SIMPLE ARTIFICAL LIFE FRAMEWORK

Jan Klusáček
Doctoral Degree Programme (2) , FEEC BUT
E-mail: xklusa00@stud.feec.vutbr.cz

Supervised by: Petr Honzík
E-mail: honzikp@feec.vutbr.cz

Abstract: Artificial life (AL) is a field of study dealing with computer models simulating life and its evolution. This article describes a simple framework created to work with an artificial life. Created framework works with simulation of behavior of single cell organism. In addition, it also simulates basic physics. Results of simulations are accessible through a web interface which allows observing the environment and details of each organism. Simple demo of realised program is aviable at address: http://klusacek.tk/~honza/sim/view.php?sim=sim.

Keywords: Artificial life, Python, Grammatical evolution, Web interface

1 INTRODUCTION

Artificial life is a very broad area covering systems simulating some properties of life. Artificial life is mostly studied using computer simulations, but there are also projects using physical hardware[1].

Many AL simulations are similar to evolutionary algorithms, particularly to genetic algorithms (GA). AL unlike GA doesn't use any explicit fitness function. Organism's ability to reproduce isn't determined by some abstract fitness function, but its direct result of individual's properties and behavior. There are many different variants of AL. Some of these variants will by described further.

1.1 EVOLVING PROGRAMS

This is probably the largest group of AL. In this case, each individual is represented by a single program. The objective of these programs is survival (being executed) and reproduction. There are many variants of simulators differing in the used language and mechanism of creating programs (individuals).

One of these simulators is Avida[2]. It simulates a virtual machine consisting of CPU, memory and operating system. All of the programs (individuals) reside in one memory space. They compete for CPU time and memory. These programs can reproduce (copy its code to another place in memory and execute it), mutate (copy itself with error) and crossover (when two programs trying to copy themselves to overlapping locations). Experiments show emergence of some interesting phenomena known from real life, for example parasitism. Some of the newly evolved programs call parts of other programs. Other programs replace concurrent programs with their own code, effectively stealing their CPU time.

Another example of evolving programs is DigiHive[3]. This simulation works with two dimensional environment populated by particles. Each particle contains a small part of a program in a declarative language. These particles can freely move and when multiple particles collide, they can join and create more complicated programs.

1.2 NEURAL NETWORK

Some simulations derive behavior of their individuals from a neural network instead of a program. One example of usage of neural network is Critterding simulator. Individuals in this simulation are defined by their body and neural network controlling it. The goal of each individual is to evolve its body and neural network to allow it to move and collect food effectively. There is no explicit fitness function, instead individual which is able to collect a specified amount of energy can reproduce.

Another example of AL simulation using neural network is OpenWorm project. This project aims to create complete simulation of one worm (Caenorhabditis elegans)[4]. It aims to simulate complete copy of its neural network (302 neurons).

2 ARTIFICAL LIFE FRAMEWORK

This paper describes a simple AL framework which was implemented and tested. This framework is designed to simulate simple single cell organisms and their interaction with the environment. The main purpose of this framework is to simplify the process of testing different algorithms simulating the organism's behavior. The whole framework is divided into two parts. One part is dealing with simulation and other provides an interface for presenting results to user in the human readable form.

3 SIMULATION

The first part of designed framework is simulation. It's divided into three modules. First part deals with simulation of an environment. Second part deals with simulation of organism: its movement, senses, energy consumption, etc. Last part simulates behavior of organisms.

3.1 ENVIROMENT

Simulated environment is represented by a grid of pixels. Each pixel contains information about amount of chemicals. Currently, the simulation works with two chemicals: food and waste products. There are also methods to simulate phenomena as diffusion and decay of chemicals, methods to create sources of chemicals and methods providing statistics.

3.2 ORGANISM

This module simulates processes related to physical organisms. It simulates its external functions as turning, movement (with inertia), food intake, sensory functions and internal functions as energy consumption and production of waste.

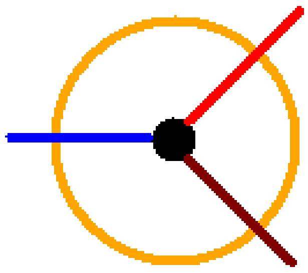


Figure 1: Schema of organism.

The simple schema of simulated is on the Figure 1. There is a circle in the center of each organism. The color of this circle gradually changes from black to white, depending on energy stockpile. Every organism need energy to survive and it dies when its energy reserve drop to zero. Energy is spent when the organism accelerates, it's necessary for the organism to reproduce and it's slowly spent even when the organism is still. The organism can obtain energy by eating food which is located on the same pixel.

Each organism has three sensors enabling it to sense the concentration of chemicals in three places relative to its coordinates. First is placed in center of organism, other two are placed at s distance of 1 pixel 45°on each side of the longitudinal axis. These sensors are shown as red lines on the schema. The brightness of these lines depends on sensed value.

The blue line on schema shows orientation of organism. The brightness of this line shows a current acceleration of the organism.

3.3 Behavior

Last module in created program controls behavior of each organism. This module uses inputs from sensors (currently only food, waste and energy stockpile) and control outputs (turning and acceleration). This module is implemented as an abstract class intended to be used as a base class.

To test the function of a whole program, one example behavior module was implemented. This module uses grammatical evolution to control outputs.

3.4 Grammatical evolution

Grammatical evolution (GE) is evolutionary algorithm which search to space of programs. It's similar to Genetic programming (GP). Both algorithms work with programs represented by tree structure and both algorithms use genetic algorithms. The main difference between these algorithms is the way they handle genetic operators (crossover and mutation). GP work directly with tree structure and can change selected node to another (mutation) or swap branches of two different trees (crossover). This means that after each modification of the program, it has to be checked if it has valid syntax.

GE keeps separate genotype and phenotype. Genotype is represented by integer string and it's mapped to phenotype using grammar in Backus-Naur form (BNF). Genetic operators are applied to integer string, which is then converted back to tree structure using grammar. This ensures that the new program is valid. On the other hand, this approach cause lower locality of method (change in one gene can change the meaning of all following genes). The process of mapping integer string to expression is on Figure 2

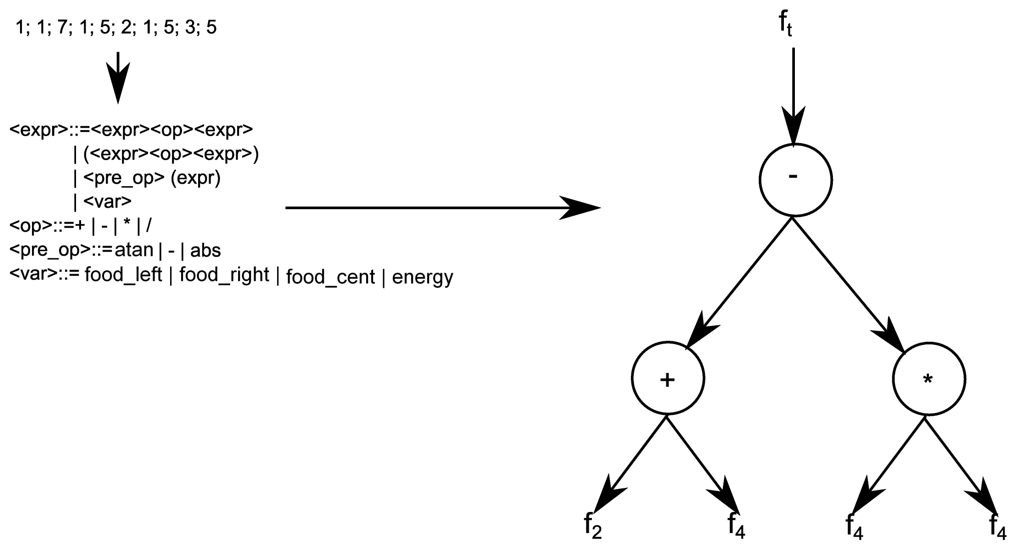
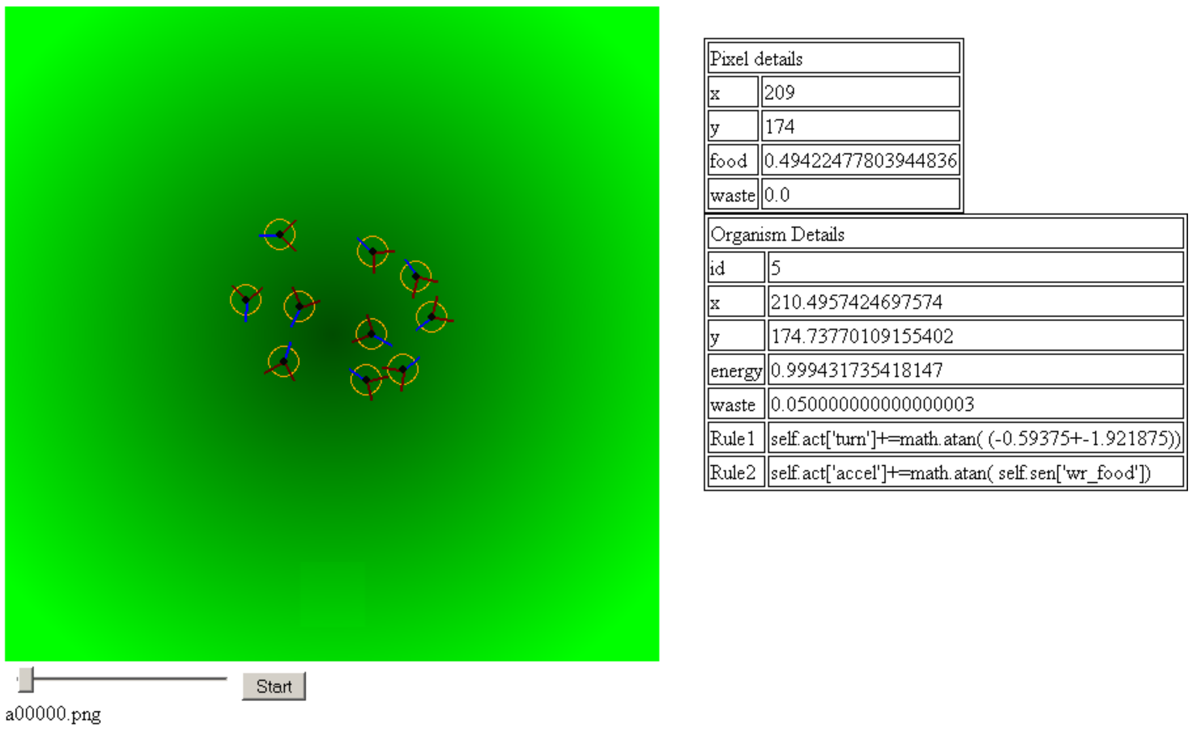


Figure 2: Example of mapping integer string to tree structure

4 WEB INTERFACE

The first part of a framework deal with simulation and it save its results in a file. But it's necessary to have some interface to show these results in human readable form. This was implemented as web interface. The web interface was selected because it allows easily check the progress of simulation on remote server and it is multiplatform. On server side PHP and Python scripts are used on server side. PHP was used to create most of the web interface. Python scripts are used only to extract information from saved results and they share parts of code with simulation. On the client side HTML and JavaScript is used. New pictures and requested details are loaded using Ajax to guarantee minimal latency.



Pixel details	
x	209
y	174
food	0.49422477803944836
waste	0.0

Organism Details	
id	5
x	210.4957424697574
y	174.73770109155402
energy	0.999431735418147
waste	0.050000000000000003
Rule1	self.act['turn']+=math.atan((-0.59375+-1.921875))
Rule2	self.act['accel']+=math.atan(self.sen['wr_food'])

Figure 3: Example of Web interface

Example of web interface is on figure 3. The scroll bar on the bottom of the page can be used to select step of the simulation. The step can be selected either manually (moving scroll bar directly) or as the animation. Animation can be started using Start button which can be used also used to pause running animation. Picture of selected step is displayed above the scroll bar. This picture shows the distribution of food (green) and waste (red). It also shows individual organisms using schema described in chapter 3.2. The user can click anywhere in this picture to get detailed information. After each click, python script is run on server side loading saved data from selected step. This information is then displayed in tables on the right side. One table shows information about selected pixel. Second table contains information about nearest organism.

5 CONCLUSION

New framework described in this paper enables easy testing of the different behaviors of single cell organisms. Simulator saves the complete state of simulation each step. This allows to review any step of the simulation after it is already finished. The interface is realized as a web interface. This allows easy access to results even if the simulation is running on the remote computer.

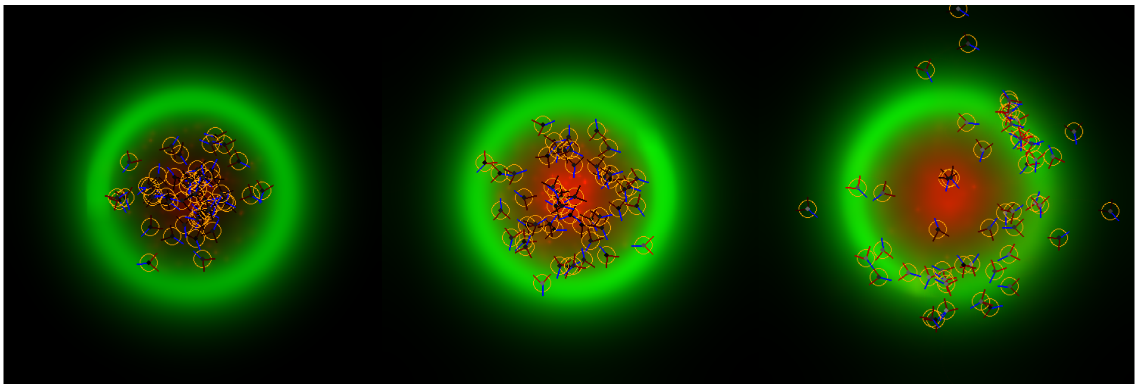


Figure 4: Evolution of GE organism.

One example organism was designed and implemented in addition to basic framework. This organism is using grammatical evolution to control its behavior. This organism was used to test the functions of the framework. This test used a simple environment with little food in the center and abundance on edges. All new organisms are spawned in the center, meaning that they have to find place with higher concentration of food.

The example of the evolutionary process of GE organism is on Figure 4. At the beginning of the simulation, there are organisms with random behavior spawned in center of map without food. These organisms keep dying, because they are not able to locate food and new random organisms are spawned to replace them. Eventually some organism which is able to locate food source appear. This organism quickly finds the location with an abundance of food and starts to multiply quickly. After a short time, most of the population is able to locate food, and its move from the center of the map.

Acknowledgement: This work was supported by grant „Research of Modern Methods and Approaches in Automation“ from the Internal Grant Agency of Brno University of Technology (grant No. FEKT-S-14-2429).

REFERENCES

[1] Hitoshi Hemmi, Jun'ichi Mizoguchi, and Katsunori Shimohara. Development and evolution of hardware behaviors. In *Towards Evolvable Hardware*, pages 250–265. Springer, 1996.

[2] Chris Adami and C Titus Brown. Evolutionary learning in the 2d artificial life system ‘avida’. In *Artificial life IV*, volume 1194, pages 377–381. Cambridge, MA: MIT Press, 1994.

[3] Rafa Sienkiewicz and Wojciech Jêdruch. Artificial environment for simulation of emergent behaviour. In B. Bieliczyñski et al, editor, *Adaptive and Natural Computing Algorithms: 8th International Conference, ICANNGA 2007, Warsaw, Poland, April 11-14, 2007, Proceedings, Part I*, volume 4431/2007 of *LNCS*, pages 386–393. Springer, 2007.

[4] Tim Busbice, Padraig Gleeson, Sergey Khayrulin, Matteo Cantarelli, Alexander Dibert, Giovanni Idili, Andrey Palyanov, and Stephen Larson. The neuroml c. elegans connectome. *Frontiers in Neuroinformatics*, (17).