"Refactoring Analysis Document - Iteration #2

In reviewing the previous build (#1), our focus was on identifying potential areas for refactoring based on several parameters:

- Identification of methods with multiple logic flows.
- Detection of repetitive logic across multiple locations.
- Examination of classes with extensive method counts.
- Analysis of deep nesting and conditional logic.

Refactoring Targets:

1. Adaptation of existing methods to align with game logic.
2. Implementation of the State Pattern.
3. Refinement of the GameState management.
4. Enhancement of game progress logging.
5. Optimization of end game logic.
6. Transformation of the Order class into an interface to facilitate card functionality.

Potential Refactoring Targets:

7. Structuring formatting functions for Phases and Orders.
8. Handling unanticipated exceptions or Error situations.
9. Augmenting test case coverage.
10. Reduction of code complexity.
11. Integration of comprehensive code commenting.
12. Mitigation of large class sizes.
13. Simplification of nested conditionals.

14. Localization of global variables.
15. Streamlining dependency management.

State Pattern:

Original: The GameEngine managed the entire gameplay process.

Refactored: Segmented into distinct phases for Start Up, Order Issuance, and Order Execution.

Rationale: Enhances code clarity and documentation.

Test Cases Added: None

Modified Test Cases: None

GameState:

Original: Tracked map and player information exclusively.

Refactored: Expanded to include player-related logs.

Rationale: Necessary for comprehensive game monitoring.

Test Cases Added: None

Modified Test Cases: None

Logging of Game Progress:

Original: Relied on print statements for user notifications.

Refactored: Implemented logging for improved debugging and monitoring capabilities.

Rationale: Enhances gameplay tracking and error identification.

Test Cases Added: None

Modified Test Cases: None

End Game Logic:

Original: Utilized an exit command for game termination.

Refactored: Game concludes when a single player conquers all countries on the map.

Rationale: Provides a definitive game endpoint.

Test Cases Added:

1. testEndOfTheGame() - Ensures all countries are conquered by a single player.

Modified Test Cases: None

Modification of Existing Methods to Suit Game Logic:

Original: Game Engine handled all gameplay operations.

Refactored: Code adjusted to incorporate distinct phases and initialization processes.

Rationale: Alignment with coding standards and improved maintainability.

Test Cases Added: None

Modified Test Cases: None

Conversion of Order Class to Interface Type:

Original: Class executed and deployed orders while updating game state.

Refactored: Interface introduced to manage deployment and advancement operations.

Rationale: Enhances abstraction between deployment and attacking functionalities.

Test Cases Added: None

Modified Test Cases: Removed order test class as it's now an interface.