TEAM GIT GOOD

FREECOL

DESIGN PATTERNS & FIRST CONTRIBUTION

Design Patterns

State Behavioural Pattern

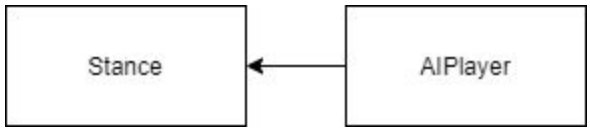

Figure 1.1 - State Behavioural Pattern of FreeCol Stance

The Stance class in FreeCol is an example of the State Behavioural pattern. The Stance class stores the current relationship between players. For example, a player can be “uncontacted” or at “war” with another player. These changes in states in the Stance class allows for different interactions depending on your relationship with the other players. In Figure 1.1, the diagram shows the relationship an AIPlayer has where the AIPlayer will make the decision to attack (or not to) depending on the state of the relationship it has with the player.

Abstract Factory Pattern

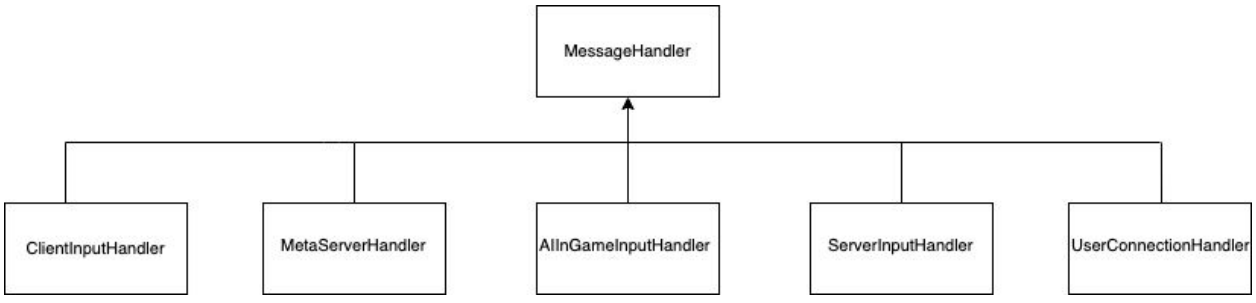

Figure 1.2 - Abstract Factory Pattern

MessageHandler is an interface in FreeCol that follows the abstract factory pattern. It sets out different forms of handling messages, and each object that uses the interface specializes in the way they handle incoming messages, respond to messages, and throw exceptions. Figure 1.2 shows the relationship between MessageHandler and its implementations.

Command Behavioural Pattern

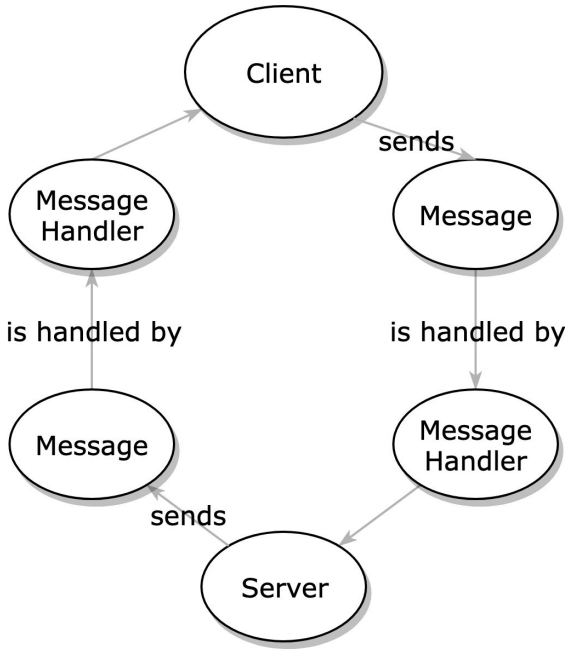

Figure 1.3 Command behavioural pattern

The Message abstract class is an example of the Command Behavioural Pattern. Actions and events in the game are turned into stand-alone objects that extend from Message, and the Message class has many subclasses that handle the different types of Messages that get sent to the InGameController.

Iterator

Figure 1.4 - Iterator

The UnitIterator class is an example of the Iterator pattern. The UnitIterator class stores a list of Player objects and all of the units associated with the players. After a player ends their turn, this class is responsible for updating the iterator to the next player and then loading all units associated with that player. Figure 1.4 demonstrates the relationship between UnitIterator and Player.

Template Method

FreeCol uses the Template Method design pattern in multiple classes. One example is the FreeColConfirmDialog, which is responsible for creating a template for confirmation dialogs for player events in-game runtime. This class is extended by other classes such as the EndTurnDialog, EditOptionDialog, and the LoadingSavegameDialog. The diagram in Figure 1.4 shows all the classes that use FreeColConfirmDialog as a template.

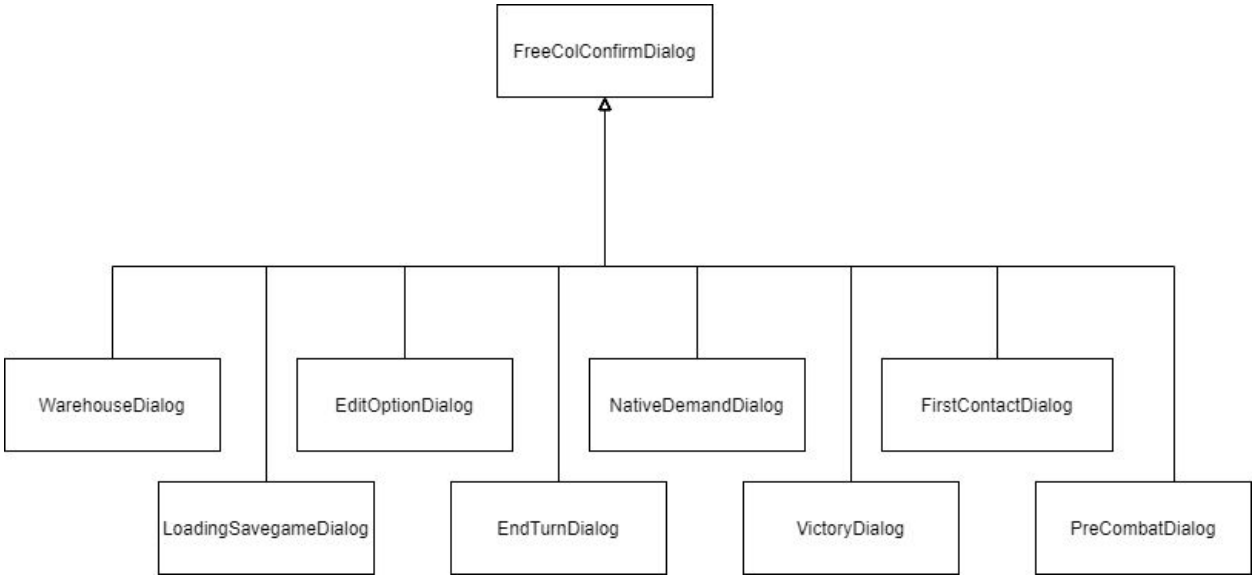

Figure 1.4 - Template Method Diagram

First Contribution

Issue

The issue we focused on was file configuration for custom maps made in the map editor, as brought up in a SourceForge issue thread. The main problem was that custom maps were saved as .fsg files when being saved, but only .fsm files were detected by the map loader when trying to play a game using a custom map and when they were open for editing in the map editor.

Fix

To solve this issue, we decided to change the default file type to .fsm when saving maps. To do this, we had to change the default file name from “my_map.fsg” to “my_map.fsm”. This also causes the file filter to change from “.fsg” to “.fsm” since it uses the default file name’s extension for the file save dialog filter.

We also noticed a related issue where the “.fsm” extension does not get automatically appended to the end of the file name if the user forgets to type it in, so in order to leave the code in a better state than it was before, we added a check to cover that case. The pull request is available at https://github.com/FreeCol/freecol/pull/57. We responded to the thread at the SourceForge forum stating that we contributed with a fix to this issue.