# SOEN 6441 – Advanced Programming Practices – Winter 2021

# Warzone Project - Build 3 – Refactoring – Team 1

## Potential Refactoring Targets

The following list of refactoring targets have been taken from the new requirements established in build 3, and based on pain points and inconsistencies encountered during the development of build 2. Also, several of the targets were taken from code analysis done using PMD Source Code Analyzer (https://pmd.github.io/).

1. \*Refactor Strategy pattern to player behavioral strategies.
2. \*Refactor Adaptor pattern for loading and saving of Domination and Conquest map types.
3. \*Rename classes
4. \*Cleanup unused imports
5. \*Remove unused variables
6. Cleanup unused methods
7. Use AssertSame instead OfAssertTrue in unit tests
8. Use AssertNull instead of AssertTrue in unit tests
9. Use AssertEquals instead of AssertTrue in unit tests
10. Rename methods
11. Remove unused formal parameter(s)
12. Use collection IsEmpty instead of size() == 0 (or size() != 0, size() > 0, size() < 1)
13. Eliminate unnecessary constructors
14. Change use of ArrayList<> to List<> wherever best
15. Refactor any formal parameter that does not conform to standards

## Actual Refactoring Targets

The list of actual refactoring taken from the target list above were chosen mainly because of the new requirements established in build 3, and on the greatest pain points and inconsistencies encountered during the development of build 2.

### Refactor Strategy pattern to player behavioral strategies

This refactoring was chosen because before refactoring the game supported only one type of player, ie. human. In build 3 the requirements asked for other types of computer based players, and therefore required a different better method of processing player commands. The method chosen was the strategy pattern.

#### Before / After Refactoring:

Before refactoring the issue\_order() method was processed one way in the Player class, i.e. directly to a human player. After refactoring the Player object is initialized with a strategy. During the issue\_order() phase the player class invokes the respective strategy to get back an order.

#### Unit Tests:

The following unit tests added in build 2 in GameStartupControllerTest:

|  |
| --- |
| test\_processGameStartupCommand\_gameplayer1\_valid  test\_processGameStartupCommand\_gameplayer2\_valid  test\_processGameStartupCommand\_gameplayer3\_valid  test\_processGameStartupCommand\_gameplayer4\_valid  test\_processGameStartupCommand\_gameplayer5\_valid  test\_processGameStartupCommand\_gameplayer\_mixed\_valid  test\_processGameStartupCommand\_gameplayer\_mixed\_1\_valid  test\_processGameStartupCommand\_gameplayer\_mixed\_1\_invalid |

These unit tests verify the gameplayer command and that the proper strategies were applied to each respective player; i.e.: PlayerAggressiveStrategy, PlayerBenevolentStrategy, PlayerCheaterStrategy, PlayerHumanStrategy, PlayerRandomStrategy

### Refactor Adaptor pattern for loading and saving of Domination and Conquest map types

This refactoring was chosen in an effort to support the new requirement that the game be able to load and save Conquest type map files in addition to Domination maps. The idea is that the existing Domination processing code be changed as little as possible, while tailoring to the different format of the Conquest map format.

#### Before / After Refactoring:

Before refactoring the application only supported the Domination map file format. The code made direct calls to the MapIoDomination class to load or save the map.

After the refactoring the code still makes direct calls to the MapIoDomination class for domination style maps, but for Conquest maps the system now uses the adaptor class MapIoAdaptor. MapIoAdaptor, invokes the appropriate method in MapIoConquest, and translates the conquest style map into the format returned by MapIoDomination (i.e. DominationMap).

The system scans the desired map file to detect what format the map file is in. Depending on the type either the MapIoDomination or the MapIoAdaptor class is instantiated. For savemap the user provides a parameter in savemap that indicates the desired output format.

#### Unit Tests:

The following unit tests were added to MapTest:

|  |
| --- |
| test\_loadmap\_command\_1  test\_loadmap\_command\_2  test\_saveMap\_1  test\_loadmap\_domination\_file\_1  test\_loadmap\_conquest\_file\_1 |

### Rename class names

This refactoring was chosen because some of the newly implemented patterns changes the role of certain classes and it makes the application code easier to read.

#### Before / After Refactoring:

Before refactoring the application code was difficult to read as some of the class names were too long. After refactoring the application code was easier to read.

Changes involved renaming of the following classes:

* GamePlayController -> SingleGameController

#### Unit Tests:

Existing unit tests were able to cover the refactoring. Invalid or unchanged class names are caught by the compiler.

### Cleanup unused imports

This refactoring was chosen because there were many warnings (130 total) indicating that there were unused imports being specified in the application. This caused pollution in the warnings list window.

#### Before / After Refactoring:

Before refactoring the application was compiling with 130 import warnings. After the refactoring there were none.

#### Unit Tests:

Existing unit tests were able to cover the refactoring. Invalid or unchanged changes are caught by the compiler.

### Remove unused variables

This refactoring was chosen because there were many warnings (42 total) indicating that there were unused variables declared in the code but not being used.. This caused pollution in the warnings list window, and used up memory unnecessarily.

#### Before / After Refactoring:

Before refactoring the application was compiling with 42 ‘The value of the variable ????? is not used’ warnings. After the refactoring there were none.

#### Unit Tests:

Existing unit tests were able to cover the refactoring. Invalid or unchanged changes are caught by the compiler.