List of potential refactoring targets:

1. Implementing strategy pattern for player behaviour
2. Implementing adapter pattern for maps loading
3. Updated save command
4. Dividing MapController class into small chunks
5. Updated player command
6. Dividing MapController further into two classes; MapController and MapEditor.
7. Adding new class to store a players’ information
8. Adding new class to store information related to command information
9. Using one console for map editing and game play phase.
10. Removing ExecuteCommandView and calling respective functions from ValidateCommandView class
11. Creating a class in controller for display console for adding command during issue order phase
12. Using ValidateCommandView for executing orders
13. Adding instance of view in model classes
14. Adding instance of model in view classes
15. 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. Implementing adapter pattern for maps loading

* It was selected because it was requirement of build 3.
* Implemented 4 test cases to check if both types of maps are loaded correctly and test cases to check if one format can be saved to another.
* Earlier, the users used to provide one type of map. Now, the map is loaded and saved and the formatting is taken into account.

1. Updated save command
   * It was selected because of requirement needs.
   * Implemented 3 test cases for checking if save works correctly. (1) 1 test case for saving to domination map. (2) 1 test case for saving to conquest map and (3) 1 test case to save using invalid command.
   * Earlier, we had used to save as domination map format. Now, we provide 1 to save as domination format or 0 to save as conquest format.
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 reading/writing of maps. (1) 1 test to check if we can read conquest map and save as domination. (2) 1 test to check if we can read domination map and save as conquest map. (3) 1 test cases to check new save command.
   * Earlier, we were reading and writing to a map from MapController class. Now, we have MapLoaderWarzone class for reading/writing domination maps.
3. Updated player command
   * It was selected because it was a requirement.
   * Implemented 2 test cases for checking if Player behaviours are properly created. (1) 1 test case to check if new command works. (2) 1 test case to check old command doesn’t work.
   * Earlier, all players were human. Now, based on the command the player can have 5 types of behaviour.