COMPUTER GAMES & ALLIED TECHNOLOGY '08

Animation, Multimedia, IPTV & Edutainment

Editor:
Edmond Brakach
Games Authoring Tool for Existing Game Engines

Liu Willy Budiman             Raymond Kosala

School of Computer Science & BINUS Business School, BINUS University
Jl. Hang Lekir 1 no. 6, Kebayoran Baru,
Jakarta, Indonesia
Contact author: rkosalab@binus.edu

Abstract

The games technology has been gaining some popularity in the industry. However, creating game-based application is not easy, laborious, and requires good knowledge of the technology and programming skill. A typical game development process might take months or years to complete. In order to improve the usage of game technology, certain methods and tools have been invented, and the most popular one is game engine. However, learning to use game engine is not simple and takes time. Moreover, we still need programming skills and technical knowledge to use the game engine. This paper focuses on building a games authoring tool that can flexibly work with the existing game engines. We propose a novel plug-in architecture so that the authoring tool can use existing game engines. The advantage of this independent authoring system is that whenever one of the systems became obsolete, the developer can replace it without changing the whole system structure. In the experiments with the prototype system, the usage of the authoring tool shows significant process reduction compared to programming it manually. Furthermore, test on the users shows that the authoring tool can help non-technical users to develop 3D games easily.

Keywords

Authoring tool, game engine, user interface, game development.

1. Introduction

The games technology has been gaining much popularity in the industry. In the early days of games development, developers used to create their game from scratch. Most of the games created back there are still done by writing the program in a manual way in the sense that the whole program is written from scratch. Even today, there are still developers or games programmer that do it in a manual way. It is a very laborious and long process to create games and the process is even longer to create complex 3D games. A typical game development process might take months or years to complete.

In order to fasten the development process, developers try to use design patterns to create games. Later the collections of design patterns are combined to establish a game engine. With a game
engine, we can create powerful games and reduce the effort and time required to build a game. This technology allows developer to make the game production even faster. However, learning to use game engine is not simple and takes time. Moreover, we still need programming skills and technical knowledge to use the game engine.

In order to solve those problems, in this paper, we develop a game authoring tool with flexible architecture. Most of the current game authoring tools, such as 3D Game Maker, 3D Game Studio, and Unity3D, do not use existing game engine but they rather use their own engine, which is usually tightly integrated with their authoring interface.

This paper focuses on building a games authoring tool that can flexibly work with the existing game engines. We propose a novel plug-in architecture so that the authoring tool can used with existing game engines. The advantage of this independent authoring system is that whenever one of the systems became obsolete, the developer can replace it without changing the whole system structure.

In the experiments with the prototype system, the usage of the authoring tool shows significant process reduction compared to programming it manually. Furthermore, test on the users shows that the authoring tool can help non-technical users to develop 3D games easily.

2. Preliminaries

2.1 Game Authoring Tool

A game authoring tool is software that helps developers to create their games with having little or no programming knowledge at all. The purpose is to allow creation of games without getting bogged down in programming detail [7]. Typically, the creation is by linking various objects with pointing and clicking manner. It simplifies the game making process as it was intended for non programmers.

2.2 Game Engine

A game engine is the program that runs all the components that comprise a game, including graphics, audio, collision detection, physics, and artificial intelligence [1]. A game engine is a large library and is subject to the software engineering principles that govern how such a large package evolves. Game engine merely just preparing all game components needed to create a game. Developers have to combine, among others, graphics engine, physics engine, networking engine, and an AI engine, into a single game [2].

In this paper, we use Irrlicht [3] game engine for authoring tool development experiment. The Irrlicht engine is an open source high performance realtime 3D engine written and usable in C++ and also available for .NET languages. It is completely cross-platform, using Direct3D, OpenGL, and its own software renderer, and has state-of-the-art features which can be found in commercial 3D engines.

3. Authoring Tool Design

Figure 1 illustrates the architectural layout of the game authoring tool, which is implemented using MVC (model-view-controller) pattern. Figure 1 shows three system components: a GUI which resides and interacts with client, an authoring tool system, and a game engine system. Inside of the authoring tool reside the system components. The controller component receives input from the user and handles the communication between the model and the view. The view component handles all
interface specific functions that need to be presented to the users. The model component contains application data that comes from the game engine function and process all content objects.

The main objective of the game authoring tool is to provide an environment which will provide the link to the game engine to simplify the mapping to a complex game engine. To do that, the game authoring tool will be divided into three parts: the game engine, the development environment, and the system connection. The game engine part (model) will prepare the required procedures to invoke certain features or functionalities. The development environment part (view) will prepare a user interface with certain functionality from the game engine. The system connection part (controller) will prepare a link from the user interface to the game engine.

The data flow of the system is shown in Figure 2. The authoring tool system consists of several modules. The modules are terrain module, model module, objects module, controls module, collision module, effects module, environment module, and render module. These modules have their specific process activity and process structure.

A render module is part of the system module because it has a specific purpose, which is to render or to draw the result to the screen. The render module is not a stand-alone module because most of its function takes parameters from other modules. However, the render module does not come from class inheritance. It is a separate class but linked through the authoring tool class using an association design pattern. It has attributes from both systems, which are the authoring tool system and the game engine. The purpose is to make information exchange between the authoring tool system and the game engine less complicated.

The system will load any subsystem by clicking in the UI. When a certain subsystem has been called, it will note the system again that this subsystem has been activated. An activated subsystem is used as a reference for the game engine to render the result. The game engine system will also select which engine that has been activated by the main system that needs to be loaded. The overall system is divided into several manageable subsystems to make it easier to develop and debug.
Figure 2. Game Authoring Tool System Data Flow Diagram

The system user interface handles system input and output from the user to the system. Here, as an example, Irrlicht is the system processor or main game engine. The link between the game engine and the UI is the system connector, which is the parameter from an activated subsystem and the game engine. Each subsystem also has a unique data flow of information from and to the main system. The use of Irrlicht is just for proofing our concept and for the first prototype. The usage of other game engines should be possible with our plug-in architecture.

In all, the main processing of each subsystem follows the same pattern, which is to extract parameters from user input and pass it to the system connector. These parameters will be used by the game engine to render the output of a particular subsystem.

Figure 3 shows the top-level GUI of the game authoring tool that we developed and an example of the game rendering.
4. Authoring Tool Evaluation

We believe that breaking the main system into subsystems has aided development step very well. With subsystems corresponding to game engine functions, author can select which function is really required and intended for it. This is also the reason why the tool has a separate system and design parts. In this design, each subsystem has its own processing system.

Introducing graphical user interface has also aided users in using the authoring tool. Users find it easier to understand and use the tool. The learning curve is quite steep as it is very straightforward to interact with it. You just press button, input some values, and change parameters without the need to know what was behind the operation.

The authoring tool use existing game engines as its game processor because game engines have the simplified functions in producing game features. Developing the authoring tool from game engines eliminates long setup and tedious game function. Using existing game engines also provide other benefits such as they have been tested for system correctness and producing the right output. From the unit test that has been conducted, the non-programmer users can learn to use the authoring tool without any difficulty.

Next we want to discuss about the authoring tool architectural design. It has been mentioned that the overall design is to introduce separate layer between the authoring tool user interface and the game engine. Both systems are linked by a plug-in architecture, which use function calls to load the appropriate functions. This allows users to change one of the components, which can be the user interface or the game engine, by creating the appropriate plug-in module. With this technique, whenever user wants to replace one of the systems, the users do not need to change the whole design structure. Independent design architecture can help developer in adding, modifying, or replacing system components without much problem.

Finally, we evaluate the authoring tool productivity and effectiveness in creating game from the start. The test conducted was to compare the steps taken to produce certain game features by the authoring tool and coding it manually with the game engine. The results are as follows:
In rendering terrain, out of 15 steps in game engine, it only requires 8 steps in authoring tool. The amount of work to do is 53% which means 47% reduction.

In rendering model, out of 15 steps in game engine, it only requires 9 steps in authoring tool. The amount of work to do is 60% which means 40% reduction.

In rendering objects, out of 13 steps in game engine, it only requires 8 steps in authoring tool. The amount of work to do is 62% which means 38% reduction.

In rendering controls, out of 14 steps in game engine, it only requires 6 steps in authoring tool. The amount of work to do is 43% which means 57% reduction.

In rendering collision, out of 14 steps in game engine, it only requires 6 steps in authoring tool. The amount of work to do is 43% which means 57% reduction.

In rendering effects, out of 16 steps in game engine, it only requires 6 steps in authoring tool. The amount of work to do is 38% which means 62% reduction.

In rendering environment, out of 11 steps in game engine, it only requires 6 steps in authoring tool. The amount of work to do is 55% which means 45% reduction.

![Game Engine](image)

<table>
<thead>
<tr>
<th>Tool</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terrain Authoring Tool</td>
<td>53%</td>
</tr>
<tr>
<td>Model Authoring Tool</td>
<td>60%</td>
</tr>
<tr>
<td>Objects Authoring Tool</td>
<td>62%</td>
</tr>
<tr>
<td>Controls Authoring Tool</td>
<td>43%</td>
</tr>
<tr>
<td>Collision Authoring Tool</td>
<td>43%</td>
</tr>
<tr>
<td>Effects Authoring Tool</td>
<td>55%</td>
</tr>
<tr>
<td>Environment Authoring Tool</td>
<td>55%</td>
</tr>
</tbody>
</table>

Figure 4. Process Effectiveness Chart

The chart in Figure 4 shows the percentage of amount of work needed to perform in authoring tool compared to 100% of work when done by manual coding. Although our method is simple, in the sense that it does not consider, for example, the actual time or the level of users' expertise in coding, but it gives an idea of effectiveness achieved by the authoring tool. Based on results above, the reduction of works are significant.

5. Conclusion

We have explained the design and development of an independent game authoring tool that answers the problems caused by the complexity of creating games. Game engines have been used to aid many developers in creating games. However, many people have not yet used this technology. They tend to build their own engine as it is easy to debug and maintain. Game engine on the other hand requires knowledge to use it effectively. An independent user interface has been shown to solve the problem with the long time needed for development and
unfamiliarity with the game engine. New user can learn and use the user interface quickly. The tests have shown a significant process reduction, which can be up to 62%.

A clear separation between the authoring tool and the game engine also has another advantage. With unique design architecture, adding, deleting, modifying and changing one of the systems can be done without changing the whole system structure.

The game authoring tool built in this paper is kind of new technology that has never been implemented before. Combining user interface as authoring tool and game engine is quite possible. Moreover using existing game engine with authoring tool and later replacing one or more of it is doable.

Our independent game authoring tool is still a prototype. The list of possible future improvements is as follows.

- Adding more game features. The seven game features discussed previously are the most basic features in producing game. Other features that can be included in the game are AI (artificial intelligence), network game, etc.
- Adding game scenario feature in the authoring tool. A game scenario is a gameplay in the game and constructs the game objective and story plot. Without this feature, a game is hardly to be called playable and does not add uniqueness to the game.
- Finally is that to create an executable file of the game produced by the authoring tool. With an executable file, users do not have to develop the game from the start again and they can use the file to distribute the game.

6. References
