Team 3

Refactoring Document for Build #3

In the analysis of the previous build (#2), focus was placed on identifying potential areas for refactoring based on the following criteria:

- Methods with multiple logical operations.
- Repetitive logic appearing in multiple locations.
- Classes containing numerous methods.
- Complex nesting and conditional logic structures.

Refactoring Targets:

1. Implementation of the Strategy Pattern.
2. Addressing Issues and Execution Order.
3. Integration of the Adapter Pattern.
4. Extraction of Game-Specific methods into GameService.
5. Management of GameState.
6. Adjustment of methods to align with tournament logic.
7. Refinement of Card Assignment.
8. Optimization of End Game Logic.
9. Streamlining Tournament Parsing.
10. Sequencing Commands effectively.

Additional Potential Refactoring Targets:

11. Refinement of the 'createOrder' Function within Strategies.
12. Consolidation of Formatting Functions shared by MapView and TournamentView.

Team 3

13. Streamlining the Tournament Main Method.

14. Improving the Mapping of Player Strategies and Players in the tournament.

15. Centralizing Common Player Logic within methods.

Adapter Pattern Refactoring:

Before: Limited support for a single type of map file format.

After: Expanded support for two types of map file formats, utilizing the Adapter Pattern to accommodate original domination and conquest formats.

Reason: Enhancing user options for map creation and access.

<u>Associated Test Cases:</u>

- Separate Testcases were added for conquestMap amd simpleMap.

Strategy Pattern Refactoring:

Before: Support for a single user-input command format.

After: Implementation of five distinct Player behaviors (Aggressive, Benevolent, Cheater, Human, Random) with corresponding logic adjustments, particularly shifting previous command input logic to the Human player.

Reason: Adapting to diverse player behavior patterns.

<u>Associated Test Cases:</u>

- Various tests covering order creation, country strength evaluation, and deployment strategies for different player types.

Team 3

Issue And Execution Order Refactoring:

Before: Handling user input commands and executing orders.

After: Revising the issue method to accommodate commands from automatic players and refining order execution to track game state changes such as winners and eliminated players.

Reason: Aligning with specified behavior patterns and game progression.

<u>Associated Test Cases:</u>

- Tests covering order creation and various player strategies.

Extraction of Game-Specific Methods to GameService:

Before: Command calls managed by individual Phase Classes.

After: Centralizing game-related operations within the GameService Class, particularly focusing on loading and saving game states.

Reason: Facilitating the implementation of load and save game functionalities.

<u>Associated Test Cases:</u>

- Tests validating the functionality of save and load game commands.

Team 3

GameState Refactoring:

Before: Tracking player information and related logs.

After: Enhancing GameState to monitor game-specific metrics such as winners, eliminated players, and turn counts.

Reason: Supporting tournament-based gameplay requirements.

Previous Methods Modification for Tournament Logic:

Before: User-based input method for issuing orders and checking for additional commands.

After: Introducing a randomized boolean logic to limit automatic player commands per turn.

Reason: Establishing constraints on automatic player actions during a turn.

<u>Associated Test Cases:</u>

Test cases were added and modified accordingly to ensure the robustness of the implemented changes."