Refactoring Document for Build 2

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,

WarMap class structure,

WarMap showMap(),

WarMap showMap(List of Players),

WarMap validateMap(),

MapEditor editMap(),

MainGameLoop run_game_loop(),

MapEditor editMapEntry(),

Player next_order(),

WarMap edit functions.

Refactoring Changes for Build 2

To select the following refactoring targets, we chose functions that made it easier to implement the requirements of build 2. 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 2:

1. GameEngine start_game(),
2. Player issue_order (),
3. Order execute(),
4. MainGameLoop run_game_loop(),
5. MapEditor editMapEntry()

1.Refactoring for (GameEngine start_game()):

We decided to refactor the play function in GameEngine because the original version didn't support the State pattern well. Before refactoring, play only controlled the program during the startup phase, and after that, it stopped receiving commands. The control was then passed to either the MapEditor or MainGameLoop. After refactoring, the play function keeps control throughout the entire game and manages all phases internally, making it much easier to apply the State pattern.

This class is tested by the following test cases:

testAssignCountries,

testAddPlayer,

testRemovePlayer,

testGetNumOfReinforcements,

testEditPhaseTransition,

testPlayPhaseTransition,

testStartUpValidation.

2.Refactoring for (Player issue_order ()):

Refactoring was necessary because the original implementation of the issue_order function in the Player class did not support the Command pattern well, and it couldn't be integrated effectively with the State pattern. Originally, the function controlled the entire issue order phase, holding control until all players had issued all their orders.

After refactoring, the issue_order function is now called repeatedly and handles one order at a time. It takes input from the player and creates a single order using the appropriate type, making it compatible with the Command pattern.

This class is tested by the following test cases:

testNextOrderEmptyList,

testNextOrder,

testInvalidCountry,

testCannotDeployMoreArmiesThanReinforcements,

testBlockadeIssueOrderValidInput,

testBlockadeIssueOrderInvalidCountry,

testBlockadeIssueOrderNoBlockadeCard,

testBlockadeCommandExecution,

testAdvanceCommandExecution,

testAdvanceCommandExecutionWithInvalidSourceCountry,

testBombCommandExecution,

testBombCommandExecutionWithInvalidCountry,

testBombCommandExecutionWithoutBombCard,

testDiplomacyCommandExecution,

testDiplomacyCommandExecutionWithoutDiplomacyCard,

testDiplomacyCommandExecutionWithInvalidTargetPlayer,

testAirliftCommandExecution,

testAirliftCommandExecutionWithInvalidSourceCountry

3.Refactoring for (Order execute()):

Refactoring was necessary because the original implementation assumed all orders were deploy orders and didn't use the Command pattern. We refactored the code to support all order types using the Command pattern.

Previously, it only worked for deploy orders. Now, it calls the appropriate execute function based on the order type. This required adding individual execute methods for each order type, and modifying the main Order class to delegate execution accordingly.

This class is tested by the following test cases:

testExecute,

testExecuteBlockadeOrderNonExistingCountry,

testExecuteBlockadeOrderCountryRemoved,

testExecuteBlockadeOrderNumOfArmies,

testBlockadeCommandExecution,

testAdvanceCommandExecution,

testAdvanceCommandExecutionWithInvalidSourceCountry,

testBombCommandExecution,

testBombCommandExecutionWithInvalidCountry,

testBombCommandExecutionWithoutBombCard,

testDiplomacyCommandExecution,

testDiplomacyCommandExecutionWithoutDiplomacyCard,

testDiplomacyCommandExecutionWithInvalidTargetPlayer,

testAirliftCommandExecution,

testAirliftCommandExecutionWithInvalidSourceCountry

4.Refactoring for (MainGameLoop run_game_loop()):

Refactoring was needed because the original implementation of the MainGameLoop's play function held control over the entire game loop, which made it difficult to implement the State pattern.

Before refactoring, the play function managed the whole game play process. After refactoring, its functionality was moved into the Play phases to align with the State pattern. Now, only one gameplay command is executed at a time before control is returned to the GameEngine, rather than the MainGameLoop holding control throughout.

This class is tested by the following test cases:

testGetNumOfReinforcements,

testPlayPhaseTransition,

testStartUpValidations,

testAssignCountries

5.Refactoring for (MapEditor editMapEntry()):

Refactoring was necessary because the original implementation of map editing held full control of the program, which conflicted with the State pattern.

Before refactoring, the map editor ran in a loop to complete all editing tasks at once. After refactoring, the editing functionality was moved into the Edit phases to support the State pattern. Now, only one edit operation is performed at a time before control returns to the GameEngine, rather than the map editor holding control the whole time.

This class is tested by the following test cases:

testIsContinentConnected,

testIsConnectedGraph,

testisEmptyContinent,

testingisCountryNoNeighbor,

testEditPhaseTransition