Chapter 5
Framsticks: Creating and Understanding Complexity of Life

Maciej Komosinski and Szymon Ulatowski

Life is one of the most complex phenomena known in our world. Researchers construct various models of life that serve diverse purposes and are applied in a wide range of areas – from medicine to entertainment. A part of artificial life research focuses on designing three-dimensional (3D) models of life-forms, which are obviously appealing to observers because the world we live in is three dimensional. Thus, we can easily understand behaviors demonstrated by virtual individuals, study behavioral changes during simulated evolution, analyze dependencies between groups of creatures, and so forth. However, 3D models of life-forms are not only attractive because of their resemblance to the real-world organisms. Simulating 3D agents has practical implications: If the simulation is accurate enough, then real robots can be built based on the simulation, as in [22]. Agents can be designed, tested, and optimized in a virtual environment, and the best ones can be constructed as real robots with embedded control systems. This way artificial intelligence algorithms can be "embodied" in the 3D mechanical constructs.

Perhaps the first well-known simulation of 3D life was Karl Sims' 1994 virtual creatures [34]. Being visually attractive, it demonstrated a successful competitive coevolutionary process, complex control systems, and interesting (evolved) behaviors. However, this work did not become available for users as documented software. A number of 3D simulation engines was developed later (see [36] for a review), but most of them are either used for a specific application or experiment (and are not available as general tools for users) or focus primarily on simulation (without built-in support for genetic encodings, evolutionary processes or complex control). Notable exceptions are the breve simulation package described in Chapter 4 and a recent version of StarLogo – StarLogo TNG (Chapter 6).

Framsticks [19, 17, 18], a software platform described in this chapter, does not address a single purpose or a single research problem. On the contrary, it is built to support a wide range of experiments and to provide all of its functionality to users, who can use this system in a variety of ways. The significance of understanding is central for the development of Framsticks. Although the system is a simplified model of reality, it is easily capable of producing phenomena more complex than a human can comprehend [14]. Thus, it is essential to provide automatic analysis and support tools. Intelligible visualization is one of the most fundamental means for human understanding of artificial life-forms, and this feature is present in the software.

The Framsticks system is designed so that it does not introduce restrictions concerning complexity and size of creatures. Therefore, neural networks can have any topology and dimension, allowing for a range of complex behaviors, some described in Section 4 of [14]. Avoiding limitations is important because Framsticks is ultimately destined to experiments with open-ended evolution, where interactions between creatures and the environment are the sources of competition, cooperation, communication, and intelligence.

Further sections of this chapter focus on the following issues:

5.1 — history of Framsticks and available software
5.2 — simulation model, morphology, control system, communication, environment
5.3 — genetics, genotype–phenotype relationship, mutation and crossing over, evolution
5.4 — scripting, experiment definitions and system framework, creating custom neurons and experiments, popular experiments
5.5 — selected tools provided to support research and education
5.6 — sample experiments that have already been performed, as well as some new ideas
5.7 — entertaining Framsticks applications
5.8 — summary.

5.1 Available Software and Tools

Framsticks was first released in late 1996, but the first official releases became available on the Internet in June 1997. Versions 1.x provided numerous parameters to customize experiments, but the experiment logic, visualization, and neurons were hard-coded.

In 2002, starting with version 2.0, the scripting language *FramScript* was introduced, which allows for flexible control of most parts of the software – on both high level (adjusting parameters) and low level (creating custom procedures). Scripting is addressed in Section 5.4.

In 2004, the first official release of the Framsticks Theater (a simple-interface, attractive graphical application) took place, with unofficial releases available since 2002.

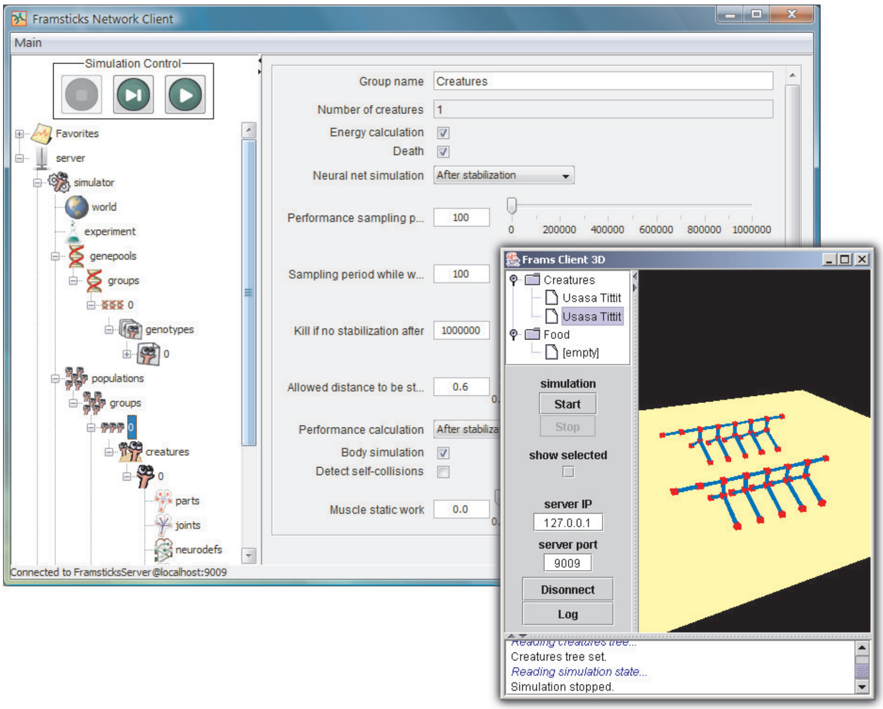
In 2008, version 3.0 introduced an accurate mechanical simulation mode. Useful features were added to the particle agents simulation mode and support for agent communication was greatly enhanced. A few applications were joined: The Framsticks Viewer was included in the Framsticks Theater, and the Framsticks Server was included in the Framsticks Command-Line Interface.

The Framsticks family of programs includes the following:

- Framsticks GUI (Graphical User Interface) – the most popular program, where simulated creatures, genotypes, and the virtual world are presented visually and allow for user interaction (dragging creatures, online genotype visualization, etc.).
- Framsticks CLI (Command-Line Interface) – a program where commands are issued using text. Useful for long, time-consuming and/or well-defined experiments, which can be performed automatically (batch processing) or remotely. This program has no overheads of the GUI and can be compiled for most operating systems. The CLI also acts as the Framsticks Network Server. The server communication protocol is published, which allows for development of third-party clients.
- Framsticks Theater – a complete Framsticks simulator with a simple menu and predefined actions ("shows"), described in more detail in Section 5.7. The Theater can also be used as a viewer of genotypes specified by users or read from files.
- Framsticks Editor (FRED) – a simple open-source graphical editor that allows users to easily design creatures without any knowledge about genetic encodings. It is mentioned in Section 5.7.
- Framsticks Network Clients – open-source programs that interact with the Framsticks Server (server(s) and client(s) can also be run on a single computer). Two basic roles of clients are (1) the GUI for the server and (2) visualization of the virtual world simulated on the server, as shown in Figs. 5.1 and 5.2. However, clients can use the server in a number of ways, including distributed evolution, modeling of ecosystems and migration, real-time interaction in mixed realities, and much more. Many clients can connect to the same server at the same time, and clients can exchange information between themselves.
- Other helper programs and scripts: brain optimizers, analyzers of experiment output data, graph generators, and so forth.

The above applications are in continuous development, with new releases coming out periodically. Framsticks genotypes and experiment proposals can be browsed, downloaded, and uploaded using the Internet database – Framsticks Experimentation Center.

Framsticks software documentation is available in many forms, including web site information [19], single-document Framsticks Manual, step-by-step tutorials (strongly recommended for beginners [20]), a web forum, and the FramScript reference that describes objects and functions useful when writing scripts.



Group name: Creatures
Number of creatures: 1
Energy calculation: ✓
Death: ✓
Neural net simulation: After stabilization
Performance sampling p...: 100
Sampling period while w...: 100
Kill if no stabilization after: 1000000
Allowed distance to be st...: 0.6
Performance calculation: After stabiliza
Body simulation: ✓
Detect self-collisions: ☐
Muscle static work: 0.0

Connected to FramsticksServer@localhost:9009

server IP: 127.0.0.1
server port: 9009

```
Reading creatures tree...
Creatures tree set.
Reading simulation state...
Simulation stopped.
```

Fig. 5.1 Two sample network clients: the GUI and the virtual world viewer.

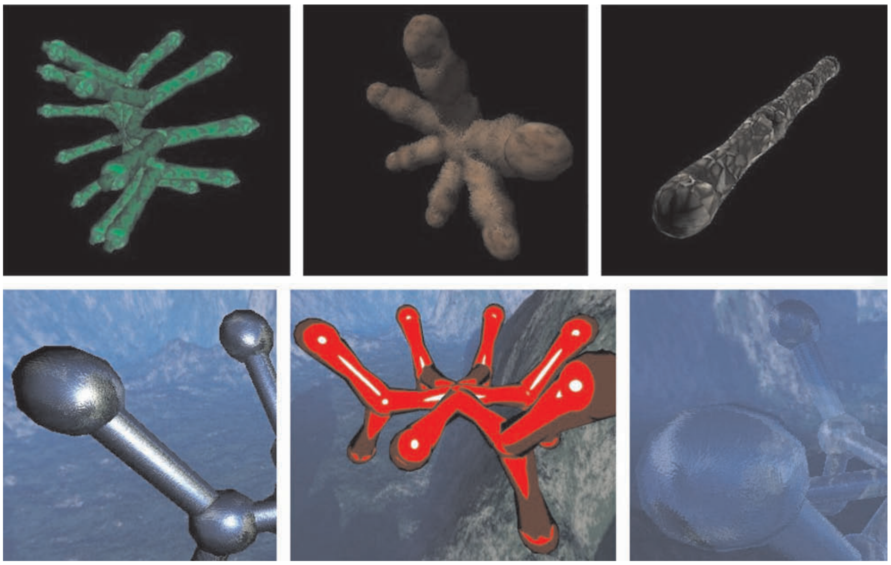


Fig. 5.2 Selected 3D real-time display styles in the Java-based network client. Top row: Basic, Fur, and Stone. Bottom row: Metal, Cartoon, and Glass.

5.2 Simulation

5.2.1 Creature Model

Creatures in Framsticks are modeled as bodies and brains. Bodies consist of body parts connected by body joints; this constitutes an undirected, spatial 3D graph structure. Brain is made of neurons (signal processing units, receptors, and effectors) and neural connections; these form a directed graph that may be connected to body where the neural units are embodied.

The four elements of the model – parts, joints, neural units, and neural connections – can be characterized by a variable number of properties like mass, position, orientation, friction, stiffness, durability, assimilation ability, weight, and neural parameters. For details on implementation of this model, refer to the GDK [37].

5.2.2 The Three Modes of Body Simulation

There is always a trade-off between simulation accuracy and simulation time. To perform evolution and evaluate thousands of individuals, fast simulation is required; on the other hand, the model should be realistic (detailed) enough to allow for realistic behaviors. When we expect emergence of more and more sophisticated phenomena, the evolution takes more time – thus simulation needs to be faster, and therefore less accurate. While some experiments are focused on realistic 3D mechanical simulations (e.g., in robotics), others are only concerned with information processing in the agent's brain (e.g., in Agent-Based Modeling applications).

In Framsticks step-by-step simulation, all major kinds of interactions between basic physical objects are considered: static and dynamic friction, damping, action and reaction forces, energy losses after deformations, gravitation, and uplift pressure – buoyancy (in a water environment). However, some experiments do not require such complexity and realism, therefore Framsticks provides three simulation modes to meet diverse requirements.

It is possible to adjust gravity force in all the three simulation modes. To improve performance, collision detection may be deactivated between agents or groups of agents (inter-creature collisions) when collisions are not of interest.

Accurate Rigid Body Dynamics

In this mode, Framsticks uses the Open Dynamics Engine [35] to simulate bodies made of boxes or cylinders that correspond to body parts and joints

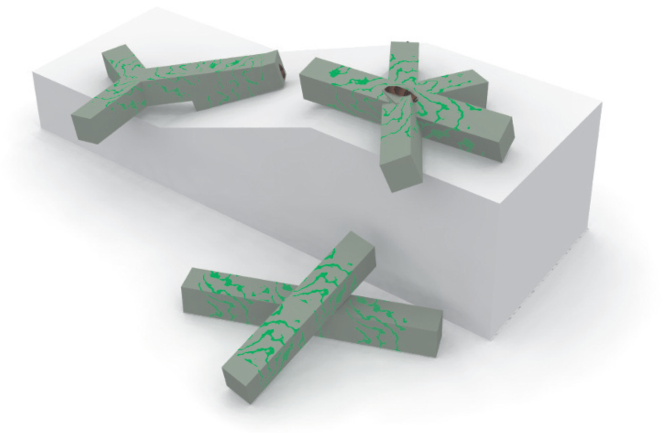


Fig. 5.3 Accurate collision detection in the rigid body dynamics mode.

(Figs. 5.3 and 5.4, left). The simulation of physics is accurate enough to expect similar behavior from the real-world robotic equivalent. A number of parameters are provided to customize simulation (cf. [16]); it is also possible to deactivate collisions among creature parts (intra-creature) when these are not required.

Simplified Elastic Body Dynamics

This mode uses the native Framsticks simulation engine, Mechastick (Fig. 5.5). It provides much better performance than ODE at the cost of realism. In order to avoid computational complexity in this simulation mode, intra-creature collisions are not detected. Creatures are built of interconnected linear segments ("sticks") corresponding to body parts and joints. Articulations exist between sticks where they share an endpoint; the articulations are unrestricted in all three degrees of freedom (bending in two planes plus twisting).

In this simulation mode, bodies exhibit elastic dynamics which limits size and complexity of constructs under normal gravity conditions. On the other

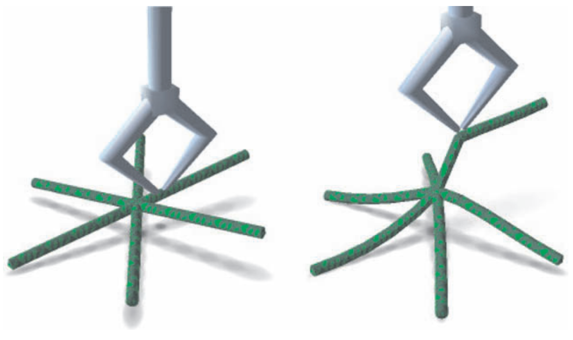


Fig. 5.4 Body dynamics simulation modes: rigid (left) and elastic (right).

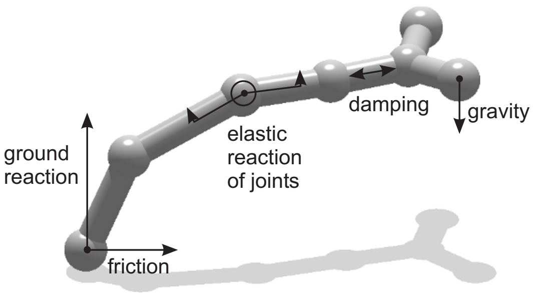


Fig. 5.5 Forces considered in Mechastick.

hand, flexibility and elastic deformations are extremely advantageous for efficient locomotion techniques (compare Fig. 5.4) and other activities. It is much harder to design or evolve naturally moving structures when rigid body dynamics is employed.

For each simulated element, Mechastick provides information about axial and angular stress values (Fig. 5.6). This information can be used to test and evolve stable, durable, and robust structures.

Particle Agents

When studying mechanics of the animal limb or robotic leg, one can spend many simulation steps to make the structure perform a single movement. However, there are many applications where morphology of agents is not considered at all, or the complete agent body is considered solid. Such experiments include navigation, communication, swarming and group behaviors, various massive multi-agent setups, and settings where agents are primarily considered as brains, not bodies.

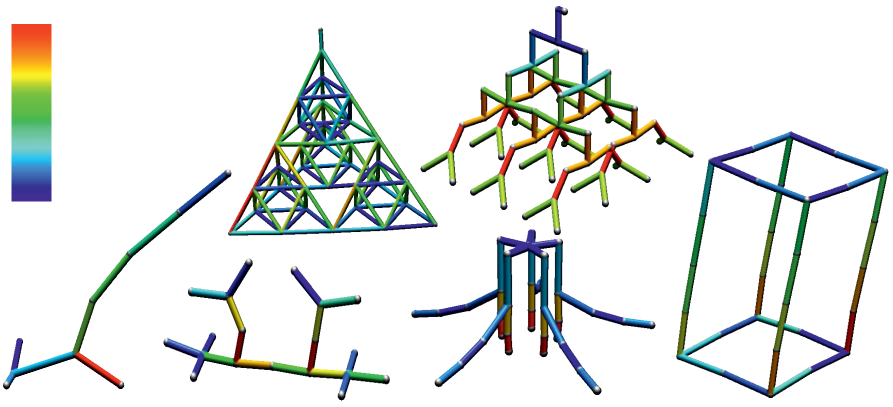


Fig. 5.6 Visualizations of axial stress in the Mechastick simulation engine.

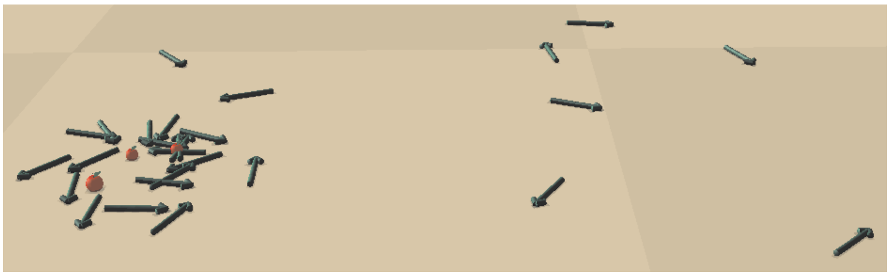


Fig. 5.7 Particle agents simulation mode. Dynamics of arrow-shaped bodies is not regarded in this experiment.

These experiments do not need true simulation of body dynamics – simple "move forward" or "turn left" commands are sufficient. In the "particle agents" mode, simple high-level methods can be used (rotating agents, setting their direction and speed) to facilitate experiment design (Figs. 5.7, 5.10, and 5.12). When bodies are not important, the simulation can be extremely simplified by making each agent a point of mass.

5.2.3 Brain

Brain (the control system) is made of neurons and their connections. A neuron may be a signal processing unit, but it may also interact with body as a receptor (sensor) or effector (actuator). There are some predefined types of neurons, for example:

- "N": the standard Framsticks neuron, which is a generalized version of the popular weighted-sum, sigmoid transfer function neuron used commonly in AI. The three additionally introduced parameters influence speed and tendency of changes of the inner neuron state and the steepness of the sigmoid transfer function. In a special case, when the three parameters are assigned specific values, the characteristics of the "N" neuron become identical to the popular, reactive AI neuron. In this case, neural output reflects instantly input signals. More information and sample neuronal runs can be found in the *simulation details* section at [19].
- "Sin": a sinusoidal generator with frequency controlled by its inputs.
- "Rnd": random noise generator.
- "Thr": thresholding neuron.
- "Delay": delaying neuron.
- "D": differentiating neuron.

It is possible to easily add custom, user-designed neurons using FramScript; an example is shown in Section 5.4.

The neural network can have any topology and complexity. Neurons can be connected with each other in any way (some may be unconnected). Inputs should be connected to outputs of another neuron (including sensors), while outputs should be connected to inputs of other neurons (including effectors). Sample control systems are shown in Fig. 5.8. Note that a single control system may contain many unconnected or independent subsystems.

Neurons can send multiple values in their output. This extension allows one to design complex neurons that provide a vector of output values, which is used, for example, in the Fuzzy and the Vector Eye neurons described later.

A section in Part I of the Framsticks tutorial [20] concerns understanding and designing control systems, and there are exercises in Part IV on the development of custom script-based neurons.

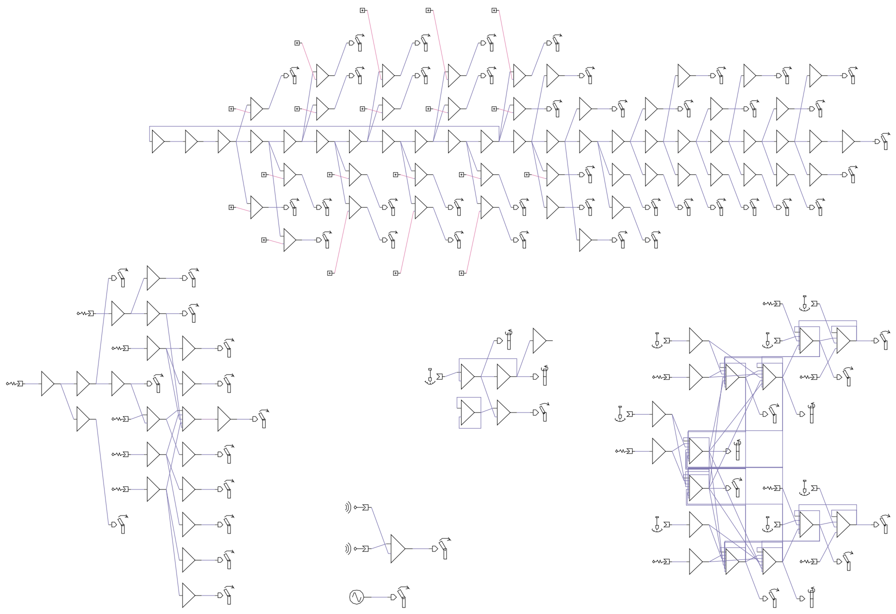


Fig. 5.8 Sample neural networks. Triangles are the standard signal-processing neurons ("N"). Receptors act as inputs (shown usually on the left side: gyroscope, touch, smell). Controlled muscles (rotating, bending) are usually on the right side. Some inputs are connected to the output of the sinus generator (⊘-) or the constant signal generator (⊞-). Note recurrent connections. Parallel connections are also allowed.

5.2.4 Receptors and Effectors

Receptors and effectors are interacting between body and brain. They must be connected to brain in order to be useful, but they also interact with creature's body and the world. The three basic Framsticks receptors (sensors) include "G" for orientation in space (equilibrium sense, gyroscope), "T" for detection of physical contact (touch), and "S" for detection of energy (smell); see Figs. 5.8 and 5.9.

The two basic Framsticks effectors are muscles: bending and rotating. Positive and negative changes of muscle control signal make adjacent sticks move in either direction, which is analogous to the natural systems of muscles, with flexors and extensors. The strength of a muscle determines its effective ability of movement and speed (acceleration). If energetic issues are considered in an experiment, then muscle strength affects energy consumption.

A sample framstick equipped with basic receptors and effectors is shown in Fig. 5.9. Other examples of receptors and effectors are energy level meter, water detector, vector eye, linear actuator, and thrust.

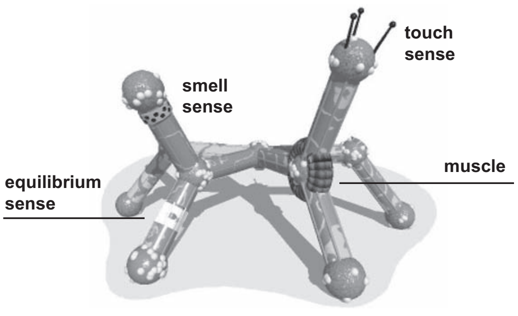


Fig. 5.9 Basic receptors (equilibrium, touch, smell) and effectors (muscles).

5.2.5 Communication

Since agents, their neurons, and the experiment logic are all script based, it is possible to implement virtually any kind of communication. Creatures, their components, and properties of these components are all accessible in script objects and can be modified by the experimenter according to their needs.

There are also dedicated objects and functions that facilitate implementation of the most common communication settings. The two basic communication objects are Channel and Signal.

- Channel objects are defined by unique names that represent either a physical medium (e.g., "light", "smell"), some abstract information ("danger", "goal"), or characterize a group of signal holders ("flock", "team_23").

- Signal objects store the value that is actually transmitted in a channel. The type of the value is arbitrary so that objects can be transmitted apart from simple values. For a signal, one can also set its power and flavor (which can be used to differentiate between multiple signals in a single channel). Signals belong to other objects – World, Creature, and Neuro – as members of their `signals` collections. World signals are stationary, while Creature signals and Neuro signals are carried by their owner.
- Functions used to receive signals can read the aggregated signal intensity from a channel (useful for physical quantities), find the strongest signal in a channel (taking into account location of transmitters and signal intensity), or enumerate all nearby signals.

Framsticks allows for visual presentation of signals in the world, which is useful both for debugging and as a part of the experiment visualization.

Communication: The Fireflies Example

In this example, Framsticks communication features are used to simulate light transmission, and two specialized neural units (the SeeLight receptor and the Light effector) are required. Agents equipped with these units are able to synchronize their flashing patterns only using local information (aggregated light intensity). See Fig. 5.10 and the Framsticks Theater show for a live demonstration.

A manually designed neural network that handles synchronization of flashing is shown in Fig. 5.11. The output of the light receptor contributes to the charging potential of neuron #6, effectively shortening the flashing cycle. Agents that flash “too late” receive most of the incoming light signal during their charging phase. This makes them charge faster and catch up with the other agents.

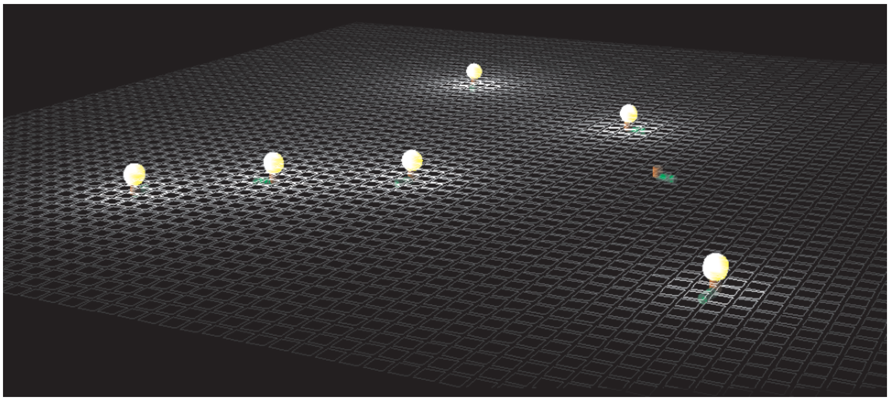


Fig. 5.10 The fireflies example and spatial intensity of the “light” signals.

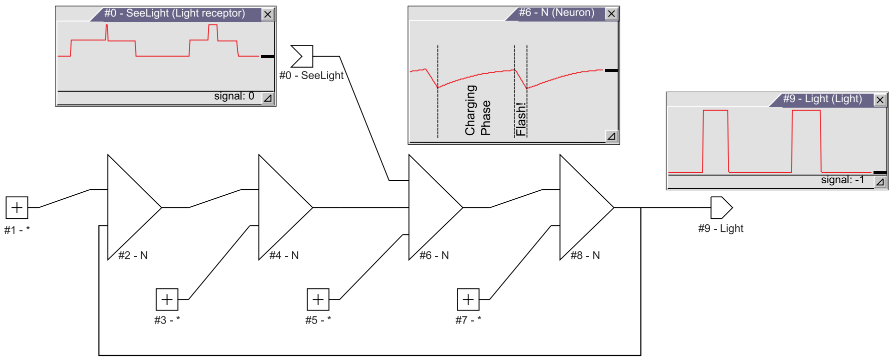


Fig. 5.11 A sample neural network that handles synchronization of flashing of a firefly.

The representation of this network in the *f1* genetic encoding (cf. Section 5.3) is as follows:

```
X [SeeLight,flavor:0] [*] [-1:2.26,6:-2,in:0.01,fo:0.01,si:1]
[*] [-2:1,-1:-0.5,si:9999,fo:1,in:0]
[*] [-2:2,-6:0.3,-1:-0.4,in:0.01,fo:0.01,si:1]
[*] [-2:1,-1:-0.5,si:9999,fo:1,in:0] [Light,-1:-1,flavor:0]
```

This genotype can be subject to evolution to obtain various flashing behaviors, depending on the fitness function used. The full source code for the the Light effector and SeeLight receptor is available in the “scripts” subdirectory of the Framsticks distribution (cf. Section 5.4 on scripting). In short, the Light effector, when created, registers one signal in the “light” channel:

```
Neuro.signals.add("light");
```

In each simulation step, this effector adjusts the power of this signal depending on its neural input value:

```
Neuro.signals[0].power=...;
```

The SeeLight receptor is very simple. In each step, it just sets its neural output to the amount of light perceived:

```
Neuro.state=Neuro.signals.receive("light");
```

Communication: The Boids Example

The implementation of boids [31] uses flexibility of the Framsticks communication to efficiently obtain the list of neighbor creatures. The data being communicated between agents are references to their own objects. The neighbor list is processed on each step by the creature handler to calculate aggregated direction of flight based on the motion of the neighbors. See Fig. 5.12 and the Framsticks Theater show for a live demonstration.

When an agent is created (the onBorn() function), it registers one signal in the “flock” channel and sets the signal value to reference its own body:

```
var signal=Creature.signals.add("flock");
signal.value=Creature.getMechPart(0);
```

To receive data, each agent in every simulation step receives a set of signals (i.e., references to neighbors within the specified range) and then can enumerate this array to perform necessary calculations:

```
var neighbors=creature.signals.receiveSet("flock",maxrange);
for(i=0;i<neighbors.size;i++) {...}
```

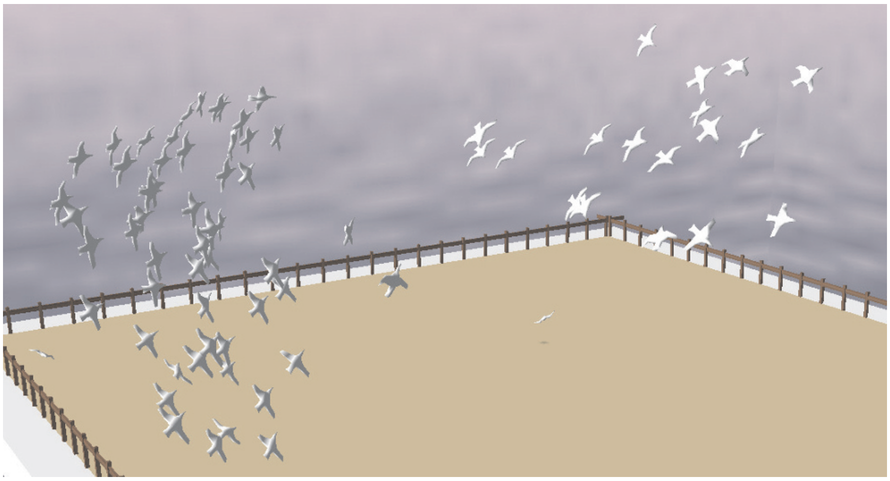


Fig. 5.12 The boids show in the Framsticks Theater. Creatures consist of one part only. They are assigned a visual style that uses a bird 3D model for display.

5.2.6 Environment

The world can be flat, built of smooth slopes, or built of blocks. It is possible to adjust the water level, so that not only walking/running/jumping creatures but also the swimming and amphibian ones are simulated. The boundaries of the virtual world can be one of three types:

- hard (surrounding wall: it is impossible to cross the boundary);
- wrap (crossing the boundary means teleportation to the other side of the world);
- none (the world is infinite).

These options are useful in various kinds of experiments and measurements of creature performance.

World heightfields are defined in a simple way, and a number of Framsticks heightfield generators exist. Real geographic information system (GIS) data from Earth or other planets can also be used: The PrettyMap conversion utilities can convert raster elevation and symbolic maps from most file formats into the Framsticks world heightfield format.

5.3 Genetics and Evolution

Framsticks supports multiple genetic encodings (also called representations or "genotype languages") [21]. The system manipulates and transforms genotype strings expressed in various representations and ultimately decodes them into the internal representation used by the simulator to construct a creature (phenotype). This means that one can describe a creature using genomes expressed in different "languages."

Any creature can be completely described using a low-level representation named *f0*, by listing all of the creature components and their properties. Other higher-level encodings convert their representations into the corresponding *f0* version (possibly through another intermediary representation), as shown in Fig. 5.13. The reverse mapping from lower-level into higher-level encodings is usually difficult or impossible to compute, which is also true in the biological realm. As a consequence, in the general case it is not possible to convert a lower-level representation into a higher-level one (or a higher-level one into another higher-level one).

Each encoding has its own genetic operators (mutation, crossover, and the optional repair) and a translation procedure that allows users to compute a phenotype from each genotype expressed in this encoding. A new encoding can be added relatively easily by implementing these components. Framsticks software is accompanied by the Genotype Development Kit (GDK)

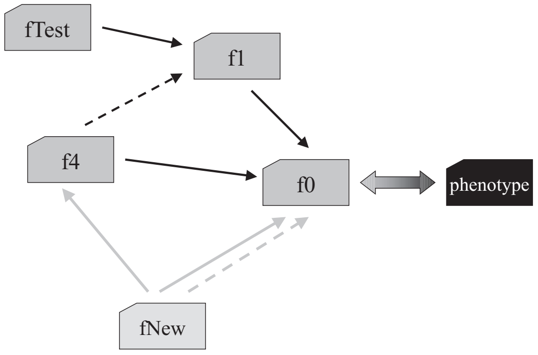


Fig. 5.13 A graph illustrating the idea of hierarchical genetic encodings (as nodes) and their translation procedures (as arcs). The dashed arrow depicts an approximate translation. Multiple, alternative translation methods may exist, as shown for the translation from *fNew* to *f0*.

which simplifies this process [37]. The most popular genetic encodings are characterized below.

The direct low-level, or *f0*, encoding describes agents exactly as they are represented in the model (Sect. 5.2.1). It does not use any higher-level features to make genotypes more compact or flexible. Its useful characteristics are that it has a minimal decoding cost and that every possible creature can be described using this encoding. Each *f0* genotype consists of a list of descriptions of all the elements a creature is composed of: parts, joints, neurons, and neural connections.

The recurrent direct encoding (labeled *f1*) also describes all the parts of the corresponding phenotype. Body properties are represented locally, so that most of the properties (and neural network connections) are maintained when a genotype section is moved to another place of the genotype. Control elements (neurons, receptors) are described near the elements under their control (muscles, sticks). Only tree-like body structures can be represented in *f1* (no cyclic bodies can be described, but arbitrary neural network topologies are possible). This concise encoding is relatively easy for humans to manipulate and manually design creatures by editing their genotypes. For example, the 'X' char means a stick, parentheses are used to branch body structure, 'r' and 'R' letters are used to rotate the branching plane, neurons are described in square brackets with their relative links and weights, and so forth.

The developmental encoding (*f4*) is development-oriented, similar to the encodings applied for evolving neural networks [6]. An interesting merit of developmental encoding is that it can incorporate symmetry and modularity, features commonly found in natural systems, yet difficult to formalize. *f4* seems similar to *f1*, but codes are interpreted as commands by cells (sticks, neurons, etc.). Cells can change their properties and divide. Each cell maintains its own pointer to the current command in the genetic code. After division, cells can execute different codes in parallel and differentiate themselves. The final body (phenotype) is the result of a development process. It starts with an undifferentiated ancestor cell and ends with a collection of interconnected differentiated cells (sticks, neurons, and connections).

Other available representations include similarity-based encoding, biological encoding (with a finite genetic alphabet and start/stop codons), chemical encoding (metabolic rules of growth), messy encoding (any sequence of symbols is valid), and a parametric Lindenmayer system encoding [9].

Each of the genetic encodings and the corresponding genetic operators have been carefully designed and tested, and each encoding was based on theoretical considerations and experimental tests. More detailed descriptions can be found in [18]. Examples of simple genotypes and corresponding phenotypes (creatures) are shown in Fig. 5.14.

The procedure of genotype translation may provide additional information regarding the relation of individual genes in the source and target encodings. If this information is available, then it is possible to track the relationship between parts of a genotype (genes) and parts of the corresponding creature

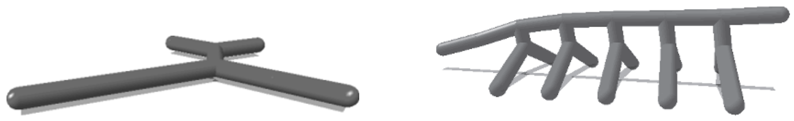


Fig. 5.14 Left: example of the *f1* genotype `XXX(XX,X(X,X))`. Right: example of the *f4* genotype with the repetition gene: `rr<X>#5<,<X>RR<<llX>LX>LX>>X`.

(phenes). Details of this process and examples are shown in [21]. Fig. 5.15 illustrates how this information can be visualized and used both ways.

In the Framsticks software, it is possible to select parts of the phenotype and genotype to get an instant visual feedback and understand their relationship; see Fig. 5.16. Users can move the cursor along the genotype to see which phenotype parts are influenced by the genotype character under the cursor. Another feature related to "genetic debugging" (a feature not yet available for biological genomes...) is the ability to modify a genotype by adding, deleting, or editing its parts while the corresponding phenotype is instantly computed and displayed. Framsticks can be used to illustrate the phenomena of polygeny and pleiotropy and to perform direct experiments with artificial genetic encodings, increasing comprehension of the genotype-to-phenotype translation process, and properties of genetic encodings – modularity, compression, redundancy, and many more.

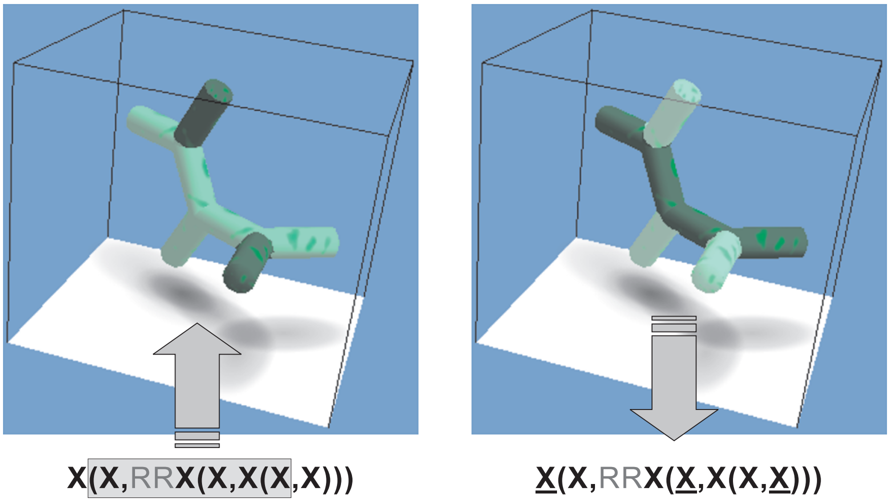
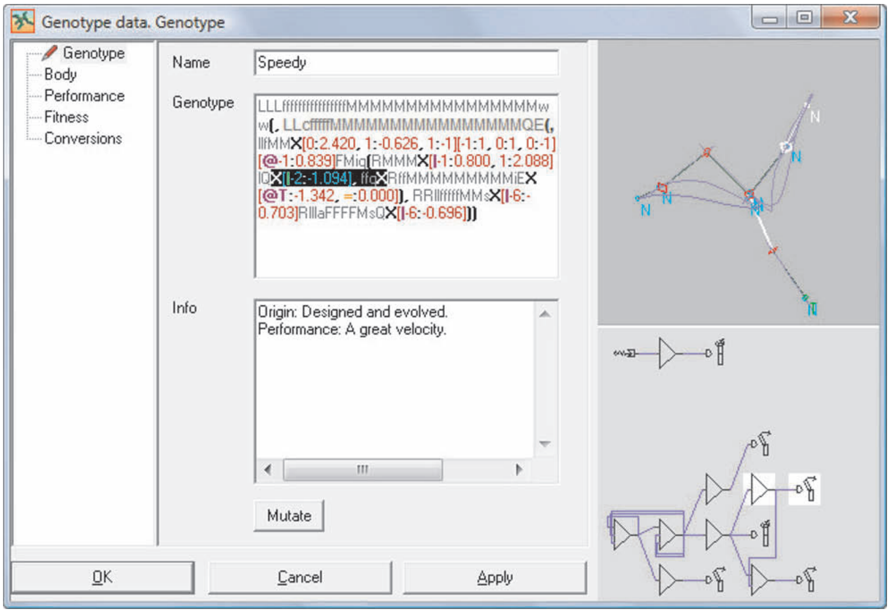


Fig. 5.15 A simple mapping between an *f1* genotype and the corresponding phenotype. Left: user selected a part of the genotype, corresponding phenes are highlighted. Right: user highlighted some parts of the body, corresponding genes are underlined.



Name: Speedy

Genotype:

```
LLLffffffffffffffffMMMMMMMMMMMMMMMMw
w(, LLcfffffMMMMMMMMMMMMMMMMMMQE(,
IIfMMX[0:2.420, 1:-0.626, 1:-1][-1:1, 0:1, 0:-1]
[@-1:0.839]FMiq(RMMMX[|-1:0.800, 1:2.088]
IQX[|-2:-1.094], ffqXRffMMMMMMMMMiEX
[@T:-1.342, =:0.000]), RRIIffffffMMsX[|-6:-
0.703]RIIIaFFFFMsQX[|-6:-0.696]))
```

Info:

Origin: Designed and evolved.
Performance: A great velocity.

Fig. 5.16 A sample genotype and the corresponding creature (body and brain). Some genes are selected by a user, and the corresponding parts of body and brain are highlighted.

5.4 Scripting

The Framsticks environment has its own virtual machine, thus it can interpret commands written in a simple language – *FramScript*. FramScript can be used for a range of tasks, from custom fitness functions, macros, and user-defined neurons, to user-defined experiment definitions, creature behaviors, events, and even 3D visualization styles. Understanding FramScript allows one to exploit the full potential of Framsticks, because scripts control the Framsticks system.

The FramScript syntax and semantics are very similar to JavaScript, JAVA, C, or PHP. In FramScript,

- All variables are untyped and declared using *var* or *global* statements.
- Functions are defined with the *function* statement.
- References can be used for existing objects.
- No structures and no pointers can be declared.
- There is the *Vector* object which handles dynamic arrays.
- FramScript code can access Framsticks object fields as "Object.field".

5.4.1 Creating Custom Script-Based Neurons

To demonstrate how scripting can be used, we will design a "noisy" neuron which generates occasional noise (random output). Otherwise it will pass the weighted sum of inputs into its output. In neuron definitions, inputs can be read, any internal function can be defined, output can be controlled directly, and internal states can be stored using "private" neuron properties. "Public" neural properties can be used to influence the neuron behavior; genetic operators (mutation, crossing over) will by default operate on public properties.

For a neuron, two functions can be defined: the initialization function (*init*) and the working function (*go*) which is executed in each simulation step. For our noisy neuron, we do not need the initialization function – there are no internal properties to initialize. However, the public "error rate" property will be useful to control how much noise is generated. For each neuron, we first have to define its name, long name, description, and the number of preferred inputs (any number in this case) and outputs (the noisy neuron provides one meaningful output signal):

```
class:
name:Nn
longname:Noisy neuron
description:Occasionally generates a random value
prefinputs:-1
prefoutput:1
```

The error rate property ("e") will be a floating-point number (f) within the range of 0.0 and 0.1:

```
prop:
id:e
name:Error rate
type:f 0.0 0.1
```

Finally, we implement the working function, which uses the `rnd01` function of the `Math` object to obtain a random value in the range from 0.0 to 1.0:

```
function go()
{
  if (Math.rnd01 < Fields.e)
    Neuro.state = Neuro.weightedInputSum;
  else
    Neuro.state = (Math.rnd01 * 2) - 1.0;
}
```

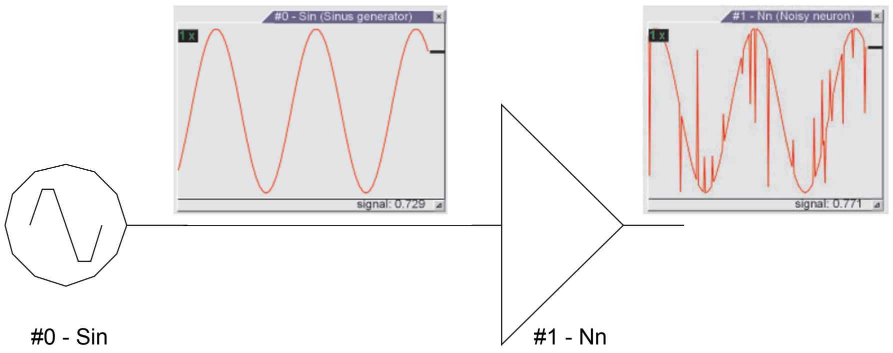


Fig. 5.17 The noisy neuron defined by the script, connected to the sinus generator. Random output values are generated with the rate of 0.1.

We join these three fragments into a single file, name it “noisy.neuro”, place it in the appropriate directory, run Framsticks, build a creature that uses the noisy neuron with the error rate set to 0.1, and start the simulation to see the result shown in Fig. 5.17. Now the FramScript source of the neuron can be easily modified to extend its functionality and to exhibit more complex behaviors. The noisy neuron is ready to be used in neural networks, and even in evolution, without any additional work.

5.4.2 Experiment Definitions

A very important feature of Framsticks is that you may define custom rules for the simulator. There are no predetermined laws, but there is a script called the *experiment definition*. It is analogous to the neuron definition explained in the previous section. The experiment definition script is more complex and defines behavior of the Framsticks system in a few related areas:

- Creation of objects in the world. The script defines where, when, and how much of which objects will be created. An object is an evolved organism, food particle, or any other element of the world designed by a researcher. Thus, depending on the script, food or obstacles might appear, move, and disappear, their location might depend on where creatures are, and so forth.
- Objects interactions. Object collision (contact) is an event, which may cause some action defined by the script developer. For example, physical contact may result in energy ingestion, pushing each other, exchange of information, destruction, or reproduction.
- Evolution. Any model of evolution or coevolution can be employed, including many gene pools and many populations (generally called groups);

independent evolutionary processes can be performed under different conditions.
- Evaluation criteria. These are flexible and do not have to be as simple as the basic performances supplied by the simulator. For example, fitness may depend on time or energy required to fulfill some task, or degree of success (distance from target, number of successful actions, etc.).

The script is built of "functions" assigned to system events, which include:

- `onExpDefLoad` – occurs after experiment definition has been loaded. This procedure should prepare the environment, create necessary gene pools and populations, etc.
- `onExpInit` – occurs at the beginning of the experiment.
- `onExpSave` – occurs on "save experiment data" request.
- `onExpLoad` – occurs on "load experiment data" request.
  The script should restore the system state saved by `onExpSave`.
- `onStep` – occurs in each simulation step.
- `onBorn` – occurs when a new organism is created in the world.
- `onKill` – occurs when a creature is removed from the world.
- `onUpdate` – occurs periodically, which is useful for efficient performance evaluation.
- `on[X]Collision` – occurs when an object of population [X] has touched some other object.

Therefore, researchers may define the behavior of the system by implementing appropriate actions within these events. A single script may introduce parameters which allows users to perform a number of diversified experiments using one experiment definition.

5.4.3 Illustrative Example ("Standard Experiment" Definition)

The file "standard.expdef" contains the source for the standard experiment definition script used to optimize creatures on a steady-state (one-at-a-time) basis, with fitness defined as a weighted sum of their performances. This script is quite versatile and complex. Below, its general concept is explained, with much simplified actions assigned to events. This digest gives an idea of what components constitute a complete experiment definition.

- `onExpDefLoad`:
  - create a single gene pool named "Genotypes"
  - create two populations: "Creatures" and "Food"

- `onExpInit`:
  - empty all gene pools and populations
  - place the initial genotype in “Genotypes”
- `onStep`:
  - if too little food: create a new object in “Food”
  - if too few organisms: select a parent from “Genotypes”; mutate, crossover, or copy it. Based on the resulting genotype, create a new individual in “Creatures”
- `onBorn`:
  - move the new object into a randomly chosen place in the world
  - set its starting energy depending on the object’s type (creature or food)
- `onKill`:
  - if “Creatures” object died, save its performance in “Genotypes” (possibly creating a new genotype). If there are too many genotypes in “Genotypes”, remove one.
- `onFoodCollision`:
  - send a piece of energy from the “Food” object to the colliding “Creature” object.

5.4.4 Popular Experiment Definitions

The most popular experiment definitions are outlined below. More specific experiments are referred to in Section 5.6.7.

- **standard** is used to perform a range of common evolutionary optimization experiments. It provides one gene pool, one population for individuals, one “population” for food, steady-state evolutionary algorithm, fitness as a weighted sum of performance values, support for custom fitness formulas, fitness scaling, roulette or tournament selection, and stagnation detection. It can log fitness and automatically produce charts using gnuplot [38].
- **generational** is a simple generational optimization experiment that resembles a popular “genetic algorithm” setup. It provides two gene pools (previous and current generation), one population for creatures, generational replacement of genotypes, roulette selection, and script-defined fitness formula.
- **reproduction** models asexual reproduction in the world. Each creature with a sufficient energy level produces an offspring, which is located to its parent. Food is created at a constant rate and placed randomly (Fig. 5.29).
- **neuroanalysis** simulates all loaded creatures one by one and computes the average and standard deviation of the output signal for each neuron

in each creature. After evaluation, a simple statistics report is printed. No evolution is performed.

- **boids** models emergent, flocking behavior of birds [31]. Users can activate or deactivate individual behavior rules (separation, alignment, cohesion) and instantly see results (Fig. 5.12).
- **evolution_demo** demonstrates the process of evolution. Individuals are placed in a circle. One individual is selected and then cloned, mutated, or crossed over. It is then evaluated in the middle of the circle and, depending on its fitness, may replace a poorer, existing individual or disappear.
- **learn_food** illustrates a social phenomenon of knowledge sharing in the context of exploration of the environment and exploitation of knowledge about the environment (Fig. 5.7). When an individual encounters food, it eats a bit, remembers its location, and gets a full amount of energy. The energy of each individual provides information on how current its information about food coordinates is. When agents collide, they learn from each other where the food was (e.g., by weighted averaging their knowledge). An individual that cannot find food for a long period of time loses all energy and dies, and a newborn one is created.

  It is interesting to see how knowledge sharing (cooperation, dependence) versus no sharing (self-sufficiency, risk) influences minimal, average, and maximal life span in various scenarios of food placement (e.g., neighboring and random). The dynamics in this experiment depends on the number of individuals, size and shape of their body (affects collisions and thus chances of sharing knowledge), world size, food eating rate, food placement, learning strategy, and the behavioral movement pattern.
- **mazes** evaluates (and evolves) creatures walking between two specified points in a maze.
- **deathmatch** is an educational tool intended for use in practical courses in evolutionary computing, evolutionary robotics, artificial life, and cognitive science. Following the "education by competition" approach, it implements a tournament among teams of creatures, as well as among teams of students. To win, a team has to provide (design or evolve) a creature that stays alive longer than creatures submitted by other teams. To survive, creatures need energy which can be collected by touching energy resources, winning fights, avoiding fights, or cooperation.
- **standard-eval** evaluates loaded genotypes thoroughly one by one and produces a report of fitness averages, standard deviations, and average evaluation times. No evolution is performed.
- **standard-log** logs genetic and evaluation operations, producing a detailed history of evolutionary process.
- **standard-tricks** serves as an example of a few advanced techniques: Random force can be applied to parts of a living creature during its life span, neural property values can be used in the fitness function, and statistical data can be acquired regarding movement of simulated body parts.

5.5 Advanced Features for Research and Education

Research works in the area of artificial life often concern studies of evolutionary and coevolutionary processes – their properties, dynamics and efficiency. Various methods and measures have been developed in order to analyze evolution, complexity, and interaction in complex adaptive systems (CAS). Another line of research focuses on artificial creatures themselves, regarding them as subjects of survey rather than “black boxes” with assigned fitness, and thus helping understand their behaviors.

Artificial life systems – especially those applied to evolutionary robotics and design [3, 4, 22] – are so complex that it is difficult or impossible to fully understand behaviors of artificial agents. The only way is to observe them carefully and use human intelligence to draw conclusions. Usually, behaviors of such agents are nondeterministic, and their control systems are sophisticated, often coupled with morphology and very strongly connected functionally [24]. Using artificial life techniques, humans are able to generate successful and efficient solutions to various problems, but they are unable to comprehend their internals and logic. This is because evolved structures are not designed by a human, not documented, and inherently complex.

Therefore, to effectively study behaviors and populations of individuals, one needs high-level, intelligent support tools [17]. It is not likely that automatic tools will soon be able to produce understandable, nontrivial explanations of sophisticated artificial agents. Nonetheless, their potential in helping researchers is huge. Even simple automatic support is of great relevance to a human; this becomes obvious after spending hours on investigating relatively simple artificial creatures.

One of the purposes of Framsticks is to allow creating and testing such tools and procedures and to develop methodology needed for their use. Realistic artificial life environments are the right place for such research. In the future, some of the advanced analysis methods developed within artificial life methodology may become useful for real-life studies, biology, and medicine.

In education, it is primarily important to make complex systems attractive and easier to understand. Visualization of relations between genotypes and phenotypes described in Section 5.3 is an example of such an educational instrument.

5.5.1 Brain Analysis and Simplification

An (artificial) creature is composed of body and brain. Bodies can be easily seen, and basic statistical information is easy to obtain (the number of parts, body size, weight, density, degree of consistency, etc.). Brains are much more difficult to present and comprehend. Framsticks provides a dedicated algorithm for laying out neural networks so that their structure can be exposed.

After it is applied, complex neural networks that initially looked chaotic have their brain structures revealed; see Fig. 5.8. Additionally, if a neuron that is embodied (located in body) is selected, it is highlighted in both the body view and the brain view (Fig. 5.16).

Signal flow charts are useful to understand how the brain works. Users can open multiple views of a single brain and connect probes to neurons, as shown in Figs. 5.17 and 5.18. It is also possible to enforce states of neurons using these probes so that parts of the brain can be turned off, oscillation can be stopped, or the desired signal shape can be interactively "drawn." In this way, muscles can be directly controlled while simulation is running.

Some sensors reflect states of the body. If the body is moved, output values of these neurons change (e.g., equilibrium or touch sensors) which is immediately seen in the neural probes. Fig. 5.18 shows a creature under such analysis.

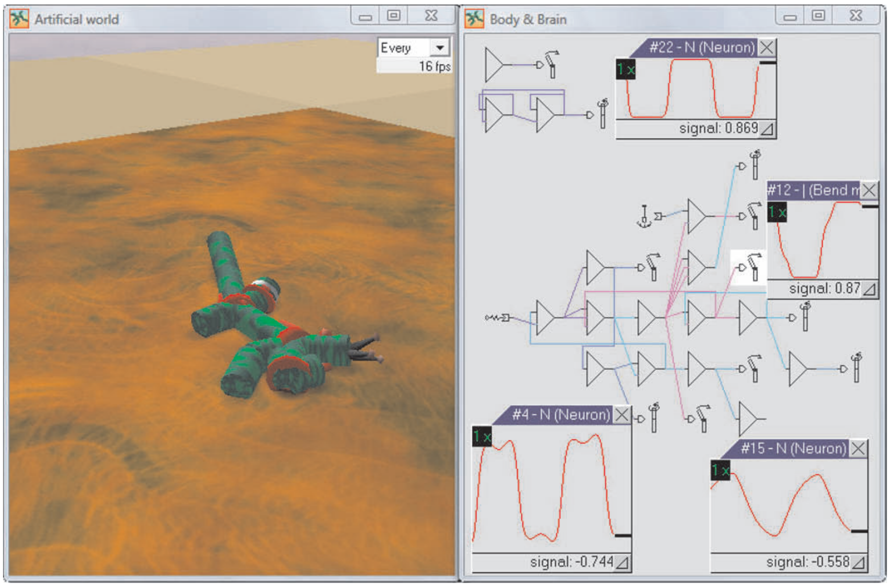


Fig. 5.18 A simulated creature with the control system under investigation. Four neural probes are attached showing signals in different locations of the neural network.

A simple automatic tool for brain analysis is the "neuroanalysis" experiment definition. During simulation, it observes each neuron in each creature and computes averages and standard deviations for all neural output signals. The final report summarizes the activity of brains and helps in the location of inactive and redundant brain areas. It also gives clues on possible ways of simplification of analyzed neural networks.

The “brain simplifier” macro is also available. It prunes neurons that do not directly or indirectly influence bodies, thus making evolved neural networks simpler and easier to apprehend.

5.5.2 Numerical Measure of Symmetry of Constructs

There is no agreement among scientists on why symmetry in biology is such a common evolutionary outcome, but this phenomenon must be certainly related to the properties of the physical world. According to one of the hypotheses [23], a bilaterally symmetrical body facilitates visual perception, as such a body is easier for the brain to recognize while in different orientations and positions. Another popular hypothesis suggests that symmetry evolved to help with mate selection. It was shown that females of some species prefer males with the most symmetrical sexual ornaments [29, 30]. For humans, there are proved positive correlations between facial symmetry and health [41] and between facial symmetry and perception of beauty [32].

It is hard to imagine the objective measure of symmetry. The only thing that can be assessed objectively is whether a construct is entirely symmetrical or not. The natural language is not sufficiently precise to express intermediate values of symmetry – we say that something is nearly symmetrical, but we are not able to the degree of symmetry numerically in the same manner as, for instance, angles can be described. This lack of expressions in natural languages describing partial symmetry is reasonable since many objects in the real world are symmetrical.

However, symmetry is not such a common concept in artificial worlds, and in order to study the phenomenon of symmetry and its implications, there was a need for defining a numerical, fully automated, and objective measure of symmetry for creatures living in artificial environments as well constructs, models, and 3D objects simulated in virtual settings. In Framsticks, the implementation of parametrized, numerical symmetry measure is provided in the Symmetry object; see [11] for a detailed description and Section 5.6.3 for a sample application.

5.5.3 Estimating Similarity of Creatures

Similarity is sometimes considered to be a basic or simple property. However, automatic measures of similarity can be extremely helpful! Similarity can be identified in many ways, including aspects of morphology (body), brain, size, function, behavior, performance, or fitness, but once a quantitative measure of similarity is established, it allows one to do the following:

- analyze structure of populations of individuals (e.g., diversity, convergence, etc.), facilitating better interpretation of experimental results,
- discover clusters in groups of organisms,
- reduce large sets or populations of creatures to small subsets of diverse representatives, thus reducing the complexity and size of experimental data and making it more comprehensible,
- infer dendrograms (and, hopefully, phylogenetic trees) based on morphological distances between organisms,
- restrict crossing over to only involve similar parents and thus avoid impaired offspring,
- introduce artificial niches, or species, by modifying fitness values in evolutionary optimization [5, 28],
- test correlation between similarity and quality of individuals, determine global convexity of the solution space [13], and develop efficient distance-preserving crossover operators [39, 27].

For the model of morphology considered in this chapter, a heuristic method was constructed that is able to estimate the degree of dissimilarity of two individuals. This method treats body as a graph (with parts as vertices and joints as edges) and then matches two body structures taking into account both body structure and body geometry. For a more detailed description of this method, see [15]; a sample application is presented in Section 5.6.4.

5.5.4 History of Evolution

In real life it is possible to trace genetic relationships within existing creatures, but we do not precisely know what happened during mutations and crossing overs of their genomes. Moreover, it is hard to trace all genetic relations over a longer timescale and in high numbers of individuals.

Framsticks as an artificial life environment allows one not only to retain all parent–child relationships but also to estimate genotype shares of related individuals (how many genes have mutated or have been exchanged). This lets users derive and render the real tree of evolution as shown in Fig. 5.19. The vertical axis is time, and the horizontal one reflects a degree of local genetic dissimilarity (between each pair of individuals). Vertices in this tree are single individuals. This way of visualizing evolution exposes milestones (genotypes with many descendants), and these can be automatically identified. The overall characteristics of the evolutionary process (convergence, high selective pressure, or random drift) can also be seen in such pictures.

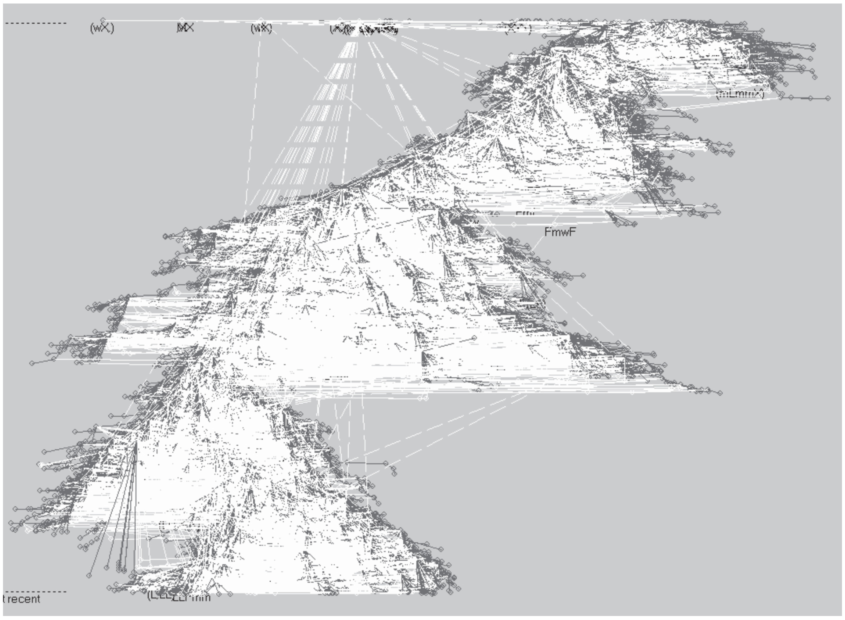


Fig. 5.19 The real tree of evolution; single ancestor and the beginning of time on top. Dark lines represent mutations, and white lines are crossovers.

5.5.5 Vector Eye

Vector Eye is a high-level sensor that provides a list of edges in the scene that are visible from some location in space. This information can be accurate – with no noise or imperfections that would exist if these edges were detected in a raster image.

Many simple sensors that are commonly used in robotics (touch, proximity, 3D orientation) provide either single-valued outputs or a constant rate of information flow (e.g., camera). Vector Eye provides a variable amount of information depending on the shape perceived and the relative position and orientation of the sensor with respect to the shape.

Vector Eye as a neuron uses multiple-valued output to send coordinates of perceived edges. This neuron is usually attached to some element of creature body, and a convex object to be watched is supplied as a parameter of this neuron; see the right part of Fig. 5.24 for illustration.

5.5.6 Fuzzy Control

Framsticks provides support for evolvable fuzzy control based on the Mamdani model: Both premises and conclusions of fuzzy rules are described by fuzzy variables [25]. The fuzzy system may be considered as a function of many variables, where input signals are first fuzzily evaluated by particular fuzzy rules, the outcomes of firing rules are then aggregated, and the resulting fuzzy set is defuzzified [33, 40].

Fuzzy sets in Framsticks are represented in a trapezoidal form, each set being defined by four real numbers within the normalized domain of $[-1, 1]$. The fuzzy rule-based control system is implemented in the Fuzzy neuron, with fuzzy sets and rules described as parameters of this neuron. Dedicated mutation and crossing-over operators have been developed; see [8, 7] for details and Section 5.6.6 for a sample application.

5.6 Research and Experiments

5.6.1 Comparison of Genotype Encodings

There are a number of studies on the evolution of simulated creatures that exhibit realistic physical behavior. In these systems, the use of a physical simulation layer implements a complex genotype–fitness relationship. Physical interactions between body parts, the coupling between control and physical body, and interactions during body development can all add a level of indirection between the genotype and its fitness. The complexity of the genotype–fitness relationship offers a potential for rich evolutionary dynamics.

The most important element of the genotype-to-fitness relationship is the genotype-to-phenotype mapping determined by the genotype encoding. There is no obvious simple way to encode a complex phenotype – which consists of a variable-size, structured body and the corresponding control system – into a simpler genotype. Moreover, performance of the evolutionary algorithm can greatly vary from encoding to encoding for reasons not immediately apparent. This makes genetic encodings and genetic operators a subject of intense research.

The flexibility of the genetic architecture in Framsticks allowed one to analyze and compare various genotype encodings in a single simulation environment. The performance of the three encodings described in detail in Section 5.3 was compared in three optimization tasks: maximization of passive height, active height, and velocity [18]. Solutions produced by evolutionary processes were examined and considered successful in these tasks for all encodings. However, there were some important differences in the degree of success. The *f0* encoding performed worse than the two higher-level encodings. The most important differences between these encodings are that *f0* has a minimal bias and is unrestrictive, while the higher-level encodings (*f1* and *f4*) restrict the search space and introduce a strong bias toward structured phenotypes. These results indicate that a more structured genotype encoding, with genetic operators working on a higher level, is beneficial in the evolution of 3D agents. The existence of a bias toward structured phenotypes can overcome the apparent limitation that entire regions of the search space are inaccessible for the optimization search. This bias may be useful in some applications (engineering and robotics, for example). The significant influence of encodings can be clearly seen in the obtained creatures: Those described by the *f0* encoding displayed neither order nor structure. The two encodings restricting morphology to a tree produced more clear constructs, with segmentation and modularity visible for developmental encoding (Fig. 5.20).

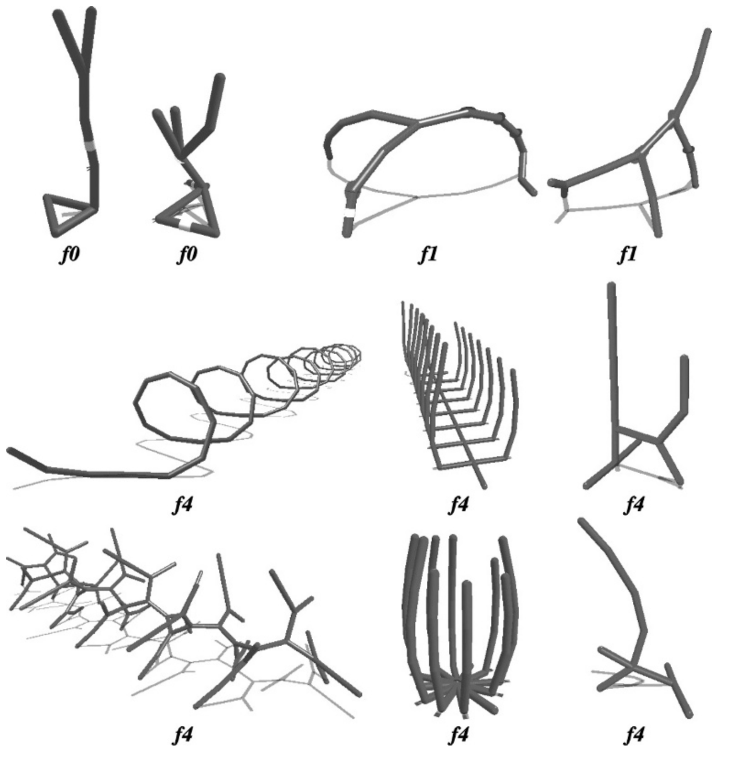


Fig. 5.20 Representative agents for three distinct genetic encodings and height maximization task.

5.6.2 Automatic Optimization Versus Human Design

Designing agents by hand is a very complex process. In professional applications, it requires extensive knowledge about the control system, sensors, actuators, mechanics, physical interactions, and the simulator employed. Designing neural control manually may be especially difficult and tedious (see for example [16]). For this reason, agents built by humans usually have lower fitness than agents generated by evolution. However, human creations are often interesting because of their explicit purpose, elegance, simplicity (a minimum of means), symmetry, and modularity. These properties are opposed to evolutionary outcomes, which are characterized by hidden purpose, complexity, implicit and very strong interdependencies between parts, as well as redundancy and randomness [18].

The difficult process of designing bodies and control systems manually can be circumvented by a hybrid solution: Bodies can be hand-constructed and control structures evolved for them. This popular approach can yield interesting creatures [14, 17, 1, 19], often resembling in behavior creatures found in nature [10].

5.6.3 Symmetry in Evolution and in Human Design

Following considerations from Section 5.5.2, the bilateral symmetry estimate allows one to compute the degree of symmetry for a construct (Fig. 5.21); it is therefore another automatic tool that helps a human examine and evaluate virtual and real creatures and designs [11].

It is also possible to rank creatures according to their symmetry value. The ranking shown in Fig. 5.22 presents 30 diversified constructs – small, big, symmetric, asymmetric, human-designed, and evolved. The horizontal axis shows values of symmetry. Creatures are oriented such that the plane of

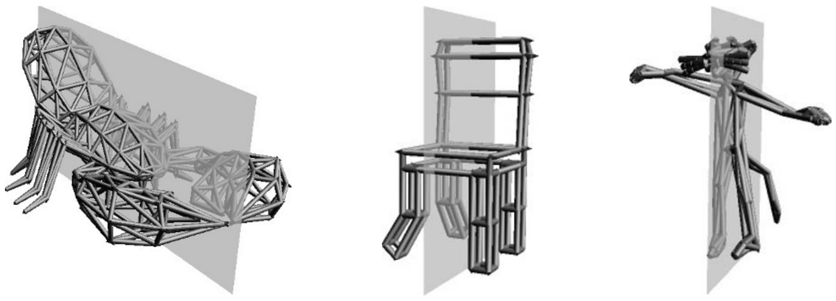


Fig. 5.21 Highest-symmetry planes for three sample constructs. Symmetry values are 1.0, 0.92, and 0.84.

symmetry for each of them is a vertical plane perpendicular to the horizontal axis. Constructs that were hand-designed and have regular shapes are the most symmetrical ones (located on the right side, with symmetry close to 1.0). On the other hand, large evolved bush-like creatures, for which symmetry planes were not obvious, are located on the far left (low values of symmetry).

Symmetry has long dominated in architecture and it is an unifying concept for all cultures of the world. Some famous examples include the Pantheon, Gothic churches, and the Sydney Opera House. A question comes up about the symmetry of human designs compared to the symmetry of evolved constructs. To investigate this issue, a set of 84 representative individuals has been examined. This set included evolved constructs originated from various evolutionary processes oriented toward speed and height and designed constructs that served various purposes – most often efficient locomotion, specific mechanical properties or aesthetic shape.

Analysis revealed that the vast majority (92%) of designed creatures appeared to be symmetrical or nearly symmetrical (symmetry higher than 0.9). Moreover, 82% of designed creatures were completely symmetrical – with the symmetry value 1.0. Clearly, human designers prefer symmetry, and there were no human designs in the set with symmetry less than 0.6.

Although half of the evolved creatures were highly symmetrical, symmetry of the rest is distributed fairly uniformly. It has to be noted that among evolved creatures with complete symmetry, many structures were very simple and symmetrical human designs were much more complex. Simple structures

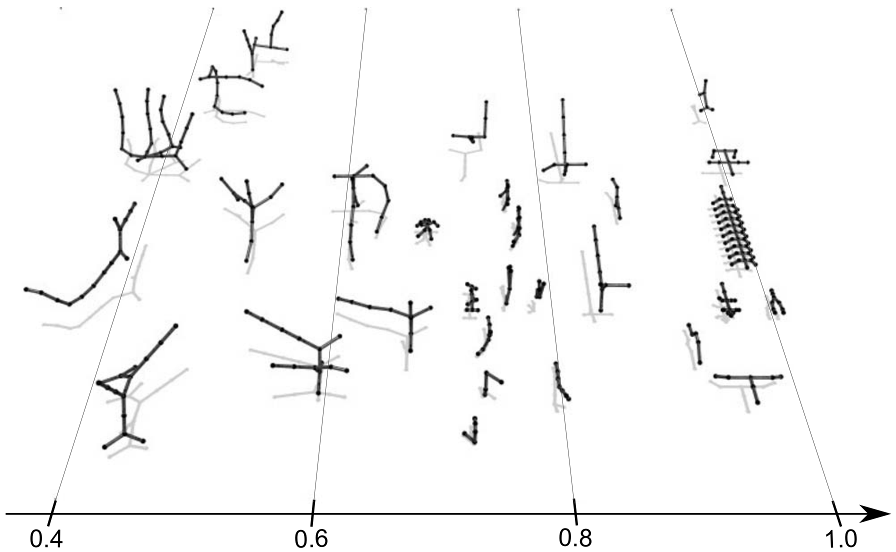


Fig. 5.22 30 diverse constructs arranged horizontally according to their values of symmetry (the most symmetrical on the right).

may exhibit symmetry by chance, while complex ones require some purpose or bias to be symmetrical.

5.6.4 Clustering with Similarity Measure

The similarity measure outlined in Section 5.5.3 allowed one to perform a number of experiments [15]; a sample clustering application is presented here. The UPGMA clustering method has been applied after computing dissimilarity for every pair of considered individuals. Fig. 5.23 shows the result of clustering of 10 individuals resulting from the experiments with maximization of body height. The clustering tree is accompanied by creature morphologies.

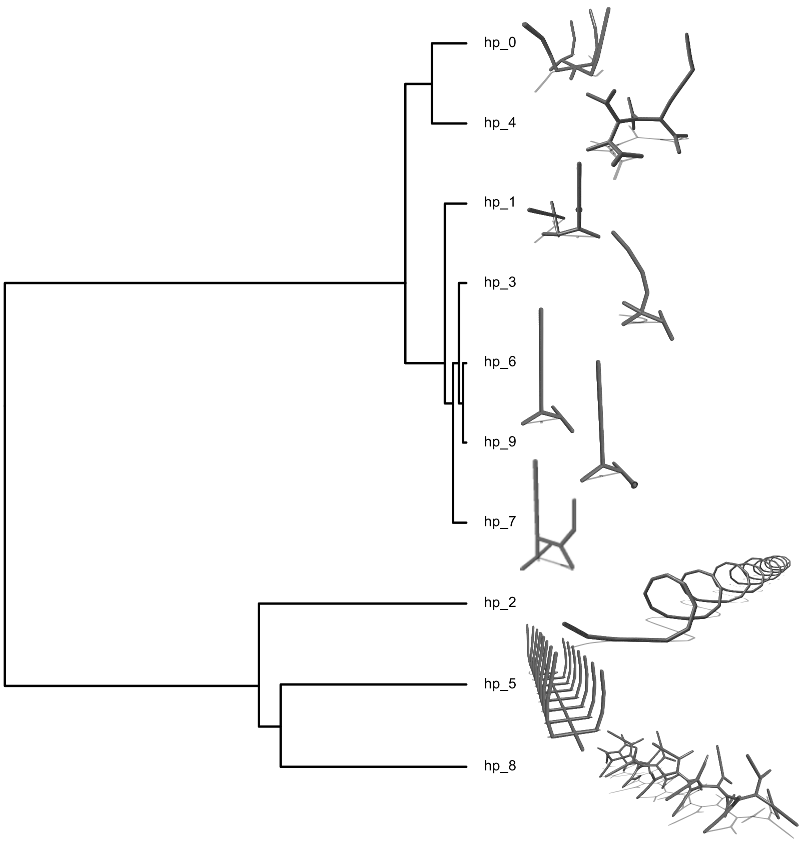


Fig. 5.23 The clustering tree for 10 best individuals in the height maximization task.

It can be seen that the three large organisms are grouped in a single, distinct cluster. They are similar in size but different in structure, so the distance in between them is high. Moreover, the measure also captured functional similarity (hp_1, 3, 6, 9, 7); all of these agents have a single stick upward and a similar base. The agents hp_0 and hp_4 are of medium size, but certainly closer to the group of small organisms than to the large ones. They are also similar in structure – this is why they constitute a separate cluster.

The similarity measure is very helpful for the study and analysis of groups of individuals. Manual work of classification of the agents shown in Fig. 5.23 yielded similar results, but it was a mundane and time-consuming process. It also lacked objectivism and accuracy, which are properties of the automatic procedure.

5.6.5 Bio-Inspired Visual Coordination

One of the factors that play an important role in the success of living organisms is the way they acquire information from the environment. Their senses are interfaces between neural systems and the outer world. Living organisms exhibit a vast number of sensor types, including olfactory, tactile, auditory, visual, electric, and magnetic ones. Among these, visual sensing provides a lot of information about the environment; it is therefore popular in natural systems and often used in artificial designs.

In the area of machine vision, considered problems usually concern object recognition and classification. The domain of artificial life adds the aspect of active exploration of the environment based on information that is perceived. The Vector Eye sensor described in Section 5.5.5 allows one to build and test models of complex cognitive systems, while Framsticks allows these systems to be embodied and situated in a virtual world. The purpose of building such biologically inspired cognitive models is twofold. First, they help understand cognitive processes in living organisms. Second, implementations of such models can cope with the complexity of real-world environments because these models are inspired by solutions that proved to be successful in nature.

A sample experiment [12] concerns a visual–motor model that facilitates stimulus–reaction behaviors, as it is the basic schema of functioning of living organisms. In this case the stimulus is visual, and the motor reaction is movement of an agent. This visual–motor system consists of three components: the Vector Eye sensor, the visual cortex, and the motor area. Vector data acquired by the sensor are transformed and aggregated by the visual cortex and fed to the motor area, which controls agent movements, as shown in Fig. 5.24. This biologically inspired visual–motor coordination model has been verified in a number of navigation experiments and perceived 3D shapes and proved to be flexible and appropriate for such purposes.

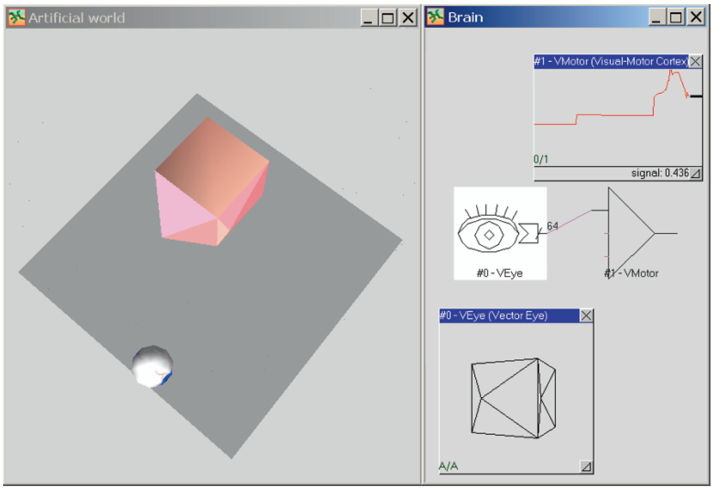


Fig. 5.24 Left: the agent equipped with the Vector Eye sensor circles around a 3D object. Right: Vector Eye perceives edges; this variable-rate information stream is processed by the visual cortex module and ultimately transformed into motor actions of the agent. Thus, agent behavior depends on what it perceives, which depends on its behavior.

5.6.6 Understanding Evolved Behaviors

Traditional neural networks with many neurons and connections are hard to understand. They are often considered "black boxes," successful but impossible to explain – and therefore not trustworthy for some applications. However, there is another popular paradigm used in control: the *fuzzy* control. It is employed in many domains of our life, including washing machines, video cameras, ABS in cars, and air conditioning. It is often applied for controlling nonlinear, fast-changing processes, where quick decisions are more important than exact ones [40]. Fuzzy control, just as neural networks, can cope with uncertainty of information. It is also attractive because of the following:

- It allows for linguistic variables (like "drive fast," where "fast" is a fuzzy term).
- It is easier to understand by humans. The fuzzy rule "if X is Big and Y is Small then Z is Medium" is much easier to follow than the crisp one "if X is between 32.22 and 43.32 and Y is less than 5.2 then Z is 19.2."

To evolve controllers whose behavior is explainable, fuzzy control has been developed in Framsticks (cf. Sect. 5.5.6). It has been applied in a number of artificial life evolutionary experiments; here, we outline a variant of the popular "inverted pendulum" problem [8, 7]. This experiment tested the efficiency of evolution in optimizing fuzzy control systems, verified if the evolved systems can really explain behaviors in a human-friendly way, and compared evolved fuzzy and neural control.

The base of the pendulum was composed of three joints (J_0, J_1, J_2) equipped with two actuators (bottom and top) working in two planes; see Fig. 5.25. The top part of the pendulum was composed of four perpendicular sticks, each having a single equilibrium sensor (G_0, G_1, G_2, G_3). The sensors provide signal values from the $[-1, 1]$ range depending on the spatial orientation of the joint in which they are located. Sensory information is further processed by the control system that controls actuators; this constitutes a loop of relations between the agent and the environment known from Section 5.6.5.

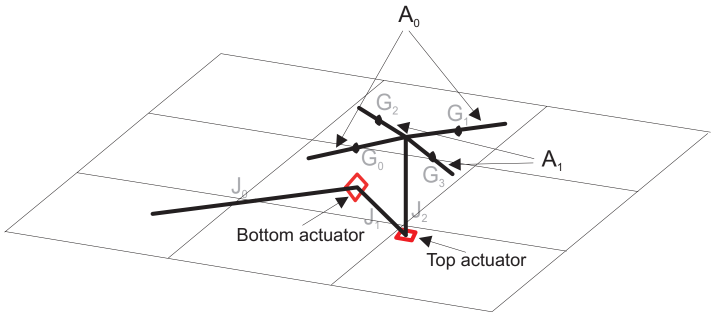


Fig. 5.25 The pendulum body structure (shown in a bent position).

The optimization task was to evolve a control system capable of keeping the head of the inverted pendulum from falling down for as long as possible. Evolved fuzzy systems were compared to evolved neural networks in the same experiment (cf. Fig. 5.26), and their performance was similar. Since evolved fuzzy systems were quite complex, they were simplified by performing an additional, short optimization phase with "add fuzzy set" and "add fuzzy rule" operators disabled. Modifications and deletions were allowed. Thus, the complexity of the control system was radically decreased without deteriorating its fitness.

Control systems considered in this experiment have four inputs and two outputs. Input signals s_0, s_1, s_2, and s_3 come from four equilibrium sensors. Based on their values, the fuzzy system sends two outputs signals for actuators: bend_bottom and bend_top. Linguistic variables for inputs (upright, leveled, and upside_down) and outputs characterizing bending directions (right, none, left) need to be defined to present the fuzzy system in a human-readable form. After they are introduced, the best evolved fuzzy system consisting of five rules can be rendered as follows:

1. **if** (s_2=leveled and s_0=leveled) **then** (bend_bottom=left and bend_top=left)
2. **if** (s_3=leveled and s_1=upside_down) **then** (bend_top=left)
3. **if** (s_1=upright) **then** (bend_bottom=left and bend_top=left)
4. **if** (s_3=upside_down) **then** (bend_bottom=right and bend_top=left)

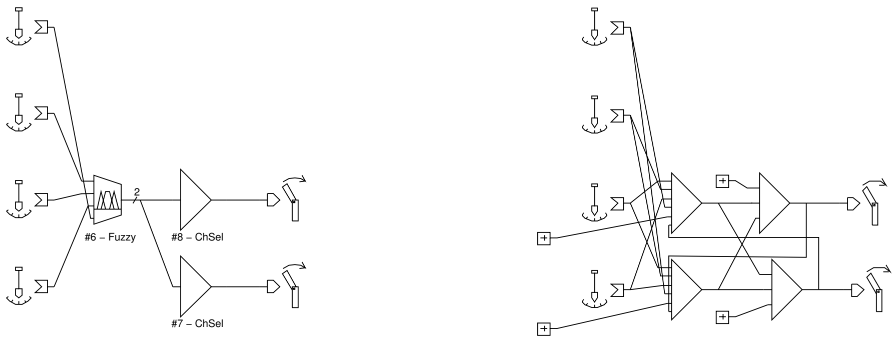


Fig. 5.26 Fuzzy and neural pendulum control. Left: a fuzzy system that provides two values in a single output (further separated by the Channel Selector neurons). Right: a sample neural network with two recurrent connections.

5. **if** (s_1=upside_down) **then** (bend_bottom=left and bend_top=none)

Although evolution of both neural and fuzzy controllers yielded similar pendulum behaviors in this experiment, careful analysis of the evolved fuzzy knowledge confirmed an additional, explanatory value of the fuzzy controller. The evolved fuzzy rules referenced to the pendulum structure and behavior are easily understandable by a human. This approach employed in evolving artificial life agents allows one to present evolved control rules in a human-readable form.

5.6.7 Other Experiments

The open architecture of Framsticks lets users define diverse genetic representations, experimental setups, interaction and communication patterns, and environments (see Sects. 5.4 and 5.4.2). Possible ideas include cooperative or competitive coevolution of species, predator–prey relationships, and multiple gene pools and populations. Sample experiment ideas related to biology include introducing geographical constraints and investigating differences in clusters obtained after a period of time, or studying two or more populations of highly different sizes. The latter, under geographical constraints, can be used to simulate and understand speciation.

A number of interesting experiments regarding evolutionary and neurocomputational bases of the emergence of complex cognition forms and a discussion about semantics of evolved neural networks, perception, and memory are presented in [26].

In the Virtual life lab at Utrecht University, Framsticks has been used to investigate evolutionary origins and emergence of behavior patterns. In

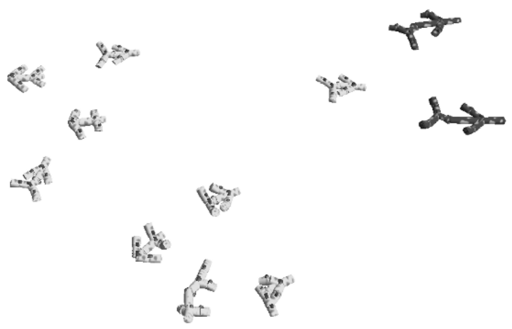


Fig. 5.27 A snapshot of the predator–prey simulation. The two dark creatures are predators, hunting down the lighter prey. Prey have evolved to smartly flee from their predators.

contrast to the standard evolutionary computation approach – with selection criteria imposed by the experimenter outside of the evolving system ("exogenous" or artificial selection) – in these studies selection emerges from within the system ("endogenous" or natural selection). Experiments are designed to include the problems of survival and reproduction in which creatures are born, survive (by eating food), reproduce (by colliding with potential mates), and die (if their energy level is insufficient). This approach enables the investigation of environmental conditions under which certain behaviors offer reproductive advantages, as in the following experiments:

- Coevolutionary processes in **predator–prey** systems are considered to result in arms races that promote *complexification*. For such complexification to emerge, the system must exhibit (semi)stable population dynamics. The primary goal of this experiment is to establish the conditions in which the simulated ecosystem is stable. Food, prey, and predator creatures are modeled in a small food chain and allowed to consume each other and reproduce (see Fig. 5.27). The resulting population dynamics are analyzed using an extended Lotka–Volterra model. When the relations between the parameters in the biological model and the simulation are established, stable conditions can be predicted which enables studies in long-term coevolutionary complexification [2].
- **Semiosis** is the establishment of connections between a sign and the signified via a situated interpretant. The segregation of the sign and the signified from the environment is not given a priori to agents: A sign becomes a part of an agent's subjective environment only if it offers benefits in terms of survival and/or reproduction. In this experiment, the evolutionary emergence of the relation between signs and signified is studied: Through natural selection, agents that have established behavioral relations between the signs and the signified are promoted. A sample of such a relation is movement toward the sign (*chemotaxis*). In varieties of this

experiment, agents can leave trails of signs (e.g., pheromones) in the environment or evolve the ability to signal to each other by using *symbols.*

- Usually, speciation occurs through geographical isolation, which disables gene flow and promotes genetic drift. Such speciation is called allopatric. This experiment reproduces another kind of speciation: **sympatric speciation** which happens in populations that live in the same geographical area.

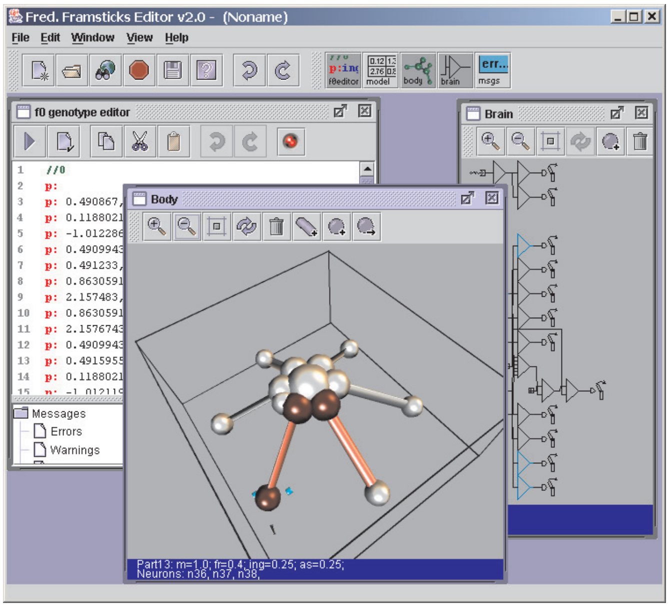
5.7 Education with Entertainment

Simulating evolution of three-dimensional agents is not a trivial task. On the other hand, 3D creatures are very attractive and appealing to both young and older users who spend much time enjoying the simulation. Many users wish to design their own creatures, simulate them, improve, and evaluate, but designing creatures is not very obvious when it takes place on the genetic level. To simplify this process, the Framsticks graphical editor (FRED) was developed: It helps in building creatures just as CAD programs support designing 3D models. The user-friendly graphical interface (shown in Fig. 5.28), drag-and-drop operations, and instant preview allow users to develop structures of their imagination. Designing neural networks lets users understand basic principles of control systems, their architecture, and their behavior. The editor can also browse and download existing genotypes from the Internet database.

Framsticks can be used to illustrate basic notions and phenomena like genes and genetics, mutation, evolution, user-driven evolution and artificial selection, walking and swimming, artificial life simulation, and virtual world interactions. However, the simulator has numerous options and parameters that make it complicated for the first-time users to handle. Thus, a predefined set of parameters and program behaviors was created primarily for the purposes of demonstration.

Since predefined simulation parameters shift users' focus from creating and modifying to observing what is happening in the virtual world, the demonstration program was named the *Framsticks Theater*. It is an easy-to-use application that includes a number of "shows," and new shows can be added by advanced users or developers using scripts (Sect. 5.4). The shows already included have their script source files available. Each show available in the Theater has a few major options to adjust (like the number of running creatures, length, and difficulty for the "Race" show) . The list of shows (see also Fig. 5.29) includes the following:

- Biomorph – illustrates user-driven (interactive, aesthetic) evolution. Users iteratively choose and double-click a creature to create eight mutated offspring.


Fred. Framsticks Editor v2.0 - (Noname)
File Edit Window View Help
f0editor model body brain msgs
f0 genotype editor
Body
Brain
Messages
Errors
Warnings

```
//0
p:
p: 0.490867,
p: 0.1188021
p: -1.012286
p: 0.4909943
p: 0.491233,
p: 0.8630591
p: 2.157483,
p: 0.8630591
p: 2.1576743
p: 0.4909943
p: 0.4915955
p: 0.1188021
p: -1.01211
```

```
Part13: m=1.0; fr=0.4; ing=0.25; as=0.25;
Neurons: n36, n37, n38,
```

Fig. 5.28 FRED: the graphical editor of the Framsticks creatures.

- Dance – effectors of all simulated creatures are forced to work synchronously.
- Evolution – shows evolutionary optimization with predefined fitness criteria, 50 genotypes in the gene pool, and tournament selection.
- Mixed world – no evolution takes place, creatures are just simulated in a mixed land-and-water environment.
- Mutation – presents a chain of subsequent mutants.
- Presentation – shows various walking and swimming methods of creatures evolved or constructed by the Framsticks users.
- Race – creatures compete in a terrain race running to the finish line.
- Reproduction – illustrates spontaneous evolution. Each creature with a sufficient energy level produces an offspring that appears near its parent. Food is created at a constant rate and placed randomly.
- Touch of life – creatures pass life from one to another by touching.
- Framsbots – the aim of this simple game is to run away from hostile creatures and make all of them hit one another.

The Framsticks Theater can be run on stand-alone workstations as a show (artistic installations, shops, fairs), as well as for education (e.g., in biology, evolution, optimization, simulation, robotics), illustration, attractive graphical background for music, advertisement, entertainment, or screen-saving mode.

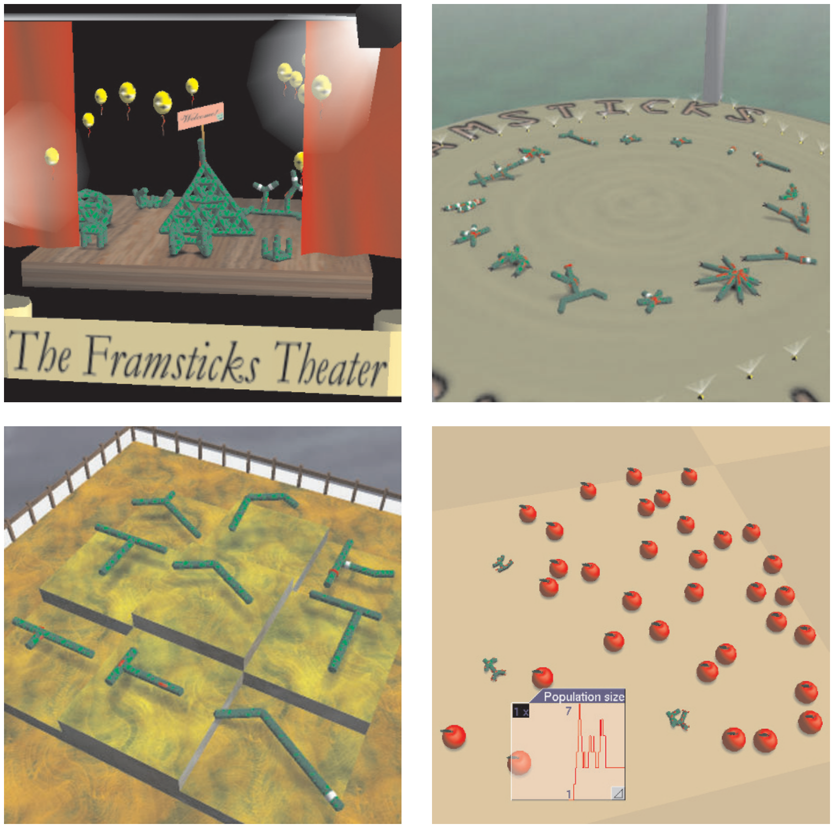


Fig. 5.29 Four Framsticks Theater shows: introduction, dance, biomorph, and reproduction.

5.8 Summary

This chapter presented Framsticks, a tool for modeling, simulation, optimization, and evolution of three-dimensional creatures. Sections 5.5, 5.6, and 5.7 demonstrated selected features and applications in education, research, and entertainment. Framsticks is developed with a vision of combining these three aspects, to make research and education attractive, playing for fun – educationally involving, and education – a demonstration and introduction to research. The software is used by cognitive scientists, biologists, roboticists, computer and other scientists and also by students and non-professionals of various ages.

Although the Framsticks system is versatile and complex, it can be simplified when some features are not needed. For example, control systems can be neglected if only static structures are of interest; genetic encoding may only allow for two-dimensional structures if 3D is not required; simulation or evolution can be restricted to a specific type of a neuron; local optimization techniques can be used if the problem at hand does not require evolutionary algorithms.

Complexity is useless when it cannot be understood or applied. This is why Framsticks software tries to present information in a human-friendly and clear way, encouraging development of automated analysis tools and helping to understand the phenomena of life and nature.

Acknowledgements This work has been supported by the Ministry of Science and Higher Education, grant No. N N519 3505 33.

References

1. Adamatzky, A.: Software review: Framsticks. Kybernetes: The International Journal of Systems & Cybernetics **29**(9/10), 1344–1351 (2000)
2. de Back, W., Wiering, M., de Jong, E.: Red Queen dynamics in a predator-prey ecosystem. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation pp. 381–382 (2006)
3. Bentley, P.: Evolutionary Design by Computers. Morgan Kaufmann (1999)
4. Funes, P., Pollack, J.B.: Evolutionary body building: adaptive physical designs for robots. Artificial Life **4**(4), 337–357 (1998)
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA (1989)
6. Gruau, F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding for genetic neural networks. In: J.R. Koza, D.E. Goldberg, D.B. Fogel, R.R. Riolo (eds.) Proceedings of the First Annual Conference, Genetic Programming 1996, pp. 81–89. MIT Press, Cambridge, MA (1996)
7. Hapke, M., Komosinski, M.: Evolutionary design of interpretable fuzzy controllers. Foundations of Computing and Decision Sciences **33**(4), 351–367 (2008)
8. Hapke, M., Komosinski, M., Waclawski, D.: Application of evolutionarily optimized fuzzy controllers for virtual robots. In: Proceedings of the 7th Joint Conference on Information Sciences, pp. 1605–1608. Association for Intelligent Machinery, North Carolina, USA (2003)
9. Hornby, G., Pollack, J.: The advantages of generative grammatical encodings for physical design. Proceedings of the 2001 Congress on Evolutionary Computation CEC2001 pp. 600–607 (2001)
10. Ijspeert, A.J.: A 3-D biomechanical model of the salamander. In: J.C. Heudin (ed.) Proceedings of 2nd International Conference on Virtual Worlds (VW2000), Lecture Notes in Artificial Intelligence No. 1834, pp. 225–234. Springer-Verlag, Berlin (2000)
11. Jaskowski, W., Komosinski, M.: The numerical measure of symmetry for 3D stick creatures. Artificial Life Journal **14**(4), 425–443 (2008)
12. Jelonek, J., Komosinski, M.: Biologically-inspired visual-motor coordination model in a navigation problem. In: B. Gabrys, R. Howlett, L. Jain (eds.) Knowledge-Based Intelligent Information and Engineering Systems. Lecture Notes in Computer Science 4253, pp. 341–348. Springer-Verlag, Berlin (2006)
13. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: L.J. Eshelman (ed.) Proceedings of the 6th International Conference on Genetic Algorithms, pp. 184–192. Morgan Kaufmann (1995)
14. Komosinski, M.: The world of Framsticks: Simulation, evolution, interaction. In: J.C. Heudin (ed.) Virtual Worlds. Lecture Notes in Artificial Intelligence No. 1834, pp. 214–224. Springer-Verlag, Berlin (2000)

15. Komosinski, M., Koczyk, G., Kubiak, M.: On estimating similarity of artificial and real organisms. Theory in Biosciences **120**(3-4), 271–286 (2001)
16. Komosinski, M., Polak, J.: Evolving free-form stick ski jumpers and their neural control systems. In: Proceedings of the National Conference on Evolutionary Computation and Global Optimization. Poland (2009)
17. Komosinski, M., Rotaru-Varga, A.: From directed to open-ended evolution in a complex simulation model. In: M.A. Bedau, J.S. McCaskill, N.H. Packard, S. Rasmussen (eds.) Artificial Life VII, pp. 293–299. MIT Press (2000)
18. Komosinski, M., Rotaru-Varga, A.: Comparison of different genotype encodings for simulated 3D agents. Artificial Life Journal **7**(4), 395–418 (2001)
19. Komosinski, M., Ulatowski, S.: Framsticks web site, `http://www.framsticks.com`
20. Komosinski, M., Ulatowski, S.: The Framsticks Tutorial. `http://www.framsticks.com/common/tutorial`
21. Komosinski, M., Ulatowski, S.: Genetic mappings in artificial genomes. Theory in Biosciences **123**(2), 125–137 (2004)
22. Lipson, H., Pollack, J.B.: Automatic design and manufacture of robotic lifeforms. Nature **406**(6799), 974–978 (2000)
23. Livio, M.: The Equation That Couldn't Be Solved. How Mathematical Genius Discovered the Language of Symmetry. Simon & Schuster, New York (2005)
24. Lund, H.H., Hallam, J., Lee, W.P.: Evolving robot morphology. In: Proceedings of IEEE 4th International Conference on Evolutionary Computation. IEEE Press, Piscataway, NJ (1997)
25. Mamdani, E.H.: Advances in the linguistic synthesis of fuzzy controllers. International Journal of Man-Machine Studies **8**(6), 669–678 (1976)
26. Mandik, P.: Synthetic neuroethology. Metaphilosophy, Special Issue on Cyberphilosophy: The Intersection of Philosophy and Computing **33**(1/2) (2002)
27. Merz, P.: Advanced fitness landscape analysis and the performance of memetic algorithms. Evolutionary Computation **12**(3), 303–325 (2004). URL `http://dx.doi.org/10.1162/1063656041774956`
28. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, New York (1996)
29. Moller, A.: Fluctuating asymmetry in male sexual ornaments may reliably reveal male quality. Animal Behaviour **40**, 1185–1187 (1990)
30. Moller, A.: Female swallow preference for symmetrical male sexual ornaments. Nature **357**, 238–240 (1992)
31. Reynolds, C.W.: Flocks, herds and schools: A distributed behavioral model. In: SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 25–34. ACM Press, New York (1987). DOI http://doi.acm.org/10.1145/37401.37406
32. Rhodes, G., Proffitt, F., Grady, J., Sumich, A.: Facial symmetry and the perception of beauty. Psychonomic Bulletin and Review **5**, 659–669 (1998)
33. Ross, T.J.: Fuzzy Logic with Engineering Applications. Wiley, New York (2004)
34. Sims, K.: Evolving 3D morphology and behavior by competition. In: R.A. Brooks, P. Maes (eds.) Proceedings of the 4th International Conference on Artificial Life, pp. 28–39. MIT Press, Boston, MA (1994)
35. Smith, R.: Open Dynamics Engine. `http://www.ode.org/` (2007)
36. Taylor, T., Massey, C.: Recent developments in the evolution of morphologies and controllers for physically simulated creatures. Artificial Life **7**(1), 77–88 (2001)
37. Ulatowski, S.: Framsticks GDK (Genotype Development Kit), `http://www.framsticks.com/dev/gdk/main`
38. Williams, T., Kelley, C.: gnuplot: a plotting utility. `http://www.gnuplot.info/`
39. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**, 67–82 (1997)
40. Yager, R., Filev, D.: Foundations of Fuzzy Control. Wiley, New York (1994)
41. Zaidel, D., Aarde, S., Baig, K.: Appearance of symmetry, beauty, and health in human faces. Brain and Cognition **57(3)**, 261–263 (2005)