Reusable Virtual Elements for Virtual Environment Simulations

Francisco Luengo and Carol Soto

Department of Computer Science, University of Zulia, Venezuela

Summary
In this paper, we present guidelines for designing virtual elements that can be used for developing intelligent virtual environments, i.e. 3D environments with autonomous agents. We consider three kinds of virtual elements (rigid objects, smart objects, and autonomous agents) and their functionalities. These functionalities let them to interact one to each other. This approach allows for dynamically scaling and adapting the virtual element's geometry and functions to different scenarios and simulations. We also explain the steps taken in order to construct and animate such environments from reusable virtual elements.

Key words:
Intelligent Virtual Environment, Autonomous agent, Reusable virtual elements

Introduction

3D Virtual Environments have been successfully used in a broad range of contexts, ranging from Virtual Reality simulations to video games, and all sorts of interactive applications. Nowadays, the term "virtual world" has become familiar for most people. These worlds should not be limited to just beautiful 3D scene, but also they should make users feel a real sense of presence. This is achieved integrating dynamic characteristics and autonomous entities that can interact with the user or other virtual entities to the environment.

There is a necessity of creating different intelligent virtual environments for several reasons such as testing experimental behavioral models, or trying agent-object or object-object interactions, or making intelligent virtual environment simulations for training, entertainment, education, etc. In most cases, designers and developers build their own platforms for constructing those virtual environments, but the implementation of this kind of application is highly expensive in terms of time and human resources. On the other hand, the virtual elements created (any 3D object or virtual agent that forms up the graphic scene) usually are closely tight to the application, which makes very expensive and sometimes unfeasible reusing them in different platforms. In addition, one obstacle to develop those applications is the complexity of the design of the virtual elements.

Some virtual reality applications require interactions between virtual objects and a user wearing a special suit, and other ones require that virtual characters are able to manipulate the objects in their environment. Those objects made for interacting are known as smart objects. The smart objects paradigm considers virtual objects as agents with their own features of behavior.

In any virtual environment simulation, realism and believability of virtual characters plays an important role in the immersion of the user. However, one of the most challenging topics is the accurate animation of the behavior of these virtual agents as well as their interactions with objects in the virtual world. Therefore, research in this field usually focuses on defining development frameworks with reusable and pluggable components.

Allowing an agent to conduct interactions with objects, a number of general approaches may be taken. One option suggests that the design of the object inform about how to handle it. The most successful implementation of this approach was presented by Kallman and Thalmann [11] and extended by Peters, Dobbyn, MacNamee and O'Sullivan [15]. In this approach, all features, geometry, behaviors, and interaction information of the smart object are described in a text-based script file. Recently works add formal description of the action semantics in smart objects to allow the agents to meaningfully interact and create action plans even with objects never encountered before and not anticipated by the agent developer [1].

In the case of autonomous agents, separating the design from the platform is harder than in smart objects, because agent's design has to integrate the different techniques required for a realistic simulation of its behavioral. Among them, we can include those for perception, motion control, 3D rendering and animation, goal selection, action execution, communications between agents, their interaction with the environment, etc. The goal is to provide the agents a high degree of autonomy, so they can evolve freely, with a minimal input from the animator. In addition, this animation is expected to be realistic; in other

Manuscript received July 5, 2006.
Manuscript revised July 25, 2006.

words, the virtual agents must behave according to reality from the point of view of a human observer.
A number of different approaches have been proposed to fulfill the previous requirements [2,6,8]. Despite these approaches make a separation among different kinds of simulation presents in an autonomous agent (physical and behavioral simulation); it is usual finding one of them implemented in the simulation platform. Other approach aims to a general model for the representation of virtual environments based on the semantics of the virtual entities and not on their geometry [7].
In this paper we propose the design of virtual elements as reusable software components in order to create virtual environment simulations. The aim is to form a library of independent virtual elements that developers can use and share for making their own simulations in a fast and easy way.
The structure of this paper is as follow: The Section 2 briefly describes the virtual elements that appear in virtual environment simulations. In Section 3 we introduce our approach to design reusable virtual elements. Some design's considerations for each kind of virtual element are commented in Section 3, and extended in Section 4 and Section 5. Al illustrative example is presented in Section 6. Conclusions close the paper.

2. Virtual Element in Virtual Environment Simulations

Autonomous agents inhabit an intelligent virtual world. This virtual world comprises two kinds of objects which the agent can interact with: rigid objects and smart objects. By *smart objects* we mean those objects whose shape, location and/or status can be modified over time, as opposed to the rigid ones. For example, a music box or a traffic light are smart objects, simply because they might be turned on/off or red/yellow/green while a wall or a tree are rigid objects. This concept, already used in previous approaches [10,12,14,17] with a different meaning, has shown to be extremely helpful in defining the interactions between the virtual actors and the objects. We point out that saying that an object is rigid does not mean it has null influence on the agents' actions. For instance, a tree could be a rigid object but it should be considered in such tasks as collision avoidance and path planning.

With the intention of generating intelligent virtual environment simulations from independent virtual elements, it should have communication among the virtual elements that are participating in the simulation. Examples of this communication are presented as follow:

- Rigid Object - Agent: a virtual human (agent) eluding a rock (rigid object). The agent needs to know where the rock is.
- Smart Object - Agent: a virtual human (agent) trying to turn on a lamp (smart object). The agent needs to know the status of the lamp (if it is off or on), and the position of the switch and how use it. The lamp needs to know if its switch has been pressed in order to change its status.
- Rigid Object - Smart Object: a ball (smart object) rolling down on an inclined road (rigid object). The ball needs to know the road's slope and its length.

Generally, rigid objects just need to offer information about them. Instead, smart objects and autonomous agents have to get information about the status of other virtual elements in order to interact with them, and to give information about their own status.

The main idea is to design and to implement each virtual element as a software component that can be used for any programmer to create in an easy and fast way an intelligent virtual environment simulation. In that sense, some considerations should be taken.

3. Reusable Approach

With the purpose of building intelligent virtual environment simulations from individual virtual element software components, the design of these elements must consider their appearance and functionality, and includes the mechanisms that let the programmer set a simulation up in an easy way.
This aim can be reach using object oriented programming (OOP). Through *classes*, it is possible to define a completely virtual element. The class would include the visual representation (shape), features (attributes), and functionalities (methods) of a type of virtual element.
The figure 1 shows a general framework for creating any kind of virtual element. The complexity of the design is related to the element's functionalities.

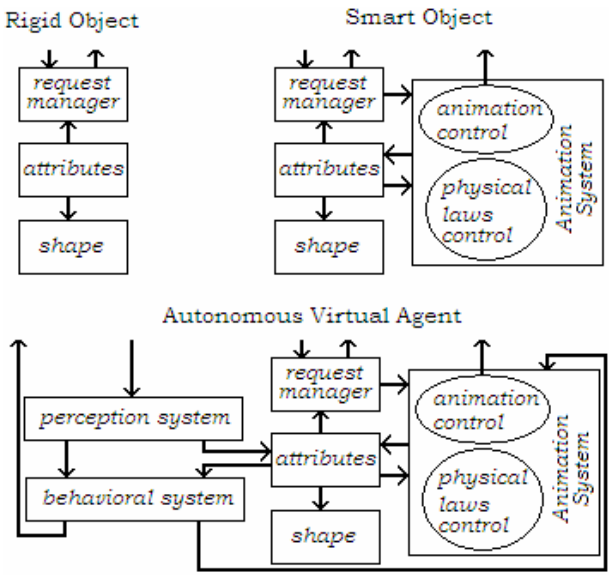

Figure 1. Structure of virtual elements

A rigid object is the simplest element. It has the basic modules of any virtual element. *Shape* is the data needed to represent the element visually. *Attributes* are shape's parameters which allow modifying some aspects of the element such as dimensions, position, orientation, color, etc. The *request manager* is formed by all methods that the programmer can used for customizing the element or getting information to other elements.

The structure of a smart object is more complicate because it includes a sort of behavior (its respond to an interaction with), and that behavior can be ruled by physical laws. A smart object needs to be animated. The animation system contains all necessary methods to get interaction information and to react accordingly.

On the other hand, an autonomous agent is able to take decisions about what to do and how to do it. This is achieved through an action selection scheme [4,14,16]. In most cases, this action selection scheme tries to simulate cognitive process with the intention of taking more realistic decisions [5,9]. The behavioral system contains methods that usually implement artificial intelligent techniques, and it uses the animation control's methods for achieve its goals. The behavioral system also requires new attributes related with emotions, feelings, thoughts, needs, and believes (about themselves, others, or the environment). Depending on those attributes, a goal is selected and different actions are taken. Furthermore, in order to interact with the 3D world, the autonomous agents must identify the different world's elements, regardless their nature (smart objects, rigid objects, other agents) and properties (location, status, etc). To this purpose, each virtual agent should have a perception mechanism that includes a set of individual sensors to analyze the environment in order to capture relevant information [3].

4. The Design

The object-oriented approach allows modeling specific features of a virtual element into individual classes; for example, the perception system and the behavior system can be implemented in separated classes (figure 1). Then, the agent's class is formed inheriting the attributes and methods from those classes. The animation system also can be implemented in its own class and to be inherited. This hierarchical model is very useful because this way, designing, testing, or modifying the virtual element's class is easier, and helps to the creation process of new classes.

We consider two levels of design: low and high level. The low level includes the virtual element's attributes and those methods that doing internal process for correctly updating the attributes. The high level correspond to the class's interface i.e. the methods used for getting information from the class, supplying information to the class, and executing some class's functionality.

Once the class of an element has been implemented, the methods of the interface are the most important ones for who are using the class because through them it is possible to customize an instance of the class and to use it. The Figure 2 shows the Class wheel's interface. This class describes a smart object used for interact with an autonomous agent (figure 3).

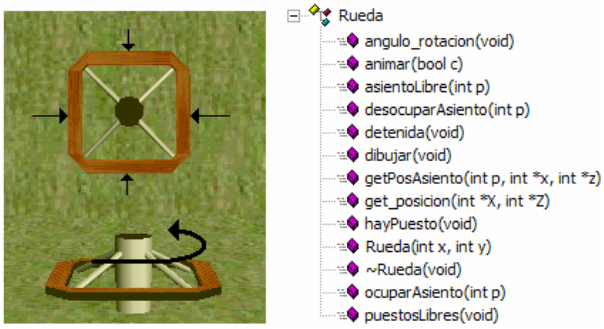

Figure 2. Class wheel's interface

Figure 3. Example of an interaction between a smart object and an autonomous agent

There are three different kinds of methods in the interface of a class: *getting methods*, *setting methods*, and *executing methods*. The *getting methods* deliver information about the virtual element's attributes to the programmer. This information depends on the kind of the element. There is basic information, that it can be getting from any element, like position, orientation, size and shape; but the programmer needs to get additional information about smart objects and virtual agents, like their status. The *setting methods* allow to customize the element (changing its size or any other attribute and to adapt it to the new environment simulation) or give information to the element that it needs to react or to interact with. A rigid object hasn't executing methods because it doesn't need to be animated. The *executing methods* are used for animate those elements that exhibit some kind of behavior.

There are common methods for all implemented classes. One of these methods is *paint method* which let draw the virtual element. Other methods depend on the virtual element and for what they were made for.

All virtual elements' classes are usually gathered in object libraries. This way, when a programmer needs a virtual element, he only has to include the corresponding object library and create an instance of the class and use it.

5. Designing Reusable Autonomous Agents

The rigid object is the perfect reusable element because it doesn't need to be animated. Smart objects aren't difficult to reuse either. On the other hand, autonomous agents can be not easier to insert into a virtual environment simulation because of their behavioral systems.

Same as smart objects, the autonomous agents respond to actions and events. But different to smart objects, the autonomous agents don't need to wait for something happen because they always are taking decisions about what to do and how to do it, and trying to achieve their goals by themselves. The agent's behavior and its capabilities of interaction depend on what the agent was created for. Each agent has its own behavioral system which to lead its actions. An executing method is associated to this system to animate the agent. Nevertheless, the behavioral system requires a lot of information to takes decisions. The perception system implements setting methods to obtain information that the behavioral system needs. If the agent is able to exhibit a lot of behaviors then it will have to supply more information about its status.

The behavioral system is usually tight to the simulation engine, making impossible to reuse this component in a different engine.

The figure 4 shows a general framework to build intelligent virtual agents (IVA). This framework, which was originally presented in [13], encloses all components that an autonomous agent needs for exhibit an autonomous and intelligent behavior.

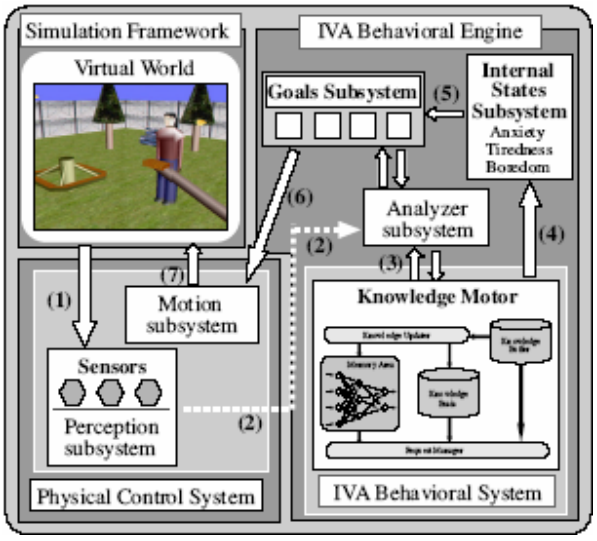

Figure 4. Scheme of the simulation framework

The physical control system implements the interface's methods. It includes methods for specific movements and others to obtain information to make decisions. The IVA behavioral engine is the "brain" of the agent. It makes the process to become information into knowledge, to select a goal to be achieved and the way to be accomplished.

Following the information flow in the figure 4, the agent receives environment's information (1) through setting methods. This information is combined with the agent's knowledge (2 and 3) to update its internal states (4) and its knowledge base (3). Internal states and knowledge are

used for establish a goal (5) and take the actions accordingly (6). Those actions are carried out using executing methods (7).

As we mentioned in the previous section, this framework can be implemented by a Class. But, it could be a hard task to add new functionalities to a Class implemented by someone else. It becomes less hard if the Class offers a set of methods that implements basic animation features because it is possible to combine some of those methods to create new ones. Also, it's important to implements independent processes in their own class. This helps to make classes easier to understand and handle.

Another aspect that makes hard creating reusable autonomous agents is that they need to identify the different virtual elements inside the virtual environment. That is important because normally its behavioral system takes decisions about interacting with a specific object or another agent. Therefore, it has to be able to identify the elements and to know how to interact with them. A solution would be using an element representation scheme like in [13].

All virtual elements' classes are gathered in object libraries. This way, when a programmer needs a virtual element, he only has to include the corresponding object library and create an instance of the class and use it.

6. An illustrative example

In this section we show how some virtual elements used for different applications can be gathering in the same virtual environment simulation.

All elements' classes were implemented using C++ and OpenGL. There are some files needed to use a class (the class's package). This package includes the source code, the element's geometry file, texture files, etc.

First, we start describing virtual elements designed for virtual park simulations. The figure 5 shows two rigid objects (a tree and a bench) and two smart objects (a wheel and a seesaw). Each element has its own class.

Figure 5. Virtual elements designed for a virtual park

The following table describes the seesaw class's interface's methods. These methods are enough for synchronizing a simulation between the seesaw and virtual agents.

Getting Methods	Description
get_is_stopped	Return true if the seesaw is being animated
get_height	Return the height of each seat
get_position	Return the position of the seesaw and its seats
get_shape	Return four vertices that surround the seesaw
get_down_seat	Return true if a specific seat is down
get_free_seat	Return the seats that aren't occupied
Setting Methods	
set_position	Let to put the seesaw in any position
set_occupy_seat	Set a seat as not available
Set_empty_seat	Set a seat as available
Executing Methods	
animate	Start or stop the animation of the seesaw

Table 1. Interface of the seesaw class

The wheel class also has an interface that allows controlling the interaction with agents.

The next figure shows two autonomous agents specially designed for virtual park simulations. Their behavioral engine includes goals that are associated to the different elements in the virtual park. Furthermore, the virtual human class, that implements the IVA, allows creating virtual agent's instances with different aspect (man, woman, child, customizing its clothes and hair).

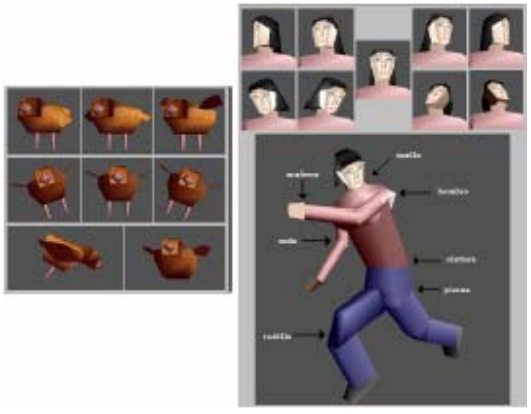

Figure 6. Two different autonomous agents for a virtual park simulation

In the figure 7 we can see some scenes of virtual park simulations, which show interactions agents – agents and agents – objects.

Figure 7. Virtual park simulation

With the purpose of testing the behavioral system's adaptability to different scenarios, a new environment was developed (figure 8). This new scenario corresponds to a shopping mall. It is possible to identify some virtual elements that have taken part in the virtual park simulation. For example, it appears the bench with a new texture, and the tree class implements several kinds of trees, like the palm tree. Obviously, it also appears the same virtual agent but news goals were added. It wasn't necessary to eliminate the virtual park's goals because in the new scenario they become unfeasible to reach.

Figure 8. Virtual shopping mall simulation

It is important to make emphasizes that all virtual human in both simulations are instances of the same class but with different attribute's values. That's reason for different agents show different behaviors.

On the other hand, in a completely different software application, virtual elements were developed to create car accident and traffic flow simulations. The figure 9 shows those elements. Most of them are rigid object, which include methods for customizing them (change their size or shape) or getting information (like position and area) in order to prevent or detect collisions. The cars were designed as autonomous agents. The car class allows selecting among two kinds of vehicles (truck or car). Furthermore, the car class implements a physical control system and a behavioral control system that let the car exhibits autonomous movement and collision responds according to physical laws.

Figure 9. Virtual elements created for car accidents simulations

A software program gathers all those elements and let to build scenarios for making car accident and traffic flow simulations (figure 10).

Figure 10. Car accident and traffic flow simulation program

However, scenarios with only rigid objects are static and bored. With the purpose of making dynamic environments for the simulations, virtual humans were added to the scenes. Those virtual humans are the same used in the virtual park simulation.

It wasn't needed making any change to the IVA class because the virtual humans just had to walking around avoiding obstacles. The figure 11 shows the agents into the car accident and traffic flow simulation.

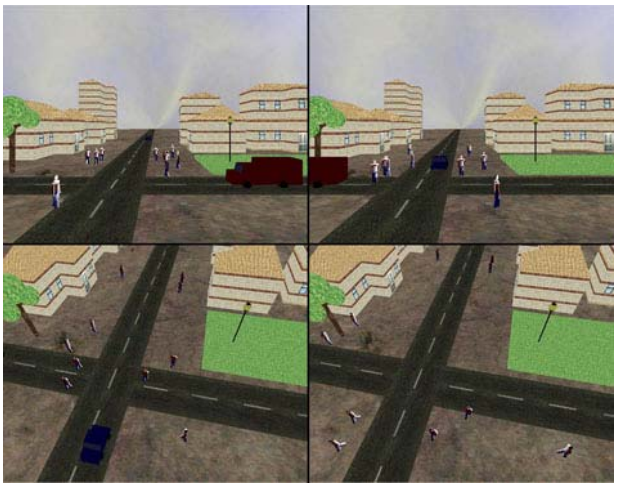

Figure 11. Integrating virtual human into the traffic flow simulation program

7. Conclusions

In this paper we have presented different considerations to the moment of designing reusable virtual elements. We use object-oriented programming approach for designing a virtual elements library where each virtual element is completely defined by its attributes and functionalities. We also show how can use this library to create different intelligent virtual environments applications.

Some approaches present platforms or tools for constructing intelligent virtual environments. Others ones present frameworks for designing such environments. But in all of them, the virtual elements are supported by an engine. This engine recognizes all the virtual elements, supporting the communication between them, and executes the simulation, controlling it.

There is no approach for the creation of reusable virtual elements that perfectly fits to all kind of virtual environment applications. The engine approach allows a better use of the computational resources but it makes difficult import the virtual elements to or exports it from other applications.

A virtual element implemented as an independent software component encloses all resources that it needs. A virtual environment simulation built from those software components couldn't be so efficient, computationally talking. Nevertheless, a careful design of the elements' classes with a good documentation allows modifying the classes adding new features when it is necessary. In any case, these classes allow creating different virtual environment applications for different purposes in an easy and fast way.

References

[1] Abaci T., Ciger J., and Thalmann D. "Action Semantics in Smart Objects". proceedings of the Workshop towards Semantic Virtual Environments 2005 workshop, March 2005, Switzerland

[2] Boulic R., Becheiraz P., Emering L., and Thalmann D. "Integration of motion and control techniques for virtual human and avatar real-time animation". ACM Symposium on Virtual Reality Software and Technology, ACM, New York, pp.111-118, 1997.

[3] Conde T. and Thalmann D. "An Artificial Life Environment for autonomous virtual agents with multi-sensorial and multi-perceptive features". Computer Animation and Virtual Worlds 2004, 15:311-318.

[4] Donnart J-Y and Meyer J-A. "A Hierarchical Classifier System Implementing a Motivationally Autonomous Animat". 3th Int. Conf. on Simulation of Adaptive Behavior. 1994. The MIT Press/Bradford Books.

[5] Farenc N., Boulic R., and Thalmann D. "An informed environment dedicated to the simulation of virtual human in urban context". Proc. EUROGRAPHICS'99, Computer Graphics Forum, pp.309-318, 1999.

[6] Funge J., Tu X., and Terzopoulos D. "Cognitive modeling: knowledge, reasoning and planning for intelligent characters". Proceedings of SIGGRAPH'99, ACM, New York, p 29-38. 1999.

[7] Gutierrez M., Vexo F., and Thalmann D. "Semantics-based representation of virtual environments". Int. J. of Computer Applications in Thecnology, Vol.23, Nos. 2/3/4, 2005.

[8] Iglesias A. and Luengo F. "A new based-on-Artificial-Intelligence Framework for behavioral animation of virtual actors". Proc. I.C. on Computer Graphics, Imaging and Visualization, pp.245-250, 2004.

[9] Iglesias A. and Luengo F. "New Goal Selection Scheme for behavioral Animation of Intelligent Virtual Agents". IEICE TRANS. INF. & SYST., Vol.E88-D, No.5 MAY, pp.865-871, 2005.

[10] Kallman M., Monzani J-S., Caicedo A., and Thalmann D. "ACE: A platform for the real time simulation of virtual human agents". EGCAS'2000, 11th Eurographics Workshop on Animation and Simulation, Interlaken, Switzerland, August, 2000.

[11] Kallman M. and Thalmann D. "Modeling Objects for Interaction Tasks". Proc. Eurographics Workshop on Animation and Simulation, 1998.
[12] Kallmann M., de Sevin E., and Thallman D."Constructing Virtual human life Simulations".Deformable Avatars, Kluwer Pub., pp.240-247,2004.
[13] Luengo F. and Iglesias A. "Framework for simulating the human behavior for intelligent virtual agents. Part II: Behavioral System". ICCS 2004, LNCS 3039, pp.237-244, 2004.
[14] Monzani J.S., Caicedo A., and Thalmann D. "Integrating behavioral animation techniques". Proc. of EUROGRAPHICS'2001, Computer Graphics Forum, Vol.20(3), pp.309-318, 2001.
[15] Peters C., Dobbyn S., Mac Namee B., and O'Sullivan C. "Smart Objects for Attentive Agents". Proc. WSCG'2003.
[16] Tyrrell T. "An Evaluation of Maes' "Bottom-Up Mechanism for Behavior Selection". Adaptive Behavior 2(4), 307—348, 1994.
[17] Vosinakis S. and Panayiotopoulos T. "SimHuman : A Platform for Real-Time Virtual Agents with Planning Capabilities". ". A. de Antonio, R. Aylett, and D. Ballin (Eds.): IVA 2001, LNAI 2190, pp 210-223, 2001. Springer-Verlag Berlin Heidelberg 2001.

Francisco Luengo is Associate Professor at the Department of Computer Science of the University of Zulia (Venezuela). He holds a B.Sc. degree in Computer Science at the University of Zulia (1992) and a M.Sc. in Computer Science at the Central University of Venezuela (1996) and a Ph.D. in Applied Mathematics and Computational Sciences at the University of Cantabria (Spain). His fields of interest are computer graphics and animation, parallel computing, artificial intelligence and numerical analysis.

Carol Soto received the B.S. degree in Computer Science at the University of Zulia (2006). She is a M.S. candidate at Applied Computer Program in the University of Zulia (Venezuela). Her research interest is artificial life, 3D simulations.