2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010)
IPCSIT vol. 53 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V53.No.2.29

Architecture Design of A Lightweight Extensible GUI Framework for Games

Zhixing Shen[+] and Jinyuan Jia

School of Software Engineering, Tongji University

Shanghai, China

Abstract—Graphical user interface (GUI) design is a core part of games development, as it is responsible for the look and feel of a game to players. However, GUI design is often overlooked and also time consuming. This paper presents a novel architecture design of a lightweight extensible GUI framework for games to ease the pains of GUI design. Firstly, the core architecture of the GUI framework is described. Secondly, main concepts about the framework, such as GUIControl, Input handling and GUISkin, are further explained. Thirdly, illustrative examples are given to demonstrate the GUI framework. Finally, we summarize characteristics and advantages of this GUI framework. These show that the GUI framework is a lightweight one which also of great extensibility.

Keywords- graphical user interface; GUI; game engine; games development; computer graphics; design patterns

1. Introduction

With the booming development of games, animation, virtual reality and other industries in the worldwide, consumers' demands for related products are also rising. Consumers not only want to see beautiful vision effects from games or virtual applications, but they also desire better user experiences. Graphical user interface (GUI), as its name implies, plays an important role in enhancing the user experience. If the GUI of one game is well designed, the players can play the game smoothly without too much effort to get used to it; if not, the players may feel bad or confused when they play it [1], [2].

On the development side, all GUI elements in a game are developed based on a module of Game Engine: GUI framework (or GUI component, GUI sub-system). GUI framework may be varied depending on the specific games. Generally, existing GUI frameworks can be classified into two genres: the first genre includes heavyweight GUI libraries such as CEGUI [3], and the second genre includes internal GUI libraries which are specified for one game. There are problems for both of the two kinds of GUI frameworks: For the heavy one, the problem is that the framework is too heavy for some games and it may take great efforts to integrate it with the specific game; for the internal one, we may need to develop new GUI framework to adapt the GUI requirements of new game once we start developing new games, this takes extra efforts too. Furthermore, GUI needs to be quickly and dramatically adapted continuously throughout a product's development cycle in order to reflect modifications to virtually any other part of the game [4].This also requires that GUI Framework should be of great extensibility.

In order to solve the problems mentioned above, we take existing GUI frameworks as references and propose a novel architecture design of GUI Framework for games, which is also of great extensibility.

[+] Corresponding author.
E-mail address: istar.shen@gmail.com

The rest of the paper is organized as: Section 2 overviews related works on GUI design; Section 3 list background information about this paper; Section 4 describes architecture design of the novel GUI framework in details; Section 5 demonstrates the framework through illustrative examples; Section 6 summarizes advantages of the GUI framework; Section 7 concludes this paper and discusses future works.

2. Related Works

To our best knowledge, there are not too many literatures on GUI design for games until now. Considering the fact that GUI design for traditional software shares some idea with GUI design for games and there have been more publications on GUI design for traditional software, we think that some techniques used to design GUI for traditional software should be good references for GUI design for games. Below are the latest works about GUI design for games and traditional software:

A visualized GUI editor is presented to create and edit game interfaces quickly without programming [5]. The combination of a hybrid dynamic and static approach is investigated to allow for round-trip maintenance of GUIs. Dynamic analysis reconstructs object relationships and static checking guides the reconciliation between the GUI editors' design view and source [6]. The problem of GUI logic flaws is formulated and a methodology is developed to uncover GUI logic flaws in software implementations [7]. A software development approach for teaching graphical user interfaces (GUI) and event handling through computer games is discussed. It is found that games could be used as a programming assignment for the purpose of introducing GUI, event handling and other programming concepts [8]. An approach towards standardization of the general rules is presented to synthesis and design of man machine interfaces that include dynamic adaptive behavior [9]. A method is proposed to concentrate upon user sequences of GUI objects and selections which collaborate, called complete interaction sequences (CIS).An empirical investigation of this method shows that substantial reduction in tests can still detect the defects in the GUI [10].

3. Background

3.1. GUI Frameworks for Games

A graphical user interface (GUI), often pronounced gooey, is a type of user interface that allows users to interact with programs in more ways than typing such as computers; hand-held devices such as MP3 players, portable media players or gaming devices; household appliances and office equipment with images rather than text commands [11]. A GUI offers graphical icons, and visual indicators, as opposed to text-based interfaces, typed command labels or text navigation to fully represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements [12].

For games, GUI plays an even more important role in connecting players with virtual game worlds. At first, players perceive game worlds through Game GUI. Then, they take actions through different input devices (such as keyboard, mouse, joystick and so on) and these inputs are parsed into specific commands. Once the game receives these commands, corresponding game logic will be updated and players may get responses through Game GUI. Thus, players can continue interacting with the game world.

Opposite to its importance, GUI design is an often overlooked and under-resourced part of game development. In fact, we need to take great efforts on it as the UI needs to be quickly and dramatically adapted continuously through a product's development cycle in order to reflect modifications to virtually any other part of the game[4]. In order to reduce the efforts to be taken, some GUI frameworks are developed by game developers. The GUI framework here generally means a collection of useful libraries which are composed of a good many re-useable codes when design GUI for games. By using such GUI frameworks, developers can design and implement new GUIs more rapidly and they can focus on other parts of the game.

3.2. Design Patterns

As the paper is about the architecture and design of GUI framework, some design patterns which are used in the GUI framework should be introduced first as below [13, 14]:

Singleton ensures a class only has one instance, and it provide a global point of access to the instance. It can be depicted as Figure 1.

Composite is used to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. It can be depicted as Figure 2.

MVC The model-view-controller (MVC) design pattern aims to ensure only a loose coupling of elements by separating the framework into three distinct constituent parts to be maintained independently:

The ***model*** refers to the actual data that we represent on the screen.

The ***view*** concerns itself only with the visual representation and rendering of that object.

The ***controller*** refers to how the object interacts with the game, user input, and general state, system, or game logic.

MVC pattern can be depicted as Figure 3.

Fig.1. Design Pattern - Singleton.

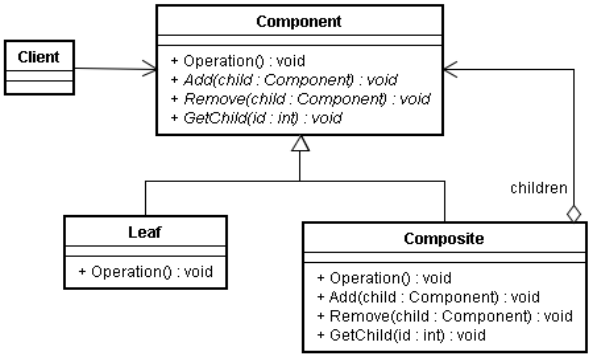

Fig.2. Design Pattern – Composite.

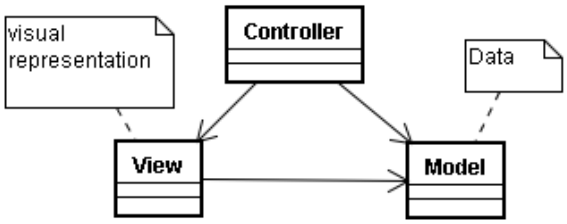

Fig.3. Design Pattern – MVC.

4. Architecture and Design

4.1. GUI FrameWork: As a Sub-Module of a Game Engine

Before heading into the architecture details of the GUI framework, it's necessary to specify how the GUI framework should be integrated with other modules of a game engine. This is depicted as Figure 4.

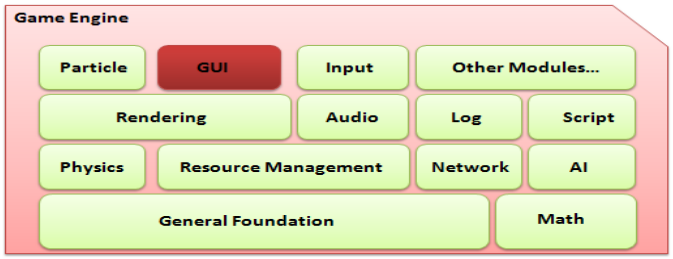

Fig.4. Architecture Overview of a Game Engine.

From Figure 4, we can see that GUI framework is a high-level module in Game Engine. GUI framework is based on the Rendering module and it also interacts with the Input module to get information about players' inputs.

4.2. Core Architecture

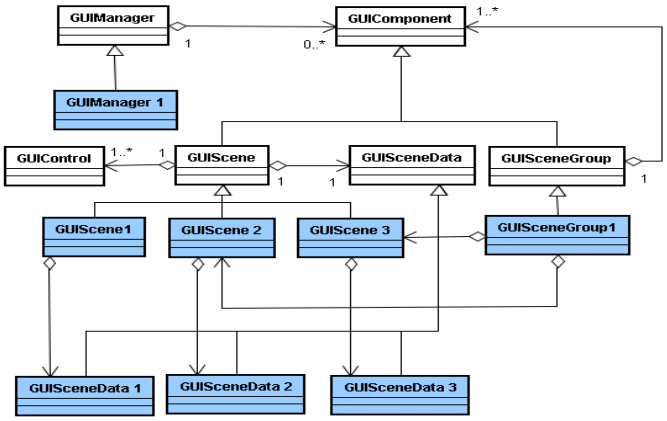

Fig.5. Core Architecture of GUI framework.

Core architecture of the GUI framework is depicted as Figure 5, and main concepts are explained as below:

GUIManager is a singleton class which manages all the GUI stuff such as loading GUI Scene, switching among GUI Scenes and communicating with Input module.

GUIScene represents a single in-game HUD screen or a menu screen. It contains many GUIControls and it is responsible for managing Update () and Render () of all child GUIControls. GUIScene is the Leaf in Composite design pattern and the Controller in MVC design pattern.

GUISceneGroup Each UISceneGroup contains a great deal of GUIScenes. It is mostly used to represent a menu group which is composed of sub-menus. GUISceneGroup is the Composite in the Composite pattern.

GUIComponent is the super class of both GUIScene and GUISceneGroup. It is the Component in Composite design pattern and it provides unified interfaces for its client.

GUIControl The basic component in GUIScene is GUIControl. Each UIControl is responsible for parsing users' inputs and rendering itself under actual game logic. So it can be regarded as the View in MVC design pattern.

GUISceneData In order to separate visual representation of a GUIControl and the actual data which may influence the representation, GUISceneData is introduced. Each GUISceneData is linked with a GUIScene, and GUIScene will first extract data from GUISceneData, then pass the data to corresponding GUIControl, finally, GUIControl will render itself with the data if needed. Here GUISceneData can be regarded as the Model which is used to hold the data in MVC design pattern.

4.3. GUIControl

The most basic component in GUIScene is GUIControl. Generally, there are two kinds of GUIControls: Common GUIControls and Specialized GUIControls.

Common GUIControls are controls which can be reused frequently. They mainly include Text, Label, Button, Slider, Scrollbar and Window. On the game side, these common controls may be combined into a more complicated GUIControl. New common controls can also be added into the base framework if needed in the later days.

Specialized GUIControls are controls which are specific to a game or a game level. Opposite to common GUIControls, specialized GUIControls are not located in the game engine but in the game side. They may inherit from GUIControl directly, or inherit from some common control. Specialized GUIControl is a supplement to the common control because it is hard to implement some complicated GUI features using common controls.

The two kinds of GUIControl can be depicted as Figure 6.

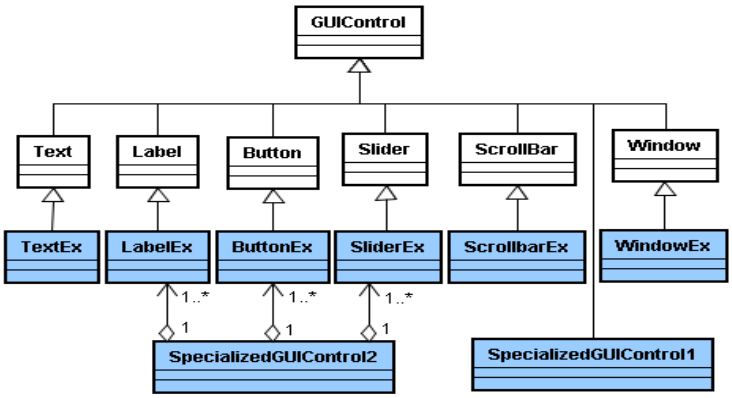

Fig.6. The Inheritance of GUIControl.

4.4. Input Handling

In Figure 4, we have already known that GUI framework will interact with Input module to get information about players' inputs. The inputs will be handled by GUI framework through dispatching messages to GUIControls. The mechanism can be depicted as Figure 7. Original inputs information is transmitted from Input module to GUIManager. Then, GUIManager filters the information and sends a message to current GUIScene. After this, GUIScene will send messages which encapsulate inputs information to all GUIControls in it. Finally, each GUIControl can parse the inputs and update itself.

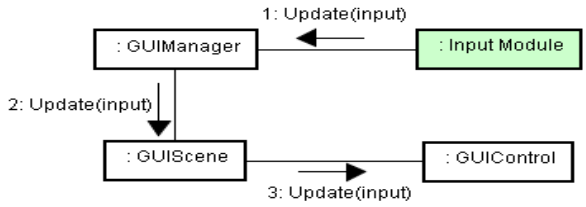

Fig.7. The mechanism of input handling.

4.5. GUISkin

In order to switch between different update and render styles of a GUIControl at run time of the game, the concept of "GUISkin" is incorporated into the GUI framework. GUISkin here shares the same ideas with "CSS" in the field of web design. Each GUIScene contains a GUISkin which described the render and update style of each GUIControl in the GUIScene. Once game developers have implemented different render and update styles for a GUIControl, they can switch among these styles though changing the settings of the corresponding GUISkin.

The main idea of GUISkin can be depicted in Figure 8.

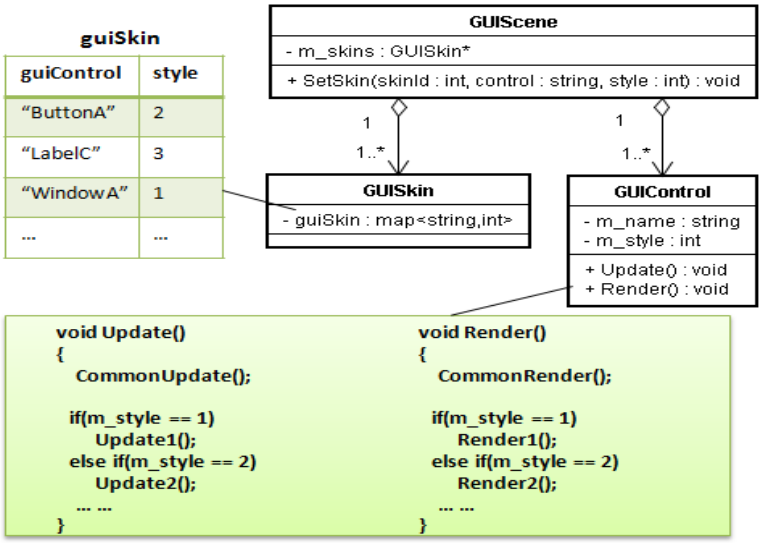

Fig.8.Overview of GUISkin

5. Illustrative Examples

In order to make the main ideas of the GUI framework more clearly, illustrative examples are given in this section. These examples are based on a racing game.

The menu screen of the racing game is depicted as Figure 9. Firstly, the menu screen itself is a GUISceneGroup which is composed of several child GUIComponents. One of these child GUIComponents is “Options” which is also a GUISceneGroup. And the “Options” is composed of 4 GUIScenes: “Audio”, “Graphics”, “Control” and “Misc”. Currently, the “Audio” GUIScene is selected and some GUIControls which are used to control volumes are displayed. GUIControls used in this menu screen are Text, Label, Button, CheckBox and Slider.

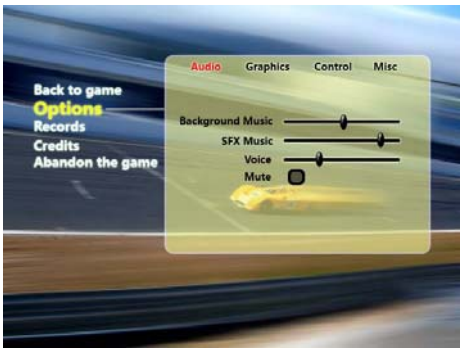

Fig.9. Menu Screen of a Racing Game

Two in-Game GUIs are depicted as Figure 10 and Figure 11. Each in-Game GUI screen is a GUIScene which manages all the GUI stuff in this GUIScene.

In Figure 10, we can see many GUI elements: two numerical indicators for gas and speed at the left-top corner, one mini-map at the right-top corner, one wheel and one shift at the bottom, a “Skin” button at the left-bottom corner and a “Menu” button at the right-bottom corner. Some of them are just common GUIControls. For example, the numerical indicators are Text controls, the “Skin” button and the “Menu” button are Button controls. The others are specialized GUIControls which inherit directly from GUIControl. For example, the wheel is a specialized control which can rotate around its center to mimic the real wheel driving, the shift is also a specialized control which is used to mimic the functionality of real shifts, and the mini-map is a powerful GUIControl which is composed of other GUIControls.

Fig.10. In-Game GUI(Skin 1)

The In-Game GUI screen in Figure 11 shares some ideas with the GUI screen in Figure 10. They can switch to each other by clicking the “Skin” button at the left-bottom corner. In Figure 11, the wheel and the shift are replaced by a virtual stick and two round buttons. For this mode, we can control the car using the virtual stick and shift by clicking the two round buttons. The functionality of “dynamic GUI switching” is implemented through GUISkin of the proposed GUI framework. With the help of “GUISkin”, game designers are able to compare different GUI control methods at run time of the game and then choose a better one as the

final control methods. In a simple word, GUISkin saves communication time between designers and programmers.

Fig.11. FiIn-Game GUI(Skin 2)

6. Discussion

From the above sections, we can find that the GUI framework is a lightweight one which also of great extensibility. The advantages of it can be summarized as below:

Highly concise architecture Core architecture of the GUI framework is highly concise as it only includes several classes: GUIManager, GUIComponent, GUIScene, GUISceneGroup, GUIControl, GUISceneData and GUISkin. Complicated functionalities are implemented through inheriting from such classes on the game side. The concise architecture ensures the GUI framework easily maintained.

Highly extensible GUIControl No matter common controls or specialized controls, they are easy to be reused and extended. For most of time, developers can simply use common controls directly. If predefined common controls can't meet the requirements, developers can also inherit GUIControl to create new specialized controls.

Clear hierarchy In this GUI framework, actual update logic and render logic are encapsulated in GUIControl, and all GUIControls are managed by a GUIScene. Furthermore, GUIScenes can comprise each other to form a tree-like structure. The clear hierarchy simplifies most operations to traversing the tree-like structure.

Dynamic GUI switching The incorporated GUISkin enables the players to switch among different GUI styles at run time of the game. This feature is necessary for some games or some game levels.

Highly portable The GUI framework is almost self-contained and it only replies on two external modules: Input and Rendering. Through writing two wrapper classes for external input and rendering module, the GUI framework can be easily integrated with any other game engines or libraries.

Separate data from render style Actual game data and the render style of a GUIControl are separated into different classes: GUIControl is responsible for visual representation, and GUISceneData is responsible for storing data which will be used by GUIControls in a GUIScene. By doing this, either the data or the render code can be easily maintained.

Easy to be used As the above advantages, the GUI framework is very easy to be used to implement new GUI features for a specific game. A junior game developer who is familiar with game development can master using the GUI framework within several days.

7. Conclusion and Future Work

In this paper, we present a novel architecture design of a lightweight and extensible GUI Framework for Games. Inspired by CSS in the field of web design, we also introduce the concept of GUISkin into this GUI framework. Then, we discussed the advantages of this GUI framework from different aspects. The main idea of this GUI framework has been used for developing an iPad game already. And we find it is very easy to use and it also saves time greatly so that we can take more time to polish other parts of the game.

The proposed architecture design is just the beginning of a more powerful GUI framework. Actually, more works can be done and more functionality can be incorporated into this framework. For example, to make this GUI framework be data-driven, to develop a GUI tool for designing new GUI and to create more common controls. These will become our future works in the next stage. We believe, with these works been done, the GUI framework should become more powerful but still lightweight enough.

8. Acknowledgment

We would like to thank those anonymous reviewers for their insightful comments. Also, we sincerely thank our WebVR team members and some friends worked in 2K Games China for their kind cooperation and constructive discussion: Ms. Zhao Zhang, Ms. Hong Zhang, Ms. Wenjun Cai, Mr. Chang Luo, Mr. Wenzhao Ren, Mr. Lei Xi, Mr. Yushi Xia, Mr. Haoyuan Zhang, Mr. Leny Liu, Mr. Shengjie Shao, Mr. Tianyi Cheng, Ms. Xi Wang. Furthermore, special thanks should be given to Shanghai Key Breakthrough Grant of Science & Technology (Grant No. 08511501000) for its supports to start this research.

This research is supported by Shanghai Key Breakthrough Grant of Science & Technology, Grant No. 08511501001.

9. References

[1] Ismail, S.F., R. Hashim, and S.Z.Z. Abidin. *Collaborative bridge game: A comparative study on user interface design*. in CAMP'10. 2010. Shah Alam, Malaysia: IEEE Computer Society.

[2] Fei, H. and J. Lixia. *On the Peak-Experience in the Game GUI Design*. in Management of e-Commerce and e-Government, 2008. ICMECG '08. International Conference on. 2008.

[3] *Crazy Eddie's GUI system*, http://www.cegui.org.uk, 2010.

[4] Adrian Hirst, *A GUI Framework and Presentation Layer*, in Game Engine Gems, Vol. 1. 2010.

[5] Tan, X., et al. *Research and implement of tree structure based graphic user interface editor*. in ICALIP 2008. 2008. Shanghai, China: Inst. of Elec. and Elec. Eng. Computer Society.

[6] Peng, L. and E. Wohlstadter. *Dynamic round-trip GUI maintenance*. in Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on. 2008.

[7] Shuo, C., et al. *A Systematic Approach to Uncover Security Flaws in GUI Logic*. in Security and Privacy, 2007. SP '07. IEEE Symposium on. 2007.

[8] Estell, J.K. *Teaching graphical user interfaces and event handling through games*. in ASEE 2004 Annual Conference and Exposition. 2004. Salt Lake City, UT, United states: American Society for Engineering Education.

[9] Kostov, V. and S. Fukuda. *Development of man-machine interfaces based on user preferences*. in CCA '01. 2001. Mexico City, Mexico: Institute of Electrical and Electronics Engineers Inc.

[10] White, L. and H. Almezen. *Generating test cases for GUI responsibilities using complete interaction sequences. in Software Reliability Engineering*, 2000. ISSRE 2000. Proceedings. 11th International Symposium on. 2000.

[11] *Graphical user interface*, http://en.wikipedia.org/wiki/Graphical_user_interface#cite_note-1, 2010.

[12] *Window manager Definition*, PC Magazine. Ziff Davis Publishing Holdings Inc. 2008.

[13] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns*, Addison-Wesley. 1995.

[14] Julian Gold, *Object-oriented Game Development*, Addison-Wesley. 2004.