2015-7-30

Concordia University

Department of Computer Science

and Software Engineering

Team B	
Name	SID
Cao, Guocai	27607059
Li, Zhihao	27252331
Lyu, Qiuyi	27293143
Na, Heya	27208243
Sun, Sai	26263704
Yin, Ge	27116829

1.Architecture.......... 2

1.1 Model.......... 2

1.2 view.......... 2

1.3 Controller.......... 2

1.4 Detailed design..........3

References..........5

1.Architecture

We use the Model-View-Controller (MVC) which is a commonly used and a powerful high level architecture system for GUIs. The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller.

1.1 Model

The model is used to manage information and notify observers when that information changes. It only contains data and functionality that are related by a common purpose. A model is meant to serve as a computational approximation or abstraction of some real world process or system. It captures not only the state of a process or system, but also how the system works. This makes it very easy to use real-world modeling techniques in defining our models.

1.2 view

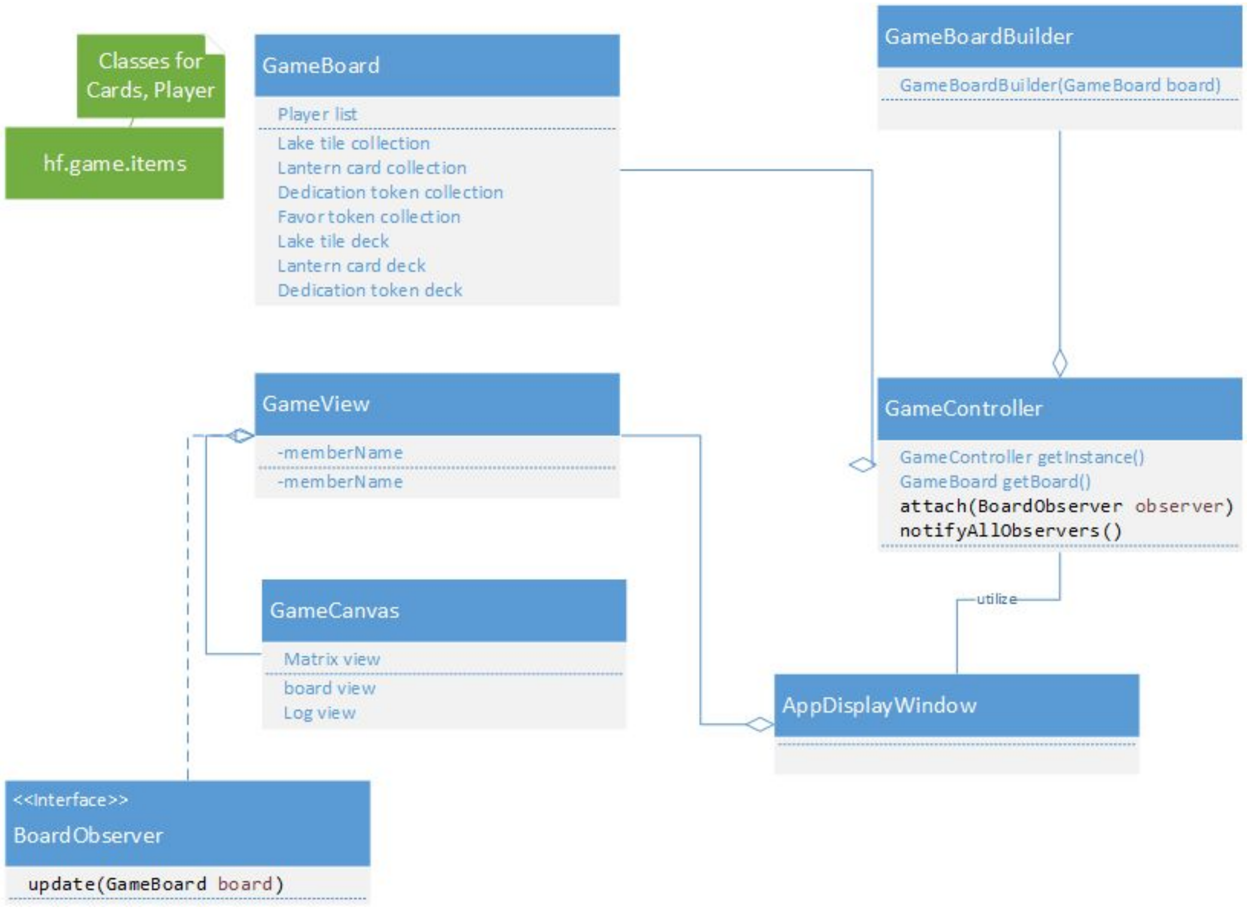
The view or viewpoint is responsible for mapping graphics onto a device. A viewpoint typically has a one to one correspondence with a display surface and knows how to render to it. A viewpoint attaches to a model and renders its content to the display surface. In addition, when the model changes, the viewpoint automatically redraws the affected part of the image to reflect those changes. There can be multiple viewpoints onto the same model and each of these viewpoints can render the contents of the model to a different display surface. A viewpoint may be a composite viewpoint containing several sub-views, which may themselves contain several sub-views.

1.3 Controller

A controller is the means by which the user interacts with the application. A controller accepts input from the user and instructs the model and viewpoint to perform actions based on that input. In fact, the controller is responsible for mapping end-user action to application response. For example, if the user clicks the mouse button or chooses a menu item, the controller is responsible for determining how the application should respond.

1.4 Detailed design

Game board is virtualized as the overall model, and all the UI views are built based on data structures kept in the model. GameBoardBuilder class is the constructor class for game board, it will contain functions to load cards collections, create decks of cards, distribute player hand cards and etc…



GameController class is defined as a singleton class to create and keep only one instance of the game board object, and it features the observer pattern to notify any interested component about a game board update, for instance a saved game has been loaded or a new game has been started.

Two 3rd party libraries have been chosen to better support the project:

- Slick 2D library to provide an easy way of creating UI components, and control UI behaviors
- XStream library to provide an easy way of loading and saving game board object from to XML documents.

We also use observer pattern, which is a key part in Model–View–Controller (MVC). It is very suitable when there is one-to-many relationship between objects, such as if one object is modified, its dependent objects are to be notified automatically. The observer pattern separates the observer and the observed perfectly, and also sets precise delineation of boundaries between modules. This greatly increases the maintainability and reusability of our application.

References

[1]Kruchten, Philippe. "Architectural Blueprints— The “4 + 1” View Model of Software Architecture." Tutorial Proceedings of Tri-Ada 95 (1995): 540-555.

[2]Hannemann, Jan, and Gregor Kiczales. "Design pattern implementation in Java and AspectJ." In ACM Sigplan Notices, vol. 37, no. 11, pp. 161-173. ACM, 2002.

[3]Patterns, D. E. S. I. G. N. (2003). Model-View-Controller.

[4]Jahnke, Jens, and Albert Zündorf. "Rewriting poor design patterns by good design patterns." Proc. ESEC/FSE. Vol. 97. 1997.

[5]Verfaillie, K. (1992). Variant points of view on viewpoint invariance. Canadian Journal of Psychology/Revue canadienne de psychologie, 46(2), 215.