Architectural Design Document

Introduction

Risk game implemented is a modified version of the Risk board game.This document is used to represent the architecture pattern used in the game.Risk game is a board game.We have used MVC as our design pattern.It was an effort to use extreme programming effort with iterative software development approach to make a modular design and deliver several working coherent modules in small increments or builds.

Model View Controller

Model View Controller design aims at decomposing an interactive system into three components, namely: Model, View and Controller.

Model - The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process.

View - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a push model, in which the view registers itself with the model for change notifications, or a pull model, in which the view is responsible for calling the model when it needs to retrieve the most current data.

Controller - The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections. Depending on the context, a controller may also select a new view -- for example, an action on a particular click event may result in rendering a completely new view.

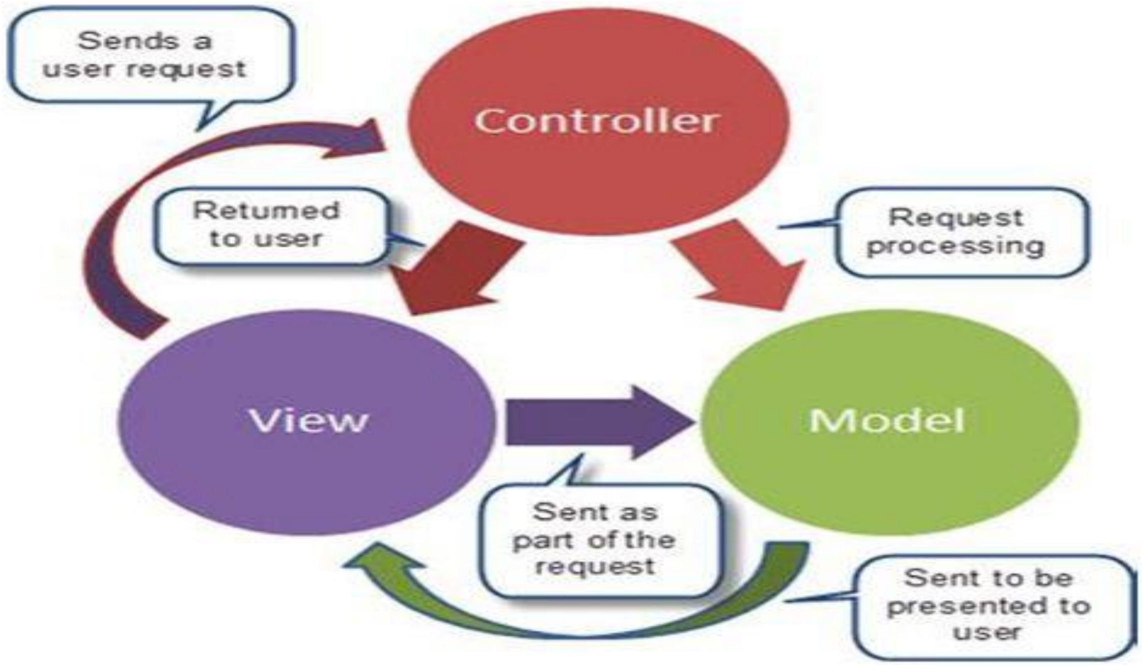

Fig1. Basic MVC architecture

The packages used in the game are shown in Fig.2. Figure 3 shows the architectural pattern followed in the game.

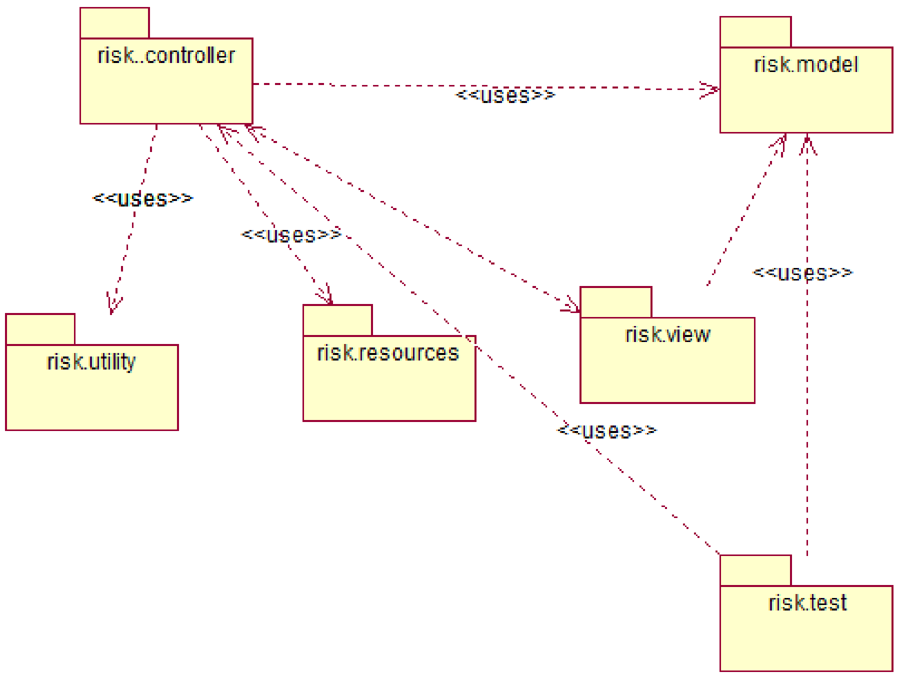

Fig. 2 Package Diagram for overall MVC structure

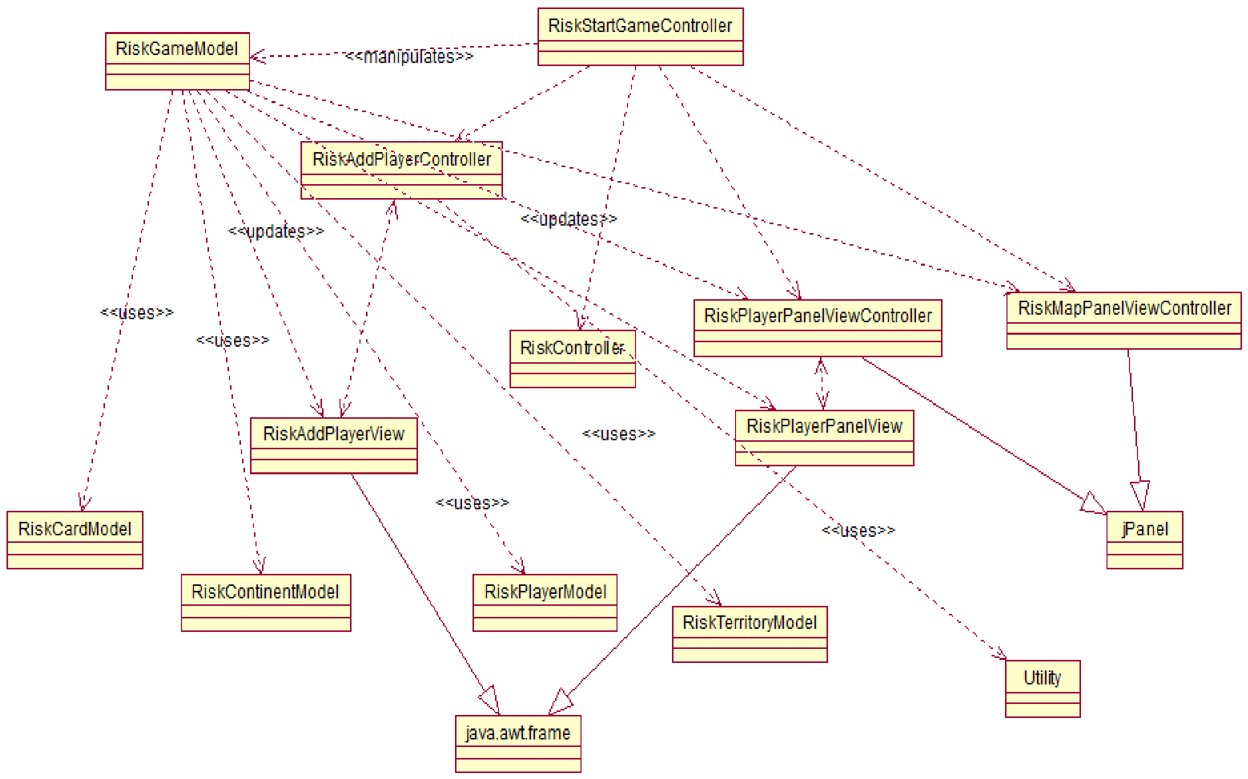

Fig.3 Overview of Class Diagram

Design Patterns Used

- Observer Pattern
- Singleton Pattern

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

This pattern has the following properties:

- Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Encapsulate the core (or common or engine) components in a Subject abstraction, and the variable (or optional or user interface) components in an Observer hierarchy.
- The "View" part of Model-View-Controller.

Singleton Pattern

Singleton design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

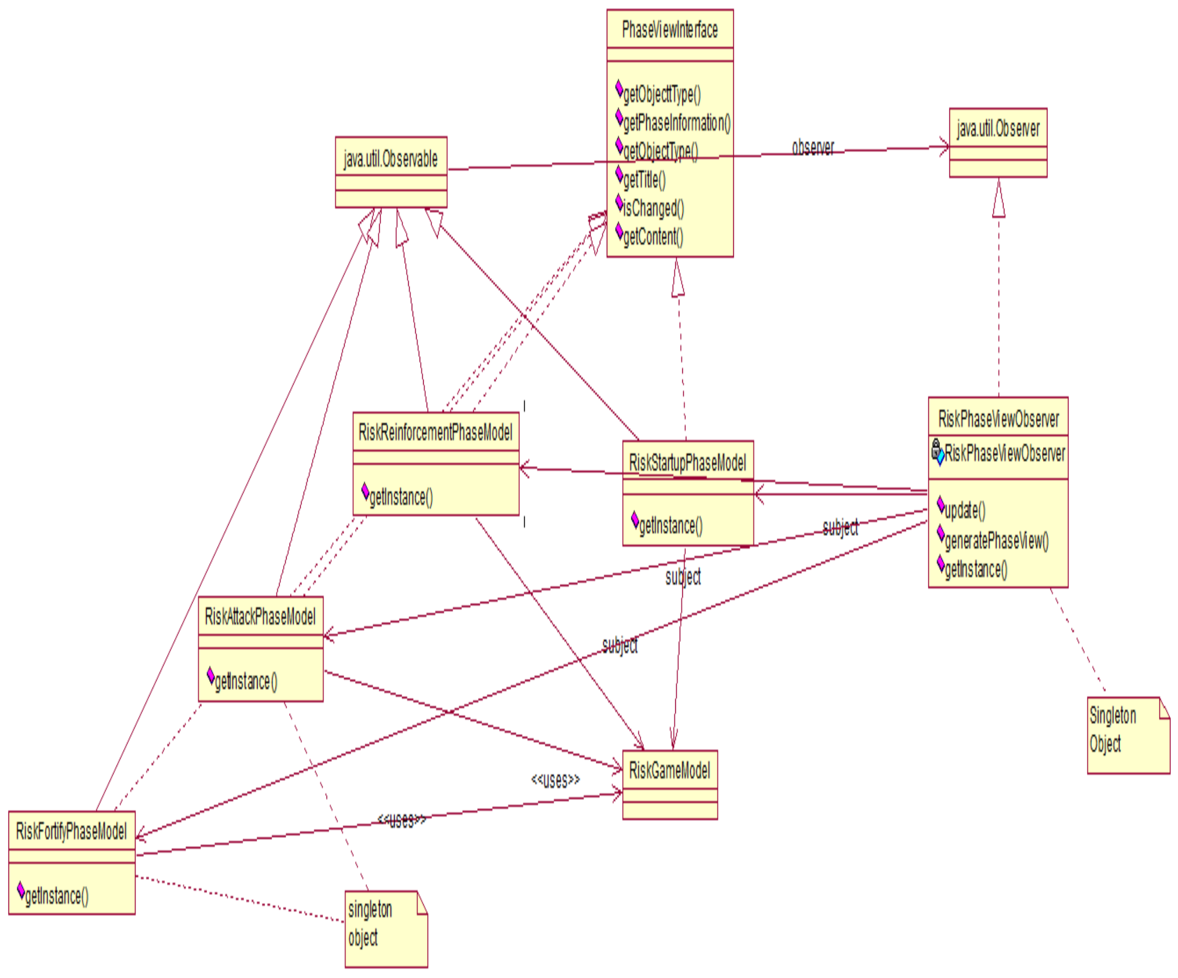

Fig.4 Risk Phase View Observer implemented in Risk game

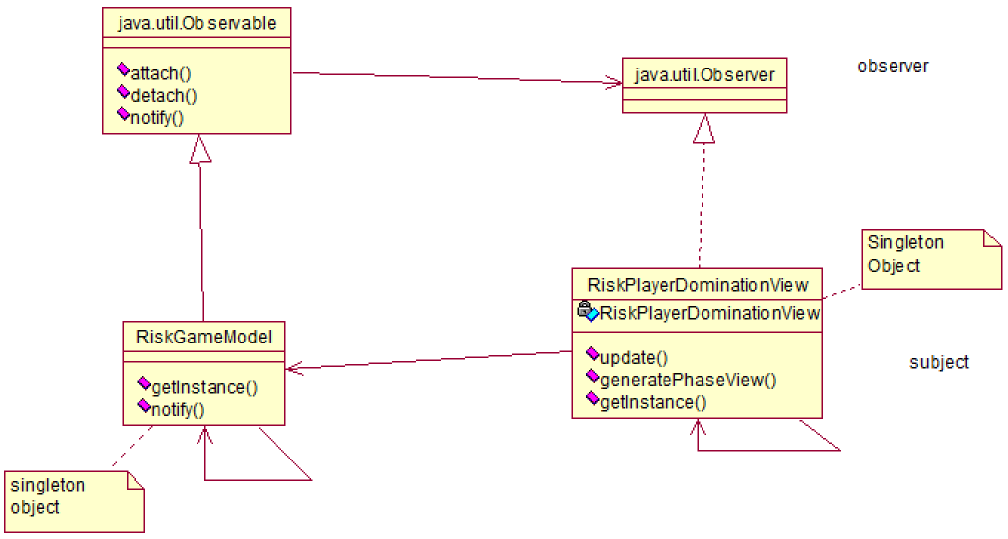

Fig.5 Risk Player Domination View Observer implemented in Risk game

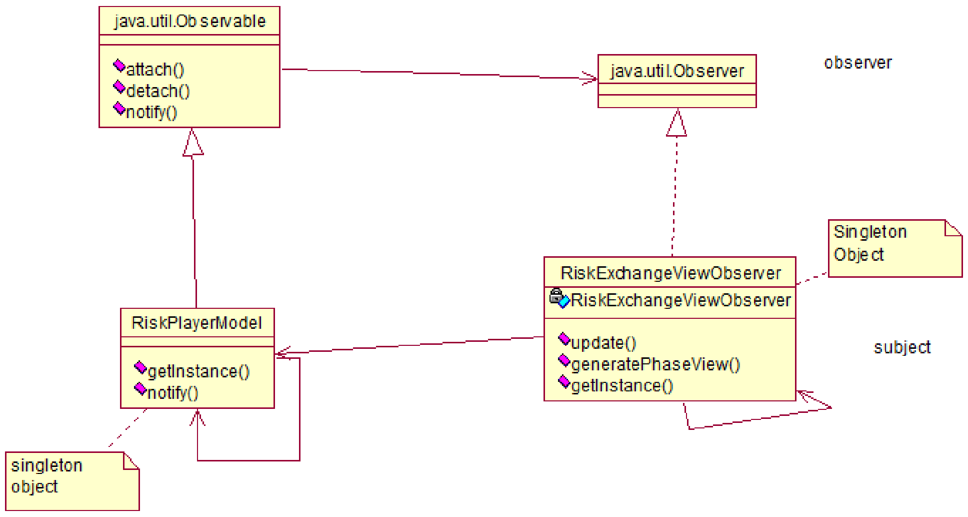

Fig.6 Risk Card Exchange View Observer implemented in Risk game

References

- http://www.oracle.com/technetwork/articles/javase/index-142890.html
- http://www.java-forums.org/attachments/ocmjea/3449d1333636384t-tutorial-review-web-tier-application-architecture-java-architect-exam-c5-conceptualmvc.jpg
- https://sourcemaking.com/design_patterns/observer
- https://en.wikipedia.org/wiki/Observer_pattern
- https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm