**A close-up of a logo

Description automatically generated**

**GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE**

**SOEN 6441 – Advanced Programming Practices**

**Project Build #3**

**Course Instructor**: Joey Paquet

**Group Number**: 17

|  |  |
| --- | --- |
| **Name** | **Student ID** |
| Hritik Nandanwar | 40268450 |
| Sweta Sarat | 40263287 |
| Vishav Teji | 40266275 |
| Muqaddaspreet Singh | 40276333 |
| Ushnish Chakravarty | 40270673 |

**Project Build #3**

**Build 3 – Refactoring Document**

The analysis was based on specific criteria, which included the following target points:

* Methods that contain multiple sets of logic.
* Similar logics are repeated in various places.
* Classes which have too many methods.
* Extensive use of nesting and complex conditional logic structures.

**Refactoring Targets:**

1. It is recommended to change the access level of the class variables in the "showmap" class, which include `d\_players`, `d\_gameState`, `d\_map`, `d\_countries`, and `d\_continents`, to private. Furthermore, it is advisable to create public getter methods to allow for controlled access to these variables and ensure encapsulation.
2. Error handling can be enhanced by making changes to the "loadFile" method

present in the "loadmap" function. The modification should include the throwing of an IOException when an error happens during file reading. The "loadFile" method is modified within the "loadmap" function to throw an IOException when an error occurs during file reading

1. The "playerservice" method called "updatePlayers" is responsible for several tasks, including parsing a player's name, managing player-related actions (such as adding or removing players), and updating the game state. To enhance the code structure, the method can be broken down into smaller and more focused functions that can handle each of these duties individually.
2. When removing a player as part of the refactoring process in the "playerservice," it's good to create a custom exception that should be thrown if the player is not found.
3. Consider dividing the "startup" class and other similar phases into smaller, more specialized classes to comply with the Single Responsibility Principle and enhance code organization. Each of these new classes should have a specific responsibility during the game's startup phase, such as input handling, map editing, game player actions, and so forth. This approach can improve the modularity and maintainability of the code.
4. The extensive "showMap" method would benefit from decomposition into smaller, more

specialized methods, each responsible for handling specific tasks.

1. The process of creating commands for game actions in Warzone can be improved by adopting the Builder pattern. This approach will allow for clear and customizable construction of commands, resulting in enhanced readability and flexibility.
2. Refactor to use Optional Instead of Null in certain methods such as findCountryByName

in Airlift.java

1. Update "executeLoadMap" method to use dependency injection for `d\_mapService`. This allows any `MapService` implementation to be provided during invocation, enhancing flexibility and testability.
2. To enhance the readability and maintainability of your code, it is good to utilize the Java Stream API wherever possible for streamlining loops and operations on collections.

**Potential Refactoring Targets:**

1. Strategy pattern
2. Adapter pattern
3. GameService class can be seperated by extractiong game state specific methods including loadGame and saveGame
4. IssueOrderphase method can be randomly updated to determine whether next order is required or not for handling automatic players, as user won’t play a role in this gameplay phase.
5. The issueOrder method can be modified to receive new orders based on the player's strategy, instead of relying solely on user input.

**11. Strategy pattern:**

**Before:** All the orders were given by the user for each player initially

**After:** The player behaviors can be categorized into five distinct types: Aggressive, Benevolent, Cheater, Human, and Random. The orders are executed based on the behavior of the players and their respective characteristics. The logic for the previous command input has been modified to align with the actions of a human player.

**Reason:**The system has been altered to accommodate the

distinct behaviors and tendencies that are commonly observed in each category of players, allowing for adjustments to be made accordingly.

**Added Test Cases:**

1. testInitialOrder: The test is used to check whether Order creation is deployed initially.

(AggressiveTest.java)

2. testDeployOnStrongestCountry: The test is used to check whether aggressive player deploys armies on strongest country. (AggressiveTest.java)

3. testDeploy: The test is used to check whether benevolent player deploy armies on weakest country. (BenevolentTest.java)

4. testAttack: The test is used to check whether benevolent player attacks to weakest neighbor. (BenevolentTest.java)

**Modified Test Cases (if any):** None

**12. Adapter Pattern:**

**Before:** There was only one map format supported by the game. (i.e. Domination map).

**After:** The system now has the ability to perform read and write operations for two types of map file formats: the original domination format and the conquest format. This was made possible by refactoring the map loading format to utilize the adapter pattern, which allows for smooth interaction and adaptation between the two map file formats.

**Reason:** The user is provided with the ability to choose from various map formats. Additionally, the process of adding new map formats to the existing code is made simple.

**Added Test Cases:**

1.testReadMapFile: This test case is used to read domination map..

(DominationMapTest.java)

2. testReadMap: This test case is used to read conquest map. (ConquestMapTest.java)

3. testAddContinent: This test case is used to add continent in conquest map.

(ConquestMapTest.java)

4. testRemoveContinent: This test case is used to remove continent in a conquest map.

(ConquestMapTest.java)

5. testAddCountry: This test case is used to add country in a conquest map.

(ConquestMapTest.java)

6. testRemoveCountry: This test case is used to remove a country in a conquest map.

(ConquestMapTest.java)

**Modified Test Cases (if any):** None.

**13. Extract game state specific methods including loadGame and**

**saveGame to separate class GameService:**

**Before:** All the game related commands were handled by phase classes that were assigned to each individual phase.

**After:** Implemented a GameService class to handle certain game related methods such as

loading and saving a game state.

**Reason:** To implement the features of load game and save game in warzone.

**Added Test Cases:**

1. testValidSaveGame(): Tests valid savegame command.

2. testInvalidSaveGame(): Tests invalid savegame command.

3. testValidLoadGame(): Tests valid loadgame command.

4. testinvalidLoadGame(): Tests invalid loadgame command.

All tests were added in GameEngineConrollerTest.java.

**Modified test cases:** None

**14. Update issueOrder phase methods to randomly update whether**

**next order is required or not for handling automatic players, as**

**user won’t play a role in this gameplay phase:**

**Before:** The user is asked to indicate whether they would like to enter more orders by responding with either "yes" or "no".

**After:** The issueorderphase method have been modified to include a random selection process that determines whether or not the next order

is required to control automatic players, thereby eliminating the need for user interaction during this phase of the game.

**Reason:** To set a limit on the number of commands an automatic player can execute during a

turn.

**Added test cases:** None.

**Modified test cases: None**

**15. Based on strategy of player issueOrder method is adapted to accept new orders rather than from user input:**

**Before:** During the issue order phase, the user was asked to input orders

which were executed in the order they were entered.

**After:** The method for addressing the problem has been modified to accommodate orders that are randomly generated from

various player strategies.

**Reason:** To adapt to different behavioural patterns

**Added test cases:**

1. testReadMapFile: This test case is used to read domination map.

(DominationMapTest.java)

2. testReadMap: This test case is used to read conquest map. (ConquestMapTest.java)

3. testAddContinent: This test case is used to add continent in conquest map.

(ConquestMapTest.java)

4. testRemoveContinent: This test case is used to remove continent in a conquest map.

(ConquestMapTest.java)

5. testAddCountry: This test case is used to add country in a conquest map.

(ConquestMapTest.java)

6. testRemoveCountry: This test case is used to remove a country in a conquest map.

(ConquestMapTest.java)

**Modified test cases:** None