Refactoring Document Build No. 3

The focus in identifying potential refactoring targets in the prior build (#2) was on highlighting specific parameters.

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

In the prior build (#2), the potential areas for improving code were outlined:

1. Strategy Pattern refinement.
2. Execution order and issue resolution.
3. Adapter Pattern adaptation.
4. Relocating game-specific methods to GamePlayService.
5. Evaluating GameState utilization.
6. Modifying methods for tournament logic.
7. Enhancing card assignment.
8. Improving end-game logic.
9. Optimizing tournament parsing.
10. Sequencing commands efficiently.

Additionally, the following areas were identified for potential improvement:

11. Refinement of createOrder function in Strategies.
12. Streamlining formatting functions for MapView and TournamentView.
13. Reviewing the main method for tournaments.
14. Aligning player strategies and mapping in tournaments.
15. Consolidating common player logic across methods.

Adapter Pattern:

Before: Previously, the game only allowed the creation and access of a single map file format.
After: Now, there's support for reading and writing two distinct map file formats: the original domination and conquest formats.
– original domination and conquest refactoring original map loading format to adapter
pattern.
Reason: This enhancement involved refactoring the original map loading format to implement the adapter pattern. This change was made to offer users a broader array of options for selecting and generating maps.

Added Test Cases:
1. testReadMapFile – MapFileReaderTest
2. testReadConquestFile – ConquestMapFileReaderTest
3. testEditMap - ConquestMapFileReaderTest
Modified Test Cases (if any): None

Strategy Pattern:
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 behaviour, previous command input logic
shifted to human player.
Reason: To accommodate the given behaviour 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

Strategy Pattern:

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 behaviour, previous command input logic
shifted to human player.

Reason: To accommodate the given behaviour 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

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