TEAM GIT GOOD

FREECOL
Second Contribution & Test Cases

Second Contribution

About

The contribution we made was to add author and map description information to custom maps in FreeCol's map editor, as requested in a thread on the issue tracker. This contribution involved two main parts: adding an interface in the client which prompts the user to enter an author name and a description, and inputting that information into the XML representation of the map.

Solution

For this improvement request, we created a new dialog class that extended their custom input dialog, so that we are following their current template design pattern. This dialog screen allows users to input the author name and description for the map file that is being saved. This is saved in a common model object we created, and then passed to the server.

The server method that writes the XML file had to be changed to allow the information to be saved in a new element tag. All the code necessary for this implementation can be obtained by checking out the pull request. Additionally, the dialog modal design can be seen in Figure 1.1.

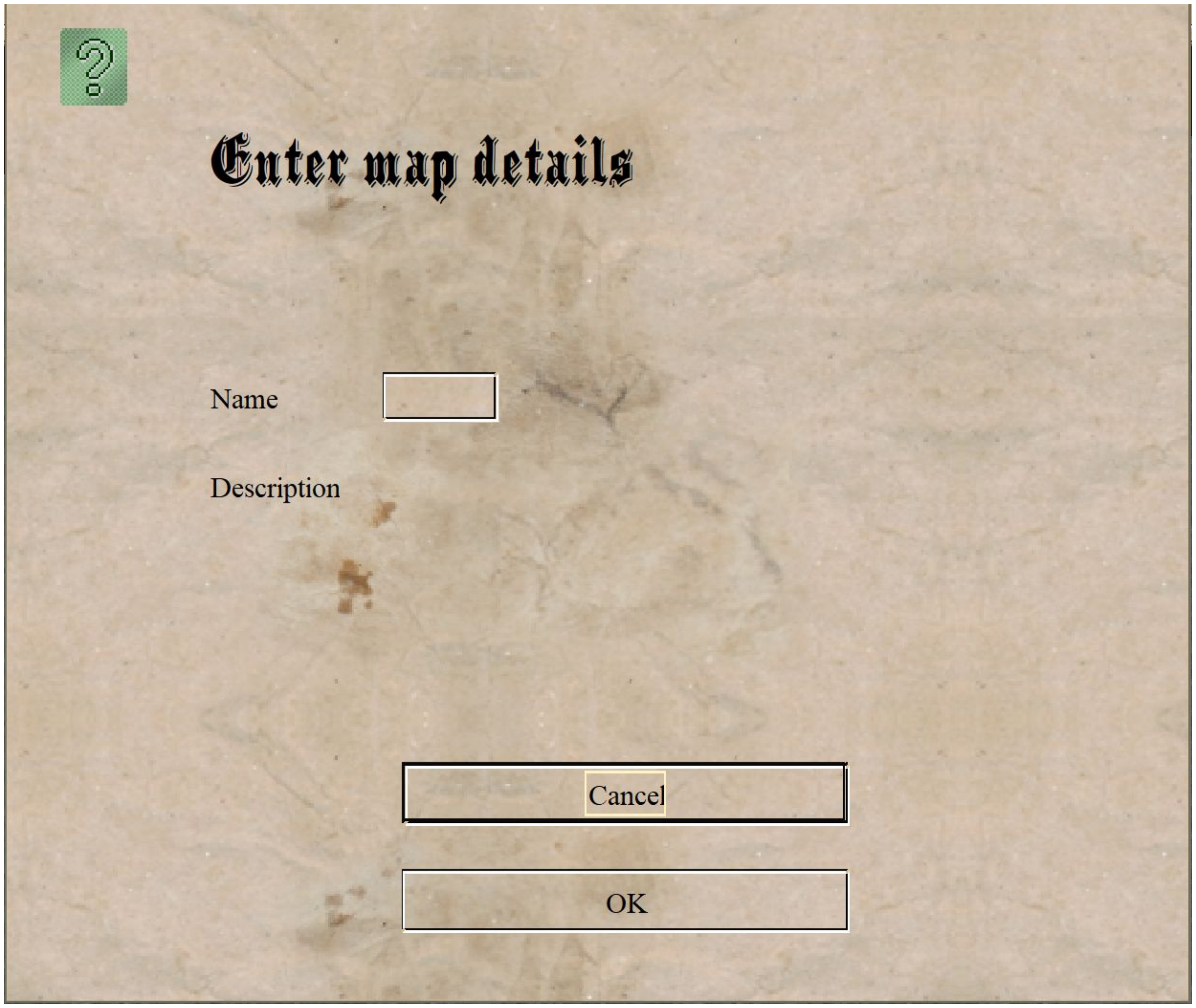

Figure 1.1 - ***Implemented map details dialog***

Discoveries

We learned that the template design pattern is widely used in FreeCol, since it is applied to almost all UI custom elements. Moreover, we learned that all of the UI is built on top of Swing, which is a Java widget toolkit.

Existing Test Cases

Introduction

FreeCol currently has 431 existing test cases, written using JUnit 3, which cover functionality in the client, server, common, and util packages. There are several helper and base classes for these tests such as FreeColTestCase, which adds some FreeCol-specific functionality to the junit.framework package's TestCase class, and ServerTestHelper, which provides stubbed server functionality. The entire test suite for the project has a locale setting which can be changed to run the tests in different locales.

Test Case #1: ServerPlayer - testMarketRecovery

This test case tests the behavior of the market after players buy and sell. It sets up two separate markets, a French market and an English market. It then sells lightly in the French market and heavily in the English market and checks to make sure that prices in the English market have dropped more drastically. Lastly, it advances time to check that the markets both eventually recover.

After studying this test case, we understood more about the expected behaviors of the "market" feature in the game, such as the fact that players' markets are separate, that prices are expected to rise when products are sold in it, and that it recovers over time. Without the test case and included comments, we would not have been able to confirm the intent behind these features without playing the original game.

Test Case #2: MonarchTest

This test cases checks the monarch's ability to raise or lower taxes given the minimum and maximum tax rates. The first test cases check for the monarch's choices when taxes are set at the minimum. If the tax is set at the minimum, then the monarch can only raise taxes through war or tax-acts. The monarch does not have the ability to lower them further. Similarly, when the taxes are set to maximum, the monarch can not raise taxes further and can only decrease taxes through war or tax-acts. Another test case checks for the default behavior where the tax is neither at its maximum or minimum, and the monarch has the ability to raise and/or lower taxes. Interestingly, the taxes are not applied to a player that is labeled a rebel.

After studying this test case, we learned that taxes have an impact on all players, except rebels, and that the monarch has limited taxing powers in relation to minimum and maximum tax rates.

Test Case #3: MissionAssignmentTest - testImpossibleConditionsForTargetSelection

This test case tests the ability, or lack of ability, for one unit to attack another unit. By studying the test case, we can learn a couple of conditions that must be preset before an attack command can even occur.

The first unit is that a map, with a board, must exist, and the two pieces must be next to each other. We can also learn that players must be in a state of war before attacking. Players cannot target their own units to attack, and land units cannot attack naval units. If two nations who have the War stance target an enemy unit who are next to them, then an attack command is valid, and the game will process the outcome.

New Test Cases

SaveMapEditorGame Test Case

Since there was no existing test case for it, we created a test case checking that a map editor game could be saved. The test case sets up a new server game, test map, and client, and tries to save that map. It then checks to see if the map was successfully saved before deleting it.

You can view the pull request for this test case here.

Disaster Test Case

FreeCol has the ability to test whether a disaster occurs in the game. There are a wide variety of disasters, however, there only exists two test cases: floods and tornadoes. We extended the test suite to include more disasters to ensure that they can occur: disease and sandstorm. Following their template pattern, we established that when creating a disaster that it is not null, the effects work, that the disaster has an id, and that the probability of the disaster occurring is greater than zero. The test cases ran with no issues and have been integrated within the existing DisasterTest class.

You can view the pull request for this test case here.

Simple Move Test Case

In this test case, FreeCol tests the movement from one tile to the next. Upon further inspection of the code, the test was not actually testing this functionality, but rather tested if tile 2 was North East of tile 1, and then proceeded to move without actually testing if the movement part was successful or not. We added additional assertions to check if the automated simple movement is able to occur. The test case integrated with the rest of the test cases.

You can view the pull request for this test case here.