System design document for the Riskyspace project (SDD)

Contents

Version: 2.0
Date: 2012-05-15

Author: Daniel Augurell, Jonatan Rapp, Sebastian Odbjer, Alexander Hederstaf

This version overrides all previous versions

1 Introduction

1.1 Design goals

1.2 Definitions, acronyms and abbrevations

- GUI, A Graphical User Interface, what the user will see when using the application.
- JRE, Java Runtime Enviroment, this is needed to run Java application.
- Java, A cross platform programming language used for this application.
- MVC, Model-View-Controller design. This is performed by splitting the View elements and the Contoller and Model elements.
- Socket, Input and Output handling object that is used to build up network connections.
- AWT, Window Toolkit for Java that is used for drawing GUI.
- OpenGL, A platform independent Graphics Library used for drawing the GUI.
- Lobby, The GUI starting point where settings can be managed and the user can start games.

2 System Design

2.1 Overview

The MVC Pattern that is used is based upon an event bus which handles events from both the view and the controller. This design separates the view from the controller and model. If Network play is used there is an additional layer of communication in between two separate event buses on either side of the server/client relation.

The controller is based upon one major event handling class GameManager.java which uses the singleton design pattern. The Server receive commands from clients and sends them to the GameManager with information about the player who sent it, and GameManager then processes or denies the event. GameManager may send events to the Server which then sends it to relevant clients.

The clients listen to the server and the view for events that it sends on in either direction.

The game starts in the Lobby and from there other players can connect, when enough have connected you may start the game.

2.1.1 Rules

Rules are to vary with game modes. At the moment we only have one game mode implemented and therefore all rules currently working is enabled.

2.1.2 The model functionality

The entire model is collected in World.java and consists of a modifiable amount of territories, these territories may be modified in play or before the game is started. World also includes four player's resources and building-queues in all games, but the controller can chose how many to use based on the amount of players in any game. The model can be saved to a file and recreated to an identical world from that file.

2.1.3 Value objects
The Model is mostly mutable through the controller and is almost purely a data storage. A helper logic package with classes take care of most controls and calculations when used by GameManager. Immutable information is easily accessible and can be sent to the view over the eventbus and possibly server introducing no risk of data change directly by the view.

2.1.4 Unique identifiers
We use unique identifiers for most object types as they are serializeable. For Fleet, Planet, and Territory keep unique ID to identify them from others that are in other matters equal.

2.1.5 Event handling
Any class may publish Events to the EventBus. The Server and Client listen to the Bus on each one instance, SERVER and CLIENT. The view publish event to the CLIENT bus and the controller publish to the SERVER bus.

2.1.6 Networking
The server and client each listen to events from controller and view and send these events to the other side. When an event is received over the network it is handled by the GameManager on the server side and by the client on the client side.

2.1.7 Networking in Lobby
When a user start a new LobbyServer it will take an argument for a number of players. It will update all connected clients with information until the number of connected players is reached. When enough players have connected the host's client will get the ability to send start game events, this event will tell all connected LobbyClients to disconnect and then connect to the gameserver.

The gameserver get all IPs that were connected in the lobby and accept only those. The LobbyServer shut down once the gameserver is started.

2.2 Software decomposition

2.2.1 General
Main.java is the application entry class, it will initiate needed classes and start a Lobby.

- view; top level package for all GUI related classes including the main View.
- view.swing; implementation of the view with awt.
- view.opengl; implementation of the view with openGL library jogl
- services, event handlers
- sound, Sound player
- model, The game's model
- logic, Utility classes for calculations and model modification.
- logic.data, Data that is created by logic systems and sent to the views
- data; Settings and Save/Load game Utility
- network; All server and Client implementations

2.2.2 Models
The model contains all information about the current state of the game. The class World is

containing all the information, and used for updates and external communication with the model.

2.2.3 Controllers
We only have one controller class, GameManager, but it has several help classes. The controller is eventbased, so there's no thread running in the background.

2.2.4 Views
Each player has their view. This view is updated through a thread to maintain a smooth fps. The view resources that are displayed are changed based on the events received from the server. The lobby view listen to clicks and changes resources based on that except for when connecting to a lobby server.

2.2.5 Layering
NA

2.2.6 Dependency analysis
Library dependencies are as follows:

- JOGL openGL Library, used for rendering in the openGL implementation of the view.
- JLayer, mp3 / ogg Library used for music playback and sound effects.

2.3 Concurrency issues
We've had quite a few issues with the server and clients with events but we have solved all those we have detected. Other than that we had no issues that laster for more than a few hours before being solved.

A know bug is that on at least Mac OSX, the game can crash if OpenGL is initialized two times on the same run. We think this has to do with OpenGL threading, see references.

2.4 Persistent data management
Not many settings are saved in between games, the only ones being sound on/off and last IP successfully connected to. Saved games are stored in a subfolder of the user home folder. We use autosaves every 2 turns just incase something goes bad. Last two autosaves are saved and others are overwritten to not clutter the load game screen and take up disc space.

2.5 Access control and security
The server can be connected to just knowing the ip and port. But the server handles the incoming objects as Event-objects and if the incoming object isn't an Event it will be discarded.

2.6 Boundary conditions
Application launched and exited as fullscreen application.

3.0 References

- JOGL problem with multiple threads. http://java.net/jira/browse/JOGL-109?focusedCommentId=265299&page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel#action_265299

Appendix I

External Libraries

JOGL openGL library for graphics. http://jogamp.org/jogl/www/
JLayer version 1.0.1 for sound. http://javazoom.net/javalayer/javalayer.html

Domain Model