Build #3 – Refactoring Document

To identify the potential refactoring targets in the previous build(#2), following parameters were laid emphasis on:

- Methods containing more than one logic.
- Similar logic being called at multiple places.
- Classes with many methods.
- Longer nesting and wraps of conditional logic.

Refactoring Targets:

1. **Strategy Pattern**
2. **Issue And Execution Order**
3. **Adapter Pattern**
4. **Extracted Game Specific methods to GameService**
5. **GameState**
6. **Previous methods modification to suit tournament logic**
7. **Card Assignment**
8. **End Game Logic**
9. **Tournament Parsing**
10. **Sequencing of Commands**

Potential Refactoring Targets:

11. createOrder Function in Strategies
12. Formatting Functions common to MapView and TournamentView
13. Tournament Main Method
14. Mapping Player Strategies and Players in tournament
15. Common Player Logic in methods.

<u>Adapter Pattern:</u>

Before: Only one type of map file format can be created and accessed throughout the game.

After: Read/Write operations can now be performed on two types of map file formats – original domination and conquest refactoring original map loading format to adapter pattern.

Reason: More options available for users to choose and create maps from.

Added Test Cases:

1. testReadMapFile – MapFileReaderTest
2. testReadConquestFile – ConquestMapFileReaderTest
3. testEditMap - ConquestMapFileReaderTest

Modified Test Cases (if any): None

<u>Strategy Pattern:</u>

Before: Support of a single user-input command based format to take orders.

After: Five types of Player behaviours - Aggressive, Benevolent, Cheater, Human and Random in accordance to their described behavior, previous command input logic shifted to human player.

Reason: To accommodate the given behavior patterns.

Added Test Cases:

1. testOrderCreation- Aggressive, Random, Cheater and Benevolent
2. testStrongestCountry – AggressivePlayer
3. testWeakestCountry – BenevolentPlayer
4. testWeakestNeighbour - BenevolentPlayer
5. testUnallocatedArmiesDeployment - CheaterPlayer
6. testCheaterOwnsAllEnemies - CheaterPlayer

Modified Test Cases (if any): None

Issue And Execution Order

Before: Issue Method to Accept Commands From User Input, execute orders to execute orders and check if any player conquered all countries

After: Issue Method to accept commands from automatic players, execute orders to keep track of winners and players out of the game in each gamestate.

Reason: To accommodate the given behavior patterns.

Added Test Cases:

7. testOrderCreation- Aggressive, Random, Cheater and Benevolent
8. testStrongestCountry – AggressivePlayer
9. testWeakestCountry – BenevolentPlayer
10. testWeakestNeighbour - BenevolentPlayer
11. testUnallocatedArmiesDeployment - CheaterPlayer
12. testCheaterOwnsAllEnemies - CheaterPlayer

Modified Test Cases (if any): None

Extracted Game Specific methods to GameService

Before: Individual Phase Classes Supporting Command Calls

After: GameService Class to Load and Save the current Game.

Reason: To implement the load and save game commands

Added Test Cases:

1. testPerformSaveGameValidCommand
2. testPerformLoadGameValidCommand

Modified Test Cases (if any): None

GameState

Before: GameState keeps track of players and logs related to players

After: GameState keeps track of winner, losing player and number of turns being played in each game

Reason: To support tournament based game play

Added Test Cases: None

Modified Test Cases (if any): None

Previous methods modification to suit tournament logic.

Before: Player's issue order to call user based input method and Y/N Input for checking more orders

After: Check for More Orders to Work on random boolean based logic to give next order or not

Reason: To keep a bound on how many commands automatic player can take in a turn

Added Test Cases: None

Modified Test Cases (if any): None