Refactoring Document for Build 3

To find areas for refactoring, we picked the parts of the code that are most important to how our project works.

Possible areas for code refactoring include the following:

GameEngine start_game(),

Player issue_order(),

GameEngine getNumOfReinforcements(),

MapEditor readMap(),

Order execute(),

Player class structure,

MapEditor editMap(),

WarMap edit functions,

MapEditor saveMap(),

MainMenu loadMap()/StartUp loadMap()

Postload editMap(),

Postload saveMap(),

OrderExecution phase,

IssueOrder phase,

MapEditor showAllMaps().

Refactoring Changes for Build 3

To select the following refactoring targets, we chose functions that made it easier to implement the requirements of build 3. Before refactoring, some of these functions were not well-suited for meeting those requirements.

The following functions were selected and refactored to enhance code clarity and make it easier to meet the requirements of Build 3:

1. GameEngine start_game(),
2. Player issue_order (),
3. GameEngine addPlayer(),
4. MapEditor readMap()/editMap(),
5. MainMenu loadMap()/StartUp loadMap()

1.Refactoring for (GameEngine start_game()):

The refactoring was required after Tournaments were introduced in build 3. In order to manage the tournaments, additional commands had to be accepted and processed. Aside from the tournament mode, in single-player mode, the addition of behavior strategies that do not require human input necessitated modifying this method. Specifically, when a non-human behavior player is involved, we no longer manually process their orders and instead, automatically perform all necessary actions.

This class is tested by the following test cases:

testAssignCountries,

testAddPlayer,

testRemovePlayer,

testGetNumOfReinforcements,

testEditPhaseTransition,

testPlayPhaseTransition,

testStartUpValidation.

2.Refactoring for (Player issue_order ()):

The introduction of behavior strategies meant that the order issuing process was entirely dependent on the behavior strategy assigned to the player. Therefore, the primary purpose of the refactoring was to implement the Strategy Pattern, which allowed control of order issuance to be passed to the appropriate behavior that the player was initialized with.

Before the refactoring, all the order issuance logic was managed by the Player class, as there was only human behavior. The previous implementation was then moved to the HumanBehaviorStrategy, and now, the Player class is only responsible for delegating the order issuance control to the corresponding behavior.

This class is tested by the following test cases:

testNextOrderEmptyList,

testNextOrder,

testInvalidCountry

3.Refactoring for (GameEngine addPlayer()):

With the introduction of player behavior strategies, it became necessary to assign a specific behavior to each player. Previously, the addPlayer method simply created a player using the player's name as input. After the refactoring, this method now also handles assigning the appropriate behavior to each player. As a result, two overloaded versions of the addPlayer method were created: one that accepts the behavior strategy as a parameter, and another that prompts the user to input the strategy, which is then assigned accordingly.

This class is tested by the following test cases:

testAssignCountries,

testAddPlayer,

testRemovePlayer,

testGetNumOfReinforcements,

testEditPhaseTransition,

testPlayPhaseTransition,

testStartUpValidation.

4.Refactoring for (MapEditor readMap()/editMap()):

These two functions modified the current WarMap of the game engine by passing the game engine's WarMap object to the functions. However, this approach was incompatible with the implementation of the adapter pattern, so changes were necessary.

After the refactoring, both readMap() and editMap() no longer take a WarMap as input. Instead, they now return a WarMap object. This adjustment enables the Adapter class to return a WarMap object by using the adaptee's readMap and editMap functions, thereby properly implementing the Adapter pattern.

This class is tested by the following test cases:

testEditPhaseTransition,

testStartUpValidations,

The WarMapTest class also tests indirectly, as it will fail if the readMap() function fails.

5.Refactoring for (MainMenu loadMap()/StartUp loadMap()):

Originally, these functions could only load domination map files, and attempting to load a conquest map file would not work. Furthermore, the refactored readMap() function of the MapEditor was incompatible with the old implementation of loadMap(), so refactoring was necessary.

After the refactoring, the loadMap() method can now load both domination and conquest map files. It determines which map editor to use (either the original or the conquest map editor) based on the specified map file type. Additionally, it was refactored to use the new method signature for readMap().

This class is tested by the following test cases:

testEditPhaseTransition,

testPlayPhaseTransition

testStart Validations.