Proc. 7th International Conference on Computer Games (CGAMES ' 2005), CNBDI, Angouleme, France, pp. 135-139, Nov. 2005

TOWARDS PROCEDURAL CREATION OF CHARACTER BEHAVIORS IN AN INTERACTIVE DRAMA SYSTEM

Keisuke Tanaka and Ruck Thawonmas
Intelligent Computer Entertainment Laboratory
Department of Human and Computer Intelligence, Ritsumeikan University
Kusatsu, Shiga 525-8577, Japan
URL: www.ice.ci.ritsumei.ac.jp
E-mail: ruck@ci.ritsumei.ac.jp

KEYWORDS

Interactive storytelling, TVML application, contents authoring

ABSTRACT

This paper discusses the architecture of a new interactive drama system that facilitates creating of story contents and extending of the system itself. The key feature of this system is that one can procedurally create characters' animations using TVML scripts without any 3D modeling software. Also we show its prototype system that implements some of the required functions.

INTRODUCTION

Interactive drama is a new type of storytelling in which viewers can interact with the virtual world to affect the story. By making believable agents play the roles of the characters, such a system realizes more flexible story. Research on interactive drama has been recently very active, and some systems are being developed [1] [2] [3]. Facade [1] is a first-person-perspective system in which the viewer can participate as one of the characters. Mimesis [2] is also a first-person-perspective system. On the other hand, I-Storytelling is a third-person-perspective system to which the viewer can interact without playing one of the characters.

The system we are developing is a third-person-perspective system capable of interaction with character agents and objects through mouse clicking or keyboard typing. We are aiming to realize a multi-purpose interactive drama system that facilitates content creation and system extension.

An important issue in development of an interactive drama system is to adopt the developing discipline that commensurates with the size of the project. Some of interactive drama projects use game engines, rather than starting from scratch. Such game engines, such as UnrealEngine, normally have a large specification. Therefore, it is difficult to use them in a project consisting of a few members with moderate programming skills, such as our project. In addition, in existing interactive drama systems, the cost for creating story's content is high. Mimesis and I-Storytelling, mentioned above, use UnrealEngine, which is, however, demanding because an animation has to be created for each action of a character with 3D modeling software. Another problem in development with game engines is due to insufficient documents on the engines. Many MOD developers use unofficial documents spreading around user's community, but if one wants complete documents and supports, they need to pay huge amount of license fee.

An approach for solving these problems is to incorporate existing tools that have sufficient documents and only necessary functions. Adopting this approach, we develop an interactive drama system by using TVML and JSHOP2 as its components.

TVML

TVML [4] is a TV program making software and language developed by NHK Science and Technical Research Laboratories. By writing an acting script in TVML and playing it with the TVML player, a show is played in real-time rendered 3DCG. The behavior of cameras, props, and characters can be defined by TVML. The most important feature for our system discussed below is that the user can make animations in a procedural way with only TVML.

JSHOP2

JSHOP2 [5] is a Java implementation of SHOP2 [6] which is a Hierarchical Task Network (HTN) Planner [7] originally implemented in LISP. The HTN planner performs planning efficiently by decomposing abstract tasks into smaller and more concrete tasks called subtasks, and finally into directly performable tasks called primitive tasks. The set of those primitive tasks are performed by a set of corresponding operators.

By introducing TVML and JSHOP2 into our system, we aim to develop a multi-purpose interactive drama system that facilitates content creation and system extension.

The final research goal of the authors is to examine a number of planners, including our modified version of JSHOP2, for the interactive drama application. To achieve this goal, an interactive drama system has to be implemented so that such planners can be tested.

In this paper, we present the architecture of our interactive drama system, and introduce a prototype system that implements a part of required functions.

REQUIREMENTS AND ARCHITECTURE

Requirements

As mentioned above, some interactive drama systems are being realized [1] [2] [3]. However, there is no clear definition of interactive drama. Common understandings are roughly as follows:

- Characters are controlled by AI's.
- Viewers can interact with those characters and the virtual world in a certain way.
- The story change as a result of such interactions.

After setting the main aim at realizing these functions, we put them into shape and list necessary functions for being implemented in our system as follows:

- Characters move around in the computer-generated 3D virtual world.
- Characters talk to each other.
- Viewers can interact with those characters and objects in the virtual world with mouse clicking and keyboard typing.
- Creators of story's content can easily define and add new animations.

Architecture

We discuss here the architecture (cf. Fig. 1) of our interactive drama system that satisfies the requirements above.

Server

The major role of the server is to control the TVML components that visualize the story. The TVML player is provided with an external-control library called TVIF, and this C library enables a server-side program to perform not only simple controls, such as play and pause, but also advanced tasks, such as interruption to execute a given script or reception of inputs from the mouse/keyboard.

The server also has some other roles, such as handling mouse/keyboard inputs, and internal processing that handles entities' informations.

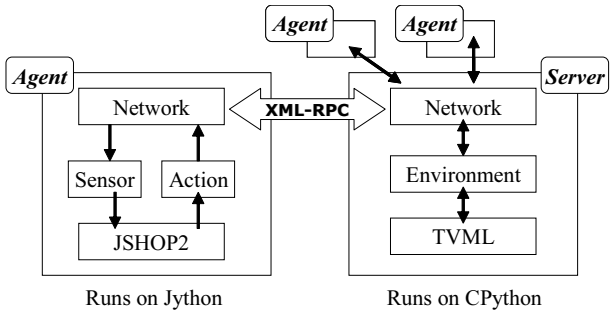

Figure 1: Architecture

Client

In our system, a client program is an autonomous agent that has an avatar in the virtual world. The role of a client is to control a character in the virtual world by using a planner. Receiving information from the server, a client performs planning based on that information. In addition, it requests the server to perform the set of actions that are generated as the result of planning.

IMPLEMENTATION

We have developed a prototype system that implements a part of the required functions as an initial version of our system. Below, we discuss this development and the current prototype system.

Programming Languages

We adopt the object-oriented language Python for development of the system. In particular, we are using both C implementation (CPython) and Java implementation (Jython) of the Python language out of need to integrate existing libraries and programs that are developed in these two languages.

The advantage of using Python is that it is simple and easy to use and also provides plenty of libraries. Therefore it suits short-term and small-group development.

In addition, Python is a capable glue language. Namely, it is easy to extend a python program with C language through CPython's interface, and Java classes can be directly used by Jython.

Prototype System

The prototype system we have developed realizes a part of the functions we required. The main implemented items are as follows:

- Visualization by TVML
- Internal processing
- Framework of server-client type system

The items we have not yet included in this prototype are as follows:

- Planning by JSHOP2
- Sensor module
- Action module

The highlights of the current implementation mainly are functions of the server side, and communication between the clients and the server. Because of the omission of JSHOP2 integration, characters are controlled by tentative codes written for demonstration.

The description of each module in the system is as follows:

Modules for the server

`idserver`
: This server side's main program starts up the system and handles the XML-RPC server.

`pytvif`
: By loading this module, a Python program can call any functions of TVIF.

`simplepytvif`
: This module manages the pytvif module and provides interfaces to the functions in pytvif.

`virtualenv`
: This module performs internal processing and also provides commands for controlling the objects and the virtual world itself.

Modules for the client

`idclient`
: This client side's main program establishes connection to the XML-RPC server and also controls the operations of characters as well as props.

Adaptation of TVML to the System

To implement this prototype system, we face with one technical issue, namely, how to integrate the C programs of TVML into the system and make them controllable by Python scripts. Our solution is that of extending TVML components in 3 steps as follows:

1. TVIF Adaptation

The first step is to make TVML available from Python. The key to this implementation is that of using the TVIF library that provides interfaces for controlling TVML. This library is a static or dynamic link library written in C language. Therefore, the system can not use it directly. CPython provides, however, an API called 'Python/C API' with which one can make a wrapper script to make the library loadable from Python. We create the wrapper script, `pytvif`, for TVIF by using the wrapper script generator called 'SWIG'.

2. Module Optimization

The next step is to make a new module that optimizes `pytvif` by enabling initialization of processes and execution of routine codes through simplified interfaces. As a result, this new module provides only needed functions. This is implemented as the `simplepytvif` module. For example, it sets up and starts up the TVML player using only a few methods and provides a method to interrupt and execute TVML scripts handed as a set of strings.

3. Extensible Implementation

The last step is to make it possible to play an animation consisting of a set of TVML scripts, by requesting only one command, and to construct the framework capable of adding new commands. This is realized by the `virtualenv` module. A command consists of a TVML script and internal processing codes that are bound to the command's name. Such a command can be called through the `command()` method, which makes it possible to avoid hard-coding of the command requests from clients.

RESULTS

Fig.2 shows a screenshot of the prototype system. This demonstration is for explanation of the implemented functions; the story itself makes no sense.

This demo is based on a sample script attached to TVML, and our system executes control commands while playing this script. The left half of Fig.2 is the screenshot of playing the original script, and the right half shows the result when the original script is controlled by our system.

The description of each screenshot and the corresponding internal processing is as follows:

Scene1——

[Original] The skit starts, and three characters (Bob, Mary, and Some guy) show up by walking into the room.

[Controlled] A guitar and a new character (Tom) are added to the scene.

[Processing] Commands for creating a guitar and Tom are requested to the server. Once created, they show up by the 'spawn' command.

Scene2——

[Original] The camera is panned to the front of the three characters, and then they start talking to each other.

[Controlled] Tom walks around the three characters, and then says "Interrrupt test one."

[Processing] By using for-loop, the `characterMove` command is repeatedly executed so that Tom walks through four points of location. In the original TVML script, control structures can not be used because commands are sequentially executed in a TVML script; thus, four move commands with predetermined parameters must be used. In the controlled one, the following commands are executed, which shows that Python's control structures can be successfully applied to the TVML script.

```
x = (-1, 1, 1, -1)
z = (-1.0, -1.0, 0.5, 0.5)
for i in zip(x, z):
    ve_server.command('characterMove',\
                'TOM', i[0], 0.0, i[1])
```

In this scene, while Tom is moving, he also says "Interrupt test one" in synthesized voice by the `characterSay` command.

Scene3——

[Original] The talk continues.

[Controlled] Tom comes close to the guitar, and stops at it.

[Processing] When the above for-loop reaches its end, the `characterMove`-command sequence ends, and the `characterPickUp` command is sent out to pick up the guitar. Receiving this command, the internal processing registers the information that the guitar object now belongs to Tom, and this is reflected on the screen with Tom having the guitar. The command to pick up a guitar is as follows:

```
ve_server.command('characterPickup',\
                    'TOM', 'GUITAR')
```

Scene4——

[Original] The talk still continues.

[Controlled] Tom walks around the three characters while he's holding the guitar, and halfway through that he says, "Interrupt test two."

[Processing] Besides Tom having the guitar, the process done here is same as Scene 2. However, this time, in the for-loop, `time.sleep()` is called right before `ve_server.command('characterMove','TOM', i[0], 0.0, i[1])` , and this makes the three characters continue their conversations even while Tom is moving.

Scene5——

[Original] 'Some guy' spins his head and then gets out of the screen.

[Controlled] Tom moves behind the three characters, and then jumps to the front of the camera.

[Processing] Basic actions, such as walking or picking up objects, are performed by the TVML scripts predefined in the server. However, in this scene, the action is performed by the `executeScript` command that executes any TVML scripts directly handed over. This means that one can define any actions and execute them at runtime. The code below requests the server to execute a TVML script that makes Tom jump and land at a specified location:

```
ve_server.command('executeScript',\
        'character: jump(name=TOM,\
          x=-0.3, y=0, z=1, d=45)')
```

DISCUSSIONS

The significance of the functions implemented in this prototype system is that animations of TVML can be controlled by Python codes. This can be perceived from the code that makes Tom move through four points of locations. Though our demonstration consists of very simple scenes, it shows that action parameters are dynamically set, which is a leap from the existing TVML. This is the basic idea which is, however, most necessary for implementing interactive drama. More specifically, if a story happens on a straight road, all one needs is to code and play TVML scripts in a conventional fashion. However, in interactive drama where characters' actions are decided at runtime, the control structures and variables are essential.

On the other hand, the current problem with our system is that it is elaborate to execute both ongoing scripts and interrupting scripts simultaneously. The actual problem in the demo of Fig. 2 was that the three's talk stopped when Tom started moving. As a tentative solution, overwrite of talk was avoided by adjusting the timing of the interruption, but it is not a fundamental solution. For dealing with the case that multiple interrupting scripts conflict, the system needs a function that

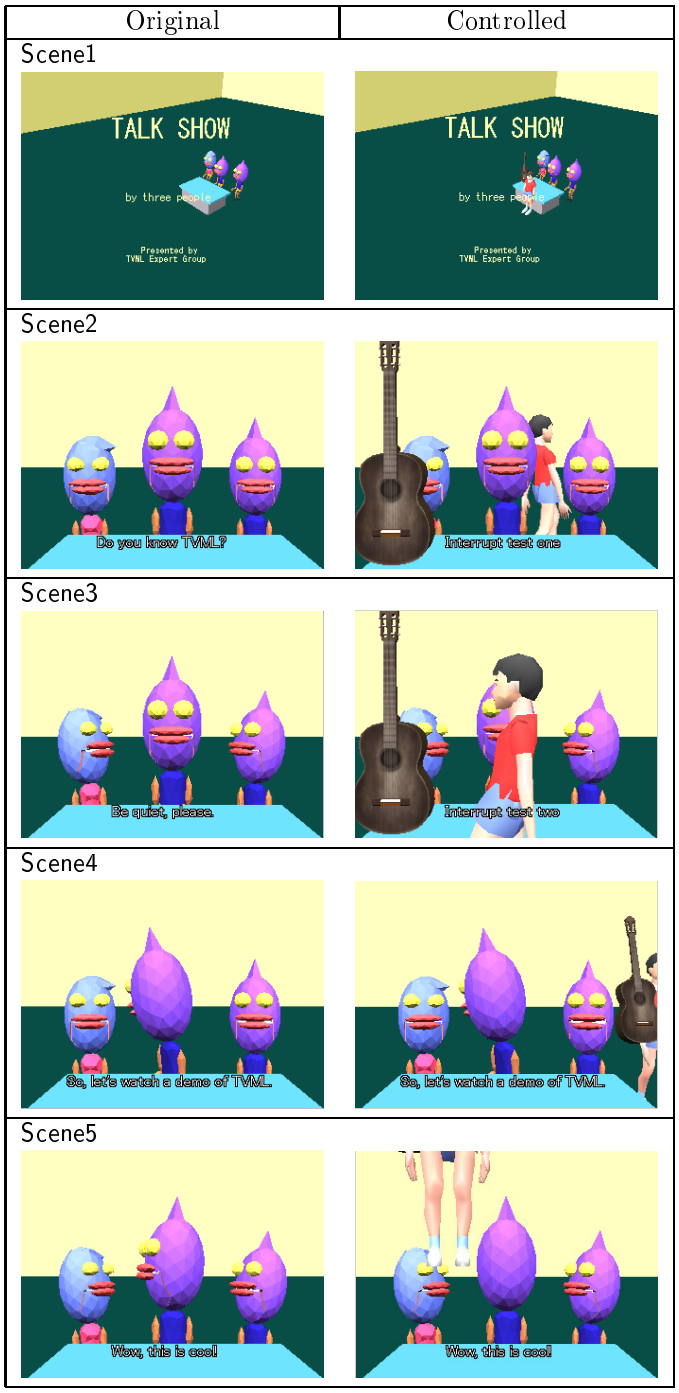

Figure 2: Screenshot

synchronizes multiple commands by buffering those requested commands and generating a combined TVML script that plays as expected.

CONCLUSION

In this paper, we have presented the architecture of a multi-purpose interactive drama system that facilitates content creation and system extension. In addition, we have developed the prototype system that implemented a part of required functions, with which we could show the significance of our extension to TVML for handling dynamic control by Python.

Our future work is that of implementing the remaining functions required for the system, such as the agent with JSHOP2 and the support for multiple clients. In addition, we are planning to verify our modified version of JSHOP2 through experiments using the developed system.

References

[1] Mateas, M. and Stern, A.: Architecture, Authorial Idioms and Early Observations of the Interactive Drama Facade. Tech report CMU-CS-02-198, Carnegie Mellon University. 2002

[2] Young, R.M. and Riedl, M.O.: Towards an architecture for intelligent control of narrative in interactive virtual worlds. In Proceedings of the 2003 International Conference on Intelligent User Interfaces, 310-313. 2003

[3] Cavazza M., Charles, F., and Mead, S.J.: Developing reusable interactive storytelling technologies. In the IFIP World Computer Congress, 2004.

[4] Hayashi, M., Ueda, H., and Kurihara, T.: TVML (TV program making language)-Automatic TV Program Generation from Text-based Script. In Imagina 99. 1999.

[5] Ilghami, O.: Documentation for JSHOP2. Tech report CS-TR-4694, University of Maryland. 2005

[6] Nau, D.S., Au, T., Ilghami, O., Kuter, U., Murdock, J., Wu, D., and Yaman, F.: SHOP2: An HTN planning system. Journal of Artificial Intelligence Research 20, 379-404. 2003.

[7] Nau, D.S., Smith, S.J.J., and Erol, K.: Control Strategies in HTN Planning: Theory versus Practice. AAAI-98/IAAI-98 Proceedings, pp. 1127-1133, 1998.