Development of an API to Create Interactive Storytelling Systems

Enrique Larios[1], Jesús Savage[1], José Larios[1], Rocío Ruiz[2]

[1] Laboratorio de Interfaces Inteligentes
National University of Mexico, School of Engineering, Valdez Vallejo Building, Ciudad Universitaria, Mexico City, Mexico
bernulli@yahoo.com, savage@servidor.unam.mx, joselarios@lycos.com
[2] Instituto Tecnológico de Estudios Superiores Monterrey, Campus Ciudad de México, Mexico City, México
caruiz@itesm.mx

Abstract. Interactive storytelling (*IS*) is an incipient field that has not been completely formalized yet. There is still a significant amount of research to be done. Especially regarding topics such as how much freedom can be granted to the user to modify the story and what are the effects this freedom will have in the quality of the story. In this paper it is presented an object-oriented expandable API that provides many of the low-level functionality requirements of an interactive storytelling system, so research regarding such topics can be done without worrying about the implementation details. The API includes the capability to present a 3D graphic representation of the of the story world. It has a basic agent class, which already provides NURB-curve-based navigation, bone-based animation and text-to-speech capabilities.

1 Introduction

An Interactive Storytelling System (*ISS*) is a software system that stores all the information regarding the virtual world where the story takes place. It includes the story's objects and characters; the *ISS* also simulates the evolution of the events that form the story itself, with diverse mechanisms and with different degrees of freedom. It allows the user to modify the story's course of the events. This is a capability necessary for a system in order to be called interactive. Another important distinction in an *ISS* is the way the story they told is created. Some systems generate a completely new story every time, while other systems may already have the basic line of development that the story events will follow, and only do small changes on it depending on the user input.

In the second type of systems, a content creator makes the general outline of events before the system is used, and it is usually not very flexible to the user input. If it gives a high amount of freedom to the user, it would had to anticipate and cover many branches that the story could take, but if the user actions have little or no effect in the final outcome of the story then the system could hardly be called interactive. On the

other hand, the systems that generate the story by themselves have to take into account the user actions and some kind of parameter set to indicate the mood, the tone and the pace of the story. Often, the problem with this kind of approach is the lack of dramatic and entertaining quality of the generated stories.

Additionally, the cost of development of an *ISS* is usually very high, because of the developers' lack of experience in this area, the short of proven design patterns and the different disciplines that take part in this kind of project. These problems can cause the scattering of efforts in several areas, like artificial intelligence, computer graphics, animation, text-to-speech conversion and voice recognition. After building these basic systems the developers have to deal with their integration with the story system that they developed. The complexity of the system that has to be built in order to test new theories is one of the main problems that affect *IS* development and research.

The purpose of this project is to provide an API for the developers of ISS that includes the basic functionality that would allow them to develop their systems easily. The use of this API would render in further research of the real issues of IS. The API was tested in the development of a virtual reality environment that represents the Mayan archeological site of Calakmul, see figure 1.

Fig. 1. Calakmul virtual environment.

The purpose of this project is to allow the users of the system to visit, virtually, a newly discovered Mayan archeological site in the state of Campeche, Mexico. Through the integration and use of the Nebula game engine [1] in the API and the help of a tourist guide controlled by the API agent class, the system lets the users to know the archeological site that is located in an almost inaccessible area.

2 Types of Interactive Storytelling Models

Different Artificial Intelligence approaches have been applied to the implementation of *ISS*, but all of them are restricted in one way or another by the interaction model the *ISS* uses. Because depending on the way the user is allowed to interact with the story and the way the story molds itself depending it, the AI methods applied may

vary greatly. In some models, the system has to respond by simply selecting a branch of the story that was already built up, depending on very few significant user actions. While in others, the agents that control the story's characters have a specific AI implementation that tries to make their reactions look real while generating a story that can be modified by the user actions. It is also necessary to state that there are several different models in between these two. This great variety of completely different systems is one of the main reasons for the lack of standard design patterns in this field, and obviously, an obstacle for deep and thorough research on *IS*. In the next paragraphs, we will present some of the more common interactive storytelling models.

2.1 Linear Story with Branching Points

This is the simplest and the less interactive of the *IS* models. In these systems, the story develops in a linear fashion and only certain specific actions of the users affect the plot of the story, branching it from a limited set of selections. These systems sometimes have an upper layer of high interactivity, but the users actions in this layer do not have a significant impact on the story itself. Actually, these systems can hardly be called interactive, unless the level of branching covers every possible action the user may perform on every possible object in the story world. Obviously, this is very hard to do, because the number of branches would increase very fast to an almost infinite amount. Due to this problem, the number of important branches in any story that runs in this type of systems is always less than six. This model is usually applied in systems such as videogames and interactive software for kids.

2.2 Emergent Story from Intelligent Agents

In this type of model, the attention is placed on the way the characters behave based on their emotions and profiles. Most of the use of AI solutions in these systems are applied to the agents, giving sometimes an unpredictable result in the quality of the story, like Aylett points out in her soccer game analogy in [2]. It is not enough that all the characters that interact with the user have a realistic behavior in order to emerge an interesting story. For example, even if it were possible to create an AI system where the virtual characters acted completely natural, even if it were impossible to distinguish them from a real person, even then, there would be no guarantee that the generated story would be good. We simply have to look at real life; does every sequence of events that happen in real life have the dramatic or comic quality to really call it a story? Even in improvisational drama an abstract plot is given to the actors. It constrains and directs the actions of the character they represent. This means that developers of interactive systems using this model still have to face with the challenging problem of generating a good story at the abstract level.

A particular problem that arises with emergent stories, and that is shared with improvisational drama, is that the boring parts of the story aren't leaved out as authors usually do in traditional narrative between scenes. It can be argued that digital emergent stories have an advantage over improvisational drama, because they can flip onto the next scene computing the boring parts that happened between scenes, as pointed

out in [2]. With this solution, however, other difficult problems may arise. Like the need to create a system module in charge of selecting which element of the story that is emerging are dramatic and interesting enough to be presented to the user, and which are boring, mundane or irrelevant so they have to be computed off-screen.

2.3 Story-Driven Interaction

The sometimes uncertain quality of the story generated by intelligent agents has motivated other researchers, like Szilas [3], to propose interaction models where the generation of quality stories is emphasized. The proposal for this kind of systems states the need of a story engine where story would be generated taking into account the user actions, but also narrative and dramatic principles in order to ensure the story's quality. This is, admittedly, no easy feat. There is still no definitive solution for this problem, even though it seems like a step in the right direction. The problem with this model is the difficulty to represent and mold many of the abstract qualities that a good dramatic story needs. There is even no agreement on exactly what makes a good story. The problems of this approach are aggravated because of the lack of a definitive narrative theory; some of the theories that are commonly applied to interactive stories are the story functions proposed by Propp [4] or Barthes' structural analysis [5]. It is also necessary to point out that a story-driven interactive system based on these theories is very difficult to implement because these theories were not intended for a computer application, and also because they have a high level of abstraction.

As concluded by [6], most of the narrative theories conceive a story as something that was authored rather than something dynamically created, an approach which would be more suitable for the application of intelligent agents to the characters' implementation. Certainly, the dynamic approach would be easier because it would allow the use of more tested and researched agent's solutions.

3 An API for Interactive Storytelling

Because the current state of interactive storytelling and the great variety of disciplines needed to develop a complete *ISS*, we have developed an API that includes the most common and difficult to implement functions of an *ISS*. This API would help to the easier and faster development of *ISS*; it would also facilitate the research of new models and the confirmation of proposed theories in this field. Although it is possible to present the stories in text, or narrated through a text-to-speech system, the computer user is accustomed to a 3D interface where he can see the graphic representation of the characters and the story environment. Due to this, an important part of any API for *IS* should include a graphic engine to help the developer to entice the graphic-centered computer user. Around this engine, and using its software architecture the API was developed integrating the basic functionality that every *ISS* requires. Much of the effort in the building of this API was put in its expandability; the use of the object-oriented approach was part of this scheme to insure that the developer of interactive stories can use the API to build his own system.

3.1 Object-Oriented Design

As mentioned in the last section, the API should help the programmers to develop completely modular and fully expandable projects. The object-oriented design is the most natural and intuitive way to do it. The design pattern is as follows, there are two main kinds of classes: the story entities, objects that control the internal state and behavior of every element that exists in the story, and the servers, that are just software elements that implement a common function and share it with all the entities of the story. To help the developers to follow this pattern, the API already includes a base class that has the minimum functionality and the most common interfaces that every story entity, either character or object, needs. All the classes are based on the root class of the game engine (nRoot) and on the base class of the API (nIstEntiy), see figure 2.

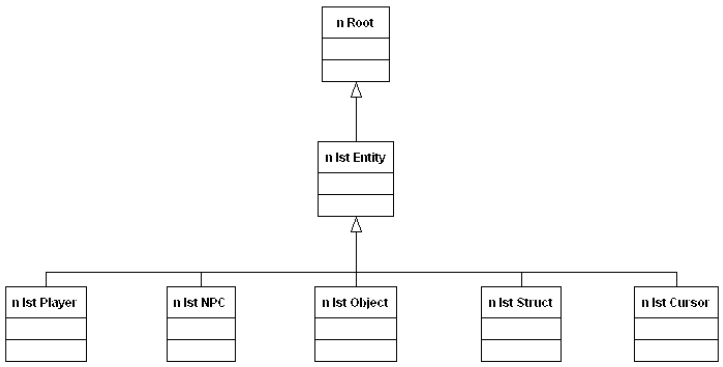

Fig. 2. Object hierarchy of the basic classes of the API.

Another design pattern promoted by the API is the division between the object that stores the status and the behavior of the object in the story world, and the one that stores its graphic representation. In this way, the API is ready for an easy change of game engine, foreseeing its obsolescence. Due to this division there is a class in the API to store the story entities and one to store its graphic representation. The world class is where all the story entities are stored and updated and the scene class is where all the graphic representations of the objects are stored and rendered.

The design pattern followed by the server class is intended to combine a set of related functions in a sole class. An instance of this class makes accessible its functionality to every entity that may need them. Classes like the time server, the graphic server and the voice server belong to this kind of classes, and are already implemented in the API. The developers that follow this design pattern can create new classes that control and generate the story, developing story servers like the ones mentioned in Section 2.3. Using all the included functionality and only developing a new server class which generates the story that all the story entities would follow is an example of the way we expect that our API would drive the development of new *ISS*.

3.2 Basic Physic Simulation System

As was already stated, in order to create a more inmersive experience, it is almost necessary that the *ISS* do a graphic representation of the events that happen in the story. A problem that arises when the events are presented this way is the need for physic simulation, because it would not be enough that the movements of the characters were controlled by the user or by the agents. If there is no physic simulation or collision detection, the user would be able to direct the character he is controlling against a wall, only to find that the character goes through it.

For that reason, the API also includes a complete physics' particle simulation system that ensures the realistic physical behavior of the objects in the story world. The API simulates gravity, inertia, friction and inelastic collisions. These are just a few simple physics laws, but more than enough to enhance the realism of the graphic representation of the virtual environment of the story.

3.3 High-Level Functionality for the Control of the Characters

Another class that the API includes is the actor class. This class is the one in charge of the actions of the story characters. And as a requirement for the developers' ease of use, all of this class functionality can be accessed through high-level commands. The purpose of the high-level access to the actor functionality is to allow inexperienced programmers to develop *ISS* through scripts, encouraging an increase in the number of artistic members in the *ISS* development groups. As Chris Crawford points out [7], part of the lack of new models and ideas in interactive storytelling is product of the lack of multidisciplinary groups that develop *ISS*.

The actor class is coupled with the 3D model class that supports the cycling and blending of bone-based animations, see figure 3. In this way, the designer can apply to several different models the same walk and run animation, while they blend it with another animation that controls the model's hands. The 3D model class is implemented over the open source Cal3D library [8], and supports models and animations exported from 3D Studio Max 4.2.

Fig. 3. The animation of the characters is bone-based in order to adapt one animation to several different models.

Often, the lack of artists and content creators in the development teams of *ISS* is caused by the difficulty they face to create the content. This difficulty is posed by the use of not fully developed design tools and the utilization of low-level languages to control the actions of the characters.

We expect that the high-level control offered by the API will help to the creation of a rule based system that will control some of the characters high level actions, while the details of those actions are controlled by the code of the API. For example, the API already has a basic navigation system based on NURB-curves. This feature of the API is meant to create a network of nodes representing the important points in a scene, which are communicated through the paths created by the NURB-curves. In this way, it is only necessary to develop an algorithm, like Dikjstra´s search, that selects the best path to reach the place the agent needs to go.

3.4 Wide Variety of Camera Styles

When someone is telling a story in a graphical way, it is not enough to show the events as they happen. In order to increment its dramatic quality, the camera plays a key role. Depending on the position and movement of the camera, the scene changes completely. The ideas and feelings that the author intends to transmit may be severely distorted by the position of the camera. Even the most epic or dramatic scene can be turned into a comedy with the right camera angle. For that reason the API also contains a very complete camera class. It lets the developers show their scene in the most appropriate way. The camera class has some automated functions that free the developer from having to control its movement every moment.

3.5 Text to Speech Conversion

Depending on the way the *ISS* is displayed, different forms of communication can be employed to inform the user, but its effectiveness may vary greatly depending on several factors. For example, if the *ISS* is displayed in a "cave", where the user is completely immersed and the developers have the intention of making the user to feel he is in some other place, then the use of text messages would be a bad choice. Because text messages would interfere with the cave display, making it look less real. Considering this possible use, we decided to integrate into the API the Microsoft Speech SDK that has text-to-speech capabilities, thus allowing the agents to say small phrases that already were written by the developer. These phrases can be converted into speech depending upon the developers' scripting.

3.6 Interactive Storytelling Models Suitable for the API

The functionality and the architecture of the API make it more suitable for some of the *IS* models mentioned early on. The API is completely suitable to implement the functionality required by a system using the linear-branching story model. In the case of the other two mentioned models of interaction, the API does not provide with all the functionality that an *ISS* implementing them requires. Nevertheless, its architec-

ture provides a foundation that covers the low-level actions that a virtual character requires. Allowing that all the required high-level functionality can be easily implemented and incorporated. In emergent story telling, a more advanced and abstract agent can be built on top of the existing functionality in order to obtain story emergence. The API is also suitable for an *ISS* that implements the story-driven model, although a whole story engine stills has to be built as a server on top of the API's low-level functionality.

3.7 Possible Roles for the Agents Created with this API

With the capabilities that the API grants to the agents, it enables them to behave like reactive agents. Their responses can be completely scripted. It also allows creating agents with a simple automaton that triggers scripts depending upon its state as their behavior and emotion control. Right now, the functionality implemented in the API enables the agents created with it to have a role as story participants, but not as creators or active modifiers of the story. For example, the API functionality was used to implement a virtual-tourist guide that answers to many possible questions about the archeological site of Calakmul.

4 Future Work: The Development of a Rule Based System

One of the most complex parts to develop of an *ISS* is the control of the characters. The API presents a script based control mechanism for every object. This feature is extended from game engine architecture to all the API classes. The scripting language is Tcl, which is flexible and powerful enough for most of the needs of the *ISS* developers. Only if the system requires real independent intelligent agents, then the kind control that Tcl scripting offers may not be suitable. To support those cases we are planning to include in the API a rule based system, using CLIPS [9] that would help enable the agents to plan and execute more complex sets of actions without the explicit programming from the developer. It would also help to create a better interpretation of the user actions through its knowledge-representation capacities, enabling the agents to react to complex situations and environments where context is important.

As stated earlier, the use of intelligent agents, would not by itself, improve the quality of the interactive story, but it would make the behavior of the characters more believable. It would depend upon the developers for the creation of a system that tells the user a real interesting story that changes accordingly to his actions.

Conclusion

The contribution of this project is to propose a basic framework and an API for the development of *ISS*, giving the developers certain basic functions and recommending a software architecture that would make its development easier. Moreover, this work

tries to promote the creation and use of common agreed concepts in the field of *IS*, because only then, most of the effort will be directed to the research and solution of the problems that affect this novel field.

There are still many unsolved issues that only focused and specialized research can discover. It is also very likely that only multidisciplinary development groups will find the right system for real interaction with quality stories. And although there are other more refined systems, like Gamebots [10], that use a game engine for research purposes, we believe that we have a distinctive characteristic in the application development process. Because all the code we used was open sourced, with the exception of the text to speech module from MS, developers can use it for any kind of application without worrying about paying any license fee.

References

1. http://www.radonlabs.de/nebula.html
2. Aylett, R.: Narrative in Virtual Environments - Towards Emergent Narrative. Narrative Intelligence Symposium, AAAI 1999 Fall Symposium Series (1999)
3. Szilas, N.: Interactive Drama on Computer: Beyond Linear Narrative. In AAAI Fall Symposium. AAAI Press, Menlo Park, CA (1999) 150-156
4. Propp, V. 1928: Morphology of the Folktale. University of Texas Press (1998)
5. Barthes, R. 1966: Communication 8, Introduction à l'analyse structurale des récits. Editions du Seuil (1981)
6. Louchart, S., Aylett, R.: Narrative Theory and Emergent Interactive Narrative. The 2nd International Workshop on Narrative and Interactive Learning Environments, Edinburgh, Scotland (2002)
7. Crawford, C.: Artist and Engineers as Cats and Dogs: Implications for interactive storytelling. Computer Graphics 26, (2002) 1
8. http://cal3d.sourceforge.net/
9. http://www.ghg.net/clips/CLIPS.html
10. http://gamebots.sourceforge.net/