Clear Population-Group 2: Food
    LiveLibrary.clearGroup(1);
    //Clear Population-Group 3: Sticks
    LiveLibrary.clearGroup(2);

    // Clear Genotype-group 1: Genotypes
    GenotypeLibrary.clearGroup(0);
    GenotypeLibrary.newGenotype(ExpParams.initialgen);
    GenotypeLibrary.copyGenotype(0);

    ExpState.totaltestedcr = 0;
}

@include "BSC_placement.inc"

function onBorn()
{
    var retry = 20; //try 20 times
    var trycentral;

    while (retry--)
    {
        //Place central only in the first trial
        trycentral = (ExpParams.placement == 1) && (retry == 19);
        place_randomly(trycentral,ExpParams.rotation);
        if(!LiveLibrary.creatBBCollisions(0))
            goto placed_ok;
    }
    Simulator.message("onBorn() could not avoid Collisions!", 2);

placed_ok:
    if(LiveLibrary.group == 0)
    {
        Creature.idleen = ExpParams.e_meta * Creature.numjoints;
        Creature.energ0 = Creature.energ0 * ExpParams.Energy0
                * Creature.numjoints;
        Creature.energy = Creature.energ0;
    }
    else
    {
        if(LiveLibrary.group == 1)
        {
            Creature.name = "Food";
            Creature.idleen = 0;
            Creature.energ0 = ExpParams.feede0;
            Creature.energy = Creature.energ0;
        }
        else
        {
            Creature.name = "Stick";
            Creature.idleen = 0;
            Creature.energ0 = ExpParams.feede0;
            Creature.energy = Creature.energ0;
        }
    }
}
```

```
function addfood()
{
    LiveLibrary.group = 1;
    if(ExpParams.foodgen == "")
        LiveLibrary.createFromString("//0\nm:Vstyle=food\np:");
    else
        LiveLibrary.createFromString(ExpParams.foodgen);

    LiveLibrary.group = 0;
}

function addstick()
{
    LiveLibrary.group = 2;
    LiveLibrary.createFromString("XX");
    LiveLibrary.group = 0;
}

function checkNrOfRuns()
{
    var counter;

    for(counter = 0; counter < GenotypeLibrary.getGroup(0).count; counter++)
    {
        if(GenotypeLibrary.getGroup(0).getGenotype(counter).popsiz < 10)
            return GenotypeLibrary.getGroup(0).getGenotype(counter);
    }
    return 0;
}

function onStep()
{
    var nrSticks = 2; //Number of sticks in the simulation
    LiveLibrary.group = 0;
    if(CreaturesGroup.creaturecount < ExpParams.MaxCreated)
    {
        if(!checkNrOfRuns())
            selectGenotype();

        if(Genotype.isValid)
            LiveLibrary.createFromGenotype();
        else
            Simulator.print("Invalid Genotype - ignored: "
                    + Genotype.info);
    }

    if(ExpParams.aging > 0)
    {
        var i = 0;
        while (i < CreaturesGroup.creaturecount)
        {
            LiveLibrary.creature = i;
            Creature.idleen = ExpParams.e_meta * Creature.numjoints
```

```
                    * Math.exp((0.6931471806 * Creature.lifespan)
                    / ExpParams.aging);
            i++;
        }
    }

    LiveLibrary.group = 1;
    if(CreaturesGroup.creaturecount < ExpParams.feed)
        addfood();

    LiveLibrary.group = 2;
    if(CreaturesGroup.creaturecount < nrSticks)
        addstick();
}

function updatePerformanceWithPopSize()
{
    GenotypeLibrary.genotype =
        GenotypeLibrary.findGenotypeForCreature();
    if(GenotypeLibrary.genotype < 0) // not found in gene pool
    {
        GenotypeLibrary.getFromCreature();
        Genotype.num = 0; // 0 = it will be filled automatically
        GenotypeLibrary.copyGenotype(0);
        Genotype.popsiz = 0;
    }
    GenotypeLibrary.addPerformanceFromCreature();
}

function onKill()
{
    if(LiveLibrary.group != 0)
        return; //You can't kill food or sticks.
    ExpState.totaltestedcr++;
    updatePerformanceWithPopSize();
    LimitGenePool();
    Simulator.checkpoint();
}

@include "BSC_select.inc"

function selectGenotype()
{
    var sel;

    sel = (ExpParams.p_nop + ExpParams.p_mut + ExpParams.p_xov)
    * Math.rnd01;
    if(sel < ExpParams.p_nop)
        GenotypeLibrary.genotype = selectedForCreation();
    else
    {
        sel = sel - ExpParams.p_nop;
        if(sel < ExpParams.p_mut)
        {
            GenotypeLibrary.genotype = selectedForCreation();
            GenotypeLibrary.mutate();
        }
        else
        {
            var other;

            GenotypeLibrary.genotype = selectedForCreation();
            if(ExpParams.xov_mins > 0.0)
                other = selectedSimilar();
            else
                other = selectedForCreation();
            if (other >= 0)
                GenotypeLibrary.crossover(other);
            else
                Simulator.print("Crossover -
                    Second genotype not found?");
        }
    }
}

function onFoodCollision()
{
    //If a creature has a stick, it gets
    //4 times as much energy from a food-source
    if(Collision.Creature2.user1 == 1)
    {
        var e = 4*(Collision.Part2.as
            * ExpParams.feedtrans), m3;
        Collision.Creature1.energy_m =
            Collision.Creature1.energy_m + e;
        Collision.Creature2.energy_p =
            Collision.Creature2.energy_p + e;
    }
    else
    {   //if it doesn't have a stick, it gets
        //the normal value
        var e = Collision.Part2.as
            * ExpParams.feedtrans, m3;
        Collision.Creature1.energy_m =
            Collision.Creature1.energy_m + e;
        Collision.Creature2.energy_p =
            Collision.Creature2.energy_p + e;
    }
}

function onSticksCollision()
{
    Collision.Creature2.user1 = 1;
    Collision.Creature1.energy = -10;
}

function ExpParams_cleardata_call()
{
    var i;

    GenotypeLibrary.group = 0;
    for(i = 0; i < GenotypeGroup.count; i++)
    {
        GenotypeLibrary.genotype = i;
        Genotype.popsiz = 0;
    }
    Simulator.print("Performance data reset.");
}

function LimitGenePool()
{
    if(GenotypeGroup.totalpop > (ExpParams.capacity + 1))
        Simulator.print("Removing "
            + (GenotypeGroup.totalpop - ExpParams.capacity)
            + " genotypes!");
    while(GenotypeGroup.totalpop > ExpParams.capacity)
        GenotypeLibrary.del1Genotype(selectedForDeletion());
}

function Expparams_capacity_set()
{
    LimitGenePool();
}

function limitCreaturesGroup(g,n)
{
    LiveLibrary.group = g;
    n = CreaturesGroup.creaturecount - n;

    while(n > 0)
    {
        LiveLibrary.creature =
            CreaturesGroup.creaturecount-1;
        LiveLibrary.delete();
        n--;
    }
}

function ExpParams_MaxCreated_set()
{
    limitCreaturesGroup(0,ExpParams.MaxCreated);
}

function ExpParams_feed_set()
{
    limitCreaturesGroup(1,ExpParams.feed);
}

function
    ExpParams_cr_c_set, ExpParams_cr_life_set,
    ExpParams_cr_v_set, ExpParams_cr_gl_set,
    ExpParams_cr_joints_set, ExpParams_cr_nnsiz_set,
    ExpParams_cr_nncon_set, ExpParams_cr_di_set,
    ExpParams_cr_vpos_set, ExpParams_cr_vvel_set,
    ExpParams_cr_norm_set, ExpParams_cr_simi_set,
update_fitformula()
{
    GenotypeLibrary.group = 0;
    var formula = "" + ExpParams.cr_c;

    formula += singlecrit("cr_life", "lifespan");
    formula += singlecrit("cr_v", "velocity");
    formula += singlecrit("cr_gl", "strsiz");
    formula += singlecrit("cr_joints", "strjoints");
    formula += singlecrit("cr_nnsiz", "nnsiz");
    formula += singlecrit("cr_nncon", "nncon");
    formula += singlecrit("cr_di", "distance");
    formula += singlecrit("cr_vpos", "vertpos");
    formula += singlecrit("cr_vvel", "vertvel");

    if(ExpParams.cr_simi)
        formula = "(" + formula + ") * this.simi";
    GenotypeGroup.fitness = "return " + formula + ";";
}

function singlecrit(crname, fieldname)
{
    var weight = ExpParams.[crname];

    if(weight == 0.0)
        return "";
    if(ExpParams.cr_norm)
        return "+this.getNormalized(Genotype:"
            + fieldname + ")*" + weight;
    else
        return "+this." + fieldname + "*" + weight;
}

@include "BSC_events.inc"
@include "BSC_loadsave.inc"

~

#include "BSC_props.inc"

prop:
id:cleardata
name:Reset performance data
type:p

state:
id:notes
name:Notes
type:s 1
help:~
You can write anything here
```

```
(it will be saved to the experiment file)~

state:
id:totaltestedcr
name:Evaluated creatures
help:Total number of the creatures evaluated
in the experiment
type:d
flags:16

state:
id:creaturesgrouploaded
name:creaturesgrouploaded
type:d
flags:34
```