FURG Smart Games: a Development Environment to NPC Decisions in Games

Carlos Alberto B. C. W. Madsen[1] , Giancarlo Lucca[1] , Guilherme B. Daniel[1] , Diana F. Adamatti[1]
[1]Universidade Federal do Rio Grande - FURG
Brazil
{carlos.madsen, dianaadamatti, guilherme.daniel}@furg.br, gico.lucca@gmail.com

ABSTRACT**:** *The following paper aims to present a proposal for development environment to decision making layer to Non-Player Characters (NPCs) in the games. It considers the intensive use of the software reuse to implement artificial intelligence techniques required in this layer, specifically the Finite State Machines (FSM), Artificial Neural Network (ANN) and the Genetic Algorithins(GA). This enviroment consists on a framework that implements the Artificial Intelligence techniques and the generation and edition tools of the object-oriented source codes.*

Keywords: Framework, Finite State Machine, Design Patterns, Games, Non-Player Character, Genetic Algorithms, Artificial Neural Networks

Received: 12 February 2012, Revised 28 March 2012, Accepted 4 April 2012

1. Introduction

Since the beginning of the games development, the industry uses the Finite StateMachines (FSM) technique to develop the decision making layer to NPC (Non-Player Characters) [1].

However, the software complexity of the games is increasing. It demands a large quantity of working hours to have the product. In this context, the software reuse becomes fundamental and it must be widespread in this process, in order to decrease production time, rework and a good quality of the product. Despite the many qualities of the FSM and their massive use, their predictability in NPC behavior does not provide a satisfactory experience to the gamers. To have a more uncertability game, an alternative solution is to aggregate Artificial Intelligence (AI) algorithms, as Artificial Neural Networks (ANN) and Genetic Algorithms (GA), into the FSM. However, the time to form a professional with these skills is long and expensive [3][2].

For that reason, this paper proposes an environment for implementation of the decision making layer, facing the software reuse, for the NPCs by the Finite StateMachines, using ANN and GA. This software consists of a framework that implements those techniques, as well as the RAD (Rapid Application Development) tools for the edition and generation of the object-oriented source codes.

The paper is divided in five sections. In section 2 the AI techniques and their application in games development are presented. Section 3 shows software reuse techniques focused on frameworks and design patterns. In Section 4 the proposed environment, called FURG Smart Games (FSG), is presented. Finally, Section 5 presents the conclusion and future work.

2. Artificial Intelligence Techniques

This section presents the three AI techniques used by the FSG. However, we focused on the FSM, because of its importance on the implementation on the decision making of the NPCs. Despite their secondary role, the GA and the ANN are very important to achieve the goal of making the FSG an undeterministic tool, making possible a more realistic NPC behavior.

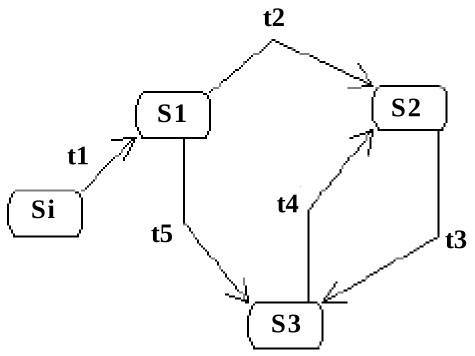


Figure 1. Graphical representation of an FSM

2.1 Finite State Machines (FSM)
The FSM have their origin in mathematics, specifically in the computability and complexity theory [2]. Its computational model had a great impact in the games industry, where normally it is an important piece in the implementation of NPC's decision making layer present in a game, being an important part in what is actually known as Game AI [1]. Through all these advantages, the FSM is widely used since its beginning until these days.

In this context, the FSM is a behavior model made of three fundamental elements: an *S* state finite set, *I* input events and *T(s,i)* transition functions, where each of these *S* states represents a specific character behavior, considering that the machine actual state is unique. The *I* possible events set represents the stimulus that NPC can receive from the environment. Lastly, the transitional functions are responsible for defining which conditions must be pleased so the machine will move from its current state to another [11] [12].

Classically, the FSM is represented by a graph, where the vertices symbolize the states and the edges symbolize the transitions, as it is illustrated in the Figure 1 (adapted from [1]).

In the FSM presented on the Figure 1, we got the set of the possible states *S* = {*Si*, *S1*, *S2*, *S3*}, and its respective set of transitions *T* = {*t1*,*t2*,*t3*,*t4*,*t5*}. In this example, the machine starts at the state *Si* and remains in it until it receives an input event associated to the input transition *t1*. If it has met its conditions, a transition will occur to the state *S1*, and so on.

There are two main ways of sorting this technique, known as Moore and Mearly machines. On the Mearly machine, the FSM output is due to transitions between the states, whereas, on the Moore machine the output is generated as a product of the actual state. The FSM typically used in games are the Moore ones, considering the system output. In this case, the character actions will be generated in their own states [13] [2].

The prolonged use and sucess of the FSM, on the games development is due to a large number of factors, as [11] [12] [14]:

• **Less developing time**: having in mind that they are conceptually simple, their project and implementation can be done relatively fast and is also easily extensible. There is also the possibility of using the project pattern State for its implementation.

• **Fast Learning curve**: the simplicity of this tool makes it easy to understand and use.

• **Predictability**: with a set of known inputs and an actual state known as well, the state transition is easily predictable, making the test and maintenance of the software easy.

• **Low Maintenance**: its implementation is performed by subdividing its source code. Typically, every state and its method, referring to the possible transitions, have their own separated code. Thus, if the implemented NPC acts somewhat unpredictably, it is easy to detect the source of the error.

• **Low computational cost**: just the code of the current state is accomplished every time, beyond a little logical code, that determines the actual state and when it will change.

• **Communication tool**: the FSM are simple enough to make it acessible to people who are not used to programming such as designers and artists.

In order to demonstrate the use of FSM in games, it presents the Figure 2 (adapted of [2]), a complete example of an NPC model from this technique. In this case, the character is a soldier who must monitor and defend a territory. Initially, he is at his base (state "*Homing*"), and, his energy is high, he begins his patrolling (state "*Patrolling*"). Then, given his vital energy level and the appearance of enemies, he passes through the other states, molding, this way, his behavior.

Lastly, it can be observed that the FSM computational model has different definitions. However, the most used by the industry is the state transition diagram of UML (Figure 2), that is basically constituted by these components [15] [16]:

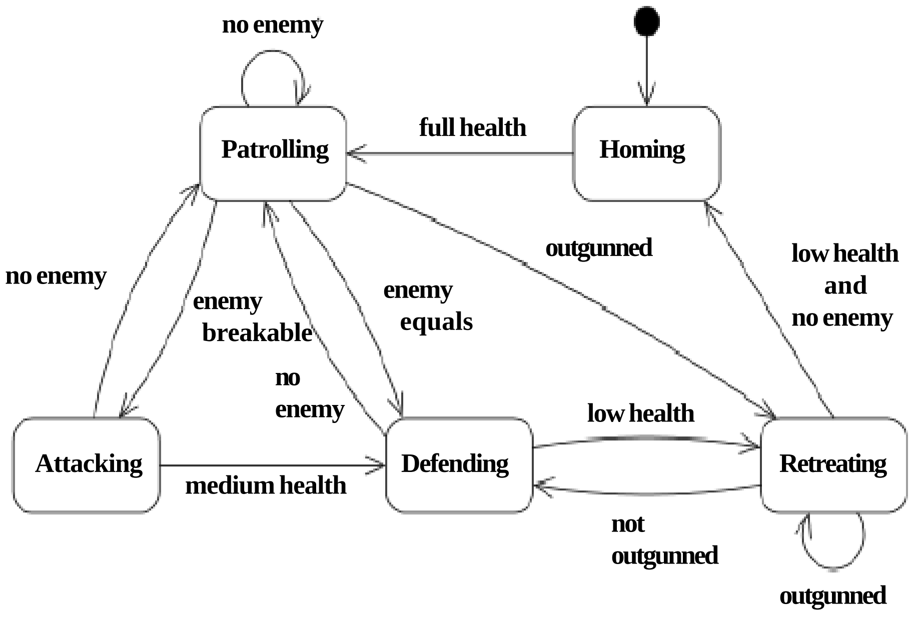


Figure 2. FSM of a soldier (NPC)

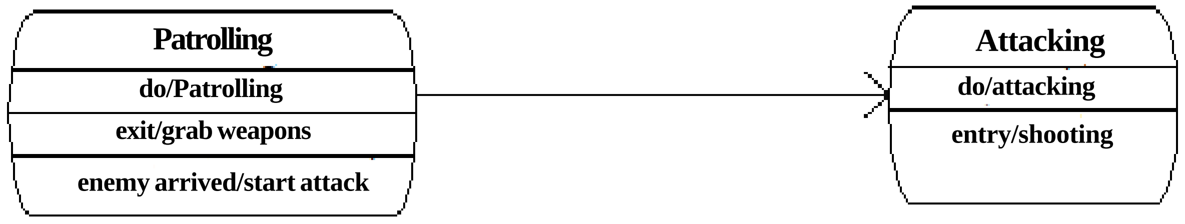


Figure 3. Transition between the state Patrolling and Attacking

• **State**: describes the internal state of the NPC at a specific instant of time, besides being related to an activity and various actions.

• **Activity**: is a continuous execution and is not associated with an atomic state, represented by the label "*do*". It has been implemented with the NPC behavior, only by being interrupted in the transition to a next state.

• **Action**: is an atomic implementation, invoked in a state change. Usually, associated with a state, it has the action "*entry*", that is executed when the NPC enters the state, and "*exit* ", that is invoked when the character leaves the state. For each transition there is the option to have a related action.

• **Transition**: is a relation between two states indicating that an NPC can pass from a "*State* 1" to a " *State* 2" given the occurrence of an event. Optionally, each transition can have a guard condition and an action.

• **Event**: environment stimulus that can unleash a state transition.

• **Guard condition**: condition that must be true so a transition stimulated by an event can happen.

• **Start state**: state, represented by a filled black circle, that indicates where the FSM starts executing.

• **Final state**: state normally represented by a hollow empty circle that symbolizes the end of the FSM execution.

With the objective of elucidating the functioning of the components presented above, in Figure 3 there is an explanation of how the transition occurs from the state "*Patrolling*" to state "*Attacking*" of FSM presented in Figure 2.

The transition presented in Figure 3 will happen according to the following steps:

1. The state "*Patrolling*" notices the occurrence of an event "*enemy arrived* ", which has been associated with the transition to "*Attacking*", governed by the guard condition "*breakable enemy*" that has to be attended.
2. The "*patrolling*" activity that was being developed by "*Patrolling*" is interrupted.
3. The output action "*grab weapons*" from "*Patrolling*" state is performed.
4. The action "*start attack*" associated with the transition is performed.
5. The input action "*shooting*" from "*Attacking*" state is performed.
6. Finally the execution of the "*attacking*" activity from "*Attacking*" state starts.

2.2 Artificial Neural Network (ANN)

The Artificial Neural Networks (ANNs) are the principal connectionism technique, AI line which studies the possibility of simulation of intelligent behavior throughout mathematic models which try to resemble to biological neural structure. These are characterized by being massively distributed processors built from a big number of units of simple processors, known as artificial neurons that are prone to store experimental knowledge and to make it available to use.

Acquiring the computational capacity throughout learning and generalization, considering an interactive process with the external environment.

This AI technique is applied mainly in approximation, precognition, classification, categorization and optimization problems, as character recognition, voice recognition, temporal series precognition, process shaping, computational vision and signal processing [19].

Among the ANN principal characteristics, can be highlighted [21] [20]:

• **Generalization**: Ability to learn through examples and generalize in order to recognize similar instances to which it has never been presented;

• **Adaptability**: Possibility to adapt sinaptical heights, absorbing environmental modifications. Like this, a network that was trained to act in specific conditions can be trained also to deal with modifications;

• **Fail Tolerance**: Capacity of reaching your objective confronting noisy signals, or even communication lost in network pieces;

• **Self-Learning**: The knowledge of a specialist is not needed to make a decision, the ANN is exclusively based on the historical examples that are offered to it;

• **Non-LinearModeling**: The mapping process of a neural network involves non-linear functions that can cover a bigger limit from the complexity of the problem.

In this paper we will prioritize the multilayer perceptrons (MLP) feedforward fully connected. These neural networks are characterized by having its neurons in multiple layers, normally one input layer, many hidden layers and finally one output layer, where the neuron of each layer is connected to all neurons of the layers immediately anterior and posterior. So, given an input signal this one propagates itself, according to the integrator and activated function of each neuron, layer by layer until the set of output is produced as a final answer of this network [21].

2.3 Genetic Algorithms (GA)
Genetic Algorithms (GA) are computationalmodels based in natural selection theory and the transmission of genetical characteristics of progenitors to descendants with the objective of solving optimization problems [22].

Based on a population, the most adapted individuals have more reproduction probability, conceiving a next generation more capable than the previous, where each individual in the population represents a possible solution to the studied problem. In this way, they are characterized as a directed search in a solution space, where the direction is given by relevant information to the problem, differently of RandomWalks techniques.

In this method, inside each generation there is an application of the principles of reproduction and selection. Throughout the selection, individuals which may reproduce themselves are determined, basing the quantification of their aptitudes, conceiving a determined number of decedents for the next generation.

The GA is characterized as global optimization algorithms, that use strategies of parallel and structured search, directing the point of high aptitude or closer to the optimal solution of the problem.

Throughout the time, they were very efficient to search optimal solutions, or close to optimal, in a big variety of problems, considering they do not present the limitations of the search methods and traditional optimizations. Highlighting the following advantages [21]:

• They work in a codification of a parameters set that should be optimized and not with the original values;

• They are highly parallelizable, because they act over a full population and not over specific sets;

• They use rules of probabilistic transition and not deterministic;

• Along the optimization process they do not use only local information. They do not take the risk of a local maxima;

• Although they have random components, they use the information of the current population to determine the next search stat

• They are not affected by abrupt discontinuities on the function to be optimized or in their derives, because they do not use derived information in their evolution or information around them;

• They are capable to deal with discrete and continuous functions.

3. Software Reuse

The reuse is intrinsic to the process of solving problems used by the human race, since its beginnings. As the solutions for certain problems were found they were recorded and, if possible, adapted to other similar problems. This was possible due to the capacity of abstraction and adaptation the human beings have [6].

Nowadays the electronic games have their development process more and more intricate and in need of human resources, mainly because of the fact that the scope and complexity of such software have become bigger and bigger.

Therefore, there is the need of reuse in software project, resulting in a systematic practice of development blocks, so that possible similarities and demands of different projects may be explored. In order to make the final product more reliable, flexible with an easy-to-do maintenance and evolution, with a better quality and with a less time consuming development [4].

Despite the reuse being a present practice in the software industry since the 1960s, with the advent of subroutines and macros, there was a technological jump in the end of the 1980s due to an intensive usage of the paradigm of programming object oriented. The orientation to objects provides an easy modularization, enabling the strong adoption of software reuse [5][4].

Nowadays we face several software reuse techniques, however among them we highlight the development based on frameworks and the design patterns.

3.1 Frameworks
A framework is a reusable and semi complete application which can be improved to produce customized applications, with specific characteristics as it gathers interrelated concrete and abstract classes, aiming at minimizing the effort of development

and maintenance of certain software, making the reuse not only in the codification, but also in the analysis and project [7]. Besides the classes previously mentioned, this technique specifies as well how such instances should interact to each other [8][5][9].

For a framework to succeed, it must present the following basic characteristics: modularity, reusability, extensibility and control inversion [6]. Modularity is achieved by encapsuling the implementation of classes, providing access, for the developer, only to its interfaces. This characteristic facilitates the finding of possible problems besides simplifying the changes in the project, reducing, thus, the maintenance effort. Concerning the reusability we can highlight the considerable advantage this technique offers. The programming through generic and preformatted components is notoriouslymore productive. The extensibility happens due to a bigger extension capacity of its components, enabling, through the heritage mechanism, the customization according to the problem domain faced. And, finally, the control inversion is important as it involves the capacity to respond to external events, keeping the application execution control [8][7].

Finally, there are two distinct forms to implement such tool, one known as the "*white box*" and the other as the "*black box*" [8]. In the White Box ones, the reuse is done exclusively by heritage of abstract classes which are available in the framework. Therefore, the developermust necessarily create subclasses to customize and implement the necessary resources for its application. On the other hand, it is necessary to have a deep knowledge of how the groups of classes work internally, causing problems for the modularity. Differently, the Black Box ones act predominantly through the composition, so the developer is only concerned about matching the instantiated objects, from the existing concrete classes, the best way possible, taking into consideration what is to be implemented. As the name suggests, the internal functioning of the classes is fully abstracted from the user, then benefiting the modularity.

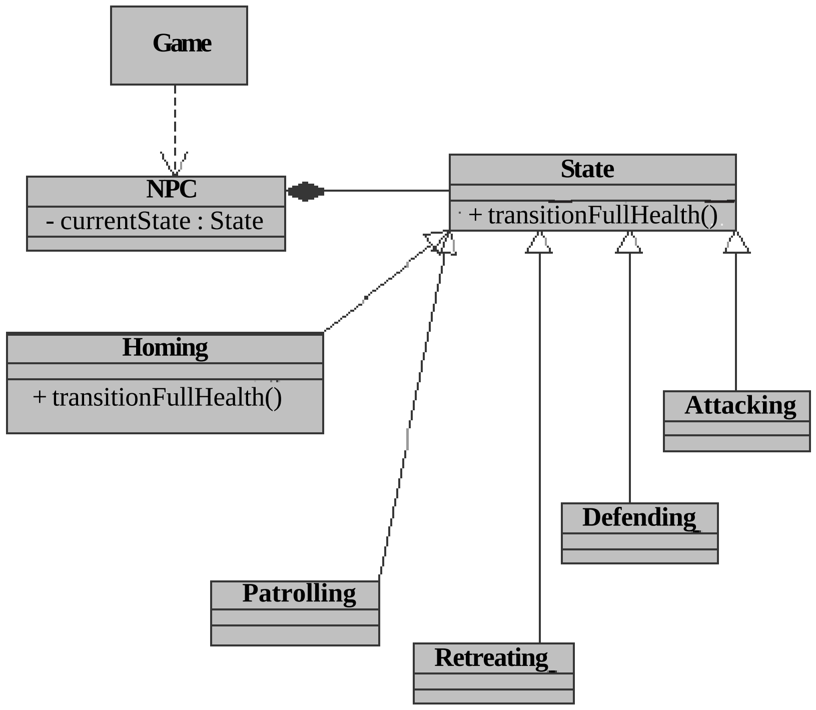


Figure 4. Example of class model of Stare Project pattern

3.2 Design Patterns

This reuse technique describes a certain problem and the essence of its solution, therefore it can be adapted for several applications. Due to its abstract nature it does not emphasize implementation details. It can be seen as a description knowledge and accumulated experiences of a certain proved solution for a common problem.

As a result, a design pattern names abstracts and identifies key aspects of a structure of common design, in order to make it useful to create object oriented software. This close connection to object oriented programming happens due to the fact the patterns have characteristics of objects, such as the heritage and polymorphism. They are defined as:

"*descriptions of communicating objects and classes which should be customized to solve a general design problem in a specific context*" [10].

Finally, according to Gamma et. al [10] and Sommerville [6] there are four essential elements of a design pattern:

• **Pattern name**: reference which may be used to describe a pattern problem, its solutions and consequences in one or two words.

• **Problem**: describes in which situation the pattern must be used, explaining the problem and its context. It may include a list of conditions which should be fulfilled so the usage of the pattern makes sense.

• **Solution**: describes the elements which compose the design pattern, its relationships, its responsibilities and collaborations. In this item specific implementations are not described, as the pattern works more as a template, so it provides an abstract description of a problemand how a group of classes and objects may solve the problem.

• **Consequences**: they are results of advantage and disadvantage analysis of pattern application. They include its impact on the flexibility, extensibility or portability of a system.

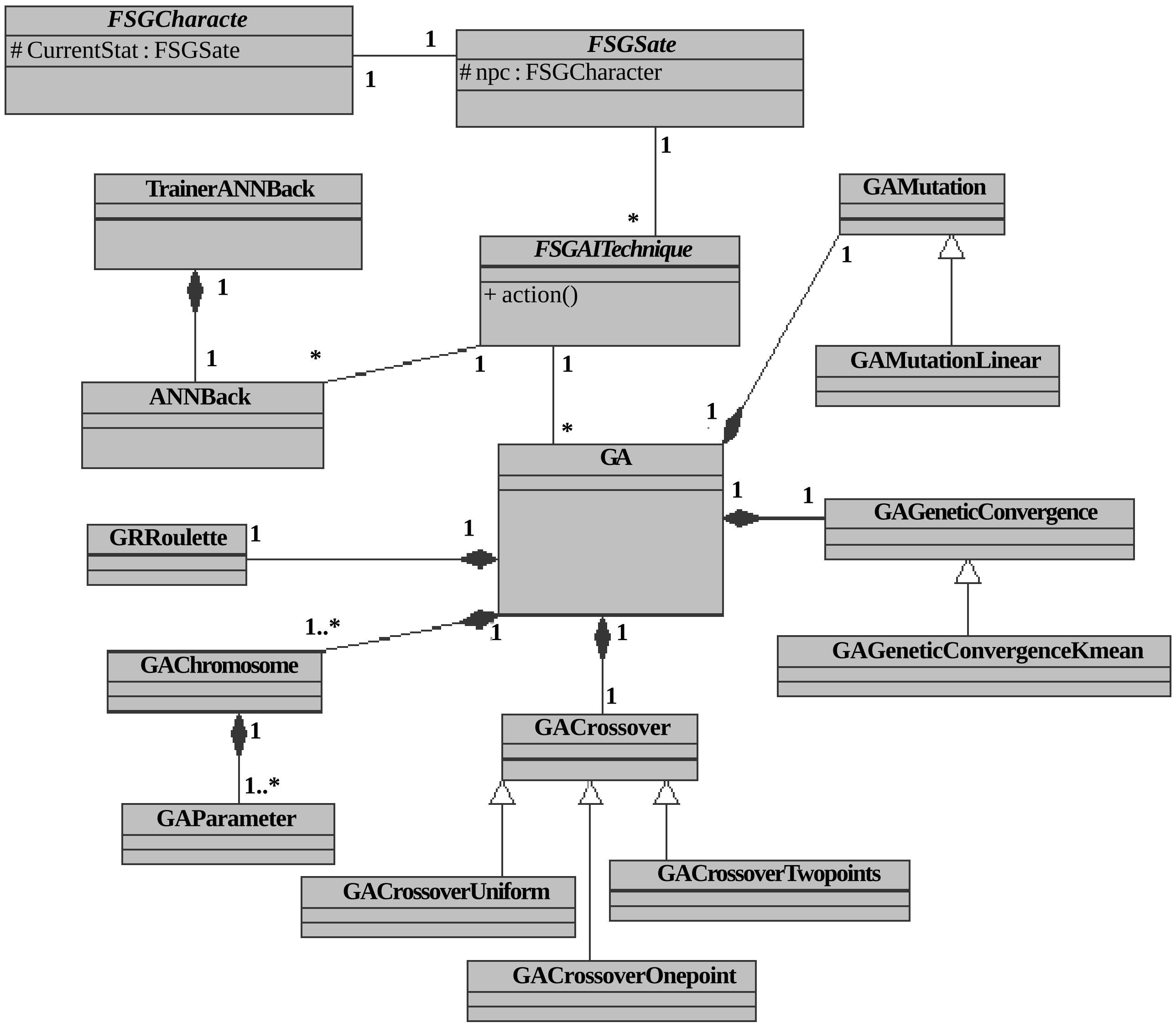


Figure 5. Class diagram of the FSG

3.3 Design Pattern State

State is a behavioral pattern, which means it aims to control the manner in which its objects. In this NPCs context, interacting among them, encapsulating interchangeable behaviors and using delegation in order to decide which behavior should be used [17]. Thus, allowing that a certain object changes its behavior according to its internal state, giving the impression that this object changed its class [10] [15]. Real time applications, as games, tend to be benefited specially from this architecture because they frequently work with a perspective of event and changes of state.

An example of use of this pattern can be seen on Figure 4, related to the FSM presented on Figure 2, in which it can be observed that the game can launch events to be served by NPC, which delegates the task to its current state.

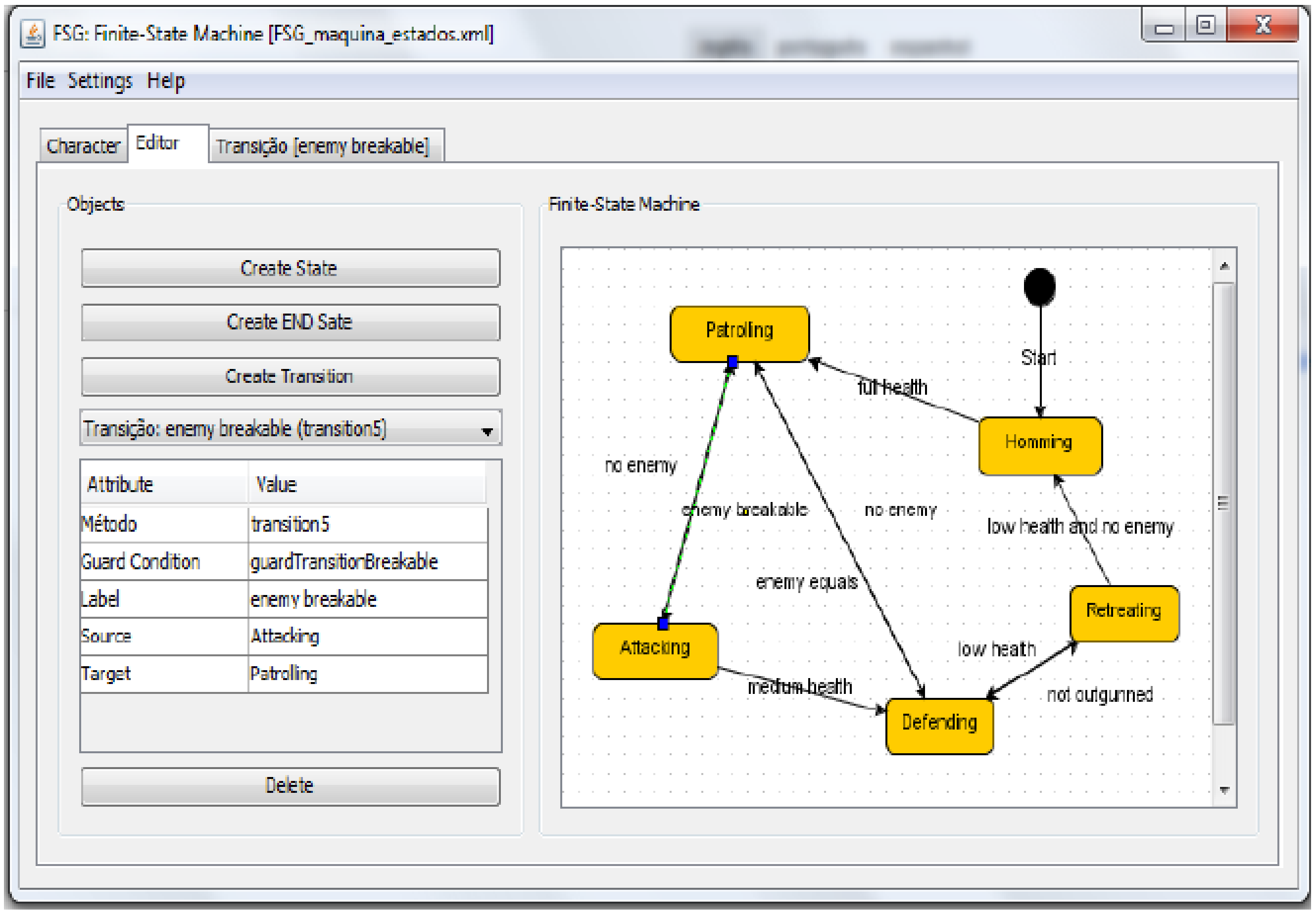
4. FSG Environment

The development environment of the decision making process, from the FSM, proposed in this paper, is divided into a framework and a RAD tool for automating source code. A previous work of the authors is presented in [18].

4.1 Framework FSG

The proposed framework aims at facilitating the incorporation techniques of AI in the development of games, assuming that the process of decision making of the NPC is implemented by a FSM, highly used technique by the industry. In the FSG each transition present in the machine may be related as an AI technique, as ANN or GA, through its guard condition. As a result, the character's behavior is not strictly deterministic, making the game more realistic.

To achieve this goal, the developer that uses the tool will be faced with two very different ways of working. First, as regards the



Transição: enemy breakable (transition5)

Attribute	Value
Método	transition5
Guard Condition	guardTransitionBreakable
Label	enemy breakable
Source	Attacking
Target	Patrolling

Figure 6. FSM Edition in RAD tool

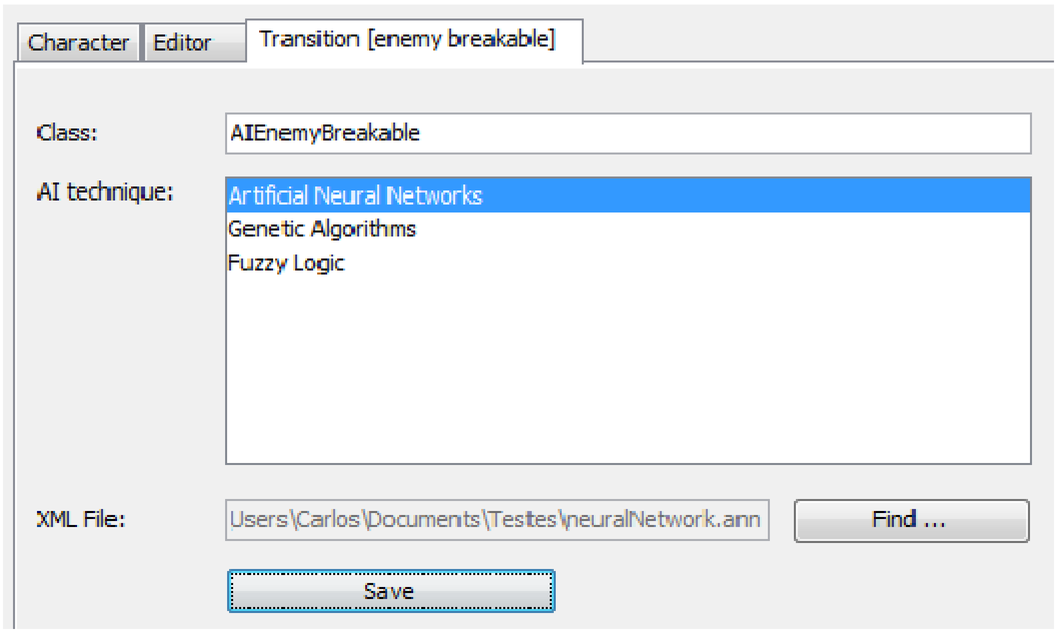
state machine, he/she must rely primarily on inheritance, and to implement the present methods in abstract super classes and, also to know in which order and time. In this case, it is a white box framework. Moreover, as regards the ANN and GA, the implementation is completely abstracted, provided through the concrete classes. So the developer is limited to linking and aggregating their instances. It is a black box framework.

It is believed that with this work methodology, the FSG understanding is facilitated as it is necessary to have a deeper knowledge concerning the FSM, which is mentioned in section 2.1 as conceptually simple and with low learning curve.

Moreover, as regards the ANN and GA, that have a higher learning curve, the framework already does this job, leaving to the developer just to figure out at what time a particular technique must be used in preference to another.

Still, due to the fact the games are implemented in several languages and are executed in distinct platforms, the choice for the proposal and implementation of the FSG was made in an XML based language, similar to Java (the formal description of this language will not be presented here, as this is not the focus of the paper). By using a RAD tool, this XML and later translation for a programming language will be created, as long as there is support for object orientation.

In the Figure 5 we have the class diagram of the FSG. We can observe the relation between the classes and the three AI



Class: AIEnemyBreakable

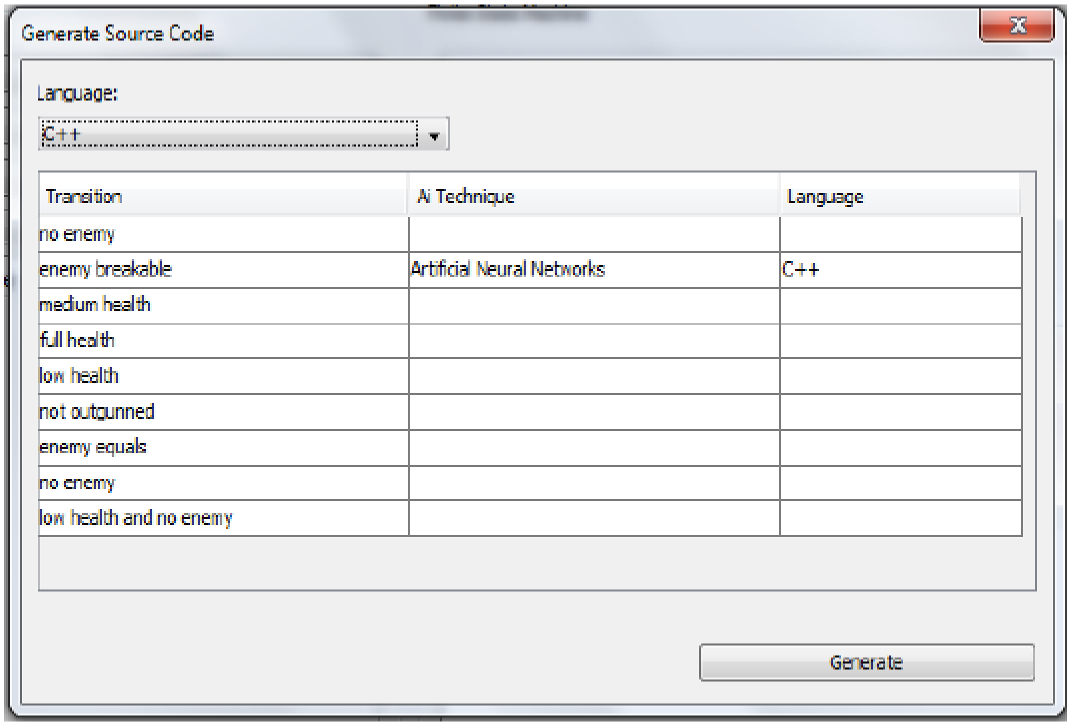
XML File: Users\Carlos\Documents\Testes\neuralNetwork.ann

Figure 7. Relating one transition with an AI technique

techniques. Regarding the FSM it can highlight the following classes: FSGCharacter, and FSGState FSGAITechnique. From the FSGCharacter class there is the NPC implementation, where we see the presence of the attribute "*currentState*" which shows in which condition the machine is. In the FSGState class, all the states found in the FSM must be implemented. We call attention for the fact that all actions, activities and guard conditions proposed in the UML study diagram, are expected in its method.

Finally, the FSGAITechnique class works as an interface for the AI techniques, present in the framework, and the FSM state transitions. Normally their objects are instantiated within the "*guardCondition*" method of a certain class inheriting FSGState. In this way, as demonstrated in the section 3, it is clear that the tool, is in accordance with the UML state model and it has a consistent implementation of the Design Pattern State.

Related to the ANN, we can highlight the following classes: ANNBack and TrainerANNBack. On the ANNBack we have the neural network itself, allowing the implementation of an MLP with one hidden layer, where three layers are defined: the neurons,



Transition	Ai Technique	Language
no enemy		
enemy breakable	Artificial Neural Networks	C++
medium health		
full health		
low health		
not outgunned		
enemy equals		
no enemy		
low health and no enemy		

Figure 8. Generating the NPC source code

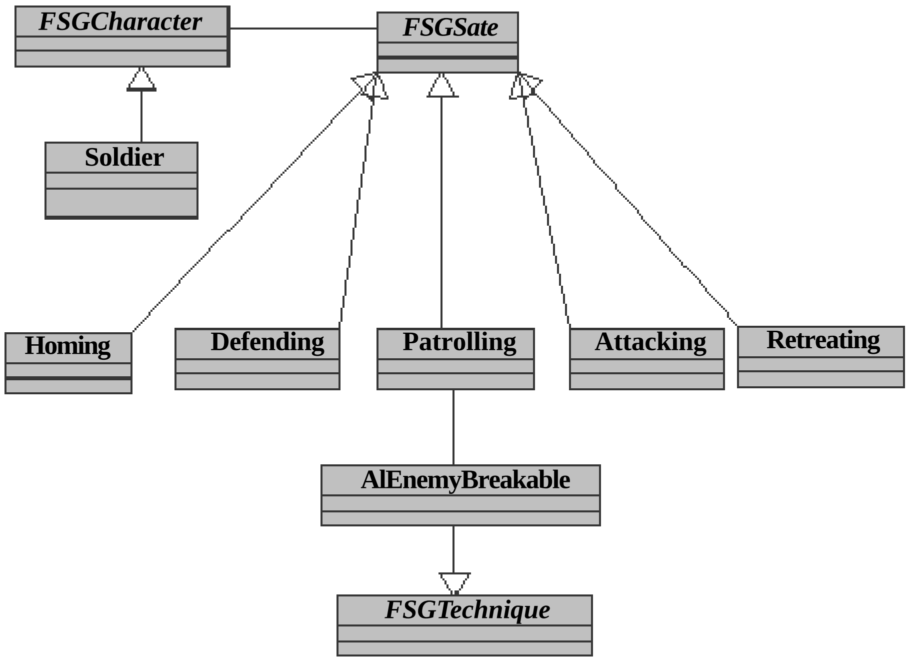
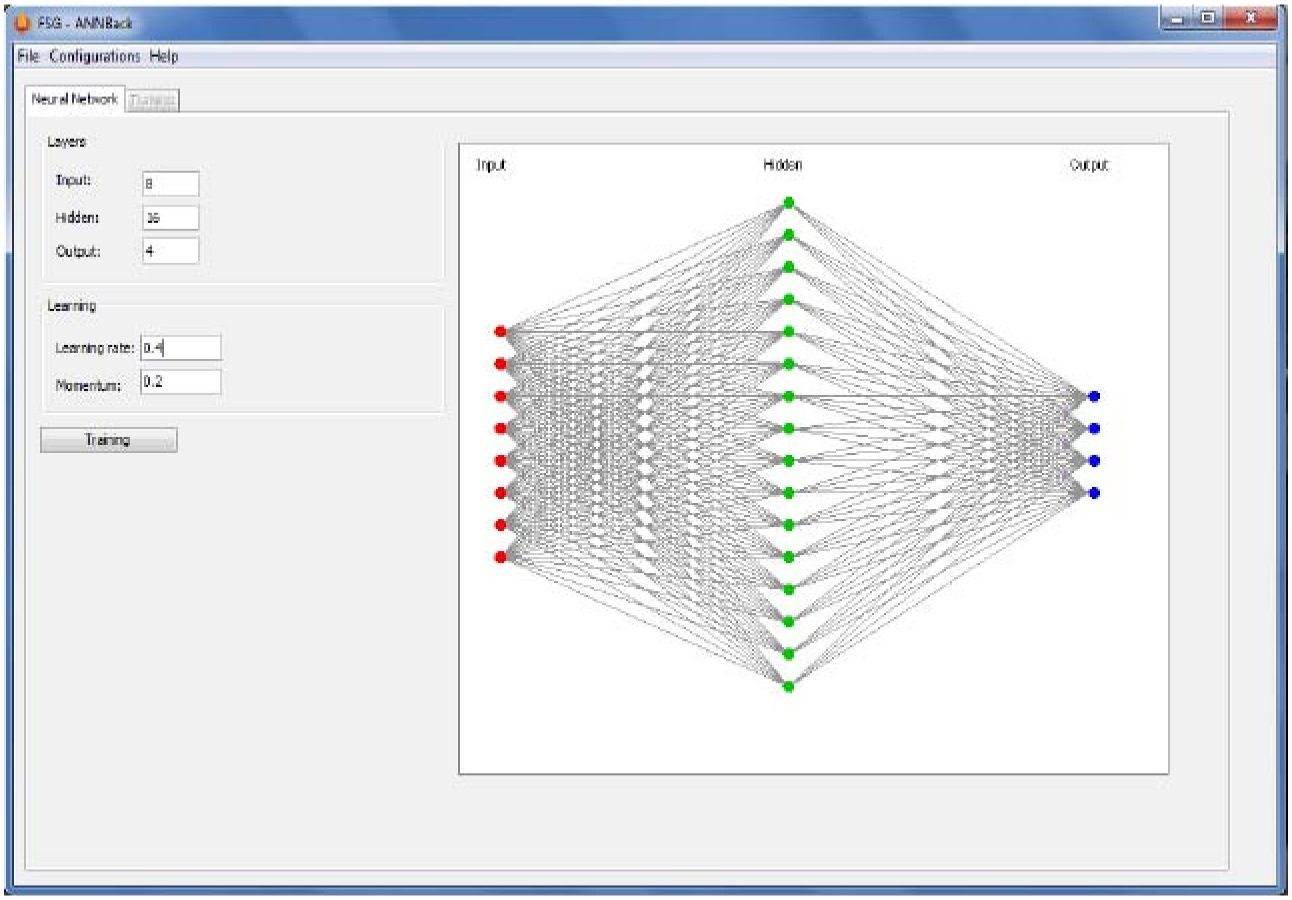


Figure 9. Class diagram of the source code generate to the FSM from Figure 6

the synaptic weights and the methods of propagation and backpropagation. The TrainerANNBack is responsible for training the network, where are defined the acceptable error, learning rate, and the maximum number of times.

Finally, concerning the GA, we can highlight the following classes: GA, GAChromosome, GACrossover, GAMutation and GAGeneticConvergence. GA is the main class of this AI technique, where the chromosomes population is found and where the



Layers

Input: 8

Hidden: 16

Output: 4

Learning

Learning rate: 0.4

Momentum: 0.2

Figure 10. Initial RAD screen of neural networks

algorithm is executed. Each individual in the population must be an instance of GAChromosome, given that this has the binary representation of the chromosome. The reproduction occurs by the class that inherits the GACrossover, which can be one point, two points or uniform. Once the reproduction is done, there is the possibility of some of the genes of the offspring through an instance of the GAMutation class. Finally, new generations will be created successively until there is a genetic convergence. For that the method to detect the event must be implemented by one inheritance of GAGeneticConvergence, that is the case of K-mean algorithm K-mean [23].

4.2 RAD

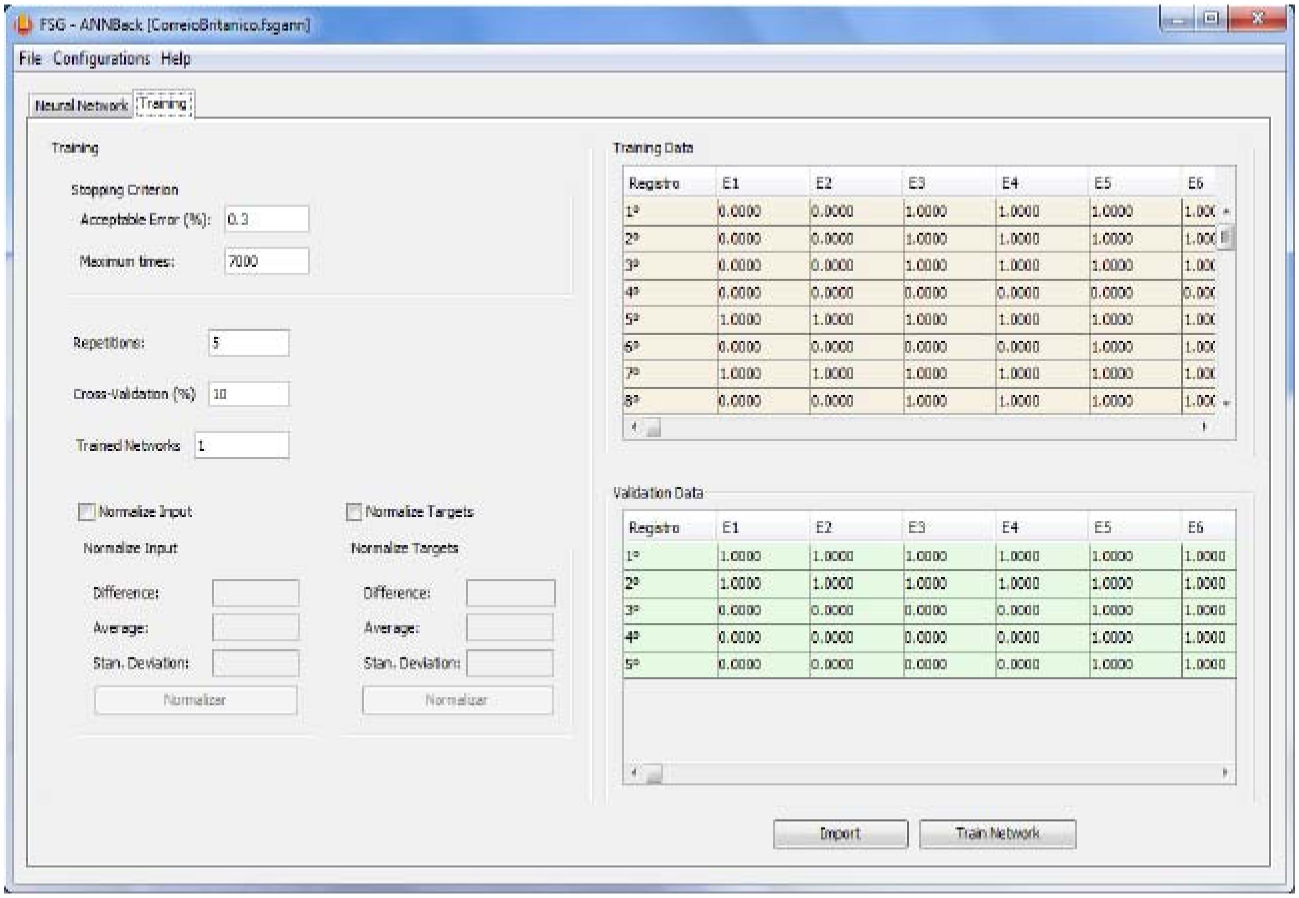
The second part of the environment of development is a RAD set of tools responsible for the edition of the AI techniques and posterior generation of the source code in the FSG framework pattern.

4.2.1 RAD (FSG: Finite-StateMachine)

The main RAD tool is responsible for the graphic edition of FSM, which occurs throughout UML simplified state diagram, where questions as actions, activities and guard conditions are not presented, considering that they are already available on the framework FSGstate class, as the Figure 6 illustrates. Still in this figure it might be observed in the left that the software allows, in state case, the specification of the class name which will inherit FSGstate, and in case of a transition of the definition of the method that triggers it, as well as the responsible methods name for implementing its guard condition.

As shown in subsection 4.1 there is the possibility of relating the guard condition to a specific AI technique, shown in Figure 7. In it the developer defines the class name which will inherit FSGAITTechnique, choose the technique and lastly load an XML file with its configuration.

In the example of Figure 7 the file must contain the artificial neural network definition as well as the values of its weights



Training

Stopping Criterion

Acceptable Error (%): 0.3

Maximum times: 7000

Repetitions: 5

Cross-Validation (%) 10

Trained Networks 1

Normalize Input

Normalize Targets

Normalize Input

Difference:

Average:

Stan. Deviation:

Normalize Targets

Difference:

Average:

Stan. Deviation:

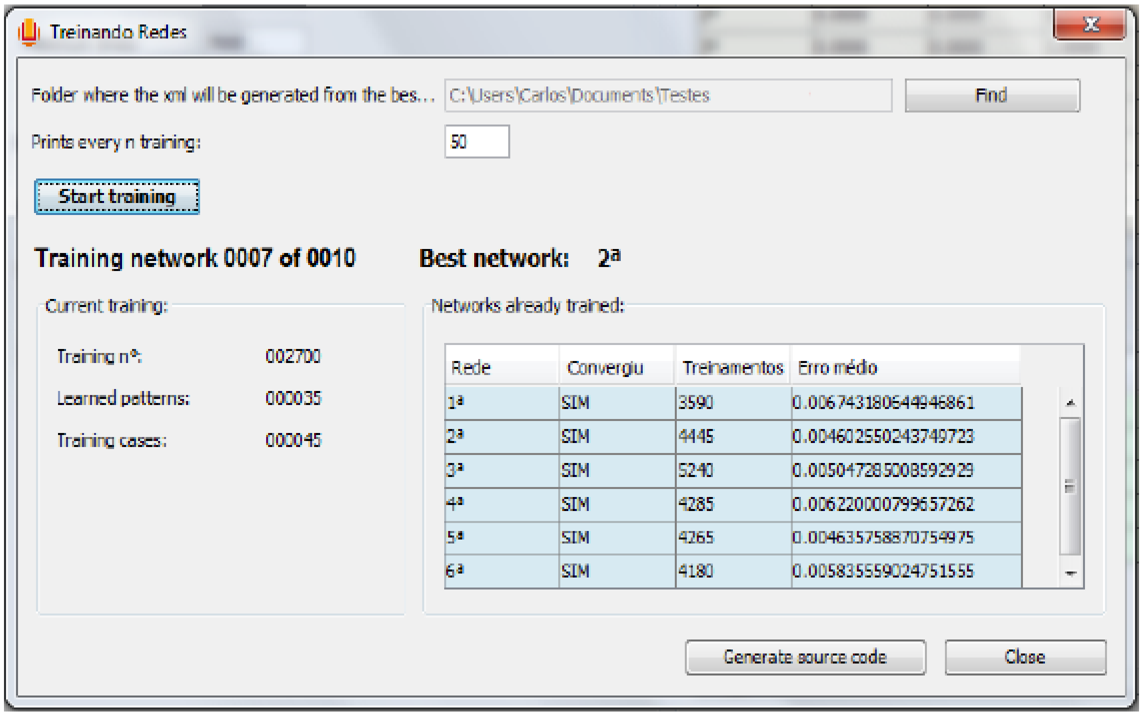
Training Data

Registro	E1	E2	E3	E4	E5	E6
1º	0.0000	0.0000	1.0000	1.0000	1.0000	1.00(
2º	0.0000	0.0000	1.0000	1.0000	1.0000	1.00(
3º	0.0000	0.0000	1.0000	1.0000	1.0000	1.00(
4º	0.0000	0.0000	0.0000	0.0000	0.0000	0.00(
5º	1.0000	1.0000	1.0000	1.0000	1.0000	1.00(
6º	0.0000	0.0000	0.0000	0.0000	1.0000	1.00(
7º	1.0000	1.0000	1.0000	1.0000	1.0000	1.00(
8º	0.0000	0.0000	1.0000	1.0000	1.0000	1.00(

Validation Data

Registro	E1	E2	E3	E4	E5	E6
1º	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2º	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3º	0.0000	0.0000	0.0000	0.0000	1.0000	1.0000
4º	0.0000	0.0000	0.0000	0.0000	1.0000	1.0000
5º	0.0000	0.0000	0.0000	0.0000	1.0000	1.0000

Figure 11. Network training area



Folder where the xml will be generated from the bes... C:\Users\Carlos\Documents\Testes

Prints every n training: 50

Training network 0007 of 0010

Best network: 2ª

Current training:

Training nº: 002700

Learned patterns: 000035

Training cases: 000045

Networks already trained:

Rede	Convergiu	Treinamentos	Erro médio
1ª	SIM	3590	0.00674318064494686l
2ª	SIM	4445	0.004602550243749723
3ª	SIM	5240	0.005047285008592929
4ª	SIM	4285	0.006220000799657262
5ª	SIM	4265	0.004635758870754975
6ª	SIM	4180	0.005835559024751555

Figure 12. Performing the network training

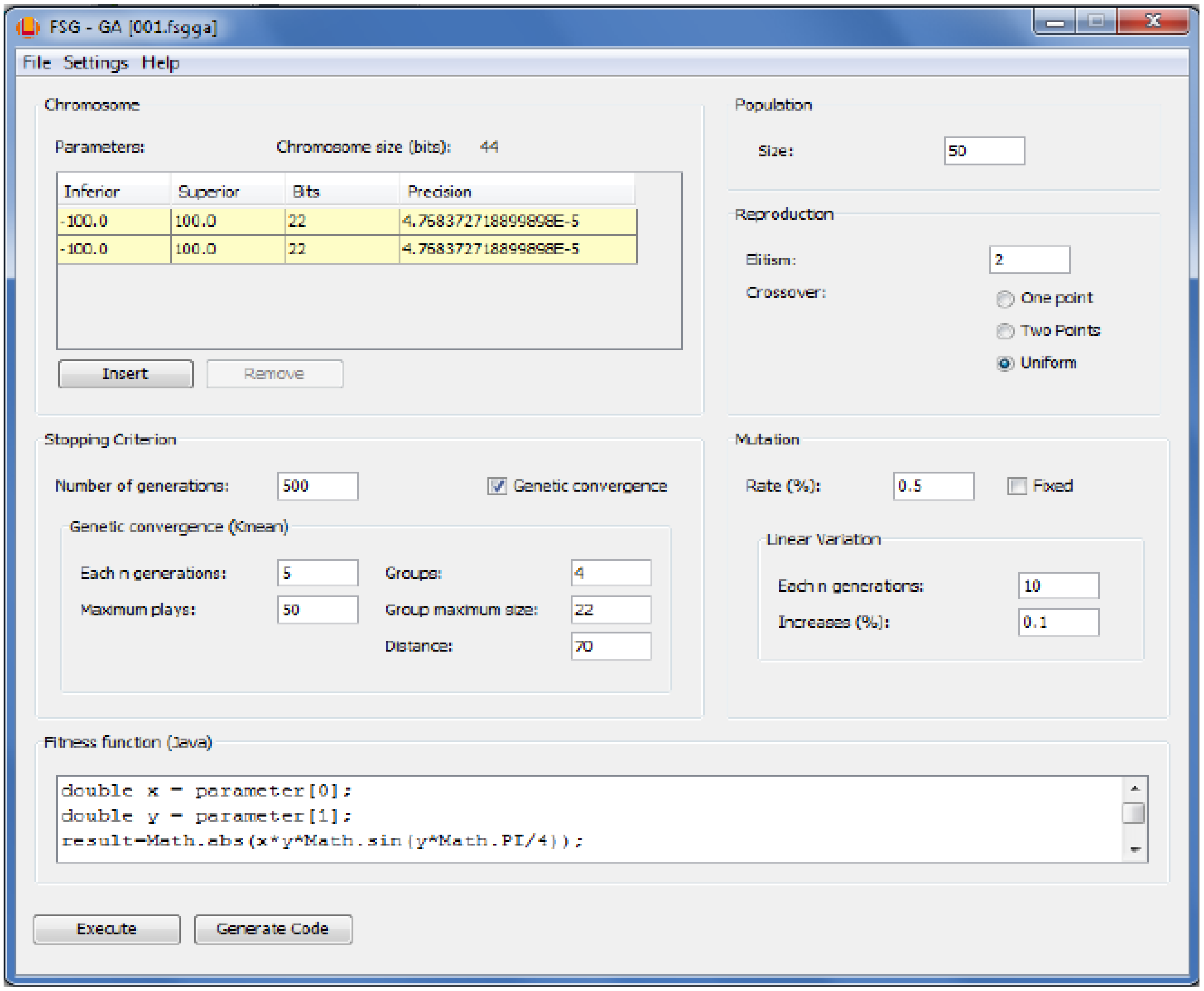
previously trained. The RAD allows other algorithms, beyond the presented, to be cataloged.

Then, the developer can request the generation of the NPC source code in the programming language of his/her preference, as presented in Figure 8. It can be observed that the tool identifies the use of an AI technique related to "*enemy breakable*" transition, this way it informs that the codes that refer to the implementation of this technique will also be generated.

The generation of this source happens in two steps. Firstly RAD generates all the NPC and AI algorithms classes connected to its XML specified based language. After that, it invokes a translator, previously cataloged in the software, corresponding to the selected programming language. The programmer translates this XML in source code.

Nowadays, it is possible to translate to the programming languages: C++, Java, PHP and Python.

Lastly, to the NPC "*soldier*" presented in Figure 6, we have a source code generated according to the diagram of classes in Figure 9.



Chromosome

Parameters: Chromosome size (bits): 44

Inferior	Superior	Bits	Precision
-100.0	100.0	22	4.76837271889989 8E-5
-100.0	100.0	22	4.768372718899898E-5

Population

Size: 50

Reproduction

Elitism: 2

Crossover: One point [✓ Uniform] Two Points

Stopping Criterion

Number of generations: 500 [✓ Genetic convergence]

Genetic convergence (Kmean)

Each n generations: 5 Groups: 4

Maximum plays: 50 Group maximum size: 22

Distance: 70

Mutation

Rate (%): 0.5 Fixed

Linear Variation

Each n generations: 10

Increases (%): 0.1

Fitness function (Java)

```
double x = parameter[0];
double y = parameter[1];
result=Math.abs(x*y*Math.sin(y*Math.PI/4));
```

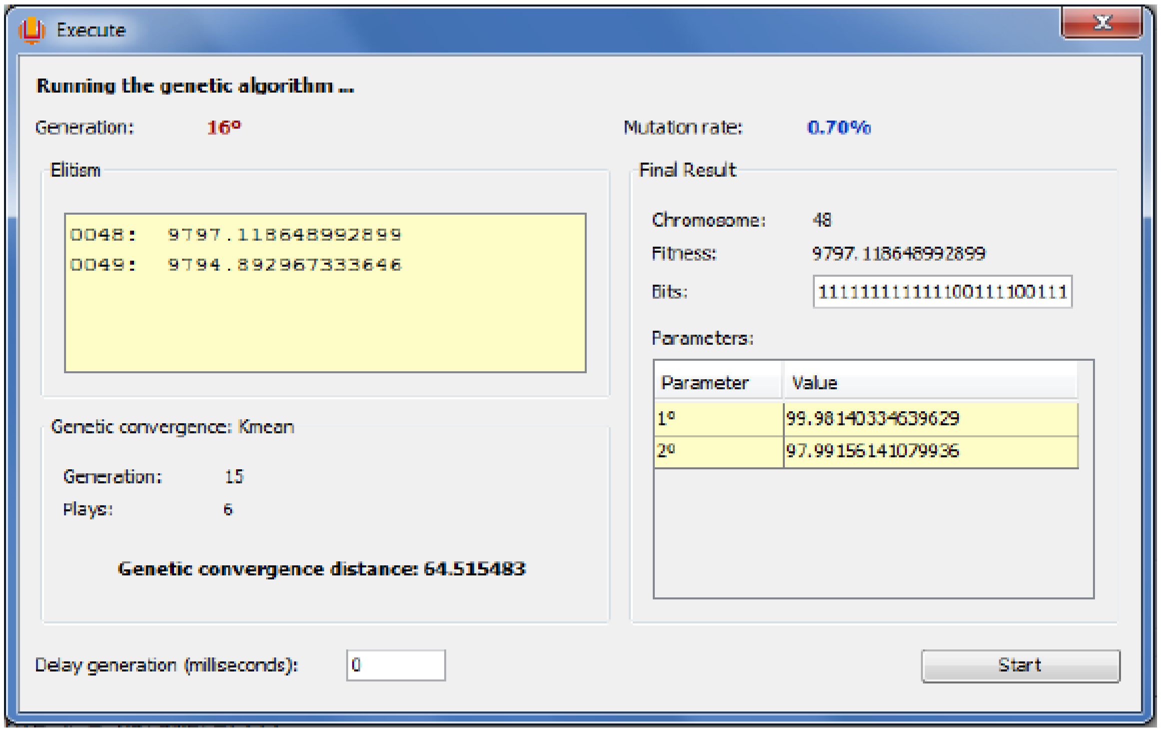
Figure 13. Initial RAD screen of genetic algorithms

4.2.2 RAD (FSG - ANNBack)

Similar what was presented on previous subsection, the ANN has a RAD tool, responsible for its configuration, training and generating the source code on the pattern FSG.

Referring to the network configuration, as observed in Figure 10, the software allows the definition of the number of neurons in each layer, as well as its learning rate and momentum.

In Figure 11 the network training interface is presented, highlighting the stopping criterion, cross-validation size, number of training networks, normalization, set of training data and a set of validation data.Once ANN has been configured, the training



Running the genetic algorithm ...

Generation: 16º

Mutation rate: 0.70%

Elitism

```
0048:  9797.118648992899
0049:  9794.892967333646
```

Final Result

Chromosome: 48

Fitness: 9797.118648992899

Bits: 11111111111100111100111

Parameters:

Parameter	Value
1º	99.98140334639629
2º	97.99156141079936

Genetic convergence: Kmean

Generation: 15

Plays: 6

Genetic convergence distance: 64.515483

Delay generation (milliseconds): 0

Figure 14. Performing the genetic algorithms

must be performed, as illustrated on Figure 12. By the end of this process, among the diverse trained networks, the software will select and save in an XML file the information of the less errors network on the output layer. This XML will be used on RAD of FSM, as presented on Figure 7. Finally, if the developer needs an ANN, he/she has the option to generate his/her own source code on the pattern of the framework.

4.2.3 RAD (FSG - GA)

At the same way what was presented in the two previous subsections the GA has a RAD tool, responsible for the configuration of the genetic algorithm, performance and validation testing, and generation of the standard source code of the FSM.

Regarding the configuration of the GA, as we can observe in the Figure 13, it allows the definition of all the necessary attributes to classes instantiation, as presented in the Figure 4. Highlighting the chromosome structure, the population size, the reproduction form, the mutation, the genetic convergence, the stop criterion, and finally, the fitness function.

Also, there is an option to generate the source code in the framework standard form.

There is the possibility to execute and validate the GA within the tool, as shown in Figure 14.

5. Conclusion and FurtherWork

The FSM is a consolidated technique in the game industry to implement decision making in NPC. However, this technique is deterministic and it needs to be used together with AI algorithms. Nowadays, the AI use and the wide complexity of new games, makes the software reuse become a crucial task. In this context, we believe that the proposed environment is relevant, because it incorporates the State project pattern to implement the FSM as well as a tool to UML state diagram of the UML. Related to AI techniques, the environment makes an abstraction of them and generates source code to several languages (multiplatform).

As further work, we will validate the proposed environment. To do that, basing the FSG, NPC will be implemented in the Neverwinter Nights game by BioWare, based on the work of Zhao et. al [24].

References

[1] Bourg, D. M., Seeman, G. (2004). AI for Game Developers. p. 400, O'Reilly.

[2] Smed, J., Hakonen, H. (2006). Algorithms and Networkinf for Computer Games. p. 286, John Wiley & Sons Ltd, University of Turku, Finland.

[3] Bittencourt, G. (2006). Inteligência Artificial - Ferramentas e Teorias. 3rd ed. p. 371, ISBN 8532801382. Universidade Federal de Santa Catarina, Florianópolis, Brazil.

[4] Oliveira, K. S., Mattos, H. D. (2006). Abordagens de Reuso de Software no Desenvolvimento de Aplicações Orientadas a Objetos. XII Escola Regional de Informática. Faculdades Luiz Meneghel (FFALM), Curitiba, Brazil.

[5] Weschter, E. O. (2008). Arquitetura do Gerador de Aplicaes Web Baseado no Framework TITAN. p. 93, Universidade Federal do Mato Grosso do Sul, Campo Grande, Brazil.

[6] Sommerville, I. (2007). Software Engineering. 8th. ed. 549 p. Addison-Wesley.

[7] Bittencourt, J. R., Osório, F. (2001). ANNEF - Artificial Neural Networks Framework: Uma Solução Software Livre para o Desenvolvimento, Ensino e Pesquisa de Aplicações de Inteligência Artificial Multiplataforma. p. 2,13–16. Free SoftwareWorkShop, Porto Alegre, Brazil.

[8] Medeiros, F. N., Medeiros, F. M., Domnguez, A. H. (2006). FA PorT: Um Framework para Sistemas Portfólio-Tutor utilizando Agentes. p. 18, 08–10. Symposium on Computers in Education.

[9] Ferreira, F. M. G. (2005). Desenvolvimento e Aplicações de um Framework Orientado a Objetos para Análise Dinâmica de Linhas de Ancoragem de Risers. p. 109. Universidade Federal de Alagoas, Alagoas, Brazil.

[10] Gamma, E., Helm, R., Johnson. R. (2005). Design Patterns - Elements of Reusable Object Oriented Software. p. 416. Bookman.

[11] Rabin, S. (2002). AI Game ProgrammingWisdom. 1st ed. p. 672. Charles River Media.

[12] Alberto, A. D. B. (2009). Uma estratégia para a minimização de máquinas de estados finitos parciais. p. 99, Universidade de São Paulo. São Paulo, Brazil.

[13] Dinízio, C. S., Simões, M. A. C. (2003). Inteligência Artificial em Jogos de Tiro em Primeira Pessoa. Universidade de Tiradentes. Revista de Iniciação Científica. .Minas Gerais, Brazil.

[14] Malfatti, S. M., Fraga, L. M. (2006). Utilizando Behaviors Para o Gerenciamento da Máquina de Estados em Jogos Desenvolvidos com Java 3D. Brazilian Symposium Games.

[15] Larman, G. (2004). Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development. p. 507, Bookman.

[16] Booch, G., Rumbaugh, J., Jacobson, I.(2000).The Unified Modeling Language User Guide. p.576, Addison-Wesley Professional.

[17] Freeman, E., Robson, E., Bates, B., Sierra, K. (2004). Head First Design Patterns. p. 688, O'Reilly Media.

[18] Madsen, C. A. B. C. W., Adamatti, D. F. (2011).Using Artificial Intelligence in Computational Games. *Journal of Information & System Management*, 1 (2) 60–67.

[19] Russel, S. (2004). Inteligência Artificial. 2nd ed. p.1056. Edit. Campus, Rio de Janeiro - RJ, Brazil.

[20] Haykin, S. (2001). Redes Neurais Princípios e Práticas. 2nd ed. p. 900, Edit. Bookman, Higienópulis - SP, Brazil

[21] Rezende, S. O. (2005). Sistemas Inteligentes Fundamentos e Aplicações. 1st ed. p. 370. Edit. Manole Ltda, Barueri - SP, Brazil.

[22] Linden, R. (2008). *Algoritmos Genéticos - Uma Importante Ferramenta da Inteligência Computacional*, 2nd ed, p. 400 ,Basil: Brasport.

[23] Hartigan, J. A., Wong, M. A. (1979). *A K-Means Clustering Algorithm*, J*ournal of the Royal Statistical Society. Series C (Applied Statistics)*, 28 (1) 100-108, England: Blackwell.

[24] Wenfeng, H., Qiang, Z., Yaqin, M. (2011). Component-BasedHierarchical StateMachine - A Reusable and Flexible Game AI Technology, *In*: Information Technology and Artificial Intelligence Conference. 2, 319-324.