REFACTORING DOCUMENT

Potential refactoring targets (Actual refactoring targets in bold):

1. **Changes needed in the mapparser for adapter design pattern implementation.**
2. **Moving the orders into a folder.**
3. **Moving checkAndCreatedirectory() to GameCommonUtils.**
4. **MapCommonUtils changes to change the getMapDetails() to generic form to accept map format of domination and conquest.**
5. **createOrders() to ensure order creation for the computerized version of the players to work.**
6. Storing continent object in Country object's continent it belongs to. - Instead of iterating through continent object list to get the desired continent object.
7. loadGameMap() in LoadMapPhase is rewritten to accept the conquest game map format.
8. executePhase() is refactored to handle both the tournament and the singlegame mode. It is done using GameMode class which in itself is altered to accommodate both the modes.
9. GameDetails class has an outer class GameMode class to accommodate the two different game modes.
10. Remove for loops for searching for a country given country name in multiple places by defining a method in Country Class. – To increase the readability of code.

11. Define a separate class to perform map command validation instead of repeating the same code in multiple edit methods. – To increase the readability of code and to reduce code duplication.
12. Grouping multiple declarations of same type in different lines to one line. – To increase the readability of code and to reduce the length of code.
13. Remove lines that cannot throw Exceptions from try and catch blocks. – Only lines that may throw exceptions should be present in try block.
14. Unused GameConstants and GameMessageConstants are removed.
15. Each game is issued a Game number which proves useful to generate the tournament result, found in the GameDetails class.

Actual refactoring targets:

1. mapparser changes: DominationGameMapReader, ConquestGameMapReader, MapReaderAdapter classes added.
Additional test cases needed: ConquestGameMapReaderTest

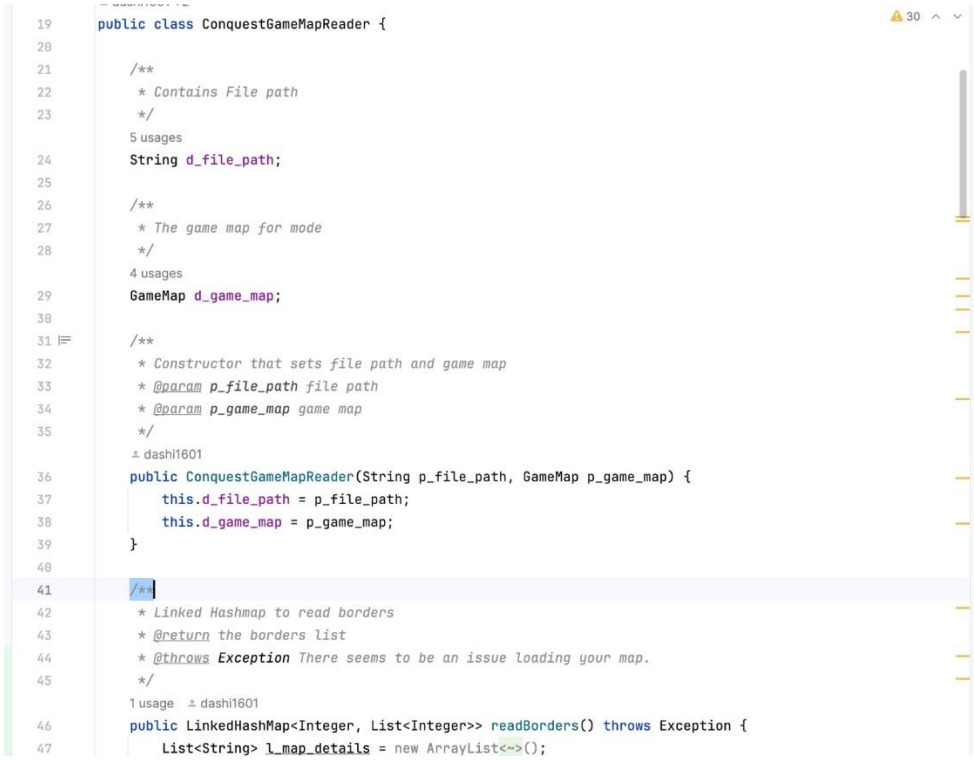
Why: Previously the GameMap object would work only with the domination map formats. The ConquestGameMapReader (adaptee) uses the MapReaderAdapter (adapter) to make appropriate calls from the adapter class translating the code into the Domination Map format. The calls for the Domination Map is done using the DominationGameMapReader. This ensures the game accepts both the Domination and the Conquesst Game Map formats.

Before:

```
+   //move load
    public void loadGameMap(GameMap p_game_map) throws Exception {

-       d_logger.addLogger("Game Map Loaded");
-       // loading map objects
-       p_game_map.loadBorders();
-       p_game_map.loadContinents();
-       p_game_map.loadCountries();
+      String l_map_type = MapCommonUtils.checkMapType(this.d_map_path);
+      if(l_map_type.compareTo("DominationMap") == 0){
+          DominationGameMapReader l_dom_reader = new DominationGameMapReader(this.d_map_path, p_game_map);
+          l_dom_reader.loadContinents();
+          l_dom_reader.loadCountries();
+          l_dom_reader.loadBorders();
+
+      }else if(l_map_type.compareTo("ConquestMap") == 0){
+           ConquestGameMapReader l_conq_reader = new ConquestGameMapReader(this.d_map_path,p_game_map);
+          MapReaderAdapter l_adapter = new MapReaderAdapter(l_conq_reader);
+          l_adapter.loadContinents();
+          l_adapter.loadCountries();
+          l_adapter.loadBorders();
+
+      }else{
+          throw new GameException(GameMessageConstants.D_MAP_LOAD_FAILED);
+      }
```

After:

```
 * Class for map reader adapter                                          ▲5 ✓1
 */
dashi1601 +1
public class MapReaderAdapter extends DominationGameMapReader {

    /**
     * Contains conquest map
     */
    5 usages
    private ConquestGameMapReader d_conquest_map;

    /**
     * Map reader adapter
     * @param p_conquest_map Conquestmapreader object
     */
    // load functions that will be overridden
    dashi1601
    public MapReaderAdapter(ConquestGameMapReader p_conquest_map) {
        super(p_conquest_map.d_file_path, p_conquest_map.d_game_map);
        this.d_conquest_map = p_conquest_map;
    }

    /**
     * Method to load borders
     * @throws Exception if any exception rises in the code block
     */
    2 usages  dashi1601
    @Override
    public void loadBorders() throws Exception {
        LinkedHashMap<Integer, List<Integer>> l_borders =  d_conquest_map.readBorders();
        this.d_game_map.d_borders = l_borders;
    }
```



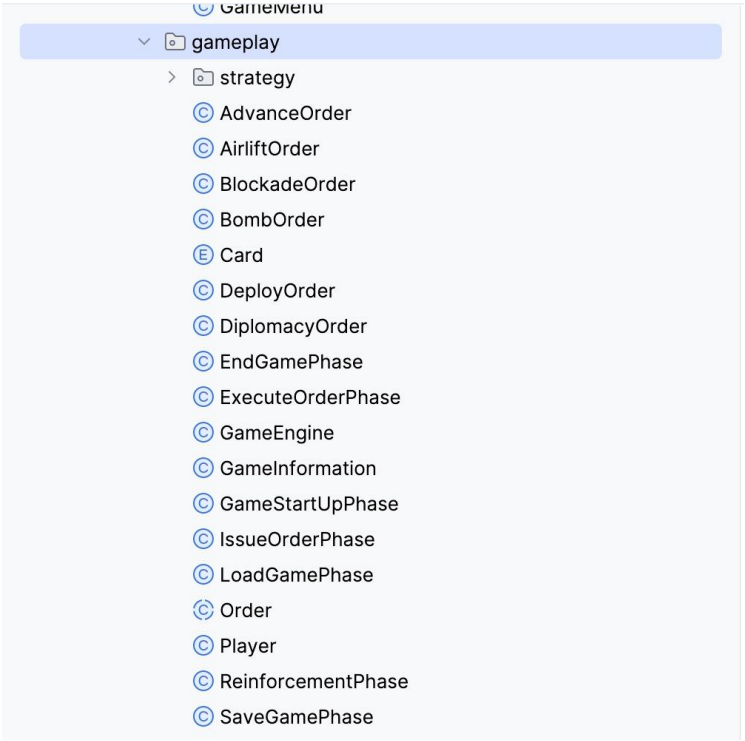
```
public class ConquestGameMapReader {

    /**
     * Contains File path
     */
    5 usages
    String d_file_path;

    /**
     * The game map for mode
     */
    4 usages
    GameMap d_game_map;

    /**
     * Constructor that sets file path and game map
     * @param p_file_path file path
     * @param p_game_map game map
     */
    dashi1601
    public ConquestGameMapReader(String p_file_path, GameMap p_game_map) {
        this.d_file_path = p_file_path;
        this.d_game_map = p_game_map;
    }

    /**
     * Linked Hashmap to read borders
     * @return the borders list
     * @throws Exception There seems to be an issue loading your map.
     */
    1 usage  dashi1601
    public LinkedHashMap<Integer, List<Integer>> readBorders() throws Exception {
        List<String> l_map_details = new ArrayList<~>();
```

2. Order changes: moved AdvancedOrder, DeployOrder, AirliftOrder, AdvanceOrder, BombOrder, DiplomacyOrder into a folder.

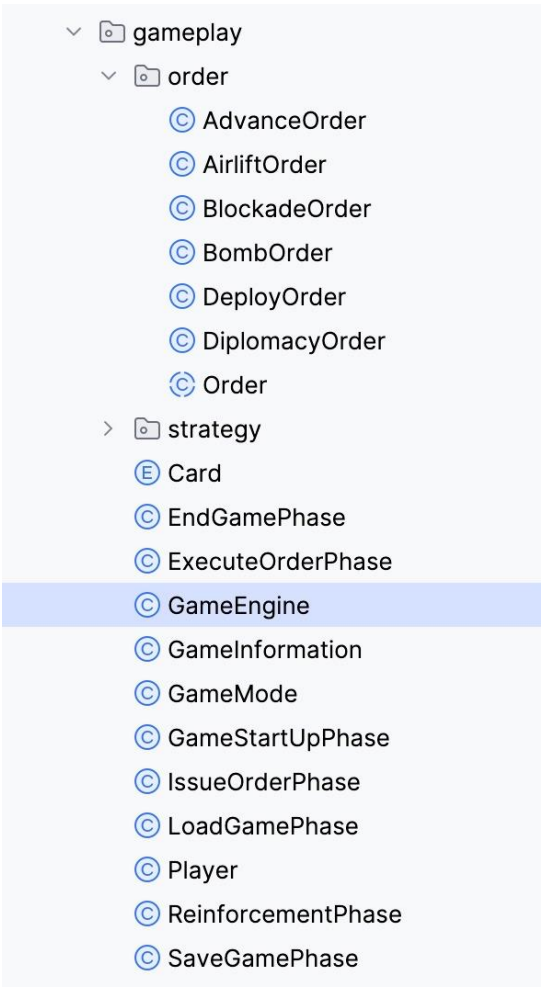
Additional test cases needed: No test cases were added, but the test folders for the Order is also put in a separate folder ensuring consistency.

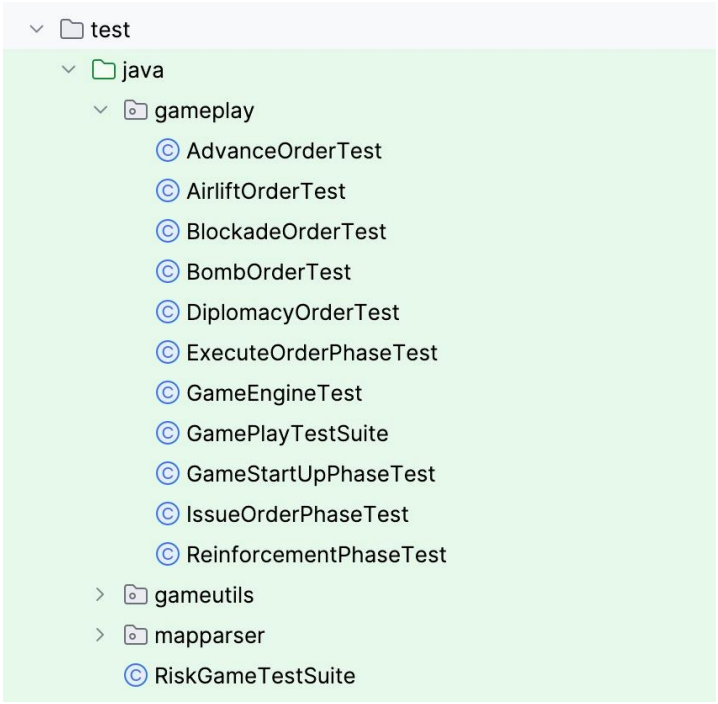
Why: Previously the Orders were in the gameplay folder. gameplay previously had most of the classes involved in the game, besides the Order is a pattern that needs to be implemented, hence the need to put in the Order commands into a folder. This ensures that the code is refactored and cleaned.

Before:



After:





3. LogEntrywriter changes: checkAndCreateDirectory() moved to GameCommonUtils.

Additional test cases needed: None

Why: There was a much higher need to create new files with the introduction of save map option. Besides, the same is needed for loading the map files, hence was put in the GameCommonUtils class of the gameutils folder.

Before:

```
26      /**
27       * Checks and create the directory if not present
28       * @param p_directory_path directory path parameter
29       */
         2 usages  AswiinR
30      public static void checkAndCreateDirectory(String p_directory_path) throws Exception {
31          File l_file_dir = new File( pathname: "").getCanonicalFile();
32          p_directory_path = l_file_dir.getParent() + p_directory_path;
33
34          File l_directory = new File(p_directory_path);
35          if (!l_directory.exists() || !l_directory.isDirectory()) {
36              l_directory.mkdirs();
37          }
38      }
39
40  }
41
```

After:

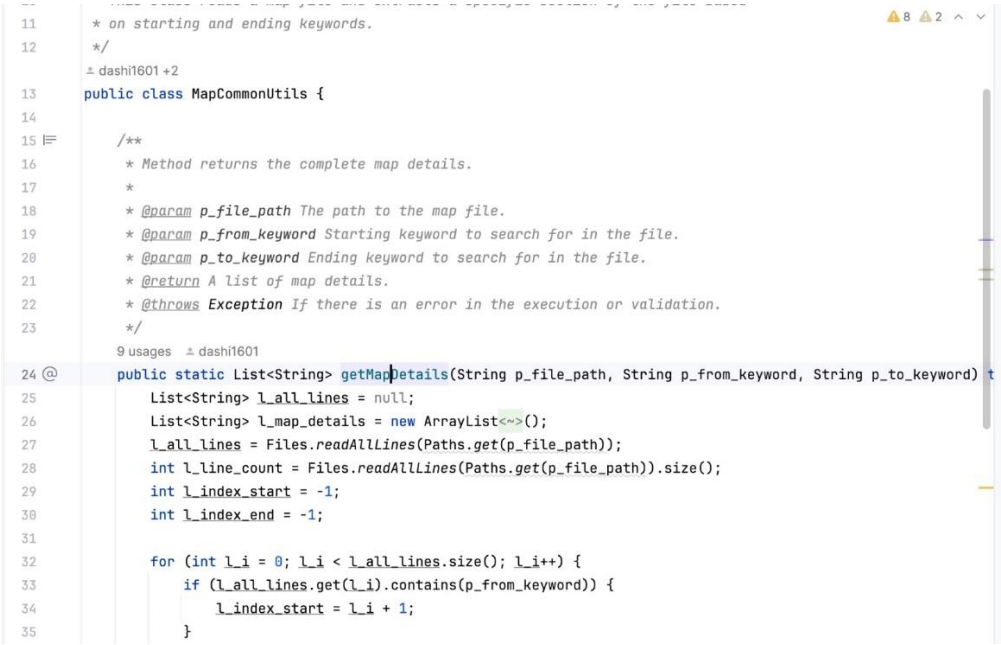
```
2   2
3   3   import constants.GameConstants;
    4 + import gameutils.GameCommonUtils;
4   5
5   6   import java.io.PrintWriter;
6   7   import java.io.BufferedWriter;
        @@ -32,7 +33,7 @@ public void writeLogFile(String p_log_message) {
32  33          PrintWriter l_writer = null;
33  34          try {
34  35
35      -           checkDirectory("logfiles");
    36  +           GameCommonUtils.checkAndCreateDirectory("logfiles");
36  37              String l_log_file_path = "logfiles" + File.separator + GameConstants.D_LOG_FILE_NAME;
37  38              l_writer = new PrintWriter(new BufferedWriter(new FileWriter(l_log_file_path, true)));
38  39              l_writer.println(p_log_message);
        @@ -44,15 +45,4 @@ public void writeLogFile(String p_log_message) {
44  45          }
45  46      }
46  47
47      -   /**
48      -    * Checks the directory
49      -    * @param p_directory_path directory path parameter
50      -    */
51      -   private void checkDirectory(String p_directory_path) {
52      -       File l_directory = new File(p_directory_path);
53      -       if (!l_directory.exists() || !l_directory.isDirectory()) {
54      -           l_directory.mkdirs();
55      -       }
56      -   }
57      -
```

4. MapCommonUtils changes: getMapDetails() changes.

Additional test cases needed: ConquestGameMapReaderTest.

Why: Method converted into a generic form so as to accept the conquest game map format. This ensures the formats of the conquest maps are also valid and working with the game.

After:



```
 * on starting and ending keywords.
 */
dashi1601 +2
public class MapCommonUtils {

    /**
     * Method returns the complete map details.
     *
     * @param p_file_path The path to the map file.
     * @param p_from_keyword Starting keyword to search for in the file.
     * @param p_to_keyword Ending keyword to search for in the file.
     * @return A list of map details.
     * @throws Exception If there is an error in the execution or validation.
     */
    9 usages  dashi1601
    public static List<String> getMapDetails(String p_file_path, String p_from_keyword, String p_to_keyword) t
        List<String> l_all_lines = null;
        List<String> l_map_details = new ArrayList<~>();
        l_all_lines = Files.readAllLines(Paths.get(p_file_path));
        int l_line_count = Files.readAllLines(Paths.get(p_file_path)).size();
        int l_index_start = -1;
        int l_index_end = -1;

        for (int l_i = 0; l_i < l_all_lines.size(); l_i++) {
            if (l_all_lines.get(l_i).contains(p_from_keyword)) {
                l_index_start = l_i + 1;
            }
```

5. createOrders() changes: in accordance to strategy pattern.

Additional test cases needed: None.

Why: Previously the game was played only by the users – humans. The strategy pattern is implemented to incorporate the various strategies – random, benevolent, cheater, aggressive and human. createOrders() is used to create orders for the computerized players of the game, if chosen. The computerized version being the four types except the human player strategy mentioned above.

Before:

```
322     @Override
323     public void executePhase() throws Exception {
324         System.out.printf("%nstart issuing your orders or enter endgame to terminate%n");
325         d_current_game_info = GameInformation.getInstance();
326
327         BufferedReader l_reader = new BufferedReader(new InputStreamReader(System.in));
328
329         LinkedHashMap<String, Player> l_player_list = d_current_game_info.getPlayerList();
330
331         for (Map.Entry<String, Player> l_player : l_player_list.entrySet()) {
332             Player l_player_obj = l_player.getValue();
333             int l_current_armies = l_player_obj.getCurrentArmies();
334             System.out.println("Start by issuing deploy order");
335
336             String l_input_command;
337             // Executing deploy order
338             while (l_current_armies > 0) {
339                 System.out.println("Player: "+ l_player.getKey() + " turn");
340                 System.out.println("Remaining Armies: " + l_current_armies);
341                 l_player_obj.d_current_order = null;
342                 try {
343                     System.out.println();
344                     l_input_command = l_reader.readLine();
345                     if (l_input_command.equals("endgame")) {
346                         d_current_game_info.setCurrentPhase(new EndGamePhase());
347                         return;
348                     }
```

After:

```
35      * Allows player to issue orders.
36      * @throws Exception If an error occurs.
37      */
        AswiinR
38      @Override
39      public void executePhase() throws Exception {
40          d_current_game_info = GameInformation.getInstance();
41
42          LinkedHashMap<String, Player> l_player_list = d_current_game_info.getPlayerList();
43          Player l_last_session_player = d_current_game_info.getLastSessionPlayer();
44          d_current_game_info.setLastSessionPlayer(null);
45
46          for (Map.Entry<String, Player> l_player : l_player_list.entrySet()) {
47              Player l_player_obj = l_player.getValue();
48              if (l_last_session_player != null && !l_player_obj.getPlayerName().equals(l_last_session_player.ge
49                  continue;
50              }
51
52              // issuing player orders
53              l_player_obj.issue_order();
54              l_last_session_player = null;
55
56              if (d_current_game_info.getCurrentPhase() instanceof EndGamePhase) {
57                  return;
58              }
59          }
60
61          d_current_game_info.setCurrentPhase(this.nextPhase());
62      }
63  }
```

```java
/**
 * Adds current order to the players list of orders.
 */
3 usages  AswiinR
public void issue_order() throws Exception {
    List<Order> l_player_orders = this.d_player_strategy.createOrders( p_player_obj: this);
    if (l_player_orders != null && !l_player_orders.isEmpty()) {
        d_orders_list.addAll(l_player_orders);
    }
}
```