List of potential refactoring targets:

1. Implementing strategy pattern for player behaviour
2. Adding a new variable in Country model to store count of potential armies as per the issued orders
3. Dividing MapController further into two classes; MapController and MapEditor.
4. Adding new class to store a players’ information
5. Adding new class to store information related to command information
6. Using one console for map editing and game play phase.
7. Removing ExecuteCommandView and calling respective functions from ValidateCommandView class
8. Creating a class in controller for display console for adding command during issue order phase
9. Using ValidateCommandView for executing orders
10. Adding instance of view in model classes
11. Adding instance of model in view classes
12. Adding instance of controller in both model classes

List of 5 refactoring targets:

1. Implementing strategy pattern for player behaviour

* It was selected because it was a requirement of build 3.
* Implemented multiple test cases for checking each of the 4 automated strategies. Aggressive test cases check if the player attacks with its strongest country. Benevolent cases check if the distribution is even. Random cases perform applies multiple tests to check randomness. Cheater test cases check if player conquers neighbours without issuing orders.
* Earlier, all players were user controlled. Now, after inclusion of the strategy pattern each player can behave differently based on the strategy assigned to them.

1. Adding a new variable in Country model to store count of potential armies as per the issued orders

* It was selected because it was required for implementation of benevolent strategy
* Implemented 1 test case to check if the variable provide accurate information as per issued orders.
* Earlier, the users used to provide the orders. Now, because of automation a variable was required to store the state of armies. This state was used to check in any more order are required or not.

1. Merging all command validation classes for different phases into one class
   * It was selected because for every command entered we had to validate the commands and calling resp. functions/methods in the Phase class.
   * Implemented 3 test cases for checking if commands are validated properly. (1) 1 test case for checking if any random incorrect command throws an error. (2) 2 test for validating correct commands and setting correct phases after execution.
   * Earlier, we had three set of classes for validating MapEditing, GamePlay start-up and GamePlay commands. Now, we have one common validation class for all command types and after validation respective methods are called from Phase class to carry out the command. Instances of Phase handles the validity of the command for the phase or not and displays appropriate message.
2. Dividing MapController class into small chunks
   * It was selected because the MapController class had become too lengthy and cumbersome to understand.
   * Implemented 3 test cases for checking the validation of map. 1 test case each for checking if map is connected graph, each continent is a connected sub-graph and check if continent is empty.
   * Earlier, we were handling all functions related to map in MapController class. Now, we have a separate class MapValidator to handle all functionality related to handling validation of the map.
3. Handling of console based on phase
   * It was selected because we had separate console for MapEditing and GamePlay phase.
   * Implemented 2 test cases for checking if the consoles are properly switching. The test cases check if the load map command initialized start-up phase, which is only initialized when console is switched.
   * Earlier, we were handling switching of console based on an integer value. Now, we have instances of phases for different phase. We are checking the type of phase instance to handle the switching.