Semiautomatic edition of behaviours in videogames *

Gonzalo Flórez-Puga and Belén Díaz-Agudo
Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
email: gflorez@fdi.ucm.es, belend@sip.ucm.es

Abstract

The edition of intelligent behaviours in games is not an easy task. Amongst other activities, it implies identifying the entities which must behave intelligently, and what kind of behaviours they must show without being too predictable; designing and integrating these new behaviours with the virtual environment, in terms of perception and actuation over the environment, and implementing them. In this paper we present an ongoing work using Case Based Reasoning (CBR) to design intelligent behaviours in videogames. We have developed a graphical editor based on hierarchical state machines that includes a CBR module to retrieve and reuse stored behaviours. The editor and the CBR module are generic and reusable for different games. We have tested our module on a soccer simulation environment (SoccerBots) to control the behaviour of the soccer players.

Keyword: Intelligent Agents, Behaviours in Games, State Machines, CBR

1. Introduction

The aim of almost any game is to provide some amusement to the player. This task can be performed in several ways. In the particular case of computer games, besides a good story or spectacular graphics, the game must be a real challenge for the player. An appropriate method for achieving this is by providing the opponents (and the allies) of the player with intelligence [1].

The edition of intelligent behaviours in games or simulation environments is a difficult task. Amongst other activities, it implies identifying the entities which must behave intelligently, and what kind of behaviours they must show (e.g. helping, aggressive, elusive), designing and implementing them, integrating them in the game and testing.

Designing new behaviours could be greatly benefited from two features that are common in most of nowadays videogames. First of all, modularity in behaviours. That means that complex behaviours can be decomposed into simpler ones, that are somehow combined. Second, and related with the former, simpler behaviours tend to recur within complex behaviours of the same game, or even in different games of the same genre. Both features are useful to build new complex behaviours based on simple behaviours as the building blocks that can be reused.

In the ongoing work described in this paper we are developing a graphical behaviour editor that is able to store, index and reuse behaviours previously designed. Our editor (eCo) is generic and applicable to

* Supported by the Spanish Committee of Science & Technology (TIN2006-15140-C03-02)

© BCS Specialist Group on Artificial Intelligence

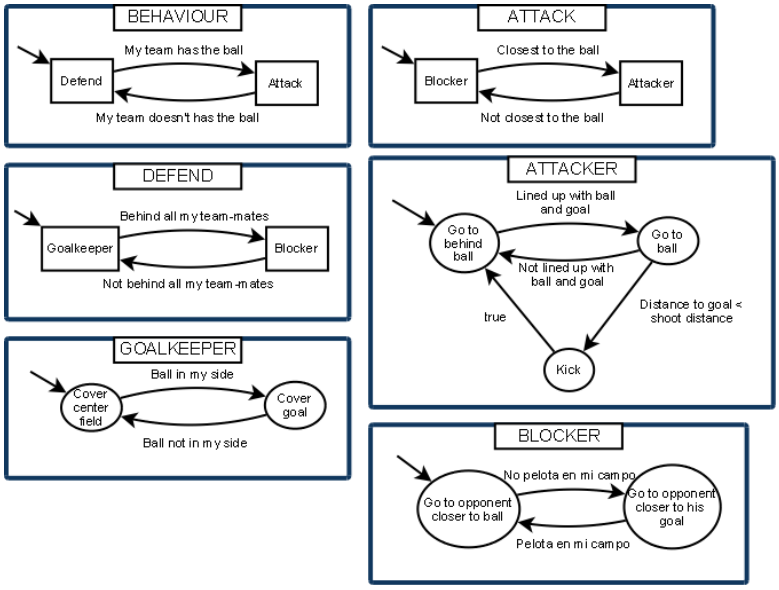

Figure 1. Example of HFSM

different games, as long as it is configured by a game model file. The underlying technologies of eCo are Hierarchical Finite State Machines (HFSMs) [2] and Case Based Reasoning (CBR).

HFSMs are appropriate and useful tools to graphically represent behaviours in games, facilitating the modular decomposition of complex behaviours into simpler ones and the reuse of simple behaviours. The eCo behaviour editor provides a graphical interface which allows the user to create or modify behaviours just by "drawing" them. On the other hand, by means of a CBR-based module, the user can make approximate searches against a case base of previously edited behaviours. Both technologies work tightly integrated. Initially, the case base is empty, so all the editing has to be done via the manual editing (graphic) tools. Once there are enough cases in the case base, new behaviours can be constructed by retrieving and adapting the stored ones.

There exist several tools oriented towards the edition of finite state machines. Most of them are general purpose state machine editors that don't allow the use of HFSMs, nor facilitates the reusing. Regarding game editors, most of them are only applicable to one game or game engina (e.g. Valve Hammer Editor). Besides, the vast majority only allow map edition. The few that allow editing the behaviours, are usually script based.

Finally, there exist some tools like BrainFrame and its later version, Simbionic, which are game oriented finite state machine editors. These editors allow the specification of the set of sensors and actuators for any game. There are two crucial differences wtih our approach. First of all, the Simbionic editor doesn't offer any assistance for reusing the behaviours, like the CBR approximate search engine integrated into the eCo editor. And second, to integrate a behaviour edited with the Simbionic editor with a game, it is mandatory to integrate the Simbionic runtime engine with the game.

In section 2 we introduce some general ideas on behaviour edition. In section 3 we present the eCo behaviour editor, and in section 4 we show a small example of application of the editor to a simulation environment: SoccerBots. Section 5 describes the CBR module integrated in the editor. As the editor and

© BCS Specialist Group on Artificial Intelligence

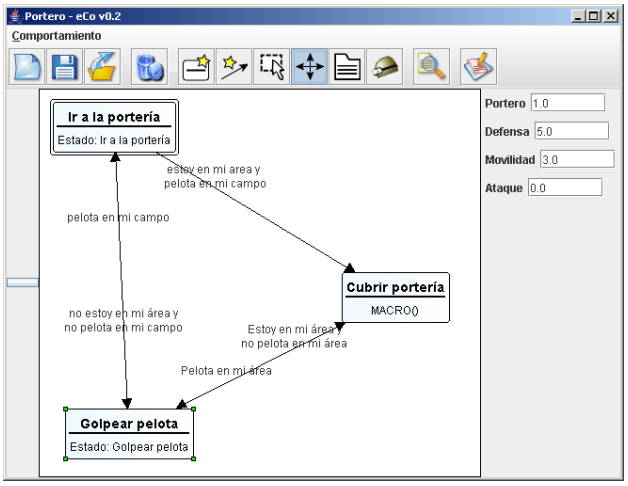

Figure 2. The eCo behaviour editor

the CBR module are reusable through different environments, in section 6 we outline the integration of the editor with different games and simulation environments. Finally, in section 7 and 8, we present related work, future goals and conclusions.

2. Behaviour editing in simulation environments

Each behaviour is typically defined by means of a set of actions or reactions performed by an entity, usually in relation with its environment. In a computer game or simulation, each entity gathers information about its environment using a set of *sensors*, which could be compared to the senses of the living beings. Depending on this information, the entity performs certain actions, using a set of *actuators*. In general, is different for each game or simulation environment, although there will be similarities between games of the same genre. For instance, commonly used sensors in a first-person-shooter (FPS) game could be the position, the health or the visibility of other entities. Regarding the actuators, the entity can shoot, look at or go to a place, talk to other entities, among others.

There are two properties, shown by game behaviours, which have been of critical importance for the development of the editor prototype: m*odularity* (complex behaviours are usually composed of simpler behaviours) and *reuse* (simpler behaviours tend to recur in complex behaviours).

Several suitable techniques exist for the representation of behaviours. Due to its expressive power and simplicity, the Finite State Machines (FSMs) is one of the most widespread of them. One of the drawbacks of the FSMs is that they can be very complex when the number of states begins to grow. To prevent this we used Hierarchical Finite State Machines (HFSMs), which are an extension to the classic FSMs. In a HFSM (like the one shown in Figure 1), besides a set of actions, the states can contain a complete HFSM, reducing the overall complexity and favouring its legibility [2]. Each HFSM can be considered as an abstract, modular component, which can be used anywhere in the hierarchy. FSMs have been used successfully in commercial games (e.g. Quake [3]), and in game editing tools (e.g. Simbionic [4]). Representation of behaviours using HFSM is very suitable to be used within a CBR system. Next we describe the basic working aspects of the eCo editor, and an example of its use in the SoccerBots simulation environment. Section 5 describes the CBR system.

© BCS Specialist Group on Artificial Intelligence

3. The eCo Behaviour Editor

The eCo Behaviour Editor (Figure 2) is a graphical editing tool which uses HFSMs to represent behaviours, allowing the user to "draw" the behaviour he wants to get. It also is able to automatically generate the code to execute the behaviour. The editor is strongly dependant on a CBR module which allows reusing behaviours previously edited. The design of the editor worked towards the achievement of three objectives, namely:

- Easiness of use: the user shouldn't need any technical or architectural knowledge about the game. This is achieved by the use of HFSMs as an intermediate graphic format.
- Applicability: the editor must be able to generate behaviours for different games or simulations, regardless of its genre. To accomplish this goal, the editor can use different configuration files (called game models) and code generators, suitable for each specific game.
- Assistance to users: this goal is met reusing previously edited behaviours, via a CBR module. This module should be able to make approximate retrieving and adaptation of the behaviours.

In section 3.1 we describe the configuration files (game models). Section 3.2 deals with the manual edition of behaviours.

3.1 Defining the game models

A game model is a configuration file that describes some details of a game or a simulation environment. The game models allow the user to use the eCo editor in different games.

Each game model includes the information about sensors and actuators, and a set of descriptors. The sensors and actuators are obtained from the game API. Regarding the descriptors, they are numeric or symbolic attributes that will be used by the CBR module to describe the behaviours and retrieve them from the case base. The descriptors are obtained through the observation of the characteristics of the different behaviours that exist in the domain of the game and must be enough extensive and representative to describe most of the behaviours we can come across for that particular game.

3.2 Editing behaviours, generating code and storing cases

The eCo editor provides a set of editing tools that allows the user to create behaviours from scratch or from previously edited behaviours stored in disk.

Once the behaviour is complete, it is possible to use the code generation tool to generate the source code corresponding to the behaviour. This tool uses the structure of the state machine together with the information in the game model to obtain the source file. As the game model and the source file required are usually different for each game, the code generator will also be unique for each game. The saving tool also allows the user to store the behaviour being edited in the case base for later reusing. We have used XML files to store the case bases. Each case is described by an attribute-value set of descriptors:

- Attributes: numeric and symbolic parameters that describe different properties of the behaviour. The attributes are different for each game, although similar games (e.g. games of the same genre) will share similar attribute sets.

© BCS Specialist Group on Artificial Intelligence

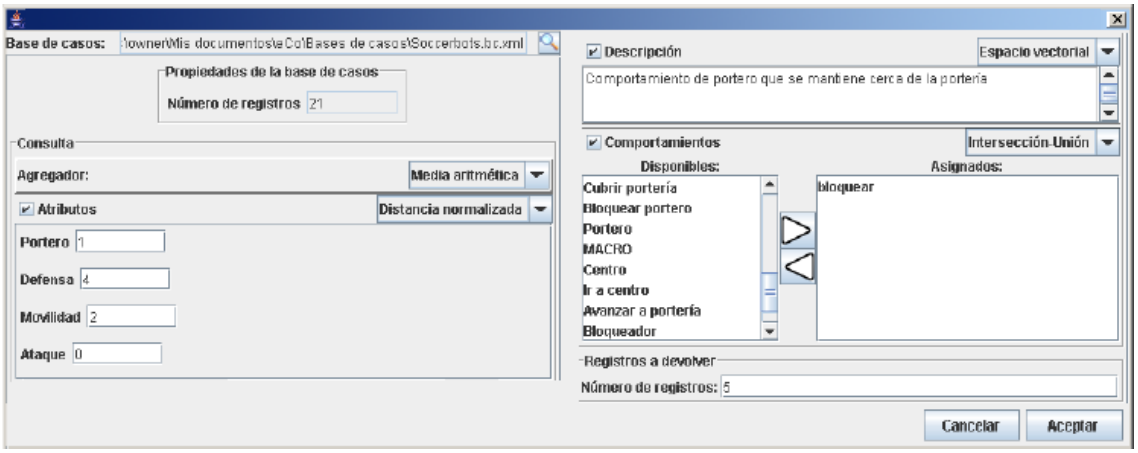

Figure 3. Functionality based queries editor

- Description: textual description of the behaviour. It serves a double purpose: the user can use it to fine tune the description given by the numeric and symbolic attributes, and it is shown to the user during the retrieval phase, so he can select the most appropriate case.
- Enclosed behaviours: specifies which behaviours are hierarchically subordinated. This allows the user to retrieve behaviours which include a specific set of sub-behaviours or actuators.

4. SoccerBots Example

The behaviour editor described in Section 3, and the CBR system that we are describing in section 5, are independent of any specific game. However, for the sake of an easier exposition we are explaining the basic ideas using a simple game. SoccerBots is a simulation environment developed by Tucker Balch, where two teams play in a soccer match. Simulation time, behaviour of robots, colours, size of field, and many other features are configured from a text file. Basically, rules are similar to those from Robocup.

The first step in using eCo to generate behaviours for the SoccerBots environment is to define the game model with the information about sensors, actuators and CBR descriptors of the SoccerBots simulation environment. In the SoccerBots API we can find sensors like `getBallX`, which checks the X, position of the ball, and actuators (i.e. actions that robots can take) like `setSteerHeading(int)`, which changes the direction the robot is facing.

As we stated before, the descriptors are obtained through the observation of the characteristics of the different possible behaviours. We used four numeric parameters to describe SoccerBots behaviours:

- Mobility: ability to move all over the playfield.
- Attack: ability of the robot to play as an attacker.
- Defence: ability of the robot to play as a defender.
- Goalkeeper: ability of the robot to cover the goal.

© BCS Specialist Group on Artificial Intelligence

5. The CBR system

The CBR system takes advantage of the modularity and reuse properties of the behaviours; it assists the user in the reuse of behaviours by allowing her to query a case base. Each case of the case base represents a behaviour. By means of these queries, the user can make an approximate retrieval of behaviours previously edited, which will have similar characteristics. The retrieved behaviours can be reused, modified and combined to get the required behaviours.

Initially, the case base is empty, so all the editing has to be done via the manual editing (graphic) tools. Once there are enough cases in the case base, new behaviours can be constructed by retrieving and adapting the stored ones.

There are two kinds of queries: functionality based queries and structure based queries. In the former, the user provides a set of parameters to specify the desired functionality for the retrieved behaviour. In the latter, a behaviour is retrieved, whose composition of nodes and edges is similar to the one specified by the query. In the current version of the editor, only functionally based retrieval is possible.

5.1 Functionality based retrieval

The most common usage of the CBR system is that the user wants to obtain a behaviour similar to query in terms of its functionality. The functionality is expressed by a set of parameters, which can be any (or all) of the descriptors of the cases presented in section 3.2 (i.e. the attributes, the textual description and the enclosed behaviours). The parameters that form the query are used to describe the behaviour, and are closely related to the game model. The more differences exist between two games, the more different the associated behaviours are and, hence, the parameters used to describe them. The eCo editor provides a query form, showed in Figure 3, for the users to enter the parameters of the query.

To obtain the global similarity value between the cases and the query, the similarity of the numeric and symbolic attributes is aggregated with the similarity due to the textual description of each behaviour. The user can select the most appropriate operator to combine them in the query form. Some examples of operators could be the arithmetic (used in this example) and the geometric mean or the maximum. Figure 3 shows an example query for the SoccerBots environment with the following parameters:

Goalkeeper	1	Attack	2	Description	Goalkeeper behaviour that stays near the goal
Mobility	4	Defence	0	Enclosed behaviours	Block

5.2 Descriptor based similarity

Using the aforementioned form the user can enter the query descriptor values and select the similarity measure used to compare them to the ones in the cases of the case base. To obtain the similarity value between two descriptors, we use the normalized difference value.

In the following table we show an example of the calculus of the similarity measure for the query in Figure 3 and a hypothetical case:

Descriptor	Range	Query	Case	Similarity
Goalkeeper	[0, 1]	1	1	1
Mobility	[0, 5]	4	2	0.6

© BCS Specialist Group on Artificial Intelligence

Attack	[0, 5]	2	3	0.8
Defence	[0, 5]	0	5	0

5.3 Textual based similarity

Description of behaviours by means of a detailed vector of descriptors can be cumbersome and difficult. It would result in excessively long descriptions. Furthermore, it is difficult to identify all them. However is useful to have this descriptors as indexes to filter and select cases.

To make the querying process easier, the user can use a textual description to fine tune the query by including in it characteristics not considered by the attributes. For instance, in the example, the user is requesting a behaviour that stays near the goal. There is no specific descriptor in the game model, as it is not relevant for most of the behaviours. Instead, the textual description is used. In the current version, we use the vector space model [5] to compute the similarity measure between the text descriptions.

6. Integration with other games

JV^2M [6] is a third-person action game conceived to teach the operation of the Java Virtual Machine (JVM). The game takes place in a space station, which acts as a metaphor of the JVM. The development of JV^2M is currently in progress and the set of sensors and actuators is not defined, so we had to sketch a sensory model, based in the model of the game FarCry.

Neverwinter Nights is a role playing computer game that takes place in the Dungeons & Dragons universe. It includes the Aurora Toolset, which allows scripting the NPC's behaviours. To carry out the integration, we have used RCEI (Remote Controlled Environments Interface), a protocol conceived to communicate a virtual environment with a remote controller application, via ASCII sockets.

Finally, we tested the editor with an AIBO pet, a multipurpose robotic pet. The code controlling the AIBO was built over the library URBI (Universal Real-time Behaviour Interface), which allows controlling the robot remotely, via a wireless connection.

In summary, we have tested the integration in environments with very different nature (a sport simulator, a role playing game, an action game and a real life multipurpose robot) and with different integrating characteristics. For instance, while in JV^2M we define the set of sensors and actuators, it is fixed for the other environments; while Neverwinter Nights is highly event-oriented, the rest of the environments are basically reactive systems.

7. Conclusions and Future Work

In this paper we have described an ongoing work using Case Based Reasoning (CBR) to design intelligent behaviours in videogames. We have developed a graphical editor based on hierarchical state machines that includes a CBR module to retrieve and reuse stored behaviours. One of the main advantages of our approach is that the editor and the CBR module are generic and reusable for different games. We have shown the applicability in a soccer simulator environment (SoccerBots) and we are working in applying our editor to JV2M, a third-person action game conceived to teach the operation of the Java Virtual Machine, that is currently being developed by our research group.

© BCS Specialist Group on Artificial Intelligence

The eCo behaviour editor is easy to use and offers a friendly interface based on a well known technique typically used to represent behaviours: HFSMs. The editor assists the user in the definition of new behaviours through a CBR module that retrieves previously stored behaviours.

We have described the current state of the work but there are many open lines of work. We have finished the graphical editor, defined the structure of the cases and the game models, and we have been working on case representation, storage and similarity based retrieval. Current lines of work are structure based retrieval, more sophisticated similarity measures, automatic reuse of behaviours and learning.

The use of hiecharchical state machines offers many possibilities to reuse and combine pieces of behaviours within other, more complex, ones. We are working on the definition of an ontology on game genres to be able to reuse behaviours, vocabulary, sets of sensors and actuators and even game models between different games of the same genre.

There exist numerous techniques, besides HFSMs, to represent behaviours, like rule based systems, or HTNs, for instance. One of the opened investigation lines is the study of the pros and cons of each one of them and the possibility of combining some of them to create the behaviours

In the current version, the learning of the CBR system is totally user guided: the user indicates which cases must be stored in the case base and also introduce the values for the descriptors. The set of values for each descriptor is a very subjective matter, so it would be a good idea to automatize this process, or, if it is not possible, make the system suggest values using machine learning approaches.

Besides the functionality based queries, presented in this paper, we are working on queries based on the structure of nodes and edges of the state machine that define the behaviour. We are studying about graph similarity measures, like the ones presented in [7], the restrictions they involve (for instance, in the case representation), and the applicable adaptation techniques.

The CBR techniques presented in this paper can also be used in runtime, to retrieve behaviours based in the defined attributes and the state of the game or simulation environment. This is another open research line that is currently being developed [8].

8. References

[1] Michael Bowling, Johannes Fürnkranz, Thore Graepel, and Ron Musick. Machine learning and games. Mach Learn, 63:211–215, 2006.

[2] A. Girault, B. Lee, and E.A. Lee. Hierarchical finite state machines with multiple concurrency models. IEEE Transactions on Computer-Aided Design, 18(6):742–760, June 1999.

[3] Jason Brownlee. Finite State Machines. AI Depot. Available from http://ai-depot.com/FiniteStateMachines/FSM.html (accessed September 18, 2007)

[4] Daniel Fu and Ryan Houlette. Putting AI in entertainment: An AI authoring tool for simulation and games. IEEE Intelligent Systems, 17(4):81–84, 2002.

[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to Information Retrieval. Cambridge University Press, 2007.

[6] Pedro Pablo Gómez-Martín, Marco Antonio Gómez-Martín, and Pedro Antonio González-Calero. Javy: Virtual Environment for Case-Based Teaching of Java Virtual Machine. LNAI Volume 2773, subseries of LNCS, pages 906–913. Springer Berlin / Heidelberg, 2003.

© BCS Specialist Group on Artificial Intelligence

[7] Gonzalo Flórez Puga, María Belén Díaz Agudo, Pedro Antonio González Calero. “Experience Based Design Of Behaviours In Videogames”. Advances in Case Based Reasoning 5239. 180–194. Springer. Dresde, 2008.

[8] Gonzalo Flórez Puga, Marco Gómez Martín, Belén Díaz Agudo, Pedro A. González Calero. “Dynamic Expansion of Behaviour Trees”. Proceedings of the 4th AIIDE Conference. 36–41. AAAI Press. Stanford, 2008.

© BCS Specialist Group on Artificial Intelligence