

ARENA and WOXBOT: Some Steps Towards the Animation of Cognitive Characters Acting in a Virtual World

Fábio Roberto Miranda

João Eduardo Kögler Junior

SENAC College of Computer Science & Technology

Rua Galvão Bueno, 430 São Paulo SP Brasil

{fabio.rmiranda,kogler}@cei.sp.senac.br

Márcio Lobo Netto

Emílio Del Moral Hernandez

Polytechnic School – University of São Paulo

Avenida Prof. Luciano Gualberto, Travessa 3, 158, São Paulo SP Brasil

{lobonett,emilio}@lsi.usp.br

Abstract

This paper reports some of the results of the WOXBOT / ARENA project on artificial life. This project to build virtual worlds was set initially aimed at the graphic simulation of an arena where WoxBots, small mobile robots, can perform requested tasks while behaving according to their own motivation and reasoning. Each robot is an intelligent agent that perceives the virtual environment via a simulated vision system and reacts moving away from or approaching to the object it sees. The conception and specification of the robots and the environment are being done very carefully to create an open distributed object architecture that can serve as a test bed freely available and ready to use for testing theories in some computational areas such as evolutionary computation, artificial life, pattern recognition, artificial intelligence, cognitive neurosciences and distributed objects architectures. Furthermore, it is a first step towards building a cognitive animated character.

Keywords

Artificial Life, Computer Cognitive Animation, Evolutionary Computation, Genetic Algorithms, Neural Networks.

1. INTRODUCTION

Artificial life is a term originally intended to mean the simulation of macroscopic behavioral aspects of living beings using microscopically simple components [Bonabeau95]. However, the term spanned to designate a wide variety of simulated living creatures, including virtual characters whose behavior emerges from hierarchically and functionally specialized complex structures like an animal's body [Terzopoulos98]. Both kinds of simulations are very interesting from the computational point of view: they offer very attractive means to computationally model complex behavior, a subject that has gained a special relevance under both the theoretical and the applied standpoints.

Artificial life worlds are computational simulations such as virtual spaces where animated characters interact with the environment and with other virtual beings of the same or of distinct categories. Different levels of sophistication can be found in these virtual creatures, from unicellular

life with minimalist models to complex animals with detailed biomechanical models. However, all of them display behavior that emerges from the dynamics of a complex system (for instance, systems with learning, reasoning, ontologies, cognitive processes, etc).

New tendencies point to behavioral animation, where characters have some degree of autonomy to decide their actions. What we are aiming at is one of the most recent approaches to animation, cognitive animation [Funge99]. In this approach, artificial intelligence techniques are mixed with computer graphics to generate a computer animated character capable of acting in response to a storyboard or screenplay and displaying naturalism when doing other tasks not specified explicitly. Following this track to cognitive animation, we propose to exploit the capability of several methodologies and techniques that employ adaptive algorithms, evolutionary programming and artificial intelligence techniques.

Results obtained in a project set to build an artificial environment for animated virtual creatures (ARENA) are reported in this paper. Although it was conceived to allow more than one robot or virtual creature to be easily introduced to co-exist in the same environment, its first version presented here is a single-robot virtual world.

Our goal is to obtain virtual creatures capable of performing specified tasks in their environment by exploring certain strategies and adapting those whenever necessary. This scenario can be useful for many purposes: for behavior modeling, as a laboratory of learning algorithms, for research on societies of virtual characters, in the study of collective dynamics of populations, etc. The ARENA robot – WoxBot (Wide Open Extensible Robot) – has a vision system consisting of a simulated camera and a neural network that classify the visual patterns to provide input to a motor system controlled by a deterministic finite state machine (FSM). This FSM is an automaton obtained from an optimization procedure implemented with a genetic algorithm.

As an example of application, a given research project can be targeting visual recognition methods, so it will be using WoxBot as a subject employing the methods under test, and will take ARENA as a laboratory for evaluation experiments. Another research project instead, can target autonomous robot traveling strategies. In this case, the ARENA floor can be the field of traveling, and WoxBots will be simulating the autonomous vehicles. Yet the WoxBot could be re-shaped to have the aspect of a given car or truck model and the ARENA could be configured to resemble a street, in a project intended to analyze vehicle traffic conditions.

The ARENA is implemented as a distributed object environment and can run on single processor platform as well as can take advantage of parallelism or multithreading in high performance computer architectures. In this later case it could be used to handle highly sophisticated and complex applications, such as the simulation of a street with many cars. It could also be used to evaluate multi-agent distributed computational models using the WoxBots as the prototypical agents.

The organization of this paper is as follows: the next section gives an overview of the ARENA project, its requirements and choices for solutions. Sections 3 to 6 review several conceptual aspects involved in the various project components and give guidelines to be used in the implementation. Section 7 discusses particular implementation issues, such as the choice of Java and Java3D as programming language and 3D application programming interface. Section 8 addresses the modeling of the character and the environment and presents the results obtained. Section 9 foresees the further steps to be taken.

2. PROJECT OVERVIEW

Our long-term research line in artificial life is aimed at the development of complex virtual worlds with realistic creatures able to exhibit sophisticated behaviors with

reasoning, learning and cognition. To achieve it, one is required to attend to the following aspects:

- Account to the wide variety of mathematical and computational models usually required to represent the complex aspects of living behavior;
- Provide an efficient environment and platform with enough computational power to represent in detail the behavior of the simulated creatures in real time, including the ability to support live interactivity with persons;
- Specify and build a system that embodies constant evolution and improvement in its constructs at the same time that enables or, moreover, promotes the reuse of well-succeeded results and implementations.

Our choice was to fulfill these requirements building an application development environment on top of an architecture of distributed communicating objects mapped on a microcomputer cluster [Bernal99]. Additionally, the ARENA implementation also exploits parallelism via multithreading, so the program can run both on multi and single processor platforms.

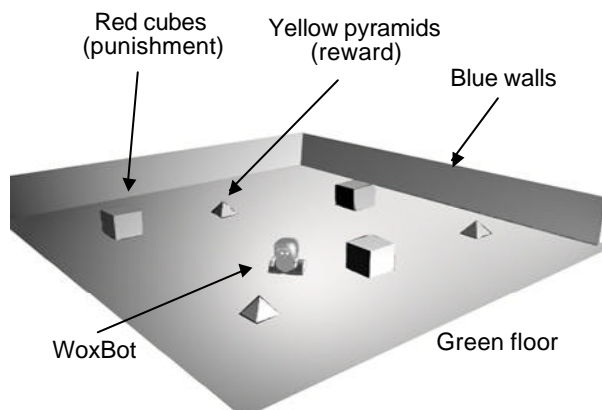


Figure 1: ARENA (Artificial Environment for Animats) – a virtual world for the development of artificial life projects. The character at the center is a robot that avoids the cubes and seeks the pyramids. A contact with a cube reduces the robot life, while the pyramids extends it. The robot goal is to prolong life as long as it can. An evolutionary computing scheme selects robots that perform better. The environment can support several robots.

The virtual space of the ARENA (figure 1) has a floor and walls encompassing the limits of the virtual world. Inside this scenario one can place objects of several kinds and functionalities such as obstacles, barriers, traps, shelters, energy or food sources etc. One or many characters can be introduced in this environment. In the first stage of the project the sole character will be the WoxBot (figure 2), whose task is to keep itself alive as long as it can.

Certain conceptual ingredients play specific roles on composing the artificial life scenario and thus should be present in the implementation: sensing and perception, knowledge use and evolution. In the following sections we examine these aspects.

3. ARTIFICIAL LIFE

Artificial life has been associated to computer science in different ways, from cellular automata theory [Adami98], [Bentley99] to computer graphics animation [Terzopoulos98], [Phillips88], [Badler92]. Some researchers have been involved with this field looking for models that would describe how real life began and evolved. In fact, they were looking for a universal life concept, which should be independent of the medium on which it existed [Adami98]. Other scientists were looking for physical models to give natural appearance to their characters. Very interesting results have been achieved through this approach [Terzopoulos98]. Although very different in purpose, these work have been related to evolution and natural selection concepts. In many of them, evolutionary computing schemes such as genetic algorithms have been an important tool to assist the conduction of transformations in the genotype of virtual creatures, allowing them to change from one generation to another. Mutation and combination (by reproduction) provide efficient ways to modify characteristics of a creature, as in real life. Selection plays a role choosing those that, by some criteria, are the best suited, and therefore are allowed to survive and reproduce themselves.

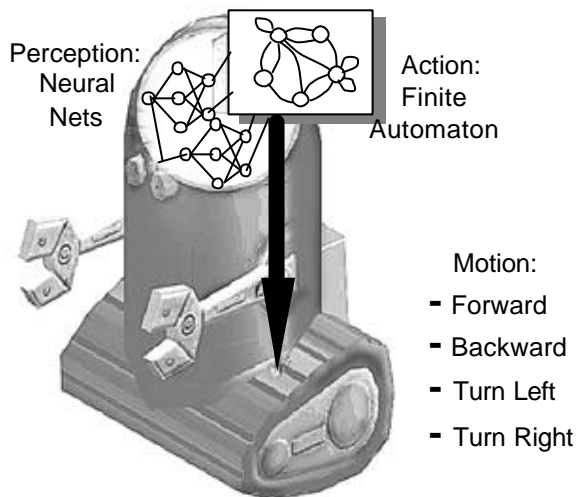


Figure 2: WoxBot (Wide Open Extensible Robot) – a virtual character that gets input from vision and reacts moving according to the decision taken by the finite state automaton.

Genetic algorithms are computational procedures capable of finding the optimal solution in particularly hard problems [Fogel95], [Holland92], [Koza92], [Mirchell97], [Beer90]. Each point of the optimization

domain is represented by a binary string that is seen as “genetic information”, in the sense that this string can be mutated, by flipping bits, as well as two such strings can be mixed by a crossover operation, in a way comparable to the actual biological reproduction. These strategies provide an efficient way to obtain global optimization in cases where it is very difficult, or not practical, to formalize the optimization problem on an analytical framework.

In this work the evolutionary computation is employed to generate the computer animated character control. Each WoxBot (Figure 2) is controlled by a finite state automaton, which evolves through generations, based on genetic mutations and crossover. Descendant robots can arise with some innovative features that may be better or worse than those from previous generations. A character being better or worse is a relative concept. In this project it is defined in a way correspondent to how well suited a character is to the environment. Those that by some criteria achieve higher grades (for instance keeping greater energy, living more time and performing their tasks faster) have higher chances to be selected for reproduction, and therefore to transfer the genotypes and, consequently, their features.

The use of knowledge in the artificial life environment can be done under two aspects: (i) it can be present in the environment and in the conception of the creatures and; (ii) it can be used by the creatures themselves when performing their actions. In both cases, the effective and rational use of the knowledge (about the world, the actions, the tasks) leads to what is called intelligent behavior. Intelligence can also be considered a property emergent from evolutionary systems [Fogel95]. Thus, the evolving characters can be modeled as intelligent agents performing tasks in the virtual environment.

4. INTELLIGENT AGENTS

Intelligent agents may be understood as computational entities that behave with autonomy in order to manipulate the information associated to its knowledge. This autonomy must regard the agent design, the environment, the goals and motivations [Russel95], [Beer90]. They have the capability of reasoning about what is going on in the environment where they are running, or of responding to some query conducted by external agents or persons.

In addition to artificial life features, our robots also exhibit features that allow their classification as intelligent agents. In this first implementation, while there is only one WoxBot in the Arena, they do not have communication, just sensing and mobility skills. And they have to decide which action they should take at every moment. A finite state machine has been evolved from random ones for this purpose. The robot should look for energy sources, while avoiding traps that make it weaker.

Based on visual information, as explained further, the robot plans its way to get closer to the energy sources. A finite state machine that can be a result of evolution through generations determines the actions of the plans.

Next versions of our robot will be able to communicate in an environment with other robots and they will implement also sociability skills, which are important features to promote a better adaptability from a creature to its environment, as in real life.

5. PATTERN RECOGNITION

One of the tasks that WoxBot has to perform in this first project is to be aware of nutrients (yellow pyramids) and hurting entities (red cubes) that are present in ARENA (figure 1). The ARENA scenario and robot vision are simulated by means of JAVA3D. The visual information is gathered from the 3D surrounding scene by projecting it to a view port with 3 color channels, namely R (red), G (green) and B (blue). Figure 3 depicts a typical input, which at the present state is taken in low resolution without any loss of accuracy, since no textures are yet used in our 3D virtual world simulation. To differentiate and spatially locate these entities, two specialized neural networks interpret the visual information received by the robot, one of them targeting nutrients and the other targeting hurting entities. These networks are named here ANN-I and ANN-II.

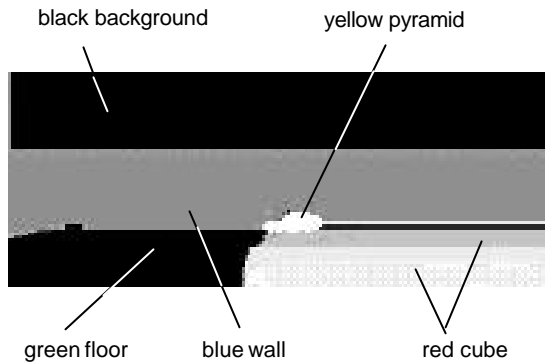


Figure 3: A typical image of the ARENA environment as seen by the WoxBot vision. The actual image has the depicted colors with shading resulting of the illumination. This image is the input of the neural nets that perform perception. It is also possible to add textures to the 3D scene and take yet more complex images.

The network ANN-I is trained for the identification of yellow pyramids as well as for a general evaluation (classification) of the position occupied by the principal yellow pyramid in the robot's visual field. To help in this task, a simple filter for the yellow color is implemented as a pre-processing stage. This is done by combining the primary sensory channels RGB so to obtain a gray scale where yellow is mapped into high values of gray and the remaining colors are mapped into low values of gray.

The monochromatic image obtained in this way is then used as the input to the ANN-I, the one that targets nutrients. After training, this network is able to activate one of 4 outputs (see Table I): output 1, indicating that no yellow pyramid is present in the visual field, output 2, indicating that the principal yellow prism is on the robot's left, output 3, when the principal yellow prism is on the center of the visual field, and output 4, when the principal yellow prism is on the right. In this way, ANN-I makes the robot aware of the presence and location of nutrients.

Code	scene situation
0	no object ahead
1	target object at left
2	target object at center
3	target object at right

Table 1: ANNI & II observers

In the same lines as ANN-I, the second neural network, ANN-II, performs a similar function for the identification and position evaluation of the red cubes representing the hurting entities. For this second network, however, the gray images that are sensed by the neural network inputs are obtained by using a different filter, which combines the channels RGB so to favor the red color.

The architecture chosen for ANN-I and ANN-II is the standard multi layer perceptron with one single hidden layer of moderate size and learning through the error back propagation algorithm.

Among the observations that we did during the training of the neural networks we want to mention that the detection and position classification of visual targets was facilitated by the use of samples (pyramids and cubes) of similar sizes. In other words, ANN-I and ANN-II don't have good abilities to implement scale invariance. That was in fact expected, since the complexity of the network is relatively limited, and its architecture does not have any specialized organization to implement scale invariance. This indicates that some kind of size invariance mechanism could be added in the pre-processing stage. Another possibility is to use a larger and more complex neural architecture so to make possible the interpretation of targets positioned at a broader range of distances.

An important issue to consider in the context of the neural networks of WoxBot is adaptability. In these initial experiments with the robot, the training of ANN-I and ANN-II was done in an isolated phase, which was performed previously to the exploration of ARENA. In other words, only after WoxBot was able to do the differentiation between nutrients and hurting entities, he started his exploration in ARENA, and then, no further adaptation of the visual recognition system was allowed.

In future phases of this project, we want the recognition system to be able to deal with unpredicted changing conditions such as change of illumination, and we will include the change of the features of the nutrients targeted by WoxBot (shape and color for example). Of course, this depends strongly on the ability of the visual recognition system to adapt, since the target recognition prototypes change with new experiences and new needs of the robot, as its life and exploration of ARENA evolves. At that point of the project, the intrinsic adaptive nature of neural networks will be crucial, and it will be fully explored.

6. ACTION, BEHAVIOR AND EVOLUTION

The WoxBot character is an intelligent agent with a simulated visual sensor to pick images of the environment from its point of observation. The two neural networks inside the agent analyze these images classifying the visual patterns. These networks outputs are tokens fed into the agent control system, which is a finite state machine (FSM). The inputs to the FSM generated by the combination of the ANN-I and ANN-II outputs are shown in Table 2.

input code to FSM	Semantics
0000	no YP and no RC
0001	no YP but RC at left
0010	no YP but RC at center
0011	no YP but RC at right
0100	YP at left but no RC
0101	YP at left and RC at left
0110	YP at left and RC at center
0111	YP at left and RC at right
1000	YP at center but no RC
1001	YP at center and RC at left
1010	YP at center and RC at center
1011	YP at center and RC at right
1100	YP at right but no RC
1101	YP at right and RC at left
1110	YP at right and RC at center
1111	YP at right and RC at right

Table 2: Input codes to the FSM and their meanings.
YP = Yellow Pyramid and RC = Red Cube

Based on the above codes, the FSM chooses one action in its repertory: going ahead, turning right, turning left or going backwards. Table 3 shows the action encoding used to build the FSM representation as a genetic code.

code	Action
00	turn left
01	go straight ahead
10	turn right
11	go backwards

Table 3: Movement encoding

The motion control is performed by the FSM automaton, which is designed without specifying in advance what interstate transitions should occur. It is initially set with a random structure that is improved based on evolutionary computation concepts. Only the maximum number of allowed FSM states is specified. The FSM is represented by a string of bits coding its states, inputs and actions. This string is named the WoxBot chromosome. For each FSM state there is a chromosome section. All these sections have the same structure: for each of the 16 possible inputs shown on Table 2, there is an entry on the chromosome state section, composed by two fields: the first one is the number of the next state after the transition caused by the corresponding input; the second field is the code of the action, following Table 3, taken upon the given input. Figure 4 gives an example of a FSM and Figure 5 is the corresponding chromosome.

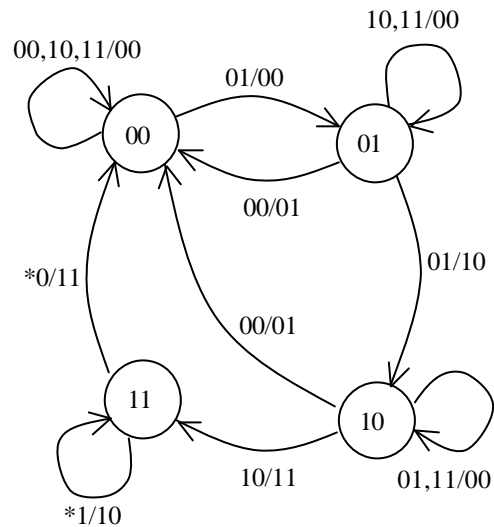


Figure 4: A sample Finite State Machine (FSM) used by WoxBot to control its reactions to the visual stimuli

At the start of the evolutionary process, a population of WoxBots is generated by sampling chromosomes at random from a uniform distribution. This population constitutes the first generation. Each member of the current generation is put into the ARENA, and it is assigned a performance value proportional to the duration of its life. A contact with red cubes shortens the life of WoxBots, while contact with yellow pyramids extends it.

State	In 00	In 01	In 10	In 11
00	00 00	01 00	00 00	00 00
01	00 01	10 10	01 00	01 00
10	00 01	10 00	11 11	10 00
11	00 11	11 10	00 11	11 10

Figure 5: Chromosome for the FSM of figure 4. There are 4 states. Each state has 4 inputs. Each Input has 2 fields, the first is the next state after the transition and the second is the action code.

After each generation of robots in the ARENA, the fittest ones are selected to be the parents of the next generation and their chromosomes are grouped in pairs. These chromosomes then undergo a mutation followed by crossover of each pair. This constitutes part of the next generation; the remnants are completed, in a heuristic method of adding variability, by drawing them from a uniform probability distribution, in the same fashion as it was done for the first generation. This procedure follows, giving thus more generations, until there is a stabilization of the fitness value of the population. Figure 6 shows schematically the whole evolutionary process.

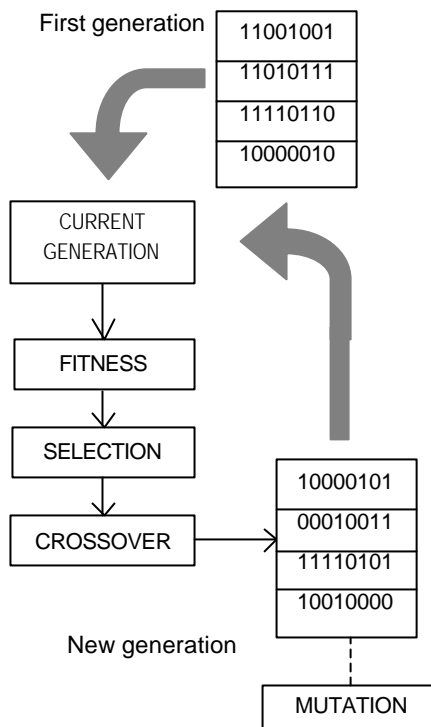


Figure 6: The evolutionary process. The first generation (top) undergoes simulation that results in fitness values. A crossover of the chromosomes of best individuals (WoxBots) yields a new generation, subject to mutation before simulation takes place again.

7. IMPLEMENTATION ISSUES

Before building an application intended for virtual world simulation, one key decision to be made is whether it will be a real time application or not. Many 3D animation packages today, like Maya™ or Softimage™, have an integrated Application Programming Interface (API), allowing algorithmic animation of characters. This approach, yet useful for some kinds of simulations, has the drawback of few or none capabilities for user interaction. These packages are also expensive, narrowing the scope of people that could later experiment with our work.

Having decided for real-time capability, the next issue is the choice of the API. Some requirements we decided to fulfill with ARENA included: portability through many platforms, share ability and open architecture, and easy of use for others. Given these issues, our choice was made for the Java Programming Language for development and Java3D as 3D graphics package. Java is a worldwide spread and freely available language strongly object-oriented. Its three-dimensional extension, Java3D, is available on a variety of platforms (Windows, Linux, Irix and others), is also freely available, and can be run through web browsers with the proper plug-in. The most powerful feature of Java3D is its scene-graph oriented approach that shifts the focus of 3D programming from vertexes to hierarchies of scene objects and events, shortening development time.

Additional advantages include the possibility of importing content authored in 3D modeling and animation packages, use of underlying OpenGL or DirectX accelerated hardware, automatic use of threads for executions, taking advantage of parallel hardware, and the availability of free Java Integrated Development Environments (IDEs), like Java Forté, from Sun Microsystems and JBuilder, from Inprise.

7.1 Application structure

Java3D organizes the objects represented in its scenes in a special kind of tree called *scene graph* [Sowizral98]. Such representation has the advantage of hierarchically organizing elements, what eases the programmer burden of implementing kinematically linked objects, such as a forearm linked to an arm, where a movement of the second implies a movement of the first. Another advantage relates to parallelism (branches of the tree potentially can be processed in parallel), and optimizations related to execution culling (whole branches of the hierarchy that aren't changed in certain ways can be optimized and made faster to process).

Some main objects were used at the ARENA, and are seen at figure 7. The objects VirtualUniverse and Locale operate as gateways to the 3D rendering engine. BranchGroup objects grant others access to the Locale and act as a grouping tool. TransformGroups represent grouped objects subject to transformations (translation, scaling and rotations).

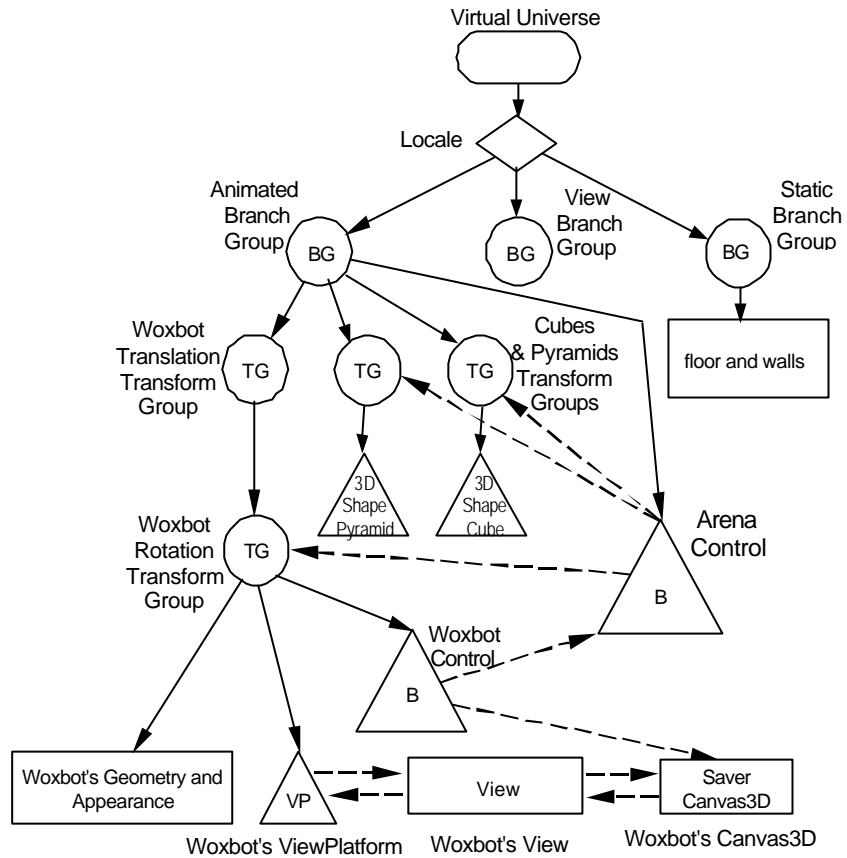


Figure 7: Java3D Scene Graph of ARENA with WoxBot. All objects linked to a VirtualUniverse through a Locale are taken into account by the rendering engine. BranchGroup (BG) objects serve as objects for grouping others. TransformGroups (TG) represents spatial transformations such as scaling, rotation and translation.

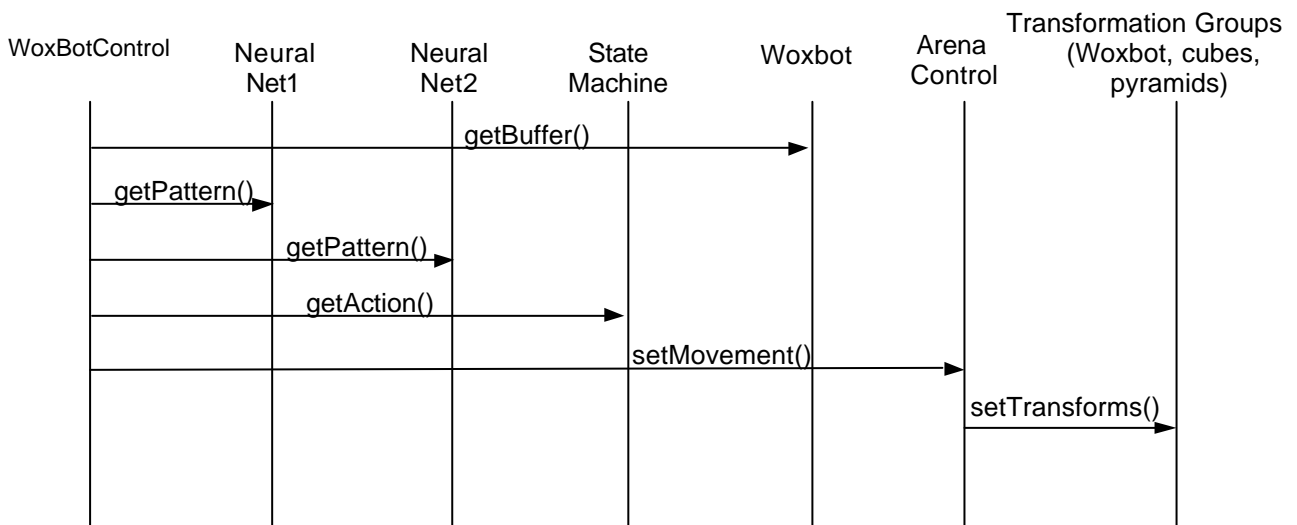


Figure 8: Sequence diagram of ARENA and WoxBot

The Java3D scene graph structure [Sowizral98] of the ARENA application is seen in figure 7. This application is composed of one Locale, with three BranchGroups directly attached to it. There is one BranchGroup for the user view of the scene (View BranchGroup), one BranchGroup holding the 3D objects in the scene that cannot move (floor and walls) and one BranchGroup holding the animated scene objects. The reason for this division is that Java3D performs a series of optimizations based on what a given BranchGroup can or cannot do, by holding all the 3D objects that cannot move in one BranchGroup, we can increase the efficiency of the optimizations. The Animated BranchGroup is where there is most to be observed in this application. The non-robot 3D objects (boxes and pyramids) have their own TransformGroup, the robot has one TransformGroup for rotation and another for translation. The WoxBot Rotation TransformGroup is the lastTransformGroup before the robot 3D mesh data, below it is also the ViewPlatform of the camera that renders the robot view (the WoxBot's View). The Animated BranchGroup also holds two control objects that are responsible for the dynamic aspect of the application: WoxBotControl and ArenaControl. WoxBotControl models the behavior of the robot, keeping track of visual data (accessing the WoxBot's Canvas3D), presents the data to the neural networks for pattern recognition and feeds the patterns into the state machine that decides robot's actions. Once robot's actions are defined, they're told the ArenaControl, that only lets valid actions take place (the robot cannot cross through walls or red cubes, for instance).

The diagram in figure 8 depicts in a simplified way the dynamic behavior of this program. An arrow from one bar to another means the first bar's object is requesting the second object to perform an operation. Return values are not represented. At extreme left, WoxBotControl can be seen. This object is responsible for most of the intricacies of the simulation. From time to time, this object polls the WoxBot Canvas3D for visual data. Then it processes the visual data and feeds the two neural networks, collecting pattern classification from them. Pattern data is then fed into state machine, which returns the action the robot should take. This desire of action is communicated to the Arena Control, that applies the results of this action to the scene. ArenaControl detects collision between the robot and other scene objects, decrementing robot's time of life when a collision is against a harmful object or incrementing it when collision is against yellow pyramids. In the latter case, the pyramids should disappear for a given amount of time, this behavior also is triggered by the ArenaControl.

8. SIMULATION AND RESULTS

The genetic algorithm was used to evolve a population of 16 individuals along 30 generations. The first generation was randomly created, every subsequent generation had the genotype of half of its members randomly generated and the other half created by a crossover of genotypes of the two most fit individuals of the previous generation, followed by a mutation. The crossover happened at a random point between the beginning and the end of the parents' chromosomes, exchanging complementary segments, so that resulting and original chromosomes have the same length. Mutation rate used in this simulation was 0.06, and it was a bit mutation rate, each bit copied from parents had a chance of 0.06 of being mutated.

A state machine with 4 states was used, which corresponds to a genotype of 258 bits (4 times 64 bits, the length of the description of each state transition / output relations, plus additional 2 bits to indicate the initial state of the machine).

The members of the generations are tested at the simulation environment one after another. In the simulation discussed in this paper each individual was given a time of 60 seconds, that could be extended or decreased on basis of the individual's behavior. Each time an individual collided against a yellow pyramid it received extra 4 seconds, that pyramid was then considered "eaten", and a new pyramid would appear at its place only after 4.5 seconds, this distribution of values was made to avoid seen behaviors where individuals has the tendency to stay at the same place indefinitely just waiting for the energy, this way they will end up dying once waiting for energy spends more energy that is gained when the waited energy arrives. The collision against a red cube caused the individual to loose 5 seconds, cubes also are considered "eaten" after the individual passes over it, and at the simulation discussed here those cubes took 3 seconds to reappear. By varying parameters such as values of the rewards and punishments related to passing over to the objects, and the time it takes the objects to reappear, we expect to see a range of strategies evolving from daring to conservative.

Figure 9 depicts the fitness plot along the generations. Some oscillation of the tendency is due to the low number of members in the populations, but the overall tendency to increasing and stabilization can be appreciated. After a long term running with bigger populations, the fitness will probably stabilize.

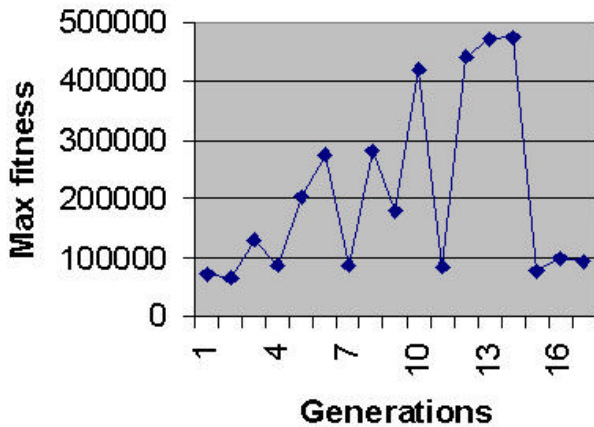


Figure 9: Fitness of the WoxBots along the evolutionary process of a population of 16 individuals.

At the present stage, the WoxBot still does not memorize the previous actions, so it actually cannot take advantage of learning. The number of states of the FSM is also low – a maximum of only 4 states is allowed. These limitations were adopted just to provide a reasonable chance for understanding the behavior of the machine by examining its structure.

Figure 10 presents a snapshot of the environment built in Java 3D to develop the ARENA / WoxBot project. Figure 11 shows the chromosome of the best-fitted individual in the above experimental run.

9. CONCLUSION AND FUTURE WORK

The first stage of the project was to build the ARENA with a single WoxBot to test the main idea and to orient the specification of the further steps. There are two main directions to follow simultaneously from now. One is to proceed with the ARENA development introducing more complexity in the environment, in the tasks and using two or more WoxBots simultaneously. The other is to improve the WoxBot to build new and more sophisticated characters.

The Jack [Adami98] project can be an inspirational experience for us, where different levels of research development and application and incremental development were successfully explored.

Concerning the actual stage of the simulations, WoxBot will have some improvements, namely to allow much more states in the FSM and to provide a memory for a finite number of previous actions. Also the ARENA will undergo some improvements concerning its size, the distribution of objects, the presence of textures and to allowance of simultaneous WoxBots. Also, concerning the WoxBots, given the use of textures and objects other than pyramids and cubes, improvements on its visual system will also have to be provided.

8. REFERENCES

- [Bonabeau95] E.W. Bonabeau and G. Therulaz, Why do we need artificial life?. *Artificial Life an Overview*, edited by C.G. Langton, MIT Press, 1995.
- [Terzopoulos98] D.Terzopoulos (org.), *Artificial Life for Graphics, Animation, Multimedia and Virtual Reality. SIGGRAPH 98 Course Notes 22*, 1998.
- [Bernal99] Volnys Bernal *et al*, PAD Cluster: An Open Modular, and Low Cost High Performance Computing System. *Proceedings of 11th Symposium on Computer Architecture and High Performance Computing* Natal, RN.Brazil, Sep 29th-Oct 2nd, 1999, 215--222.
- [Adami98] Adami, *Introduction to Artificial Life*, Springer Verlag, 1998.
- [Phillips88] C.Phillips and N.I. Badler, Jack: A toolkit for manipulating articulated figures, *in ACM/SIGGRAPH Symposium on User Interface Software*, Banff, Canada, Oct. 1988.
- [Badler92] N.I. Badler, C.B. Philips and B.L. Webber, *Simulating Humans: Computer Graphics, Animation and Control*, Oxford University Press, 1992.
- [Bentley99] P. J. Bentley (ed.), *Evolutionary Design by Computers*, Morgan Kaufmann, 1999.
- [Fogel95] D.B. Fogel, *Evolutionary Computation - Toward a new philosophy of machine intelligence*, IEEE Press, 1995.
- [Holland92] J.H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1992.
- [Koza92] J. Koza, *Genetic Programming*, MIT Press, 1992.
- [Russel95] S. Russell and P. Norvig, *Artificial Intelligence – A Modern Approach*, Prentice-Hall, 1995.
- [Mirchell97] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1997.
- [Beer90] R.D.Beer, H.J.Chiel and L.S.Sterling, A biological perspective on autonomous agent design, *in Designing Autonomous Agents*, P. Maes (ed.), MIT Press, 1990.
- [Sowizral98] Henry Sowizral (org), “Introduction to Programming in Java3D”, *SIGGRAPH 98, Course Notes 37*, 1998.
- [Funge99] J. Funge, X.Tu, D. Terzopoulos. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. *SIGGRAPH 99 Proceedings*, 1999.

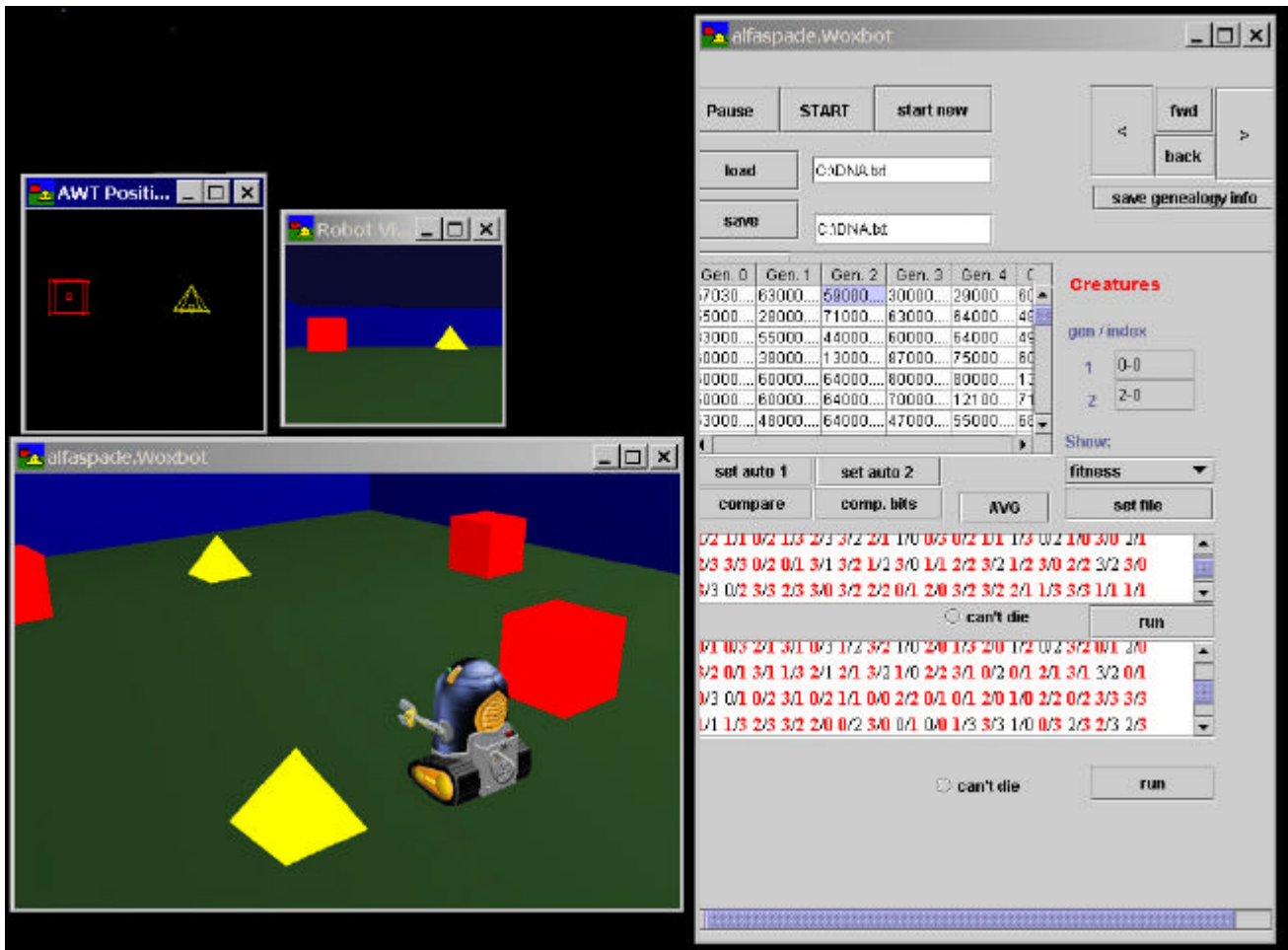


Figure 10: The Development desktop environment built with Java/Java3D. It allows the visualization of the evolutionary process live action, data logging for the population parameters, chromosomes, etc. It enables one to load a previously evolved individual and watch its performance again.

Initial state:3

```

3/2 3/0 0/3 1/0 0/3 3/3 2/1 3/3 2/3 3/0 3/2 3/1 2/0 3/1 3/1 0/2
2/2 0/3 2/3 3/0 0/3 0/1 3/0 0/3 1/0 2/2 1/0 1/1 0/2 3/2 1/0 1/3
1/0 2/3 2/2 0/3 3/1 1/0 2/0 1/0 1/3 1/0/0/0 1/1 3/2 0/0 3/2 0/0
0/1 2/0 2/3 3/3 3/2 3/1 3/3 2/1 0/0 1/0 1/0 0/0 0/2 1/3 1/1 3/2

```

Figure 11: Chromosome of the best fitted individual of the experiments run with the specifications provided in the text.