THE <E-GAME> PROJECT: FACILITATING THE DEVELOPMENT OF EDUCATIONAL ADVENTURE GAMES

P. Moreno-Ger[1], I. Martínez-Ortiz[2], B. Fernández-Manjón[1]
[1]*Universidad Complutense de Madrid (Facultad de Informática)**
C/ Prof. García Santesmases sn, 28040, Madrid, Spain.
pablom@fdi.ucm.es , balta@sip.ucm.es

[2]*Centro de Estudios Superiores Felipe II*
C/ Capitán 39, 28300 Aranjuez, Madrid, Spain
imartinez@cesfelipesegundo.com

ABSTRACT

Game based learning can be seen as an interesting approach to learning, but the complexity of today's videogames demands from developers a robust foundation in a variety of technologies. On the other hand, the educational part of game based learning requires the active participation of field experts. The collaboration between those experts and game developers is always difficult. The <e-Game> project addresses this problem by offering an authoring environment for educational adventure games that does not demand a deep formation in Information and Communication Technologies. The author (an expert in a specific field) only needs to write documents that describe the contents of the videogame following the <e-Game> XML syntax and feed them to the engine. In turn, the engine produces a fully functional game from those documents.

KEYWORDS

e-learning, game-based learning, XML, Domain Specific Languages, e-Game.

1. INTRODUCTION

The limitations of typical web-based learning processes for exploratory or constructivist learning have caused a growing interest for alternative approaches to e-learning. Such approaches could include collaborative learning processes, blended-learning, complex AI tutors, highly interactive environment, etc. Among those approaches, we would like to point out the potential of game-based learning. Videogames and interactive simulations can provide engaging and motivational environments that, if exploited carefully, can provide grounding for alternative learning paradigms.

However, this is not an easy task. Developing such environments is a challenging endeavor, and adding valuable educational material will necessarily require the collaboration of field experts that, often, will not have a technical background. This paper presents the <e-Game> project and how it can facilitate the development of content-rich educational games by using previous experiences in developing content-rich applications using a document driven approach.

2. GAMES AND EDUCATION

Games are here; our students play them, and usually do it after spending huge amounts of money buying titles and gaming equipment. As of today, the videogame industry already generates more income than Hollywood's box office sales (Snider, 2002 and ESA, 2005) and the trend is towards further growth. This

fact alone should settle any possible discussion on whether they are motivational and attractive or not. But, can they really be used as a platform for learning processes?

First, we should consider whether they *should* be used for learning. Most schools have a fixed learning model in which teachers act as content *deliverers.* The teacher presents a set of isolated concepts (be it facts, theorems, formulas, etc.) and then the student's grasp of these concepts is evaluated. (Shaffer *et al*, 2005) reflect that these typical skill-and-drill learning models (based in artifact learning) do not provide "good" learning. In order to achieve deep and useful learning the learner must immerse in the domain to be learnt. Of course, such an immersion cannot be achieved using the traditional learning model with the teacher in the role of content deliverer. This immersion, to be effective, must be perceived by the learner directly. A different learning model based on direct experience (embodied learning) with a constructivist (Tobin and Tippins, 1993) approach can yield great results if applied carefully.

However, putting the learner in a real environment is often expensive and in many cases unpractical. Videogames are a perfect medium for this kind of immersion, because they can provide a fast and simplified experience in any domain. While playing a videogame, the learner projects herself onto the characters in the videogame and this produces an almost-direct constructivist learning experience. In addition, videogames are cheap, safe and only bounded by our imagination (e.g. imagine a learning process with real immersion in the field of astronautics: It wouldn't be cheap nor safe to train new recruits in the moon, while a videogame could be inspired in Saturn).

Nevertheless, it is not that simple. Creating good learning videogames can be a daunting task for a variety of reasons. We want to explore one of them: How to deliver high quality learning content with a videogame.

2.1 Developers and field experts: A cultural clash

If we want to produce "good" *educative* games, it is necessary to include experts in pedagogy and/or the field to be learnt (referred hereafter as *field experts*) into the development teams. However, bringing field experts directly into a game development team is not necessarily a good idea, producing situations in which the expert spends most of her effort in understanding the technologies, terminology and procedures of game development. We need means to allow field experts to provide their knowledge to the system without needing to know barely anything about the technology underneath. It is necessary to seek collaboration models and processes that facilitate the interaction between people with different backgrounds, formations and views of how the final product should behave.

2.2 Reducing barriers: Document driven approaches

Recent development processes involving technicians and field experts have been successful in using a document driven approach. In such projects, field experts build the applications themselves by writing documents using a fixed syntax. Ideally, that syntax should correspond to a language that is close to the domain being considered in order to facilitate the experts' task. Those documents are then fed to a compiler or an interpreter that, in turn, generates the application.

This development process implies a deep collaboration between field experts and developers, but clearly separates their roles and functions: developers help field experts to define the language and then develop the tool that understands that language. Field experts write the applications using that language.

The only skill that is required from field experts is a good understanding of the language, which, ideally, should be kept reasonably simple. The best way to achieve this is to keep the language closely tied to the application domain and thus, these languages are often referred in the literature as Domain Specific Languages (DSL) (Van Deursen *et al.* 2000). For more information about document oriented approaches to software development consult (Sierra *et al.* 2004).

3. THE <E-GAME> PROJECT

The <e-Game> research project is currently being developed by the authors as part of the broader <e-Aula> project. The objective is to apply the advantages of DSLs and document driven developments to facilitate the

direct development of educational games. Because the key idea is simplicity, we do not aim to propose a language that would permit the development of any kind of game: The field is so rich, that such a language would require the power (and complexity) of a modern programming language such as C++ or Java, probably flavoured with game-specific additions (like graphic-related commands).

We can simplify the process by restricting the genre and functionality of the games, which means we should try to focus on one single, well-defined genre. We consider that Adventure Games can be a valuable genre for learning. Studies like (Ju and Wagner, 1997) indicate its main traits and identify a clear bias for content instead of just plain action. Although high adrenaline action can be attractive and motivational, we must not forget that we are ultimately pursuing a learning process, and content will be fundamental. Moreover, this genre lived a golden age in the 90s, with many successful titles published by LucasArts™ and Sierra™. Those titles were not only very famous, but also very similar in structure and functionality. Of course, this fact has narrowed the traits of the genre, making all titles very similar and uniform. This uniformity will be very valuable for our objective of simplifying the development language, because all its commonalities can be assumed and abstracted from the language.

The following subsections describe the development of educative games following our approach and then we will discuss some specific traits of the design language itself.

3.1 Designing an adventure

The objective is to allow that author to build and execute the game without needing a previous background in programming. All he/she would need is basic knowledge on how to use a computer, open a text editor and a few notions of XML and the <e-Game> syntax (the language itself will be detailed later).

We should start presuming that the author (probably a teacher specialised in the corresponding field) already has a game concept prepared in the form of a graphical adventure with a style and a user interface similar to those found in classical adventure games. This game concept is probably a storyboard or script, where the author specifies game scenarios and adds content to them. This content includes objects, characters and conversations. The conversations include a certain degree of interactivity in the form of multiple-choice answers that affect the flow of the conversations. These conversations and the actions that the user performs should be associated with the content to be learnt. All actions and conversations have “outcomes”, and the achieved outcomes represent the state of the game (and of the learning process).

This is all the knowledge about game mechanics that the author should need to know. There is a bit more of sugar in the syntax, mostly related with artwork. However, the artworks are decoupled from the general syntax. This allows the author to fully write the game and then ask for the help of a designer/musician or use available art assets.

3.2 Turning a document into a game

After the previous stage, the available assets are a set of documents describing the game with all its content and a few external files containing the artworks. Such external files are referenced by the <e-Game> documents.

Now the author only needs to run the <e-Game> software and indicate the game’s main file. The tool will analyze the document and all its dependencies (other <e-Game> documents or external files). If the syntax is correct and all the assets are correctly located, the document is parsed and loaded into memory. The outcome will be a game that can be directly executed in the <e-Game> environment.

4. THE <E-GAME> TECHNOLOGY

We have described the development cycle, insisting on how a simplified syntax will simplify the creation of games without requiring previous ICT background. The following sub-sections describe the nature and the syntax of the <e-Game> language and its engine.

4.1 Static models

The basic modeling units of the language are the “scenarios”. A scenario is a bounded gameplay area, defined in 2D. Typical examples would be a bedroom, a tavern or a street. These units have exits that lead to other “scenarios”. This allows us to create networks of these units, thus yielding the entire environment for the game.

However, scenarios are static: A background artwork and the exits leading to other scenarios. Although it is possible to include learning content in the scenarios (embedded in the artwork, for instance), that is not their purpose. Scenarios are populated with objects and characters. From a learning perspective, these objects and characters are the interface that delivers most of the learning content. The actual content is delivered by the possible interactions between the learner and those objects and characters.

An object can be examined roughly (a brief description), examined in detail (a more detailed description), combined with another object or given to another character. On the other hand, interactions with characters include examination, offering objects or, most important of all, conversations.

Conversations are dialogues between a character and the learner’s avatar. Usually, the learner chooses a phrase from a fixed list (defined by the author for each conversation) and the character answers with a corresponding response. This will often allow the player to answer by choosing from another selection of phrases, thus engaging in an interactive intercourse that provides a sense of freedom and learner involvement without compromising the author’s capacity to manage the possible outcomes of the conversation.

4.2 Narrative scripting: Game state

However, this is too limiting. A number of freely navigable scenarios with characters and objects scattered around may contain all the desired content but yield no learning at all. Although free will and decision capabilities are necessary in order to achieve a motivational environment, they should be limited. An absolute lack of order could lead to incoherency and could make it difficult to perceive any progress in the game. We must give a sense of narrative coherence to the succession of events.

We can achieve this by introducing a notion of “state”. All the actions that we perform in the game should be able to affect future actions. Some objects may be hidden until something happens (the object appears only if the learner has performed action X) or some exits may be locked (you can’t enter the museum until you buy a ticket or talk the janitor into letting you in).

This can be conceptually modelled by assigning each interaction (with an object or character) an identifier. We can then add preconditions to anything we want in the form of a list of identifiers that indicates which actions must have already been performed. The state in any given point of the game is the set of actions that have already been performed.

4.3 Using XML for game design

Traditionally, game development programming comes in two flavours: High-level programming languages (C, Java…) and scripting languages for specific tasks. Actually, it is common to find games where there is a hard-coded game engine and the game itself is programmed using script files. It can be argued that this kind of development process is very similar to our approach.

However, those scripting languages are designed to make things easier for the game developer or content creator. Those are usually skilled programmers and the scripting language are usually very complex. The main objectives are richness, flexibility and expressivity. Unlike scripting languages, we do not want to model functionality. Our focus is on content. In addition, we want the syntax to be very simple and intuitive in order to be easily reachable by field experts without programming background.

These needs suggest that XML technologies could be an interesting candidate. XML is focused on describing hierarchically structured content, which is closer to our needs than a full programming language. Also, XML is widely used by the e-learning community in their standardization processes and learning models (Fernández-Manjón and Sancho, 2004).

XML even provides mechanisms such as DTD and XML-Schema that allow the formalization of the language in a machine-readable way. This facilitates enormously the construction of the <e-Game> system. It also facilitates the author’s task by providing means to verify the correctness of the documents before feeding

them to the system. Besides, the wide acceptance of XML means that there is an assortment of commercial products supporting this kind of authoring process.

Finally, XML also provides mechanisms for identification and reference of elements. This allows, for instance, to model scenarios, characters and conversations separately and then link them using references (fig. 1). In addition, this functionality gives a straightforward solution to the previously mentioned issue of modelling the state.

```
<scenario id="SCE-1">
  <character-ref idRef="CHA-1">
    <position>...</position>
     ...
  </character>
   ...
</scenario>
<character id="CHA-1">
  <conversation-ref idRef="CON-1" />
   ...
</character>
<conversation id="CON-1">
   ...
</conversation>
```

Figure 1: A simplified XML excerpt showing the basic structure of the language. All elements (scenarios, characters, objects, conversations, etc.) can be defined separately and then they can be cross-referenced using the *id* and *idRef* attributes. In the example, the conversation CON-1 is triggered when speaking with the character CHA-1 who appears in the scenario SCE-1

4.4 The <e-Game> engine

The <e-Game> engine is used to execute the games from their describing <e-Game> documents. This processor has been designed to facilitate its maintenance and evolution. As depicted in Figure 2, it is designed in terms of:

- A tree builder. This element is based on a standard DOM parser and it builds a tree representation of the input <e-Game> document.
- A component repository. This repository contains a set of game components which are instantiated according to the documents and assembled for the production of the final videogame.
- A game generator, which is the core of the engine and which processes the document tree to assemble the game components.

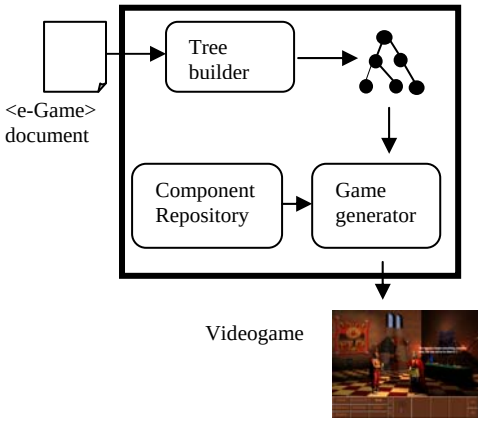

Figure 2: High-level architecture of the <e-Game> engine

5. CONCLUSIONS AND RELATED WORK

Game-based learning offers a very rich field that has been exponentially getting more attention following the exponential growth of the videogame industry during the last ten years. (Gee, 2005) provides a theoretical study of good learning principles potentially available in videogames, while (Virvou et al, 2005) puts together some practical notions. Important entities like ADL and the Department of Defence of the United States are exploring the feasibility of the model (Bonk and Dennen, 2005). And yet, even though there are a few serious initiatives, most initiatives have found limited success.

One of the reasons for this is the difficulty of getting field experts involved in the development. Our project suggests a development process in which game developers and field experts collaborate and in which both groups can focus their efforts in the areas in which they are more proficient.

Currently, the <e-Game> project has been used successfully to develop a number of small educational videogames, mainly with testing purposes. Although the tool is functional, the methodology still forces the author to create documents directly using the <e-Game> syntax, which may still be a drawback for some authors. In the near future, it will be necessary to develop the tools to facilitate even more the authoring process by automating some of the tasks involved in writing the documents.

It must also be noted that, as of today, the system is rather limited and produces games that are appealing when compared to previous experiences, but still a few steps behind the current state-of-the-art in the videogame market. Nevertheless, the separation of roles and the modular architecture make it is possible for developers to further improve the quality of the game without affecting the processes that authors must follow (hence this means higher game quality without increasing complexity for field experts).

ACKNOWLEDGEMENTS

The Spanish Committee of Science and Technology (TIC2001-1462 and TIN2004-08367-C02-02) has partially supported this work. The Spanish National Center of Information and Educative Communication (CNICE) provided the art assets displayed in the examples.

REFERENCES

Bonk, C. J. and Dennen, V.P. 2005. Massive Multiplayer Online Gaming: A Research Framework for Military Training and Education. *Technical Report 2005-1, Department of Defence of the U.S.A.*

ESA, 2005. *Entertainment Software Association, Sales & Genre Data.* Retrieved online April 2, 2005 from http://www.theesa.com/facts/sales_genre_data.php

Fernández-Manjón, B. and Sancho, P. 2004. Creating Cost-effective Adaptative Educational Hypermedia Based on Markup Technologies and E-Learning Standards. *Interactive Educational Multimedia* Vol. 2, pp.1-11.

Gee, J.P. 2003. *What Videogames have to teach us about learning and literacy?* MacMillan, New York.

Ju, E. and Wagner, C., 1997. Personal computer adventure games: Their structure, principles and applicability for training. *The Database for Advances in Information Systems* Vol. 28, No. 2, pp. 78-92.

Sierra, J.L. et al, 2004. ADDS: A Document Oriented Approach for Application Development. *Journal of Universal Computer Science.* Vol. 10, No. 9, pp. 1302-1324.

Shaffer, D. et al, 2005. Videogames and the future of learning (In submission).

Snider, M., 2002. Where movies end, games begin. *USA Today*, May 23. Retrieved online April 2, 2005 from http://www.usatoday.com/life/cyber/tech/review/2002/5/23/e3.htm

Tobin, K. and Tippins, D. 1993. Constructivism as a Referent for Teaching and Learning. In: K. Tobin (Ed) *The Practice of Constructivism in Science Education*, pp 3-21, Lawrence-Erlbaum, Hillsdale, NJ.

Van Deursen, A. et al 2000. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices* Vol. 35, No. 6, pp. 26-36.

Virvou, M. et al, 2005. Combining Software Games with Education: Evaluation of its Educational Efectiveness. *Educational Technology & Society* Vol. 8, No. 2, pp. 54-65.