NTNU
Innovation and Creativity

Evolution of Control System(s) for a Multi Joint Snake

Transformer #13

Karl Hatteland

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Gunnar Tufte, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Evolving a Control System for a Multi Joint Snake (for artist purposes only). Make a graphical representation of the snake for simulation and implement a genetic algorithm to control it. And run it with different fitness functions to get different behavior from the snake.

Assignment given: 12. January 2007
Supervisor: Gunnar Tufte, IDI

Karl Hatteland

Evolution of Control System(s) for a Multi Joint Snake (for Artists purpose only)

Department of Computer and Information Science
Norwegian University of Science and Technology
N-7491 Trondheim, Norway

This is version 1.0, June 9, 2007.

Abstract

This project is about evolving a control system for a snake called Transformer <-> # 13. This is a mechanical snake with several body parts. The choise was to use a cellular genetic algorithm where each body part is a cell. These contain "DNA", one ruleset for each degree of freedom in the joints, which decides how it will behave in relation to its neighbour body parts. Three different fitness functions have been implemented which each gives a distinct and different behaviour. The goal of the different fitness functions is; crawling far, rising high and making geometry. The crawling part was successfull, while the other two goals was much harder for the snake and didnt provide great results. Concluding that the snake is appropriate for crawling around and making an impression of different cubic forms. Which for artist purposes is adequate, but it fails on getting into specific shapes.

Contents

List of Figures

Preface

“Trans<->Former # 13”

This project is a further development of the sculpture 11 cubes from 1998.

A 3.5 meter high steel piece constructed from one single line that due to it’s movements round the axis XYZ, based on a partically algorithm, forms a volume which becomes the modelled sculpture. The project are trying to develop computer controlled joints for this sculpture. The assembled sculpture will then be able to change its shape by automatically repositioning it’s parts. The sculpture parts mutual positions will be altered by various forms of external or internal trigging. The sculpture can interact with the audience at sight, by instance transforming it’s positions and shape according to the wievers amount and distance. The sculpture is powered by solarpanels and is undependent of external wiring.

As an art piece this sculpture origin within a constructuvist tradition, but as a moving object it can also be seen upon as part of a futurist movement. The object cant be read from just one angle of vision, it is in constant change despite it’s frozen state (originally). I like to think that it lives it own live, and that it suggest an alternative place of origin: in time , in space, in dimension.

A snapshot of an otherwise unavailable world.

Espen Gangvik, 2007

Acknowledgments

Thanks to Gunnar Tufte, Espen Gangvik and others who have given support to this thesis.

Karl Hatteland
June 9, 2007

Part I

Setting

Chapter 1

Introduction

This chapter will introduce the snake, touch some of the basics of Evolutionary Computation (EC) and comment on the choise of the evolutionary algorithm to be used. Evolutionary Computation will be explained more thoroughly in the Theory part, chapter 2. And give an introduction to the Breve simulation environment.

1.1 The Snake

The snake is a mechanical construct designed by artist Espen Gangvik, see figure 1.1. It is made up by body's which are linked together by joints. An earlier thesis by Ragnar Melz [8] has done an analysis of the snake and how it can be realized. Some of his findings have been used when simulating the snake to make the simulation as realistic as possible.

1.1.1 Mechanics

The snake used in the simulations consists of 9 bodies which are linked together by joints, an example can be seen in figure 1.2. The joints have two degrees of freedom they can rotate around their own axis and it can bend in one direction. This effectively makes it possible to reach all directions possible for the joint.

The rotating joint is restricted to rotate from neg 180 to pos 180 degrees which is a full circle. There were discussions in Melz thesis about rotation joints that didnt need wires running thru it, which would avoid the problem of wires getting twinned by the rotation. This would enable the rotating joint to be able to go on and on, without ever having to think about twisting wires. It was however decided that the restriction was to be used when simulating, a solution which would work in either way the snake was made rather than implementing a solution that might not work.
In his thesis Melz set the two motors in different locations. The rotating one was in

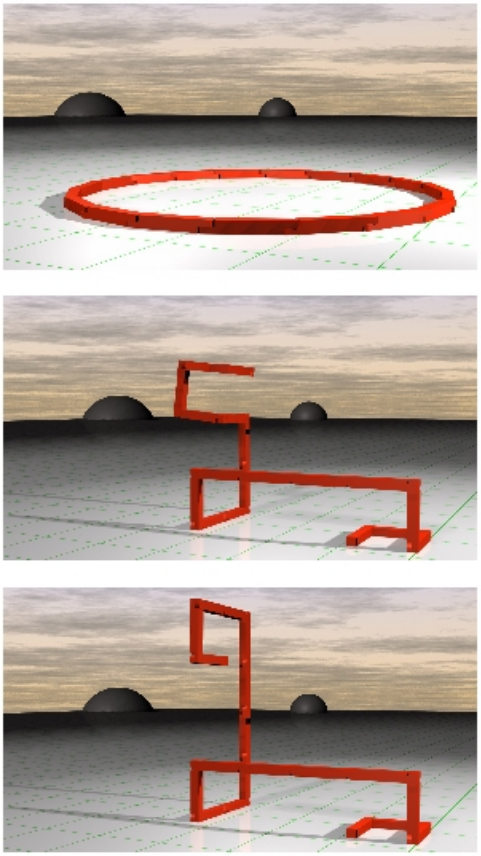

Figure 1.1: Possible positions of the snake as imagined by Espen Gangvik.

Figure 1.2: A picture of the simulated snake.

the midle of the body part while the bending one was the one who actually linked it to the next body part. In the simulation both rotation and bending is done in the same place. This difference in location is trivial; there are no changes in the behavior of the snake from this difference.

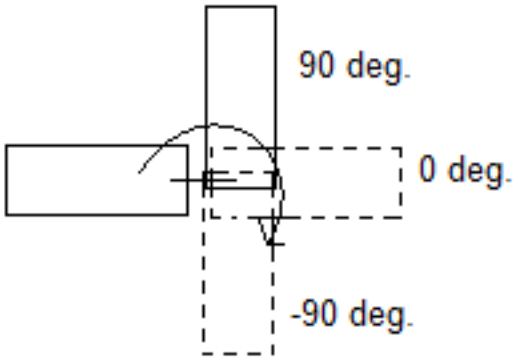

Figure 1.3: A picture showing the possible positions of the bending joint.

The bending joint bends from neg 90 degrees to pos 90 degrees. The reason for this is that we use 90 degrees steps to get the cubic look of the snake and the -180 or 180 degrees would bend the snake over and it would crash with its neighbour so the possible possitions would be -90, 0 and 90degrees, see figure 1.3.

1.1.2 Behaviour

The behaviour of the snake is not forced to any particular pattern. It will react and adapt to the environment and in some cases input from onlookers.
Each of the snake's body parts contains two rule sets which are used to decide the next movement of the snake. These rules are changed by an evolutionary algorithm. The bodies are awarded their own fitness value depending on how well they accomplish the goal that is set for the snake. The rulesets are copied between the bodies and every time that happens there is a small chance for mutations of the rulesets.
So thru these copying and mutation of the rulesets the snake change its behavior over time, adapting to the environment.

1.1.3 Art and Ideas

To get elegant movement, the speeds of the joints are restricted to a maximum of 30 degrees per second. And they are restricted to increments of 90 degrees per movements so as to make the figure the snake makes appear volumetric. This restriction is not set in stone and can be changed depending on how the snake will be used. Possible changes would be to allow increments of 45 degrees. Or even free movement.

The first idea that prompted this snake was the sculpture 11 cubes from 1998. A snake that would rotate and bend itself until it finally settled into that position. See figure 1.4 for an example of the 11 cubes shape, designes by Espen Gangvik.

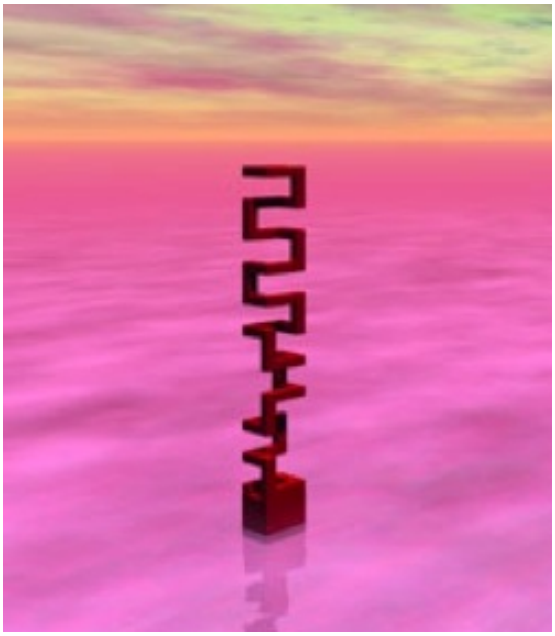

Figure 1.4: Trans<->Former #13. Envisioned by Artist Espen Gangvik

Other ideas have been to put the snake in glass cubicle in public areas and have the snake move inside it and change shape depending on the number of onlookers and how close they are or thru the onlookers choosing its decided goal (fitness function) thru options on a computer.
Think about hanging the snake up on some loose wires one at each end, and have it take different shapes in the middle of the air, when it contracts it would go higher and when it stretches out it would go lower. Or one could hang it from one end so when all stretched out touches the ground. The artistic possibilities for the snake is great.

1.2 Evolutionary Computation

Evolutionary Computation (EC) is the section of computer science which uses the principles of evolution to find solutions to non-linear and complex systems. There is several different approaches to evolutionary computation, some of these are; genetic algorithms (GA) [4], genetic programming (GP) [7], evolutionary strategies (ES) [10] and evolutionary programming (EP) [2]. All of these implement in different ways two functions 1) random search and 2) selection.

Figure 1.5 is showing a fictional search space and the steps of a genetic algorithm.

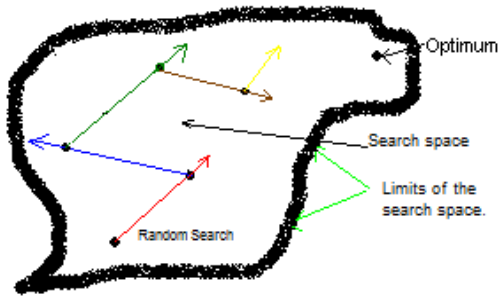

Figure 1.5: An example search space, showing the initial random search, the arrows show the direction each generation and the convergance towards the optimum.

Usually the optimum is not known, but it was added to show how the algorithm can converge on local or global maximums. Even if it's stuck on a local maximum, mutations make it possible to escape them and eventually find the global max.

1.2.1 Evolutionary Computation in control systems

Evolutionary Computation is being used more and more recently to control systems, often making the behavior seem realistic. Such effects are used in computer games where there are computer controlled agents, they then learn and adapt to a changing situations. Sushil J. Louis and Chris Mules used a method they called CIGARs (**c**ase-**i**njected **g**enetic **a**lgo**r**ithms) [9]. To improve computer game play.
Yannakakis and Hallam write about evolving opponents for interresting interactive computer games. Using the familiar Pac-Man game they comment that the emergent near-optimal behaviours of the predators makes the game less interresting to play, being too hard [3].
Maybe the most famous use of evolutionary computation to design and control agents is the work done by Karl Sims. His creations has evolved physically by using a genetic algorithm and then teached to walk, swim, jump and follow [11]. See figure 1.6 for some examples of creatures evolved for swimming.

As a commercial product that uses evolutionary computing one has Sonys dog Aibo which uses neural nets to learn to recognize people and situations and reacting thereafter [13].

1.2.2 Evolutionary Computation and the environment

Using a genetic algorithm as an example, it creates a generation of several individuals. These individuals have some traits, which is evaluated and gives them a fitness value depending on how well they perform. When the environment change, then the traits that was good before may now be bad for the individual. And those traits that were

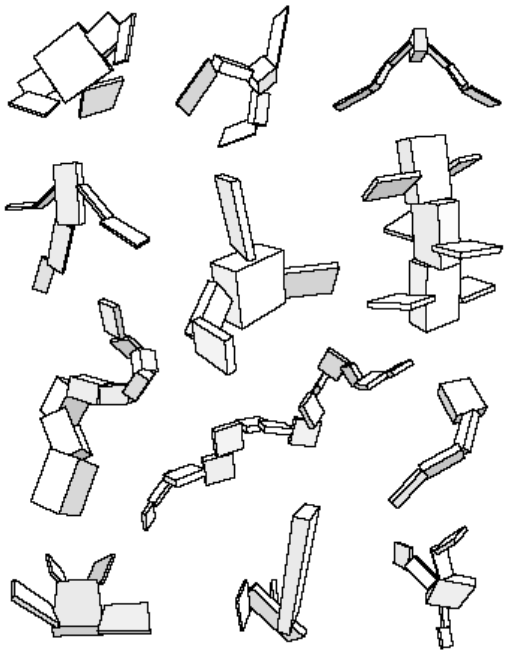

Figure 1.6: The creations of Karl Sims which has been tought to swim [11].

hurting an individual may now be good. This will affect the fitness value for the individuals, because they do no longer work as good. Thus changing which individuals gets selected for next generation. And after a couple of generation the individuals that live now have adapted to the new environment. The communication between the environment and the genetic algorithm is illustrated in figure 1.7.

1.2.3 Choice of evolutionary algorithm

The evolutionary algorithm that is going to be used in the snake has to be adaptive to environment and that the number of body parts of the snake change, making the snake longer or shorter. It cannot take too much computing power per body part, because the computing power is limited. The computer power is limited by more than the speed and effectivity of the micro controller, it is also limited by the available power source, batteries or solar panels. So the lesser amount of time and power needed to do the calculations the better.
To meet these requirements the choise was a cellular genetic algorithm. This algorithm meets the requirements. It is adaptable to the environment; it's easy to add body parts to the snake. In addition it can also provide the movement desired by artist Espen Gangvik. The computing power needed is relative small because there are no global control, every body part acts individually. The body parts just look at its position relative to its neighbour and looks up its rulesets and acts as the ruleset says. A simple example shows how the internal communication between the body parts work, see figure 1.8. The workings of the cellular genetic algorithm is explained in detail in section 2.3.

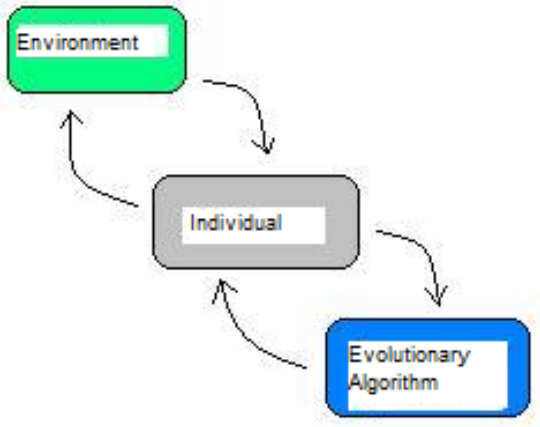

Figure 1.7: An illustration of how the interaction goes. The environment affects the individual, making it perform differently which in turn changes the individuals fitness value. And in turn affect the outcome of the selection made by the genetic algorithm.

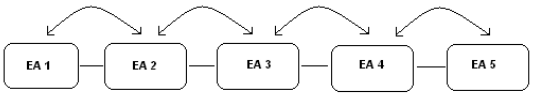

Figure 1.8: Each body part can only communicate to the neighbouring body parts. And the communication is limited to exchanging rulesets and asking about the neighbours' fitness. And when communicating rulesets mutations can occur.

1.3 Breve – a 3D simulation environment

Breve is a simulation environment with a physics engine. It is open source and free to use. It also includes several classes for setting up floor, objects, connect the objects to eachother, write to file and apply physics-like conditions like collision and gravity. It comes in two versions, IDE and CLI [6].

1.3.1 IDE and CLI mode

The IDE version contains a source code editor and a window showing the simulation in real time. It also has the ability to record movies of the simulations or take snapshots. This is really nice when developing the snake. Being able to check that the snake behaves correctly and see the changes directly. Breve parses the source file at simulation start, so after you have done a change it's just to stop and restart the simulation for changes to take effect. A picture displaying a screenshot of Breve IDE can be seen in figure 1.9.

The CLI version is the command line interface, when used the simulation goes alot faster, but no graphics are shown. When running longer simulations this is the only way, else the simulations would take very long time.

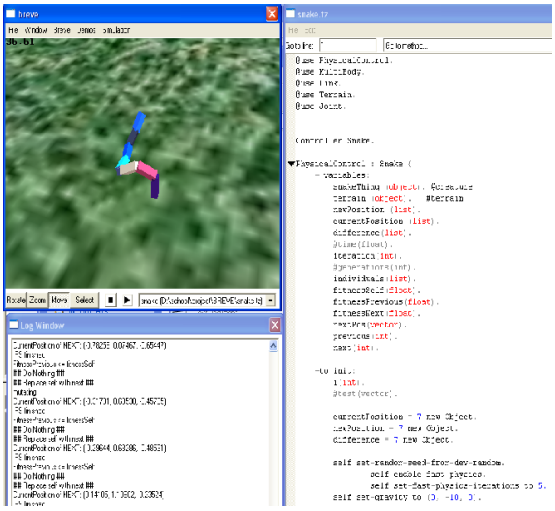

Figure 1.9: A screenshot of the snake in an early implemetation showing the 3D simulation window in the upper left. A log window at the lower left and the source code on the right side.

1.3.2 Programming language

From the documentation of the programming language of Breve: "Simulations in breve are written using a language called "steve". Steve is a simple language which aims to allow rapid construction of advanced simulations while avoiding a great deal of the programming overhead associated with constructing simulations in other languages."
Steve is an object-oriented language. This means that programming in steve involves working with components called objects which contain data (variables) and behaviors (methods). In breve, objects can be either real, meaning they have a presence in the simulated world; or abstract meaning that they are used to store data or to perform computations, but do not appear in the simulated world.
For documentation on the steve language check out the Breve webpage [5].

1.4 Objectives

The objectives for this thesis are to create the snake in a simulation environment. Implement the cellular genetic algorithm on the simulated snake. And run experiments to test the implemented algorithm in different situations. These differnt situations are created by changing the fitness function to get the snake to act towards different goals.

1.4.1 Setting up the snake in Breve

The first objective is implementing the snake and the cellular genetic algorithm in Breve.

1.4.2 Experiment 1: GoFar

This experiment is trying to make the snake go as far as possible. This is the simplest experiment, because very many movements of the snake will achieve this.
The distance is calculated using $\sqrt{x^2 + z^2}$. This gives the position in the floor plane, the height isnt neccesary to include. This function also does so that the snake gets better fitness nomatter which direction it chooses. The only thing that matters is that it gets futher away from the start position.

1.4.3 Experiment 2: GoHigh

This experiment is trying to get the snake to strech itself upwards. For the snake this is a harder experiment. There is less choises that lead to success compared to experiment 1. Many of the movements the snake can do will lead to loosing balance and fall, thus loosing heigth. Can there be evolved a rule set which manage to rise the snake high?
The global fitness here is the fitness of the highest body part of the snake. The fitness of the individual body parts are 0, 0.5 of 1. 0 if its lower than its neighbour, 0.5 if its equal and 1 if its better.

1.4.4 Experiment 3: GoCircle

This experiment will try to get the snake into a specific position. A circle is defines as the state where all the angles are the same. The fitness is calculated after how close to the needed angle the snake is. Using 9 body parts, the angle between each body part would have to be 36degrees to get the closest possible to a circle for the snake used in the simulations.

1.5 Results

The results are varied between the different experiments. As expected the first experiment proved to be easier for the snake and it managed well to keep going the same way when it first started a direction. Even if it kept going further and further in the long run it sometimes took lapses back, before going forward again. See section 3.1.1 for complete results and fitness over time graphs of the simulation.
The second experiment was not as successfull; the snake lifted itself from the ground, but kept falling down. From the logs it can be seen that the highest it got was 4 body parts length up. While normally it was only one or two body parts length above the ground. See section 3.1.2 for complete results.
The snake managed to partially make a circle but very often it moved into something resembling a circle and then after the next movement would go out of formation again.

1.6 Thesis Outline

Part 1 contains introductions.
Part 2 is the Theory part. It explains genetic algorithm and the cellular genetic algorithm in more detail.
Part 3 is experiments and results. This part will go thru exactly what was done and the results from the experiments.
Part 4 contains a discussion and conclusions and possible future work.

Part II

Theory

Chapter 2

Biologically Inspired

All the evolutionary algorithms are inspired by biology. We observe complex living organisms grow, adapt and learn. By examining how they accomplish this in nature we can try to replicate these techniques within computer programs to make the programs resilient, adaptable and able to find solutions in complex search spaces. Sipper et al [1] sets up a POE model which classify and describe the different aspects of biology and their properties.

2.1 POE model

"If one considers life on Earth since its very beginning, then the following three levels of organization can be distinguished" - Sipper et al [1]. These three levels are called **P**hylogenetic, **O**ntogenetic and **E**pigenetic (POE).

The first level Phylogeny concerns the evolution of the genetic code, similar to the evolution of species. Which is done by recombination of the genes and subject it to a low error (mutation) rate? The recombination and mutation provide diversity which is indispensible for the survival of a species. While it also makes the species able to adapt to the environment over time.

The second level Ontogeny concerns growth, similar to the division of a cell. Each cell being divided has the same code as the mother cell. The behavior of each new cell is deceided by the surrounding cells; this behavior is called cellular differentiation. Ontogeny thus makes creatures very robust, in the way that all the information of the whole is in each cell. And even if a great part of it is destroyed it can grow back.

The third level Epigenesis concerns accumulated knowledge, similar to a brain, nervous system and the immune system. Upon reaching a certain lever of complexity it is no longer possible to store all information in the cells directly, so there emerge a different process that permits the individual to integrate the interactions with the outside world. This process is called epigenesis. These are structures which are defined by the genome

and modified by experience.

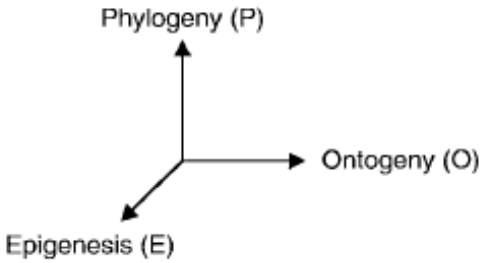

Figure 2.1: The three levels described as different planes.

2.2 Genetic Algorithms

Belonging to the phylogenetic level the genetic algorithm works by recombination and mutation. These algorithms encode a potential solution to a specific problem on a simple chromosome like data structure. This data structure is then modified by recombination of different potential solutions with an added small chance of mutation. Often the best potential solution is transfered to the next generation without change to not loose good solutions.

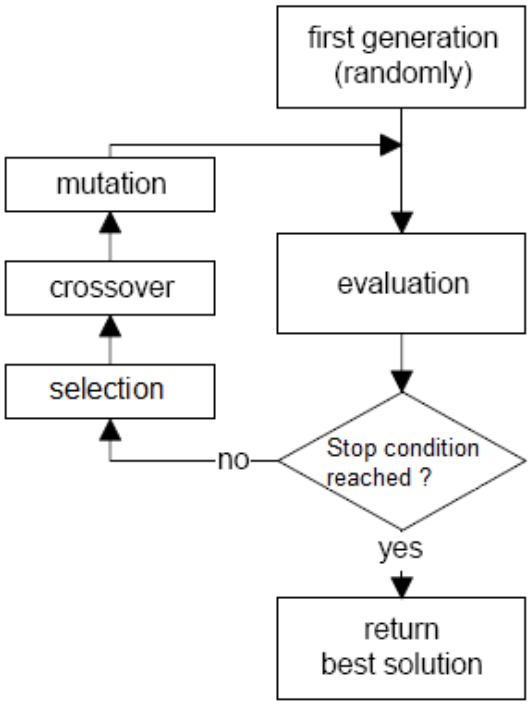

Figure 2.2: The flow of a general genetic algorithm.

Following figure 2.2 one first creates a generation and initializes them randomly. These are then evaluated and get assigned a fitness value. The fitness value will represent the solution we are searching for in such a manner that the closer the potential solutions comes to wanted solution the higher fitness the potential solution is assigned.

Then we check if a stop condition is reached, this might be how many generations has passed, the best fitness got over some level or simply a time constraint. If this condition is not met the GA goes on by making the next generation.

This is done by selection from the old generation; this is often used with roulette-wheel selection where the chance for being selected is proportional with the fitness of the potential solution. So the better fitness the greater chance at being selected. Two are selected and are crossovered and when crossover happens there is a small chance of mutation. The resulting new potential solution is added to the new generation. This is repeated until the new generation is as big as the previous one was.

The new generation is then evaluated and the cycle repeates until stop condition stops the algorithm.

2.3 Cellular Genetic Algorithms

To understand how Cellular Genetic Algorithms works, imagine the cells in your body. They communicate by chemicals; they can only communicate to their nearest neightbours. The number of neighbours also changes depending on where in the body the cells are located and what type of cells it is.

Moshe Sipper divides the cellular genetic algorithms into two; the uniform and the non-uniform cellular genetic algorithms. The uniform has the same rules in every cell while the non-uniform has different rules in every cell [12]. In non-uniform cellular genetic algorithms the evolution takes place not only in the state space but also in the rule space, meaning the rules will evolve and adapt. The state space is the physical position or the state of each cell while the rule space is the rules that dictate the cells behavior.

In this thesis the non-uniform cellular genetic algorithm will be used. The rulesets in each part of the snake will be different and they will be evolved over time.

The cellular genetic algorithm works in principle like the genetic algorithm from section 2.2. At the start each cells ruleset is randomized. Then the system goes on for a defined number of steps, the cells react and change depending on their neighbours as the rulesets decide. After the decided number of steps, all cells are given a fitness value telling how good each cell did to better the whole. Each cell then looks at its neightbours and three things may happen:

1. None of the neighbouring cells are better than self. If so, do nothing.
2. One and only one of the neighbours are better than self. If so replace its rulesets with that found in the one that was better.

3. If more than one of the neighbours are better than self. Using roulette wheel or similar selection method to select two of the neighbours, the ones with better fitness has better chance to be selected. Do a crossover of the two selected neighbours and replace the old ruleset with the result from the crossover.

Moshe Sipper supplies this pseudo code on page 81 of his book on cellular genetic algorithms [12].

```
for each cell i in CA do in parallel
   initialize rule table of cell i
   fi = 0 { fitness value }
end parallel for
c = 0 { initial configurations counter }
while not done do
   generate a random initial configuration
   run CA on initial configuration for M time steps
   for each cell i do in parallel
      if cell i is in the correct final state then
         fi = fi + 1
      end if
   end parallel for
   c = c + 1
   if c mod C = 0 then { evolve every C configurations }
      for each cell i do in parallel
         compute nfi(c) { number of fitter neighbors }
         if nfi(c) = 0 then rule i is left unchanged
         else if nfi(c) = 1 then replace rule i with the fitter neighboring
                            rule, followed by mutation.
         else if nfi(c) = 2 then replace rule i with the crossover of the
                            two fitter neighboring rules, followed by mutation.
         else if nfi(c) > 2 then replace rule i with the crossover of two
                            randomly chosen fitter neighboring rules, followed by mutation (this case c
         end if
         fi = 0
      end parallel for
   end if
end while
```

2.4 Implemented fitness functions

The cellular genetic algorithm was implemented in the snake so that each body part of the snake became its own cell. These cell contains rulesets which tells the body part how to react. The snake moves every 10 seconds. When it comes into position it waits a second or so before doing next move. In the implemented cellular genetic algorithm runs 10 time steps before it calculates new fitness and perform its genetic operations.

The fitness function for experiment one GoFar is the distance from the starting point. The floor created by Breve is only 1000x1000 units big and the start point is in the middle. This does so that the snake can fall over the side, when that happens the

current fitness is saved and the snake is transported back to the middle and continues its run from there. The shape of the snake is unchanged, in a way it doenst notice its been moved. One simulation atleast was ruined by this, the snake fell over and fell further and further away from the start point looking really good. Until it was too good to be true. As was mentioned earlier the global fitness is the distance the snake has traveled away from the start point. The fitness of the individual body parts is -1, 0 or 1 depending on their position compared to their previous position. If they are better than last time they get 1, similar they get 0 and if worse they got -1.

The fitness function for experiment two GoHigh works by giving fitness to the first body part using its height over the ground. For the other body parts its relative position to the previous neighbour is used. The second body part got fitness 1 if it was lower than the first body part, 0 if similar height and -1 if itself was higher. And for third it looked at the second. This would make the best possible fitness if the snake stood on its last body part and all the others was straight up.

The fitness function for the third experiment GoCircle counts the number of body parts, adds one, then takes 360degrees and divide it on the number of body parts + 1. This gives the angle that must be between each one if it is to make a circle. The fitness is then calculated from how close to this value it is. This experiment also require the snake to not only move in cubic 90 degree angles.

Part III

Results

Chapter 3

Results

3.1 Results

Each experiment was intended simulated for 8days in real time, which equates to about 10 million seconds simulated (115days). The snake move once every 10 seconds. And every 10 times it moves new fitness is calculated and genetic operations is performed. Giving about 100000 genetic operations per simulation. On the graphs the Y axis displays the global fitness of the snake while the X axis shows simulated time in seconds. Several of the simulation will however not be as long as 10 million seconds due to the computers running the simulations getting turned off, restarted or canceled for different reasons.
The rulesets found in the appendix is two rules for each body part. RuleX and RuleZ. X is the bending joint, while Z is the rotation. The rulesets look like:

```
RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57
```

The body part checks the angle, called A, between itself and the previous body part then the the angle, called B, between itself and the next body part. First for the X joint, if A is -1.57 and B is -1.57 then the next position will be the first value in the RuleX list, -1.57 and 0 the second etc. for all 9 combinations. This is then repeated for the Z angle. The resulting values are then used for the join between itself and the next body part.

3.1.1 Experiment 1: GoFar

Experiment GoFar was about getting the snake to go as far as possible. The graph in figure 3.1 shows the results from the first GoFar simulation. The resulting rulesets after the last genetic operation of the simulation one, can be found in the appendix A.1.1. The graph in figure 3.2 shows the results from the second simulation. And the evolved

ruleset from the second simulation can be found in the appendix A.1.2. The graph in figure 3.3 shows the results from the third simulation. The evolved rulesets from the third simulation can be found in the appendix A.1.3. The graph in figure 3.4 shows the results from the fourth simulation. The evolved ruleset from the fourth simulation can be found in the appendix A.1.4.

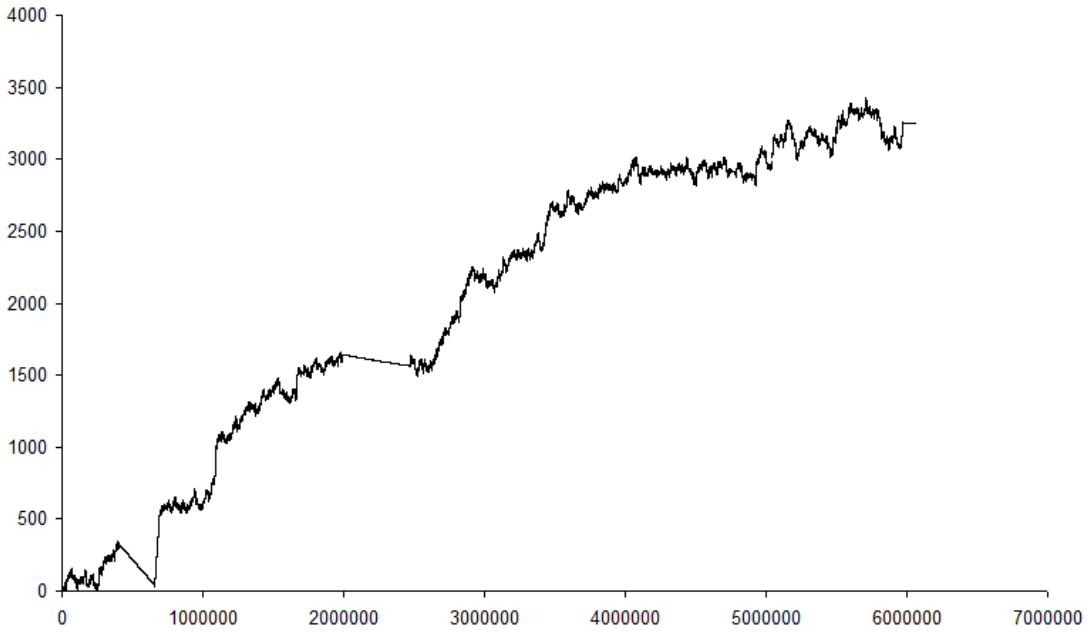

Figure 3.1: A graph showing results from the first GoFar experiment.

3.1.2 Experiment 2: GoHigh

The purpose of this experiment was to get the snake to try to rise as high as possible. Stretch one of its ends as high up as possible, doing this the snake need to balance well or it will fall. The results from one simulation can be seen in figure 3.5. While the evolved rulesets can be found in the appendix A.1.5.

3.1.3 Experiment 3: GoCircle

This experiment tries to make the snake attain certain shapes, in this case a circle. The result from this experiment can be seen in the figure 3.6. The resulting ruleset from this experiment can be found in the appendix A.1.6.

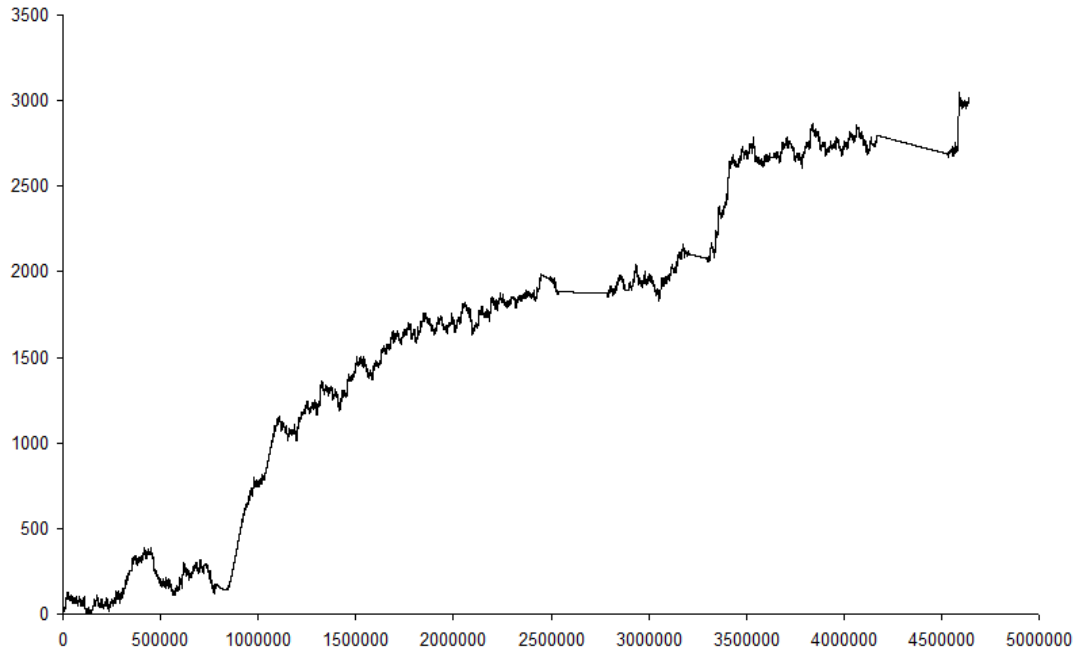

Figure 3.2: A graph showing results from the second GoFar experiment

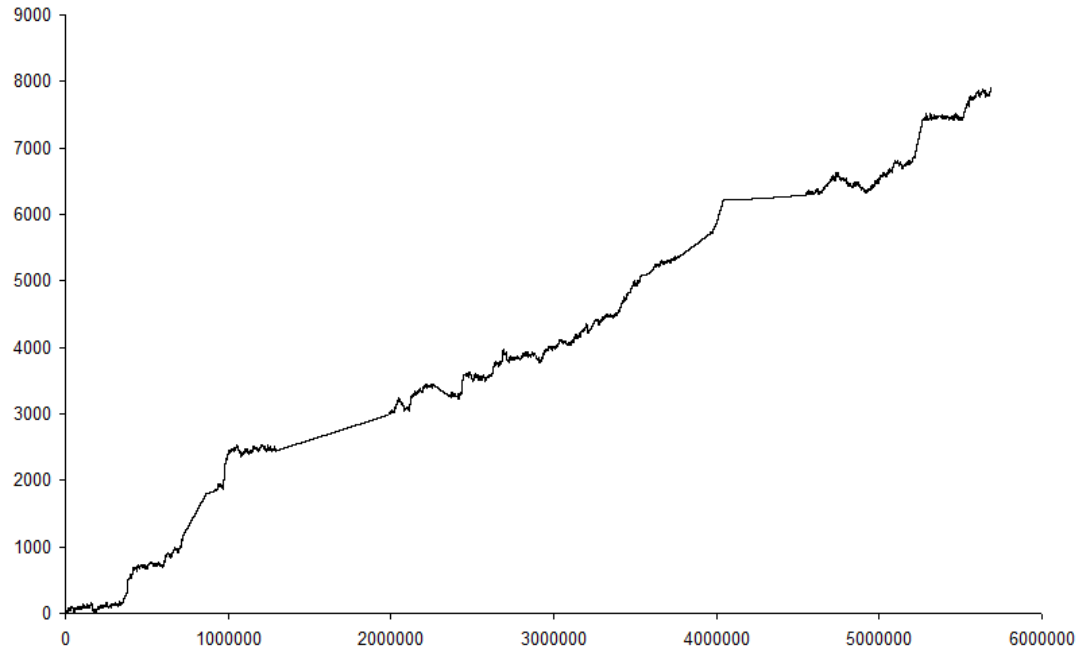

Figure 3.3: A graph showing results from the third GoFar experiment.

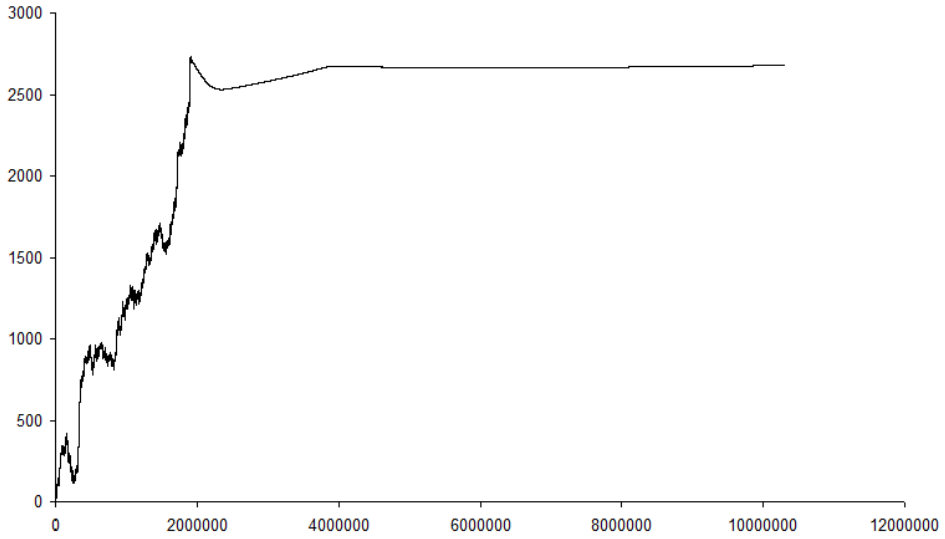

Figure 3.4: A graph showing results from the fourth GoFar experiment.

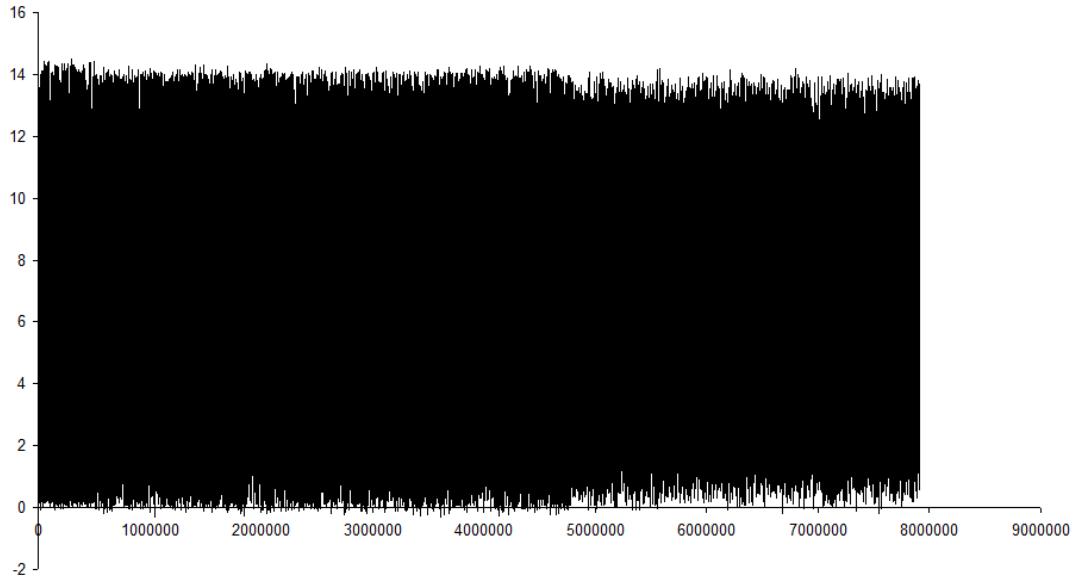

Figure 3.5: A graph showing results from the first GoHigh experiment.

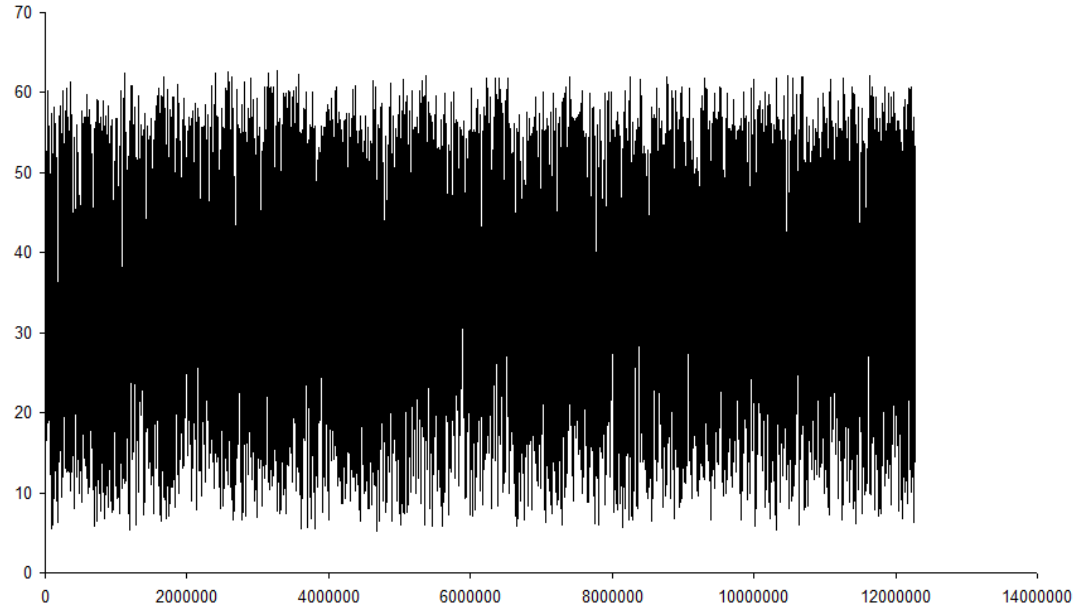

Figure 3.6: A graph showing results from the first GoCircle experiment.

Part IV

Synopsis

Chapter 4

Synopsis

4.1 Discussion

The results are varied. The expected graph for the first experiment would be a graph that average at the start then improved, making the graph steeper and steeper as the rulesets are optimized by the cellular genetic algorithm and its fitness function. Looking at the graphs in the figures 3.1 to 3.4 it is clear that the graphs goes in steps with some periods where they are actually going backwards. Even if the task is completed one can see from the graphs that there is no continuing progression where it improves much over time. If you look closer at figure 3.2 you can see that it found good rulesets to accomplish the task at about 800000 seconds into the simulation and it lasted until about 1.1million seconds into the simulation. One reason that it changed for worse can be that the mutation is too severe. At first it only changed one of values in the lists. The problem with this was that the snake ended up in repeating movement, and even after very long times, millions of seconds simulated, didnt get out of them. So the mutation was change to have greater effect and changed one of the rulesets in the individual completely. It looks like this change were too much and that something in the middle would be better. In figure 3.4 there is a period after about 2million seconds simulated where it gets stuck in a pattern of movement where it never really changes position.
Evaluating the rulesets from the four simulations from experiment one there is no obvious pattern. They all seem very different. This means that either the simulations need longer time to converge towards a similar ruleset or the cellular genetic algorithm isnt set properly to make the convergance. This could be because of the 10 steps between fitness evaluation or the mutation effect being so big it makes the cellular genetic algorithm almost like random search. To analyze this there need to be done more simulations with different settings.
The graph in figure 3.5 is not very nice. The fitness function gives fitness depending on how high above the ground the first body part is. The snake has a tendancy to get it somewhat high up and then falling down again. And you can see this in the graph where it varies between 0 and about 14 all the time. Something interresting happened

at 4.9milliong seconds simulated; the extreme positions of 0 and 14 didnt happen so often anymore. It came into a position where it kept the first body part up more reliable over time.
The third experiment was a challenge aswell for the snake, it tries to find a particular shape, while at the same time it changes form every 10 seconds. This resulted in the snake getting close, then next move it was no longer close to being a circle. A circle for the snake was defined to be 360degree divided on body parts + 1. So for the snake that was 9 body parts long it was 36degrees between each bodypart. The closer the angle got to 36degrees the better fitness it got. The graph in figure 3.6 is very similar to the graph for experiment GoHigh. This is because it moved into and out of position all the time.
The two experiments GoHigh and GoCircle did not go very well with the snake. It seemed to restless and didnt manage to follow a sequence of steps for GoHigh or sit in a position in GoCircle. For artist purpose it still looked great when moving and sometimes looked like its living. If the puprose of the snake was to do particular tasks the current control system would be inadequate. But for the purpose of electronic art in a glass cubicle og on the floor in a store it accomplish the goal of using movments which look cubic and give the impression of volume.

4.2 Conclusion

This document try to evolve a control system for a multi joint snake. With the imformation provided by Melz master thesis [8] the snake is implemented in the Breve simulation environment. Which provide a suitable environment to simulate the snake with realistic physics. A cellular genetic algorithm is implemented in the snake, due to its low demand on processing power, expandability and adaptibility. For simple tasks like going far in one direction the algorithm does well. The results of simulations can be seen in the result section 3.1. The second experiment were the snake was supposed to balance upwards and stretch itself as high as possible and the third where it was supposed to attain a special shape it performed poorly. When trying to stretch high it always did some movements that made it fall down. While when trying to find a specific shape, like the circle in experiment three, it sometimes got close, but then next time it moved again it moved out of the shape.

That been said the snake in experiment one does give the impression of volume when it moves with 90 degrees angles. So for artist purposes it functions well.

4.3 Future work

For experiment one what is needed as future work would be to run simulations and fine tune the mutation function. It does seem too severe at this moment. It could also be an idea to also figure in the speed of the movement when calculating the fitness. As now it gets good fitness if it moves away from the starting point, but it doesnt take into

account the speed it moves. This should be able to improve the snakes performance and perhaps attain the fast speed the snake shows from time to time. One example of this was in experiment two, figure 3.2, during the time from 800000 seconds to 1.1million second simulated time.

For movment to specific positions like in experiment two and three a change in the algorithm could make it possible to reliably solve the problems. Maybe evolve a sequence of movements that takes the snake from a position to the goal position. This could be done by a regular genetic algorithm and the snake performed one of the potential solutions in each generation, seeing how far it got.

Alot more simulations should be done aswell, as the low number of simulations here does not make the results statistically significant. It could be good luck or bad luck, thus making it impossible to generalise with too much certanity. This simulation would take a dedicated computer and alot of time to perform. With more simulations done with different settings of the cellular genetic algorithm it could be analyzed if other settings would make the evolved rulesets converge better toward an optimal configuration.

Part V

Appendices

Appendix A

Rulesets for the result section

A.1 Rulesets

Rulesets as they were when the simulation was ended.

A.1.1 GoFar one

The rulesets from the first GoFar simulation.

```
RuleX -1.57,  1.57,  1.57,  0.00,  0.00, -1.57, -1.57, -1.57,  1.57
RuleZ  1.57,  1.57,  1.57,  0.00,  1.57,  1.57, -1.57,  0.00,  1.57

RuleX -1.57,  1.57,  1.57,  0.00,  0.00, -1.57, -1.57, -1.57,  1.57
RuleZ  0.00, -1.57, -1.57,  1.57,  1.57,  0.00, -1.57,  0.00,  0.00

RuleX  0.00,  0.00,  1.57,  1.57,  0.00,  0.00, -1.57,  0.00, -1.57
RuleZ  1.57,  1.57,  1.57, -1.57,  1.57,  0.00, -1.57,  1.57,  0.00

RuleX  1.57,  1.57,  1.57,  0.00,  0.00,  0.00, -1.57,  0.00,  0.00
RuleZ  0.00,  0.00, -1.57,  0.00, -1.57, -1.57,  0.00,  1.57,  1.57

RuleX  1.57,  1.57,  1.57,  0.00,  0.00,  0.00, -1.57,  0.00,  0.00
RuleZ  0.00,  0.00, -1.57,  0.00, -1.57, -1.57,  0.00,  1.57,  1.57

RuleX  1.57,  1.57,  1.57,  0.00,  0.00,  0.00, -1.57,  0.00,  0.00,
RuleZ  0.00,  0.00, -1.57,  0.00, -1.57, -1.57,  0.00,  1.57,  1.57

RuleX  0.00, -1.57, -1.57,  0.00,  0.00,  0.00,  0.00,  0.00, -1.57
RuleZ  0.00, -1.57,  1.57,  1.57,  0.00,  0.00, -1.57, -1.57,  0.00
```

```
RuleX  0.00, -1.57,  1.57,  1.57, -1.57,  0.00,  1.57,  1.57, -1.57
RuleZ  0.00,  0.00, -1.57,  0.00,  0.00,  1.57,  0.00,  0.00,  1.57

RuleX  0.00, -1.57,  1.57,  1.57, -1.57,  0.00,  1.57,  1.57, -1.57
RuleZ  0.00,  0.00, -1.57,  0.00,  0.00,  1.57,  0.00,  0.00,  1.57
```

A.1.2 GoFar two

The rulesets from the second GoFar simulation.

```
RuleX  1.57,  1.57,  1.57,  0.00,  1.57, -1.57, -1.57,  1.57,  1.57,
RuleZ -1.57,  0.00, -1.57,  0.00,  1.57,  1.57,  1.57, -1.57, -1.57,

RuleX  1.57,  1.57,  1.57,  0.00,  1.57, -1.57, -1.57,  1.57,  1.57
RuleZ -1.57,  0.00, -1.57,  0.00,  1.57,  1.57,  1.57, -1.57, -1.57

RuleX  1.57,  0.00,  0.00, -1.57, -1.57,  1.57,  0.00, -1.57, -1.57
RuleZ -1.57, -1.57, -1.57, -1.57, -1.57,  1.57,  1.57,  1.57, -1.57

RuleX  1.57,  1.57,  1.57,  0.00,  1.57, -1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  1.57, -1.57,  1.57,  1.57,  0.00, -1.57,  1.57

RuleX  1.57,  1.57,  1.57,  0.00,  1.57, -1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  1.57, -1.57,  1.57,  1.57,  0.00, -1.57,  1.57

RuleX  1.57,  1.57,  1.57,  0.00,  1.57, -1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  1.57, -1.57,  1.57,  1.57,  0.00, -1.57,  1.57

RuleX  0.00,  1.57,  1.57,  1.57,  1.57, -1.57, -1.57,  0.00,  1.57
RuleZ  1.57,  0.00, -1.57,  1.57, -1.57, -1.57,  0.00, -1.57,  1.57

RuleX -1.57, -1.57,  1.57, -1.57,  0.00, -1.57,  0.00,  1.57,  0.00
RuleZ -1.57, -1.57,  1.57, -1.57,  1.57,  1.57,  0.00, -1.57,  1.57
```

A.1.3 GoFar three

The rulesets from the third GoFar simulation.

```
RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57
```

```
RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57

RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57

RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57

RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57

RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57

RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57

RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57

RuleX  1.57,  1.57,  0.00,  1.57,  0.00,  0.00, -1.57,  1.57,  0.00
RuleZ  0.00, -1.57,  0.00, -1.57,  1.57,  0.00,  0.00,  0.00, -1.57
```

A.1.4 GoFar four

The rulesets from the fourth GoFar simulation.

```
RuleX  1.57,  0.00,  0.00, -1.57,  1.57,  1.57,  1.57, -1.57,  1.57
RuleZ  0.00, -1.57, -1.57,  1.57,  1.57, -1.57,  1.57,  0.00, -1.57

RuleX  0.00,  0.00,  0.00,  0.00,  0.00,  1.57,  0.00,  0.00,  0.00
RuleZ -1.57,  1.57, -1.57,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57

RuleX  0.00,  0.00,  0.00,  0.00,  0.00,  1.57,  0.00,  0.00,  0.00
RuleZ -1.57,  1.57, -1.57,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57

RuleX  0.00,  0.00,  0.00,  0.00,  0.00,  1.57,  0.00,  0.00,  0.00
RuleZ -1.57,  1.57, -1.57,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57

RuleX  0.00,  0.00,  0.00,  0.00,  0.00,  1.57,  0.00,  0.00,  0.00
RuleZ -1.57,  1.57, -1.57,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57

RuleX  0.00,  0.00,  0.00,  0.00,  0.00,  1.57,  0.00,  0.00,  0.00
```

```
RuleZ -1.57,  1.57, -1.57,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57

RuleX  1.57, -1.57,  1.57,  0.00,  1.57,  0.00, -1.57, -1.57,  0.00
RuleZ  1.57,  1.57,  0.00,  0.00, -1.57,  1.57, -1.57,  0.00,  0.00

RuleX -1.57, -1.57, -1.57,  1.57,  0.00, -1.57,  0.00,  1.57,  0.00
RuleZ -1.57, -1.57,  0.00,  1.57,  1.57,  1.57, -1.57,  1.57,  0.00

RuleX  1.57, -1.57,  1.57,  0.00,  1.57,  0.00, -1.57, -1.57,  0.00
RuleZ  0.00,  1.57, -1.57, -1.57,  1.57,  0.00, -1.57, -1.57,  0.00
```

A.1.5 GoHigh one

The rulesets from the first GoHigh simulation.

```
RuleX  0.00,  0.00,  0.00,  0.00, -1.57, -1.57,  1.57, -1.57,  0.00
RuleZ  1.57,  0.00,  1.57,  1.57, -1.57, -1.57,  1.57,  1.57, -1.57

RuleX -1.57,  0.00, -1.57,  1.57,  0.00,  1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  0.00,  0.00,  0.00,  1.57,  0.00, -1.57, -1.57

RuleX -1.57,  0.00, -1.57,  1.57,  0.00,  1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  0.00,  0.00,  0.00,  1.57,  0.00, -1.57, -1.57

RuleX -1.57,  0.00,  0.00,  1.57, -1.57, -1.57,  0.00,  1.57,  1.57
RuleZ -1.57,  1.57,  1.57,  1.57, -1.57, -1.57,  0.00,  0.00, -1.57

RuleX -1.57,  0.00, -1.57,  1.57,  0.00,  1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  0.00,  0.00,  0.00,  1.57,  0.00, -1.57, -1.57

RuleX -1.57,  0.00, -1.57,  1.57,  0.00,  1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  0.00,  0.00,  0.00,  1.57,  0.00, -1.57, -1.57

RuleX -1.57,  0.00, -1.57,  1.57,  0.00,  1.57, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57,  0.00,  0.00,  0.00,  1.57,  0.00, -1.57, -1.57

RuleX  0.00, -1.57,  1.57, -1.57,  1.57,  0.00,  0.00, -1.57,  1.57
RuleZ  0.00,  1.57,  0.00,  0.00,  1.57,  0.00,  0.00,  0.00,  1.57
```

A.1.6 GoCircle one

```
RuleX  1.57,  1.57, -1.57, -1.57,  0.00,  0.00, -1.57, -1.57,  1.57
RuleZ  0.00,  1.57,  1.57, -1.57,  1.57, -1.57,  1.57,  1.57,  0.00

RuleX  1.57,  1.57, -1.57, -1.57,  0.00,  0.00, -1.57, -1.57,  1.57
RuleZ  0.00,  1.57,  1.57, -1.57,  1.57, -1.57,  1.57,  1.57,  0.00

RuleX  0.00, -1.57, -1.57,  0.00, -1.57, -1.57,  0.00, -1.57,  1.57
RuleZ  0.00,  1.57,  1.57, -1.57,  1.57, -1.57,  1.57,  1.57,  0.00

RuleX  0.00, -1.57, -1.57,  0.00, -1.57, -1.57,  0.00, -1.57,  1.57
RuleZ  0.00, -1.57,  0.00,  0.00,  0.00, -1.57,  1.57, -1.57,  1.57

RuleX  0.00, -1.57, -1.57,  0.00, -1.57, -1.57,  0.00, -1.57,  1.57
RuleZ  0.00, -1.57,  0.00,  0.00,  0.00, -1.57,  1.57, -1.57,  1.57

RuleX  0.00,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57,  1.57,  1.57
RuleZ  0.00, -1.57,  0.00,  0.00,  0.00, -1.57,  1.57, -1.57,  1.57

RuleX  0.00,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57,  1.57,  1.57
RuleZ  0.00, -1.57,  0.00,  0.00,  0.00, -1.57,  1.57, -1.57,  1.57

RuleX  0.00,  0.00, -1.57,  1.57, -1.57, -1.57,  1.57,  1.57,  1.57
RuleZ  0.00, -1.57,  0.00,  0.00,  0.00, -1.57,  1.57, -1.57,  1.57

RuleX -1.57, -1.57, -1.57, -1.57, -1.57,  0.00, -1.57,  1.57,  1.57
RuleZ -1.57, -1.57, -1.57,  1.57, -1.57,  1.57, -1.57, -1.57,  0.00
```

Bibliography

[1] Moshe Sipper et al. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, april 1997.

[2] Back T. Rudolph G. and Schwefel H. P. Evolutionary programming and evolution strategies: Similarities and differences. *Proceedings of the Second Annual Conference on Evolutionary Programming, Evolutionary Programming Society*, pages 11–22, 1993.

[3] Georgios N. Yannakakis & John Hallam. Evolving opponents for interesting interactive computer games. *not known*, not known.

[4] Holland J.H. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press., 1975.

[5] Jon Klein. Breve: a 3d simulation enviroment for multi-agent simulations and artificial life. [http://www.spiderland.org/breve/].

[6] Jon Klein. Breve: a 3d environment for the simulation of decentralized systems and artificial life. Technical report, Complex Systems Group at Department of Physical Resource Theory, Goteborg Univeristy Sweeden and Cognitive Science, Hampshire COllege Amherst, MA 01002, 2005. [http://www.spiderland.org/breve/].

[7] Koza J.R. Bennett F.H. III Andre D. Keane M.A. and Dunlop F. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1(2):109–128, 1997.

[8] Ragnar Melz. Skulptur i bevegelse. MSc thesis, Norwegian University of Science and Technology, Institutt for kybernetikk, June 2003.

[9] Sushil J. Louis & Chris Miles. Playing to learn: Case-injected genetic algorithms for learning to play computer games. *IEEE Transactions on Evolutionary Computation, VOL 9, NO.6*, pages 669–681, 2005.

[10] I. Rechenberg. *Evolution Strategy '94.* Frommann-Holzboog, Stuttgart., 1994.

[11] Karl Sims. Evolving virtual creatures. *Computer Graphics Annual Conference Series (SIGGRAPH 94 Proceedings)*, pages 15–22, 1994.

[12] Moshe Sipper. *Evolution of Parallel Cellular Machines*. Springer, 1997.

[13] Sony. Aibo. [http://www.sony.net/Products/aibo/].