**Refactoring for Build 2**

**Potential List for Refactoring Build 2 –**

1. Implementation of **validateCommand** method in edit, load and save phase.
2. Implementation of **Enum** in Commands.java.
3. **Magic** **String** replacement with add, remove command.

Example: -add, -remove.

1. Implementation of the **Magic** **String** replacement in the GameInitialization.java.
2. Use of **Dependency** **Injection**: The Commands class is directly instantiated in the main method. This could be replaced with dependency injection to make the code more testable and flexible.
3. **User** **Interface** for Validation Strategy: Use of an MapValidationStratergy interface to define the validation logic.
4. **Null** **pointer** exceptions were removed during the refactoring phase. These were removed by introducing a specific syntax for. equals method all throughout our project.
5. **Removed** the Commands.java file as the code is **integrated** with the GameEngine.java.
6. Use of **TextTable** library for displaying the output map in the console.
7. **State** **Pattern** Implementation - The state pattern in Java is a behavioural software design pattern that allows an object to alter its behaviour when its internal state changes. The state design pattern is generally used in cases when an object depends on its state and its behavior must be changed during run time depending on its internal state.

In the project, refactoring for the build 1 is done to support state pattern design.

1. Resource Management with **try**-**with**-**resources**: The Scanner object is currently closed using a finally block. You can simplify this by using a try-with-resources statement. This ensures proper resource management and eliminates the need for an explicit close call.
2. Use of **StringBuilder** for Efficiency - Use of StringBuilder can be done instead of using l\_dataString while reading the lines. String Concatenation in Java is inefficient as it creates new strings every time.
3. Remove **Explicit** **Type** **Argument** in Continent.java, Country.java, GameEngine.java, GameMap.java, MapValidation.java and MapValidationTest.java
4. Replace assignment with **Compound** **Operators** in the Commands.java.
5. Split of **Multiple** **Variable** **declaration** in the ReadMap.java and WriteMap.java.

**Main Refactoring Operations -**

1. validateCommand method implemented in edit, load and save phase.

Screenshot –

A screen shot of a computer code

Description automatically generated

Test Cases - Before - 

Test Cases – After - 

1. Enum implemented in Commands.java.

Screenshot –

A screenshot of a computer screen

Description automatically generatedA screen shot of a computer program

Description automatically generated

Test Cases - Before - 

Test Cases – After - 

1. Magic String replacement implemented in GameInitialization.java.

Screenshot - Before

A computer screen shot of text

Description automatically generated

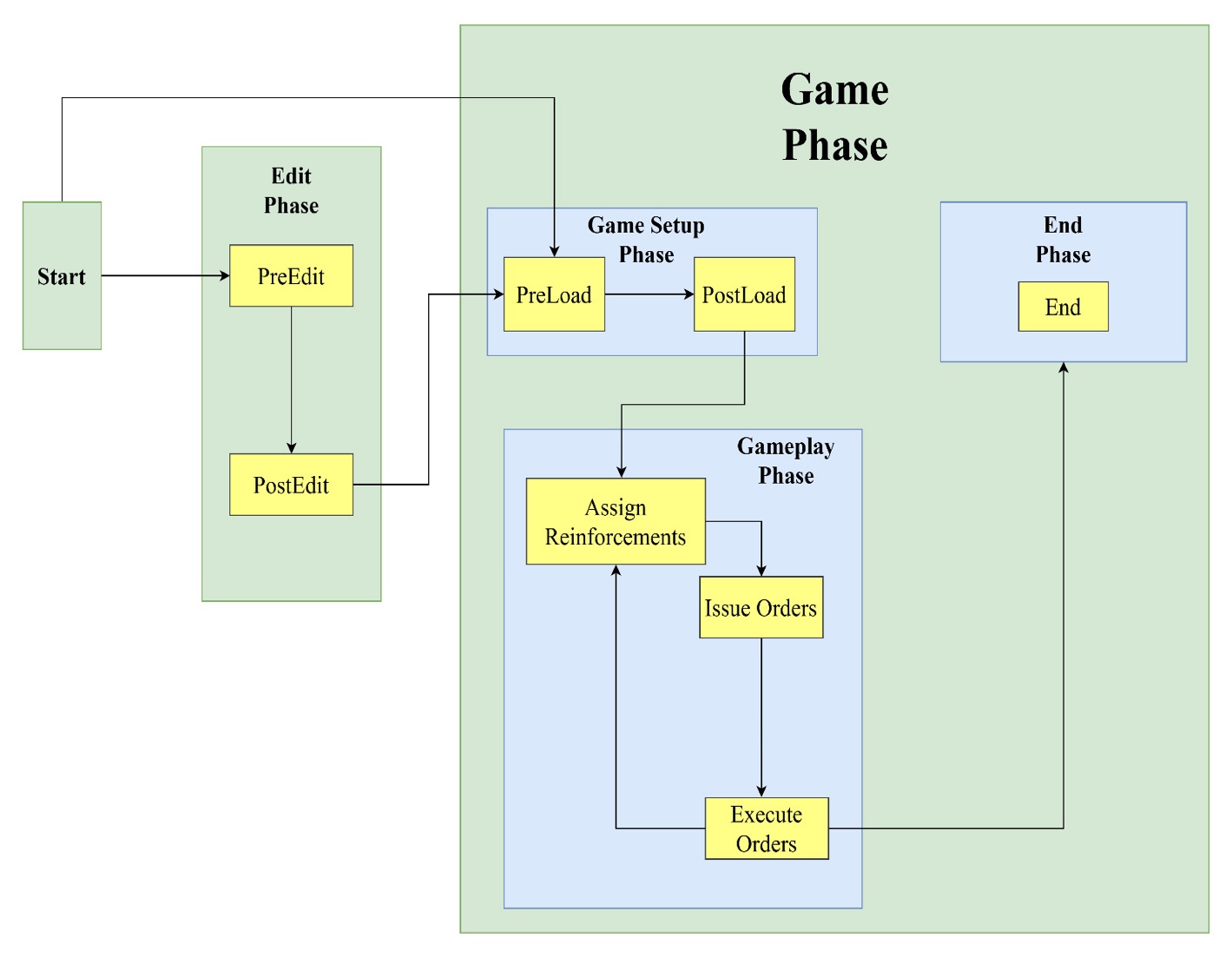
Screenshot – AfterA computer screen with text

Description automatically generatedA computer screen with text

Description automatically generated

1. State Pattern Implementation – The state pattern in Java is a behavioural software design pattern that allows an object to alter its behaviour when its internal state changes. The state design pattern is generally used in cases when an object depends on its state and its behavior must be changed during run time depending on its internal state. In the project, refactoring for the build 1 is done to support state pattern design.

Screenshot –



1. Null pointer exceptions were removed during the refactoring phase. These were removed by introducing a specific syntax for equals method all throughout our project.

Screenshot –

A computer screen with white and orange text

Description automatically generated